# A #SAT Algorithm for Small Constant-Depth Circuits with PTF gates.

Nutan Limaye
Computer Science and Engineering Department,
Indian Institute of Technology, Bombay, (IITB) India.
Joint work with

Swapnam Bajpai, Vaibhav Krishan, Deepanshu Kush, and Srikanth Srinivasan.
IIT Bombay, India.

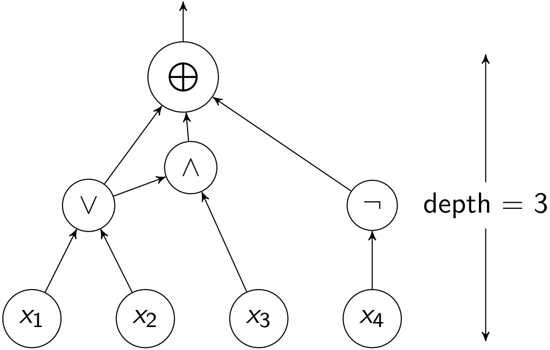
Complexity, Algorithms, Automata and Logic Meet (CAALM 2019)
Chennai Mathematical Institute, January 2019.

# Circuit satisfiability algorithms

## Boolean circuits

# Circuit satisfiability algorithms

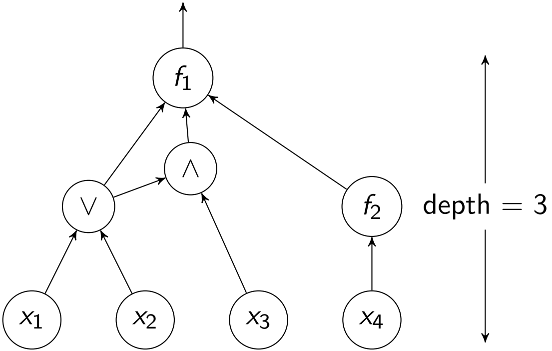
## Boolean circuits




Size = number of gates = 4

# Circuit satisfiability algorithms

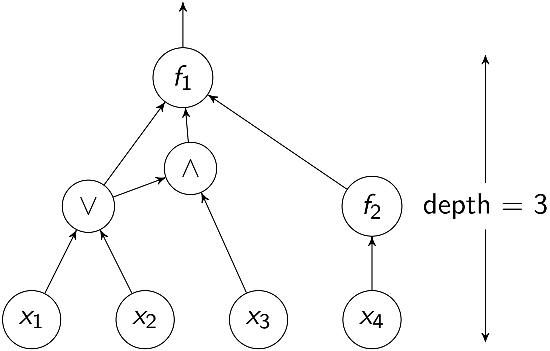
## Boolean circuits




Size = number of gates = 4

# Circuit satisfiability algorithms

## Boolean circuits

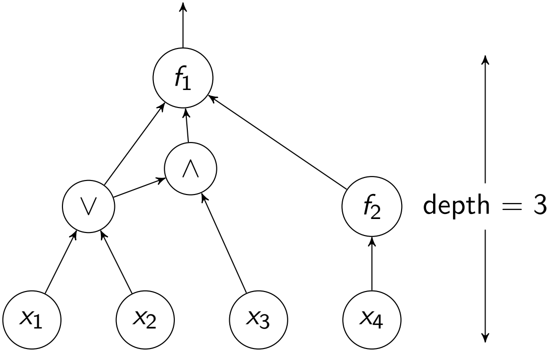



Size = number of gates = 4

Number of input variables: $n$

# Circuit satisfiability algorithms

## Boolean circuits




Size = number of gates = 4

Number of input variables: $n$

Constant-depth circuits: $d$ is independent of $n$.

# Our focus

Task:

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

Given: $C \in \mathcal{C}$

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

Given: $C \in \mathcal{C}$ computing $f : \{-1, 1\}^n \to \{-1, 1\}$

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

Given: $C \in \mathcal{C}$ computing $f : \{-1,1\}^n \to \{-1,1\}$

Count: $\#\{a \in \{-1,1\}^n \mid f(a) = -1\}$

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

Given: $C \in \mathcal{C}$ computing $f : \{-1,1\}^n \to \{-1,1\}$

Count: $\#\{a \in \{-1,1\}^n \mid f(a) = -1\}$

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

Given: $C \in \mathcal{C}$ computing $f : \{-1,1\}^n \to \{-1,1\}$

Count: $\#\{a \in \{-1,1\}^n \mid f(a) = -1\}$

Here $-1$ stands for True and $1$ stands for False.

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

Given: $C \in \mathcal{C}$ computing $f : \{-1, 1\}^n \to \{-1, 1\}$

Count: $\#\{a \in \{-1, 1\}^n \mid f(a) = -1\}$

Here $-1$ stands for True and $1$ stands for False.

Trivial brute-force algorithm exists.

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

Given: $C \in \mathcal{C}$ computing $f : \{-1,1\}^n \to \{-1,1\}$

Count: $\#\{a \in \{-1,1\}^n \mid f(a) = -1\}$

Here $-1$ stands for True and $1$ stands for False.

Trivial brute-force algorithm exists. It takes time $\mathrm{poly}(|C|) \cdot 2^n$.

# Our focus

Task: Fix a class of circuits $\mathcal{C}$.

#SAT(C)

Given: $C \in \mathcal{C}$ computing $f : \{-1, 1\}^n \to \{-1, 1\}$

Count: $\#\{a \in \{-1, 1\}^n \mid f(a) = -1\}$

Here $-1$ stands for True and $1$ stands for False.

Trivial brute-force algorithm exists. It takes time $\mathrm{poly}(|C|) \cdot 2^n$.

Can we design an algorithm that takes time $2^n / n^{\omega(1)}$ when $|C|$ is small, say $\mathrm{poly}(n)$?

# Circuit satisfibaility algorithms

Connections to circuit lower bounds

# Circuit satisfibaility algorithms

## Connections to circuit lower bounds

Better than brute-force circuit-satisfiability algorithms for a class $\mathcal{C}$ reveals some weaknesses of functions computable by $\mathcal{C}$.

# Circuit satisfibaility algorithms

## Connections to circuit lower bounds

Better than brute-force circuit-satisfiability algorithms for a class $\mathcal{C}$ reveals some weaknesses of functions computable by $\mathcal{C}$.

This intuitive connection has been formalised to derive lowerbounds for various interesting classes of circuits.

[Paturi, Pudlák, Zane 1997],[Paturi, Pudlák, Saks, Zane 2005], [Williams 2010], [Williams 2011].

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

A function $f : \{-1, 1\}^n \to \{-1, 1\}$ is called a degree-$k$ Polynomial Threshold Function ($k$-PTF) if there is a multilinear degree-$k$ polynomial

$$P(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| \leq k} \alpha_S x_S$$

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

A function $f : \{-1, 1\}^n \to \{-1, 1\}$ is called a degree-$k$ Polynomial Threshold Function ($k$-PTF) if there is a multilinear degree-$k$ polynomial

$$P(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| \leq k} \alpha_S x_S$$

where $X_S = \prod_{i \in S} x_i$

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

A function $f : \{-1, 1\}^n \to \{-1, 1\}$ is called a degree-$k$ Polynomial Threshold Function ($k$-PTF) if there is a multilinear degree-$k$ polynomial

$$P(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| \leq k} \alpha_S x_S$$

where $X_S = \prod_{i \in S} x_i$, $P(X) \in \mathbb{R}[X]$ such that

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

A function $f : \{-1,1\}^n \to \{-1,1\}$ is called a degree-$k$ Polynomial Threshold Function ($k$-PTF) if there is a multilinear degree-$k$ polynomial

$$P(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| \leq k} \alpha_S x_S$$

where $X_S = \prod_{i \in S} x_i$, $P(X) \in \mathbb{R}[X]$ such that $\text{sign}(P(a)) = f(a)$ for every $a \in \{-1,1\}^n$.

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

A function $f : \{-1,1\}^n \to \{-1,1\}$ is called a degree-$k$ Polynomial Threshold Function ($k$-PTF) if there is a multilinear degree-$k$ polynomial

$$P(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| \leq k} \alpha_S x_S$$

where $X_S = \prod_{i \in S} x_i$, $P(X) \in \mathbb{R}[X]$ such that $\mathrm{sign}(P(a)) = f(a)$ for every $a \in \{-1,1\}^n$.

We assume that $P(a) \neq 0$ for each $a \in \{-1,1\}^n$.

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

A function $f : \{-1,1\}^n \to \{-1,1\}$ is called a degree-$k$ Polynomial Threshold Function ($k$-PTF) if there is a multilinear degree-$k$ polynomial

$$P(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| \leq k} \alpha_S x_S$$

where $X_S = \prod_{i \in S} x_i$, $P(X) \in \mathbb{R}[X]$ such that $\text{sign}(P(a)) = f(a)$ for every $a \in \{-1,1\}^n$.

We assume that $P(a) \neq 0$ for each $a \in \{-1,1\}^n$.

Let $w(P)$ denote the bit-complexity of $\sum_{S \subseteq [n], |S| \leq k} |\alpha_S|$.

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

A function $f : \{-1, 1\}^n \to \{-1, 1\}$ is called a degree-$k$ Polynomial Threshold Function ($k$-PTF) if there is a multilinear degree-$k$ polynomial

$$P(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| \leq k} \alpha_S x_S$$

where $X_S = \prod_{i \in S} x_i$, $P(X) \in \mathbb{R}[X]$ such that $\text{sign}(P(a)) = f(a)$ for every $a \in \{-1, 1\}^n$.

We assume that $P(a) \neq 0$ for each $a \in \{-1, 1\}^n$.

Let $w(P)$ denote the bit-complexity of $\sum_{S \subseteq [n], |S| \leq k} |\alpha_S|$.

Example: $\text{AND}(x_1, x_2) = \text{sign}(x_1 + x_2 + 1)$

# Polynomial Threshold Functions

## Definition (Polynomial Threshold Functions)

Let $X = \{x_1, \ldots, x_n\}$.

A function $f : \{-1, 1\}^n \to \{-1, 1\}$ is called a degree-$k$ Polynomial Threshold Function ($k$-PTF) if there is a multilinear degree-$k$ polynomial

$$P(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| \leq k} \alpha_S x_S$$

where $X_S = \prod_{i \in S} x_i$, $P(X) \in \mathbb{R}[X]$ such that $\text{sign}(P(a)) = f(a)$ for every $a \in \{-1, 1\}^n$.

We assume that $P(a) \neq 0$ for each $a \in \{-1, 1\}^n$.

Let $w(P)$ denote the bit-complexity of $\sum_{S \subseteq [n], |S| \leq k} |\alpha_S|$.

Example: $\text{AND}(x_1, x_2) = \text{sign}(x_1 + x_2 + 1) = \text{sign}(100x_1 + 100x_2 + 1)$.

# Polynomial Threshold Circuits

A circuit consisting of PTF gates.

# Polynomial Threshold Circuits

A circuit consisting of PTF gates.

**Definition ($k$-PTF circuits)**

A $k$-PTF circuit on $n$ variables is a Boolean circuit, where each gate of fan-in $m$ computes a fixed $k$-PTF of its inputs.

# Polynomial Threshold Circuits

A circuit consisting of PTF gates.

**Definition ($k$-PTF circuits)**

A $k$-PTF circuit on $n$ variables is a Boolean circuit, where each gate of fan-in $m$ computes a fixed $k$-PTF of its inputs.

Size of the circuit is the number of gates in it.

Depth of the circuit is the longest input to output path.

# Polynomial Threshold Circuits

A circuit consisting of PTF gates.

**Definition ($k$-PTF circuits)**

A $k$-PTF circuit on $n$ variables is a Boolean circuit, where each gate of fan-in $m$ computes a fixed $k$-PTF of its inputs.

Size of the circuit is the number of gates in it.

Depth of the circuit is the longest input to output path.

Weight of the circuit is the maximum among the weights of $k$-PTFs in the circuit.

# Polynomial Threshold Circuits

A circuit consisting of PTF gates.

**Definition ($k$-PTF circuits)**

A $k$-PTF circuit on $n$ variables is a Boolean circuit, where each gate of fan-in $m$ computes a fixed $k$-PTF of its inputs.

Size of the circuit is the number of gates in it.

Depth of the circuit is the longest input to output path.

Weight of the circuit is the maximum among the weights of $k$-PTFs in the circuit.

# Polynomial Threshold Circuits

Suppose each gate is a Linear Threshold function, the class is called $\mathsf{TC}^0$.

# Polynomial Threshold Circuits

Suppose each gate is a Linear Threshold function, the class is called $\mathsf{TC}^0$.

- They are powerful.
  - Integer arithmetic can be done in $\mathsf{TC}^0$.

# Polynomial Threshold Circuits

Suppose each gate is a Linear Threshold function, the class is called $\mathsf{TC}^0$.

They are powerful.

Integer arithmetic can be done in $\mathsf{TC}^0$.
[Beame, Cook, Hoover 1986],[Hesse, Allender,Barrington, 2002].

# Polynomial Threshold Circuits

Suppose each gate is a Linear Threshold function, the class is called $\mathsf{TC}^0$.

They are powerful.

Integer arithmetic can be done in $\mathsf{TC}^0$.
[Beame, Cook, Hoover 1986],[Hesse, Allender,Barrington, 2002].

At the frontier of lower bound techniques.

# Polynomial Threshold Circuits

Suppose each gate is a Linear Threshold function, the class is called $\mathsf{TC}^0$.

- They are powerful.
  - Integer arithmetic can be done in $\mathsf{TC}^0$.
  - [Beame, Cook, Hoover 1986],[Hesse, Allender,Barrington, 2002].
- At the frontier of lower bound techniques.
  - For instance [Kane, Williams, 2015], [Chen 2018].

# Polynomial Threshold Circuits

Suppose each gate is a Linear Threshold function, the class is called $\mathsf{TC}^0$.

They are powerful.

Integer arithmetic can be done in $\mathsf{TC}^0$.

[Beame, Cook, Hoover 1986],[Hesse, Allender,Barrington, 2002].

At the frontier of lower bound techniques.

For instance [Kane, Williams, 2015], [Chen 2018].

Polynomial Threshold circuits are a natural generalization of $\mathsf{TC}^0$.

# Satisfiability algorithms for a single $k$-PTF

Better than brute-force satisfiability algorithms.

# Satisfiability algorithms for a single $k$-PTF

Better than brute-force satisfiability algorithms.

Algorithms that run in time $2^{n-s}$, where $s$ is non-trivial.

# Satisfiability algorithms for a single $k$-PTF

Better than brute-force satisfiability algorithms.
Algorithms that run in time $2^{n-s}$, where $s$ is non-trivial.

Known results for a single PTF gate

# Satisfiability algorithms for a single $k$-PTF

Better than brute-force satisfiability algorithms.
Algorithms that run in time $2^{n-s}$, where $s$ is non-trivial.

Known results for a single PTF gate

A single 2-PTF satisfiability.
[Williams, 2004], [Williams, 2014].

# Satisfiability algorithms for a single $k$-PTF

Better than brute-force satisfiability algorithms.
Algorithms that run in time $2^{n-s}$, where $s$ is non-trivial.

Known results for a single PTF gate

A single 2-PTF satisfiability.
[Williams, 2004], [Williams, 2014].

#SAT for a single $k$-PTF when the weights are small.
[Sakai, Seto, Tamaki, Teruyama, 2016].

# Satisfiability algorithms for a single $k$-PTF

Better than brute-force satisfiability algorithms.
Algorithms that run in time $2^{n-s}$, where $s$ is non-trivial.

Known results for a single PTF gate

A single 2-PTF satisfiability.
[Williams, 2004], [Williams, 2014].

#SAT for a single $k$-PTF when the weights are small.
[Sakai, Seto, Tamaki, Teruyama, 2016].

# Satisfiability algorithms for $TC^0$ and $k$-PTF circuits

Known results for depth-2

# Satisfiability algorithms for $TC^0$ and $k$-PTF circuits

## Known results for depth-2

For depth-2 $TC^0$ circuits with $O(n)$ gates.

[Impagliazzo, Paturi, Schneider, 2013], [Impagliazzo, Lovett, Paturi, Schneider, 2014].

# Satisfiability algorithms for $\mathsf{TC}^0$ and $k$-PTF circuits

## Known results for depth-2

For depth-2 $\mathsf{TC}^0$ circuits with $O(n)$ gates.
[Impagliazzo, Paturi, Schneider, 2013], [Impagliazzo, Lovett, Paturi, Schneider, 2014].

For depth-2 $\mathsf{TC}^0$ circuits with almost quadratic number of gates.
[Alman, Chan, Williams, 2016], [Tamaki 2016].

# Satisfiability algorithms for $\mathsf{TC}^0$ and $k$-PTF circuits

## Known results for depth-2

For depth-2 $\mathsf{TC}^0$ circuits with $O(n)$ gates.
[Impagliazzo, Paturi, Schneider, 2013], [Impagliazzo, Lovett, Paturi, Schneider, 2014].

For depth-2 $\mathsf{TC}^0$ circuits with almost quadratic number of gates.
[Alman, Chan, Williams, 2016], [Tamaki 2016].

# Satisfiability algorithms for $TC^0$ and $k$-PTF circuits

## Known results for constant-depth

For constant depth $TC^0$ circuits of size $n^{1+\epsilon_d}$, where $d$ is the depth of the circuit.

[Chen, Santhanam, Srinivasan, 2018].

# Satisfiability algorithms for $\mathsf{TC}^0$ and $k$-PTF circuits

## Known results for constant-depth

For constant depth $\mathsf{TC}^0$ circuits of size $n^{1+\epsilon_d}$, where $d$ is the depth of the circuit.

[Chen, Santhanam, Srinivasan, 2018].

The paper proved the first average case lower bound for constant depth $\mathsf{TC}^0$ circuits.

# Satisfiability algorithms for $TC^0$ and $k$-PTF circuits

## Known results for constant-depth

For constant depth $TC^0$ circuits of size $n^{1+\epsilon_d}$, where $d$ is the depth of the circuit.

[Chen, Santhanam, Srinivasan, 2018].

The paper proved the first average case lower bound for constant depth $TC^0$ circuits.

The lower bound was extended to a much more powerful class of constant depth PTF circuits.

# Satisfiability algorithms for $TC^0$ and $k$-PTF circuits

## Known results for constant-depth

For constant depth $TC^0$ circuits of size $n^{1+\epsilon_d}$, where $d$ is the depth of the circuit.

[Chen, Santhanam, Srinivasan, 2018].

The paper proved the first average case lower bound for constant depth $TC^0$ circuits.

The lower bound was extended to a much more powerful class of constant depth PTF circuits. [Kane, Kabanets, Lu, 2017].

# Satisfiability algorithms for $TC^0$ and $k$-PTF circuits

## Known results for constant-depth

For constant depth $TC^0$ circuits of size $n^{1+\epsilon_d}$, where $d$ is the depth of the circuit.

[Chen, Santhanam, Srinivasan, 2018].

The paper proved the first average case lower bound for constant depth $TC^0$ circuits.

The lower bound was extended to a much more powerful class of constant depth PTF circuits. [Kane, Kabanets, Lu, 2017].

For constant depth PTF circuits of size $n^{1+\epsilon_d}$, where $d$ depends on the depth of the circuit

# Satisfiability algorithms for $TC^0$ and $k$-PTF circuits

## Known results for constant-depth

For constant depth $TC^0$ circuits of size $n^{1+\epsilon_d}$, where $d$ is the depth of the circuit.

[Chen, Santhanam, Srinivasan, 2018].

The paper proved the first average case lower bound for constant depth $TC^0$ circuits.

The lower bound was extended to a much more powerful class of constant depth PTF circuits. [Kane, Kabanets, Lu, 2017].

For constant depth PTF circuits of size $n^{1+\epsilon_d}$, where $d$ depends on the depth of the circuit, and sparsity $n^{2-\Omega(1)}$.

[Kabanets and Lu 2018].

# Satisfiability algorithms for $\mathrm{TC}^0$ and $k$-PTF circuits

## Known results for constant-depth

For constant depth $\mathrm{TC}^0$ circuits of size $n^{1+\epsilon_d}$, where $d$ is the depth of the circuit.

[Chen, Santhanam, Srinivasan, 2018].

The paper proved the first average case lower bound for constant depth $\mathrm{TC}^0$ circuits.

The lower bound was extended to a much more powerful class of constant depth PTF circuits. [Kane, Kabanets, Lu, 2017].

For constant depth PTF circuits of size $n^{1+\epsilon_d}$, where $d$ depends on the depth of the circuit, and sparsity $n^{2-\Omega(1)}$.

[Kabanets and Lu 2018].

The last two algorithms also work for #SAT.

# A simple question

Question left open by previous works.

# A simple question

Question left open by previous works.

Is there a better than brute-force #SAT algorithm for degree-$k$ PTFs?

# A simple question

Question left open by previous works.

Is there a better than brute-force #SAT algorithm for degree-$k$ PTFs?

Answered affirmatively here.

# A simple question

Question left open by previous works.

Is there a better than brute-force #SAT algorithm for degree-$k$ PTFs?

Answered affirmatively here.

Our result

**Theorem (#SAT for a single $k$-PTF)**

*Fix any constant $k$, there is a zero-error radomized algorithm that solves the #SAT problem for a single $k$-PTF in time $\mathrm{poly}(n, M) \cdot 2^{n-s}$, where $s = \tilde{\Omega}(n^{1/k+1})$.*

# A simple question

Question left open by previous works.

Is there a better than brute-force #SAT algorithm for degree-$k$ PTFs?

Answered affirmatively here.

Our result

**Theorem (#SAT for a single $k$-PTF)**

*Fix any constant $k$, there is a zero-error radomized algorithm that solves the #SAT problem for a single $k$-PTF in time $\mathrm{poly}(n, M) \cdot 2^{n-s}$, where $s = \tilde{\Omega}(n^{1/k+1})$.*

*Here $n$ is the number of variables and $M = w(P)$.*

# A simple question

Question left open by previous works.

Is there a better than brute-force #SAT algorithm for degree-$k$ PTFs?

Answered affirmatively here.

Our result

**Theorem (#SAT for a single $k$-PTF)**

*Fix any constant $k$, there is a zero-error radomized algorithm that solves the #SAT problem for a single $k$-PTF in time $\mathrm{poly}(n, M) \cdot 2^{n-s}$, where $s = \tilde{\Omega}(n^{1/k+1})$.*

*Here $n$ is the number of variables and $M = w(P)$.*

*$w(P)$: bit-complexity of sum of absolute values of the coeffients of the $k$-PTF.*

# A simple question

Question left open by previous works.

Is there a better than brute-force #SAT algorithm for degree-$k$ PTFs?

Answered affirmatively here.

Our result

**Theorem (#SAT for a single $k$-PTF)**

*Fix any constant $k$, there is a zero-error radomized algorithm that solves the #SAT problem for a single $k$-PTF in time $\mathrm{poly}(n, M) \cdot 2^{n-s}$, where $s = \tilde{\Omega}(n^{1/k+1})$.*

*Here $n$ is the number of variables and $M = w(P)$.*

*$w(P)$: bit-complexity of sum of absolute values of the coeffients of the $k$-PTF.*

Some comments on zero-error randomized algorithms.

# #SAT algorithm for $k$-PTF circuits

Our result

**Theorem (#SAT for constant depth $k$-PTF circuits)**

*Fix any constants $k, d$, we have the following for some fixed constants $\varepsilon_{k,d}, \beta_{k,d}$ depending only on $k, d$.*

# #SAT algorithm for $k$-PTF circuits

Our result

**Theorem (#SAT for constant depth $k$-PTF circuits)**

*Fix any constants $k, d$, we have the following for some fixed constants $\varepsilon_{k,d}, \beta_{k,d}$ depending only on $k, d$.*

*There is a zero-error randomized algorithm that solves #SAT problem for $k$-PTF circuits of depth $d$ and size $n^{(1+\varepsilon_{k,d})}$ in time $\mathrm{poly}(n, M) \cdot 2^{n-s}$, where $s = n^{\beta_{k,d}}$.*

# #SAT algorithm for $k$-PTF circuits

## Our result

### Theorem (#SAT for constant depth $k$-PTF circuits)

*Fix any constants $k, d$, we have the following for some fixed constants $\varepsilon_{k,d}, \beta_{k,d}$ depending only on $k, d$.*

*There is a zero-error randomized algorithm that solves #SAT problem for $k$-PTF circuits of depth $d$ and size $n^{(1+\varepsilon_{k,d})}$ in time $\mathrm{poly}(n, M) \cdot 2^{n-s}$, where $s = n^{\beta_{k,d}}$. Here $n$ is the number of inputs, $M$ is the weight of the circuit.*

# #SAT algorithm for $k$-PTF circuits

## Our result

**Theorem (#SAT for constant depth $k$-PTF circuits)**

*Fix any constants $k, d$, we have the following for some fixed constants $\varepsilon_{k,d}, \beta_{k,d}$ depending only on $k, d$.*

*There is a zero-error randomized algorithm that solves #SAT problem for $k$-PTF circuits of depth $d$ and size $n^{(1+\varepsilon_{k,d})}$ in time $\mathrm{poly}(n, M) \cdot 2^{n-s}$, where $s = n^{\beta_{k,d}}$. Here $n$ is the number of inputs, $M$ is the weight of the circuit.*

*Weight of a $k$-PTF circuit is the maximum among the weights of $k$-PTFs in the circuit.*

# #SAT algorithm for a single $k$-PTF

For simplicity of presentation, we will discuss SAT algorithm.

# #SAT algorithm for a single $k$-PTF

For simplicity of presentation, we will discuss SAT algorithm.

Memoization

# #SAT algorithm for a single $k$-PTF

For simplicity of presentation, we will discuss SAT algorithm.

### Memoization

A technique to solve satisfiability problems.

# #SAT algorithm for a single $k$-PTF

For simplicity of presentation, we will discuss SAT algorithm.

Memoization

A technique to solve satisfiability problems.

A 2-step procedure to solve satisfiability for class $\mathcal{C}$ of circuits.

# #SAT algorithm for a single $k$-PTF

For simplicity of presentation, we will discuss SAT algorithm.

## Memoization

A technique to solve satisfiability problems.

A 2-step procedure to solve satisfiability for class $\mathcal{C}$ of circuits.

# Memoization

A 2-step procedure to solve satisfiability for class $\mathcal{C}$ of circuits.

# Memoization

A 2-step procedure to solve satisfiability for class $\mathcal{C}$ of circuits.

Step 1 Use brute-force to solve all instances on $m$ inputs. Typically $m = n^{\varepsilon}$.

# Memoization

A 2-step procedure to solve satisfiability for class $\mathcal{C}$ of circuits.

Step 1 Use brute-force to solve all instances on $m$ inputs. Typically $m = n^{\varepsilon}$.

Store all answers (SAT or not SAT) for each in a table $\mathcal{T}$.

Takes time $\exp(m^{O(1)}) \ll 2^n$.

# Memoization

A 2-step procedure to solve satisfiability for class $\mathcal{C}$ of circuits.

Step 1 Use brute-force to solve all instances on $m$ inputs. Typically $m = n^{\varepsilon}$.

Store all answers (SAT or not SAT) for each in a table $\mathcal{T}$.

Takes time $\exp(m^{O(1)}) \ll 2^n$.

Step 2 On input $C \in \mathcal{C}$,
set variables $x_{m+1}, \ldots, x_n$ to all possible Boolean values.

Each setting creates an instance on $m$ inputs.

# Memoization

A 2-step procedure to solve satisfiability for class $\mathcal{C}$ of circuits.

Step 1 Use brute-force to solve all instances on $m$ inputs. Typically $m = n^{\varepsilon}$.

Store all answers (SAT or not SAT) for each in a table $\mathcal{T}$.

Takes time $\exp(m^{O(1)}) \ll 2^n$.

Step 2 On input $C \in \mathcal{C}$,
set variables $x_{m+1}, \ldots, x_n$ to all possible Boolean values.

Each setting creates an instance on $m$ inputs.

Look-up $\mathcal{T}$ and figure out whether it is satisfiable.

# Memoization

A 2-step procedure to solve satisfiability for class $\mathcal{C}$ of circuits.

Step 1 Use brute-force to solve all instances on $m$ inputs. Typically $m = n^{\varepsilon}$.

Store all answers (SAT or not SAT) for each in a table $\mathcal{T}$.

Takes time $\exp(m^{O(1)}) \ll 2^n$.

Step 2 On input $C \in \mathcal{C}$,
set variables $x_{m+1}, \ldots, x_n$ to all possible Boolean values.

Each setting creates an instance on $m$ inputs.

Look-up $\mathcal{T}$ and figure out whether it is satisfiable.

If look-up can be done in $\mathrm{poly}(|C|)$ time, then this step takes time $O(2^{n-m} \cdot \mathrm{poly}(|C|)) \ll 2^n$.

# Memoization for $k$-PTF

Given as input $f$ specified by a degree-$k$ polynomial $P$ on $n$ variables with integer coefficients.

Can memoization be made to work for a single $k$-PTF?

# Memoization for $k$-PTF

Given as input $f$ specified by a degree-$k$ polynomial $P$ on $n$ variables with integer coefficients.

Can memoization be made to work for a single $k$-PTF?

Step 1 The number of $k$-PTF on $m$ variables is $2^{m^{k+1}}$. [Chow 1961].

# Memoization for $k$-PTF

Given as input $f$ specified by a degree-$k$ polynomial $P$ on $n$ variables with integer coefficients.

Can memoization be made to work for a single $k$-PTF?

Step 1 The number of $k$-PTF on $m$ variables is $2^{m^{k+1}}$. [Chow 1961].

Hence this step can be implemented in time $2^{m^{k+1}} \ll 2^n$ time.

# Memoization for $k$-PTF

Given as input $f$ specified by a degree-$k$ polynomial $P$ on $n$ variables with integer coefficients.

Can memoization be made to work for a single $k$-PTF?

Step 1 The number of $k$-PTF on $m$ variables is $2^{m^{k+1}}$. [Chow 1961].

Hence this step can be implemented in time $2^{m^{k+1}} \ll 2^n$ time.

Step 2 For this to work, the look-up (into the functions stored in Step 1) need to happen quickly.

# Memoization for $k$-PTF

Given as input $f$ specified by a degree-$k$ polynomial $P$ on $n$ variables with integer coefficients.

Can memoization be made to work for a single $k$-PTF?

Step 1 The number of $k$-PTF on $m$ variables is $2^{m^{k+1}}$. [Chow 1961].

Hence this step can be implemented in time $2^{m^{k+1}} \ll 2^n$ time.

Step 2 For this to work, the look-up (into the functions stored in Step 1) need to happen quickly.

This step not obvious.

# Memoization for $k$-PTF

Given as input $f$ specified by a degree-$k$ polynomial $P$ on $n$ variables with integer coefficients.

Can memoization be made to work for a single $k$-PTF?

Step 1 The number of $k$-PTF on $m$ variables is $2^{m^{k+1}}$. [Chow 1961].

Hence this step can be implemented in time $2^{m^{k+1}} \ll 2^n$ time.

Step 2 For this to work, the look-up (into the functions stored in Step 1) need to happen quickly.

This step not obvious.

Quick Look-up?

# Memoization for $k$-PTF

## Quick Look-up: A possible approach.

# Memoization for $k$-PTF

## Quick Look-up: A possible approach.

Every $k$-PTF on $m$ variables can be sign represented by a polynomial with coefficients bounded by $2^{O(\mathrm{poly}(m))}$. [Muroga 1971].

# Memoization for $k$-PTF

## Quick Look-up: A possible approach.

Every $k$-PTF on $m$ variables can be sign represented by a polynomial with coefficients bounded by $2^{O(\mathrm{poly}(m))}$. [Muroga 1971].

Simply store all polynomials with small weights in the table.

# Memoization for $k$-PTF

## Quick Look-up: A possible approach.

Every $k$-PTF on $m$ variables can be sign represented by a polynomial with coefficients bounded by $2^{O(\mathrm{poly}(m))}$. [Muroga 1971].

Simply store all polynomials with small weights in the table.

Doable in time $2^{O(\mathrm{poly}(m))} \ll 2^n$.

# Memoization for $k$-PTF

## Quick Look-up: A possible approach.

Every $k$-PTF on $m$ variables can be sign represented by a polynomial with coefficients bounded by $2^{O(\mathrm{poly}(m))}$. [Muroga 1971].

Simply store all polynomials with small weights in the table.

Doable in time $2^{O(\mathrm{poly}(m))} \ll 2^n$.

May not wok.

# Memoization for $k$-PTF

Quick Look-up: A possible approach.

Every $k$-PTF on $m$ variables can be sign represented by a polynomial with coefficients bounded by $2^{O(\mathrm{poly}(m))}$. [Muroga 1971].

Simply store all polynomials with small weights in the table.

Doable in time $2^{O(\mathrm{poly}(m))} \ll 2^n$.

May not wok.

A $k$-PTF $P$ on $n$ variables is reduced to a $k$-PTF $P'$ on $m$ variables by Step 1.

# Memoization for $k$-PTF

## Quick Look-up: A possible approach.

Every $k$-PTF on $m$ variables can be sign represented by a polynomial with coefficients bounded by $2^{O(\mathrm{poly}(m))}$. [Muroga 1971].

Simply store all polynomials with small weights in the table.

Doable in time $2^{O(\mathrm{poly}(m))} \ll 2^n$.

## May not wok.

A $k$-PTF $P$ on $n$ variables is reduced to a $k$-PTF $P'$ on $m$ variables by Step 1.

The coeffiecients of $P'$ can be as large as $2^{\mathrm{poly}(n)}$.

Not clear how to find a polynomial with small coefficients that sign-represents $P'$.

# Memoization for $k$-PTF

## Quick Look-up: A possible approach.

Every $k$-PTF on $m$ variables can be sign represented by a polynomial with coefficients bounded by $2^{O(\mathrm{poly}(m))}$. [Muroga 1971].

Simply store all polynomials with small weights in the table.

Doable in time $2^{O(\mathrm{poly}(m))} \ll 2^n$.

## May not wok.

A $k$-PTF $P$ on $n$ variables is reduced to a $k$-PTF $P'$ on $m$ variables by Step 1.

The coeffiecients of $P'$ can be as large as $2^{\mathrm{poly}(n)}$.

Not clear how to find a polynomial with small coefficients that sign-represents $P'$.

# Memoization for $k$-PTF

Quick Look-up: Another possible approach.

# Memoization for $k$-PTF

Quick Look-up: Another possible approach.

A $k$-PTF on $m$ variables can be represented by $\mathrm{poly}(m)$ many numbers of $O(m)$ bit-complexity.

The numbers are called Chow parameters. [Chow 1961].

# Memoization for $k$-PTF

Quick Look-up: Another possible approach.

A $k$-PTF on $m$ variables can be represented by $\mathrm{poly}(m)$ many numbers of $O(m)$ bit-complexity.

The numbers are called Chow parameters. [Chow 1961].

Expensive to compute

Even for LTFs computing Chow parameters is known to be NP-hard. [O'Donnell, Servedio 2011].

# Linear Decision Tree

Our approach:

# Linear Decision Tree

Our approach:

Linear Decision Trees

# Linear Decision Tree

Our approach:

Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

# Linear Decision Tree

Our approach:

Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

There is an algorithm that given a positive integer $r$ and a set $H \subseteq \{-1, 1\}^r$

# Linear Decision Tree

Our approach:

Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

There is an algorithm that given a positive integer $r$ and a set $H \subseteq \{-1, 1\}^r$, produces a decision tree $T$ in time $2^{O(\Delta)}$, where $\Delta = O(r \log r \log |H|)$

# Linear Decision Tree

Our approach:

Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

> There is an algorithm that given a positive integer $r$ and a set $H \subseteq \{-1, 1\}^r$, produces a decision tree $T$ in time $2^{O(\Delta)}$, where $\Delta = O(r \log r \log |H|)$ and $T$ that has the following properties:

# Linear Decision Tree

Our approach:

Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

There is an algorithm that given a positive integer $r$ and a set $H \subseteq \{-1,1\}^r$, produces a decision tree $T$ in time $2^{O(\Delta)}$, where $\Delta = O(r \log r \log |H|)$ and $T$ that has the following properties:

Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w$ is the input to $T$ and $\alpha_i \in \{-2,-1,0,1,2\}$.

# Linear Decision Tree

Our approach:

Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

There is an algorithm that given a positive integer $r$ and a set $H \subseteq \{-1,1\}^r$, produces a decision tree $T$ in time $2^{O(\Delta)}$, where $\Delta = O(r \log r \log |H|)$ and $T$ that has the following properties:

Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w$ is the input to $T$ and $\alpha_i \in \{-2,-1,0,1,2\}$.

It computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ such that

given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

# Linear Decision Tree

Our approach:

Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

There is an algorithm that given a positive integer $r$ and a set $H \subseteq \{-1,1\}^r$, produces a decision tree $T$ in time $2^{O(\Delta)}$, where $\Delta = O(r \log r \log |H|)$ and $T$ that has the following properties:

Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w$ is the input to $T$ and $\alpha_i \in \{-2,-1,0,1,2\}$.

It computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ such that

given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

The depth of the decision tree is $\Delta$.

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter,
$H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter,
$H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test ($\sum_{i=1}^{r} \alpha_i w_i \geq \theta$), where $w$ is the input to $T$

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter,
$H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test ($\sum_{i=1}^{r} \alpha_i w_i \geq \theta$), where $w$ is the input to $T$ and $\alpha_i \in \{-2, -1, 0, 1, 2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1, 1\}^{|H|}$ such that

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter,
$H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w$ is the input to $T$ and $\alpha_i \in \{-2, -1, 0, 1, 2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1, 1\}^{|H|}$ such that
- Given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter,
$H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test ($\sum_{i=1}^{r} \alpha_i w_i \geq \theta$), where $w$ is the input to $T$ and $\alpha_i \in \{-2, -1, 0, 1, 2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1, 1\}^{|H|}$ such that
- Given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

Solving a learning problem.

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter, $H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test ($\sum_{i=1}^{r} \alpha_i w_i \geq \theta$), where $w$ is the input to $T$ and $\alpha_i \in \{-2, -1, 0, 1, 2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1, 1\}^{|H|}$ such that
- Given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

Solving a learning problem.

Given $w \in \mathbb{R}^r$,

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter, $H \subseteq \{-1,1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w$ is the input to $T$ and $\alpha_i \in \{-2,-1,0,1,2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ such that
- Given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

Solving a learning problem.

Given $w \in \mathbb{R}^r$,

think of $w$ as a linear test $f_w(h) := \langle w, h \rangle$

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter, $H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test ($\sum_{i=1}^{r} \alpha_i w_i \geq \theta$), where $w$ is the input to $T$ and $\alpha_i \in \{-2, -1, 0, 1, 2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1, 1\}^{|H|}$ such that
- Given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

Solving a learning problem.

Given $w \in \mathbb{R}^r$,

think of $w$ as a linear test $f_w(h) := \langle w, h \rangle$

We want to learn $\text{sign}(f_w)$ at every point in $H$.

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter, $H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w$ is the input to $T$ and $\alpha_i \in \{-2, -1, 0, 1, 2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1, 1\}^{|H|}$ such that
- Given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

Solving a learning problem.

Given $w \in \mathbb{R}^r$,

think of $w$ as a linear test $f_w(h) := \langle w, h \rangle$

We want to learn $\mathrm{sign}(f_w)$ at every point in $H$.

Types of queries allowed: for $h, h' \in H$ is $f_w(h) \geq f_w(h')$?

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter, $H \subseteq \{-1, 1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w$ is the input to $T$ and $\alpha_i \in \{-2, -1, 0, 1, 2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1, 1\}^{|H|}$ such that
- Given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

Solving a learning problem.

Given $w \in \mathbb{R}^r$,

think of $w$ as a linear test $f_w(h) := \langle w, h \rangle$

We want to learn $\mathrm{sign}(f_w)$ at every point in $H$.

Types of queries allowed: for $h, h' \in H$ is $f_w(h) \geq f_w(h')$?

Small depth decision tree for this

# A little bit about Linear Decision Tree [KLMZ 2017]

## Linear Decision Trees

Let $r$ be a parameter, $H \subseteq \{-1,1\}^r$.

The linear decision tree $T$ has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w$ is the input to $T$ and $\alpha_i \in \{-2,-1,0,1,2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ such that
- Given an input $w \in \mathbb{R}^r$, $F(w)$ is the truth table of the LTF defined by $w$ on all points in $H$.

Solving a learning problem.

Given $w \in \mathbb{R}^r$,

think of $w$ as a linear test $f_w(h) := \langle w, h \rangle$

We want to learn $\text{sign}(f_w)$ at every point in $H$.

Types of queries allowed: for $h, h' \in H$ is $f_w(h) \geq f_w(h')$?

Small depth decision tree for this implies a fast learning algorithm.

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\text{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\text{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

For a point $b \in \{-1, 1\}^m$, let $e_b \in \{-1, 1\}^r$ denote the evaluation vector of all monomials of degree at most $k$ on the point $b$.

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\text{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

For a point $b \in \{-1, 1\}^m$, let $e_b \in \{-1, 1\}^r$ denote the evaluation vector of all monomials of degree at most $k$ on the point $b$.

Let $H = \{e_b \mid b \in \{-1, 1\}^m\}$.

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\text{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

For a point $b \in \{-1,1\}^m$, let $e_b \in \{-1,1\}^r$ denote the evaluation vector of all monomials of degree at most $k$ on the point $b$.

Let $H = \{e_b \mid b \in \{-1,1\}^m\}$. $|H| \leq 2^m$.

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\text{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

For a point $b \in \{-1,1\}^m$, let $e_b \in \{-1,1\}^r$ denote the evaluation vector of all monomials of degree at most $k$ on the point $b$.

Let $H = \{e_b \mid b \in \{-1,1\}^m\}$. $|H| \leq 2^m$.

Note that

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\text{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

For a point $b \in \{-1, 1\}^m$, let $e_b \in \{-1, 1\}^r$ denote the evaluation vector of all monomials of degree at most $k$ on the point $b$.

Let $H = \{e_b \mid b \in \{-1, 1\}^m\}$. $|H| \leq 2^m$.

Note that

Given a polynomial $P'$ of degree $k$ on $m$ variables

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\mathrm{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

For a point $b \in \{-1, 1\}^m$, let $e_b \in \{-1, 1\}^r$ denote the evaluation vector of all monomials of degree at most $k$ on the point $b$.

Let $H = \{e_b \mid b \in \{-1, 1\}^m\}$. $|H| \leq 2^m$.

Note that

Given a polynomial $P'$ of degree $k$ on $m$ variables, the truth table of the function sign-represented by $P'$

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\mathrm{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

For a point $b \in \{-1,1\}^m$, let $e_b \in \{-1,1\}^r$ denote the evaluation vector of all monomials of degree at most $k$ on the point $b$.

Let $H = \{e_b \mid b \in \{-1,1\}^m\}$. $|H| \leq 2^m$.

Note that

Given a polynomial $P'$ of degree $k$ on $m$ variables, the truth table of the function sign-represented by $P'$ is given by the LTF defined by $\mathrm{coeff}(P')$ evaluated at $H$.

# Using Linear Descision Tree

## Definition

Given a $k$-PTF $P'$ on $m$ variables, let $\text{coeff}(P') \in \mathbb{R}^r$ denote a vector of coefficients of all monomials in $P'$ in lexicographical order, where $r = \sum_{i=0}^{k} \binom{m}{i}$.

For a point $b \in \{-1,1\}^m$, let $e_b \in \{-1,1\}^r$ denote the evaluation vector of all monomials of degree at most $k$ on the point $b$.

Let $H = \{e_b \mid b \in \{-1,1\}^m\}$. $|H| \leq 2^m$.

Note that

Given a polynomial $P'$ of degree $k$ on $m$ variables, the truth table of the function sign-represented by $P'$ is given by the LTF defined by $\text{coeff}(P')$ evaluated at $H$.

# Linear Decision Tree for $k$-PTF

## Linear Decision Trees

# Linear Decision Tree for $k$-PTF

Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

# Linear Decision Tree for $k$-PTF

## Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

Given a $k$-PTF $P'$ on $m$.

Run the algorithm of [KLMZ 2017] with $r = \sum_{i=1}^{m} \binom{m}{i}$ and a set $H = \{e_b \mid b \in \{-1,1\}^m\} \subseteq \{-1,1\}^r$

# Linear Decision Tree for $k$-PTF

## Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

Given a $k$-PTF $P'$ on $m$.

Run the algorithm of [KLMZ 2017] with $r = \sum_{i=1}^{m} \binom{m}{i}$ and a set $H = \{e_b \mid b \in \{-1,1\}^m\} \subseteq \{-1,1\}^r$.

This produces a decision tree $T$ that has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w_i$s are the inputs and $\alpha_i \in \{-2,-1,0,1,2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ such that

# Linear Decision Tree for $k$-PTF

## Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

Given a $k$-PTF $P'$ on $m$.

Run the algorithm of [KLMZ 2017] with $r = \sum_{i=1}^{m} \binom{m}{i}$ and a set $H = \{e_b \mid b \in \{-1,1\}^m\} \subseteq \{-1,1\}^r$.

This produces a decision tree $T$ that has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w_i$s are the inputs and $\alpha_i \in \{-2,-1,0,1,2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ such that
- Given an input $\text{coeff}(P') \in \mathbb{R}^r$, $F(\text{coeff}(P'))$ is the truth table of the $k$-PTF defined by $P'$.

# Linear Decision Tree for $k$-PTF

## Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

Given a $k$-PTF $P'$ on $m$.

Run the algorithm of [KLMZ 2017] with $r = \sum_{i=1}^{m} \binom{m}{i}$ and a set $H = \{e_b \mid b \in \{-1,1\}^m\} \subseteq \{-1,1\}^r$.

This produces a decision tree $T$ that has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w_i$s are the inputs and $\alpha_i \in \{-2,-1,0,1,2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ such that
- Given an input $\text{coeff}(P') \in \mathbb{R}^r$, $F(\text{coeff}(P'))$ is the truth table of the $k$-PTF defined by $P'$.
- The decision tree depth is $\Delta = O(r \log r \log |H|) = O(m^{k+1} \log m)$.

# Linear Decision Tree for $k$-PTF

## Linear Decision Trees [Kane, Lovett, Moran, Zhang 2017]

Given a $k$-PTF $P'$ on $m$.

Run the algorithm of [KLMZ 2017] with $r = \sum_{i=1}^{m} \binom{m}{i}$ and a set $H = \{e_b \mid b \in \{-1,1\}^m\} \subseteq \{-1,1\}^r$.

This produces a decision tree $T$ that has the following properties:

- Each internal node of the tree is a linear test $(\sum_{i=1}^{r} \alpha_i w_i \geq \theta)$, where $w_i$s are the inputs and $\alpha_i \in \{-2,-1,0,1,2\}$.
- It computes a function $F : \mathbb{R}^r \to \{-1,1\}^{|H|}$ such that
- Given an input $\mathrm{coeff}(P') \in \mathbb{R}^r$, $F(\mathrm{coeff}(P'))$ is the truth table of the $k$-PTF defined by $P'$.
- The decision tree depth is $\Delta = O(r \log r \log |H|) = O(m^{k+1} \log m)$.

The tree can be constructed in time $2^{O(\Delta)} = \exp(m^{k+1})$ time.

# Memoization for $k$-PTF

## Our approach

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

For each $\sigma : \{x_{m+1}, \ldots, x_n\} \to \{-1, 1\}$,

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

For each $\sigma : \{x_{m+1}, \ldots, x_n\} \to \{-1, 1\}$,

Compute $k$-PTF $P'_\sigma$ obtained from $P$ after restricting the last $n - m$ variables.

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

For each $\sigma : \{x_{m+1}, \ldots, x_n\} \to \{-1, 1\}$,

Compute $k$-PTF $P'_\sigma$ obtained from $P$ after restricting the last $n - m$ variables.
Time: $\mathrm{poly}(n, M)$.

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

For each $\sigma : \{x_{m+1}, \ldots, x_n\} \to \{-1, 1\}$,

Compute $k$-PTF $P'_\sigma$ obtained from $P$ after restricting the last $n - m$ variables.
Time: $\mathrm{poly}(n, M)$.
Query the tree $T$ using $\mathrm{coeff}(P'_\sigma)$ and compute the answer.

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

For each $\sigma : \{x_{m+1}, \ldots, x_n\} \to \{-1, 1\}$,

Compute $k$-PTF $P'_\sigma$ obtained from $P$ after restricting the last $n - m$ variables.
Time: $\mathrm{poly}(n, M)$.
Query the tree $T$ using $\mathrm{coeff}(P'_\sigma)$ and compute the answer.
Time: $O(m^{k+1} \log m)$.

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

For each $\sigma : \{x_{m+1}, \ldots, x_n\} \to \{-1, 1\}$,

Compute $k$-PTF $P'_\sigma$ obtained from $P$ after restricting the last $n - m$ variables.
Time: $\mathrm{poly}(n, M)$.
Query the tree $T$ using coeff($P'_\sigma$) and compute the answer.
Time: $O(m^{k+1} \log m)$.

Time: $2^{n-m} \times \mathrm{poly}(n, M)$.

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

For each $\sigma : \{x_{m+1}, \ldots, x_n\} \to \{-1, 1\}$,

Compute $k$-PTF $P'_\sigma$ obtained from $P$ after restricting the last $n - m$ variables.
Time: $\mathrm{poly}(n, M)$.
Query the tree $T$ using $\mathrm{coeff}(P'_\sigma)$ and compute the answer.
Time: $O(m^{k+1} \log m)$.

Time: $2^{n-m} \times \mathrm{poly}(n, M)$.

Time: $\exp(m^{k+1}) + 2^{n-m}\,\mathrm{poly}(n, M) = 2^{n-m}\,\mathrm{poly}(n, M)$

# Memoization for $k$-PTF

## Our approach

Step 1 Construct Linear Decision Tree $T$ for $k$-PTFs on $m$ variables.
Time: $\exp(m^{k+1})$.

Step 2 Let $P$ be a $k$-PTF on $n$ variables and $w(P) = M$.

For each $\sigma : \{x_{m+1}, \ldots, x_n\} \to \{-1, 1\}$,

Compute $k$-PTF $P'_\sigma$ obtained from $P$ after restricting the last $n - m$ variables.
Time: $\mathrm{poly}(n, M)$.
Query the tree $T$ using coeff$(P'_\sigma)$ and compute the answer.
Time: $O(m^{k+1} \log m)$.

Time: $2^{n-m} \times \mathrm{poly}(n, M)$.

Time: $\exp(m^{k+1}) + 2^{n-m}\,\mathrm{poly}(n, M) = 2^{n-m}\,\mathrm{poly}(n, M)$ if $m = n^{1/k+1}/\log n$.

# Small-depth $k$-PTF circuits

## Our Approach

# Small-depth $k$-PTF circuits

## Our Approach

Use the template of [Kabanets, Lu 2018].

# Small-depth $k$-PTF circuits

## Our Approach

Use the template of [Kabanets, Lu 2018].

Note that there are two crucial steps where [KL 2018] use the sparcity assumption.

# Small-depth $k$-PTF circuits

## Our Approach

Use the template of [Kabanets, Lu 2018].

Note that there are two crucial steps where [KL 2018] use the sparcity assumption.

Develop strategies that work for these two steps even when the $k$-PTF gates are not sparse.

# Satisfiability algorithms vs. SAT solving

| Circuit satisfiability algorithms | SAT solving |
| --- | --- |
| worst case guarantees | guarantees for a subset of instances |

# Satisfiability algorithms vs. SAT solving

| Circuit satisfiability algorithms | SAT solving |
|---|---|
| worst case guarantees | guarantees for a subset of instances |
| connections to/from circuit lower bounds | connections to/from proof complexity |

# Satisfiability algorithms vs. SAT solving

| Circuit satisfiability algorithms | SAT solving |
|---|---|
| worst case guarantees | guarantees for a subset of instances |
| connections to/from circuit lower bounds | connections to/from proof complexity |

Both fields have witnessed many interesting developments.

# Satisfiability algorithms vs. SAT solving

| Circuit satisfiability algorithms | SAT solving |
|---|---|
| worst case guarantees | guarantees for a subset of instances |
| connections to/from circuit lower bounds | connections to/from proof complexity |

Both fields have witnessed many interesting developments.

Some techniques in this talk could be of general interest!

# Conclusion and open problems

Better than brute-force algorithms for

- $k$-PTFs.
- Constant depth $k$-PTF circuits with slightly superlinear size.

# Conclusion and open problems

Better than brute-force algorithms for

- $k$-PTFs.
- Constant depth $k$-PTF circuits with slightly superlinear size.

Open problems

# Conclusion and open problems

Better than brute-force algorithms for

- $k$-PTFs.
- Constant depth $k$-PTF circuits with slightly superlinear size.

Open problems

- Can the techniques here give better-than brute-force algorithms in the subquadratic regime?

# Conclusion and open problems

Better than brute-force algorithms for

- $k$-PTFs.
- Constant depth $k$-PTF circuits with slightly superlinear size.

Open problems

- Can the techniques here give better-than brute-force algorithms in the subquadratic regime?
- Other connections from learning algorithms to satisfiability algorithms?

# Conclusion and open problems

Better than brute-force algorithms for

- $k$-PTFs.
- Constant depth $k$-PTF circuits with slightly superlinear size.

Open problems

- Can the techniques here give better-than brute-force algorithms in the subquadratic regime?
- Other connections from learning algorithms to satisfiability algorithms?
- Can this technique inspire any SAT solving heuristic?

Thank You!