A Polynomial Time Algorithm for 3SAT

ROBERT QUIGLEY

This paper includes the presentation of a polynomial time algorithm to solve 3SAT. It begins with the problem definition and format of various parts of the problem. It then offers lemmas of important aspects of the 3SAT problem along with their corresponding proofs. Next, it describes the algorithm that uses these lemmas to solve 3SAT in polynomial times. And finally, it proves that the algorithm works for any general case of 3SAT. The algorithm relies on the fact that an instance of 3SAT is unsatisfiable iff contradicting 1-terminal clauses can be derived in polynomial time.

Additional Key Words and Phrases: test1, test2

ACM Reference Format:
Robert Quigley. 2024. A Polynomial Time Algorithm for 3SAT. 1, 1 (January 2024), 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

TODO: find who the heck wrote '3SAT is NP-complete' read https://dl.acm.org/doi/10.1145/800157.805047

1 DEFINITIONS

Definition 1.1. Terminals: symbols used in the 3SAT problem that can be assigned a value of either 0 or 1, True or False, or any other binary assignment. They usually take the form x_i where i is a natural number.

Definition 1.2. Terms: terminals in either the positive or negated form. If a term is positive, then the value of the terminal will be the same as the value of the term. If a term is negated, then the value of the terminal will be the opposite of the value of the term.

Definition 1.3. Clauses: a set of terms combined by logical or operators. Clauses usually take the form $(x_i \vee \neg x_j \vee x_k)$ where $x_i \neq x_j \neq x_k$

Definition 1.4. (3SAT) Instance: a set of any number of clauses combined by logical and operators. Note that entire clauses may not be negated. Terminals may not repeat within a clause, but they are free to repeat between clauses. Instances usually take the form: $(x_i \vee \neg x_j \vee x_k) \wedge (x_l \vee x_m \vee x_n)$

Definition 1.5. Assignment: A list of values in which each value represents either True or False such that each item in the list corresponds to a terminal and all terminals are assigned a value.

Definition 1.6. Satisfying Assignment: An assignment, A, is said to satisfy the instance, if applying A will make the instance evaluate to True.

Author's address: Robert Quigley, robertquigley21@gmail.com.

Manuscript submitted to ACM

Definition 1.7. The 3SAT Problem: Given a 3SAT instance, does there exist a satisfying assignment?

Definition 1.8. Satisfiable: an instance is satisfiable iff there exists a satisfying assignment.

Definition 1.9. Unsatisfiable: an instance is unsatisfiable iff there does not exist a satisfying assignment.

Definition 1.10. Blocking an Assignment: An assignment, A, is said to be blocked if, given a clause, C, there is no way that A can satisfy the instance.

Definition 1.11. TODO: define a more 'specifically in polytime' type of implication Implication: A clause, C, is said to imply another clause, D, if all assignments blocked by D are also blocked by C

Definition 1.12. Given Clauses: clauses which were given in the original instance.

Definition 1.13. Implied Clauses: clauses which are implied by the clauses in the original instance.

Definition 1.14. k-terminal (k-t) clause: a clause is described as a k-terminal or a k-t clause if there are k terminals in the clause

Definition 1.15. Reduction: A special type of implication in which the implied clause is shorter than the implying clause(s).

Definition 1.16. Expansion: A special type of implication in which the implied clause is longer than the implying clause(s).

Definition 1.17. Contradicting Clauses: A set of clauses is said to be contradicting if they block every assignment

2 REFORMATTING

Since there are a lot of constant characteristics about an instance of 3SAT, we can remove most of them to allow ourselves to focus only on what changes from instance to instance. A list of unchanging characteristics follows:

- the symbol x
- logical and operators
- logical or operators

The only difference between instances, therefore, is the subscript of the terminal. The following items will be changed to improve compatibility with python code: *possible footnote to describe how to represent it as list of lists

- parentheses will become square brackets
- negation symbols will become minus signs
- an instance may be surrounded with square brackets to show it is a list of lists

For example, the instance:
$(\neg x_a \vee x_b \vee x_c) \wedge (x_a \vee x_d \vee x_e)$
will be written as:
$[[-a, b, c], [a, d, e]]$

3 LEMMAS

LEMMA 3.1. *For a given instance with n terminals, there are 2^n possible assignments.*

PROOF. An assignment for this instance consists of n values, each with two possible values, True or False. Therefore, there are 2^n possible assignments. □

LEMMA 3.2. *If a clause contains the same terminal in its negated and positive form, it will always be true.*

PROOF. Given: a clause containing the same terminal in both the positive and negative form.

We know that terminal must either be True or False. Consider both cases:

That terminal's value is True: The positive form of the terminal will be true and the clause will evaluate to true

That terminal's value is False: The negative form of the terminal will be true and the clause will evaluate to true □

LEMMA 3.3. *For a given instance with n terminals, there are $8 * \binom{n}{3}$ possible 3-terminal clauses that block at least one instance.*

PROOF. Recall from Lemma 3.2 that a clause containing the same terminal in both the positive and negated form will always be true.

Therefore, it will never block any assignments.

Recall also that an instance containing the same terminal in the same form twice is not a valid 3SAT clause.

Therefore we do not have to consider clauses containing the same terminal twice since we are concerned with clauses that block assignments.

Given an instance with n terminals, choose three of them to make a clause

Once the terminals are selected, each terminal can have one of two values, either True or False

There are $2 * 2 * 2$ possible clauses based on these terminals

Since there are 8 clauses for every selection of three terminals, there are $8 * \binom{n}{3}$ possible clauses □

LEMMA 3.4. *A clause can block an assignment*

PROOF. Recall the definition of a clause blocking an assignment: An assignment is blocked if it does not allow the clause to evaluate to True.

Since terms in a clause are combined by logical OR operators, a clause cannot evalute to True iff all terms evalute to False.

A term evaluates to False if it's either (1) negated and the terminal's value is True or (2) positive and the terminal's value is False

Given a clause, we know there are three unique terminals and we can find assignments where all terms evalute to False.

This would force the clause to evalute to False.

Since the instance is composed of clauses combined by logical AND operators and one clause evalutes to False, then the entire instance evalutes to False.

Since the instance evaluates to False, the assignment cannot satisfy the instance.

□

Lemma 3.5. *Each assignment can be blocked by at most* $\binom{n}{3}$ *3-terminal clauses*

Proof. Consider an assignment, A. WTS it is blocked by $\binom{n}{3}$ clauses.

Recall from Lemma 3.4 that an assignment is blocked based on values of the three terminals in the clause blocking it.

Notice also that each of these values only has one possible option since assignments have a fixed value for each terminal.

Recall assignments are of length n.

We can choose three of the n values in the assignments to fix and make a clause out of it in the following manner:

if the value is False, the term is positive

if the value is True, the term is negated

This will create a clause that blocks the assignment.

This allows us to have $\binom{n}{3}$ possible clauses that block an assignment.

□

Lemma 3.6. *Each clause of length k blocks* 2^{n-k} *assignments*

Proof. In this proof, I will use a, b, c instead of x_a, x_b, x_c and I will use 0 and 1 instead of False and True

Now, consider the general k-terminal clause, C in an instance with n terminals.

As seen in Lemma 3.4, this blocks all assignments where all of the terms evaluate to False

Since the values for k terminals are set, there are $n - k$ terminals left whose values could be 0 or 1.

Since an assignment exists for every possible way to assign values to each terminal, we know an assignment exists for every possible way to assign a value for these $n - k$ terminals

Since the values of k terminals are fixed, there are 2^{n-k} unique assignments that are blocked by C. □

Lemma 3.7. *Adding clauses to an unsatisfiable instance will never make it satisfiable. Similarly, removing clauses from a satisfiable instance will never make it unsatisfiable.*

Proof. 1. Recall that clauses block assignments and an instance is unsatisfiable if none of the possible assignments satisfy the instance, ie, an instnace is unsatifiable if all of the assignments are blocked.

Therefore, adding clauses to an instance in which all assignments are blocked will never be able to unblock an assignment and it will remain unsatisfiable.

2. An instance is satisfiable if there exists a satisfying assignment, ie, an instance is satisfiable if there is at least one assignment that's not blocked.

Therefore, removing clauses will never be able to block that assignment and the instance will remain satisfiable. □

Lemma 3.8. *For any clause, C, if we select a terminal, t, that's not in C then half of the assignments blocked by C will assign True to t and the other half will assign False to t*

Proof. We have a clause, C, of arbitrary length:

[1, 2, 3, ...]

Now we select a terminal, t, that's not in C, say we call this terminal t

WTS half of the assignments blocked by C assign True to t and the other half assign False to t.

We know there will be no overlap between these assignments because a single assignment cannot assign both the values True and False to the same terminal.

Now we just have to show that exactly half of the assignments are blocked by assigning either True or False to t. Let's say we assign True to t.

Let's say there are k terms in C, then we know it blocks assignments where all of the terms evaluate to False.

By Lemma 3.6, this clause blocks 2^{n-k} assignments.

If we fix the calue of t to True, then there are only $n-k-1$ terminals whose values could be 0 or 1. Since we have two choices per terminal and there are $n-k-1$ terminals, then there are 2^{n-k-1} assignments blocked by C where the value of t is fixed.

Divide

$2^{n-k-1}/2^{n-k}$

$= 2^{n-k-1-(n-k)}$

$= 2^{-1}$

$= 1/2$

This shows that half of the assignments blocked by C assign a fixed value to t.

Recall that a terminal can be assigned the value True or False.

Therefore, for any clause, C, if we select a terminal, t, that's not in C then half of the assignments blocked by C will assign True to t and the other half will assign False to C.

□

Lemma 3.9. *Given a clause of length k, say C, all of the assignments blocked by another clause, say D, which contains all of the terms in C, are also blocked by C.*

Proof. Given a clause, C, of an arbitrary length, say k:

[1, 2, 3, ...]

And another clause, D, containing all the terms in C with possibly additional terms, say of length l:

[1, 2, 3, ... a, b, c, ...]

WTS all the assignments blocked by D are also blocked by C.

We know that C blocks all assignments that cause all the terms to evaluate to False.

We know there are 2^{n-k} remaining terminals, each with a possible value of True or False.

For each of the terminals in D that are not in C, we know the value is fixed in the assignments blocked by D.

Since there are 2^{n-k} remaining terminals, each with a possible value of True and False, and the assignments blocked by D have set values for potentially up to the rest of the terminals, all of the assignments blocked by D are also blocked by C □

Lemma 3.10. *Two clauses of length k that share all the same terms, except for one in which the same terminal is positive in one clause and negated in the other, implies a clause of length k-1 which is composed of all the shared terminals.*

Proof. Given clauses consistent with the description:

A := [1, 2, 3, ... i] B := [1, 2, 3, ..., -i]

where 1, 2, 3, ... is shared between the clauses and i is a terminal not in 1, 2, 3, ...

Recall that the same terminal cannot appear twice within the same clause.

Consider the clause

C := [1, 2, 3, ...]

We know by Lemma 3.8 that if we select a terminal that's not in C, say t, then half of the assignments blocked by C assign True to t and the other half of the assignments blocked by C assign False to t

Let this terminal t that's not in C be the terminal i that's in A and B

We know that A blocks all assignments blocked by C where i is assigned the value of 0.

We know that B blocks all assignments blocked by C where i is assigned the value of 1.

Since A and B both block mutually exclusive halves of C, we can say that A and B imply C.

□

Lemma 3.11. *Given a clause, C, and a terminal, t, that's not in C, two new clauses can be implied consisting of all the terms of C appended to either the positive form of t or the negated form of t.*

Proof. Given a clause, C, and a terminal, t, that's not in C:

C := [a, b, c, ...]

By Lemma 3.8, half of the assignments blocked by C assign True to t and the other half of the assignments blocked by C assign False to t.

We can compose two new clauses:

D := [a, b, c, ..., t]

E := [a, b, c, ..., -t]

Since D and E both contain all the terminals from C, by Lemma 3.9 then D and E are implied by C.

□

Lemma 3.12 (Stronger Lemma 3.10). *If two clauses have the same terminal, t, such that t is positive in one clause and negated in the other, then these clauses imply a new clause which is composed of all the terms in both clauses that do not have t.*

Proof. Consider two clauses,

C := [1, 2, 3, ..., t]

D := [a, b, c, ..., -t]

Where within a clause, the same terminal does not repeat, but between clauses the same terminal can repeat.

WTS we can make a clause like this:

E := [1, 2, 3, ..., a, b, c, ...]

Let's define some additional clauses:

E' := [1, 2, 3, ..., a, b, c, ..., t]

E" := [1, 2, 3, ..., a, b, c, ..., -t]

By Lemma 3.9, we know that C implies E', ie, all assignments blocked by C are also blocked by E.

By Lemma 3.9, we know that D implies E".

By Lemma 3.10, since E' and E" share all the same terms except for t, which is positive in one clause and negated in the other, we can create a new clause composed of all the shared terms in E' and E".

Such a clause is already defined as E.

Now there are a couple extra cases to consider:

- There is some overlap between 1, 2, 3, ... and a, b, c, ...
- There is the same terminal that's positive in 1, 2, 3, ... and negated in a, b, c, ...

First, if the same term exists in 1, 2, 3, ... and a, b, c, ..., then we can just remove one of the duplicates since one term being true implies an identical term being True

Secondly, if the same terminal exists, but is of the opposite form in 1, 2, 3, ... and a, b, c, ... then by Lemma 3.2, this clause will always be True and thus blocks no assignments so C and D do not imply E, but this type of clause is worthless so we can simply discard it.

□

LEMMA 3.13. *Given two clauses of length k and l that share a terminal, t, which is positive in one clause and negated in the other, you will be able to directly imply clauses of length max(k, l)-1 to (k + l - 2)*

PROOF. The smallest clause that can be implied by clauses of length k and l occur when all of the terms in one clause exist in the other.

As such, the unique terms will come from the clause that's longer.

Removing t, you are left with 1 less than the maximum of k and l.

The largest clause can be implied if there are no terms shared between the two clauses. In this case you subtract 1 from the length of each clause to remove t and since no duplicates will be removed, the resulting clause's length is 2 less than the sum of the lengths of the clauses. □

LEMMA 3.14. *Given an instance of 3SAT, you can expand all of the given clauses to the point where you are considering clauses of length n.*

PROOF. TODO: PROVE □

LEMMA 3.15. *If you expand given 3-t clauses as described in 3.14, you will derive 2^n unique clauses of length n iff the instance is unsatisfiable.*

PROOF. TODO: PROVE □

LEMMA 3.16. *Using the algo,*

- *A 1-terminal clause can only be derived by a 2-t clause.*
- *A 2-t clause can only be derived by a 2-t or 3-t clause.*
- *A 3-t clause can only be derived by a 3-t or 4-t clause.*

PROOF. TODO Prove □

LEMMA 3.17. *Given four clauses of length k - 1, A, B, C, and D, and one clause of length k, E, such that*

- *A and B imply E*
- *C and E imply D*

Then A, B, and C, imply D without having to process a clause of length k

Proof. Given an implied clause of length k, say C:

[1, 2, 3, ...]

By the algorithm, it requires two clauses to imply another clause. Let's call these clauses D and E. The lengths of D and E can be:

D: length k

E: length ≥ 2

or

D: length k + 1

E: length any

Okay.

C, along with another clause, F of length m, can imply clauses of the following length:

- if k > m, [k-1, k+m-2]
- if m > k, [m-1, k+m-2]

Let the clause implied by C and F be called G.

Half of the assignments blocked by C are blocked by D and the other half are blocked by E.

Half of the assignments blocked by G are blocked by C and the other half are blocked by F.

WTS some combo of D, E, and F, imply G.

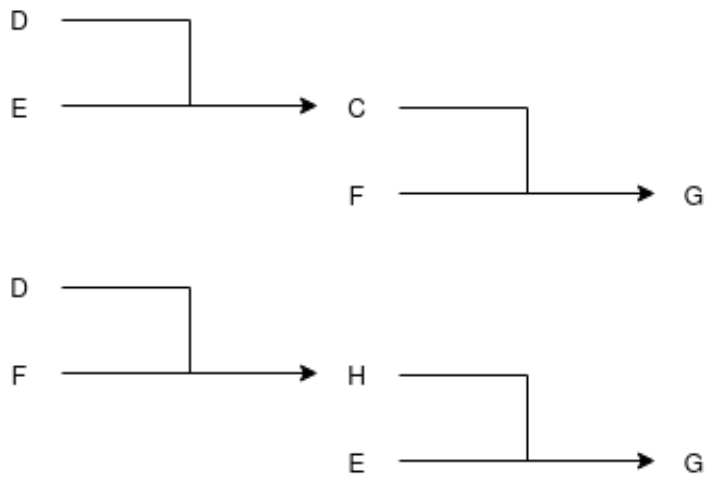
D + E

D + F

E + F

Known:

- All terms in C exist in either D or E
- D and E contain an opposite term, say i
- All terms in D or E that's not i exist in C
- F must contain an opposite term from C
- F must contain an opposite term from D or E that's not i
- D + F can imply another clause, say H
- E + F can imply another clause, say I
- H and I can derive G



Define the clauses in the following manner:

D := [a, b, ..., i]

E := [c, d, ... -i]

C := [a, b, ..., c, d, ...]

F := [-a, e, f, ...]

G := [b..., c, d, ..., e, f, ...]

H := [b, ..., i, e, f, ...]

Define lengths for the ... after b, d, and f

... after b: β

... after d: δ

... after f: ϕ

We know C is of length k and G is of length k - 1 or k.

Let lowercase alphabetical letters represent that terminal with a count of 1

Let letters in parentheses represent the count of terminals in both sets

Let length of G be k

$k = a + b + \beta + c + d + \delta - (\beta\delta)$ (by clause C)

$k = b + c + d + e + f + \beta + \delta + \phi - (\beta\delta) - (\beta\phi) - (\delta\phi) + (\beta\delta\phi)$ (by clause G)

$len(H) = b + i + e + f + \beta + \phi - (\beta\phi)$

$len(H) < k$

$b + i + e + f + \beta + \phi - (\beta\phi) < b + c + d + e + f + \beta + \delta + \phi - (\beta\delta) - (\beta\phi) - (\delta\phi) + (\beta\delta\phi)$

$\rightarrow i + \beta + \phi - (\beta\phi) < c + d + \beta + \delta + \phi - (\beta\delta) - (\beta\phi) - (\delta\phi) + (\beta\delta\phi)$

$\rightarrow i < c + d + \delta - (\beta\delta) - (\delta\phi) + (\beta\delta\phi)$

By Venn Diagram shenanigans, $\delta - (\beta\delta) - (\delta\phi) + (\beta\delta\phi)$, represents the number of terminals in δ not in β or ϕ

The lowest case for the right hand side of the equation is where this is 0, ie, all of the terminals in δ are in β or ϕ. In this case, the equation becomes:

$\rightarrow$ i $<$ c + d

which is guaranteed to be true since c and d both appear in E, they have to be different so the length of H will always be less than k, so you do not have to process clauses of length k to get the same implications.

However, you may have to process clauses up to length k - 1.

Let length of G be k - 1

This would imply $\rightarrow$ i $<$ c + d - 1

Alas this does not hold, so we get

$\rightarrow$ i = c + d - 1

Which means the length of H will be k - 1, which is still shorter than C, but is it short enough?

This is only valuable if D, E, and F are of length k - 1 because then this shows you will not have to process clauses of length k to get implications that that clause + another clause of length k - 1 will imply.

This proves that if we have four 3-t clauses, A, B, C, and D, and one 4-t clause, E st A and B -¿ E and C and E imply D, then A, B, and C will imply D without having to process a 4-terminal clause.

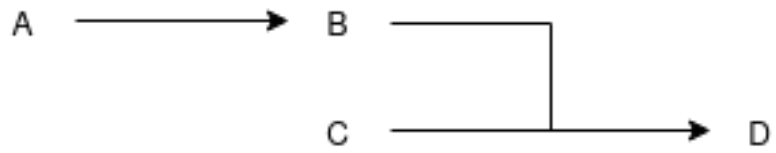
□

LEMMA 3.18. *Given the following clauses:*

- *A, of length shorter than k*
- *B, of length k*
- *C, of length shorter than k*
- *D, of length k or k - 1*
- *A expands to imply B*
- *B and C imply D*

Then A and C can imply D by processing clauses of at most length k - 1

PROOF. Consider the following case:



A := [a, b, ...]

B := [a, b, ..., c, d, ...]

C_1 := [-a, e, f, ...]

C_2 := [-c, e, f, ...]

D_1 := [b, ..., c, d, ..., e, f, ...]

D_2 := [a, b, ..., d, ..., e, f, ...]

E := [b, ..., e, f, ...]

A' := [a, b, ..., c]

F := [a, b, ..., e, f, ...]

Where C has to contain an opposite terminal that's either in A or not in A

Define the following:

β := set of terminals after b δ := set of terminals after d ϕ := set of terminals after f

Consider C_1 and D_1

Notice A and C_1 share an opposite term

Since B is of length k, we know

$k = a + b + c + d + \beta + \delta - \beta\delta$

Consider D is of length k

$k = b + c + d + e + f + \beta + \delta + \phi - \beta\delta - \beta\phi - \delta\phi + \beta\delta\phi$

WTS length of E is less than k

len(E) $= b + e + f + \beta + \phi - \beta\phi$

len(E) $<$ k

$b + e + f + \beta + \phi - \beta\phi < b + c + d + e + f + \beta + \delta + \phi - \beta\delta - \beta\phi - \delta\phi + \beta\delta\phi$

$\rightarrow 0 < c + d + \delta - \beta\delta - \delta\phi + \beta\delta\phi$

$\delta - \beta\delta - \delta\phi + \beta\delta\phi$ represents the number of terminals in δ not in β or ϕ. The lowest this value can be is 0.

$\rightarrow 0 < c + d$

This is always true, so len(E) is indeed less than k

Consider D is of length k - 1

Add a -1 to the inequality:

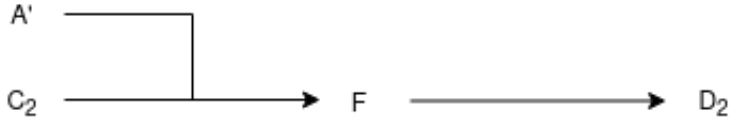
$b + e + f + \beta + \phi - \beta\phi < b + c + d + e + f + \beta + \delta + \phi - \beta\delta - \beta\phi - \delta\phi + \beta\delta\phi - 1$

$\rightarrow 0 < c + d - 1$

Which is again always true since c and d have to be different because they exist in the same clause, so len(E) is indeed less than k.

Since expanding a clause only directly implies clauses of length 1 greater, you only have to process clauses of length 1 shorter than D to get D

Consider C_2 and D_2



WTS A' and F are shorter than k

Consider D is of length k

$k = b + c + d + e + f + \beta + \delta + \phi - \beta\delta - \beta\phi - \delta\phi + \beta\delta\phi$

len(A') $= a + b + \beta + c$

len(A') $<$ k

$\rightarrow a + b + \beta + c < b + c + d + e + f + \beta + \delta + \phi - \beta\delta - \beta\phi - \delta\phi + \beta\delta\phi$

$\rightarrow a < d + e + f + \delta + \phi - \beta\delta - \beta\phi - \delta\phi + \beta\delta\phi$

In the smallest case for the R.H.S., d overlaps with e or f. $\delta + \phi - \beta\delta - \beta\phi - \delta\phi + \beta\delta\phi$ represents the number of terminals in $\delta + \phi$ - overlap. The smallest value for this is when $\delta = \phi$

$\rightarrow a < e + f + \delta$

Which is true as long as there are at least two terminals in e, f, δ or ϕ

Consider D is of length k - 1

Similarly as before, we will derive

$\rightarrow a < e + f + \delta - 1$

Which is true as long as there are at least three terminals in e, f, δ or ϕ

WTS F is shorter than k

len(F) $= a + b + e + f + \beta + \phi - \beta\phi$

len(F) $<$ k

$a + b + e + f + \beta + \phi - \beta\phi < b + c + d + e + f + \beta + \delta + \phi - \beta\delta - \beta\phi - \delta\phi + \beta\delta\phi$

$\rightarrow a < c + d + \delta - \beta\delta - \delta\phi + \beta\delta\phi$

Where $\delta - \beta\delta - \delta\phi + \beta\delta\phi$ represents the terminals in δ that are not in β or ϕ

Which is true if there are at least two terminals spread across c, d, or (δ and not β and not ϕ)

Which translates to at least two terminals in D_2 that are not in A or C_2

□

Lemma 3.19. *Using Lemma 3.12, a 1-t clause can only be directly implied by two 2-t clauses sharing the same terminal with the other terminal positive in one clause and negated in the other.*

Proof. Given a 1-t clause,

[i]

WTS that it can only be directly implied by 2-t clauses like

[i, j]

[i, -j]

By Lemma 3.13, the smallest clause that can be derived from two clauses of length k and l is max(k, l) - 1.

With k = l = 2, this gives us a clause of length 1.

If we increase by the lowest possible amount, say k = 3, l = 2, this gives us a clause of length 2.

So we have to use two clauses of length 2 if we want to directly imply a clause of length 1 by using the algorithm.

By the algorithm, both clauses must share the same terminal, where it's positive in one clause and negative in the other.

Now each clause has one remaining terminal and they could either be (1) the same terminal or (2) different terminals.

(1) The same term:
    By Lemma 3.12 or more directly Lemma 3.10, this will derive a 1-t clause
(2) Different term:
    By Lemma 3.12, this will derive a 2-t clause

Therefore, using Lemma 3.12, a 1-t clause can only be directly implied by two 2-t clauses sharing the same terminal with the other terminal positive in one clause and negated in the other.

□

Lemma 3.20. *The n-terminal clauses described in Lemma 3.14 can be reduced to derive any pair of contradicting 1-terminal clauses.*

Proof. TODO Prove □

Lemma 3.21. *Contradicting 1-terminal clauses can be expanded to imply every possible 3-terminal clause.*

Proof. Let the following clauses be a pair of contradicting 1-t clauses:
[a]
[-a]
Let the following be a generic 3-terminal clause we want to derive using the contradicting 1-t clauses:
[b, c, d]
By Lemma 3.9, we know the 1-terminal clauses imply the following 3-t clauses:
[a, b, c]
[-a, b, d]
By Lemma 3.12, these clauses imply:
[b, c, d] □

Lemma 3.22. *Any unsatifiable instance can be converted to a form such that if one 3-terminal clause is removed, the instance will be satisfiable.*

Proof. TODO Prove □

Lemma 3.23.

Proof. □

Lemma 3.24. *Maybe dumb but: Given an instance of 3SAT, we can add another terminal, x_i, then for each clause, we create two new 4-terminal clauses with either form of x_i appended to the current clause.*

Proof. Given:
[1, 2, 3], [1, -2, 3], [-4, -5, 6], ...
We can add the terminal, A, and create the 4-t clauses
[1, 2, 3, A], [1, 2, 3, -A],
[1, -2, 3, A], [1, -2, 3, -A],
[-4, -5, 6, A], [-4, -5, 6, -A],
...
Then what do we do? I guess we want to show [[A], [-A]] exists iff the instance is unsat idk

□

Lemma 3.25. *All of the clauses implied by Lemma 3.12 are all of the clauses that could possibly be implied*

Proof. "possibly be implied" - what does this mean?
WTS the given clauses of length 3 can be processed to gain all of the possible clauses of length 3 that are implied
again, what does "implied" mean? are you relying on the definition of "imply" in your algorithm?
WTS an unsatisfiable instance implies contradicting 1-t clauses.

Unsatisfiable -¿ all assignments are blocked
Contradicting 1-t clauses -¿ 2-t clauses
2-t clauses -¿ 2-t or 3-t clauses
3-t clauses -¿ 3-t clauses or 4-t clauses, but all my homies hate 4-t clauses
Proving this is another million dollar proof □

Lemma 3.26. *An instance is unsatifiable iff contradicting 1-terminal clauses can be derived using the algorithm from the Algorithm section.*

Proof. TODO Prove the million dollar proof □

4 ALGORITHM

(1) For each clause in the instance, C, of length 3 or less:
  (a) For each clause in the instance, D, of length 3 or less:
    (i) Get all clauses implied by C and D according to Lemma 3.12 and add them to the instance
  (b) For each clause in the instance, E, of length 1:
    (i) For each clause in the instance, F, of length 1:
      (A) if E and F contain the same terminal in which it is positive in one clause and negated in the other, the clauses are contradicting and the instance is unsatisfiable, end
(2) Repeat (1) until no new clauses are added
(3) If it reaches here, the instance is satisfiable, end

5 TIME COMPLEXITY ANALYSIS

In this section I will analyze the time complexity of the algorithm in section 4
(1) At most $\binom{n}{3} * 8 + \binom{n}{2} * 4 + \binom{n}{2} * 2$ clauses to iterate which is on the order of
$O(n^3)$
(1.a) At most $\binom{n}{3} * 8 + \binom{n}{2} * 4 + \binom{n}{2} * 2$ clauses to iterate which is on the order of
$O(n^3)$
(1.a.i) For each terminal in C, check if it's opposite form is in D
$O(3^2)$ which is just constant time which is on the order of
$O(1)$
(1.b) At most $\binom{n}{1} * 2$ clauses of length 1 exist, which is on the order of
$O(n)$
(1.b.i) At most $\binom{n}{1} * 2$ clauses of length 1 exist, which is on the order of
$O(n)$
(1.b.i.A) Constant time to check if two 1-terminal clauses contain the same terminal in the opposite form
$O(1)$
(2) Worst case, we add one new clause each time so we have to do (1) $\binom{n}{3} * 8 + \binom{n}{2} * 8 + \binom{n}{1} * 2$ which is on the order of
$O(n^3)$
(3) Constant time to check and return satisfiable
$O(1)$

The time complexity breaks down:
$(2) * ((1) * ((1.a) * ((1.a.i)) + (1.b) * ((1.b.i) * (1.b.i.A)))) + (3)$
$= O(n^3) * (O(n^3) * (O(n^3) * (O(1)) + O(n) * (O(n) * O(1)))) + O(1)$
$= O(n^3) * (O(n^3) * (O(n^3) * (O(1)) + O(n) * O(n))) + O(1)$
$= O(n^3) * (O(n^3) * (O(n^3) * (O(1)) + O(n^2))) + O(1)$
$= O(n^3) * (O(n^3) * (O(n^3) + O(n^2))) + O(1)$
$= O(n^3) * (O(n^3) * O(n^3)) + O(1)$
$= O(n^3) * O(n^6) + O(1)$
$= O(n^9) + O(1)$
$= O(n^9)$

6 PROOF OF CORRECTNESS

By Lemma 3.18/Lemma 3.17, if we can derive a clause of length k, say C, then any clause of length k - 1, say D, can be implied by the clauses implying C without the need to process a clause of length k or greater.

Now, WTS an unsatisfiable instance means we will imply contradicting 1-t clauses.

Given an unsatisfiable instance, we can expand the 3-t clauses to produce all possible n-terminal clauses.

These n-terminal clauses imply clauses of length n or n-1 using the algo.

They cannot imply any new n-terminal clauses since we already have all possible n-terminal clauses since it's unsatisfiable.

And by Lemma 3.18, the n-1-terminal clauses that are implied can be implied without having to process clauses longer than n - 1.

Now we have all the n-1-terminal clauses. These imply clauses between lengths n-2 and 2(n-1) - 2 by Lemma 3.13.

We already have the clauses longer than n-1 so we really only care about implying the clauses of length n-2.

Again by Lemma 3.18, these clauses can be implied without having to process clauses greater than length n-2.

Continue this pattern...

We already have the clauses of length n-w and we already derived any clause longer than n-w so the only clauses left are the clauses of length n-w-1. By Lemma 3.18/Lemma 3.17, we can derive the clauses of length n-w-1 without processing clauses longer than n-w-1.

Continue ...

We already have clauses of length 4 and we already derived any clause greater than length 4 so the only clauses left to imply are those of length 3. By Lemma 3.18/3.17, we can derive the clauses of length 3 without processing clauses greater than length 3.

Now we have all the clauses of length 3, and clearly a path exists to derive contradicting 1-terminal clauses.

And since we do not have to process clauses of length greater than 3, the time complexity analysis in section 5 holds and it is indeed polynomial time.

But wait! What if we get the retort: Consider "we already have all clauses of length 3, doesn't Lemma 3.17/18 mean any 2-terminal clause implied by these can already be implied without processing a clause

of length 3?" No, because the given 3-terminal clauses were not derived, so some processing does have to happen.

7 CONCLUSION

In this paper, we present an algorithm to solve 3SAT in polynomial time.

By the Cook-Levin theorem, 3SAT is NP-complete, meaning 3SAT is at least as hard as every problem in NP.

Since we have an algorithm to solve 3SAT in polynomial time, then all problems in NP can be solved in polynomial time.

Thus, the class of problems, P, that can be solved in polynomial time is equivalent to the class of problems, NP, that can be solved in non-deterministic polynomial time.

TODO:

- reorder definitions
- redo Lemma 3.18
- Perhaps modify the algo to use expansion or add another lemma to say 'expansion is equivalent to the current algo if we only deal with 1-, 2-, 3-, terminal clauses'