CS124 **Lecture 15**

15.1 2SAT

We begin by showing yet another possible way to solve the 2SAT problem. Recall that the input to 2SAT is a logical expression that is the conjunction (AND) of a set of clauses, where each clause is the disjunction (OR) of two literals. (A literal is either a Boolean variable or the negation of a Boolean variable.) For example, the following expression is an instance of 2SAT:

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1}).$$

A solution to an instance of a 2SAT formula is an assignment of the variables to the values T (true) and F (false) so that all the clauses are satisfied– that is, there is at least one true literal in each clause. For example, the assingment $x_1 = T, x_2 = F, x_3 = F, x_4 = T$ satisfies the 2SAT formula above.

Here is a simple randomized solution to the 2SAT problem. Start with some truth assignment, say by setting all the variables to false. Find some clause that is not yet satisfied. Randomly choose one the variables in that clause, say by flipping a coin, and change its value. Continue this process, until either all clauses are satisfied or you get tired of flipping coins.

In the example above, when we begin with all variables set to F, the clause $(x_1 \vee x_2)$ is not satisfied. So we might randomly choose to set x_1 to be T. In this case this would leave the clause $(x_4 \vee \overline{x_1})$ unsatisfied, so we would have to flip a variable in the clause, and so on.

Why would this algorithm tend to lead to a solution? Let us suppose that there is a solution, call it S. Suppose we keep track of the number of variables in our current assignment A that match S. Call this number k. We would like to get to the point where $k = n$, the number of variables in the formula, for then A would match the solution S. How does k evolve over time?

At each step, we choose a clause that is unsatisfied. Hence we know that A and S disagree on the value of at least one of the variables in this clause– if they agreed, the clause would have to be satisfied! If they disagree on both, then clearly changing either one of the values will increase k. If they disagree on the value one of the two variables, then with probability 1/2 we choose that variable and make increase k by 1; with probability 1/2 we choose the other variable and decrease k by 1.

Hence, in the worse case, k behaves like a *random walk*– it either goes up or down by 1, randomly. This leaves us with the following question: if we start k at 0, how many steps does it take (on average, or with high probability) for k to stumble all the way up to n, the number of variables?

We can check that the average amount of steps to walk (randomly) from 0 to n is just n^2. In fact, the average amount of time to walk from i to n is $n^2 - i^2$. Note that the time average time $T(i)$ to walk from i to n is given by:

$$\begin{aligned} T(n) &= 0 \\ T(i) &= \frac{T(i-1)}{2} + \frac{T(i+1)}{2} + 1, \; i \geq 1 \\ T(0) &= T(1) + 1. \end{aligned}$$

These equations completely determine $T(i)$, and our solution satisfies these equations!

Hence, on average, we will find a solution in at most n^2 steps. (We might do better– we might not start with all of our variables wrong, or we might have some moves where we must improve the number of matches!)

We can run our algorithm for say $100n^2$ steps, and report that no solution was found if none was found. This algorithm might return the wrong answer– there may be a truth assignment, and we have just been unlucky. But most of the time it will be right.