A Fast Algorithm for SAT in Terms of Formula Length

Junqiang Peng[1], Mingyu Xiao[1]

[1]University of Electronic Science and Technology of China

jqpeng0@foxmail.com, myxiao@gamil.com

Our contribution

- An improved parameterized algorithm for SAT running in $O^*(1.0646^L)$, where L is length of the input CNF-formula.

Our contribution

- An improved parameterized algorithm for SAT running in $O^*(1.0646^L)$, where L is length of the input CNF-formula.
    - Branch and Search: New branching rules.

Our contribution

- An improved parameterized algorithm for SAT running in $O^*(1.0646^L)$, where L is length of the input CNF-formula.
    - Branch and Search: New branching rules.
    - Measure and Conquer: Some assumptions on weights.

Outline

Problem Definition

The Satisifiability Problem

Given a CNF formula $\mathcal{F} = C_1 \wedge \cdots \wedge C_m$ on n boolean variables $x_1, \cdots, x_n$, decide if there is an assignment to $x_1, \cdots, x_n$ that makes $\mathcal{F} = 1$.

Problem Definition

The Satisifiability Problem

Given a CNF formula $\mathcal{F} = C_1 \wedge \cdots \wedge C_m$ on n boolean variables $x_1, \cdots, x_n$, decide if there is an assignment to $x_1, \cdots, x_n$ that makes $\mathcal{F} = 1$.

Example

- $\mathcal{F}_1 = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_3 \vee x_4)$.
  Solution: $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1 \Rightarrow \mathcal{F}_1$ is satisifiable.

Problem Definition

The Satisifiability Problem

Given a CNF formula $\mathcal{F} = C_1 \wedge \cdots \wedge C_m$ on n boolean variables $x_1, \cdots, x_n$, decide if there is an assignment to $x_1, \cdots, x_n$ that makes $\mathcal{F} = 1$.

Example

- $\mathcal{F}_1 = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_3 \vee x_4)$.
  Solution: $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1 \Rightarrow \mathcal{F}_1$ is satisifiable.
- $\mathcal{F}_2 = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_2})$.
  Solution: Not exist $\Rightarrow \mathcal{F}_2$ is not satisifiable.

Background

The SAT problem has been extensively and intensively studied in many fields:

- heuristic algorithms
- randomized algorithms
- approximation algorithms
- exact and parameterized algorithms
- ...

Background

The SAT problem has been extensively and intensively studied in many fields:

- heuristic algorithms
- randomized algorithms
- approximation algorithms
- exact and parameterized algorithms
- ...

Background

There are three popular parameters to measure the running time of algorithms for SAT:

Parameter/Measure	Best Running Time Bound
n: the number of variables	$O^*(2^n)$ [1]
m: the number of clauses	$O^*(1.2226^m)$ [2]
L: the number of literals (length)	$O^*(1.0646^L)$ [3]

Strong Exponential Time Hypothesis: The SAT Problem can not be solved in time $O^*(2^n)$.

Our contribution is improving the running time bound in terms of the number of literals (formula length).

[1] Strong Exponential Time Hypothesis (SETH)
[2] AAAI'2021 Chu, Xiao and Zhang
[3] This paper

Background

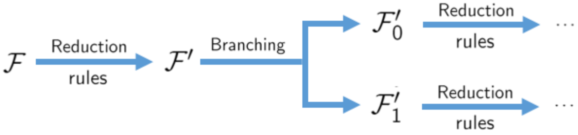
Table: Previous and our upper bound for SAT

Running time bounds	References
$O^*(1.0927^L)$	Van Gelder 1988
$O^*(1.0801^L)$	Kullmann and Luckhardt 1997
$O^*(1.0758^L)$	Hirsch 1998
$O^*(1.074^L)$	Hirsch 2000
$O^*(1.0663^L)$	Wahlström 2005
$O^*(1.0652^L)$	Chen and Liu 2009
$O^*(1.0646^L)$	This paper 2021

Our Algorithm - Overview

Our algorithm is a standard branch-and-search algorithm (Davis–Putnam–Logemann–Loveland (DPLL) algorithm):

- We first apply some reduction rules to reduce the instance.
  - Reduce the size (measure) of the formula and bring us some properties
  - Take polynomial time
- When no reduction rules can be applied, we will search for a solution by branching.
  - Assign value(s) to variable(s) or literal(s)
  - Exponentially increase the running time



Preliminaries

- (i,j)-literal: a literal z is called an (i,j)-literal in a formula $\mathcal{F}$ if z appears i times and $\overline{z}$ appears j times in the formula $\mathcal{F}$.
  (i^+,j)-literal, (i,j^+)-literal, (i^+,j^+)-literal, ...
- Degree: For a variable x in a formula $\mathcal{F}$, the degree of it, denoted by $deg(x)$, is the number of times it appears in the formula. We say a variable is an i-variable if the degree of it is i.
  i^+-variable, i^--variable, ...
- Length: The length of a clause C, denoted by $|C|$, is the number of literals in it. We call a clause k-clause if the length of it is k.
  k^+-clause, ...

$$\mathcal{F} = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_3 \vee x_4) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4})$$

x_3 is a $(2,1)$-literal, $deg(x_3) = 3$.

Preliminaries

If we assign value 1 to literal $x(x = 1, \overline{x} = 0)$, then

- All clauses containing literal x will be removed from the formula.
- All literals $\overline{x}$ will be removed from the clauses.

We use $\mathcal{F}_{x=1}$ to indicate the formula after assigning $x = 1$.

$$\mathcal{F} = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_3 \vee x_4) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4})$$

$$\mathcal{F}_{x_3=1} = (x_1 \vee x_2) \wedge (\overline{x_2} \vee \overline{x_4})$$

Reduction Rules

R-Rule 1 (Elimination of duplicated literals). *If a clause C contains duplicated literals z, remove all but one z in C.* $\mathcal{F}' \wedge (zzD) \to \mathcal{F}' \wedge (zD)$.

R-Rule 2 (Elimination of subsumptions). *If there are two clauses C and D such that $C \subseteq D$, remove clause D.* $\mathcal{F}' \wedge C \wedge D \to \mathcal{F}' \wedge C$.

R-Rule 3 (Elimination of tautology). *If a clause C contains two opposite literals z and $\bar{z}$, remove clause C.* $\mathcal{F}' \wedge (z\bar{z}C) \wedge D \to \mathcal{F}' \wedge D$.

R-Rule 4 (Elimination of 1-clauses and pure literals). *If there is a 1-clause $\{x\}$ or a $(1^+, 0)$-literal x, assign $x = 1$.* $\mathcal{F}' \wedge (x) \to \mathcal{F}'_{x=1}$.

And some other reduction rules... (**R-Rule 6** $\sim$ **R-Rule 10**)

A CNF-formula is called reduced, if none of reduction rules can be applied on it.

Reduction Rules

Lemma

In a reduced CNF-formula $\mathcal{F}$, all variables are 3^+-variables.

Lemma

In a reduced CNF-formula $\mathcal{F}$, if there is a 2-clause xy, then no other clause in $\mathcal{F}$ contains xy, $\overline{x}y$, or $x\overline{y}$.

And some other properties...

Our Algorithm

For a literal x, we have two kinds of branching:

- simple branching: $x = 1$ and $x = 0$
- strong branching: $x = 1$ & $C = 0$ and $x = 0$, where x is a $(1, i)$-literal and xC is the only clause containing literal x.

Our Algorithm

Algorithm 1: SAT($\mathcal{F}$)

Input: a CNF-formula $\mathcal{F}$
Output: 1 or 0 to indicate the satisfiability of $\mathcal{F}$
Step 1. If $\mathcal{F} = \emptyset$, return 1. If $\mathcal{F}$ contains an empty clause, return 0.
Step 2. If $\mathcal{F}$ is not a reduced CNF-formula, iteratively apply the reduction rules to reduce it.
Step 3. If there is a d-variable x with $d \geq 6$, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 4. If there is a $(1,4)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1\ \&\ C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 5. If there is a 5-variable x contained in a 2-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 6. If there is a 5-variable x contained in a 4^+-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 7. If there is a clause containing both a 5-variable x and a 4^--variable, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 8. If there are still some 5-variables, then $\mathcal{F} = \mathcal{F}^* \wedge \mathcal{F}'$, where $\mathcal{F}^*$ is a 3-CNF with $var(\mathcal{F}^*)$ be the set of 5-variables in $\mathcal{F}$ and $var(\mathcal{F}^*) \cap var(\mathcal{F}') = \emptyset$. We return SAT($\mathcal{F}^*$) $\wedge$ SAT($\mathcal{F}'$) and solve $\mathcal{F}^*$ by using the 3-SAT algorithm by Liu [14].
Step 9. If there is a $(1,3)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1\ \&\ C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 10. If there is a $(2,2)$-literal x, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 11. Apply the algorithm by Wahlström [18] to solve the instance.

Our Algorithm

Algorithm SAT($\mathcal{F}$)
Input: a CNF-formula $\mathcal{F}$
Output: 1 or 0 to indicate the satisfiability of $\mathcal{F}$

Our Algorithm

Algorithm SAT($\mathcal{F}$)
Input: a CNF-formula $\mathcal{F}$
Output: 1 or 0 to indicate the satisfiability of $\mathcal{F}$
Step 1. If $\mathcal{F} = \emptyset$, return 1. If $\mathcal{F}$ contains an empty clause, return 0.
Step 2. If $\mathcal{F}$ is not a reduced CNF-formula, iteratively apply the reduction rules to reduce it.
Step 3. If there is a d-variable x with $d \geq 6$, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Algorithm SAT($\mathcal{F}$)
Input: a CNF-formula $\mathcal{F}$
Output: 1 or 0 to indicate the satisfiability of $\mathcal{F}$
Step 1. If $\mathcal{F} = \emptyset$, return 1. If $\mathcal{F}$ contains an empty clause, return 0.
Step 2. If $\mathcal{F}$ is not a reduced CNF-formula, iteratively apply the reduction rules to reduce it.
Step 3. If there is a d-variable x with $d \geq 6$, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Property: Now all variables have a degree ≤ 5.

Algorithm SAT($\mathcal{F}$)

Property: Now all variables have a degree ≤ 5.

Step 4. If there is a $(1,4)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1\ \&\ C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Step 5. If there is a 5-variable x contained in a 2-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Step 6. If there is a 5-variable x contained in a 4^+-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Our Algorithm

Algorithm SAT($\mathcal{F}$)

Property: Now all variables have a degree ≤ 5.

Step 4. If there is a $(1, 4)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1\ \&\ C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Step 5. If there is a 5-variable x contained in a 2-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Step 6. If there is a 5-variable x contained in a 4^+-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Property: Now all clauses containing 5-variables have a length of exactly 3.

Our Algorithm

Algorithm SAT($\mathcal{F}$)

Property: Now all variables have a degree ≤ 5.

Step 4. If there is a $(1,4)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1 \text{ \& } C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).

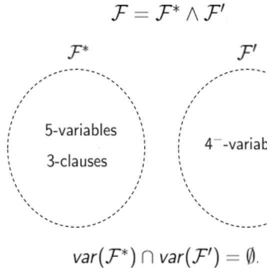
Step 5. If there is a 5-variable x contained in a 2-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Step 6. If there is a 5-variable x contained in a 4^+-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Property: Now all clauses containing 5-variables have a length of exactly 3.

Step 7. If there is a clause containing both a 5-variable x and a 4^--variable, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Algorithm SAT($\mathcal{F}$)
Step 8. If there are still some 5-variables, then $\mathcal{F} = \mathcal{F}^* \wedge \mathcal{F}'$, where $\mathcal{F}^*$ is a 3-CNF with $var(\mathcal{F}^*)$ be the set of 5-variables in $\mathcal{F}$ and $var(\mathcal{F}^*) \cap var(\mathcal{F}') = \emptyset$. We return SAT($\mathcal{F}^*$) $\wedge$ SAT($\mathcal{F}'$) and solve $\mathcal{F}^*$ by using the 3-SAT algorithm with time $O^*(1.3279^n)$ by Liu[4].



[4] Liu, S.: Chain, generalization of covering code, and deterministic algorithm for k-SAT.(ICALP 2018)

Algorithm SAT($\mathcal{F}$)

Property: Now all variables have a degree ≤ 4.

Step 9. If there is a $(1, 3)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1\ \&\ C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Step 10. If there is a $(2, 2)$-literal x, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).

Algorithm SAT($\mathcal{F}$)
Property: Now all variables have a degree ≤ 4.
Step 9. If there is a $(1,3)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1\ \&\ C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 10. If there is a $(2,2)$-literal x, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Property: Now all variables have a degree exactly 3.
Step 11. Apply the algorithm with time $O^*(1.1279^{(d-2)n}) = O^*(1.1279^n)$ by Wahlström[5] to solve the instance.

[5] Wahlström, M.: Faster exact solving of SAT formulae with a low number of occur-rences per variable. (SAT2005)

Our Algorithm

Algorithm SATSolver($\mathcal{F}$)
INPUT: a CNF formula $\mathcal{F}$
OUTPUT: a report whether $\mathcal{F}$ is satisfiable

1. $\mathcal{F} = \text{Reduction}(\mathcal{F})$;
2. pick a $d(\mathcal{F})$-variable x;
3. **if** $d(\mathcal{F}) > 5$ **then**
   return SATSolver($\mathcal{F}[x]$) $\vee$ SATSolver($\mathcal{F}[\overline{x}]$);
4. **else if** $d(\mathcal{F}) > 3$ **then**

   4.1 **if** x is a $(2,2)$-variable with clauses $x\overline{y}_1z_1$, xz_2z_3, $\overline{x}y_1$, and $\overline{x}y_2$ such that y_1 is a 4-variable and y_2 is a 3-variable **then**
   let $\overline{y}_2C_0$ be a clause containing $\overline{y}_2$;
   return SATSolver($\mathcal{F}[C_0 = \text{true}]$) $\vee$ SATSolver($\mathcal{F}[C_0 = \text{false}]$);

   4.2 **if** both x and $\overline{x}$ are 2^+-literals **then**
   return SATSolver($\mathcal{F}[x]$) $\vee$ SATSolver($\mathcal{F}[\overline{x}]$);

   4.3 **else** (* assume the only clause containing $\overline{x}$ is $\overline{x}z_1 \cdots z_h$ *)
   return SATSolver($\mathcal{F}[x]$) $\vee$ SATSolver($\mathcal{F}[\overline{x}, \overline{z}_1, \ldots, \overline{z}_h]$);
5. **else if** $d(\mathcal{F}) = 3$ **then**
   Apply the algorithm by Wahlström [12];
6. **else return** true;

Chen and Liu's algorithm[6] in 2009

Algorithm 1: SAT($\mathcal{F}$)

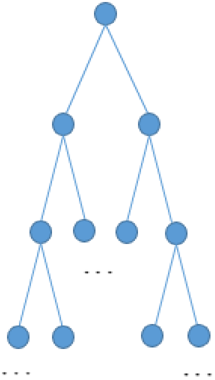
Input: a CNF-formula $\mathcal{F}$
Output: 1 or 0 to indicate the satisfiability of $\mathcal{F}$
Step 1. If $\mathcal{F} = \emptyset$, return 1. If $\mathcal{F}$ contains an empty clause, return 0.
Step 2. If $\mathcal{F}$ is not a reduced CNF-formula, iteratively apply the reduction rules to reduce it.
Step 3. If there is a d-variable x with $d \geq 6$, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 4. If there is a $(1,4)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1\ \&\ C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 5. If there is a 5-variable x contained in a 2-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 6. If there is a 5-variable x contained in a 4^+-clause, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 7. If there is a clause containing both a 5-variable x and a 4^--variable, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 8. If there are still some 5-variables, then $\mathcal{F} = \mathcal{F}^* \wedge \mathcal{F}'$, where $\mathcal{F}^*$ is a 3-CNF with $var(\mathcal{F}^*)$ be the set of 5-variables in $\mathcal{F}$ and $var(\mathcal{F}^*) \cap var(\mathcal{F}') = \emptyset$. We return SAT($\mathcal{F}^*$) $\wedge$ SAT($\mathcal{F}'$) and solve $\mathcal{F}^*$ by using the 3-SAT algorithm by Liu [14].
Step 9. If there is a $(1,3)$-literal x (assume xC is the only clause containing x), return SAT($\mathcal{F}_{x=1\ \&\ C=0}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 10. If there is a $(2,2)$-literal x, return SAT($\mathcal{F}_{x=1}$)$\vee$SAT($\mathcal{F}_{x=0}$).
Step 11. Apply the algorithm by Wahlström [18] to solve the instance.

Our algorithm

[6] Chen, J., Liu, Y.: An improved SAT algorithm in terms of formula length. (WADS2009)

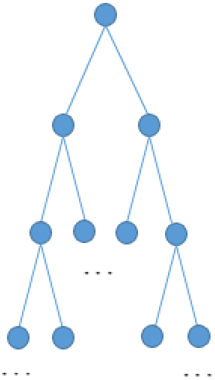
Running Time Bound Analysis

To determine the worst-case running time of a branching algorithm, we can analyze the size of the search tree generated in the algorithm.



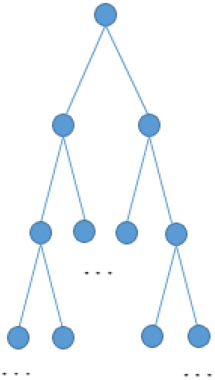
- First a measure μ is defined.
- We use $T(\mu)$ to indicate the maximum size or the number of leaves of the search tree for the input with the measure being at most μ.
- If the algorithm branches into l sub-branches with the measure decreasing at least a_i in the i-th sub-branch, we get a recurrence relation: $T(\mu) \leq T(\mu - a_1) + T(\mu - a_2) + \cdots + T(\mu - a_l)$.

Running Time Bound Analysis



- If the algorithm branches into l sub-branches with the measure decreasing at least a_i in the i-th sub-branch, we get a recurrence relation: $T(\mu) \leq T(\mu - a_1) + T(\mu - a_2) + \cdots + T(\mu - a_l)$.
- $[a_1, a_2, ..., a_l]$ is called a branching vector.
- The largest root of the function $f(x) = 1 - \sum_{i=1}^{l} x^{-a_i}$ is called the branching factor of the recurrence.
- $T(\mu) = O(\gamma^{\mu})$, where γ is the maximum branching factor of all branching factors.

Running Time Bound Analysis



- If the algorithm branches into l sub-branches with the measure decreasing at least a_i in the i-th sub-branch, we get a recurrence relation: $T(\mu) \leq T(\mu - a_1) + T(\mu - a_2) + \cdots + T(\mu - a_l)$.
- $[a_1, a_2, ..., a_l]$ is called a branching vector.
- The largest root of the function $f(x) = 1 - \sum_{i=1}^{l} x^{-a_i}$ is called the branching factor of the recurrence.
- $T(\mu) = O(\gamma^{\mu})$, where γ is the maximum branching factor of all branching factors.
- Running time bound: $O^*(\gamma^{\mu})$.

Measure and Conquer

The main idea is to adopt a new measure instead of measure L.

Measure and Conquer

The main idea is to adopt a new measure instead of measure L.

We introduce a weight to each variable in the formula according to the degree of the variable:

$$w : \mathbb{Z}^+ \to \mathbb{R}^+$$

w_i denote the weight of a variable with degree i.

Measure and Conquer

The main idea is to adopt a new measure instead of measure L.

We introduce a weight to each variable in the formula according to the degree of the variable:

$$w : \mathbb{Z}^+ \rightarrow \mathbb{R}^+$$

w_i denote the weight of a variable with degree i.

In our algorithm, the measure μ of a formula $\mathcal{F}$ is defined as:

$$\mu(\mathcal{F}) = \sum_{x \in \mathcal{F}} w_{deg(x)}$$

Measure and Conquer

The main idea is to adopt a new measure instead of measure L.

We introduce a weight to each variable in the formula according to the degree of the variable:

$$w : \mathbb{Z}^+ \to \mathbb{R}^+$$

w_i denote the weight of a variable with degree i.

In our algorithm, the measure μ of a formula $\mathcal{F}$ is defined as:

$$\mu(\mathcal{F}) = \sum_{x \in \mathcal{F}} w_{deg(x)}$$

Let n_i denote the number of i-variables in $\mathcal{F}$. We also have

$$\mu(\mathcal{F}) = \sum_{i} n_i \cdot w_i$$

Measure and Conquer

If we ensure that $w_i \leq i$, then we have

$$\mu(\mathcal{F}) = \sum_i n_i \cdot w_i \leq \sum_i n_i \cdot i \leq L(\mathcal{F})$$

This tells us if we get a running time bound of $O^*(c^{\mu(\mathcal{F})})$ for a real number c, we also get a running time bound of $O^*(c^{L(\mathcal{F})})$.

Measure and Conquer

We also define $\delta_i = w_i - w_{i-1}$, this is roughly the weight of a literal with its corresponding variable have a degree of i.

For each branching rule, we will analyze how much the new measure $\mu(\mathcal{F})$ decreases in each sub-branch to get the running time bound.

An example:

$$\mathcal{F} = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_3 \vee x_4) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4})$$
$$\mathcal{F}_{x_3=1} = (x_1 \vee x_2) \wedge (\overline{x_2} \vee \overline{x_4})$$
$$\mathcal{F}_{x_3=0} = (\overline{x_2} \vee \overline{x_4}) \wedge (x_4) \wedge (\overline{x_1} \vee \overline{x_4})$$

The branching vector is:

$$[(w_3) + (\delta_2) + (\delta_3 + \delta_2), (w_3) + (\delta_2) + (\delta_2)]$$

Measure and Conquer

How does the new measure work? Consider two cases when the maximum degree is 4.

Measure and Conquer

How does the new measure work? Consider two cases when the maximum degree is 4.

Adopt L as the measure

$w_4 = 4, w_3 = 3 \Rightarrow \delta_4 = 1$:

Branching vector	**Branching factor**
$[w_4 + 2w_3, w_4 + 6\delta_4]$	1.0718
$[w_4 + 2\delta_4, w_4 + 6\delta_4]$	1.0926

Measure and Conquer

Adopt μ as the measure

If we set $w_5 = 5, w_4 = 3.84682, w_3 = 1.92341 \Rightarrow \delta_4 = 1.92341$:

Branching vector	**Branching factor**
$[w_4 + 2w_3, w_4 + 6\delta_4]$	1.0646
$[w_4 + 2\delta_4, w_4 + 6\delta_4]$	1.0646

Running Time Bound Analysis

Assumptions:

- $w_1 = w_2 = 0$
- $w_i \leq i (3 \leq i \leq 4), w_4 = 2w_3$
- $w_i = i (i \geq 5)$
- $\delta_i > 0 (i \geq 3)$
- ...

In **Step 8**, the literals of all 5-variables form a 3-SAT instance $\mathcal{F}^*$. We apply the $O^*(1.3279^n)$-time algorithm for 3-SAT to solve our problem, where n is the number of variables in the instance. Since $w_5 = 5$, we have that $n = \mu(\mathcal{F}^*)/w_5 = \mu(\mathcal{F}^*)/5$. So the running time for this part will be

$$O^*(1.3279^{\mu(\mathcal{F}^*)/w_5}) = O^*(1.0584^{\mu(\mathcal{F}^*)}).$$

Running Time Bound Analysis

In **Step 11**, all variables are 3-variables. We apply the $O^*(1.1279^n)$-time algorithm by Wahlström to solve this special case, where n is the number of variables. For this case, we have that $n = \mu(\mathcal{F})/w_3$. So the running time of this part is

$$O^*((1.1279^{1/w_3})^{\mu(\mathcal{F})}).$$

Table 2. The weight setting

$w_1 = w_2 = 0$	
$w_3 = 1.9234132344759123$	$\delta_3 = 1.9234132344759123$
$w_4 = 3.8468264689518246$	$\delta_4 = 1.9234132344759123$
$w_5 = 5$	$\delta_5 = 1.1531735310481754$
$w_i = i (i \geq 6)$	$\delta_i = 1 (i \geq 6)$

Table 3. The branching vector and factor for each step

Steps	Branching vectors	Branching factors
Step 3	$[w_6 + \delta_6, w_6 + 11\delta_6]$	1.0636
Step 4	$[w_5 + 2w_3, w_5 + 8\delta_5]$	1.0632
Step 5	$[w_5 + 3\delta_5, w_5 + 2w_3 + 5\delta_5]$ $[w_5 + 2\delta_5, w_5 + 4w_3 + 4\delta_5]$	1.0618 1.0636
Step 6	$[w_5 + 4\delta_5, w_5 + 7\delta_5]$	1.0636
Step 7	$[w_5 + 4\delta_5, w_5 + 5\delta_5 + w_3]$	1.0646
Step 8	$O^*((1.3279^{1/w_5})^{\mu})$	1.0584
Step 9	$[w_4 + 2w_3, w_4 + 6\delta_4]$	1.0646
Step 10	$[w_4 + 2\delta_4, w_4 + 6\delta_4]$	1.0646
Step 11	$O^*((1.1279^{1/w_3})^{\mu})$	1.0646

The best choice of w_i can be found by solving a quasi-convex program problem.

The Final Result

Table 2. The weight setting

$w_1 = w_2 = 0$	
$w_3 = 1.9234132344759123$	$\delta_3 = 1.9234132344759123$
$w_4 = 3.8468264689518246$	$\delta_4 = 1.9234132344759123$
$w_5 = 5$	$\delta_5 = 1.1531735310481754$
$w_i = i (i \geq 6)$	$\delta_i = 1 (i \geq 6)$

Table 3. The branching vector and factor for each step

Steps	Branching vectors	Branching factors
Step 3	$[w_6 + \delta_6, w_6 + 11\delta_6]$	1.0636
Step 4	$[w_5 + 2w_3, w_5 + 8\delta_5]$	1.0632
Step 5	$[w_5 + 3\delta_5, w_5 + 2w_3 + 5\delta_5]$ $[w_5 + 2\delta_5, w_5 + 4w_3 + 4\delta_5]$	1.0618 1.0636
Step 6	$[w_5 + 4\delta_5, w_5 + 7\delta_5]$	1.0636
Step 7	$[w_5 + 4\delta_5, w_5 + 5\delta_5 + w_3]$	1.0646
Step 8	$O^*((1.3279^{1/w_5})^\mu)$	1.0584
Step 9	$[w_4 + 2w_3, w_4 + 6\delta_4]$	1.0646
Step 10	$[w_4 + 2\delta_4, w_4 + 6\delta_4]$	1.0646
Step 11	$O^*((1.1279^{1/w_3})^\mu)$	1.0646

The best choice of w_i can be found by solving a quasi-convex program problem.

$$\Downarrow$$

$$c^\mu \leq c^{\mu-(w_6+\delta_6)} + c^{\mu-(w_6+11\delta_6)}$$
$$c^\mu \leq c^{\mu-(w_5+2w_3)} + c^{\mu-(w_5+8\delta_5)}$$
$$\cdots$$
$$c^\mu \leq c^{\mu-(w_4+2\delta_4)} + c^{\mu-(w_4+6\delta_4)}$$

The Final Result

Table 2. The weight setting

$w_1 = w_2 = 0$	
$w_3 = 1.9234132344759123$	$\delta_3 = 1.9234132344759123$
$w_4 = 3.8468264689518246$	$\delta_4 = 1.9234132344759123$
$w_5 = 5$	$\delta_5 = 1.1531735310481754$
$w_i = i (i \geq 6)$	$\delta_i = 1 (i \geq 6)$

Table 3. The branching vector and factor for each step

Steps	Branching vectors	Branching factors
Step 3	$[w_6 + \delta_6, w_6 + 11\delta_6]$	1.0636
Step 4	$[w_5 + 2w_3, w_5 + 8\delta_5]$	1.0632
Step 5	$[w_5 + 3\delta_5, w_5 + 2w_3 + 5\delta_5]$	1.0618
	$[w_5 + 2\delta_5, w_5 + 4w_3 + 4\delta_5]$	1.0636
Step 6	$[w_5 + 4\delta_5, w_5 + 7\delta_5]$	1.0636
Step 7	$[w_5 + 4\delta_5, w_5 + 5\delta_5 + w_3]$	1.0646
Step 8	$O^*((1.3279^{1/w_5})^\mu)$	1.0584
Step 9	$[w_4 + 2w_3, w_4 + 6\delta_4]$	1.0646
Step 10	$[w_4 + 2\delta_4, w_4 + 6\delta_4]$	1.0646
Step 11	$O^*((1.1279^{1/w_3})^\mu)$	1.0646

The best choice of w_i can be found by solving a quasi-convex program problem.

$$\Downarrow$$

$$c^\mu \leq c^{\mu-(w_6+\delta_6)} + c^{\mu-(w_6+11\delta_6)}$$
$$c^\mu \leq c^{\mu-(w_5+2w_3)} + c^{\mu-(w_5+8\delta_5)}$$
$$\cdots$$
$$c^\mu \leq c^{\mu-(w_4+2\delta_4)} + c^{\mu-(w_4+6\delta_4)}$$
$$w_1 = w_2 = 0, w_4 = 2w_3, \ldots$$

Some other assumptions as constraints

The Final Result

Table 2. The weight setting

$w_1 = w_2 = 0$	
$w_3 = 1.9234132344759123$	$\delta_3 = 1.9234132344759123$
$w_4 = 3.8468264689518246$	$\delta_4 = 1.9234132344759123$
$w_5 = 5$	$\delta_5 = 1.1531735310481754$
$w_i = i (i \geq 6)$	$\delta_i = 1 (i \geq 6)$

Table 3. The branching vector and factor for each step

Steps	Branching vectors	Branching factors
Step 3	$[w_6 + \delta_6, w_6 + 11\delta_6]$	1.0636
Step 4	$[w_5 + 2w_3, w_5 + 8\delta_5]$	1.0632
Step 5	$[w_5 + 3\delta_5, w_5 + 2w_3 + 5\delta_5]$ $[w_5 + 2\delta_5, w_5 + 4w_3 + 4\delta_5]$	1.0618 1.0636
Step 6	$[w_5 + 4\delta_5, w_5 + 7\delta_5]$	1.0636
Step 7	$[w_5 + 4\delta_5, w_5 + 5\delta_5 + w_3]$	1.0646
Step 8	$O^*((1.3279^{1/w_5})^\mu)$	1.0584
Step 9	$[w_4 + 2w_3, w_4 + 6\delta_4]$	1.0646
Step 10	$[w_4 + 2\delta_4, w_4 + 6\delta_4]$	1.0646
Step 11	$O^*((1.1279^{1/w_3})^\mu)$	1.0646

The best choice of w_i can be found by solving a quasi-convex program problem.

$$\Downarrow$$

$$c^\mu \leq c^{\mu-(w_6+\delta_6)} + c^{\mu-(w_6+11\delta_6)}$$
$$c^\mu \leq c^{\mu-(w_5+2w_3)} + c^{\mu-(w_5+8\delta_5)}$$
$$\cdots$$
$$c^\mu \leq c^{\mu-(w_4+2\delta_4)} + c^{\mu-(w_4+6\delta_4)}$$
$$w_1 = w_2 = 0, w_4 = 2w_3, \ldots$$

Some other assumptions as constraints

Minimize c

The Final Result

Table 2. The weight setting

$w_1 = w_2 = 0$	
$w_3 = 1.9234132344759123$	$\delta_3 = 1.9234132344759123$
$w_4 = 3.8468264689518246$	$\delta_4 = 1.9234132344759123$
$w_5 = 5$	$\delta_5 = 1.1531735310481754$
$w_i = i (i \geq 6)$	$\delta_i = 1 (i \geq 6)$

Table 3. The branching vector and factor for each step

Steps	Branching vectors	Branching factors
Step 3	$[w_6 + \delta_6, w_6 + 11\delta_6]$	1.0636
Step 4	$[w_5 + 2w_3, w_5 + 8\delta_5]$	1.0632
Step 5	$[w_5 + 3\delta_5, w_5 + 2w_3 + 5\delta_5]$	1.0618
	$[w_5 + 2\delta_5, w_5 + 4w_3 + 4\delta_5]$	1.0636
Step 6	$[w_5 + 4\delta_5, w_5 + 7\delta_5]$	1.0636
Step 7	$[w_5 + 4\delta_5, w_5 + 5\delta_5 + w_3]$	1.0646
Step 8	$O^*((1.3279^{1/w_5})^\mu)$	1.0584
Step 9	$[w_4 + 2w_3, w_4 + 6\delta_4]$	1.0646
Step 10	$[w_4 + 2\delta_4, w_4 + 6\delta_4]$	1.0646
Step 11	$O^*((1.1279^{1/w_3})^\mu)$	1.0646

The best choice of w_i can be found by solving a quasi-convex program problem.

$$\Downarrow$$

$$c^\mu \leq c^{\mu-(w_6+\delta_6)} + c^{\mu-(w_6+11\delta_6)}$$
$$c^\mu \leq c^{\mu-(w_5+2w_3)} + c^{\mu-(w_5+8\delta_5)}$$
$$\cdots$$
$$c^\mu \leq c^{\mu-(w_4+2\delta_4)} + c^{\mu-(w_4+6\delta_4)}$$
$$w_1 = w_2 = 0, w_4 = 2w_3, \ldots$$

Some other assumptions as constraints

Minimize c

Bottleneck: Step 7, 9, 10, 11.

The Final Result

Theorem

Our algorithm SAT($\mathcal{F}$) solves the SAT problem in $O^*(1.0646^L)$ time.

Thanks for listening!