# Hard satisfiable formulas for splittings by linear combinations

**Authors:**
Dmitry Itsykson, Alexander Knop

**Institute:**
UC San Diego

# DPLL **algorithms**

DPLL algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.

# DPLL algorithms

DPLL algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a partial substitution to the variables $x_1$, ..., $x_n$.

# DPLL algorithms

DPLL algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a partial substitution to the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

# DPLL algorithms

DPLL algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a partial substitution to the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return "no";

# DPLL **algorithms**

DPLL algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.

Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a partial substitution to the variables $x_1$, ..., $x_n$.

Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return “no”;
- if $\rho$ satisfies $\varphi$, then return “yes”;

# DPLL algorithms

DPLL algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a partial substitution to the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return “no”;
- if $\rho$ satisfies $\varphi$, then return “yes”;
- choose a variable $x_i$ using $A$ and choose a Boolean value $b \in \{0, 1\}$ using $B$;

# DPLL algorithms

DPLL algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a partial substitution to the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return "no";
- if $\rho$ satisfies $\varphi$, then return "yes";
- choose a variable $x_i$ using $A$ and choose a Boolean value $b \in \{0, 1\}$ using $B$;
- If $\mathcal{D}_{A,B}(\varphi, \rho \cup \{x_i = b\}) =$ "yes", then return "yes";

# DPLL algorithms

DPLL algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \dots, x_n)$ be a formula in CNF and $\rho$ be a partial substitution to the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return "no";
- if $\rho$ satisfies $\varphi$, then return "yes";
- choose a variable $x_i$ using $A$ and choose a Boolean value $b \in \{0, 1\}$ using $B$;
- If $\mathcal{D}_{A,B}(\varphi, \rho \cup \{x_i = b\}) =$ "yes", then return "yes";
- return $\mathcal{D}_{A,B}(\varphi, \rho \cup \{x_i = 1 - b\})$.

# DPLL algorithms and unsatisfiable formulas

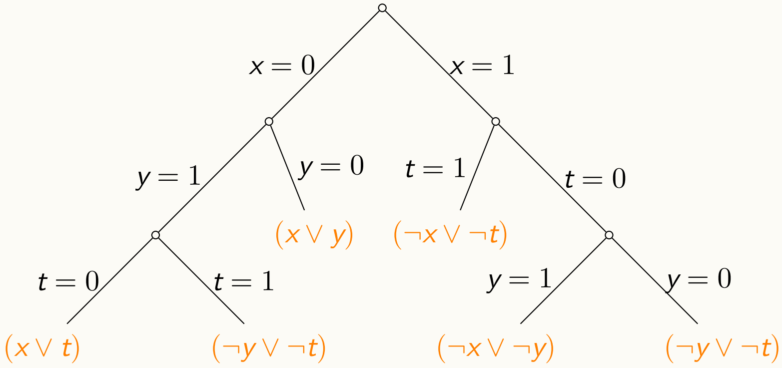
Let us run some DPLL algorithm on

$$(x \vee y) \wedge (\neg x \vee \neg y) \wedge (\neg y \vee t) \wedge (y \vee \neg t) \wedge (x \vee t) \wedge (\neg x \vee \neg t).$$

# DPLL algorithms and unsatisfiable formulas

Let us run some DPLL algorithm on

$$(x \vee y) \wedge (\neg x \vee \neg y) \wedge (\neg y \vee t) \wedge (y \vee \neg t) \wedge (x \vee t) \wedge (\neg x \vee \neg t).$$



# Complexity of unsatisfiable formulas

Because of the connections with resolution, the lower bounds on unsatisfiable instances are relatively easy.

# Complexity of unsatisfiable formulas

Because of the connections with resolution, the lower bounds on unsatisfiable instances are relatively easy.

**THEOREM**

*Tseitin formulas are exponentially hard for* DPLL

# Complexity of unsatisfiable formulas

Because of the connections with resolution, the lower bounds on unsatisfiable instances are relatively easy.

**THEOREM**

*Tseitin formulas are exponentially hard for* DPLL, *i.e., unsatisfiable systems of linear equations are hard for* DPLL.

# Complexity of satisfiable formulas

Satisfiable formulas are more interesting and easier for solvers.

# Complexity of satisfiable formulas

Satisfiable formulas are more interesting and easier for solvers.
If $\mathbf{P} = \mathbf{NP}$, then there are no superpolynomial lower bounds for DPLL algorithms since heuristic $B$ may choose the correct value.

# Complexity of satisfiable formulas

Satisfiable formulas are more interesting and easier for solvers.
If $\mathbf{P} = \mathbf{NP}$, then there are no superpolynomial lower bounds for DPLL algorithms since heuristic $B$ may choose the correct value.

**DEFINITION**

We say that $\mathcal{D}_{A,B}$ is myopic iff $A$ and $B$ can read $K = n^{1-\varepsilon}$ clauses precisely but they see other clauses without negations and can query the number of occurrences of literals.

# Complexity of satisfiable formulas

Satisfiable formulas are more interesting and easier for solvers.
If $\mathbf{P} = \mathbf{NP}$, then there are no superpolynomial lower bounds for DPLL algorithms since heuristic $B$ may choose the correct value.

**DEFINITION**

We say that $\mathcal{D}_{A,B}$ is myopic iff $A$ and $B$ can read $K = n^{1-\varepsilon}$ clauses precisely but they see other clauses without negations and can query the number of occurrences of literals.

**DEFINITION**

We say that $A$ and $B$ are drunken iff $B$ chooses the value $1$ with probability $\frac{1}{2}$.

# Complexity of satisfiable formulas

Satisfiable formulas are more interesting and easier for solvers.
If $\mathbf{P} = \mathbf{NP}$, then there are no superpolynomial lower bounds for DPLL algorithms since heuristic $B$ may choose the correct value.

**DEFINITION**

We say that $\mathcal{D}_{A,B}$ is myopic iff $A$ and $B$ can read $K = n^{1-\varepsilon}$ clauses precisely but they see other clauses without negations and can query the number of occurrences of literals.

**DEFINITION**

We say that $A$ and $B$ are drunken iff $B$ chooses the value $1$ with probability $\frac{1}{2}$.

**THEOREM (ALEKHNOVICH, HIRSCH, AND ITSYKSON, 2005)**

*Satisfiable linear systems are hard for myopic and drunken* DPLL *algorithms.*

# DPLL($\oplus$) **algorithms**

DPLL($\oplus$) algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.

# DPLL($\oplus$) **algorithms**

DPLL($\oplus$) algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.

Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a system of linear equations over the variables $x_1$, ..., $x_n$.

# DPLL($\oplus$) **algorithms**

DPLL($\oplus$) algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a system of linear equations over the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

# DPLL($\oplus$) **algorithms**

DPLL($\oplus$) algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.

Let $\varphi(x_1, \dots, x_n)$ be a formula in CNF and $\rho$ be a system of linear equations over the variables $x_1$, ..., $x_n$.

Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return "no" ($\rho$ contradicts to a clause $x_{i_1} = \sigma_1 \vee \dots \vee x_{i_k} = \sigma_k$ iff $\rho \wedge (x_{i_j} = \sigma_j)$ is unsatisfiable for all $j \in [k]$);

# DPLL($\oplus$) algorithms

DPLL($\oplus$) algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a system of linear equations over the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return “no” ($\rho$ contradicts to a clause $x_{i_1} = \sigma_1 \vee \cdots \vee x_{i_k} = \sigma_k$ iff $\rho \wedge (x_{i_j} = \sigma_j)$ is unsatisfiable for all $j \in [k]$);
- if $\rho$ has only one solution which satisfies $\varphi$, then return “yes”;

# DPLL($\oplus$) algorithms

DPLL($\oplus$) algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a system of linear equations over the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return "no" ($\rho$ contradicts to a clause $x_{i_1} = \sigma_1 \vee \cdots \vee x_{i_k} = \sigma_k$ iff $\rho \wedge (x_{i_j} = \sigma_j)$ is unsatisfiable for all $j \in [k]$);
- if $\rho$ has only one solution which satisfies $\varphi$, then return "yes";
- choose a linear combination $\ell$ of variables $x_1$, ..., $x_n$ using $A$ and choose a Boolean value $b \in \{0, 1\}$ using $B$;

# DPLL($\oplus$) algorithms

DPLL($\oplus$) algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a system of linear equations over the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return "no" ($\rho$ contradicts to a clause $x_{i_1} = \sigma_1 \vee \cdots \vee x_{i_k} = \sigma_k$ iff $\rho \wedge (x_{i_j} = \sigma_j)$ is unsatisfiable for all $j \in [k]$);
- if $\rho$ has only one solution which satisfies $\varphi$, then return "yes";
- choose a linear combination $\ell$ of variables $x_1$, ..., $x_n$ using $A$ and choose a Boolean value $b \in \{0, 1\}$ using $B$;
- If $\mathcal{D}_{A,B}(\varphi, \rho \cup \{\ell = b\}) =$ "yes", then return "yes";

# DPLL($\oplus$) **algorithms**

DPLL($\oplus$) algorithm is an algorithm $\mathcal{D}_{A,B}$ parametrized by two heuristics $A$ and $B$.
Let $\varphi(x_1, \ldots, x_n)$ be a formula in CNF and $\rho$ be a system of linear equations over the variables $x_1$, ..., $x_n$.
Then $\mathcal{D}_{A,B}(\varphi, \rho)$ determines if $\varphi \wedge \rho$ is satisfiable and works as follows.

- if $\rho$ contradicts to a clause of $\varphi$, then return "no" ($\rho$ contradicts to a clause $x_{i_1} = \sigma_1 \vee \cdots \vee x_{i_k} = \sigma_k$ iff $\rho \wedge (x_{i_j} = \sigma_j)$ is unsatisfiable for all $j \in [k]$);
- if $\rho$ has only one solution which satisfies $\varphi$, then return "yes";
- choose a linear combination $\ell$ of variables $x_1$, ..., $x_n$ using $A$ and choose a Boolean value $b \in \{0, 1\}$ using $B$;
- If $\mathcal{D}_{A,B}(\varphi, \rho \cup \{\ell = b\}) =$ "yes", then return "yes";
- return $\mathcal{D}_{A,B}(\varphi, \rho \cup \{\ell = 1 - b\})$.

# DPLL($\oplus$) **algorithms and unsatisfiable formulas**

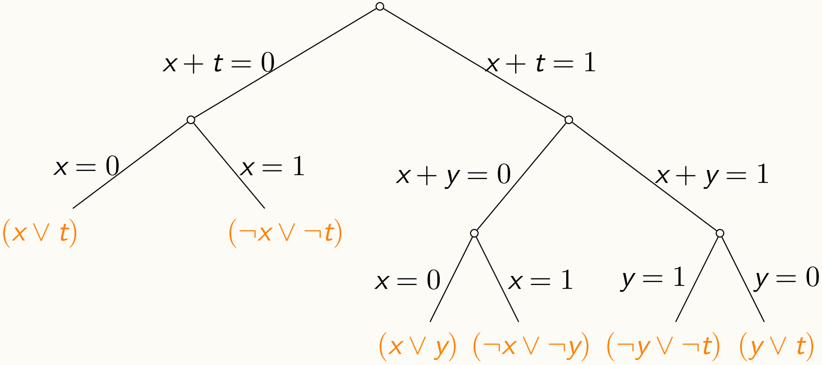
Let us run some DPLL($\oplus$) algorithm on

$$(x \vee y) \wedge (\neg x \vee \neg y) \wedge (\neg y \vee t) \wedge (y \vee \neg t) \wedge (x \vee t) \wedge (\neg x \vee \neg t).$$

# DPLL($\oplus$) **algorithms and unsatisfiable formulas**

Let us run some DPLL($\oplus$) algorithm on

$$(x \vee y) \wedge (\neg x \vee \neg y) \wedge (\neg y \vee t) \wedge (y \vee \neg t) \wedge (x \vee t) \wedge (\neg x \vee \neg t).$$



# Complexity of unsatisfiable formulas

| Formula | DPLL | Res | DPLL($\oplus$) |
|---|---|---|---|
| $\mathbb{F}_2$-linear systems | hard | hard | easy<br>[Itsykson and Sokolov 2014] |
| Perfect matching in $K_{2n+1}$ | $2^{\Theta(n \log n)}$ | $2^{\Theta(n)}$ | poly(n)<br>[Itsykson and Sokolov 2014] |
| $\mathrm{PHP}^n_{n+1}$ | $2^{\Theta(n \log n)}$ | $2^{\Theta(n)}$ | $2^{\Theta(n)}$<br>[Itsykson and Sokolov 2014]<br>[Oparin 2016] |
| $\mathrm{TS}^{\wedge}_{G,c}$ | $2^{\Theta(n)}$ | $2^{\Theta(n)}$ | $2^{\Omega(n^{\epsilon})}$<br>[Itsykson and Sokolov 2014] |
| Random 3-CNF | $2^{\Theta(n)}$ | $2^{\Theta(n)}$ | $2^{\Theta(n)}$<br>[Garlik and Kolodziejczyk 2017] |
| Lifted Pebbling | $2^{\Omega(n/\log n)}$ | $poly(n)$ | $2^{\Omega(n/\log n)}$<br>[Itsykson and Sokolov 2017] |

# Complexity of satisfiable formulas

**THEOREM**

*There exists an explicit family of satisfiable CNF formulas $\Psi_n$ such that any drunken* DPLL($\oplus$) *runs on $\Psi_n$ at least $2^{\Omega(n)}$ steps with probability at least $1 - 2^{-\Omega(n)}$.*

# Complexity of satisfiable formulas

**THEOREM**

*There exists an explicit family of satisfiable CNF formulas $\Psi_n$ such that any drunken* DPLL *runs on $\Psi_n$ at least $2^{\Omega(n)}$ steps with probability at least $1 - 2^{-\Omega(n)}$.*

# Complexity of satisfiable formulas

## THEOREM

*There exists an explicit family of satisfiable CNF formulas* $\Psi_n$ *such that any drunken* DPLL *runs on* $\Psi_n$ *at least* $2^{\Omega(n)}$ *steps with probability at least* $1 - 2^{-\Omega(n)}$.

## PLAN OF THE PROOF

- $\Psi_n$ is $\mathtt{PHP}^n_{n+1}$ plus one satisfying assignment;
- Prove that w.h.p. a drunken DPLL will make an incorrect substitution;
- Adopt the lower bound technique for $\mathtt{PHP}^n_{n+1}$.

# The Prover-delayer Game

Let $\varphi(x_1, x_2, \ldots, x_n)$ be a CNF formula; Prover and Delayer are playing the following game.

- Prover asks for the value of $x_i$ for some $i \in [n]$;
- Delayer gives an answer from $\{0, 1\}$ or "Choose any"; In the case of "Choose any" Prover chooses the value from $\{0, 1\}$
- Delayer earns $1$ coin for every answer "Choose any";
- The game ends if the current substitution contradicts some clause of $\varphi$.

# The Prover-delayer Game

Let $\varphi(x_1, x_2, \dots, x_n)$ be a CNF formula; Prover and Delayer are playing the following game.

- Prover asks for the value of $x_i$ for some $i \in [n]$;
- Delayer gives an answer from $\{0, 1\}$ or "Choose any"; In the case of "Choose any" Prover chooses the value from $\{0, 1\}$
- Delayer earns $1$ coin for every answer "Choose any";
- The game ends if the current substitution contradicts some clause of $\varphi$.

**THEOREM (CF. PUDLAK AND IMPAGLIAZZO, 2001)**

*If there is a strategy for Delayer such that for every Prover's strategy, Delayer earns at least $t$ coins, then the size of any decision tree for $\varphi$ is at least $2^t$.*

# The Pigeonhole Principle

Let us recall the definition of the pigeonhole principle formula ($\mathrm{PHP}^n_{n+1}$).

# The Pigeonhole Principle

Let us recall the definition of the pigeonhole principle formula ($\mathrm{PHP}_{n+1}^{n}$). The formula states that it is impossible to put $n+1$ pigeons into $n$ holes.

- The formulas has $n(n+1)$ variables $\mathfrak{P}_n = \{p_{i,j} \ : \ i \in [n+1], j \in [n]\}$ (informally, $p_{i,j} = 1$ iff the $i$th pigeon is in the $j$th hole).

# The Pigeonhole Principle

Let us recall the definition of the pigeonhole principle formula ($\mathrm{PHP}^n_{n+1}$). The formula states that it is impossible to put $n+1$ pigeons into $n$ holes.

- The formulas has $n(n+1)$ variables $\mathfrak{P}_n = \{p_{i,j} \ : \ i \in [n+1], j \in [n]\}$ (informally, $p_{i,j} = 1$ iff the $i$th pigeon is in the $j$th hole).
- Long clauses: $p_{i,1} \vee p_{i,2} \cdots \vee p_{i,n}$ for all $i \in [n+1]$

# The Pigeonhole Principle

Let us recall the definition of the pigeonhole principle formula ($\mathtt{PHP}^n_{n+1}$). The formula states that it is impossible to put $n+1$ pigeons into $n$ holes.

- The formulas has $n(n+1)$ variables $\mathfrak{P}_n = \{p_{i,j} \;:\; i \in [n+1], j \in [n]\}$ (informally, $p_{i,j} = 1$ iff the $i$th pigeon is in the $j$th hole).
- Long clauses: $p_{i,1} \vee p_{i,2} \cdots \vee p_{i,n}$ for all $i \in [n+1]$
- Short clauses: $\neg p_{i,k} \vee \neg p_{j,k}$ for all $i \neq j \in [n+1]$ and $k \in [n]$.

# The Pigeonhole Principle

Let us recall the definition of the pigeonhole principle formula ($\mathrm{PHP}^n_{n+1}$). The formula states that it is impossible to put $n+1$ pigeons into $n$ holes.

- The formulas has $n(n+1)$ variables $\mathfrak{P}_n = \{p_{i,j} \; : \; i \in [n+1], j \in [n]\}$ (informally, $p_{i,j} = 1$ iff the $i$th pigeon is in the $j$th hole).
- Long clauses: $p_{i,1} \vee p_{i,2} \cdots \vee p_{i,n}$ for all $i \in [n+1]$
- Short clauses: $\neg p_{i,k} \vee \neg p_{j,k}$ for all $i \neq j \in [n+1]$ and $k \in [n]$.

- We say that a substitution $\pi$ is proper if it satisfies all the short clauses.
- $\pi$ properly implies $\rho$ if any proper assignment that satisfy $\pi$ also satisfies $\rho$;
- A proper rank of a substitution is the minimal number of equalities that properly imply all the other equalities.

# A Lower Bound on Unsatisfiable Instances

**LEMMA**

*Let a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $n-1$ and can be extended to a proper substitution. Then for all $i \in [n+1]$ there is a proper solution that satisfies $p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$.*

# A Lower Bound on Unsatisfiable Instances

**LEMMA**

*Let a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $n-1$ and can be extended to a proper substitution. Then for all $i \in [n+1]$ there is a proper solution that satisfies $p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$.*

**THEOREM**

*If a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $\frac{n-1}{2}$ and can be extended to a proper substitution, then any decision tree for $\mathtt{PHP}^n_{n+1} \wedge \pi$ has size at least $2^{\frac{n-1}{2}}$*

# A Lower Bound on Unsatisfiable Instances

**LEMMA**

*Let a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $n-1$ and can be extended to a proper substitution. Then for all $i \in [n+1]$ there is a proper solution that satisfies $p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$.*

**THEOREM**

*If a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $\frac{n-1}{2}$ and can be extended to a proper substitution, then any decision tree for $\mathrm{PHP}_{n+1}^{n} \wedge \pi$ has size at least $2^{\frac{n-1}{2}}$*

**PROOF.**

Strategy of Delayer is the following: if the value of $x_i$ is properly implied from the current substitution, then return it, otherwise, return "Choose any".

# A Lower Bound on Unsatisfiable Instances

## LEMMA

*Let a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $n-1$ and can be extended to a proper substitution. Then for all $i \in [n+1]$ there is a proper solution that satisfies $p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$.*

## THEOREM

*If a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $\frac{n-1}{2}$ and can be extended to a proper substitution, then any decision tree for $\mathtt{PHP}^n_{n+1} \wedge \pi$ has size at least $2^{\frac{n-1}{2}}$*

## PROOF.

Strategy of Delayer is the following: if the value of $x_i$ is properly implied from the current substitution, then return it, otherwise, return "Choose any". "Choose any" may increase the proper rank by at most $1$. When the game ends the current substitution contradicts to all the long clauses.

# A Lower Bound on Unsatisfiable Instances

## LEMMA

*Let a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $n-1$ and can be extended to a proper substitution. Then for all $i \in [n+1]$ there is a proper solution that satisfies $p_{i,1} \lor p_{i,2} \lor \dots \lor p_{i,n}$.*

## THEOREM

*If a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank at most $\frac{n-1}{2}$ and can be extended to a proper substitution, then any decision tree for $\mathrm{PHP}^n_{n+1} \land \pi$ has size at least $2^{\frac{n-1}{2}}$*

## PROOF.

Strategy of Delayer is the following: if the value of $x_i$ is properly implied from the current substitution, then return it, otherwise, return "Choose any".
"Choose any" may increase the proper rank by at most $1$. When the game ends the current substitution contradicts to all the long clauses.
Hence, Delayer earns at least $\frac{n-1}{2}$ coins. □

# Hard Satisfiable Formula

Let $\Phi = \bigwedge_{i \in I} C_i$ be a CNF formula on the variables $x_1$, ..., $x_n$.

# Hard Satisfiable Formula

Let $\Phi = \bigwedge_{i \in I} C_i$ be a CNF formula on the variables $x_1$, ..., $x_n$.

$\Phi + \sigma = \bigwedge_{i \in I, j \in [n]} C_i \vee x_j^{\sigma(x_j)}$, where $\sigma$ is an assignment, $x^0$ denotes $\neg x$, and $x^1$ denotes $x$.

# Hard Satisfiable Formula

Let $\Phi = \bigwedge_{i \in I} C_i$ be a CNF formula on the variables $x_1$, ..., $x_n$.

$\Phi + \sigma = \bigwedge_{i \in I, j \in [n]} C_i \vee x_j^{\sigma(x_j)}$, where $\sigma$ is an assignment, $x^0$ denotes $\neg x$, and $x^1$ denotes $x$.

**LEMMA**

*If $\Phi$ is unsatisfiable, then $\sigma$ is the only satisfying assignment of $\Phi + \sigma$.*

# Hard Satisfiable Formula

Let $\Phi = \bigwedge_{i \in I} C_i$ be a CNF formula on the variables $x_1$, ..., $x_n$.

$\Phi + \sigma = \bigwedge_{i \in I, j \in [n]} C_i \vee x_j^{\sigma(x_j)}$, where $\sigma$ is an assignment, $x^0$ denotes $\neg x$, and $x^1$ denotes $x$.

**LEMMA**

*If $\Phi$ is unsatisfiable, then $\sigma$ is the only satisfying assignment of $\Phi + \sigma$.*

**THEOREM**

*If $\sigma$ is a proper assignment, then $\mathtt{PHP}_{n+1}^{n} + \sigma$ is hard for drunken* DPLL *algorithms.*

# Hard Satisfiable Formula

**THEOREM**

*If* $\sigma$ *is a proper assignment, then* $\mathtt{PHP}^n_{n+1} + \sigma$ *is hard for drunken* DPLL *algorithms.*

# Hard Satisfiable Formula

**THEOREM**

*If $\sigma$ is a proper assignment, then $\mathtt{PHP}_{n+1}^{n} + \sigma$ is hard for drunken* DPLL *algorithms.*

**LEMMA**

*Let a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank $k \leq n - 1$ and can be extended to a proper substitution. Then there are at least two proper extensions of $\pi$.*

# Hard Satisfiable Formula

**THEOREM**

*If $\sigma$ is a proper assignment, then $\mathrm{PHP}_{n+1}^{n} + \sigma$ is hard for drunken* DPLL *algorithms.*

**LEMMA**

*Let a substitution $\pi$ to the variables $\mathfrak{P}_n$ has a proper rank $k \leq n - 1$ and can be extended to a proper substitution. Then there are at least two proper extensions of $\pi$.*

**PROOF.**

Consider the moment when the solution is found, the current substitution has proper rank at least $n - 1$. □

# Hard Satisfiable Formula

**THEOREM**

*If $\sigma$ is a proper assignment, then $\mathtt{PHP}^n_{n+1} + \sigma$ is hard for drunken* DPLL *algorithms.*

**PROOF.**

Consider the moment when the solution is found, the current substitution has proper rank at least $n - 1$. Consider the moments on the acceptance branch when the proper rank grows $0 \to 1$, $1 \to 2$, ..., $\frac{n-1}{2} - 1 \to \frac{n-1}{2}$.

# Hard Satisfiable Formula

**THEOREM**

*If $\sigma$ is a proper assignment, then $\mathtt{PHP}^n_{n+1} + \sigma$ is hard for drunken* DPLL *algorithms.*

**PROOF.**

Consider the moment when the solution is found, the current substitution has proper rank at least $n-1$. Consider the moments on the acceptance branch when the proper rank grows $0 \to 1$, $1 \to 2$, ..., $\frac{n-1}{2} - 1 \to \frac{n-1}{2}$.
Note that the probability that the algorithm deviates from the acceptance path in one of these moments is $1 - 2^{-\frac{n-1}{2}}$.

# Hard Satisfiable Formula

**THEOREM**

*If $\sigma$ is a proper assignment, then $\mathtt{PHP}^n_{n+1} + \sigma$ is hard for drunken* DPLL *algorithms.*

**PROOF.**

Consider the moment when the solution is found, the current substitution has proper rank at least $n-1$. Consider the moments on the acceptance branch when the proper rank grows $0 \to 1$, $1 \to 2$, ..., $\frac{n-1}{2} - 1 \to \frac{n-1}{2}$.
Note that the probability that the algorithm deviates from the acceptance path in one of these moments is $1 - 2^{-\frac{n-1}{2}}$. After the deviation: $(\mathtt{PHP}^n_{n+1} + \sigma) \wedge \pi$ is unsatisfiable, $\pi$ can be extended to a proper substitution and has a proper rank $\frac{n-1}{2}$.

# Hard Satisfiable Formula

**THEOREM**

*If $\sigma$ is a proper assignment, then $\mathtt{PHP}^n_{n+1} + \sigma$ is hard for drunken* DPLL *algorithms.*

**PROOF.**

Consider the moment when the solution is found, the current substitution has proper rank at least $n-1$. Consider the moments on the acceptance branch when the proper rank grows $0 \to 1$, $1 \to 2$, ..., $\frac{n-1}{2} - 1 \to \frac{n-1}{2}$.
Note that the probability that the algorithm deviates from the acceptance path in one of these moments is $1 - 2^{-\frac{n-1}{2}}$. After the deviation: $(\mathtt{PHP}^n_{n+1} + \sigma) \wedge \pi$ is unsatisfiable, $\pi$ can be extended to a proper substitution and has a proper rank $\frac{n-1}{2}$.
Decision tree for $(\mathtt{PHP}^n_{n+1} + \sigma) \wedge \pi$ also a Decision tree for $\mathtt{PHP}^n_{n+1} \wedge \pi$.

# Hard Satisfiable Formula

**THEOREM**

*If $\sigma$ is a proper assignment, then $\mathtt{PHP}^n_{n+1} + \sigma$ is hard for drunken* DPLL *algorithms.*

**PROOF.**

Consider the moment when the solution is found, the current substitution has proper rank at least $n-1$. Consider the moments on the acceptance branch when the proper rank grows $0 \to 1$, $1 \to 2$, ..., $\frac{n-1}{2} - 1 \to \frac{n-1}{2}$.
Note that the probability that the algorithm deviates from the acceptance path in one of these moments is $1 - 2^{-\frac{n-1}{2}}$. After the deviation: $(\mathtt{PHP}^n_{n+1} + \sigma) \wedge \pi$ is unsatisfiable, $\pi$ can be extended to a proper substitution and has a proper rank $\frac{n-1}{2}$.
Decision tree for $(\mathtt{PHP}^n_{n+1} + \sigma) \wedge \pi$ also a Decision tree for $\mathtt{PHP}^n_{n+1} \wedge \pi$. And hence it has size at least $2^{\frac{n-1}{2}}$. □

# Open Questions

1. Good models of CDCL algorithms.

# Open Questions

1. Good models of CDCL algorithms.
2. Lower bounds for DPLL($\oplus$) algorithms for satisfiable $O(1)$-CNF formulas.

# Open Questions

1. Good models of CDCL algorithms.
2. Lower bounds for $\mathsf{DPLL}(\oplus)$ algorithms for satisfiable $O(1)$-CNF formulas. Or even SETH lower bounds on $\mathsf{DPLL}(\oplus)$ algorithms.
3. Lower bounds for myopic $\mathsf{DPLL}(\oplus)$ algorithms.

# Open Questions

1. Good models of CDCL algorithms.
2. Lower bounds for DPLL($\oplus$) algorithms for satisfiable $O(1)$-CNF formulas. Or even SETH lower bounds on DPLL($\oplus$) algorithms.
3. Lower bounds for myopic DPLL($\oplus$) algorithms.
4. Lower bounds for Res($\oplus$), the resolution that operates with disjunctions of linear equations.

## Open Questions

1. Good models of CDCL algorithms.
2. Lower bounds for DPLL($\oplus$) algorithms for satisfiable $O(1)$-CNF formulas. Or even SETH lower bounds on DPLL($\oplus$) algorithms.
3. Lower bounds for myopic DPLL($\oplus$) algorithms.
4. Lower bounds for Res($\oplus$), the resolution that operates with disjunctions of linear equations.
5. DPLL($\oplus$) or even CDCL($\oplus$) solvers working well on the industrial instances.