Assume that a CNF formula with n variables has k satisfying assignments. What is the maximum possible number of leaves in the recursion tree of a backtracking algorithm solving satisfiability of this formula?

- $2^n - k + 1$

Correct
That's right! Each leaf of the recursion tree corresponds to a partial assignment that either satisfies the formula or falsifies it. The algorithm stops if it finds a satisfying assignment. Hence, in the worst case it will first go through all $2^n - k$ falsifying assignments, then will find a satisfying assignment and halt.

- k
- 2^k
- $2^n - k$

3-SAT: Backtracking

Have a question? Discuss this lecture in the week forums. >

Interactive Transcript

Search Transcript | English

0:00
Welcome to the next lesson of coping with NP-Completeness model.

0:04
In which we are going to design exact algorithms for NP-complete problems. Exact algorithms are also called sometimes intelligent exhaustive search. As opposed to brute force search algorithm. Which to find an optimal solution, just enumerates all possible candidate solutions, and selects the best one. Instead, we are going to find an optimal solution

0:32
without actually going through all possible candidate solutions.

0:37
The first problem for which we illustrate such an algorithm is 3-satisfiability problem. Recall that the input of this problem consists of a CNF formula, and it is 3-satisfiability each clause contains at most three literals. So this formula is in three conjuctive normal form, in 3-CNF. And our goal is to find out whether it is possible to assign Boolean values to all the variables of this formula so, as to satisfy all the clauses. Let me quickly show you once again, two for example. So the first formula here is satisfiable, because we can set for example, the value 1 to x, y, and z. Or the value 1 to x, and the value 0 to y and z. And the second formula is unsatisfiable. We cannot set the values of the variables

x, y, and z to satisfy all these five clauses. And I used brute force search algorithm for these three satisfiability problem, does the following. It just goes through all possible candidate solutions that is through assignments, through all the variables of a formula and checks whether any of them satisfies the input formula. If n is the number of variables of an input formula F, then the running time of this algorithm is proportional to the length of F times 2 to the n.

2:03
Because we have 2 to the n assignments, and for all of them we need this kind of formula to check whether all of the is fired or not. In particular if your formula is unsatisfiable then your algorithm will be forced to check all these assignments. So it's running time will be 2 to the n. So our goal for this module is to avoid going through all possible 2 to the n assignments and still to check whether the input formula is satisfiable or not. The first technique that we are going to use. This is called backtracking. So this is a well known technique which proceeds as follows. You construct your solution piece by piece and then at each step when you realize that your current solution cannot be extended. Your current partial solution cannot be extended to a valid solution you backtrack, you go back. This idea is used in many situations, from many combinatorial optimization problems and for many puzzles, and games. So imagine, for example, that you are playing a game and you are staying in some room, and you are in a maze, and you are looking for an exit from a maze. And in the room that you are currently staying there are three doors leading out of this room. So you use one of these rooms and you enter another room for which you see that there are no other doors. So you are in a deadlock. So you then go back, you backtrack, and you decide to use some other room. So this is way of solving a combinatorial Problem, it is called backtracking once again. We will now show an example as usual how this technique named backtracking is used to check the stability of Boolean formulas. So consider the following formula, we need to find a satisfying assignment and we don't know what to do so let's try to assign zero to h1. What happens in this case is that

4:03
x1 disappears from this clause because it is already falsified.

4:09
On the other hand this clause is already satisfied so this clause disappears entirely. X1 also disappears from this clause and it becomes just x2 or negation of x3. Then x1 disappears also from this clause, and this clause is left untouched.

4:26
We then proceed as follows. Now let's try to assign the value 0 to x2. What happens is that, well this clause is satisfied, x2 disappears from this clause because it is falsified already from this clause and from this clause. So what we get is x3 or x4, and then the negation of x3, and the negation of x4. Then we try to assign the value 0 to x3. What is left, so this clause disappears, x3 disappears from this clause. So what is left is one clause containing x4, and one clause containing the negation of x4. We then try to assign the value to x4. If we assign the value 0 to x4, then we falsify this clause. So this clause disappears, and this clause becomes empty. Meaning that there is no satisfied literal in this clause, that there is no way to satisfy this clause anymore, right. The same happens if we assign the value 1 to x4. In this case, we satisfy this clause but at the same time, we falsify this clause. Once again, this clause becomes empty, which means that there is not way at this point to satisfy this clause.

5:39
So at this point we backtrack farther. We just realized that it was a wrong move to assign the value 0 to x3. It leads to an unsatisfiable formula. So we now try to assign the value 1 to x3. So now we returned back here, and we are trying to assign the value 1 to x3. But in this case we falsify this clause. This clause becomes 17 which means that the roll this formula is unsatisfiable. Which means that it was a wrong move to assign the value zero to x2, and now we need to try to assign the value 1 to x2. We do this, but we immediately falsify the following clause, right? So in all other clauses we have x2 so all of them are satisfied but what is left is an empty clause which cannot be satisfied. So at this point we backtrack back to the initial formula and at this point we know already that this we must assign the value one if we want to satisfy this formula. So we do this and see again there is an empty clause. So at this point we conclude that the initial formula is unsatisfiable.

6:56
Note the following important fact. So we realized that is unsatisfiable without considering all possible 16 through assignments.

7:12
To our four input variables, right? This is how the of the corresponding algorithm looks like. So we first check whether F is empty or not. So if F has no clauses and we have nothing to satisfy, then we just return that the formula is satisfiable. We then check if F is not empty, we check if F is an empty clause. As we've discussed before, an empty clause means a clause which cannot be satisfied. So all the literals in this clause have already been falsified. In this case we return that the formula is unsatisfiable. Otherwise we take some variable which is still present in the formula F. So x is some unassigned variable of F. And then we try two cases. We first try to assign the value 0 to x. So this notation means that we assign the value zero to x. When assigning the value zero to x we remove all the clauses which contain the negation effects. All such clauses already satisfied, and we remove all occurrences of x without negation from all other clauses. Because x is all ready falsified. If the formula turns out to be satisfiable, we return immediately that the real formula is satisfied.

8:31
Otherwise, we backtrack. We try to assign the value one to x. Again, so with F assigns the value one to x, meaning that we remove all the clauses that contain x from the formula. And we remove the x, negate it from all other clauses. If the recursive call returns that the formula is satisfiable we return that the initial formula is also satisfiable. In the remaining case we just report that the formula is unsatisfiable. Because no matter how we assign the value to x the resulting formula is unsatisfiable. Meaning that there is no way of

assigning the value to x to get a satisfiable formula. So as we've seen in our toy example such a strategy arose to find a satisfying assignment to conclude that the formula is unsatisfiable without actually checking all possible 2 to the n assignments, 2 to the n candidate solutions. I'm sorry. So we do this as follows. We build each solution piece by piece. We try to extend each partial solution and whenever we see that this is a deadend, that the current partial solution cannot be extended to a valid solution we backtrack immediately. We cut the correspondent branch of the tree and we do not extend it. So this is a technique which is used in many, many modern and state of the art SAT-solvers. We did not prove any upper bound on the running time of the algorithm. Well as you expect the running time of the algorithm might be exponential. Because if we prove that it is polynomial, we would show that could be solved in polynomial time. This would resolve the p versus np question. However, this technique is very useful in practice. However, in practice SAT-solvers also use complicated heuristics for simplifying formula and for selecting the next variable for branching and then in selecting the next value for branching. So in practice this idea improved by various heuristics of simplifying a formula chosen as the next variable and chosen as value for the next variable lead through very efficient algorithms that are able to solve formulas with thousands of variables. And other commonly used technique is called local search, and this is a technique that we will consider in the next part.

Downloads

Lecture Video	mp4
Subtitles (English)	WebVTT
Transcript (English)	txt