# Boolean Satisfiability—
# Combinatorics and Algorithms

Emo Welzl, ETH Zürich

Chapter 1 (October 22, 2004)
Chapter 2 (November 5, 2004)
Chapter 3 (November 12, 2004)
Chapter 4 (November 26, 2004)
Chapter 5 (December 17, 2004)
Chapter 6 (January 21, 2005)
Chapter 7 (January 28, 2005)
Chapters 8 & 9 (February 4, 2005)

*I can't get no satisfaction*
*I can't get no satisfaction*
*'Cause I try and I try and I try and I try*
*I can't get no, I can't get no*

[Jagger, Richards, 1965]

# Contents

## 0.1 Course Description

**Satisfiability of Boolean Formulas—Combinatorics and Algorithms**
(37-491 Erfüllbarkeit logischer Formeln—Kombinatorik und Algorithmen)
Emo Welzl (lecturer) and Robert Berke (assistant); ETH Zürich, Winter 2004/05, Friday, 10-12 (IFW B42, lectures) and 13-14 (IFW B42, exercises)
`http://www.ti.inf.ethz.ch/ew/courses/SAT04/index.html`

**Contents.** Satisfiability (SAT) is the problem of deciding whether a boolean formula in propositional logic has an assignment that evaluates to true. SAT occurs as a problem and is a tool in applications (e.g. Artificial Intelligence and circuit design) and it is considered a fundamental problem in theory, since many problems can be naturally reduced to it and it is the 'mother' of NP-complete problems. Therefore, it is widely investigated and has brought forward a rich body of methods and tools, both in theory and practice (including software packages tackling the problem). This course concentrates on the theoretical aspects of the problem. We will treat basic combinatorial properties (employing the probabilistic method including a variant of the Lovász Local Lemma), recall a proof of the Cook-Levin Theorem of the NP-completeness of SAT, discuss and analyze several deterministic and randomized algorithms and treat the threshold behavior of random formulas. In order to set the methods encountered into a broader context, we will deviate to the more general set-up of constraint satisfaction and to the problem of proper $k$-coloring of graphs.

**Prerequisites.** The course assumes basic knowledge in propositional logic (cf. [Schöning, 1992]), in probability theory and in discrete mathematics (cf. [Matoušek, Nesetril, 1998], [Steger, 2001, Schickinger, Steger, 2001]), as it is supplied in the *Vordiplomstudium.*

**Literature.** There exists no book that covers the many facets of the topic. Lecture notes covering the material of the course will be distributed.

**List of books** with material related to the course:

George Boole, *An Investigation of the Laws of Thought on which are Founded the Mathematical Theories of Logic and Probabilities*, Dover Publications (1854, reprinted 1973).

Peter Clote, Evangelos Kranakis, *Boolean Functions and Computation*

*Models*, Texts in Theoretical Computer Science, An EATCS Series, Springer Verlag, Berlin (2002).

Nadia Creignou, Sanjeev Khanna, Madhu Sudhan, *Complexity Classifications of Boolean Constrained Satisfaction Problems*, SIAM Monographs on Discrete Mathematics and Applications, SIAM (2001).

Harry R. Lewis, Christos H. Papadimitriou, *Elements of the Theory of Computation*, Prentice Hall (1998).

Rajeev Motwani, Prabhakar Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, (1995).

Uwe Schöning, *Logik für Informatiker*, BI-Wissenschaftsverlag (1992).

Uwe Schöning, *Algorithmik*, Spektrum Akademischer Verlag, Heidelberg, Berlin (2001).

Michael Sipser, *Introduction to the Theory of Computation*, PWS Publishing Company, Boston (1997).

Klaus Truemper, *Design of Logic-based Intelligent Systems*, Wiley-Interscience, John Wiley & Sons, Inc., Hoboken (2004).

## 0.2 Notation

$\mathbb{Z}, \mathbb{N}, \mathbb{N}_0$
$\{i..j\}$

$\mathbb{Z}$ denotes the set of integers, $\mathbb{N}$ the positive integers (natural numbers), and $\mathbb{N}_0$ the nonnegative integers. For $i, j \in \mathbb{Z}$, $\{i..j\} := \{n \in \mathbb{Z} \mid i \leq n \leq j\}$.

$|A|$, $2^A$
$\binom{A}{k}$, $A \oplus B$
$A \times B$
$B^A$

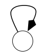
$A, B$ sets; $k \in \mathbb{N}_0$: $|A|$ denotes the cardinality of $A$, $2^A$ the power set of $A$ (set of all subsets of $A$), $\binom{A}{k}$ the set of all $k$-element subsets of $A$, $A \oplus B$ the symmetric difference, and $A \times B$ the Cartesian product. The set of functions $A \longrightarrow B$ is denoted by $B^A$.

$\Pr(\mathcal{E})$
$[P]$

$\Pr(\mathcal{E})$ denotes the probability of event $\mathcal{E}$. $\mathrm{E}(X)$ denotes the expected value of random variable $X$. "u.a.r." is short for "uniformly at random," and "i.i.d." for "identically and independently distributed." For $P$ a predicate, $[P]$ is 1 if $P$ holds, and 0, otherwise (*indicator function*).

$\deg_G(v)$

A (simple undirected) *graph* is a pair $G = (V, E)$, where $V$ is a finite set (the *vertices*) and $E \subseteq \binom{V}{2}$ (the *edges*). Vertex $u$ is termed *adjacent* to vertex $v$ (or $u$ is a *neighbor* of $v$), if $\{u, v\} \in E$. The *degree*, $\deg_G(v)$, of vertex $v$ in graph $G$ is the number of neighbors of $v$ in $G$.



A (simple) *directed graph* is a pair $G = (V, E)$, where $V$ is a finite set and $E \subseteq V \times V$ (the *directed edges*[1] or *arcs*). For $u, v \in V$, $u$ is an *out-neighbor* of $v$ if $(v, u) \in E$, and $u$ is an *in-neighbor* of $v$ if $(u, v) \in E$; $u$ is a *neighbor* of $v$ (or *adjacent* to $v$), if it is an out- or in-neighbor of $v$.

$\wedge$, $\vee$
$\neg$, $\overline{x}$
$\leftrightarrow$, $\rightarrow$, $\oplus$

The binary boolean operators 'and' (*conjunction*) and 'or' (*disjunction*) are denoted by $\wedge$ and $\vee$, resp. Negation is denoted by $\neg$ or by a horizontal bar over the operand (e.g. $\overline{x}$). We employ the usual shortcuts $\rightarrow$ (*implication*), $\leftrightarrow$ (*equivalence*), and $\oplus$ (*exclusive or*, '*xor*') ($x \rightarrow y$ for $\overline{x} \vee y$, $x \leftrightarrow y$ for $(x \rightarrow y) \wedge (y \rightarrow x)$ and $x \oplus y$ for $\neg(x \leftrightarrow y)$).

$f \equiv g$

Two boolean formulas $f$ and $g$ are *equivalent*[2], $f \equiv g$ in symbols, if every assignment of the variables involved in $f$ and $g$ lets the two formulas evaluate to the same value. A boolean formula $f$ is *satisfiable*, if there is an assignment under which $f$ evaluates to $\mathsf{true}$ (i.e. $f \not\equiv \mathsf{false}$). $f$ is a *tautology*, if it evaluates to $\mathsf{true}$ under all assignments (i.e. $f \equiv \mathsf{true}$).

---

[1] In a directed graph (as defined here), a vertex can have an edge to itself (a *loop*), while this is outruled for undirected graphs.

[2] Note the difference between '$\equiv$' and '$\leftrightarrow$': The latter is an operator symbol occurring in boolean formulas of propositional logic, while the former is an equivalence relation on the set of such formulas. They are connected in that $f \equiv g$ holds iff $f \leftrightarrow g$ is a tautology.

# Chapter 1

# Basics, Examples, and Set-Up

*Satisfiability (SAT)* is the problem of deciding whether a boolean[1] formula in propositional logic, take

$$(\overline{x} \to y) \wedge (y \to z) \wedge (z \to \overline{x}) \wedge (x \vee \overline{y} \vee \overline{z}) \tag{1.1}$$

or

$$(((x \wedge y) \vee (\overline{y} \wedge z)) \wedge \neg(x \leftrightarrow y)) \oplus (x \wedge \overline{y} \wedge z) \tag{1.2}$$

for examples, allows a $\{\mathsf{true}, \mathsf{false}\}$-assignment to variables so that the formula evaluates to $\mathsf{true}$—a so-called *satisfying assignment.* The formula (1.1) has such a satisfying assignment, namely

$$x \mapsto 1, \quad y \mapsto 0, \quad z \mapsto 0 \ ,$$

as can be easily checked (we let 1 represent $\mathsf{true}$ and 0 represent $\mathsf{false}$). (1.2) is not satisfiable and we see that it takes an argument more elaborate than in the case of satisfiability to certify this unsatisfiability.

Satisfiability has numerous applications, let it be in Artificial Intelligence or Formal Verification, but we will focus here on the theoretical aspects of the problem which plays a key role in Theoretical Computer Science, amongst others, as the 'mother' of NP-complete problems. Nevertheless, in the beginning of this first chapter we will briefly touch links to possible applications. Next we will justify our restriction to formulas

[1] George Boole, 1815-1864 (Queens College, Cork, Ireland)

in so-called *conjunctive normal form*, which will lead us to a tailored set-terminology for our treatise. In order to familiarize ourselves with this terminology, we will discuss first simple algorithms (for counting the number of satisfying assignments of a formula), and we will recapitulate the fundamental *resolution* method (for proving unsatisfiability of a formula).

**Exercise 1.1** *Show that the formula (1.2) is not satisfiable.*

**Exercise 1.2** *A binary boolean operator* $\circ$ *is* commutative *if*

$$x \circ y \equiv y \circ x \ .$$

*Which of the operators* $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$ *are commutative?*

**Exercise 1.3** *A binary boolean operator* $\circ$ *is* associative *if*

$$(x \circ y) \circ z \equiv x \circ (y \circ z) \ .$$

*Which of the operators* $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$ *are associative?*

*For the associative operators* $\circ$ *characterize when a boolean assignment of the variables lets*

$$\bigcirc_{i=1}^{n} x_i \ := \ x_1 \circ x_2 \circ \ldots \circ x_n$$

*evaluate to* $\mathsf{true}$.

**Exercise 1.4** *Is* $(\{\mathsf{true}, \mathsf{false}\}, \vee, \wedge)$ *a ring? A field?*

**Exercise 1.5** *Is* $(\{\mathsf{true}, \mathsf{false}\}, \oplus, \wedge)$ *a ring? A field?*

## 1.1 Examples

### 1.1.1 Circuit Verification

If we want to test equality of two binary signals (bits, "booleans") $x$ and $y$, that is, whether

$$x \leftrightarrow y \quad \equiv \quad (\overline{x} \vee y) \wedge (\overline{y} \vee x)$$

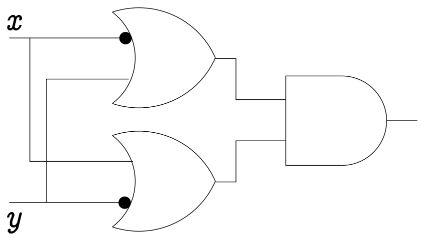



*Figure 1.1: A circuit testing equality of signals $x$ and $y$. ⊐ is the AND-gate, ⊃ is the OR-gate, and • negates. The output is 1 iff the inputs $x$ and $y$ are equal.*

is true, then the circuit in Figure 1.1 serves the purpose; its output is 1 iff $x$ equals $y$.

For testing equality of $x$, $y$ and $z$, we can build a circuit based on the boolean formula

$$(x \leftrightarrow y) \wedge (x \leftrightarrow z) \quad \equiv \quad (\overline{x} \vee y) \wedge (\overline{y} \vee x) \wedge (\overline{x} \vee z) \wedge (\overline{z} \vee x) \tag{1.3}$$

with 4 OR-gates and 3 AND-gates, but there is an alternative way by using the equivalent boolean formula

$$(x \vee \overline{y}) \wedge (y \vee \overline{z}) \wedge (z \vee \overline{x}) \; . \tag{1.4}$$

Is it really equivalent[2]?

Generally speaking, we have here the problem of deciding whether two boolean formulas $f_1$ and $f_2$ (here (1.3) and (1.4)) are equivalent. This amounts to deciding whether there is *no* satisfying assignment for

$$f_1 \oplus f_2$$

(recall that $\oplus$ is true iff its arguments are not equal).

[2]This can be seen by a case analysis. Suppose in an assignment of the variables $x$, $y$ and $z$, the formula (1.4) is true. If $x$ is false, then $y$ has to be false (because of $(x \vee \overline{y})$), and $y$ false forces $z$ to be false (because of $(y \vee \overline{z})$; thus, all three have to be false in this case. Similarly, if $x$ is true, then $z$ has to be true (because of $(z \vee \overline{x})$), and $z$ true implies $y$ true (because of $(y \vee \overline{z})$); therefore, all three have to be true. We have shown that formula (1.4) is true iff all three variables are assigned the same value. But see also Exercise 1.6.

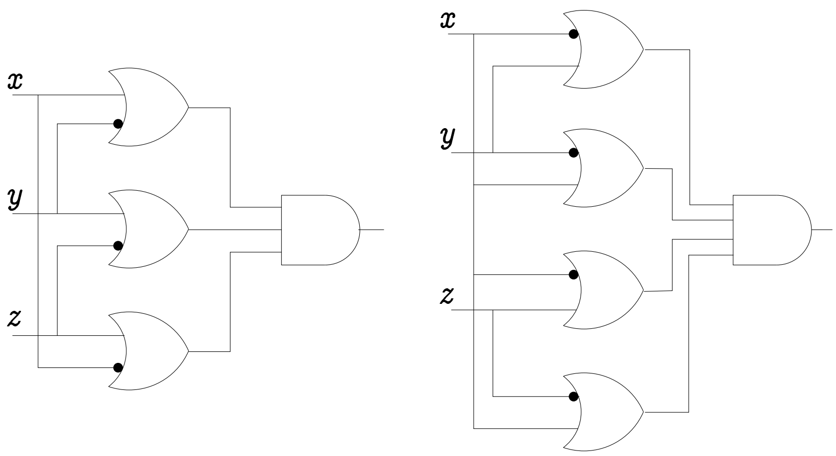



*Figure 1.2: Two equivalent circuits?*

In order not to let the example get too big, let us restrict ourselves to showing that the formula from (1.4), we call it $f$, implies that $x \leftrightarrow y$, that is, we have to show that the following formula is unsatisfiable (the proof of unsatisfiability is postponed to Section 1.5).

$$\begin{aligned} & \neg(f \rightarrow (x \leftrightarrow y)) \\ \equiv\ & \neg(\neg f \vee (x \leftrightarrow y)) \ \equiv\ f \wedge \neg(x \leftrightarrow y) \\ \equiv\ & f \wedge (\overline{x} \leftrightarrow y) \ \equiv\ f \wedge (\overline{x} \rightarrow y) \wedge (y \rightarrow \overline{x}) \\ \equiv\ & (x \vee \overline{y}) \wedge (y \vee \overline{z}) \wedge (z \vee \overline{x}) \wedge (x \vee y) \wedge (\overline{x} \vee \overline{y}) \end{aligned} \tag{1.5}$$

We have gone through the effort of transforming the formula in the form of (1.5) since this is the type of formulas we will treat in this course.

A boolean formula is in *conjunctive normal form* if it is the conjunction of clauses, where a *clause* is the disjunction of literals; a *literal* is a variable or the negation of a variable. We will also use *CNF formula*, for short. A CNF formula with $m$ clauses is of the form

$$\bigwedge_{i=1}^{m} \left( \bigvee_{j=1}^{k_i} u_{i,j} \right) , \quad u_{i,j}\text{'s literals.}$$

(1.4) and (1.5) are examples of CNF formulas. (1.1) can easily be written as an equivalent CNF formula

$$(x \vee y) \wedge (\overline{y} \vee z) \wedge (\overline{z} \vee \overline{x}) \wedge (x \vee \overline{y} \vee \overline{z}) \ .$$

What we have achieved in (1.5) is generally possible—every boolean formula can be transformed to an equivalent one in conjunctive normal form. There is a caveat though! The formula obtained may have size exponential in the size of the original formula. That would make the restriction to CNF formulas quite severe from a complexity point of view. However, it is possible to find for every formula $f$ efficiently a CNF formula $f'$ with size linear in that of $f$ so that $f'$ is satisfiable iff $f$ is satisfiable. So for the purpose of testing satisfiability of a formula $f$, this formula $f'$ perfectly serves the purpose. This will be discussed in more detail in Section 1.2 below.

**Exercise 1.6** Inequalities for Implications

*Observe that $x \to y$ is equivalent to the predicate $x \leq y$ for $x, y$ considered as real variables in $\{0, 1\}$. Rewrite the clauses in (1.4) with $\to$ and use this observation to immediately conclude that the formula is* true *iff all variables get the same value assigned.*

**Exercise 1.7** $n \in \mathbb{N}$. *Characterize the assignments that satisfy*

$$(\overline{x} \vee y_1) \wedge (\overline{y_1} \vee y_2) \wedge (\overline{y_2} \vee y_3) \wedge \ldots \wedge (\overline{y_{n-1}} \vee y_n) \wedge (\overline{y_n} \vee x) \ .$$

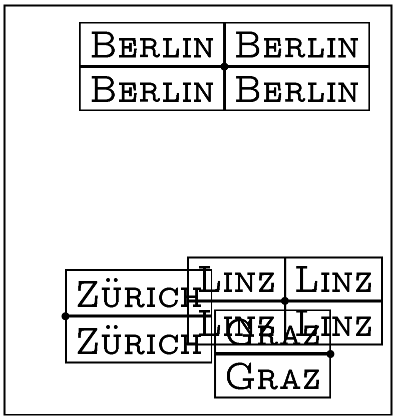
**Exercise 1.8** $n \in \mathbb{N}$. *Characterize the assignments that satisfy*

$$(\overline{x} \vee y_1) \wedge (\overline{y_1} \vee y_2) \wedge (\overline{y_2} \vee y_3) \wedge \ldots \wedge (\overline{y_{n-1}} \vee y_n) \wedge (\overline{y_n} \vee \overline{x}) \ .$$

### 1.1.2 Map Labeling

Points in a map have to be labeled. For each point, there are four possibilities to attach the label in the four quadrants around the point. Clearly, we want the labels not to overlap or to extend beyond the boundary of the map. We can express these conditions in a boolean formula as follows. For each point $p$ we introduce four variables $p_1$, $p_2$, $p_3$, $p_4$ representing the four possible placements ($p_i$ for the $i$th quadrant). The expression

$$(p_1 \vee p_2 \vee p_3 \vee p_4)$$



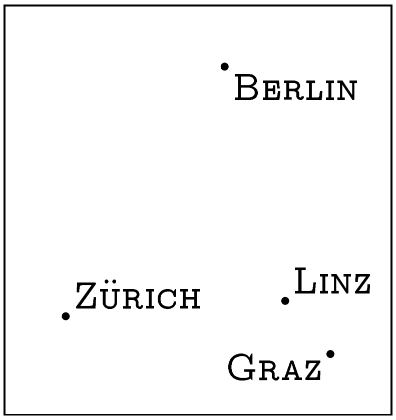




*Figure 1.3: The formula* $(b_1 \vee b_2 \vee b_3 \vee b_4) \wedge (g_2 \vee g_3) \wedge (l_1 \vee l_2 \vee l_3 \vee l_4) \wedge (z_1 \vee z_4) \wedge (\overline{z_1} \vee \overline{l_2}) \wedge (\overline{z_1} \vee \overline{l_3}) \wedge (\overline{z_4} \vee \overline{l_3}) \wedge (\overline{l_3} \vee \overline{g_2}) \wedge (\overline{l_4} \vee \overline{g_2})$ *has a satisfying assignment with* $b_4, g_3, l_1, z_1 \mapsto 1$ *and the remaining variables mapped to* $0$.

is satisfied only if at least one placement is set to 1 (here we can omit placements that extend beyond the boundary). If placements $p_i$ of point $p$ and placement $q_j$ of point $q$ overlap, we do not want both to be set to 1, which is ensured by the condition

$$\neg(p_i \wedge q_j) \quad \equiv \quad (\overline{p_i} \vee \overline{q_j}) \ .$$

The conjunction of all these constraints gives a formula which is satisfiable iff all conditions can be simultaneously satisfied. (Note that we didn't bother to include the condition that each point is labeled exactly once, but, of course, we can always omit extra options suggested by a satisfying assignment at our pleasing.)

We see here that the resulting formula comes in conjunctive normal form for free. Moreover, we observe that every clause involves at most 4 literals.

It is interesting to note that in circuit verification under normal (successful) circumstances we do not expect to find a satisfying assignment, while for map labeling, we do. We have seen that there is an asymmetry, in theory expressed by the fact that satisfiability is in NP, while unsatisfiability is in co-NP. If you are not familiar with these terms, this is best 'experienced' by the fact that it is easy to certify satisfiability (this does not mean that it is easy to find the proof), while in general this is hard for unsatisfiability.

## 1.2 Conjunctive Normal Form

Every boolean formula over a variable set $V$ has an equivalent formula in conjunctive normal form. To prove this we first observe that every boolean formula $f$ induces a function $\varphi_f : \{0,1\}^V \longrightarrow \{0,1\}$ by

$$\varphi_f(\alpha) := \text{ value to which } f \text{ evaluates under } \alpha \in \{0,1\}^V$$

and two formulas are equivalent, if they induce the same function. So we are done, if we find for every function $\varphi : \{0,1\}^V \longrightarrow \{0,1\}$ a CNF formula that induces $\varphi$. Such a formula is given by

$$\bigwedge_{\alpha \in \{0,1\}^V : \varphi(\alpha)=0} \underbrace{\left( \bigvee_{x \in V : \alpha(x)=0} x \ \vee \bigvee_{x \in V : \alpha(x)=1} \overline{x} \right)}_{=: C_\alpha} . \tag{1.6}$$

The clause $C_\alpha$ has been chosen so that it is $\mathsf{false}$ for $\alpha$ and, in fact, $\alpha$ is the only assignment on $V$ for which $C_\alpha$ is $\mathsf{false}$. It follows that the formula evaluates to $\mathsf{false}$ under an assignment $\alpha$ iff $\varphi(\alpha) = 0$. See Figure 1.4 for an example.

This existence statement is not really satisfactory from a complexity point of view, since it takes us exponential time to generate the CNF formula, and the resulting formula may have size exponential in that of the formula we started with. In the worst case this is inherently so: There are formulas of which all equivalent CNF formulas have exponential size (generalize Exercise 1.9). But there is a way out.

**Theorem 1.1** *For every boolean formula $f$ we can generate in time polynomial in the size of $f$ a CNF formula $f'$, so that $f'$ is satisfiable iff $f$ is satisfiable.*

| $x$ | $y$ | $z$ | $x \leftrightarrow (y \wedge z)$ | |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 1 | |
| 0 | 1 | 1 | 0 | $(x \vee \overline{y} \vee \overline{z})$ |
| 1 | 0 | 0 | 0 | $\wedge\ (\overline{x} \vee y \vee z)$ |
| 1 | 0 | 1 | 0 | $\wedge\ (\overline{x} \vee y \vee \overline{z})$ |
| 1 | 1 | 0 | 0 | $\wedge\ (\overline{x} \vee \overline{y} \vee z)$ |
| 1 | 1 | 1 | 1 | |

*Figure 1.4: A boolean formula, its induced boolean function, and an equivalent CNF formula.*

*If $f$ has $m$ occurrences of binary operators and $n$ variables then $f'$ has $m+n$ variables and $O(m)$ clauses, all of which are disjunctions of at most 3 literals.*

*Proof (sketch).* We confine ourselves to a sketch, among others, since we avoided to (re-)provide a formal definition of what a boolean formula is as we will not need it in our further proceeding.

What we need here is that every such formula (as every arithmetic expression) has an evaluation tree whose leaves are literals and whose inner nodes are labeled by operators. See Figure 1.5 for an example of such an evaluation tree for the formula (1.2). (We assume that a $\neg$-node does not have a $\neg$-node or a literal-node as a child. In the first case we could as well omit both nodes in the tree, in the latter we could omit the $\neg$-node and replace the literal by its negation.[3])

So let $f$ be a formula with variable set $V$ and $m$ binary operator occurrences. Consider an evaluation tree of $f$. To each node $i$ which is labeled by a binary operator we associate a new variable $a_i$. Each $\neg$-node gets associated the literal $\overline{a}$, where $a$ is the variable associated with its child,

[3] Actually, we can propagate the negations to the leaves by DeMorgan rules and relatives ($\neg(u \wedge v) \equiv (\overline{u} \vee \overline{v})$, $\neg(u \vee v) \equiv (\overline{u} \wedge \overline{v})$, $\neg(u \leftrightarrow v) \equiv (\overline{u} \leftrightarrow v)$, $\neg(u \oplus v) \equiv (\overline{u} \oplus v)$, and $\neg(u \rightarrow v) \equiv (u \wedge \overline{v})$), so that in the end the only negations that occur are in the literals.

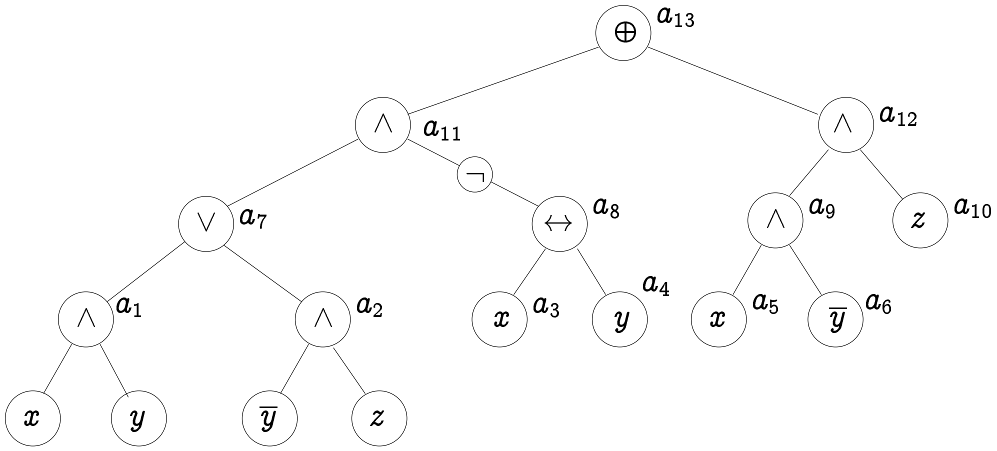



*Figure 1.5: The evaluation tree of the boolean formula (1.2) with associated new variables for inner nodes. 'Towards' $f'$ we consider*
$(a_{13}) \wedge (a_1 \leftrightarrow (x \wedge y)) \wedge \cdots (a_{11} \leftrightarrow (a_7 \wedge \overline{a_8})) \wedge \cdots (a_{13} \leftrightarrow (a_{11} \oplus a_{12}))$.

which has to be a binary operator node by our convention. Now all nodes in the tree have associated some literal, the leaves from variables in the original set $V$, and the inner nodes from some newly introduced variable; the number of such new variables is obviously exactly the number of nodes labeled by a binary operator.

Now we define a formula $f''$ (not the final $f'$ yet) as follows. If $i$ is a binary operator node labeled $\circ$ associated with $a_i$, its left child associated with literal $u$ and the right child with $v$, then we let $f_i$ be the formula $a_i \leftrightarrow (u \circ v)$. Given that, we set

$$f'' := (a_{\text{root}}) \wedge \bigwedge_{i \text{ inner node}} f_i \ .$$

Any assignment $\alpha$ on $V$ for $f$ can be extended to an assignment for the new variables by proceeding bottom up in the tree and observing the 'rules' $f_i = a_i \leftrightarrow (u \circ v)$, i.e. we set $a_i$ to the value of $u \circ v$. In this way we get an assignment for $f''$ that satisfies all $f_i$'s. Hence, the 'fate' of the assignment is decided by the value of $a_{\text{root}}$ which equals the value of $f$ under $\alpha$. That is, we have shown that if $f$ is satisfiable, then $f''$ is.

For the reverse direction, a satisfying assignment $\alpha''$ for $f''$ must observe all $f_i$'s (and it must obey $a_{\text{root}} \mapsto 1$), and thus the restriction of $\alpha''$ to $V$ is a satisfying assignment for $f$.

Therefore, $f$ is satisfiable iff $f''$ is satisfiable. $f''$ is a conjunction of $m+1$ formulas, one consisting of one variable, and the remaining $m$ involving three literals. Along the discussion before the theorem, we can replace each such formula $f_i$ by an equivalent conjunction of at most 8 disjunctions[4]. This finally gives the desired CNF formula $f'$, which involves $|V| + m$ variables and has at most $8m + 1 = O(m)$ clauses.

The generation of an evaluation tree for a given boolean formula can be done in polynomial time by standard parsing techniques. Further steps are trivially doable in polynomial time. ◻

In the next section we will redefine some of the terminology, tailored to our scenario of CNF formulas. Basically, we will remove some of the redundancies in the representation of a CNF formula. We write (represent) a clause as the set of its literals, e.g. $(x \vee \overline{y} \vee z)$ 'becomes' $\{x, \overline{y}, z\}$, and we write a conjunction of such clauses simply as the set of these clauses (now sets themselves). This is justified by the commutativity and associativity of $\vee$ and $\wedge$. For example, the formula in (1.5) 'becomes' $\{\{x, \overline{y}\}, \{y, \overline{z}\}, \{z, \overline{x}\}, \{x, y\}, \{\overline{x}, \overline{y}\}\}$. In this representation also redundancies as in $(x \vee y \vee x)$ disappear; in set notation this becomes $\{x, y\}$. We move even further in that direction by disallowing[5] a variable and its negation to appear in a clause: Such a clause is trivial in the sense that it will always evaluate to true under every assignment of the variables in the clause; hence, we may as well remove it from the formula and forget about it.

**Exercise 1.9** Parity as CNF

*Show that*

$$(x \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee z)$$

*is the unique CNF formula over $\{x, y, z\}$ equivalent to $x \oplus y \oplus z$. Here we mean 'unique' up to permutation of literals in clauses, permutation of clauses, and not considering clauses that repeat literals of the*

[4] In fact, four disjunctions suffice, see Exercise 1.12.

[5] Well, at one point we will bring it back into play for the sake of convenience.

*same variable—all of those things that are so nicely filtered out by our terminology in the forthcoming section.*

**Exercise 1.10** At Least One Couple

$n \in \mathbb{N}$. *For*

$$(x_{11} \wedge x_{12}) \vee (x_{21} \wedge x_{22}) \vee \ldots \vee (x_{n1} \wedge x_{n2})$$

*find an equivalent CNF formula. Try to get it as short as possible.*

**Exercise 1.11** Some Two

$n \in \mathbb{N}$. *For*

$$\bigvee_{1 \leq i < j \leq n} (x_i \wedge x_j)$$

*find an equivalent CNF formula. Try to get it as short as possible.*

**Exercise 1.12** Four Clauses Suffice

*Show that every boolean formula over* $\{x, y, z\}$ *has an equivalent CNF formula which is the conjunction of at most four clauses.*

**Exercise 1.13** Fewer Clauses

*Show that the formulas* $x \leftrightarrow (y \circ z)$, $\circ \in \{\wedge, \vee\}$, *have equivalent CNF formulas with three clauses, one of which is the disjunction of* 3 *literals, and the others of* 2. *Do there exist equivalent CNF formulas with two clauses?*

**Exercise 1.14** Few 3-Clauses

*Show: For every boolean formula* $f$ *with operators* $\wedge$, $\vee$ *and* $\neg$ *we can generate in time polynomial in the size of* $f$ *a CNF formula* $f'$, *so that* $f'$ *is satisfiable iff* $f$ *is satisfiable.*

*If* $m$ *is the number of occurrences of binary operators in* $f$ *and* $n$ *is the number of variables then* $f'$ *has* $m+n$ *variables and* $3m+1$ *clauses,* $m$ *of which are disjunctions of at most* 3 *literals, and the remaining ones of at most* 2 *literals.*

**Exercise 1.15** Disjunctive Normal Form
*A boolean formula is in* disjunctive normal form *(is a* DNF formula*), if it is the disjunction of conjunctions of literals, i.e. of the form* $\bigvee_{i=1}^{m}\left(\bigwedge_{j=1}^{k_i} u_{i,j}\right)$, $u_{i,j}$*'s literals.*

*Show that every boolean formula has an equivalent DNF formula.*

**Exercise 1.16** *What do you think is a harder problem? Deciding unsatisfiability of a CNF formula or deciding whether a DNF formula is a tautology?*
Or: Is it equally hard?

## 1.3 Terminology

**Variables and Literals.** A finite set $V$ of $n$ boolean variables gives rise to $2n$ *literals* $V \,\dot\cup\, \overline{V}$, where $\overline{V} := \{\overline{x} \mid x \in V\}$ is the set of *negations* of the variables in $V$. Elements in $V$ are called *positive literals*, elements in $\overline{V}$ *negative literals*. We extend negation to literals $u \in V \cup \overline{V}$ by

$$\overline{u} := \begin{cases} \overline{x} & \text{if } u = x \in V \text{, and} \\ x & \text{if } u = \overline{x} \in \overline{V}. \end{cases}$$

Literals $u$ and $v$ are *strictly distinct*, if $u \neq v$ and $u \neq \overline{v}$. We mostly use $x, y, z$ for variables and $u, v, w$ for literals (not exclusively, though).

**Clauses.** For $V$ a set of boolean variables, a *clause $C$ over $V$* is a subset of pairwise strictly distinct literals in $V \cup \overline{V}$; the *empty clause* is denoted by $\square$. $C$ is a *$k$-clause* if $|C| = k$ and it is a *$(\leq k)$-clause* if $|C| \leq k$. The set of variables that occur in $C$, i.e. $\{x \in V \mid x \in C \text{ or } \overline{x} \in C\}$, is denoted by $\mathrm{vbl}(C)$. Clauses $C$ and $C'$ are called *strictly disjoint* or *independent* if $\mathrm{vbl}(C) \cap \mathrm{vbl}(C') = \emptyset$, and they are termed *dependent*, otherwise. A set of pairwise strictly disjoint clauses is called an *independent set of clauses*.

**CNF Formulas.** For $V$ a set of boolean variables, a *CNF formula $F$ over $V$* is a set of clauses over $V$. $F$ is a *$k$-CNF formula* (a *$(\leq k)$-CNF formula*) if all clauses in $F$ are $k$-clauses ($(\leq k)$-clauses, respectively). We let $\mathrm{vbl}(F) := \bigcup_{C \in F} \mathrm{vbl}(C)$.

**Assignments.** For $V$ a set of boolean variables, an *assignment* $\alpha$ *on* $V$ is a mapping $\alpha : V \longrightarrow \{0,1\}$, i.e. $\alpha \in \{0,1\}^V$. Assuming some canonical ordering $\{x_1, x_2, \ldots, x_n\}$ on $V$ we will sometimes represent (or refer to) $\alpha$ simply as a vector in $\{0,1\}^n$ with the obvious interpretation. $\alpha$ extends to $V \cup \overline{V}$ by $\alpha(\overline{x}) := 1 - \alpha(x)$ for $x \in V$.

We will typically specify an assignment by specifying the images for literals in a non contradicting way. For example, by

$$\alpha := (x \mapsto 1, y \mapsto 0, z \mapsto 1)$$

or

$$x \mapsto 1, \overline{y} \mapsto 1, z \mapsto 1$$

which denotes the same assignment.

An assignment $\alpha$ on $V$ *satisfies a clause* $C$ over some $U$, if $\alpha(u) = 1$ for at least one literal $u \in (V \cup \overline{V}) \cap C$. $\alpha$ *satisfies* a CNF formula if it satisfies all of its clauses. A formula $F$ is *satisfiable* if there exists an assignment satisfying $F$.

(Caveat: Note that we do not require $\mathrm{vbl}(C) \subseteq V$ or $\mathrm{vbl}(F) \subseteq V$. For example, the assignments $(x \mapsto 1, z \mapsto 0)$ or $(x \mapsto 0, y \mapsto 0)$ both satisfy the clause $\{x, \overline{y}, z\}$. Note also that no assignment can satisfy the empty clause $\square$, and thus no formula that contains the empty clause is satisfiable. However, the empty formula $\{\}$ is satisfiable! [6])

An assignment $\alpha$ on $V$ is *total* for a clause $C$ (for a formula $F$) if $\mathrm{vbl}(C) \subseteq V$ ($\mathrm{vbl}(F) \subseteq V$, resp.).

For $C$ a clause and $\alpha$ an assignment on $V$, we let

$$C^{[\alpha]} := \begin{cases} \mathbf{1} & \text{if } \alpha \text{ satisfies } C \text{, and} \\ C \setminus (V \cup \overline{V}) & \text{otherwise.} \end{cases}$$

If $F$ is a CNF formula, $F^{[\alpha]} := \{C^{[\alpha]} \mid C \in F\} \setminus \{\mathbf{1}\}$. (For example,

$$\begin{aligned} \{x, \overline{y}, z\}^{[x \mapsto 1, z \mapsto 0]} &= \mathbf{1} \text{ and} \\ \{\{x, \overline{y}, z\}, \{\overline{x}, \overline{z}\}\}^{[x \mapsto 0, y \mapsto 1]} &= \{\{z\}\} \text{ .)} \end{aligned}$$

[6] Convention: We will use $\square$ for an empty clause, $\{\}$ for an empty formula, and $\emptyset$ for an empty set in general.

Given a CNF formula $F$ or a clause $C$ over $V$, the set of assignments on $V$ satisfying $F$ ($C$, resp.) is denoted by $\mathsf{sat}_V(F)$ ($\mathsf{sat}_V(C)$, resp.). Moreover, for the set of assignments not satisfying we use $\overline{\mathsf{sat}}_V(F) := \{0,1\}^V \setminus \mathsf{sat}_V(F)$ and $\overline{\mathsf{sat}}_V(C) := \{0,1\}^V \setminus \mathsf{sat}_V(C)$, resp. Two CNF formulas $F$ and $G$ are *equivalent*, in symbols $F \equiv G$, if $\mathsf{sat}_V(F) = \mathsf{sat}_V(G)$ for all $V \supseteq \mathrm{vbl}(F) \cup \mathrm{vbl}(G)$.[7]

**Observation 1.2** *For $V$ a finite set of boolean variables, let $F$ be a CNF formula over $V$, let $\alpha$ be an assignment on $V$ and let $x \in V$. Then*

(i) *$\alpha$ satisfies $F$ iff $F^{[\alpha]} = \{\}$,*

(ii) *$F$ is satisfiable iff $F^{[x \mapsto 0]}$ or $F^{[x \mapsto 1]}$ is satisfiable, and*

(iii) $|\mathsf{sat}_V(F)| = |\mathsf{sat}_{V \setminus \{x\}}(F^{[x \mapsto 0]})| + |\mathsf{sat}_{V \setminus \{x\}}(F^{[x \mapsto 1]})|$.

In the forthcoming two sections we will briefly familiarize ourselves with the CNF terminology as introduced here by discussing simple procedures for counting the number of satisfying assignments of a CNF formula, and by recapitulating the method of resolution, which you may remember from your introductory logic course.

**Exercise 1.17** Counting Clauses

*(1) Given a set $V$ of $n$ variables, how many clauses over $V$ are there? (2) For $k \in \mathbb{N}_0$, how many $k$-clauses over $V$ are there?*

HINTS: For (1), think of generating a clause by going through the list of $n$ variables, deciding for each one whether (i) it is omitted or (ii) included as a positive literal or (iii) included as a negative literal. For (2) first choose the subset of variables involved in the $k$-clause, and then decide for each variable picked whether it appears as positive or negative literal.

**Exercise 1.18** Counting Formulas

*(1) Given $n \in \mathbb{N}_0$, how many functions $\{0,1\}^n \longrightarrow \{0,1\}$ are there? (2) Given $k \in \mathbb{N}_0$ and a set $V$ of $n$ variables, how many $k$-CNF formulas over $V$ are there? (3) Show that there is a function $\{0,1\}^{10} \longrightarrow \{0,1\}$ that cannot be realized by a 3-CNF formula.*

HINT: Use (1) and (2) for (3).

[7] Here you might expect the definition of a CNF formula that is a tautology ... think about it.

**Exercise 1.19** *(1) Show that CNF formulas $F$ and $G$ are equivalent iff* $\mathrm{sat}_V(F) = \mathrm{sat}_V(G)$ *for* $V := \mathrm{vbl}(F) \cup \mathrm{vbl}(G)$. *(2) Give two equivalent CNF formulas $F$ and $G$ with* $\mathrm{vbl}(F) \neq \mathrm{vbl}(G)$.

## 1.4 Counting Satisfying Assignments

#SAT is the problem of computing the number of satisfying assignments of a given CNF formula. Clearly, it cannot be easier than deciding whether such a satisfying assignment exists. We will use this problem to get acquainted with programming terminology as employed in these notes.

Here is a first attempt of a procedure $\mathsf{cs}(F, V)$ that takes a CNF formula $F$ over $V$ and returns the number of assignments on $V$ satisfying $F$. Note that if $x \in V \setminus \mathrm{vbl}(F)$, then we cannot simply leave out $x$ in the second argument. Doing so will halve the returned value, since in this case every satisfying assignment on $V \setminus \{x\}$ can be extended in two ways to a satisfying assignment on $V$. We assume that the work to be done in $\mathsf{cs}()$ apart from

| | |
|---|---|
| $V$ boolean variables, $F$ CNF formula over $V$. | **function** $\mathsf{cs}(F, V)$<br>**if** $V = \emptyset$ **then**<br>    **if** $F = \{\}$ **then return** $1$;<br>    **else return** $0$; |
| POSTCONDITION: returns $\lvert\mathrm{sat}_V(F)\rvert$. | **else**<br>    $x \leftarrow_{\text{some in}} V$;<br>    **return** $\mathsf{cs}(F^{[x\mapsto 0]}, V \setminus \{x\}) \;+\; \mathsf{cs}(F^{[x\mapsto 1]}, V \setminus \{x\})$; |

the recursive calls (but including the computation of $F^{[x\mapsto 0]}$ and $F^{[x\mapsto 1]}$) is bounded by some monotone polynomial in $|F| + |V|$ (observe, that $|F|$ may be exponential in $|V|$). This is a reasonable assumption[8]. A more specific statement will have to refer to the representation of the pair $(F, V)$—details we like to sweep under the rug here. Given that, let us account 1 (step) for each call of the procedure $\mathsf{cs}()$. Whatever bound we prove for this number

[8] In fact, this can be done in time linear in the size of $F$ and $V$; see Exercise 1.27.

of steps, we can multiply it by a polynomial in order to get a bound on the overall running time of a call.

Let $t(n)$, $n \in \mathbb{N}_0$, be the maximum number of steps to be performed if $|V| \leq n$. Then

$$t(n) \leq \begin{cases} 1 & \text{if } n = 0, \text{ and} \\ 1 + 2t(n-1) & \text{otherwise.} \end{cases}$$

Hence, $t(n) \leq 2^{n+1} - 1$. We conclude that the running time of the procedure cs() for a formula of $m$ clauses over $n$ variables is bounded by

$$O(2^n \ \mathrm{poly}(m+n)) \ .$$

This is, of course, not too surprising. Simply going through all $2^n$ assignments and incrementing a counter for each one that is satisfying would have given the same bound. After all, that's basically what our recursive procedure does. Moreover, since the decision problem SAT is $NP$-complete, we should not expect anything better than exponential for the counting version of it.

Confronted with this fact, we can either try to find subclasses of CNF formulas, where the problem can be solved efficiently, or we can stay with the general problem and at least try to improve on the basis of the exponential function (or even the order of the exponent).

| | |
|---|---|
| $V$ boolean variables,<br>$F$ CNF formula over $V$.<br>PRECONDITION:<br>$F$ is $(\leq 1)$-CNF<br>POSTCONDITION:<br>returns $\lvert \mathrm{sat}_V(F) \rvert$. | **function** c1s$(F, V)$<br>**if** $(\Box \in F$ **or** $\exists x \in V : \{\{x\}, \{\overline{x}\}\} \subseteq F)$ **then**<br>    **return** 0;<br>**else**<br>    **return** $2^{\lvert V \rvert - \lvert F \rvert}$; |

A very easy instance of #SAT is a $(\leq 1)$-CNF formula $F$ over a variable set $V$, see procedure c1s() which computes the number of satisfying assignments in polynomial time—if you do it right, in linear time.

Things get a little bit more interesting for $(\leq 2)$-CNF formulas, see c2s(). This procedure builds on the following fact. If $\{u, v\}$ is a 2-clause

$V$ boolean variables, $F$ CNF formula over $V$.
PRECONDITION: $F$ is ($\leq 2$)-CNF
POSTCONDITION: returns $|\mathrm{sat}_V(F)|$.

**function** $\mathrm{c2s}(F, V)$
**if** $F$ is ($\leq 1$)-CNF **then**
  **return** $\mathrm{c1s}(F, V)$;
**else**
  $\{u, v\} \leftarrow_{\text{some 2-clause in}} F$;
  $\alpha \leftarrow (u \mapsto 1, v \mapsto 1)$;
  $\beta \leftarrow (u \mapsto 0, v \mapsto 1)$;
  $\gamma \leftarrow (u \mapsto 1, v \mapsto 0)$;
  $U \leftarrow V \setminus \mathrm{vbl}(\{u, v\})$;
  **return** $\mathrm{c2s}(F^{[\alpha]}, U) + \mathrm{c2s}(F^{[\beta]}, U) + \mathrm{c2s}(F^{[\gamma]}, U)$;

in a CNF formula $F$, then every satisfying assignment must assign 1 to at least one of the two literals $u$ and $v$. All three possibilities are recursively pursued in c2s(), and the results are accumulated for the number to be returned.

Let $t'(n)$, $n \in \mathbb{N}_0$, be the worst case bound for the number of calls to c2s() implied by a call with a ($\leq 2$)-CNF formula over $n$ variables (we do not count the call to c1s()). Then

$$t'(n) \leq \begin{cases} 1 & \text{if } n \in \{0, 1\}, \text{ and} \\ 1 + 3t'(n-2) & \text{otherwise.} \end{cases}$$

This gives $t'(n) \leq 1 + 3^1 + \ldots + 3^{\lfloor n/2 \rfloor} = \frac{1}{2}\left(3^{\lfloor n/2 \rfloor + 1} - 1\right)$. A bound of $O(3^{n/2}\,\mathrm{poly}(n+m)) = O(1.733^n\,\mathrm{poly}(m))$ for #SAT restricted to ($\leq 2$)-CNF formulas follows. This bound allows a simple and surprising improvement.

Let us go back to a 2-clause $\{u, v\}$. In order to satisfy this clause, we must either assign $u \mapsto 1$, or if not, then we have to assign $u \mapsto 0$ and $v \mapsto 1$. This is just the same as before, put into somewhat different terms, and in fib_c2s() put into a slightly revised procedure, whose analysis leads to a recursion

$$t''(n) \leq \begin{cases} 1 & \text{if } n \in \{0, 1\}, \text{ and} \\ 1 + t''(n-1) + t''(n-2) & \text{otherwise.} \end{cases}$$

| | |
|---|---|
| | **function** fib_c2s$(F, V)$ |
| $V$ boolean variables, $F$ CNF formula over $V$. | **if** $F$ is $(\leq 1)$-CNF **then** |
| | **return** c1s$(F, V)$; |
| PRECONDITION: $F$ is $(\leq 2)$-CNF | **else** |
| | $\{u, v\} \leftarrow_{\text{some } 2-\text{clause in}} F$; |
| POSTCONDITION: returns $\lvert \text{sat}_V(F) \rvert$. | $\alpha \leftarrow (u \mapsto 1)$; $\beta \leftarrow (u \mapsto 0, v \mapsto 1)$; |
| | $U \leftarrow V \setminus \text{vbl}(\{u\})$; $W \leftarrow V \setminus \text{vbl}(\{u, v\})$; |
| | **return** fib_c2s$(F^{[\alpha]}, U)$ + fib_c2s$(F^{[\beta]}, W)$; |

It gives us a bound of $t''(n) \leq 2f_{n+1} - 1$ for $f_m$, $m \in \mathbb{N}_0$, the $m$th Fibonacci number defined by $f_0 = 0$, $f_1 = 1$ and $f_{m+2} = f_{m+1} + f_m$ (see Appendix A.1). Thus we have an improved performance bound which we summarize as the result of this section.

**Theorem 1.3 ([W. Zhang, 1996])** *The number of satisfying assignments of a $(\leq 2)$-CNF formula with $m$ clauses over $n$ variables can be computed in time*

$$O(f_n \ \text{poly}(n + m)) = O\left(1.619^n\right)$$

*where $f_n$ is the $n$th Fibonacci number.*

NOTE 1.1 Counting satisfying assignments is #P-hard, even when restricted to 2-CNF formulas [Valiant, 1979]. [W. Zhang, 1996] supplies improvements on the obvious $O(2^n)$ bound; the material presented here is taken from that paper. Other contributions to the subject can be found in [Lozinskii, 1992, Dahllöf *et al.*, 2002, Williams, 2003], the latter ones with improvements on Theorem 1.3.

Polynomial time solvable subclasses are treated in [Courcelle *et al.*, 2001] and [Makowsky, Ravve, 2003].

**Exercise 1.20** *Apply the ideas from this section to counting satisfying assignments of $(\leq 3)$-CNF formulas.*

**Exercise 1.21** Representing Satisfying Assignments
*If instead of counting we want to* list *(output) the set of all satisfying*

*assignments, then we can't possibly do better than $O(2^n)$ in the worst case since this may be the size of the output, for an empty formula over $n$ variables, say. This becomes a more interesting problem, if we ask for a succinct representation of the output set. For this sake, let a* hyper-assignment *be a function $A : V \longrightarrow \{0, 1, *\}$. It represents all assignments $\alpha$ on $V$ with $A(x) \in \{\alpha(x), *\}$, i.e. $\alpha$ must agree with $A$ if $A$ maps to a non-$*$, but has either choice in $\{0, 1\}$, if $A$ maps to $*$.*

*For example, the set of satisfying assignments of a $(\leq 1)$-CNF formula can always be represented by a single hyper-assignment.*

*The goal is now to succinctly represent the set of all satisfying assignments of a $(\leq 2)$-CNF formula $F$— "succinctly" means that you represent $\mathrm{sat}_V(F)$ as the union of substantially less than $2^n$ hyper-assignments.*

**Exercise 1.22** Minimally Satisfying

*Given a CNF formula $F$, a (not necessarily total!) assignment $\alpha$ is called* minimally satisfying *$F$ if it satisfies $F$, and no proper restriction of $\alpha$ is satisfying $F$. What can you say about the number of minimally satisfying assignments of a 2-CNF formula?*

**Exercise 1.23** 2-CNF as DNF

*Show that every $(\leq 2)$-CNF formula over $n$ variables has an equivalent DNF formula with $O(1.619^n)$ conjunctions. (For definition of DNF see Exercise 1.15.)*

**Exercise 1.24** *Relate Exercises 1.21, 1.22 and 1.23 to each other, if possible.*

**Exercise 1.25** Number of Vertex Covers

*Given a graph $G = (V, E)$, a subset $N$ of its vertices is called a* vertex cover *if every edge contains at least one vertex in $N$, i.e. $e \cap N \neq \emptyset$ for all $e \in E$.*

*Design and analyze an algorithm for counting the number of vertex covers of a graph. Your goal should, of course, be to get it as efficient as you can.*

REMARK: I did not ask for "as efficient *as possible*"—I guess nobody knows, definitely I don't. Also, you should head for an algorithm for which you can provide an analysis. Something that you somehow believe to be efficient is not so much appreciated here.

**Exercise 1.26** *Don't hesitate to improve on Theorem 1.3!*

**Exercise 1.27** *Show that a CNF formula $F$ can be dynamically maintained so that the work to be done in the procedures of this section can be done in time $O(|V| + \sum_{C \in F} |C|)$.*

## 1.5 Resolution

Resolution is a correct and complete method for demonstrating that a CNF formula is unsatisfiable.

Let $C$ and $D$ be clauses so that there is a unique variable $x$ that occurs as literal $x$ in one of the two clauses and as literal $\overline{x}$ in the other one; say $x \in C$ and $\overline{x} \in D$. For $C' := C \setminus \{x\}$ and $D' := D \setminus \{\overline{x}\}$, we define $C' \cup D'$ as the *resolvent* of $C$ and $D$. For example,

$$\{x_1, \overline{x_2}, x_3\} \text{ is the resolvent of } \{x_0, x_1, \overline{x_2}\} \text{ and } \{\overline{x_0}, x_1, x_3\} .$$

**Lemma 1.4** *With the set-up as above*

$$\{C, D\} \equiv \{C, D, C' \cup D'\} .$$

*Proof .* We have to show that an assignment $\alpha$ satisfies $\{C, D\}$ iff it satisfies $\{C, D, C' \cup D'\}$.

Clearly, if $\alpha$ satisfies $\{C, D, C' \cup D'\}$, then it satisfies every subset of it, in particular $\{C, D\}$.

On the other hand, if $\alpha$ satisfies $\{C, D\}$, that is both $C = C' \cup \{x\}$ and $D = D' \cup \{\overline{x}\}$, then it has to satisfy $D'$ or $C'$, since $\alpha(x) = 1$ and $\alpha(\overline{x}) = 1$ exclude each other. Hence, $\alpha$ satisfies $C' \cup D'$ as well. ◻

We conclude that if $R$ is the resolvent of two clauses in a CNF-formula $F$, then $F \equiv F \cup \{R\}$. That is, if we keep successively adding resolvents to a formula, then the thus growing formula remains equivalent to the one we started with. If, at some point, the empty clause appears as a resolvent, it becomes evident that the formula is not satisfiable. We apply the procedure to a previously discussed example ((1.5) from Section 1.1).

$$\begin{array}{rcl}
F & = & \{\underbrace{\{x,\overline{y}\}}_{(1)},\underbrace{\{y,\overline{z}\}}_{(2)},\underbrace{\{z,\overline{x}\}}_{(3)},\underbrace{\{x,y\}}_{(4)},\underbrace{\{\overline{x},\overline{y}\}}_{(5)}\} \\
(1),(2) & : & \{x,\overline{z}\} \;\; (6) \\
(1),(3) & : & \{\overline{y},z\} \;\; (7) \\
(1),(4) & : & \{x\} \;\; (8) \\
(1),(5) & : & \{\overline{y}\} \;\; (9) \\
(2),(3) & : & \{\overline{x},y\} \;\; (10) \\
(2),(5) & : & \{\overline{x},\overline{z}\} \;\; (11) \\
(3),(4) & : & \{y,z\} \;\; (12) \\
(5),(10) & : & \{\overline{x}\} \;\; (13) \\
(8),(13) & : & \Box \;\; (14)
\end{array}$$

This proves that $F$ is not satisfiable. In retrospective, we have been inefficient in generating the proof, since resolvents (8), (10), (13) and (14) would have sufficed.

For a more formal set-up (for the sake of the next proof), given a CNF formula $F$, we call a sequence of distinct clauses

$$(C_0, C_1, \ldots, C_m)$$

a *resolution deduction of $C_m$ from $F$* if the following holds for all $i$, $0 \leq i \leq m$: Either $C_i \in F$, or there are indices $j, k$, $0 \leq j < k < i$, such that $C_i$ is the resolvent of $C_j$ and $C_k$.

**Theorem 1.5** *A CNF formula $F$ is unsatisfiable iff there is a resolution deduction of the empty clause $\Box$ from $F$.*

*Proof.* We have discussed the implication "$\Leftarrow$" above. For "$\Rightarrow$" let $F$ be an unsatisfiable CNF formula that, contrary to the claim of the theorem, does not allow a resolution deduction of $\Box$ from $F$; we choose such a formula which is minimal with respect to $|\mathrm{vbl}(F)|$.

$\mathrm{vbl}(F)$ cannot be empty, since then $F = \{\}$ or $F = \{\Box\}$. In the first case $F$ is satisfiable, in the second case $F$ contains already the empty clause and $(\Box)$ is a resolution deduction of $\Box$ from $F$. So we may assume that $\mathrm{vbl}(F)$ contains at least one variable $x$. Now consider the formulas $F_0 := F^{[x \mapsto 0]}$ and $F_1 := F^{[x \mapsto 1]}$.

Both $F_0$ and $F_1$ are unsatisfiable (see Observation 1.2), $|\mathrm{vbl}(F_0)| < |\mathrm{vbl}(F)|$, and $|\mathrm{vbl}(F_1)| < |\mathrm{vbl}(F)|$. Therefore, there is a resolution deduction of $\Box$ both from $F_0$ and from $F_1$ (by the minimality assumption on our formula $F$).

Consider such a resolution deduction $(C_0, C_1, \ldots, C_m)$ of $\Box = C_m$ from $F_0$. We define a resolution deduction $(\hat{C}_0, \hat{C}_1, \ldots, \hat{C}_m)$ from $F$ with

$$\hat{C}_\ell = C_\ell \quad \text{or} \quad \hat{C}_\ell = C_\ell \cup \{x\} \tag{1.7}$$

for all $\ell$, $0 \leq \ell \leq m$, as follows. For $i = 0, 1, \ldots, m$ proceed inductively:

> If $C_i \in F_0$, then there exists some clause $C \in F$ with $C_i = C^{[x \mapsto 0]}$. Let $\hat{C}_i := C$; note that indeed (1.7) holds for $\ell = i$.
>
> If $C_i \notin F_0$ and $C_i$ is resolvent of $C_j$ and $C_k$, $j, k < i$, then let $\hat{C}_i$ be the resolvent of $\hat{C}_j$ and $\hat{C}_k$. Note that if (1.7) holds for $\ell = j$ and for $\ell = k$, then the clauses $\hat{C}_j$ and $\hat{C}_k$ do indeed have a resolvent, and, in fact, (1.7) holds for $\ell = i$ as well.

Now we have a resolution deduction of $\hat{C}_m = \Box$ or $\hat{C}_m = \{x\}$ from $F$. In the first of the two cases we have a contradiction. In the latter case we have to perform the same procedure for $F_1$, yielding a resolution deduction of $\Box$ or $(\overline{x})$. Either we have a contradiction right away, or we merge the two deductions to one containing both $\{x\}$ and $\{\overline{x}\}$, which we can extend by $\Box$. So we have established a contradiction in either case, and the theorem is shown. ⊔⊓

So much for the good news: there is an easy automatic procedure that lets us conclude unsatisfiability, if we only wait sufficiently long. The only problem is that our patience may be stretched extensively. It may take an exponential number of steps until we obtain the empty clause. Even if somebody would be kind enough to tell us the shortest deduction of $\Box$, it may be exponentially long, [Haken, 1985, Urquhart, 1987]. That is, in general resolution cannot provide polynomial certificates for unsatisfiability. Just to make things even worse, [Chvátal, Szeméredi, 1988] showed that for any $k \geq 3$ a random formula of $cn$, $c \geq 0.7 \cdot 2^k$, $k$-clauses (u.a.r. from the set of all $k$-clauses over an $n$-element variable set) is unsatisfiable with high probability, but every resolution deduction of $\Box$ has length $(1+\varepsilon)^n$, where $\varepsilon > 0$ is a parameter depending on $c$ and $k$, (see also [Ben-Sasson, Widgerson, 2001]).

NOTE 1.2 Let me quote from [Chvátal, Szeméredi, 1988] for the history of the resolution method: "This observation [i.e. the resolution method] is often credited to [Robinson, 1965], even though it follows instantly from an analysis of a procedure designed by [Davis, Putnam, 1960]; Robinson defined the notion of resolution and proved a more general theorem concerning predicate calculus. Incidentally, the Davis-Putnam procedure was also proposed some fifty years earlier by [Löwenheim, 1908, Löwenheim, 1910, Löwenheim, 1913]. An account of Löwenheim's work can be found in [Hammer, Rudeanu, 1968]; we are grateful to Yves Crama for this information. We are also grateful to Wolfgang Bibel for informing us that resolution was introduced before by [Blake, 1937]; see [Bibel, 1987, p. 205]."

**Exercise 1.28** Horn Formulas

*A CNF formula in which every clause contains at most one positive literal is called a* Horn formula. Unit resolution *restricts to taking resolvents of two clauses, one (or both) of which contains exactly one literal. Show that unit resolution is a correct and complete method for proving unsatisfiability of Horn formulas.*

**Exercise 1.29** *(1) Show that if $C$ and $D$ are clauses with $C \subseteq D$ then $\{C\} \equiv \{C, D\}$. (2) Let $C$ be a clause and let $x$ be a variable with $x \notin \mathrm{vbl}(C)$. Then $C$ is the resolvent of $C \cup \{x\}$ and $C \cup \{\overline{x}\}$. Show that $\{C\} \equiv \{C \cup \{x\}, C \cup \{\overline{x}\}\}$.*
*(3) Employ (1) and (2) to simplify the CNF formula*

$$\{\{\overline{x_1}, x_2, x_3\}, \{\overline{x_1}, \overline{x_2}, x_3\}, \{\overline{x_1}, x_3, x_4, x_5\}, \{x_1, x_3\}\}$$

*i.e. transform it to an equivalent smaller formula.*

# Chapter 2

# Extremal Properties

There are simple quantitative criteria that allow to immediately conclude that a CNF formula is satisfiable. For example, when one tries to find a small example of an unsatisfiable 3-CNF formula, one soon realizes that it takes a few clauses to achieve this—at least 8 as we will see.

Another feature one quickly discovers in the construction of unsatisfiable CNF formulas is that the clauses have to be strongly interleaved (share variables); we will discover a clean quantitative assertion supporting this experience. This comes with the interesting phenomenon, that there is an easy checkable sufficient criterion for satisfiability, but up to now nobody knows how such an assured satisfying assignment can be found efficiently.

The third part treats the fact that in any CNF formula it is always possible to satisfy at least some fraction of the clauses—assuming that there is no empty clause present. And under mild extra conditions this fraction can be pushed to the reciprocal of the golden ratio.

## 2.1 Number of Clauses

**Theorem 2.1** $k \in \mathbb{N}$. *Every $k$-CNF formula with less than $2^k$ clauses is satisfiable.*

*Proof (probabilistic).* Consider a $k$-CNF formula $F$ with $m$ clauses over $V$. We show that if $m < 2^k$, then $F$ is satisfiable.

For $\alpha$ an assignment u.a.r. in $\{0,1\}^V$ and for $C$ a clause in $F$ we have

$$\Pr(\alpha \text{ does not satisfy } C) = 2^{-k} .$$

We employ the indicator variables[1]

$$X_C := [\alpha \text{ does not satisfy } C], \quad \text{for } C \in F.$$

Then

$$\mathrm{E}(\text{number of unsatisfied clauses}) = \mathrm{E}\left(\sum_{C \in F} X_C\right) = \sum_{C \in F} \mathrm{E}(X_C) = m\, 2^{-k} .$$

If $m < 2^k$, then the expected number of unsatisfied clauses is strictly less than 1, and thus there has to exist an assignment where no clause is unsatisfied—in other words, a satisfying assignment. ◻

*Proof (counting).* Consider a $k$-CNF formula $F$ with $m$ clauses over $V$, $n := |V|$. We show that if $m < 2^k$, then $F$ is satisfiable.

For a clause $C \in F$, let $\overline{\mathrm{sat}}(C)$ be the set of assignments on $V$ that do not satisfy $C$. Then $|\overline{\mathrm{sat}}(C))| = 2^{n-k}$ since the values of the $k$ variables occurring in $C$ are prescribed, while the values of the remaining $n-k$ variables are arbitrary. Hence,

$$|\bigcup_{C \in F} \overline{\mathrm{sat}}(C)| \le \sum_{C \in F} |\overline{\mathrm{sat}}(C)| = m\, 2^{n-k} .$$

If $m < 2^k$ there exists an assignment in $\{0,1\}^V \setminus \bigcup_{C \in F} \overline{\mathrm{sat}}(C)$ which satisfies all clauses and thus the formula $F$. ◻

I have put these two proofs side to side in order to emphasize that they are just two variants of essentially the same proof, the first one employing the notation of probability theory, which comes handy if you got used to it.

We can easily show that the result is tight by considering a $k$-CNF formula that consists of all $2^k$ $k$-clauses over some set of $k$ variables. In such a formula, each assignment satisfies all but one clause.

[1] Recall the indicator function $[P]$ for a predicate $P$ (cf. Section 0.2).

In order to supply an efficient procedure for finding a satisfying assignment as guaranteed in Theorem 2.1 we switch to a more general version of it.

**Theorem 2.2** *Let $F$ be a CNF formula with*

$$\sum_{C \in F} 2^{-|C|} < 1 \ .$$

*Then $F$ is satisfiable and a satisfying assignment can be computed in time polynomial in $|F|$ and $|V|$.*

*Proof of existence.* Let $V := \mathrm{vbl}(F)$. For $\alpha$ an assignment u.a.r. from $\{0,1\}^V$ and $C \in F$ we set $X_C := [\alpha \text{ does not satisfy } C]$. Then $\mathrm{E}(X_C) = 2^{-|C|}$. Let $Y_F$ be the random variable for the number clauses not satisfied by $\alpha$. We have

$$\mathrm{E}(Y_F) = \mathrm{E}\left(\sum_{C \in F} X_C\right) = \sum_{C \in F} \mathrm{E}(X_C) = \sum_{C \in F} 2^{-|C|} \ .$$

Since this is strictly less than 1, there has to exist an assignment that satisfies all clauses. ◻

*Proof of efficient computation, derandomization via conditional probabilities.* Choose some $x_1 \in V$.

$$\begin{aligned} \mathrm{E}(Y_F) &= \mathrm{E}(Y_F \,|\, x_1 \mapsto 0) \cdot \Pr(x_1 \mapsto 0) + \mathrm{E}(Y_F \,|\, x_1 \mapsto 1) \cdot \Pr(x_1 \mapsto 1) \\ &= \frac{1}{2}\left( \mathrm{E}(Y_F \,|\, x_1 \mapsto 0) + \mathrm{E}(Y_F \,|\, x_1 \mapsto 1)\right) \end{aligned}$$

Hence there is a choice of $t_1 \in \{0,1\}$ such that

$$\mathrm{E}(Y_F \,|\, x_1 \mapsto t_1) \leq \mathrm{E}(Y_F) \ .$$

Since[2]

$$\mathrm{E}(Y_F \,|\, x_1 \mapsto 0) = \sum_{C \in F} \mathrm{E}(X_{C^{[x_1 \mapsto 0]}})$$

[2] $C^{[x_1 \mapsto 0]}$ may be $\mathbf{1}$. We set $X_{\mathbf{1}} = 0$ under all assignments.

can be computed in polynomial time, we can decide for $t_1$ in polynomial time. Now we progress in this manner for all variables in some arbitrary order. In general, we have already chosen some partial assignment

$$\alpha_i = (x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_i \mapsto t_i) \quad , \quad i < n,$$

with $\mathrm{E}(Y_F \,|\, \alpha_i) \leq \mathrm{E}(Y_F)$. And since

$$\mathrm{E}(Y_F \,|\, \alpha_i) = \frac{1}{2} \left( \mathrm{E}(Y_F \,|\, \alpha_i, x_{i+1} \mapsto 0) + \mathrm{E}(Y_F \,|\, \alpha_i, x_{i+1} \mapsto 1) \right) \; ,$$

we can efficiently find a $t_{i+1} \in \{0, 1\}$ that carries us to $i + 1$, i.e. allows us to extend $\alpha_i$ by $x_{i+1} \mapsto t_{i+1}$ to obtain $\alpha_{i+1}$ with $\mathrm{E}(Y_F \,|\, \alpha_{i+1}) \leq \mathrm{E}(Y_F)$.

In the end we have obtained a *total* assignment $\alpha_n$ for $F$ with $\mathrm{E}(Y_F \,|\, \alpha_n) \leq \mathrm{E}(Y_F) < 1$. Since $\mathrm{E}(Y_F \,|\, \alpha_n) = |\{C \in F \mid C^{[\alpha_n]} = \Box\}|$ is a nonnegative integer, it has to be 0 and the assignment $\alpha_n$ is satisfying. ◻

*Proof of efficient computation, scoring function.* Given an assignment $\alpha$ on $V' \subseteq \mathrm{vbl}(F)$, we define the score of a clause $C$ by

$$\mathrm{sc}(C, \alpha) := \begin{cases} 0 & \text{if } C^{[\alpha]} = \mathbf{1}, \text{ and} \\ 2^{-|C^{[\alpha]}|} & \text{otherwise.} \end{cases}$$

Note that for $x \notin V'$

$$\mathrm{sc}(C, \alpha) = \frac{\mathrm{sc}(C, (\alpha, x \mapsto 0)) + \mathrm{sc}(C, (\alpha, x \mapsto 1))}{2} \; .$$

If we extend now the mapping sc to CNF formulas

$$\mathrm{sc}(F, \alpha) := \sum_{C \in F} \mathrm{sc}(C, \alpha)$$

we observe that

- If $\alpha$ is the empty assignment, then $\mathrm{sc}(F, \alpha) = \sum_{C \in F} 2^{-|C|}$.

- If $\alpha$ is total for $F$ then $\mathrm{sc}(F)$ is the number of clauses in $F$ not satisfied by $\alpha$.

- For $x \notin V'$,

$$\mathrm{sc}(F, \alpha) = \frac{\mathrm{sc}(F, (\alpha, x \mapsto 0)) + \mathrm{sc}(F, (\alpha, x \mapsto 1))}{2} .$$

- For $x \notin V'$, a value $t \in \{0, 1\}$ with $\mathrm{sc}(F, (\alpha, x \mapsto t)) \leq \mathrm{sc}(F, \alpha)$ can be computed in polynomial time.

It follows that we can successively assign values to the variables in $\mathrm{vbl}(F)$ without ever increasing the score. Since we start with a score less than 1, in the end, when the score is a nonnegative integer, this score has to be 0 and the obtained assignment is satisfying. ◻

Clearly, the two proofs we saw are basically the same. While you may happen to prefer the second, you should still appreciate the first one as well as it reveals a general principle which can be applied in many instances, and it explains where the score function in the second proof "comes from" (which we would have had to accept out of the blue, otherwise).

NOTE 2.1 The method of derandomization via conditional probabilities can be found, at least implicitly, in [Erdős, Selfridge, 1973], and it was developed as a method for obtaining polynomial time deterministic algorithms by [Spencer, 1987] and [Rhagavan, 1988].

**Exercise 2.1** Almost Satisfiable

*Show that the following properties are equivalent for an unsatisfiable CNF formula $F$:*

(1) $\sum_{C \in F} 2^{-|C|} = 1$.

(2) *Every total assignment for $F$ satisfies all but one clause in $F$.*

(3) *For any pair $C$ and $D$ of clauses in $F$, their union $C \cup D$ contains a complementary pair $x, \overline{x}$ of literals.*

**Exercise 2.2** Branching Rules

*Let $G$ be an unsatisfiable CNF formula over variable set $V$ with $\sum_{C \in G} 2^{-|C|} = 1$. Show that for every CNF formula $F$ over $V$ we have*

$$|\mathrm{sat}_V(F)| = \sum_{C \in G} |\mathrm{sat}_{V \setminus \mathrm{vbl}(C)}(F^{[C \mapsto 0]})| = \sum_{C \in G \setminus F} |\mathrm{sat}_{V \setminus \mathrm{vbl}(C)}(F^{[C \mapsto 0]})| ,$$

*where* $(C \mapsto 0)$ *is short for the unique assignment on* $\mathrm{vbl}(C)$ *that does not satisfy* $C$.

REMARK: Consider $\{\{x,y\},\{x,\overline{y}\},\{\overline{x},y\},\{\overline{x},\overline{y}\}\}$ and $\{\{x,y\},\{\overline{x}\},\{x,\overline{y}\}\}$ for $G$ and observe the relation to our procedures c2s() and fib_c2s(), resp., in Section 1.4 (see [W. Zhang, 1996]).

**Exercise 2.3** *True or false? A* $k$*-CNF formula,* $k \in \mathbb{N}_0$*, with* $2^k$ *clauses is unsatisfiable iff it consists of all* $k$*-clauses over a set of* $k$ *variables.*

**Exercise 2.4** Property B

*A* hypergraph *is a pair* $(V, E)$*,* $V$ *a finite set and* $E \subseteq 2^V$*. It is called* $k$-uniform *if all* hyperedges $e \in E$ *contain exactly* $k$ *elements.*

*Hypergraph* $(V, E)$ *has* property B *if the elements of* $V$ *can be 2-colored so that no hyperedge is monochromatic, i.e. in no hyperedge all elements get the same color.*

*Show that every* $k$*-uniform hypergraph,* $k \in \mathbb{N}$*, with less than* $2^{k-1}$ *hyperedges has property* $B$.

BEWARE: You cannot just employ our results, you have to use the proof ideas.

## 2.2 Number of Dependent Clauses

Here we will considerably improve on the result from the previous section. Instead of considering $k$-CNF formulas with few clauses, we will allow arbitrarily many clauses, but impose some local restriction on the dependency between the clauses.

For $F$ a CNF formula and $C$ a clause, we define the *neighborhood of* $C$ *in* $F$ as the set of clauses distinct from $C$ in $F$ that depend on $C$, formally

$$\Gamma_F(C) := \{C' \in F \mid C' \neq C \text{ and } \mathrm{vbl}(C) \cap \mathrm{vbl}(C') \neq \emptyset\} \, .$$

**Theorem 2.3 ("Lovász Local Lemma")** $k \in \mathbb{N}$*. If* $|\Gamma_F(C)| \leq 2^{k-2}$ *for all clauses* $C$ *in a* $k$*-CNF formula* $F$ *then* $F$ *is satisfiable.*

*Proof.* Let $F$ be as in the presumption of the theorem and let $V := \mathrm{vbl}(F)$. We use $\mathrm{sat}()$ and $\overline{\mathrm{sat}}()$ short for $\mathrm{sat}_V()$ and $\overline{\mathrm{sat}}_V()$, respectively (see Section 1.3).

Consider $G \subseteq F$ and $C \in F \setminus G$. If $C$ is independent from all clauses in $G$ then

$$\frac{|\overline{\mathsf{sat}}(C) \cap \mathsf{sat}(G)|}{|\mathsf{sat}(G)|} = \frac{1}{2^k} . \tag{2.1}$$

Under the assumptions of the theorem we claim that

$$\frac{|\overline{\mathsf{sat}}(C) \cap \mathsf{sat}(G)|}{|\mathsf{sat}(G)|} \leq \frac{1}{2^{k-1}} \tag{2.2}$$

always. This claim is equivalent to

$$\frac{|\mathsf{sat}(G \cup \{C\})|}{|\mathsf{sat}(G)|} = \frac{|\mathsf{sat}(C) \cap \mathsf{sat}(G)|}{|\mathsf{sat}(G)|} \geq 1 - \frac{1}{2^{k-1}}$$

and so it follows (by starting with $G = \{\}$ and successively adding the clauses from $F$) that

$$|\mathsf{sat}(F)| = |\bigcap_{C \in F} \mathsf{sat}(C)| \geq 2^{|V|} \left(1 - \frac{1}{2^{k-1}}\right)^{|F|} > 0 ,$$

therefore $\mathsf{sat}(F) \neq \emptyset$ and the existence of a satisfying assignment is certified.

It remains to prove (2.2). We employ induction on the cardinality of $G$. If $\Gamma_G(C)$ is empty then we can refer to (2.1) which implies (2.2) readily; this also settles the case of $G$ empty.

So let us assume that the claim holds for all sets of clauses smaller than $|G|$ and that $\Gamma_G(C) \neq \emptyset$. Set $G' := G \setminus \Gamma_G(C)$, the set of all clauses in $G$ that are independent of $C$. Due to the induction hypothesis, which may be employed due to $|G'| < |G|$, we have

$$\frac{|\overline{\mathsf{sat}}(D) \cap \mathsf{sat}(G')|}{|\mathsf{sat}(G')|} \leq \frac{1}{2^{k-1}} \qquad \text{for all } D \in \Gamma_G(C). \tag{2.3}$$

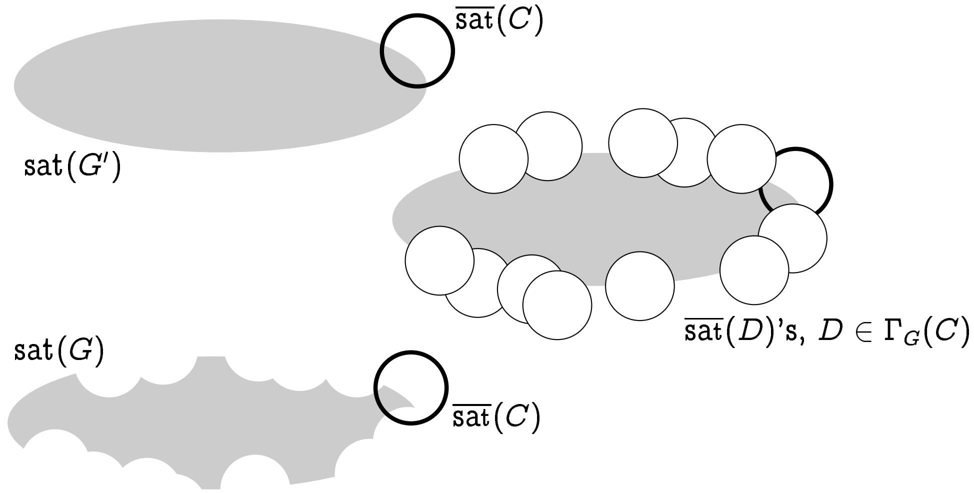



*Figure 2.1: Illustrating the proof of Theorem 2.3*
*We start with $\mathsf{sat}(G')$ of which $\overline{\mathsf{sat}}(C)$ occupies a fraction of $1/2^k$. Then we remove the $\overline{\mathsf{sat}}(D)$'s, $D \in \Gamma_G(C)$, which leaves us with $\mathsf{sat}(G)$, which still has at least half the size of $\mathsf{sat}(G')$. Hence, $\overline{\mathsf{sat}}(C)$ covers at most a fraction $2(1/2^k) = 1/2^{k-1}$ of $\mathsf{sat}(G)$.*

Hence,

$$\begin{aligned}
|\mathsf{sat}(G)| &= \left| \mathsf{sat}(G') \setminus \bigcup_{D \in \Gamma_G(C)} (\overline{\mathsf{sat}}(D) \cap \mathsf{sat}(G')) \right| \\
&\geq |\mathsf{sat}(G')| - \sum_{D \in \Gamma_G(C)} |\overline{\mathsf{sat}}(D) \cap \mathsf{sat}(G')| \\
&\stackrel{(2.3)}{\geq} |\mathsf{sat}(G')| - |\Gamma_G(C)| \cdot \frac{|\mathsf{sat}(G')|}{2^{k-1}} \\
&= |\mathsf{sat}(G')| \left(1 - \frac{|\Gamma_G(C)|}{2^{k-1}}\right) \\
&\geq \frac{|\mathsf{sat}(G')|}{2}
\end{aligned}$$

since $|\Gamma_G(C)| \leq 2^{k-2}$. Now

$$\frac{|\overline{\mathsf{sat}}(C) \cap \mathsf{sat}(G)|}{|\mathsf{sat}(G)|} \leq \frac{|\overline{\mathsf{sat}}(C) \cap \mathsf{sat}(G')|}{|\mathsf{sat}(G')|/2} = \frac{1}{2^{k-1}} ,$$

and (2.2) is established. ◻

The proof discloses no efficient procedure for finding a satisfying assignment under the premises of the theorem. It has been a remarkable breakthrough, when it was shown by [Beck, 1991] that for neighborhoods of size at most $2^{k/48}$ a polynomial procedure exists (for $k$ large enough).

NOTE 2.2 What we called here Lovász Local Lemma is actually just one of the incarnations of it. The lemma was first published in [Erdős, Lovász, 1975], and is usually formulated in terms of probability theory (cf. [Spencer, 1987, Sect. 8], see Exercise 2.7).

**Exercise 2.5** $k \in \mathbb{N}$. *Show that every $k$-CNF formula where every variable occurs (as positive or negated literal) at most $k$ times is satisfiable.*

**Exercise 2.6** Many Variables
*Let $F$ be a CNF formula so that $|G| \leq |\mathrm{vbl}(G)|$ for all $G \subseteq F$. Show that $F$ is satisfiable.*

**Exercise 2.7** $p \in \mathbb{R}$, $0 \leq p < 1$. *Let $\mathcal{A}$ be set of events, so that for all events $A \in \mathcal{A}$ we have $\Pr(A) \leq p$ and $A$ depends on at most $d$ other events in $\mathcal{A}$. Show: If $4dp \leq 1$ then $\Pr\left(\bigwedge_{A\in\mathcal{A}} \overline{A}\right) > 0$, where $\overline{A}$ denotes the complementary event of $A$.*

**Exercise 2.8** $k \in \mathbb{N}$. *Let $d \in \mathbb{N}$ be such that there exists a real number $\delta$, $1 < \delta < 2^k$, with $\delta\left(1 - \frac{\delta}{2^k}\right)^d \geq 1$. Show that every $k$-CNF formula $F$ with $|\Gamma_F(C)| \leq d$ for all $C \in F$ is satisfiable. (Via optimal choice of $d \approx \frac{2^k}{\mathrm{e}}$ and $\delta \approx \mathrm{e}$, this loosens the presumption in Theorem 2.3.)*

## 2.3 Partial Satisfaction

If a CNF formula is not satisfiable, it is sometimes desired to at least satisfy as many clauses as possible. Here we want to investigate, what fraction of the clauses can always be satisfied.

Before we do so, we want to change the set-up in that we are now considering sequences of clauses—called *CNF sequence-formulas*—instead of sets of clauses, allowing repetitions of the same clauses in the sequence. In fact, it is not the order we are concerned about, rather we want to have the possibility to give more weight to certain clauses than others[3]. The reason for doing so is two-fold. On the one hand, in applications it may indeed be so that some clauses are more important than others. On the other hand, this more general set-up will impose itself in the proof of our main result anyways.

Having said this, we can have a look at the sequence of empty clauses only, where we cannot satisfy anything. So let us exclude empty clauses. In a 1-CNF sequence-formula of the form

$$(\{x_1\}, \{\overline{x_1}\}, \{x_2\}, \{\overline{x_2}\}, \ldots, \{x_n\}, \{\overline{x_n}\})$$

we can never satisfy more than half of the clauses and this fraction can be achieved (by every total assignment).

**Theorem 2.4** $k \in \mathbb{N}$. *For every $k$-CNF sequence-formula there exists an assignment that satisfies at least a fraction $\left(1 - \frac{1}{2^k}\right)$ of all clauses. This bound is tight in the worst case, even for repetition-free arbitrarily long CNF sequence-formulas.*

*Proof.* Consider a random assignment, further details omitted (cf. proof of Theorem 2.1).

For proving the bound tight, choose some set $V$ of at least $k$ variables, and consider the sequence with each $k$-clause over $V$ occurring exactly once. $\square$

What can be said in general? If all we exclude are empty clauses, nothing better than a fraction of $\frac{1}{2}$ can be guaranteed. Is it only the "notorious" pairs $\{x\}, \{\overline{x}\}$ that can cause this fraction to be so small?

We call a CNF sequence-formula 2-*satisfiable*, if for any two clauses in the sequence there is an assignment that simultaneously satisfies both. As it can be easily seen, being 2-satisfiable is equivalent to excluding empty clauses and pairs of the form $\{x\}, \{\overline{x}\}$.

[3]We might as well have considered multi-sets or weighted sets.

**Theorem 2.5 ([Lieberherr, Specker, 1981])** *Every 2-satisfiable CNF sequence-formula allows an assignment that satisfies a fraction*[4] $h := (\sqrt{5} - 1)/2) \approx 0.618$ *of all clauses. This bound is tight.*

*Proof.* We transform a given 2-satisfiable sequence-formula $S$ into a 2-satisfiable sequence-formula $S_1$ and then into $S_2$ so that if the claim of the theorem holds for $S_2$, then it holds for $S_1$, and if it holds for $S_1$ then it holds for $S$. The clue is that $S_2$ will have a symmetric structure that will allow us to verify the claim (although, as it may come as a surprise, $S_2$ is going to be huge compared to $S$).

A CNF sequence-formula is called a *RED1 sequence-formula*, if all of its clauses are of the form $\{x\}$ or $\{\overline{x}, \overline{y}\}$, for $x$ and $y$ variables.

We transform $S$ into a RED1 sequence-formula $S_1$ in three steps.

(i) First, as long as there is a clause $\{\overline{x}\}$, $x$ a variable, we substitute $x$ in all clauses by $\overline{x}$ and vice versa. Since for no variable $x$ both clauses $\{x\}$ and $\{\overline{x}\}$ appear, this process will eventually terminate and eliminate all clauses of the form $\{\overline{x}\}$. And we also see that the number of simultaneously satisfiable clauses does not alter in this way. (ii) In the next step, in all clauses that contain a positive literal, we remove all literals but one such positive literal. Having done this wherever possible, we are left with 1-clauses containing a positive literal each, and clauses that contain negative literals only, at least two of them. (iii) For clauses of the latter type, we remove all literals but two. The resulting sequence is denoted by $S_1$. It is a RED1 sequence-formula. In steps (ii) and (iii) of the process, the number of simultaneously satisfiable clauses cannot increase. Note that the length of $S_1$ is still the same as the length of $S$. However, clauses that have been distinct in $S$ may be equal now due to our removals of literals.[5] A possible way of proceeding

[4] $h$ is the reciprocal of the golden ratio.

[5] If we hadn't done it before, here we would have been forced to take multiplicities into consideration.

would be

$$\begin{aligned} S &= (\{\overline{x}\},\{\overline{x},y\},\{x,\overline{y},\overline{z}\}) \\ &\overset{(i)}{\leadsto} (\{x\},\{x,y\},\{\overline{x},\overline{y},\overline{z}\}) \\ &\overset{(ii)}{\leadsto} (\{x\},\{x\},\{\overline{x},\overline{y},\overline{z}\}) \\ &\overset{(iii)}{\leadsto} (\{x\},\{x\},\{\overline{x},\overline{y}\}) = S_1 \ . \end{aligned}$$

A CNF sequence-formula over a set $V$ of variables is called a *RED2 sequence-formula over* $V$, if it is RED1, and the number of occurrences of $\{x\}$ is the same for all $x \in V$, and the number of occurrences of $\{\overline{x},\overline{y}\}$ is the same for all $\{x,y\} \in \binom{V}{2}$.

We choose some $V \supseteq \mathrm{vbl}(S_1)$. For $\pi$ a permutation of $V$, we let $\pi(S_1)$ denote the sequence-formula which is obtained by substituting $\pi(x)$ for $x$ (and $\overline{\pi(x)}$ for $\overline{x}$) for all $x$ in $V$. Now $S_2$ is the sequence-formula obtained by concatenating (in some order) $\pi(S_1)$ over all permutations $\pi$ of $V$. For example, if $S_1 = (\{x\},\{x\},\{\overline{x},\overline{y}\})$ and $V = \{x,y,z\}$, then

$$\begin{aligned} S_2 \;=\; & (\overbrace{\{x\},\{x\},\{\overline{x},\overline{y}\}}^{x\mapsto x,y\mapsto y,z\mapsto z},\overbrace{\{y\},\{y\},\{\overline{y},\overline{x}\}}^{x\mapsto y,y\mapsto x,z\mapsto z},\overbrace{\{z\},\{z\},\{\overline{z},\overline{y}\}}^{x\mapsto z,y\mapsto y,z\mapsto x}, \\ & \underbrace{\{x\},\{x\},\{\overline{x},\overline{z}\}}_{x\mapsto x,y\mapsto z,z\mapsto y},\underbrace{\{y\},\{y\},\{\overline{y},\overline{z}\}}_{x\mapsto y,y\mapsto z,z\mapsto x},\underbrace{\{z\},\{z\},\{\overline{z},\overline{x}\}}_{x\mapsto z,y\mapsto x,z\mapsto y}) \ . \end{aligned}$$

Let $n := |V|$, $m_1$ the number of 1-clauses in $S_1$, $m_2$ the number of 2-clauses in $S_1$, and $m := m_1 + m_2$ the length of $S_1$. $S_2$ "explodes" to a formula of $n!\,m$ clauses. For each $x \in V$, the number of occurrences of $\{x\}$ in $S_2$ is $(n-1)!\,m_1$ since every 1-clause $\{z\}$ in $S_1$ maps to $\{x\}$ in $\pi(S_1)$ for $(n-1)!$ permutations $\pi$ out of the $n!$ permutations of $V$. Similarly, for each $\{x,y\} \in \binom{V}{2}$, the number of occurrences of $\{\overline{x},\overline{y}\}$ in $S_2$ is[6] $2\,(n-2)!m_2$. Therefore, $S_2$ is RED2. If an assignment satisfies a fraction $h$ of the clauses in $S_2$, then it has to satisfy a fraction $h$ of the clauses of at least one of the $\pi(S_1)$, and hence, there is also an assignment that satisfies a fraction $h$ of the clauses in $S_1$ (and thus in $S$, as we have argued before).

The RED2 sequence-formula $S_2$ is described for our purposes by three integers: $n := \mathrm{vbl}(S_2)$; $\mu_1$, the multiplicities of clauses $\{x\}$ in $S_2$; and $\mu_2$,

[6]Let's check that indeed $n\,(n-1)!\,m_1 + \binom{n}{2}\,2\,(n-2)!\,m_2 = n!\,m$.

the multiplicities of clauses $\{\overline{x}, \overline{y}\}$ in $S_2$. The length of $S_2$ is

$$n\,\mu_1 + \binom{n}{2}\,\mu_2$$

and if an assignment on $V$ sets exactly $k$ variables to 1, then the number of satisfied clauses is

$$k\,\mu_1 + \left(\binom{n}{2} - \binom{k}{2}\right)\,\mu_2\ .$$

That is, the fraction of satisfied clauses is

$$\frac{k\varphi + \binom{n}{2} - \binom{k}{2}}{n\varphi + \binom{n}{2}} \tag{2.4}$$

where $\varphi := \frac{\mu_1}{\mu_2}$ (here we have to assume $\mu_2 > 0$; if $\mu_2 = 0$, then we can satisfy all clauses). All that is left to show is that for every positive rational number $\varphi$ there is an integer $k$, $0 \le k \le n$, such that the expression (2.4) is at least $h$ from the assertion of the theorem. This can be shown by a tedious elementary calculus argument, which we can spare ourselves as follows. Recall that $\mu_1 = (n-1)!\,m_1$ and $\mu_2 = 2\,(n-2)!\,m_2$ and thus $\varphi = \frac{(n-1)}{2}\frac{m_1}{m_2}$. Hence, if we choose $V$ such that $n-1$ is even and an integer multiple of $m_2$ then $\varphi$ is an integer.

**Case 1**: $\varphi \ge n$ *(i.e. when* $\frac{m_1}{2m_2} > 1$*)*. Then we set $k = n$ and (2.4) becomes

$$\frac{n\varphi}{n\varphi + \binom{n}{2}}$$

which exceeds $\frac{2}{3} > h$ for $\varphi \ge n$.

**Case 2**: $0 \le \varphi < n$ *(i.e. when* $\frac{m_1}{2m_2} \le 1$*)*. We can assume that $n$ is as large as we like. Observe that for $k = \varphi$ and for[7] $n \to \infty$

$$(2.4) \to \frac{\varphi^2 + n^2}{2n\varphi + n^2}\ . \tag{2.5}$$

[7] What kind of limit "$f(n) \to g(n)$ for $n \to \infty$" is this in (2.5)? It stands for $\lim_{n\to\infty}(f(n) - g(n)) = 0$.

For $\varphi \in \mathbb{R} \setminus \{-\frac{n}{2}\}$, this expression has its extrema when $\varphi^2 + \varphi n - n^2 = 0$; the only positive root is $\varphi = n \, \frac{-1+\sqrt{5}}{2} = n\, h$ which lets (2.5) evaluate to (its minimum over $\varphi \in \mathbb{R}$) $\frac{h^2+1}{2h+1}$ which equals $h$. Therefore, for $n$ large enough, the fraction exceeds[8] $h$ for $k = \varphi$. This concludes the proof of the first part of the theorem (recall that we can make $n$ as large as we like as part of the proof—we do not rely on the original $S$ to be large enough).

It remains to show $h$ to be best possible. For $n \in \mathbb{N}$, $n \geq 2$, we consider a RED2 sequence-formula with $n$ variables, $nf_n$ repetitions of 1-clauses, and $f_{n+1}$ repetitions of 2-clauses, where $f_n$ is the $n$th Fibonacci number. Note that $\varphi_n := \frac{nf_n}{f_{n+1}} \to hn$ as $n \to \infty$.

If $k$ variables are set to 1 then the fraction of satisfied clauses is

$$\frac{k\varphi_n + \binom{n}{2} - \binom{k}{2}}{n\varphi_n + \binom{n}{2}} \to \frac{2k\varphi_n + n^2 - k^2}{2n\varphi_n + n^2} \quad \text{as } n \to \infty \text{ and for } 0 \leq k \leq n \; .$$

The right hand term is maximized for $k = \varphi_n$, when it equals (2.5) for $\varphi = \varphi_n$. Since $\varphi_n$ approaches $nh$ as $n \to \infty$, and (2.5) evaluates to $h$ for $\varphi = nh$, we are done. ⊡

NOTE 2.3 Theorem 2.5 and its proof are from [Lieberherr, Specker, 1981]. The main method employed in the transformation from $S_1$ to $S_2$ is called *symmetrization* (see [Erdős, Kleitman, 1968] and also Exercise 2.17).

As presented here, it appears that the proof does not supply us with a polynomial method for finding an assignment that satisfies a fraction $h$ of the clauses in a 2-satisfiable CNF sequence-formula, primarily, since $S_2$ has size exponential in that of $S$. That can be repaired as shown by [Lieberherr, Specker, 1981] (see Exercises 2.13, 2.14 and 2.15).

In fact, they show the following. There is a polynomial algorithm that, given a 2-satisfiable CNF sequence-formula, finds an assignment that satisfies a fraction $h$ of all clauses. And for $h' > h$, it is NP-complete to decide whether there is an assignment that satisfies a fraction $h'$ of the clauses.

NOTE 2.4 It is perhaps tempting[9] to consider the following procedure for proving the bound. Consider a RED1 sequence-formula $S$ with $m_1$ the number of 1-clauses

[8] It does indeed exceed, since the fraction is rational and $h$ is irrational.

[9] So it obviously was for the author.

and $m_2$ the number of 2-clauses in $S$. Moreover, for $x \in V$, let $m_{1,x}$ be the number of clauses $\{x\}$ in $S$, and let $m_{2,x}$ be the number of clauses that contain $\overline{x}$. Note that $m_1 = \sum_{x \in \text{vbl}(S)} m_{1,x}$ and $m_2 = \frac{1}{2} \sum_{x \in \text{vbl}(S)} m_{2,x}$ .

Let $\lambda$, $0 < \lambda < 1$ be a real number to be determined later (it is the fraction of satisfied clauses we want to guarantee to be satisfied with the procedure). Suppose there is a variable $x$ with

$$m_{2,x} \geq \lambda(m_{1,x} + m_{2,x}) \qquad (\Leftrightarrow m_{1,x} \leq \tfrac{1-\lambda}{\lambda} m_{2,x}). \tag{2.6}$$

Then the assignment $x \mapsto 0$ satisfies $m_{2,x}$ clauses and makes $m_{1,x}$ clauses unsatisfiable. The ratio "satisfied clauses versus all clauses determined" is at least $\lambda$. Now we continue with $S^{[x \mapsto 0]} \setminus \{\Box\}$ which is again a RED1 multi-formula (by the way, $S^{[x \mapsto 1]}$ is not). As long as we can continue this way we know that a fraction $\lambda$ of the clauses we lose on the way is being satisfied by our choices.

At some point there may be no $x$ satisfying (2.6) (all of our notation – $S$, $m_1$, $m_{1,x}$, etc. – refers to the current multi-formula in the process). It follows that

$$m_1 = \sum_{x \in \text{vbl}(S)} m_{1,x} > \frac{1-\lambda}{\lambda} \sum_{x \in \text{vbl}(S)} m_{2,x} = 2\frac{1-\lambda}{\lambda} m_2$$

i.e.

$$m_1 > 2\frac{1-\lambda}{2-\lambda}(m_1 + m_2) \ .$$

So if we set now all remaining variables to 1, the overall process has generated an assignment that satisfies a fraction $\min\{\lambda, 2\frac{1-\lambda}{2-\lambda}\}$ of the clauses. We set $\lambda$ to $2 - \sqrt{2} \approx 0.586$ (from the equality $\lambda = 2\frac{1-\lambda}{2-\lambda}$) which guarantees us this value as the fraction of satisfied clauses; close but short of the best possible bound.

NOTE **2.5** Yet another attempt via the probabilistic method. Again we start with a RED1 sequence-formula $S$ over $V$ of length $m$ with $m_1$ 1-clauses (of type $\{x\}$) and thus $m_2 := m - m_1$ the number of 2-clauses (of type $\{\overline{x}, \overline{y}\}$). We assume $m_2 > 0$; otherwise it is trivial to satisfy all clauses.

For some probability $p \in \mathbb{R}$, $0 \leq p \leq 1$, we let $\alpha$ be a random assignment that lets each variable $x \in V$ map to 1 with probability $p$, all events mutually independent.

Thus a clause of type $\{x\}$ will be satisfied with probability $p$, and a clause of type $\{\overline{x}, \overline{y}\}$ will be satisfied with probability $1 - p^2$. Consequently, the expected number of satisfied clauses is $pm_1 + (1 - p^2)m_2$, the fraction of satisfied clauses is

$$f(p, \eta) := (1 - \eta)p + \eta(1 - p^2) \quad \text{where } \eta := \tfrac{m_2}{m},\ 0 < \eta \leq 1.$$

For $\eta$ fixed, this function attains its maximum at $p = p^* := \frac{1-\eta}{2\eta}$. $p^*$ fails to be a probability ($p^* > 1$) iff $\eta < \frac{1}{3}$. But then $f(1, \eta) = 1 - \eta > \frac{2}{3} > h$, that is, setting all variables to 1 satisfies at least two thirds of the clauses. We consider

$$g(\eta) := f(p^*, \eta) = \frac{1}{4}\left(5\eta + \frac{1}{\eta} - 2\right) \quad \text{in the range } \tfrac{1}{3} \leq \eta \leq 1.$$

There $g$ attains its smallest value $\frac{\sqrt{5}-1}{2} = h$ for $\eta = \frac{1}{\sqrt{5}}$.

Summing up, for all $\eta$, $0 \leq \eta < \frac{1}{3}$, setting all variables to 1 satisfies at least $\frac{2}{3}$ of the clauses, and for all $\eta$, $\frac{1}{3} \leq \eta \leq 1$, setting each variable to 1 with probability $p^* = \frac{1-\eta}{2\eta}$ has an expected fraction at least $h$ of the clauses satisfied. Hence, there always exists an assignment that satisfies at least a fraction $h$ of the clauses of a 2-satisfiable CNF sequence-formula.

**Exercise 2.9** $k$-Satisfiable

*$k \in \mathbb{N}$. A CNF sequence-formula is* $k$-satisfiable *if for any $k$ clauses of the sequence there is an assignment that simultaneously satisfies all of them.*

*True or false? A CNF sequence-formula is 3-satisfiable iff it neither contains $\Box$, nor pairs $\{u\}, \{\overline{u}\}$, nor triples $\{u\}, \{\overline{u}, v\}, \{\overline{u}, \overline{v}\}$, nor triples $\{u\}, \{v\}, \{\overline{u}, \overline{v}\}$ for literals $u$ and $v$.*

**Exercise 2.10** 3-Satisfiable

*Show that in a 3-satisfiable sequence-formula it is always possible to satisfy $\frac{2}{3}$ of the clauses.*

**Exercise 2.11 (Open problem)** *What fraction of simultaneously satisfiable clauses can be guaranteed for $k$-satisfiable sequence-formulas?* REMARK: This fraction cannot go beyond $\frac{3}{4}$ [Huang, Lieberherr, 1985].

**Exercise 2.12 (Open problem)** *Is $h$ the tight fraction for arbitrarily long repetition-free 2-satisfiable CNF sequence-formulas?*

**Exercise 2.13** Doubly Transitive Permutation Groups

*Let $V$ be a set. A permutation group $G$ (in the algebraic sense) of permutations $V \longrightarrow V$ is called* doubly transitive *if for all $x, x', y, y' \in V$*

*with $x \neq x'$ and $y \neq y'$ there is a permutation $\pi \in G$ such that $\pi(x) = y$ and $\pi(x') = y'$.*

*Let $G$ be a doubly transitive permutation group (on some finite $V$ with $|V| \geq 2$). Let $|V| \geq 2$ be finite, and fix some $x, y \in V$, $x \neq y$. Let $\ell_1$ be the number of permutations $\pi$ in $G$ with $\pi(x) = x$ and let $\ell_2$ be the number of permutations $\pi$ in $G$ with $\pi(x) = x$ and $\pi(y) = y$. Show that for all $x, y \in V$ the number of permutations $\pi$ with $\pi(x) = y$ is $\ell_1$, and for all $x, x', y, y' \in V$ with $x \neq x'$ and $y \neq y'$ the number of permutations $\pi$ in $G$ with $\pi(x) = y$ and $\pi(x') = y'$ is $\ell_2$.*

**Exercise 2.14** *Convince yourself that doubly transitive permutation groups serve the purpose in the proof of Theorem 2.5 just as well as the full symmetric group we have employed.*

**Exercise 2.15** *Fix some prime number $p$ and let $V := \{0..p-1\}$. Show that the set of permutations*

$$\pi_{a,b} : \quad i \mapsto (a\, i + b) \bmod p \ ,$$

*for $1 \leq a \leq p-1$ and $0 \leq b \leq p-1$, forms a doubly transitive permutation group of order $p(p-1)$.*

**Exercise 2.16** *Let $n, m \in \mathbb{N}$, $m \leq n$ and $r := n \bmod m$. Show that the vertices of the complete $n$-vertex graph $K_n$ can be colored with $m$ colors so that the fraction of edges incident to two distinctly colored vertices is*

$$h(n, m) := \frac{n^2(m-1) + r(r-m)}{mn(n-1)} \ .$$

**Exercise 2.17** *Let $n, m \in \mathbb{N}$, $m \leq n$. Show that for every $n$-vertex graph $G$ there is a coloring with $m$ colors so that the fraction of edges incident to two distinctly colored vertices is at least $h(n, m)$ (from Exercise 2.16).*

**Exercise 2.18** *Give an efficient derandomization of the probabilistic proof in Note 2.5.*

**Exercise 2.19** One of Few always Works

*Let $V$, $|V| = n$, a set of variables be given. Then there exists a set $A$ of at most $2n$ assignments on $V$ such that for every 2-CNF formula $F$ over $V$ there is an assignment $\alpha$ on $V$ which satisfies at least $\frac{3}{4}$ of all clauses in $F$.*

HINT: Use Hadamard matrices.

# Chapter 3

# Algorithms for 2-SAT

A major part of the course will investigate the *$k$-SAT problem*[1] for $k \in \mathbb{N}$, which reads as follows.

$k$-SAT
Input: *A $(\leq k)$-CNF formula $F$.*
Output: *'Yes', if $F$ is satisfiable. And 'No', otherwise.*

Note that we do not explicitly require a satisfying assignment in case of 'Yes', but that's the way it goes with decision problems. Along the result from Section 2.2 we are quite aware that this may indeed make a difference[2]—but see Exercise 3.1.

In this section we focus on 2-SAT. For a starter we observe that if two $(\leq 2)$-clauses have a resolvent, then this resolvent is still a $(\leq 2)$-clause. So if we keep adding resolvents to a $(\leq 2)$-CNF formula $F$ over $n$ variables, the resulting formulas are still $(\leq 2)$-CNF formulas over $n$ variables. It follows that at most $1 + 2n + 4\binom{n}{2} = 2n^2 + 1$ clauses can ever occur in the process (see Exercise 1.17), and thus the process terminates adding at most $O(n^2)$

[1]You may argue, that this should actually be called $(\leq k)$-SAT problem. I do agree. On the other hand, the notion of $k$-SAT has been pretty much established—although not even that consistently—in the way I present it now.

[2]A prominent example of that type is the question whether a number is composite. This can be decided in polynomial time, while nobody knows how the factors of a large composite number can be found efficiently—and we, as a more and more crypto-dependent society, hope that it stays this way.

resolvents. If we encounter the empty clause, then $F$ is not satisfiable; if not, it is. Hence, resolution is a *polynomial* method for $(\leq 2)$-CNF formulas and the polynomial time solvability of 2-SAT is established.

It seems like we are done with 2-SAT. Nevertheless we will spend the whole chapter on the topic. First we will present another simple algorithm for 2-SAT, which will offer a chance to discuss *unit clause reduction*. Then we introduce a randomized algorithm for 2-SAT that is interesting from several points of view, since it comes as a very natural procedure, and since it is a good (and smooth) introduction to randomized algorithms for $k$-SAT, $k > 2$, to be treated later in this course. Finally, we will present a method that is not only polynomial, but actually linear in the number of clauses. It will force us (quite opposed to the randomized algorithm) to really understand the structure of $(\leq 2)$-CNF formulas. Finally, we will touch a few examples where we can exploit the polynomiality of 2-SAT.

**Exercise 3.1** Decision vs. Construction
*Show that if we can decide satisfiability of CNF formulas in polynomial time, then we can actually find a satisfying assignment of satisfiable CNF formulas in polynomial time.*

**Exercise 3.2** *Provide a concrete asymptotic bound in terms of $n$ for an implementation of the resolution method for 2-SAT. Can you find an implementation that is faster than $O(n^3)$?*
REMARK: By an 'implementation' I do not mean an actual program that runs on a computer, rather I mean a more specific description of the algorithm with data structures that allow a detailed analysis of the asymptotic running time. It is part of the exercise to develop a good representation of a CNF formula. After having said this, let me emphasize that there is nothing wrong with writing actual programs and running experiments with them. In fact, I encourage you to do so.

**Exercise 3.3** Satisfiable with many Clauses
*What is the maximum number of clauses in a satisfiable $(\leq 2)$-CNF formula over $n$ variables?*

## 3.1 Unit Clause Reduction

This section builds on the following observation. If a CNF formula $F$ contains some 1-clause $\{u\}$ then in order to satisfy $F$ (if at all possible) we have no choice other than assigning 1 to $u$. So we may as well do so, that is, we consider satisfiability of the smaller CNF formula $F^{[u\mapsto 1]}$. In fact, even the number of satisfying assignments is preserved in this way (see Observation 1.2).

**Observation 3.1** *Let $F$ be a CNF formula over $V$ containing a 1-clause $\{u\}$. Then $|\mathrm{sat}_V(F)| = |\mathrm{sat}_{V\setminus \mathrm{vbl}(\{u\})}\left(F^{[u\mapsto 1]}\right)|$. In particular, $F$ is satisfiable iff $F^{[u\mapsto 1]}$ is satisfiable.*

Clearly, we can iterate this step, called *unit clause rule* until no 1-clause remains, as done in procedure uc(). If we ever encounter the empty clause in the process, we may conclude unsatisfiability of the formula we started with. Otherwise, other measures have to be taken.

| | |
|---|---|
| $V$ variables, $F$<br>CNF formula over $V$.<br>POSTCONDITION:<br>returns $U \subseteq V$ and<br>$G = F^{[\alpha]}$, for some<br>$\alpha \in \{0,1\}^{V\setminus U}$, without<br>1-clauses s.t.<br>$\lvert\mathrm{sat}_V(F)\rvert = \lvert\mathrm{sat}_U(G)\rvert$. | **function** uc($F, V$)<br>$G \leftarrow F$; $U \leftarrow V$;<br>**while** $\exists$ 1-clause in $G$ **do**<br>    $\{u\} \leftarrow_{\text{some 1-clause in}} G$;<br>    $G \leftarrow G^{[u\mapsto 1]}$;<br>    $U \leftarrow U \setminus \mathrm{vbl}(\{u\})$;<br>**return** $(G, U)$; |

What we have described so far is true for all CNF formulas, so let us now specialize to a ($\leq 2$)-CNF formula $F$. If we apply uc() to $F$, then we get back $G$ which either contains $\Box$—which lets us conclude unsatisfiability—, or $G$ has 2-clauses only. The following lemma suggests how to further proceed with a 2-CNF formula.

**Lemma 3.2** *Let $F$ be a 2-CNF formula over $V$ and let $x \in V$. Let $G$ be the formula returned by* uc$(F^{[x\mapsto 1]}, V \setminus \{x\})$.

(i) *In case $\Box \notin G$: $F$ is satisfiable iff $G$ is satisfiable.*

(ii) *In case* $\Box \in G$*: $F$ is satisfiable iff $F^{[x\mapsto 0]}$ is satisfiable.*

*Proof.* For (ii), we recall that by the nature (and postcondition) of procedure uc(), CNF formula $G$ is satisfiable iff $F^{[x\mapsto 1]}$ is satisfiable. Hence, if $\Box \in G$, $F^{[x\mapsto 1]}$ is not satisfiable, and along Observation 1.2(ii), $F$ is satisfiable iff $F^{[x\mapsto 0]}$ is satisfiable.[3]

For (i), let us first observe that uc() implicitly builds up an assignment $\alpha$ on $(V \setminus \{x\}) \setminus U$ so that $G = F^{[x\mapsto 1][\alpha]}$. The key observation is that if $\Box \notin G$, then $G$ has to be a 2-CNF formula, and $G \subseteq F$.

Therefore, if $F$ is satisfiable, then $G$ is satisfiable and "$\Rightarrow$" is shown.

Note that $\alpha$ satisfies $F^{[x\mapsto 1]} \setminus G$. So to show "$\Leftarrow$", let $\beta$ be an assignment on $\mathrm{vbl}(G) \subseteq U$ (this is disjoint from $V \setminus U$) that satisfies $G$. Then $x \mapsto 1$, $\alpha$ and $\beta$ can be combined to an assignment that satisfies $F$. ◻

As important as understanding the proof, it is to understand why it fails for a general CNF formula: in general, we cannot conclude $G \subseteq F$ in case of $\Box \notin G$, and thus it may indeed happen that $G \not\ni \Box$ is not satisfiable, while $F$ is.

| | |
|---|---|
| $V$ variables,<br>$F$ CNF formula over $V$.<br>PRECONDITION:<br>$F$ is $(\leq 2)$-CNF.<br>POSTCONDITION:<br>returns 'Yes', if $F$ is satisfiable, and 'No', otherwise. | **function** uc_2s$(F, V)$<br>$(F, V) \leftarrow \mathrm{uc}(F, V)$;<br>**if** $\Box \in F$ **then return** 'No';<br>**elseif** $F = \{\}$ **then return** 'Yes';<br>**else**<br>    $x \leftarrow_{\text{some in}} V$;<br>    $(G, U) \leftarrow \mathrm{uc}(F^{[x\mapsto 1]}, V \setminus \{x\})$;<br>    **if** $\Box \notin G$ **then return** uc_2s$(G, U)$;<br>    **else return** uc_2s$(F^{[x\mapsto 0]}, V \setminus \{x\})$; |

We are ready to describe uc_2s(), which exploits the claims of Lemma 3.2 for an efficient satisfiability test for $(\leq 2)$-CNF formulas. Note that the work done in the procedure can be performed in polynomial time, and there is just one recursive call to uc_2s(), where the second argument (the variable

[3] Note that (ii) does not rely on $F$ being a 2-CNF formula.

set) decreases by one element at least. Hence, the procedure is polynomial for ($\leq 2$)-CNF formulas.

The algorithm of this section can be found in [Lewis, Papadimitriou, 1998, Section 6.3].

**Exercise 3.4** *Apply* uc() *to (the set version of) CNF formula*

$$(\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge \ldots \wedge (\overline{x_{n-1}} \vee x_n) \wedge (\overline{x_n} \vee x_1) \wedge (x_1) \ ,$$

*with variable set* $\{x_1, x_2, \ldots, x_n\}$.

**Exercise 3.5** *Given a CNF formula* $F$ *over* $V$, *procedure* $\mathrm{uc}(F, V)$ *returns in polynomial time a pair* $(G, U)$, *such that* $G$ *is a CNF formula over* $U$ *without* 1*-clauses,* $|U| \leq |V|$, *and* $|\mathrm{sat}_U(G)| = |\mathrm{sat}_V(F)|$. *Provide an alternative polynomial time procedure, that in addition to the above properties guarantees that no two* 2*-clauses have the same variable set.*
HINT: If there are four 2-clauses with the same variable set, the formula is not satisfiable, and we can safely return $(\{\square\}, \emptyset)$. If there are three 2-clauses with the same variable set, then the values of the involved variables are determined in a satisfying assignment, and we can simplify. But what can we say about the variables involved, if two 2-clauses have the same variable set?

**Exercise 3.6** *Provide an example for the claim made after the proof of Lemma 3.2. That is, you have to find a* satisfiable *CNF formula* $F$ *over some variable set* $V$ *and a variable* $x \in V$, *so that for the formula* $G$ *returned by* $\mathrm{uc}(F^{[x\mapsto 1]}, V \setminus \{x\})$ *we have* $\square \notin G$ *and* $G$ *is not satisfiable.*

**Exercise 3.7** *Given what you have seen in this section, any ideas for (further) improvements on counting the number of satisfying assignments of a* ($\leq 2$)*-CNF formula?*

## 3.2 A Randomized Algorithm

Suppose we are confronted with a CNF formula which we are promised[4] to be satisfiable. In order to find such an assignment, [Papadimitriou, 1991]

[4] Even without trust to an authority, we may know of the existence of a satisfying assignment without having one in our hands, see Theorem 2.3 and Exercise 2.5.

suggested the following simple and natural approach.

> *"Start with any truth assignment. While there are unsatisfied clauses, pick any one, and flip a random literal in it."*

A more formal description of the method is given via procedure rf().

| | |
|---|---|
| $V$ variables, $F$ CNF formula over $V$.<br>PRECONDITION:<br>$F$ is satisfiable.<br>POSTCONDITION:<br>returns assignment $\alpha$ on $V$ satisfying $F$, if it terminates. | **function** rf$(F, V)$<br>$\alpha \leftarrow_{\text{some in}} \{0,1\}^V$;<br>**while** $\alpha$ does not satisfy $F$ **do**<br>    $C \leftarrow_{\text{some unsatisfied in}} F$;<br>    $u \leftarrow_{\text{u.a.r. in}} C$;<br>    flip assignment for $u$ in $\alpha$;<br>**return** $\alpha$; |

Note here the fine distinction between the "$\leftarrow_{\text{some}}$"- and the "$\leftarrow_{\text{u.a.r.}}$"-assignment. While in the first one we allow even an adversary to make the choice (as bad as it may be for the success of the procedure), the latter allows for probabilistic reasoning as done below. Also, the postcondition has to be understood with care: it is not clear that the procedure ever finds a satisfying assignment, and it will be the goal of this section to show that for a satisfiable ($\leq 2$)-CNF formula this is indeed the case with probability 1, and the expected time that elapses till then is bounded by a polynomial.

For the analysis of rf$(F, V)$ for a satisfiable ($\leq 2$)-CNF formula we fix some satisfying assignment $\alpha^* \in \{0,1\}^V$ and observe the *Hamming distance* $d_H(\alpha, \alpha^*) := |\{x \in V \mid \alpha(x) \neq \alpha^*(x)\}|$ between $\alpha$ and $\alpha^*$.

- $d_H(\alpha, \alpha^*) \leq n := |V|$, always.

- At any step before termination, $d_H(\alpha, \alpha^*)$ increases or decreases by 1, with the probability of decrease at least $\frac{1}{2}$. This holds, since $\alpha$ has to disagree with $\alpha^*$ in at least one literal $v$ in the chosen clause $C$—after all, $\alpha$ does not satisfy $C$ while $\alpha^*$ does. Because $C$ is a ($\leq 2$)-clause, we choose $v$ with probability at least $\frac{1}{2}$ and in that case the Hamming distance decreases.

- The algorithm stops when $d_H(\alpha, \alpha^*) = 0$ the latest (but possibly before that, since there may be satisfying assignments other than $\alpha^*$).

So the Hamming distance moves in $\{0..n\}$ with step size 1, always has a tendency towards 0 of probability at least $\frac{1}{2}$ and the process has stopped when it is 0 the latest. For an analysis of this process we recapitulate symmetric random walks.

### 3.2.1 Symmetric Random Walks

A *symmetric random walk on* $\mathbb{Z}$ "moves" in state space $\mathbb{Z}$ with the rule that in each step, the current number (state) will be increased or decreased with equal probability $\frac{1}{2}$, independently from the history of the motion. More formally, if we let $W_t \in \mathbb{Z}$ denote the state of the process at time $t \in \mathbb{N}_0$, then $(W_t)_{t\in\mathbb{N}_0}$ is a homogeneous Markov chain (see Appendix A.3) with $W_0 \in \mathbb{Z}$ and

$$\Pr(W_t = i+1 \,|\, W_{t-1} = i) = \Pr(W_t = i-1 \,|\, W_{t-1} = i) = \frac{1}{2}$$

for all $t \in \mathbb{N}$, $i \in \mathbb{Z}$.

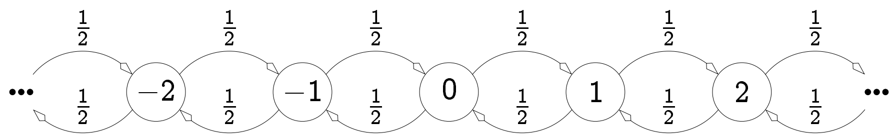



*Figure 3.1: Symmetric random walk.*

**Lemma 3.3** $(W_t)_{t\in\mathbb{N}_0}$ *as above,* $a, b, s \in \mathbb{Z}$, $a \le s \le b$.
$\mathrm{E}(\min\{t \mid W_t \in \{a,b\}\} \,|\, W_0 = s)$, *the expected number of steps up to the first encounter of* $a$ *or* $b$ *provided* $W_0 = s$, *is* $(s-a)(b-s)$.

*Proof (sketch).* For $a, b$ fixed, let us denote the expectation under consideration by $e_s$, $s \in \mathbb{Z}$. It observes

$$e_s = \begin{cases} 0 & \text{if } s \in \{a,b\}, \text{ and} \\ 1 + \frac{1}{2}(e_{s-1} + e_{s+1}) & \text{otherwise.} \end{cases} \tag{3.1}$$

And $e_s = (s-a)(b-s)$ does fulfill these identities.

For a full proof we would have to argue that for $a \le s \le b$ the equations (3.1) have a unique solution, and that $e_s$ is indeed finite.[5] $\square$

For $n \in \mathbb{N}_0$, a *symmetric random walk reflecting at $n$* is a homogeneous Markov chain $(R_t)_{t\in\mathbb{N}_0}$ with state space $\mathbb{Z}_{\le n} := \{i \in \mathbb{Z} \mid i \le n\}$ and transition probabilities

$$\Pr(R_t = i+1 \,|\, R_{t-1} = i) = \Pr(R_t = i-1 \,|\, R_{t-1} = i) = \frac{1}{2}$$

for all $i < n$, $t \in \mathbb{N}$, and $\Pr(R_t = n-1 \,|\, R_{t-1} = n) = 1$ for all $t \in \mathbb{N}$.

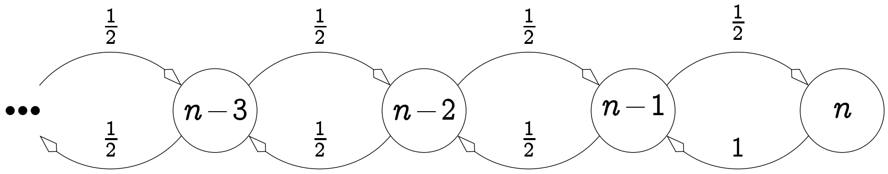



*Figure 3.2: Symmetric random walk reflecting at $n$.*

**Lemma 3.4** *$n$, $(R_t)_{t\in\mathbb{N}_0}$ as above, $s \in \mathbb{N}_0$, $s \le n$.*
$\mathrm{E}(\min\{t \mid R_t = 0\} \,|\, R_0 = s)$*, the expected number of steps up to the first encounter of* 0 *provided $R_0 = s$, is $s(2n-s)$.*

*Proof.* Note that if $(W_t)_{t\in\mathbb{N}_0}$ is a symmetric random walk on $\mathbb{Z}$, then $(n - |W_t|)_{t\in\mathbb{N}_0}$ is a symmetric random walk reflecting at $n$. It starts at $n - |W_0|$ which equals $s$ for $W_0 = n - s$, and it reaches 0 iff $W_t \in \{-n, +n\}$. Due to Lemma 3.3 it follows that the expected number of steps until this happens is $((n-s) - (-n))(n - (n-s)) = s(2n-s)$. $\square$

Intuition suggests that rf() terminates at least as fast as $(R_t)_{t\in\mathbb{N}_0}$ reaches 0. But how can we produce a strict argument comparing two random processes?

[5] If you dismiss such scrutiny as mathematical pedantry, have a look at what happens for $s = a - 1$. (3.1) holds for all $s$, and also $e_s = (s-a)(b-s)$ satisfies it for all $s \in \mathbb{Z}$. Therefore $e_{a-1} = -(b-a+1)$, or what?

### 3.2.2 Coupling

**Theorem 3.5** *If $F$ is a satisfiable $(\leq 2)$-CNF formula over $V$, $n := |V|$, then* rf() *finds a satisfying assignment in an expected number of at most $n^2$ iterations of its while-loop.*

*Proof.* We employ a sequence $(S_i)_{i\in\mathbb{N}}$ of mutually independent random variables with $\Pr(S_i = -1) = \Pr(S_i = +1) = \frac{1}{2}$ for all $i \in \mathbb{N}$.

On the one hand, let $\hat{R}_0 \in \mathbb{Z}_{\leq n}$ and for $t \in \mathbb{N}$

$$\hat{R}_t := \begin{cases} n-1 & \text{if } \hat{R}_{t-1} = n, \text{ and} \\ \hat{R}_{t-1} + S_t & \text{otherwise.} \end{cases}$$

Then $(\hat{R}_t)_{t\in\mathbb{N}_0}$ is a symmetric random walk reflecting at $n$.

On the other hand, we let the random variables $S_t$ from the above sequence steer the random choice of the literal $u$ from $C_t$, where $C_t$ is the clause chosen for $C$ in the $t^{\text{th}}$ iteration of the while-loop of procedure rf(). We first fix some assignment $\alpha^*$ satisfying $F$ and some ordering of the literals in $V \cup \overline{V}$.

If $C_t$ is a 1-clause, $u$ is determined. If $C_t$ is a 2-clause and $S_t = -1$, we let $u$ be the smaller of the literals in $C_t$ for which $\alpha^*$ and the current assignment $\alpha$ disagree (there may be one or two). If $S_t = +1$, we let $u$ be the other literal in $C_t$. Note that indeed $u$ is chosen u.a.r. from $C_t$, since $S_t$ is u.a.r. from $\{-1, +1\}$.

Fix a satisfying assignment $\alpha^* \in \{0,1\}^V$ and let $T$ be (the random variable for) the number of iterations of rf(). Let $D_0$ denote the Hamming distance between $\alpha^*$ and the initial choice of $\alpha$, and for $1 \leq t \leq T$ let $D_t$ denote the Hamming distance between $\alpha^*$ and $\alpha$ after the $t^{\text{th}}$ iteration of the while-loop. We have $D_0 \in \{0..n\}$ and

$$D_t = \begin{cases} D_{t-1} - 1 & \text{if } \forall v \in C_t : \alpha(v) \neq \alpha^*(v), \text{ and} \\ D_{t-1} + S_t & \text{otherwise.} \end{cases}$$

It follows by induction that if[6] $\hat{R}_0 = D_0$, then $\hat{R}_t \geq D_t$ for all $0 \leq t \leq T$ and thus $T \leq \min\{t \in \mathbb{N}_0 \mid \hat{R}_t = 0\}$. (It is essential for this proof to

[6]We really need equality here, $\hat{R}_0 \geq D_0$ is not sufficient to make the conclusion. Consider $\hat{R}_0 = n$ and $D_0 = n - 1$.

observe that if $\hat{R}_0 = D_0$, then $\hat{R}_t - D_t$ is always even, and thus $\hat{R}_t = n$ and $D_t = n - 1$ cannot occur.) With the help of Lemma 3.4 we conclude $\mathrm{E}(T) \leq \max_{s=0}^{n} s(2n - s) = n^2$. ◻

The method used in this proof is called *coupling*. We have generated a sequence $((\hat{R}_t, D_t))_{t \in \mathbb{N}_0}$ of 2-dimensional random variables, both components based on the same random experiment $(S_i)_{i \in \mathbb{N}}$. The projection to the first component is the well understood symmetric walk reflecting at $n$, and the second component represents the process we want to analyze. And we have used the carefully implemented correlation between the two for the analysis.

NOTE 3.1 Can we use the process described in this section to decide satisfiability of an $(\leq 2)$-CNF formula? The answer is yes, if we are willing to accept some *error probability*.

We let $F$ be $(\leq 2)$-CNF formula. We drop the precondition in rf() that $F$ is satisfiable. Let $T$ be the random variable for the number of iterations of the while-loop in rf()—it is $\infty$, if $F$ is not satisfiable. If $F$ is satisfiable, we know that $\mathrm{E}(T) \leq n^2$, and thus, by Markov's Inequality $\Pr\left(T \geq 3n^2\right) \leq \frac{1}{3}$ (see Appendix A.4). In other words, the probability that we go through more than $3n^2$ iterations while the formula is satisfiable is at most $\frac{1}{3}$. Or, if we claim unsatisfiability after not succeeding within $3n^2$ steps, then we fail to be right with probability at most $\frac{1}{3}$.

$\frac{1}{3}$ may not be sufficiently assuring, so we allow more iterations: $\lambda n^2$ iterations let the error probability goes down to $\frac{1}{\lambda}$. However, we may start anew for $r$ rounds, in each round waiting for $3n^2$ iterations. The probability of never finding a satisfying assignment, even though there is one, is at most $(\frac{1}{3})^r$, while having gone through $3r\,n^2$ iterations of the while-loop altogether. So for the expense of $\lambda\, n^2$ iterations we get an error probability of at most $\left(\frac{1}{3}\right)^{\lfloor \lambda/3 \rfloor}$. For $\lambda = 9$ this gives $\frac{1}{27}$ compared to $\frac{1}{9}$. And for $\lambda = 40$, this is already less than $\frac{1}{1'000'000}$.

We have described here a general principle (see also [Alt *et al.*, 1996] or Exercise 3.13). Clearly the concrete problem at hand allows for several alternatives.

An algorithm as we have developed it here is called *Monte Carlo algorithm*. It is a randomized algorithm whose output may be erroneous with some error probability. In our case the error is *one-sided*: Answer “satisfiable” is guaranteed to be correct, while “unsatisfiable” has a small error probability. We will see another example in Section 3.4.2 shortly.

NOTE 3.2 An issue that may be troubling you in the proof of Theorem 3.5 is that we refer to a satisfying assignment $\alpha^*$ even though such an assignment is not available to rf()—after all, it is the goal of the algorithm to find some satisfying assignment. However, $\alpha^*$ is used only by us as external observers of the process, and all we need is that a satisfying assignment exists. It is the great advantage of randomized algorithms that they can do things like "moving towards an unknown object." It also becomes clear that derandomization of the procedure along the ideas from Section 2.1 is impossible.

Perhaps even more puzzling is how we use $\alpha^*$ for steering the random choices of literals in clauses. Here it is important to realize that for an outside observer the choice is indeed random in the required sense, as long as the sequence $(S_i)_{i\in\mathbb{N}}$ is a sequence of independently and uniformly chosen bits.

NOTE 3.3 What if we start with an assignment $\alpha$ u.a.r. in $\{0,1\}^V$? Then, for a fixed satisfying assignment $\alpha^*$,

$$\Pr(d_H(\alpha^*,\alpha)=s)=\tbinom{n}{s}2^{-n} \quad \text{for all } s\in\{0..n\}.$$

Hence, the expected time until a satisfying assignment is reached is bounded by $\sum_{s=0}^{n} s(2n-s)\binom{n}{s}2^{-n}$. We derive

$$\begin{aligned}
\sum_{s=0}^{n} s(2n-s)\binom{n}{s} &= \sum_{s=1}^{n}\frac{s(2n-s)n!}{s!(n-s)!}\\
&= \sum_{s=1}^{n}\frac{(n+n-s)n!}{(s-1)!(n-s)!}\\
&= \sum_{s=1}^{n}\frac{n^2(n-1)!}{(s-1)!(n-s)!}+\sum_{s=1}^{n-1}\frac{n(n-1)(n-2)!}{(s-1)!(n-s-1)!}\\
&= n^2\sum_{s=1}^{n}\binom{n-1}{s-1}+n(n-1)\sum_{s=1}^{n-1}\binom{n-2}{s-1}\\
&= n^2\,2^{n-1}+n(n-1)\,2^{n-2}\\
&= \frac{3n^2-n}{4}\,2^n\ ,
\end{aligned}$$

and a bound of $\frac{3n^2-n}{4}$ is established—admittedly, a modest improvement by roughly a factor $\frac{3}{4}$.

For an estimate there is an easier way as follows. Observe that we want to analyze $\mathrm{E}(S(2n-S))$ where $S\in\{0..n\}$ is a random variable with $\Pr(S=s)=$

$\binom{n}{s}2^{-n}$. Now

$$\mathrm{E}(S(2n-S)) = 2n\,\mathrm{E}(S) - \mathrm{E}\Big(S^2\Big) \le 2n\,\mathrm{E}(S) - \mathrm{E}(S)^2 = 2n\,\frac{n}{2} - \left(\frac{n}{2}\right)^2 = \frac{3}{4}\,n^2$$

Here we have used $\mathrm{E}(S) = \frac{n}{2}$ which follows from its distribution, and $\mathrm{E}(S^2) \ge \mathrm{E}(S)^2$, which follows from Jensen's Inequality.

**Exercise 3.8** *Complete the proof of Lemma 3.3.*

**Exercise 3.9** *Show that for a symmetric random walk on $\mathbb{Z}$, return to the starting state is guaranteed with probability 1, but it takes an infinite number of steps on the average to do so, i.e.*

$$\begin{aligned}\Pr(\exists t \in \mathbb{N} : W_t = 0 \,|\, W_0 = 0) &= 1 \quad \textit{and}\\ \mathrm{E}(\min\{t \in \mathbb{N} \mid W_t = 0\} \,|\, W_0 = 0) &= \infty\ .\end{aligned}$$

**Exercise 3.10** *For $t \in \mathbb{N}_0$, determine $\mathrm{E}((W_t)^2 \,|\, W_0 = 0)$ for a symmetric random walk $(W_t)_{t\in\mathbb{N}_0}$ on $\mathbb{Z}$.*
HINT: For a sequence $(S_i)_{i\in\mathbb{N}}$ of mutually independent random variables with $\Pr(S_i = -1) = \Pr(S_i = +1) = \frac{1}{2}$ for all $i \in \mathbb{N}$, show that $(\sum_{i=1}^{t} S_i)_{t\in\mathbb{N}_0}$ is a symmetric random walk on $\mathbb{Z}$.

**Exercise 3.11** *Analyze the expected number of flips procedure* rf() *takes for*

$$\{\{\overline{x_1}, x_2\}, \{\overline{x_2}, x_3\}, \ldots, \{\overline{x_{n-1}}, x_n\}, \{\overline{x_n}, x_1\}\}$$

*until a satisfying assignment is reached, (i) when started with an assignment with $\ell$ 1's, $0 \le \ell \le n$, and (ii) when started with an assignment u.a.r. in $\{0,1\}^n$.*

**Exercise 3.12** *Prove that the bound in Theorem 3.5 is tight for all $n \in \mathbb{N}$. That is, you are required to describe for every $n \in \mathbb{N}$ a satisfiable $(\le 2)$-CNF formula $F_n$ over a set $V_n$ of $n$ variables, an initial assignment $\alpha_n$ and a strategy for choosing the clauses in the while-loop, so that the expected number of iterations of the while-loop implied by a call* $\mathrm{rf}(F_n, V_n)$ *is exactly $n^2$.*

**Exercise 3.13** Optimal Patience

*In Note 3.1, you may have wondered why we decided to restart after* $3n^2$ *rounds, rather than* $2n^2$ *or* $10n^2$ *rounds. Good point, so what would you suggest instead?*

*Here is the set-up. We have a randomized algorithm that finds a desired object—provided it exists—in* $T$ *steps,* $T$ *a random variable. We know an upper bound* $b$ *on the expectation of* $T$ *and we have to decide upon some* $\delta$*, such that restarting a new incarnation of the algorithm after* $\delta b$ *unsuccessful steps, provides us with a small error probability for the claim of non-existence of the object, in case we never find one within* $r$ *rounds. The cost is* $r\delta b$*, and the goal is to get the smallest error probability for the given cost.*

**Exercise 3.14** Dwelling at $n$

*For* $n \in \mathbb{N}_0$*, a* symmetric random walk dwelling on $n$ *is a homogeneous Markov chain* $(D_t)_{t\in\mathbb{N}_0}$ *with state space* $\{i \in \mathbb{Z} \mid i \leq n\}$ *and for all* $t \in \mathbb{N}$

$$\Pr(D_t = i+1 \,|\, D_{t-1} = i) = \Pr(D_t = i-1 \,|\, D_{t-1} = i) \;=\; \frac{1}{2} \quad \textit{if } i < n, \textit{ and}$$
$$\Pr(D_t = n-1 \,|\, D_{t-1} = n) = \Pr(D_t = n \,|\, D_{t-1} = n) \;=\; \frac{1}{2}\,.$$

*For* $s \in \mathbb{N}_0$*,* $s \leq n$*, determine* $\mathrm{E}(\min\{t \mid D_t = 0\} \,|\, D_0 = s)$*, the expected number of steps up to the first encounter of* $0$ *provided* $D_0 = s$*.*

Hint: Consider $\left(n - \left|W_t + \frac{1}{2}\right| + \frac{1}{2}\right)_{t\in\mathbb{N}_0}$, $(W_t)_{t\in\mathbb{N}_0}$ a symmetric random walk on $\mathbb{Z}$; but, of course, there are other (more direct) ways.

## 3.3 A Linear Time Algorithm

In this section let $F$ be an $(\leq 2)$-CNF formula, $\Box \notin F$, over variables $V$.

Recall that

$$(u \vee v) \;\equiv\; \overline{u} \to v \;\equiv\; \overline{v} \to u\;,$$

and we also have

$$(u) \;\equiv\; \overline{u} \to u\;.$$

This "implication" information of the clauses in $F$ will be collected in a directed graph (digraph) $D = D_F$ with

- vertex set $V(D) := V \cup \overline{V}$, and
- edge set $E(D) := \{(u, v) \mid \{\overline{u}, v\} \in F\}$.

Note that each clause $\{u, v\}$ with $u \neq v$ gives rise to two edges, while a unit clause $\{u\}$ leads to one edge $(\overline{u}, u)$. $D$ has $2|V|$ vertices and at most $2|F|$ directed edges.

With slight (but as we have seen, consistent) abuse of notation, we write $u \to v$ if $(u, v) \in E(D)$, and we write $u \rightsquigarrow v$ if there is a directed path

$$u = w_0 \to w_1 \to \ldots \to w_\ell = v, \quad \ell \in \mathbb{N}_0,$$

from $u$ to $v$ in $D$.

**Observation 3.6** $u, v \in V \cup \overline{V}$.

**(i)** *$u \rightsquigarrow v$ implies that every satisfying assignment setting $u$ to 1 has to set $v$ to 1 as well.*

**(ii)** *If $u \rightsquigarrow v \rightsquigarrow u$, then every satisfying assignment sets $u$ and $v$ to the same value.*

**(iii)** *If $u \rightsquigarrow \overline{u}$, then every satisfying assignment sets $u$ to 0.*

**(iv)** *$u \rightsquigarrow v$ iff $\overline{v} \rightsquigarrow \overline{u}$.*

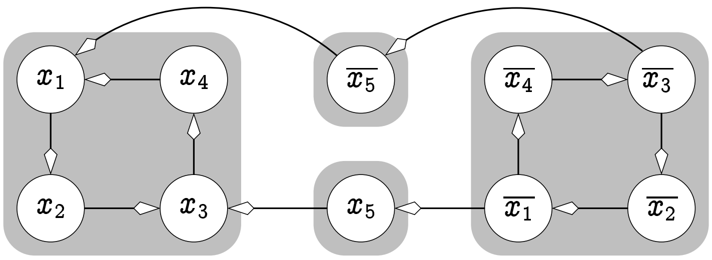



*Figure 3.3: The directed graph $D$ for the satisfiable 2-CNF formula $\{\{\overline{x_1}, x_2\}, \{\overline{x_2}, x_3\}, \{\overline{x_3}, x_4\}, \{\overline{x_4}, x_1\}, \{\overline{x_5}, x_3\}, \{x_5, x_1\}\}$ with its strongly connected components. Note, e.g., $[\overline{x_2}] \leq [x_5]$ and $[\overline{x_1}] \leq [\overline{x_5}]$.*

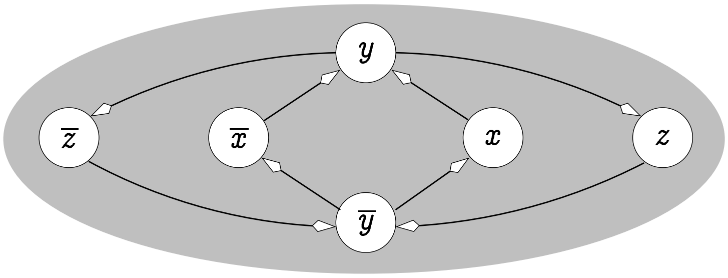



*Figure 3.4: The directed graph $D$ for the unsatisfiable 2-CNF formula $\{\{x,y\},\{\overline{x},y\},\{z,\overline{y}\},\{\overline{z},\overline{y}\}\}$. The whole graph forms one strongly connected component.*

**Lemma 3.7** *$F$ is not satisfiable iff there is a variable $x \in V$ such that $x \rightsquigarrow \overline{x} \rightsquigarrow x$.*

*Proof.* Since $x \rightsquigarrow \overline{x}$ forces $x$ to be set to 0 in a satisfying assignment, and $\overline{x} \rightsquigarrow x$ forces $\overline{x}$ to be 0, clearly $x \rightsquigarrow \overline{x} \rightsquigarrow x$ excludes the existence of a satisfying assignment. We are left to produce a satisfying assignment otherwise.

Consider the equivalence relation

$$u \sim v \quad :\Leftrightarrow \quad u \rightsquigarrow v \rightsquigarrow u$$

on $V \cup \overline{V}$. The equivalence classes $[u]$, $u \in V \cup \overline{V}$, of $\sim$ are the strongly connected components of the digraph $D$, i.e. the maximal (w.r.t. set inclusion) subsets of mutually reachable vertices.

Recall that our presumption means that no equivalence class contains both $x$ and $\overline{x}$ for an $x \in V$. Hence, $\overline{[u]} := \{\overline{v} \mid v \in [u]\}$ is disjoint from $[u]$. Note that $\overline{[u]} = [\overline{u}]$.

Define a partial order on the equivalence classes by

$$[u] \leq [v] \quad :\Leftrightarrow \quad u \rightsquigarrow v\ .$$

If $[u]$ is a minimal element with respect to that partial order, then $[\overline{u}]$ is maximal, since

$$\overline{u} \rightsquigarrow v \quad \Rightarrow \quad \overline{v} \rightsquigarrow u$$

and so $[\overline{v}] = [u]$ (because of minimality) and therefore $[\overline{u}] = [v]$. We set now $u' \mapsto 0$ and $\overline{u'} \mapsto 1$ for all $u' \in [u]$. Then, clearly every clause containing $\overline{u'}$, where $u' \in [u]$, is satisfied. A clause $\{u'\}$ cannot exist, since then $\overline{u'} \leadsto u'$ contradicts the minimality of $[u]$. For a clause $\{v, u'\}$, $v \neq u'$, we have $\overline{v} \to u'$, hence $\overline{v} \in [u]$ (because of minimality of $[u]$) and thus $v$ is set to 1 and also $\{v, u'\}$ is satisfied. We can therefore remove all clauses containing $u'$ or $\overline{u'}$ for $u' \in [u]$. The remaining clauses are satisfied by an inductive argument. ◻

It follows that for deciding the satisfiability of $F$ we have to construct a digraph with $2n$ vertices and at most $2m$ edges, $n := |V|$, $m := |F|$, then compute the strongly connected components, and decide whether one of them contains both $x$ and $\overline{x}$ for a variable $x$. All of this can be done in time $O(n+m)$ by standard graph algorithms, see e.g. [Cormen *et al.*, 1990, Section 23.5]. Finally, if there is a satisfying assignment, the procedure indicated in the proof above can produce a satisfying assignment in time $O(n+m)$ as well.

**Theorem 3.8 ([Aspvall et al., 1979])** *The satisfiability of a $(\leq 2)$-CNF formula over $n$ variables with $m$ clauses can be decided in time $O(n+m)$. Within the same time bound a satisfying assignment can be found, provided it exists.*

NOTE **3.4** In fact, [Aspvall *et al.*, 1979] show that the satisfiability of *quantified* $(\leq 2)$-CNF formulas can be decided in linear time.

NOTE **3.5** Here is an alternative *Proof of Lemma 3.7 via resolution.* Consider the resolvent $\{u, v\}$ of $\{x, u\}$ and $\{\overline{x}, v\}$. This clause introduces the edges $\overline{u} \to v$ and $\overline{v} \to u$ to the graph $D$. But since $\overline{v} \to \overline{x} \to u$ and $\overline{u} \to x \to v$ were present in the original graph, the new edges do not alter the relation $\leadsto$. In fact, if $F'$ is the CNF formula obtained after adding resolvents as long as possible, in the resulting graph $D_{F'}$, we have $u \to v$ iff $u \leadsto v$ and $u \neq v$ (verify!). This clearly proves now the claim, since $F$ is unsatisfiable iff $F' \supseteq \{\{x\}, \{\overline{x}\}\}$ for some $x \in V$, which is equivalent to $x \to \overline{x} \to x$ in the digraph of $F'$, which—in turn—is equivalent to $x \leadsto \overline{x} \leadsto x$ in the graph of $F$. ◻

While somewhat more concise, this proof has the drawback, that it does not tell us how to obtain a satisfying assignment efficiently.

**Exercise 3.15** Dependency Bounds for 2-CNF
*Improve on Theorem 2.3 for $k = 2$. That is, find the largest $b \in \mathbb{N}$ such that the following is true: Every 2-CNF formula where every clause depends on at most $b$ other clauses is satisfiable.*

## 3.4 Examples

### 3.4.1 Solving 3-SAT in less than $2^n$ Steps

Consider an $(\leq 3)$-CNF formula $F$ over $n$ variables. A subset $G$ of clauses in $F$ is called *independent*, if any two clauses in $G$ are strictly disjoint.

Consider now a maximal set $G$ of independent 3-clauses in $F$, i.e. any other 3-clause in $F$ shares a variable with at least one of the clauses in $G$.

**Observation 3.9** *If $G$ is a maximal set of independent 3-clauses in $F$, then*

- $|G| \leq \frac{n}{3}$,
- *for any assignment $\alpha \in \{0,1\}^{\mathrm{vbl}(G)}$, $F^{[\alpha]}$ is a $(\leq 2)$-CNF formula, and*
- *the number of assignments in $\{0,1\}^{\mathrm{vbl}(G)}$ satisfying $G$ is $7^{|G|} \leq 7^{n/3}$.*

In order to decide satisfiability of $F$, we simply go through all assignments $\alpha \in \{0,1\}^{\mathrm{vbl}(G)}$ satisfying $G$, and check satisfiability of $(\leq 2)$-CNF formula $F^{[\alpha]}$.

**Theorem 3.10** *Satisfiability of a $(\leq 3)$-CNF formula can be decided in time $O(\sqrt[3]{7}^{\,n}\,\mathrm{poly}(n)) = O(1.913^n)$.*

This bound will be significantly improved upon. In fact, your solution to Exercise 1.20 may have already suggested a better bound even for counting the number of satisfying assignments in a $(\leq 3)$-CNF formula. Nevertheless, we will come back to the basic idea used here.

**Exercise 3.16** 4-SAT, First Bound
*Derive a bound of $O(c^n \text{poly}(n))$, for some $c < 2$, for deciding satisfiability of a $(\leq 4)$-CNF formula over $n$ variables.*

### 3.4.2 Coloring Graphs

For $k \in \mathbb{N}$, a *proper $k$-coloring* of a graph $G = (V, E)$ is a mapping $\chi : V \longrightarrow \{1..k\}$ such that $\chi(u) \neq \chi(v)$ for all edges $\{u, v\} \in E$. The problem of deciding whether a graph has a proper $k$-coloring is NP-complete for $k \geq 3$.

Here we concentrate on

PROPER 3-COLORING

Input: *A graph $G$.*
Output: *'Yes', if $G$ has a proper 3-coloring. And 'No', otherwise.*

If $n$ is the number of vertices of $G$, a trivial algorithm checks all $3^n$ colorings $\chi : V \to \{1, 2, 3\}$ and tests each one, whether it is proper. This gives a complexity of[7] $O(3^n \text{poly}(n))$. We will describe a simple Monte Carlo algorithm that improves on that bound.

It is easily seen that deciding the existence of a proper 2-coloring of a graph $G$ can be done in linear time. We consider the following generalization: Let $\gamma$ be a mapping that assigns to every vertex $v$ in $V$ a set $\gamma(v)$ of two colors. We want to find a proper coloring that assigns to every vertex $v$ one of the two colors in $\gamma(v)$ – we call such a coloring a *proper list coloring of $G$ consistent with $\gamma$*. This problem is equivalent to the satisfiability of the following 2-CNF formula over the variables $\{x_{v,i} \mid v \in V,\, i \in \gamma(v)\}$

$$\bigwedge_{v,i,j:\ v \in V,\, \{i,j\} = \gamma(v)} (x_{v,i} \vee x_{v,j}) \wedge \bigwedge_{v,w,i:\ \{v,w\} \in E,\, i \in \gamma(v) \cap \gamma(w)} (\overline{x_{v,i}} \vee \overline{x_{w,i}})$$

and thus can be solved in time $O(|V| + |E|)$.

We employ this procedure for 3-coloring. Suppose $G$ does indeed have a proper 3-coloring $\chi$, and we choose for each vertex $v$ a $\gamma(v)$ u.a.r. in

[7] This can be immediately improved to $O(2^n \text{poly}(n))$ by a backtracking algorithm.

$\binom{\{1,2,3\}}{2}$. Then, for all $v$, $\Pr(\chi(v) \in \gamma(v)) = \frac{2}{3}$ and, hence,

$$\Pr(\forall v \in V : \chi(v) \in \gamma(v)) = \left(\frac{2}{3}\right)^n .$$

Consequently, given that there exists a proper 3-coloring of $G$,

$$\Pr(\exists \text{ a proper coloring consistent with } \gamma) \geq \left(\frac{2}{3}\right)^n ,$$

and, if we repeatedly try random mappings $\gamma$, then the expected number of attempts until we find a proper 3-coloring in this way is at most $\left(\frac{3}{2}\right)^n$.

Note that, provided a proper 3-coloring exists, we fail to find one despite of $r$ attempts with probability at most

$$\left(1 - \left(\frac{2}{3}\right)^n\right)^r < \mathrm{e}^{-(2/3)^n r} = \mathrm{e}^{-\lambda} \quad \text{for } r = \lambda \left(\tfrac{3}{2}\right)^n.$$

**Theorem 3.11** *(1) There is a randomized algorithm that finds a proper* 3*-coloring of an* $n$*-vertex graph in expected time* $O(\left(\frac{3}{2}\right)^n \mathrm{poly}(n))$*, provided such a coloring exists.*

*(2) For every positive real number* $\lambda$ *(possibly dependent on* $n$*), there is a randomized algorithm that delivers a proper* 3*-coloring of an* $n$*-vertex graph or claims that no such coloring exists in time*

$$O(\lambda \left(\frac{3}{2}\right)^n \mathrm{poly}(n)) .$$

*In the latter case, the probability that a proper* 3*-coloring exists is at most* $\mathrm{e}^{-\lambda}$.

This algorithm is of the same type as the one from Note 3.1—a Monte Carlo algorithm: The result claimed by the algorithm may fail to be correct with some (small) error probability. In our case the error is one-sided—an error can occur only if the existence of a proper 3-coloring is denied.

The ideas for this section can be found in [Beigel, Eppstein, 1995]. Theorem 3.11 is not the end of the story for 3-coloring. Improvements follow.

**Exercise 3.17** Deterministic 3-coloring

*Design a deterministic algorithm that decides the existence of a proper* 3*-coloring of an* $n$*-vertex graph in time* $O(c^n \mathrm{poly}(n))$ *for some* $c < 2$.

**Exercise 3.18** Save Variables

*In the above description we have shown that the problem of proper list coloring of an $n$-vertex graph consistent with 2-element lists can be solved via a 2-CNF formula over $2n$ variables. Can you find a 2-CNF formula over $n$ (instead of $2n$) variables to serve the purpose (perhaps even with the extra benefit that the number of satisfying assignments for this formula is exactly the number of consistent colorings for the graph)?*

**Exercise 3.19** Covering all 3-Colorings

*Let $V$ be a finite set, $|V| = n \in \mathbb{N}$. Now consider a set $\Gamma$ of mappings $V \to \binom{\{1,2,3\}}{2}$. We call $\Gamma$* exhaustive, *if for every mapping $\chi : V \to \{1, 2, 3\}$ there is a mapping $\gamma \in \Gamma$ such that $\chi(v) \in \gamma(v)$ for all $v \in V$ (we say that every $\chi \in \{1, 2, 3\}^V$ is* covered *by some $\gamma \in \Gamma$).*

*Show that there is an exhaustive set of mappings of size at most $\left\lceil (\ln 3) \cdot n \cdot \left(\frac{3}{2}\right)^n \right\rceil$.*

HINT: Take $\Gamma$ as a random set obtained by drawing $\lceil (\ln 3) \cdot n \cdot (\frac{3}{2})^n \rceil$ times an element u.a.r. from $\binom{\{1,2,3\}}{2}^V$ (with replacement). And consider the expected number of mappings in $\{1, 2, 3\}^V$ not covered by such a $\Gamma$.

**Exercise 3.20** No Monochromatic Triangles

*Suppose we are given a graph $G = (V, E)$ with a proper 3-coloring $\chi \in \{1, 2, 3\}^V$, which we don't know, though. This implies that there is a 2-coloring of the vertices so that no triangle is monochromatic (i.e. if three vertices are pairwise adjacent, then they are not all assigned the same color); for example, take $v \mapsto \chi(v)$ if $\chi(v) \in \{1, 2\}$, and $v \mapsto 1$, otherwise.*

*Design a randomized algorithm that finds such a 2-coloring without monochromatic triangles in expected polynomial time, provided a proper 3-coloring exists (but is not known to the algorithm).*

HINT: Do the obvious: Start with an arbitrary 2-coloring in $\{1, 2\}^V$; $v \mapsto 1$, say. As long as there is a monochromatic triangle, switch the color of a random vertex u.a.r. among the three in the triangle. And analyze with the help of the proper 3-coloring, which we were promised to exist.

# Chapter 4

# SAT and the Class $\mathcal{NP}$

*"A good proof is easy to verify—and*
*it is easy to falsify, in case it is wrong."*

This[1] is one of the themes of this chapter, although in the end we aim at demonstrating that

SAT

Input: *A boolean formula $f$ in propositional logic.*
Output: *'Yes', if $f$ is satisfiable. And 'No', otherwise.*

is $a$[2] most difficult problem in a whole family of problems (the class $\mathcal{NP}$), in the sense that all problems in this family allow polynomial time solutions

[1]The statement may sound a bit strange, so let me illustrate this by an everyday example. By the end of each month we get account statements, telling us the end of month balance. In order to substantiate this number a bit, we also get the list of account movements. So this is a 'proof' we can check.

What happens though if all movements are plausible but the final balance is wrong? You call your customer consultant and complain. You'll hear: "You typed the wrong numbers into your calculator!" or similar. One way out is to ask for a better proof, where after *each* movement the resulting balance is displayed. Then a mistake can be pointed out locally.

As it goes, I realized that the account statements I get in Switzerland do implement this better type of proof, while this used not to be the case in Germany or Austria. So *"Swiss banks supply better proofs"*, in case it needed any proof that they are better!

[2]Talking about '$a$' most difficult problem sounds perhaps a bit awkward, but that's because everyday language (at least for those languages I know) is not prepared for an extremum to occur more than once.

if and only if SAT does.[3] The key concept is that of a (polynomial time) *reduction*, which comes with a number of interesting insights on the side.

A problem with outputs 'Yes' or 'No' is called a *decision problem.* Given any problem, a possible input to the problem is called an *instance* of the problem. In a decision problem, the instances mapped to 'Yes' are called *Yes-instances.*

## 4.1 Verifying and Falsifying

We start with the following recently encountered combinatorial problem, and investigate how its time complexity relates to that of SAT.

PROPER COLORING
Input: $k \in \mathbb{N}_0$ *and a graph* $G = (V, E)$.
Output: *'Yes', if* $G$ *has a proper* $k$*-coloring, i.e. a mapping* $\chi \in \{1..k\}^V$ *such that* $\chi(u) \neq \chi(v)$ *for all* $\{u, v\} \in E$. *And 'No', otherwise.*

**Lemma 4.1** *There is a polynomial time algorithm that, given an instance* $(k, G)$ *of* PROPER COLORING, *produces a formula* $\mathsf{PropCol}(k, G)$ *such that* $G$ *has a proper* $k$*-coloring iff* $\mathsf{PropCol}(k, G)$ *is satisfiable.*

*Proof.* Let $\ell \in \mathbb{N}_0$. Given a sequence $\mathrm{x} = (x_i)_{i=1}^{\ell} = (x_1, x_2, \ldots, x_\ell)$ of $\ell$ boolean variables, we consider

$$\mathsf{OneIsOne}_\ell(\mathrm{x}) := \left(\bigvee_{i=1}^{\ell} x_i\right) \;\wedge \bigwedge_{1 \le i < j \le \ell} \neg(x_i \wedge x_j) \ . \tag{4.1}$$

An assignment satisfies this formula iff it is total and it assigns 1 to exactly one variable in $\{x_i\}_{i=1}^{\ell}$. For a graph $G = (V, E)$ and $k \in \mathbb{N}_0$, $k \le |V|$, let

$$\mathsf{PropCol}(k, G) := \bigwedge_{v \in V} \mathsf{OneIsOne}_k((x_{v,i})_{i=1}^{k}) \;\wedge \bigwedge_{\{u,v\} \in E,\ i \in \{1..k\}} \neg(x_{u,i} \wedge x_{v,i}) \ .$$

[3] This will serve as a perfect excuse for the poor (exponential) time bounds we will derive for the problem.

This formula over $k \cdot |V|$ variables has size polynomial in $|V|$ and it can be constructed in polynomial time.

If $\chi$ is a proper $k$-coloring of $G$, then the assignment $x_{v,i} \mapsto [\chi(v) = i]$ yields a satisfying assignment for the formula. And if $\alpha$ is a satisfying assignment, then for every $v \in V$ there is a unique $i \in \{1..k\}$ with $\alpha(x_{v,i}) = 1$; this is guaranteed by the OneIsOne-subexpressions. Assign this color $i$ to $v$, and the resulting coloring is proper, as assured by the remaining subexpressions.

If $k > |V|$, we define $\mathsf{PropCol}(k, G) := \mathtt{true}$. □

Here are first immediate conclusions from the lemma and its proof.[4]

*If* SAT *can be solved in polynomial time*
*then* PROPER COLORING *can be solved in polynomial time.*

Moreover, if $k \leq |V|$, the number of assignments on $\{x_{v,i} \mid v \in V,\ i \in \{1..k\}\}$ satisfying $\mathsf{PropCol}(k, G)$ equals[5] the number of proper $k$-colorings of $G$. Consequently,

*If the number of satisfying assignments of a boolean formula can be counted in polynomial time*
*then the number of proper $k$-colorings of a graph can be counted in polynomial time.*

Similar to the situation for SAT, we can easily prove that a graph has a proper $k$-coloring by supplying such a coloring. To be more precise: If a graph $G$ has a proper $k$-coloring, then there is a certificate (namely a proper $k$-coloring) with the help of which we can verify proper $k$-colorability in polynomial time (we ignore the issue of finding the certificate). The proof of Lemma 4.1 suggests a clean binary format for such a certificate, namely a satisfying assignment $\alpha$ for $\mathsf{PropCol}(k, G)$, i.e., assuming a vertex set

[4] $\mathsf{PropCol}(2, G)$ can be written as an equivalent ($\leq 2$)-CNF formula, and thus proper 2-colorability can be decided in polynomial time due to the results in Chapter 3. This can be derived much easier in a more direct fashion, though.

[5] You may observe that we could have replaced '$\mathsf{OneIsOne}_k((x_{v,i})_{i=1}^k)$' in $\mathsf{PropCol}(k, G)$ by the shorter expression '$\bigvee_{i=1}^k x_{v,i}$' while still yielding the assertion of the lemma. However, we could no longer make this claim about the number of satisfying assignments.

$V := \{1..n\}$, the string $\left((\alpha(x_{j,i}))_{i=1}^{k}\right)_{j=1}^{n}$. Verification is easy: the formula is a conjunction of subexpressions, and we simply check whether all of them are satisfied.

But now let us assume that we are confronted with a wrong certificate, how do we falsify it: we point at a subset of the bits which are in contradiction with one of the subexpressions. This may be either 2 or $k$ bits. If we prefer to remain with a constant number of bits, we can go back to Theorem 1.1, which lets us transfer $f := \mathsf{PropCol}(k, G)$ to a $(\leq 3)$-CNF formula $f'$ such that $f'$ is satisfiable iff $f$ is satisfiable. In fact, in our case it suffices to replace the $\mathsf{OneIsOne}_k((x_{v,i})_{i=1}^{k})$-subexpressions by

$$\begin{aligned}\mathsf{OneIsOne}'_k((x_{v,i})_{i=1}^{k}) \;:=\; & (c_{v,0} \leftrightarrow 0)\\ & \wedge \bigwedge_{i=1}^{k} (c_{v,i} \leftrightarrow (x_{v,i} \vee c_{v,i-1}))\\ & \wedge (c_{v,k} \leftrightarrow 1)\\ & \wedge \bigwedge_{1 \leq i < j \leq k} \neg(x_{v,i} \wedge x_{v,j})\end{aligned}$$

with auxiliary variables $\{c_{v,i}\}_{i=0}^{k}$. (If you think about it, this follows exactly the idea we advocated in Footnote 1 for bank account statements). For the certificate we still have to agree upon some order of the variables involved, e.g.

$$(c_{j,0}, x_{j,1}, c_{j,1}, x_{j,2}, c_{j,2}, \ldots, x_{j,k}, c_{j,k})_{j=1}^{n} \;.$$

The length of the certificate is $(2k+1)n$, i.e. polynomial in the size of the underlying graph, and it can be falsified by exhibiting at most 3 bits. Note that this fact immediately implies that the certificate can be checked in polynomial time, since there are at most $\ell + \binom{\ell}{2} + \binom{\ell}{3}$ subpatterns to check, with $\ell$ the length of the certificate.

PROPER COLORING *has a polynomial size certificate format that can be falsified by exhibiting at most* 3 *bits.*

After having discussed all the wonderful implications of our ability to express PROPER COLORING in terms of SAT, there is still hope that PROPER COLORING could be much easier to solve than SAT. Not so, as the following lemma shows.

**Lemma 4.2** *There is a polynomial time algorithm that, given a 3-CNF formula $F$, produces a graph* 3ColGraph($F$) *such that $F$ is satisfiable iff* 3ColGraph($F$) *has a proper 3-coloring.*

*Proof.* Let $V := \mathrm{vbl}(F)$. For 3ColGraph($F$) we choose a graph $G = (V_G, E_G)$ with vertex set $V_G := \{t, c\} \cup V \cup \overline{V} \cup \bigcup_{C \in F} \{a_{C,1}, a_{C,2}, a_{C,3}, b_{C,1}, b_{C,2}, b_{C,3}\}$. For the edge set we refer to Figure 4.1. All edges indicated there are in $E_G$ and no edges other than indicated exist. Note that $|V_G| = 2 + 2|V| + 6|F|$ and $|E_G| = 1 + 3|V| + 12|F|$. The size of $G$ is polynomial in $|V| + |F|$, and it can be generated from $F$ in polynomial time.

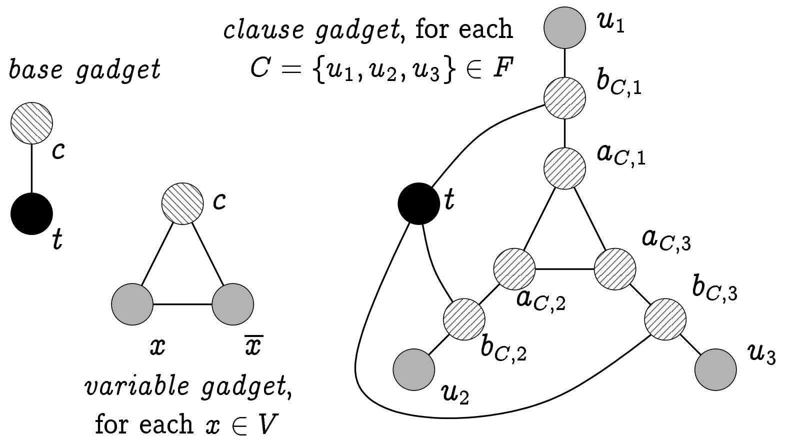



*Figure 4.1: The edges in* 3ColGraph($F$) *for 3-CNF formula $F$.*

First we show that a proper 3-coloring $\chi$ of $G$ implies a satisfying assignment for $F$. We decide to use colors $\{0, 1, 2\}$, and w.l.o.g. we assume that $\chi(t) = 1$ and $\chi(c) = 2$ (because of the base gadget, see Figure 4.1, $t$ and $c$ must receive distinct colors, and by permuting the colors we may assume that these are 1 and 2, resp.). Then $\chi(u) \in \{0, 1\}$ and $\chi(u) \neq \chi(\overline{u})$ for all literals $u$ because of the variable gadgets. So that's a good start: The restriction $\alpha_\chi$ of $\chi$ to $V \cup \overline{V}$ is an assignment, and it remains to verify that every clause $C \in F$ is satisfied. To this end consider the clause gadget of $C = \{u_1, u_2, u_3\}$. One of the vertices $a_{C,i}$, $i \in \{1, 2, 3\}$, must be colored by 2. But then $b_{C,i}$ has to be colored 0, since it is adjacent to $a_{C,i}$ and $t$.

And then $u_i$ must be colored 1, since it is adjacent to $b_{C,i}$ and $c$. Therefore, $\alpha_\chi$ satisfies $C$.

Secondly and finally, we show that a satisfying assignment $\alpha$ for $F$ can be translated to a proper 3-coloring of $G$. In this coloring let $t \mapsto 1$, $c \mapsto 2$, and $u \mapsto \alpha(u)$ for all literals $u$. In this way we have no conflicts with edges in the base or the variable gadgets. We extend further to the clause gadget of $C = \{u_1, u_2, u_3\}$ as follows. Some $u_i$ must be assigned 1 by $\alpha$, w.l.o.g., let it be $u_1$. Then $b_{C,1} \mapsto 0$, $b_{C,2} \mapsto 2$, $b_{C,3} \mapsto 2$, $a_{C,1} \mapsto 2$, $a_{C,2} \mapsto 0$, and $a_{C,3} \mapsto 1$ constitutes an extension to a proper 3-coloring of the clause gadget, no matter which colors from $\{0, 1\}$ are assigned to $u_2$ and $u_3$. ⊡

Suppose now we want to decide satisfiability of a boolean formula $f$ (an instance of SAT). We can first employ Theorem 1.1 to generate a 3-CNF formula $F'$ (an instance of 3-SAT that is satisfiable iff $f$ is). Then we can check satisfiability of $F'$ by checking whether 3ColGraph($F'$) has a proper 3-coloring. In other words, if proper 3-colorability is decidable in polynomial time, then SAT is as well. Note, that via a chain of three steps, we have actually shown that polynomial 3-colorability implies a polynomial solution for PROPER COLORING in general. We summarize:

*Among the problems* SAT, 3-SAT,
PROPER COLORING, *and* PROPER 3-COLORING,
*either all have a polynomial time solution or none.*

**Exercise 4.1** Hamiltonian Path

*Consider the problem*[6]

HAMILTONIAN PATH

Input: *A graph* $G = (V, E)$.

Output: *'Yes', if* $G$ *has a Hamiltonian path, i.e. there is an ordering* $v_1, v_2, \ldots, v_n$, $n := |V|$, *of the vertices in* $V$, *such that* $\{v_i, v_{i+1}\} \in E$ *for all* $i \in \{1..n-1\}$. *And 'No', otherwise.*

[6] William Roger Hamilton, 19th century.

*Show that there is a polynomial time algorithm that, given a graph $G$, produces a boolean formula* $\mathsf{HamPath}(G)$ *such that $G$ has a Hamiltonian path iff* $\mathsf{HamPath}(G)$ *is satisfiable.*

*Check whether your solution guarantees that the number of satisfying assignments of* $\mathsf{HamPath}(G)$ *gives the number of Hamiltonian paths in $G$. (For $n \geq 2$, $v_1, v_2, \ldots, v_n$ and $v_n, v_{n-1}, \ldots, v_1$ are counted as two distinct paths.)*

**Exercise 4.2** Short Path Orientations of Edges
*The format of a certificate is by no means determined by a problem, although the problem statement sometimes suggests a particular one. For the problem* PROPER COLORING *first show that a graph $G$ has a proper $k$-coloring, iff there is an orientation of the edges so that the longest directed path contains at most $k$ vertices. Use this characterization to reprove Lemma 4.1.*

**Exercise 4.3** *Suppose, the number of proper $k$-colorings of a graph $G = (V, E)$ can be counted in polynomial time, provided $k \leq |V|$. Based on this, discuss the issue of counting the number of proper $k$-colorings in case $k > |V|$. Can it be done in polynomial time (under the initial supposition)?*

## 4.2 The Class $\mathcal{NP}$ and Relatives

For a further discussion of the concepts we observed in the previous section we switch to the following framework.

**The Class $\mathcal{P}$.** An *alphabet* $\Sigma$ is a finite set of symbols. $\Sigma^*$ denotes the set of all finite strings of symbols in $\Sigma$ (including the *empty string* $\varepsilon$). Elements $w$ in $\Sigma^*$ are sometimes called *words*, and the *length of a word* $w \in \Sigma^*$ is denoted by $|w|$. For words $u$ and $w$, the concatenation of $u$ followed by $w$ is denoted by $uw$. A *language* is a (possibly infinite) subset of $\Sigma^*$ for some alphabet $\Sigma$.

Let $A$ be an algorithm that maps every $w \in \Sigma^*$ to 'Yes' or 'No'. In the former case we say $A$ *accepts* $w$, in the latter $A$ *rejects* $w$.

Every decision problem can be represented by a language (actually in several ways): We agree on some encoding of the inputs over some alphabet $\Sigma$, and we associate with every problem the language of encodings of all Yes-instances.

For example a graph $G = (\{1..n\}, E)$ can be encoded by the $\{0,1\}$-string $\mathrm{bin}(G) := (([\{i,j\} \in E])_{i=1}^{n})_{j=1}^{n}$ of length $n^2$. An integer $k \in \mathbb{N}_0$ can be encoded by its shortest binary representation $\mathrm{bin}(k)$ (which is of length $\lceil \log_2(k+1) \rceil$; 0 is encoded by the empty string $\varepsilon$).

Now the problem HAMILTONIAN PATH can be represented by the language

$$\{\mathrm{bin}(G) \mid G \text{ has a Hamiltonian path}\} \subseteq \{0,1\}^*$$

PROPER COLORING by

$$\{\mathrm{bin}(k)\$\mathrm{bin}(G) \mid G \text{ has a proper } k\text{-coloring}\} \subseteq \{0,1,\$\}^* \ .$$

Via an appropriate binary encoding of finite alphabets, it suffices to consider languages over a binary alphabet $\{0,1\}$, so we will assume all languages to confine to this for the rest of this section.

$\mathcal{P}$ The class $\mathcal{P}$ (of polynomially recognizable languages) is the set of all languages over $\{0,1\}$ such that there is a polynomial time algorithm $A$ with

$$L = \{w \mid A \text{ accepts } w\} \ .$$

'Polynomial time' means that there is a polynomial $p_A$ such that $A$ terminates on any input $w$ in at most $p_A(|w|)$ steps.

**Polynomial Verifiers (Good Proofs).** A *verifier* for a language $L$ is an algorithm $V$ such that

$$L = \{w \mid \exists c \in \{0,1\}^* : \ V \text{ accepts } \langle w, c \rangle\} \ .$$

If $V$ accepts $\langle w, c \rangle$, then $c$ is called a *certificate* (or proof) for membership of $w$ in $L$. A *falsifier* for a language $L$ is an algorithm $F$ such that

$$\{0,1\}^* \setminus L = \{w \mid \forall c \in \{0,1\}^* : \ F \text{ rejects } \langle w, c \rangle\} \ .$$

Some moments of reflection reveal[7] that every verifier for $L$ is also a falsifier for $L$, and vice versa, but let us accept this notational redundancy for the time being. A verifier (or falsifier) is called a *polynomial verifier* (*falsifier*, resp.), if it runs in time polynomial in $|w|$ on input $\langle w, c\rangle$ (thus it can inspect at most a polynomial number of positions in $c$).

$\mathcal{NP}$ is[8] the class of all languages with a polynomial verifier. Clearly, $\mathcal{P} \subseteq \mathcal{NP}$. It is easy to see that also[9] SAT and PROPER COLORING are in $\mathcal{NP}$, while we do not know of a polynomial time algorithm for these problems.[10] $\mathcal{NP}$

**Polynomial Constant Falsifiers (Even Better Proofs).** Let $k \in \mathbb{N}_0$. A *$k$-falsifier for $L$ is an algorithm $F$* such that

$$\Sigma^* \setminus L = \{w \mid \forall c \in \{0,1\}^* \, \exists S \subseteq \mathbb{N} : \ |S| \leq k \text{ and } F \text{ rejects } \langle w, c_S, S\rangle\}\ ,$$

where $c_S$ is the substring of $c$ of symbols at positions in $S \cap \{1..|c|\}$ (positions exceeding the length of $c$ are simply ignored). Note that we could have equivalently written

$$L = \{w \mid \exists c \in \{0,1\}^* \ \forall S \subseteq \mathbb{N} \text{ with } |S| \leq k : \ F \text{ accepts } \langle w, c_S, S\rangle\}\ .$$

If $F$ accepts $\langle w, c_S, S\rangle$ for all $S$ with $|S| \leq k$, then we call $c$ an *$F$-certificate for membership of $w$ in $L$.* That is, $F$ can reject a certificate by exhibiting

[7] At least under the assumption that we consider algorithms that terminate on all inputs.

[8] $\mathcal{NP}$ stands for 'nondeterministic polynomial time', from an equivalent definition via nondeterministic Turing machines.

[9] Note that we identify the decision problems with their respective languages. However, the encoding is not unique. Still, for our purposes, this is not an issue as long as we use an encoding whose length is polynomial in the most succinct encoding. For example, for numbers a binary encoding will do, or for SAT any encoding whose size is polynomial in the number of variables and operator occurrences fulfills the requirement. If an encoding deviating from this requirement is presumed—like unary encodings of numbers—this calls for explicit mention.

[10] It is by no means clear that the complement of a language in $\mathcal{NP}$ is also in $\mathcal{NP}$—take, e.g., the language of unsatisfiable formulas. Hence, besides $\mathcal{NP}$ also the class co-$\mathcal{NP}$ := $\{\{0,1\}^* \setminus L \mid L \in \mathcal{NP}\}$ is considered.

at most $k$ bits (knowing also their positions in the certificate). $F$ is a *polynomial $k$-falsifier* if there are polynomials $p_F$ and $\hat{p}_F$ such that, on input $\langle w, a, S\rangle$, it terminates in at most $p_F(|w|)$ steps, and it always accepts if $S \not\subseteq \{1..\hat{p}_F(|w|)\}$ (hence the falsifier ignores positions in a certificate that exceed $\hat{p}_F(|w|)$).

$\mathcal{FP}_k$ For $k \in \mathbb{N}_0$, $\mathcal{FP}_k$ is the class of languages with a polynomial $k$-falsifier.

We observe that

$$\mathcal{P} \subseteq \mathcal{FP}_{k-1} \subseteq \mathcal{FP}_k \subseteq \mathcal{NP}$$

for all $k \in \mathbb{N}$. '$\mathcal{FP}_k \subseteq \mathcal{NP}$' follows, since a polynomial $k$-falsifier $F$ can be transformed to a polynomial falsifier $F'$ by letting

$$F' \text{ rejects } \langle w, c\rangle \text{ iff}$$
$$F \text{ rejects } \langle w, c_s, S\rangle \text{ for all } S \subseteq \{1..\hat{p}_F(|w|)\} \text{ with } |S| \le k\ .$$

Since there are only $O(\hat{p}_F(|w|)^k)$ sets $S$ to consider, $F'$ can do its computations in polynomial time based on $F$ being polynomial.

Some considerations for small $k$ are instructive. A 0-falsifier will never be given a chance to have a look at the string $c$, so it is of no help, and the falsifier has to do its computations based on $w$ only. Therefore, if it is polynomial, it can recognize $L$ in polynomial time—we have $\mathcal{FP}_0 = \mathcal{P}$.

Now consider a 1-falsifier $F$ and some input $w$. There is an $F$-certificate iff there is an $m \le \hat{p}_F(|w|)$ such that $F$ accepts $\langle w, \varepsilon, \{i\}\rangle$ for all $i > m$ and $F$ accepts $\langle w, 0, \{i\}\rangle$ or $\langle w, 1, \{i\}\rangle$ for all $i \le m$. $\mathcal{FP}_1 = \mathcal{P}$ follows.

**Theorem 4.3**
(1) $\mathcal{FP}_k = \mathcal{P}$ *for all* $k \le 2$.
(2) $\mathcal{FP}_k = \mathcal{FP}_3$ *for all* $k \ge 3$.
(3) $\mathcal{FP}_3 = \mathcal{P}$ *iff* SAT$\in \mathcal{P}$.

*Proof.* Let $L \in \mathcal{FP}_k$, and let $F$ be a polynomial $k$-falsifier for $L$. We will show that, given a word $w$ and $m \le \hat{p}_F(|w|)$, we can produce in polynomial time—assuming the help of $F$—a ($\le k$)-CNF formula $G_m(w)$ which is satisfiable iff there is an $F$-certificate of length $m$. Then an $F$-certificate exists iff

$$\mathsf{Formula}(w) := \bigvee_{m=0}^{\hat{p}_F(|w|)} G_m(w)$$

is satisfiable. That is, $w \in L$ iff $\mathsf{Formula}(w)$ is satisfiable.

Let us first assume the validity of this claim and see how the assertions of the theorem are implied.

Assertion (1) follows readily: If $k \leq 2$ we can decide satisfiability of $\mathsf{Formula}(w)$ in polynomial time by checking whether one of the $(\leq 2)$-CNF formulas $G_m(w)$, $0 \leq m \leq \hat{p}_F(|w|)$, is satisfiable.

For (2), let us assume $k \geq 4$. We derive a polynomial 3-falsifier $F'$ from $F$: First $F'$ constructs $\mathsf{Formula}(w)$ as above, and then it constructs in polynomial time a $(\leq 3)$-CNF formula $G'$ over some variable set $U := \{x_i\}_{i=1}^{\ell}$ such that $G'$ is satisfiable iff $G$ is satisfiable. Now the $F'$-certificates for $w$ are the strings $(\alpha(x_i))_{i=1}^{\ell}$ for assignments $\alpha$ satisfying $G'$. Since $G'$ is a $(\leq 3)$-CNF formula, such a certificate can be falsified by exhibiting at most 3 bits.

A proof of (3) is now also straightforward. On the one hand we can decide $w \in L$ by checking satisfiability of $\mathsf{Formula}(w)$, so the direction '$\Leftarrow$' is established. On the other hand, for '$\Rightarrow$' we invoke once more Theorem 1.1 and the fact that 3-SAT is in $\mathcal{FP}_3$.

We are left with the actual construction of $G_m(w)$. We employ a variable set $\{x_i\}_{i=1}^{m}$. For all $a \leq b \leq k$, all $\{i_j\}_{j=1}^{b} \subseteq \mathbb{N}$ with

$$1 \leq i_1 < i_2 < \ldots < i_a \leq m < i_{a+1} < i_{a+2} < \ldots < i_b \leq \hat{p}_F(|w|)$$

and all $\gamma : \{i_j\}_{j=1}^{a} \to \{0,1\}$ we test whether $F$ rejects

$$\langle w,\ \gamma(i_1)\gamma(i_2)\ldots\gamma(i_a),\ \{i_j\}_{j=1}^{b}\rangle.$$

If so, we add the clause $\{u_{i_j}\}_{j=1}^{a}$ to $G_m(w)$, where

$$u_{i_j} := \begin{cases} x_{i_j} & \text{if } \gamma(i_j) = 0, \text{ and} \\ \overline{x_{i_j}} & \text{otherwise.} \end{cases}$$

Now it is easy to see that $c = (c_i)_{i=1}^{m} \in \{0,1\}^m$ is a $F$-certificate iff $x_i \mapsto c_i$ is a satisfying assignments for $G_m(w)$. Finally, $G_m(w)$ is a $(\leq k)$-CNF formula over $m \leq \hat{p}_F(|w|)$ variables, and thus its size is polynomial in $|w|$ (since $k$ is a constant), and it can be constructed in polynomial time. ◻

$\leq_{\mathrm{P}}$

**Polynomial Reductions.** A language $L_1$ is *polynomial time reducible* to language $L_2$, in symbols $L_1 \leq_{\mathrm{P}} L_2$, if there is a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that $w \in L_1$ iff $f(w) \in L_2$.

We have seen a number of such polynomial reductions. Lemma 4.1 describes a polynomial reduction from PROPER COLORING to SAT (via the polynomial function PropCol), and Lemma 4.2 gave a reduction from 3-SAT to PROPER COLORING (via 3ColGraph).

Theorem 4.3 provided polynomial time reductions from all languages in $\mathcal{FP}_k$ to SAT (via Formula). We drew conclusions which we now reformulate in this abstract setting.

**Lemma 4.4**
(1) *If $L_1 \leq_{\mathrm{P}} L_2$ and $L_2 \leq_{\mathrm{P}} L_3$ then $L_1 \leq_{\mathrm{P}} L_3$*
(2) *If $L_1 \leq_{\mathrm{P}} L_2$ and $L_2 \in \mathcal{P}$, then $L_1 \in \mathcal{P}$.*
(3) *Let $\mathcal{L}$ be a class of languages with a language $L \in \mathcal{L}$ such that all languages in $\mathcal{L}$ are polynomial time reducible to $L$ (we call such a language* $\mathcal{L}$-complete*). Then $\mathcal{L} \subseteq \mathcal{P}$ iff $L \in \mathcal{P}$.*

$\mathcal{L}$-complete

In fact, the proof of theorem employed a reduction from any problem in $\mathcal{FP}_k$ to SAT and then to 3-SAT. Since this will be the main insight used in the next section, it deserves a lemma.

**Lemma 4.5** SAT *and* 3-SAT *are $\mathcal{FP}_3$-complete (and thus complete in in $\bigcup_{k \in \mathbb{N}_0} \mathcal{FP}_k$, cf. Theorem 4.3(2)).*

It may appear that $\mathcal{FP}_3$ is very much tuned towards 3-SAT and confined to a few combinatorial problems, where a solution (certificate) can be falsified in a constant number of places while $\mathcal{NP}$ allows more elaborate calculations. But ...

**Exercise 4.4** Conflict Free Responsibilities in CNF

*Observe that a CNF formula is satisfiable, if we can declare one literal in each clause 'responsible', so that for no variable $x$ both $x$ and $\overline{x}$ are made responsible. Exploiting this, deduce an alternative certificate format (other than a satisfying assignment) for* 3-SAT.

**Exercise 4.5** Three Symbol Certificates

*$k, l \in \mathbb{N}_0$. Let $\mathcal{FP}_k^{(\ell)}$ be defined in the same way as $\mathcal{FP}_k$ with the exception that a certificate $c$ for a word $w$ is a string over an $\ell$-symbol alphabet.*

*Show that (i) $\mathcal{FP}_k^{(3)} = \mathcal{P}$ for $k = 0, 1$ and (ii) 3-SAT $\in \mathcal{FP}_2^{(3)}$.*

REMARK: This shows that the definition of $\mathcal{FP}_k$ is sensitive to the size of certificate-alphabets.

**Exercise 4.6** Many Symbol Certificates

*Show that $\bigcup_{k,\ell \in \mathbb{N}_0} \mathcal{FP}_k^{(\ell)} = \mathcal{FP}_3$.*

HINT: For $\ell > 2$, show $\mathcal{FP}_k^{(\ell)} \subseteq \mathcal{FP}_{k\lceil \log_2 \ell \rceil}$.

**Exercise 4.7** $\mathcal{FP}_3$-Characterization

*Show that $L \leq_{\mathrm{P}}$ SAT iff $L \in \mathcal{FP}_3$.*

**Exercise 4.8** $\mathcal{FP}_2^{(3)}$-Characterization

*Show that $L \leq_{\mathrm{P}}$ SAT iff $L \in \mathcal{FP}_2^{(3)}$ (that is, given Exercise 4.7, $\mathcal{FP}_3 = \mathcal{FP}_2^{(3)}$).*

**Exercise 4.9** Vertex Cover

*Show that the problem*

VERTEX COVER

> Input: *$k \in \mathbb{N}_0$ and a graph $G = (V, E)$.*
> Output: *'Yes', if $G$ has a vertex cover of size at most $k$, i.e. there is a subset $U$ of $V$ with $|U| \leq k$ such that $e \cap U \neq \emptyset$ for all $e \in E$.*

*is in $\mathcal{FP}_3$.*

## 4.3 SAT is $\mathcal{NP}$-complete

We want to prove that SAT is $\mathcal{NP}$-complete, i.e. that $L \leq_{\mathrm{P}}$ SAT for all $L \in \mathcal{NP}$. We will do so by proving $\mathcal{NP} \subseteq \bigcup_{k \in \mathbb{N}_0} \mathcal{FP}_k$ which suffices due to Lemma 4.5. This means that we have to be able to rewrite certificates that can be verified in polynomial time as certificates that can be falsified by a polynomial $k$-falsifier, for some $k \in \mathbb{N}_0$. The crucial idea is to write out the whole 'protocol' of the (polynomial) computation of the verifier as the new certificate. Due to the 'finite nature' of algorithms, the protocol will be falsifiable by exhibiting a constant number of entries in the protocol.

So far we are lacking a model of algorithms in order to make formal statements about *all polynomial verifiers*. The model we choose is that of a Turing[11] machine. It is generally accepted that anything that is computable by an algorithm (in any reasonable model of computation) is computable by a Turing machine (the Church-Turing thesis). Moreover, anything that is computable by an algorithm in polynomial time is computable by a Turing machine in polynomial time—a working *thesis* that has so far proven to be true, although it rests on shaky grounds in view of quantum computing, for example. This, however, should not lead to the impression that $\mathcal{NP}$-completeness of SAT is a matter of taste, or such like, rather the result ought to be be read with appropriate care: Based on the model of Turing machines for algorithms, this fact is a theorem, and it will stay a valid theorem even if the thesis should ever be refuted.[12]

The following two-page introduction to Turing machines is in principle self-contained, but it may be a bit concise for a first encounter. We will see that the preparation of the set-up is the major part of our project, the final conclusion will turn out to be a relatively simple observation.

### 4.3.1 Turing Machines

A *Turing machine* is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{Yes}}, q_{\text{No}})$, where $Q$, $\Sigma$ and $\Gamma$ are all finite sets and $Q$ is the set of *states*, $\Sigma$ is the *input alphabet*,

[11] Alan Turing, 1912-1954

[12] Let us go on even further with the mumbling: The fact that the whole theory of algorithms and $\mathcal{NP}$-completeness rests on a *thesis* should not be accounted for as a weak point of the theory. Rather, the theory has clearly identified the link to 'reality' (whatever that may be), and has thus supplied all means to *falsify* it. Any exact science that wants to say something about the real world needs such theses (was this another thesis?), and the collapse of a thesis is usually not a disaster for science, rather more often than not these are the most exciting moments in science.

$\Gamma \supseteq \Sigma$ is the *tape alphabet* containing the special *blank symbol* $\sqcup \notin \Sigma$,

$$\delta: \quad (Q \setminus \{q_{\text{Yes}}, q_{\text{No}}\}) \times \Gamma \quad \longrightarrow \quad Q \times (\Gamma \setminus \{\sqcup\}) \times \{-1, +1\}$$

is the *transition function*, $q_{\text{start}} \in Q$ is the *start state*, $q_{\text{Yes}} \in Q$ the *accepting state*, and $q_{\text{No}} \in Q$ the *rejecting state*.

The Turing machine operates on a bi-infinite tape that—at any point of time—contains some current word $w \in (\Gamma \setminus \{\sqcup\})^*$ and $\sqcup$'s otherwise. It has a (read/write) *head* that resides at some cell of the tape within the range of $w$ or at the blank cell to the left or to the right of $w$. It is in some current state in $Q$. Let $q$ be this state and let $a$ be the symbol at the position of the head. If $q \in \{q_{\text{Yes}}, q_{\text{No}}\}$ the machine halts. Otherwise, the next step is prescribed by $\delta(q, a) = (q', b, \sigma)$: Symbol $a$ is replaced by $b$, the head moves one position to the right if $\sigma = +1$, moves one position to the left if $\sigma = -1$, and it changes its state to $q'$. The Turing machine accepts a word $w \in \Sigma^*$, if—when started in state $q_{start}$ with $w$ on the tape and the head positioned at the first symbol of $w$—the machine eventually halts in state $q_{\text{Yes}}$. Similarly, it rejects $w$ if it halts in state $q_{\text{No}}$. Note that there is a third possibility of never terminating.

**Configurations.** For a formal description, we assume the tape to be indexed by $\mathbb{Z}$. A *configuration of $M$* is a tuple $(q, \ddot{w}, \ell, h)$, $q \in Q$, $\ddot{w} := \sqcup w \sqcup$ for $w \in (\Gamma \setminus \{\sqcup\})^*$, $\ell \in \mathbb{Z}$, and $h \in \{\ell..(\ell + |\ddot{w}| - 1)\}$ (with the interpretation that the word $w$ occupies the cells $\ell + 1$ to $\ell + |w|$ on the tape, all other cells hold $\sqcup$, and $M$ is in state $q$ with the head positioned at cell $h$).

If $q \in \{q_{\text{Yes}}, q_{\text{No}}\}$, configuration $(q, \ddot{w}, \ell, h)$ is called a *halting configuration of $M$*. Otherwise, let us write $\ddot{w}$ as $uav$ with $u, v \in \Gamma^*$, $a \in \Gamma$ and $|u| = h - \ell$ (i.e. $a$ is the symbol at the position of the head). Given this and[13] $(q', b, \sigma) := \delta(q, a)$, we define

$$(q, \ddot{w}, \ell, h) \vdash_M \begin{cases} (q', \sqcup bv, \ell - 1, h + \sigma) & \text{if } h = \ell, \\ (q', ub\sqcup, \ell, h + \sigma) & \text{if } h = \ell + |\ddot{w}| - 1, \text{ and} \\ (q', ubv, \ell, h + \sigma) & \text{otherwise.} \end{cases}$$

[13] Note that according to our definition, $b \neq \sqcup$.

We say that the left-hand configuration *directly yields* the right-hand configuration.[14] $\vdash_M^*$ is the transitive and reflexive closure of relation $\vdash_M$ on the set of configurations, and if $C \vdash_M^* C'$, we say that configuration $C$ *yields* configuration $C'$.

**Accept, Reject, Recognize, Decide.** The Turing machine $M$ *accepts* (*rejects*) a word $w$ if $(q_{\text{start}}, \sqcup w \sqcup, -1, 0)$, the *starting configuration with input* $w$, yields a halting configuration in state $q_{\text{Yes}}$ (in state $q_{\text{No}}$, resp.). The set of all words in $\Sigma^*$ accepted by $M$ is the language $L(M)$ *recognized* by $M$. The Turing machine *decides* $L \subseteq \Sigma^*$ if $L = L(M)$ and the machine halts on all inputs in $\Sigma^*$.

If $M$ accepts $w$, there is a sequence of configurations $(C_i)_{i=0}^t$, with $C_0 = (q_{\text{start}}, \sqcup w \sqcup, -1, 0)$, $q_{\text{Yes}}$ being the state of $C_t$, and $C_0 \vdash_M C_1 \vdash_M \ldots \vdash_M C_t$. $t$ is called the *number of steps* (the time, if you like) *it takes for* $M$ *to accept* $w$, and $(C_i)_{i=0}^t$ is called the *protocol of the acceptance of* $w$.

Assuming our initial thesis about polynomial algorithms versus polynomial Turing machines, the class $\mathcal{NP}$ is the set of all languages $L \subseteq \{0,1\}^*$ such that there is a Turing machine $M$ and a polynomial $p_M$ with

$$L = \{w \mid \exists c \in \{0,1\}^* : \ M \text{ accepts } w\$c \text{ in at most } p_M(|w|) \text{ steps}\}. \quad (4.2)$$

(\$ serves here as an auxiliary symbol that lets us write the pair $\langle w, c\rangle$ as a string.) We will now write the protocol of acceptance of $w\$c$ in a format that can be falsified by a polynomial $k$-falsifier, for some $k$ depending on $M$.

## 4.3.2 Representing Computations for $k$-Falsifiers

For configuration $C = (q, \ddot{w}, \ell, h)$ and $m \geq \max\{-h, h, -\ell, \ell + |\ddot{w}| - 1\}$, the *$m$-representation of $C$* is the string

$$\triangleright^{m+h}\, q\, \triangleleft^{m-h}\, \sqcup^{m+\ell}\, \ddot{w}\, \sqcup^{m-\ell-|\ddot{w}|+1}$$

[14] Hopefully, the reader finds this formulistic firework in accordance with the previously supplied intuitive description.

of length $4m + 2$ over $Q \cup \Gamma \cup \{\lhd, \rhd\}$. (As usual, we write $a^i$ for the sequence of $a$'s of length $i$.) Obviously, the $m$-representation fully encodes a configuration, and it is best visualized by writing the second half of the string aligned below the first half, see Figure 4.2. Then the state $q$ ends up exactly above the symbol in $\ddot{w}$ as prescribed by the position of the head defined by the configuration $C$. Note also that given $m$, we can easily

$$\begin{array}{ccccccc}
\rhd & \rhd & q & \lhd & \lhd & \lhd & \lhd \\
\sqcup & 1 & 0 & 1 & 0 & \sqcup & \sqcup \\
\rhd & q' & \lhd & \lhd & \lhd & \lhd & \lhd \\
\sqcup & 1 & 1 & 1 & 0 & \sqcup & \sqcup \\
q & \lhd & \lhd & \lhd & \lhd & \lhd & \lhd \\
\sqcup & 1 & 1 & 1 & 0 & \sqcup & \sqcup
\end{array}$$

*Figure 4.2: The* 3*-representation of* $(q, \sqcup 1010 \sqcup, -3, -1)$ *followed by the* 3*-representations of the next two (yielded) configurations, assuming* $\delta(q, 0) = (q', 1, -1)$ *and* $\delta(q', 1) = (q, 1, -1)$.

falsify a string as not being the $m$-representation of some configuration by exhibiting at most 3 positions of it. In fact, we can do so by ascertaining that

- a symbol in position $i$ is not in $Q \cup \{\lhd, \rhd\}$ for some $i \in \{1..2m+1\}$,
- position 1 contains $\lhd$, or position $2m+1$ contains $\rhd$,
- there is $\rhd\lhd$ or $\lhd\rhd$ or $q\rhd$ or $\lhd q$ or $qq'$ for $q, q' \in Q$ in consecutive positions in the range $\{1..2m+1\}$,
- a symbol in position $i$ is not in $\Gamma$ for some $i \in \{2m+2..4m+2\}$,
- a symbol other than $\sqcup$ is in positions $2m+2$ or $4m+2$, or
- there is a (not necessarily contiguous) subsequence $a \ldots \sqcup \ldots b$ for $a, b \in \Gamma \setminus \{\sqcup\}$ in the range $\{2m+2..4m+2\}$.

Now let $M$ and $p_M$ be as employed as a verifier for some language $L$ in $\mathcal{NP}$, see (4.2). Given a word $w \in L$, there is a certificate $c$ so that $|w\$c| \leq p_M(|w|)$ and $w\$c$ is accepted in at most $p_M(|w|)$ steps. For each such $c$ we let $\tilde{c}$ be the concatenation of the $m$-representations in the protocol of acceptance of $w\$c$, for $m := 1 + p_M(|w|)$. If $t$ is the number of steps it takes for $M$ to accept $w\$c$, $\tilde{c}$ is a string of length $t(4m + 2)$. We have to show that we can falsify a string which does not come from such a protocol by exhibiting at most $k$ positions; in fact, 3 will suffice.

First we have already discussed how to falsify, if a string is not an $m$-representation of a configuration, and we can apply this to substrings in positions $1 + i(4m + 2)$ to $(i + 1)(4m + 2$, $i \in \{0..t\}$. It is also easy to falsify by exhibiting one position, that the first configuration is a starting configuration for $w\$c$, for some $c \in \{0, 1\}^*$. We also detect if the state of the last configuration is not $q_{\text{Yes}}$.

It remains to ensure that each configuration apart from the last one directly yields the next configuration. For that it suffices to consider the symbols in positions $i(4m+2)+j$ and $i(4m+2)+(2m+1)+j$ for $i \in \{0..t-1\}$ and $j \in \{1..2m + 1\}$. Two such symbols entail one or two symbols later in the string as indicated in Figure 4.3. Again this refers to the situation

$$\begin{array}{ccccccccccc}
\triangleleft & \quad & \triangleright & \quad & \cdot & q & \quad & q & \cdot \\
a & & a & & \cdot & a & & a & \cdot \\
\cdot & & \cdot & & q' & \cdot & & \cdot & q' \\
a & & a & & \cdot & b & & b & \cdot
\end{array}$$

*Figure 4.3: The top two symbols force the other symbols, unconditioned in the first two patterns, in case of $\delta(q, a) = (q', b, -1)$ in the third pattern, and in case of $\delta(q, a) = (q', b, +1)$ in the fourth pattern. Exhibiting three positions suffices to show violations of these rules.*

where we write the string $\tilde{c}$ in rows of length $2m + 1$ aligned above each other. Violations to these rules can be checked by a 3-falsifier, and any string that satisfies the rules comes from a protocol of a computation as above.

We are done, except that the derived certificate for the 3-falsifier uses an alphabet other than $\{0, 1\}$ which we assumed in our definition of the $\mathcal{FP}_k$'s. However, we can employ a binary encoding which then leads to a $k$-falsifier for some $k > 3$ depending on the sizes of the alphabets of the Turing machine $M$ (see Exercise 4.6).

**Theorem 4.6** *$\mathcal{NP} = \mathcal{FP}_3$ and* SAT *is $\mathcal{NP}$-complete.*

NOTE **4.1** The fact that SAT is in $\mathcal{P}$ iff $\mathcal{NP} = \mathcal{P}$ was first proved by [Cook, 1971] and independently by [Levin, 1973]. The full impact of the result was realized after [Karp, 1972] supplied a whole list of $\mathcal{NP}$-complete problems, mainly from combinatorial optimization. A compendium of $\mathcal{NP}$-complete problems can be found in [Garey, Johnson, 1979], which—amazingly enough—is still pretty much up-to-date.

Strictly speaking, Cook did not prove $\mathcal{NP}$-completeness in the sense we defined it here via polynomial reductions—these were introduced by Karp in his seminal paper. This point deserves a short discussion. Note that we use $L_1 \leq_{\mathrm{P}} L_2$ to conclude that if $L_2 \in \mathcal{P}$ then $L_1 \in \mathcal{P}$. But for that, we did not have to restrict ourselves to the strict protocol '$w \in L_1 \Leftrightarrow f(w) \in L_2$'; note that this protocol does not allow to ask two questions about membership in $L_2$, or to decide on $w \in L_1$ based on $f(w) \notin L_2$. Consequently, in general, we do not have $\{0,1\}^* \setminus L \leq_{\mathrm{P}} L$, although definitely $\{0,1\}^* \setminus L \in \mathcal{P}$ iff $L \in \mathcal{P}$. A more powerful reduction is that of a Turing reduction. Here we say that $L_1$ is *polynomial time Turing reducible* to $L_2$, if there is a polynomial time bounded Turing machine that decides membership in $L_1$, but which is also allowed to ask membership in $L_2$ at unit cost; we say, the machine uses a *membership oracle* for $L_2$.

# Chapter 5

# The Cube

*Mehr Licht!*
Johann Wolfgang von Goethe[1], famous last words

*More structure!*
Mathematics, first words

Cubes in 3-space generalize to any dimension. On the real line these are intervals and in the plane these are squares. 4-Dimensional cubes are perhaps the most frequently "seen" 4-dimensional objects, see Figure 5.1. Although terminology (and original motivation) for cubes comes from geometry, they are frequently used as a tool in discrete mathematics for the sake of adding structure—so will we. The treatment of cubes in this section relates to satisfiability in that we reconsider facts from previous chapters in the cube setting—hopefully with extra insight—, and we prepare the ground for chapters to come.

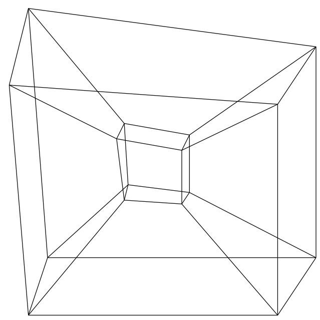


Figure 5.1: Hand-crafted 4-cube.

Often the *graph of an $n$-dimensional hypercube*, here *$n$-cube* for short, is defined as the graph with vertex set $\{0,1\}^n$, and with an edge between each pair of

[1] Johann Wolfgang von Goethe, 1749–1832

sequences that differ in exactly one position (i.e. have Hamming distance 1).

Given its defining parameter $n$, this graph has $2^n$ vertices each of which has degree $n$. Thus the number of edges is $n2^n/2 = n2^{n-1}$. The *diameter* (i.e. the largest distance between any two vertices) is $n$. This is easily seen, since the distance between two vertices $u$ and $v$ in $\{0,1\}^n$ is the Hamming distance between the two $\{0,1\}$-sequences—thus at most $n$; and this bound is attained by the pair $0^n$ and $1^n$. Such pairs at distance $n$, there are[2] $2^{n-1}$ of them, are called *antipodal vertices*.

The fact that the $n$-cube is a graph with few edges while it has small diameter is one of its highly appreciated properties in applications (e.g. for networks). Set $N = 2^n$; then all vertices have degree $\log_2 N$ and the diameter is $\log_2 N$.

Another advantageous feature of the $n$-cube is its connectivity: Clearly, removing the $n$ edges incident to a vertex makes the graph disconnected; but removal of any $n-1$ edges leaves the graph connected (the graph is *$n$-edge connected*). Already the proof of this simple fact calls for extra insight, the *facial structure* of cubes.

## 5.1 Faces

We employ a small adaption of our cube definition. For a set $A$, we let the *$A$-cube* be the graph with the vertex set $\{0,1\}^A$ (all mappings $A \longrightarrow \{0,1\}$) and with an edge between $\alpha$ and $\beta$ iff $\alpha(v) \neq \beta(v)$ for exactly one $v \in A$; we label the edge with this unique $v$. The *dimension* of the $A$-cube is $|A|$.

We observe that two cubes are isomorphic (ignoring the edge labels) iff they agree in dimension. This "new" definition is in perfect agreement with the one above[3]. And in this setting the link to assignments becomes even more apparent. We keep using $n$-cube short for the $\{1..n\}$-cube, the generic representative of all cubes of dimension $n$.

---

[2] If you think about it, $0^5 = 00000$, $1^5 = 11111$, and $2^5 = 32$, ... sometimes it is better not to think about it.

[3] After all $\{0,1\}^n$ is just a shorthand for $\{0,1\}^{\{1..n\}}$. We will identify the two without further explicit mention.

To every mapping $\varphi : A \longrightarrow \{0, 1, *\}$ we associate the set of vertices

$$V_\varphi := \{\alpha \in \{0,1\}^A \mid \alpha|_B = \varphi|_B\} \qquad \text{with } B := \varphi^{-1}(\{0,1\}).$$

(So these are all mappings that agree with $\varphi$ on the elements that are mapped to 0 or 1 by $\varphi$.) The subgraph of the $A$-cube induced by $V_\varphi$ is called the *$\varphi$-induced face of the $A$-cube.* This face is isomorphic to a cube: The mapping

$$V_\varphi \ni \quad \alpha \ \mapsto \ \alpha|_{A \setminus B} \quad \in \{0,1\}^{A \setminus B}$$

constitutes an isomorphism to the $(A \setminus B)$-cube, as can be easily verified. Hence, the dimension of the $\varphi$-induced face is $|\varphi^{-1}(*)|$, "the number of $*$'s in $\varphi$."

The 0-dimensional faces of the $A$-cube are its vertices, the 1-dimensional faces are the edges[4], and the $(n-1)$-dimensional faces, $n := |A|$, are called the *facets*. Note that the $A$-cube is a face of itself, the unique $n$-dimensional face (induced by the constant map to $*$).

For every $a \in A$, the removal of all $a$-labeled edges creates two connected components of the $A$-cube. These are the facets induced by $\varphi'$ and $\varphi''$, resp., with $\varphi'(a) = 0$, $\varphi''(a) = 1$, and $\varphi'(b) = \varphi''(b) = *$ for $b \in A \setminus \{a\}$. This suggests also an instruction for building an $n$-dimensional cube inductively: A 0-dimensional cube is a single vertex. An $n$-dimensional cube can be obtained from two isomorphic copies of an $(n-1)$-dimensional cube with "corresponding" vertices connected, see Figure 5.2.

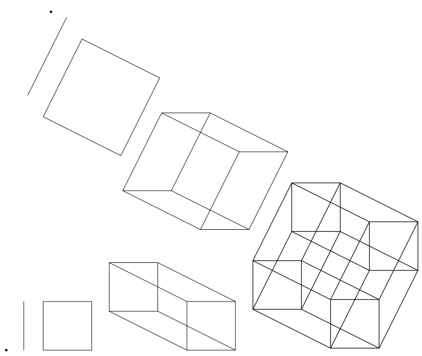


Figure 5.2: Building the 4-cube, in two of many ways (how many?).

[4]Strictly speaking, according to our definition a 0-dimensional face is a graph $(\{v\}, \emptyset)$ (and not a vertex $v$) and a 1-dimensional face is a graph $(\{u,v\}, \{\{u,v\}\})$ (and not an edge $\{u,v\}$).

**Connectivity.** We are ready to settle the connectivity of the $n$-cube.

**Lemma 5.1** $n \in \mathbb{N}$. *Removal of at most* $n-1$ *edges from the* $n$*-cube leaves the graph connected.*

*Proof.* The statement is trivial for $n = 1$.

Now let $n \geq 2$. If at most $n-1$ edges are removed, there must be an $i \in \{1..n\}$ such that all edges labeled $i$ are still present. These edges partition the $n$-cube into two facets $F'$ and $F''$, isomorphic copies of the $(n-1)$-cube. If all removed edges lie in just one copy , $F'$ say, the other copy $F''$ is still connected and thus the whole graph is still connected since every vertex in $F'$ is connected to a vertex in $F''$ via an $i$-labeled edge. Otherwise, in each of the two facets at most $n-2$ edges are removed which leaves them connected by the induction hypothesis—and they are mutually connected by the $i$-labeled edges. ◻

**Faces in Cubes and Clauses in CNF Formulas.** Let $F$ be a $CNF$ formula over $V$. If $C = \{u_1, u_2, \ldots, u_k\}$ is a clause of $F$, then all assignments with $u_i \mapsto 0$, for all $i \in \{1..k\}$, will not satisfy formula $F$, no matter what they do to the remaining variables. The set of all such total assignments on $V$ (which we denoted by $\overline{\mathrm{sat}}_V(C)$) forms the vertex set of an $(n-k)$-dimensional face of the $V$-cube. It is induced by

$$V \ni \quad x \mapsto \begin{cases} 0 & \text{if } x \in C, \\ 1 & \text{if } \overline{x} \in C, \text{ and} \\ * & \text{otherwise.} \end{cases}$$

With this relation in mind, $k$-SAT can be equivalently formulated as

$k$-Codim Cube Noncover

Input: $n \in \mathbb{N}$, *and a set* $F$ *of faces (given as sequences in* $\{0, 1, *\}^n$*) of the* $n$*-cube, each of dimension at least* $n-k$.

Output: *'Yes', if there is a vertex of the* $n$*-cube that lies in none of the faces in* $F$*. And 'No', otherwise (i.e. if the faces in* $F$ *cover the whole* $n$*-cube).*

Previous complexity results can now be read as follows: It can be decided in polynomial time whether a set of $(n-2)$-faces[5] covers the $n$-cube. And it is $\mathcal{NP}$-complete to decide whether a set of $(n-3)$-faces does not cover the $n$-cube.

This fresh view allows several (now) easy observations.

Consider the formula $f = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ (and recall Exercise 1.9). A total assignment on $\{x_1, x_2, \ldots, x_n\}$ satisfies $f$ iff it maps an odd number of variables to 1. In 'cube'-language it means that no unsatisfying assignment is adjacent to any unsatisfying assignment. Thus the set of unsatisfying assignments contains no edges and does not contain all vertices of any $(n-k)$-face for $k < n$. Therefore no CNF formula over $\{x_i\}_{i=1}^n$ with a $k$-clause for some $k < n$ can be equivalent to $f$. In fact, there is a unique CNF-formula equivalent to $f$: An $n$-CNF formula with $2^{n-1}$ clauses.

For another observation recall that at some point (Exercise 1.12) we asked whether every 3-variable formula has an equivalent CNF-formula with at most 4 clauses. In the cube setting this is easy to see: Consider the 3-cube with some of its vertices marked as 'unsatisfying'. Our goal is to cover them with few faces, without covering any of the other vertices. For that take four disjoint edges of the cube (a perfect matching), choose for each of them the subface of marked vertices, and we are done. We exploit here the fact that all nonempty[6] subsets of vertices of a 1-dimensional face form faces; this is peculiar to at most 1-dimensional faces.

Figure 5.3 displays the answer to Exercise 2.3 in the cube setting.

**Packing and Covering the Cube with Faces.** Let us recall two trivialities, which are precious assets, nonetheless. Consider a family $\mathcal{M}$ of subsets of some finite ground set $U$. (1) If $\bigcup_{M \in \mathcal{M}} M = U$ ($\mathcal{M}$ is a *cover* of $U$) then $\sum_{M \in \mathcal{M}} |M| \geq |U|$. (2) If the sets in $\mathcal{M}$ are pairwise disjoint ($\mathcal{M}$ is a *packing* in $U$) then $\sum_{M \in \mathcal{M}} |M| \leq |U|$.

Observation (1) about covers applied to sets of faces of a cube yields the following lemma, a reincarnation of Theorem 2.2.

---

[5] $(n-2)$-Faces are sometimes called *ridges*.

[6] Sometimes, also the *empty face* is introduced which comes handy here and there (see also Exercise 5.1).

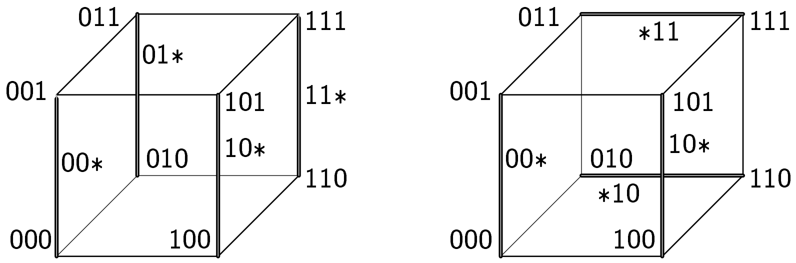



*Figure 5.3: Two ways of covering the vertices of a 3-cube with four edges. Or two unsatisfiable 2-CNF formulas with four clauses (answering Exercise 2.3):* $\{\{x,y\},\{\overline{x},y\},\{x,\overline{y}\},\{\overline{x},\overline{y}\}\}$ *corresponds to the left picture,* $\{\{x,y\},\{\overline{x},y\},\{\overline{y},z\},\{\overline{y},\overline{z}\}\}$ *to the right one.*

**Observation 5.2** $n, m \in \mathbb{N}_0$. *Let* $\{F_1, F_2, \ldots, F_m\}$ *be a set of faces of the* $n$*-cube, with* $F_i$ *of dimension* $d_i$. *If* $\sum_{i=1}^{m} 2^{d_i} < 2^n$ *(or, equivalently,* $\sum_{i=1}^{m} 2^{-(n-d_i)} < 1$*), these faces do not cover the whole* $n$*-cube and there is a vertex which appears in none of the faces.*

Sometimes it takes some extra effort to see that a statement amounts to a simple observation about covers or packings. Two illustrating facts follow. They will come handy in forthcoming sections.

**Lemma 5.3 (Kraft Inequality)** *Let* $S \subseteq \{0,1\}^*$*, finite and prefix-free (i.e. for no* $s \in S$ *there exists* $s' \in S \setminus \{s\}$ *and* $t \in \{0,1\}^*$ *such that* $s = s't$.)

*Then* $\sum_{s \in S} 2^{-|s|} \leq 1$.

*Proof.* Set $n := \max_{s \in S} |s|$ and define $\hat{S} := \{s{*}^{n-|s|} \mid s \in S\}$ (i.e. every sequence in $S$ is extended to length $n$ by appending $*$'s). The elements in $\hat{S}$ can be interpreted as faces of the $n$-cube in the obvious way, and due to the prefix-freeness of $S$, these faces are pairwise disjoint (form a packing). It follows that $\sum_{s \in S} 2^{n-|s|} \leq 2^n$ which—after division by $2^n$—is the assertion of the lemma. ◻

**Lemma 5.4** *Let* $S \subseteq \{0,1\}^n$*, nonempty. For* $s \in S$*, let* $\deg(s)$ *be the number of sequences in* $S$ *that differ from* $s$ *in exactly one position.*

*Then* $\sum_{s \in S} 2^{-\deg(s)} \geq 1$.

*Proof.* Consider the elements in $S$ as mappings in $\{0,1\}^{\{1..n\}}$, i.e. vertices of the $n$-cube. For $s \in S$ we define a $\varphi_s \in \{0,1,*\}^{\{1..n\}}$: For $i \in \{1..n\}$, if the $i$-labeled edge incident to $s$ leads to a neighbor *not* in $S$ then we set $\varphi_s(i) := *$; and $\varphi_s(i) := s(i)$, otherwise. $\varphi_s$ induces a face of dimension $n - \deg(s)$ of the $n$-cube. If we can show that these faces cover the cube, then $\sum_{s \in S} 2^{n-\deg(s)} \geq 2^n$ holds. The claim of the lemma readily follows.

Now consider a vertex $u$ in the $n$-cube. We have to exhibit an $s \in S$ so that the face of $\varphi_s$ contains $u$. Choose $s$ to be some element in $S$ that minimizes the Hamming distance to $u$ (a nearest vertex in $S$, so to say). The neighbors of $s$ that are closer to $u$ than $s$ cannot be in $S$ by the choice of $s$. Thus, indeed, $\varphi_s$ maps to $*$ for every $i$ for which $s$ and $u$ differ, and therefore the face induced by $\varphi_s$ contains $u$. ◻

**Exercise 5.1** Euler-Poincaré

*For $i \in \{0..n\}$ let $f_i$ be the number of $i$-dimensional faces of the $n$-cube. Show that*

$$\sum_{i=0}^{n} f_i = 3^n \quad \textit{and} \quad \sum_{i=0}^{n} (-1)^i f_i = 1 \ .$$

REMARK: The right identity holds for all convex polytopes in $n$-space, where $f_i$ denotes the number of $i$-dimensional faces of the polyhedron. Often the empty face is introduced, which is accounted for by $f_{-1} := 1$. Then the formula reads $\sum_{i=-1}^{n} (-1)^i f_i = 0$.

**Exercise 5.2** "Representing Satisfying Assignments" Revisited

*Reconsider Exercise 1.21.*

**Exercise 5.3** "Almost Satisfiable" Revisited

*Reconsider Exercise 2.1.*

**Exercise 5.4** *(1) Given two clauses $C$ and $D$ over variables $V$. What property of $C$ and $D$ is equivalent to the fact that the two faces of the $V$-cube corresponding to $C$ and $D$ are disjoint?*
*(2) Explain resolution in terms of faces in a cube.*

**Exercise 5.5** *Which ternary boolean functions have no equivalent CNF-formula with at most 3 clauses?*

**Exercise 5.6** *Reformulate Theorem 2.5 about the ratio of satisfiable clauses in a 2-satisfiable CNF sequence-formula in the cube setting.*

**Exercise 5.7** $n \in \mathbb{N}, n \geq 3$. *Show that removal of n edges disconnects the n-cube iff the edges are incident to a common vertex.*

**Exercise 5.8** *Show that in an n-cube there are always n edge-disjoint paths between any pair of distinct vertices. Argue that this also implies Lemma 5.1. (Actually, by Menger's Theorem, this statement is equivalent to n-edge connectivity.)*

**Exercise 5.9** Vertex Connectivity of the Cube
$n \in \mathbb{N}, n \geq 3$. *Show that removal of at most* $n-1$ *vertices leaves the n-cube connected.*

**Exercise 5.10** Universal for Non-Satisfaction
*Let* $V$ *be a set of variables,* $n := |V|$. *We are interested in a set* $\mathcal{A}$ *of assignments in* $\{0,1\}^V$ *such that each 2-clause over* $V$ *is* not *satisfied by some assignment in* $\mathcal{A}$.
*(1) Specify such a set* $\mathcal{A}$ *as small as you can get it (in terms of n).*
*(2) Rephrase the problem in the cube-setting.*

**Exercise 5.11** Large Degree Forces Many Vertices
*Show that an induced subgraph of the n-cube with minimum degree k has at least* $2^k$ *vertices.*
HINT: Lemma 5.4.

## 5.2 Hamming Balls

Every graph is equipped with a metric: The distance between two vertices $u$ and $v$ is the length (number of edges) of a shortest path between the two vertices. In cubes we know that this distance is the Hamming distance, $d_H(u,v)$, between the respective vertices.

And wherever there is a metric, there are balls. The ball of radius $r \in \mathbb{R}$ with center $z \in \{0,1\}^n$ in the $n$-cube is defined as the set

$$\{u \in \{0,1\}^n \mid d_H(z,u) \leq r\} .$$

Note that given $z$ and $d \in \mathbb{N}_0$, we can 'create' a vertex at distance $d$ by flipping exactly $d$ positions in $z$; there are $\binom{n}{d}$ ways to do so. Hence, the number of vertices in a ball of radius $r$ in the $n$-cube, we like to call this its *volume*, is[7]

$$\mathrm{vol}(n,r) := \sum_{i=0}^{r} \binom{n}{i}$$

(obviously independent from the center due to the symmetry of the $n$-cube).

It is worthwhile to try to understand this quantity $\mathrm{vol}(n,r)$, and that's what we will do for the rest of this chapter. For a couple of trivial observations, (i) $\mathrm{vol}(n,r) = \mathrm{vol}(n,\lfloor r \rfloor)$, (ii) $\mathrm{vol}(n,r) = 0$ iff $r < 0$, (iii) $\mathrm{vol}(n,0) = 1$, (iv) $\mathrm{vol}(n,r) = \mathrm{vol}(n,n) = 2^n$ for $r \geq n$, (v) $\mathrm{vol}(n,r) + \mathrm{vol}(n,n-r-1) = 2^n$ for $r \in \mathbb{Z}$, and (vi) $\mathrm{vol}(n,\frac{n}{2}) = 2^{n-1}$ if $n$ is odd and $\mathrm{vol}(n,\frac{n}{2}) = 2^{n-1} + \frac{1}{2}\binom{n}{n/2}$ if $n$ is even.

Despite the exact formula we have for the volume, the sum involved seems of little help in answering a number of questions. For example, while we see that a ball of radius $\frac{n}{2}$ covers roughly half of the cube, what proportion of the cube is covered by a ball of radius $\frac{n}{4}$ or of radius $\frac{n}{2} - \sqrt{n}$? This calls for estimates in terms of functions more 'transparent' than a sum of binomial coefficients. We will discriminate according to how the magnitude of radius and dimension relate to each other.

**Very Small Radius.** Right from the definition of the binomial coefficient we have $\binom{n}{k} \leq n^k$ and thus an upper bound of $(r+1)n^r$ for $\mathrm{vol}(n,r)$ follows. That is, for $r$ constant, the volume of balls is polynomial in the dimension. We refine.

**Lemma 5.5** *$n \in \mathbb{N}$, $r \in \mathbb{R}^+$, $r \leq n$. With the convention $\left(\frac{n}{0}\right)^0 = 1$*

$$\left(\frac{n}{\lfloor r \rfloor}\right)^{\lfloor r \rfloor} \leq \mathrm{vol}(n,r) < \left(\frac{\mathrm{e}n}{r}\right)^r .$$

*Proof.* The lower bound is left as Exercise 5.12. For the upper bound, let $\lambda \in \mathbb{R}^+$, $\lambda \leq 1$. We get (with $1+\lambda < \mathrm{e}^\lambda$ for $\lambda \in \mathbb{R}\setminus\{0\}$ and the binomial

[7] If $r \notin \mathbb{Z}$, read $\sum_{i=0}^{r}$ as $\sum_{i=0}^{\lfloor r \rfloor}$.

theorem)—driven by a magic hand—

$$e^{n\lambda} > (1+\lambda)^n = \sum_{i=0}^{n} \binom{n}{i} \lambda^i \geq \sum_{i=0}^{r} \binom{n}{i} \lambda^i \geq \sum_{i=0}^{r} \binom{n}{i} \lambda^r . \qquad (5.1)$$

Setting $\lambda := \frac{r}{n}$, we have established $e^r > \mathrm{vol}(n,r) \left(\frac{r}{n}\right)^r$ which yields the claimed bound. ◻

For $r > 0.33 \cdot n$, the upper bound exceeds[8] $2^n$ and is of no help at all.

**Radius Proportional to the Dimension.** For the next bound we employ the *binary entropy function*

$$H(x) := -x \log_2 x - (1-x) \log_2 (1-x)$$

for $x \in \mathbb{R}$, $0 < x < 1$, with the continuous extension $H(0) := H(1) := 0$. The function is symmetric about $\frac{1}{2}$, where $H(\frac{1}{2}) = 1$. It is increasing for $0 \leq x \leq \frac{1}{2}$, see Figure 5.4.

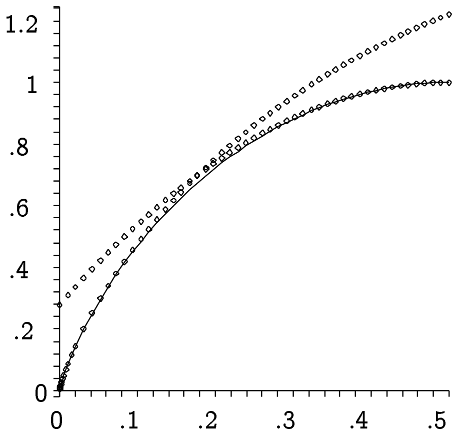



Figure 5.4: Graphs of $H(x)$ (lower solid curve), of $1 - 2(\frac{1}{2} - x)^2 \ln 2$ (upper curve ending in $(0.5, 1)$) and $x \log_2(e/x)$ (upper curve starting in $(0,0)$)—see Note 5.2 below.

**Lemma 5.6** $n \in \mathbb{N}$, $\rho \in \mathbb{R}^+, \rho \leq \frac{1}{2}$.

$$\frac{2^{n \cdot H(\rho)}}{\sqrt{8n\rho(1-\rho)}} \leq \mathrm{vol}(n, \rho n) \leq 2^{n \cdot H(\rho)} .$$

*Proof .* The argument for the lower bound is omitted, cf. [MacWilliams, Sloane, 1977]. For the upper bound we derive with

[8] $(\frac{e}{0.33})^{0.33} = 2.00543\ldots$.

reminiscence of the previous proof

$$\begin{aligned}
1 &= (\rho + (1-\rho))^n \\
&= \sum_{i=0}^{n} \binom{n}{i} \rho^i (1-\rho)^{n-i} \quad \text{(binomial theorem)} \\
&= \sum_{i=0}^{n} \binom{n}{i} (1-\rho)^n \left(\frac{\rho}{1-\rho}\right)^i \\
&\geq \sum_{i=0}^{\rho n} \binom{n}{i} (1-\rho)^n \left(\frac{\rho}{1-\rho}\right)^i \quad \text{(truncation of sum)} \\
&\geq \sum_{i=0}^{\rho n} \binom{n}{i} (1-\rho)^n \left(\frac{\rho}{1-\rho}\right)^{\rho n} \quad \left(\tfrac{\rho}{1-\rho} \leq 1 \text{ and } i \leq \rho n \text{ always}\right) \\
&= \underbrace{\sum_{i=0}^{\rho n} \binom{n}{i}}_{\mathrm{vol}(n,\rho n)} \cdot \underbrace{\rho^{\rho n}(1-\rho)^{(1-\rho)n}}_{2^{-n\cdot H(\rho)}}
\end{aligned}$$

which implies the desired inequality. ◻

For $\rho$ constant, we see that the upper bound is sharp up to a polynomial factor in $n$ (of the order $\frac{1}{\sqrt{n}}$). After calculating $H(\frac{1}{4}) = 2 - \frac{3}{4}\log_2 3$, we conclude that $\mathrm{vol}(n, \frac{n}{4})$ is roughly $(4 \cdot 3^{-3/4})^n = 1.7548\ldots^n$, a neglectable fraction compared to the overall number $2^n$ of vertices (the probability that a vertex u.a.r. from the $n$-cube falls in a given ball of radius $\frac{n}{4}$ is roughly $0.8774\ldots^n$).

Again the estimate has its limits. We still have no clue of how fast the volume decreases as we decrease the radius $\frac{n}{2}$ a bit. For which radii a ball covers only a polynomially small fraction, $\frac{1}{n^2}$ say, of the $n$-cube?

**Radius Close to Half the Dimension.** Before we proceed let us establish a simple relation between $\mathrm{vol}(n,r)$ and a sequence of $n$ mutually independent head-tail experiments with a fair coin. For $i \in \{1..n\}$, let $X_i$ be the random variable which attains $+1$ if the $i$th experiment shows head, and $-1$ if it shows tail; that is $\Pr(X_i = +1) = \Pr(X_i = -1) = \frac{1}{2}$. Then $X = \sum_{i=1}^{n} X_i$ tells us how many more heads than tails we encountered in the course of the experiment. It is easily seen that $\mathrm{E}(X) = 0$.

**Lemma 5.7** $n \in \mathbb{N}$, $\lambda \in \mathbb{R}$. *With $X$ the random variable defined above*

$$\Pr(X \geq \lambda) = \frac{\mathrm{vol}(n, \frac{n-\lambda}{2})}{2^n} \; .$$

*Proof.* $2^n \cdot \Pr(X \geq \lambda)$ is the number of sequences $s$ in $\{-1,+1\}^n$ where the number $|s|_\oplus$ of $+1$'s in $s$ exceeds the number $|s|_\ominus$ of $-1$'s in $s$ by $\lambda$ at least, i.e. $|s|_\oplus - |s|_\ominus \geq \lambda$, or equivalently, $|s|_\ominus \leq \frac{n-\lambda}{2}$ (since $|s|_\oplus = n - |s|_\ominus$). Now if we substitute 1 for $-1$ and 0 for $+1$ then these are the vertices of the $n$-cube in the ball of radius $\frac{n-\lambda}{2}$ centered at $0^n$, the number of which we denoted by $\mathrm{vol}(n, \frac{n-\lambda}{2})$. The claimed relation is established. ∎

We are ready for the next bound.

**Lemma 5.8 (Chernoff Bound)** $n \in \mathbb{N}$, $\delta \in \mathbb{R}^+$.

$$\mathrm{vol}(n, \frac{n}{2} - \delta) < \mathrm{e}^{-2\delta^2/n} \cdot 2^n \; .$$

*Proof.* For $X$ as above we show[9] $\Pr(X \geq \lambda) < \mathrm{e}^{-\lambda^2/(2n)}$ for any $\lambda \in \mathbb{R}^+$ which will give the bound in the lemma via Lemma 5.7 (by setting $\lambda := 2\delta$).

For $t \in \mathbb{R}^+$ and $i \in \{1..n\}$, $\mathrm{E}\left(\mathrm{e}^{tX_i}\right) = \frac{1}{2}\left(\mathrm{e}^t + \mathrm{e}^{-t}\right) < \mathrm{e}^{t^2/2}$. In order to justify the inequality, we investigate the Taylor series of the terms involved.[10]

$$\frac{1}{2}\left(\mathrm{e}^t + \mathrm{e}^{-t}\right) = \frac{1}{2}\sum_{i=0}^{\infty}\left(\frac{t^i}{i!} + \frac{(-t)^i}{i!}\right) = \frac{1}{2}\sum_{i=0}^{\infty} 2\frac{t^{2i}}{(2i)!} < \sum_{i=0}^{\infty}\frac{t^{2i}}{2^i\, i!} = \mathrm{e}^{t^2/2} \; .$$

Mutual independence of the $X_i$'s allows the following estimate.

$$\mathrm{E}\left(\mathrm{e}^{tX}\right) = \mathrm{E}\left(\prod_{i=1}^{n} \mathrm{e}^{tX_i}\right) = \prod_{i=1}^{n} \mathrm{E}\left(\mathrm{e}^{tX_i}\right) < \prod_{i=1}^{n} \mathrm{e}^{t^2/2} = \mathrm{e}^{t^2 n/2} \; .$$

It is time for Markov's Inequality.

$$\Pr(X \geq \lambda) \; = \; \Pr\left(\mathrm{e}^{tX} \geq \mathrm{e}^{t\lambda}\right) \; \leq \; \frac{\mathrm{E}\left(\mathrm{e}^{tX}\right)}{\mathrm{e}^{t\lambda}} \; < \; \mathrm{e}^{t^2 n/2 - t\lambda} \; ,$$

[9] With an argument almost verbatim from [Spencer, 1987].

[10] $\left(\mathrm{e}^t + \mathrm{e}^{-t}\right)/2$, also denoted as $\cosh(t)$, *hyperbolic cosine*.

for all $t \in \mathbb{R}^+$. The parameter $t$ can be chosen so that $\frac{t^2 n}{2} - t\lambda$ is as small as possible. This is attained for $t = \frac{\lambda}{n}$, which yields the inequality claimed in the beginning of the proof. ◻

We now can conclude

$$\mathrm{vol}(n, \frac{n}{2} - \nu\sqrt{n}) < \mathrm{e}^{-2\nu^2} 2^n$$

and

$$\mathrm{vol}(n, \frac{n}{2} - \nu\sqrt{n \ln n}) < n^{-2\nu^2} 2^n \ .$$

For a radius of $\frac{n}{4}$ the bound is

$$\mathrm{vol}(n, \frac{n}{4}) = \mathrm{vol}(n, \frac{n}{2} - \frac{n}{4}) < (\mathrm{e}^{-1/8} 2)^n = 1.7649..^n$$

which is actually close to the right base of 1.7548... we had derived earlier.

**Note 5.1** For a comparison of the bounds in Lemmas 5.5 and 5.6 observe that (5.1) actually shows that $\mathrm{vol}(n, r) \leq \frac{(1+\lambda)^n}{\lambda^r} =: f(\lambda)$ for all $\lambda \in \mathbb{R}$, $0 < \lambda \leq 1$. With $\lambda = \frac{r}{n}$ the bound in Lemma 5.5 evolves, but a more ambitious attack will search for the $\lambda$ which minimizes $f$ in the given range. This is easily done by setting the first derivative $f'(\lambda) = \frac{(1+\lambda)^{n-1}}{\lambda^{r+1}}((n-r)\lambda - r)$ to 0 which suggests $\lambda = \frac{r}{n-r} = \frac{\rho}{1-\rho}$ (for $\rho = \frac{r}{n}$) as long as $r \leq \frac{n}{2}$. This actually yields the bound of Lemma 5.6, which thus is always superior to the bound in Lemma 5.5.

**Note 5.2** Let us relate the upper bounds of Lemmas 5.6 and 5.8. The first one is $\mathrm{vol}(n, \rho n) \leq (2^{H(\rho)})^n$, the second one we can rewrite as

$$\mathrm{vol}(n, \rho n) = \mathrm{vol}(n, \frac{n}{2} - (\frac{1}{2} - \rho)n) < \left(2^{1 - 2(\frac{1}{2} - \rho)^2 \ln 2}\right)^n \ .$$

(and, for the sake of completeness, the bound in Lemma 5.5 as $(2^{\rho \log_2(\mathrm{e}/\rho)})^n$). Therefore the relation of the functions $H(\rho)$ and $1 - 2(\frac{1}{2} - \rho)^2 \ln 2$ in the range $0 \leq \rho \leq \frac{1}{2}$ decides whether the first or the second bound is better, see Figure 5.4 for their graphs. I leave it as an exercise to prove or disprove that the second function indeed majorizes the first one in this range as the figure suggests.

Even if so, the Chernoff bound in Lemma 5.8 has its merits, because we can retrieve from it a better understanding for the volume when the radius is of the form $\frac{n}{2} - o(n)$.

**Covering the Cube with Balls (Covering Codes).** A code $C$ of length $n$ is a subset of $\{0,1\}^n$. The *covering radius* of $C$ is the smallest radius $r$ such that the balls of radius $r$ centered at the elements of $C$ cover all of $\{0,1\}^n$; or equivalently, the smallest $r$ such that every element of $\{0,1\}^n$ is at distance at most $r$ from at least one of the elements in $C$, or simply

$$r := \max_{u \in \{0,1\}^n} \min_{v \in C} d_H(u,v) \ .$$

For example, the covering radius of the code $\{0^n, 1^n\}$ is $\left\lfloor \frac{n}{2} \right\rfloor$. The *normalized covering radius* is $\rho := \frac{r}{n}$.

For $r$ the covering radius of $C$, we see that $|C| \cdot \mathrm{vol}(n,r)$ has to be at least $2^n$, and so

$$|C| \geq \frac{2^n}{\mathrm{vol}(n,r)} \geq 2^{(1-H(\rho))n} \ ,$$

with $\rho$ the normalized covering radius. This constitutes a lower bound on the size of a code with covering radius $r$ or smaller. Up to a factor of $n$ this bound can be achieved.

**Lemma 5.9** $n \in \mathbb{N}, r \in \mathbb{N}_0$. *There exists a code of length $n$ of covering radius at most $r$ with at most*

$$\left\lceil \frac{n \cdot 2^n}{\mathrm{vol}(n,r)} \right\rceil = O\left(2^{(1-H(\rho))n} \operatorname{poly}(n)\right)$$

*elements.*

*Proof.* We choose the elements of the code u.a.r. from $\{0,1\}^n$ with replacement, $\left\lceil \frac{n \cdot 2^n}{\mathrm{vol}(n,r)} \right\rceil$ of them, and we show that there is a positive probability that this code has covering radius at most $r$. This shows the existence of a code as claimed.

Let $u$ be some element in $\{0,1\}^n$. The probability that it has distance exceeding $r$ from all elements in the randomly generated code—we say $u$ is not covered—is

$$\left(1 - \frac{\mathrm{vol}(n,r)}{2^n}\right)^{\lceil n \cdot 2^n / \mathrm{vol}(n,r) \rceil} \leq \mathrm{e}^{-n}.$$

Consequently, the probability that there is an element in $\{0,1\}^n$ not covered is at most $2^n \mathrm{e}^{-n}$, and thus the probability that all elements are covered is at least $1-\left(\frac{2}{\mathrm{e}}\right)^n > 0$. In fact, the probability quickly tends to 1 as $n$ grows. $\square$

**Exercise 5.12** Lower Bound for Binomial Coefficient

*For $k, n \in \mathbb{N}$, show that $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$ (thereby establishing the lower bound in Lemma 5.5).*

**Exercise 5.13** Volume versus Boundary

*$k, n \in \mathbb{N}_0$, $k \leq \frac{n}{2}$. Show that*

$$\binom{n}{k} \leq \sum_{i=0}^{k} \binom{n}{i} \leq \binom{n}{k}\left(1+\frac{k}{n-2k+1}\right) .$$

**Exercise 5.14** Far from Home

*We recall the definition of a symmetric random walk $(W_t)_{t\in\mathbb{N}_0}$ on $\mathbb{Z}$ from Chapter 3. For $n, k \in \mathbb{N}$, show that*

$$\Pr(|W_n| \geq k \mid W_0 = 0) = \frac{\mathrm{vol}(n, \frac{n-k}{2})}{2^{n-1}} .$$

*Is the identity true for $k = 0$?*

**Exercise 5.15** *Prove or refute. For all $0 \leq x < \frac{1}{2}$,*

$$H(x) < 1 - 2\left(\frac{1}{2} - x\right)^2 \ln 2 .$$

**Exercise 5.16** Ellipsoids in the Cubical World

*Given $k, n \in \mathbb{N}_0$ and vertices $u$ and $v$ in the $n$-cube, we let*

$$\mathrm{ell}(u, v, k) := \{w \in \{0,1\}^n \mid d_H(w, u) + d_H(w, v) \leq k\} .$$

*(1) What can you say about $\mathrm{ell}(u, v, d_H(u, v))$?*
*(2) What is the smallest $k$ (in terms of $n$ and $d_H(u, v)$) such that $\mathrm{ell}(u, v, k)$ comprises the whole $n$-cube?*
*(3) In general, how many vertices are in (what is the volume of) $\mathrm{ell}(u, v, k)$?*

**Exercise 5.17** Football Pools

*(from [Cohen* et al.*, 1997]) Assume we wish to place bets on the winners of n football matches. A bet is a prediction of the winners of these n matches—no ties allowed. We are interested in the smallest number of bets needed to guarantee that at least one of them has at most r incorrect predictions.*

*Relate this to covering codes.*

**Exercise 5.18** $n \in \mathbb{N}_0$. *n real numbers, i.e. points on the real line* $\mathbb{R}$ *induce* $n+1$ *intervals between these points, two of which are infinite (unless* $n = 0$*). Show the generalizations to higher dimensions.*

*(1) The complement of the union of n lines in* $\mathbb{R}^2$ *in general position (i.e. no two parallel and no three through a common point) consists of* $1+n+\binom{n}{2}$ *connected components (called* cells *of the line arrangement).*

*(2)* $d \in \mathbb{N}$. *The complement of the union of n hyperplanes in* $\mathbb{R}^d$ *in general position*[11] *(i.e. any d of them have a unique common point) consists of* $\sum_{i=0}^{d} \binom{n}{i}$ *connected components (cells of the hyperplane arrangement).*

HINT: Employ proof by induction over number of hyperplanes and dimension. When adding a hyperplane to $n-1$ hyperplanes, note that the intersection of this new hyperplane with the old ones is an arrangement of dimension $d-1$. And the cells in this $(d-1)$-dimensional arrangement account exactly for the cells which get split by the newly added hyperplane.

**Exercise 5.19** And now Generalize

$q \in \mathbb{N}$. *Consider the graph with vertex set* $\{0..q-1\}^n$ *where two sequences are adjacent if they disagree on exactly one position.*

*(1) What is the number of vertices and edges of this graph? What is the diameter? What is the connectivity?*

*(2) Define faces? How many are there?*

*(3) What is the volume of balls of radius r in this graph? Can you derive estimates for this quantity?*

*(4) etc.*

[11] The right definition of general position is: Any $i \le d$ of the hyperplanes intersect in a $(d-i)$-flat. This is the same as requiring that any $d$ intersect in a unique point (a 0-flat), unless $n < d$.

# Chapter 6

# Coding and $k$-SAT Algorithms

We start with the question "How efficiently (in terms of number of bits necessary) can we encode satisfying assignments of a given CNF formula?" The "usual" encoding agrees on some ordering of the variables $V$ and represents a (satisfying) assignment by a sequence in $\{0,1\}^n$ of length $n$, $n := |V|$. Actually, in general, there is no space for improvement: First, the empty CNF formula over $V$ has $2^n$ satisfying assignments. Second, we have the following fundamental information-theoretic fact.

**Lemma 6.1** *If $e : S \to \{0,1\}^*$ is a prefix-free encoding (i.e. injective), then the average code length $\left(\sum_{x \in S} |e(x)|\right)/|S|$ is at least $\log_2 |S|$.*

*Proof.* The Kraft Inequality (Lemma 5.3) tells us that $\sum_{x \in S} 2^{-|e(x)|} \leq 1$. Employ this in

$$-\log_2\left(\frac{\sum_{x \in S} 2^{-|e(x)|}}{|S|}\right) \leq \frac{\sum_{x \in S} -\log_2 2^{-|e(x)|}}{|S|} = \frac{\sum_{x \in S} |e(x)|}{|S|} \tag{6.1}$$

and the lemma follows. The inequality in (6.1) is a consequence of Jensen's Inequality (see Lemma A.2) since $a \mapsto -\log_2 a$ is a convex function on $\mathbb{R}^+$. $\boxminus$

So as long as we want to stay with prefix-free codes, $n$ bits are necessary on the average and we are done. What we will do, though, is to show that under a certain condition (that remains to be specified) efficient encodings are possible. And if these conditions are not met, there will be many satisfying assignments.

These ideas will suggest the following for an algorithm. Either there are many satisfying assignments, when it is easy to find one. Or there are few, when we try to decode all "short" code words to find a satisfying assignment. The actual algorithm we will consider will be simple and natural—the challenge is to supply an analysis for it.

The mysterious condition states that there is a satisfying assignment that has only few satisfying neighbors in the cube.

So much for a rough outline, we will develop towards more specific definitions and facts.

## 6.1 Encoding Satisfying Assignments

Along an example we start with a refined discussion of the ideas we want to pursue.

**An Example.** Consider the CNF formula with clauses

$$\{x_1, x_2\}, \quad \{\overline{x_2}, \overline{x_3}, \overline{x_4}\}, \quad \{x_1, x_4\}, \quad \{\overline{x_1}, x_3\}\ .$$

The sequence 0101 is a satisfying assignment (with the convention that $x_1$ gets assigned the first bit in the sequence, $x_2$ the second, and so on). If you start telling this sequence to somebody else (who knows the formula, the order of variables, and that the assignment is *satisfying*), she can conclude the second bit after seeing the first bit: This is so, because $x_1 \mapsto 0$ leaves $x_2 \mapsto 1$ as the only possibility to satisfy clause $\{x_1, x_2\}$. After $x_1 \mapsto 0$ and $x_2 \mapsto 1$ is set, she may want to know the bit for $x_3$, which is 0, while $x_4$ is forced to be 1 because of the third clause. That is, the sequence $0 * 0 *$ carries enough information to specify the sequence 0101. In fact, 00 (without specifying to which positions the bits correspond) suffices, if we agree to skip a value exactly if it is determined, because the corresponding variable or its negation are left as the last possible satisfying literal in a clause. Here it is important to stick to this protocol, and not to try to be too smart: You may have realized that actually $x_3 \mapsto 0$ is already implied by $x_1 \mapsto 0$ and $x_2 \mapsto 1$ (why?)—still, we list $x_3$'s value in 00. With this convention, it is possible to efficiently decode the sequence.[1]

The example also illustrates that the encoding, and even its length depend on the underlying agreed upon ordering of the variables. For example, if the order were $(x_1, x_2, x_4, x_3)$, then the single bit 0 encodes the above satisfying assignment (which is now 0110 since the order changed). And with the order $(x_2, x_3, x_4, x_1)$ the same assignment encodes as 101.

Another observation: The satisfying assignment above is rigid in the sense that flipping any single value of it will make it unsatisfying. With the cube-setting in mind, we call such an assignment isolated. Note that this means that every variable or its negation appear as the only satisfying literal in one of the clauses. Clearly, this is a necessary condition for its value to be determined in an encoding, but whether it is indeed, depends on the order in which we consider the variables.

The goal of this section is to show that isolated (or close to isolated) satisfying assignments have short encodings, and to conclude (with Lemma 6.1) an upper bound on the possible number of isolated satisfying assignments.

**The Formal Set-Up.** We let $\mathcal{S}_V = \mathcal{S}_V$ denote the set of bijective mappings $\{1..n\} \longrightarrow V$, and we will be sloppy[2] in referring to this as the set of permutations of $V$.

Let us first specify the procedures for encoding and decoding, see function procedures encode() and decode(), respectively.

Assuming $F$ and $V$ to be fixed, we will denote the output of the call encode$(\alpha, F, V, \pi)$ by enc$(\alpha, \pi)$. Decoding just reverts the process, as specified by procedure decode(). Here are two basic facts that follow quite easily.

**Observation 6.2** *(i) Given a CNF formula $F$ over $V$, the mapping* enc$(\cdot,\cdot)$

[1] We could, of course, agree to skip exactly those bits which are implied by previous bits. However, deciding this is an $\mathcal{NP}$-complete problem, and thus neither efficient encoding nor decoding is possible, at least not according to current state of the art.

[2] Recall that, strictly speaking, a permutation of a set $M$ is a bijective map in $M^M$, and not an ordering of the elements in $M$ or such like, although it is sometimes convenient to think of it this way.

| | |
|---|---|
| $F$ CNF formula over $V$, $\alpha \in \{0,1\}^V$, $\pi \in \mathcal{S}_V$.<br>PRECONDITION:<br>$\alpha \in \mathrm{sat}_F(V)$.<br>POSTCONDITION:<br>returns a string in $\{0,1\}^*$. | **function** encode$(\alpha, F, V, \pi)$<br>$\varepsilon \leftarrow$ empty string;<br>**for** $i \leftarrow 1$ **to** $\lvert V\rvert$ **do**<br>    $x \leftarrow \pi(i)$;<br>    **if** $\{x\} \notin F$ **and** $\{\overline{x}\} \notin F$ **then**<br>        $\varepsilon \leftarrow \varepsilon\, \alpha(x)$;<br>    $F \leftarrow F^{[x \mapsto \alpha(x)]}$;<br>**return** $\varepsilon$; |

| | |
|---|---|
| $\varepsilon = (\varepsilon_j)_{j=1}^{\lvert\varepsilon\rvert} \in \{0,1\}^*$, $F$ CNF formula over $V$, $\pi \in \mathcal{S}_V$.<br>POSTCONDITION:<br>returns $\alpha$ with $\varepsilon = \mathrm{enc}(\alpha, \pi)$, if it exists, and the empty map, otherwise. | **function** decode$(\varepsilon, F, V, \pi)$<br>$j \leftarrow 0$; $\alpha \leftarrow \emptyset$;<br>**for** $i \leftarrow 1$ **to** $\lvert V\rvert$ **do**<br>    $x \leftarrow \pi(i)$;<br>    **if** $\{x\} \in F$ **then** $b \leftarrow 1$;<br>    **elseif** $\{\overline{x}\} \in F$ **then** $b \leftarrow 0$;<br>    **else**<br>        **if** $j = \lvert\varepsilon\rvert$ **then return** $\emptyset$;<br>        $j \leftarrow j+1$; $b \leftarrow \varepsilon_j$;<br>    $\alpha \leftarrow \alpha \cup \{x \mapsto b\}$; $F \leftarrow F^{[x \mapsto b]}$;<br>**if** $\Box \in F$ **or** $j \neq \lvert\varepsilon\rvert$ **then return** $\emptyset$;<br>**else return** $\alpha$; |

*is a prefix-free encoding of* $\mathrm{sat}_F(V)$.
*(ii) Procedure* decode() *computes the inverse of* $\mathrm{enc}(\cdot,\cdot)$ *on the image of* $\mathrm{sat}_F(V)$ *under* $\mathrm{enc}(\cdot,\cdot)$.

Given an assignment $\alpha \in \mathrm{sat}_F(V)$, we call a variable $x \in V$ *critical for* $\alpha$ if flipping the value of $x$ in $\alpha$ stops it being satisfying. This requires at least one clause in $F$ that has $x$ or $\overline{x}$ as the unique literal that is set to 1 by $\alpha$. Such a clause is called *critical for*[3] $x$. By[4] $j(\alpha)$ we denote the number

[3] Every critical variable has a clause which is critical for it, and every critical clause is critical for exactly one variable

[4] For $\alpha \in \mathrm{sat}_F(V)$, let $\deg(\alpha)$ denote the number of neighbors of $\alpha$ in the $V$-cube that

of critical variables for $\alpha$, and we say that $\alpha$ is *$j$-isolated*, if $j(\alpha) \geq j$. $n$-isolated assignments (thus $j(\alpha) = n$) are called *isolated*.

**Lemma 6.3 (Satisfiability Coding Lemma)** *If $\alpha$ is a $j$-isolated satisfying assignment of a $(\leq k)$-CNF formula over $n$ variables, then its expected coding length (for the order of variables chosen u.a.r. from all $n!$ permutations) is at most $n - \frac{j}{k}$.*

*Proof.* Since $\alpha$ is $j$-isolated, there are $j$ variables with a critical clause; we fix one such critical clause $C_x$ for each such variable $x$. The probability for $x$ to appear as the last variable of $\mathrm{vbl}(C_x)$ in a random permutation is $\frac{1}{|C_x|} \geq \frac{1}{k}$. If it is last, its bit will be skipped in the encoding. Therefore, the expected number of bits skipped in an encoding with a random permutation is at least $\frac{j}{k}$ which proves the lemma. ◻

**Lemma 6.4** $j \in \mathbb{N}_0, k, n \in \mathbb{N}$. *Any $(\leq k)$-CNF formula over $n$ variables has at most $2^{n-j/k}$ satisfying assignments that are $j$-isolated. (In particular, there are at most $2^{n-n/k}$ isolated satisfying assignments.)*

*Proof.* Given a $j$-isolated satisfying assignment, its average code length over all variable permutations is at most $n-\frac{j}{k}$. Therefore, the expected code length of a u.a.r. chosen $j$-isolated satisfying assignment for a u.a.r. variable permutation is at most $n - \frac{j}{k}$. Consequently, there exists a permutation, for which the expected code length of a u.a.r. chosen $j$-isolated satisfying assignments is at most $n - \frac{j}{k}$. Now, since the codes we consider are prefix-free, Lemma 6.1 steps in and shows that there cannot be more than $2^{n-j/k}$ satisfying assignments that are $j$-isolated.

If that was a bit fast, let us do it more explicitely—an example of double counting. Let $F$ be the $(\leq k)$-CNF formula under consideration, let $V$ be the variable set, and Let $A$ be the set of $j$-isolated satisfying assignments. For $\alpha \in A$ we know that $\frac{1}{n!}\sum_{\sigma \in \mathcal{S}_V} |\mathrm{enc}(\alpha, \sigma)| \leq n - \frac{j}{k}$. Hence,

$$\frac{1}{n!}\sum_{\sigma \in \mathcal{S}_V} \underbrace{\left(\frac{1}{|A|}\sum_{\alpha \in A} |\mathrm{enc}(\alpha, \sigma)|\right)}_{(*)} = \frac{1}{|A|}\sum_{\alpha \in A}\left(\frac{1}{n!}\sum_{\sigma \in \mathcal{S}_V} |\mathrm{enc}(\alpha, \sigma)|\right) \leq n - \frac{j}{k}$$

---

are satisfying (recall Lemma 5.4). Then $\deg(\alpha) = n - j(\alpha)$.

and thus for at least one permutation $\sigma \in \mathcal{S}_V$ the term $(*)$ has to be at most $n - \frac{j}{k}$. This permutation certifies that there is a prefix-free encoding of $A$ of average length at most $n - \frac{j}{k}$. ◻

Finally, we show that the bound cannot be improved upon for the case of isolated satisfying assignments. Let $n := mk$, $m, k \in \mathbb{N}$. The formula

$$\bigwedge_{i=0}^{m-1} (x_{ik+1} \oplus x_{ik+2} \oplus \ldots \oplus x_{ik+k})$$

has an equivalent $k$-CNF formula over variable set $\{x_1, x_2, \ldots, x_n\}$ (with $m2^{k-1}$ clauses). All satisfying assignments to this formula are isolated, and there are $(2^{k-1})^m = 2^{mk-m} = 2^{n-n/k}$ of them.

**Exercise 6.1** Harmonic vs. Geometric Mean

*Prove the harmonic versus geometric mean inequality*

$$\left(\frac{\sum_{i=1}^{n} \frac{1}{x_i}}{n}\right)^{-1} \leq \sqrt[n]{\prod_{i=1}^{n} x_i} \quad \text{for } n \in \mathbb{N} \text{ and } x_i\text{'s in } \mathbb{R}^+,$$

*with the help of Jensen's Inequality.*

**Exercise 6.2** Many $j$-Isolated Satisfying Assignments

*$j \in \mathbb{N}_0$, $k, n \in \mathbb{N}$. Find $(\leq k)$-CNF formulas over $n$ variables with many $j$-isolated satisfying assignments also for $j < n$.*

## 6.2 A Randomized $k$-SAT Algorithm

Consider procedure ppz() for finding a satisfying assignment. We will analyze its probability of success, provided the CNF formula is satisfiable. Along the usual lines we can repeat the procedure in order to boost the success probability.

On the one hand, the procedure is a natural way to proceed: It simply builds a random assignment variable by variable, except that it refrains from making too obvious mistakes (namely to set a literal to 0 although it forms a 1-clause in the current restriction of $F$). On the other hand, it can be viewed as the decoding procedure decode() applied to a random $\{0,1\}$-string with a random permutation.

$F$ CNF formula over $V$.
POSTCONDITION: returns an assign't satisfying $F$ or the empty map.

```
function ppz(F, V)
π ←random S_V;  α ← ∅;
for i ← 1 to |V| do
    x ← π(i);
    if {x} ∈ F then b ← 1;
    elseif {x̄} ∈ F then b ← 0;
    else b ←random {0,1};
    α ← α ∪ {x ↦ b};  F ← F^[x↦b];
    if □ ∈ F then return ∅;
return α;
```

**Lemma 6.5** *The probability, that* ppz() *finds a satisfying assignment in a* $(\leq k)$*-CNF formula over* $n$ *variables is at least* $2^{-n+n/k}$.

*Proof.* Fix a $(\leq k)$-CNF formula $F$ over $V$, $|V| = n$. The main observation for the proof is as follows: If the algorithm decides for a permutation $\sigma$ then in order to generate a specific satisfying assignment $\alpha$, it has $|\mathrm{enc}(\alpha, \sigma)|$ free choices, which—since made at random—come out right with probability $2^{-|\mathrm{enc}(\alpha,\sigma)|}$. Thus

$$\begin{aligned}
\Pr(\mathrm{ppz}() \text{ returns } \alpha) &= \sum_{\sigma \in \mathcal{S}_V} \Pr(\mathrm{ppz}() \text{ returns } \alpha \,|\, \pi = \sigma) \cdot \Pr(\pi = \sigma) \\
&= \sum_{\sigma \in \mathcal{S}_V} 2^{-|\mathrm{enc}(\alpha,\sigma)|} \cdot \frac{1}{n!} \\
&\geq 2^{\frac{1}{n!}\sum_{\sigma \in \mathcal{S}_V} -|\mathrm{enc}(\alpha,\sigma)|} \\
&\geq 2^{-n+j(\alpha)/k}
\end{aligned}$$

The penultimate inequality uses Jensen's Inequality (with the function $a \mapsto 2^a$ which is convex on $\mathbb{R}$). The last inequality uses the monotonicity of $a \mapsto 2^a$ and the fact that the average code length of a $j$-isolated satisfying assignment is at most $n - \frac{j}{k}$.

Note that if there is an isolated satisfying assignment $\alpha$ (i.e. $j(\alpha) = n$) then are done. If not, we have to go through the following estimate. It exploits the fact that if there are no $j$-isolated satisfying assignments for large $j$, then there have to be many (as quantified by Lemma 5.4).

$$\begin{aligned}
\Pr(\text{ppz() returns some } \alpha \in \mathrm{sat}_V(F)) &= \sum_{\alpha \in \mathrm{sat}_V(F)} \Pr(\text{ppz() returns } \alpha) \\
&\geq \sum_{\alpha \in \mathrm{sat}_V(F)} 2^{-n+j(\alpha)/k} \\
&= 2^{-n+n/k} \sum_{\alpha \in \mathrm{sat}_V(F)} 2^{-(n-j(\alpha))/k} \\
&\geq 2^{-n+n/k} \sum_{\alpha \in \mathrm{sat}_V(F)} 2^{-(n-j(\alpha))} \\
&\geq 2^{-n+n/k}
\end{aligned}$$

For the last inequality we recall that $n - j(\alpha) = \deg(\alpha)$ (see Footnote 4), and with this in mind we use Lemma 5.4. ⊒

We are by now well-experienced[5] to turn this into the following Monte-Carlo type of result.

**Theorem 6.6** *For a $(\leq k)$-CNF formula $F$ over $n$ variables, the probability that $\lambda 2^{n-n/k}$ independent repetitions of* ppz() *find no satisfying assignment, even though one exists, is at most* $e^{-\lambda}$.

Concrete Bounds that follow from this theorem are Monte-Carlo algorithms for $k$-SAT with running time $O(2^{(2/3)n}\,\mathrm{poly}(n)) = O(1.588^n)$ for $k = 3$ and $O(2^{(3/4)n}\,\mathrm{poly}(n)) = O(1.682^n)$ for $k = 4$.

Note **6.1** The results of this section are from [Paturi *et al.*, 1997]. That paper supplies also a derandomized version of ppz() whose time complexity approaches $O(2^{n-n/(2k)})$ for large values of $n$ and $k$. The randomized procedure was further improved by these authors in [Paturi *et al.*, 1998] ($O(1.447^n)$ Monte-Carlo for 3-SAT).

Some of these bounds will be improved upon in the next chapter, where we will also summarize the currently best known bounds.

**Exercise 6.3** *Recall the example of a $k$-CNF formula over $n$ variables with $2^{n-n/k}$ isolated satisfying assignments. What can you say about the success probability of* ppz() *for such a formula.*

[5] Recall that if we repeat an experiment with success probability $p$ for $\lceil \lambda/p \rceil$ rounds, then the probability of always failing is $(1-p)^{\lceil \lambda/p \rceil} \leq e^{-p\,\lambda/p} = e^{-\lambda}$.

**Exercise 6.4** Make it Hard for ppz()

*Describe $(\leq k)$-CNF formulas over $n$ variables for which you can prove a small success probability for* ppz() *(ideally as small as $2^{-n+n/k}$).*

# Chapter 7

# Hamming Balls and $k$-SAT Algorithms

We start with a procedure that searches for a satisfying assignment of a CNF formula $F$. Besides $F$, it holds an assignment $\alpha$ as parameter. If this assignment satisfies $F$ we are done, otherwise we take a clause $C$ not satisfied—thus all literals in $C$ are mapped to 0 by $\alpha$—and try for each literal $u$ in $C$ recursively whether setting $u$ to 1 yields a satisfiable formula. There is a third parameter $r \in \mathbb{N}_0$, though, whose benefits will materialize promptly. Namely, we search only for a satisfying assignment that differs from $\alpha$ in at most $r$ variables. In other words, in the cube of all assignments we search for a satisfying assignment within the Hamming ball of radius $r$ centered at $\alpha$.

---

| | |
|---|---|
| $F$ CNF, $\alpha$ assign't,<br>$r \in \mathbb{N}_0$.<br>PRECONDITION:<br>$\alpha$ total for $F$.<br>POSTCONDITION:<br>returns true if there is an assign't satisfying $F$ at distance $\leq r$ from $\alpha$. And false, otherwise. | **function** $\mathrm{sb}(F, \alpha, r)$<br>**if** $\alpha$ satisfies $F$ **then return** true;<br>**elseif** $r = 0$ **then return** false;<br>**else**<br>    $C \leftarrow_{\text{some in}} \{D \in F \mid \alpha \text{ does not satisfy } D\}$;<br>    **for all** $u \in C$ **do**<br>        **if** $\mathrm{sb}(F^{[u \mapsto 1]}, \alpha, r-1)$ **then return** true;<br>    **return** false; |

---

It is easy to see that procedure sb() ( *"search ball"*) serves its purpose:

Given $F$ and $\alpha$, either $\alpha$ satisfies $F$, or for a clause $C$ not satisfied by $\alpha$ at least one literal in $C$ has to change (compared to $\alpha$) its value in any satisfying assignment. (In particular, if $C$ is an empty clause, the procedure can return false right away.)

For the analysis, we get

**Lemma 7.1** $r \in \mathbb{N}_0, k \in \mathbb{N}, k \geq 2$. *For a* $(\leq k)$*-CNF formula* $F$*, a call* $\mathrm{sb}(F, \alpha, r)$ *with* $\alpha$ *an assignment on* $V \supseteq \mathrm{vbl}(F)$ *entails at most* $\frac{k^{r+1}-1}{k-1}$ *calls to* $\mathrm{sb}()$ *(including the initial call). Thus it can be implemented to run in time* $O(k^r \operatorname{poly}(n))$*,* $n := |V|$*.*

*Proof.* Let $t(r)$ be the maximum possible number of calls entailed by a call with a $(\leq k)$-CNF formula and parameter $r$. Then

$$t(r) \leq \begin{cases} 1 & \text{if } r = 0, \text{ and} \\ 1 + k \cdot t(r-1) & \text{otherwise.} \end{cases}$$

This readily gives the asserted bound. ◻

**An Easy Deterministic Procedure for 3-SAT.** We see that for a $(\leq 3)$-CNF formula $F$ over $n$ variables and $r = \lfloor n/2 \rfloor$, the procedure takes time $O(3^{n/2}\operatorname{poly}(n)) = O(1.733^n)$ which is substantially less than the number of assignments at distance $\lfloor n/2 \rfloor$ which is roughly $2^{n-1}$. This can be exploited by deciding the existence of a satisfying assignment by two calls $\mathrm{sb}(F, \alpha_0, \lfloor n/2 \rfloor)$ and $\mathrm{sb}(F, \alpha_1, \lfloor n/2 \rfloor)$ where $\alpha_0$ maps all variables to 0 and $\alpha_1$ all variables to 1. Since every assignment has distance at most $\lfloor n/2 \rfloor$ either from $\alpha_0$ or from $\alpha_1$, the search will not miss a satisfying assignment if one exists at all. The overall running time is still upper bounded by $O(1.733^n)$.

**The Volume of Hamming Balls.** We recall from Chapter 5 that for $r \in \mathbb{R}$ and $n \in \mathbb{N}_0$, $\mathrm{vol}(n, r) := \sum_{i=0}^{r} \binom{n}{r}$ is the number of assignments at distance at most $r$ from a given assignment on $n$ variables—the "volume" of a closed Hamming ball of radius $r$ in the $n$-cube. And for $\rho := r/n$ (the normalized radius) with $0 < \rho \leq \frac{1}{2}$ we have the estimate (Lemma 5.6)

$$(8n\rho(1-\rho))^{-1/2}\, 2^{H(\rho)n} \leq \mathrm{vol}(n, r) \leq 2^{H(\rho)n} \ ,$$

where $H(\rho) := -\rho\log_2\rho - (1-\rho)\log_2(1-\rho)$, the *binary entropy function.*

A value of interest shortly is $\rho = \frac{1}{4}$; we have

$$H(\tfrac{1}{4}) = -\tfrac{1}{4}\log_2\tfrac{1}{4} - \tfrac{3}{4}\log_2\tfrac{3}{4} = 2 - \tfrac{3}{4}\log_2 3.$$

That is, $\mathrm{vol}(n, \frac{n}{4})$ is $\left(4\cdot 3^{-3/4}\right)^n$ up to a polynomial factor.

**A Simple Randomized Improvement.** Let $F$ be a satisfiable ($\leq 3$)-CNF formula over $V$, $n := |V|$. If we choose an assignment $\alpha$ u.a.r. in $\{0,1\}^V$, then the ball of radius $r := \lfloor n/4 \rfloor$ centered at $\alpha$ contains a fixed satisfying assignment $\alpha^*$ with probability $\frac{\mathrm{vol}(n,r)}{2^n} \geq \left(2\cdot 3^{-3/4}\right)^n/\mathrm{poly}(n)$, which will be found by a call $\mathrm{sb}(F,\alpha,r)$ in time $O(3^{n/4}\mathrm{poly}(n))$. Hence, if we repeat this search with a new random assignment, then the expected number of trials until a satisfying assignment is found is at most $(3^{3/4}/2)^n\,\mathrm{poly}(n)$ and the expected time is at most $O(\left(3^{n/4}\,3^{3n/4}/2^n\right)\mathrm{poly}(n)) = O((3/2)^n\,\mathrm{poly}(n))$.

If we take a not necessarily satisfiable ($\leq 3$)-CNF formula over $n$ variables, and we repeat the above search for $\lceil \lambda\, 2^n/\mathrm{vol}(n,r)\rceil$ rounds without success, then—along by now familiar lines[1]—we can conclude that there is no satisfying assignment with error probability at most $e^{-\lambda}$.

**Lemma 7.2** *There is a deterministic algorithm that decides satisfiability of a ($\leq 3$)-CNF formula over $n$ variables in time $O(\sqrt{3}^n\mathrm{poly}(n)) = O(1.733^n)$.*

$\lambda \in \mathbb{R}, \lambda > 0$. *There is a randomized algorithm with running time $O(\lambda\cdot 1.5^n\,\mathrm{poly}(n))$ that either finds a satisfying assignment of a ($\leq 3$)-CNF formula over $n$ variables, or concludes unsatisfiability with error probability at most $\mathrm{e}^{-\lambda}$.*

Both results will now we improved upon, and extended to arbitrary $k \geq 3$.

## 7.1 A Deterministic Algorithm

The deterministic procedure searches from the two antipodal constant assignments $\mapsto 0$ and $\mapsto 1$. What about choosing more starting points? In

[1] Recall Footnote 5 in Chapter 6.

fact, that's what we did in the randomized procedure. We will now show that we can choose them in a deterministic fashion. For that we switch to the language of covering codes from Section 5.2.

### 7.1.1 Employing Covering Codes

We showed that for $n \in \mathbb{N}$ and $r \in \mathbb{N}_0$ there exists a code of length $n$ of covering radius at most $r$ with at most $O\left(2^{(1-H(\rho))n}\operatorname{poly}(n)\right)$ elements (Lemma 5.9).

So the bound looks promising and helpful, but it isn't satisfactory either, since the proof of the bound did not reveal a method for generating a good code in a deterministic fashion.

Let us suppose we can for a moment. Then we choose some $r$, generate a code of covering radius $r$ or smaller of the size given in Lemma 5.9, and use the elements of the code as starting assignments for a search with radius $r$—this takes $k^r$ time for a $(\leq k)$-CNF formula up to a polynomial factor. That is, altogether the time is (up to a polynomial factor)

$$2^{(1-H(\rho))n}k^{\rho n} = 2^{(1-H(\rho)+\rho\log_2 k)n} \quad .$$

The first derivative of $g(\rho) := 1 - H(\rho) + \rho\log_2 k$ is

$$\log_2 \rho + \frac{1}{\ln 2} - \log_2(1-\rho) - \frac{1}{\ln 2} + \log_2 k = \log_2 \frac{k\rho}{1-\rho}$$

which is 0 for $\rho = \frac{1}{k+1}$, where $g$ attains its minimum value for $0 < \rho < 1$.

$$\begin{aligned} g\left(\frac{1}{k+1}\right) &= 1 + \tfrac{1}{k+1}\log_2\tfrac{1}{k+1} + \tfrac{k}{k+1}\log_2\tfrac{k}{k+1} + \tfrac{1}{k+1}\log_2 k \\ &= 1 + \left(\tfrac{1}{k+1} + \tfrac{k}{k+1}\right)\log_2\tfrac{1}{k+1} + \left(\tfrac{k}{k+1} + \tfrac{1}{k+1}\right)\log_2 k \\ &= 1 + \log_2\frac{k}{k+1} \end{aligned}$$

and $2^{g(1/(k+1))n} = \left(2\frac{k}{k+1}\right)^n = \left(2 - \frac{2}{k+1}\right)^n$. This is $1.5^n$ for 3-SAT, $1.6^n$ for 4-SAT, etc. Hence, it seems worthwhile to invest in efficient constructions of such codes.

### 7.1.2 Constructing Covering Codes.

We present two deterministic algorithms of varying time complexity and varying guarantee concerning the size of the code.

**Greedy Algorithm.** Constructing a small covering code is a set covering problem: We want to cover the vertices $\{0,1\}^n$ of the $n$-cube by few balls of given radius $r$. The greedy algorithm successively chooses a next ball so that as many as possible yet uncovered vertices are covered. It is well known, that this gives a cover of size at most a factor $(1 + \ln 2^n) = O(n)$ off the size of the optimal cover.[2]

In a naïve implementation we associate with each vertex the number of yet uncovered elements in the ball of radius $r$ centered at this element. Each time we select a new vertex for the code, we can update these numbers in time $O(2^{2n}\text{poly}(n))$. There are at most $2^n$ iterations, so the overall time is $O(2^{3n}\text{poly}(n))$ (all of these are crude over-estimates, of course). We have

**Lemma 7.3** *$n \in \mathbb{N}$, $\rho \in \mathbb{R}$, $0 < \rho \leq \frac{1}{2}$. A code of length $n$ with covering radius at most $\rho n$ and size $O(2^{(1-H(\rho))n}\,\text{poly}(n))$ can be constructed in time $O(2^{3n}\,\text{poly}(n))$.*

**Splitting into Linear Size Blocks.** Fix some $d \in \mathbb{N}$ and let $n$ be an integer multiple of $d$. First we construct a code $C'$ of length $\frac{n}{d}$ via the procedure described above. Then we let $C := C'^d$, that is the set of all possible concatenations of $d$ (not necessarily distinct) elements in $C'$. We have spent time $O\left(\left(2^{3n/d} + |C|\right)\text{poly}(n)\right)$ and the size of $C$ is

$$O\left(\left(2^{(1-H(\rho))n/d}\,\text{poly}(n/d)\right)^d\right) = O\left(2^{(1-H(\rho))n}\,\text{poly}_d(n)\right) \ ,$$

where $\text{poly}_d$ indicates that the degree of the polynomial depends on $d$.

[2] A result due to [Johnson, 1974] and [Lovász, 1975], see e.g. [Cormen *et al.*, 1990, Section 37.3]. It states that if $\mathcal{F} \subseteq 2^U$, $U$ some finite ground set, then the greedy algorithm produces a cover of $U$ with sets in $\mathcal{F}$ of size at most $(1 + \ln \max_{S\in\mathcal{F}} |S|)$ times the size of the smallest such cover.

Therefore, if $3/d \leq (1 - H(\rho))$, that is $d \geq 3/(1 - H(\rho))$, then the time needed for the construction is the size of the code times a polynomial factor (with the degree depending on $\rho$).

**Lemma 7.4** *$n \in \mathbb{N}$, $\rho \in \mathbb{R}, 0 < \rho \leq \frac{1}{2}$. A code of length $n$ with covering radius at most $\rho n$ and of size $O\left(2^{(1-H(\rho))n}\,\mathrm{poly}_\rho(n)\right)$ can be constructed in time and space $O\left(2^{(1-H(\rho))n}\,\mathrm{poly}_\rho(n)\right)$. ($\mathrm{poly}_\rho$ is a polynomial of degree $O\left(\frac{1}{1-H(\rho)}\right)$.)*

Procedure cov() represents the deterministic approach based on covering codes developed in this section (due to [Dantsin *et al.*, 2002]) with the time complexity summarized in Theorem 7.5.

---

| | |
|---|---|
| | **function** cov($F$) |
| $F$ CNF formula. | $k \leftarrow \max_{C \in F} \lvert C \rvert$;   $\rho \leftarrow \frac{1}{k+1}$; |
| POSTCONDITION: | $n \leftarrow \lvert \mathrm{vbl}(F) \rvert$; |
| returns true if $F$ is satisfiable. And false, otherwise. | $\mathcal{C} \leftarrow$ code of length $n$ with cov. radius $\leq \rho n$; |
| | **for all** $\alpha \in \mathcal{C}$ **do** |
| | **if** sb($F, \alpha, \lfloor \rho n \rfloor$) **then return** true; |
| | **return** false; |

---

**Theorem 7.5** *$k, n \in \mathbb{N}$. Satisfiability of a $(\leq k)$-CNF formula $F$ over $n$ variables can be decided in time $O\left(\left(2 - \frac{2}{k+1}\right)^n \mathrm{poly}_k(n)\right)$.*

NOTE **7.1** Further tailored improvements for $(\leq 3)$-CNF formulas lead to a variant of sb() with running time $O(2.848^r\,\mathrm{poly}(n))$ which can be employed for a variant of cov() with running time $O(1.481^n)$. This bound for 3-SAT, and the bounds in Theorem 7.5 for $k$-SAT, $k \geq 4$, are the best known and have been established in [Dantsin *et al.*, 2002]. Before that, the best bound known for 3-SAT, for example, was $O(1.505^n)$ from [Kullmann, 1999] following a series of improvements starting with a bound of $O(1.619^n)$ by [Dantsin, 1981, Monien, Speckenmeyer, 1985].

**Exercise 7.1** $m \in \mathbb{N}$. *What is the covering radius of the code*

$$\{0^{3m}, 0^m 1^{2m}, 1^m 0^m 1^m, 1^{2m} 0^m\}$$

*of length* $n := 3m$*?*

**Exercise 7.2** $n \in \mathbb{N}, r \in \mathbb{N}_0$. *Show that there exists a code of length* $n$ *of covering radius at most* $r$ *with at most* $\left\lceil \frac{(\ln 2)\, n \cdot 2^n}{\mathrm{vol}(n,r)} \right\rceil$ *elements. (*$\ln 2 \approx 0.6931\ldots$*)*

**Exercise 7.3** $r \in \mathbb{N}_0, n \in \mathbb{N}, r \le n$. *Show that if there is a code of size* $M$ *of length* $n$ *with covering radius at most* $r$*, then there is a code of size at most* $M$ *of length* $n$ *with covering radius exactly* $r$.

**Exercise 7.4** *(Challenge) Adapt* sb() *so that it runs for a* $(\le 3)$*-CNF formula over* $n$ *variables and radius argument* $r$ *in* $O(c^r \mathrm{poly}(n))$*, for some* $c < 3$.

## 7.2 A Randomized Algorithm

For the next randomized procedure not only will we start our search for a satisfying assignment from random assignments, but we will also do the search itself in a randomized manner. Actually, we will take over the random-flip procedure from Section 3.2, with the exception that we will let it run only for a limited number $s$ of steps, see procedure rfb() (*"random flip bounded"*).

The idea is to repeatedly call procedure rfb() with appropriate $s$. With clairvoyant foresight, let us use $s := 3n$, $n := |\mathrm{vbl}(F)|$, as a choice for the number of steps we are willing to patiently wait for success, before we repeat the search with a new random initial assignment, see procedure sch() (*"Schöning's"*).

In order to determine the expected number of iterations needed by sch() to find a satisfying assignment of a $(\le k)$-CNF formula, we will have to analyze the success probability of rfb() for a satisfiable $(\le k)$-CNF formula.

$F$ CNF formula,
$s \in \mathbb{N}_0$, $\alpha$ return parameter.
POSTCONDITION:
if true is returned then $\alpha$ is an assign't on $\mathrm{vbl}(F)$ satisfying $F$.

**function** $\mathrm{rfb}(F, s, \alpha)$
$\alpha \leftarrow_{\text{random}} \{0,1\}^{\mathrm{vbl}(F)}$;
**if** $\alpha$ satisfies $F$ **then return** true;
**while** $s > 0$ **do**
  $C \leftarrow_{\text{some unsatisfied by } \alpha \text{ in}} F$;
  $u \leftarrow_{\text{random}} C$;
  flip assignment for $u$ in $\alpha$;
  **if** $\alpha$ satisfies $F$ **then return** true;
  $s \leftarrow s - 1$;
**return** false;

$F$ CNF formula.
POSTCONDITION:
returns assignment $\alpha$ on $\mathrm{vbl}(F)$ satisfying $F$, if it terminates.

**function** $\mathrm{sch}(F)$
$n \leftarrow |\mathrm{vbl}(F)|$;
**repeat**
**until** $\mathrm{rfb}(F, 3n, \alpha)$;
**return** $\alpha$;

### 7.2.1 Random Walks Revisited

Similarly to the situation with procedure rf() in Section 3.2 we consider a satisfiable ($\leq k$)-CNF formula $F$ and we fix some satisfying assignment $\alpha^*$ on $\mathrm{vbl}(F)$. Let $\alpha$ be some assignment not satisfying $F$. If the value of a random literal in an unsatisfied clause $C$ is flipped in $\alpha$, then with probability at least[3] $\frac{1}{|C|} \geq \frac{1}{k}$ the Hamming distance between $\alpha$ and $\alpha^*$ decreases by 1.

Note also that the initial random assignment will have Hamming distance $j$ from the fixed satisfying assignment with probability $\binom{n}{j}\frac{1}{2^n}$ for $0 \leq j \leq n$, $n := |\mathrm{vbl}(F)|$. That is, the Hamming distance between $\alpha$ and $\alpha^*$ starts with the symmetric binomial distribution as just described, and then in each step decreases by 1 with probability at least $q := \frac{1}{k}$ or increases by 1 with probability at most $1 - q = 1 - \frac{1}{k}$.

[3] Since $F$ is satisfiable, $C \neq \Box$.

We model now a Markov chain that will be pessimistic compared to rfb() in the sense that—via appropriate coupling—rfb() will find a satisfying assignment with higher probability than the Markov chain will reach its designated goal state 0.

The state set of the Markov chain is $\{S\} \cup \mathbb{N}_0$, with transition probabilities

$$\begin{aligned} p_{S,j} &:= \binom{n}{j}\frac{1}{2^n}, \text{ for } 0 \leq j \leq n, \\ p_{i,i+1} &:= 1-q, \quad \text{for } i \in \mathbb{N}, \\ p_{i,i-1} &:= q, \quad \text{for } i \in \mathbb{N} \\ p_{0,0} &:= 1, \quad \text{and} \\ &:= 0, \quad \text{otherwise.} \end{aligned}$$

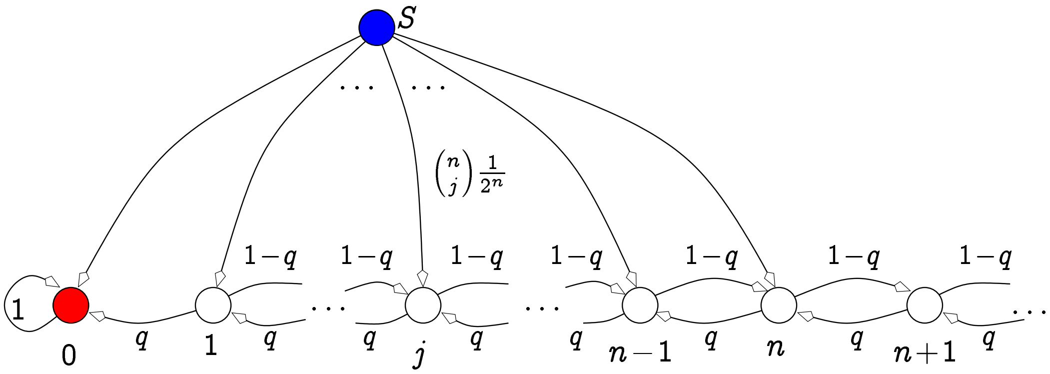



*Figure 7.1: The "pessimistic" Markov chain for* rfb().

Note that this chain allows motion into states greater than $n$, while this is not feasible for the Hamming distance between $\alpha$ and $\alpha^*$.

For $n \in \mathbb{N}$ and $q \in \mathbb{R}$, $0 < q < 1$, fixed, we consider random variables $Y$ and $X_j$, $j \in \mathbb{N}_0$, so that

- $Y + 1$ is the number of steps from S to the first encounter of 0, and
- $X_j$ is the number of steps from $j$ to the first encounter of 0.

Our goal is to lower estimate $\Pr(Y \le 3n)$, since this will provide a lower bound for the success probability of rfb(). A problem we face here is that for $q \le \frac{1}{2}$ the expectation of $Y$ is infinite, since—as we will see—there is a positive probability that $Y = \infty$.

Let us first evaluate our chances to reach 0 at all starting from some state $j \in \mathbb{N}_0$, and the expected number of steps until we reach 0, if we reach it at all.

**Lemma 7.6** *If $q < \frac{1}{2}$, then for all $j \in \mathbb{N}$,*

$$\Pr(X_j < \infty) = \left(\frac{q}{1-q}\right)^j \quad \textit{and} \quad \mathbb{E}(X_j \mid X_j < \infty) = \frac{j}{1-2q} \ .$$

*Proof.* $j = 0$ is trivial, so let us assume $j \in \mathbb{N}$. According to the ballot theorem, cf. [Grimmett, Stirzaker, 1992], for $i \in \mathbb{N}_0$, the number of walks (with step size one) of length $2i + j$ from $j$ to 0 where the first encounter of 0 happens in the last step is[4] $\binom{2i+j}{i}\frac{j}{2i+j}$. Hence,

$$\begin{aligned} \Pr(X_j < \infty) &= \sum_{i=0}^{\infty} \binom{2i+j}{i} \frac{j}{2i+j} (1-q)^i q^{i+j} \\ &= q^j \sum_{i=0}^{\infty} \binom{2i+j}{i} \frac{j}{2i+j} \ (q(1-q))^i \\ &= q^j \ (\mathcal{B}_2(q(1-q)))^j \, , \end{aligned} \tag{7.1}$$

for $\mathcal{B}_2(z)$ the generalized Binomial series defined by

$$\mathcal{B}_2(z) = \sum_i \binom{2i+1}{i} \frac{z^i}{2i+1} = \frac{1-\sqrt{1-4z}}{2z} \ ,$$

for which

$$(\mathcal{B}_2(z))^r = \sum_i \binom{2i+r}{i} \frac{r}{2i+r} z^i \ ,$$

[4]This is the same as the number of walks of length $2i + j - 1$ from $j$ to 1 which *never* encounter 0. Without the latter restriction, this number is $\binom{2i+j-1}{i}$, since we simply have to choose when we make the $i$ increasing steps. Now take such a walk that does encounter 0. Reflect the walk *after* the first encounter of 0 in the sense that we make an increasing step for a decreasing step in the original walk, and vice versa. In this way we get a walk from $j$ to $-1$. In fact, we have described a bijection to such walks of length $2i + j - 1$. That is, the number in question is $\binom{2i+j-1}{i} - \binom{2i+j-1}{i-1} = \binom{2i+j}{i}\frac{j}{2i+j}$.

for all $r \in \mathbb{N}$, cf. [Graham *et al.*, 1989, Section 5.4, page 293]. So

$$\Pr(X_j < \infty) \;=\; q^j \left(\frac{1-\sqrt{1-4q+4q^2}}{2\,(1-q)\,q}\right)^j \;=\; q^j \left(\frac{1}{1-q}\right)^j .$$

For the expectation we get

$$\begin{aligned}
\mathrm{E}(X_j | X_j < \infty) &= \frac{1}{\Pr(X_j < \infty)} \sum_{i=0}^{\infty} (2i+j) \binom{2i+j}{i} \frac{j}{2i+j} (1-q)^i \, q^{i+j} \\
&= \frac{j\, q^j}{\Pr(X_j < \infty)} \sum_{i=0}^{\infty} \binom{2i+j}{i} \, (q(1-q))^i \\
&= \frac{j\, q^j}{q^j \, (\mathcal{B}_2(q(1-q)))^j} \frac{(\mathcal{B}_2(q(1-q)))^j}{\sqrt{1-4q(1-q)}} \quad \text{see (7.1) and (7.2)} \\
&= \frac{j}{1-2q}
\end{aligned}$$

exploiting the identity

$$\frac{\mathcal{B}_2(z)^r}{\sqrt{1-4z}} = \sum_i \binom{2i+r}{i} z^i \, , \tag{7.2}$$

for all $r \in \mathbb{N}$, cf. [Graham *et al.*, 1989]. ◻

**Lemma 7.7** *If* $q < \frac{1}{2}$,

$$\Pr(Y < \infty) \;=\; \left(\frac{1}{2(1-q)}\right)^n \, . \tag{7.3}$$

*Proof.*

$$\begin{aligned}
\Pr(Y < \infty) &= \sum_{j=0}^{n} \binom{n}{j} \frac{1}{2^n} \Pr(X_j < \infty) \\
&= \sum_{j=0}^{n} \binom{n}{j} \frac{1}{2^n} \left(\frac{q}{1-q}\right)^j \\
&= \frac{1}{2^n} \left(1 + \frac{q}{1-q}\right)^n
\end{aligned}$$

with the help of the Binomial Theorem $\sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i} = (x+y)^n$. ◻

**Lemma 7.8** *If* $q < \frac{1}{2}$,

$$\mathrm{E}(Y \,|\, Y < \infty) \;=\; \frac{qn}{1-2q}\;.$$

*Proof.* We use use $\Pr(A|B) = \Pr(A \wedge B)/\Pr(B)$ for

$$\begin{aligned}
\mathrm{E}(Y \,|\, Y < \infty) &= \sum_{i=0}^{\infty} i \,\Pr(Y = i \,|\, Y < \infty) \\
&= \frac{1}{\Pr(Y < \infty)} \sum_{i=0}^{\infty} i \,\Pr(Y = i) \\
&\qquad \text{(since } (Y = i) \wedge (Y < \infty) \Leftrightarrow (Y = i)) \\
&= \frac{1}{\Pr(Y < \infty)} \sum_{i=0}^{\infty} i \left( \sum_{j=0}^{n} \binom{n}{j} \frac{1}{2^n} \Pr(X_j = i) \right) \\
&= \frac{1}{2^n \Pr(Y < \infty)} \sum_{j=0}^{n} \binom{n}{j} \sum_{i=0}^{\infty} i \,\Pr(X_j = i) \\
&= \frac{1}{2^n \Pr(Y < \infty)} \sum_{j=0}^{n} \binom{n}{j} \mathrm{E}(X_j \,|\, X_j < \infty) \Pr(X_j < \infty) \\
&\qquad \text{(since } \Pr(X_j = i) = \Pr(X_j = i \wedge X_j < \infty) \\
&\qquad\qquad\qquad\qquad = \Pr(X_j = i \,|\, X_j < \infty) \Pr(X_j < \infty)) \\
&= (1-q)^n \sum_{j=0}^{n} \binom{n}{j} \frac{j}{1-2q} \left( \frac{q}{1-q} \right)^j \\
&\qquad \text{(see Lemmas 7.7 and 7.6)} \\
&= \frac{n(1-q)^n}{1-2q} \sum_{j=1}^{n} \binom{n-1}{j-1} \left( \frac{q}{1-q} \right)^j \\
&= \frac{n(1-q)^n}{1-2q} \frac{q}{1-q} \sum_{j=1}^{n} \binom{n-1}{j-1} \left( \frac{q}{1-q} \right)^{j-1} \\
&= \frac{n(1-q)^n}{1-2q} \frac{q}{1-q} \left( 1 + \frac{q}{1-q} \right)^{n-1} \\
&= \frac{n(1-q)^n}{1-2q} \frac{q}{1-q} \left( \frac{1}{1-q} \right)^{n-1} \\
&= \frac{nq}{1-2q}\;.
\end{aligned}$$

□

Finally, we employ Markov's inequality to get an estimate for reaching 0 fast.

**Lemma 7.9** *If $q < \frac{1}{2}$,*

$$\Pr\left(Y \leq \frac{\lambda q n}{1-2q}\right) \;>\; \left(1-\frac{1}{\lambda}\right)\left(\frac{1}{2(1-q)}\right)^n$$

*for $\lambda \geq 1$.*

*Proof.* Write $\mu$ for $\mathrm{E}(Y\,|Y<\infty)$. Observe that

$$\Pr\left(Y > \lambda\mu \,|\, Y < \infty\right) \;<\; \frac{1}{\lambda}$$

and

$$\Pr\left(Y \leq \lambda\mu\right) \;=\; \Pr\left(Y \leq \lambda\mu \,|\, Y<\infty\right)\cdot\Pr\left(Y<\infty\right),$$

since $(Y \leq \lambda\mu \;\wedge\; Y<\infty) \Leftrightarrow (Y \leq \lambda\mu)$. □

For $q = \frac{1}{k}$, $k \geq 3$, (i.e. $\frac{q}{1-2q} \leq 1$)—thus returning to the parameters in the motivating process—we get

$$\Pr\left(Y \leq 3n\right) \;\geq\; \Pr\left(Y \leq \frac{3qn}{1-2q}\right) \;>\; \frac{2}{3}\left(\frac{1}{2}\left(\frac{k}{k-1}\right)\right)^n .$$

We skip the formal coupling argument for arguing that the success probability of rfb() is indeed lower bounded by $\Pr\left(Y \leq 3n\right)$, see Section 3.2. Accepting this, we conclude that for 3-SAT we need an expected number of at most $\frac{3}{2}\left(\frac{4}{3}\right)^n$ iterations. And the probability of not finding a satisfying assignment within $\frac{3\lambda}{2}\left(\frac{4}{3}\right)^n$ iterations is at most $\mathrm{e}^{-\lambda}$. Note that an iteration here needs polynomial time only. We summarize.

**Theorem 7.10** *The expected time for* sch() *for finding a satisfying assignment in a satisfiable $(\leq k)$-CNF formula over $n$ variables is at most $O(\left(\frac{2(k-1)}{k}\right)^n \mathrm{poly}(n))$.*

*For $\lambda \in \mathbb{R}^+$ there is a Monte Carlo algorithm that needs time $O(\lambda\left(\frac{2(k-1)}{k}\right)^n \mathrm{poly}(n))$ for a $(\leq k)$-CNF formula over $n$ variables. If it claims satisfiability, the answer is always correct. If it claims unsatisfiability, there is an error probability of at most $\mathrm{e}^{-\lambda}$.*

NOTE 7.2 In order to avoid case distinctions, we ignored $q \geq \frac{1}{2}$ (justifiably, since $\frac{1}{k} < \frac{1}{2}$ for $k \geq 3$). The calculation simply would have to proceed with $2q - 1$ for $\sqrt{1 - 4q + 4q^2}$, whenever that expression occurs. Also some special care has to be taken for $q = \frac{1}{2}$, when $\mathrm{E}(Y \mid Y < \infty) = \infty$ (although $\Pr(Y < \infty) = 1$).

NOTE 7.3 The randomized algorithm of this section is due to [Schöning, 2002], an ingenious follow-up of Papdimitriou's algorithm from Section 3.2 (after all, it took roughly ten years for this to surface). Improvements to $O(1.3302^n)$, to $O(1.32793^n)$ and to $O(1.324^n)$ (best known) for 3-SAT have been developed by [Hofmeister *et al.*, 2002], [Rolf, 2003], and [Iwama, Tamaki, 2003], respectively. The latter paper has also an improvement to $O(1.474^n)$ (best known) for 4-SAT. It combines the approaches of [Paturi *et al.*, 1997] (with improvement in [Paturi *et al.*, 1998]) and of [Schöning, 2002] (with improvements mentioned above).

The best bounds for randomized algorithms for $k$-SAT, $k > 4$, can be found in [Paturi *et al.*, 1998].

The deterministic algorithm from Section 7.1 is a derandomized offspring of Schöning's randomized algorithm.

# Chapter 8

# Constraint Satisfaction

In a *constraint satisfaction problem (CSP)* we are given a finite set of variables, each of which has a finite list of possible colors. Moreover, there is a set of constraints, each one a set of variable-color pairs. If $V$ denotes the set of variables and $L_v$ denotes the list of possible colors of $v \in V$, then a constraint is a set

$$C = \{(v_1, c_1), (v_2, c_2), \ldots, (v_k, c_k)\}\ ,\ k \in \mathbb{N}_0,$$

with $v_j \in V$, pairwise distinct, and $c_j \in L_{v_j}$ for $j \in \{1..k\}$.

A *coloring* $\chi$ assigns colors to variables from their respective lists of possible colors ($\chi(v) \in L_v$ for all $v \in V$). It *satisfies a constraint* $C$, if there is $(v, c) \in C$ with $\chi(v) \neq c$. And it satisfies the CSP if it satisfies all of its constraints. We say that $\chi$ *violates constraint* $C$ if $\chi(v) = c$ for all $(v, c) \in C$.

Consider the CNF formula

$$\underbrace{(x \vee y)}_{C_1} \wedge \underbrace{(\overline{x} \vee y \vee z)}_{C_2} \wedge \underbrace{(y \vee \overline{z})}_{C_3} \wedge \underbrace{(x \vee \overline{y} \vee z)}_{C_4}\ .$$

We can represent this as an equivalent CSP in two ways. The more obvious one looks for truth values (now called colors) for the variables so that none of the clauses is violated.

- Variables: $x$,$y$, and $z$ with color lists $L_x = L_y = L_z = \{0, 1\}$.

- Constraints: $\{(x,0),(y,0)\}$ (for $C_1$), $\{(x,1),(y,0),(z,0)\}$ (for $C_2$), $\{(y,0),(z,1)\}$ (for $C_3$), and $\{(x,0),(y,1),(z,0)\}$ (for $C_4$).

A possible satisfying coloring is given by $\{x \mapsto 1, y \mapsto 1, z \mapsto 0\}$.

The alternative representation is heading for a "responsible" variable for each clause, so that whenever a variable is chosen for several clauses, then it appears as the same literal in all of these clauses. (Variable $v_i$ represents clause $C_i$.)

- Variables: $v_1$, $v_2$, $v_3$, and $v_4$ with color lists $L_{v_1} = \{x,y\}$, $L_{v_2} = \{x,y,z\}$, $L_{v_3} = \{y,z\}$, and $L_{v_4} = \{x,y,z\}$.
- Constraints: $\{(v_1,x),(v_2,x)\}$, $\{(v_2,x),(v_4,x)\}$, $\{(v_1,y),(v_4,y)\}$, $\{(v_2,y),(v_4,y)\}$, $\{(v_3,y),(v_4,y)\}$, $\{(v_2,z),(v_3,z)\}$, and $\{(v_3,z),(v_4,z)\}$.

Here $\{v_1 \mapsto x, v_2 \mapsto y, v_3 \mapsto y, v_4 \mapsto x\}$ constitutes a satisfying coloring.

A constraint of cardinality $k$ is called a *$k$-constraint.* A CSP with a set $S$ of 2-constraints can be visualized as a graph $G$ with vertex set $V(G) := \bigcup_{v \in V} (\{v\} \times L_v)$ and edge set $E(G) := S$. The CSP is satisfiable iff there is an independent set of vertices which has one representative in each set $\{v\} \times L_v$; see Figure 8.1.

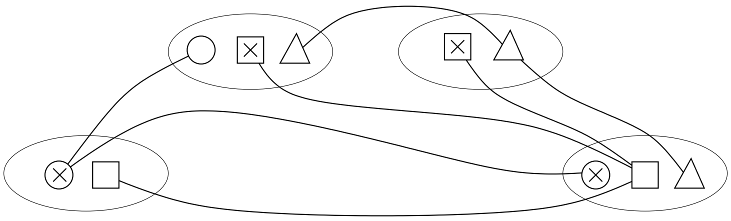


*Figure 8.1: Visualizing a CSP with 2-constraints as a graph. The $\times$'s indicate a possible solution—an independent set which has one vertex (color) in each variable.*

Proper 3-colorability of a graph $G = (V, E)$ translates to CSP as follows.

- Variables: $V$, all with the same color list $\{1,2,3\}$.

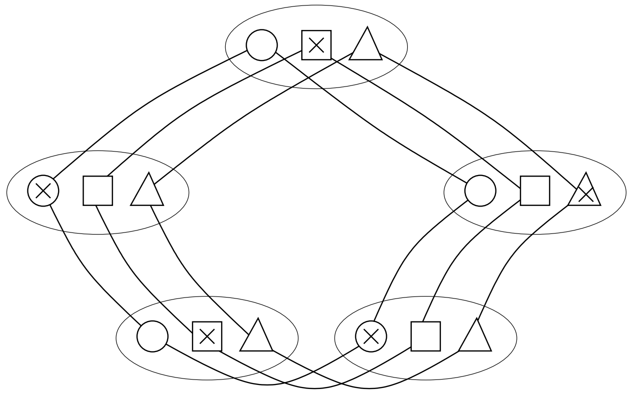


*Figure 8.2: Visualizing a CSP with 2-constraints (from proper 3-colorability of a cycle of length 5) as a graph ($\times$'s indicate a possible solution).*

- Constraints: $\{(u,i),(y,i)\}$ for all $\{u,v\} \in \binom{V}{2}$ and $i = 1,2,3$.

If in a CSP $max_{v\in V}|L_v| \leq \ell$ and $\max_{C\in S}|C| \leq k$, then we call it a $(\leq\ell, \leq k)$-CSP, and it should be clear as well what $(\ell,k)$-CSP, $(\leq\ell,k)$-CSP, and $(\ell,\leq k)$-CSP stands for.

**Exercise 8.1** CSP vs. CNF

*$k,\ell \in \mathbb{N}$. Show that for every $(\leq\ell,\leq k)$-CSP $S$ with $n$ variables we can construct in polynomial time a $(\leq \max\{\ell,k\})$-CNF formula $F$ with at most $\ell n$ variables so that $S$ is satisfiable iff $F$ is satisfiable.*

**Exercise 8.2** Two Colors

*$k \in \mathbb{N}$. Show that for every $(2,\leq k)$-CSP $S$ with $n$ variables we can construct in polynomial time a $(\leq k)$-CNF formula $F$ with at most $n$ variables so that $S$ is satisfiable iff $F$ is satisfiable.*

**Exercise 8.3** Unsatisfiability Needs many Constraints

*$k,\ell \in \mathbb{N}$. Find $b(\ell,k)$ (as large as possible), such that every unsatisfiable $(\ell,k)$-CSP must have at least $b(\ell,k)$ constraints.*

## 8.1 An Algorithm for $(3, 2)$-CSP

Note that $(3, 2)$-CSP is exactly the set-up we get from representing proper 3-colorability of graphs. We will show how to find a satisfying coloring—provided it exists—in expected time $O(\sqrt{2}^n \mathrm{poly}(n))$, $n$ the number of variables; thus the same bound follows for proper 3-coloring of $n$-vertex graphs. As we will see at hindsight, the result for 3-coloring inherently exploits the embedding into the more general setting of CSPs (meaning, there is no obvious way how to specialize the result to 3-coloring without talking about CSP or similar).

The result relies on two lemmas due to [Beigel, Eppstein, 1995].

**Lemma 8.1** *If a $(\leq\ell, 2)$-CSP $S$ has $n$ variables, one of which has at most two possible colors, then we can transform it in polynomial time into a $(\leq\ell, 2)$-CSP $S'$ with $n-1$ variables, so that $S$ is satisfiable iff $S'$ is satisfiable.*

*Proof.* If the distinguished variable $v$ has no possible color, the problem is trivial, and if there is one possible color $i$, then we can assume that $v$ is colored $i$, and remove the pair $(v, i)$ in constraints, wherever it occurs.

So let us assume that $v$ has colors $i$ and $j$ at disposal. Now we add a constraint $\{(u, i'), (w, j')\}$ whenever

$$\{(u, i'), (v, i)\} \in S \text{ and } \{(w, j'), (v, j)\} \in S \ ,$$

unless $u = w$ and $i' \neq j'$, see Figure 8.3. On the one hand this does not further restrict our set of legal colorings: $u \mapsto i'$ implies $v \mapsto j$ because of $\{(u, i'), (v, i)\}$, and $w \mapsto j'$ implies $v \mapsto i$ because of $\{(w, j'), (v, j)\}$. So $u \mapsto i'$ *and* $w \mapsto j'$ was illegal before adding the new constraint. (This reminds us of resolution for CNF formulas.)

On the other hand, let $\chi$ be a coloring on $V \setminus \{v\}$ that respects all constraints including the newly added ones. Suppose, we cannot extend this coloring to $\chi(v) := i$ because of a constraint $\{(u, \chi(u)), (v, i)\}$ and the same for $\chi(v) := j$ because of a constraint $\{(w, \chi(w)), (v, j)\}$. But then $\chi$ violates the newly introduced constraint $\{(u, \chi(u)), (w, \chi(w))\}$; contradiction.

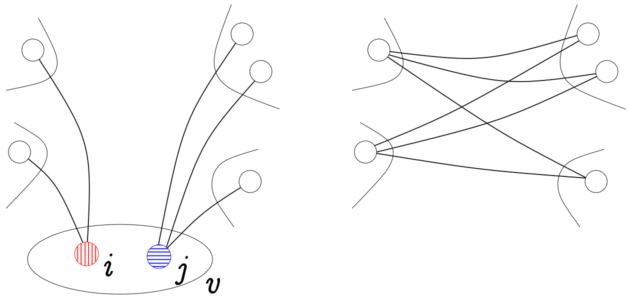



*Figure 8.3: Removal of a variable with two possible colors.*

That is, we can add constraints as indicated above, and then remove $v$ and all constraints involving $v$ without altering satisfiability. In order to obtain a $(\leq \ell, 2)$-CSP, we have to remove 1-constraints $\{(u, i')\}$ that we might have added in the process ($\{(u, i'), (v, i)\} \in S$ and $\{(u, i'), (v, j)\} \in S$ yields constraint $\{(u, i')\}$). But this can be overcome by removing color $i'$ from $L_u$ and then removing the constraint. $\square$

Note that we do not have to worry about an explosion of the constraint set, since its size is bounded by $\ell^2 \binom{n}{2}$.

**Lemma 8.2** *$n \in \mathbb{N}, n \geq 2$. There is a randomized polynomial time procedure, that—given a $(3, 2)$-CSP $S$ over $n$ variables—produces a $(3, 2)$-CSP $S'$ over at most $n - 2$ variables such that (i) if $S'$ is satisfiable, so is $S$, and (ii) if $S$ is satisfiable, so is $S'$ with probability at least $\frac{1}{2}$.*

*Proof.* Choose a constraint $\{(v, i_1), (w, j_1)\}$ (if there is none, the problem is trivial), and let $\{i_1, i_2, i_3\}$ and $\{j_1, j_2, j_3\}$ be the color lists for $v$ and $w$, respectively. Now randomly restrict the color lists of $v$ and $w$ to subsets $L''_v$ and $L''_w$, respectively, u.a.r. among the four possibilities where either $L''_v = \{i_2, i_3\}$ or $L''_w = \{j_2, j_3\}$, but not both; this gives a new $(\leq 3, 2)$-CSP $S''$.

Clearly, if $S''$ is satisfiable then $S$ is satisfiable. Now suppose $S$ is satisfiable with a coloring with $v \mapsto i$ and $w \mapsto j$; hence, $(i, j) \neq (i_1, j_1)$. By our choice of $L''_v$ and $L''_w$, the probability of $(i, j) \in L''_v \times L''_w$ is $\frac{1}{2}$, see Figure 8.4. If this event materializes, $S''$ is satisfiable as well. Now, since

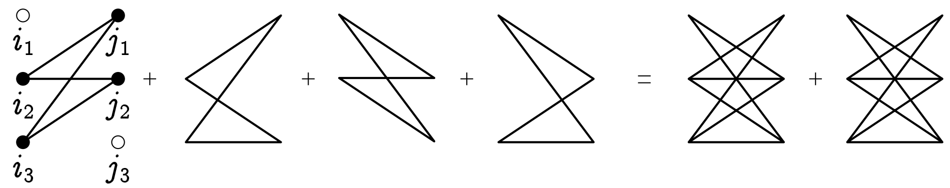



*Figure 8.4: The four sets $\{i_2, i_3\} \times \{j_1, j_2\}$, $\{i_2, i_3\} \times \{j_1, j_3\}$, $\{i_1, i_2\} \times \{j_2, j_3\}$, and $\{i_1, i_3\} \times \{j_2, j_3\}$ cover each pair in $(\{i_1, i_2, i_3\} \times \{j_1, j_2, j_3\}) \setminus \{(i_1, j_1)\}$ exactly twice.*

$S''$ has two vertices with two allowable colors only, we can transform $S''$ into a $(3,2)$-CSP $S'$ with at least two fewer vertices (via Lemma 8.1), so that $S''$ is satisfiable iff $S'$ is satisfiable. ◻

For solving a $(3,2)$-CSP $S$ with $n$ variables we apply Lemma 8.2 to reduce the number of variables by at least 2 for at most $\left\lfloor \frac{n}{2} \right\rfloor$ rounds until we reach a CSP $\hat{S}$ with at most 1 variable; now satisfiability can be solved trivially. If $\hat{S}$ is satisfiable, we know that $S$ is satisfiable, and we are done (and, in fact, a satisfying coloring of $S$ can be reconstructed in polynomial time). If $S$ is satisfiable, then $\hat{S}$ is satisfiable with probability at least $\left(\frac{1}{2}\right)^{\lfloor n/2 \rfloor}$. The usual arguments entail now

**Theorem 8.3** $\lambda \in \mathbb{R}^+$. *There is a randomized algorithm with running time $O(\lambda\sqrt{2}^n \mathrm{poly}(n))$ that either determines satisfiability of a $(3,2)$-CSP over $n$ variables, or concludes unsatisfiability with error probability at most $\mathrm{e}^{-\lambda}$.*

**Corollary 8.4** $\lambda \in \mathbb{R}^+$. *There is a randomized algorithm with running time $O(\lambda\sqrt{2}^n \mathrm{poly}(n))$ that either determines proper 3-colorability of an $n$-vertex graph, or concludes that no such coloring exists with error probability at most $\mathrm{e}^{-\lambda}$.*

Note 8.1 The best bound known for $(3,2)$-CSP is $O(1.365^n)$, the best for proper 3-coloring of graphs is $O(1.329^n)$, both from [Eppstein, 2001].

# Chapter 9

# Random $k$-CNF Formulas

$k, n, m \in \mathbb{N}$, $n \geq k$. A *random $k$-CNF formula with $m$ clauses over $n$ variables, random CNF-formula $F_k(n, m)$ for short,* is a CNF formula $\{C_1, C_2, \ldots, C_m\}$ where each $C_i$ is drawn u.a.r. from all $k$-clauses over some (common) set of $n$ variables. Note that we do not require the $C_i$'s to be pairwise distinct, but we insist on all clauses having exactly $k$ literals, no two from the same variable.

For quite some time now there has been active research investigating the probability of such a random CNF-formula to be satisfiable (in dependence of the parameters $k$, $n$ and $m$). It is conjectured that there is a *sharp threshold* in the following sense: For each $k \geq 2$ there is a constant $r_k$ such that for $k$ and $c < r_k$ constant, the probability of satisfiability tends to 1 as $n$ goes to $\infty$ if $m < cn$, while for $c > r_k$ constant it tends to 0 if $m > cn$. So far the answer is known for $k = 2$ only. One reason for interest in the matter is that around the transition from satisfiable to unsatisfiable, one apparently gets algorithmically hard CNF formulas: Satisfying assignments are hard to find, and evidence for unsatisfiability is hard to provide.

## 9.1 First Linear Bounds

Let $o(1)$ stand for a function that tends to 0 as $n$ tends to infinity. First we show that at most a linear number of clauses is allowed if we want a "fair" chance of satisfiability.

**Theorem 9.1 ([Franco, Paull, 1983])** *$k \in \mathbb{N}$, $(\ln 2) \cdot 2^k < c \in \mathbb{R}$, both constant, and $n \in \mathbb{N}$. A random CNF formula $F_k(n,m)$ with $m \geq cn$ is unsatisfiable with probability $1 - o(1)$.*

*Proof.* Let the random formula $F$ be over variable set $V$. For an assignment $\alpha \in \{0,1\}^V$, the probability that a random $k$-clause over $V$ (u.a.r. from all such $k$-clauses) is satisfied is $(1-2^{-k})$. To see this we sample first a set of variables u.a.r. in $\binom{V}{k}$, and then we decide for each variable whether it appears as a positive or negative literal. The probability of always choosing the sign which is not satisfied by $\alpha$ is $2^{-k}$.

It follows that the probability of $\alpha$ satisfying $F$ is $(1-2^{-k})^m$, and thus the expected number of satisfying assignments of $F$ is

$$\begin{aligned} \mathrm{E}(|\mathsf{sat}_V(F)|) &= 2^n(1-2^{-k})^m \leq 2^n(1-2^{-k})^{cn} \\ &< 2^n\,\mathrm{e}^{-2^{-k}cn} = 2^n\,\mathrm{e}^{-(\ln 2)n}\,\mathrm{e}^{-2^{-k}c'n} = \mathrm{e}^{-2^{-k}c'n} \end{aligned}$$

for $c' = c - (\ln 2)2^k$. For $c$ a constant as presumed in the assertion of the theorem, we have $c' > 0$ and the expected number of satisfying assignments tends to 0 as $n$ tends to infinity. Thus, the probability of existence[1] of a satisfying assignment tends to 0. ◻

If $c < (ln 2) \cdot 2^k$, then we expect many (an exponential number of) satisfying assignments. Can we thus conclude that there is a good chance that there is one at all? Not so in general, and it is instructive to investigate the matter on the otherwise not so interesting case of random 1-CNF formulas.

A random CNF formula $F := F_1(n,m)$ expects $2^n 2^{-m}$ satisfying assignments (so this is 1 for $m = n$).

We take a closer look at the process. Let us sample $F$ by first choosing a sequence of variables (of length $m$), and then we choose for each element in the sequence a sign which gives the 1-clauses. The formula is satisfiable if and only if each variable gets consistently the same sign for all of its occurrences. If $N'$ (a random variable) denotes the number of distinct variables occurring in the sequence, then

$$\Pr(F \text{ is satisfiable} \mid N' = n') = \left(\frac{1}{2}\right)^{m-n'}$$

[1]For $X$ a random variable with $X \in \mathbb{N}_0$, we have $\Pr(X \geq 1) = \sum_{i=1}^{\infty} \Pr(X = i) \leq \sum_{i=1}^{\infty} i \cdot \Pr(X = i) = \mathrm{E}(X)$.

and

$$\mathrm{E}(|\mathrm{sat}(F)\,|\,F \text{ is satisfiable and } N' = n'|) = 2^{n-n'}$$

But[2]

$$\mathrm{E}(N') = n\left(1 - \left(1 - \frac{1}{n}\right)^m\right) \le n\left(1 - \mathrm{e}^{-m/(n-1)}\right) = n(1 - \mathrm{e}^{-\lambda})$$

if $m = \lambda(n-1)$. This follows since the probability of a variable $x$ *not* to occur is $\left(1 - \frac{1}{n}\right)^m$.

That is, for $m = n$, on the average we have $N' \approx \left(1 - \frac{1}{\mathrm{e}}\right) n$, for which the probability of satisfiability is $2^{-n/e}$, which[3] is small but compensated by a huge supply of satisfying assignments in the rare case of satisfiability.

We will show now that—after all—starting $k = 2$ a linear number of clauses (with the right constant in linearity) gives a satisfiable formula almost surely.

**Theorem 9.2 ([Chvátal, Reed, 1992])** $1 > c \in \mathbb{R}^+$, *constant, and* $n \in \mathbb{N}$. *A random CNF formula* $F_2(n, m)$ *with* $m \le cn$ *is satisfiable with probability* $1 - o(1)$.

*Proof.* We will define so-called bicycles (which are certain sequences of clauses) and show that (i) absence of bicycles implies satisfiability and (ii) a random formula with at most $cn$ 2-clauses contains a bicycle with probability $o(1)$, if $c < 1$.

An *s-bicycle*, $s \ge 2$, is a sequence of $s + 1$ clauses

$$\{v, u_1\}, \{\overline{u_1}, u_2\}, \{\overline{u_2}, u_3\}, \ldots, , \{\overline{u_{s-1}}, u_s\}, \{\overline{u_s}, w\} \tag{9.1}$$

where $(u_1, u_2, \ldots, u_s)$ is a sequence of literals that come from $s$ distinct variables, and $\mathrm{vbl}(\{v\}) \subseteq \mathrm{vbl}(\{u_2, u_3, \ldots, u_s\})$, $\mathrm{vbl}(\{w\}) \subseteq \mathrm{vbl}(\{u_1, u_2, \ldots, u_{s-1}\})$.

With reference to the representation of a 2-CNF formula over $V$ as a directed graph on vertex set $V \cup \overline{V}$ in Section 3.3, we observe that a bicycle appears as directed walk

$$\overline{v} \to u_1 \to u_2 \to \ldots \to u_s \to w\ .$$

[2] We use here that $(1 - \frac{1}{n})^{n-1} > \mathrm{e}^{-1}$ for all $2 \le n \in \mathbb{N}$, which is equivalent to $(1 + \frac{1}{n-1})^{n-1} < \mathrm{e}$.

[3] An analysis of $\Pr(F \text{ is satisfiable})$ and $\mathrm{E}(|\mathrm{sat}(F)|\,|\,F \text{ is satisfiable})$ is not yet obtained. We leave this as an exercise.

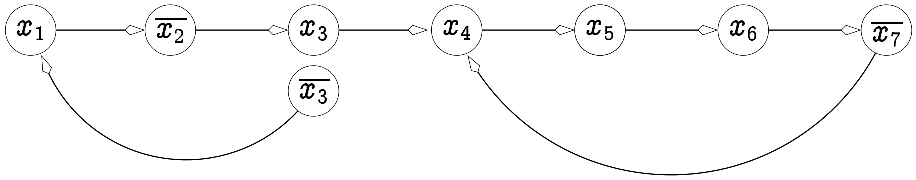



*Figure 9.1: The appearance of a bicycle in the directed graph (as defined in Section 3.3) of a 2-CNF formula.*

We recall that a 2-CNF formula is unsatisfiable if and only if $x \rightsquigarrow \overline{x} \rightsquigarrow x$ for some variable $x$, see Lemma 3.7. That is, there is a directed walk

$$x \to v_1 \to v_2 \to \ldots \to v_t \to x \qquad \text{with } \overline{x} \in \{v_2, v_3, \ldots, v_{t-1}\}. \tag{9.2}$$

Some part of this walk has to be a bicycle, so claim (i) is established.

For showing (ii), we first observe that the number of bicycles equals

$$\binom{n}{s} 2^s s! (2(s-1))^2 < n^s 2^s (2s)^2$$

(first we choose the set of variables for $(u_1, u_2, \ldots, u_s)$, then we decide upon their signs, then on their order and finally we select $v$ among the $2(s-1)$ literals at disposal, and the same for $w$).

The probability that a specific $s$-bicycle occurs is at most

$$\binom{m}{s+1}(s+1)! \left(\frac{1}{4\binom{n}{2}}\right)^{s+1} < m^{s+1} \left(\frac{1}{2n(n-1)}\right)^{s+1} .$$

For a slow development of this bound, let us denote by $(C_1, C_2, \ldots, C_m)$ the sequence of clauses generated for $F$. We define an occurrence of an $s$-bicycle (as given in (9.1)) as a sequence $\sigma = (i_1, i_2, \ldots, i_{s+1})$ of distinct integers in $\{1..m\}$ such that

$$C_{i_1} = \{v, u_1\}, C_{i_2} = \{\overline{u_1}, u_2\}, \ldots, C_{i_{s+1}} = \{\overline{u_s}, w\} .$$

There are $\binom{m}{s+1}(s+1)!$ sequences as $\sigma$, and each one is an occurrence of a specific $s$-bicycle with probability $\left(4\binom{n}{2}\right)^{-(s+1)}$. Therefore, the left hand side in (9.2) is exactly the expected number of occurrences of a specific $s$-bicycle, and the probability of it occurring is upper bounded by this expectation.

Consequently, the probability that a bicycle occurs is at most

$$\begin{aligned}\sum_{s=2}^{n} n^s 2^s (2s)^2 m^{s+1} \left(\frac{1}{2n(n-1)}\right)^{s+1} &= \frac{2m}{n(n-1)} \sum_{s=2}^{n} s^2 \left(\frac{m}{n-1}\right)^s \\ &= O\left(\frac{1}{n}\right) \ , \ \text{provided } c < 1.\end{aligned}$$

Summing up, we have shown that for a random 2-CNF formula $F$ with at most $cn$ clauses over $n$ variables

$$\Pr(F \text{ is unsatisfiable}) \leq \Pr(F \text{ has a bicycle}) = o(1) \ , \quad \text{provided } c < 1.$$

□

The theorem readily implies that the same is true for all $k \geq 2$. For $k = 2$ one can even show that the bound of 1 is tight, that is, a random 2-CNF formula with at least $cn$ clauses is unsatisfiable with probability at least $1 - o(1)$, provided $c > 1$; [Chvátal, Reed, 1992], but see also [Goerdt, 1999] and [Fernandez de la Vega, 92] for independent proofs.

## 9.2 Improved Upper Bound for 3-CNF

Theorem 9.1 shows that a random 3-CNF formula with at least $5.19\,n$ clauses is almost surely unsatisfiable. Next we want to improve on this by demonstrating that is the case already for a smaller constant.

We have to intervene with the sudden flood of satisfying assignments in case of satisfiability, as we observed it in the previous section. We will do so by carefully selecting a usually much smaller subset of it. Given a CNF formula $F$ over $V$, we call a satisfying assignment maximal if flipping any of its 0's to 1 leads to an assignment that does not satisfy. The set of maximal satisfying assignments on $V$ is denoted by $\widehat{\mathsf{sat}}_V(F)$. Note that $\widehat{\mathsf{sat}}_V(F) \subseteq \mathsf{sat}_V(F)$ and $\widehat{\mathsf{sat}}_V(F) \neq \emptyset$ iff $\mathsf{sat}_V(F) \neq \emptyset$.[4]

[4] Indeed, for a 1-CNF formula $|\widehat{\mathsf{sat}}_V(F)| \in \{0,1\}$ and therefore $\Pr(F \text{ is satisfiable}) = \mathrm{E}\left(|\widehat{\mathsf{sat}}_V(F)|\right)$.

Let $c \in \mathbb{R}^+$. For $n \in \mathbb{N}$ we consider a random CNF formula $F_3(n, m)$ over $V$ with $m \geq cn$ clauses. Our goal is to find an upper bound on $\mathrm{E}(|\widehat{\mathsf{sat}}(F)|)$ (we skip the subindex $V$ in $\widehat{\mathsf{sat}}_V()$ for the time being).

As a first step we analyze for $\alpha = \{1\}^V$ and $\ell \leq n$ distinct assignments $\alpha_i$, $i \in \{1..\ell\}$, with $d_H(\alpha_i, \alpha)$ the probability

$$\begin{aligned}
&\Pr\left(\alpha \in \mathsf{sat}(F) \wedge \bigwedge_{i=1}^{\ell} \alpha_i \notin \mathsf{sat}(F)\right) \\
= \quad &\Pr\left(\bigwedge_{i=1}^{\ell} \alpha_i \notin \mathsf{sat}(F) \,|\, \alpha \in \mathsf{sat}(F)\right) \cdot \underbrace{\Pr\left(\alpha \in \mathsf{sat}(F)\right)}_{(7/8)^m}
\end{aligned}$$

which is the same as $\Pr(\alpha \in \widehat{\mathsf{sat}}(F))$ for an assignment with $\ell$ zeros.

Let us try $\Pr(\alpha_1 \notin \mathsf{sat}(F) \,|\, \alpha \in \mathsf{sat}(F))$ first. We generate the $m$ clauses as ordered tuples of literals. This time we will first choose for each clause the sign pattern u.a.r. in $\{+,-\}^3 \setminus \{- \; - \; -\}$, i.e. among all sign patterns that ensure that $\alpha$ satisfies the clause. Then we choose a variable for each of the three positions.

Now for $\alpha_1$ not to satisfy the clause, (i) there has to be a unique $+$ in the sign pattern and (ii) this $+$-sign must be associated with the variable $x$ with $\alpha(x) = 0$. (i) happens with probability $\frac{3}{7}$, and conditioned on this, (ii) happens with probability $\frac{1}{n}$. That is,

$$\Pr\left(\alpha_1 \notin \mathsf{sat}(F) \,|\, \alpha \in \mathsf{sat}(F)\right) = 1 - \left(1 - \frac{3}{7n}\right)^m$$

In order to continue we need a lemma.

**Lemma 9.3** *$m \in \mathbb{N}$ and $S$ a finite set. Let $X_i \in S$, $i \in \{1..m\}$, be i.i.d.*[5] *random variables with $\Pr(X_i = a) = p_a$ for $a \in S$, and let $\mathcal{X} := \{X_i\}_{i=1}^m$. For every subset $T$ of $S$,*

$$\Pr\left(\bigwedge_{a \in T} (a \in \mathcal{X})\right) \leq \prod_{a \in T} \Pr(a \in \mathcal{X}) = \prod_{a \in T} \left(1 - (1 - p_a)^m\right) \ .$$

[5] identically and independently distributed

*Proof (intuition).* $\Pr(a \notin \mathcal{X}) = (1-p_a)^m$, since the $X_i$'s are independent. Hence, the equality to the right follows.

However, the events $a \in \mathcal{X}$ are not independent. In fact, if $m < |T|$, the probability to the left is zero, since then $m$ trials do not allow to include $|T|$ elements in $\mathcal{X}$. It turns out that the events $a \in \mathcal{X}$ have the so-called "negative dependency" property which allows to conclude the inequality above. Intuitively speaking, the fact that we know already $a \in \mathcal{X}$ can only decrease the chances for $b \in \mathcal{X}$ for $b \neq a$.

Perhaps surprisingly, the proof is non-trivial, still elementary, no worry, but beyond the scope of this course (see [Dubhashi, Ranjan, 1996]). ⊡

To employ the lemma we consider random variables $X_i \in \{0..\ell\}$, $i \in \{1..m\}$, one for each clause in the sequence produced in the random process for generating $F$. We let $X_i = j$ if the variable $x$ with $\alpha_j(x) = 0$ appears as the unique positive literal in the $i$th clause, and $X_i = 0$, otherwise. The scenario is set for

$$\begin{aligned}\Pr\left(\bigwedge_{i=1}^{\ell} \alpha_i \notin \mathsf{sat}(F) \,|\, \alpha \in \mathsf{sat}(F)\right) &\leq \left(1-\left(1-\frac{3}{7n}\right)^m\right)^\ell \\ &\leq \left(1-\mathrm{e}^{-3c/7}+o(1)\right)^\ell\end{aligned}$$

for $m > cn$, $c \in \mathbb{R}^+$. Now

$$\begin{aligned}\mathrm{E}(|\widehat{\mathsf{sat}}(F)|) &\leq \left(\frac{7}{8}\right)^{cn} \sum_{\ell=0}^{n} \binom{n}{\ell} \left(1-\mathrm{e}^{-3c/7}+o(1)\right)^\ell \\ &= \left(\frac{7}{8}\right)^{cn} \left(2-\mathrm{e}^{-3c/7}+o(1)\right)^n\end{aligned}$$

which tends to zero for $n$ growing, as long as $c$ satisfies

$$\left(\frac{7}{8}\right)^c \left(2-\mathrm{e}^{-3c/7}\right) < 1 \;;$$

this holds for $c \geq 4.667$.

**Theorem 9.4 ([Kirousis et al., 1998])** $n \in \mathbb{N}$. *A random CNF formula* $F_3(n,m)$ *with* $m \geq 4.667\,n$ *is unsatisfiable with probability* $1 - o(1)$.

NOTE **9.1** Currently known bounds say that a random CNF formula $F_3(n, m)$ with $m \leq 3.42\,n$ is satisfiable , and with $m \geq 4.506n$ is unsatisfiable with probability $1 - o(1)$, [Kaporis *et al.*, 2002] and [Dubois *et al.*, 2000], respectively; consult also the surveys [Franco, 2001], [Dubois, 2001], and [Clote, Kranakis, 2002, Chapter 4]. There are recent developments for general $k$-CNF formulas in [Friedgut, 1999] and [Achlioptas, Peres, 2002].

# Bibliography


[Alt *et al.*, 1996] Helmut Alt, Leonidas J. Guibas, Kurt Mehlhorn, Richard M. Karp, Avi Widgerson: A method for obtaining randomized algorithms with small tail probabilities, *Algorithmica* **16:4-5** (1996) 543–547.

[Alon, Spencer, 1992] Noga Alon, Joel H. Spencer: *The Probabilistic Method*, John Wiley and Sons, New York (1992).

[Aspvall *et al.*, 1979] Bengt Aspval, Michael F. Plass, Robert Endre Tarjan: A linear-time algorithm for testing the truth of certain quantified boolean formulas, *Information Processing Letters* **8:3** (1979) 121–123.

[Achlioptas, 2001] Dimitris Achlioptas: Lower bounds for random 3-SAT via differential equations, *Theoretical Computer Science* **265** (2001) 159–185.

[Achlioptas, Moore, 2002] Dimitris Achlioptas, Cristopher Moore: The asymptotic order of the random $k$-SAT threshold, *Proc. 43th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2002) 779–788.

[Achlioptas, Peres, 2002] Dimitris Achlioptas, Yuval Peres: The threshold for random $k$-SAT is $2^k(\ln 2 + o(1))$, manuscript (2002).

[Beck, 1991] József Beck: An algorithmic approach to the Lovász Local Lemma, I., *Random Structures and Algorithms* **2:4** (1991) 343–365.

[Beigel, Eppstein, 1995] Richard Beigel, David Eppstein: 3-Coloring in time $O(1.3446^n)$: A no MIS algorithm, *Proc. 36th Annual IEEE*

*Symposium on Foundations of Computer Science (FOCS)* (1995) 444–453.

[Bibel, 1987] WOLFGANG BIBEL: *Automated Theorem Proving*, Vieweg, Braunschweig (1987).

[Ben-Sasson, Widgerson, 2001] ELI BEN-SASSON, AVI WIDGERSON: Short proofs are narrow—resolution made simple, *Journal of the Association for Computing Machinery* **48:2** (2001) 149–169.

[Birnbaum, Lozinskii, 1999] ELAZAR BIRNBAUM, ELIEZER L. LOZINSKII: The good old Davis-Putnam procedure helps counting models, *Journal of Artificial Intelligence Research* **10** (1999) 457–477.

[Blake, 1937] A. BLAKE: *Canonical Expressions in Boolean Algebra*, Ph.D. dissertation, University of Chicago, Chicago, Illinois (1937). (Reviewed in *Journal of Symbolic Logic* **3** (1938) 93–93.)

[Bläser, Manthey, 2002] MARKUS BLÄSER, BODO MANTHEY: Improved approximation algorithms for Max-2SAT with cardinality constraints, *Proc. 13th International Symposium on Algorithms and Computation (ISAAC)*, *Lecture Notes in Computer Science* **2518** (2002) 187–198.

[Bollobás, 1998] BÉLA BOLLOBÁS: *Modern Graph Theory*, Graduate Texts in Mathematics **184**, Springer-Verlag New York Inc. (1998).

[Bollobás, Brightwell, 2001] BÉLA BOLLOBÁS, GRAHAM BRIGHTWELL: The number of $k$-SAT functions, manuscript (2001).

[Bollobás *et al.*, 2003] BÉLA BOLLOBÁS, GRAHAM BRIGHTWELL, IMRE LEADER: The number of 2-SAT functions, *Israel Journal of Mathematics* **133** (2003) 45–60.

[Boole, 1854] GEORGE BOOLE: *An Investigation of the Laws of Thought on which are Founded the Mathematical Theories of Logic and Probabilities*, Dover Publications (1854, reprinted 1973).

[Chen, Kanj, 2002] JIANER CHEN, IYAD A. KANJ: Improved exact algorithms for MAX-SAT, *Proc. 5th Latin American Symposium on Theoretical Informatics (LATIN)*, *Lecture Notes in Computer Science* **2286** (2002) 341–355.

[Chvátal, Reed, 1992] VAŠEK CHVÁTAL, BRUCE REED: Mick gets some (the odds are on his side), *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1992) 620–627.

[Chvátal, Szemérédi, 1988] VAŠEK CHVÁTAL, ENDRE SZEMÉREDI: Many hard examples for resolution, *Journal of the Association for Computing Machinery* **35:4** (1988) 759–768.

[Clote, Kranakis, 2002] PETER CLOTE, EVANGELOS KRANAKIS: *Boolean Functions and Computation Models*, Texts in Theoretical Computer Science, An EATCS Series, Spinger Verlag, Berlin (2002).

[Cohen *et al.*, 1997] GÉRARD COHEN, IIRO HONKALA, SIMON LITSYN, ANTOINE LOBSTEIN: *Covering Codes*, North-Holland, Amsterdam (1997).

[Cook, 1971] STEPHEN A. COOK: The complexity of theorem-proving procedures, *Proc. 3rd Annual ACM Symposium on Theory of Computation (STOC)* (1971) 151–158.

[Cormen *et al.*, 1990] THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST: *Introduction to Algorithms*, MIT Press, Cambridge (1990).

[Creignou, Hermann, 1996] NADIA CREIGNOU, M. HERMANN: Complexity of generalized satisfiability counting problems, *Information and Computation* **125** (1996) 1–12.

[Creignou *et al.*, 2001] NADIA CREIGNOU, SANJEEV KHANNA, MADHU SUDHAN: *Complexity Classifications of Boolean Constrained Satisfaction Problems*, SIAM Monographs on Discrete Mathematics and Applications, SIAM (2001).

[Courcelle *et al.*, 2001] BRUNO COURCELLE, JOHANN A. MAKOWSKY, UDI ROTICS: On the fixed parameter complexity of graph enumeration problems definable in monadic second order logic, *Discrete Applied Mathematics* **108:1-2** (2001) 23–52.

[Dahllöf, Jonsson, 2002] VILHELM DAHLLÖF, PETER JONSSON: An algorithm for counting maximum weighted independent sets and its applications, *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2002) 292–298.

[Dahllöf *et al.*, 2002] VILHELM DAHLLÖF, PETER , MAGNUS WAHLSTRÖM:

2-SAT and 3-SAT, *Proc. 8th Annual International Conference on Computing and Combinatorics (COCOON)*, *Lecture Notes in Computer Science* **2387** (2002) 535–543.

[Dantsin, 1981] EVGENY DANTSIN: Two propositional proof systems based on the splitting method (in Russian), *Zapiski Nauchnykh Seminarov LOMI* **105** (1981) 24–44. Also (English translation): *Journal of Soviet Mathematics* **22:3** (1983) 1293–1305.

[Dantsin *et al.*, 2000] EVGENY DANTSIN, ANDREAS GOERDT, EDWARD A. HIRSCH, UWE SCHÖNING: Deterministic algorithms for $k$-SAT based on covering codes and local search, *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP)*, *Lecture Notes in Computer Science* **1853** (2000) 236–247.

[Dantsin *et al.*, 2001] EVGENY DANTSIN, EDWARD A. HIRSCH, SERGEI IVANOV, MAXIM VSEMIRNOV: Algorithms for SAT and upper bounds on their complexity, *Electronic Colloquium on Computational Complexity*, Report No. 12 (2001) 19 pages.

[Dantsin *et al.*, 2002] EVGENY DANTSIN, ANDREAS GOERDT, EDWARD A. HIRSCH, , R. KANNAN, J. KLEINBERG, CHRISTOS H. PAPADIMITRIOU, UWE SCHÖNING: A deterministic $(2 - 2/(k+1))^n$ algorithm for $k$-SAT based on local search, *Theoretical Computer Science* **289:1** (2002) 69–83.

[Dantsin *et al.*, 2003] EVGENY DANTSIN, EDWARD A. HIRSCH, ALEXANDER WOLPERT: Algorithms for SAT based on search in Hamming balls, *Electronic Colloquium on Computational Complexity*, Report No. 72 (2003) 11 pages.

[Dantsin, Wolpert, 2004] EVGENY DANTSIN, ALEXANDER WOLPERT: Derandomization of Schuler's algorithm for SAT, *Electronic Colloquium on Computational Complexity*, Report No. 17 (2004) 6 pages.

[Davis, Putnam, 1960] MARTIN DAVID, HILARY PUTNAM: A computing procedure for quantification theory, *Journal of the Association for Computing Machinery* **7**:**3** (1960) 201–215.

[De Ita, Morales, 1997] GUILLERMO DE ITA, GUILLERMO MORALES: $\#2,2$-SAT is solvable in linear-time, manuscript (1997).

[Diestel, 2000] REINHARD DIESTEL: *Graph Theory*, Graduate Texts in Mathematics **173**, Springer-Verlag New York Inc. (2nd ed. 2000).

[Dubhashi, Ranjan, 1996] DEVDATT DUBHASHI, DESH RANJAN: Ball and bins: A study in negative dependence, *Basic Research in Computer Science Report Series* BRICS RS-96-25 (1996).

[Dubois, 1991] OLIVIER DUBOIS: Counting the number of solutions for instances of satisfiability, *Theoretical Computer Science* **81**:**1** (1991) 49–64.

[Dubois *et al.*, 2000] OLIVIER DUBOIS, Y. BOUFKHAD, J. MANDLER: Typical random 3-SAT formulae and the satisfiability threshold, *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2000) 124–126.

[Dubois, 2001] OLIVIER DUBOIS: Upper bounds on the unsatisfibility threshold, *Theoretical Computer Science* **265** (2001) 187–197.

[Ehrenfeucht, Karpinski, 1990] ANDRZEJ EHRENFEUCHT, MAREK KARPINSKI: The computational complexity of (XOR,AND)-counting problems, manuscript (1990).

[Eppstein, 2001] DAVID EPPSTEIN: 3-coloring in time $O(1.3289^n)$, *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2001) 329–337.

[Eppstein, 2000] DAVID EPPSTEIN: Small maximal independent sets and faster exact graph coloring, manuscript (2000).

[Eppstein, 2000a] DAVID EPPSTEIN: Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction, manuscript (2000).

[Erdős, Kleitman, 1968] PAUL ERDŐS, DANIEL J. KLEITMAN: On coloring graphs to maximize the proportion of multicolored $K$-edges, *Journal of Combinatorial Theory, Series A* **5**:**2** (1968) 164–169.

[Erdős, Lovász, 1975] PAUL ERDŐS, LASZLO LOVÁSZ: Problems and results on 3-chromatic hypergraphs and some related questions, in *Infinite and Finite Sets*, vol. **10** of Colloqia Mathematica Soc. János Bolyai, (András Hajnal, Richard Rado, Vera T. Sós, eds.), North Holland, Amsterdam, New York (1975) 609–628.

[Erdős, Selfridge, 1973] PAUL ERDŐS, JOHN L. SELFRIDGE: On a combinatorial game, *Journal of Combinatorial Theory, Series A* **14** (1973) 298–301.

[Fernandez de la Vega, 92] W. FERNANDEZ DE LA VEGA: On random 2-SAT, manuscript (1992).

[Franco, 2001] JOHN FRANCO: Results related to threshold phenomena research in satisfiability: lower bounds, *Theoretical Computer Science* **265** (2001) 147–157.

[Franco, Paull, 1983] JOHN FRANCO, M. PAULL: Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem, *Discrete Applied Mathematics* **5:1** (1983) 77–87.

[Friedgut, 1999] AHUD FRIEDGUT, (JEAN BOURGAIN): Sharp thresholds of graph properties and the $k$-SAT problem, *Journal of the American Mathematical Society* **12:4** (1999) 1017–1054.

[Formann, Wagner, 1991] MICHAEL FORMANN, FRANK WAGNER: A packing problem with applications to lettering of maps, *Proc. 7th Annual ACM Symposium on Computational Geometry (SoCG)* (1991) 281–288.

[Freeman, 1996] JON W. FREEMAN: Hard random 3-SAT problems and the Davis-Putnam procedure, *Artificial Intelligence* **81** (1996) 183–198.

[Garey, Johnson, 1974] MICHAEL R. GAREY, DAVID S. JOHNSON: Some simplified NP-complete problems, *Proc. 6th Annual ACM Symposium on Theory of Computation (STOC)* (1974) 47–63.

[Garey, Johnson, 1979] MICHAEL R. GAREY, DAVID S. JOHNSON: *Computers and Intractability – A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, New York (1979).

[Goerdt, 1999] ANDREAS GOERDT: A remark on random 2-SAT, *Discrete Applied Mathematics* **96-97** (1999) 107–110.

[Goldberg *et al.*, 1995] MARK K. GOLDBERG, THOMAS H. SPENCER, DAVID A. BERQUE: A low-exponential algorithm for counting vertex covers, *Graph Theory, Combinatorics, Algorithms, and Applications* **1** (1995) 431–444.

[Graham *et al.*, 1989] RONALD L. GRAHAM, DONALD E. KNUTH, OREN PATASHNIK: *Concrete Mathematics*, Addison-Wesley Publishing Company Inc., Boston, San Francisco, ... (1989).

[Grimmett, Stirzaker, 1992] GEOFFREY R. GRIMMET, DAVID R. STIRZAKER: *Probability and Random Processes*, Oxford University Press, Oxford (1992, 2nd Edition).

[Haken, 1985] ARMIN HAKEN: The intractability of resolution, *Theoretical Computer Science* **39** (1985) 297–308.

[Hall, 1935] PHILIP HALL: On representation of subsets, *Journal of the London Mathematical Society* **10** (1935) 26–30.

[Hammer, Rudeanu, 1968] PETER L. HAMMER (IVANESCU), SERGIU RUDEANU: *Boolean Methods in Operations Research and Related Areas*, Springer Verlag, Berlin, Heidelberg, New York (1968).

[Hansen, Jaumard, 1990] PIERRE HANSEN, BRIGITTE JAUMARD: Algorithms for maximum satisfiability, *Computing* **44** (1990) 279–303.

[Håstad, 1997] JOHAN HÅSTAD: Some optimal in-approximability results, *Proc. 29th Annual ACM Symposium on Theory of Computation (STOC)* (1997) 1–10.

[Hirsch, 2000] EDWARD A. HIRSCH: New worst-case upper bounds for SAT, *Journal of Automated Reasoning* **24:4** (2000) 397–420.

[Hirsch, Kojevnikov, 2002] EDWARD A. HIRSCH, ARIST KOJEVNIKOV: UnitWalk: A new SAT solver that uses local search guided by unit clause elimination, *Proc. 5th International Symposium on the Theory and Applications of Satisfiability Testing (SAT)* (2002) 35–42.

[Hofmeister, 2003] THOMAS HOFMEISTER: An approximation algorithm for MAX-2-SAT with cardinality constraints, *Proc. 11th Annual European Symposium on Algorithms (ESA)*, *Lecture Notes in Computer Science* **2832** (2003) 301–312.

[Hofmeister *et al.*, 2002] THOMAS HOFMEISTER, UWE SCHÖNING, RAINER SCHULER, OSAMU WATANABE: A probabilistic 3-SAT algorithm further improved, *Proc. 19th Symposium on Theoretical Aspects of Computer Science (STACS)*, *Lecture Notes in Computer Science* **2285** (2002) 192–202.

[Huang, Lieberherr, 1985] MING-DEH A. HUANG, KARL J. LIEBERHERR: Implications if forbidden structures for extremal algorithmic problems, *Theoretical Computer Science* **40** (1985) 1995–210.

[Itai, Makowsky, 1987] ALON ITAI, JOHANN A. MAKOWSKY: Unification as complexity measure for logic programming, *Journal of Logic Programming* **4:2** (1987) 105–117.

[Iwama, Tamaki, 2003] KAZUO IWAMA, SUGURU TAMAKI: Improved upper bounds for 3-SAT, *Electronic Colloquium on Computational Complexity*, Report No. 53 (2003) 3 pages.

[Jagger, Richards, 1965] MICK JAGGER, KEITH RICHARDS: (I can't get no) satisfaction, *Out of our Heads*, Rolling Stones record label (1965).

[Johnson, 1974] DAVID S. JOHNSON: Approximation algorithms for combinatorial problems, *Journal of Computer System Sciences* **9** (1974) 256–278.

[Kaporis *et al.*, 2002] A. KAPORIS, LEFTERIS M. KIROUSIS, E.G. LALAS: The probabilistic analysis of a greedy satisfiability algorithm, *Proc. 10th Annual European Symposium on Algorithms (ESA)*, *Lecture Notes in Computer Science* **2461** (2002) 574–585.

[Karp, 1972] RICHARD M. KARP: Reducibility among combinatorial problems, in: *Complexity of Computer Computations*, (Raymond E. Miller, James W. Thatcher, eds.), Plenum Press, New York-London (1972) 85–104.

[Khanna *et al.*, 1997] SANJEEV KHANNA, MADHU SUDAN, DAVID P. WILLIAMSON: A complete classification of the approximability of maximation problems derived from boolean constraint satisfaction, *Proc. 29th Annual ACM Symposium on Theory of Computation (STOC)* (1997) 11–20.

[Khanna *et al.*, 2001] SANJEEV KHANNA, MADHU SUDAN, LUCA TREVISAN, DAVID P. WILLIAMSON: The approximability of constraint satisfaction problems, *SIAM Journal on Computing* **30:6** (2001) 1863–1920.

[Kirousis *et al.*, 1998] LEFTERIS M. KIROUSIS, EVANGELOS KRANAKIS, DANNY KRIZANC, YANNIS C. STAMATIOU: Approximating the unsatisfiability threshold of random formulas, *Random Structures and Algorithms* **12:3** (1998) 253–269.

[Kullmann, 1999] OLIVER KULLMANN: New methods for 3-SAT decision and worst case analysis, *Theoretical Computer Science* **223:1–2** (1999) 1–72.

[Lawler, 1976] EUGENE L. LAWLER: A note on the complexity of the chromatic number problem, *Information Processing Letters* **5:3** (1976) 66–67.

[Levin, 1973] LEONID A. LEVIN: Universal sorting problems, *Problemy Peredaci Informacii* **9** (1973) 115–116. Also (English translation): *Problems of Information Transmission* **9** (1973) 265–266.)

[Lewis, Papadimitriou, 1998] HARRY R. LEWIS, CHRISTOS H. PAPADIMITRIOU: *Elements of the Theory of Computation*, Prentice Hall (1998).

[Lieberherr, Specker, 1981] KARL J. LIEBERHERR, ERNST SPECKER: Complexity of partial satisfaction, *Journal of the Association for Computing Machinery* **28:2** (1981) 411–421. Also: *Proc. 20th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1979) 132–139.

[Lieberherr, Specker, 1982] KARL J. LIEBERHERR, ERNST SPECKER: Complexity of partial satisfaction, II, Technical Report 293, Dept. of EECS, Princeton University (1982).

[Lovász, 1975] LÁSZLO LOVÁSZ: On the ratio of optimal integral and fractional covers, *Discrete Mathematics* **13** (1975) 383–390.

[Löwenheim, 1908] LEOPOLD LÖWENHEIM: Über das Auflösungsproblem im logischen Klassenkalkül, *Sitzungsberichte der Berliner Mathematischen Gesellschaft* **7** (1908) 89–94.

[Löwenheim, 1910] LEOPOLD LÖWENHEIM: Über die Auflösung von Gleichungen im logischen Gebietkalkül, *Mathematische Annalen* **68** (1910) 169–207.

[Löwenheim, 1913] LEOPOLD LÖWENHEIM: Über Transformationen im Gebietkalkül, *Mathematische Annalen* **73** (1913) 245–272.

[Lozinskii, 1992] E. LOZINSKI: Counting propositional models, *Information Processing Letters* **41** (1992) 327–332.

[MacWilliams, Sloane, 1977] F. JESSIE MACWILLIAMS, NEIL J.A. SLOANE: *The Theory of Error-Correcting Codes* North-Holland, Amsterdam (1977).

[Makowsky, Ravve, 2003] JOHANN A. MAKOWSKY, ELENA V. RAVVE: Counting truth assignments of formulas of bounded tree-width, manuscript (2003).

[Matoušek, Nesetril, 1998] JIRÍ MATOUŠEK, JAROSLAV NESETRIL: *Invitation to Discrete Mathematics*, Oxford University Press (1998) Also (German translation): *Diskrete Mathematik—Eine Entdeckungsreise*, Springer Verlag (2002).

[McCuaig, Shepherd, 1989] WILLIAM MCCUAIG, BRUCE SHEPHERD: Domination in graphs with minimum degree two, *Journal of Graph Theory* **13:6** (1989) 749–762.

[Monien, Speckenmeyer, 1985] BURKHARD MONIEN, EWALD SPECKENMEYER: Solving satisfiability in less than $2^n$ steps, *Discrete Applied Mathematics* **10:3** (1985) 287–295.

[Motwani, Raghavan, 1995] RAJEEV MOTWANI, PRABHAKAR RAGHAVAN: *Randomized Algorithms*, Cambridge University Press, Cambridge, (1995).

[Papadimitriou, 1991] CHRISTOS H. PAPADIMITRIOU: On selecting a satisfying truth assigment, *Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1991) 163–169.

[Paturi *et al.*, 1997] RAMAMOHAN PATURI, PAVEL PUDLÁK, FRANCIS ZANE: Satisfiability coding lemma, *Proc. 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1997) 566–574.

[Paturi *et al.*, 1998] RAMAMOHAN PATURI, PAVEL PUDLÁK, MICHAEL E. SAKS, FRANCIS ZANE: An improved exponential-time algorithm for $k$-SAT, *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1998) 628–637.

[Pudlák, Impagliazzo, 1999] PAVEL PUDLÁK, RUSSEL IMPAGLIAZZO: A lower bound for DLL algorithms for $k$-SAT, manuscript (1999).

[Reed, 1996] BRUCE REED: Paths, stars and the number three, *Combinatorics, Probability and Computing* **5** (1996) 277–295.

[Rhagavan, 1988] PRABHAKAR RHAGAVAN: Probabilistic construction of deterministic algorithms: Approximating packing integer programs, *Journal of Computer System Sciences* **37** (1988) 130–143.

[Robinson, 1965] JOHN ALAN ROBINSON: A machine-oriented logic based on the resolution principle, *Journal of the Association for Computing Machinery* **12:1** (1965) 23–41.

[Rolf, 2003] DANIEL ROLF: 3-SAT $\in$ RTIME($O(1.32793^n)$)—Improving randomized local search by initializing strings of 3-clauses, *Electronic Colloquium on Computational Complexity*, Report No. 54 (2003) 22 pages.

[Roth, 1996] DAN ROTH: On the hardness of approximate reasoning, *Artificial Intelligence* **82** (1996) 273–302.

[Schaefer, 1978] THOMAS J. SCHAEFER: The complexity of satisfiability problems, *Proc. 10th Annual ACM Symposium on Theory of Computation (STOC)* (1978) 216–226.

[Schickinger, Steger, 2001] THOMAS SCHICKINGER, ANGELIKA STEGER: *Diskrete Strukturen (Band 2), Wahrscheinlichkeitstheorie und Statistik*, Springer Verlag (2001).

[Schiermeyer, 1994] INGO SCHIERMEYER: Deciding 3-colourability in less than $O(1.415^n)$ steps, *Proc. 19th International Workshop on Graph*

*Theoretic Concepts in Computer Science (WG)*, *Lecture Notes in Computer Science* **790** (1994) 177–182.

[Schöning, 1992] UWE SCHÖNING: *Logik für Informatiker*, BI-Wissenschaftsverlag (1992).

[Schöning, 2001] UWE SCHÖNING: *Algorithmik*, Spektrum Akademischer Verlag, Heidelberg, Berlin (2001).

[Schöning, 2002] UWE SCHÖNING: A probabilistic algorithm for $k$-SAT based on limited local search and restart, *Algorithmica* **32** (2002) 615–623. Also: A probabilistic algorithm for $k$-SAT and constrained satisfaction problems, *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (1999) 410–414.

[Schuler, 2005] RAINER SCHULER: An algorithm for the satisfiability problem of formulas in conjunctive normal form, *Journal of Algorithms* (2004) to–appear.

[Sipser, 1997] MICHAEL SIPSER: *Introduction to the Theory of Computation*, PWS Publishing Company, Boston (1997).

[Spencer, 1987] JOEL H. SPENCER: *Ten Lectures on the Probabilistic Method*, CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia (1987; 2nd ed. 1994).

[Steger, 2001] ANGELIKA STEGER: *Diskrete Strukturen (Band 1), Kombinatorik–Graphentheorie–Algebra* Springer Verlag (2001).

[Trümper, 1998] KLAUS TRÜMPER: *Effective Logic Computation*, Wiley, (1998)

[Tovey, 1984] CRAIG A. TOVEY: A simplified NP-complete satisfiability problem, *Discrete Applied Mathematics* **8** (1984) 85–89.

[Urquhart, 1987] ALASDAIR URQUHART: Hard examples for resolution, *Journal of the Association for Computing Machinery* **34:1** (1987) 209–219.

[Urquhart, 1999] ALASDAIR URQUHART: The symmetry rule in propositional logic, *Discrete Applied Mathematics* **96-97** (1999) 177–193.

[Valiant, 1979] LESLIE G. VALIANT: The complexity of enumeration and reliability problems, *SIAM Journal on Computing* **8** (1979) 410–421.

[West, 2001] DOUGLAS B. WEST: *Introduction to Graph Theory*, Prentice-Hall, Inc., Upper Saddle River (2nd ed. 2001).

[Williams, 2003] RYAN WILLIAMS: On computing $k$-CNF formula properties, manuscript (2003).

[Yannakakis, 1994] MIHALIS YANNAKAKIS: On the aproximation of maximum satisfiability, *Journal of Algorithms* **17** (1994) 475–502. Also: *Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (1992) 1–9.

[H. Zhang *et al.*, 2003] HANTAO ZHANG, HAIOU SHEN, FELIP MANYÁ: Exact algorithms for MAX-SAT, *Electronic Notes in Theoretical Computer Science* **86:1** (2003) 14 pages.

[W. Zhang, 1996] WENHUI ZHANG: Number of models and satisfiability of sets of clauses, *Theoretical Computer Science* **155** (1996) 277–288.

# Appendix A

# Glossary of Notions and Facts

## A.1 Fibonacci Numbers, Golden Ratio

The sequence $(f_n)_{n\in\mathbb{N}_0}$ of Fibonacci[1] numbers is defined by

$$f_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \text{ and} \\ f_{n-1} + f_{n-2} & \text{otherwise.} \end{cases}$$

It often occurs in mathematics, and even outside of it. We know that for $n \in \mathbb{N}_0$

$$f_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right) .$$

Therefore, the limit of the ratio $\frac{f_{n+1}}{f_n}$ is $\frac{1+\sqrt{5}}{2} \approx 1.618033\ldots$, called the *golden ratio*.

## A.2 Some Recurrences

In our analyses we frequently encounter functions $g : \mathbb{N}_0 \longrightarrow \mathbb{N}_0$ observing recurrences of the form

$$g(n) \leq \begin{cases} a_n & \text{if } n < k, \text{ and} \\ a + \sum_{i=1}^{k} c_{k-i} \cdot g(n-i) & \text{otherwise.} \end{cases}$$

[1] Leonardo di Pisa, called Fibonacci, around 1175-1245 (Pisa, Italy)

Here, in general, $a$, $a_i$, and $c_i$, $0 \le i < k$ are real numbers with $c_0 \ne 0$. *For our applications we assume that these are non-negative numbers—thus* $c_0 > 0$— *and that* $\sum_{i=0}^{k-1} c_i > 1$. The goal is to derive an asymptotic upper bound for $g(n)$ from these inequalities.

First, we want to get rid of the constant term $a$. So, if $a > 0$, we multiply the inequality by $\lambda > 0$ and add 1 on both sides. Then

$$\underbrace{\lambda g(n)+1}_{f(n)} \le \begin{cases} \lambda a_n + 1 & \text{if } n < k\text{, and} \\ \lambda a + 1 + \sum_{i=1}^{k} c_{k-i} \cdot (\underbrace{\lambda g(n-i)+1}_{f(n-i)} - 1) & \text{otherwise.} \end{cases}$$

We substitute $f(n)$ for $\lambda g(n)+1$, and we choose $\lambda = \frac{\left(\sum_{i=0}^{k-1} c_i\right)-1}{a}$ (which is indeed positive by our assumptions) so that $\lambda a + 1 - \sum_{i=1}^{k} c_{k-i}$ vanishes. Then homogeneous inequalities for $f()$, $n \ge k$, evolve

$$f(n) \le \begin{cases} \lambda a_n + 1 & \text{if } n < k\text{, and} \\ \sum_{i=1}^{k} c_{k-i} \cdot f(n-i) & \text{otherwise.} \end{cases}$$

Via $g(n) = \frac{f(n)-1}{\lambda}$ any asymptotic upper bund for $f()$ implies the same asymptotic upper bound for $g()$.

If $r$ is a positive real root[2] of

$$x^k = \sum_{i=0}^{k-1} c_i x^i, \quad \text{where } c_0 > 0,\ c_i \ge 0 \text{ for } 1 \le i < k\text{, and } \textstyle\sum_{i=0}^{k-1} c_i > 1,$$

then there exists a $b > 0$ such that $f(n) \le b \cdot r^n$ for all $n \in \mathbb{N}_0$. For a proof choose $b := \max_{i=0}^{k-1} \frac{\lambda a_i + 1}{r^i}$, and then induction readily goes through.

Consequently, $f(n) = O(r^n)$ and thus $g(n) = O(r^n)$.

For obtaining the roots we can employ programs like Maple. For example, let us look at

$$t(n) \le \begin{cases} 1 & \text{if } n < 3\text{, and} \\ 1 + t(n-1) + t(n-2) + t(n-3) & \text{otherwise.} \end{cases}$$

[2] In case you wonder or care: There is exactly one such positive root which has to exceed 1. Why? The polynomial $p(x) := x^k - \sum_{i=0}^{k-1} c_i x^i$ has exactly one sign change in its coefficients, so Descartes' rule allows for one positive root at most. But $p(1) < 0$ and $p(\infty) > 0$, so there has to be a real root exceeding 1.

Then we are supposed to type in Maple

```
> evalf(solve(x^3-x^2-x-1,x));
```

which provides us with the answer

```
1.839286755, -0.4196433777+0.6062907300 I,
        -0.4196433777-0.6062907300 I
```

and so $t(n) = O(1.8393^n)$ is established.

This treatment is very much tailored to our needs. A more profound and general derivation can be found, e.g. in [Matoušek, Nesetril, 1998, Section 10.3].

## A.3 Markov Chains

Given some countable (infinite or finite) set $S$, we consider an infinite sequence of random variables

$$(X_t)_{t\in\mathbb{N}_0} = X_0, X_1, X_2, \ldots \quad \text{with } X_t \in S$$

such that $X_t$, $t \in \mathbb{N}$, depends on $X_{t-1}$ only (and not on any of the previous history $X_0, X_1, \ldots, X_{t-2}$). More formally,

$$\begin{aligned}\Pr(X_t = j \mid X_0 = j_0, X_1 = j_1, \ldots, X_{t-2} = j_{t-2}, X_{t-1} = i)\\ = \Pr(X_t = j \mid X_{t-1} = i) =: p_{tij}\end{aligned}$$

for all $t \in \mathbb{N}$ and all $(j_0, j_1, \ldots, j_{t-2}, i, j) \in S^{t+1}$. A sequence of random variables with this memorylessness property is called a *Markov*[3] *chain*. The elements in $S$ are the *states* of the Markov chain, $X_0$ either equals some *initial state* $s_0 \in S$ or, more generally, attains a state in $S$ with some given *initial probability distribution* $q^{(0)} = (q_i^{(0)})_{i\in S}$ *on* $S$ ( $\Pr(X_0 = i) = q_i^{(0)}$).

A Markov chain is *homogeneous*, if for all $t \in \mathbb{N}$,

$$\Pr(X_t = j \mid X_{t-1} = i) = \Pr(X_1 = j \mid X_0 = i) =: p_{ij} \ .$$

The $(p_{ij})$'s are called *transition probabilities* – they are real numbers that satisfy $0 \le p_{ij} \le 1$ and $\sum_{k\in S} p_{ik} = 1$ for all $i, j \in S$. (A matrix $(p_{ij})_{i,j\in S}$ with this property is called *stochastic*.)

[3]Andrei Andreyevich Markov, 1856-1922 (St. Petersburg University, Russia)

## A.4 Tail Estimates

**Lemma A.1 (Markov's Inequality)** *Let $X$ be a nonnegative random variable for which* $\mathrm{E}(X) > 0$ *exists. Then, for all* $\lambda \in \mathbb{R}^+$,

$$\Pr(X \geq \lambda\,\mathrm{E}(X)) \leq \frac{1}{\lambda}.$$

*Proof.* Let $t \in \mathbb{R}^+$.

$$\begin{aligned}
\mathrm{E}(X) &= \sum_x x \cdot \Pr(X = x) \\
&= \sum_{x<t} \underbrace{x}_{\geq 0} \cdot \Pr(X = x) + \sum_{x \geq t} \underbrace{x}_{\geq t} \cdot \Pr(X = x) \\
&\geq t \cdot \sum_{x \geq t} \Pr(X = x) \\
&= t \cdot \Pr(X \geq t)
\end{aligned}$$

That is,

$$\Pr(X \geq t) \leq \frac{\mathrm{E}(X)}{t}, \quad \text{for all } t \in \mathbb{R}^+.$$

Now set $t = \lambda\,\mathrm{E}(X)$ to conclude the statement of the lemma. ∎

## A.5 Jensen's Inequality

In order to state Jensen's[4] Inequality we have to clarify two notions. First, a *real interval* is a convex subset $I$ of $\mathbb{R}$, i.e. whenever $x, y \in I$ then $\lambda x + (1-\lambda)y \in I$ for all $\lambda \in [0,1]$.[5] Second, a real-valued function $f$ defined on a real interval $I$ is called *a convex function on* $I$, if

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$$

for all $x, y \in I$ and all $\lambda \in [0,1]$. (That is, the segment connecting any two points of the graph of $f$ in the range $I$ lies above or on the graph of $f$.)

Examples of convex functions are $x \mapsto x^2$ on $\mathbb{R}$, $x \mapsto |x|$ on $\mathbb{R}$, $x \mapsto -\ln x$ on $\mathbb{R}^+$, and $x \mapsto \sin(x)$ on $[\pi, 2\pi]$.

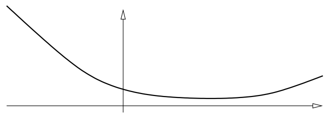


Figure A.1: A convex function.

An equivalent definition of a convex function $f$ on $I$ requires that for every $a \in I$ there exists a $\lambda_a \in \mathbb{R}$ such that

$$f(x) \geq f(a) + \lambda_a (x - a)$$

for all $x \in I$. That is, in every point of the graph of $f$ in the range of $I$ there exists a tangent to the curve that is majorized by $f$ in the range of $I$.

**Lemma A.2 (Jensen's Inequality)** *Let $f$ be a convex function on a real interval $I$. Then*

$$f\left(\sum_{i=1}^{n} \lambda_i x_i\right) \leq \sum_{i=1}^{n} \lambda_i f(x_i) \tag{A.1}$$

*for any $(x_i)_{i=1}^n \in I^n$ and any $(\lambda_i)_{i=1}^n \in (\mathbb{R}_0^+)^n$ with $\sum_{i=1}^n \lambda_i = 1$.*

*And (provided the expectations involved exist)*

$$f(\mathrm{E}(X)) \leq \mathrm{E}(f(X)) \tag{A.2}$$

*for any random variable $X$ attaining values in $I$ only.*

[4] Johan Ludwig Jensen, 1859-1925 (Denmark)

[5] Why don't we simply say that $I = [a,b]$ for $a, b \in \mathbb{R}$ with $a \leq b$? Well, we want also intervals of the form $[a,b)$, $(a,b]$ and $(a,b)$. And we want to allow infinite intervals (e.g. $\mathbb{R}$, $\mathbb{R}^+$ or $\mathbb{R}_0^+$), i.e. $a$ possibly $-\infty$ and $b$ possibly $\infty$. But then, of course, we don't want $-\infty$ or $\infty$ to occur in a real interval, i.e. $[-\infty, \infty]$ ...is outruled. Perhaps you like now the definition we chose.

If the $\lambda_i$'s in (A.1) are read as probabilities, we see that (A.1) is just a written-out special case of the probabilistic formulation (A.2).

Many useful inequalities follow from Jensen's. For example, the relation between geometric and arithmetic mean

$$\sqrt[n]{\prod_{i=1}^{n} x_i} \leq \frac{\sum_{i=1}^{n} x_i}{n} \qquad \text{for } x_i\text{'s in } \mathbb{R}^+,$$

can be rewritten as (by taking ln on both sides, with the monotonicity of ln in mind)

$$\frac{1}{n}\ln\prod_{i=1}^{n} x_i = \sum_{i=1}^{n}\frac{1}{n}\ln x_i \;\leq\; \ln\sum_{i=1}^{n}\frac{1}{n}x_i\ .$$

Now we multiply both sides with $-1$ (thus reverting the inequality) and we have Jensen's Inequality for the convex function $x \mapsto -\ln x$.

*Proof of (A.2) of Lemma A.2.* Let $\mu := \mathrm{E}(X)$ (hence $\mu \in I$). Let $\lambda$ be such that $f(x) \geq f(\mu) + \lambda(x-\mu)$ for all $x \in I$. Due to linearity of expectation

$$\mathrm{E}(f(X)) \geq \mathrm{E}(f(\mu) + \lambda(X-\mu)) = f(\mu) + \lambda(\underbrace{\mathrm{E}(X)}_{\mu} - \mu) = f(\mu) = f(\mathrm{E}(X))$$

and we are done. ◻