19.4 The *k*-SAT problem

Let $x_1, \ldots, x_n$ be boolean variables. A literal is a boolean variable x_i or its negation $\overline{x_i}$. A *k-CNF formula* (conjunctive normal form) is an And of clauses, each being an Or of k literals. Such a CNF formula φ is *satisfiable* if these exists a truth assignment $a \in \{0,1\}^n$ for which $\varphi(a) = 1$.

The k-SAT problem is, given a k-CNF, to decide whether it is satisfiable or not; here $k \geq 3$ is assumed to be a fixed constant. Of course, this question can always be solved in 2^n trials: just test all 2^n possible assignments one by one. This dummy strategy will, however, take a huge amount of time on formulas with, say, $n = 100$ variables. Are there any quicker algorithms working, say in time n^c for some constant c? To show that no such algorithm exists is one of the central problems (if not the central) of the whole of computer science, and is known under the name "P vs. NP problem."

On the other hand, the Lovász Local Lemma gives us a tool to quickly recognize some satisfiable CNFs by just looking at their structure!

We say that two clauses *overlap* if they have a common variable x_i, regardless of whether the variable is negated or not in the clauses. In this case we also say that the clauses *share* the variable x_i.

Lemma 19.8. *Let φ be a k-CNF formula. If each of its clauses overlaps with at most 2^{k-2} clauses, then φ is satisfiable.*

Note that the total number n of variables is irrelevant here!

Proof. Consider a random experiment where the variables in φ are assigned truth values by independent tosses of a fair coin. Let A_i be the event that the i-th clause of φ is *not* satisfied. For this event to happen, all k literals in C_i must receive a "wrong" value. Hence, $p = \Pr[A_i] = 2^{-k}$. Further, each A_i is mutually independent of the set of all A_j such that the i-th clause C_i and the j-th clause C_j of φ do not overlap. Hence, the dependency graph of the events A_i has degree $d \leq 2^k/4$. Since $4dp \leq 4 2^{k-2} 2^{-k} = 1$, Lemma 19.1 applies and φ will be satisfied with non-zero probability. $\square$

This lemma is "non-constructive:" it gives no clue on how to find a satisfying assignment in a reasonable (polynomial) time.

Beck (1991) achieved a breakthrough by proving that a polynomial-time algorithm exists which finds a satisfying assignment to every k-CNF formula in which each clause has a neighbourhood of at most $2^{k/48}$ other clauses. His approach was deterministic and used the nonconstructive version of the Lovász Local Lemma as a key ingredient, basically proving that even after truncating clauses to a 48th of their size (a step used to simplify the formula and make it fall apart into small components), a solution remains guaranteed and can then be looked for by exhaustive enumeration. Alon (1991) simplified Beck's algorithm and analysis by introducing randomness and presented an algorithm that works up to neighbourhoods of $2^{k/8}$ in size. Czumaj and Scheideler (2000) later demonstrated that a variant of the method can be made to work for the non-uniform case where clause sizes vary. Srinivasan (2008) improved the bound of what was polynomial-time feasible to essentially $2^{k/4}$ by a more accurate analysis. Finally, Moser (2009) published a polynomial-time algorithm that can cope with neighbourhood size up to 2^{k-5} neighbours, which is asymptotically optimal with a constant gap. Recently, Moser and Tardos (2010) gave a randomized algorithm for the general (non-symmetric) version of the Lovász Local Lemma.

Theorem 19.9 (Moser 2009)**.** *There is a constant c such that, given a k-CNF formula φ with m clauses, none of which overlaps with more than $r = 2^{k-c}$ other clauses, one can find a satisfying assignment for φ in expected time polynomial in m.*

We are not going to prove this theorem in full detail. We rather give a coarse, intuitive and convincing argument that this "must" hold.

Moser's algorithm Solve(φ) is a randomized algorithm consisting of recursive calls of (also recursive) procedures Fix(C) for clauses C of φ: Pick a random assignment $a \in \{0,1\}^n$; while there is an unsatisfied clause C, call Fix(C). The procedure Fix(C) itself is the following recursive procedure:

Step 1: Replace the variables of C with new random values.
Step 2: While there is a clause D that shares a variable with C that is not satisfied, call Fix(D).

First, observe that, *if* Fix(C) *terminates*, then every clause A that was satisfied before Fix(C) is called will remain satisfied after Fix(C) is called. This holds because each flipping of the variables in a clause C can only affect the values of clauses that share a common variable with C. So, if the value of A is turned from true to false at some moment of the execution of Fix(C), then Fix(A) is called.

By this observation, Solve makes at most m calls to Fix, *if* Fix(C) *always terminates.* So we need to show all the Fix(C) terminate. Suppose the algorithm makes at least s Fix calls including all the recursive ones. We will show that s is bounded by $O(m \log m)$, and thus the algorithm terminates in almost linear expected time.

This can be proved (at least at an intuitive level) by Kolmogorov complexity arguments. An extensive account concerning these arguments can be found in the book by Li and Vitányi (2008). Here we just describe this argument on an informal level.

The Kolmogorov complexity, $K(x)$, of a string x is the length of the string's shortest description in some fixed universal description language. Such a description language can be based on any programming language. If P is a program which outputs a string x, then P is a description of x. The length of the description is just the length of P as a character string. In determining the length of P, the lengths of any subroutines used in P must be accounted for. The length of any integer constant n which occurs in the program P is the number of bits required to represent n, that is (roughly) $\log_2 n$. For example, a huge string $x = 010101\cdots01$ with 01 repeated 2^{100} times can be described as a program "repeat 01 2^{100} times", whose length (after binary encoding) is only about 100 bits. In general, any string containing some repeating patterns has small Kolmogorov complexity. On the other hand, random strings are "resistant" against compression, and hence, have large Kolmogorov complexity.

It is straightforward to compute upper bounds for $K(x)$: simply compress the string x with some method, implement the corresponding decompresser in the chosen language, concatenate the decompresser to the compressed string, and measure the resulting string's length.

A string x is compressible by a number c if it has a description whose length does not exceed $|x| - c$, that is, if $K(x) \leq |x| - c$, where $|x|$ is the length of (number of symbols in) x. Otherwise x is incompressible by c. A string incompressible by 1 is said to be simply incompressible or *Kolmogorov random*; by the pigeonhole principle, incompressible strings must exist, since there are 2^n bit strings of length n but only $2^n - 1$ shorter strings, that is strings of length $n - 1$ or less.

For the same reason, "most" strings are complex in the sense that they cannot be significantly compressed: $K(x)$ is not much smaller than $|x|$, the length of x in bits. To make this precise, fix a value of n. There are 2^n binary strings of length n. Let x be a string chosen uniformly at random with probability 2^{-n}. It is easy to show that the probability that x *is* compressible by c is negligible: it is $2^{-c+1} - 2^{-n}$. To see this, it is enough to observe that the number of descriptions of length not exceeding $n - c$ is given by the geometric series: $1 + 2 + 2^2 + \cdots + 2^{n-c} = 2^{n-c+1} - 1$, and there remain at least $2^n - 2^{n-c+1} + 1$ binary strings of length n that are incompressible by c.

Now, the so-called *incompressibility argument* works as follows: In order to show that some condition holds, assume it does not hold and use this assumption to show that then some Kolmogorov random string x would have a description much shorter than $K(x)$.

After this short excursion into Kolmogorov complexity, let us return to Moser's algorithm. Fix a Kolmogorov random string x of length $n+sk$ (where n is the total number of variables) and assume the algorithm uses the first n bits as the initial assignment a, and k bits each to replace the variables in each Fix call. (If we choose the string x randomly then it will be Kolmogorov random with high probability.) The random string x is used in Step 1 of Fix(C) to replace the values of variables in C by "fresh" random values, and each time next k bits of x are used.

If we know which clause is being fixed, we know the clause is violated so we know all the bits of this clause and thus we learn k bits of x (recall that assignments used by an algorithm are from the string x). We then replace those bits with another part of x.

So we can describe x by the list of clauses we fix plus the remaining n bits of the final assignment. We can describe each clause C such that $\mathrm{Fix}(C)$ is called by Solve using $O(m \log m)$ bits. The remaining fixed clauses can be described by $\log_2 r + \alpha$ bits (for a constant α) because either it is one of r clauses that intersects the previous clause or we indicate the end of a recursive call (keeping track of the recursion stack). This is exactly the place where the compression comes: Since the clause was not satisfied, we reveal k bits of information about the string x, but since $r \leq 2^{k-c}$, we can describe this information using only $k - c$ bits. Since the string x was Kolmogorov random, we must have

$$O(m \log m) + s(\log r + \alpha) + n \geq n + sk$$

or $s(k - \log r - \alpha) \leq O(m \log m)$. Now, if $r \leq 2^{k-c}$ for $c > \alpha$, then $k - \log r - \alpha$ is a positive constant (not exceeding k, which is also a constant), implying that $s = O(m \log m)$.

Exercises

19.1. Let $A_1, \ldots, A_n$ be events, and suppose that each of them is mutually independent of all the other events except for at most d of them. Let $0 < \epsilon < 1$, and assume that

$$\Pr[A_i] \leq \frac{\epsilon}{n}\left(1 - \frac{\epsilon}{n}\right)^d$$

for all $i = 1, \ldots, n$ Prove that then $\Pr[\cap_i \overline{A_i}] \geq 1 - \epsilon$.

19.2. Let $\mathcal{F}$ be a k-uniform k-regular family, i.e., each set has k points and each point belongs to k sets. Let $k \geq 10$. Show that then at least one 2-coloring of points leaves no set of $\mathcal{F}$ monochromatic.

19.3. The van der Waerden number $W(2, k)$ is the least number n such that any coloring of $\{1, 2, \ldots, n\}$ in two colors gives a monochromatic arithmetic progression with k terms. Prove that $W(2, k) > 2^k/(2\mathrm{e}k)$. *Hint*: Assume that $n \leq 2^k/(2\mathrm{e}k)$ and observe that one progression with k terms intersects at most nk others.

19.4. (Erdős–Lovász 1975). Consider the colorings of real numbers in r colors. Say that a set of numbers is *multicolored* if it contains elements of all r colors. Fix a finite set X of real numbers, and let m be such that

$$4rm(m-1)\left(1 - \frac{1}{r}\right)^m < 1.$$

Using Lemma 19.1 prove that then, for any set S of m numbers there is an r-coloring under which every translate $x + S := \{x + y \,:\, y \in S\}$, with $x \in X$,