# Tight upper bound on splitting by linear combinations for pigeonhole principle *

Vsevolod Oparin[1,2]

[1] St. Petersburg Academic University
8/3 Khlopina, St.Petersburg, 194021, Russia
[2] Steklov Institute of Mathematics at St. Petersburg,
27 Fontanka, St.Petersburg, 191023, Russia
oparin.vsevolod@gmail.com

**Abstract.** The usual DPLL algorithm uses splittings (branchings) on single Boolean variables. We consider an extension to allow splitting on linear combinations mod 2, which yields a search tree called a linear splitting tree. We prove that the pigeonhole principle has linear splitting trees of size $2^{O(n)}$. This is near-optimal since Itsykson and Sokolov [1] proved a $2^{\Omega(n)}$ lower bound. It improves on the size $2^{\Theta(n \log n)}$ for splitting on single variables; thus the pigeonhole principle has a gap between linear splitting and the usual splitting on single variables. This is of particular interest since the pigeonhole principle is not based on linear constraints. We further prove that the perfect matching principle has splitting trees of size $2^{O(n)}$.

## 1 Introduction

Splitting is a well known method for solving NP-hard problems. In the case of the satisfiability problem for a Boolean CNF formula, the highly successful DPLL algorithms are based on splitting [2,3]. DPLL works in the following way to search for satisfying assignments for a CNF formula $\phi$. The algorithm chooses a variable $x$ and a first value $\alpha$ to substitute. The algorithm substitutes $x = \alpha$ and runs recursively on the simplified formula $\phi|_{x=\alpha}$. If this does not succeed, it tries $\phi|_{x=1\oplus\alpha}$. If there is no success again, the algorithm returns FAIL. Otherwise, it returns a satisfying assignment.

There is extensive research on hard examples for DPLL algorithms. It is well known that systems of linear equations mod 2, such as the Tseitin tautologies, are hard for DPLL and resolution [4,5,6,1]. However, they can be quickly solved by splitting on linear combinations mod 2. Thus it is natural to consider generalizing DPLL to use linear splitting.

A linear splitting algorithm maintains a system of linear equations over $\mathbb{F}_2$. Initially, the system is empty. Instead of choosing a single variable, the algorithm chooses a linear form $\sum_i \alpha_i \cdot x_i$ and a first value $\beta$. The algorithm adds the

* Research is partially supported by the Government of the Russian Federation under Grant 14.Z50.31.0030.

equation $\sum_i \alpha_i \cdot x_i = \beta$ to the system and runs itself recursively. The second call runs with the equation $\sum_i \alpha_i \cdot x_i = 1 + \beta$.

On every step of the recursion the algorithm checks three conditions before splitting again.

- If the system is inconsistent, the algorithm backtracks. This condition can be checked in polynomial time by Gaussian elimination.
- If the system violates any single clause, the algorithm backtracks. The system violates a clause $C = l_1 \vee l_2 \vee \cdots \vee l_k$, if for every $i \in [k]$ the system plus the equation $l_i = 1$ is inconsistent. All clauses can be checked in polynomial time.
- If the system has exactly one solution, the algorithm returns a satisfying assignment. This is also can be checked by Gaussian elimination.

Otherwise, the algorithm selects a linear form for the next splitting. The entire execution tree is a binary tree called a linear splitting tree.

Several prior works combine linear substitutions and splitting. For example, the recent algorithm of Seto and Tamaki [7] solves the satisfiability problem for an arbitrary formula of linear size $c \cdot n$ in $2^{(1-\mu_c)\cdot n}$ steps using splitting by linear forms. Kulikov and Demenkov [8] use a formula which is non-trivial up to the linear number of linear substitutions to show a lower bound of $3n - o(n)$ for the circuit complexity over the full binary basis.

Itsykson and Sokolov [1] provide a family of hard formulas for linear splitting. In particular, they prove that the pigeonhole principle, where $m$ pigeons fly to $n$ holes, has no linear splitting tree of size less than $2^{\Omega(n)}$. On the other hand, it is known that splitting by single Boolean variables gives a tree of size $2^{O(n \log n)}$ and this bound is tight [9]. So it is natural to ask whether the bound for the pigeonhole principle with linear splitting is tight.

We answer this by showing that the pigeonhole principle has a linear splitting tree of size $2^{O(n)}$. The pigeonhole principle is not based on linear constraints, so this gap between splitting by single variables and by linear forms is interesting.

We also consider the perfect matching principle built on arbitrary graphs. Any graph of odd cardinality has a polynomial-size splitting tree [1], but nothing is known about graphs of even cardinality. For an arbitrary $n$, we prove a $2^{O(n)}$ upper bound on splitting trees for graphs on $n$ vertices.

## 2 Preliminaries

We will use the following notation: $[n] = \{1, 2, \ldots, n\}$. Let $X = \{x_1, \ldots, x_n\}$ be a set of variables that take values from $\mathbb{F}_2$. A linear form is a polynomial $\sum_{i=1}^{n} \alpha_i \cdot x_i$ over $\mathbb{F}_2$.

Consider a binary tree $T$ with edges labeled by linear equalities. For every vertex $v$ of $T$ we denote by $\Phi_v^T$ the system of all equalities that are written along the path from the root to vertex $v$.

A *linear splitting tree* for a CNF formula $\varphi$ is a binary tree $T$ with the following properties. Every internal node is labeled by a linear form that depends on variables from $\varphi$. For every internal node labeled by a linear form $f$, one of the incident edges going to the children is labeled by $f = 0$, and the other one is labeled by $f = 1$.

For every leaf $v$ of the tree exactly one of the following conditions holds: 1) The system $\Phi_v^T$ has no solution. We call such leaf degenerate. 2) The system $\Phi_v^T$ is satisfiable but violates a clause $C$ of the formula $\varphi$. We say that such leaf violates clause $C$. 3) The system $\Phi_v^T$ has exactly one solution and the solution satisfies the formula $\varphi$. We call such leaf satisfying.

A linear splitting tree may be viewed as a tree of recursive calls for the algorithm solving SAT for the CNF formula $\varphi$. The algorithm maintains a system of linear equations $\Phi$ and starts with a given formula $\varphi$ and $\Phi = \mathtt{True}$. Given a formula $\varphi$ and a system of linear equations $\Phi$, the algorithm looks for a satisfying assignment of $\varphi \wedge \Phi$. At every step the algorithm chooses a linear form $f$ and a value $\alpha \in \mathbb{F}_2$ and makes two recursive calls: on the input $(\varphi, \Phi \wedge (f = \alpha))$ and on the input $(\varphi, \Phi \wedge (f = 1 + \alpha))$.

The algorithm backtracks in one of the three cases: 1) The system $\Phi$ has no solution; 2) The system $\Phi$ contradicts a clause $C$ of the formula $\varphi$. (A system $\Psi$ contradicts a clause $(l_1 \vee l_2 \vee \dots \vee l_k)$ iff for all $i \in [k]$ the system $\Psi \wedge (l_i = 1)$ is unsatisfiable.) 3) The system $\Phi$ has a unique solution that satisfies $\varphi$. All three cases can be checked in polynomial time.

Note that if it is enough to find merely one satisfying assignment, the algorithm may stop at the first satisfying leaf. In the case of unsatisfiable formulas, the algorithm must traverse the whole splitting tree.

**Proposition 1.** *[1] For every linear splitting tree $T$ for a formula $\varphi$ it is possible to construct a splitting tree without degenerate leaves. The number of vertices in the new tree is at most the number of vertices in $T$.*

## 3 Upper bound for the pigeonhole principle

Let we have $m$ pigeons and $n$ holes. Every pigeon should fly to at least one hole. The pigeonhole principle states that if $m > n$, there exists a hole with at least two pigeons inside.

We encode the reverse statement into an unsatisfiable CNF formula. For $i \in [m]$ and $j \in [n]$ let $x_{i,j}$ be a variable such that the $i$-th pigeon flies to the $j$-th hole iff $x_{i,j} = 1$.

We encode the fact that the $i$-th pigeon flies somewhere by the clause

$$\bigvee_{j \in [n]} x_{i,j}.$$

Also we encode, that the $j$-th hole accepts at most one pigeon by the set of clauses

$$\neg x_{i_1,j} \vee \neg x_{i_2,j}$$

for every $i_1 \neq i_2 \in [m]$

We denote the conjunction of all these clauses by $\mathrm{PHP}_n^m$. Obviously, the formula $\mathrm{PHP}_n^m$ is unsatisfiable if $m > n$.

**Theorem 1.** *For all $m > n$ there exists a linear splitting tree for PHP$_n^m$ of size $2^{O(n)}$.*

*Proof.* The formula $\mathrm{PHP}_n^{n+1}$ is a subformula of $\mathrm{PHP}_n^m$. So it is enough to build a tree for $\mathrm{PHP}_n^{n+1}$ only.

We construct the tree by induction on $n$. The base $n = 1$ is trivial.

For $n > 1$, we reduce $\mathrm{PHP}_n^{n+1}$ to multiple copies of $\mathrm{PHP}_{n/2}^{n/2+1}$. This is done by building a linear splitting tree $T$ of size $2^{O(n)}$. Every leaf of $T$ either will violate one of the clauses or will correspond to an instance of $\mathrm{PHP}_{n/2}^{n/2+1}$. (The second kind of leaves will become the root of a tree for $\mathrm{PHP}_{n/2}^{n/2+1}$.)

The logarithm $\mathrm{LG}(n)$ of the size of the whole splitting tree for $\mathrm{PHP}_n^{n+1}$ can be expressed by the inequality

$$\mathrm{LG}(n) \le \log(2^{O(n)} \cdot 2^{\mathrm{LG}(\lfloor n/2 \rfloor)}) = O(n) + \mathrm{LG}(\lfloor n/2 \rfloor).$$

Hence $\mathrm{LG}(n) = O(n)$. So the size of the tree is $2^{O(n)}$.

We split the pigeons into two almost equal parts $L = [1, \lfloor (n+1)/2 \rfloor]$ and $R = [\lfloor (n+1)/2 \rfloor + 1, n+1]$. We refer to these parts as "left" and "right", respectively. For every hole $j$ we define two linear forms:

$$\begin{aligned} \mathrm{LEFT}(j) &= \textstyle\bigoplus_{i \in L} x_{i,j}, \\ \mathrm{RIGHT}(j) &= \textstyle\bigoplus_{i \in R} x_{i,j}. \end{aligned}$$

The tree $T$ starts with a full binary tree $T_Q$ of height $2n$. Every branch in $T_Q$ queries the values $\mathrm{LEFT}(j)$ and $\mathrm{RIGHT}(j)$ for every hole $j$. So $T_Q$ has $2^{2n}$ leaves. $T$ will be defined from $T_Q$ by replacing each leaf $\ell$ of $T_Q$ with a polynomial-size subtree $T_\ell$. In each $T_\ell$, all but possibly one of its leaves will be labeled by violated clauses (see Figure 1).

Fix a leaf $\ell$ of tree $T_Q$. For each hole $j$, we have fixed values of $\mathrm{LEFT}(j)$ and $\mathrm{RIGHT}(j)$. There are four cases.

1. $\mathrm{LEFT}(j) = 1$, $\mathrm{RIGHT}(j) = 1$.
2. $\mathrm{LEFT}(j) = 0$, $\mathrm{RIGHT}(j) = 1$.
3. $\mathrm{LEFT}(j) = 1$, $\mathrm{RIGHT}(j) = 0$.
4. $\mathrm{LEFT}(j) = 0$, $\mathrm{RIGHT}(j) = 0$.

If Case 1 holds for any hole $j$, the splitting tree $T_\ell$ has size $O(n^2)$ and finds a violated clause. $T_\ell$ can be described as a tree of recursive calls of the following algorithm. First, we go through all pigeons in the left part and split by $x_{i,j}$ for $i \in L$. Once $x_{i,j} = 1$ is found, we go through the pigeons of the right part and do the same until we find $x_{i',j} = 1$. Both variables exist since $\mathrm{LEFT}(j) = \mathrm{RIGHT}(j) = 1$. Once two non-zero variables are found, we return a violated clause.

Otherwise, we form $T_\ell$ by chaining together splitting trees $T_j$, one for each hole $j$. $T_j$ is formed depending on which of the Cases 2-4 holds.

2. Suppose Case 2 holds, so LEFT$(j) = 0$. The leaves of the tree $T_j$ either will violate an injectivity clause for hole $j$ or will ensure that no left pigeon flies to hole $j$. The tree $T_j$ has the following structure. For every left pigeon $i$ we split by the variable $x_{i,j}$. If $x_{i,j} = 1$, we can find a violated clause. Since LEFT$(j) = 0$, there must be another $x_{i',j} = 1$ for $i' \in L$. We split by $x_{i',j}$ for every pigeon $i' \in L \backslash \{i\}$ and find a violated clause.
Otherwise, the values for all left pigeons $i \in L$ are zero. In this case, we come to a leaf at which we know no left pigeon flies to the $j$-th hole.
3. Suppose Case 3 holds, so RIGHT$(j) = 0$. The tree $T_j$ is formed dually as above, and each leaf of $T_j$ either will violate an injectivity clause for hole $j$ or will ensure that no right pigeon flies to hole $j$.
4. Suppose Case 4 holds, so both LEFT$(j) = 0$ and RIGHT$(j) = 0$. $T_j$ is formed as in the previous two cases, but now we split on $x_{i,j}$ for all pigeons $i$. Each leaf of $T_j$ either will violate an injectivity clause or will ensure that no pigeon flies to hole $j$.

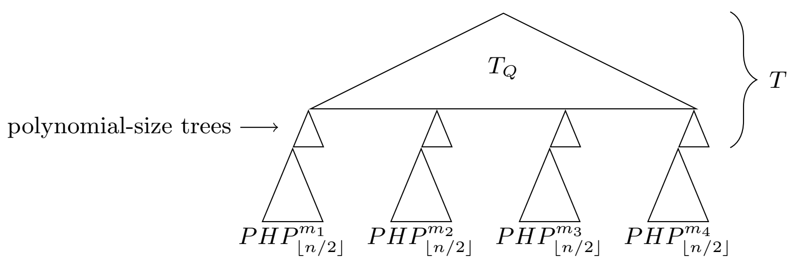


**Fig. 1.** Tree structure for PHP$^m_n$. The small polynomial-size trees contain trees $T_j$ chained together.

By design every tree $T_j$ has exactly one leaf not labeled by a violated clause. We call such a leaf *free*. We connect all trees $T_j$ each to the next one using free leaves, forming a *chain of trees*. The chain of trees forms the tree $T_\ell$ and has size $O(n^3)$. We attach chain $T_\ell$ to the leaf $\ell$.

The last tree in the chain has exactly one free leaf. At this leaf if any hole $j$ has LEFT$(j) = 0$, then no pigeon flies there from the left part $L$. Likewise, if any hole $j$ has RIGHT$(j) = 0$, then no pigeon flies there from the right part $R$. We separate holes into two disjoint parts: the first part has the holes $j$ with LEFT$(j) = 1$, the second part has the holes $j$ with RIGHT$(j) = 1$. The pigeons in $L$ can fly only to the first part, the pigeons in $R$ can fly only to the second part.

We show that at least one part of holes is less than the number of pigeons that fly there. Let $h_l$ and $h_r$ be the number of holes with LEFT$(j) = 1$ and RIGHT$(j) = 1$, respectively. We prove that either $h_l < |L|$ or $h_r < |R|$ by contradiction. Suppose $h_l \geq |L|$ and $h_r \geq |R|$. Since the sets of holes of the subformulas are distinct, $h_l + h_r \leq n$.

So

$$n \geq h_l + h_r \geq |L| + |R| = n + 1,$$

which is impossible.

Since $L$ and $R$ are less than $\lceil n/2 \rceil$, we can take a set of holes and pigeons that form a formula $PHP_{n/2}^{n/2+1}$ and attach a tree for this formula to the free leaf of the chain (see Fig. 1).

## 4 Upper bound on the perfect matching principle

In terms of CNF encoding, the perfect matching principle is similar to the pigeonhole principle. The formula $\mathrm{PMP}_G$, built on an arbitrary graph $G = \langle V, E \rangle$, encodes that every vertex has exactly one edge, taken into the matching. Formally, we provide a variable $x_e$ for each edge $e \in E$. For every vertex $v$ we encode that there exists at least one edge taken into the matching:

$$\bigvee_{u \in V : (u,v) \in E} x_{(u,v)}.$$

Also for every pair of edges $(u, v)$ and $(w, v)$ with a common endpoint $v$ we encode, that they can not be both taken into the matching:

$$\neg x_{(u,v)} \vee \neg x_{(w,v)}.$$

The formula $\mathrm{PMP}_G$ is the conjunction of all these clauses. Obviously, if the graph $G$ has no perfect matching, the formula is unsatisfiable.

Itsykson and Sokolov proved the following proposition.

**Proposition 2 ([1]).** *Let $G$ be a graph on an odd number of vertices. Then the formula $\mathrm{PMP}_G$ has a splitting tree of a polynomial size.*

Using Theorem 1 and Proposition 2, we prove the following theorem. Note that $n$ can be even.

**Theorem 2.** *Let $G = \langle V, E \rangle$ be a graph on $n$ vertices, which has no perfect matching. Then the formula $\mathrm{PMP}_G$ has a splitting tree of the size $2^{O(n)}$.*

*Proof.* We use Tutte's criterion to prove the theorem.

**Criterion 1 (Tutte, 1947)** *A graph $G$ has a perfect matching iff for any set $S \subseteq V$ the following statement holds: $o(G - S) \leq |S|$, where $G - S$ denotes the graph $G$ without vertices of the set $S$ and $o(G - S)$ denotes the number of connected components with odd cardinality in the obtained graph.*

We reduce the problem to the pigeonhole principle. Suppose the graph $G$ has no perfect matching. Let $S \subseteq V$ be a set such that $|S| < o(G - S)$.

Let $v_1, v_2, \cdots, v_n$ be the vertices of the set $S$ and $C_1, C_2, \cdots, C_m$ be the odd-cardinality connected components of the graph $G - S$. For every vertex $v_j$ and connected component $C_i$ we introduce a variable

$$y_{i,j} = \bigoplus_{(u,v_j) \in E, u \in C_j} x_{u,v_j}.$$

By the criterion $m > n$. Let us construct a formula $\mathrm{PHP}_n^m$, built on $y$'s, and build a splitting tree $T_y$ of size $2^{O(n)}$ as it was done in Theorem 1. Every node in the tree $T_y$ has a linear form on $y$'s.

We build a tree $T_x$ using the structure of $T_y$. We expand all $y$'s into the xor of $x$'s. At some nodes of $T_x$ we may have empty linear forms: no edge connects a vertex of $S$ and a connected component of $G - S$. In this case, one of the outgoing edges is labeled by the equation $0 = 1$. We truncate the corresponding subtree since the system becomes inconsistent.

We replace all leaves of $T_y$ by polynomial-size trees that finds violated clauses of $\mathrm{PMP}_G$. Fix a leaf $\ell$ of $T_y$ labeled by a clause $C_\ell$. We replace corresponding leaf of $T_x$ by a tree $T_\ell$. The structure of $T_\ell$ depends on clause $C_\ell$. There are two possible cases.

1. Clause $C_\ell$ is of type $\neg y_{i_1,j} \vee \neg y_{i_2,j}$. Then there exist two connected components $C_{i_1}$ and $C_{i_2}$ and vertex $v_j \in S$ s.t. $y_{i_1,j} = 1$ and $y_{i_2,j} = 1$.
2. Clause $C_\ell$ is of type $\bigvee_j y_{i,j}$. Then there exists connected component $C_i$ s.t. $y_{i,j} = 0$ for every vertex $v_j \in S$.

1. We have at least two edges in the matching coming to the vertex $v_j$. Tree $T_\ell$ corresponds to the recursive tree of the following algorithm that finds these edges. Check every edge $e$ between $v_j$ and $C_{i_1}$. Once the edge $e_1$ with $x_{e_1} = 1$ is found, switch to the second component $C_{i_2}$ and repeat the search. Once the second edge $e_2$ with $x_{e_2} = 1$ is found, return falsified clause $\neg x_{e_1} \vee \neg x_{e_2}$. Both edges exist since $y_{i_1,j} = \bigoplus_{(u,v_j) \in E, u \in C_{i_1}} x_{(u,v_j)} = 1$ and $y_{i_2,j} = \bigoplus_{(u,v_j) \in E, u \in C_{i_2}} x_{(u,v_j)} = 1$. Tree $T_\ell$ has size $O(n^2)$.

2. We have $y_{i,j} = 0$ for every $v_j \in S$. It means that either $x_{u,v} = 0$ for every $u \in C_i$ and $v \in S$ or there are at least two $x_{(u,v_j)} = 1$ for a fixed vertex $v_j$.

First, we ensure that the variables $x_{u,v} = 0$. Tree $T_\ell$ begins with a splitting tree $T_i$ that corresponds to the following algorithm. The algorithm goes through all variables $x_e$ for all edges between $S$ and $C_i$. Once, the algorithm finds $x_{(u,v_j)} = 1$ for a vertex $v_j$, it starts to look for the second $x_{(u',v_j)} = 1$ for all $u' \in C_i \setminus \{u\}$. There must exist such a variable since $y_{i,j} = \bigoplus_{(u,v_j) \in E, u \in C_i} x_{(u,v_j)} = 0$. Once the variable is found, the algorithm returns a violated clause.

If all variables are zero, we end up at a free leaf of $T_i$ where $C_i$ has no outgoing edge taken into the matching. We consider $C_i$ as a graph of the odd-cardinality and use Proposition 2 to get a polynomial-size splitting tree $T_{C_i}$. We attach $T_{C_i}$ to the free leaf of tree $T_i$ forming $T_\ell$.

## 5 Open question

Tight bounds on splitting trees for perfect matching is still an open question. Itsykson and Sokolov provided polynomial-size splitting trees for graphs on odd number of vertices. We have just proved, that formula built on an arbitrary graph has a splitting tree of size $2^{O(n)}$. It is an interesting question if the formula $PMP_G$ has exponential lower bounds for arbitrary graphs or even such case can be solved with polynomial-size splitting trees.


## Acknowledgements

The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University. The author is grateful to Dmitry Itsykson for fruitful discussions and to Sam Buss for valuable advices that improve readability of the paper.


## References


1. D. Itsykson and D. Sokolov, *Mathematical Foundations of Computer Science 2014: 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, ch. Lower Bounds for Splittings by Linear Combinations, pp. 372–383. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.
2. M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, pp. 394–397, 1962.
3. M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, pp. 201–215, 1960.
4. G. S. Tseitin, "On the complexity of derivation in the propositional calculus," *Zapiski nauchnykh seminarov LOMI*, vol. 8, pp. 234–259, 1968. English translation of this volume: Consultants Bureau, N.Y., 1970, pp. 115–125.
5. A. Urquhart, "Hard examples for resolution," *JACM*, vol. 34, no. 1, pp. 209–219, 1987.
6. M. Alekhnovich, E. A. Hirsch, and D. Itsykson, "Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas," *J. Autom. Reason.*, vol. 35, no. 1-3, pp. 51–72, 2005.
7. K. Seto and S. Tamaki, "A satisfiability algorithm and average-case hardness for formulas over the full binary basis," in *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pp. 107–116, June 2012.
8. E. Demenkov and A. S. Kulikov, *Mathematical Foundations of Computer Science 2011: 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, ch. An Elementary Proof of a 3n - o(n) Lower Bound on the Circuit Complexity of Affine Dispersers, pp. 256–265. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.
9. S. Dantchev and S. Riis, "Tree resolution proofs of the weak pigeon-hole principle," in *Computational Complexity, 16th Annual IEEE Conference on, 2001.*, pp. 69–75, 2001.