1 Randomized Algorithms for 3-SAT

Summary: 3-SAT is an NP-complete problem, so we do not expect to have a polynomial-time algorithm (deterministic or randomized) for it. Today's topic is on just trying to beat the brute-force 2^n-work algorithm of trying all possible solutions. We will discuss a randomized algorithm with work roughly 1.733^n, then we'll reduce that to 1.5^n, then we'll reduce that to 1.33^n. The current best bound is roughly 1.31^n. Note that this has to do with guarantees on worst-case instances. There are a number of heuristic strategies that work *extremely* well on instances that arise in many applications, but today's topic is not on those.

The 3-SAT problem: The 3-SAT problem is the following. You are given a 3-CNF formula (an AND of ORs, where each OR contains at most 3 literals) over n Boolean variables. Your goal is to find an assignment to the n variables that satisfies the formula, if one exists. For example, consider $n = 4$ and the formula:

$$(x_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee x_2 \vee x_4)(x_2 \vee \bar{x}_4)(x_3 \vee x_4)(x_1 \vee \bar{x}_3).$$

This formula has three satisfying assignments. Using bit-vector notation (1=true, 0=false), assignments 1110, 1101, and 1111 satisfy the formula, and the rest do not. If we add the clause $(\bar{x}_1 \vee \bar{x}_2)$ to it, getting:

$$(x_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee x_2 \vee x_4)(x_2 \vee \bar{x}_4)(x_3 \vee x_4)(x_1 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_2)$$

then the formula is unsatisfiable. We typically use m to denote the number of clauses. Note that $m = O(n^3)$.

Brute force: A brute-force algorithm is to try all 2^n possible assignments. This can be done in parallel so the span is not too large, but the work is $O(m2^n)$.

Randomization: We're going to look at randomized algorithms with low work that succeed with some tiny probability p. We'll then repeat them many times to boost up the success probability. Before going into the algorithms, let's look at the question of how many times we would have to repeat such an algorithm in order to have high probability that at least one of the runs was successful. Note that this repetition can be done in parallel so it doesn't really increase the span by much at all; the main issue is the total work.

Claim 1 *Suppose we have a randomized algorithm with some small probability p of success. Then $O(\frac{1}{p}\log n)$ repetitions of the algorithm are sufficient so that we have probability $\geq 1 - 1/n^{10}$ that at least one of the runs was successful.*

Proof: We will use the fact that $1 - p \leq e^{-p}$ for any real number p.[1] Now, suppose we repeat the algorithm t times. The probability that all repetitions fail equals $(1-p)^t \leq e^{-pt}$. If we plug

[1] This can be seen by plotting the two functions and is actually a really helpful fact to know that comes up in a lot of analyses.

in $t = \frac{10}{p} \ln n$ we get n^{-10}. So, the chance that at least one of the repetitions succeeds is at least $1 - n^{-10}$. ■

We now look at a randomized algorithm that beats the brute-force bound. This algorithm and analysis are due to Uwe Schöning. We will begin with a simpler version that gets a work bound of $poly(n) \times 1.733^n$, then improve this to $poly(n) \times 1.5^n$, then improve this to $poly(n) \times (4/3)^n$. In all this discussion we will assume there exists at least one satisfying assignment, and our goal will be to find a satisfying assignment with some probability p, which we will then boost using repetition.

Algorithm 1 Schöning's algorithm, version 1

1. Pick a random initial assignment x.
2. While there is at least one unsatisfied clause and have done this at most n times:
   (a) Pick an *arbitrary* unsatisfied clause c.
   (b) Flip the bit of a *random* x_i in the clause.

Analysis #1: Consider some arbitrary correct satisfying assignment α. Let A be the event that the initial assignment x agrees with α on at least $n/2$ variables. Note that $\Pr[A] \geq 1/2$ by symmetry.

Now, every iteration of Step 2 has at least a $1/3$ chance of increasing the agreement with α by 1 (can you see why?). So, the probability p that this procedure succeeds is at least

$$\Pr[A] \cdot \Pr[succeed|A] \geq \frac{1}{2} \left(\frac{1}{3}\right)^{n/2}.$$

Using our claim from before, to succeed with high probability requires only $O(\frac{1}{p} \log n)$ repetitions. Putting these together with the fact that each repetition runs in polynomial time, we get a total work of $poly(n) \times 3^{n/2} = O(1.733^n)$.

Analysis #2: We can get a better bound by keeping the same algorithm but just doing a less crude analysis. Again consider some arbitrary correct satisfying assignment α, but now let A_k be the event that the initial assignment x disagrees with α on exactly k variables. Note that

$$\Pr[A_k] = \binom{n}{k} 2^{-n}.$$

(Do you see why?) The probability p that we succeed is at least the probability that in Step 2 we make a beeline towards α (reducing disagreement by 1 on each step until we reach a satisfying assignment). This means we have:

$$p \geq \sum_{k=0}^{n} \binom{n}{k} 2^{-n} \left(\frac{1}{3}\right)^k = 2^{-n} \sum_{k=0}^{n} \binom{n}{k} \left(\frac{1}{3}\right)^k.$$

As a cute trick, we can see that the RHS above is exactly $2^{-n}(1 + \frac{1}{3})^n$. So, $p \geq (\frac{2}{3})^n$.

Putting this together with our claim and the fact that each repetition runs in polynomial time, we get a total work of $poly(n) \times (\frac{3}{2})^n$.

An improved algorithm: We now slightly modify the algorithm. The only change will be to run Step 2 for $3n$ iterations instead of n iterations.

Algorithm 2 Schöning's algorithm, version 2

---

1. Pick a random initial assignment x.
2. While there is at least one unsatisfied clause and have done this at most $3n$ times:
   (a) Pick an *arbitrary* unsatisfied clause c.
   (b) Flip the bit of a *random* x_i in the clause.

To analyze this algorithm, we consider the events A_k as before, but instead of looking at the chance of making a beeline towards α, we examine the chance of making at most k incorrect steps within our first $3k$ steps. Note that this also is sufficient to reach a satisfying assignment. The probability we make k or fewer incorrect steps within our first $3k$ steps is at least

$$\binom{3k}{k}(1/3)^{2k}(2/3)^k.$$

We can use Stirling's formula for approximating $n!$ plus some algebra to get that this quantity is at least $\frac{1}{\sqrt{10k}} \cdot 2^{-k}$ for all $k \geq 1$. Combining this with the probability of event A_k and summing over k we get a success probability:

$$\begin{aligned} p &\geq 2^{-n} + \sum_{k=1}^{n} \left(\frac{1}{\sqrt{10k}} \cdot 2^{-k} \right) \binom{n}{k} 2^{-n} \\ &\geq \frac{2^{-n}}{\sqrt{10n}} \sum_{k=0}^{n} \binom{n}{k} 2^{-k} \\ &= \frac{2^{-n}}{\sqrt{10n}} \left(1 + \frac{1}{2} \right)^n \\ &= \frac{1}{\sqrt{10n}} \left(\frac{3}{4} \right)^n. \end{aligned}$$

So, putting this together with our claim and the fact that each repetition runs in polynomial time, we get a total work of $poly(n) \times (\frac{4}{3})^n$.

Interestingly, this is pretty much the best worst-case guarantee known, with the current best bound being roughly $poly(n) \times 1.31^n$.