Good but still Exp Algorithms for 3-SAT and MIS

Exposition by William Gasarch
This talk is based on parts of the following AWESOME books:

**The Satisfiability Problem SAT, Algorithms and Analyzes**
by
Uwe Schoning and Jacobo Torán

**Exact Exponential Algorithms**
by
Fedor Formin and Dieter Kratsch
What is 3SAT?

**Definition:** A Boolean formula is in 3CNF if it is of the form

\[ C_1 \land C_2 \land \cdots \land C_k \]

where each \( C_i \) is an \( \lor \) of three or less literals.

**Definition:** A Boolean formula is in 3SAT if it in 3CNF form and is also SATisfiable.
OUR GOAL

We will show algorithms for 3SAT that

1. Run in time $O(\alpha^n)$ for various $\alpha < 1$. Some will be randomized algorithms. **NOTE:** By $O(\alpha^n)$ we really mean $O(p(n)\alpha^n)$ where $p$ is a poly. We ignore such factors.

2. Quite likely run even better in practice, or modifications of them do.
2SAT is in P:
Convention For All of our Algorithms

Definition:
1. A Unit Clause is a clause with only one literal in it.
2. A Pure Literal is a literal that only shows up as non negated or only shows up as negated.

Conventions:
1. If have unit clause immediately assign its literal to TRUE.
2. If have POS-pure literal then immediately assign it to be TRUE.
3. If have NEG-pure literal then immediately assign it to be FALSE.
4. If we have a partial assignment \( z \).
   4.1 If \( \forall C \)[\( C(z) = TRUE \)] then output YES.
   4.2 If \( \exists C \)[\( C(z) = FALSE \)] then output NO.

META CONVENTION: Abbreviate doing this STAND (for STANDARD).
DPLL (Davis-Putnam-Logemann-Loveland) ALGORITHM

ALG($F$: 3CNF $\text{fml}$; $z$: Partial Assignment)

STAND
Pick a variable $x$ (VERY CLEVERLY)

$\text{ALG}(F; z \cup \{x = T\})$

$\text{ALG}(F; z \cup \{x = F\})$
Key Idea Behind Recursive 7-ALG

**KEY1:** If $F$ is a 3CNF formula and $z$ is a partial assignment either

1. $F(z) = \text{TRUE}$, or
2. there is a clause $C = (L_1 \lor L_2)$ or $(L_1 \lor L_2 \lor L_3)$ that is not satisfied. (We assume $C = (L_1 \lor L_2 \lor L_3)$.)

**KEY2:** In ANY extension of $z$ to a satisfying assignment ONE of the 7 ways to make $(L_1 \lor L_2 \lor L_3)$ true must happen.
Recursive-7 ALG

ALG($F$: 3CNF $\text{fml}$; $z$: Partial Assignment)

STAND
if $F(z)$ in 2CNF use 2SAT ALG
find $C = (L_1 \lor L_2 \lor L_3)$ a clause not satisfied for all 7 ways to set $(L_1, L_2, L_3)$ so that $C=\text{TRUE}$
Let $z'$ be $z$ extended by that setting
ALG($F; z'$)

VOTE: IS THIS BETTER THAN $O(2^n)$?
Recursive-7 ALG

ALG($F$: 3CNF $fml$; $z$: Partial Assignment)

STAND
if $F(z)$ in 2CNF use 2SAT ALG
find $C = (L_1 \lor L_2 \lor L_3)$ a clause not satisfied
for all 7 ways to set $(L_1, L_2, L_3)$ so that $C = \text{TRUE}$
Let $z'$ be $z$ extended by that setting
ALG($F; z'$)

VOTE: IS THIS BETTER THAN $O(2^n)$?
IT IS!
The Analysis

\[ T(0) = O(1) \]
\[ T(n) = 7T(n - 3). \]

so
\[ T(n) = 7^{n/3}O(1) = O(((7^{1/3})^n)) = O((1.913)^n) \]

1. Good News: BROKE the \(2^n\) barrier. Hope for the future!
2. Bad News: Still not that good a bound.
Key Ideas Behind Recursive-3 ALG

KEY1: Given $F$ and $z$ either:

1. $F(z) = TRUE$, or
2. there is a clause $C = (L_1 \lor L_2)$ or $(L_1 \lor L_2 \lor L_3)$ that is not satisfied. (We assume $C = (L_1 \lor L_2 \lor L_3)$.)

KEY2: in ANY extension of $z$ to a satisfying assignment either:

1. $L_1$ TRUE.
2. $L_1$ FALSE, $L_2$ TRUE.
3. $L_1$ FALSE, $L_2$ FALSE, $L_3$ TRUE.
Recursive-3 ALG

ALG($F$: 3CNF $\text{fm}$; $z$: Partial Assignment)

STAND
if $F(z)$ in 2CNF use 2SAT ALG
find $C = (L_1 \lor L_2 \lor L_3)$ a clause not satisfied
ALG($F; z \cup \{L_1 = T\}$)
ALG($F; z \cup \{L_1 = F, L_2 = T\}$)
ALG($F; z \cup \{L_1 = F, L_2 = F, L_3 = T\}$)

VOTE: IS THIS BETTER THAN $O((1.913)^n)$?
Recursive-3 ALG

ALG($F$: 3CNF $fml$; $z$: Partial Assignment)

STAND
if $F(z)$ in 2CNF use 2SAT ALG
find $C = (L_1 \lor L_2 \lor L_3)$ a clause not satisfied
ALG($F; z \cup \{ L_1 = T \}$)
ALG($F; z \cup \{ L_1 = F, L_2 = T \}$)
ALG($F; z \cup \{ L_1 = F, L_2 = F, L_3 = T \}$)

VOTE: IS THIS BETTER THAN $O((1.913)^n)$?
IT IS!
The Analysis

\[ T(0) = O(1) \]
\[ T(n) = T(n - 1) + T(n - 2) + T(n - 3). \]
\[ T(n) = O((1.84)^n). \]
So Where Are We Now?

1. Good News: BROKE the \((1.913)^n\) barrier. Hope for the future!

2. Bad News: \((1.84)^n\) Still not that good. Good News: Can modify to work better in theory!!
**Definition:** If $F$ is a fml and $z$ is a partial assignment then $z$ is COOL if every clause that $z$ affects is made TRUE.

**BILL:** Do examples and counterexamples.

Prove to yourself:

**Lemma:** Let $F$ be a 3CNF fml and $z$ be a partial assignment.

1. If $z$ is COOL then $F \in 3SAT$ iff $F(z) \in 3SAT$.
2. If $z$ is NOT COOL then $F(z)$ will have a clause of length 2.
ALG($F$: 3CNF fml, $z$: partial assignment)

COMMENT: This slide is when a 2CNF clause not satisfied.

STAND
if $(\exists C = (L_1 \lor L_2))$ not satisfied then
    $z_1 = z \cup \{L_1 = T\}$
    if $z_1$ is COOL then ALG($F;z_1$)
    else
        $z_{01} = z \cup \{L_1 = F, L_2 = T\}$
        if $z_{01}$ is COOL then ALG($F;z_{01}$)
        else
            ALG($F;z_1$)
            ALG($F;z_{01}$)
else (COMMENT: The ELSE is on next slide.)
(COMMENT: This slide is when a 3CNF clause not satisfied)

if \((\exists C = (L_1 \lor L_2 \lor L_3))\) not satisfied then

\[ z_1 = z \cup \{L_1 = T\} \]

if \(z_1\) is COOL then \(\text{ALG}(F; z_1)\)
else

\[ z_{01} = z \cup \{L_1 = F, L_2 = T\} \]

if \(z_{01}\) is COOL then \(\text{ALG}(F; z_{01})\)
else

\[ z_{001} = z \cup \{L_1 = F, L_2 = F, L_3 = T\} \]

if \(z_{001}\) is COOL then \(\text{ALG}(F; z_{001})\)
else

\(\text{ALG}(F; z_1)\)
\(\text{ALG}(F; z_{01})\)
\(\text{ALG}(F; z_{001})\)
IS IT BETTER?

VOTE: IS THIS BETTER THAN $O((1.84)^n)$?
IS IT BETTER?

VOTE: IS THIS BETTER THAN $O((1.84)^n)$?
IT IS!
IT IS BETTER!

**KEY1:** If any of $z_1$, $z_{01}$, $z_{001}$ are COOL then only ONE recursion: $T(n) = T(n-1) + O(1)$.

**KEY2:** If NONE of the $z_0$, $z_{01}$ $z_{001}$ are COOL then ALL of the recurrences are on fml’s with a 2CNF clause in it.

$T(n)= $ Time alg takes on 3CNF formulas.

$T'(n)= $ Time alg takes on 3CNF formulas that have a 2CNF in them.

$T(n) = \max\{ T(n-1), T'(n-1) + T'(n-2) + T'(n-3) \}.$

$T'(n) = \max\{ T(n-1), T'(n-1) + T'(n-2) \}.$

Can show that worst case is:

$T(n) = T'(n-1) + T'(n-2) + T'(n-3).$

$T'(n) = T'(n-1) + T'(n-2).$
The Analysis

\[ T'(0) = O(1) \]
\[ T'(n) = T'(n - 1) + T'(n - 2). \]

\[ T'(n) = O((1.618)^n). \]

So

\[ T(n) = O(T(n)) = O((1.618)^n). \]

**VOTE:** Is better known?

**VOTE:** Is there a proof that these techniques cannot do any better?
**Definition** If \( x, y \) are assignments then \( d(x, y) \) is the number of bits they differ on.

**BILL: DO EXAMPLES**

**KEY TO NEXT ALGORITHM:** If \( F \) is a fml on \( n \) variables and \( F \) is satisfiable then either

1. \( F \) has a satisfying assignment \( z \) with \( d(z, 0^n) \leq n/2 \), or
2. \( F \) has a satisfying assignment \( z \) with \( d(z, 1^n) \leq n/2 \).
HAM ALG

HAMALG($F$: 3CNF fml, $z$: full assignment, $h$: number) $h$ bounds $d(z, s)$ where $s$ is SATisfying assignment $h$ is distance

STAND
if $\exists C = (L_1 \lor L_2)$ not satisfied then
   $\text{ALG}(F; z \oplus \{L_1 = T\}; h - 1)$
   $\text{ALG}(F; z \oplus \{L_1 = F, L_2 = T\}; h - 1)$

if $\exists C = (L_1 \lor L_2 \lor L_3)$ not satisfied then
   $\text{ALG}(F; z \oplus \{L_1 = T\}; h - 1)$
   $\text{ALG}(F; z \oplus \{L_1 = F, L_2 = T\}; h - 1)$
   $\text{ALG}(F; z \oplus \{L_1 = F, L_2 = F, L_3 = T\}; h - 1)$
HAMALG($F; 0^n; n/2$)
If returned NO then HAMALG($F; 1^n; n/2$)

**VOTE:** IS THIS BETTER THAN $O((1.61)^n)$?
HAMALG\((F; 0^n; n/2)\)
If returned NO then HAMALG\((F; 1^n; n/2)\)

**VOTE:** IS THIS BETTER THAN $O((1.61)^n)$?
**IT IS NOT!** Work it out in groups anyway NOW.
KEY: We don’t care about how many vars are assigned since they all are. We care about $h$.

$T(0) = 1$.

$T(h) = 3 \cdot T(h - 1)$.

$T(h) = 3^i \cdot T(h - i)$.

$T(h) = 3^h$.

$T(n/2) = 3^{n/2} = O((1.73)^n)$.
KEY TO HAM ALGORITHM: Every element of \([0, 1]^n\) is within \(n/2\) of either \(0^n\) or \(1^n\).

Definition: A covering code of \([0, 1]^n\) of SIZE \(s\) with RADIUS \(h\) is a set \(S \subseteq \{0, 1\}^n\) of size \(s\) such that

\[
(\forall x \in \{0, 1\}^n)(\exists y \in S)[d(x, y) \leq h].
\]

Example: \([0^n, 1^n]\) is a covering code of SIZE 2 of RADIUS \(n/2\).
ASSUME ALG

Assume we have a Covering code of \(\{0, 1\}^n\) of size \(s\) and radius \(h\). Let Covering code be \(S = \{v_1, \ldots, v_s\}\).

\(i = 1\)
FOUND = FALSE
while (FOUND = FALSE) and \((i \leq s)\)
    HAMALG(\(F; v_i; h\))
    If returned YES then FOUND = TRUE
    else
        \(i = i + 1\)
end while
Each iteration satisfies recurrence

\[ T(0) = 1 \]
\[ T(h) = 3 T(h - 1) \]
\[ T(h) = 3^h. \]

And we do this \( s \) times.

**ANALYSIS:** \( O(s3^h) \).

Need covering codes with small value of \( O(s3^h) \).
RECAP: Need covering codes of size $s$, radius $h$, with small value of $O(s^{3^h})$. 
IN SEARCH OF A GOOD COVERING CODE

RECAP: Need covering codes of size $s$, radius $h$, with small value of $O(s^{3^h})$.
THATS NOT ENOUGH: We need to actually CONSTRUCT the covering code in good time.
RECAP: Need covering codes of size $s$, radius $h$, with small value of $O(s^{3h})$.
THATS NOT ENOUGH: We need to actually CONSTRUCT the covering code in good time.
YOU”VE BEEN PUNKED: We’ll just pick a RANDOM subset of $\{0,1\}^n$ and hope that it works.
RECAP: Need covering codes of size $s$, radius $h$, with small value of $O(s3^h)$.

THATS NOT ENOUGH: We need to actually CONSTRUCT the covering code in good time.

YOU’VE BEEN PUNKED: We’ll just pick a RANDOM subset of $\{0, 1\}^n$ and hope that it works.

SO CRAZY IT MIGHT JUST WORK!
IN SEARCH OF A GOOD COVERING CODE-RANDOM!

CAN find with high prob a covering code with

- Size $s = n^2 2^{1.4063n}$
- Distance $h = 0.25n$.

Can use to get SAT in $O((1.5)^n)$.

Note: Best known: $O((1.306)^n)$.  
What is Maximum Ind Set?

**Definition:** If $G = (V, E)$ is a graph then $I \subseteq V$ is an *Ind. Set* if $(\forall x, y \in V)[(x, y) \notin E]$. The set $I$ is a MAXIMUM IND SET if it is an Ind Set and there is NO ind set that is bigger.

**Goal:** Given a graph $G$ we want the SIZE of the Maximum Ind. Set. Obtaining the set itself will be an easy modification of the algorithms which we will omit.

**Abbreviation:** MIS is the Maximum Ind Set problem.
OUR GOAL

1. Will we show that MIS is in P?
OUR GOAL

1. Will we show that MIS is in P?
   
   NO.

2. We will show algorithms for MIS that
   
   2.1 Run in time $O(\alpha^n)$ for various $\alpha < 1$. NOTE: By $O(\alpha^n)$ we really mean $O(p(n)\alpha^n)$ where $p$ is a poly. We ignore such factors.

   2.2 Quite likely run even better in practice.
If all of the degrees are $\leq 2$ then the problem is EASY.
(WE OMIT)
If $G = (V, E)$ is a graph and $v \in V$ then

$$N[v] = \{v\} \cup \{u \mid (v, u) \in E\}.$$ 

The NEIGHBORS of $v$ AND $v$ itself.
MIN DEG ALGORITHM

ALG(G = (V, E): A Graph)

v = vertex of min degree
for u ∈ N[v]
    \( m_u = ALG(G - N[m_u]) \)
\( m = \min\{m_u | u \in N[v]\} \).
RETURN(1 + m)
Let $N[v] = \{v, x_1, \ldots, x_{d(v)}\}$.

\[
T(n) \leq 1 + T(n - d(v) - 1) + \sum_{i=1}^{d(v)} T(n - d(x_i) - 1) \\
\leq 1 + T(n - d(v) - 1) + \sum_{i=1}^{d(v)} T(n - d(v) - 1) \\
\leq 1 + (d(v) + 1) T(n - (d(v) + 1))
\]

1. Runs in $T(n) = O((3^{1/3})^n) \leq O((1.42)^n)$.

2. Works well on high degree graphs until they become low degree graphs.


4. Makes more sense to take High degree nodes.
MAX DEG ALG

ALG(G)

1. If ($\exists v)[d(v) = 0]$ then RETURN($1 + ALG(G - v)$).
2. If ($\exists v)[d(v) = 1]$ then RETURN($1 + ALG(G - N[v])$).
3. If ($\forall v)[d(v) \leq 2$] then CALL 2-MIS ALG.
4. If ($\exists v)[d(v) \geq 3$] then
   4.1 Let $v^*$ be of max degree
   4.2 Return MAX of $1 + ALG(G - N[v^*]), ALG(G - v^*)$. 
ANALYSIS

\[ T(n) \leq T(n - d(v) - 1) + T(n - 1) \]
\[ T(n) \leq T(n - 4) + T(n - 1) \]

1. Runs in \( T(n) = O((1.38)^n) \).
2. Works well on high degree graphs until they become low degree graphs. But better than Min-Degree alg.
3. WORKS really well in practice.
Need to MEASURE progress better.

1. We measure a node of degree \( \leq 1 \) as having weight ZERO.
2. We measure a node of degree 2 as having weight \( \frac{1}{2} \).
3. We measure a node of degree \( \geq 3 \) as having weight ONE.

SO we view \( |V| \) as

\[
\frac{1}{2} (\text{number of verts of degree 2}) + (\text{number of verts of degree 3})
\]

We still refer to this as \( n \).
BETTER ANALYSIS

Have picked $v^*$.

1. Assume there are no vertices of degree $\leq 1$ (else would not be in $v^*$ case)
2. Assume $v^*$ has $d_2$ vertices of degree 2.
3. Assume $v^*$ has $d_3$ vertices of degree 3.
4. Assume $v^*$ has $d_{\geq 4}$ vertices of degree $\geq 4$. 
BETTER ANALYSIS OF \( G - N[v] \) CASE

\( G - N[v^*] \):

1. Loss of \( v^* \) is loss of 1.
2. Loss of \( d_2 \) vertices of degree 2: Loss is \( \frac{d_2}{2} \).
3. Loss of \( d_3 \) vertices of degree 3: Loss is \( d_3 \).
4. Loss of \( d_{\geq 4} \) vertices of degree \( \geq 4 \): Loss is \( d_{\geq 4} \).

Total Loss: \( 1 + \frac{d_2}{2} + d_3 + d_{\geq 4} \).

Work to do:

\[
T(n - (1 + \frac{d_2}{2} + d_3 + d_{\geq 4}))
\]
BETTER ANALYSIS OF $G - v$ CASE

$G - v^*$:

1. Loss of $v^*$ is loss of 1.
2. The $d_2$ verts of deg 2 become $d_2$ verts of deg $\leq 1$. Loss is $\frac{d_2}{2}$.
3. The $d_3$ verts of deg 3 become $d_3$ verts of deg $\leq 2$. Loss is $\frac{d_3}{2}$.
4. The $d_{\geq 4}$ verts of deg $\geq 4$. No Loss.

Total Loss: $1 + \frac{d_2}{2} + \frac{d_3}{2}$.

Work to do:

$$T(n - (1 + \frac{d_2}{2} + \frac{d_3}{2}))$$
TOTAL ANALYSIS

\[ T(n) \leq T(n - (1 + \frac{d_2}{2} + d_3 + d_{\geq 4})) + T(n - (1 + \frac{d_2}{2} + \frac{d_3}{2})) \]
\[ \leq T(n - 1) + T(n - (1 + d_2 + \frac{3d_3}{2} + d_{\geq 4})) \]
\[ \leq T(n - 1) + T(n - (d(v^*) + 1)) \]

1. If \( d(v^*) \geq 4 \) then get

\[ T(n) \leq T(n - 1) + T(n - 5) \]

2. If \( d(v^*) = 3 \) then get

\[ T(n) \leq T(n - 1) + T(n - 4) \]
HOW GOOD?

1. Runs in $T(n) \leq O((1.3248)^n)$.
2. Using Deg2 weight 0.596601, Deg3 weigh 0.928643, Deg4 weight 1 can get $O((1.2905)^n)$.
3. Works well on high degree graphs until they become low degree graphs. But better than Min-Degree alg.
4. WORKS really well in practice, and this analysis may say why.
Best known runs in time

\[ O((1.2109)^n). \]

1. Order constant is REASONABLE.
2. LOTS of cases depending on degree.
3. Sophisticated analysis.