Color Structures and the Monotone Satisfiability Problem with Bounded Variable Occurrence

Ronald de Haan
University of Amsterdam, ILLC

Hannah Van Santvliet
University of Tübingen, Department of Computer Science

Abstract

We study Monotone 3-Sat-$(\leq k, 1)$, a restricted variant of the Satisfiability problem where clauses consist of three variables and are monotone (every clause contains either only unnegated or only negated variables) with up to k positive and exactly one negative occurrence per variable in the formula. We resolve a challenge posed by Darmann and Döcker (On simplified NP-complete variants of Monotone 3-Sat, Discrete Applied Mathematics 292:45–58, 2021) by proving that for $k \in \{3, 4\}$, the problem is trivial in the sense that every instance satisfying the given restrictions is satisfiable. This result closes the remaining gap in a dichotomy theorem: Triviality for $k \in \{1, 2\}$ follows by a result by Tovey (A simplified NP-complete satisfiability problem, Discrete Applied Mathematics 8(1):85–89, 1984), while NP-completeness for $k \geq 5$ was shown by Darmann and Döcker. To obtain our result, we introduce the notion of *color structures* and show that a satisfying assignment can always be constructed in $\mathcal{O}(n \cdot m)$ time, where n and m denote the number of negative and positive clauses, respectively.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Satisfiability problem

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

The Boolean satisfiability problem (Sat) is a central decision problem in theoretical computer science. Given a Boolean formula in conjunctive normal form (CNF), the problem asks whether there exists an assignment that satisfies all clauses. Cook's seminal result established that Sat is NP-complete [2], and it has since served as a benchmark for studying computational hardness.

A common approach to gaining a finer understanding of the complexity of Sat is to study restricted subclasses of CNF formulas. By imposing syntactic or structural constraints, the resulting satisfiability problem may become tractable or even trivial.

A well-known example for such a jump is the restriction to clauses of size two: While 3-Sat is NP-complete, 2-Sat is solvable in polynomial time [1]. Further restrictions, such as bounding the number of variable occurrences, can render the problem trivial in the sense that every restricted formula is either always satisfiable or always unsatisfiable. Tovey showed that any CNF formula with clause size r in which each variable occurs at most r times is always satisfiable [9]. More recent work in this direction—aimed at identifying tractable or in particular trivial subclasses—has explored restrictions on the graph representation of formulas [7] and on the number of potential conflicts arising from truth value assignments [8].

Among the restricted variants of Sat, monotone formulas—formulas in which each clause contains either only positive or only negative literals—have received considerable attention. Since Monotone 3-Sat is NP-complete [5, 6], monotonicity alone does not suffice to render satisfiability easy. Consequently, additional restrictions are required to identify subclasses where satisfiability is guaranteed.

In the monotone setting, particular attention has also been paid to formulas where the number of occurrences of each variable is explicitly bounded. Darmann, Döcker, and Dorn [4] showed that Monotone 3-Sat remains hard if each variable appears exactly four times. Darmann and Döcker [3] distinguish positive and negative occurrences of the variables. They studied the class Monotone 3-Sat-(p, q), consisting of monotone 3-CNF formulas in which each variable occurs exactly p times positively and q times negatively, and obtained hardness results for every fixed pair (p, q) with $p \geq 2$ and $q \geq 2$ as well as for the case that each variable appears exactly three times positively and once negatively, or three times negatively and once positively.

They also showed that Monotone 3-Sat-$(k, 1)$ is NP-complete for $k \geq 5$, and posed the question of the complexity of Monotone 3-Sat-$(k, 1)$ with $k < 5$. By a result of Tovey [9, Theorem 2.4], the problem is trivial for $k \leq 2$, but the cases $k \in \{3, 4\}$ remained open. This question is particularly intriguing as it probes the precise threshold between hardness and guaranteed satisfiability within a structurally simple class of formulas.

In this paper, we resolve the open question of Darmann and Döcker by proving that all instances of Monotone 3-Sat-$(k, 1)$ with $k \leq 4$ are satisfiable, establishing that the satisfiability problem for this class is trivial. Our proof introduces a new combinatorial framework, called a *color structure*, which captures the interaction between positive and negative clauses under bounded variable occurrences. We present an algorithm that iteratively builds such a color structure from a given monotone formula with bounded occurrences running in $\mathcal{O}(n \cdot m)$ time, where n and m denote the number of negative and positive clauses, respectively. We prove termination of the algorithm and show that a satisfying assignment can be extracted directly from the resulting color structure.

2 Definitions and Notation

Let $x_1, x_2, \ldots$ be Boolean variables taking values in $\{\texttt{true}, \texttt{false}\}$. A *literal* is either a variable x_i or its negation $\overline{x_i}$. A *clause* is a disjunction of literals, and a formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. If every clause in a CNF formula contains exactly k literals, the formula is called a k-Sat formula. A *variable occurrence* is a specific appearance of a variable in a clause of the formula, possibly negated, and possibly appearing multiple times. A CNF formula is *monotone* if every clause consists entirely of positive literals or entirely of negative literals. Accordingly, the clauses of a monotone CNF-formula partition into *positive clauses*, containing only unnegated variables, and *negative clauses*, containing only negated variables.

We further bound the number of variable occurrences. The class Monotone 3-Sat-(p, q) consists of all monotone 3-Sat formulas in which each variable occurs exactly p times positively and exactly q times negatively. Analogously, Monotone 3-Sat-$(\leq p, \leq q)$ consists of all monotone 3-Sat formulas in which each variable occurs at most p times positively and at most q times negatively. Throughout this paper, we consider instances of Monotone 3-Sat-$(\leq k, 1)$ or Monotone 3-Sat-$(k, 1)$ for $k \in \{3, 4\}$.

To simplify such formulas, we encode the information carried by negative clauses by a *color* (denoted by a letter) and say that variables in the same negative clause *have the same color*. If two variables have the same color, we call them *relatives* and a variable is not relative to itself. Since the negated literals occur exactly once, we omit them and carry this information into the positive clauses via their respective colors. The simplified formula consisting of positive (and colored) clauses only is called a *colored* formula.

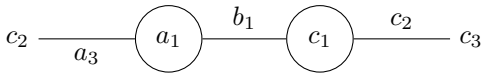


Figure 1 A color structure of the formula $(a_1 \vee a_3 \vee c_2) \wedge (a_1 \vee b_1 \vee c_1) \wedge (c_1 \vee c_2 \vee c_3)$

▶ **Example 1.** For instance, the formula

$$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_4} \vee \overline{x_5} \vee \overline{x_6}) \wedge (\overline{x_7} \vee \overline{x_8} \vee \overline{x_9}) \wedge (x_1 \vee x_3 \vee x_8) \wedge (x_1 \vee x_4 \vee x_7) \wedge (x_7 \vee x_8 \vee x_9)$$

has the colors a, b and c and hence contains the negative clauses $(\overline{a_1} \vee \overline{a_2} \vee \overline{a_3})$, $(\overline{b_1} \vee \overline{b_2} \vee \overline{b_3})$, $(\overline{c_1} \vee \overline{c_2} \vee \overline{c_3})$ and the positive clauses $(a_1 \vee a_3 \vee c_2), (a_1 \vee b_1 \vee c_1)$ and $(c_1 \vee c_2 \vee c_3)$ and the colored formula consists of the positive (and now colored) clauses. The variable a_1 has the two relatives a_2 and a_3 that both occur once.

Central to our structure is the notion of *guards*. A *guard* is a variable that is explicitly chosen to be assigned to the value `false`. Intuitively, the set of guards represents the choices we make in constructing the assignment. A negative clause is satisfied if and only if at least one of its variables is assigned the value `false`. We may therefore assume, without loss of generality, that any satisfying assignment sets exactly one variable per negative clause to false: if a satisfying assignment sets more than one variable in some negative clause to `false`, we can flip the excess false variables to `true`. Therefore, the negative clauses of a formula are satisfied if there is one guard for each color. To reason about a formula and a variable assignment simultaneously, we introduce a combined representation called a *color structure*. Specifically, we translate a monotone formula (or its colored variant) together with a variable assignment into such a structure, using guards to encode the relationship between the two.

▶ **Definition 2.** *A* color structure *of a colored 3-SAT formula is a triple $\langle G, N, E\rangle$ of sets of variables where the elements of G are called guards, those of N non-guards, and those of E edge labels, respectively. A color structure must satisfy the following properties:*

- *For every clause $c = (x_i \vee x_k \vee x_j)$, one variable x_k of the clause belongs to E, and the remaining two belong to $G \cup N$.*
- *No two guards have the same color.*
- *The sets G and $E \cup N$ do not share variables.*

Given a color structure $\langle G, N, E\rangle$, the induced assignment *sets all variables appearing in G to `false` and all remaining variables to `true`. For two variables that are both not guards, we may freely choose which one to place as an edge label and which as a non-guard node, since the disjunction $\vee$ of the formula is commutative.*

We represent a color structure as a graph whose nodes and edges are both labeled with variables from the formula. Nodes are either encircled *guard nodes* or unencircled *non-guard nodes*. Nodes and edges are labeled according to the clauses of the formula: if two nodes x_i and x_j are connected by an edge labeled x_k, then the formula contains the clause $(x_i \vee x_k \vee x_j)$. The graph representation is unique and throughout this paper, we use the term *color structure* to refer both to the graph representation and to the triple $\langle G, N, E\rangle$.

▶ **Example 3.** The color structure of the (colored) formula $(a_1 \vee a_3 \vee c_2) \wedge (a_1 \vee b_1 \vee c_1) \wedge (c_1 \vee c_2 \vee c_3)$ from above can be depicted as in Figure 1. Applying the induced assignment, we set a_1 and c_1, corresponding to the guard variables x_1 and x_7, to `false` and all remaining variables to `true`.

Sometimes variables are locked in between guards, and therefore cannot be added to G without violating the first property of the color structure. We say that a variable is *locked* if it appears in a positive clause whose two remaining variables are both guards. A color is *locked* if all variables of the color are locked. In particular, we cannot add a locked variable to G because the corresponding clause would not be satisfied then. In other cases, it is sufficient to swap variables between E and N so that the variable of each clause that belongs to E is an edge label in the graph representation. Further, we can make sure that G and $N \cup E$ are disjoint after we added a variable x to G if we add all variables from the clauses that contain x to the sets G, N and E. We abbreviate the process of swapping and adding variables from the same clause by the term *expansion*. *Expanding* a variable x means to add it to G, perform the required swaps between N and E, and add all variables from clauses containing x to G, N, E as appropriate. Note that even though expansion of a variable is global in the sense that either a variable is in the set G or not, due to property one, expansion has an influence on the variables that are in the same clauses as the expanded variable. Conversely, *unexpanding* a variable reverts the process of expansion: We remove the respective variable from the set G. Since unexpansion is the revert process of expansion, we refrain from expanding variables that have just been expanded and then unexpanded. As a convention, if the unexpansion produces an edge both of whose incident nodes are non-guards, this edge is omitted from the depicted color structure for a neater graph representation. Note that the process of expanding a variable modifies the current color structure in a unique way.

Further, we define a measure of how close a color is to being locked. We call the *lock number of a variable* the number of relatives that have at least one locked occurrence or that are contained in a clause where one variable is a guard. By construction and the variable occurrence bound of four, the lock number ranges between zero and three. In Example 3, the lock number of variable b_1 is zero, while the lock number of b_2 and b_3 is one, since b_1 is locked between a_1 and c_1. Intuitively, a high lock number indicates that the color of a variable is close to being locked. We therefore prioritize expanding variables with a high lock number. Similarly, we define the *expand number of a variable x_i of a color structure* $\langle G, N, E\rangle$ as the number of locked colors that are 'gained' if we expand x_i. For example, a color structure with the only locked color x and expand number of x_i zero, has no locked colors after expanding x_i.

3 A Color Structure Algorithm

Our first algorithmic approach for constructing a color structure is to greedily expand variables color by color. The lock number serves as a heuristic to avoid locking a color: if four variables of a color are locked, no satisfying assignment can be derived. If three variables of a color are locked, the only way to obtain a satisfying assignment is to expand the one remaining unlocked variable. The lock number ensures that this variable is prioritized for expansion. Nevertheless, this greedy approach may still produce a locked color, and thus an unsatisfying assignment. In that case, we need to modify the current color structure to eliminate locked colors. If this fails, no satisfying assignment can be derived from the color structure. In particular, a color structure is *satisfying* if each color has exactly one guard. The color structure depicted in Figure 2 (belonging to Example 5 below) is not satisfying, since no variable of color g is a guard. The following lemma formalizes the correspondence between satisfying assignments and satisfying color structures. We leave the proof to the reader.

▶ **Lemma 4.** *A monotone 3-SAT formula F in which each variable occurs exactly once*

negated is satisfiable if and only if it admits a satisfying color structure.

In sequel, we present the algorithm (Listing 1) that takes a colored variant of a monotone 3-SAT formula F where each variable occurs once negated and up to four times unnegated together as input and returns a color structure. The following example illustrates the individual steps of the algorithm. We resolve ties in line 4 of the algorithm due to the lowest index of the variables and ties in line 6 due to the highest index of the variables. Other tie breaks are possible but these make a connected color structure more likely.

Listing 1 Color Structure Construction Algorithm

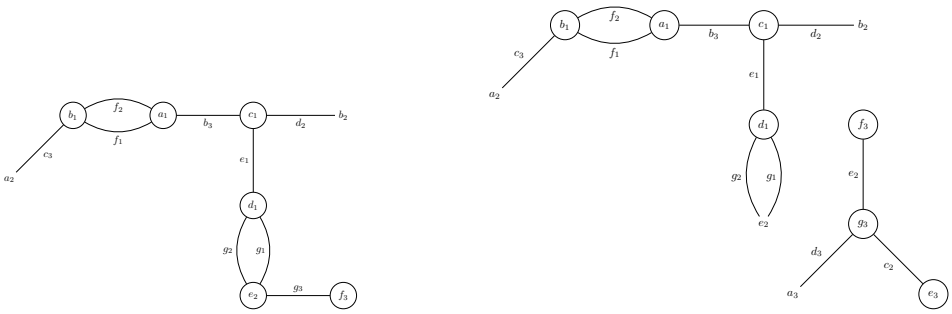
```
Pick a variable q_1 and set Q = {q_1}                                    1
WHILE Q is not empty DO                                                  2
    Expand element in Q with highest lock number                         3
    if the color does not have a guard yet                               4
    WHILE there exists a locked color a DO                               5
        Choose variable a_i with the lowest expand number and            6
        occurrence in the current color structure                        7
        FOR each occurrence of a_i DO                                    8
            Let c = (g ∨ a_i ∨ u) be the clause containing a_i           9
            Among g and u, unexpand the one with the lower lock number  10
            Expand a_i                                                  11
        END FOR                                                         12
    END WHILE                                                           13
    Add all variables in the color structure that are not locked to Q   14
    IF Q is empty DO                                                    15
        Pick new variable q_i that is not locked and whose              16
        color does not have a guard yet                                 17
        Q = {q_i}                                                       18
    END IF                                                              19
    Remove variables G and all their relatives from Q                   20
END WHILE                                                               21
RETURN Color structure constructed from the set of expanded variables   22
```

▶ **Example 5.** Consider the formula

$$(a_1 \vee b_1 \vee f_1) \wedge (a_1 \vee b_1 \vee f_2) \wedge (a_1 \vee b_3 \vee c_1) \wedge (a_2 \vee b_1 \vee c_3) \wedge (a_3 \vee d_3 \vee g_3) \wedge (b_2 \vee c_1 \vee d_2)$$
$$\wedge (b_2 \vee f_2 \vee g_1) \wedge (c_1 \vee d_1 \vee e_1) \wedge (d_1 \vee e_2 \vee g_1) \wedge (d_1 \vee e_2 \vee g_2) \wedge (c_2 \vee e_3 \vee g_3) \wedge (e_2 \vee f_3 \vee g_3).$$

First, we expand a_1 since the lock number of each variable is zero. Then, we have the following queue annotated with the current lock number in the second coordinate: $(b_1, 1), (b_3, 1), (c_1, 0), (f_1, 1)$ and $(f_2, 1)$. Subsequently, we expand b_1 since it has the highest lock number and the lowest index. In the course of the algorithm, we expand the variables a_1, b_1, c_1, d_1, e_2, f_3 in this order. This results in the color structure depicted in Figure 2a, locking the color g and hence, the variables g_1, g_2, and g_3. Since all variables g_1, g_2, and g_3 occur exactly once, we plan to expand g_3. The variable g_3 is locked between e_2 and f_3. Since the lock number of e_2 is one and the lock number of f_3 is two, we unexpand e_2 and then expand g_3. The outer while loop then terminates since we expanded the variable e_3 and with it the last color. This results in the color structure depicted in Figure 2b. Now, we can extract the satisfying assignment: We set the guard variables a_1, b_1, c_1, d_1, e_3, f_3, g_3 to `false` and all remaining ones to `true`. Since there are exactly as many negative clauses as guards, we have that Q is empty and the formula is satisfiable by Lemma 4.

The outer while loop greedily expands a variable for each color. Since the inner while loop may create new locked colors, we record its operations in a *state diagram*. An operation is



(a) Translating the colored formula from Example 5 to a color structure

(b) Modified color structure for the formula from Example 5

Figure 2 Example Color Structure Construction Algorithm

depicted as rectangle and connected with the operation that is subsequent in the inner while loop. The inner while loop contains the operations expand or unexpand. The minimal size consists of six operations since we have that unexpansions are always followed by expansions and by definition, we do not immediately unexpand a variable that was just expanded or the other way around. The number of states in the state diagram depends on the input formula. In order to facilitate a later counting argument, we section an arbitrary state diagram into the base cycle and the ℓ-block. The base cycle consists of the minimal size state diagram. We rename the variables such that the minimal size of a state diagram consists of the operations 'expand a_1', 'unexpand b_1', 'expand c_1', 'unexpand a_1', 'expand d_1', 'unexpand e_1'. We call the remaining part of the state diagram ℓ-block consisting of the remaining k expand operations and k unexpand operations. We rename the variables of the ℓ-block to $y_1^1, \ldots, y_1^\ell$ that are expanded and the variables $x_1^1, \ldots, x_1^\ell$ that are unexpanded. Hence, the state diagram is labeled with variables of the colors $a, b, c, d, e, y^1, \ldots, y^\ell, x^1, \ldots, x^\ell$. This naming convention helps us to overview the states of the inner while loop (see Figure 3).

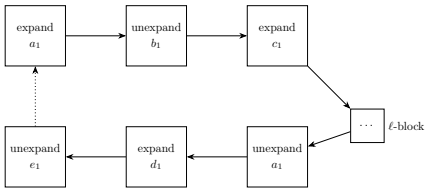


Figure 3 State diagram of the inner while loop

In order to prove correctness, we want to abstract from a concrete color structure given an input formula to the algorithm. Instead, we look at a hypothetical color structure that is constructed at a certain time in the algorithm. We specify this via the states of the state diagram. The state diagram fixes parts of the color structure due to the naming which variable is expanded or unexpanded when. We know that the variables that were expanded are guards and the variables that were unexpanded are edge labels connected to the guards. In particular, if 'expand a_1' and 'unexpand b_1' are consecutive, some clause must contain both a_1 and b_1. We call these clauses *connectors* because they connect the states in the state diagram. For each subsequent operation pair, we choose one specific connector if more than one clause contains both variables. In addition to connectors, there are clauses where we cannot specify its guards due to the naming of the state diagram but that contain variables

that need to be locked somewhere in the current color structure. Throughout the algorithm, we only unexpand a variable if the corresponding color is locked in the first place. A color is locked if all four variables of the color have at least one occurrence that is locked. If we unexpand b_1 in order to expand c_1, we can assume two relatives c_2 and c_3 to be locked in the first place. For each unexpansion followed by an expansion of a variable, we call the relative variables of the variable *detainee relatives*.

▶ **Lemma 6.** *There are $6 + 2\ell$ detainee relatives for the current color structure $\langle G, N, E\rangle$ in the last state of the state diagram consisting of a base cycle and an ℓ-block.*

Proof. In the base cycle, we have that the colors c, d and a were locked. Consequently, six variables—c_2, c_3, d_2, d_3, a_2, and a_3—must be locked in addition to the three variables a_1, c_1 and d_1 that were locked in the course of the algorithm. In each iteration of the ℓ-block, two additional variables y_2^i and y_3^i must be locked alongside y_1^i. In total, this yields $6 + 2\ell$ variables that need to be locked for a cycle consisting of a base cycle and an ℓ-block. ◀

So far, we have reasoned about the parts of the current color structure at a certain state of the state diagram with fixed naming. In contrast, we can look at the parts of the color structure where we have not yet named the variables or fixed the location of the clauses. To illustrate the hypothetical color structure, imagine a color structure that consists of placeholders for the guards, edge labels and node labels were some are filled with variables and some are blank. The blank spots arise because no variable names have been assigned/specified in the inner while loop yet. We call edges whose labels have not yet been specified in the inner while loop *prisoner spots*. A variable is locked if it is contained in a clause with two other guards and hence, prisoner spots are the locations where variables can be locked if the edge label is contained in a clause with two guards.

▶ **Lemma 7.** *If each variable occurs positively at most k times, the color structure $\langle G, N, E\rangle$ constructed after the first completion of the inner-while loop has at most $\lfloor \frac{3(k-1)+(k-1)\ell}{2} \rfloor$ prisoner spots.*

Proof. First, we count all possible edge labels in the color structure $\langle G, N, E\rangle$. We have three guards a_1, c_1, d_1, and ℓ additional ones for each iteration of the ℓ-block, all of which are necessarily contained in $\langle G, N, E\rangle$. Since each guard may occur at most k times in the formula, we obtain $k(3 + \ell)$ tuples of the form (guard, edge label). Here, we double count each edge label since two tuples (guard 1, edge label) and (edge label, guard 2) together correspond to a clause (guard 1, edge label, guard 2). Subsequently, there are at most $\lfloor \frac{k(3+\ell)}{2} \rfloor$ many edge labels in $\langle G, N, E\rangle$. There might be less edge labels since there could be clauses that contain only one guard from $\langle G, N, E\rangle$ and the remaining variables of the clause are not a guard or a guard that is not contained in $\langle G, N, E\rangle$. Note that possibly, there must be clauses with variables that are not part of the loop color structure if k is uneven.

Second, we subtract from the number of all possible edge labels the edge labels that were already labeled in the course of the algorithm. Specifically, for each expansion in the cycle that is followed by an unexpansion, there must exist a connector clause that enforces the expansion in the first place. In the hypothetical cycle, we would only unexpand b_1 and expand c_1 due to a clause $(a_1 \vee c_1 \vee b_1)$ which results in at least one edge labeled with c_1. Since a clause contains three variables, for each operation pair, we subtract half a clause from the total number of prisoner spots. We therefore subtract $\frac{3}{2}$ for the edges originating from the clauses $(a_1 \vee c_1 \vee b_1)$, $(y_1^i \vee d_1 \vee a_1)$ and (d_1, a_1, e_1), and subtract a further $\frac{\ell}{2}$ since each expansion in the ℓ-block must be connected to a subsequent expansion in the ℓ-block. In sum, there are at most $\lfloor \frac{k(3+\ell)-3-\ell}{2} \rfloor = \lfloor \frac{3(k-1)+(k-1)\ell}{2} \rfloor$ prisoner spots. ◀

In the following, we use the counting arguments to prove correctness of the Color Structure Construction Algorithm.

▶ **Theorem 8.** *Every monotone 3-Sat formula F in which each variable occurs once negated and at most four times unnegated is satisfiable. A satisfying assignment can be constructed in time $\mathcal{O}(n \cdot m)$, where n is the number of negative clauses and m is the number of positive clauses of F.*

Proof. Without loss of generality, we assume that each variable in a negative clause occurs in at least one positive clause. If this is not the case, we can immediately satisfy the negative clause by assigning `false` to any variable that does not appear in a positive clause and `true` to all remaining variables. In addition, we recall that it is only beneficial to set at most one variable per negative clause to `false`. As a result, we assume that no two variables of a positive clause occur in the same negative clause since then, the positive clause is automatically satisfied.

First, we show that the algorithm terminates. The outer while loop terminates once a variable has been marked as a guard for each color. In each iteration, one differently colored variable is added to G since in line 14 - 19 the algorithm adds variables to Q as long as there are colors that do not have a guard yet. The inner while loop may exchange variables in G, but does not decrease the number of guards. After all colors have been expanded, Q is empty since all variables are in G or are relatives of variables in G. For termination of the inner while loop, we look at the state diagram. The maximum length of the state diagram is reached when all clauses of the formula are in the color structure that is modified in the inner while loop. Afterwards, states have to be modified that were modified before. We depicted this with a dotted line in the state diagram (Figure 3) and show that we exit the while loop before entering a state that was entered before.

Let $\langle G, N, E\rangle$ be the current color structure in the last state of the state diagram restricted to the part of the color structure that contains variables $\mathcal{C} = \{a, b, c, d, e, x^1, y^1, x^2, y^2, \ldots, x^\ell, y^\ell\}$ or is locked by variables in $\mathcal{C}$. In the following, we count the number of edge labels where we can lock variables, and compare this with the minimum number of detainee relatives. Due to Lemma 6, we have that there are $6 + 2\ell$ detainee relatives. Furthermore, we have that the detainee relatives must be locked between the variables in $\mathcal{C}$ if we have a state diagram consisting of a base cycle and an ℓ-block: Suppose there exists a color $h \in \mathcal{C}$ with a locked variable where one guard is not contained in $\mathcal{C}$. Then, this variable has an expand number of zero and hence, is prioritized in the algorithm, and the inner while loop is exited after this iteration. But then, the color structure is not the one constructed in the last state of the state diagram.

Consequently, we want to count the number of edge labels where variables can be locked. Due to Lemma 7, we have at most $\lfloor 4.5 + 1.5\ell \rfloor$ prisoner spots for where variables occur positively at most four times and $6 + 2\ell$ prisoner spots for variables that occur positive at most five times. Thus, a cycle in the state diagram is possible when each variable occurs positively at most five times, but impossible when occurrences are bounded by four.

Further, we want to show the algorithm is well-defined. First, note that we can always expand a color (line 3-4) without violating property one of the definition of a color structure since variables of positive clauses have different colors. We have property two since in line 20, we remove variable with colors that already have a guard and property three is given due to the expansion procedure. Similarly, we have that the current color structure is a valid color structure before and after each iteration of the inner while loop because unexpanding all occurrence of a variable unlocks a variable and hence, allows the expansion of it yielding a valid color structure. We have that the inner while loop terminates in $\mathcal{O}(m)$ after–in the

worst case–all clauses that have been added as edges in the color structure were modified. The outer while loop terminates in $\mathcal{O}(n)$ once a variable has been marked as a guard for each color. By Lemma 4, this yields an $\mathcal{O}(n \cdot m)$ algorithm producing a satisfying assignment for any monotone 3-SAT formula with at most four positive occurrences and one negative occurrence per variable. ◀

As a result, we obtain an algorithm that iteratively builds a color structure from a given colored formula and thereby derives a satisfying assignment. Together with the equivalence proved by Darmann and Döcker [3, Theorem 1] and the aforementioned hardness result showing that Monotone 3-Sat-$(k, 1)$ is NP-complete for all $k \geq 5$ [3, Corollary 7], we obtain the following dichotomy:

▶ **Corollary 9.** *The problems Monotone 3-Sat-($\leq k$,1) and Monotone 3-Sat-(k,1) are NP-hard for $k \geq 5$ and trivial for $1 \leq k \leq 4$.*

4 Conclusion

In this work, we introduced the concept of a color structure for encoding assignments for formulas of the type Monotone 3-Sat-$(k, 1)$ with $k \geq 1$. We presented an algorithm that translates a formula of this type into a color structure and used it to establish the existence of a satisfying assignment. This completes the complexity classification of Monotone 3-Sat-($\leq k$,1) and Monotone 3-Sat-(k,1), and yields a polynomial-time algorithm for computing satisfying assignments in the tractable cases. We believe that the concept of color structures is interesting in itself and may prove fruitful for other Sat solving approaches.

References

1 Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters*, 8(3):121–123, 1979.

2 Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158. Association for Computing Machinery, 1971.

3 Andreas Darmann and Janosch Döcker. On simplified NP-complete variants of Monotone 3-Sat. *Discrete Applied Mathematics*, 292:45–58, 2021.

4 Andreas Darmann, Janosch Döcker, and Britta Dorn. The monotone satisfiability problem with bounded variable appearances. *International Journal of Foundations of Computer Science*, 29(06):979–993, 2018.

5 E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

6 W.N. Li. Two-segmented channel routing is strong NP-complete. *Discrete Applied Mathematics*, 78(1–3):291–298, 1997.

7 Alexander Pilz. Planar 3-SAT with a Clause/Variable Cycle. *Discrete Mathematics & Theoretical Computer Science*, 21, 2017.

8 Dominik Scheder. Trivial, Tractable, Hard. A Not So Sudden Complexity Jump in Neighborhood Restricted CNF Formulas. In *Algorithms and Computation*, pages 251–261, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

9 Craig A. Tovey. A Simplified NP-Complete Satisfiability Problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.