NP2: 3-SAT

Notes for CS-8803-GA: Introduction to Graduate Algorithms

Georgia Tech (Dr. Eric Vigoda), Fall 2017

as recorded by Brent Wagenseller

Overview

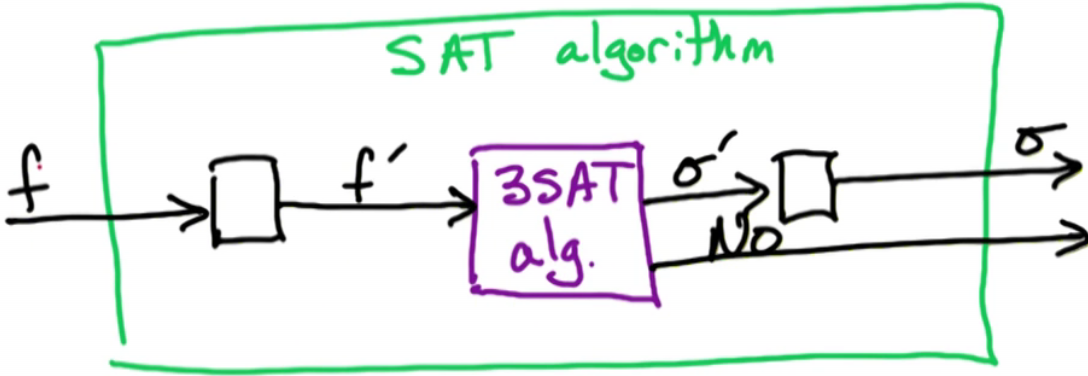
* 3SAT is NP-Complete; we are going to show this
* We are going to use the fact that SAT is NP-Complete to show that 3SAT is also complete
  + This method is known as the **Cook-Levin Theorem** (1971), and it showed that SAT was NP-Complete
  + BRENTS NOTE: Its possible SAT was the first algorithm shown to be NP-Complete, then the rest followed
  + Once it was shown that SAT was NP-Complete, this thought was used to show 21 other problems were also NP-Complete in 1972 (Karp)
    - 3SAT was one of these 21
* Recall 3SAT
  + Similar to SAT
    - Input is a Boolean formula f in **conjunctive normal form** (**CNF**) (where there are clauses that connect Booleans via OR in a single clause, and multiple clauses are joined with AND)
      * n variables are used and m clauses, where each clause has <= 3 Boolean literals
    - Output: satisfying assignment if one exists, otherwise NO
  + Difference from SAT is SAT used 2 Booleans in a clause, 3SAT uses 3
* Outline of proof that 3SAT is NP-Complete
  + Show 3SAT ∈ NP
  + Show SAT → 3SAT
    - Here we have to take a known NP-Complete problem and transform it to our target problem we are trying to show is NP-Complete (in this case its 3SAT)
    - Once we can show this, we can conclude ∀ A ∈ NP, A → 3SAT
    - The implication is if we have a polynomial time algorithm for 3SAT, we have one for every problem in NP as every problem in NP can be reduced to 3SAT

3SAT ∈ NP

* We must show that we can verify solutions efficiently to prove 3SAT ∈ NP
* Given 3SAT input f and a Boolean T/F assignment for {x1, …, xn}, we need to check if the variables satisfy the formula
* For each clause c ∈ f: in O(1) time we can check that at least one literal in C is satisfied
  + O(1) because every clause has at most 3 literals
  + Therefore, it tames O(m) time to verify if the Booleans satisfy the statement since there are m clauses
  + This shows 3SAT ∈ NP

SAT → 3SAT Setup

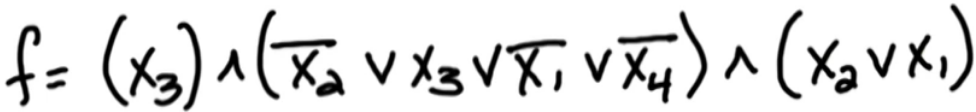
* We are going to try to reduce SAT to 3SAT
  + We are going to try to construct an algorithm for SAT using the 3SAT algorithm as a subroutine
* This completely relies on the assumption that there is, in fact, a 3SAT algorithm that exists and we can use it as a black box



* + We need to take input f from the SAT problem and convert it to input to the 3SAT problem f`
    - This may be tricky as f may have some big clauses which contain, potentially, n literals, but our input for 3SAT has to have clauses of size at most 3
    - These big clauses MUST be transformed into smaller clauses
    - This must be done in such a way that if we have a satisfying assignment σ` (this is sigma prime), which is a satisfying assignment for f` (our 3SAT formula), then we can transform this satisfying assignment σ for the original SAT input.
    - Its also critical that if 3SAT cannot solve the problem and outputs NO, f would ALSO be a NO for SAT
      * f` has no satisfying assignment if and only if f has no satisfying assignment
      * σ` satisfies f` if and only if σ satisfies f
        + σ` satisfies f` **⇔** σ satisfies f
      * σ` is a satisfying assignment for f` (the 3SAT input) if and only if the transformed output σ satisfies the original set formula f
        + Its critically important that ‘NO’ instances for the 3SAT formula correspond exactly to ‘NO’ instances for the SAT formula
    - Creating this f` seems to be the hardest task

Intuition for the reduction from SAT to 3SAT

* Our Example:



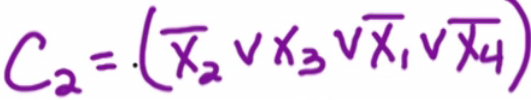
* + 4 variables and 3 clauses; its important that there are only 3 variables at most
* Clauses 1 and 3 we can keep the same, but clause 2 MUST be altered as we can ONLY have 3 variables
* How do we handle clause 2?
  + We need to break the second clause (denoted C2) into two clauses, and then create a new variable y
  + The new clause C2` will be



* + - The first two literals are joined with y in the first clause
    - The last two literals are joined with bar(y) in the second clause
  + The key claim is that C2 is satisfiable if and only if C2` is satisfiable
    - C2 is satisfiable ⇔ C2` is satisfiable
    - This means C2 and the pair of clauses in C2` makes an equivalent formula and there is at most 3 Booleans in each clause

Proving: C2 is satisfiable if and only if C2` is satisfiable

* Knowing this will allow us to have an idea about how to generalize the construction
  + Forward: Take satisfying assignment for C2 and construct a satisfying assignment for C2`
    - We are assuming C2 is satisfied
    - Take a satisfying assignment for C2:



* + - Find a satisfying collection of these Booleans and then show there is a satisfying assignment for C2`
    - In order for the assignment to satisfy C2, either x1 = F, x2 = T, x3 = F, or x4 = F
    - Break these possibilities into their respective breakdowns in the two new clauses in C2`:
      * If x2 = F or x3 = T: set y = F
        + If x2 = F or x3 = T, y does not matter so set y = F
        + Note that this WILL satisfy the other clause, as the other clause uses bar(y) so the second clause in C2` will work
        + So either case here will satisfy BOTH C2 AND C2` due to how we set up y
      * If x1 = F or x4 = F: set y = T
        + If x1 = F or x4 = F, y does not matter so set y = T (bar(y) will ultimately make this false so y will be a nonfactor)
        + Note that this WILL satisfy the other clause, as the other clause uses y so the first clause in C2` will work
        + Again, either case here will satisfy BOTH C2 AND C2` due to how we set up y
    - This setup shows that satisfying C2 also satisfies C2`
  + Reverse: Take a satisfying assignment for C2` and construct a satisfying assignment for C2



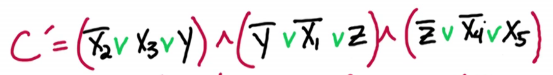
* + - We are assuming C2` is satisfied
    - If we ignore y, anything that satisfies x1, x2, x3, or x4 will satisfy C2
    - There will be two cases
      * If y = T:
        + We know that C2` is satisfied, so if y = T the first clause is satisfied; then either x1 must be false or x4 must be false, both of which would satisfy C2
      * If y = F:
        + We know that C2` is satisfied, so if y = F the second clause is satisfied; then either x2 must be false or x3 must be true, both of which would satisfy C2
      * Every satisfying assignment for C2` also satisfies C2
  + Since we have shown both directions, we have shown the if and only if statement σ` satisfies f` **⇔** σ satisfies f which is necessary for the proof

5 → 3

* We have done 4SAT to 3SAT (above), but what about 5SAT to 3SAT?
* Generally, you must introduce n variables to reduce k-SAT to 3SAT, where n = k – 3
* An example given in the quiz: Reduce the following to 3SAT:



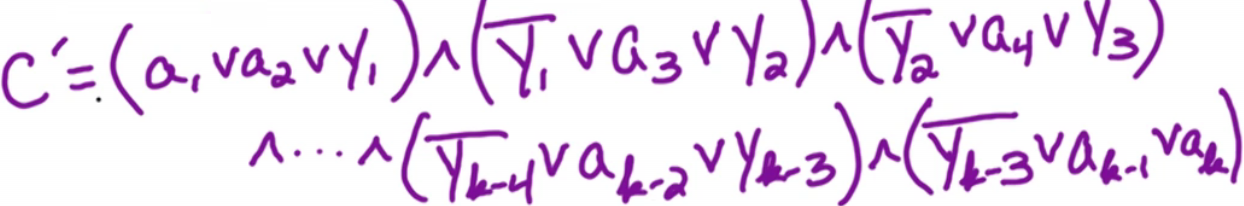
* + We need to keep in mind that C is satisfiable **⇔** C` is satisfiable
  + To do this, we will need 2 new variables (y, z) and 3 clauses:



* + - Notice that in the middle clause, we only have one original variable
    - The two new variables are compliments

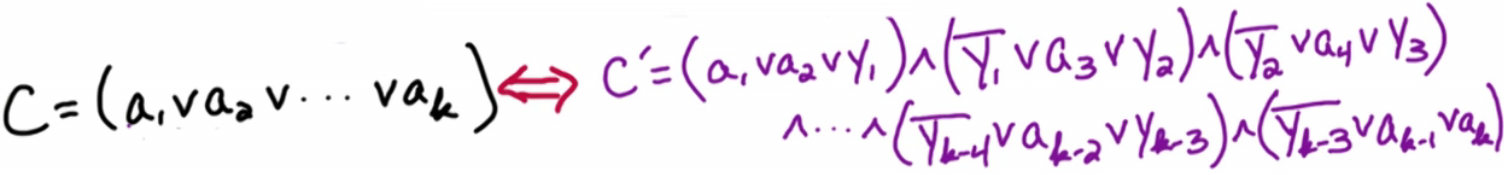
Big Clauses: reducing k-SAT to 3SAT

* Generalize and make a clause C = (a1 ∨ a2 ∨ … ∨ ak) where a1, a2, … ak are literals
* We will need to create k-3 new variables Y1, Y2, … Yk-3
  + Every clause >3 creates new variables, and they are distinct for each clause
    - This means that EVERY clause will get new, unique variables
      * These variables are known as **auxiliary variables**
    - Every clause may make n new variables; if there are m clauses, there will be n\*m new variables
  + For every clause >3, we will have k – 2 new clauses

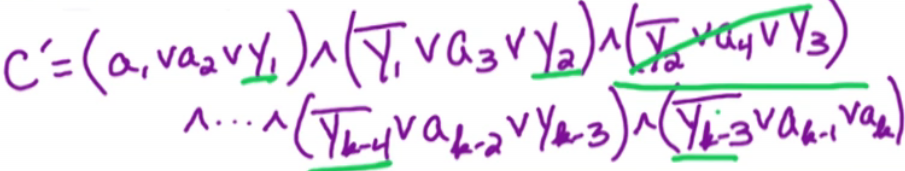


* + - Take the first two variables and the first Y in the first clause
    - Take the negative of the first Y, the third variable, and the positive of the second Y for the second clause
    - Keep doing this until there are 2 variables left, at which point the clause will be comprised of the negative of Yk-3 and then the last two variables
* C is satisfiable ⇔ C` is satisfiable

Proving k-SAT → 3SAT Generally



* We aim to prove the two above clauses are satisfiable if and only if the other clause is satisfiable
  + This can be abstracted to all clauses
* First prove the forward: Take satisfying assignment for C and construct a satisfying assignment for C`
  + We are assuming C is satisfied
  + One of the literals must be satisfiable in C; let ai be satisfiable
  + ai is the minimum i where ai is satisfied
  + Since ai is satisfied (ai = T), that will satisfy one of the clauses in C`
    - Matter of fact, satisfies the (i-1)st clause in C`
  + Now we can satisfy earlier clauses by setting earlier versions of Y to T
    - Matter of fact, this will be Y1 = Y2 = … = Yi-2 = T to satisfy the first (i - 2) clauses
  + For the clauses after (i-1), we can set Yi-1 = Yi = … = Yk-2 = F
  + The punchline is we use the literal ai to satisfy the (i-1) st clause, and we then use one set of auxiliary variables to satisfy the earlier clauses (setting =T) and the remaining set of auxiliary variables to satisfy the later clauses (setting = F)
  + We only need one literal of C to be satisfied; then we use the auxiliary variables to satisfy all other clauses of C`
  + Example how this could work if i = 4:



* + - ai = T
    - Y1 to Y2 (aka Yi-2) = T
    - Y3 (aka Yi-1) to Yk-2 = F
* Now to prove the reverse: Take satisfying assignment for C` and construct a satisfying assignment for C
  + We are assuming C` is satisfied
  + Since the auxiliary variables do not matter for C, all we need to do is show ONE variable is true in C`
  + Since we are assuming C` is satisfied, this means at least one ai = T
    - We need to show that the auxiliary variables do not matter
    - Suppose a1 = a2 = … = ak = F
    - This means that for the first clause to be true, the first auxiliary variable Y1 MUST be true
      * This means that bar(Y1) in the second clause will be false, but Y2 will save the clause so the entire clause will be true
      * Unfortunately, once we get to the last clause we will have only one auxiliary variable , and this will be false
        + since all variables in this last clause are false this means the clause is false which is not possible if we assumed C` is satisfied, therefore this is a contradiction

Therefore at least one ai MUST be set to true

* + Since we proved by contradiction that at least ONE ai MUST be set to true, so ai will satisfy the clause C

Returning to SAT → 3SAT

* Consider f for SAT
* Create input f` for 3SAT:
  + For each clause C ∈ f:
    - If |C| <= 3: add C to f`
      * If C has less than or equal to 3 literals, add it as-is to f`
    - If |C| > 3
      * We will do what we just did previously
      * Create k-3 new variables and add C` as defined before
  + Now that we have defined the problem, we need to prove



* + - This is straightforward to prove, given we already proved C is satisfiable if and only if C` is satisfiable
    - We still have to show that given a satisfying assignment to f`, we can construct a satisfying assignment to f
      * This is also straightforward, as we can ignore the auxiliary variables and the setting of the original variables will give us a satisfying assignment to the original formula
* Prove:



* + Start with the forward implication
    - Given assignment to x1, …., xn satisfying f
    - Show that by keeping these assignments the same, there is an assignment to the new variables so this new f` is satisfied
      * Since the assignments x1, …., xn are the same AND the new variables are distinct for each clause, we can look at this clause by clause
      * Lets examine clause C ∈ f
        + If C has at most 3 literals its obvious
        + If |C| > 3:

Replace C by the (k-2) clauses (as discussed earlier) and call them C`

C` uses (k-3) new variables AND they are distinct for C`

They do NOT appear elsewhere

As such, these variables can be set however we want wrt C and will not affect any other clauses

We have seen previously that there is an assignment to these (k-3) new variables so that C` is satisfied

* + Second, look at the reverse implication
    - Given satisfying assignment for f`, construct a satisfying assignment for f
    - Show the assignment wrt the original variables satisfies f
      * We can ignore the auxiliary variables in f`
    - Consider the sequence of clauses that correspond to C` in f
    - We previously showed that at least one of the literals in C MUST be satisfied
      * If all literals were false, there would be no way to satisfy C`, but satisfying C` was a requirement
      * This means that if C` is satisfied, C MUST be satisfied (As previously shown)

Satisfying Assignment

* So far we have shown
  + How to take an input formula f for SAT and transform it into f` for 3SAT
  + We then proved f is satisfiable if and only if f` is satisfiable
* Now, we have to convert σ` for f` to σ
  + We can simply take σ` and ignore the auxiliary variables
  + The assignment for the original variables gives us a satisfying assignment for f
* Our reduction is now complete!
* One other things to note: what is the size of f`?
  + f, our original input, has n variables and m clauses
  + in the worst case, we may make n new variables in each clause
    - It may have n\*m variables in the worst case
      * O(nm)
    - We are also replacing every clause with n clauses in the worst case
      * O(nm)
  + This means the size of f` is polynomial in terms of the size of f
    - If we have an algorithm that is polynomial in the size of f`, its still polynomial in the size of f

Practice Problems

* 8.3: Stingy SAT
* 8.8: Exact 4-SAT