

BOOLEAN SATISFIABILITY— COMBINATORICS AND ALGORITHMS

EMO WELZL, ETH ZÜRICH
with Chapters by

Timon Hertli,
Robin Moser,
and,
Dominik Scheder

Version Spring 2016

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	viii
0.2	Notation	x
1	Basics, Examples, and Set-Up	1
1.1	Examples	2
1.1.1	Circuit Verification	2
1.1.2	Map Labeling	5
1.2	Conjunctive Normal Form	7
1.3	Terminology	11
1.4	Counting Satisfying Assignments	14
1.5	Resolution	19
2	Extremal Properties	23
2.1	Number of Clauses	24
2.2	Number of Dependent Clauses	28
2.3	Partial Satisfaction	31
2*	The Algorithmic Lovász Local Lemma	41
2*.1	A very simple algorithm	42
2*.2	A bit of information theory	44
2*.3	Bounding the recursion	45
2**	Minimal Unsatisfiable Formulas	53
2**.1	Matchings and Formulas	53
2**.2	Strongly Minimal Unsatisfiable Formulas	57
2**.3	More On Resolution	62
2**.3.1	Davis-Putnam Resolution	62
2**.3.2	The Structure of MU(1)-Formulas	66
2**.4	Short Resolution Proofs	67

2**	5 Universal Patterns For Unsatisfiability	69
3	Algorithms for 2-SAT	77
3.1	Unit Clause Reduction	78
3.2	A Randomized Algorithm	81
3.2.1	Symmetric Random Walks	82
3.2.2	Coupling	84
3.3	A Linear Time Algorithm	89
3.4	Examples	92
3.4.1	Solving 3-SAT in less than 2^n Steps	92
3.4.2	Coloring Graphs	93
4	SAT and the Class \mathcal{NP}	97
4.1	Verifying and Falsifying	98
4.2	The Class \mathcal{NP} and Relatives	103
4.3	SAT is \mathcal{NP} -complete	109
4.3.1	Turing Machines	110
4.3.2	Representing Computations for k-Falsifiers	111
5	The Cube	115
5.1	Faces	116
5.2	Hamming Balls	122
6	Coding and k-SAT Algorithms	131
6.1	Encoding Satisfying Assignments	132
6.2	A Randomized k-SAT Algorithm	136
7	Hamming Balls and k-SAT Algorithms	141
7.1	A Deterministic Algorithm	143
7.1.1	Employing Covering Codes	143
7.1.2	Constructing Covering Codes	144
7.2	A Randomized Algorithm	147
7.2.1	Random Walks Revisited	148
7*	Derandomizing Schönning's Algorithm	155
7*.1	The Algorithm	155
7*.2	Hindsight: Why and How?	160

8	The PPSZ Algorithm	171
8.1	Having a Unique Satisfying Assignment	173
8.1.1	Building Critical Clause Trees	175
8.1.2	Placements, Critical Clause Trees and Forced Variables	176
8.2	Random Deletion in Binary Trees	180
8.2.1	Monotone Convergence	180
8.2.2	Numerical Integration and Riemann Sums	181
8.2.3	The FKG Inequality	182
8.2.4	Of Infinite and Finite Trees	184
8.2.5	Of Independent and Dependent Labels	187
8.2.6	Integrating over the Rank of the Root	189
8.3	Multiple Satisfying Assignments	192
8.3.1	Problem Assessment	193
8.3.2	Definition of a Cost Function	196
8.3.3	Gathering a few Simple Facts	197
8.3.4	Bounding Correlations in Weighted Sums	201
8.3.5	Trading Survival Probability for Cost Savings	202
8*	Sparsification and ETH	211
8*.1	k-SAT in Subexponential Time?	211
8*.2	The Need for Sparsification	212
8*.3	The Sparsification Algorithm	214
9	Constraint Satisfaction	221
9.1	An Algorithm for $(3, 2)$ -CSP	223
10	Random k-CNF Formulas	227
10.1	First Linear Bounds	227
10.2	Improved Upper Bound for 3-CNF	231
A	Glossary of Notions and Facts	i
A.1	Fibonacci Numbers, Golden Ratio	i
A.2	Some Recurrences	i
A.3	Markov Chains	iii
A.4	Tail Estimates	iii
A.5	Jensen's Inequality	v

0.1 Course Description

Satisfiability of Boolean Formulas—Combinatorics and Algorithms

(252-0491-00 Erfüllbarkeit logischer Formeln—Kombinatorik und Algorithmen)

Emo Welzl (lecturer) and Chidambaram Annamalai (assistant); ETH Zürich, Spring 2016, Tuesday, 10-12 (CAB G59) and Thursday 9-10 (CAB G59) for lectures, and Tuesday, 13-15 (CAB G57) for exercises.

<http://www.ti.inf.ethz.ch/ew/courses/SAT16/>

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 with its recent algorithmic version), recall a proof of the Cook-Levin Theorem of the NP-completeness of SAT, discuss and analyze several deterministic and randomized algorithms. 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, Nešetřil, 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).

Uwe Schöning, Jacobo Torán, *The Satisfiability Problem*, Lehmanns Verlag (2013).

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

$\mathbf{Z}, \mathbf{N}, \mathbf{N}_0$	\mathbf{Z} denotes the set of integers, \mathbf{N} the positive integers (natural numbers), and \mathbf{N}_0 the nonnegative integers. For $i, j \in \mathbf{Z}$, $\{i..j\} := \{n \in \mathbf{Z} \mid i \leq n \leq j\}$.
$\{i..j\}$	A, B sets; $k \in \mathbf{N}_0$: $ 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 .
$ A , 2^A$	$\Pr(\mathcal{E})$ denotes the probability of event \mathcal{E} . $E(X)$ 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 (<i>indicator function</i>).
$\binom{A}{k}, A \oplus B$	A (simple undirected) <i>graph</i> is a pair $G = (V, E)$, where V is a finite set (the <i>vertices</i>) and $E \subseteq \binom{V}{2}$ (the <i>edges</i>). Vertex u is termed <i>adjacent</i> to vertex v (or u is a <i>neighbor</i> of v), if $\{u, v\} \in E$. The <i>degree</i> , $\deg_G(v)$, of vertex v in graph G is the number of neighbors of v in G .
$A \times B$	A (simple) <i>directed graph</i> is a pair $G = (V, E)$, where V is a finite set and $E \subseteq V \times V$ (the <i>directed edges</i> ¹ or <i>arcs</i>). For $u, v \in V$, u is an <i>out-neighbor</i> of v if $(v, u) \in E$, and u is an <i>in-neighbor</i> of v if $(u, v) \in E$; u is a <i>neighbor</i> of v (or <i>adjacent</i> to v), if it is an out- or in-neighbor of v .
B^A	The binary boolean operators ‘and’ (<i>conjunction</i>) and ‘or’ (<i>disjunction</i>) are denoted by \wedge and \vee , resp. Negation is denoted by \neg or by a horizontal bar over the operand (e.g. \bar{x}). We employ the usual shortcuts \rightarrow (<i>implication</i>), \leftrightarrow (<i>equivalence</i>), and \oplus (<i>exclusive or</i> , ‘ <i>xor</i> ’) ($x \rightarrow y$ for $\bar{x} \vee y$, $x \leftrightarrow y$ for $(x \rightarrow y) \wedge (y \rightarrow x)$ and $x \oplus y$ for $\neg(x \leftrightarrow y)$).
$\Pr(\mathcal{E})$	Two boolean formulas f and g are <i>equivalent</i> 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 <i>satisfiable</i> , if there is an assignment under which f evaluates to true (i.e. $f \not\equiv \text{false}$). f is a <i>tautology</i> , if it evaluates to true under all assignments (i.e. $f \equiv \text{true}$).
$[P]$	
$\deg_G(v)$	
	
\wedge, \vee	
$\neg, \bar{}$	
$\leftrightarrow, \rightarrow, \oplus$	
$f \equiv g$	

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

²Note 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 boolean¹ formula in propositional logic, take

$$(\bar{x} \rightarrow y) \wedge (y \rightarrow z) \wedge (z \rightarrow \bar{x}) \wedge (x \vee \bar{y} \vee \bar{z}) \quad (1.1)$$

or

$$(((x \wedge y) \vee (\bar{y} \wedge z)) \wedge \neg(x \leftrightarrow y)) \oplus (x \wedge \bar{y} \wedge z) \quad (1.2)$$

for examples, allows a $\{\text{true}, \text{false}\}$ -assignment to variables so that the formula evaluates to **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 **true** and 0 represent **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 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

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

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

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

evaluate to true.

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

Exercise 1.5 *Is $(\{\text{true}, \text{false}\}, \oplus, \wedge)$ 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

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

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

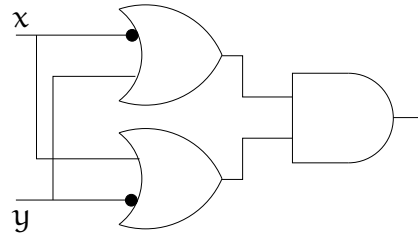


Figure 1.1: A circuit testing equality of signals x and y . \sqcap is the AND-gate, \supset is the OR-gate, and \bullet negates. The output is 1 iff the inputs x and y are equal.

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

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

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

$$(x \vee \bar{y}) \wedge (y \vee \bar{z}) \wedge (z \vee \bar{x}) . \quad (1.4)$$

Is it really equivalent²?

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).

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

²This 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 $(x \vee \bar{y})$), and y false forces z to be false (because of $(y \vee \bar{z})$); thus, all three have to be false in this case. Similarly, if x is true, then z has to be true (because of $(z \vee \bar{x})$), and z true implies y true (because of $(y \vee \bar{z})$); 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.

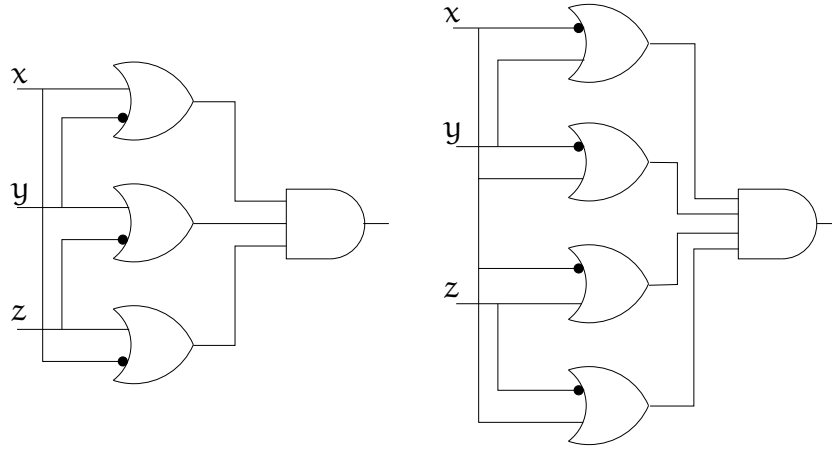


Figure 1.2: Two equivalent circuits?

unsatisfiability is postponed to Section 1.5).

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

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

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

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 $x \rightarrow y$ 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 \rightarrow 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

$$(\bar{x} \vee y_1) \wedge (\bar{y}_1 \vee y_2) \wedge (\bar{y}_2 \vee y_3) \wedge \dots \wedge (\bar{y}_{n-1} \vee y_n) \wedge (\bar{y}_n \vee x) .$$

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

$$(\bar{x} \vee y_1) \wedge (\bar{y}_1 \vee y_2) \wedge (\bar{y}_2 \vee y_3) \wedge \dots \wedge (\bar{y}_{n-1} \vee y_n) \wedge (\bar{y}_n \vee \bar{x}) .$$

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

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

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

$$\neg(p_i \wedge q_j) \quad \equiv \quad (\bar{p}_i \vee \bar{q}_j) .$$

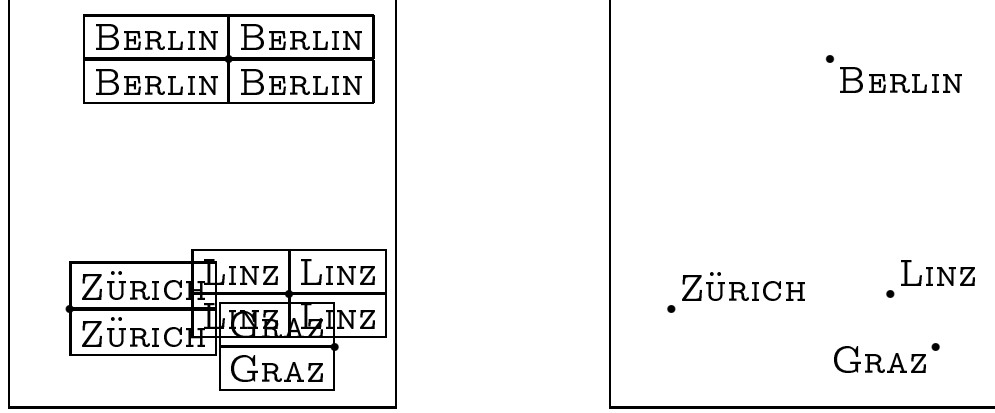


Figure 1.3: The formula $(b_1 \vee b_2 \vee b_3 \vee b_4) \wedge (g_2 \vee g_3) \wedge (l_1 \vee l_2 \vee l_3 \vee l_4) \wedge (z_1 \vee z_4) \wedge (\bar{z}_1 \vee \bar{l}_2) \wedge (\bar{z}_1 \vee \bar{l}_3) \wedge (\bar{z}_4 \vee \bar{l}_3) \wedge (\bar{l}_3 \vee \bar{g}_2) \wedge (\bar{l}_4 \vee \bar{g}_2)$ has a satisfying assignment with $b_4, g_3, l_1, z_1 \mapsto 1$ and the remaining variables mapped to 0.

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 $\varphi_f : \{0, 1\}^V \rightarrow \{0, 1\}$ 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 $\varphi : \{0, 1\}^V \rightarrow \{0, 1\}$ a CNF formula that induces φ . Such a formula is given by

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

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

x	y	z	$x \leftrightarrow (y \wedge z)$	
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	$(x \vee \bar{y} \vee \bar{z})$
1	0	0	0	$\wedge (\bar{x} \vee y \vee z)$
1	0	1	0	$\wedge (\bar{x} \vee y \vee \bar{z})$
1	1	0	0	$\wedge (\bar{x} \vee \bar{y} \vee z)$
1	1	1	1	

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

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.*

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 \ominus -node does not have a \ominus -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 \ominus -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 α_i . Each \ominus -node gets associated the literal $\bar{\alpha}$, where α is the variable associated with its child, 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 α_i , its left child associated with literal u and the right child with v , then we let f_i be the formula $\alpha_i \leftrightarrow (u \circ v)$. Given that, we set

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

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

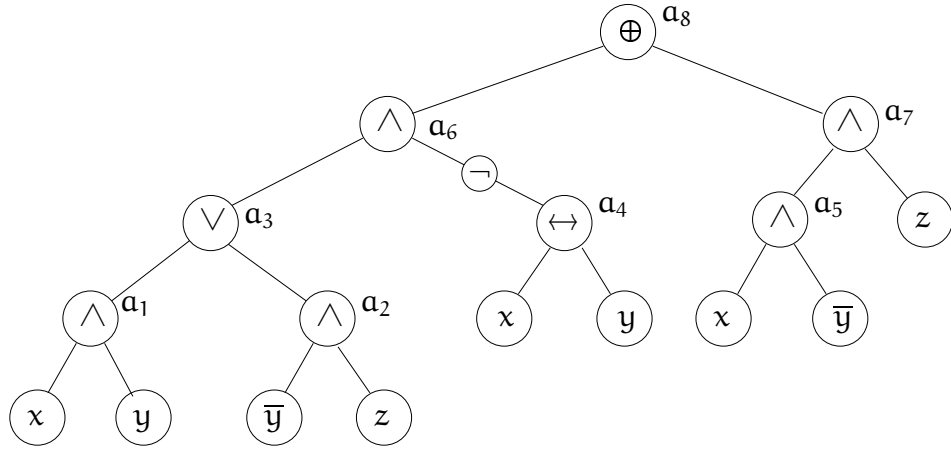


Figure 1.5: The evaluation tree of the boolean formula (1.2) with associated new variables for inner nodes. ‘Towards’ f' we consider

$$(a_8) \wedge (a_1 \leftrightarrow (x \wedge y)) \wedge \cdots (a_6 \leftrightarrow (a_3 \wedge \overline{a_4})) \wedge \cdots (a_8 \leftrightarrow (a_6 \oplus a_7)).$$

Any assignment α 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_{root} which equals the value of f under α . That is, we have shown that if f is satisfiable, then f'' is.

For the reverse direction, a satisfying assignment α'' for f'' must observe all f_i ’s (and it must obey $a_{\text{root}} \mapsto 1$), and thus the restriction of α'' 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 disjunctions⁴. 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. \square

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

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. $(x \vee \bar{y} \vee z)$ ‘becomes’ $\{x, \bar{y}, z\}$, 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’ $\{\{x, \bar{y}\}, \{y, \bar{z}\}, \{z, \bar{x}\}, \{x, y\}, \{\bar{x}, \bar{y}\}\}$. 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 disallowing⁵ 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

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

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 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

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

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

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

Exercise 1.11

Some Two

 $n \in \mathbf{N}$. *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.

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

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 $V \cup \bar{V}$, where $\bar{V} := \{\bar{x} \mid x \in V\}$ is the set of *negations* of the variables in V . Elements in V are called *positive literals*, elements in \bar{V} *negative literals*. We extend negation to literals $u \in V \cup \bar{V}$ by

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

Literals u and v are *strictly distinct*, if $u \neq v$ and $u \neq \bar{v}$. 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 $V \cup \bar{V}$; 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. $\{x \in V \mid x \in C \text{ or } \bar{x} \in C\}$, is denoted by $\text{vbl}(C)$. Clauses C and C' are called *strictly disjoint* or *independent* if $\text{vbl}(C) \cap \text{vbl}(C') = \emptyset$, 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 $\text{vbl}(F) := \bigcup_{C \in F} \text{vbl}(C)$.

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

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

which denotes the same assignment.

An assignment α on V *satisfies a clause* C over some U , if $\alpha(u) = 1$ for at least one literal $u \in (V \cup \bar{V}) \cap C$. α *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 $\text{vbl}(C) \subseteq V$ or $\text{vbl}(F) \subseteq V$. For example, the assignments $(x \mapsto 1, z \mapsto 0)$ or $(x \mapsto 0, y \mapsto 0)$ both satisfy the clause $\{x, \bar{y}, z\}$. 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 α on V is *total* for a clause C (for a formula F) if $\text{vbl}(C) \subseteq V$ ($\text{vbl}(F) \subseteq V$, resp.).

For C a clause and α an assignment on V , we let

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

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

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

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

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

- (i) α satisfies F iff $F^{[\alpha]} = \{\}$,
- (ii) F is satisfiable iff $F^{[x \mapsto 0]}$ or $F^{[x \mapsto 1]}$ is satisfiable, and
- (iii) $|\text{sat}_V(F)| = |\text{sat}_{V \setminus \{x\}}(F^{[x \mapsto 0]})| + |\text{sat}_{V \setminus \{x\}}(F^{[x \mapsto 1]})|$.

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.

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

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

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 $n \in \mathbb{N}$, how many functions $\{0, 1\}^n \rightarrow \{0, 1\}$ 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 $\{0, 1\}^{10} \rightarrow \{0, 1\}$ that cannot be realized by a 3-CNF formula.

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

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

1.4 Counting Satisfying Assignments

$\#\text{SAT}$ 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 $\text{cs}(F, V)$ that takes a CNF formula F over V and returns the number of assignments on V satisfying F . Note that if $x \in V \setminus \text{vbl}(F)$, 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 $\text{cs}()$ apart from 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 assumption⁸. A more specific statement will have to refer to the representation of the pair (F, V) —details

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

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

we like to sweep under the rug here. Given that, let us account 1 (step) for each call of the procedure cs(). Whatever bound we prove for this number of steps, we can multiply it by a polynomial in order to get a bound on the overall running time of a call.

Let $t(n)$, $n \in \mathbb{N}$, 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

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

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).

A very easy instance of #SAT is a (≤ 1) -CNF formula F over a variable set V , see procedure cls() 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 (≤ 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.	function c1s(F, V)
PRECONDITION: F is (≤ 1) -CNF	if $(\square \in F \text{ or } \exists x \in V : \{\{x\}, \{\bar{x}\}\} \subseteq F)$ then
POSTCONDITION: returns $ \text{sat}_V(F) $.	return 0;
	else
	return $2^{ V - F }$;

	function c2s(F, V)
	if F is (≤ 1) -CNF then
	return c1s(F, V);
	else
V boolean variables, F CNF formula over V.	$\{u, v\} \leftarrow \text{some 2-clause in } F$;
PRECONDITION: F is (≤ 2) -CNF	$\alpha \leftarrow (u \mapsto 1, v \mapsto 1)$;
POSTCONDITION: returns $ \text{sat}_V(F) $.	$\beta \leftarrow (u \mapsto 0, v \mapsto 1)$;
	$\gamma \leftarrow (u \mapsto 1, v \mapsto 0)$;
	$U \leftarrow V \setminus \text{vbl}(\{u, v\})$;
	return c2s($F^{[\alpha]}$, U) + c2s($F^{[\beta]}$, U) + 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 $\text{c2s}()$, and the results are accumulated for the number to be returned.

Let $t'(n)$, $n \in \mathbb{N}$, be the worst case bound for the number of calls to $\text{c2s}()$ implied by a call with a (≤ 2) -CNF formula over n variables (we do not count the call to $\text{c1s}()$). 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 $t'(n) \leq 1 + 3^1 + \dots + 3^{\lfloor n/2 \rfloor} = \frac{1}{2} (3^{\lfloor n/2 \rfloor + 1} - 1)$. A bound of $O(3^{n/2} \text{poly}(n+m)) = O(1.733^n \text{poly}(m))$ for $\#SAT$ restricted to (≤ 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

	function <code>fib_c2s(F, V)</code>
	if F is (≤ 1) -CNF then
	return <code>c1s(F, V)</code> ;
	else
	$\{u, v\} \leftarrow \text{some 2-clause in } F$;
	$\alpha \leftarrow (u \mapsto 1); \quad \beta \leftarrow (u \mapsto 0, v \mapsto 1)$;
	$U \leftarrow V \setminus \text{vbl}(\{u\}); \quad W \leftarrow V \setminus \text{vbl}(\{u, v\})$;
	return <code>fib_c2s(F^[α], U) + fib_c2s(F^[β], W)</code> ;
V boolean variables, F CNF formula over V . PRECONDITION: F is (≤ 2) -CNF POSTCONDITION: returns $ \text{sat}_V(F) $.	

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

It gives us a bound of $t''(n) \leq 2f_{n+1} - 1$ for f_m , $m \in \mathbf{N}$, 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 (≤ 2) -CNF formula with m clauses over n variables can be computed in time*

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

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 (≤ 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 $A : V \rightarrow \{0, 1, *\}$. It represents all assignments α on V with $A(x) \in \{\alpha(x), *\}$, i.e. α 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 (≤ 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 (≤ 2) -CNF formula F —“succinctly” means that you represent $\text{sat}_V(F)$ 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 α is called minimally satisfying F if it satisfies F , and no proper restriction of α 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 (≤ 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 \bar{x} in the other one; say $x \in C$ and $\bar{x} \in D$. For $C' := C \setminus \{x\}$ and $D' := D \setminus \{\bar{x}\}$, we define $C' \cup D'$ as the *resolvent* of C and D . For example,

$\{x_1, \bar{x}_2, x_3\}$ is the resolvent of $\{x_0, x_1, \bar{x}_2\}$ and $\{\bar{x}_0, x_1, x_3\}$.

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 α satisfies $\{C, D\}$ iff it satisfies $\{C, D, C' \cup D'\}$.

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

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

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).

$$\begin{aligned}
 F &= \underbrace{\{x, \bar{y}\}}_{(1)}, \underbrace{\{y, \bar{z}\}}_{(2)}, \underbrace{\{z, \bar{x}\}}_{(3)}, \underbrace{\{x, y\}}_{(4)}, \underbrace{\{\bar{x}, \bar{y}\}}_{(5)} \\
 (1), (2) &: \{x, \bar{z}\} \quad (6) \\
 (1), (3) &: \{\bar{y}, z\} \quad (7) \\
 (1), (4) &: \{x\} \quad (8) \\
 (1), (5) &: \{\bar{y}\} \quad (9) \\
 (2), (3) &: \{\bar{x}, y\} \quad (10) \\
 (2), (5) &: \{\bar{x}, \bar{z}\} \quad (11) \\
 (3), (4) &: \{y, z\} \quad (12) \\
 (5), (10) &: \{\bar{x}\} \quad (13) \\
 (8), (13) &: \square \quad (14)
 \end{aligned}$$

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

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

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 \square 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 \square from F ; we choose such a formula which is minimal with respect to $|\text{vbl}(F)|$.

$\text{vbl}(F)$ cannot be empty, since then $F = \{\}$ or $F = \{\square\}$. In the first case F is satisfiable, in the second case F contains already the empty clause and (\square) is a resolution deduction of \square from F . So we may assume that $\text{vbl}(F)$

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), $|\text{vbl}(F_0)| < |\text{vbl}(F)|$, and $|\text{vbl}(F_1)| < |\text{vbl}(F)|$. Therefore, there is a resolution deduction of \square 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, \dots, C_m) of $\square = C_m$ from F_0 . We define a resolution deduction $(\hat{C}_0, \hat{C}_1, \dots, \hat{C}_m)$ from F with

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

for all ℓ , $0 \leq \ell \leq m$, as follows. For $i = 0, 1, \dots, m$ 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 $\hat{C}_m = \square$ 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 \square or $\{\bar{x}\}$. Either we have a contradiction right away, or we merge the two deductions to one containing both $\{x\}$ and $\{\bar{x}\}$, which we can extend by \square . So we have established a contradiction in either case, and the theorem is shown. \square

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 \square , 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 \square 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 $x \notin \text{vbl}(C)$. Then C is the resolvent of $C \cup \{x\}$ and $C \cup \{\bar{x}\}$. Show that $\{C\} \equiv \{C \cup \{x\}, C \cup \{\bar{x}\}\}$.

(3) Employ (1) and (2) to simplify the CNF formula

$$\{\{\bar{x}_1, x_2, x_3\}, \{\bar{x}_1, \bar{x}_2, x_3\}, \{\bar{x}_1, x_3, x_4, x_5\}, \{x_1, x_3\}\}$$

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 now¹ 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.

Exercise 2.1

Satisfaction Probabilities

For an assignment α drawn uniformly at random from all assignments on $V = \{x_1, x_2, \dots, x_{100}\}$ determine the following probabilities.

- (1) $\Pr(\alpha \text{ satisfies } \{x_1, \overline{x_3}, x_7\})$.
- (2) $\Pr(\alpha \text{ satisfies } \{\{x_1, \overline{x_5}, x_{11}\}, \{\overline{x_2}, x_4\}, \{x_6, \overline{x_{12}}, x_{13}\}\})$.
- (3) $\Pr(\alpha \text{ satisfies } \{\{x_1, x_2, x_3\}, \{\overline{x_3}, x_4, x_5\}\})$.

¹Well, you'll see. This was right a couple of years ago, but the world has changed.

- (4) $\Pr(\alpha \text{ satisfies } \{\{x_1, x_2, x_3\}, \{x_3, x_4, x_5\}\})$.
- (5) $\Pr(\alpha \text{ satisfies } \{\{x_1, x_2, \overline{x_3}\}, \{\overline{x_1}, x_3, x_4, x_5, \overline{x_7}\}\})$.

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 α 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 variables²

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

Then

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

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. \square

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

$$\left| \bigcup_{C \in F} \overline{\text{sat}}(C) \right| \leq \sum_{C \in F} |\overline{\text{sat}}(C)| = m 2^{n-k}.$$

²Recall the indicator function $[P]$ for a predicate P (cf. Section 0.2).

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

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.

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

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

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

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

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

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

$$E(Y_F | x_1 \mapsto t_1) \leq E(Y_F) .$$

Since³

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

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

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

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

$$\mathbb{E}(Y_F | \alpha_i) = \frac{1}{2} (\mathbb{E}(Y_F | \alpha_i, x_{i+1} \mapsto 0) + \mathbb{E}(Y_F | \alpha_i, x_{i+1} \mapsto 1)) \quad ,$$

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

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

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

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

Note that for $x \notin V'$

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

If we extend now the mapping sc to CNF formulas

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

we observe that

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

³ $C[x_1 \mapsto 0]$ may be 1. We set $X_1 = 0$ under all assignments.

- If α is total for F then $\text{sc}(F, \alpha)$ is the number of clauses in F not satisfied by α .
- For $x \notin V'$,

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

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

It follows that we can successively assign values to the variables in $\text{vbl}(F)$ 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. \square

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 [Raghavan, 1988].

Exercise 2.2

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, \bar{x} of literals.

Exercise 2.3

Almost Satisfiable Versus Balanced⁴

Let F be an unsatisfiable k -CNF formula with $\sum_{C \in F} 2^{-|C|} = 1$.

⁴due to Andreas Razen

- (1) Show that F has to be balanced in the sense that each variable occurs equally often as positive and as negative literal in F .
- (2) Is the statement in (1) true for F an arbitrary unsatisfiable CNF formula with $\sum_{C \in F} 2^{-|C|} = 1$?

Exercise 2.4

Branching Rules

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

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

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

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

Exercise 2.5 True or false? A k -CNF formula, $k \in \mathbb{N}$, with 2^k clauses is unsatisfiable iff it consists of all k -clauses over a set of k variables.

Exercise 2.6

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.

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

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

Theorem 2.3 (“Lovász Local Lemma”) $k \in \mathbf{N}$. 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 $V := \text{vbl}(F)$. We use $\text{sat}()$ and $\overline{\text{sat}}()$ short for $\text{sat}_V()$ and $\overline{\text{sat}}_V()$, 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

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

Under the assumptions of the theorem we claim that

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

always. This claim is equivalent to

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

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

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

therefore $\text{sat}(F) \neq \emptyset$ 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

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

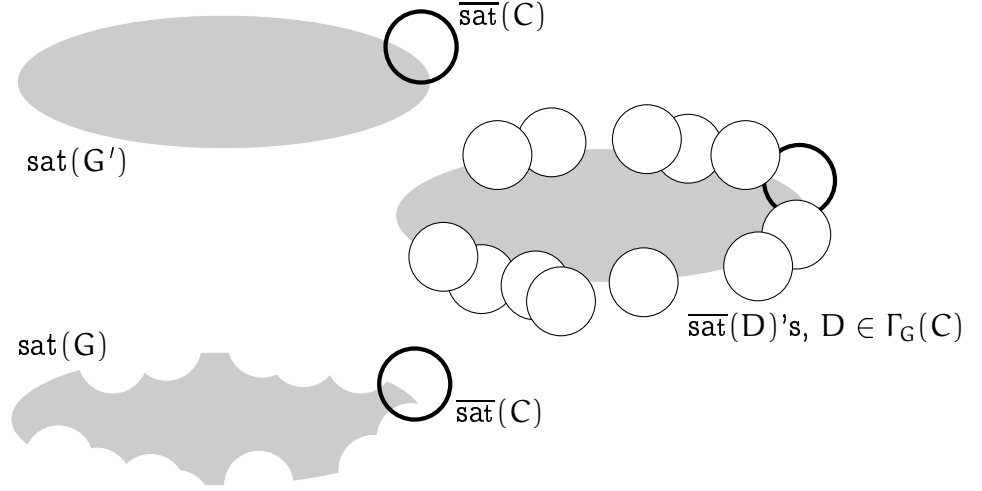


Figure 2.1: Illustrating the proof of Theorem 2.3

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

Hence,

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

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

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

and (2.2) is established. \square

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 2.9).

Exercise 2.7 $k \in \mathbf{N}$. Show that every k -CNF formula where every variable occurs (as positive or negated literal) at most k times is satisfiable.

Exercise 2.8

Many Variables

Let F be a CNF formula so that $|G| \leq |\text{vbl}(G)|$ for all $G \subseteq F$. Show that F is satisfiable.

Exercise 2.9 $p \in \mathbf{R}$, $0 \leq p < 1$. Let \mathcal{A} be a 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 $\Pr(\bigwedge_{A \in \mathcal{A}} \overline{A}) > 0$, where \overline{A} denotes the complementary event of A .

Exercise 2.10 $k \in \mathbf{N}$. Let $d \in \mathbf{N}$ be such that there exists a real number δ , $1 < \delta < 2^k$, 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 $d \approx \frac{2^k}{e}$ and $\delta \approx e$, 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, given certain preconditions on the formula. In fact, some clauses (constraints) may be considered more important than others—we will model this by giving weights to clauses (proportional to our urge to see them satisfied).

Weighted Formulas. A *weighted CNF formula* is a pair (F, μ) where F is a CNF formula and $\mu : F \rightarrow \mathbf{R}_0^+$. We extend μ to subsets G of F by $\mu(G) := \sum_{C \in G} \mu(C)$. For α an assignment, we set

$$\mu^{[\alpha]}(G) := \sum_{C \in G} \mu(C) \cdot [\alpha \text{ satisfies } C] .$$

If α is a random assignment from some distribution then

$$\mathbb{E}(\mu^{[\alpha]}(F)) = \sum_{C \in F} \mu(C) \cdot \Pr(\alpha \text{ satisfies } C) . \quad (2.4)$$

It immediately follows that if F is an $(\geq k)$ -CNF formula then, for α uniformly at random from $\{0, 1\}^{\text{var}(F)}$,

$$\mathbb{E}(\mu^{[\alpha]}(F)) \geq \left(1 - \frac{1}{2^k}\right) \mu(F),$$

and, therefore, in such a formula we can always satisfy a $1 - \frac{1}{2^k}$ portion of the whole weight.

Instead of forcing a lower bound on the size of the clauses, we consider now formulas that are “locally satisfiable” in the sense that small subsets of the clauses are simultaneously satisfiable.

k-Satisfiable Formulas. Given $k \in \mathbf{N}_0$, a CNF formula F is called *k-satisfiable* if every subset G of F with $|G| \leq k$ is satisfiable.

Every CNF formula F is 0-satisfiable and it is 1-satisfiable iff $\square \notin F$. Note that “1-satisfiable” is equivalent to “ (≥ 1) -CNF” and so at least half of the weight of a 1-satisfiable CNF formula can be satisfied. This $\frac{1}{2}$ -bound is tight, as can be seen from

$$F := \{\{x_1\}, \{\overline{x_1}\}, \{x_2\}, \{\overline{x_2}\}, \dots, \{x_n\}, \{\overline{x_n}\}\}$$

with all weights equal 1.

Weighted 2-Satisfiable Formulas and the Golden Ratio. We observe that a CNF formula F is 2-satisfiable iff it contains neither the empty clause \square nor a pair $\{x\}, \{\overline{x}\}$ of complementary 1-clauses. That, as we will show next, allows us to satisfy significantly more than half of the weight of the formula.

Theorem 2.4 ([Lieberherr, Specker, 1981]) *Every weighted 2-satisfiable CNF formula (F, μ) has an assignment α with*

$$\mu^{[\alpha]}(F) \geq \Phi \cdot \mu(F), \quad \Phi := \frac{-1+\sqrt{5}}{2} \approx 0.618, \text{ the golden ratio}^5 \text{ (conjugate).}$$

Proof. We assume that no $\{\bar{x}\}$, $x \in \text{vbl}(F)$, occurs in F . If such a negative 1-clause $\{\bar{x}\}$ appears, we switch all occurrences of x to \bar{x} and vice versa.

Our goal is to choose a random assignment so that every clause in F is satisfied with some large probability. If that probability is supposed to be larger than $\frac{1}{2}$, then we cannot choose an assignment uniformly at random because of the 1-clauses.

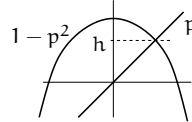
We fix $p \in \mathbf{R}$, $0 \leq p \leq 1$, (what value p should attain will evolve from the proof) and we define a random assignment α on $\text{vbl}(F)$ by

$$\Pr(x \mapsto 1) = p \quad \text{for all } x \in \text{vbl}(F) \text{ (mutually independent).}$$

Then, for $C \in F$, we have:

- If C contains a positive literal, then α satisfies C with probability at least p .
- If C contains no positive literal, then it must contain at least two negative literals and α satisfies C with probability at least $1 - p^2$.

Therefore, for $h := \min\{p, 1 - p^2\}$, every clause in F is satisfied with probability at least h . Hence, we should set p in $[0, 1]$ to maximize h .



This is easily seen to happen when $p = 1 - p^2$, $p \geq 0$, i.e. $p = \frac{-1+\sqrt{5}}{2}$. Then $h = \frac{-1+\sqrt{5}}{2} = \Phi$. Now $\Pr(\alpha \text{ satisfies } C) \geq \Phi$ for all $C \in F$ and the theorem follows. \square

We want to argue that the golden ratio bound is tight. For that purpose, given $n \in \mathbf{N}$, we choose some set V of n variables and some $v \in \mathbf{R}^+$ (and

⁵Usually, $\varphi := \frac{1+\sqrt{5}}{2} \approx 1.618$ is called the *golden ratio*, while $\Phi := \frac{1}{\varphi} = \varphi - 1$ is called the *golden ratio conjugate*. The golden ratio is omnipresent, among others in architecture and art, and the letter “phi” is used to honor the Greek sculptor Phidias (ca. 430 BC).

again, we decide on ν 's fate as the time is ripe). Let

$$F = F_n := \{\{x\} \mid x \in V\} \cup \{\{\bar{x}, \bar{y}\} \mid \{x, y\} \in \binom{V}{2}\};$$

the weight function $\mu = \mu_{n,\nu}$ assigns ν to every 1-clause and 1 to the 2-clauses. Note that the overall weight is $n\nu + \binom{n}{2}$.

F is convenient for our analysis, since, for $\alpha \in \{0, 1\}^V$, the weight $\mu^{[\alpha]}(F)$ of clauses satisfied by α is determined by the number, k , of variables in V that are set to 1. We have

$$\mu^{[\alpha]}(F) = s(k) := k\nu + \binom{n}{2} - \binom{k}{2}.$$

(You may view the rest of the argument as a game: Given ν , player A chooses k as to maximize the weight satisfied. And given the optimal strategy of A, player B chooses ν so that A's success, the ratio of the weight satisfied, is minimized. The best strategy of B yields the ν for the hardest formula.)

The function $s(k)$ as a function in \mathbf{R} attains its maximum for $k = \nu + \frac{1}{2}$ (obtained by setting the first derivative of $s(k)$ to 0). That is, no more than weight

$$s(\nu + \tfrac{1}{2}) = \frac{\nu^2}{2} + \binom{n}{2} + O(\nu)$$

can be satisfied and for every assignment the ratio of the weight satisfied is at most

$$\frac{s(\nu + \frac{1}{2})}{\mu(F)} = \frac{\frac{\nu^2}{2} + \frac{n^2}{2} + O(n + \nu)}{n\nu + \frac{n^2}{2} + O(n)} = \frac{(\frac{\nu}{n})^2 + 1 + O(\frac{1}{n} + \frac{\nu}{n^2})}{2\frac{\nu}{n} + 1 + O(\frac{1}{n})}.$$

We fix some $\lambda \in \mathbf{R}^+$ and we set $\nu = \lambda n$. As n goes to infinity, the ratio above converges to $\frac{\lambda^2+1}{2\lambda+1}$. Going through the usual exercise of setting the first derivative to 0, we get that in the range $\lambda \geq 0$ this expression minimizes for $\lambda = \frac{-1+\sqrt{5}}{2} = \Phi$. For that value of λ we get also⁶ $\frac{\lambda^2+1}{2\lambda+1} = \Phi$. This means that

⁶As if it needed another proof that there is no escape from the golden ratio.

if $v = \Phi n$ then the ratio of the optimal weight satisfiable in F_n approaches Φ as $n \rightarrow \infty$; formally,⁷

$$\lim_{n \rightarrow \infty} \max_{\alpha} \frac{\mu_{n, \Phi n}^{[\alpha]}(F_n)}{\mu_{n, \Phi n}(F_n)} = \Phi .$$

Therefore, no bound exceeding Φ can be guaranteed.

An Algorithmic Implication. Employing the technique of derandomization via conditional probabilities, we can turn the proof of Theorem 2.4 into a polynomial time algorithm that computes for every weighted 2-satisfiable CNF formula an assignment that satisfies Φ of the overall weight.

Given (F, μ) , let α^* be an assignment that maximizes the satisfied weight, i.e. $\mu^{[\alpha^*]}(F) = \max_{\alpha} \mu^{[\alpha]}(F)$. No algorithm is known that computes such an optimal α^* in polynomial time. We want to show that there is a polynomial algorithm that computes an assignment α that satisfies

$$\mu^{[\alpha]}(F) \geq \Phi \mu^{[\alpha^*]}(F) .$$

Just to avoid any confusion: If F is 2-satisfiable, then—given what we know—this is obvious since we can produce an α with $\mu^{[\alpha]}(F) \geq \Phi \mu(F)$ and clearly $\mu(F) \geq \mu^{[\alpha^*]}(F)$. So it remains to tackle the presence of the empty clause and complementary pairs of 1-clauses.

For that we decompose a weighted CNF formula (F, μ) into weighted formulas (F_0, μ_0) , (F_1, μ_1) , and (F_2, μ_2) . The idea behind this decomposition is that F_0 and F_1 are trivial in the sense that every total assignment satisfies the same weight of it, and F_2 is 2-satisfiable. For the specification, we extend μ to all clauses over $\text{vbl}(F)$ simply by setting $\mu(C) = 0$ for $C \notin F$:

$$F_0 := \begin{cases} \{\square\} & \text{if } \square \in F, \text{ and} \\ \{\} & \text{otherwise.} \end{cases}$$

In the first case, $\mu_0(\square) := \mu(\square)$.

$$F_1 := \bigcup_{x \in V : \min\{\mu(\{x\}), \mu(\{\bar{x}\})\} > 0} \{\{x\}, \{\bar{x}\}\}$$

and for $\{u\} \in F_1$, $\mu_1(\{u\}) := \min\{\mu(\{u\}), \mu(\{\bar{u}\})\}$.

$$F_2 := F \setminus (\{\square\} \cup \{\{u\} \mid \mu(\{u\}) = \min\{\mu(\{u\}), \mu(\{\bar{u}\})\}\})$$

⁷If you are concerned that we actually proved only “ $\leq \Phi$ ”, recall Theorem 2.4.

and for $C \in F_2$,

$$\mu_2(C) := \begin{cases} \mu(\{u\}) - \mu(\{\bar{u}\}) & \text{if } C \text{ is a 1-clause } \{u\}, \text{ and} \\ \mu(C) & \text{otherwise.} \end{cases}$$

That is, we remove the empty clause, if there. For every complementary pair $\{x\}, \{\bar{x}\}$ in F , we keep only the one of the two with larger weight, and the new weight is the excess of the old weight over the complementary 1-clause.

Clearly, these weighted formulas can be computed in polynomial time. For every clause C , we have $\mu_0(C) + \mu_1(C) + \mu_2(C) = \mu(C)$ (again, we extend weight mappings to all clauses by setting it to 0 if not defined). It follows that for every total assignment α

$$\mu^{[\alpha]}(F) = \mu_0^{[\alpha]}(F_0) + \mu_1^{[\alpha]}(F_1) + \mu_2^{[\alpha]}(F_2) = \frac{1}{2}\mu_1(F_1) + \mu_2^{[\alpha]}(F_2)$$

Now we can use that F_2 is 2-satisfiable, and we compute an assignment α on $\text{vbl}(F)$ with $\mu_2^{[\alpha]}(F_2) \geq \Phi\mu_2(F_2)$. It holds that

$$\begin{aligned} \mu^{[\alpha]}(F) &\geq \frac{1}{2}\mu_1(F_1) + \Phi\mu_2(F_2) \geq \frac{1}{2}\mu_1(F_1) + \Phi\mu_2^{[\alpha^*]}(F_2) \\ &\geq \Phi \left(\frac{1}{2}\mu_1(F_1) + \mu_2^{[\alpha^*]}(F_2) \right) = \Phi\mu^{[\alpha^*]}(F) . \end{aligned}$$

We have established the following result.

Theorem 2.5 *There is an algorithm that computes for every weighted CNF formula (F, μ) in polynomial time an assignment α with*

$$\mu^{[\alpha]}(F) \geq \Phi\mu^{[\alpha^*]}(F) .$$

NOTE 2.3 The best ratio known for a polynomial approximation algorithm is 0.7968 due to [Avidor *et al.*, 2005].

2-Satisfiable CNF Formulas—no Weights. What happens if we go back to unweighted CNF formulas (or we force all weights to be the same)? The CNF formula

$$\{\{x\}, \{y\}, \{\bar{x}, \bar{y}\}\}$$

shows that in the worst case we cannot guarantee to satisfy more than $\frac{2}{3}$ of the clauses in an unweighted 2-satisfiable CNF formula. It turns out that this is already the worst case.

Theorem 2.6 ([Käppeli, 2005]) *Every 2-satisfiable CNF formula has an assignment that satisfies at least $\frac{2}{3}$ of its clauses.*

Proof. Given a CNF formula F over V , the proof will juggle with occurrences of literals in the formula, so we prepare a formal basis for this.

Every pair (u, C) with $u \in C \in F$ is called *an occurrence of u in F* . Let us fix some assignment α on V . An occurrence (u, C) is called *critical* (relative to α) if either u is the only literal with $\alpha(u) = 1$ in C (a *positive critical occurrence of u*) or α does not satisfy C —in particular, $\alpha(u) = 0$ (a *negative critical occurrence of u*).

Suppose we switch α for a single variable x . Then a clause C switches from satisfied to unsatisfied or vice versa iff (x, C) or (\bar{x}, C) is a critical occurrence.

Let now α be an assignment with the property that (i) switching a single variable cannot increase the number of clauses satisfied, and (ii) if the number of clauses satisfied stays the same for a switch, then the number of satisfied 1-clauses cannot increase. Clearly, such an assignment exists by starting with an arbitrary assignment and then successively switching assignments for variables where the property is violated.

For ease of presentation, let us assume that α is the all-1 assignment. Then the required property of α is equivalent to the following:

- (i') For every variable x , the number of positive critical occurrences of x is at least the number of negative critical occurrences of \bar{x} .
- (ii') If, for a variable x , the number of positive critical occurrences of x equals the number of negative critical occurrences of \bar{x} then $\{\bar{x}\} \notin F$.

We will partition the clauses of F in such a way that it becomes obvious that α satisfies $\frac{2}{3}$ of each part from which the claim of the theorem clearly follows.

Property (i') above shows that there is an injective mapping κ from negative critical occurrences to positive critical occurrences. Moreover, along Property (ii'), we can guarantee that if $\{\bar{x}\} \in F$ then there is a spare positive critical occurrence $(x, C^{(x)})$ of x that does not appear in the image of κ .

Let $C = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_k\}$ be an unsatisfied clause with $\kappa(\bar{x}_i, C) = (x_i, C_i)$. Then we put C into one part of our partition with C_1, C_2, \dots, C_k ; and we add $C^{(x_1)}$, in case $k = 1$. So this part contains one unsatisfied clause and at

least two satisfied clauses. Finally, we put into one extra part all clauses we have not used along with unsatisfied clauses. All these clauses are satisfied by α . So, indeed, we have found the partition that entails the statement we wanted to prove. \square

Larger k . For $k \in \mathbb{N}_0$, let r_k be the supremum of all real numbers r so that every weighted k -satisfiable CNF formula has an assignment α with $\mu^{[\alpha]}(F) \geq r \mu(F)$. It is a simple exercise⁸ to prove that every weighted k -satisfiable CNF formula F has an assignment α with $\mu^{[\alpha]}(F) \geq r_k \mu(F)$, i.e. the supremum is a maximum. Clearly, $r_{k+1} \geq r_k$. $r_0 = 0$ and $r_1 = \frac{1}{2}$ was easy, and we proved also $r_2 = \Phi$. Let us move one step further to 3-satisfiable formulas.

Theorem 2.7 *Every weighted 3-satisfiable CNF formula (F, μ) has an assignment α with*

$$\mu^{[\alpha]}(F) \geq \frac{2}{3} \mu(F) ,$$

i.e. $r_3 \geq \frac{2}{3}$.

Proof. By switching occurrences of x and \bar{x} , if necessary, we can assume that F contains no clause $\{\bar{x}\}$, $x \in \text{vbl}(F)$. We define a random assignment α on $\text{vbl}(F)$ following the rule

$$\text{for } x \in \text{vbl}(F) \quad x \mapsto 1 \quad \text{with probability} \quad \begin{cases} \frac{2}{3} & \text{if } \{x\} \in F, \text{ and} \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

With this we get for C in F :

- If C is a 1-clause, then it is satisfied with probability $\frac{2}{3}$.
- If C is a 2-clause $\{u, v\}$, then $\Pr(u \mapsto 0) = \Pr(v \mapsto 0) = \frac{2}{3}$ is not possible, since then $C = \{\bar{x}, \bar{y}\}$ with $\{x\} \in F$ and $\{y\} \in F$ for variables x and y —a contradiction to 3-satisfiability. Thus at least one of $\Pr(u \mapsto 0)$ and $\Pr(v \mapsto 0)$ is at most $\frac{1}{2}$, and, therefore, C is satisfied with probability at least $1 - \frac{2}{3} \cdot \frac{1}{2} = \frac{2}{3}$.

⁸If you wonder, why this is an exercise at all, please think about it until you see it. For example, the supremum α^* of all numbers α with $\alpha < 1$ is obviously 1, so $\alpha^* < 1$ does not hold.

- If C is a (≥ 3) -clause, then it is satisfied with probability at least $1 - (\frac{2}{3})^3 = \frac{19}{27} \geq \frac{2}{3}$.

That is, the assignment satisfies every clause in F with probability at least $\frac{2}{3}$ which establishes the theorem. \square

The proof exploits the fact that a 3-satisfiable CNF formula has no \square , has no complementary pair $\{x\}, \{\bar{x}\}$, and it has no triple $\{u\}, \{v\}, \{\bar{u}, \bar{v}\}$. Note that 3-satisfiability excludes $\{u\}, \{\bar{u}, v\}, \{\bar{u}, \bar{v}\}$ as well, which was not used in the argument.

NOTE 2.4 As we observed, $r_{k+1} \geq r_k$ for $k \in \mathbb{N}$. [Huang, Lieberherr, 1985] showed that $r_k \leq \frac{3}{4}$ for all k , and [Trevisan, 2004] proved $\lim_{k \rightarrow \infty} r_k = \frac{3}{4}$.

For concrete values of r_k , we have seen r_0, r_1, r_2 , and a bound⁹ $r_3 \geq \frac{2}{3}$. [Král', 2004] proved

$$r_4 = \frac{3}{5 + \sqrt[3]{\frac{3\sqrt{69}-11}{2}} - \sqrt[3]{\frac{3\sqrt{69}+11}{2}}} \approx 0.6992 .$$

Interestingly, [Král', 2004] also showed that a probabilistic argument along the lines of our proofs of Theorems 2.4 and 2.7 will not give the tight result for r_4 !

Theorems 2.4 and 2.7 are originally from [Lieberherr, Specker, 1981] and from [Lieberherr, Specker, 1982], resp.; they use a different method called symmetrization, which is somewhat more involved, but still interesting. The simpler probabilistic approach we took here was proposed in [Yannakakis, 1994] (cf. [Jukna, 2001, Section 20.6]).

NOTE 2.5 Similarly to r_k , let us define s_k as the largest ratio of clauses that can always be satisfied in a k -satisfiable CNF formula (unweighted). Clearly, $s_k \geq r_k$. $s_1 = \frac{1}{2}$ is easy, but we have seen that $s_2 = \frac{2}{3}$ already deviates from r_2 . The proof of $r_k \leq \frac{3}{4}$ in [Huang, Lieberherr, 1985] carries over to s_k , so $\lim_{k \rightarrow \infty} s_k = \frac{3}{4}$ follows.

In [Käppeli, 2006] the bounds $0.6809 \approx \frac{32}{47} \leq s_3 \leq 0.7$ were established

Exercise 2.11 *Work out the derandomization of the proof of Theorem 2.4.*

⁹It is interesting to note that in [Král', 2004] and [Trevisan, 2004] $r_3 = \frac{2}{3}$ is attributed to [Lieberherr, Specker, 1982], but they write on page 18: “We find a lower bound on the fraction τ_3 of the clauses which can always be satisfied in a 3-satisfiable formula by showing that $\tau_3 \geq \frac{2}{3}$. Unfortunately, we have not been able to determine τ_3 exactly.” Probably $r_3 \leq \frac{2}{3}$ is easy, but I don't see the example that provides this upper bound.

Exercise 2.12

How Much Independence is Necessary?

In the proof of Theorem 2.4 we defined a random assignment α by choosing the values for the variables mutually independently. Is mutual independence necessary, or is perhaps pairwise independence sufficient?

Exercise 2.13

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 $\alpha \in A$ on V which satisfies at least $\frac{3}{4}$ of all clauses in F .

HINT: Use Hadamard matrices.

Exercise 2.14 Prove that every weighted k -satisfiable CNF formula F has an assignment α with $\mu^{[\alpha]}(F) \geq r_k \mu(F)$.

Exercise 2.15

All in 3—at Most 7 in 10

Show that the CNF formula

$$F := \{\{a\}, \{b\}, \{c\}, \{d\}, \{\bar{a}, x\}, \{\bar{b}, x\}, \{\bar{c}, \bar{x}\}, \{\bar{d}, \bar{x}\}, \{\bar{a}, \bar{b}, \bar{x}\}, \{\bar{c}, \bar{d}, x\}\}$$

is 3-satisfiable and no assignment satisfies more than 7 clauses in F .

REMARK: That establishes $s_3 \leq 0.7$.

Chapter 2*

The Algorithmic Lovász Local Lemma

Chapter by Dominik Scheder, with additions by Robin Moser.

In Chapter 2, we have stated and proved the Lovász Local Lemma for SAT. It has turned out that a k -CNF formula F in which every clause C has a neighbourhood $\Gamma_F(C)$ of at most 2^{k-2} clauses which it shares common variables with is always satisfiable. However, the proof of Theorem 2.3 is completely non-constructive and does not disclose any way in which such an assignment could be discovered efficiently. In the sequel, we will demonstrate that an efficient algorithmic procedure for solving such formulas can be devised quite easily. In order to give the proof in a simplest possible form, we will restrict the neighbourhoods of clauses to at most $2^{k-3} - 1$ in size. Asymptotically, this is no different. With somewhat more effort, it can be demonstrated that such a restriction is not necessary.

Theorem 2*.1 $k \in \mathbf{N}$. If $|\Gamma_F(C)| \leq 2^{k-3} - 1$ for all clauses C in a k -CNF formula F , then F is satisfiable and a satisfying assignment can be found in expected polynomial time by a randomized algorithm.

The objective of the present chapter is to prove this claim by describing and analyzing an appropriate algorithm.

Exercise 2*.1

Random sampling doesn't work

Prove: For every k , there exists a family of arbitrarily large formulas F having $\forall C \in F : |\Gamma_F(C)| \leq 2^{k-3} - 1$ which are satisfied with exponentially small probability (in the size of each formula) when sampling assignments u.a.r. This justifies why the theorem we are proving in

this chapter is non-trivial.

HINT: Select the formulas in a 'semi-random' manner.

2*.1 A very simple algorithm

Suppose we do not know much about SAT and try to find a satisfying assignment for a formula F we know admits one. What would we do? A natural way of proceeding is the following: we try a random assignment and then as long as there is some violated clause, we try assigning new values for the variables in that clause. It turns out that this very basic approach is successful in the case of formulas with small clause neighbourhoods.

Note that for the purpose of simplifying the proof of efficiency in this chapter, we prescribe an exact rule as to which violated clause to select for correction if there are multiple such clauses available. With somewhat more effort, however, it can be demonstrated that such rule is not needed.

The recursive procedure `locally_correct(C)` attempts to correct one violated clause and then invokes itself recursively. We assume that there is a global variable $\alpha \in \{0, 1\}^V$ storing the assignment currently under consideration and that the formula F and the set of variables V are, as well, globally accessible. We use the notation $\Gamma_F^+(C) := \Gamma_F(C) \cup \{C\}$ to denote the *inclusive neighbourhood* of a clause $C \in F$. Moreover, we assume that the clauses of F are somehow ordered in a fixed (but arbitrary) way and call this ordering the *lexicographic ordering*. The function is supposed to be called for some clause $C \in F$ which is currently violated under α .

```

function locally_correct(C)
  replace in  $\alpha$  the values for  $\text{vbl}(C)$  with new samples u.a.r.
  while  $\exists D \in \Gamma_F^+(C) : D$  violated under  $\alpha$  do
     $D \leftarrow$  the lexicographically first violated clause in  $\Gamma_F^+(C)$ ;
    locally_correct(D);

```

On a global scale, we start from a random point and invoke the local correction procedure as long as there is something to correct. The complete

solver is then summarized as `solve_lll(F, V)` below.

At first sight, it might look surprising that such a strategy should work. In particular, there appears to be the possibility for the recursive procedure to run indefinitely, but we will show that this won't happen. Our claim is that the algorithm `solve_lll(F, V)` has a polynomial expected running time on formulas with a clause neighbourhood of size at most 2^{k-3} .

```

function solve_lll(F, V)
   $\alpha \leftarrow$  u.a.r. from  $\{0, 1\}^V$ ;
  while  $\exists C \in F : C$  violated under  $\alpha$  do
     $C \leftarrow$  the lexicographically first violated clause in  $F$ ;
    locally_correct( $C$ );
  return  $\alpha$ ;

```

To see this, let us look at the recursive local correction procedure. We want to establish that when this procedure returns on the topmost invocation level, then it has made a genuine improvement to the assignment, in terms of the number of violated clauses.

Lemma 2*.2 *Let $\alpha \in \{0, 1\}^V$ be any assignment and $C \in F$ a clause violated under α . Suppose we call `locally_correct(C)` with α as the assignment to start from. If this procedure ever returns, then the assignment α' it has constructed satisfies all clauses which were affected by any changes made, i.e.*

$$\forall D \in F : (\exists x \in \text{vbl}(D) : \alpha(x) \neq \alpha'(x)) \rightarrow \alpha' \text{ satisfies } D$$

and in particular, α' satisfies C .

Proof. Let $V' := \{x \in V \mid \alpha(x) \neq \alpha'(x)\}$ and let $D \in F$ be any clause that contains at least one variable from V' . Consider the variable $x \in \text{vbl}(D)$ which is the last one to be resampled among all variables occurring in D . This resampling happens inside some recursive call of `locally_correct(E)` for some clause $E \in F$ with $x \in \text{vbl}(E)$. When that recursive call returns,

D , being a member of $\Gamma_F^+(E)$, must be satisfied. Ever after, none of the variables occurring in D change their value anymore. \square

From the lemma we can conclude that if a call to `locally_correct()` ever returns, then the set of clauses violated under α' is a strict subset of the set of clauses violated under α , since the clause we initially called the procedure for was violated and is now satisfied, and no new violated clauses can appear. Therefore, the outer loop in `solve_lll()` cannot be repeated any more than $|F|$, that is, a linear number of times. It remains to prove that the total number of recursive calls to `locally_correct()` cannot become too large.

Exercise 2*.2

Fewer top-level invocations

For simplicity, we only stated the weakest fact necessary, i.e. that at most $|F|$ top-level invocations are being made. Prove that in reality, the number of such invocations is even bounded by n/k , where n is the number of variables.

Exercise 2*.3

Even fewer top-level invocations

Following up on the last exercise, prove that on average, no more than $\frac{n}{8k}$ top-level invocations are made.

Exercise 2*.4

The procedure stops

Before reading on, try to prove (or at least find a plausible argument) that the probability for the algorithm to run indefinitely is zero.

2*.2 A bit of information theory

The proof of a strong bound on the number of recursive invocations bases on one of the simplest principles in information theory, namely that uniformly random data is not effectively compressible. In order to formalize this intuition, we have to say what it means to effectively compress. Suppose we fix two integers $t, t' \in \mathbf{N}$ such that $t' < t$. Now we consider a compression function $C : \{0, 1\}^t \rightarrow (\{0, 1\}^{t'} \cup \{\text{'no'}\})$ that takes as input a string of length t and outputs either a string of shorter length t' or the failure message 'no'. Its counterpart is a decompression function $D : \{0, 1\}^{t'} \rightarrow \{0, 1\}^t$ that takes a compressed bitstring of length t' and decodes the longer string of length t . What we require for soundness is that whenever $C(s) \neq \text{'no'}$ for

some $s \in \{0, 1\}^t$, then $\mathcal{D}(\mathcal{C}(s)) = s$. For practical compression, we would of course not allow the mapping to 'refuse' compressing and we would require many more things, e.g. that \mathcal{C} and \mathcal{D} be efficiently computable. Here, such aspects do not matter. The following lemma states the simple observation that there is no perfect compression.

Lemma 2*.3 *Let $t, t' \in \mathbf{N}$ with $t' < t$ and let $\mathcal{C} : \{0, 1\}^t \rightarrow (\{0, 1\}^{t'} \cup \{\text{'no'}\})$ and $\mathcal{D} : \{0, 1\}^{t'} \rightarrow \{0, 1\}^t$ be arbitrary mappings such that*

$$\forall s \in \{0, 1\}^t : \mathcal{C}(s) \neq \text{'no'} \Rightarrow \mathcal{D}(\mathcal{C}(s)) = s.$$

If $S \in \{0, 1\}^t$ is distributed uniformly at random among all bitstrings of length t , then

$$\Pr(\mathcal{C}(S) = \text{'no'}) \geq 1 - 2^{t'-t}.$$

Proof. The proof is by simple counting. There cannot be any two distinct strings $s, s' \in \{0, 1\}^t$ for which $\mathcal{C}(s) = \mathcal{C}(s') \neq \text{'no'}$ because that would imply $\mathcal{D}(\mathcal{C}(s)) = \mathcal{D}(\mathcal{C}(s'))$ and thus $\mathcal{D}(\mathcal{C}(s)) \neq s$ or $\mathcal{D}(\mathcal{C}(s')) \neq s'$. But since there are 2^t strings of length t and only $2^{t'}$ strings of length t' , at most $2^{t'}$ of the input strings can be mapped to an output other than 'no'. Therefore, if S is chosen u.a.r. from $\{0, 1\}^t$, the probability that $\mathcal{C}(S) \neq \text{'no'}$ is at most $2^{t'-t}$, as claimed. \square

In the last section, we will apply this lemma to bound the expected running time of our algorithm. The idea will be that we decorate the algorithm with a few extra statements that encode the random bits it uses in a compact fashion. Then we prove that *either* the algorithm solves our formula quickly, *or else* it effectively compresses the random bits it has received.

2*.3 Bounding the recursion

Suppose we want to record, as our algorithm runs, a compact log of which clauses are being corrected at what time. We could write down the index of each clause of which we reassign the variables. However, doing this in a trivial way wastes a certain amount of information: if there are $m = |F|$ clauses in total, then we need $\lceil \log(m) \rceil$ bits to represent a clause. But if a clause C is corrected first and then recursively a clause $D \in \Gamma_F^+(C)$, then

identifying D should need substantially fewer bits since $\Gamma_F^+(C)$ contains only a small number of clauses.

The following extended version of the algorithm makes use of this fact to produce a log in the form of a string $s \in \{0, 1\}^*$ documenting its actions. As for notation, if $F' \subseteq F$ is a set of clauses and $C \in F'$ some clause in that set, we denote by $\text{brcode}(C, F')$ the binary encoding of the *index of C in F'* , that is the number of clauses in F' that occur before C in the lexicographic ordering, padded with zeroes in such a way that the length of each code is $\lceil \log(|F'|) \rceil$. Moreover, we use the operator \circ to denote the catenation of binary strings and we assume that the function `append_to_log(..)` is some arbitrary means of output.

Whenever a recursive call is made for correction of a clause $D \in \Gamma_F^+(C)$ in the neighbourhood of a previously corrected clause C , we write the index of D in $\Gamma_F^+(C)$ to the log. We prepend this index by a single '1'-bit indicating that a deeper recursion level is being created. Let us call this a *message of type I*. Whenever `locally_correct()` finds that all clauses in the currently inspected neighbourhood are satisfied and thus returns control to a higher recursion level, we append a single '0'-bit to the log to represent this fact. We call this a *message of type II*.

On a global scale, whenever the outermost loop starts a new recursive correction process, we write the index of the clause C corrected on the topmost recursion level to the log. As there is no previous information available as to which clause this could be, we have to use a full encoding of C in F . We call this a *message of type III*. As the recursion depth before the top-level call can be reconstructed by counting previous messages of types I and II, no extra bit is needed to indicate the type of message appended. Note that the top-level call, too, will automatically output a '0'-bit in the end to indicate termination.

Let us precisely quantify the number of bits that are needed to store the log being output. A message of type I needs $\lceil \log(2^{k-3}) \rceil = k - 3$ bits for encoding the clause to be corrected next and one additional bit being prepended. A message of type II needs exactly one bit. A message of type III needs $\lceil \log m \rceil$ bits. In order to facilitate the calculation, let us note that messages of type II always occur paired up with messages of type I or III, which are uniquely associated with a specific invocation of the local correction procedure. Therefore each top-level invocation incurs a total of $\lceil \log m \rceil + 1$ bits of output, each recursive invocation then produces

an additional $(k-3) + 2 = k-1$ bits of output to the log.

```

function locally_correct(C)
  replace in  $\alpha$  the values for  $\text{vbl}(C)$  with new samples u.a.r.
  while  $\exists D \in \Gamma_F^+(C) : D$  violated under  $\alpha$  do
     $D \leftarrow$  the lexicographically first violated clause in  $\Gamma_F^+(C)$ ;
    append_to_log('1'  $\circ$  bincode( $D, \Gamma_F^+(C)$ ));
    locally_correct( $D$ );
  append_to_log('0');

function solve_III(F, V)
   $\alpha \leftarrow$  u.a.r. from  $\{0, 1\}^V$ ;
  while  $\exists C \in F : C$  violated under  $\alpha$  do
     $C \leftarrow$  the lexicographically first violated clause in  $F$ ;
    append_to_log(bincode( $C, F$ ));
    locally_correct( $C$ );
  return  $\alpha$ ;
```

Suppose the algorithm makes a total of t invocations of the correction procedure, including the top-level calls (of which a maximum of m can be done according to Lemma 2*.2). Then the size of the log we produce is smaller than $m(\lceil \log m \rceil + 1) + (k-1)t$. Note that we are very generous here and the top-level calls are being overcounted.

It is very easy to see that from the log output, it is possible to unambiguously reconstruct the entire history of clause corrections. Now, the key point of the proof follows. By just looking at the log produced, we can therefore also reconstruct histories of random values for each variable occurring in corrected clauses. Let C be the first corrected clause. The clause was corrected because it was violated in the beginning, therefore we know the initial values that all variables in $\text{vbl}(C)$ were assigned before any corrections. Suppose D is the next corrected clause, i.e. D was violated

after the correction of C . D might have some variables in common with C and some variables not encountered yet. For those not yet encountered, we can reconstruct their initial values, for the other ones we can now reconstruct the replacement values they received when correcting C . Continuing in this way, each entry of the clause correction log allows us to reconstruct the respective next values of the k variables in the clause. After reading the whole log, we will have reconstructed for each variable $x \in V$, *all* the values it has received during the procedure, *except* for its final value as our reconstruction always lags behind by one.

We now claim that for any $t \in \mathbf{N}$, the probability that the algorithm does not terminate before making t invocations of the local correction procedure is at most $2^{m(\lceil \log m \rceil + 1) - t}$. From this it immediately follows that the expected number of such invocations the algorithm makes is $\mathcal{O}(m \log m)$.

Fix some $t \in \mathbf{N}$. We incorporate our algorithm into a compression scheme consisting of functions

$$\mathcal{C} : \{0, 1\}^{n+tk} \rightarrow (\{0, 1\}^{n+(m\lceil \log m \rceil + 1)+t(k-1)} \cup \{\text{'no'}\})$$

and

$$\mathcal{D} : \{0, 1\}^{n+(m\lceil \log m \rceil + 1)+t(k-1)} \rightarrow \{0, 1\}^{n+tk}.$$

The compression \mathcal{C} of a string $s \in \{0, 1\}^{n+tk}$ is defined as follows: run the algorithm `solve_lll(F, V)`, using s to replace the randomness, i.e. the first n bits of s are used as the initial assignment and the remaining bits are used, in chunks of k bits each, for replacing values within the local correction procedure. We let the algorithm go on for up to t calls to `locally_correct(C)`. If it has not finished its job by then, we interrupt it and make \mathcal{C} output the log produced so far, which we know has size at most $m(\lceil \log m \rceil + 1) + t(k - 1)$, followed by n bits representing the current value of α when the algorithm was interrupted. If necessary, we can pad the string with zeroes for it to have the required length. If, however, the algorithm is successful in finding a satisfying assignment before reaching t invocations of the correction procedure, then we let \mathcal{C} output 'no'. This way, *either* the solver is successful *or* the compression is.

The decompression \mathcal{D} does the reverse. As we have described above, each of the t calls to `locally_correct(C)` we have recorded in our log allows for the reconstruction of k bits at determined positions from s . As further described, the only bits we have not reconstructed this way are the final

values of each variable in α . But since $C(s)$ has the n bits describing α appended to its end, we know these values too. Finally, we have reconstructed all $n + tk$ bits from s correctly and thus whenever $C(s) \neq \text{'no'}$, we have $D(C(s)) = s$.

Since this is a compression scheme satisfying the conditions in 2*.3, the probability that $C(s) = \text{'no'}$ if s is chosen uniformly at random has to be at least $1 - 2^{m(\lceil \log m \rceil + 1) - t}$. But those are exactly the cases when a satisfying assignment is output, concluding the proof of Theorem 2*.1. \square

NOTE 2*.1 The first algorithmic versions of the Lovász Local Lemma were found in breakthrough papers by Beck in [Bec91] and Alon in [Alo91]. Further progress was made in various publications over time ([MR98], [CS00], [Mos06], [Sri08], [Mos08]). The proof described in this chapter has first been given in a slightly different form in [Mos09]. After publication, it became apparent that arguments of this sort were previously known under the name of the 'incompressibility method' and had even been applied in amazingly similar fashions to Local Lemma type problems (cf., e.g., [Sch09]) however without the realisation of their important algorithmic consequences. Finally, in [MT09], generalizations of the method are provided that allow making almost all known applications of the Local Lemma beyond satisfiability constructive and that get rid of the remaining constant gap left open here. Very recently, it has been demonstrated that the method can be derandomized (see [CGH09]).

Exercise 2*.5

An more concise log

Prove that the factor of $\log m$ can be gotten rid of, i.e. that the expected running time is in fact $\mathcal{O}(m)$, by encoding the clauses for which top-level calls are made in a more compact fashion.

Exercise 2*.6

An even more concise log

Think of ways in which the log can be further compressed! Where is the description still somewhat wasteful? Consider expressing logs by objects other than binary strings. If you find the right formalism, it is possible to prove that the algorithm also works for formulas with $2^k/e$ inclusive neighbors per clause and that in fact the recursive structure of the algorithm is not necessary: a single loop suffices. You can also read the solution in [MT09].

Bibliography

- [Knu69] Donald E. Knuth. *The Art of Computer Programming*, Vol. I, Addison Wesley, London, 1969, p. 396 (Exercise 11).
- [EL75] Paul Erdős and László Lovász. *Problems and results on 3-chromatic hypergraphs and some related questions*. In A. Hajnal, R. Rado and V.T. Sós, editors, *Infinite and Finite Sets* (to Paul Erdős on his 60th birthday), volume II, pages 609-627. North-Holland, 1975.
- [Bec91] József Beck. *An Algorithmic Approach to the Lovász Local Lemma*. *Random Structures and Algorithms*, 2(4):343-365, 1991.
- [Alo91] Noga Alon. *A parallel algorithmic version of the local lemma*. *Random Structures and Algorithms*, 2(4):367-378, 1991.
- [KST93] Jan Kratochvíl and Petr Savický and Zsolt Tuza. *One more occurrence of variables makes satisfiability jump from trivial to NP-complete*. *SIAM J. Comput.*, Vol. 22, No. 1, pp. 203-210, 1993.
- [MR98] Michael Molloy and Bruce Reed. *Further Algorithmic Aspects of the Local Lemma*. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 524-529, 1998.
- [CS00] Artur Czumaj and Christian Scheideler. *Coloring non-uniform hypergraphs: a new algorithmic approach to the general Lovász local lemma*. *Symposium on Discrete Algorithms*, 30-39, 2000.

- [Mos06] Robin A. Moser. *On the Search for Solutions to Bounded Occurrence Instances of SAT*. Not published. Semester Thesis, ETH Zürich. 2006.
- [Sri08] Aravind Srinivasan. *Improved algorithmic versions of the Lovász Local Lemma*. Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms (SODA), San Francisco, California, pp. 611-620, 2008.
- [Mos08] Robin A. Moser. *Derandomizing the Lovász Local Lemma more Effectively*. Eprint arXiv:0807.2120v2, 2008.
- [Sch09] Pascal Schweitzer. *Using the Incompressibility Method to obtain Local Lemma results for Ramsey-type Problems*. Information Processing Letters, 109(4):229-232, 2009.
- [Mos09] Robin A. Moser. *A Constructive Proof of the Lovász Local Lemma*. Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, pp. Pages 343-350, 2009.
- [MT09] Robin A. Moser and Gábor Tardos. *A Constructive Proof of the General Lovász Local Lemma*. Submitted. Eprint available at arXiv:0903.0544v3, 2009.
- [CGH09] Karthekeyan Chandrasekaran, Navin Goyal and Bernhard Haeupler. *Deterministic Algorithms for the Lovász Local Lemma*. Submitted. Eprint available at arXiv:0908.0375, 2009.

Chapter 2**

Minimal Unsatisfiable Formulas

Chapter by Dominik Scheder.

A CNF formula is *minimal unsatisfiable* if it is unsatisfiable, but removing any clause makes it satisfiable. Clearly, every unsatisfiable formula has a minimal unsatisfiable subformula. For example, the formula

$$\{\{x_1\}, \{\bar{x}_1, x_2\}, \{\bar{x}_1, \bar{x}_2\}, \{x_2, x_3, x_4, x_5\}\}$$

is unsatisfiable, but not minimal unsatisfiable, because we can remove the last clause and obtain

$$\{\{x_1\}, \{\bar{x}_1, x_2\}, \{\bar{x}_1, \bar{x}_2\}\},$$

which is still unsatisfiable. This formula, however, is minimal unsatisfiable. In this chapter we will investigate minimal unsatisfiable formulas. We introduce some notation. For a CNF formula F and a literal u , we will write

$$\text{occ}_F(u) := |\{C \in F \mid u \in C\}|,$$

and for a variable x , $\deg_F(x) := \text{occ}_F(x) + \text{occ}_F(\bar{x})$, i.e. the number of clauses containing x , irrespective of its sign.

2**.1 Matchings and Formulas

Definition 2.1 (Clause-Variable Incidence Graph)** *Let F be a CNF formula. The clause-variable incidence graph is a bipartite graph on the vertex set $F \cup \text{var}(F)$, where we connect a clause C and a variable x if $x \in \text{var}(C)$.*

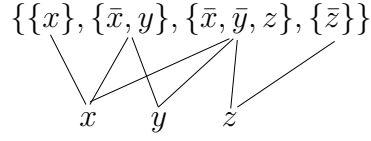


Figure 2**.1: The bipartite clause-variable incidence graph between a CNF formula F and $\text{var}(F)$.

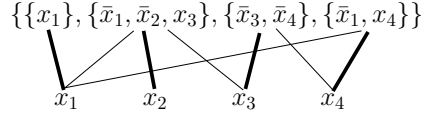


Figure 2**.2: A formula with a saturating matching. Matching edges are drawn bold.

A *matching* in a graph is a set M of edges such that each vertex is incident to at most one of them. If a vertex is incident to one of them it is *matched*, otherwise it is *unmatched*. We also say "a matching in F " as a short-hand for "a matching in the clause-variable incidence graph of F ". If G is a subformula of F and a matching M matches every clause in G , we say it *saturates* G . If it matches every clause in F , we simply say it is saturating. The following observation makes the connection with satisfiability.

Observation 2.2** *If a CNF formula F has a saturating matching, it is satisfiable. See Figure 2**.2.*

Proof . Let $F = \{C_1, \dots, C_m\}$. Since there is a matching saturating all clauses of F , there are m distinct variables x_1, \dots, x_m such that $x_i \in \text{var}(C_i)$. We define a (possibly partial) assignment α by either (i) setting x_i to 1, if $x_i \in C_i$, or (ii) setting x_i to 0, if $\bar{x}_i \in C_i$, for all $1 \leq i \leq m$. This assignment satisfies F . \square

The condition in Observation 2**.2 is, of course, only sufficient, not necessary. Take for example the formula

$$\{\{x, y\}, \{\bar{x}, y\}, \{\bar{x}, z\}, \{\bar{x}, \bar{y}, \bar{z}\}\},$$

which does not admit a saturating matching, but still is satisfiable.

There is an elegant characterization for when a formula has a matching that matches all clauses. Although this characterization holds for general bipartite graphs, we state it in terms of CNF formulas.

Theorem 2.3 (Hall's Theorem in SAT terms)** *A CNF formula F has a saturating matching if and only if for all subformulas $G \subseteq F$, it holds that*

$$|\text{var}(G)| \geq |G|. \quad (2^{**}.1)$$

It is easy to see that condition (2**.1) is necessary for the existence of such a matching. That it is also sufficient requires a proof, which we do not give here. Also note that the naive approach to test whether condition (2**.1) holds, namely iterating over all subformulas, would take $2^{|F|}$ steps. Fortunately, there are efficient algorithms for finding maximum matchings in graphs, and thus determining whether a formula F has a matching saturating F , and also finding such a matching. Therefore, if F fulfills the precondition of Observation 2**.2, we can efficiently find a satisfying assignment.

This discussion motivates some definitions. For a CNF formula F , we call $\delta(F) := |F| - |\text{var}(F)|$ the *deficiency* of F , and $\delta^*(F) := \max_{G \subseteq F} \delta(G)$ the *maximum deficiency* of F . Observation 2**.2 can now be re-stated as follows: Every CNF formula F with $\delta^*(F) \leq 0$ is satisfiable. What does $\delta(F)$ tell us about satisfiability? At first sight, pretty little. Take for example the formula

$$\{\{x_1\}, \{\bar{x}_1, x_2\}, \{\bar{x}_1, \bar{x}_2\}, \{x_2, x_3, x_4, x_5\}\}. \quad (2^{**}.2)$$

Its deficiency is $4 - 5 = -1$, but still it is unsatisfiable. Sure, it contains the subformula $\{\{x_1\}, \{\bar{x}_1, x_2\}, \{\bar{x}_1, \bar{x}_2\}\}$, which is unsatisfiable (and has deficiency 1). Does every unsatisfiable formula F contain an unsatisfiable subformula $G \subseteq F$ with $\delta(G) > 0$? The answer turns out to be yes.

Lemma 2.4** *If F is a minimal unsatisfiable CNF formula, then for every proper subformula $G \subsetneq F$, it holds that $\delta(G) \leq \delta(F) - 1$.*

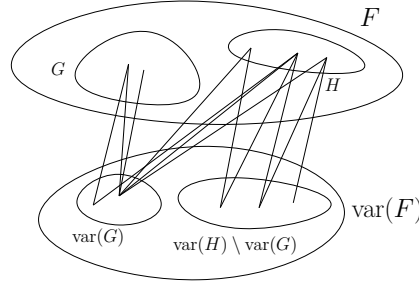


Figure 2**.3: Illustration of the proof of Lemma 2**.4

Proof . Suppose this does not hold, i.e. F is minimal unsatisfiable, and there exists a formula $G \subsetneq F$ with $\delta(G) \geq \delta(F)$. Choose G to be inclusion-wise maximal with this property. Consider any $H \subseteq F \setminus G$. See Figure 2**.3 for an illustration. If $H \subsetneq F \setminus G$, then by maximality of G we conclude $\delta(G \cup H) \leq \delta(F) - 1$. If $H = F \setminus G$, then $\delta(G \cup H) = \delta(F)$. Hence in any case $\delta(G \cup H) \leq \delta(F)$. We can now calculate:

$$\begin{aligned}
 \delta(F) &\geq \delta(G \cup H) = |G \cup H| - |\text{var}(G \cup H)| = \\
 &= |G| + |H| - |\text{var}(G)| - |\text{var}(H) \setminus \text{var}(G)| \\
 &= \delta(G) + |H| - |\text{var}(H) \setminus \text{var}(G)| .
 \end{aligned}$$

Since $\delta(G) \geq \delta(F)$, we conclude that $|\text{var}(H) \setminus \text{var}(G)| \geq |H|$ for all $H \subseteq F \setminus G$. By Hall's Theorem, this means we can find a matching in F that (i) matches all clauses of $F \setminus G$ and (ii) only matches variables that do not occur in G . We can thus define a partial assignment $\alpha : \text{var}(F) \setminus \text{var}(G) \rightarrow \{0, 1\}$ that satisfies $F \setminus G$. Since F is minimal unsatisfiable, G is satisfiable, hence let $\beta : \text{var}(G) \rightarrow \{0, 1\}$ satisfy G . Since α and β are defined over disjoint sets of variables, we can combine them to obtain a satisfying assignment for F . This is a contradiction. \square

We conclude that for any minimal unsatisfiable formula F , it holds that $\delta(F) = \delta^*(F) \geq 1$. Are there minimal unsatisfiable formulas with deficiency 1? Actually, we have already seen one: $\{\{x_1\}, \{\bar{x}_1, x_2\}, \{\bar{x}_1, \bar{x}_2\}\}$. There is an even simpler one: $\{\square\}$. We introduce some notation: We denote by MU the class of all minimal unsatisfiable formulas, and define $\text{MU}(k) := \{F \in \text{MU} \mid \delta(F) = k\}$. By the above results, we see that $\text{MU}(0)$, $\text{MU}(-1)$, $\text{MU}(-2)$

and so on are all empty. It turns out that formulas in MU(1) have very interesting properties, besides having the smallest possible deficiency.

2**.2 Strongly Minimal Unsatisfiable Formulas

Removing clauses from a formula makes the formula "more satisfiable" in the following sense: If it was satisfiable before, it is still satisfiable afterwards. Conversely, adding clauses makes it "less satisfiable". Now suppose we pick some clause in the formula, and add some new literal to it. Surely, the resulting formula is more satisfiable: If the old one was satisfiable, the new one is as well. Similarly, removing literals makes a formula only less satisfiable. Let us make this very vague discussion more precise.

Definition 2.5** *Let F be a CNF formula. A CNF formula F' is called an extension of F if there is a surjective function $\Phi : F \rightarrow F'$ such that*

$$\forall C \in F : C \subseteq \Phi(C) .$$

If $F' \neq F$, we say F' is a proper extension of F , otherwise it is a trivial extension.

Example. Consider the two formulas

$$\begin{aligned} F &= \{\{x\}, \{\bar{x}, y, z\}, \{\bar{x}, \bar{y}, \bar{z}\}\} \\ F' &= \{\{x, y\}, \{\bar{x}, y, z\}, \{\bar{x}, \bar{y}, \bar{z}\}\} \end{aligned}$$

Here, F' is an extension of F , and, as it happens, both are satisfiable. We justify the above statement that an extension is always "more satisfiable" than the old formula by the following observation.

Observation 2.6** *Let F' be an extension of F . Any assignment α satisfying F also satisfies F' .*

The function Φ in Definition 2**.5 need not be injective. In fact, by adding literals to clauses, two clauses can become the same. Consider for example

$$\begin{aligned} F &= \{\{x\}, \{y\}, \{\bar{x}, \bar{y}\}\} \\ F' &= \{\{x, y, z\}, \{\bar{x}, \bar{y}\}\} . \end{aligned}$$

F' is an extension of F , but by adding the literals y and z to the clause $\{x\}$ and the literals x and z to $\{y\}$, these two clauses become the same in F' . In general, for a CNF formula F and an extension F' of it, it holds that

$$\begin{aligned} |F| &\geq |F'| \\ \text{var}(F) &\subseteq \text{var}(F') , \end{aligned}$$

and both inequalities can be strict, as the above example shows. However, under certain circumstances we will have equality.

Lemma 2.7** *Suppose F is minimal unsatisfiable and F' is an extension of F . If F' is unsatisfiable, then*

$$|F| = |F'| \tag{2**.3}$$

$$\text{var}(F) = \text{var}(F') . \tag{2**.4}$$

Proof . Let F be minimal unsatisfiable, and F' be an extension of F . To show (2**.3), we assume that $|F'| < |F|$ and show that F' is satisfiable. Let $\Phi : F \rightarrow F'$ be as in Definition 2**.5. For every clause $C' \in F'$, pick one clause $C \in F$ with $C' = \Phi(C)$. Collect those clauses in a CNF formula G . It holds that $G \subseteq F$ and $|G| = |F'|$. Since $|F'| < |F|$ this means that $G \subsetneq F$. But F is minimal unsatisfiable, so G is satisfiable. Observe that F' is also an extension of G , via the same function Φ , and by Observation 2**.6, F' is satisfiable. To show (2**.4), assume that $\text{var}(F) \subsetneq \text{var}(F')$ and let $x \in \text{var}(F') \setminus \text{var}(F)$. We will show that F' is satisfiable. There is a clause $C' \in F'$ with $x \in \text{var}(C')$. We choose a truth value $b \in \{0, 1\}$ as follows: If $x \in C'$, we let $b = 1$, and if $\bar{x} \in C'$, let $b = 0$. In any case, setting x to b satisfies C' . We observe that $F'^{[x \mapsto b]}$ is an extension of some proper subformula $G \subsetneq F$. Hence G is satisfiable, and by Observation 2**.6, F' is.

□

Definition 2.8** *A CNF formula is called strongly minimal unsatisfiable if it is unsatisfiable, but every proper extension of it is satisfiable.*

The term "strongly minimal unsatisfiable" suggests that a strongly minimal unsatisfiable CNF formula is automatically minimal unsatisfiable. This is indeed the case, and we will prove it in a minute. We denote

by SMU the class of all strongly minimal unsatisfiable formulas. Again, $\text{SMU}(k)$ denotes those SMU-formulas with deficiency k .

Example. $F_1 = \{\{\bar{x}\}, \{x, \bar{y}\}, \{x, y, z\}, \{x, y, \bar{z}\}\}$.

We claim that F_1 is minimal unsatisfiable (in fact, it is in $\text{SMU}(1)$). We could do this by a detailed case analysis, but there is a much more elegant way. We evaluate

$$\sum_{C \in F_1} 2^{-|C|} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1.$$

If F'_1 is a proper extension of F_1 , it is immediate that $\sum_{C' \in F'_1} 2^{-|C'|} < 1$, and by Theorem 2.2, F'_1 is satisfiable. Be warned though that the above sum does not in general evaluate to 1 for SMU formulas. For example,

$$F_2 = \{\{\bar{x}_1, x_2\}, \{\bar{x}_2, x_3\}, \{\bar{x}_3, x_4\}, \{\bar{x}_4, x_1\}, \{x_1, x_2, x_3, x_4\}, \{\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4\}\}$$

is strongly minimal unsatisfiable (on a rainy saturday afternoon, you should check this), but $\sum_{C \in F_2} 2^{-|C|} = \frac{9}{8}$.

Observation 2.9** *Every strongly minimal unsatisfiable formula is also minimal unsatisfiable. In other words, the term strongly minimal is justified.*

Proof. Let F be strongly minimal unsatisfiable. We will show that for any $C \in F$, the formula $F \setminus \{C\}$ is satisfiable. Let x be a new variable, i.e., $x \notin \text{var}(F)$, and define $F' := F \setminus \{C\} \cup \{C \cup \{x\}\}$. Since F' is a proper extension of F , it is by assumption satisfiable. Furthermore, $F \setminus \{C\}$ is satisfiable, since it is a subformula of F' \square

Every minimal unsatisfiable CNF formula F has a strongly minimal unsatisfiable extension F' . To see this, we define a sequence F_1, F_2, \dots of extensions of F as follows: Start with $F_1 = F$. If F_i is SMU, it is a strongly minimal unsatisfiable extension of F . Otherwise, there is a proper extension of F_i that is unsatisfiable. Let F_{i+1} be that formula. By Lemma 2**.7, $\text{var}(F_i) = \text{var}(F)$ for all i . Since there are only finitely many formulas over $\text{var}(F)$, the process terminates eventually. We summarize this argument:

Observation 2.10** *Every minimal unsatisfiable CNF formula F has a (not necessarily unique) strong minimal unsatisfiable extension F' . Furthermore, $|F| = |F'|$ and $\text{var}(F) = \text{var}(F')$ for any such extension.*

If F is an unsatisfiable formula, and $x \in \text{var}(F)$, then clearly $F^{[x \mapsto 0]}$ and $F^{[x \mapsto 1]}$ are both unsatisfiable. Now suppose F is *minimal* unsatisfiable. Are $F^{[x \mapsto 0]}$ and $F^{[x \mapsto 1]}$ also minimal unsatisfiable? Unfortunately, this is not the case: For example, the formula

$$\{\{x\}, \{\bar{x}, \bar{y}\}, \{y\}\}$$

is minimal unsatisfiable. When we set x to 0, we obtain $\{\square, \{y\}\}$, which is clearly not minimal unsatisfiable. For consolation, we do however have following lemma.

Lemma 2.11** *Let F be a strongly minimal unsatisfiable CNF formula, $x \in \text{var}(F)$, and $b \in \{0, 1\}$. Then $F^{[x \mapsto b]}$ is minimal unsatisfiable.*

Proof. Assume $b = 0$, the other case is similar. Clearly that $F^{[x \mapsto 0]}$ is unsatisfiable. We will show that for any $C \in F^{[x \mapsto 0]}$, $F^{[x \mapsto 0]} \setminus \{C\}$ is satisfiable. There is a clause $D \in F$ such that $C = D^{[x \mapsto 0]}$, so either $D = C$ or $D = C \cup \{x\}$. In the first case, consider $F' := F \setminus \{D\} \cup \{D \cup \{\bar{x}\}\}$. This is a proper extension of F , hence it is satisfiable, so at least one of $F'^{[x \mapsto 0]}$ or $F'^{[x \mapsto 1]}$ is satisfiable. Note that $F'^{[x \mapsto 1]} = F^{[x \mapsto 1]}$, which is unsatisfiable. Hence $F'^{[x \mapsto 0]}$ is satisfiable (in other words, if adding literal u to a clause makes F satisfiable, then every satisfying assignment must also satisfy u). But $F'^{[x \mapsto 0]} = F^{[x \mapsto 0]} \setminus \{C\}$, so the latter is satisfiable. In the other case, $D = C \cup \{x\}$. Consider $F'' := F \setminus \{D\}$. Since F is, by Observation 2**.9, minimal unsatisfiable, F'' is satisfiable, and again at least one of $F''^{[x \mapsto 0]}$ and $F''^{[x \mapsto 1]}$ is satisfiable. The latter equals $F^{[x \mapsto 1]}$, which is unsatisfiable, thus $F''^{[x \mapsto 0]}$ is satisfiable. But this equals $F^{[x \mapsto 0]} \setminus \{C\}$. \square

We will now prove a surprising fact that, as it turns out, will completely describe the structure of MU(1) formulas.

Theorem 2.12** *Let F be an MU(1) formula. Then either $F = \{\square\}$, or there exists a variable x such that $\text{occ}_F(x) = \text{occ}_F(\bar{x}) = 1$.*

Proof. We will first prove the theorem for the case that F is SMU(1). Choose a variable $x \in \text{var}(F)$ that minimizes $d_F(x)$. We claim that

$$\text{var}(F^{[x \mapsto 0]}) = \text{var}(F) \setminus \{x\}. \quad (2**.5)$$

In fact, suppose this is not the case, and let $y \in \text{var}(F) \setminus \{x\} \setminus \text{var}(F^{[x \mapsto 0]})$. How can it happen that y appears in F but not in $F^{[x \mapsto 0]}$? The only possibility is that every clause containing y or \bar{y} also contains \bar{x} , and is satisfied and disappears when setting x to 0. This implies

$$d_F(y) \leq \text{occ}_F(\bar{x}) < d_F(x) ,$$

contradicting the choice of x . The latter inequality comes from the fact that in a minimal unsatisfiable formula contains no *pure* literals, i.e. if \bar{x} occurs, then x occurs as well. We conclude that (2**.5) holds. Now the proof is just a calculation (we will explain all inequalities afterwards):

$$1 \leq \delta(F^{[x \mapsto 0]}) \quad (2**.6)$$

$$= |F^{[x \mapsto 0]}| - |\text{var}(F^{[x \mapsto 0]})|$$

$$= |F| - \text{occ}_F(\bar{x}) - |\text{var}(F)| + 1 \quad (2**.7)$$

$$= 2 - \text{occ}_F(\bar{x})$$

$$\Rightarrow \text{occ}_F(\bar{x}) = 1$$

Inequality (2**.6) comes from the fact that $F^{[x \mapsto 0]}$ is minimal unsatisfiable (Lemma 2**.11) and that minimal unsatisfiable formulas have deficiency at least one (Lemma 2**.4). Equality (2**.7) comes from (2**.5) combined with the observation that the only clauses that disappear when setting x to 0 are those containing \bar{x} , and that, in a minimal unsatisfiable formula, no two clauses "collapse", i.e., become identical, when setting a variable. We conclude that $\text{occ}_F(\bar{x}) = 1$. A similar argument shows that $\text{occ}_F(x) = 1$. Hence we have showed the theorem for *strongly* minimal unsatisfiable formulas.

If F is minimal unsatisfiable, but not strongly minimal unsatisfiable, let F' be a strongly minimal unsatisfiable extension of F . By the above argument, there exists a variable $x \in \text{var}(F')$ with $\text{occ}_{F'}(x) = \text{occ}_{F'}(\bar{x}) = 1$. By Observation 2**.10, $\text{var}(F') = \text{var}(F)$, thus $x \in \text{var}(F)$. Since $\text{occ}_F(u) \leq \text{occ}_{F'}(u)$ for any literal u , and x is not pure in F , it follows that $\text{occ}_F(x) = \text{occ}_F(\bar{x}) = 1$. \square

2**.3 More On Resolution

We have already seen resolution as a complete and correct method for determining unsatisfiability of formulas. Let F be a CNF formula and $C, D \in F$ be two clauses. If there is a unique literal u with $u \in C$ and $\bar{u} \in D$, then the *resolvent* of C and D is defined to be $\text{res}(C, D) := (C \setminus \{u\}) \cup (D \setminus \{\bar{u}\})$, and $\text{var}(u)$ is called the *resolved variable*. It is easy to check that $F \equiv F \cup \{\text{res}(C, D)\}$, in the sense that any total assignment $\alpha : \text{var}(F) \mapsto \{0, 1\}$ satisfies F if and only if it satisfies $F \cup \{\text{res}(C, D)\}$. A *resolution refutation* of a formula F is defined to be a sequence C_1, \dots, C_m with $C_m = \square$ such that for every $1 \leq j \leq m$, either $C_j \in F$, or there are indices $i, i' < j$ such that $C_j = \text{res}(C_i, C_{i'})$. Such a refutation can naturally be represented as a tree.

Definition 2.13** *Let F be a CNF formula. A resolution tree of F is a complete binary tree T in which each node a is labeled with a clause C_a such that*

- *if a is a leaf of T , then $C_a \in F$,*
- *if a has children b and c , then C_a is the resolvent of C_b and C_c .*

If the root of T is labeled with clause C , we say T ends in C .

Further, each inner node a of T is naturally associated with a variable: If b and c are the children of a , and $C_a = (C_b \setminus \{u\}) \cup (C_c \setminus \{\bar{u}\})$, we say the variable $\text{var}(u)$ is *resolved at node a* . We write x_a to denote that variable.

Observation 2.14** *A CNF formula F is unsatisfiable if and only if it has a resolution tree ending in \square .*

2**.3.1 Davis-Putnam Resolution

We introduce variant of resolution, called *Davis-Putnam resolution*. Let F be a CNF formula, and let $x \in \text{var}(F)$. Write $r := \text{occ}_F(x)$ and $s := \text{occ}_F(\bar{x})$ and let C_1, \dots, C_r be the clauses of F containing the positive literal x , and let D_1, \dots, D_s be the clauses of F containing \bar{x} . For $1 \leq i \leq r$ and $1 \leq j \leq s$, we define

$$B_{ij} := C_i \setminus \{x\} \cup D_j \setminus \{\bar{x}\},$$

the resolvent of C_i and D_j . Note that it is possible that B_{ij} contains complementary literals, for example if $y \in C_i$ and $\bar{y} \in D_j$ for some $y \neq x$. In this case we say B_{ij} is trivial. We obtain a formula $DP_x(F)$ by removing all C_i and D_j and adding all non-trivial resolvents B_{ij} . Formally,

$$DP_x(F) := F \setminus \{C_1, \dots, C_r, D_1, \dots, D_s\} \\ \cup \{B_{ij} \mid 1 \leq i \leq r, 1 \leq j \leq s, B_{ij} \text{ non-trivial}\} \quad (2**.8)$$

We say we obtain $DP_x(F)$ from F by applying Davis-Putnam resolution with respect to x . Contrary to resolution as we are used to, we do not only add the resolvents, but remove the original clauses. This is justified by the following fact:

Lemma 2.15 (Correctness of Davis-Putnam resolution)** $F \equiv_{SAT} DP_x(F)$.

Proof. One direction is easy. Suppose α satisfies F . As we already know, α satisfies all possible resolvents, hence satisfies $DP_x(F)$. For the other direction, suppose α satisfies $DP_x(F)$. Since $x \notin DP_x(F)$, we assume α does not assign any value to x . We claim that at least one of the following holds: (i) α satisfies all clauses $C_i \setminus \{x\}$, $1 \leq i \leq r$, or (ii) α satisfies all clauses $D_j \setminus \{\bar{x}\}$, $1 \leq j \leq s$. Indeed, if this were not the case, then there would be i and j such that α does not satisfy B_{ij} . This B_{ij} is non-trivial, since α satisfies every trivial resolvent. Hence $B_{ij} \in DP_x(F)$, contradicting the assumption that α satisfies $DP_x(F)$. So if (i) holds, then the combination $\alpha[x \mapsto 0]$ satisfies F . If (ii) holds, then $\alpha[x \mapsto 1]$ satisfies F . \square

Exercise 2.1** Use Lemma 2**.15 to give an alternative proof of the completeness of resolution (of resolution, not DP-resolution).

Exercise 2.2** Convince yourself that Lemma 2**.15 holds as well for the extreme cases where $occ_F(x) = 0$ and/or $occ_F(\bar{x}) = 0$.

Exercise 2.3** Give an example of a CNF formula F with $x \in \text{var}(F)$ such that F and $DP_x(F)$ are not equivalent, i.e. $\text{sat}_V(F) \neq \text{sat}_V(DP_x(F))$ for $V = \text{var}(F)$.

There is an even more special kind of resolution: Suppose $occ_F(x) = 1$ or $occ_F(\bar{x})$. In this case, $|DP_x(F)| < |F|$. We say $DP_x(F)$ is obtained from F by *singular resolution*.

Lemma 2.16** *Suppose F is a CNF formula, u a literal with $\text{occ}_F(u) = 1$. Let $x = \text{var}(u)$ and $F' := \text{DP}_x(F)$. Then $F \in \text{MU}$ if and only if $F' \in \text{MU}$ and $|F'| = |F| - 1$. Further, if $F \in \text{MU}$, then $\delta(F) = \delta(F')$.*

Proof. Let $s := \text{occ}_F(\bar{u})$ and write $F = \{C, D_1, \dots, D_s\} \cup R$, $u \in C$, $\bar{u} \in D_j$ for $1 \leq j \leq s$, and R is the “rest”, i.e. the clauses neither containing u nor \bar{u} . We set

$$B_j := C \setminus \{u\} \cup D_j \setminus \{\bar{u}\},$$

such that

$$F' := \{B_j \mid 1 \leq j \leq s, B_j \text{ not trivial}\} \cup R.$$

For the “only if” direction, suppose $F \in \text{MU}$. We already know from Lemma 2**.15 that F' is unsatisfiable. We will show that $|F'| = |F| - 1$ holds. We claim that (i) all B_j are nontrivial, (ii) all B_j are distinct, and (iii) $B_j \notin R$, for all $1 \leq j \leq s$. If (i) is violated, then some B_j is trivial. If (ii) is violated, then there are $j \neq j'$ with $B_j = B_{j'}$. If (iii) is violated, then there is some j with $B_j \in R$. In any case, there is some B_j such that $F \equiv_{\text{SAT}} \text{DP}_x(F) = \text{DP}_x(F \setminus \{B_j\}) \equiv_{\text{SAT}} F \setminus \{B_j\}$, which is a contradiction, since a CNF formula in MU cannot be SAT-equivalent to a subformula. It remains to show that every proper subformula of F' is satisfiable. Let $E' \in F'$ be a clause. We show that $F' \setminus \{E'\}$ is satisfiable. If $E' = B_j$ for some j , define $E := D_j$. If $E' \in R$, define $E := E'$. In any case $E \in F$, and $F' \setminus \{E'\} \subseteq \text{DP}_x(F \setminus \{E\}) \equiv_{\text{SAT}} F \setminus \{E\}$. The latter formula is satisfiable, since F is minimal unsatisfiable. The inclusion \subseteq is actually an equality, which follows from points (i), (ii) and (iii) above. The equivalence \equiv_{SAT} follows from Lemma 2**.15.

For the “if” direction, suppose $F' \in \text{MU}$, and $|F'| = |F| - 1$. The latter implies that all B_j are nontrivial, distinct, and not in R . We will show that $F \in \text{MU}$. Let $E \in F$ be a clause. If $E \in R$, then $F \setminus \{E\} \equiv_{\text{SAT}} \text{DP}_x(F \setminus \{E\}) = \text{DP}_x(F) \setminus \{E\}$. The last equality holds since $E \neq D_j$, for any j . The latter formula is a proper subset of F' , hence satisfiable. If $E = D_j$, then $F \setminus \{D_j\} \equiv_{\text{SAT}} \text{DP}_x(F \setminus \{D_j\}) = \text{DP}_x(F) \setminus \{B_j\}$, where in the last equality we use the fact that all B_j are non-trivial, distinct, and not in R . The latter formula is a proper subset of $\text{DP}_x(F)$, thus satisfiable, hence $F \setminus \{B_j\}$ is satisfiable, too. Finally, $F \setminus \{C\}$ is easily seen to be satisfiable: In this formula, \bar{u} is a pure literal. We satisfy it, and arrive at the formula R , which is satisfiable.

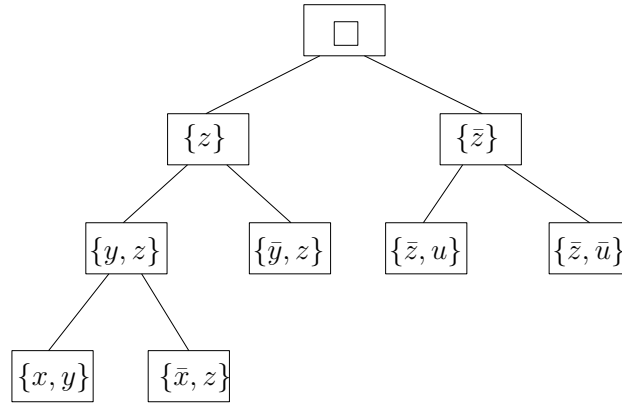


Figure 2**.4: A strict resolution tree of the formula $\{x, y\}, \{\bar{x}, z\}, \{\bar{y}, z\}, \{\bar{z}, u\}, \{\bar{z}, \bar{u}\}$.

Finally, we show that $F \in \text{MU}$ implies $\delta(F) = \delta(F')$. We already know that $|F'| = |F| - 1$. Since every B_j is non-trivial, no variable “gets lost”, i.e. $\text{var}(F') = \text{var}(F) \setminus \{x\}$. Therefore $\delta(F) = \delta(F')$. \square

Exercise 2.4** Show that the assumption $\text{occ}_F(u) = 1$ in Lemma 2**.16 is necessary: Give an example of a CNF formula $F \in \text{MU}$ with $\text{occ}_F(x) = \text{occ}_F(\bar{x}) = 2$, such that $\text{DP}_x(F) \notin \text{MU}$.

To go one step further, an even more restricted version of Davis-Putnam resolution could require that $\text{occ}_F(x) = \text{occ}_F(\bar{x}) = 1$. Of course, Lemma 2**.16 covers this, as well. In this case, we say $F' = \text{DP}_x(F)$ has been obtained from F by applying $(1, 1)$ -resolution and $F \rightarrow_{\text{DP}}^1 F'$.

We define a type of “especially beautiful” resolution trees that are closely related to $(1, 1)$ -resolution. Recall that in a resolution tree T , every inner node a corresponds to the variable x_a that is resolved at that node.

Definition 2.17** Let T be a resolution tree ending in clause C . We say T is strict if no variable is resolved at more than one node of T , and no variable of C is resolved in T (See Figure 2**.4).

Clearly, if T ends in the empty clause, then the second condition in Definition 2**.17 is void.

Exercise 2.5** *Show that in a strict resolution tree, no two nodes can be labeled with the same clause.*

2**.3.2 The Structure of MU(1)-Formulas

We will show a connection between strict resolution trees, (1, 1)-resolution and MU(1)-formulas: These three notions are (almost) equivalent.

Theorem 2.18** *The following are equivalent for any CNF formula F .*

(i) *F is minimal unsatisfiable, and there is a sequence*

$$F = F_1 \rightarrow_{\text{DP}}^1 F_2 \rightarrow_{\text{DP}}^1 \dots \rightarrow_{\text{DP}}^1 F_m = \{\square\}.$$

(ii) *There is a strict resolution tree T ending in \square such that F is exactly the set of clauses appearing as labels at the leaves of T .*

(iii) *$F \in \text{MU}(1)$.*

Proof. We first show (iii) \Rightarrow (i), which is the most surprising fact. We proceed by induction on $|\text{var}(F)|$. If this is 0, then $F = \{\square\}$, and we are done. By Theorem 2**.12, there is a variable x with $\text{occ}_F(x) = \text{occ}_F(\bar{x}) = 1$, and we let $F' := \text{DP}_x(F)$. By Lemma 2**.16, $F' \in \text{MU}$, and $\delta(F') = \delta(F)$, therefore $F' \in \text{MU}(1)$. By induction, there is a sequence $F' = F_2 \rightarrow_{\text{DP}}^1 \dots \rightarrow_{\text{DP}}^1 F_m = \{\square\}$, and the claim follows.

To show (i) \Rightarrow (ii), we use induction on m , the length of the sequence. If $m = 1$, then $F = \{\square\}$ and an isolated root vertex labeled with \square will serve as T . Otherwise, induction yields a strict resolution tree T_2 for F_2 . Let $C \in F_2$ be the clause obtained by singular resolution from $D_1, D_2 \in F_1$, and let x be the resolved variable. Exactly one leaf of T_2 is labeled by C . We append two children to it and label them with D_1 and D_2 , respectively. This new tree is a strict resolution tree for T_1 , since $x \notin \text{var}(F_2)$ and thus does not appear in T_2 .

To show (ii) \Rightarrow (iii), we use induction on the size of T . If T consists of the root only, we are done. Otherwise, pick a node in T both whose children are leaves. Let those children be labeled with C and D . Their

parent is labeled with $\text{res}(C, D)$. Let x be the resolved variable. Since T is strict, x cannot appear at any other nodes, and $\text{occ}_F(x) = \text{occ}_F(\bar{x}) = 1$. Therefore $F \rightarrow_{\text{DP}}^1 \text{DP}_x(F) =: F'$. By removing these two leaves from T , we obtain a strict resolution tree T' for F' . By induction, $F' \in \text{MU}(1)$, and clearly $|F'| = |F| - 1$. By Lemma 2**.16, $F \in \text{MU}(1)$, too. \square

This gives us an efficient algorithm for deciding whether a given CNF formula F is in $\text{MU}(1)$: Apply $(1, 1)$ -resolution as long as possible. If the deficiency does not stay 1 throughout this process, F is not $\text{MU}(1)$. Otherwise, F is $\text{MU}(1)$ if and only if the process terminates in $\{\square\}$.

2**.4 Short Resolution Proofs

A resolution refutation of a CNF formula F is a proof that F is unsatisfiable. Since we believe (but do not know) that $\text{NP} \neq \text{coNP}$, we cannot expect every unsatisfiable CNF formula to have a small resolution refutation. In fact, although we cannot yet prove that $\text{NP} \neq \text{coNP}$, one can prove that certain formulas require exponentially large resolution refutations. In this section we will show that $\text{MU}(k)$ -formulas have short resolution refutations. More precisely, we want to prove the following theorem:

Theorem 2.19** *Every CNF formula $F \in \text{MU}(k)$ has a resolution refutation of size $O(|F|^2 2^{k-1})$. More generally, every unsatisfiable CNF formula F has a resolution refutation of size $O(|F|^2 2^{\delta^*(F)-1})$.*

Please recall the definition $\delta^*(F) = \max_{G \subseteq F} \delta(G)$. The second statement follows immediately from the first: If F is unsatisfiable, there is some minimal unsatisfiable $G \subseteq F$. Clearly $\delta(G) \leq \delta^*(F)$, and by the first statement, G has a resolution refutation of size $O(n^2 2^{\delta(G)-1})$. This is also a refutation for F . Using the methods we have developed so far, the proof of the theorem is not difficult.

*Proof of Theorem 2**.19.* Let us denote by $\text{res-size}(F)$ the smallest m such that there is a resolution refutation C_1, \dots, C_m such that $C_m = \square$ of F . We will show that if $F \in \text{MU}(k)$, then $\text{res-size}(F) \leq 2^{k-1}|F|^2$. We use induction on $|F|$ and k . If $k = 1$, then $F \in \text{MU}(1)$ and has a strict resolution tree. This translates into a resolution proof of size $2|F| - 1$. So assume

$F \in \text{MU}(k)$ for some $k \geq 2$. We distinguish two cases:

Case 1. If F contains a literal u with $\text{occ}_F(u) = 1$, let $s := \text{occ}_F(\bar{u})$. Let C be the unique clause containing u , and D_1, \dots, D_s the clauses containing \bar{u} . We can apply singular resolution and obtain F' . By Lemma 2**.16, $F' \in \text{MU}(k)$. It is smaller than F , hence by induction there exists a resolution refutation $C'_1, \dots, C'_m = \square$ for F' , with $m \leq 2^{k-1}|F'|^2$. The sequence $C, D_1, \dots, D_s, C'_1, \dots, C'_m$ is a resolution refutation of F of length $m + s + 1 \leq 2^{k-1}|F'|^2 + |F| \leq 2^{k-1}|F|^2$.

Case 2. This is the more interesting case. Suppose there is no literal u with $\text{occ}_F(u) = 1$. We let F' be a strongly minimal extension of F . As we know, $|F'| = |F|$ and $\text{var}(F) = \text{var}(F')$, hence $F' \in \text{SMU}(k)$. Furthermore, there does not exist any literal u with $\text{occ}_{F'}(u) = 1$. We claim that $\text{res-size}(F) \leq \text{res-size}(F')$. An exercise below asks you to verify this. The theorem will follow in a straightforward manner from the following lemma:

Lemma 2.20** *Let $F \in \text{SMU}(k)$ for $k \geq 2$, and suppose $\text{occ}_F(u) \neq 2$ for any literal u . Then for a variable $x \in \text{var}(F)$ minimizing $\deg_F(x)$ and for any $b \in \{0, 1\}$, it holds that $F^{[x \mapsto b]}$ is minimal unsatisfiable and $\delta(F^{[x \mapsto b]}) < k$.*

Applying this lemma to F' , the strongly minimal unsatisfiable extension of F , we see that $F_0 := F'^{[x \mapsto 0]}$ and $F_1 := F'^{[x \mapsto 1]}$ are both minimal unsatisfiable, and $\delta(F_0), \delta(F_1) < k$. By induction, we obtain two resolution refutations of F_0 and F_1 of length at most $2^{k-2}(|F| - 2)^2$ each. The term $|F| - 2$ comes from the fact that $|F_0|, |F_1| \leq |F| - 2$. These refutations translate into resolution derivations of the clauses $\{x\}$ and $\{\bar{x}\}$, respectively (as in the proof of completeness of resolution). We combine those two derivations, add a last resolution step to obtain \square . This is a refutation of size at most $2^{k-1}|F|^2$. It remains to prove the above lemma.

*Proof of Lemma 2**.20.* We already know from Lemma 2**.11 that $F^{[x \mapsto b]}$ is minimal unsatisfiable. It remains to show that its deficiency is strictly smaller than that of F . We proceed in a similar fashion as in the proof of Theorem 2**.12. Assume without loss of generality that $b = 0$. First note that $\text{var}(F^{[x \mapsto 0]}) = \text{var}(F) \setminus \{x\}$. If there were a variable $y \neq x$ with $y \in \text{var}(F) \setminus \text{var}(F^{[x \mapsto 0]})$, then $\deg_F(y) \leq \text{occ}_F(\bar{x}) < \deg_F(x)$, contradicting

the choice of x . We calculate:

$$\begin{aligned}\delta(F^{[x \mapsto 0]}) &= |F^{[x \mapsto 0]}| - |\text{var}(F^{[x \mapsto 0]})| = \\ &= |F| - \text{occ}_F(\bar{x}) - |\text{var}(F)| + 1 = \\ &= \delta(F) - \text{occ}_F(\bar{x}) + 1 .\end{aligned}$$

Since $\text{occ}_F(\bar{x}) \geq 2$ by assumption, the claim follows. \square

This also finishes the proof of Theorem 2**.19. \square

Exercise 2.6** Let F be minimal unsatisfiable, and let F' be a strongly minimal unsatisfiable extension of F . Show that $\text{res-size}(F) \leq \text{res-size}(F')$.

Exercise 2.7** Prove the following fact implicitly used in the last paragraph of the proof of Theorem 2**.19: Let F be a CNF formula, $x \in \text{var}(F)$. If $F^{[x \mapsto 0]}$ has a resolution derivation of length m of the clause C , then F has a resolution derivation of $C \cup \{x\}$ of length at most m .

2**.5 Universal Patterns For Unsatisfiability

Consider the formula $F_1 = \{\{x, y\}, \{\bar{x}, z\}, \{\bar{y}, z\}, \{\bar{x}, \bar{w}\}\}$, which is satisfiable. You might agree that the formula $F_2 = \{\{a, b\}, \{\bar{a}, c\}, \{\bar{b}, c\}, \{\bar{a}, \bar{d}\}\}$ is, in some way, the same as F_1 . Sure, we simply renamed x by a , y by b and so on. Clearly, when doing so, we should be consistent, meaning renaming \bar{x} by \bar{a} . Also, the formula

$$F_3 = \{\{\bar{x}, b\}, \{x, c\}, \{\bar{b}, c\}, \{x, \bar{d}\}\}$$

is not really different. We renamed a by \bar{x} and consequently \bar{a} by x . Nobody will be surprised that all those formulas have the same number of clauses, variables, and even the same number of satisfying assignments. What happens if we rename b by x in F_3 , but keep all other names? Let us see:

$$F_4 = \{\{\bar{x}, x\}, \{x, c\}, \{\bar{x}, c\}, \{x, \bar{d}\}\} .$$

You may protest that this is not even a proper CNF formula, since $\{\bar{x}, x\}$ is not even a real clause! But being pragmatic, we will soon see that we benefit from admitting clauses like $\{\bar{x}, x\}$ to the party. So let us agree that in

this section, clauses are allowed to contain complementary literals. Surely, F_4 is different from F_3 . It has, for example, one variable less than F_3 . Now we replace c and d by x as well:

$$F_5 = \{\{\bar{x}, x\}, \{x, x\}, \{\bar{x}, x\}, \{x, \bar{x}\}\} = \{\{\bar{x}, x\}, \{x\}\} .$$

This is indeed a very simple formula. In particular, it is satisfiable. In fact it turns out that a formula F is satisfiable if and only if its variables can be renamed (in the above sense) to obtain the formula $\{\{\bar{x}, x\}, \{x\}\}$. Let us make this precise.

Let V and V' be sets of variables. By \bar{V} and \bar{V}' we denote the set of negated variables. A *renaming* is a map $\Phi : V \rightarrow V' \cup \bar{V}'$. It naturally extends to \bar{V} via $\Phi(\bar{x}) := \overline{\Phi(x)}$. It extends to clauses over the variables V via $\Phi(C) = \{\Phi(u) \mid u \in C\}$ and to whole CNF formulas via $\Phi(F) = \{\Phi(C) \mid C \in F\}$. Let us start to justify why these definitions do not come out of the blue.

Lemma 2.21** *Let F be a CNF formula and $\Phi : \text{var}(F) \rightarrow V' \cup \bar{V}'$ be some renaming. If $\Phi(F)$ is satisfiable, then F is as well.*

Proof . Write $G := \Phi(F)$ and let $\alpha : \text{var}(G) \rightarrow \{0, 1\}$ satisfy G . Then the assignment $\alpha \circ \Phi$ satisfies F . In other words, we define $\beta : \text{var}(F) \rightarrow \{0, 1\}$ via

$$\beta(x) := \alpha(\Phi(x)) .$$

β satisfies every clause $C \in F$: α satisfies some literal in $\Phi(C)$. This literal is of the form $\Phi(u)$ for some $u \in C$. Thus $\beta(u) = \alpha(\Phi(u)) = 1$, and β satisfies C . \square

Corollary 2.22** *A CNF formula F is satisfiable if and only if there is a renaming Φ such that $\Phi(F) \subseteq \{\{\bar{x}, x\}, \{x\}\}$.*

Proof . The “if” direction follows from Lemma 2**.21. For the “only if” direction, assume that F is satisfiable and let α be a satisfying assignment. Define $\Phi : \text{var}(F) \rightarrow \{x, \bar{x}\}$ by

$$\Phi(y) := \begin{cases} x & \text{if } \alpha(y) = 1 , \\ \bar{x} & \text{else.} \end{cases}$$

In every clause of F , at least one literal evaluates to 1 under α . Hence $\Phi(C)$ is either $\{x\}$ or $\{\bar{x}, x\}$. \square

Theorem 223** *A CNF formula F is unsatisfiable if and only if there exists an $MU(1)$ -formula G and a renaming Φ such that $\Phi(G) \subseteq F$. Furthermore, the minimum number of clauses in such a formula G equals the minimum number of leaves of a resolution tree of F ending in \square .*

Proof. For the “if” direction, suppose $G \in MU(1)$, and $\Phi(G) \subseteq F$ for some renaming Φ . By Lemma 2**21, $\Phi(G)$ is unsatisfiable, and so is F . We now show that F has a resolution tree with $|G|$ leaves. By Theorem 2**18, there is a strict resolution tree T_G for G , having $|G|$ leaves. We can extend Φ to operate on resolution trees as well: For a resolution tree T , we let $\Phi(T)$ be the same tree, but we replace every clause label C by $\Phi(C)$. Clearly, if T is a resolution tree of some formula F ending in C , then $\Phi(T)$ is a resolution tree of $\Phi(F)$ ending in $\Phi(C)$.¹ In our case, $\Phi(G) \subseteq F$, and therefore $\Phi(T_G)$ is a resolution tree for F ending in the empty clause, having $|G|$ many leaves.

For the “only if” direction, let F be unsatisfiable, and let T be a resolution tree of F with a minimum number of leaves. We will construct an $MU(1)$ -formula G having as many clauses as T has leaves. We prove a more general statement:

Lemma 224** *Let T be a resolution tree ending in clause C . Then there exists a strict resolution tree T' ending in clause C , and a renaming Φ such that $\Phi(T') = T$ and $\Phi(x) = x$ for any $x \in \text{var}(C)$.*

Proof. We proceed by induction on the size of T . If T consists of the root only, labeled with clause C , it is already strict, and we are done, by letting Φ be the identity on $\text{var}(C)$. Otherwise, suppose the root is labeled with clause C , and its two children with $D_1 \cup \{x\}$ and $D_2 \cup \{\bar{x}\}$, respectively, such that $C = D_1 \cup D_2$. Let T_1, T_2 be the two resolution trees rooted at the children of the root. By induction, there are strict resolution trees T'_1 and T'_2 ending in $D_1 \cup \{x\}$ and $D_2 \cup \{\bar{x}\}$, respectively. By renaming variables, we can make sure that the only variables occurring in both T'_1 and T'_2 are the variables $D_1 \cup \{x\}$ and $D_2 \cup \{\bar{x}\}$. Also by induction, there are renamings Φ_1, Φ_2 with $\Phi_1(T'_1) = T_1$ and $\Phi_2(T'_2) = T_2$. Since they are the identity function

¹Some clauses appearing as labels in T might become trivial under $\Phi(C)$, i.e. possibly $u, \bar{u} \in \Phi(C)$. It is not difficult to get rid of these trivial clauses in T without making the tree larger.

on the variables $\text{var}(D_1) \cup \text{var}(D_2) \cup \{x\}$, there is a “big” renaming Φ doing the job for both. We construct T' by glueing T'_1 and T'_2 to a new root, which we label with C . This is the desired strict tree. \square

Apply the lemma to the minimum size resolution tree T of F ending in \square . We obtain a strict resolution tree T' with the same number of leaves. Collect all clauses appearing as labels of leaves of T' in a formula G . By construction, T' is a strict resolution tree for G , ending in \square . By Theorem 2**.18, this means that $G \in \text{MU}(1)$. \square

Exercise 2.8** Show the limits of Lemma 2**.11: Give an example of an SMU formula F such that $F^{[x \mapsto 0]}$ is MU, but not SMU.

Exercise 2.9** Prove the following “splitting theorem”:

Theorem 2.25** If $F \in \text{MU}(1)$, then either $F = \{\square\}$, or there exists a variable $x \in \text{var}(F)$ and a partition $F = F_0 \uplus F_1$ such that

- $\text{var}(F_0) \cap \text{var}(F_1) = \{x\}$,
- $F_0^{[x \mapsto 0]} \in \text{MU}(1)$ and $F_1^{[x \mapsto 1]} \in \text{MU}(1)$.

Exercise 2.10** Prove the following theorem for SMU(1)-formulas:

Theorem 2.26** Let $F \in \text{SMU}(1)$. Then either $F = \{\square\}$, or there exists a variable $x \in \text{var}(F)$ such that $x \in \text{var}(C)$ for every $C \in F$, and $F^{[x \mapsto 0]}$ and $F^{[x \mapsto 1]}$ are both again in $\text{SMU}(1)$, and

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

Further, for any two distinct clauses $C, D \in F$, it holds that C and D have a 1-conflict, i.e. $|C \cap \overline{D}| = 1$, where $\overline{D} = \{\bar{u} \mid u \in D\}$.

Exercise 2.11** We call a CNF formula monotone if all clauses contain either only positive or only negative literals. A monotone (k, ℓ) -CNF formula consists of positive k -clauses and negative ℓ -clauses.

1. Give an example of a monotone $(2,2)$ -CNF formula in $MU(1)$.
2. Show that any monotone (k,ℓ) -CNF formula in $MU(1)$ has at least $\binom{k+\ell-1}{k} + \binom{k+\ell-1}{\ell}$ clauses. Hint: use Theorem 2**.25
3. Show that this is tight. Give a construction of monotone $(k,-\ell)$ -CNF formulas in $MU(1)$ with exactly $\binom{k+\ell-1}{k} + \binom{k+\ell-1}{\ell}$ clauses.

NOTE 2**.1 Aharoni and Linial [1] were the first to investigate MU formulas using matching arguments. However, they attribute the fact that $\delta(F) \geq 1$ for all $F \in MU$ to an unpublished result of Michael Tarsi. Also, strongly minimal unsatisfiable formulas as well as Theorem 2**.26 are from [1]. Theorem 2**.25 and Theorem 2**.12 were first proven by Davydov, Davydova and Kleine Büning [3]. However, their proof does not use strongly minimal formulas, and is rather complicated. Hoory and Szeider [5] state that a proof of Theorem 2**.25 “is implicitly present in” [1], and by this they probably mean that Theorem 2**.12 follows rather painlessly once proven for $SMU(1)$ -formulas. Theorem 2**.19 was originally proved by Kleine Büning [2]. The connection between “renamings”, resolution trees and $MU(1)$ -formulas we described in Section 2**.5 was first investigated by Szeider [7].

There is much more to be said about $MU(k)$ -formulas. As we have seen, deciding membership in $MU(k)$ is in NP, since we can give short resolution proofs. It turns out that they can actually be found efficiently (Kullmann [6] and Fleischner, Kullmann and Szeider [4]), for any constant k . The first paper works by somehow “guessing” the structure of the resolution proof, while the latter one gives a general algorithm for deciding satisfiability in time $\binom{n}{k} \text{poly}(n)$, where $n = |\text{var}(F)|$ and $k = \delta^*(F)$. However, even for moderate values of $\delta^*(F)$, say $\delta^*(F) = 10$, this is of order n^{10} or larger. One would like to have an algorithm with running time $f(k)\text{poly}(n)$, where the degree of the polynomial in n does not depend on k . Such an algorithm has indeed been given by Szeider [8], and this one again works more by using resolution.

Bibliography

- [1] R. Aharoni and N. Linial. Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. *J. Combin. Theory Ser. A*, 43(2):196–204, 1986.
- [2] H. K. Büning. An upper bound for minimal resolution refutations. In G. Gottlob, E. Grandjean, and K. Seyr, editors, *CSL*, volume 1584 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 1998.
- [3] G. Davydov, I. Davydova, and K. Büning. An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. *Ann. Math. Artificial Intelligence*, 23(3-4):229–245, 1998.
- [4] H. Fleischner, O. Kullmann, and S. Szeider. Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. *Theoret. Comput. Sci.*, 289(1):503–516, 2002.
- [5] S. Hoory and S. Szeider. Computing unsatisfiable k-SAT instances with few occurrences per variable. *Theoretical Computer Science*, 337(1-3):347–359, 2005.
- [6] O. Kullmann. An application of matroid theory to the sat problem. In *COCO '00: Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, page 116, Washington, DC, USA, 2000. IEEE Computer Society.
- [7] S. Szeider. Homomorphisms of conjunctive normal forms. *Discrete Appl. Math.*, 130(2):351–365, 2003.

- [8] S. Szeider. Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. *J. Comput. Syst. Sci.*, 69(4):656–674, 2004.

Chapter 3

Algorithms for 2-SAT

A major part of the course will investigate the *k-SAT problem*¹ for $k \in \mathbf{N}$, 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 difference²—but see Exercise 3.1.

In this section we focus on 2-SAT. For a starter we observe that if two (≤ 2) -clauses have a resolvent, then this resolvent is still a (≤ 2) -clause. So if we keep adding resolvents to a (≤ 2) -CNF formula F over n variables, the resulting formulas are still (≤ 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 $O(n^2)$ resolvents. If we encounter the empty clause, then F is not satisfiable; if not,

¹You 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.

²A 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.

it is. Hence, resolution is a *polynomial* method for (≤ 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 (≤ 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 (≤ 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 $F^{[u \rightarrow 1]}$. 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 $|\text{sat}_V(F)| = |\text{sat}_{V \setminus \text{vbl}(\{u\})}(F^{[u \rightarrow 1]})|$. In particular, F is satisfiable iff $F^{[u \rightarrow 1]}$ is satisfiable.*

Clearly, we can iterate this step, called *unit clause rule* until no 1-clause remains, as done in procedure $\text{uc}()$. 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 CNF formula over V . POSTCONDITION: returns $U \subseteq V$ and $G = F^{[\alpha]}$, for some $\alpha \in \{0, 1\}^{V \setminus U}$, without 1-clauses s.t. $ \text{sat}_V(F) = \text{sat}_U(G) $.	function $\text{uc}(F, V)$ $G \leftarrow F; U \leftarrow V;$ while \exists 1-clause in G do $\{u\} \leftarrow \text{some 1-clause in } G;$ $G \leftarrow G^{[u \rightarrow 1]}$; $U \leftarrow U \setminus \text{vbl}(\{u\});$ return $(G, U);$
--	--

What we have described so far is true for all CNF formulas, so let us now specialize to a (≤ 2) -CNF formula F . If we apply $\text{uc}()$ to F , then we get back G which either contains \square —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 $\text{uc}(F^{[x \rightarrow 1]}, V \setminus \{x\})$.*

- (i) *In case $\square \notin G$: F is satisfiable iff G is satisfiable.*
- (ii) *In case $\square \in G$: F is satisfiable iff $F^{[x \rightarrow 0]}$ is satisfiable.*

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

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

For (i), let us first observe that $uc()$ implicitly builds up an assignment α on $(V \setminus \{x\}) \setminus U$ so that $G = F^{[x \mapsto 1]}[\alpha]$. The key observation is that if $\square \notin G$, 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 α satisfies $F^{[x \mapsto 1]} \setminus G$. So to show “ \Leftarrow ”, let β be an assignment on $vbl(G) \subseteq U$ (this is disjoint from $V \setminus U$) that satisfies G . Then $x \mapsto 1$, α and β can be combined to an assignment that satisfies F . \square

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 $\square \notin G$, and thus it may indeed happen that $G \not\subseteq F$ is not satisfiable, while F is.

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

We are ready to describe $uc_2s()$, which exploits the claims of Lemma 3.2 for an efficient satisfiability test for (≤ 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 set) decreases by one element at least. Hence, the procedure is polynomial for (≤ 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

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

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

Exercise 3.5 Given a CNF formula F over V , procedure $uc(F, V)$ returns in polynomial time a pair (G, U) , such that G is a CNF formula over U without 1-clauses, $|U| \leq |V|$, and $|\text{sat}_U(G)| = |\text{sat}_V(F)|$. 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 $uc(F^{[x \mapsto 1]}, V \setminus \{x\})$ 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 (≤ 2) -CNF formula?

3.2 A Randomized Algorithm

Suppose we are confronted with a CNF formula which we are promised⁴ to be satisfiable. In order to find such an assignment, [Papadimitriou, 1991] 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 $rf()$.

Note here the fine distinction between the “ \leftarrow_{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

⁴Even 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 2.7.

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

a satisfying assignment, and it will be the goal of this section to show that for a satisfiable (≤ 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 $\text{rf}(F, V)$ for a satisfiable (≤ 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 α and α^* .

- $d_H(\alpha, \alpha^*) \leq n := |V|$, 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 α has to disagree with α^* in at least one literal v in the chosen clause C —after all, α does not satisfy C while α^* does. Because C is a (≤ 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 α^*).

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 Z* “moves” in state space 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 $W_t \in \mathbf{Z}$ denote the state of the process at time $t \in \mathbf{N}_0$, then $(W_t)_{t \in \mathbf{N}_0}$ is a homogeneous Markov chain (see Appendix A.3) with $W_0 \in \mathbf{Z}$ and

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

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

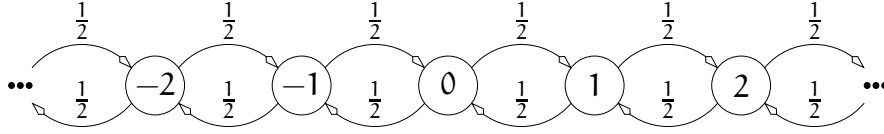


Figure 3.1: Symmetric random walk.

Lemma 3.3 $(W_t)_{t \in \mathbf{N}_0}$ as above, $a, b, s \in \mathbf{Z}$, $a \leq s \leq b$.

$E(\min\{t \mid W_t \in \{a, b\}\} | W_0 = s)$, 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 , $s \in \mathbf{Z}$. It observes

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

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

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

For $n \in \mathbf{N}_0$, a *symmetric random walk reflecting at n* is a homogeneous Markov chain $(R_t)_{t \in \mathbf{N}_0}$ with state space $\mathbf{Z}_{\leq n} := \{i \in \mathbf{Z} \mid i \leq n\}$ 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 $i < n$, $t \in \mathbf{N}$, and $\Pr(R_t = n - 1 | R_{t-1} = n) = 1$ for all $t \in \mathbf{N}$.

⁵If 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 $s \in \mathbf{Z}$. Therefore $e_{a-1} = -(b - a + 1)$, or what?

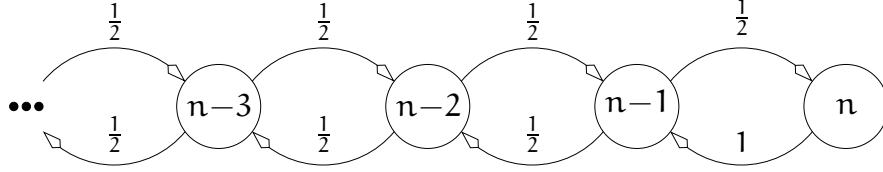


Figure 3.2: Symmetric random walk reflecting at n .

Lemma 3.4 $n, (R_t)_{t \in \mathbb{N}_0}$ as above, $s \in \mathbb{N}_0, s \leq n$.

$E(\min\{t \mid R_t = 0\} \mid R_0 = s)$, 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 $\text{rf}()$ 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?

3.2.2 Coupling

Theorem 3.5 If F is a satisfiable (≤ 2) -CNF formula over V , $n := |V|$, then $\text{rf}()$ 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^{th} iteration of the while-loop of procedure

$\text{rf}()$. We first fix some assignment α^* 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 α^* and the current assignment α 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 $\{-1, +1\}$.

Let T be (the random variable for) the number of iterations of $\text{rf}()$. Let D_0 denote the Hamming distance between the fixed satisfying assignment α^* and the initial choice of α , and for $1 \leq t \leq T$ let D_t denote the Hamming distance between α^* and α after the t^{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 if⁶ $\hat{R}_0 = D_0$, 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 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 $E(T) \leq \max_{s=0}^n s(2n - s) = n^2$. \square

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 (≤ 2) -CNF formula? The answer is yes, if we are willing to accept some *error probability*.

We let F be (≤ 2) -CNF formula. We drop the precondition in $\text{rf}()$ that F is satisfiable. Let T be the random variable for the number of iterations of the while-loop in $\text{rf}()$ —it is ∞ , if F is not satisfiable. If F is satisfiable, we know that $E(T) \leq n^2$, and thus, by Markov's Inequality $\Pr(T \geq 3n^2) \leq \frac{1}{3}$ (see Appendix A.4). In

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

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: λn^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 $3rn^2$ iterations of the while-loop altogether. So for the expense of λn^2 iterations we get an error probability of at most $(\frac{1}{3})^{\lceil \lambda/3 \rceil}$. 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 α^* 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, α^* 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 α^* 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 α u.a.r. in $\{0, 1\}^V$? Then, for a fixed satisfying assignment α^* ,

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

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 $E(S(2n-S))$ where $S \in \{0..n\}$ is a random variable with $\Pr(S=s) = \binom{n}{s} 2^{-n}$. Now

$$E(S(2n-S)) = 2n E(S) - E(S^2) \leq 2n E(S) - E(S)^2 = 2n \frac{n}{2} - \left(\frac{n}{2}\right)^2 = \frac{3}{4} n^2$$

Here we have used $E(S) = \frac{n}{2}$ which follows from its distribution, and $E(S^2) \geq E(S)^2$, 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 \mathbf{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.

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

Exercise 3.10 For $t \in \mathbf{N}_0$, determine $E((W_t)^2 | W_0 = 0)$ for a symmetric random walk $(W_t)_{t \in \mathbf{N}_0}$ on \mathbf{Z} .

HINT: For a sequence $(S_i)_{i \in \mathbf{N}}$ of mutually independent random variables with $\Pr(S_i = -1) = \Pr(S_i = +1) = \frac{1}{2}$ for all $i \in \mathbf{N}$, show that $(\sum_{i=1}^t S_i)_{t \in \mathbf{N}_0}$ is a symmetric random walk on \mathbf{Z} .

Exercise 3.11 *Analyze the expected number of flips procedure $\text{rf}()$ takes for*

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

until a satisfying assignment is reached, (i) when started with an assignment with ℓ 1's, $0 \leq \ell \leq n$, 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 $n \in \mathbf{N}$. That is, you are required to describe for every $n \in \mathbf{N}$ a satisfiable (≤ 2)-CNF formula F_n over a set V_n of n variables, an initial assignment α_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 $\text{rf}(F_n, V_n)$ is exactly n^2 .*

Exercise 3.13

Optimal Patience

In Note 5.1, 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 δ , such that restarting a new incarnation of the algorithm after δb 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 $n \in \mathbf{N}_0$, a symmetric random walk dwelling on n is a homogeneous Markov chain $(D_t)_{t \in \mathbf{N}_0}$ with state space $\{i \in \mathbf{Z} \mid i \leq n\}$ and for all $t \in \mathbf{N}$

$$\begin{aligned} \Pr(D_t = i + 1 \mid D_{t-1} = i) &= \Pr(D_t = i - 1 \mid D_{t-1} = i) = \frac{1}{2} \quad \text{if } i < n, \text{ and} \\ \Pr(D_t = n - 1 \mid D_{t-1} = n) &= \Pr(D_t = n \mid D_{t-1} = n) = \frac{1}{2}. \end{aligned}$$

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

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

3.3 A Linear Time Algorithm

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

Recall that

$$(u \vee v) \equiv \bar{u} \rightarrow v \equiv \bar{v} \rightarrow u ,$$

and we also have

$$(u) \equiv \bar{u} \rightarrow u .$$

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

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

Note that each clause $\{u, v\}$ with $u \neq v$ gives rise to two edges, while a unit clause $\{u\}$ leads to one edge (\bar{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 $u \rightarrow v$ if $(u, v) \in E(D)$, and we write $u \rightsquigarrow v$ if there is a directed path

$$u = w_0 \rightarrow w_1 \rightarrow \dots \rightarrow w_\ell = v, \quad \ell \in \mathbf{N}_0,$$

from u to v in D .

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

- (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 $u \rightsquigarrow \bar{u}$, then every satisfying assignment sets u to 0.

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

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

Proof. Since $x \rightsquigarrow \bar{x}$ forces x to be set to 0 in a satisfying assignment, and $\bar{x} \rightsquigarrow x$ forces \bar{x} to be 0, clearly $x \rightsquigarrow \bar{x} \rightsquigarrow x$ 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 $V \cup \bar{V}$. The equivalence classes $[u]$, $u \in V \cup \bar{V}$, 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 \bar{x} for an $x \in V$. Hence, $[\bar{u}] := \{\bar{v} \mid v \in [u]\}$ is disjoint from $[u]$. Note that $[\bar{u}] = [\bar{u}]$.

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 $[\bar{u}]$ is maximal, since

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

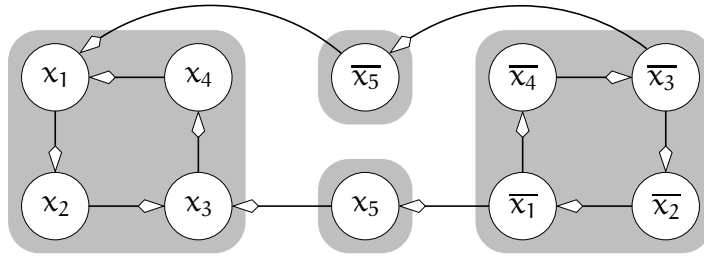


Figure 3.3: The directed graph D for the satisfiable 2-CNF formula $\{\{\bar{x}_1, x_2\}, \{\bar{x}_2, x_3\}, \{\bar{x}_3, x_4\}, \{\bar{x}_4, x_1\}, \{\bar{x}_5, x_3\}, \{x_5, x_1\}\}$ with its strongly connected components. Note, e.g., $[\bar{x}_2] \leq [x_5]$ and $[\bar{x}_1] \leq [\bar{x}_5]$.

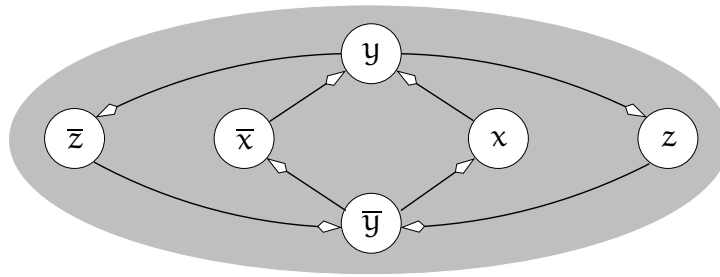


Figure 3.4: The directed graph D for the unsatisfiable 2-CNF formula $\{\{x, y\}, \{\bar{x}, y\}, \{z, \bar{y}\}, \{\bar{z}, \bar{y}\}\}$. The whole graph forms one strongly connected component.

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

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 \bar{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 (≤ 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* (≤ 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 $\{\bar{x}, v\}$. This clause introduces the edges $\bar{u} \rightarrow v$ and

$\bar{v} \rightarrow u$ to the graph D . But since $\bar{v} \rightarrow \bar{x} \rightarrow u$ and $\bar{u} \rightarrow x \rightarrow v$ 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 $u \rightarrow v$ iff $u \leadsto v$ and $u \neq v$ (verify!). This clearly proves now the claim, since F is unsatisfiable iff $F' \supseteq \{\{x\}, \{\bar{x}\}\}$ for some $x \in V$, which is equivalent to $x \rightarrow \bar{x} \rightarrow x$ in the digraph of F' , which—in turn—is equivalent to $x \leadsto \bar{x} \leadsto x$ in the graph of F . \square

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 $b \in \mathbf{N}$ 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 (≤ 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 $\alpha \in \{0, 1\}^{\text{var}(G)}$, $F^{[\alpha]}$ is a (≤ 2) -CNF formula, and
- the number of assignments in $\{0, 1\}^{\text{var}(G)}$ satisfying G is $7^{|G|} \leq 7^{n/3}$.

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

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

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 (≤ 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 (≤ 4) -CNF formula over n variables.

3.4.2 Coloring Graphs

For $k \in \mathbf{N}$, a *proper k-coloring* of a graph $G = (V, E)$ is a mapping $\chi : V \rightarrow \{1..k\}$ 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 $\chi : V \rightarrow \{1, 2, 3\}$ and tests each one, whether it is proper. This gives a complexity of⁷ $O(3^n \text{poly}(n))$. 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 γ 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 γ* . 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 $O(|V| + |E|)$.

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

We employ this procedure for 3-coloring. Suppose G does indeed have a proper 3-coloring χ , and we choose for each vertex v a $\gamma(v)$ u.a.r. in $\binom{\{1,2,3\}}{2}$. 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 γ , 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

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

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

(2) *For every positive real number λ (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*

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

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

This algorithm is of the same type as the one from Note 5.1—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 $O(c^n \text{poly}(n))$ 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, $|V| = n \in \mathbf{N}$. Now consider a set Γ of mappings $V \rightarrow \binom{\{1,2,3\}}{2}$. We call Γ exhaustive, if for every mapping $\chi: V \rightarrow \{1,2,3\}$ 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 $\lceil (\ln 3) \cdot n \cdot \left(\frac{3}{2}\right)^n \rceil$.

HINT: Take Γ as a random set obtained by drawing $\lceil (\ln 3) \cdot n \cdot \left(\frac{3}{2}\right)^n \rceil$ 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 Γ .

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}

to

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

This¹ 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

¹The 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!

²Talking 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 $\text{PropCol}(k, G)$ such that G has a proper k -coloring iff $\text{PropCol}(k, G)$ is satisfiable.*

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

$$\text{OnelsOne}_\ell(\mathbf{x}) := \left(\bigvee_{i=1}^\ell x_i \right) \wedge \bigwedge_{1 \leq i < j \leq \ell} \neg(x_i \wedge x_j). \quad (4.1)$$

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 $k \in \mathbb{N}_0$, $k \leq |V|$, let

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

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

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

If χ 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 α 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 *OnelsOne*-subexpressions. Assign this color i to v , and the resulting coloring is proper, as assured by the remaining subexpressions.

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

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 $\text{PropCol}(k, G)$ equals⁵ 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 α for $\text{PropCol}(k, G)$, i.e., assuming a vertex set $V := \{1..n\}$, 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

⁴ $\text{PropCol}(2, G)$ can be written as an equivalent (≤ 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.

⁵You may observe that we could have replaced ' $\text{OnelsOne}_k((x_{v,i})_{i=1}^k)$ ' in $\text{PropCol}(k, G)$ 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.

we prefer to remain with a constant number of bits, we can go back to Theorem 1.1, which lets us transfer $f := \text{PropCol}(k, G)$ to a (≤ 3) -CNF formula f' such that f' is satisfiable iff f is satisfiable. In fact, in our case it suffices to replace the $\text{OnelsOne}_k((x_{v,i})_{i=1}^k)$ -subexpressions by

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

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.

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

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 ℓ 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 $3\text{ColGraph}(F)$ such that F is satisfiable iff $3\text{ColGraph}(F)$ has a proper 3-coloring.*

Proof. Let $V := \text{vbl}(F)$. For $3\text{ColGraph}(F)$ we choose a graph $G = (V_G, E_G)$ with vertex set $V_G := \{t, c\} \cup V \cup \bar{V} \cup \bigcup_{C \in F} \{a_{C,1}, a_{C,2}, a_{C,3}, b_{C,1}, b_{C,2}, b_{C,3}\}$. 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

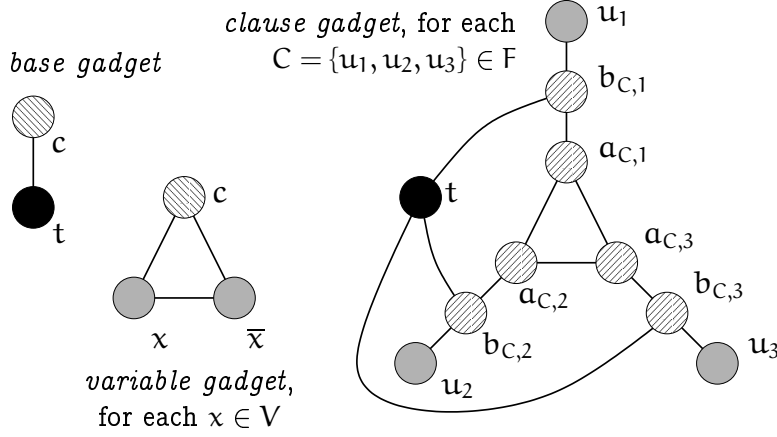


Figure 4.1: The edges in $3\text{ColGraph}(F)$ for 3-CNF formula F .

$|E_G| = 1 + 3|V| + 12|F|$. The size of G is polynomial in $|V| + |F|$, and it can be generated from F in polynomial time.

First we show that a proper 3-coloring χ 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 $\chi(u) \neq \chi(\bar{u})$ for all literals u because of the variable gadgets. So that's a good start: The restriction α_χ of χ to $V \cup \bar{V}$ 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, α_χ satisfies C .

Secondly and finally, we show that a satisfying assignment α 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 α , 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 . \square

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 $3\text{ColGraph}(F')$ 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 problem⁶

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, \dots, 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.

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

Check whether your solution guarantees that the number of satisfying assignments of $\text{HamPath}(G)$ gives the number of Hamiltonian paths in G . (For $n \geq 2$, v_1, v_2, \dots, v_n and v_n, v_{n-1}, \dots, 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 for the case that k is a fixed constant.

⁶William Roger Hamilton, 19th century.

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* Σ is a finite set of symbols. Σ^* denotes the set of all finite strings of symbols in Σ (including the *empty string* ε). Elements w in Σ^* 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 Σ^* for some alphabet Σ .

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

Now the problem HAMILTONIAN PATH can be represented by the language

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

PROPER COLORING by

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

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 L over $\{0, 1\}$ such that there is a polynomial time algorithm A_L with

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

‘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 reveal⁷ 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}

\mathcal{NP} is⁸ the class of all languages with a polynomial verifier. Clearly, $\mathcal{P} \subseteq \mathcal{NP}$. It is easy to see that also⁹ SAT and PROPER COLORING are in \mathcal{NP} , while we do not know of a polynomial time algorithm for these problems.¹⁰

⁷At 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.

⁹Note 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.

¹⁰It 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 $\text{co-}\mathcal{NP} := \{\{0, 1\}^* \setminus L \mid L \in \mathcal{NP}\}$ is considered.

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

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

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

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

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 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 $\hat{p}_F(|w|)$).

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

\mathcal{FP}_k

We observe that

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

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

$$\begin{aligned} V \text{ accepts } \langle w, c \rangle &\text{ iff} \\ F \text{ rejects } \langle w, c_S, S \rangle &\text{ for no } S \subseteq \{1..\hat{p}_F(|w|)\} \text{ with } |S| \leq k. \end{aligned}$$

Since there are only $O(\hat{p}_F(|w|)^k)$ sets S to consider, V 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 $m \leq \hat{p}_F(|w|)$ such that F accepts $\langle w, \epsilon, \{i\} \rangle$ for all $i > m$ and F accepts $\langle w, 0, \{i\} \rangle$ or $\langle w, 1, \{i\} \rangle$ for all $i \leq m$. $\mathcal{FP}_1 = \mathcal{P}$ follows.

Theorem 4.3

- (1) $\mathcal{FP}_k = \mathcal{P}$ for all $k \leq 2$.
- (2) $\mathcal{FP}_k = \mathcal{FP}_3$ for all $k \geq 3$.
- (3) $\mathcal{FP}_3 = \mathcal{P}$ iff $\text{SAT} \in \mathcal{P}$.

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 $m \leq \hat{p}_F(|w|)$, we can produce in polynomial time—assuming the help of F —a $(\leq k)$ -CNF formula $G_m(w)$ which is satisfiable iff there is an F -certificate of length m . Then an F -certificate exists iff

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

is satisfiable. That is, $w \in L$ iff $\text{Formula}(w)$ 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 $\text{Formula}(w)$ in polynomial time by checking whether one of the (≤ 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 $\text{Formula}(w)$ as above, and then it constructs in polynomial time a (≤ 3) -CNF formula G' over some variable set $U := \{x_i\}_{i=1}^\ell$ such that G' is satisfiable iff $\text{Formula}(w)$ is satisfiable. Now the F' -certificates for w are the strings $(\alpha(x_i))_{i=1}^\ell$ for assignments α satisfying G' . Since G' is a (≤ 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 $\text{Formula}(w)$, 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 $\{i_j\}_{j=1}^b \subseteq \mathbf{N}$ with

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

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

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

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

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

Now it is easy to see that $c = (c_i)_{i=1}^m \in \{0, 1\}^m$ is an 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. \square

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

 \leq_P

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 $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$ then $L_1 \leq_P L_3$
- (2) If $L_1 \leq_P L_2$ 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 4.3 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 $\bigcup_{k \in \mathbb{N}_0} \mathcal{FP}_k$, 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 \bar{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 $\mathcal{FP}_k^{(l)}$ 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 l -symbol alphabet.

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

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 $L \leq_P \text{SAT}$ iff $L \in \mathcal{FP}_3$.

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

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

Exercise 4.9 Let $\mathcal{FP}_{2|1}$ be defined the same as \mathcal{FP}_3 except that the position sets S are further restricted to $\{i, i+1, j\}$ for $i, j \in \mathbb{N}$. Show that $3\text{-SAT} \in \mathcal{FP}_{2|1}$.

Exercise 4.10

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 $L \leq_P \text{SAT}$ 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 Turing¹¹ 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 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.

¹¹Alan Turing, 1912-1954

¹²Let 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.

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 , Σ and Γ are all finite sets and Q is the set of *states*, Σ is the *input alphabet*, $\Gamma \supseteq \Sigma$ is the *tape alphabet* containing the special *blank symbol* $\sqcup \notin \Sigma$,

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

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_{yes} . Similarly, it rejects w if it halts in state q_{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}, ℓ, h) , $q \in Q$, $\ddot{w} := \sqcup w \sqcup$ for $w \in (\Gamma \setminus \{\sqcup\})^*$, $\ell \in \mathbb{Z}$, and $h \in \{\ell, \ell + 1, \dots, \ell + |\ddot{w}| - 1\}$ (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}, ℓ, 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 and¹³ $(q', b, \sigma) := \delta(q, a)$, we define

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

¹³Note 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_{yes} (in state q_{no} , resp.). The set of all words in Σ^* 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 Σ^* .

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_{yes} being the state of C_t , and $C_0 \vdash_M C_1 \vdash_M \dots \vdash_M C_t$. 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

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

of length $4m + 2$ over $Q \cup \Gamma \cup \{\triangleleft, \triangleright\}$. (As usual, we write α^i for the sequence of α 's of length i .) Obviously, the m -representation fully encodes a configuration, and it is best visualized by writing the second half of the

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

string aligned below the first half, see Figure 4.2. Then the state q ends up exactly above the symbol in \tilde{w} as prescribed by the position of the head defined by the configuration C . Note also that given m , we can easily

\triangleright	\triangleright	q	\triangleleft	\triangleleft	\triangleleft	\triangleleft
\sqcup	1	0	1	0	\sqcup	\sqcup
\triangleright	q'	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft
\sqcup	1	1	1	0	\sqcup	\sqcup
q	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft
\sqcup	1	1	1	0	\sqcup	\sqcup

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 $Q \cup \{\triangleleft, \triangleright\}$ for some $i \in \{1..2m+1\}$,
- position 1 contains \triangleleft , or position $2m+1$ contains \triangleright ,
- there is $\triangleright\triangleleft$ or $\triangleleft\triangleright$ or $q\triangleright$ or $\triangleleft q$ or qq' for $q, q' \in Q$ in consecutive positions in the range $\{1..2m+1\}$,
- a symbol in position i is not in Γ 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 $a \dots \sqcup \dots b$ 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_{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

\triangleleft	\triangleright	\cdot q	q \cdot
a	a	\cdot a	a \cdot
\cdot	\cdot	q' \cdot	\cdot q'
a	a	\cdot b	b \cdot

Figure 4.3: The top two symbols force the other symbols, unconditionally 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 $L_1 \leq_P L_2$ 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 $\{0, 1\}^* \setminus L \leq_P L$, 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 Goethe¹, 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.

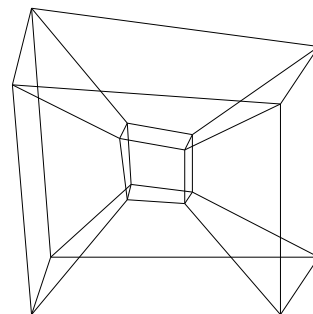


Figure 5.1: Hand-crafted 4-cube.

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

¹Johann Wolfgang von Goethe, 1749–1832

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 are² 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 $A \rightarrow \{0, 1\}$) and with an edge between α and β 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 above³. 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 .

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

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

(So these are all mappings that agree with φ on the elements that are mapped to 0 or 1 by φ .) The subgraph of the *A-cube* induced by V_φ is

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

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

called the φ -induced face of the A -cube. This face is isomorphic to a cube: The mapping

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

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

The 0-dimensional faces of the A -cube are its vertices, the 1-dimensional faces are the edges⁴, 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 φ' and φ'' , 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.

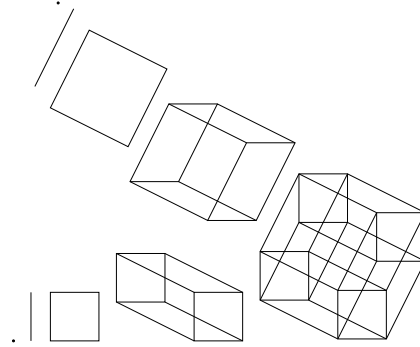


Figure 5.2: Building the 4-cube, in two of many ways (how many?).

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

Lemma 5.1 $n \in \mathbf{N}$. *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 φ' and φ'' , isomorphic copies of the

⁴Strictly 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 $\{u, v\}$).

$(n - 1)$ -cube. If all removed edges lie in just one copy, φ' say, the other copy φ'' is still connected and thus the whole graph is still connected since every vertex in φ' is connected to a vertex in φ'' 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. \square

Faces in Cubes and Clauses in CNF Formulas. Let F be a CNF formula over V . If $C = \{u_1, u_2, \dots, u_k\}$ 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 $\overline{\text{sat}}_V(C)$) forms the vertex set of an $(n - k)$ -dimensional face of the V -cube. It is induced by

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

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

k -CODIM CUBE NONCOVER

Input: $n \in \mathbf{N}$, 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 $(n - 2)$ -faces⁵ 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 $f = x_1 \oplus x_2 \oplus \dots \oplus x_n$ (and recall Exercise 1.9). A total assignment on $\{x_1, x_2, \dots, x_n\}$ 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

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

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 nonempty⁶ 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 2.5 in the cube setting.

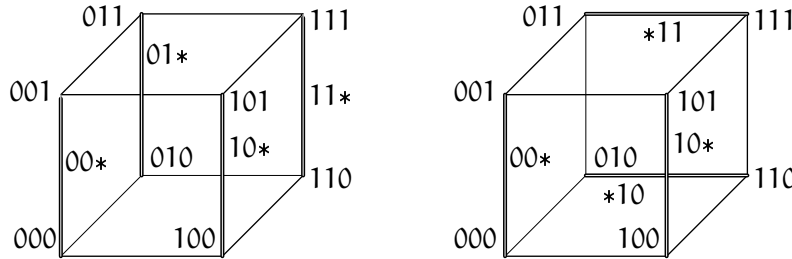


Figure 5.3: Two ways of covering the vertices of a 3-cube with four edges. Or two unsatisfiable 2-CNF formulas with four clauses (answering Exercise 2.5): $\{\{x, y\}, \{\bar{x}, y\}, \{x, \bar{y}\}, \{\bar{x}, \bar{y}\}\}$ corresponds to the left picture, $\{\{x, y\}, \{\bar{x}, y\}, \{\bar{y}, z\}, \{\bar{y}, \bar{z}\}\}$ to the right one.

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.

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

Observation 5.2 $n, m \in \mathbb{N}$. Let $\{\varphi_1, \varphi_2, \dots, \varphi_m\}$ be a set of faces of the n -cube, with φ_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. \square

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. φ_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 φ_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, φ_s maps to $*$ for every i for which s and u differ, and therefore the face induced by φ_s contains u . \square

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

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

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 2.2.

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 2.4 about the ratio of satisfiable clauses in a 2-satisfiable CNF sequence-formula in the cube setting.

Exercise 5.7 $n \in \mathbf{N}, n \geq 3$. 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

$n \in \mathbf{N}$. 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*, is⁷

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

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

It is worthwhile to try to understand this quantity $\text{vol}(n, r)$, and that’s what we will do for the rest of this chapter. For a couple of trivial observations, (i) $\text{vol}(n, r) = \text{vol}(n, \lfloor r \rfloor)$, (ii) $\text{vol}(n, r) = 0$ iff $r < 0$, (iii) $\text{vol}(n, 0) = 1$, (iv) $\text{vol}(n, r) = \text{vol}(n, n) = 2^n$ for $r \geq n$, (v) $\text{vol}(n, r) +$

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

$\text{vol}(n, n - r - 1) = 2^n$ for $r \in \mathbf{Z}$, and (vi) $\text{vol}(n, \frac{n}{2}) = 2^{n-1}$ if n is odd and $\text{vol}(n, \frac{n}{2}) = 2^{n-1} + \frac{1}{2} \binom{n}{n/2}$ 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 $\text{vol}(n, r)$ follows. That is, for r constant, the volume of balls is polynomial in the dimension. We refine.

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

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

Proof. The lower bound is left as Exercise 5.12. For the upper bound, let $\lambda \in \mathbf{R}^+$, $\lambda \leq 1$. We get (with $1 + \lambda < e^\lambda$ for $\lambda \in \mathbf{R} \setminus \{0\}$ and the binomial theorem)—driven by a magic hand—

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

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

For $r > 0.33 \cdot n$, the upper bound exceeds⁸ 2^n and is of no help at all.

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

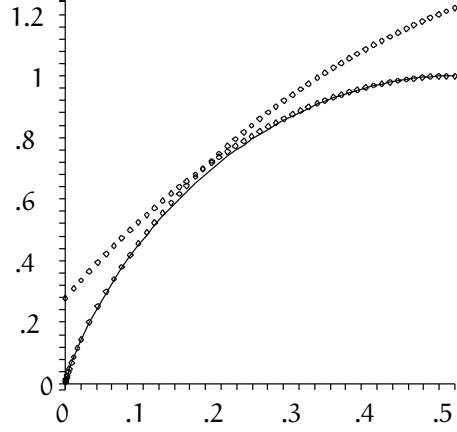
⁸ $\left(\frac{e}{0.33}\right)^{0.33} = 2.00543\dots$

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

for $x \in \mathbf{R}$, $0 < x < 1$, 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.

Lemma 5.6 $n \in \mathbf{N}$, $\rho \in \mathbf{R}^+$, $\rho \leq \frac{1}{2}$.

$$\frac{2^{n \cdot H(\rho)}}{\sqrt{8n\rho(1-\rho)}} \leq \text{vol}(n, \rho n) \leq 2^{n \cdot H(\rho)}.$$



Proof. The argument for the lower bound is omitted, cf. [MacWilliams, Sloane, 1977, Chapter 10, Corollary 9]. For the upper bound we derive with reminiscence of the previous proof

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.

$$\begin{aligned} 1 &= (\rho + (1 - \rho))^n \\ &= \sum_{i=0}^n \binom{n}{i} \rho^i (1 - \rho)^{n-i} \quad (\text{binomial theorem}) \\ &= \sum_{i=0}^n \binom{n}{i} (1 - \rho)^n \left(\frac{\rho}{1 - \rho} \right)^i \\ &\geq \sum_{i=0}^{\rho n} \binom{n}{i} (1 - \rho)^n \left(\frac{\rho}{1 - \rho} \right)^i \quad (\text{truncation of sum}) \\ &\geq \sum_{i=0}^{\rho n} \binom{n}{i} (1 - \rho)^n \left(\frac{\rho}{1 - \rho} \right)^{\rho n} \quad \left(\frac{\rho}{1 - \rho} \leq 1 \text{ and } i \leq \rho n \text{ always} \right) \\ &= \underbrace{\sum_{i=0}^{\rho n} \binom{n}{i}}_{\text{vol}(n, \rho n)} \cdot \underbrace{\rho^{\rho n} (1 - \rho)^{(1-\rho)n}}_{2^{-n \cdot H(\rho)}} \end{aligned}$$

which implies the desired inequality. \square

For ρ 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 $\text{vol}(n, \frac{n}{4})$ is roughly $(4 \cdot 3^{-3/4})^n = 1.7548 \dots^n$, 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 $0.8774 \dots$).

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 $\text{vol}(n, r)$ 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 $E(X) = 0$.

Lemma 5.7 $n \in \mathbf{N}$, $\lambda \in \mathbf{R}$. With X the random variable defined above

$$\Pr(X \geq \lambda) = \frac{\text{vol}(n, \frac{n-\lambda}{2})}{2^n}.$$

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 λ 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 $\text{vol}(n, \frac{n-\lambda}{2})$. The claimed relation is established. \square

We are ready for the next bound.

Lemma 5.8 (Chernoff Bound) $n \in \mathbf{N}$, $\delta \in \mathbf{R}^+$.

$$\text{vol}(n, \frac{n}{2} - \delta) < e^{-2\delta^2/n} \cdot 2^n.$$

Proof. For X as above we show⁹ $\Pr(X \geq \lambda) < e^{-\lambda^2/(2n)}$ for any $\lambda \in \mathbf{R}^+$ which will give the bound in the lemma via Lemma 5.7 (by setting $\lambda := 2\delta$).

⁹With an argument almost verbatim from [Spencer, 1987].

For $t \in \mathbf{R}^+$ and $i \in \{1..n\}$, $E(e^{tX_i}) = \frac{1}{2}(e^t + e^{-t}) < e^{t^2/2}$. In order to justify the inequality, we investigate the Taylor series of the terms involved.¹⁰

$$\frac{1}{2}(e^t + e^{-t}) = \frac{1}{2} \sum_{i=0}^{\infty} \left(\frac{t^i}{i!} + \frac{(-t)^i}{i!} \right) = \frac{1}{2} \sum_{i=0}^{\infty} 2 \frac{t^{2i}}{(2i)!} < \sum_{i=0}^{\infty} \frac{t^{2i}}{2^i i!} = e^{t^2/2}.$$

Mutual independence of the X_i 's allows the following estimate.

$$E(e^{tX}) = E\left(\prod_{i=1}^n e^{tX_i}\right) = \prod_{i=1}^n E(e^{tX_i}) < \prod_{i=1}^n e^{t^2/2} = e^{t^2 n/2}.$$

It is time for Markov's Inequality.

$$\Pr(X \geq \lambda) = \Pr(e^{tX} \geq e^{t\lambda}) \leq \frac{E(e^{tX})}{e^{t\lambda}} < e^{t^2 n/2 - t\lambda},$$

for all $t \in \mathbf{R}^+$. 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. \square

We now can conclude

$$\text{vol}(n, \frac{n}{2} - v\sqrt{n}) < e^{-2v^2} 2^n$$

and

$$\text{vol}(n, \frac{n}{2} - v\sqrt{n \ln n}) < n^{-2v^2} 2^n.$$

For a radius of $\frac{n}{4}$ the bound is

$$\text{vol}(n, \frac{n}{4}) = \text{vol}(n, \frac{n}{2} - \frac{n}{4}) < (e^{-1/8} 2)^n = 1.7649\dots^n$$

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 $\text{vol}(n, r) \leq \frac{(1+\lambda)^n}{\lambda^r} =: f(\lambda)$ for all $\lambda \in \mathbf{R}$, $0 < \lambda \leq 1$. With $\lambda = \frac{r}{n}$ the bound in Lemma 5.5 evolves, but a more ambitious attack will search for the λ 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 $\lambda = \frac{r}{n-r} = \frac{\rho}{1-\rho}$

¹⁰ $(e^t + e^{-t})/2$, also denoted as $\cosh(t)$, *hyperbolic cosine*.

(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 $\text{vol}(n, \rho n) \leq (2^{H(\rho)})^n$, the second one we can rewrite as

$$\text{vol}(n, \rho n) = \text{vol}\left(n, \frac{n}{2} - \left(\frac{1}{2} - \rho\right)n\right) < \left(2^{1-2\left(\frac{1}{2}-\rho\right)^2 \ln 2}\right)^n.$$

(and, for the sake of completeness, the bound in Lemma 5.5 as $(2^{\rho \log_2(e/\rho)})^n$). Therefore the relation of the functions $H(\rho)$ and $1 - 2\left(\frac{1}{2} - \rho\right)^2 \ln 2$ 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 \mathcal{C} of length n is a subset of $\{0, 1\}^n$. The *covering radius* of \mathcal{C} is the smallest radius r such that the balls of radius r centered at the elements of \mathcal{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 \mathcal{C} , or simply

$$r := \max_{u \in \{0,1\}^n} \min_{v \in \mathcal{C}} d_H(u, v).$$

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 \mathcal{C} , we see that $|\mathcal{C}| \cdot \text{vol}(n, r)$ has to be at least 2^n , and so

$$|\mathcal{C}| \geq \frac{2^n}{\text{vol}(n, r)} \geq 2^{(1-H(\rho))n},$$

with ρ 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 $n \in \mathbf{N}, r \in \mathbf{N}_0$. *There exists a code of length n of covering radius at most r with at most*

$$\left\lceil \frac{n \cdot 2^n}{\text{vol}(n, r)} \right\rceil = O\left(2^{(1-H(\rho))n} \text{poly}(n)\right)$$

elements.

Proof. We choose the elements of the code u.a.r. from $\{0, 1\}^n$ with replacement, $\left\lceil \frac{n \cdot 2^n}{\text{vol}(n, r)} \right\rceil$ 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

$$\left(1 - \frac{\text{vol}(n, r)}{2^n}\right)^{\lceil n \cdot 2^n / \text{vol}(n, r) \rceil} \leq e^{-n}.$$

Consequently, the probability that there is an element in $\{0, 1\}^n$ not covered is at most $2^n e^{-n}$, and thus the probability that all elements are covered is at least $1 - \left(\frac{2}{e}\right)^n > 0$. In fact, the probability quickly tends to 1 as n grows. \square

Exercise 5.12

Lower Bound for Binomial Coefficient

For $k, n \in \mathbf{N}$, $k \leq n$, 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

$k, n \in \mathbf{N}$, $k \leq \frac{n}{2}$. 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 $(W_t)_{t \in \mathbf{N}_0}$ on \mathbf{Z} from Chapter 3. For $n, k \in \mathbf{N}$, show that

$$\Pr(|W_n| \geq k | W_0 = 0) = \frac{\text{vol}(n, \frac{n-k}{2})}{2^{n-1}}.$$

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 $k, n \in \mathbf{N}_0$ and vertices u and v in the n -cube, we let

$$\text{ell}(u, v, k) := \{w \in \{0, 1\}^n \mid d_H(w, u) + d_H(w, v) \leq k\}.$$

- (1) What can you say about $\text{ell}(u, v, d_H(u, v))$?
- (2) What is the smallest k (in terms of n and $d_H(u, v)$) such that $\text{ell}(u, v, k)$ comprises the whole n -cube?
- (3) In general, how many vertices are in (what is the volume of) $\text{ell}(u, v, k)$?

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.

HINT: Relate this to covering codes.

Exercise 5.18 $n \in \mathbf{N}_0$. n real numbers, i.e. points on the real line \mathbf{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 \mathbf{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) $d \in \mathbf{N}$. The complement of the union of n hyperplanes in \mathbf{R}^d in general position¹¹ (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.

¹¹The right definition of general position is: Any $i \leq d$ 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 $n < d$.

Exercise 5.19

And now Generalize

$q \in \mathbf{N}$. 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.

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 $e : S \rightarrow \{0, 1\}^*$ is a prefix-free encoding (i.e. injective), then the average code length $(\sum_{x \in S} |e(x)|) / |S|$ 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

$$-\log_2 \left(\frac{\sum_{x \in S} 2^{-|e(x)|}}{|S|} \right) \leq \frac{\sum_{x \in S} -\log_2 2^{-|e(x)|}}{|S|} = \frac{\sum_{x \in S} |e(x)|}{|S|} \quad (6.1)$$

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 \mathbf{R}^+ .

□

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

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

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

¹We 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.

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 S_V denote the set of bijective mappings $\{1..n\} \rightarrow V$, and we will be sloppy² 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.

	function <code>encode</code> (α, F, V, π)
F CNF formula over	$\varepsilon \leftarrow$ empty string;
$V, \alpha \in \{0, 1\}^V, \pi \in S_V$.	for $i \leftarrow 1$ to $ V $ do
PRECONDITION:	$x \leftarrow \pi(i);$
$\alpha \in \text{sat}_F(V)$.	if $\{x\} \not\subset F$ and $\{\bar{x}\} \not\subset F$ then
POSTCONDITION:	$\varepsilon \leftarrow \varepsilon \alpha(x);$
returns a string in	$F \leftarrow F^{[x \mapsto \alpha(x)]};$
$\{0, 1\}^*$.	return $\varepsilon;$

Assuming F and V to be fixed, we will denote the output of the call `encode`(α, F, V, π) by $\text{enc}(\alpha, \pi)$. Decoding just reverts the process, as speci-

²Recall 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.

fied by procedure `decode()`. Here are two basic facts that follow quite easily.

$\varepsilon = (\varepsilon_j)_{j=1}^{ \varepsilon } \in \{0, 1\}^*$, F CNF formula over V , $\pi \in S_V$. POSTCONDITION: returns α with $\varepsilon = \text{enc}(\alpha, \pi)$, if it exists, and the empty map, otherwise.	<pre> function decode(ε, F, V, π) $j \leftarrow 0$; $\alpha \leftarrow \emptyset$; for $i \leftarrow 1$ to V do $x \leftarrow \pi(i)$; if $\{x\} \in F$ then $b \leftarrow 1$; elseif $\{\bar{x}\} \in F$ then $b \leftarrow 0$; else if $j = \varepsilon$ then return \emptyset; $j \leftarrow j + 1$; $b \leftarrow \varepsilon_j$; $\alpha \leftarrow \alpha \cup \{x \mapsto b\}$; $F \leftarrow F^{[x \mapsto b]}$; if $\square \in F$ or $j \neq \varepsilon$ then return \emptyset; else return α; </pre>
---	--

Observation 6.2 (i) Given a CNF formula F over V and $\pi \in S_V$, the mapping $\text{enc}(\cdot, \pi)$ is a prefix-free encoding of $\text{sat}_V(F)$.
 (ii) Given $\pi \in S_V$, procedure `decode()` with permutation π computes the inverse of $\text{enc}(\cdot, \pi)$ on the image of $\text{sat}_V(F)$ under $\text{enc}(\cdot, \pi)$.

Given an assignment $\alpha \in \text{sat}_V(F)$, we call a variable $x \in V$ *critical* for α if flipping the value of x in α stops it being satisfying. This requires at least one clause in F that has x or \bar{x} as the unique literal that is set to 1 by α . Such a clause is called *critical for*³ x . By⁴ $j(\alpha)$ we denote the number of critical variables for α , and we say that α 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 α 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 $n - \frac{j}{k}$.*

³Every critical variable has a clause which is critical for it, and every critical clause is critical for exactly one variable

⁴ For $\alpha \in \text{sat}_V(F)$, let $\deg(\alpha)$ denote the number of neighbors of α in the V -cube that are satisfying (recall Lemma 5.4). Then $\deg(\alpha) = n - j(\alpha)$.

Proof. Since α 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 $\text{vbl}(C_x)$ 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. \square

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 explicitly—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 $\frac{1}{n!} \sum_{\sigma \in S_V} |\text{enc}(\alpha, \sigma)| \leq n - \frac{j}{k}$. Hence,

$$\frac{1}{n!} \sum_{\sigma \in S_V} \underbrace{\left(\frac{1}{|A|} \sum_{\alpha \in A} |\text{enc}(\alpha, \sigma)| \right)}_{(*)} = \frac{1}{|A|} \sum_{\alpha \in A} \left(\frac{1}{n!} \sum_{\sigma \in S_V} |\text{enc}(\alpha, \sigma)| \right) \leq n - \frac{j}{k}$$

and thus for at least one permutation $\sigma \in S_V$ 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 $n - \frac{j}{k}$. \square

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

$$\bigwedge_{i=0}^{m-1} (x_{ik+1} \oplus x_{ik+2} \oplus \dots \oplus x_{ik+k})$$

has an equivalent k -CNF formula over variable set $\{x_1, x_2, \dots, x_n\}$ (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

$$\left(\frac{\sum_{i=1}^n \frac{1}{x_i}}{n} \right)^{-1} \leq \sqrt[n]{\prod_{i=1}^n x_i} \quad \text{for } n \in \mathbf{N} \text{ and } x_i \text{'s in } \mathbf{R}^+,$$

with the help of Jensen's Inequality.

Exercise 6.2Many j -Isolated Satisfying Assignments

$j \in \mathbf{N}_0$, $k, n \in \mathbf{N}$. 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 $\text{ppz}()$ 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.

F CNF formula over V .

POSTCONDITION:
returns an assign't
satisfying F or the
empty map.

function $\text{ppz}(F, V)$

$\pi \leftarrow_{\text{random}} S_V; \alpha \leftarrow \emptyset;$

for $i \leftarrow 1$ **to** $|V|$ **do**

$x \leftarrow \pi(i);$

if $\{x\} \in F$ **then** $b \leftarrow 1;$

elseif $\{\bar{x}\} \in F$ **then** $b \leftarrow 0;$

else $b \leftarrow_{\text{random}} \{0, 1\};$

$\alpha \leftarrow \alpha \cup \{x \mapsto b\}; F \leftarrow F^{[x \mapsto b]};$

if $\square \in F$ **then return** $\emptyset;$

return $\alpha;$

On the one hand, the procedure is a natural way to proceed: It simply builds a random satisfying assignment variable by variable, except that it

refrains from making too obvious mistakes with respect to generating a satisfying assignment (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.

Lemma 6.5 *The probability, that `ppz()` finds a satisfying assignment in a satisfiable $(\leq k)$ -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 σ then in order to generate a specific satisfying assignment α , it has $|\text{enc}(\alpha, \sigma)|$ free choices, which—since made at random—come out right with probability $2^{-|\text{enc}(\alpha, \sigma)|}$. Thus

$$\begin{aligned} \Pr(\text{ppz}() \text{ returns } \alpha) &= \sum_{\sigma \in \mathcal{S}_V} \Pr(\text{ppz}() \text{ returns } \alpha | \pi = \sigma) \cdot \Pr(\pi = \sigma) \\ &= \sum_{\sigma \in \mathcal{S}_V} 2^{-|\text{enc}(\alpha, \sigma)|} \cdot \frac{1}{n!} \\ &\geq 2^{\frac{1}{n!} \sum_{\sigma \in \mathcal{S}_V} -|\text{enc}(\alpha, \sigma)|} \\ &\geq 2^{-n+j(\alpha)/k} \end{aligned}$$

The penultimate inequality uses Jensen's Inequality (with the function $a \mapsto 2^a$ which is convex on \mathbf{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 α (i.e. $j(\alpha) = n$) then we 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).

$$\begin{aligned}
\Pr(\text{ppz}() \text{ returns some } \alpha \in \text{sat}_V(F)) &= \sum_{\alpha \in \text{sat}_V(F)} \Pr(\text{ppz}() \text{ returns } \alpha) \\
&\geq \sum_{\alpha \in \text{sat}_V(F)} 2^{-n+j(\alpha)/k} \\
&= 2^{-n+n/k} \sum_{\alpha \in \text{sat}_V(F)} 2^{-(n-j(\alpha))/k} \\
&\geq 2^{-n+n/k} \sum_{\alpha \in \text{sat}_V(F)} 2^{-(n-j(\alpha))} \\
&\geq 2^{-n+n/k}
\end{aligned}$$

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. \square

We are by now well-experienced⁵ 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 $\text{ppz}()$ 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 $O(2^{(2/3)n} \text{poly}(n)) = O(1.588^n)$ for $k = 3$ and $O(2^{(3/4)n} \text{poly}(n)) = O(1.682^n)$ 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 $\text{ppz}()$ 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 $\text{ppz}()$ 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 $(1-p)^{\lceil \lambda/p \rceil} \leq e^{-p \lceil \lambda/p \rceil} = e^{-\lambda}$.

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 α 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 α —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 α 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 α .

F CNF, α assign't, $r \in \mathbb{N}_0$. PRECONDITION: α total for F . POSTCONDITION: returns <code>true</code> if there is an assign't satisfying F at distance $\leq r$ from α . And <code>false</code> , otherwise.	function <code>sb</code> (F, α, r) if α satisfies F then return <code>true</code> ; elseif $r = 0$ then return <code>false</code> ; else $C \leftarrow_{\text{some in}} \{D \in F \mid \alpha \text{ does not satisfy } D\}$; for all $u \in C$ do if <code>sb</code> ($F^{[u \rightarrow 1]}$, $\alpha, r - 1$) then return <code>true</code> ; return <code>false</code> ;
--	--

It is easy to see that procedure `sb()` (“*search ball*”) serves its purpose: Given F and α , either α satisfies F , or for a clause C not satisfied by α at least one literal in C has to change (compared to α) 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 $r \in \mathbb{N}, k \in \mathbb{N}, k \geq 2$. For a $(\leq k)$ -CNF formula F , a call $\text{sb}(F, \alpha, r)$ with α an assignment on $V \supseteq \text{vbl}(F)$ entails at most $\frac{k^{r+1}-1}{k-1}$ calls to $\text{sb}()$ (including the initial call). Thus it can be implemented to run in time $O(k^r \text{poly}(n))$, $n := |V|$.

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. \square

An Easy Deterministic Procedure for 3-SAT. We see that for a (≤ 3) -CNF formula F over n variables and $r = \lfloor n/2 \rfloor$, the procedure takes time $O(3^{n/2} \text{poly}(n)) = O(1.733^n)$ 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 $\text{sb}(F, \alpha_0, \lfloor n/2 \rfloor)$ and $\text{sb}(F, \alpha_1, \lfloor n/2 \rfloor)$ where α_0 maps all variables to 0 and α_1 all variables to 1. Since every assignment has distance at most $\lfloor n/2 \rfloor$ either from α_0 or from α_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 $n \in \mathbb{N}$, $\text{vol}(n, r) := \sum_{i=0}^r \binom{n}{i}$ 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)

$$(8n\rho(1-\rho))^{-1/2} 2^{H(\rho)n} \leq \text{vol}(n, r) \leq 2^{H(\rho)n},$$

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

$$H\left(\frac{1}{4}\right) = -\frac{1}{4} \log_2 \frac{1}{4} - \frac{3}{4} \log_2 \frac{3}{4} = 2 - \frac{3}{4} \log_2 3.$$

That is, $\text{vol}(n, \frac{n}{4})$ is $(4 \cdot 3^{-3/4})^n$ up to a polynomial factor.

A Simple Randomized Improvement. Let F be a satisfiable (≤ 3) -CNF formula over V , $n := |V|$. If we choose an assignment α u.a.r. in $\{0, 1\}^V$, then the ball of radius $r := \lfloor n/4 \rfloor$ centered at α contains a fixed satisfying assignment α^* with probability $\frac{\text{vol}(n, r)}{2^n} \geq (2 \cdot 3^{-3/4})^n / \text{poly}(n)$, which will be found by a call $\text{sb}(F, \alpha, r)$ in time $O(3^{n/4} \text{poly}(n))$. 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 $(3^{3/4}/2)^n \text{poly}(n)$ and the expected time is at most $O((3^{n/4} 3^{3n/4}/2^n) \text{poly}(n)) = O((3/2)^n \text{poly}(n))$.

If we take a not necessarily satisfiable (≤ 3) -CNF formula over n variables, and we repeat the above search for $\lceil \lambda 2^n / \text{vol}(n, r) \rceil$ rounds without success, then—along by now familiar lines¹—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 (≤ 3) -CNF formula over n variables in time $O(\sqrt{3}^n \text{poly}(n)) = O(1.733^n)$.*

$\lambda \in \mathbb{R}, \lambda > 0$. *There is a randomized algorithm with running time $O(\lambda \cdot 1.5^n \text{poly}(n))$ that either finds a satisfying assignment of a (≤ 3) -CNF formula over n variables, or concludes unsatisfiability with error probability at most $e^{-\lambda}$.*

Both results will now be 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 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 $O(2^{(1-H(\rho))n} \text{poly}(n))$ elements (Lemma 5.9).

¹Recall Footnote 5 in Chapter 6.

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)

$$2^{(1-H(\rho))n} k^{\rho n} = 2^{(1-H(\rho)+\rho \log_2 k)n}.$$

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 $0 < \rho < 1$.

$$\begin{aligned} g\left(\frac{1}{k+1}\right) &= 1 + \frac{1}{k+1} \log_2 \frac{1}{k+1} + \frac{k}{k+1} \log_2 \frac{k}{k+1} + \frac{1}{k+1} \log_2 k \\ &= 1 + \left(\frac{1}{k+1} + \frac{k}{k+1}\right) \log_2 \frac{1}{k+1} + \left(\frac{k}{k+1} + \frac{1}{k+1}\right) \log_2 k \\ &= 1 + \log_2 \frac{k}{k+1} \end{aligned}$$

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 $n \in \mathbf{N}$, $\rho \in \mathbf{R}$, $0 < \rho \leq \frac{1}{2}$. *A code of length n with covering radius at most ρn 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 $d \in \mathbf{N}$ and let n be an integer multiple of d . First we construct a code \mathcal{C}' of length $\frac{n}{d}$ via the procedure described above. Then we let $\mathcal{C} := \mathcal{C}'^d$, that is the set of all possible concatenations of d (not necessarily distinct) elements in \mathcal{C}' . We have spent time $O((2^{3n/d} + |\mathcal{C}'|) \text{poly}(n))$ and the size of \mathcal{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 poly_d indicates that the degree of the polynomial depends on d .

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 ρ).

Lemma 7.4 $n \in \mathbf{N}$, $\rho \in \mathbf{R}$, $0 < \rho \leq \frac{1}{2}$. *A code of length n with covering radius at most ρn and of size $O(2^{(1-H(\rho))n} \text{poly}_\rho(n))$ can be constructed in time and space $O(2^{(1-H(\rho))n} \text{poly}_\rho(n))$. (poly_ρ is a polynomial of degree $O(\frac{1}{1-H(\rho)})$.)*

Procedure $\text{cov}()$ 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.

²A 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.

	function cov(F)
F CNF formula.	$k \leftarrow \max_{C \in \mathcal{F}} C ; \quad \rho \leftarrow \frac{1}{k+1};$
POSTCONDITION:	$n \leftarrow \text{vbl}(F) ;$
returns true if F is	$\mathcal{C} \leftarrow \text{code of length } n \text{ with cov. radius } \leq \rho n;$
satisfiable. And false ,	for all $\alpha \in \mathcal{C}$ do
otherwise.	if sb(F, α , $\lfloor \rho n \rfloor$) then return true;
	return false;

Theorem 7.5 $k, n \in \mathbf{N}$. *Satisfiability of a $(\leq k)$ -CNF formula F over n variables can be decided in time $O\left(\left(2 - \frac{2}{k+1}\right)^n \text{poly}_k(n)\right)$.*

NOTE 7.1 Further tailored improvements for (≤ 3) -CNF formulas lead to a variant of sb() with running time $O(2.848^r \text{poly}(n))$ 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 $m \in \mathbf{N}$. *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 $n \in \mathbf{N}, r \in \mathbf{N}_0$. *Show that there exists a code of length n of covering radius at most r with at most $\left\lceil \frac{(\ln 2) n \cdot 2^n}{\text{vol}(n, r)} \right\rceil$ elements. ($\ln 2 \approx 0.6931\dots$)*

Exercise 7.3 $r \in \mathbf{N}_0, n \in \mathbf{N}, r \leq n$. *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 (≤ 3) -CNF formula over n variables and radius argument r in $O(c^r \text{poly}(n))$, 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*”).

F CNF formula, $s \in \mathbb{N}$,
 α return parameter.
POSTCONDITION:
 if `true` is returned then
 α is an assign't on
 $\text{vbl}(F)$ satisfying F .

```

function rfb( $F, s, \alpha$ )
 $\alpha \leftarrow_{\text{random}} \{0, 1\}^{\text{vbl}(F)}$ ;
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;

```

The idea is to repeatedly call procedure `rfb()` with appropriate s . With clairvoyant foresight, let us use $s := 3n$, $n := |\text{vbl}(F)|$, 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 $(\leq k)$ -CNF formula, we will have to analyze the success probability of `rfb()` for a satisfiable $(\leq k)$ -CNF formula.

F CNF formula.
POSTCONDITION:
 returns assignment α
 on $\text{vbl}(F)$ satisfying F ,
 if it terminates.

```

function sch( $F$ )
 $n \leftarrow |\text{vbl}(F)|$ ;
repeat
  until rfb( $F, 3n, \alpha$ );
return  $\alpha$ ;

```

7.2.1 Random Walks Revisited

Similarly to the situation with procedure $\text{rf}()$ in Section 3.2 we consider a satisfiable ($\leq k$)-CNF formula F and we fix some satisfying assignment α^* on $\text{vbl}(F)$. Let α be some assignment not satisfying F . If the value of a random literal in an unsatisfied clause C is flipped in α , then with probability at least³ $\frac{1}{|C|} \geq \frac{1}{k}$ the Hamming distance between α and α^* 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 $0 \leq j \leq n$, $n := |\text{vbl}(F)|$. That is, the Hamming distance between α and α^* 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 $1 - q = 1 - \frac{1}{k}$.

We model now a Markov chain that will be pessimistic compared to $\text{rfb}()$ in the sense that—via appropriate coupling— $\text{rfb}()$ 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

$$\begin{aligned} p_{S,j} &:= \binom{n}{j} \frac{1}{2^n}, \text{ for } 0 \leq j \leq n, \\ p_{i,i+1} &:= 1 - q, \text{ for } i \in \mathbb{N}, \\ p_{i,i-1} &:= q, \text{ for } i \in \mathbb{N} \\ p_{0,0} &:= 1, \text{ and} \\ &:= 0, \text{ otherwise.} \end{aligned}$$

Note that this chain allows motion into states greater than n , while this is not feasible for the Hamming distance between α and α^* .

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 $\Pr(Y \leq 3n)$, since this will provide a lower bound for the success probability of $\text{rfb}()$. A problem we face here is that

³Since F is satisfiable, $C \neq \square$.

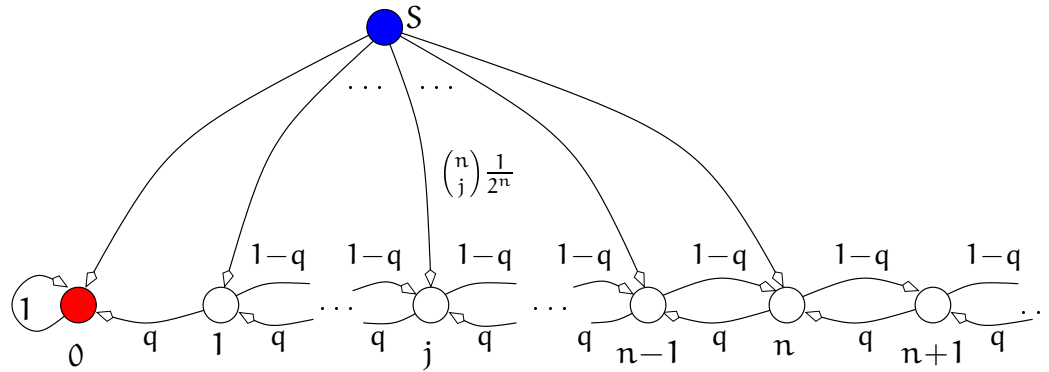


Figure 7.1: The “pessimistic” Markov chain for $\text{rfb}()$.

for $q \leq \frac{1}{2}$ 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 $j \in \mathbb{N}$, 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 $j \in \mathbb{N}$,*

$$\Pr(X_j < \infty) = \left(\frac{q}{1-q} \right)^j \quad \text{and} \quad \mathbb{E}(X_j | X_j < \infty) = \frac{j}{1-2q}.$$

Proof. $j = 0$ is trivial, so let us assume $j \in \mathbb{N}$. According to the ballot theorem, cf. [Grimmett, Stirzaker, 1992], for $i \in \mathbb{N}$, 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 is⁴ $\binom{2i+j}{i} \frac{j}{2i+j}$. 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

$$(\mathcal{B}_2(z))^r = \sum_i \binom{2i+r}{i} \frac{r}{2i+r} z^i,$$

for all $r \in \mathbf{N}$, 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

$$\begin{aligned} \mathbb{E}(X_j | X_j < \infty) &= \frac{1}{\Pr(X_j < \infty)} \sum_{i=0}^{\infty} (2i+j) \binom{2i+j}{i} \frac{j}{2i+j} (1-q)^i q^{i+j} \\ &= \frac{j q^j}{\Pr(X_j < \infty)} \sum_{i=0}^{\infty} \binom{2i+j}{i} (q(1-q))^i \\ &= \frac{j q^j}{q^j (\mathcal{B}_2(q(1-q)))^j} \frac{(\mathcal{B}_2(q(1-q)))^j}{\sqrt{1-4q(1-q)}} \quad \text{see (7.1) and (7.2)} \\ &= \frac{j}{1-2q} \end{aligned}$$

⁴This 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 $\binom{2i+j-1}{i} - \binom{2i+j-1}{i-1} = \binom{2i+j}{i} \frac{j}{2i+j}$.

exploiting the identity

$$\frac{\mathcal{B}_2(z)^r}{\sqrt{1-4z}} = \sum_i \binom{2i+r}{i} z^i, \quad (7.2)$$

for all $r \in \mathbf{N}$, cf. [Graham *et al.*, 1989]. \square

Lemma 7.7 *If $q < \frac{1}{2}$,*

$$\Pr(Y < \infty) = \left(\frac{1}{2(1-q)} \right)^n. \quad (7.3)$$

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 $\sum_{i=0}^n \binom{n}{i} x^i y^{n-i} = (x+y)^n$. \square

Lemma 7.8 *If $q < \frac{1}{2}$,*

$$\mathbb{E}(Y | Y < \infty) = \frac{qn}{1-2q}.$$

Proof. We use $\Pr(A|B) = \Pr(A \wedge B) / \Pr(B)$ for

$$\begin{aligned} \mathbb{E}(Y | Y < \infty) &= \sum_{i=0}^{\infty} i \Pr(Y = i | Y < \infty) \\ &= \frac{1}{\Pr(Y < \infty)} \sum_{i=0}^{\infty} i \Pr(Y = i) \\ &\quad (\text{since } (Y = i) \wedge (Y < \infty) \Leftrightarrow (Y = i)) \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{\Pr(Y < \infty)} \sum_{i=0}^{\infty} i \left(\sum_{j=0}^n \binom{n}{j} \frac{1}{2^n} \Pr(X_j = i) \right) \\
&= \frac{1}{2^n \Pr(Y < \infty)} \sum_{j=0}^n \binom{n}{j} \sum_{i=0}^{\infty} i \Pr(X_j = i) \\
&= \frac{1}{2^n \Pr(Y < \infty)} \sum_{j=0}^n \binom{n}{j} \mathbb{E}(X_j | X_j < \infty) \Pr(X_j < \infty) \\
&\quad (\text{since } \Pr(X_j = i) = \Pr(X_j = i \wedge X_j < \infty)) \\
&\quad \quad \quad = \Pr(X_j = i | X_j < \infty) \Pr(X_j < \infty)) \\
&= (1-q)^n \sum_{j=0}^n \binom{n}{j} \frac{j}{1-2q} \left(\frac{q}{1-q} \right)^j \\
&\quad (\text{see Lemmas 7.7 and 7.6}) \\
&= \frac{n(1-q)^n}{1-2q} \sum_{j=1}^n \binom{n-1}{j-1} \left(\frac{q}{1-q} \right)^j \\
&= \frac{n(1-q)^n}{1-2q} \frac{q}{1-q} \sum_{j=1}^n \binom{n-1}{j-1} \left(\frac{q}{1-q} \right)^{j-1} \\
&= \frac{n(1-q)^n}{1-2q} \frac{q}{1-q} \left(1 + \frac{q}{1-q} \right)^{n-1} \\
&= \frac{n(1-q)^n}{1-2q} \frac{q}{1-q} \left(\frac{1}{1-q} \right)^{n-1} \\
&= \frac{nq}{1-2q}.
\end{aligned}$$

□

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 μ for $E(Y|Y < \infty)$. Observe that

$$\Pr(Y > \lambda\mu | Y < \infty) < \frac{1}{\lambda}$$

and

$$\Pr(Y \leq \lambda\mu) = \Pr(Y \leq \lambda\mu | Y < \infty) \cdot \Pr(Y < \infty),$$

since $(Y \leq \lambda\mu \wedge Y < \infty) \Leftrightarrow (Y \leq \lambda\mu)$. \square

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

$$\Pr(Y \leq 3n) \geq \Pr\left(Y \leq \frac{3qn}{1-2q}\right) > \frac{2}{3} \left(\frac{1}{2} \left(\frac{k}{k-1}\right)\right)^n.$$

We skip the formal coupling argument for arguing that the success probability of `rfb()` is indeed lower bounded by $\Pr(Y \leq 3n)$, 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 $e^{-\lambda}$. 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 $O\left(\left(\frac{2(k-1)}{k}\right)^n \text{poly}(n)\right)$.*

For $\lambda \in \mathbf{R}^+$ there is a Monte Carlo algorithm that needs time $O\left(\lambda \left(\frac{2(k-1)}{k}\right)^n \text{poly}(n)\right)$ 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 $e^{-\lambda}$.

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 $E(Y|Y < \infty) = \infty$ (although $\Pr(Y < \infty) = 1$).

NOTE 7.3 The randomized algorithm of this section is due to [Schöning, 2002], an ingenious follow-up of Papadimitriou'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 7*

Derandomizing Schönning's Algorithm

Chapter by Dominik Scheder, with additions by Robin Moser.

In Chapter 7 we have seen the deterministic algorithm `cov()` (page 110) deciding k -SAT in time $O\left(\left(\frac{2k}{k+1}\right)^n \text{poly}(n)\right)$ and the randomized algorithm `sch()` (page 111), Schönning's algorithm, which has an expected running time of $O\left(\left(\frac{2(k-1)}{k}\right)^n \text{poly}(n)\right)$. In this chapter, we will derandomize Schönning's algorithm, i.e., give a deterministic algorithm with a running time that (almost) matches that of `sch()`. In Section 7*.1 we describe and analyze that algorithm. In Section 7*.2 we give some hindsight why this approach works while others do not.

7*.1 The Algorithm

We restate algorithm `sb()` below and take a closer look at it. It runs in time $O(k^r \text{poly}(n))$ and decides whether the Hamming ball $B_r(\alpha)$ contains a satisfying assignment. We give a faster algorithm which solves a somewhat simpler problem still good enough for our purposes.

Lemma 7*.1 *There is a deterministic algorithm `sb-fast` that takes as input an $(\leq k)$ -CNF formula F , an assignment α on $\text{vbl}(F)$, and a number $r \in \mathbb{N}_0$, runs in time $(k-1)^{r+o(n)}$, and*

- *returns `true` if $B_\alpha(r)$ contains a satisfying assignment of F ,*
- *returns `false` if F is unsatisfiable,*

F CNF, α assign't, $r \in \mathbb{N}_0$.	function $\text{sb}(F, \alpha, r)$
PRECONDITION: α total for F .	if α satisfies F then return true ;
POSTCONDITION: returns true if there is an assign't satisfying F at distance $\leq r$ from α . And false , otherwise.	elseif $r = 0$ then return false ;
	else
	$C \leftarrow_{\text{some in}} \{D \in F \mid \alpha \text{ does not satisfy } D\}$;
	for all $u \in C$ do
	if $\text{sb}(F^{[u \rightarrow 1]}, \alpha, r - 1)$ then return true ;
	return false ;

and otherwise returns either true or false .

Together with the covering code construction in Chapter 7, this yields the main result of this chapter:

Theorem 7*.2 *There is a deterministic algorithm deciding k -SAT in time $\left(\frac{2(k-1)}{k}\right)^{n+o(n)}$.*

For the rest of this chapter, we will prove Lemma 7*.1. We start with a definition and two basic observations.

Definition 7*.3 *Let F be an $(\leq k)$ -CNF formula and α be an assignment on $\text{vbl}(F)$. We say F is good with respect to α and k if every clause $C \in F$ that is not satisfied by α has at most $k - 1$ literals.*

Observation 7*.4 *If F is good with respect to α and k , and u is some literal, then $F^{[u \rightarrow 1]}$ is also good with respect to α and k .*

Observation 7*.5 *If F is good with respect to α and k , then $\text{sb}(F, \alpha, r)$ runs in time $O((k - 1)^r \text{poly}(n))$.*

This observation implies that Lemma 7*.1 holds for good formulas. What do we do if F is not good, i.e., if F contains unsatisfied clauses of size k ? Collect greedily a maximal set $G = \{C_1, \dots, C_m\} \subseteq F$ of pairwise disjoint clauses that are unsatisfied by α .

Observation 7*.6 *Let β be an assignment on $\text{vbl}(G)$. Then $F^{[\beta]}$ is good with respect to α and k .*

Exercise 7*.1 *Prove Observation 7*.4, 7*.5, and 7*.6.*

Our algorithm `sb-fast` considers two cases. First, if $m \leq \log_k \log_2 n$, it iterates over all $2^{k|G|}$ assignments β on $\text{vbl}(G)$ and calls $\text{sb}(F^{[\beta]}, \alpha, r)$. Since each of the formulas $F^{[\beta]}$ is good, this takes time at most

$$2^{km}(k-1)^r \text{poly}(n) \leq (k-1)^r \text{poly}(n) .$$

Otherwise, if $m > \log_k \log_2 n$, set $t = \lfloor \log_k \log_2 n \rfloor$ and $G' = \{C_1, \dots, C_t\}$. At this point we have to take a little detour.

k-ary Covering Codes

The set $\{1, \dots, k\}^t$ is endowed with a Hamming distance, just as the Hamming cube is: For $w, w' \in \{1, \dots, k\}^t$, we define

$$d_H(w, w') := |\{i \in \{1, \dots, t\} \mid w_i \neq w'_i\}|.$$

A k -ary Hamming ball around w of radius s is the set

$$B_s^{(k)}(w) := \{w' \in \{1, \dots, k\}^t \mid d_H(w, w') \leq s\} .$$

The number of elements in such a ball is independent of w . We define and observe

$$\text{vol}^{(k)}(t, s) := |B_s^{(k)}(w)| = \sum_{i=0}^s \binom{t}{i} (k-1)^i .$$

The equation is easy to see: There are $\binom{t}{i}$ ways to choose i positions on which w' is supposed to differ from w . On each such position, we have $k-1$ ways to choose w'_i . We can and also must define covering codes. A set $\mathcal{C} \subseteq \{1, \dots, k\}^t$ is called a k -ary code of length t and covering radius s if

$$\bigcup_{w \in \mathcal{C}} B_s^{(k)}(w) = \{1, \dots, k\}^t$$

and s is the smallest integer with this property.

Lemma 7*.7 *Let $t, k \in \mathbb{N}$ and $s \in \mathbb{N}_0$. There exists a k -ary code of length t and covering radius s with at most*

$$\left\lceil \frac{t \ln(k) k^t}{\text{vol}^{(k)}(t, s)} \right\rceil$$

elements.

The proof is almost identical to the proof of Lemma 5.9 on page 91 of the lecture notes. Observe that for our choice $t = \lceil \log_k \log_2 n \rceil$ it holds that $|\{1, \dots, k\}^t| = k^t \leq \log_2 n$, thus there are at most $2^{\log_2 n} = n$ subsets $C \subseteq \{1, \dots, k\}^t$. By iterating through all of them, we can easily find a smallest code of covering radius s .

Consider our set $G' = \{C_1, \dots, C_t\}$ of unsatisfied, pairwise disjoint clauses. Any satisfying assignment α^* of F must satisfy at least one literal in each $C_i \in G'$. Since they are pairwise disjoint, this implies $d_H(\alpha, \alpha^*) \geq t$. There are exactly k^t assignments that satisfy G' and have distance exactly t from α . Each such assignment can be represented by a $w \in \{1, \dots, k\}^t$ in the obvious way. To be more precise, for $w \in \{1, \dots, k\}^t$ we define $\alpha[G', w]$ to be the assignment which we obtain from α by flipping the w_i th literal in C_i , for $1 \leq i \leq t$. For this, we need to define some arbitrary but fixed order on the literals in each C_i . If G' is understood from the context, we write $\alpha[w]$ instead of $\alpha[G', w]$.

Example. Consider $G' = \{\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \{x_3, y_3, z_3\}\}$, $\alpha = (0, \dots, 0)$ and $t = 3$. Let $w = (2, 3, 3)$. Then $\alpha[w]$ is the assignment that sets y_1, z_2 , and z_3 to 1 and all other variables to 0.

Observation 7*.8 *We observe the following facts about $\alpha[w]$:*

- $d_H(\alpha, \alpha[w]) = t$ for every $w \in \{1, \dots, k\}^t$.
- If α^* satisfies F , then there is some $w^* \in \{1, \dots, k\}^t$ such that $d_H(\alpha[w^*], \alpha^*) = d_H(\alpha, \alpha^*) - t$.
- Let $w, w' \in \{1, \dots, k\}^t$. Then $d_H(\alpha[w], \alpha[w']) = 2d_H(w, w')$.

Lemma 7*.9 *Let t and G' be defined as above, and let $C \subseteq \{1, \dots, k\}^t$ be a k -ary code of covering radius s . If α^* is a satisfying assignment of F , then there is some $w \in C$ such that $d_H(\alpha[w], \alpha^*) \leq d_H(\alpha, \alpha^*) - t + 2s$.*

In particular, if $B_r(\alpha)$ contains a satisfying assignment, then there is some $w \in C$ such that $B_{r-t+2s}(\alpha[w])$ contains it, too.

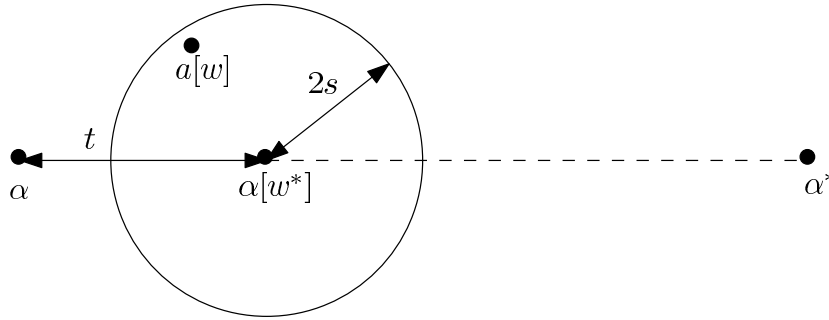


Figure 7*.1: Illustration of Lemma 7*.9. The distance from $\alpha[w]$ to α^* is at most the distance from $\alpha[w^*]$ to α^* plus $2s$.

Proof of Lemma 7.9.* By Observation 7*.8, there is some $w^* \in \{1, \dots, k\}^t$ such that $d_H(\alpha[w^*], \alpha^*) = d_H(\alpha, \alpha^*) - t \leq r - t$. Since \mathcal{C} has covering radius s , there is some $w \in \mathcal{C}$ such that $d_H(w, w^*) \leq s$, and by Observation 7*.8, $d_H(\alpha[w], \alpha[w^*]) \leq 2s$. The lemma now follows from the triangle inequality. The proof is illustrated in Figure 7*.1. \square

Below we state *sb-fast* formally. We compute an optimal k -ary code \mathcal{C} of length t and covering radius $s = t/k$.

	function <i>sb-fast</i> ($F, \alpha, r, \mathcal{C}$)
	if α satisfies F then return <i>true</i> ;
	elseif $r = 0$ then return <i>false</i> ;
	else
	$G \leftarrow$ a maximal set of pairwise disjoint
	k -clauses of F unsatisfied by α ;
	if $ G \leq t := \lceil \log_k \log_2 n \rceil$ then
	return $\bigvee_{\beta \in \{0,1\}^{\text{vbl}(G)}} \text{sb}(F^{[\beta]}, \alpha, r)$;
	else
	$G' \leftarrow$ the first t clauses of G
	return $\bigvee_{w \in \mathcal{C}} \text{sb-fast}(F, \alpha[G', w], r - t + 2t/k, \mathcal{C})$;
F an $(\leq k)$ -CNF, α	
assign't, $r \in \mathbb{N}_0$, \mathcal{C} a	
k -ary code.	
PRECONDITION:	
α total for F , \mathcal{C} has	
covering radius at	
most t/k .	
POSTCONDITION:	
returns <i>true</i> if $B_\alpha(r)$	
contains a satisfying	
assignment of F , <i>false</i>	
if F is unsatisfiable,	
else returns either <i>true</i>	
or <i>false</i> .	

Correctness of *sb-fast* follows from the above discussion: If there is some $\alpha^* \in B_r(\alpha)$ that satisfies F , then for at least one $w \in \mathcal{C}$ it holds that $d_H(\alpha[w], \alpha^*) \leq r - t + 2t/k$, thus the corresponding recursive call to *sb-fast* will be successful.

What about the running time? If $|G| \leq t$, then every call to $\text{sb}(F^{[B]}, \alpha, r)$ runs in time $O((k-1)^r \text{poly}(n))$. Otherwise, the procedure *sb-fast* calls itself recursively for each $w \in \mathcal{C}$. Thus, every level further into the recursion, the parameter r decreases by $t - 2t/k$. The running time is therefore

$$|\mathcal{C}|^{r/(t-2t/k)} \text{poly}(n) .$$

To compute this, we have to estimate the size of \mathcal{C} . Recall that $s = t/k$.

$$\begin{aligned} |\mathcal{C}| &\leq \frac{k^t \text{poly}(t)}{\text{vol}^{(k)}(t, s)} \leq \frac{k^t \text{poly}(t)}{\binom{t}{s} (k-1)^s} \\ &\leq \frac{k^t \text{poly}(t)}{\left(\frac{t}{s}\right)^s \left(\frac{t}{t-s}\right)^{t-s} (k-1)^s} \\ &= \frac{k^t \text{poly}(t)}{k^s \left(\frac{k}{k-1}\right)^{t-s} (k-1)^s} \\ &= (k-1)^{t-2s} \text{poly}(t) . \end{aligned}$$

Note that we have used the convenient approximation

$$\binom{t}{s} \geq 2^{tH(\frac{s}{t})} = 2^{t(-\frac{s}{t} \log \frac{s}{t} - \frac{t-s}{t} \log \frac{t-s}{t})} = \left(\frac{t}{s}\right)^s \left(\frac{t}{t-s}\right)^{t-s} .$$

Therefore,

$$|\mathcal{C}|^{r/(t-2t/k)} \leq \left((k-1)^{t-2s} \text{poly}(t)\right)^{r/(t-2s)} = (k-1)^r \text{poly}(t)^{r/(t-2s)} .$$

Since t is a growing function in n , the term $\text{poly}(t)^{1/(t-2s)}$ converges to 1 as n grows, and the running time is at most $(k-1)^{r+o(n)}$. This completes the proof of Lemma 7*.1.

7*.2 Hindsight: Why and How?

At this point, we want to give some intuition on why *sb-fast*() is faster than *sb*() and why it took so long for *sb-fast*() to be discovered.

An Easier Problem?

The first reason is that `sb-fast()` solves a *simpler* problem than `sb()`: We define the problem BALL-k-SAT:

Ball-k-SAT: Given an $(\leq k)$ -CNF formula F over n variables, a truth assignment α to these variables, and a natural number r . Decide whether $B_r(\alpha)$ contains a satisfying assignment.

The function `sb()` solves BALL-k-SAT in time $O(k^r \text{poly}(n))$. Some minutes of thought reveal that `sb-fast()` does not decide BALL-k-SAT.

Exercise 7*.2 *Explain why `sb-fast()` does not solve BALL-k-SAT. What has to happen for it to return a wrong answer?*

What is the exact problem that `sb-fast()` solves, if it is not BALL-k-SAT? Well, it turns out to be a *promise* version thereof:

Promise-Ball-k-SAT: Given an $(\leq k)$ -CNF formula F , an assignment α , and a radius r , PROMISE-BALL-k-SAT is the problem of distinguishing between the case that $B_r(\alpha)$ contains a satisfying assignment for F and the case that F is unsatisfiable.

In this context, *distinguishing* means the following: If $B_r(\alpha)$ contains a satisfying assignment for F , the algorithm must return `true`. If F is unsatisfiable, it must return `false`. Otherwise, i.e., if F is satisfiable, but not within $B_r(\alpha)$, the answer is arbitrary.

Exercise 7*.3 *Give a good argument why BALL-k-SAT is in general more difficult than PROMISE-BALL-k-SAT.*

Hint: Consider the case $k = 2$.

Allowing Mistakes

In Chapter 7 we analyzed `rfb()` (the random walk algorithm on page 111), by fixing some satisfying assignment α^* and modelling the evolution of $d_H(\alpha, \alpha^*)$ as a Markov chain. At any given step of `rfb()` let us say that it *makes a mistake* if this distance increases by 1.

The probability that $\text{rfb}()$ is successful is at least $\Omega((k-1)^{-r})$ where r is the initial distance $d_H(\alpha, \alpha^*)$. However, if we insist on $\text{rfb}()$ being successful *without making any mistakes*, this probability drops to k^{-r} : In the worst case, there is exactly one correct choice in each step! It is much more realistic to reach the goal after having made (and corrected) some mistakes on the way than to reach the goal without any mistakes (that's a lesson for life). Note that $\text{sb-fast}()$ allows mistakes: It performs t correction steps at once, and within this window, it tolerates up to t/k mistakes.

Correction Strings

Imagine $\text{rfb}()$ (the random walk algorithm on page 111) being called with an additional parameter: $\text{rfb}(F, t, \alpha, w)$, where t is the number of corrections steps to be made, and $w \in \{1, \dots, k\}^t$ is a *correction string* telling the algorithm which corrections to make. That is, if $w_i = j$, then in the i^{th} step, the algorithm flips the assignment of the j^{th} literal in C . Clearly, if $w \in \{1, \dots, k\}^t$ is sampled uniformly at random, this is the same as the original $\text{rfb}(F, t, \alpha)$. The function $\text{rfb}(F, t, \alpha)$ is called with $t = 3n$, thus $\{1, \dots, k\}^{3n}$ is the space of all possible correction strings. Why can we not simply compute a covering code for this space?

Imagine two iterations $j_1 < j_2$ of the loop in $\text{rfb}()$. In step j_1 the algorithm selects an unsatisfied clause C_{i_1} , and in step j_2 it selects C_{j_2} . Which clause C_{j_2} it picks may depend on the random choice it made at step j_1 . Therefore, what it means for a correction string to be *correct at its j_2^{th} position* depends on its j_1^{th} position. Therefore, two correction strings might have Hamming distance 1, but still it is possible that the first string is correct in every position and the second is wrong everywhere. This renders useless the whole machinery of covering codes.

Note that the picture becomes less bleak if there are disjoint unsatisfied clauses C_1, \dots, C_t and our correction string tells us only what to do on those clauses: Which literal of C_{j_2} is the "correct one" does in fact not depend on which literal the algorithm flipped in clause C_{i_1} . Therefore, in this scenario the covering code machinery works, and this is exactly what we exploit in sb-fast .

Taking a Closer Look: Typical Executions

In order to enable the reader to retrace the thinking steps that lead to the discovery of the three key ideas we have just sketched, let us dive somewhat deeper into the analysis of the randomized version of Schönig's algorithm. Place yourself in the position of someone who has just studied Schönig and is trying to derandomize it properly.

For the analysis of the randomized variant in Chapter 7, we have coupled the algorithm to the execution of a Markov Chain $M_n = \{D_i\}_{i \geq 0}$ with an initial starting state $D_0 \sim \text{Bin}(n, \frac{1}{2})$ (that is the distance of the randomly chosen starting assignment from α^*) and such that¹ for $i \geq 1$,

$$D_i = \begin{cases} D_{i-1} - 1 & \text{with probability } \frac{1}{k} \\ D_{i-1} + 1 & \text{with probability } \frac{k-1}{k} \end{cases},$$

where these steps correspond to the flips of literals in violated clauses which can either be good and take you closer to α^* or bad and take you further away. We have then determined that for the probability for this process to reach state 0 within the first $3n$ steps, we have

$$\Pr(\exists i \leq 3n : D_i = 0) \geq \left(\frac{k}{2(k-1)} \right)^n \cdot \frac{1}{\text{poly}(n)}.$$

However, to reach this performance, we use lots of random bits in every run of the algorithm: first n random bits to select the assignment to start from, then another $\log k$ random bits for choosing the literal to flip in each step. If we make at most $3n$ steps, this yields a total of $n + 3n \log k$ random bits that are being used and there are thus $2^n \cdot k^{3n}$ possibilities to select those. For a trivial derandomization, we can just explore *all* possibilities of selecting the random bits yielding a very slow algorithm. On the other hand, the probabilistic analysis has demonstrated that not just a single, but a good portion of these possibilities lead to the zero state. We are thus in the situation that we have lots of possibilities to explore, many of which are good. What we are looking for is a *hitting set* for the good runs of the algorithm, that is a smaller set of random inputs, small enough for us

¹for *this* section, assume that the Markov Chain M_n is also extended to the left, into the negative numbers. This does not have an impact on the coupling or any statements we have made earlier.

to cycle through all members, for which we have a guarantee that one of its members leads to a success. In order to identify such a hitting set, we set out by asking ourselves whether really *all* ways of reaching the solution are relevant, or whether restricting to 'typical successful runs' of Schöning already yields a good success probability.

Let E be any event determined by the random experiment of executing Markov Chain M_n . We say that E is *typical* for Schöning if

$$\Pr(E \wedge (\exists i \leq 3n : D_i = 0)) \geq \left(\frac{k}{2(k-1)} \right)^{n+o(n)},$$

i.e. intuitively if (in our definition of 'success') requiring E to occur in addition to just reaching state zero lowers the success probability only by a negligible amount. Note that for such an asymptotic statement to make sense, $E = E(n)$ must represent a whole sequence of events, one for every n , but we will omit the index for simplicity.

Here are two basic observations.

Observation 7*.10 *If $E_1 \cup E_2 \cup E_3 \cup \dots \cup E_r$ for some suitably small $r = 2^{o(n)}$ is a cover of the probability space, i.e. if*

$$\Pr(E_1 \cup E_2 \cup E_3 \cup \dots \cup E_r) = 1,$$

then there exists i such that E_i is typical for Schöning.

This is a trivial consequence of a probabilistic pigeonhole principle: if there are only $2^{o(n)}$ events covering the probability space, then one of them must intersect with at least $2^{-o(n)}$ fraction of the event that Schöning succeeds. Note that in this observation, again, all $E_j = E_j(n)$ and $i = i(n)$ are sequences depending on n such that we can make the necessary asymptotic statement, but we leave out the indices.

Observation 7*.11 *If E is an event which is typical for Schöning and $E_1 \cup E_2 \cup E_3 \cup \dots \cup E_r = E$ for some $r = 2^{o(n)}$ form a cover of that event, then there exists i such that E_i is typical for Schöning.*

This follows immediately by the same pigeonhole argument, together with noting that $2^{-o(n)} \cdot 2^{-o(n)} = 2^{-o(n)}$.

Let us make a first example for such a typical event. We want to find out what is a typical distance D_0 of the initial assignment from α^* . For $i = 0, 1, \dots, n$, let E_i be the event that $D_0 = i$, i.e. that the initial assignment α_0 has distance exactly i from α^* . Observation 7*.10 immediately yields that *one* of these events must be typical. We claim that $E_{n/k}$ is typical for Schöning, i.e. that in a typical successful run, α_0 has distance n/k from α^* . To see this, recall that in the probabilistic analysis, we have proved that when starting from initial state j , the probability that we reach the state zero within $3n$ steps is at least $(k-1)^{-j+o(n)}$, hence for the total success probability we get

$$\Pr \left(E_{\frac{n}{k}} \wedge (\exists i \leq 3n : D_i = 0) \right) = \frac{\binom{n}{n/k}}{2^n} \cdot (k-1)^{-\frac{n}{k}+o(n)}.$$

Using Lemma 5.6, we obtain that this is

$$\begin{aligned} &\geq \frac{2^{n \cdot H(\frac{1}{k})+o(n)}}{2^n} \cdot (k-1)^{-\frac{n}{k}+o(n)} = \\ &= \left(2 \cdot \left(\frac{1}{k} \right)^{\frac{1}{k}} \left(\frac{k-1}{k} \right)^{\frac{k-1}{k}} (k-1)^{\frac{1}{k}} \right)^{-n+o(n)} = \\ &= \left(\frac{2(k-1)}{k} \right)^{-n+o(n)}, \end{aligned}$$

as claimed. So it is typical that $D_0 = n/k$, this means that we can forget about all the runs of the algorithm which start anywhere else than at distance n/k from α^* . The conclusion is that in a deterministic variant, we should use a (covering) code which is small enough to cycle through all codewords, still guaranteeing that one assignment from the code is at the required distance (or even closer).

Let us conduct a similar study of the number of steps being made. Consider the cover $E_{n/k} = \bigcup_{0 \leq i, j \leq 3n, i-j=n/k} H_{i,j}$, where $H_{i,j}$ is the event that

$E_{n/k}$ occurs *and* within the first $i+j$ steps, i steps go to the left and j steps go to the right in M_n . Since $E_{n/k}$ itself is typical, using Observation 7*.11, at least one of the $H_{i,j}$ with $0 \leq i, j \leq 3n, i+j = n/k$ is typical. We claim that H_{i_0, j_0} is typical for example if $i_0 = \frac{k-1}{k(k-2)} \cdot n$ and $j_0 = \frac{1}{k(k-2)} \cdot n$, i.e. that in a typical successful run of Schöning, we are making exactly i_0 'good' flips which take us closer to α^* and exactly j_0 'bad' flips which take us further away. To see this, we again directly compute the probability of $H_{i_0, j_0} \wedge (\exists i \leq 3n : D_i = 0)$. A beautiful thing to note here is that H_{i_0, j_0} *implies* success of the algorithm because then, $D_{i+j} = \frac{n}{k} - i_0 + j_0 = 0$ where $i+j \leq 3n$. So it suffices to calculate the probability of H_{i_0, j_0} alone. For this we obtain

$$\begin{aligned} \Pr(H_{i_0, j_0}) &= \Pr(E_{n/k} \wedge (i_0 \text{ flips to the left}) \wedge (j_0 \text{ flips to the right})) = \\ &= \frac{\binom{n}{n/k}}{2^n} \cdot \binom{i_0 + j_0}{i_0} \cdot \left(\frac{1}{k}\right)^{i_0} \left(\frac{k-1}{k}\right)^{j_0}. \end{aligned}$$

Noting that $i_0 + j_0 = \frac{n}{k-2}$ and using Lemma 5.6 again, this yields

$$\begin{aligned} \Pr(H_{i_0, j_0}) &= \frac{2^{nH(\frac{1}{k})+o(n)}}{2^n} \cdot 2^{\frac{n}{k-2}H(\frac{1}{k})+o(n)} \cdot \left(\frac{1}{k}\right)^{\frac{k-1}{k(k-2)}} \left(\frac{k-1}{k}\right)^{\frac{1}{k(k-2)}} = \\ &= \left(2 \left(\frac{1}{k}\right)^{\frac{1}{k} + \frac{1}{k(k-2)}} \left(\frac{k-1}{k}\right)^{\frac{k-1}{k} + \frac{k-1}{k(k-2)}} \left(\frac{1}{k}\right)^{\frac{k-1}{k(k-2)}} \left(\frac{k-1}{k}\right)^{\frac{1}{k(k-2)}}\right)^{-n+o(n)} = \\ &= \left(2 \frac{k-1}{k}\right)^{-n+o(n)}. \end{aligned}$$

Not only have we proved this way that H_{i_0, j_0} is typical; since H_{i_0, j_0} implies success of the algorithm, the last line is also a complete (!) proof for the success probability of Schöning's algorithm, so as long as you are happy with losing the small $o(n)$ factor in the exponent, you can supplant the entire calculation in Chapter 7 with these two lines. This shows that in a typical successful run of Schöning, the initial assignment has distance $\frac{n}{k}$ from α^* and then the algorithm makes exactly $\frac{n}{k-2}$ steps, out of which $\frac{1}{k}$ -fraction goes to the right and the rest goes to the left. Even if we consider all runs of the algorithm not having these characteristics 'failures',

the success probability stays the same for the relevant part.

For a last example, let $t \in \omega(1) \cap o(n)$ be any slowly growing function. Within H_{i_0, j_0} , consider the set of events $\{E_{i_1, i_2, \dots, i_t}\}_{0 \leq i_1, i_2, \dots, i_t \leq t}$, where we split the first $N := \frac{n}{k-2}$ steps which M_n carries out into N/t phases of t steps each and E_{i_1, i_2, \dots, i_t} denote the event that for all $1 \leq j \leq N/t$, there are exactly i_j steps in phase j which go to the left and $t - i_j$ steps in phase j which go to the right. Then, since the total number of events in the set is $t^{N/t}$ which is $2^{o(n)}$, Observation 7*.11 tells us that there exist i_1, \dots, i_t such that E_{i_1, \dots, i_t} is typical. And in fact, using the very same argument as in the previous example phase by phase, we can easily prove that E_{i_1, \dots, i_t} is typical if $i_j = \frac{k-1}{k}t$ for all j .

That is: in a typical successful run of Schöning,

- α_0 has distance $\frac{n}{k}$ from α^*
- the algorithm makes a total of $N := \frac{n}{k-2}$ flips
- if we group the flips into N/t phases of t flips each, then in each such phase, we go $\frac{t}{k}$ steps to the right and $\frac{(k-1)t}{k}$ steps go to the left, finally arriving at state zero.

Now that you *understand* which are the relevant executions of Schöning's algorithm, you know what you must look for in a deterministic variant: find (using a covering code) a starting assignment at distance at most $\frac{n}{k}$, then for some slowly growing function t , make phases of t flips each and in each phase, make sure (using a covering code for the flips) that you decrease your distance from α^* by at least $\frac{k-2}{k}t$. Once you also realise that for applying covering codes to flips you must make sure that the t flips in one phase are independent from one another, you have all the inspiration necessary and the rest is a matter of transpiration.

NOTE 7*.1 The algorithm $\text{cov}()$ has discovered by Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan and Schöning [DGH⁺02]. Since then, several papers have improved $\text{cov}()$, mainly focusing on 3-SAT. The running time has been improved to $O(1.481^n)$ by Dantsin et al. [DGH⁺02], $O(1.473^n)$ by Brueggemann and Kern [BK04], $O(1.465^n)$ by Scheder [Sch08], $O(1.439^n)$ by Scheder and Kutzkov [KS10]. The algorithm presented in this chapter is due to Moser and Scheder [MS10]

Bibliography

- [BK04] Tobias Brueggemann and Walter Kern. An improved deterministic local search algorithm for 3-sat. *Theor. Comput. Sci.*, 329(1-3):303–313, 2004.
- [DGH⁺02] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, O. Raghavan, and U. Schöning. A deterministic $(2 - 2/(k + 1))^n$ algorithm for k-SAT based on local search. In *Theoretical Computer Science 289*, pages 69–83, 2002.
- [KS10] Konstantin Kutzkov and Dominik Scheder. Using csp to improve deterministic 3-sat. *CoRR*, abs/1007.1166, 2010.
- [MS10] Robin A. Moser and Dominik Scheder. A full derandomization of schoening’s k-sat algorithm. *CoRR*, abs/1008.4067, 2010.
- [Sch08] Dominik Scheder. Guided search and a faster deterministic algorithm for 3-SAT. In *Proc. of the 8th Latin American Symposium on Theoretical Informatics (LATIN’08), Lecture Notes In Computer Science, Vol. 4957*, pages 60–71, 2008.

Chapter 8

The PPSZ Algorithm

Chapter by Timon Hertli, Robin Moser and Dominik Scheder.

In this chapter we present an improvement of the ppz algorithm. As ppz is called after its inventors, Paturi, Pudlák and Zane [PPZ99], the improved version, ppsz, is called after Paturi, Pudlák, Saks, and Zane [PPSZ05]. The idea behind the improvement is very beautiful and simple, the analysis however becomes quite involved (though still beautiful).

The basic idea of the ppz-algorithm is very simple: Given a CNF formula F , choose a random variable $x \in \text{vbl}(F)$. If $\{x\} \in F$ or $\{\bar{x}\} \in F$, set x the obvious way. Otherwise, set x randomly. Then iterate. Here is a simple idea for an improvement.

Instead of checking whether $\{x\} \in F$ or $\{\bar{x}\} \in F$, check whether x or \bar{x} can be derived from F in some way. Of course, checking this is again NP-complete and no big help. However, we can check whether x or \bar{x} follows from F in some simple, polynomial-time checkable way. Indeed, $\{x\}$ being a unit clause in F is probably the simplest way of x following from F .

Definition 8.1 *Let F be a CNF formula, let $D \in \mathbb{N}$, and let u be a literal. We say that F implies u , in writing $F \models u$, if any assignment α satisfying F also satisfies u . We say F D -implies u , in writing $F \models_D u$, if there is some $G \subseteq F$ with $|G| \leq D$ having $G \models u$.*

In this terminology, in the ppz-algorithm we check whether x is 1-

implied in F . The ppsz-algorithm applies an obvious improvement: Check whether x is D -implied for some (large) constant D . This can be done in polynomial time. To state the ppsz-algorithm formally, we think of it choosing a random permutation π of $\text{vbl}(F)$ and an assignment $\beta \in \{0, 1\}^n$ in advance and then processing the variables according to π and assigning them values according to β , unless they turn out to be D -implied.

F a (≤ 3) -CNF
formula over variables
 V , α_0 a partial
assignment over
variables from V
whose values we want
to fix in advance (by
default an empty
assignment), $D \in \mathbb{N}_0$.
POSTCONDITION:
returns a satisfying
assignment or
failure

```
function ppsz( $F, V, \alpha_0, D$ )
 $\pi \leftarrow$  a permutation of  $V \setminus \text{dom}(\alpha_0)$  chosen u.a.r.;
 $\beta \leftarrow$  an assignment from  $\{0, 1\}^{V \setminus \text{dom}(\alpha_0)}$  chosen u.a.r.;
return ppsz( $F, V, \alpha_0, \pi, \beta, D$ );
```

F a (≤ 3) -CNF
formula over V , α_0 a
partial assignment
over V ,
 $\pi = (x_1, x_2, \dots, x_j)$, a
permutation of
 $V \setminus \text{dom}(\alpha_0)$,
 $\beta \in \{0, 1\}^n$, $D \in \mathbb{N}_0$.
POSTCONDITION:
returns a satisfying
assignment or
failure

```
function ppsz( $F, V, \alpha_0, D, \pi, \beta$ )
 $\alpha \leftarrow \alpha_0$ ;
for  $i \leftarrow 1$  to  $|\pi|$  do
  if  $F^{[\alpha]} \models_D x_i$  then  $\alpha(x_i) \leftarrow 1$ ;
  elseif  $F^{[\alpha]} \models_D \bar{x}_i$  then  $\alpha(x_i) \leftarrow 0$ ;
  else  $\alpha(x_i) \leftarrow \beta(x_i)$ ;
if  $\alpha$  satisfies  $F$  then return  $\alpha$ ;
else return failure;
```

Note that in these algorithms, there is an additional parameter α_0 that represents a partial assignment over V allowing us to previously fix the

values of certain variables. You might ask yourself what this parameter is good for as we will start the algorithm with $\alpha_0 = \emptyset$ under normal circumstances. Once we go towards the end of the chapter, we will however see that the additional parameter gets very useful for the analysis. For this reason, we carry the parameter through the whole chapter even if right now, you can think of it as being an empty assignment. To simplify notation, let us write $U(\alpha_0) := V \setminus \text{dom}(\alpha_0)$ as well as $n(\alpha_0) := |U(\alpha_0)|$. Moreover, if α is a total assignment and α_0 a partial assignment, let us say that α_0 and α are *incompatible* if $\exists x \in V : \{\alpha(x), \alpha_0(x)\} = \{0, 1\}$, and *compatible*, otherwise.

When does this algorithm succeed in finding a satisfying assignment? To understand this, we need the concept of *forced* and *guessed* variables.

Definition 8.2 Let F be a CNF formula over n variables, $\alpha \in \{0, 1\}^n$ a satisfying assignment, $\pi = (x_1, \dots, x_{n(\alpha_0)})$ a permutation of $U(\alpha_0)$, and $D \geq 0$. A variable x_i is called *forced* (with respect to F , α_0 , α , π , and D) if $F^{\alpha_0 \cup \{x_1 \mapsto \alpha(x_1), \dots, x_{i-1} \mapsto \alpha(x_{i-1})\}} \vdash D$ implies x_i or \bar{x}_i . Otherwise the variable is called *guessed*. We denote the set of forced (guessed) variables within $U(\alpha_0)$ by $\text{Forced}(F, \alpha_0, \alpha, \pi, D)$ ($\text{Guessed}(F, \alpha_0, \alpha, \pi, D)$).

Observe the following.

Observation 8.3 Let F be a CNF formula and α be a satisfying assignment compatible with α_0 . Then $\text{ppsz}(F, \alpha_0, \pi, \beta, D)$ returns α if and only if $\alpha(x) = \beta(x)$ for all $x \in \text{Guessed}(F, \alpha_0, \alpha, \pi, D)$.

8.1 Having a Unique Satisfying Assignment

In this section we analyze the special case that F has a *unique* satisfying assignment α . This case is easier to analyze and at the same time we can introduce all the concepts we will need for the general case. For the entire chapter moreover, we restrict ourselves to (≤ 3) -CNF formulas. The case for $k > 3$ is totally analogous throughout the chapter, just some calculations of integrals later in the chapter become more complicated and we want to spare the reader this distraction. So for now, let F be a fixed (≤ 3) -CNF formula having a unique satisfying assignment.

By the above observation, ppsz returns α if and only if β agrees with α on all variables from $\text{Guessed}(F, \alpha_0, \alpha, \pi, D)$. This set is a random object, since it depends on π , which is a random permutation. We can write

$$\Pr_{\beta, \pi}[\text{ppsz returns } \alpha] = \mathbb{E}_{\pi} \left[2^{-|\text{Guessed}(F, \alpha_0, \alpha, \pi, D)|} \right]$$

This expression is rather difficult to work with. It would be easier to bound $\mathbb{E} [|\text{Guessed}(F, \alpha_0, \alpha, \pi, D)|]$, for example. Luckily, we have Jensen's inequality, which in this case states that for a random variable X , it holds that

$$\mathbb{E} [2^{-X}] \geq 2^{-\mathbb{E}[X]},$$

since the function $x \mapsto 2^{-x}$ is a convex function. With this, we obtain

$$\Pr_{\beta, \pi}[\text{ppsz returns } \alpha] \geq 2^{-\mathbb{E}_{\pi} [|\text{Guessed}(F, \alpha_0, \alpha, \pi, D)|]} \quad (8.1)$$

The right-hand side here is a much nicer expression, since by linearity of expectation, we have

$$\mathbb{E}_{\pi} [|\text{Guessed}(F, \alpha_0, \alpha, \pi, D)|] = \sum_{x \in U(\alpha_0)} \Pr[x \in \text{Guessed}(F, \alpha_0, \alpha, \pi, D)] . \quad (8.2)$$

And these probabilities we will be able to bound, although it will be quite some effort to do this.

Exercise 8.1 (Re-)Analyze the PPZ algorithm for the uniquely satisfiable case using the terminology we have introduced so far and the equations and inequalities just stated.

Exercise 8.2 Consider the 3-SAT formula F_n^- over variables $\{x_1, x_2, \dots, x_n\}$ defined as follows: F_n^- contains all 7 clauses over $\{x_1, x_2, x_3\}$ containing at least one positive literal as well as all clauses $\{\bar{x}_{i-2}, \bar{x}_{i-1}, x_i\}$ for $4 \leq i \leq n$. Prove that PPZ needs (expected) exponential time to solve this formula and that on the other hand PPSZ for $D = \log_2 n$ solves the formula in (expected) subexponential time.

HINT: You may want to use a Chernoff Bound at some point.

8.1.1 Building Critical Clause Trees

Fix some $x \in U(\alpha_0)$. We will now bound $\Pr[x \in \text{Guessed}(F, \alpha_0, \alpha, \pi, D)]$ from above. Please remember that we are still in the case of a uniquely satisfiable formula F , so there is exactly one $\alpha \in \text{sat}_V(F)$. We assume that α and α_0 are compatible, or, equivalently here, that $F^{[\alpha_0]}$ is satisfiable: Only this case is useful, and only in this case we can actually show something.

We now construct a collection of binary trees $\{T_x\}_{x \in U(\alpha_0)}$, each one called a *critical clause tree of x* , which in some way represents the multiple ways x can become D -implied during a run of the ppsz algorithm.

T_x is a rooted binary tree (where every node has at most two children) where every node $u \in V(T)$ is labelled both with a variable $x \in V$, which we denote by $\text{var-label}(u)$, and with a clause $C \in F^{[\alpha_0]}$, denoted by $\text{clause-label}(u)$. Even though there is some choice in how these trees are built, we make such choices arbitrarily once and for all and then consider the collection $\{T_x\}_{x \in U(\alpha_0)}$ to be fixed for the remainder of the section. Here is how T_x is built for a fixed $x \in U(\alpha_0)$:

1. Start with T_x consisting of a single root. This root has variable label x , and does not have a clause label yet.
2. As long as there is a leaf $u \in V(T)$ that does not yet have a clause label, do the following:
 - (a) Define $W := \{\text{var-label}(v) \mid v \in V(T) \text{ is an ancestor of } u \text{ in } T\}$, where *ancestor* includes u itself and the root.
 - (b) Define the assignment β as

$$\beta : U(\alpha_0) \rightarrow \{0, 1\}, \quad z \mapsto \begin{cases} 1 - \alpha(z) & \text{if } z \in W, \\ \alpha(z) & \text{otherwise.} \end{cases}$$

- (c) Let $C \in F^{[\alpha_0]}$ be a clause not satisfied by β . Since $\beta \neq \alpha$ and α is the unique satisfying assignment, such a clause exists. Set $\text{clause-label}(u) := C$.
- (d) For each literal $w \in C$ which has $\alpha(w) = 0$, create a new leaf, label it with the variable underlying w , and attach it to u as a child. The new leaf does not yet have a clause label.

See Figures 8.1– 8.5 for the illustration of an example.

We denote the resulting tree by T_x . Note that every clause in $F^{[\alpha]}$ has at most two literals violated by α , and thus in step (d) we append at most two children. Suppose v is an ancestor of u and $\text{var-label}(v) = y$. Since $\beta(y) \neq \alpha(y)$, there is no α -violated literal over y contained in $\text{clause-label}(u)$. Therefore,

Observation 8.4 *In T_x , no node has the same var-label as one of its ancestors.*

This also implies that the height of the tree cannot exceed $n - 1$, thus the process terminates, making T_x well-defined.

8.1.2 Placements, Critical Clause Trees and Forced Variables

We now establish a connection between critical clause trees and forced variables useful for estimating the probability with which a variable is forced.

To simplify the calculations that await us, we introduce the notion of *placements*. A *placement* of the variables in $U(\alpha_0)$ is a function $\pi : U(\alpha_0) \rightarrow [0, 1]$. It is not coincidental that we overload the letter π with this object: from now on, we do not think about π as a permutation anymore, but rather as a placement, which is roughly the same. If the values $\pi(x)$ are chosen independently and uniformly at random from $[0, 1]$ for each $x \in U(\alpha_0)$, then with probability 1, π is injective, and by sorting the values $\pi(x)$ we obtain a uniformly distributed permutation of $U(\alpha_0)$.

Let $\gamma \in [0, 1]$ and T_x be the critical clause tree of some fixed variable. We call a node $u \in T_x$ *reachable at time γ w.r.t. π* if there exists a path $\text{root} = v_0, v_1, \dots, v_m = u$ in T_x such that $\pi(\text{var-label}(v_i)) \geq \gamma$ for all $1 \leq i \leq m$. Let us denote by $\text{Reachable}(T_x, \gamma, \pi)$ the set of all nodes in T reachable at time γ w.r.t. π .

Lemma 8.5 *If we have $|\text{Reachable}(T_x, \pi(x), \pi)| \leq D$, then it holds as well that $x \in \text{Forced}(F, \alpha_0, \alpha, \pi, D)$.*

Proof. Let α' be the restriction of α to the variables $y \in U(\alpha_0)$ with $\pi(y) < \pi(x)$. By definition, x is forced if there is a formula $F' \subseteq F^{[\alpha_0 \cup \alpha']}$



Figure 8.1: Consider the formula $F = \{\{x, \bar{y}, \bar{z}\}, \{x, \bar{v}, \bar{w}\}, \{x, y, v\}, \{x, z, \bar{a}\}, \dots\}$ and the unique satisfying assignment $\alpha = (1, 1, \dots, 1)$. We begin the construction of T_x with a tree consisting of a root. It is labeled with x , but does not yet have a clause label. We consider the assignment $\alpha[x \mapsto 0]$ and observe that this leaves $\{x, \bar{y}, \bar{z}\}$ unsatisfied. We attach two new leaves to the root, labeled with y and z , respectively.

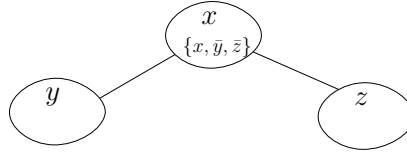


Figure 8.2: We take the leaf labeled with z . The assignment $\alpha[x, z \mapsto 0]$ leaves the clause $\{x, z, \bar{a}\}$ unsatisfied. Thus, we attach one new child to z and label it with a .

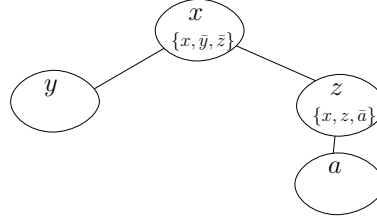


Figure 8.3: We take the leaf labeled with y . The assignment $\alpha[x, y \mapsto 0]$ leaves the clause $\{x, \bar{v}, \bar{w}\}$ unsatisfied. Thus, we attach two new children to y .

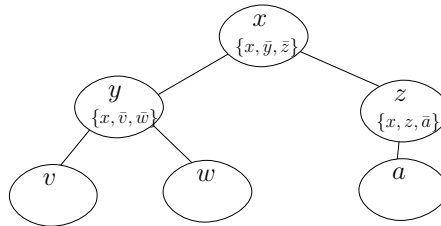


Figure 8.4: We pick the leftmost leaf. The assignment $\alpha[x, y, v \mapsto 0]$ leaves the clause $\{x, y, v\}$ unsatisfied. We do not attach further leaves here.

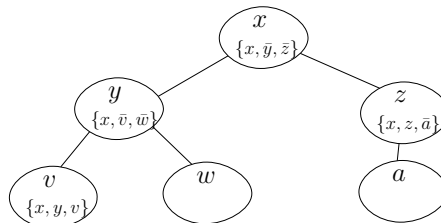


Figure 8.5: The tree now has 6 nodes. Two leaves are still do not have a clause label, thus the process is not finished yet.

with $|F'| \leq D$ that implies x . Let $G := \text{clause-label}(\text{Reachable}(T_x, \pi(x), \pi))$, i.e., the subformula of F consisting of all clause labels of reachable nodes in T_x . Clearly $|G| \leq D$. We claim that $G^{\alpha_0 \cup \alpha'}$ implies x .

Suppose, for the sake of contradiction, that $G^{\alpha_0 \cup \alpha'}$ does not imply x or \bar{x} . Then we can fix an assignment $\beta : V \rightarrow \{0, 1\}$ which is compatible with $\alpha_0 \cup \alpha'$, which has $\beta(x) \neq \alpha(x)$ and which satisfies G . Choose a maximal path in T_x starting at the root and containing only nodes v such that $\beta(\text{var-label}(v)) \neq \alpha(\text{var-label}(v))$. Since $\beta(x) \neq \alpha(x)$, this path is non-empty. Let u be its endpoint. The node u is reachable as β is compatible with α on all variables ranking before x in π . All children of u are labeled with some variable z for which $\beta(z) = \alpha(z)$, and all ancestors with some variable y for which $\beta(y) \neq \alpha(y)$. One checks that by the definition of T_x , $\text{clause-label}(u)$ is unsatisfied by β , a contradiction. \square

The lemma tells us how to proceed with bounding the probability that a variable is forced. Namely, it follows immediately that over uniform choice of π , we have

$$\Pr[x \in \text{Forced}(F, \alpha_0, \alpha, \pi, D)] \geq \Pr[|\text{Reachable}(T_x, \pi(x), \pi)| \leq D]. \quad (8.3)$$

This reduces the problem to a probabilistic calculation on binary trees: what is the probability that when sorting the nodes of a fixed binary tree according to a random permutation (caveat: some nodes have the same var-label and are prescribed to get assigned the same rank) and deleting all nodes which rank after the root, there will be at most D nodes reachable? The answer is nontrivial unfortunately, and we will have to devote the whole of Section 8.2 to this problem. There, we will establish the following.

Define

$$S := \frac{1}{2} - \int_0^{\frac{1}{2}} \frac{p^2}{(1-p)^2} dp = 2 \ln 2 - 1 \leq 0.3863.$$

Theorem 8.6 *For any $\epsilon > 0$ there exists a $D_\epsilon \in \mathbf{N}$ depending only on ϵ such that the following holds. Let $X_1, X_2, \dots, X_r \in [0, 1]$ be real random variables distributed uniformly and mutually independently. Let T be any finite binary tree and $\sigma : V(T) \rightarrow \{1 \dots r\}$ a labelling of the nodes of T such that on each path from the root to a leaf of T , σ is injective. Consider the experiment of drawing X_1, X_2, \dots, X_r according to their distribution and then deleting all nodes u from T (along with the corresponding subtrees) for which $X_{\sigma(u)} < X_{\sigma(\text{root})}$. Then for the*

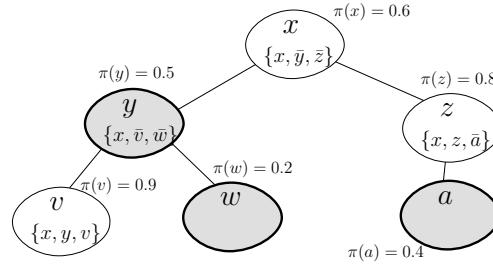


Figure 8.6: Illustration of Lemma 8.5. Nodes u with $\pi(u) < \pi(x)$ are shown gray. Two nodes are reachable, those var-labeled with x and z . Their clause labels yield the subformula $\{\{x, \bar{y}, \bar{z}\}, \{x, z, \bar{a}\}\}$. This formula implies the clause $\{x, \bar{y}, \bar{a}\}$ (which can be seen by resolution, for example). Therefore, every assignment satisfying this formula and setting all gray variables to 1 must set x to 1.

probability that the resultant tree T' contains more than D_ϵ nodes we have

$$\Pr(|V(T')| > D_\epsilon) \leq S + \epsilon.$$

For later use, let us also introduce the notation

$$S_D := \sup_T (\Pr(|V(T')| > D)),$$

where the supremum is over all choices of finite trees with labels T (as in the theorem) and T' is the random tree arising from T by conducting the experiment described in the theorem. In this language, the theorem states that $\lim_{D \rightarrow \infty} S_D \leq S$, where the limit exists because S_D is trivially monotonic and bounded.

Before we go on to prove this, let us finish the algorithmic part and see how we can use the result for estimating the success probability of ppsz.

We were interested in the expected number of variables that need to be guessed. If we consider $\alpha_0 = \emptyset$ in the input, then combining (8.2) with (8.3), we get

$$\begin{aligned} & E_\pi [|\text{Guessed}(F, \alpha_0, \alpha, \pi, D)|] = \\ & \leq n - \sum_{x \in U(\alpha_0)} \Pr[|\text{Reachable}(T_x, \pi(x), \pi)| \leq D]. \end{aligned}$$

Using Theorem 8.6, this yields

$$E_{\pi} [\|\text{Guessed}(F, \alpha_0, \alpha, \pi, D)\|] \leq S_D \cdot n,$$

where $S_D \rightarrow S$ for $D \rightarrow \infty$. This means: on the average, only 38.63% of the variables need to be guessed, while the remaining 61.37% of variables are forced. The ratio of guessed variables is higher if D is small, but the larger we make D , the closer we get to this ratio. In particular if $D \rightarrow \infty$, then we converge towards this number. We can easily achieve this by selecting D to be some function that grows very slowly in n . For example, if $D = \log_2 n$, then examining all $G \subseteq F$ with $|G| \leq D$ and checking for D -implications is still doable in subexponential time. If we plug this result into (8.1), we obtain the final theorem.

Theorem 8.7 *For any (≤ 3) -CNF formula F on n variables V which has a unique satisfying assignment, $\text{ppsz}(F, V, \emptyset, \log_2 n)$ returns this assignment with probability $\Omega(1.3071^{-n})$.*

There are two things left to do in this chapter: firstly, we have to prove Theorem 8.6 on which this result heavily depends. We will do this in Section 8.2. And secondly, we would of course like to remove the assumption that F has a unique satisfying assignment, this will be the topic in Section 8.3.

8.2 Random Deletion in Binary Trees

This section is aimed at proving Theorem 8.6. Note that this theorem speaks purely about probabilities in a combinatorial random experiment on binary trees some of whose vertices get deleted. There is no reference to the ppsz algorithm, so this is a self-contained proof which you can understand independently. However, the proof is non-trivial and we will first have to recall to memory some basic tools which we know from discrete probability theory and analysis.

8.2.1 Monotone Convergence

A fundamental theorem from probability and measure theory that you might already know is the *theorem of monotone convergence*. The proof

is more involved if you need it for functions in continuous space. Luckily, a largely simplified version for events in a discrete space which will be sufficient for our purposes.

Theorem 8.8 (Monotone Convergence Theorem) *Let, in a probability space, B_1, B_2, \dots be an infinite sequence of events such that $B_i \supseteq B_{i+1}$ for all $i \geq 1$. Then we have*

$$\lim_{n \rightarrow \infty} \Pr(B_n) = \Pr\left(\bigcap_{i=1}^{\infty} B_i\right).$$

Proof. We write

$$\bigcap_{i=1}^{\infty} B_i = B_1 \setminus \bigcup_{i=1}^{\infty} (B_i \setminus B_{i+1}).$$

Since the union is disjoint and moreover contained in B_1 , this implies¹

$$\Pr\left(\bigcap_{i=1}^{\infty} B_i\right) = \Pr(B_1) - \sum_{i=1}^{\infty} \Pr(B_i \setminus B_{i+1}).$$

By definition, the infinite sum on the right hand side is the limit of sums truncated after the first n summands and since $\Pr(B_1)$ does not depend on this truncation, we can write

$$\Pr\left(\bigcap_{i=1}^{\infty} B_i\right) = \lim_{n \rightarrow \infty} \left(\Pr(B_1) - \sum_{i=1}^n \Pr(B_i \setminus B_{i+1}) \right).$$

But as we easily check, the expression inside the limit is $\Pr(B_{n+1})$, establishing the claim. \square

8.2.2 Numerical Integration and Riemann Sums

In our calculations, we will use integrals in order to average probabilities over a continuum of cases. To circumvent the discussion of tricky questions

¹note that what we are using here is the definition of probability spaces: the probability of a countable disjoint union of events is the infinite sum of the corresponding probabilities

of bounded and monotone convergence and the exchange of limits with integration which are better discussed in an analysis course, we will use the following simple result for the estimation of integrals using Riemann sums. All functions that we will encounter in this chapter are well-behaved (continuous, bounded and most often monotonic) and therefore trivially integrable in the Riemann sense.

Lemma 8.9 *Let $\varphi : [0, 1] \rightarrow [0, 1]$ be a continuous and monotonically non-decreasing function. Then for any $N \geq 1$,*

$$\frac{1}{N} \sum_{i=0}^{N-1} \varphi\left(\frac{i}{N}\right) \leq \int_0^1 \varphi(x) dx \leq \frac{1}{N} \sum_{i=0}^{N-1} \varphi\left(\frac{i}{N}\right) + \frac{1}{N}.$$

Proof. Since φ is monotonically increasing, we have for all $0 \leq x_0 < x < x_1 \leq 1$ that $\varphi(x_0) \leq \varphi(x) \leq \varphi(x_1)$. In particular, if $x_0 = i/N$ and $x_1 = (i+1)/N$ for some $i \in \{0..N-1\}$, then

$$\frac{1}{N} \varphi(x_0) \leq \int_{x_0}^{x_1} \varphi(x) dx \leq \frac{1}{N} \varphi(x_1) = \frac{1}{N} \varphi(x_0) + \frac{1}{N} [\varphi(x_1) - \varphi(x_0)].$$

By summing over all i , we obtain

$$\begin{aligned} \frac{1}{N} \sum_{i=0}^{N-1} \varphi\left(\frac{i}{N}\right) &\leq \int_0^1 \varphi(x) dx \leq \\ &\leq \frac{1}{N} \sum_{i=0}^{N-1} \varphi\left(\frac{i}{N}\right) + \frac{1}{N} \sum_{i=0}^{N-1} \left[\varphi\left(\frac{i+1}{N}\right) - \varphi\left(\frac{i}{N}\right) \right]. \end{aligned}$$

Noting that the last term is a telescopic sum yielding $\varphi(1) - \varphi(0) \leq 1$ completes the proof. \square

8.2.3 The FKG Inequality

A quite powerful tool we will apply to bound the probability we are interested in is the famous *FKG correlation inequality* due to Fortuin, Kasteleyn and Ginibre. It basically states that monotone events are positively correlated.

Let $\mathcal{A} = \{A_1, A_2, \dots, A_r\}$ be a collection of independent binary random variables. An event E is said to be *determined* by \mathcal{A} , if there exists a fixed

list $S_E \subseteq 2^{\mathcal{A}}$ such that $E = \{A \in \mathcal{A} \mid A = 1\} \in S_E$, or, informally speaking, if knowing the values of \mathcal{A} leads to knowing whether E occurs. Moreover, if S_E is upwards hereditary, i.e. if

$$\forall \mathcal{A} \supseteq \mathcal{U} \supseteq \mathcal{V} : \mathcal{V} \in S_E \Rightarrow \mathcal{U} \in S_E,$$

then E is called *monotonically increasing* in \mathcal{A} .

Theorem 8.10 FKG Inequality *Let $\mathcal{A} = \{A_1, A_2, \dots, A_r\}$ be a collection of mutually independent binary random variables and E_1 and E_2 events which are determined by \mathcal{A} and monotonically increasing in \mathcal{A} . Then*

$$\Pr(E_1 \wedge E_2) \geq \Pr(E_1) \cdot \Pr(E_2).$$

Proof. We proceed by induction on r .

For a simple base case, let $r = 1$. Then there are only two non-empty monotonically increasing events determined by A_1 : either an event that occurs for both values of A_1 or an event that occurs only if $A_1 = 1$. Now if $E_1 = E_2$ the statement is trivial, so the only thing we have to check is if E_1 is the first of these cases and E_2 is the other. If p is the probability that $A_1 = 1$, then the probability that both events occur is p , the probability that E_1 occurs is 1 and the probability that E_2 occurs is p , establishing the claim.

For the induction step, let us assume that the statement holds for smaller values of r and let p be the probability that $A_1 = 1$. We can rewrite the right hand side of our claim using the law of total probability as

$$\begin{aligned} \Pr(E_1) \cdot \Pr(E_2) &= \\ &= (p \underbrace{\Pr(E_1 \mid A_1 = 1)}_{=: e_{11}} + (1-p) \underbrace{\Pr(E_1 \mid A_1 = 0)}_{=: e_{10}}) \cdot \\ &\cdot (p \underbrace{\Pr(E_2 \mid A_1 = 1)}_{=: e_{21}} + (1-p) \underbrace{\Pr(E_2 \mid A_1 = 0)}_{=: e_{20}}) = \\ &= p^2 e_{11} e_{21} + p(1-p)(e_{11} e_{20} + e_{10} e_{21}) + (1-p)^2 e_{10} e_{20}. \end{aligned}$$

Now note that the mutual independence of \mathcal{A} together with the monotonicity of both E_1 and E_2 in A_1 implies that both events can only be more likely in the conditional space determined by $\{A_1 = 1\}$ than in the case $\{A_1 = 0\}$, thus we get $e_{11} \geq e_{10}$ and $e_{21} \geq e_{20}$. We can use this to estimate the mixed term in our expansion: consider $e_{11} \geq e_{10}$ to be weights and the mixed term to be a weighted sum of $e_{21} \geq e_{20}$. Currently, the larger weight accompanies the smaller summand. Thus if we exchange the weights so that the larger summand gets the larger weight, the weighted sum can only increase. This yields

$$\Pr(E_1) \cdot \Pr(E_2) \leq p^2 e_{11} e_{21} + p(1-p)(e_{10} e_{20} + e_{11} e_{21}) + (1-p)^2 e_{10} e_{20}$$

which after some term recollection becomes

$$\Pr(E_1) \cdot \Pr(E_2) \leq p e_{11} e_{21} + (1-p) e_{10} e_{20}. \quad (8.4)$$

It is now time to invoke the induction hypothesis. We have assumed that the statement holds for smaller r . The events E_1 and E_2 are, *in the conditional space of $\{A_1 = 1\}$* , determined by A_2, A_3, \dots, A_r and monotonically increasing in these variables. Therefore, in the conditional space of $\{A_1 = 1\}$, the FKG inequality holds for E_1 and E_2 by the induction hypothesis, yielding that

$$e_{11} e_{21} \leq \Pr(E_1 \wedge E_2 \mid A_1 = 1). \quad (8.5)$$

Analogously

$$e_{10} e_{20} \leq \Pr(E_1 \wedge E_2 \mid A_1 = 0). \quad (8.6)$$

Plugging in (8.5) and (8.6) into (8.4) and applying once more the law of total probability, we can complete the induction step. \square

8.2.4 Of Infinite and Finite Trees

So far the preliminaries. For finally proving Theorem 8.6, we will proceed in four high-level steps. First, we consider a much simpler case: a totally symmetric infinite full binary tree from which every node is deleted independently with probability p . Next, we proceed to showing that if our tree is not infinite but finite, this does not change too much. Then we go on to show that introducing dependencies between the nodes can only help our case. Finally, we translate these results to the case where the root's rank is also random.

So to set out, let us prove the following simpler variant. Define for all $p \in [0, 1]$,

$$\zeta(p) := \begin{cases} \frac{p^2}{(1-p)^2} & \text{if } p < \frac{1}{2}, \\ 1 & \text{otherwise.} \end{cases}$$

It is important to note that ζ is continuous on $[0, 1]$ and monotonically non-decreasing. Now we claim the following.

Lemma 8.11 *Let T_∞ be the infinite rooted full binary tree. Consider the following random experiment: each non-root node from T_∞ is deleted (along with its subtree) independently from all other nodes with probability p . Then the probability that the resultant tree T' is of finite size is*

$$\Pr(T' \text{ finite}) \geq \zeta(p).$$

Proof. For the probability $q := \Pr(T' \text{ finite})$, since the two subtrees of the root of T are again infinite rooted full binary trees which we subject to the same random experiment, we know that the relation

$$q = (p + (1-p) \cdot q)^2$$

must hold. As you can easily verify, this quadratic equation has the two solutions

$$q = 1 \text{ and } q = \frac{p^2}{(1-p)^2}.$$

This proves that q is always at least the smaller of the two solutions, which readily establishes the claim. \square

If T is any (finite or infinite) binary tree, let us denote by $h(T)$ the height of T , i.e. the length of the longest path from the root downwards. If T is infinite, then $h(T) = \infty$. We can extend the previous lemma as follows.

Lemma 8.12 *Let T_∞ be the infinite rooted full binary tree. Consider the following random experiment: each non-root node from T_∞ is deleted (along with its subtree) independently from all other nodes with probability p . Then the probability that the resultant tree T' has height at most $d \geq 1$ converges as*

$$\lim_{d \rightarrow \infty} \Pr(h(T') \leq d) = \Pr(T' \text{ finite}).$$

Proof. Let B_i be the event that there exists a path from the root to some node at depth i . We first claim that

$$\bigcap_{i \geq 1} B_i = \{T' \text{ infinite}\}.$$

Suppose that an outcome is contained in the left hand side. This means that T' contains finite paths of arbitrary length. Of course, by consequence, T' cannot be finite. The other direction is trivial.

From Theorem 8.8, it now follows that

$$\lim_{i \rightarrow \infty} \Pr(B_i) = \Pr(T' \text{ infinite}).$$

Taking complements,

$$\lim_{d \rightarrow \infty} \Pr(h(T') \leq d) = \lim_{d \rightarrow \infty} \bar{B}_d = 1 - \lim_{d \rightarrow \infty} B_d = 1 - \Pr(T' \text{ infinite}),$$

the claim readily follows. \square

The two last lemmas can now be combined to obtain a result on finite trees T .

Lemma 8.13 *Let $p \in [0, 1]$ be a fixed number. There exists a sequence $\epsilon_1(p), \epsilon_2(p), \dots \in \mathbf{R}^+$ of numbers depending only on p , having $\epsilon_d(p) \rightarrow 0$ for $d \rightarrow \infty$ such that the following holds. Let T be any finite (and not necessarily full) binary tree. Consider the following random experiment: each non-root node from T is deleted (along with its subtree) independently from all other nodes with probability p . Then the probability that the resultant tree T' has height at most $d \geq 1$ satisfies*

$$\Pr(h(T') \leq d) \geq \zeta(p) - \epsilon_d(p).$$

Please note an important subtlety in this statement: instead of using a limit in the resulting inequality, we have introduced the sequence of errors $\epsilon_i(p)$. This is crucial in order to stress that the rate with which the probability converges to the numbers given is independent of which tree T we are considering (“uniform convergence”). On the other hand, this rate must depend on p , as anything stronger would not be provable from the statements we made above.

Proof. The random experiment we are conducting on T can be coupled to a random experiment conducted on T_∞ in the obvious way: T is embeddable

into T_∞ with the two roots coinciding and then if we delete every node from T_∞ independently with probability p , the same experiment is taking place on T . Note that for the tree T'' resulting from the deletions in T_∞ and the tree T' resulting from the deletions in T , since T' is a subtree of T'' we have $h(T') \leq h(T'')$ and therefore $\Pr(h(T') \leq d) \geq \Pr(h(T'') \leq d)$. Define, for all $d \geq 1$,

$$\epsilon_d(p) := \max\{\zeta(p) - \Pr(h(T'') \leq d), 0\}.$$

Then we find that

$$\Pr(h(T') \leq d) \geq \Pr(h(T'') \leq d) \geq \zeta(p) - \epsilon_d(p)$$

and from the previous lemma,

$$\lim_{d \rightarrow \infty} \epsilon_d(p) = 0,$$

as required. \square

8.2.5 Of Independent and Dependent Labels

What we have obtained so far does not yet totally reflect what we are after: in the experiments studied so far, each node was deleted independently of all other nodes. This is not the case in the result we are eventually after, there dependencies can occur of the sort that nodes of the tree are linked to one another and if one of these linked vertices is deleted, all of them are deleted. We now generalize Lemma 8.13 to obtain the following.

Lemma 8.14 *Let $Z_1, Z_2, \dots, Z_r \in \{0, 1\}$ be mutually independent binary random variables, each of which takes value one with probability p . Let T be any finite (and not necessarily full) binary tree with a labelling $\sigma: V(T) \setminus \{\text{root}\} \rightarrow \{1 \dots r\}$ of the non-root nodes of T with indices with the property that on each path from the root to a leaf, σ is injective. Consider the experiment of drawing Z_1, \dots, Z_r according to their distribution and then deleting all nodes u from T (along with their subtrees) for which $Z_{\sigma(u)} = 1$. Call the resulting tree T' .*

Juxtapose the experiment where in T , every non-root node is deleted independently from all other nodes with probability p . Call the random tree resulting from this experiment T'' . Then for any d ,

$$\Pr(h(T') \leq d) \geq \Pr(h(T'') \leq d).$$

Proof. The statement is trivial if σ is globally injective. We now use the FKG inequality to demonstrate that correlations arising from duplicate labels cannot decrease this probability if none of these duplicates are in an ancestor-descendant relation.

To achieve this, we proceed by induction on d . For $d = 0$ the statement is trivial.

For the induction step, let $d > 0$. If the root of T has no child, the statement is trivial. If the root of T has only one child, then the statement is a direct consequence of the induction hypothesis, as the whole tree has height d iff the unique subtree of the root has height $d - 1$. The only interesting case arises if the root of T has two children u and v .

In this case, let us look at each of the root's subtrees separately first. Let T_u be the subtree rooted at u and let $Z_i = \sigma(u)$. Let T'_u denote the subtree of T' rooted at u and T''_u the subtree of T'' rooted at u (empty trees if u is deleted). Let T_v, Z_j, T'_v, T''_v be the corresponding objects for v . The hypothesis on σ entails that no other node in T_u is labelled with Z_i , so whatever happens in the non-root nodes of T_u is independent of whether u itself is being deleted or not. The same holds at v . Let

$$E_1 := \{Z_i = 1 \vee (Z_i = 0 \wedge h(T'_u) \leq d - 1)\}$$

as well as

$$E_2 := \{Z_j = 1 \vee (Z_j = 0 \wedge h(T'_v) \leq d - 1)\}.$$

We have that

$$\{h(T') \leq d\} = E_1 \wedge E_2.$$

Moreover, E_1 and E_2 are events which are determined by $\{Z_1, \dots, Z_r\}$ and, as you can easily check, monotonically increasing in those events. Therefore we can use the FKG Inequality as in Theorem 8.10 to obtain that

$$\Pr(h(T') \leq d) = \Pr(E_1 \wedge E_2) \geq \Pr(E_1) \cdot \Pr(E_2).$$

For these separate events, we have, due to the independence of T_u from Z_i and T_v from Z_j ,

$$\Pr(E_1) = p + (1 - p) \Pr(h(T'_u) \leq d - 1) \geq p + (1 - p) \Pr(h(T''_u) \leq d - 1)$$

and

$$\Pr(E_2) = p + (1 - p) \Pr(h(T'_v) \leq d - 1) \geq p + (1 - p) \Pr(h(T''_v) \leq d - 1),$$

where we have used the induction hypothesis for the inequalities. Combining the last three inequalities, we obtain

$$\begin{aligned} \Pr(h(T') \leq d) &\geq \\ &\geq (p + (1 - p) \Pr(h(T''_u) \leq d - 1)) \cdot (p + (1 - p) \Pr(h(T''_v) \leq d - 1)) = \\ &= \Pr(h(T'') \leq d), \end{aligned}$$

as claimed. \square

Moreover, we will need the following simple observation for technical reasons.

Observation 8.15 *In the setting of Lemma 8.14, $\Pr(h(T') \leq d)$ as a function of p is continuous and monotonically non-decreasing.*

8.2.6 Integrating over the Rank of the Root

Using the previous lemma we can now finally approach the proof of Theorem 8.6. Fix $\epsilon > 0$.

An arbitrary tree T is labelled with real-valued random variables as $\sigma : V(T) \rightarrow \{X_1, X_2, \dots, X_r\}$ in such a way that σ is injective on any path from the root to a leaf of T . Now each X_i is drawn independently and uniformly from among $[0, 1]$. Then all nodes are deleted (along with their subtrees) which have a value smaller than that of the root, producing the resultant random tree T' .

Let $D \in \mathbf{N}$ be some number which we will fix appropriately later. To estimate the probability that T' has at most D vertices, we estimate the probability that T' has at most depth $d := \lfloor \log_2 D \rfloor$. Clearly, having depth at most d implies that we cannot have any more than D vertices. The specific relation between D and d is not relevant; the only thing we will need is that d grows (in any way) with D . In turn, to estimate the probability

of having height at most d , we will condition on the value $\sigma(\text{root})$ of the random variable at the root node and then use total probability.

Without loss of generality, suppose $\sigma(\text{root}) = X_r$. Note that this value is independent of all other values used because the root is part of all paths and σ is injective on each path, so X_r does not occur a second time as a label. Now once we condition on $X_r = \gamma$ for some fixed value $\gamma \in [0, 1]$, we can consider for $1 \leq i \leq r-1$ the binary random variables

$$Z_i := \begin{cases} 1 & \text{if } X_i < \gamma \\ 0 & \text{otherwise.} \end{cases}$$

The $\{Z_1, Z_2, \dots, Z_{r-1}\}$ are mutually independent binary random variables, each of which takes value 1 with probability exactly γ . This is the situation we have in Lemma 8.14 and so we can use this result to conclude that

$$\Pr(h(T') \leq d \mid X_r = \gamma) \geq \zeta(\gamma) - \epsilon_d(\gamma). \quad (8.7)$$

To convert this conditional into an unconditional probability, we invoke the law of total probability, which, since Z_r is uniformly distributed, reads

$$\Pr(h(T') \leq d) = \int_0^1 \Pr(h(T') \leq d \mid X_r = \gamma) d\gamma.$$

Of course, we would be happy to just plug in (8.7) to estimate the integrand, then take limits and be done. The problem is that although we have proved $\epsilon_d(\gamma)$ to exist and to converge to zero pointwise for all fixed γ , the dependency of $\epsilon_d(\gamma)$ on γ remains unknown and since the rate of convergence could be different for different γ , we are unable² to establish convergence of the whole integral.

What we can do instead however is making use of the monotonicity of $\Pr(h(T') \leq d \mid X_r = \gamma)$ which we have established in Observation 8.15. Since the integrand is monotonically non-decreasing in γ as well as continuous, we may, for any $N \in \mathbf{N}$, approximate the integral by a Riemann sum

²we could alternatively use the Dominated Convergence Theorem from measure theory and use the fact that the ϵ_d are uniformly bounded functions, but we prefer not to fall back on such sophisticated machinery here since measure theory is not usually taught to computer scientists

using Lemma 8.9 to obtain

$$\int_0^1 \Pr(h(T') \leq d \mid X_r = \gamma) d\gamma \geq \sum_{i=0}^{N-1} \frac{1}{N} \Pr\left(h(T') \leq d \mid X_r = \frac{i}{N}\right).$$

The resulting finite mesh resolution will enable us to establish the necessary convergence property. Plugging (8.7) into the Riemann sum representation, we obtain that

$$\Pr(h(T') \leq d) \geq \sum_{i=0}^{N-1} \frac{1}{N} \zeta\left(\frac{i}{N}\right) - \sum_{i=0}^{N-1} \frac{1}{N} \epsilon_d\left(\frac{i}{N}\right).$$

Now note that ζ is, as well, monotonically non-decreasing on $[0, 1]$. This yields that the first summand on the right hand side is an approximation to S which involves an error of at most $\frac{1}{N}$ according to Lemma 8.9, yielding that

$$\Pr(h(T') \leq d) \geq \int_0^1 \zeta(x) dx - \underbrace{\left(\frac{1}{N} + \sum_{i=0}^{N-1} \frac{1}{N} \epsilon_d\left(\frac{i}{N}\right)\right)}_{=: \varphi(N, d)}.$$

This holds for arbitrary N . Note the very important fact that the error term $\varphi(N, d)$ depends only on N and on d but not on the choice of T' . Therefore, given $\epsilon > 0$ as prescribed by the theorem, we can simply pick an $N = N(\epsilon)$ which is large enough such that $1/N < \epsilon/2$, and then pick a $d = d(\epsilon, N)$ large enough such that $\epsilon_d(x) < \epsilon/2$ for all of the N points x where the function is evaluated, yielding that $\varphi(N, d) < \epsilon$ and thus all possible trees have a probability of having at most height d of at least $S - \epsilon$. Setting $D_\epsilon > 2^{d(\epsilon, N(\epsilon))}$ concludes the proof of Theorem 8.6. \square

We have finally established a strong bound on the running time of ppsz on uniquely satisfiable (≤ 3)-CNF formulas. The computation was admittedly quite cumbersome. Here is an open exercise.

Exercise 8.3 (*Open challenge*) Simplify the calculations in this section.

Exercise 8.4 Let F be a ($\leq k$)-CNF formula F over variables V . Let us call a satisfying assignment $\alpha \in \text{sat}_V(F)$ r -superisolated if for every variable $x \in V$, $F^{[x \mapsto (1-\alpha(x))]}$ contains at least r independent clauses violated by α . Prove that PPZ has subexponential success probability on formulas that have a $(\log_2 n)$ -superisolated assignment.

Exercise 8.5 *Given a k -uniform hypergraph H over n vertices V , a proper k -coloring is a coloring $\gamma : V \rightarrow \{1..k\}$ of the vertices V with k colors such that no edge becomes monochromatic. We are interested in the computational problem of deciding whether a given hypergraph is properly k -colorable.*

Consider the following PPZ-like approach to the problem: process the vertices in a uniformly random ordering and choose for each vertex a uniformly random color among those colors that do not lead to closing a monochromatic edge. Assume that H admits an n -isolated coloring, i.e. a coloring for which, if you change any single color, the coloring is no longer proper. Determine a bound (you can give this bound as a formula still involving integrals, you need not evaluate them) depending on k for the success probability of this algorithm in finding a proper coloring.

Compare this approach to the method of random downsampling, where we randomly forbid $k-2$ of k colors at each vertex and then run Schöning's algorithm (or for $k=3$ PPSZ) on the resulting $(\leq k)$ -CNF formula. Which is the method of choice for which k ? You can do the comparison numerically; use a computer algebra program to evaluate your formula for a number of values until you see a tendency.

8.3 Multiple Satisfying Assignments

So far in the chapter, we dealt with the case where F has a unique satisfying assignment. In general, we cannot rely on such a strong assumption. This makes the whole analysis considerably more complicated and for roughly ten years, it was open whether it could at all be carried out without a loss in terms of an inferior running time.

In fact, the original analysis by Paturi et al. [PPSZ05] attempts at circumventing the problem of multiple satisfying assignments by partitioning the solution space into disjoint subregions within which the formula may be considered 'uniquely satisfiable' and then averaging success probabilities over these regions. This results in an algorithm running in time $\mathcal{O}(1.364^n)$ for general 3-SAT, clearly inferior to the case of uniquely satisfiable formulas and also inferior to the running time of the very simple randomized

algorithm due to Schönig which we have studied in Chapter 7.

This loss in efficiency was disturbing and attracted a lot of research attention: how could it be that an algorithm was performing *worse* in the search for a satisfying assignment when there were *more* of these assignments around? A series of publications tried to alleviate the gap occurring ([IT04], [Rol06], [ISTT10], [Her10]), but each of them can be merely considered an attempt at *bypassing* the actual problem by adding preprocessing steps, parallelly executing Schönig's algorithm and other forms of added complexity. None of them succeeded in addressing the actual underlying issue: that an algorithm as natural as ppsz 'cannot possibly' behave as counterintuitively.

Finally, in a breakthrough result [Her11], Timon Hertli has been able to demonstrate what was long conjectured: that the ppsz algorithm does *not* become less successful if there are multiple satisfying assignments, resulting finally in the following currently best known performance for 3-SAT.

Theorem 8.16 *For any satisfiable (≤ 3)-CNF formula F over n variables $\text{ppsz}(F, V, \emptyset, \log_2 n, \pi, \beta)$ returns some satisfying assignment with probability $\Omega(1.3071^{-n})$.*

The present section is to prove this theorem.

8.3.1 Problem Assessment

Where does the analysis go wrong when we have multiple satisfying assignments?

Recall the tree construction process 175. Suppose we are assuming α to still be *a* (not anymore *the*) satisfying assignment and suppose we are trying to build a critical clause tree T_x for x . If the current intermediate tree T has a leaf u that does not yet have a clause label, it defines U to be the set of all variable labels of ancestors of u . Then it defines an assignment β via

$$\beta : \text{vbl}(F) \rightarrow \{0, 1\}, \quad z \mapsto \begin{cases} 1 - \alpha(z) & \text{if } z \in W, \\ \alpha(z) & \text{otherwise,} \end{cases}$$

chooses a clause $C \in F$ that is not satisfied by β and sets C as clause label of u . The problem is that if α is just *some* but not *the unique* satisfying assignment, β might as well satisfy F , and thus such a clause C need not exist and we are stuck.

Still, most of the arguments from the first section go through with a milder assumption than uniqueness of the satisfying assignment. Note that in the present case $\beta(x) \neq \alpha(x)$ according to definition, because $x \in W$ holds always in this tree. So in order to construct a full tree without getting stuck, we only require that there are no satisfying assignments β with $\beta(x) \neq \alpha(x)$, i.e. that all satisfying assignments give the same value to x .

Definition 8.17 *Let F be any (≤ 3) -CNF formula and $x \in \text{vbl}(F)$. x is said to be frozen in F if either $F \models x$ or $F \models \bar{x}$, that is all satisfying assignments of F send x to the same value.*

For any formula F , this partitions the variables into three categories

$$V_{\text{fo}}(F) \dot{\cup} V_{\text{fr}}(F) \dot{\cup} V_{\text{nf}}(F) = \text{vbl}(F),$$

where

- $V_{\text{fo}}(F) = \{x \in \text{vbl}(F) \mid F \models_D x \vee F \models_D \bar{x}\}$ are the variables that are *currently forced*, that is which are frozen *and* follow from F via D-implication,
- $V_{\text{fr}}(F) := \{x \in \text{vbl}(F) \mid F \models x \vee F \models \bar{x}, F \not\models_D x \wedge F \not\models_D \bar{x}\}$ are the variables that are frozen (but not forced) and
- $V_{\text{nf}}(F) := \{x \in \text{vbl}(F) \mid F \not\models x \vee F \not\models \bar{x}\}$ are the remaining variables that are non-frozen, i.e. those which you can assign either way keeping the formula satisfiable.

The arguments of the last section show that at least if a variable is frozen, the probability that it needs to be guessed during a run of ppsz is bounded.

Lemma 8.18 *Let F be a (≤ 3) -CNF formula and $x \in V_{\text{fo}}(F) \cup V_{\text{fr}}(F)$ be a variable that is frozen. Let furthermore α be any satisfying assignment. Then $\Pr(x \in \text{Guessed}(F, \emptyset, \alpha, \pi, D)) \leq S_D$, with S_D defined as in Theorem 8.6.*

The proof of the lemma consists of reading through the entire first section again and just convincing yourself that none of the arguments we used there fail for the present case. This is left to you as an exercise.

One could be inclined to think that we are already done now. The non-frozen variables are totally harmless: we can assign them either way and stay satisfiable. And since the previous lemma, we now know that whenever a variable becomes frozen during execution, the probability that it needs to be guessed in the further execution of the algorithm is bounded by the same number it was bounded in the case of uniquely satisfiable formulas. Or not?

The big problem arising is that we were able to bound this probability only for a *fixed* satisfying assignment α . So as long as the algorithm chooses to assign values according to α , the probability that a frozen variable needs to be guessed is appropriately bounded. But there could be many satisfying assignments around and *which* of them the algorithm is going to steer towards is decided only while it executes and assigns values to the non-frozen variables.

In particular, the probability with which each of the assignments is being selected can depend heavily on the ordering in which we process variables. And although *each* satisfying assignment α has, according to the lemma, the property that frozen variables need to be guessed with probability no higher than S_D for a *uniformly* random ordering π , the probability of α at all becoming *relevant* for us depends heavily on π and so there could be nasty correlations causing those permutations π that make lots of variables forced for α to be unfortunately those permutations that make α very unlikely to become selected at all.

This effect has to be taken into account in our analysis which is the reason why there is still quite some work left to do. It will however turn out that there is a nice tradeoff between the potential 'badness' of the correlations occurring and the number of non-frozen variables we encounter during execution. The more non-frozen variables are present, the stronger the correlations can potentially get. But having lots of non-frozen variables is on the other hand very good for the algorithm as each guessing step then has a higher probability to preserve satisfiability. The remainder of the present section is to demonstrate that those two effects compensate

sufficiently nicely to obtain the same running time for the general case which we were able to prove in the uniquely satisfiable case.

8.3.2 Definition of a Cost Function

To do the calculation, we will associate a *cost* to each state of the algorithm as follows. Let us consider the formula F and its variables V to be fixed from here onwards so we can omit F and V in all definitions. An ‘intermediate state’ of the algorithm can be represented by a partial assignment α_0 over V containing all the values we have already selected in previous steps. This is where, finally, the additional parameter α_0 we have ignored so far comes handy.

Remember $U(\alpha_0) := V \setminus \text{dom}(\alpha_0)$ and $n(\alpha_0) := |U(\alpha_0)|$. To carry over the notions from the last subsection, we define for notational convenience

- $V_{\text{nf}}(\alpha_0) := V_{\text{nf}}(F^{[\alpha_0]})$,
- $V_{\text{fr}}(\alpha_0) := V_{\text{fr}}(F^{[\alpha_0]})$ and
- $V_{\text{fo}}(\alpha_0) := V_{\text{fo}}(F^{[\alpha_0]})$.

Now, we first define a per-assignment and per-variable cost as follows.

Definition 8.19 *Let α_0 be a partial and α be a total assignment and let $x \in V$ be any variable. We define the cost of x when completing α_0 to α , in writing $\text{cost}(\alpha_0, \alpha, x)$, as follows.*

- If $x \notin U(\alpha_0)$, then $\text{cost}(\alpha_0, \alpha, x) := 0$.
- If α_0 and α are incompatible, i.e. $\exists x : \{\alpha_0(x), \alpha(x)\} = \{0, 1\}$, then $\text{cost}(\alpha_0, \alpha, x) := 0$.
- If α does not satisfy F , then $\text{cost}(\alpha_0, \alpha, x) := 0$.
- Otherwise, if
 - $x \in V_{\text{fo}}(F^{[\alpha_0]})$, then $\text{cost}(\alpha_0, \alpha, x) := 0$.
 - $x \in V_{\text{fr}}(F^{[\alpha_0]})$, then

$$\text{cost}(\alpha_0, \alpha, x) := \Pr(x \in \text{Guessed}(F, \alpha_0, \alpha, \pi, D)).$$

– $x \in V_{\text{nf}}(F^{[\alpha_0]})$, then $\text{cost}(\alpha_0, \alpha, x) := S_D$.

Next, we define the *likelihood* of each assignment.

Definition 8.20 *Let α_0 be a partial and α a total assignment over V . The likelihood of completing α_0 to α , in writing $\text{lkhd}(\alpha_0, \alpha)$, is defined as the probability that $\text{ppsz}(F, V, \alpha_0, D)$ for $D = |F|$ outputs α .*

Note that the version of ppsz used in this definition is very inefficient because D is so large that the time needed for discovering D -implications is largely exponential. But as a definition used only in the analysis, it is just fine. Since D is as large as the whole formula, *every* implication is a D -implication and thus whenever a variable is frozen, it is automatically forced. The likelihood of α is thus in a certain sense the probability that ppsz selects α during execution, under the assumption that it never makes mistakes and that there is no gap between frozen and forced variables. For this reason, if α_0 is incompatible with α or if α does not satisfy F , then $\text{lkhd}(\alpha_0, \alpha) = 0$.

Finally, the cost of completing α_0 to *any* satisfying assignment arises from summing over all possible variables and all possible assignments and adding up individual per-variable per-assignment costs, weighted by the relevance, i.e. likelihood, of the corresponding assignment, i.e.

$$\text{cost}(\alpha_0, x) := \sum_{\alpha \in \{0,1\}^V} \underbrace{\text{lkhd}(\alpha_0, \alpha) \cdot \text{cost}(\alpha_0, \alpha, x)}_{=: \text{wcost}(\alpha_0, \alpha, x)}$$

and

$$\text{cost}(\alpha_0) := \sum_{x \in V} \text{cost}(\alpha_0, x).$$

8.3.3 Gathering a few Simple Facts

As with most new definitions, we first gather some basic and intuitive facts about the things we defined now. To begin with, consider the following.

Observation 8.21 *For any α_0, α, x , we have $\text{cost}(\alpha_0, \alpha, x) \leq S_D$. Furthermore $\text{cost}(\alpha_0) \leq S_D n(\alpha_0)$.*

Proof. The statement follows directly from the definition of the cost and Lemma 8.18. \square

In the end, we will show that the probability that PPSZ finds a satisfying assignment of F is $2^{-\text{cost}(\emptyset)}$, which, given the above observation, is exactly what we want.

Lemma 8.22 *Let α_0 and α be fixed. For any fixed variable $x \in \mathcal{U}(\alpha_0)$, if we set x according to α ,*

(i) *the likelihood of α can only increase, i.e.*

$$\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha) \geq \text{lkhd}(\alpha_0, \alpha)$$

with equality if x is frozen in $F^{[\alpha_0]}$, and

(ii) *the cost of a fixed variable $y \in V$ w.r.t. α can only decrease, i.e.*

$$\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y) \leq \text{cost}(\alpha_0, \alpha, y).$$

On the other hand, when choosing $x \in \mathcal{U}(\alpha_0)$ uniformly at random and setting it according to α ,

(iii) *the likelihood of α increases on average as*

$$\mathbb{E}(\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)) = \left(1 + \frac{|\mathcal{V}_{\text{nf}}(\alpha_0)|}{n(\alpha_0)}\right) \text{lkhd}(\alpha_0, \alpha),$$

whereas

(iv) *the cost of a fixed variable $y \in \mathcal{U}(\alpha_0)$ decreases on expectation as*

$$\mathbb{E}(\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)) \leq \text{cost}(\alpha_0, \alpha, y) - \frac{s}{n(\alpha_0)},$$

where

$$s = \begin{cases} 1 & \text{if } y \in \mathcal{V}_{\text{fr}}(\alpha_0) \\ s_D & \text{if } y \in \mathcal{V}_{\text{nf}}(\alpha_0) \\ 0 & \text{if } y \in \mathcal{V}_{\text{fo}}(\alpha_0). \end{cases}$$

Proof.

- (i) Assume for the moment that the permutation π is fixed. What is $\text{lkhd}(\alpha_0, \alpha)$? To output α , every non-frozen variable that ppsz encounters has to be set to the right value, which happens with probability $1/2$. If $D = |F|$, all frozen variables are automatically forced. Hence, for a fixed π , $\text{lkhd}(\alpha_0, \alpha)$ is $2^{-\text{nf}(\pi, \alpha_0, \alpha)}$, where $\text{nf}(\pi, \alpha_0, \alpha)$ denotes the number of non-frozen variables encountered. Therefore we have

$$\text{lkhd}(\alpha_0, \alpha) = \mathbb{E}_\pi[2^{-\text{nf}(\pi, \alpha_0, \alpha)}].$$

Moving x to the beginning in π can only decrease the number of non-frozen variables. Furthermore, if x is frozen in $F^{[\alpha_0]}$, the number of non-frozen variables remains the same.

Now observe that if we remove x from π , the resulting permutation π' has a uniform distribution from permutations over $\mathcal{U}(\alpha_0) \setminus \{x\}$. The number of non-frozen variables can only decrease in $\mathcal{U}(\alpha_0) \setminus \{x\}$; removal of x might decrease this even further. Therefore

$$\mathbb{E}_\pi[2^{-\text{nf}(\pi, \alpha_0, \alpha)}] \leq \mathbb{E}_{\pi'}[2^{-\text{nf}(\pi', \alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)}],$$

with equality if x is frozen, as in this case x is always assigned $\alpha(x)$. The latter term is equal to $\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha)$, and we are done.

- (ii) We consider the three cases of Definition 8.19. Note that if $x = y$, the statement holds trivially.

If $y \in V_{\text{nf}}(\alpha_0)$, then $\text{cost}(\alpha_0, \alpha, y) = S_D$. By Observation 8.21, we have $\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y) \leq S_D$.

If $y \in V_{\text{fr}}(\alpha_0)$ or $y \in V_{\text{fo}}(\alpha_0)$, then $\text{cost}(\alpha_0, \alpha, y)$ is the probability that y is guessed in the remainder of ppsz. If we now fix another variable x to $\alpha(x)$, then this probability cannot decrease (why?), so $\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y) \leq \text{cost}(\alpha_0, \alpha, y)$.

- (iii) We have $\text{lkhd}(\alpha_0, \alpha) = \mathbb{E}[\text{ppsz}(F, V, \alpha_0, |F|) = \alpha]$. Let x be the variable selected next in ppsz (x is obviously u.a.r. in $\mathcal{U}(\alpha_0)$). If x is frozen, then ppsz always sets x to $\alpha(x)$. If x is non-frozen, then ppsz sets x to $\alpha(x)$ with probability $1/2$. x is non-frozen with probability $|V_{\text{nf}}(\alpha_0)|/n(\alpha_0)$. Therefore

$$\begin{aligned}
& \text{lkhd}(\alpha_0, \alpha) \\
&= \Pr(x \in V_{\text{nf}}(\alpha_0)) \cdot \frac{1}{2} \cdot \mathbb{E}[\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}) \mid x \in V_{\text{nf}}(\alpha_0)] + \\
&\quad \Pr(x \notin V_{\text{nf}}(\alpha_0)) \mathbb{E}[\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}) \mid x \notin V_{\text{nf}}(\alpha_0)] \\
&= \frac{|V_{\text{nf}}(\alpha_0)|}{n(\alpha_0)} \cdot \frac{1}{2} \cdot \mathbb{E}[\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}) \mid x \in V_{\text{nf}}(\alpha_0)] + \\
&\quad \left(1 - \frac{|V_{\text{nf}}(\alpha_0)|}{n(\alpha_0)}\right) \mathbb{E}[\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}) \mid x \notin V_{\text{nf}}(\alpha_0)] \\
&= \frac{1}{2} \mathbb{E}[\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\})] \\
&\quad + \frac{1}{2} (1 - |V_{\text{nf}}(\alpha_0)|/n(\alpha_0)) \mathbb{E}[\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}) \mid x \notin V_{\text{nf}}(\alpha_0)].
\end{aligned}$$

From (i), we know that if $x \notin V_{\text{nf}}(\alpha_0)$, then $\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}) = \text{lkhd}(\alpha_0)$, and so we have

$$\begin{aligned}
& \text{lkhd}(\alpha_0, \alpha) = \\
&= \frac{1}{2} \mathbb{E}[\text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\})] + \\
&\quad + \frac{1}{2} \left(1 - \frac{|V_{\text{nf}}(\alpha_0)|}{n(\alpha_0)}\right) \text{lkhd}(\alpha_0, \alpha).
\end{aligned}$$

Multiplying the equation by 2 and collecting the $\text{lkhd}(\alpha_0, \alpha)$ -terms on the left-hand side yields the statement.

- (iv) If $y \in V_{\text{nf}}(\alpha_0)$ or $y \in V_{\text{fo}}(\alpha_0)$, then the statement is trivial: With probability $1/n(\alpha_0)$, y is selected in ppsz, and the cost of y reduces to 0. If y is not selected, then by (ii) the cost does not increase.

The interesting case is if $y \in V_{\text{fr}}(\alpha_0)$, where x is frozen but not forced. In that case, the cost of y is the probability that y is guessed, and the statement tells us that this probability is reduced by $1/n(\alpha_0)$ after

one step. This is because with probability $1/n(\alpha_0)$, y comes next in π and is guessed now (as x is not forced now). This $1/n(\alpha_0)$ is counted in $\text{cost}(\alpha_0, \alpha, y)$, but not in $E(\text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y))$.

□

From these basic facts, we will finally be able to prove the main theorem.

8.3.4 Bounding Correlations in Weighted Sums

But first we have to combine the observed expected changes in likelihood and in cost and investigate their combined effects.

Let us start with the simple cases. If some variable $y \in V_{fo}(\alpha_0)$ is forced, then the cost of y is always zero and will remain zero at all times. This is true for all satisfying assignments, so we do not care about shifts in the likelihood distribution.

A similar fact is true for the non-frozen variables. We have bounded the expected drop in cost for a non-frozen variable in Lemma 8.22 as $S_D/n(\alpha_0)$, and this drop stems exclusively from the case when the variable in question is being assigned and eliminated, causing its cost to drop to zero. If this happens, it happens in all assignments at once and so we do not care if at the same time there are shifts in the likelihood distribution.

The difficult case arises for the variables which are frozen but not currently forced, $y \in V_{fr}(\alpha_0)$. There, each assignment α contributes a potentially different cost for y and so if at the same time the likelihood distribution over the various satisfying assignments changes, there can be potentially harmful correlations (as we have announced during the introduction to this section). The following correlation inequality will come handy in bounding these possibly negative effects.

Lemma 8.23 *Let $A, B \in \mathbf{R}$ be random variables and $a, b, \bar{a}, \bar{b} \in \mathbf{R}$ fixed numbers such that $A \geq a$ and $B \leq b$ always and $E(A) = \bar{a}$ and $E(B) = \bar{b}$. Then*

$$E(A \cdot B) \leq a\bar{b} + b\bar{a} - ab.$$

Proof. We can write

$$\mathbb{E}(A \cdot B) = \mathbb{E}((A - a) \cdot B) + a \mathbb{E}(B)$$

and then use $B \leq b$ and $A \geq a$ to obtain

$$\mathbb{E}(A \cdot B) \leq b \mathbb{E}(A - a) + a \mathbb{E}(B) = b\bar{a} - ba + a\bar{b},$$

as claimed. \square

Applying this result, we can prove the following extension of Lemma 8.22 for the frozen variables.

Lemma 8.24 *Let α be a fixed satisfying assignment and $\alpha_0 \subseteq \alpha$. Let $y \in V_{\text{fr}}(\alpha_0)$ be a fixed frozen variable. If we select $x \in U(\alpha_0)$ uniformly at random and assign it according to α , then*

$$\begin{aligned} \mathbb{E}(\text{wcost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)) &\leq \\ &\leq \left(1 + \frac{|V_{\text{nf}}(\alpha_0)|}{n(\alpha_0)}\right) \text{wcost}(\alpha_0, \alpha, y) - \frac{\text{lkhd}(\alpha_0, \alpha)}{n(\alpha_0)}. \end{aligned}$$

Proof. To apply Lemma 8.23 to the random experiment indicated, we set $A := \text{lkhd}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, y)$, $a := \text{lkhd}(\alpha_0, \alpha)$ and $\bar{a} := a(1 + |V_{\text{nf}}(\alpha_0)|/n(\alpha_0))$. We have $A \geq a$ and $\mathbb{E}(A) = \bar{a}$ from Lemma 8.22. Further, we set $B := \text{cost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, y, \alpha, y)$, $b := \text{cost}(\alpha_0, \alpha, y)$ and $\bar{b} := \text{cost}(\alpha_0, \alpha, y) - 1/n$. Again, we have $B \leq b$ and $\mathbb{E}(B) = \bar{b}$ from Lemma 8.22. Invoking Lemma 8.23, we deduce that

$$\mathbb{E}(\text{wcost}(\alpha_0 \cup \{x \mapsto \alpha(x)\}, \alpha, y)) \leq a\bar{b} + b\bar{a} - ab,$$

which matches the claim. \square

8.3.5 Trading Survival Probability for Cost Savings

Suppose α_0 is the current state of the algorithm. In a single following step, ppsz selects a variable $x \in U(\alpha_0)$ uniformly at random. If x is D-implied, it is being set accordingly, if not, it is assigned a value uniformly at random. Call $b \in \{0, 1\}$ the random variable representing the value ppsz chooses to

assign to x . Two things happen during such a step.

Firstly, there is a certain *survival probability* with which ppsz will make an assignment that preserves satisfiability such that $\alpha_0 \cup \{x \mapsto b\}$ can still be completed to some satisfying assignment and we stay in the game. Let us write

$$\mathcal{S} := \{(x', b') \in \mathcal{U}(\alpha_0) \times \{0, 1\} \mid F^{\alpha_0 \cup \{x' \mapsto b'\}} \text{ is satisfiable}\}$$

for the set of all variable-value pairs (x', b') such that $x' \mapsto b'$ preserves satisfiability. Then the survival probability of this step of the algorithm is the probability that $(x, b) \in \mathcal{S}$. This survival probability is easy to compute: whenever x is a forced variable, we choose the correct value by D-implication and so the probability is one, whenever x is a non-frozen variable, both values are okay and we survive as well with probability one. Only if x is a frozen but not yet forced variable, we have a probability of $\frac{1}{2}$ to make a mistake.

Secondly, there is a *drop in the costs*: an additional variable x gets a value which, for one, removes the cost of x itself and may additionally help to force other variables, such that the costs of frozen variables go down. How large are the new costs on average?

In principle, we are interested in the average drop in costs *given* that the current step makes the algorithm survive. However, since the survival probability as outlined above is different for different types of variables, entering such a conditional space would introduce a bias in the choice of x and b which is difficult to handle and would complicate matters. Instead, we here suggest to analyse the *uniform* distribution over \mathcal{S} . This distribution neither ideally reflects what the algorithm is actually doing, nor is the processed variable selected uniformly such that our cost decrease estimates from the last subsection could be applied seamlessly. But it is a handy intermediate distribution that can be related sufficiently easily to both aspects. We now claim the following.

Lemma 8.25 *If $(X, B) \in \mathcal{S}$ is selected uniformly at random, then*

$$\mathbb{E}(\text{cost}(\alpha_0 \cup \{X \mapsto B\})) \leq \text{cost}(\alpha_0) - \frac{|V_{\text{fr}}(\alpha_0)|}{|\mathcal{S}|} - \frac{2S_D \cdot |V_{\text{nf}}(\alpha_0)|}{|\mathcal{S}|}.$$

Proof. We analyze the cost decrease contribution of the different types of variables separately. Each variable forced in state α_0 contributes zero cost both before and after the step.

For a non-frozen variable $y \in V_{\text{nf}}(\alpha_0)$, note that \mathcal{S} features two pairs containing y , in contrast to the other types of variables of which \mathcal{S} features only one pair each. This means that with probability $2|V_{\text{nf}}(\alpha)|/|\mathcal{S}|$, a pair featuring y is selected. In that case the cost contribution of y drops from S_D to zero in all assignments. No matter what happens to these costs otherwise (they can certainly not increase by definition), the non-frozen variables hence contribute the last term in the claimed inequality.

Now consider the frozen variables. If we fix some satisfying and α_0 -compatible assignment $\alpha \in \{0, 1\}^V$ and condition the experiment on the event that $\alpha(X) = B$, then X becomes uniformly at random among $U(\alpha_0)$ because \mathcal{S} features exactly one pair (x', b') per variable $x' \in U(\alpha_0)$ such that $\alpha(x') = b'$. This is where the uniform choice over \mathcal{S} comes very handy. As now, we can directly apply our previous Lemma 8.24 to find that conditioning on $\alpha(X) = B$, the cost of each frozen variable drops on average as

$$\begin{aligned} & \mathbb{E}(\text{wcost}(\alpha_0 \cup \{X \mapsto B\}, \alpha, y) \mid \alpha(X) = B) \leq \\ & \leq \underbrace{\left(1 + \frac{|V_{\text{nf}}(\alpha_0)|}{n(\alpha_0)}\right)}_{|\mathcal{S}|/n} \text{wcost}(\alpha_0, \alpha, y) - \frac{\text{lkhd}(\alpha_0, \alpha)}{n(\alpha_0)}. \end{aligned}$$

The condition itself is satisfied with probability $n/|\mathcal{S}|$. If it does not apply, then the cost contribution of α drops to zero altogether. Therefore the unconditional change can be obtained by multiplying the right-hand side with $n/|\mathcal{S}|$ which then yields

$$\mathbb{E}(\text{wcost}(\alpha_0 \cup \{X \mapsto B\}, \alpha, y)) \leq \text{wcost}(\alpha_0, \alpha, y) - \frac{\text{lkhd}(\alpha_0, \alpha)}{|\mathcal{S}|}.$$

If we, on both sides, take the sum over all assignments $\alpha \in \{0, 1\}^V$, the claim follows. \square

Now we are almost done. For the final lemma, we need the following inequality about logarithms:

Lemma 8.26 For $x \geq 0$,

$$\log_2(1+x) \geq \log_2(e) \frac{x}{1+x}.$$

Proof. As $\log_2(x)$ is an antiderivative of $\log_2(e)1/x$ and $\log_2(1) = 0$, we have

$$\log_2(1+x) = \int_1^{1+x} \log_2(e) \frac{1}{t} dt \geq \int_1^{1+x} \log_2(e) \frac{1}{1+x} dt = \log_2(e) \frac{x}{1+x}.$$

□

The following lemma immediately implies Theorem 8.16, as $\text{cost}(\alpha_0) \leq S_{\text{DN}}(\alpha_0)$ by Observation 8.21.

Lemma 8.27 Let α_0 s.t. $F^{[\alpha_0]}$ is satisfiable. Then the overall probability of ppsz to output some satisfying assignment when starting in state α_0 is at least $2^{-\text{cost}(\alpha_0)}$

Proof. We apply induction and suppose that the claim holds for all α_0 that fix a larger number of variables. If α_0 is total, then the statement holds trivially.

Let us denote by $p(\alpha_0)$ the probability that ppsz outputs some satisfying assignment when starting from state α_0 . Let x and b be random variables as defined before, i.e. $x \in U(\alpha_0)$ u.a.r.; and $b \in \{0, 1\}$ the forced value by D-implication if $x \in V_{f_0}(\alpha_0)$ and u.a.r. otherwise. Let

$$w(x, b) := \begin{cases} 2, & x \in V_{f_0}(\alpha_0) \\ 1, & \text{otherwise} \end{cases}$$

be the weight of (x, b) in ppsz (note that iff x is forced, b is *not* chosen u.a.r.).

We have

$$\begin{aligned} p(\alpha_0) &= \mathbb{E}(p(\alpha_0 \cup \{x \mapsto b\})) \\ &= \Pr((x, b) \in \mathcal{S}) \mathbb{E}(p(\alpha_0 \cup \{x \mapsto b\}) \mid (x, b) \in \mathcal{S}) \\ &= \frac{|\mathcal{S}| + |V_{f_0}(\alpha_0)|}{2n(\alpha_0)} \mathbb{E}(p(\alpha_0 \cup \{x \mapsto b\}) \mid (x, b) \in \mathcal{S}). \end{aligned}$$

Now we change the expectation to $(X, B) \in \mathcal{S}$ u.a.r. It is clear that each $(X, B) \in \mathcal{S}$ has weight $w(X, B)$ in the previous expectation. However, we

have to normalize the weights, i.e. the weights have to be 1 on average. We have $\mathbb{E}(w(X, B)) = \frac{\sum_{(X', B') \in \mathcal{S}} w(X', B')}{|\mathcal{S}|}$, so the normalization factor is $\frac{|\mathcal{S}|}{\sum_{(X', B') \in \mathcal{S}} w(X', B')}$. Hence

$$\begin{aligned} p(\alpha_0) &= \frac{|\mathcal{S}| + |V_{fo}(\alpha_0)|}{2n(\alpha_0)} \mathbb{E} \left(\frac{|\mathcal{S}|}{\sum_{(X', B') \in \mathcal{S}} w(X', B')} w(X, B) p(\alpha_0 \cup \{X \mapsto B\}) \right) \\ &= \frac{|\mathcal{S}| + |V_{fo}(\alpha_0)|}{2n(\alpha_0)} \frac{|\mathcal{S}|}{|\mathcal{S}| + |V_{fo}(\alpha_0)|} \mathbb{E}(w(X, B) p(\alpha_0 \cup \{X \mapsto B\})) \\ &= \frac{|\mathcal{S}|}{2n(\alpha_0)} \mathbb{E}(w(X, B) p(\alpha_0 \cup \{X \mapsto B\})). \end{aligned}$$

Using Jensen's inequality gives

$$\begin{aligned} p(\alpha_0) &\geq 2^{\log_2 \left(\frac{|\mathcal{S}|}{2n(\alpha_0)} \right)} 2^{\mathbb{E}(\log_2(w(X, B) p(\alpha_0 \cup \{X \mapsto B\})))} \\ &= 2^{\log_2 \left(\frac{|\mathcal{S}|}{2n(\alpha_0)} \right)} 2^{\mathbb{E}(\log_2(w(X, B))) - \mathbb{E}(-\log_2(p(\alpha_0 \cup \{X \mapsto B\})))}, \end{aligned}$$

by linearity of expectation. Observe that

$$\begin{aligned} \mathbb{E}(\log_2(w(X, B))) &= \Pr(w(X, B) = 2) \\ &= \frac{|V_{fo}(\alpha_0)|}{|\mathcal{S}|}, \end{aligned}$$

and by induction and Lemma 8.25

$$\begin{aligned} \mathbb{E}(-\log_2(p(\alpha_0 \cup \{X \mapsto B\}))) &\leq \mathbb{E}(\text{cost}(\alpha_0 \cup \{X \mapsto B\})) \\ &\leq \text{cost}(\alpha_0) - \frac{|V_{fr}(\alpha_0)|}{|\mathcal{S}|} - \frac{2S_D \cdot |V_{nf}(\alpha_0)|}{|\mathcal{S}|}. \end{aligned}$$

Putting things together, we obtain

$$p(\alpha_0) \geq 2^{\log_2 \left(\frac{|\mathcal{S}|}{2n(\alpha_0)} \right) + \frac{|V_{fo}(\alpha_0)|}{|\mathcal{S}|} - \text{cost}(\alpha_0) + \frac{|V_{fr}(\alpha_0)|}{|\mathcal{S}|} + \frac{2S_D \cdot |V_{nf}(\alpha_0)|}{|\mathcal{S}|}}.$$

As we want to show that $p(\alpha_0) \geq 2^{-\text{cost}(\alpha_0)}$, we need to show

$$\log_2 \left(\frac{|\mathcal{S}|}{2n(\alpha_0)} \right) + \frac{|V_{fo}(\alpha_0)|}{|\mathcal{S}|} - \text{cost}(\alpha_0) + \frac{|V_{fr}(\alpha_0)|}{|\mathcal{S}|} + \frac{2S_D \cdot |V_{nf}(\alpha_0)|}{|\mathcal{S}|} \geq -\text{cost}(\alpha_0).$$

$\text{cost}(\alpha_0)$ conveniently cancels and we are left to show

$$\log_2 \left(\frac{|\mathcal{S}|}{2n(\alpha_0)} \right) + \frac{|V_{fo}(\alpha_0)|}{|\mathcal{S}|} + \frac{|V_{fr}(\alpha_0)|}{|\mathcal{S}|} + \frac{2S_D \cdot |V_{nf}(\alpha_0)|}{|\mathcal{S}|} \geq 0.$$

We write

$$\begin{aligned} \log_2 \left(\frac{|\mathcal{S}|}{2n(\alpha_0)} \right) &= -1 + \log_2 \left(\frac{|\mathcal{S}|}{n(\alpha_0)} \right) \\ &= -1 + \log_2 \left(1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)} \right) \\ &\geq -1 + \log_2(e) \frac{\frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)}}{1 + \frac{|V_{nf}(\alpha_0)|}{n(\alpha_0)}} \\ &= -1 + \log_2(e) \frac{|V_{nf}(\alpha_0)|}{n + |V_{nf}(\alpha_0)|} \\ &= -1 + \log_2(e) \frac{|V_{nf}(\alpha_0)|}{|\mathcal{S}|}, \end{aligned}$$

where the inequality is by Lemma 8.26.

Inserting this, it remains to show

$$-1 + \log_2(e) \frac{|V_{nf}(\alpha_0)|}{|\mathcal{S}|} + \frac{|V_{fo}(\alpha_0)|}{|\mathcal{S}|} + \frac{|V_{fr}(\alpha_0)|}{|\mathcal{S}|} + \frac{2S_D \cdot |V_{nf}(\alpha_0)|}{|\mathcal{S}|} \geq 0.$$

Multiplying by $|\mathcal{S}|$ gives equivalently

$$-|\mathcal{S}| + \log_2(e)|V_{nf}(\alpha_0)| + |V_{fo}(\alpha_0)| + |V_{fr}(\alpha_0)| + 2S_D \cdot |V_{nf}(\alpha_0)| \geq 0.$$

Using $\log_2(e) + 2S_D > 1.44 + 2 \cdot 0.38 > 2$, the left-hand side of the inequality is at least 0 as $|\mathcal{S}| = 2V_{nf}(\alpha_0) + V_{fo}(\alpha_0) + V_{fr}(\alpha_0)$ and we are done. \square

Bibliography

- [Her10] Timon Hertli. Investigating and improving the PPSZ algorithm for SAT, master's thesis. ETH Zürich, 2010. doi: <http://dx.doi.org/10.3929/ethz-a-006206989>.
- [Her11] Timon Hertli. 3-SAT faster and simpler—unique-SAT bounds for PPSZ hold in general. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science—FOCS 2011*, pages 277–284. IEEE Computer Soc., Los Alamitos, CA, 2011.
- [ISTT10] Kazuo Iwama, Kazuhisa Seto, Tadashi Takai, and Suguru Tamaki. Improved randomized algorithms for 3-SAT. In *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin / Heidelberg, 2010.
- [IT04] Kazuo Iwama and Suguru Tamaki. Improved upper bounds for 3-sat. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 328–328, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.
- [Mil12] Sebastian Millius. Towards a generalization of the PPSZ algorithm for large domains and multiple solutions, master's thesis. ETH Zürich, 2012.
- [PPSZ05] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.
- [PPZ99] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chicago J. Theoret. Comput. Sci.*, pages Article 11, 19 pp. (electronic), 1999.

- [Rol06] Daniel Rolf. Improved Bound for the PPSZ/Schöning-Algorithm for 3-SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:111–122, 2006.

Chapter 8*

Sparsification and ETH

by Dominik Scheder (with small adaptations by Emo Welzl)¹

8*.1 k-SAT in Subexponential Time?

We have seen several clever algorithms for solving k-SAT. For 3-SAT, we learned about a deterministic $O(1.5^n)$ algorithm and a randomized $O(1.334^n)$ algorithm. And we mentioned that the $O(1.334^n)$ algorithm can be derandomized with the same bound, and that a randomized bound of $O(1.308^n)$ can be shown (where n denotes the number of variables).

It is a natural question whether this goes on forever, or whether there is some natural barrier which we cannot cross.

We say that k-SAT *can be solved in subexponential time* if for every $\epsilon > 0$, there is an algorithm for k-SAT running in time $O(2^{\epsilon n})$? So the question is:

Can k-SAT be solved in subexponential time?

Note that if $\mathcal{P} = \mathcal{NP}$, then the answer is obviously *yes*. However, even if $\mathcal{P} \neq \mathcal{NP}$, there might be an algorithm for 3-SAT running in time $O(2^{10n/\log n})$ or even $O(2^{1000\sqrt{n}})$, say. Many people believe that the answer to the above question is no. This belief is called the exponential time hypothesis.

Exponential Time Hypothesis (ETH): 3-SAT cannot be solved in subexponential time. In other words, there is an $\epsilon > 0$ such that 3-SAT cannot be solved in time $O(2^{\epsilon n})$.

¹Based on: Impagliazzo, Russell; Paturi, Ramamohan; Zane, Francis (2001), "Which problems have strongly exponential complexity?", Journal of Computer and System Sciences 63 (4): 512-530.

Unless we hope to settle the \mathcal{P} versus \mathcal{NP} question, we cannot expect to prove ETH. However, we can investigate what consequences ETH (or its negation) has. In this chapter, we want to prove the following theorem.

Theorem 8*.1 *If 3-SAT can be solved in subexponential time, then so can k-SAT, for every $k \geq 3$.*

Compare this theorem to the following well-known one:

Theorem 8*.2 *If 3-SAT can be solved in polynomial time, then so can k-SAT, for every $k \geq 3$.*

Let us recall the proof of Theorem 8*.2. We will see that it fails to prove Theorem 8*.1. We prove Theorem 8*.2 by replacing each k-clause by a set of 3-clauses, introducing new variables. That is, for $k \geq 4$, consider a k-clause:

$$C = (x_1 \vee \cdots \vee x_k) .$$

We introduce $k-3$ new variables y_3, y_4, \dots, y_{k-1} (dedicated to this clause C only) and replace the above k-clause by

$$\begin{aligned} & (x_1 \vee x_2 \vee y_3) \wedge (\bar{y}_3 \vee x_3 \vee y_4) \wedge (\bar{y}_4 \vee x_4 \vee y_5) \wedge \dots \\ & \wedge (\bar{y}_{k-2} \vee x_{k-2} \vee y_{k-1}) \wedge (\bar{y}_{k-1} \vee x_{k-1} \vee x_k) . \end{aligned}$$

If all x_i are set to 0, this 3-CNF becomes unsatisfiable. On the other hand, if some x_i is set to 1, we can set the y_j s in a way such that those $k-2$ clauses become satisfied. By replacing every k-clause $C \in F$ (with $k \geq 4$) by $k-2$ many 3-clauses as above, we obtain a 3-CNF formula F' . Note that F is satisfiable if and only if F' is. Also, F' can be obtained from F in polynomial time. Thus, a polynomial time algorithm for 3-SAT gives one for k-SAT. This proves Theorem 8*.2.

8*.2 The Need for Sparsification

If we use the above reduction to prove Theorem 8*.1, we will fail. Let us see why. Let a $(\leq k)$ -CNF F have m clauses and n variables. For each ℓ -clause of F we introduce $\ell-3$ new variables. Thus, F' has at most $n + (k-3)m$ variables (and we have that many, if all clauses are k-clauses). Suppose 3-SAT can be solved in subexponential time. That means, pick your favorite $\epsilon > 0$ and I give you an algorithm solving 3-SAT in time $O(2^{\epsilon n})$. Now use this algorithm to solve F' . Since F' has up to $n + (k-3)m$ variables, a running time of $O(2^{\epsilon(n+(k-3)m)})$ can be concluded. We see it depends on m , the number of clauses in F , and this number

may be as large as $O(n^k)$. If for example $m = n^2$, then this running time is much worse than 2^n , for any constant $\epsilon > 0$. Obviously, our problem is that too many new variables are introduced in the reduction.

What would we actually need for a successful reduction? Suppose there is some algorithm $\text{sparsify}(F, \epsilon)$ that takes a positive real number ϵ and a $(\leq k)$ -CNF formula F on n variables and outputs a $(\leq k)$ -CNF formula F' such that

- F is satisfiable if and only if F' is;
- $|\text{vbl}(F')| \leq n$;
- $|F'| \leq c(\epsilon)n$ for some function $c : \mathbf{R}^+ \rightarrow \mathbf{N}$;
- sparsify runs in time $O(2^{\epsilon n})$.

(Here, think of $c(\epsilon)$ possibly quite large for ϵ small.) Then we can do the obvious:

Compute $F' := \text{sparsify}(F, \epsilon)$.

Transform F' into a (≤ 3) -CNF formula F'' where the number of variables is

$$|\text{vbl}(F'')| \leq |\text{vbl}(F')| + (k-3)|F'| \leq n + (k-3)c(\epsilon)n \leq kc(\epsilon)n .$$

Pick a small value $\delta > 0$. If 3-SAT can be solved in subexponential time, then there is an algorithm for 3-SAT with running time $O(2^{\delta n})$. This algorithm takes time $O(2^{\delta kc(\epsilon)n})$ on F'' .

By choosing δ very small, we can make $\delta kc(\epsilon) \leq \epsilon$. Thus, the total running time – sparsify , transformation to a 3-CNF formula, and running the 3-SAT algorithm, is

$$O(2^{\epsilon n}) + \text{poly}(n) + O(2^{\epsilon n}) .$$

So we can solve k -SAT in subexponential time.

We are left to prove that such an algorithm sparsify exists. Actually, we don't know this. However, something almost as good, still sufficiently good, exists.

Theorem 8*.3 *For every $k \in \mathbf{N}$ there is an algorithm $\text{sparsify}(F, \epsilon)$ which takes as input a $(\leq k)$ -CNF formula over n variables and outputs a list of $(\leq k)$ -CNF formulas $\{F_1, F_2, \dots, F_t\}$ with the following properties.*

1. F is satisfiable if and only if at least one of the F_i is satisfiable;
2. $|\text{vbl}(F_i)| \leq n$.

3. $|F_i| \leq c(k, \epsilon)n$;

4. *sparsify* runs in time $O(2^{\epsilon n})$ and $t \leq 2^{\epsilon n}$.

$c(k, \epsilon)$ is some function depending on k and ϵ (independent of n).

If 3-SAT can be solved in subexponential time, we can now solve k -SAT in time $O(2^{\gamma n})$ as follows: For an ϵ to be determined, compute $\{F_1, \dots, F_t\} = \text{sparsify}(F, \epsilon)$ (in time $O(2^{\epsilon n})$). Then transform each F_i into a 3-CNF formula F'_i with at most $kc(k, \epsilon)n$ variables (altogether in $O(2^{\epsilon n} \text{poly}(n))$). We check for satisfiability of each F'_i using a fast 3-SAT algorithm that runs in time $O(2^{\delta N})$ for N variables; for all the $O(2^{\epsilon n})$ F_i s this takes time $O(2^{\epsilon n} \cdot 2^{\delta kc(k, \epsilon)n})$ altogether. If we have chosen ϵ and δ such that $\epsilon + \delta kc(k, \epsilon) < \gamma$, we are done.

8*.3 The Sparsification Algorithm

The algorithm *sparsify* is quite simple. The analysis is not difficult, but somewhat tedious involving a potpourri of constants.

Let k and ϵ be given. Depending on these numbers, we use some threshold values

$$2 = \theta_0 < \theta_1 < \dots < \theta_{k-1}.$$

(Their exact value shall not bother us at the moment. Just imagine that θ_i is much larger than θ_{i-1} .)

A *sunflower* is a set of clauses $G = \{C_1, \dots, C_s\}$ such that

- all clauses C_i have the same size, call it ℓ ;
- the intersection is non-empty: $C := C_1 \cap \dots \cap C_s \neq \emptyset$.

The set C is the *heart* of the sunflower, and the sets $C_i \setminus C$ are called *petals*. Every petal has size $p := \ell - |C|$. We say the sunflower is *good* if s is large: $s \geq \theta_p$. (At this point it is already clear that a good sunflower cannot consist of one clause, since then $p = 0$ and $\theta_0 = 2$.)

Let G_1 and G_2 be sunflowers of ℓ_1 -clauses and ℓ_2 -clauses, respectively. We say G_1 is *better than* G_2 if (1) $\ell_1 < \ell_2$; or (2) if $\ell_1 = \ell_2$, but the heart of G_1 is bigger than the heart of G_2 . In words, we prefer shorter clauses, and given equal clause length, we prefer bigger hearts (and thus smaller petals).

We are now ready to state the algorithm *sparsify*. It makes use of a small subroutine called *reduce*, that simply removes redundant clauses: A clause $D \in F$ is redundant if F contains some other clause C such that $C \subseteq D$. In that case, we can remove D , and obtain an equivalent formula.

<p>F a CNF formula, POSTCONDITION: returns a CNF formula equivalent to F</p>	<pre> function reduce(F) while $\exists C, D \in F : C \subsetneq D$ do $F \leftarrow F \setminus \{D\};$ return F; </pre>
---	--

<p>F a $(\leq k)$-CNF formula, $\epsilon > 0$ POSTCONDITION: returns a set $\{F_1, \dots, F_t\}$ of $(\leq k)$-CNF formulas</p>	<pre> function sparsify(F, ϵ) if F contains no good sunflower then return $\{F\};$ else $\{C_1, \dots, C_s\} \leftarrow$ a best good sunflower in F; $C \leftarrow C_1 \cap \dots \cap C_s;$ $F_{\text{heart}} \leftarrow \text{reduce}(F \cup \{C\});$ $F_{\text{petals}} \leftarrow \text{reduce}(F \cup \{C_1 \setminus C, \dots, C_s \setminus C\});$ return $\text{sparsify}(F_{\text{heart}}, \epsilon) \cup \text{sparsify}(F_{\text{petals}}, \epsilon);$ </pre>
---	---

For a CNF formula F , a literal u , and a number $\ell \in \mathbb{N}$, define

$$d_\ell(u, F) := |\{C \in F \mid |C| = \ell, u \in C\}| \quad \text{and} \quad d_\ell(F) := \max_u d_\ell(u, F).$$

Observation 8*.4 *Let F be a $(\leq k)$ -CNF formula that contains no good sunflower of ℓ -clauses. Then $d_\ell(F) \leq \theta_{\ell-1} - 1$.*

Proof. We prove the contrapositive. Suppose $d_\ell(F) \geq \theta_{\ell-1}$. Then there is some literal u occurring in at least $\theta_{\ell-1}$ many ℓ -clauses. These ℓ -clauses form a sunflower with petal size at most $\ell - 1$. By definition, this sunflower is good. \square

An immediate corollary of this is

Corollary 8*.5 *Let F be an $(\leq k)$ -CNF formula (with $n := |\text{vbl}(F)|$) not containing any good sunflower. Then for every $1 \leq \ell \leq k$, F contains at most $2\theta_{\ell-1}n$ clauses of size ℓ . In particular, $|F| \leq 2k\theta_{k-1}n$.*

Proof. By the previous observation, $d_\ell(u, F) \leq \theta_{\ell-1}$ for all literals u and every ℓ , $1 \leq \ell \leq k$. Thus, F contains at most $2\theta_{\ell-1}n$ ℓ -clauses. Let us sum up over all k : $|F| \leq 2(\theta_0 + \theta_1 + \dots + \theta_{k-1})n \leq 2k\theta_{k-1}n$. \square

Note that we could improve the bound $2k\theta_{k-1}n$ in various ways, but there is no point in doing so here. In any case, we observe that $2k\theta_{k-1}$ is a number independent of n . (There is a dependence of the θ_i s on k and ϵ which we will clarify only later.)

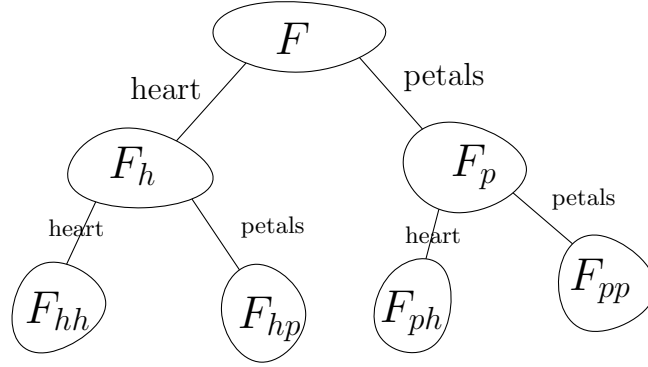


Figure 8*.1: The sparsification tree, top three levels.

We are ready to prove some parts of Theorem 8*.3. Point 2 is clear: We never add variables. Point 3 follows from Corollary 8*.5 by setting $c(k, \epsilon) = 2k\theta_{k-1}$. Let us argue why Point 1 holds. The function `sparsify` computes two new formulas F_{heart} and F_{petals} by adding clauses to F . Clearly, if F is unsatisfiable then both subformulas are unsatisfiable and Point 1 follows by induction. If F is satisfiable, let α be a satisfying assignment. Consider the sunflower $\{C_1, \dots, C_t\}$ used for branching. There are two possibilities: If α satisfies the heart C , then it satisfies F_{heart} . Otherwise, α satisfies all the petals, so it satisfies F_{petals} , and again Point 1 follows from induction.

Here, we swept something under the rug: Induction on what? Well, induction on the recursion tree built by the algorithm. Fair enough, but for this we would have to show that the tree is finite. This should not be too difficult, but let's not bother with it: We will show that it has at most $2^{\epsilon n}$ leaves (if the θ_i s are chosen carefully). This will also prove Point 4.

Proof of Point 4 of Theorem 8*.3

This is by far the most challenging part of the proof, with new constants entering the picture. First, take a look at Figure 8*.1. Every inner node in the recursion tree has two children; one is formed by adding the heart, the other by adding the petals. We can encode a path in the tree as a string in $\{h, p\}^*$. We will show two things: First, every path from the root to a leaf has length at most $a \cdot n$. Second, the number of p 's in such a path is at most n/b . Here, a and b are numbers

depending on k and ϵ ; we will determine them later. This shows that the tree has at most

$$\sum_{i=0}^{n/b} \binom{an}{i} \leq 2^{H(1/ab)an}$$

leaves. It will turn out that $H(1/ab)a \leq \epsilon$, as required. Note that every node of the recursion tree is associated with a CNF formula. Along a path of the tree, we add clauses to our formula (hearts or petals of sunflowers) and might also remove some (since they became redundant).

Lemma 8*.6 *Let F be a formula occurring at an inner node in the recursion tree and let $\{C_1, \dots, C_s\}$ be the sunflower of ℓ -clauses used at this node. Let F' be a child of F . Note that either $F' = F_{\text{heart}}$ or $F' = F_{\text{petals}}$. In any case, F' was formed by adding to F one or more clauses of equal size j , $1 \leq j < \ell$, and then performing the reduce procedure. Then*

$$d_j(F') \leq 2\theta_{j-1}.$$

Proof. The call $\text{sparsify}(F, \epsilon)$ used a sunflower of ℓ -clauses. Since it chooses the best sunflower in the formula, we conclude that F does not contain a good sunflower of j -clauses. By Observation 8*.4, $d_j(F) \leq \theta_{j-1} - 1$. If $F' = F_{\text{heart}}$, then it is formed from F by adding one clause and then removing some. So clearly $d_j(F') \leq \theta_{j-1}$, and we are done.

If $F' = F_{\text{petals}}$, things are somewhat more subtle. Denote the heart by $C := C_1 \cap \dots \cap C_s$ and the petals by $D_i := C_i \setminus C$. By assumption, the petals have size j . Suppose for the sake of contradiction that $d_j(F') > 2\theta_{j-1}$. Then some literal u appears in more than $2\theta_{j-1}$ of the j -clauses of F' . Since $d_j(u, F) \leq \theta_{j-1} - 1$, we conclude that u occurs in many petals: $d_j(u, \{D_1, \dots, D_s\}) \geq \theta_{j-1}$ (actually $> \theta_{j-1} + 1$, but we don't need this here). Without loss of generality, suppose that $u \in D_1 \cap \dots \cap D_{\theta_{j-1}}$. Then

$$C_1 \cap \dots \cap C_{\theta_{j-1}} \supseteq C \cup \{u\}$$

and therefore

$$\{C_1, \dots, C_{\theta_{j-1}}\}$$

is a sunflower of ℓ -clauses with petal size $j - 1$. It is a good sunflower, since it consists of θ_{j-1} clauses. Furthermore, it is better than C_1, \dots, C_s , since its petals are smaller. This is a contradiction, since the algorithm picks the best sunflower.

We conclude that in any case, $d_j(F') \leq 2\theta_{j-1}$. \square

This means that as soon as we add a j -clause to a formula, we can be sure that d_j will be low, i.e., at most $2\theta_{j-1}$ afterwards. It will stay low forever: Adding sets of other size does not change d_j , and adding new sets of size j again makes the lemma apply.

Note that the formulas in the tree contain two types of clauses: *Native* clauses, that have already been in the original formula at the root, and *immigrant* clauses, that have been added either as hearts or petals. Further, when a clause D gets removed from a formula by the procedure *reduce*, there is some other clause $C \subseteq D$. We say C *removes* D . (Note that C might not be unique.)

Lemma 8*.7 *In a call to $\text{reduce}(F)$, a clause C can remove at most $2\theta_{j-1}$ immigrant j -clauses of F .*

Proof. If F contains no immigrant j -clause, we are done. Otherwise, Lemma 8*.6 tells us that $d_j(u, F) \leq 2\theta_{j-1}$. This also implies that any non-empty clause C is contained in at most $2\theta_{j-1}$ j -clauses. Since C removes exactly those clauses that contain it, we are done. \square

Now we have all the tools to finish the proof. Recall that we want to show (1) that every path has at most n/b edges, and (2) the number of petal branches along a path is at most n/b . To show (1), we bound the number of immigrant clauses added to our formula along the path. We define numbers β_ℓ , $\ell = 1, 2, \dots, k-1$, recursively by $\beta_1 = 2$ and $\beta_\ell = 4\theta_{\ell-1}\beta_{\ell-1}$.

Lemma 8*.8 *Consider a root-leaf path in the tree. For $1 \leq \ell \leq k-1$, let $I_{\leq \ell}$ denote the number of immigrant ($\leq \ell$)-clauses that are added along the path. Then $I_{\leq \ell} \leq \beta_\ell n$.*

Since every edge adds at least one immigrant ($\leq k-1$)-clause, we conclude:

Corollary 8*.9 *Any path has at most $\beta_{k-1}n$ edges.*

Proof of Lemma 8.8.* We use induction on ℓ . The case $\ell = 1$ is obvious, since we never add an empty clause and the number of 1-clauses is $2n$ (and we never add a clause twice). So let us assume that $\ell \geq 2$.

Let F be the formula at the root and F' be the formula at the leaf. Let R_ℓ the number of immigrant ℓ -clauses that are removed along the path. An immigrant ℓ -clause either (i) is removed by some immigrant ($\leq \ell-1$)-clause, or (ii) contained in F' . Therefore

$$I_{\leq \ell} \leq I_{\leq \ell-1} + R_\ell + \text{number of } \ell\text{-clauses in } F'.$$

The latter is at most $2\theta_{\ell-1}n$, by Corollary 8*.5. Let us bound R_ℓ from above. Suppose the algorithm removes an immigrant ℓ -clause D . This means that just before it has added an immigrant clause C with $C \subsetneq D$. By Lemma 8*.7, C removes at most $2\theta_{\ell-1}$ other immigrant clauses. By induction, the algorithm adds at most $\beta_{\ell-1}n$ ($\leq \ell - 1$)-clauses along the path. Together, they remove at most $2\theta_{\ell-1}\beta_{\ell-1}n$ immigrant ℓ -clauses. Thus,

$$R_\ell \leq 2\theta_{\ell-1}\beta_{\ell-1}n ,$$

and finally

$$I_{\leq \ell} \leq \underbrace{I_{\leq \ell-1}}_{\beta_{\ell-1}n} + \underbrace{R_\ell}_{2\theta_{\ell-1}\beta_{\ell-1}n} + \underbrace{\ell\text{-clauses in } F'}_{2\theta_{\ell-1}n} \leq 4\theta_{\ell-1}\beta_{\ell-1}n = \beta_\ell n .$$

□

We have shown that every path has at most $\beta_{k-1}n$ edges. Next, we show that every path has at most n/b petal edges.

Lemma 8*.10 *Assume that there is a constant μ with $\frac{\theta_j}{\beta_j} = \mu$ for all $j = 1, 2, \dots, k-1$. Then, along any root-leaf path, at most $(k-1)n/\mu$ edges are due to adding petals.*

Proof. How often does the algorithm add petals of size j , along a given path? Let P_j be that number. A good sunflower with petals of size j has at least θ_j petals. Therefore

$$I_j \geq P_j \cdot \theta_j .$$

with I_j the number of j -clauses added along this path. We know that

$$I_j \leq I_{\leq j} \leq \beta_j n$$

from Lemma 8*.8. We conclude that $P_j \leq (\beta_j/\theta_j)n = n/\mu$ edges of the path are due to adding j -petals. □

All of our claims so far go through if the constants μ , β_i , $i = 1, 2, \dots, k-1$, and θ_j , $j = 0, 1, \dots, k-1$ satisfy the conditions

$$\theta_0 = 2, \beta_1 = 2, \theta_1 = 2\mu;$$

$$\beta_\ell = 4\beta_{\ell-1}\theta_{\ell-1} \text{ and } \theta_\ell = \mu\beta_\ell \text{ for } \ell = 2, 3, \dots, k-1.$$

This recurrence determines the sequence $\beta_\ell = 2(8\mu)^{2^{\ell-1}-1}$ (and $\theta_\ell = \mu\beta_\ell$). The parameter μ is left open, and it will be chosen in dependence on ϵ and k .

Every path in the tree has at most $\beta_{k-1}n$ edges; of these edges, at most $(k-1)n/\mu$ are petal-edges. Thus, with $K := k-1$, the number of paths is at most

$$\sum_{r=0}^{Kn/\mu} \binom{\beta_K n}{r} \leq 2^{H\left(\frac{K}{\mu\beta_K}\right) \beta_K n}.$$

Recall the binary entropy function:

$$H(x) = -x \log x - (1-x) \log(1-x).$$

One checks that for small x , the first term dominates, and therefore

$$H(x) \leq -2x \log x.$$

In our case, $x = K/(\mu\beta_K)$, thus

$$H\left(\frac{K}{\mu\beta_K}\right) \beta_K \leq 2 \frac{K}{\mu\beta_K} \log\left(\frac{\mu\beta_K}{K}\right) \beta_K = \frac{2K}{\mu} \log\left(\frac{\mu\beta_K}{K}\right) \quad (8^*.1)$$

Note that

$$\begin{aligned} \log\left(\frac{\mu\beta_K}{K}\right) &= \log \mu + \log \beta_K - \log K \\ &= \log \mu + \log 2 + (2^{K-1} - 1) \log(8\mu) - \log K \\ &\leq 2^{K-1} \log(8\mu). \end{aligned}$$

Plugging this into (8*.1), we see that

$$H\left(\frac{K}{\mu\beta_K}\right) \beta_K \leq \frac{2K}{\mu} 2^{K-1} \log(8\mu) = \frac{K 2^K \log(8\mu)}{\mu} \leq \frac{k 2^k \log(8\mu)}{\mu}$$

Given k , we simply choose μ large enough such that this is at most ϵ . Then

- The tree contains at most $2^{\epsilon n}$ leaves.
- It returns the set of formulas occurring at the leaves.
- Each such formula has at most n variables and at most $2k\theta_{k-1}n$ clauses.

This completes the argument for Theorem 8*.3.

Exercise 8*.1

Subexponential in Number of Clauses

Suppose one can show that for some $k \geq 3$ and every $\epsilon > 0$ there is an $O(2^{\epsilon m})$ time algorithm deciding satisfiability of $(\leq k)$ -CNF formulas with m clauses. Does this have any implications on the Exponential Time Hypothesis?

Exercise 8*.2

How large does it get?

Determine an explicit value of μ (in the proof of sparsification) in terms of ϵ and k . The value does not have to be as small as possible, but not exceedingly large either.

Chapter 9

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

$$C = \{(v_1, c_1), (v_2, c_2), \dots, (v_k, c_k)\}, \quad k \in \mathbb{N}_0,$$

with $v_j \in V$, pairwise distinct, and $c_j \in L_{v_j}$ for $j \in \{1..k\}$.

A *coloring* χ 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 χ *violates constraint* C if $\chi(v) = c$ for all $(v, c) \in C$.

Consider the CNF formula

$$\underbrace{(x \vee y)}_{C_1} \wedge \underbrace{(\bar{x} \vee y \vee z)}_{C_2} \wedge \underbrace{(y \vee \bar{z})}_{C_3} \wedge \underbrace{(x \vee \bar{y} \vee z)}_{C_4}.$$

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 9.1.

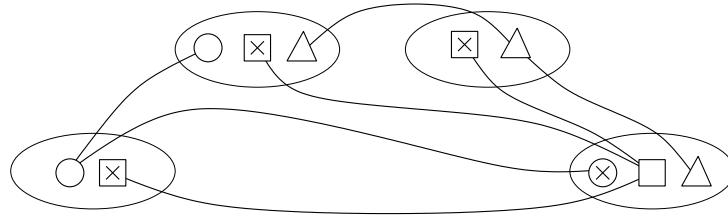


Figure 9.1: Visualizing a CSP with 2-constraints as a graph. The \times 's indicate a possible solution—an independent set which has one vertex (color) in each variable.

Proper 3-colorability of a graph $G = (V, E)$ translates to CSP as follows.

- Variables: V , all with the same color list $\{1, 2, 3\}$.
- Constraints: $\{(u, i), (v, i)\}$ for all $\{u, v\} \in E$ 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 (ℓ, k) -CSP, $(\leq \ell, k)$ -CSP, and $(\ell, \leq k)$ -CSP stands for.

Exercise 9.1

CSP vs. CNF

$k, \ell \in \mathbf{N}$. 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 ℓn variables so that S is satisfiable iff F is satisfiable.

Exercise 9.2

Two Colors

$k \in \mathbf{N}$. 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.

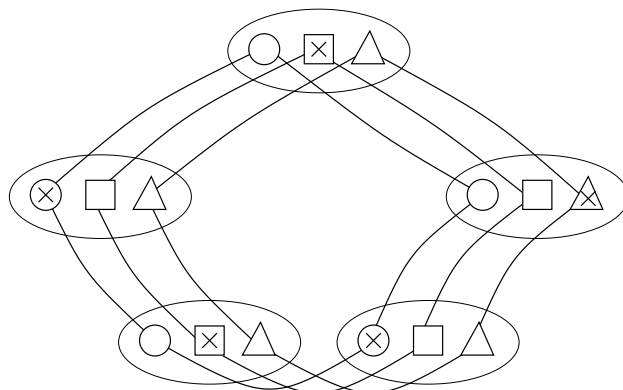


Figure 9.2: Visualizing a CSP with 2-constraints (from proper 3-colorability of a cycle of length 5) as a graph (\times 's indicate a possible solution).

Exercise 9.3

Unsatisfiability Needs many Constraints

$k, \ell \in \mathbf{N}$. Find $b(\ell, k)$ (as large as possible), such that every unsatisfiable (ℓ, k) -CSP must have at least $b(\ell, k)$ constraints.

9.1 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 $O(\sqrt{2}^n \text{poly}(n))$, 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 9.1 *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 9.3. 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.)

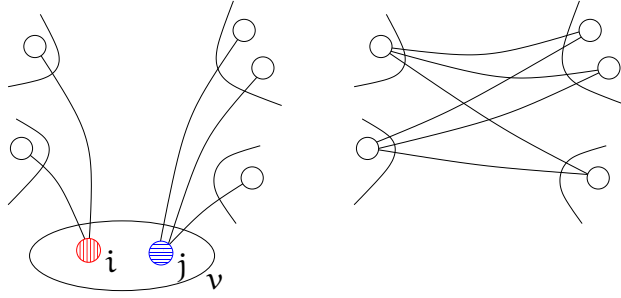


Figure 9.3: Removal of a variable with two possible colors.

On the other hand, let χ 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 χ violates the newly introduced constraint $\{(u, \chi(u)), (w, \chi(w))\}$; contradiction.

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 9.2 $n \in \mathbf{N}, n \geq 2$. *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'' .

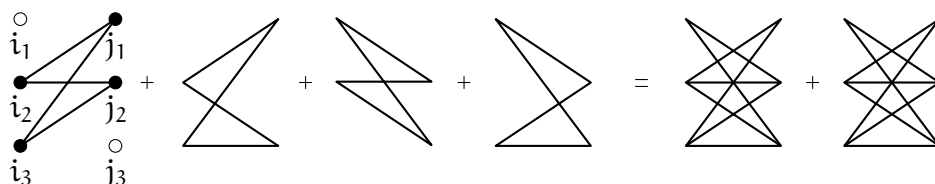


Figure 9.4: The four sets $\{i_2, i_3\} \times \{j_1, j_2\}$, $\{i_2, i_3\} \times \{j_1, j_3\}$, $\{i_1, i_2\} \times \{j_2, j_3\}$, and $\{i_1, i_3\} \times \{j_2, j_3\}$ cover each pair in $(\{i_1, i_2, i_3\} \times \{j_1, j_2, j_3\}) \setminus \{(i_1, j_1)\}$ exactly twice.

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 9.4. If this event materializes, S'' is satisfiable as well. Now, since 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 9.1), so that S'' is satisfiable iff S' is satisfiable. \square

For solving a $(3, 2)$ -CSP S with n variables we apply Lemma 9.2 to reduce the number of variables by at least 2 for at most $\lfloor \frac{n}{2} \rfloor$ 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 9.3 $\lambda \in \mathbb{R}^+$. There is a randomized algorithm with running time $O(\lambda\sqrt{2}^n \text{poly}(n))$ that either determines satisfiability of a $(3, 2)$ -CSP over n variables, or concludes unsatisfiability with error probability at most $e^{-\lambda}$.

Corollary 9.4 $\lambda \in \mathbb{R}^+$. There is a randomized algorithm with running time $O(\lambda\sqrt{2}^n \text{poly}(n))$ that either determines proper 3-colorability of an n -vertex graph, or concludes that no such coloring exists with error probability at most $e^{-\lambda}$.

NOTE 9.1 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 10

Random k -CNF Formulas

$k, n, m \in \mathbf{N}$, $n \geq k$. A *random k -CNF formula with m clauses over n variables*, *random CNF-formula $F_k(n, m)$ for short*, is a CNF formula $\{C_1, C_2, \dots, C_m\}$ 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 ∞ 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.

10.1 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 10.1 ([Franco, Paull, 1983]) $k \in \mathbf{N}$, $(\ln 2) \cdot 2^k < c \in \mathbf{R}$, both constant, and $n \in \mathbf{N}$. 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 α is 2^{-k} .

It follows that the probability of α satisfying F is $(1 - 2^{-k})^m$, and thus the expected number of satisfying assignments of F is

$$\begin{aligned} \mathbb{E}(|\text{sat}_V(F)|) &= 2^n (1 - 2^{-k})^m \leq 2^n (1 - 2^{-k})^{cn} \\ &< 2^n e^{-2^{-k}cn} = 2^n e^{-(\ln 2)n} e^{-2^{-k}c'n} = e^{-2^{-k}c'n} \end{aligned}$$

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 existence¹ of a satisfying assignment tends to 0. \square

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

$$\Pr(F \text{ is satisfiable} \mid N' = n') = \left(\frac{1}{2}\right)^{m-n'}$$

and

$$\mathbb{E}(|\text{sat}(F)| \mid F \text{ is satisfiable and } N' = n') = 2^{n-n'}$$

But²

$$\mathbb{E}(N') = n \left(1 - \left(1 - \frac{1}{n}\right)^m\right) \leq n \left(1 - e^{-m/(n-1)}\right) = n(1 - e^{-\lambda})$$

¹For X a random variable with $X \in \mathbb{N}_0$, we have $\Pr(X \geq 1) = \sum_{i=1}^{\infty} \Pr(X = i) \leq \sum_{i=1}^{\infty} i \cdot \Pr(X = i) = \mathbb{E}(X)$.

²We use here that $(1 - \frac{1}{n})^{n-1} > e^{-1}$ for all $2 \leq n \in \mathbb{N}$, which is equivalent to $(1 + \frac{1}{n-1})^{n-1} < e$.

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 $N' \approx \left(1 - \frac{1}{e}\right)n$, for which the probability of satisfiability is $2^{-n/e}$, which³ 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 with $k = 2$ a linear number of clauses (with the right constant in linearity) gives a satisfiable formula almost surely.

Theorem 10.2 ([Chvátal, Reed, 1992]) $1 > c \in \mathbb{R}^+$, constant, and $n \in \mathbb{N}$. A random CNF formula $F_2(n, m)$ with $m \leq cn$ 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, $s \geq 2$, is a sequence of $s + 1$ clauses

$$\{v, u_1\}, \{\overline{u_1}, u_2\}, \{\overline{u_2}, u_3\}, \dots, \{\overline{u_{s-1}}, u_s\}, \{\overline{u_s}, w\} \quad (10.1)$$

where (u_1, u_2, \dots, u_s) is a sequence of literals that come from s distinct variables, and $\text{vbl}(\{v\}) \subseteq \text{vbl}(\{u_2, u_3, \dots, u_s\})$, $\text{vbl}(\{w\}) \subseteq \text{vbl}(\{u_1, u_2, \dots, u_{s-1}\})$.

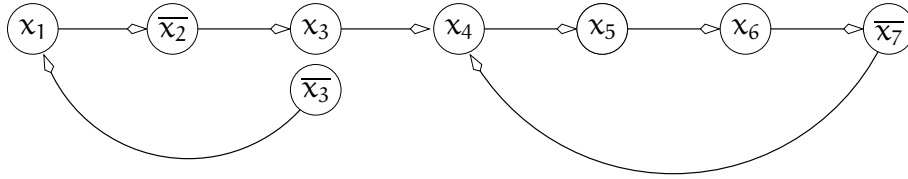


Figure 10.1: The appearance of a bicycle in the directed graph (as defined in Section 3.3) of a 2-CNF formula.

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

$$\overline{v} \rightarrow u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_s \rightarrow w.$$

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

$$x \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_t \rightarrow x \quad \text{with } \overline{x} \in \{v_2, v_3, \dots, v_{t-1}\}.$$

³An analysis of $\Pr(F \text{ is satisfiable})$ and $E(|\text{sat}(F)| | F \text{ is satisfiable})$ is not yet obtained. We leave this as an exercise.

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, \dots, 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

$$\binom{m}{s+1} (s+1)! \left(\frac{1}{4 \binom{n}{2}} \right)^{s+1} < m^{s+1} \left(\frac{1}{2n(n-1)} \right)^{s+1}. \quad (10.2)$$

For a slow development of this bound, let us denote by (C_1, C_2, \dots, C_m) the sequence of clauses generated for F . We define an occurrence of an s -bicycle (as given in (10.1)) as a sequence $\sigma = (i_1, i_2, \dots, i_{s+1})$ of distinct integers in $\{1..m\}$ such that

$$C_{i_1} = \{v, u_1\}, C_{i_2} = \{\bar{u}_1, u_2\}, \dots, C_{i_{s+1}} = \{\bar{u}_s, w\}.$$

There are $\binom{m}{s+1} (s+1)!$ sequences as σ , and each one is an occurrence of a specific s -bicycle with probability $(4 \binom{n}{2})^{-(s+1)}$. Therefore, the left hand side in (10.2) 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

$$\Pr(F \text{ is unsatisfiable}) \leq \Pr(F \text{ has a bicycle}) = o(1), \text{ provided } c < 1.$$

□

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 [Goerd, 1999] and [Fernandez de la Vega, 92] for independent proofs.

10.2 Improved Upper Bound for 3-CNF

Theorem 10.1 shows that a random 3-CNF formula with at least $5.19n$ clauses is almost surely unsatisfiable. Next we want to improve on this by demonstrating that this 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 $\widehat{\text{sat}}_V(F)$. Note that $\widehat{\text{sat}}_V(F) \subseteq \text{sat}_V(F)$ and $\widehat{\text{sat}}_V(F) \neq \emptyset$ iff $\text{sat}_V(F) \neq \emptyset$.⁴

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 $\mathbb{E}(|\widehat{\text{sat}}(F)|)$ (we skip the subindex V in $\widehat{\text{sat}}_V()$ for the time being).

As a first step we analyze for $\alpha = \{1\}^V$ and $\ell \leq n$ distinct assignments α_i , $i \in \{1.. \ell\}$, with $d_H(\alpha_i, \alpha) = 1$ the probability

$$\begin{aligned} & \Pr \left(\alpha \in \text{sat}(F) \wedge \bigwedge_{i=1}^{\ell} \alpha_i \notin \text{sat}(F) \right) \\ &= \Pr \left(\bigwedge_{i=1}^{\ell} \alpha_i \notin \text{sat}(F) \mid \alpha \in \text{sat}(F) \right) \cdot \underbrace{\Pr(\alpha \in \text{sat}(F))}_{(7/8)^m} \end{aligned}$$

which is the same as $\Pr(\alpha \in \widehat{\text{sat}}(F))$ for an assignment with ℓ zeros.

Let us try $\Pr(\alpha_1 \notin \text{sat}(F) \mid \alpha \in \text{sat}(F))$ 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 α satisfies the clause. Then we choose a variable for each of the three positions.

Now for α_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 $\alpha_1(x) = 0$. (i) happens with probability $\frac{3}{7}$, and conditioned on this, (ii) happens with probability $\frac{1}{n}$. That is,

$$\Pr(\alpha_1 \notin \text{sat}(F) \mid \alpha \in \text{sat}(F)) = 1 - \left(1 - \frac{3}{7n}\right)^m$$

In order to continue we need a lemma.

⁴Indeed, for a 1-CNF formula $|\widehat{\text{sat}}_V(F)| \in \{0, 1\}$ and therefore $\Pr(F \text{ is satisfiable}) = \mathbb{E}(|\widehat{\text{sat}}_V(F)|)$.

Lemma 10.3 $m \in \mathbf{N}$ 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 ,

$$\Pr\left(\bigwedge_{a \in T} (a \in \mathcal{X})\right) \leq \prod_{a \in T} \Pr(a \in \mathcal{X}) = \prod_{a \in T} (1 - (1 - p_a)^m) .$$

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]). \square

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

$$\begin{aligned} \Pr\left(\bigwedge_{i=1}^{\ell} \alpha_i \notin \text{sat}(F) \mid \alpha \in \text{sat}(F)\right) &\leq \left(1 - \left(1 - \frac{3}{7n}\right)^m\right)^{\ell} \\ &\leq \left(1 - e^{-3c/7} + o(1)\right)^{\ell} \end{aligned}$$

for $m > cn$, $c \in \mathbf{R}^+$. Now

$$\begin{aligned} \mathbb{E}(|\widehat{\text{sat}}(F)|) &\leq \left(\frac{7}{8}\right)^{cn} \sum_{\ell=0}^n \binom{n}{\ell} \left(1 - e^{-3c/7} + o(1)\right)^{\ell} \\ &= \left(\frac{7}{8}\right)^{cn} \left(2 - e^{-3c/7} + o(1)\right)^n \end{aligned}$$

which tends to zero for n growing, as long as c satisfies

$$\left(\frac{7}{8}\right)^c \left(2 - e^{-3c/7}\right) < 1 ;$$

this holds for $c \geq 4.667$.

⁵identically and independently distributed

Theorem 10.4 ([Klousis et al., 1998]) $n \in \mathbf{N}$. A random CNF formula $F_3(n, m)$ with $m \geq 4.667n$ is unsatisfiable with probability $1 - o(1)$.

NOTE 10.1 Currently known bounds say that a random CNF formula $F_3(n, m)$ with $m \leq 3.42n$ 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).
- [Asano, Williamson, 2002] TAKAO ASANO, DAVID P. WILLIAMSON: Improved approximation algorithms for MAX SAT, *Journal of Algorithms* **42:1** (2002) 173–202.
- [Avidor *et al.*, 2005] ADI AVIDOR, IDO BERKOVITCH, AND URI ZWICK: Improved Approximation Algorithms for MAX NAE-SAT and MAX SAT, *Proc. 3rd International Workshop on Approximation and*

Online Algorithms (WAOA), Lecture Notes in Computer Science
3879 (2005) 27–40.

- [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, Wigderson, 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, Szemerédi, 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, Springer 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 VSEMIROV: 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. BOUFKHAH, 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 unsatisfiability threshold, *Theoretical Computer Science* **265** (2001) 187–197.
- [Ehrenfeucht, Karpinski, 1990] ANDRZEJ EHRENFUCHT, 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 Colloquia 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).
- [Goerdts, 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. GRIMMETT, 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 SCHÜLER, 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 of forbidden structures for extremal algorithmic problems, *Theoretical Computer Science* **40** (1985) 199–210.
- [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).
- [Jukna, 2001] SATASYS JUKNA: *Extremal Combinatorics - With Applications in Computer Science*, Springer Verlag, Berlin, Heidelberg (2001).
- [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.
- [Käppeli, 2005] CLAUDIA KÄPPELI: *Satisfiability in 2-Satisfiable Formulas*, Semester Thesis, Inst. of Theoretical Computer Science, ETH Zurich, Switzerland (2005).
- [Käppeli, 2006] CLAUDIA KÄPPELI: *Ratio of Satisfiable Clauses under Local Constraints*, Master Thesis, Inst. of Theoretical Computer Science, ETH Zurich, Switzerland (2006).
- [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 maximization 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.
- [Kráľ', 2004] DANIEL KRÁL': Locally satisfiable formulas, *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2004) 330–339.
- [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, Nešetřil, 1998] JIRÍ MATOUŠEK, JAROSLAV NEŠETRIL: *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 assignment, *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).
- [Trevisan, 2004] LUCA TREVISAN: On local versus global satisfiability, *SIAM Journal on Discrete Mathematics* **17:4** (2004) 541–547.
- [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 approximation 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, HAIYOU 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 Fibonacci¹ 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 $\frac{1+\sqrt{5}}{2} \approx 1.618033\dots$, 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

$$g(n) \leq \begin{cases} a_n & \text{if } n < k, \text{ and} \\ a + \sum_{i=1}^k c_{k-i} \cdot g(n-i) & \text{otherwise.} \end{cases}$$

Here, in general, a , a_i , and c_i , $0 \leq i < k$ are real numbers with $c_0 \neq 0$. *For our applications we assume that these are non-negative numbers—thus $c_0 > 0$ —*

¹Leonardo di Pisa, called Fibonacci, around 1175-1245 (Pisa, Italy)

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

$$\underbrace{\lambda g(n) + 1}_{f(n)} \leq \begin{cases} \lambda a_n + 1 & \text{if } n < k, \text{ and} \\ \lambda a + 1 + \sum_{i=1}^k c_{k-i} \cdot \underbrace{(\lambda g(n-i) + 1 - 1)}_{f(n-i)} & \text{otherwise.} \end{cases}$$

We substitute $f(n)$ for $\lambda g(n) + 1$, and we choose $\lambda = \frac{(\sum_{i=0}^{k-1} c_i) - 1}{a}$ (which is indeed positive by our assumptions) so that $\lambda a + 1 - \sum_{i=1}^k c_{k-i}$ vanishes. Then homogeneous inequalities for $f()$, $n \geq k$, evolve

$$f(n) \leq \begin{cases} \lambda a_n + 1 & \text{if } n < k, \text{ and} \\ \sum_{i=1}^k c_{k-i} \cdot f(n-i) & \text{otherwise.} \end{cases}$$

Via $g(n) = \frac{f(n)-1}{\lambda}$ any asymptotic upper bound for $f()$ implies the same asymptotic upper bound for $g()$.

If r is a positive real root² of

$$x^k = \sum_{i=0}^{k-1} c_i x^i, \quad \text{where } c_0 > 0, c_i \geq 0 \text{ for } 1 \leq i < k, \text{ and } \sum_{i=0}^{k-1} c_i > 1,$$

then there exists a $b > 0$ such that $f(n) \leq b \cdot r^n$ for all $n \in \mathbb{N}$. 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

$$t(n) \leq \begin{cases} 1 & \text{if } n < 3, \text{ and} \\ 1 + t(n-1) + t(n-2) + t(n-3) & \text{otherwise.} \end{cases}$$

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, Nešetřil, 1998, Section 10.3].

²In 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.

A.3 Markov Chains

Given some countable (infinite or finite) set S , we consider an infinite sequence of random variables

$$(X_t)_{t \in \mathbf{N}_0} = X_0, X_1, X_2, \dots \quad \text{with } X_t \in S$$

such that X_t , $t \in \mathbf{N}$, depends on X_{t-1} only (and not on any of the previous history X_0, X_1, \dots, X_{t-2}). More formally,

$$\begin{aligned} \Pr(X_t = j | X_0 = j_0, X_1 = j_1, \dots, X_{t-2} = j_{t-2}, X_{t-1} = i) \\ = \Pr(X_t = j | X_{t-1} = i) =: p_{tij} \end{aligned}$$

for all $t \in \mathbf{N}$ and all $(j_0, j_1, \dots, j_{t-2}, i, j) \in S^{t+1}$. A sequence of random variables with this memorylessness property is called a *Markov³ 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 $t \in \mathbf{N}$,

$$\Pr(X_t = j | X_{t-1} = i) = \Pr(X_1 = j | X_0 = i) =: p_{ij}.$$

The (p_{ij}) 's are called *transition probabilities* – they are real numbers that satisfy $0 \leq p_{ij} \leq 1$ 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*.)

A.4 Tail Estimates

Lemma A.1 (Markov's Inequality) *Let X be a nonnegative random variable for which $E(X) > 0$ exists. Then, for all $\lambda \in \mathbf{R}^+$,*

$$\Pr(X \geq \lambda E(X)) \leq \frac{1}{\lambda}.$$

Proof. Let $t \in \mathbf{R}^+$.

$$\begin{aligned} E(X) &= \sum_x x \cdot \Pr(X = x) \\ &= \sum_{x < t} \underbrace{x}_{\geq 0} \cdot \Pr(X = x) + \sum_{x \geq t} \underbrace{x}_{\geq t} \cdot \Pr(X = x) \\ &\geq t \cdot \sum_{x \geq t} \Pr(X = x) \\ &= t \cdot \Pr(X \geq t) \end{aligned}$$

³Andrei Andreyevich Markov, 1856-1922 (St. Petersburg University, Russia)

That is,

$$\Pr(X \geq t) \leq \frac{\mathbb{E}(X)}{t}, \quad \text{for all } t \in \mathbf{R}^+.$$

Now set $t = \lambda \mathbb{E}(X)$ to conclude the statement of the lemma.

□

A.5 Jensen's Inequality

In order to state Jensen's⁴ Inequality we have to clarify two notions. First, a *real interval* is a convex subset I of \mathbf{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 \mathbf{R} , $x \mapsto |x|$ on \mathbf{R} , $x \mapsto -\ln x$ on \mathbf{R}^+ , and $x \mapsto \sin(x)$ on $[\pi, 2\pi]$.

An equivalent definition of a convex function f on I requires that for every $a \in I$ there exists a $\lambda_a \in \mathbf{R}$ 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*

$$f\left(\sum_{i=1}^n \lambda_i x_i\right) \leq \sum_{i=1}^n \lambda_i f(x_i) \quad (\text{A.1})$$

for any $(x_i)_{i=1}^n \in I^n$ and any $(\lambda_i)_{i=1}^n \in (\mathbf{R}_0^+)^n$ with $\sum_{i=1}^n \lambda_i = 1$.

And (provided the expectations involved exist)

$$f(\mathbf{E}(X)) \leq \mathbf{E}(f(X)) \quad (\text{A.2})$$

for any random variable X attaining values in I only.

⁴Johan Ludwig Jensen, 1859-1925 (Denmark)

⁵Why don't we simply say that $I = [a, b]$ for $a, b \in \mathbf{R}$ 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. \mathbf{R} , \mathbf{R}^+ or \mathbf{R}_0^+), i.e. a possibly $-\infty$ and b possibly ∞ . But then, of course, we don't want $-\infty$ or ∞ to occur in a real interval, i.e. $[-\infty, \infty]$... is outruled. Perhaps you like now the definition we chose.

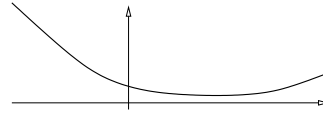


Figure A.1: A convex function.

If the λ_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

$$\sqrt[n]{\prod_{i=1}^n x_i} \leq \frac{\sum_{i=1}^n x_i}{n} \quad \text{for } x_i\text{'s in } \mathbb{R}^+,$$

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 $\mu := \mathbb{E}(X)$ (hence $\mu \in I$). Let λ be such that $f(x) \geq f(\mu) + \lambda(x - \mu)$ for all $x \in I$. Due to linearity of expectation

$$\mathbb{E}(f(X)) \geq \mathbb{E}(f(\mu) + \lambda(X - \mu)) = f(\mu) + \lambda(\underbrace{\mathbb{E}(X)}_{\mu} - \mu) = f(\mu) = f(\mathbb{E}(X))$$

and we are done. □