Solving the Boolean k-SAT Problem in Polynomial-Time

Caleb P. Nwokocha

July 9, 2025

1 Overview

In this article, I explore a structured reduction of arbitrary k-CNF formulas to a disjunction of 2-CNF subformulas—what I call a *3-DNF-of-2-CNF* representation—via three key transformations:

1. Clause-length reduction: Any clause of length > 3 is split into multiple 3-clauses by introducing fresh variables, preserving satisfiability.
2. Odd-Clause Adjustment (Theorem 3): If the total number of 3-clauses is odd, append a tautological 3-clause $(x \lor \neg x \lor \ell)$ to make it even without affecting logical equivalence.
3. Pairwise Expansion (Theorem 2): Consecutive pairs of 3-clauses are rewritten—via Theorem 1—into a disjunction of three 2-CNF formulas

$$\{p(i), q(i), r(i)\}.$$

   Conjoining these across all pairs yields a compact 3-DNF-of-2-CNF representation.

Once in this representation, one may either:

(a) Choose one 2-CNF block per pair by minimizing negative-literal counts and preferring rarely occurring variables, then solve the resulting 2-SAT conjunction in polynomial time.

(b) Systematically explore all $3^{m/2}$ possible choices of 2-CNF blocks to guarantee finding a satisfying assignment if one exists.

A central challenge lies in validating (a): constructing *counter-examples* that force backtracking. Since the space of possible clause interactions grows rapidly, identifying such counter-examples can be intricate. This overview thus conclude with emphasis on the importance of developing and analyzing hard instances—formulas for which local, greedy decisions lead to dead ends—in order to understand the true limits of this reduction algorithm.

2 Constructing the 3-DNF-of-2-CNF

Theorem 1. *The 3-SAT formula $(a \vee b \vee c) \wedge (d \vee e \vee f)$ is logically equivalent to $\big[(a \vee b) \wedge (d \vee e)\big] \vee \big[(a \vee c) \wedge (d \vee f)\big] \vee \big[(b \vee c) \wedge (e \vee f)\big]$.*

Proof. Start with the conjunction of two 3-clauses:

$$(a \vee b \vee c) \wedge (d \vee e \vee f). \tag{1}$$

1. Associative Law of $\vee$:

   $$(a \vee b \vee c) = \big((a \vee b) \vee c\big), \quad (d \vee e \vee f) = \big((d \vee e) \vee f\big). \tag{2}$$

   Hence

   $$(a \vee b \vee c) \wedge (d \vee e \vee f) = \big((a \vee b) \vee c\big) \wedge \big((d \vee e) \vee f\big). \tag{3}$$

2. Distributive Law of $\wedge$ over $\vee$: Let $X = (a \vee b)$, $Y = (d \vee e)$. Then observe that $\big((a \vee b) \vee c\big) \wedge \big((d \vee e) \vee f\big)$ can be written by changing the left disjunction to $X \vee c$ and the right disjunction to $Y \vee f$. By the distributive law,

   $$(X \vee c) \wedge (Y \vee f) = (X \wedge (Y \vee f)) \vee \big(c \wedge (Y \vee f)\big). \tag{4}$$

   Substituting back $X = (a \vee b)$ and $Y = (d \vee e)$ yields

   $$\begin{aligned} &\big((a \vee b) \vee c\big) \wedge \big((d \vee e) \vee f\big) \\ = &\Big((a \vee b) \wedge \big((d \vee e) \vee f\big)\Big) \vee \Big(c \wedge \big((d \vee e) \vee f\big)\Big). \end{aligned} \tag{5}$$

3. Distributive Law again (inner): Now distribute inside each of the two resulting conjunctions:

   $$\begin{aligned} (a \vee b) \wedge \big((d \vee e) \vee f\big) &= \big((a \vee b) \wedge (d \vee e)\big) \vee \big((a \vee b) \wedge f\big), \\ c \wedge \big((d \vee e) \vee f\big) &= \big(c \wedge (d \vee e)\big) \vee (c \wedge f). \end{aligned} \tag{6}$$

   Thus altogether:

   $$\begin{aligned} (a \vee b \vee c) \wedge (d \vee e \vee f) &= T_1 \vee T_2 \vee T_3 \vee T_4, \\ T_1 &= (a \vee b) \wedge (d \vee e), \\ T_2 &= (a \vee b) \wedge f, \\ T_3 &= c \wedge (d \vee e), \\ T_4 &= c \wedge f. \end{aligned} \tag{7}$$

4. Distributive Law (atomic expansion): Expand each of T_1, T_2, T_3 into atomic conjunctions by distributing $\wedge$ over $\vee$:

   $$\begin{aligned} &T_1 = (a \vee b) \wedge (d \vee e) = (a \wedge d) \vee (a \wedge e) \vee (b \wedge d) \vee (b \wedge e), \\ &T_2 = (a \vee b) \wedge f = (a \wedge f) \vee (b \wedge f), \\ &T_3 = c \wedge (d \vee e) = (c \wedge d) \vee (c \wedge e), \\ &T_4 = c \wedge f \quad \text{(already atomic)}. \end{aligned} \tag{8}$$

5. Re-grouping into three 2-clauses: The following disjunction of atomic conjunctions is obtained:

$$\begin{gathered}(a \wedge d) \ \vee \ (a \wedge e) \ \vee \ (b \wedge d) \ \vee \ (b \wedge e) \ \vee \\ (a \wedge f) \ \vee \ (b \wedge f) \ \vee \ (c \wedge d) \ \vee \ (c \wedge e) \ \vee \ (c \wedge f).\end{gathered} \tag{9}$$

To see that these nine atomic conjuncts can be *re-grouped* into exactly the three 2-clause conjunctions

$$(a \vee b) \wedge (d \vee e), \quad (a \vee c) \wedge (d \vee f), \quad (b \vee c) \wedge (e \vee f), \tag{10}$$

proceed by expanding each of those three candidates and checking that, when taken together, they produce precisely the same set of atomic conjunctions, counting each distinct conjunction only once (even if it appears multiple times):

1. Expand $(a \vee b) \wedge (d \vee e)$. Using distributivity,

$$(a \vee b) \wedge (d \vee e) = \ (a \wedge d) \ \vee \ (a \wedge e) \ \vee \ (b \wedge d) \ \vee \ (b \wedge e). \tag{11}$$

Hence from $(a \vee b) \wedge (d \vee e)$, obtain exactly the four atomic conjuncts

$$a \wedge d, \ a \wedge e, \ b \wedge d, \ b \wedge e. \tag{12}$$

2. Expand $(a \vee c) \wedge (d \vee f)$. Again by distributivity,

$$(a \vee c) \wedge (d \vee f) = \ (a \wedge d) \ \vee \ (a \wedge f) \ \vee \ (c \wedge d) \ \vee \ (c \wedge f). \tag{13}$$

Thus $(a \vee c) \wedge (d \vee f)$ yields the atomic conjuncts

$$a \wedge d, \ a \wedge f, \ c \wedge d, \ c \wedge f. \tag{14}$$

3. Expand $(b \vee c) \wedge (e \vee f)$. By the same rule,

$$(b \vee c) \wedge (e \vee f) = \ (b \wedge e) \ \vee \ (b \wedge f) \ \vee \ (c \wedge e) \ \vee \ (c \wedge f). \tag{15}$$

Hence $(b \vee c) \wedge (e \vee f)$ contributes the atomic conjuncts

$$b \wedge e, \ b \wedge f, \ c \wedge e, \ c \wedge f. \tag{16}$$

Now collect all atomic conjuncts appearing in the three expansions above:

$$\begin{aligned}\text{From } (a \vee b) \wedge (d \vee e): &\quad \{a \wedge d, \ a \wedge e, \ b \wedge d, \ b \wedge e\}, \\ \text{From } (a \vee c) \wedge (d \vee f): &\quad \{a \wedge d, \ a \wedge f, \ c \wedge d, \ c \wedge f\}, \\ \text{From } (b \vee c) \wedge (e \vee f): &\quad \{b \wedge e, \ b \wedge f, \ c \wedge e, \ c \wedge f\}.\end{aligned} \tag{17}$$

Notice that some atomic conjuncts (namely $a \wedge d$ and $b \wedge e$ and $c \wedge f$) appear more than once. However, in propositional logic a disjunction of identical atomic conjuncts collapses to a single copy. Therefore the *union* of all atomic conjuncts produced by the three 2-clause expansions is

$$\{\, a \wedge d,\ a \wedge e,\ b \wedge d,\ b \wedge e,\ a \wedge f,\ b \wedge f,\ c \wedge d,\ c \wedge e,\ c \wedge f \,\}, \tag{18}$$

which is exactly the same multiset of nine atomic conjuncts obtained from the original expansion of $(a \vee b \vee c) \wedge (d \vee e \vee f)$.

Hence, taking the disjunction of the three expanded 2-clause formulas yields nine-term disjunction

$$\begin{gathered} (a \wedge d) \ \vee \ (a \wedge e) \ \vee \ (b \wedge d) \ \vee \ (b \wedge e) \ \vee \\ (a \wedge f) \ \vee \ (b \wedge f) \ \vee \ (c \wedge d) \ \vee \ (c \wedge e) \ \vee \ (c \wedge f), \end{gathered} \tag{19}$$

no more and no fewer. Because disjunction is idempotent, any duplicate atomic conjunct appears only once. Therefore, the formula

$$\big[(a \vee b) \wedge (d \vee e)\big] \ \vee \ \big[(a \vee c) \wedge (d \vee f)\big] \ \vee \ \big[(b \vee c) \wedge (e \vee f)\big] \tag{20}$$

is equivalent to the nine-term disjunction by repeated application of distributivity. Formula (20) is a 3-DNF-of-2-CNF representation.

This completes the proof by systematic application of the Associative and Distributive Laws of propositional logic. □

Theorem 2 (Pairwise 3-SAT Expansion)**.** *Let*

$$F \ = \ \bigwedge_{j=1}^{2k} C_j \tag{21}$$

be a 3-CNF formula with an even number $2k > 2$ of clauses, where each

$$C_j = (\ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3}) \tag{22}$$

is a disjunction of three literals. Partition the clauses into k consecutive pairs

$$(C_{2i-1}, C_{2i}), \quad i = 1, \ldots, k. \tag{23}$$

Then

$$F \ = \bigwedge_{i=1}^{k} \Big[p(i) \ \vee \ q(i) \ \vee \ r(i) \Big], \tag{24}$$

and

$$p(i) = (\ell_{2i-1,1} \vee \ell_{2i-1,2}) \wedge (\ell_{2i,1} \vee \ell_{2i,2}), \tag{25}$$

$$q(i) = (\ell_{2i-1,1} \vee \ell_{2i-1,3}) \wedge (\ell_{2i,1} \vee \ell_{2i,3}), \tag{26}$$

$$r(i) = (\ell_{2i-1,2} \vee \ell_{2i-1,3}) \wedge (\ell_{2i,2} \vee \ell_{2i,3}). \tag{27}$$

Proof. Proceed by applying Theorem 1 to each consecutive pair of clauses in F. Fix $i \in \{1, \ldots, k\}$. Write

$$C_{2i-1} \;=\; (\ell_{2i-1,1} \vee \ell_{2i-1,2} \vee \ell_{2i-1,3}), \quad C_{2i} \;=\; (\ell_{2i,1} \vee \ell_{2i,2} \vee \ell_{2i,3}). \tag{28}$$

By Theorem 1, with the renaming

$$a \mapsto \ell_{2i-1,1},\; b \mapsto \ell_{2i-1,2},\; c \mapsto \ell_{2i-1,3},\; d \mapsto \ell_{2i,1},\; e \mapsto \ell_{2i,2},\; f \mapsto \ell_{2i,3}, \tag{29}$$

gives the logical equivalence

$$C_{2i-1} \,\wedge\, C_{2i} \;\equiv\; \big[(a \vee b) \wedge (d \vee e)\big] \;\vee\; \big[(a \vee c) \wedge (d \vee f)\big] \;\vee\; \big[(b \vee c) \wedge (e \vee f)\big]. \tag{30}$$

Substituting back the ℓ–notation yields exactly

$$C_{2i-1} \wedge C_{2i} \;\equiv\; p(i) \;\vee\; q(i) \;\vee\; r(i). \tag{31}$$

Since

$$F \;=\; \bigwedge_{i=1}^{k} \big(C_{2i-1} \wedge C_{2i}\big), \tag{32}$$

conjoining these k equivalences gives equation (24) as claimed. □

Theorem 3 (Odd-Clause Adjustment)**.** *Let*

$$F \;=\; \bigwedge_{j=1}^{2k+1} C_j \tag{33}$$

be a 3-CNF formula with an odd number $2k+1$ of clauses. Fix any fresh variable x (not appearing in F) and let

$$C_{2k+2} \;=\; (x \vee \neg x \vee \ell) \tag{34}$$

be an arbitrary clause (for some literal ℓ). Define $F' = F \wedge C_{2k+2}$. Then F' has $2(k+1)$ clauses and

$$F \;\equiv\; F', \tag{35}$$

and F' can be converted pairwise into multiple 3-DNF-of-2-CNF representations.

Proof. Since $(x \vee \neg x \vee \ell)$ is a tautology, $F' \equiv F$. Moreover, F' has exactly $2k+2$ clauses, which is even and > 2. Hence, by Theorem 2, the clauses in F' may be grouped into $(2k+2)/2$ pairs and replace each pair by the corresponding disjunction of three binary conjunctions, preserving logical equivalence throughout. □

3 Minimizing Negative Literals Per Clause

A clause with fewer negative literals admits more *positive* assignments (i.e. setting literals to `true`), which tends to increase the space of satisfying assignments. Concretely, if one 2-SAT candidate has clauses $(x \lor y)$ and $(\neg x \lor \neg z)$, its maximum negative count is 2. Another candidate with $(u \lor v)$ and $(\neg u \lor w)$ has maximum negative count 1. It is preferable to choose the latter candidate, because $(u \lor v) \ \land \ (\neg u \lor w)$ has at most one negation, making it "easier" to satisfy. Consider the three 2-SAT candidates arising from a pair (C_3, C_4):

$$\begin{aligned} p &= (a \lor \neg c) \ \land \ (\neg a \lor c), \\ q &= (a \lor d) \ \land \ (\neg a \lor \neg d), \\ r &= (\neg c \lor d) \ \land \ (c \lor \neg d). \end{aligned} \tag{36}$$

Counting negatives per clause:

	1st clause	2nd clause
p	1 $(\neg c)$	1 $(\neg a)$
q	0	2 $(\neg a, \neg d)$
r	1 $(\neg c)$	1 $(\neg d)$

(37)

Here, $\max\{\neg \text{ per clause}\}(p) = \max\{\neg\}(r) = 1$, while for q the maximum is 2. Thus both p and r are more preferable than q.

4 Tie-break by Preferring Rare Variables

If two 2-SAT formulas are tied on their negative-literal metric (e.g. both have exactly one negation per clause), then compare the *global* frequency of each variable in the entire collection of 2-SAT candidates across all pairs. It is preferable to choose the formula whose variables appear least often overall.

- *Motivation:* A rare variable provides *new* information when enforced, whereas a very frequent variable is likely already constrained by other clauses.

- *Example:* Suppose across all pairs the variable a, b, c appears 12 times but d, e, f only 4 times. Between two candidates $(a \lor b) \land (\neg a \lor c)$ versus $(d \lor e) \land (\neg d \lor f)$, both have exactly one negation, but it is preferable to choose the latter because d, e, f occur less frequently, thereby maximally reducing uncertainty about constraints by other clauses.

```
ALGORITHM: PAIRWISE REDUCTION OF 3-SAT TO 2-SAT
Require: A 3-CNF formula $F = \bigwedge_{j=1}^{m} C_j$, where each $C_j = (\ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3})$ and
    $m > 1$.
Ensure: A satisfying assignment for $F$, or UNSATISFIABLE.

 1: STEP 1: Normalize formula.
        1. Remove any duplicate of a clause so that the formula consist of only distinct
           clauses.
        2. Remove any duplicate of a variable in any $C_j$ so that each clause truly has
           three distinct variables.
 2: STEP 2: Make number of clauses even.
 3: if $m$ is odd then
 4:     Introduce a fresh variable $x$ and add the tautology $C_{m+1} = (x \vee \neg x \vee \ell)$ for
        arbitrary literal $\ell$. {By Theorem 3 (Odd-Clause Adjustment), $F \equiv F \wedge C_{m+1}$}
 5:     Set $m \leftarrow m + 1$.
 6: end if

 7: STEP 3: Pair and expand.
 8: for $i = 1, \ldots, \frac{m}{2}$ do
 9:     Let $(C_{2i-1}, C_{2i})$ be the $i$th consecutive pair.
10:     Apply Theorem 2 (Pairwise 3-SAT Expansion) to obtain three 2-SAT formulas

                            $p(i),\ q(i),\ r(i),$

        each of the form $(\alpha \vee \beta) \wedge (\gamma \vee \delta)$.
11: end for

12: STEP 4: Select one 2-SAT per pair.
13: for $i = 1, \ldots, \frac{m}{2}$ do
14:     From $\{p(i), q(i), r(i)\}$ choose the formula that
        1.   By section 3, minimizes the maximum number of negative literals in any
             single clause, and
        2.   By section 4, prefers clauses whose variables occur least frequently overall.
15:     Denote the chosen 2-SAT by $S_i$.
16: end for

17: STEP 5: Solve the resulting 2-SAT.
18: Let $S = \bigwedge_{i=1}^{m/2} S_i$.
19: Run any standard polynomial-time 2-SAT algorithm (e.g. strongly-connected com-
    ponents).
20: if $S$ is satisfiable then
21:     return the satisfying assignment (which also satisfies the original $F$).
22: else
23:     return UNSATISFIABLE.
24: end if=0
```

5 Time Complexity Analysis

I analyze the running time of the five-step procedure on an input 3-CNF formula F with n variables and m clauses.

1. Step 1 (Normalization): Removing duplicate clauses can be done by sorting or hashing in $O(m \log m)$ time. Eliminating repeated literals within each clause takes $O(m)$ time (each clause has constant size). Hence Step 1 runs in $O(m \log m)$.
2. Step 2 (Even-Clause Adjustment): Checking parity of m and, if odd, adding one tautological clause is $O(1)$. Thus $O(1)$.
3. Step 3 (Pair and Expand): It form $\frac{m}{2}$ pairs and, for each, apply Theorem 2 to produce three 2-SAT formulas. Each expansion involves only a constant number of literal-relabeling and conjunctions/disjunctions. Therefore Step 3 is $O(m)$.
4. Step 4 (Selection Procedure): Precompute the frequency of each variable across all $3m/2$ generated 2-SAT clauses in $O(m)$ time. For each of the $\frac{m}{2}$ pairs, compute the maximum negation count in constant time and compare frequencies in $O(1)$. Thus overall Step 4 is $O(m)$.
5. Step 5 (Solve 2-SAT): The conjunction S contains exactly $\frac{m}{2}$ clauses of size two. A standard implication-graph Strongly Connected Components algorithm runs in $O(n + m)$ time.

Total cost by summing contributions:

$$O(m \log m) \;+\; O(1) \;+\; O(m) \;+\; O(m) \;+\; O(n+m) \tag{38}$$

$$= \; O\big(m \log m + n + m\big) \;=\; O\big(m \log m + n\big). \tag{39}$$

In particular, for $m = \Omega(n)$ this is $O(m \log m)$, i.e. *polynomial-time* in the input size.

Remark. If one were to exhaustively enumerate all three choices per pair, the resulting time would blow up to $O\big(3^{m/2} \cdot (n+m)\big)$, but the greedy variant above remains $O(m \log m + n)$.

6 Solving k-SAT via 3-DNF-of-2-CNF

Let

$$F \;=\; \bigwedge_{j=1}^{m} C_j, \qquad C_j = (\ell_{j,1} \vee \ell_{j,2} \vee \cdots \vee \ell_{j,k}), \tag{40}$$

be an arbitrary k-CNF formula. I speculate the following high-level strategy to reduce F to a (possibly large) disjunction of 2-CNF blocks, each of which can be solved in polynomial time.

1. Clause-length reduction: If $k > 3$, introduce fresh variables $x_{j,4}, \dots, x_{j,k-1}$ to split each k-clause

$$(\ell_{j,1} \vee \ell_{j,2} \vee \cdots \vee \ell_{j,k}) \longrightarrow (\ell_{j,1} \vee \ell_{j,2} \vee x_{j,4}) \wedge (\neg x_{j,4} \vee \ell_{j,3} \vee \ell_{j,4}), \quad (41)$$

   and iterate until every clause is of length 3. This preserves satisfiability and increases the clause count linearly in k.

2. Even-clause adjustment: If the total number of 3-clauses is odd, append a tautological clause $(x \vee \neg x \vee y)$ for a fresh x and arbitrary y, as in Theorem 3.

3. Pairwise expansion: Group the resulting 3-clauses into $\frac{1}{2}$(even count) pairs and apply Theorem 2 to each pair, obtaining a disjunction of three 2-CNF formulas per pair.

4. Global 3-DNF assembly: Taking the conjunction over all pairs, one obtains

$$F \equiv \bigwedge_{i=1}^{\lfloor m/2 \rfloor} \big[p(i) \vee q(i) \vee r(i)\big], \quad (42)$$

   which, when fully distributed, is a 3-DNF (disjunction of conjunctions) of 2-CNF blocks.

5. Selection or enumeration: Select one 2-CNF per pair by minimizing negative-literal count and preferring rare variables (as in Step 4). Solve the conjunction of the chosen blocks in polynomial time.

Remark. The entire reduction (splitting into 3-clauses, odd-clause adjustment, and pairwise expansion) runs in time $O\big(m \cdot \text{poly}(n)\big)$, where n is the number of variables and m the original clause-count. However, if one wishes to exhaustively enumerate all choices of 2-CNF per pair, the worst-case time becomes $O\big(3^{m/2} \cdot \text{poly}(n)\big)$. The greedy variant (selecting one block per pair) stays in polynomial time but may require backtracking, in the presence of a counter-example that it always fails.

7 Proving the $P = NP$

An algorithmic result of finding *at least a unique satisfying assignment* in polynomial time for arbitrary 3-SAT does prove that $P = NP$. The key point here is that finding *at least a unique satisfying assignment* is sufficient to show that the formula is satisfiable. If this task can be completed in polynomial time, then 3-SAT is solvable in polynomial time, and therefore, $P = NP$. The focus on *at least a unique satisfying assignment* is not a trivial modification; it is, in fact, an essential part of solving the 3-SAT problem. In essence, finding *at least a unique satisfying assignment* is equivalent to solving the problem in the decision version of 3-SAT.

At the heart of NP-completeness lies the challenge of searching through a vast solution space. For NP-complete problems, such as 3-SAT, this search is thought to require exponential time in the worst case. However, if a polynomial-time algorithm can find *at least a unique satisfying assignment*, it implies that the search space for finding that solution is much smaller than previously thought.

The result challenges the assumption that NP-complete problems, in general, require exponential time for solution search. Finding *at least a unique satisfying assignment* is sufficient to show that a problem is solvable, and if this solution can be found efficiently, the entire class of NP problems becomes subject to polynomial-time algorithms.