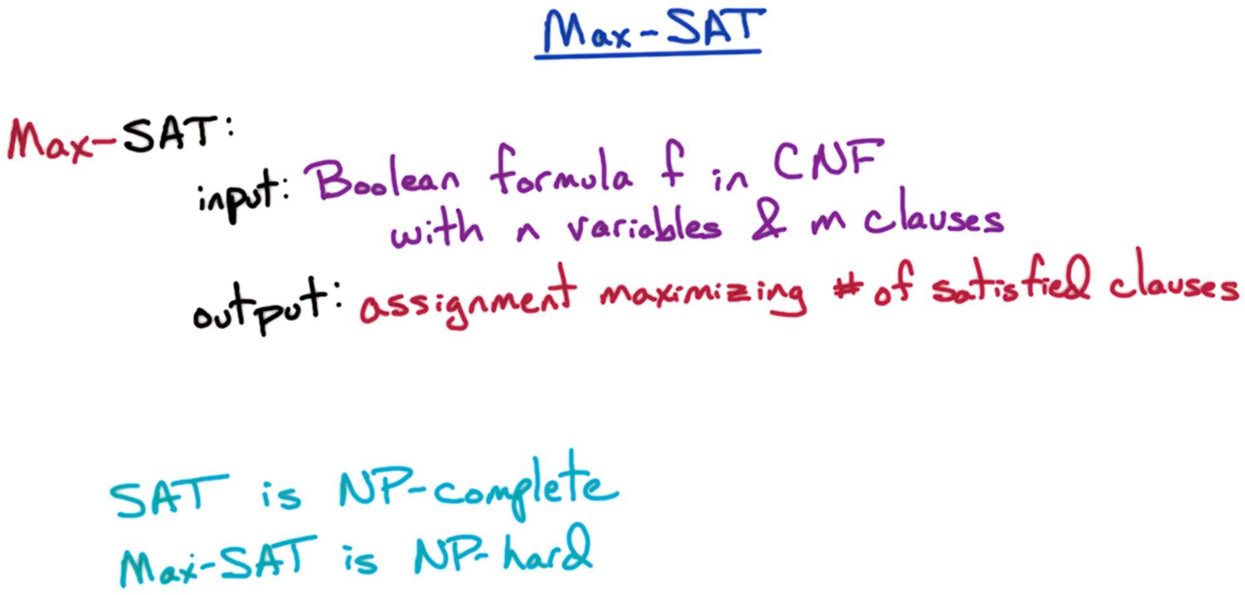
LESSON: LP4: Max-SAT Approximation

***

(Slide 1) Max-SAT



Max-SAT

Max-SAT:

input: Boolean formula f in CNF with n variables & m clauses

output: assignment maximizing # of satisfied clauses

SAT is NP-complete

Max-SAT is NP-hard

We've seen now, several times, the satisfiability problem.

- The input once again is a boolean formula in Conjunctive Normal Form with n variables and m clauses.
- And the output from the SAT problem is an assignment, a True/False assignment, for the n variables, so that the formula evaluates True or we simply output NO, if there's no such satisfying assignment.

Now, as we know the SAT problem is NP-complete, we can't hope to find a polynomial time algorithm for the SAT problem. Now, this is a search problem. Let's look at the optimization version.

It's the Max-SAT problem.

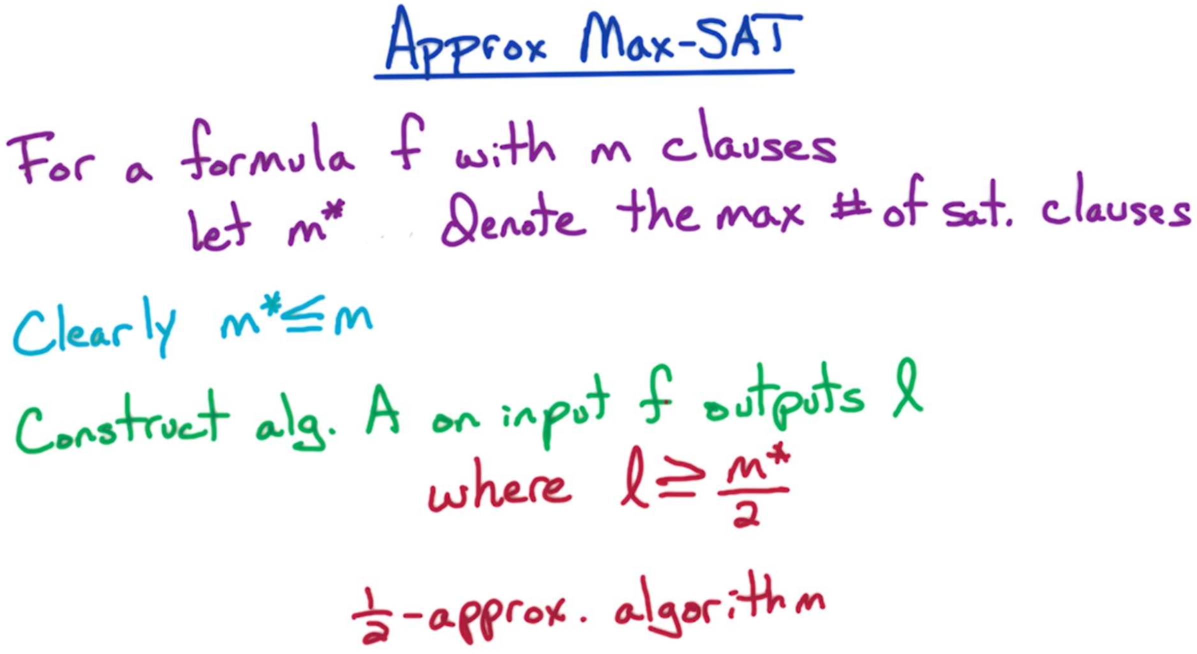
- Now, in the Max-SAT problem, the input is the same. It's a boolean formula in Conjunctive Normal Form. And, once again, will use n to denote the number of variables and m for the number of clauses.
- The difference is in terms of the output. Even if there's no satisfying assignment, we're still going to output an assignment. And here, we're going to output an assignment which maximizes the number of satisfied clauses.

In the optimization version of the SAT problem, we want to find an assignment which satisfies as many clauses as possible. Now, this is still a hard problem. In particular the Max-SAT problem is NP-hard. It's no longer a search problem – so, it is no longer in the class NP because we have no way of verifying that the number of clauses satisfied is maximum.

But, clearly, this Max-SAT problem is at least as hard as the SAT problem. It's straightforward to reduce SAT to Max-SAT and therefore Max-SAT is NP-hard. So once again, we can't hope to solve the Max-SAT problem in polynomial time. Instead, we're going to aim to approximate the Max-SAT problem and to do that we're going to use Linear Programming.

***

(Slide 2) Approximate Max-SAT



Approx Max-SAT

For a formula f with m clauses
let m^* Denote the max # of sat. clauses

Clearly $m^* \leq m$

Construct alg. A on input f outputs ℓ
where $\ell \geq \frac{m^*}{2}$

$\frac{1}{2}$-approx. algorithm

Let's look in detail at what we mean by an approximation algorithm for the Max-SAT problem.

Consider an input to the Max-SAT - a formula f where it has m clauses and let's let m* denote the maximum number of satisfied clauses. So if we look over all assignments to the variables in f, the maximum number of clauses satisfied by any of those assignments is m* and this is, of course, dependent on f, so, let's denote it as m*(f).

Now let's simplify the notation a little bit and let's drop this “of f” part and let's simply denote it as m*, which is the solution to the Max-SAT problem. Now clearly m* is at most m. If m* = m, then that means formula f is satisfiable.

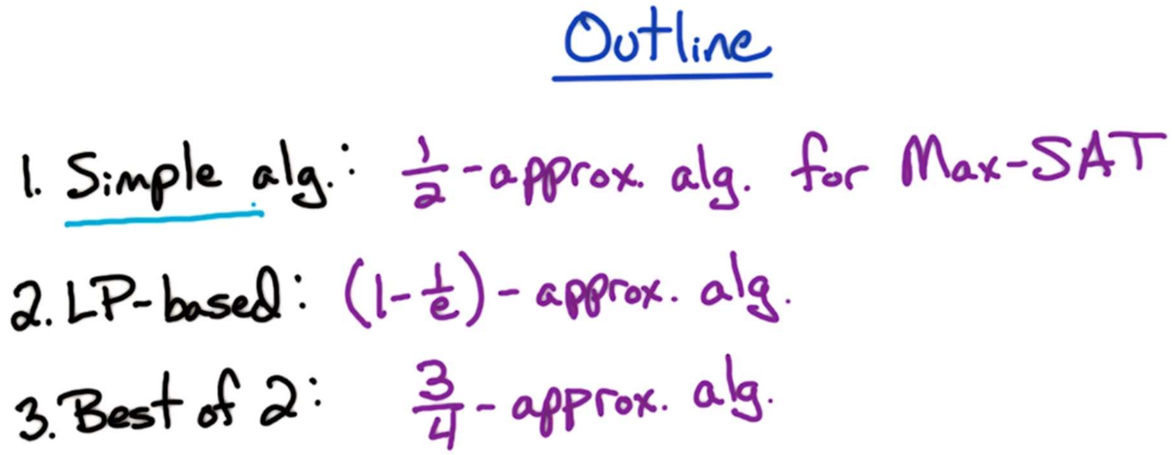
Now, we're going to construct an algorithm which we will denote it as A. It's going to take a formula f as input and it's going output ℓ (lowercase L) on this input f, and, to be precise, actually, its going to output an assignment which satisfies ℓ clauses of f. So, ℓ is the number of clauses satisfied by these assignments outputted by A.

Now how does l compare to the optimal solution m*? Well, in the first instance, we're going to guarantee that l is at least m*/2. Even though we don't know m*, we're going to guarantee that the output of our algorithm is at least within a factor of 1/2 of the optimal number of satisfied clauses. And if this holds for every formula f. Then, this is a 1/2 approximation algorithm.

And, then, later in this lecture, we're going to prove ½ to ¾ and we're going to get to ¾ approximation algorithm for the Max-SAT.

***

(Slide 3) Outline



Outline

1. Simple alg.: $\frac{1}{2}$-approx. alg. for Max-SAT
2. LP-based: $(1-\frac{1}{e})$-approx. alg.
3. Best of 2: $\frac{3}{4}$-approx. alg.

And we're going to start by looking at a very simple randomized algorithm, and we'll show that this simple algorithm achieves a one-half - approximation algorithm for Max-SAT.

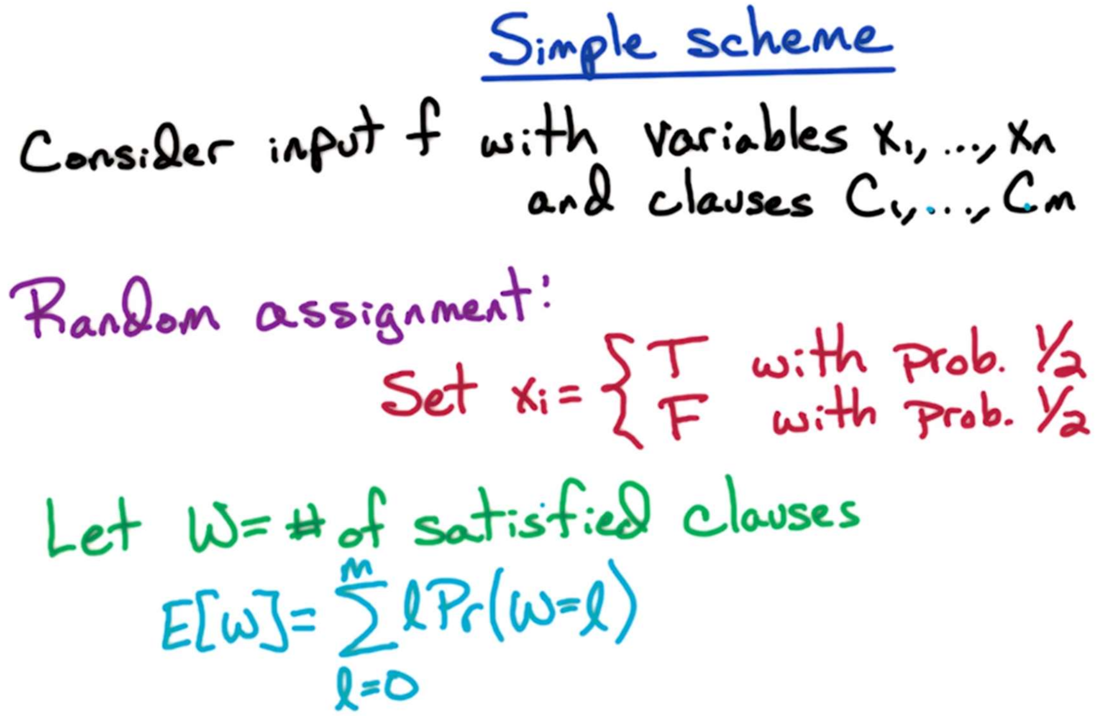
After that, we're going to look at an LP-based algorithm and we're going to prove that this new LP-based algorithm achieves a (1-1/e) - approximation algorithm, so to improve the previous algorithm by a little bit.

Finally, we're going to look at an algorithm which is a combination of these two algorithms, and this combined algorithm is going to achieve a ¾ - approximation.

So let's dive into the simple scheme.

***

(Slide 4) Simple Scheme



Simple scheme

Consider input f with variables $x_1, \ldots, x_n$
and clauses $C_1, \ldots, C_m$

Random assignment:

$$\text{Set } x_i = \begin{cases} T & \text{with Prob. } 1/2 \\ F & \text{with Prob. } 1/2 \end{cases}$$

Let $W = \#$ of satisfied clauses

$$E[W] = \sum_{\ell=0}^{m} \ell \Pr(W=\ell)$$

Now let's consider some input f to the max stack problem. And once again, let's denote the variables as x1 thru xn. So, there are n variables. And there are m clauses which we'll denote as C1 through Cm.

Now, we're going to do the simplest possible scheme for making a True/False assignment for these n variables. We're not even going to look at the formula f. We're simply going to assign these variables randomly to True or False.

For each variable xi, independently of the other ones, we're going to set xi to be True with probability half, and False with probably half. So, we simply flip a fair coin and if heads comes up, we set xi to be True; and, if tails comes up, we set xi to be False. And we do that for all n variables.

Now, how does this random assignment perform? We want to look at the expected performance of this random assignment. How do we measure the performance of the algorithm? Well, we measure the performance by looking at the number of clauses satisfied.

Therefore, let W denote the number of satisfied clauses by this random assignment. Now, since the assignment is random, W is also a random variable. Now, since W is a random variable, we want to look at its Expected value which in some sense is the average value of W. In particular, the Expectation is the average value of W weighted by its probabilities.

Now, what are the possible values for W? Well, the minimum number of clauses we can satisfy is of course zero and the maximum number is at most m. So let's sum over the possible values. Let ℓ (lowercase L) denote the number of satisfied clauses, and ℓ is going to vary between 0 and m. And then we get this value ℓ and we have to weight it by the probability that the random variable W is equal to ℓ. This is simply the definition of the expectation of a random variable.

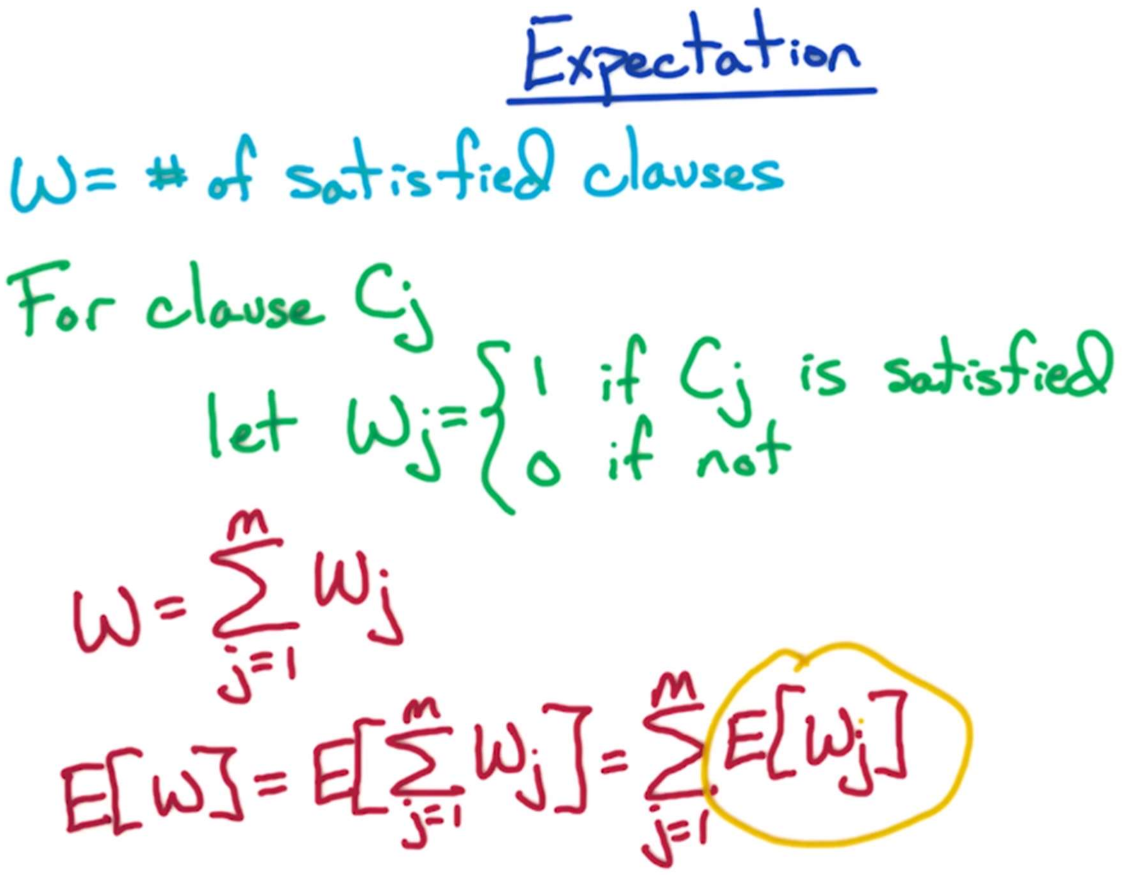
Now, the Expectation in this form is quite difficult to analyze. Why? Because whether a particular clause is satisfied or not is related to whether another clause is satisfied or not. Because they might have variables in common.

Now, what we're going to do is we're going to break up this random variable, which is the total number of satisfied clauses, into a clause-by-clause quantity. And in this manner we're going to be able to analyze each clause in isolation, independent of what happens for the other clauses.

And, in that way, it's going to be straightforward to analyze the Expected performance of the algorithm. So, let's look at how exactly we do that.

***

(Slide 5) Expectation



Expectation

W = # of satisfied clauses

For clause C_j

$$\text{let } W_j = \begin{cases} 1 & \text{if } C_j \text{ is satisfied} \\ 0 & \text{if not} \end{cases}$$

$$W = \sum_{j=1}^{m} W_j$$

$$E[W] = E\left[\sum_{j=1}^{m} W_j\right] = \sum_{j=1}^{m} \boxed{E[W_j]}$$

Once again, capital W denote the number of satisfied clauses for a random assignment. Each variable is assigned True or False with probability one half.

Now, we want to look at a simpler quantity. In particular, we want to look at one clause in isolation. So, consider the jth clause, Cj, and let's make a new random variable Wj.

Now, we're just talking about one clause – so, either that clause is satisfied or not. Therefore, this new random variable is going to take either value 1 or 0. It takes value 1 if the clause Cj is satisfied and it takes value 0 if it's not satisfied.

Now, what happens if we sum this quantity over all clauses? So, we look at the sum for j going from 1 to m of Wj - Σ Wj - where we get a 1 for every clause which is satisfied and 0 if it's not satisfied. So, this gives us a total count of the number of satisfied clauses. Therefore, it's equal to capital W.

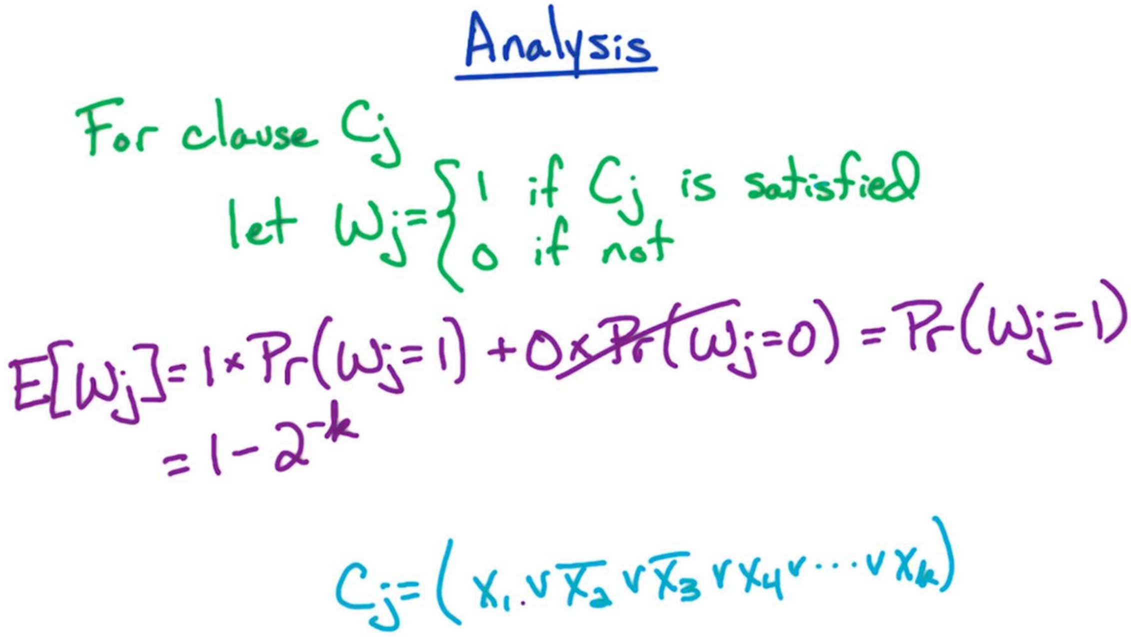
Now, recall that our original goal was to analyze the expected performance of the algorithm. So, we want to look at the Expectation of capital W, E[W], the expected number of satisfied clauses.

Now, we can plug in this identity, so we can replace capital W by this sum: E[W} = W[Σ Wj]. We have the expectation of the sum over j of Wj and now recall the linearity of expectations (*linearity of expectations: the expected value of the sum is equal to the sum of the expectations*) . So, we can take the sum outside. The expectation of two random variables x and y. Expectation of x + y is the same as expectation of x plus the expectation of y. So, now we have that the expectation of W equals the sum over j of the expectation of Wj.

Now, we can analyze this quantity E[Wj]. Notice, this just depends on the clause Cj. It doesn't have anything to do with the other clauses. So, now we can focus on this clause in isolation. We don't have to look at any of the other clauses.

***

(Slide 6) Analysis



Analysis

For clause C_j

let $W_j = \begin{cases} 1 & \text{if } C_j \text{ is satisfied} \\ 0 & \text{if not} \end{cases}$

$$E[W_j] = 1 \times Pr(W_j = 1) + 0 \cancel{\times Pr}(W_j = 0) = Pr(W_j = 1)$$
$$= 1 - 2^{-k}$$

$$C_j = (x_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4 \vee \cdots \vee x_k)$$

We're looking at this random variable Wj, which takes value 1 or 0 depending on whether the clause Cj is satisfied or not by the random assignment. We want to analyze the expectation of Wj.

So, we take the two possible values weighted by the probability of achieving that value. So, we have value 1 times the probability that Wj equals 1 plus 0 times the probability Wj equals zero. Then, clearly, the 0 term goes away and we're left with the probability Wj = 1.

This is the beauty of considering a random variable which takes value 0 or 1. The expectation is simply the probability that that random variable equals value 1.

Now, considering a sample of clause Cj, suppose this (x1 ∨ $\overline{x2}$ ∨ $\overline{x3}$ ∨ x4 ∨ ··· ∨ xk). So, the clause has size k. Now, in order to achieve value 1, the clause has to be satisfied. Let's look at the compliment - what's the probability the clause is not satisfied? Now, in order to not satisfy this clause, we have to set x1 to false, x2 to true, x3 to true and so on. That's exactly one setting of these variables - these k variables - so that this clause is not satisfied.

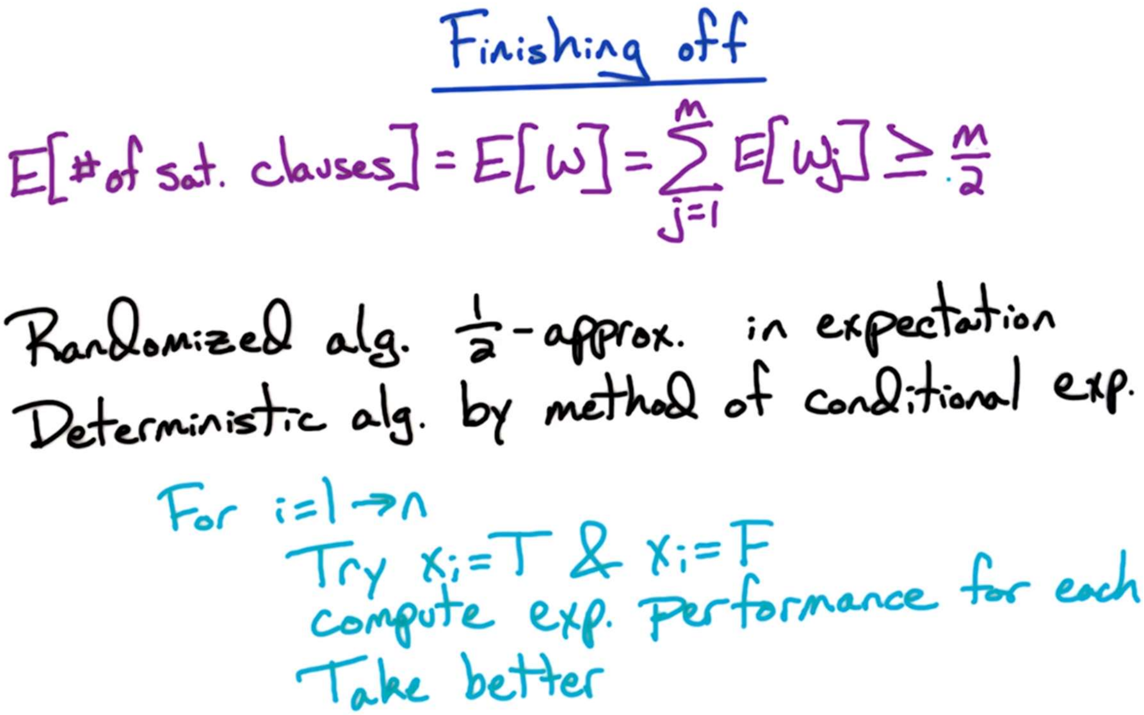
The chance x1 is set to false is one-half. The chance that x1 is set to false, and x2 is set to true, and x3 is set to true and so on - so these k variables are set exactly to this one assignment - the probability of that one assignment is 2^{-k}. That's the probability that this clause is not satisfied.

The probability that the clause is satisfied is a compliment of that, so it's $1 – 2^{-k}$. The probability that Wj = 1 is equal to $1 – 2^{-k}$. Now, notice something interesting, this gets better as k grows.

The worst case is when k = 1 - it's a unit clause. So, suppose the clause is just (x1), then the probability that this clause does not satisfy is one-half and the probability it is satisfied is one-half. So, this quantity is always at least one-half since k is at least 1.

***

(Slide 7) Finishing Off



Finishing off

$$E[\#\text{of sat. clauses}] = E[W] = \sum_{j=1}^{m} E[W_j] \geq \frac{m}{2}$$

Randomized alg. $\frac{1}{2}$-approx. in expectation

Deterministic alg. by method of conditional exp.

```
For $i=1 \rightarrow n$
    Try $x_i=T$ & $x_i=F$
    compute exp. performance for each
    Take better
```

Now, once again, we wanted to look at the expected number of satisfied clauses for this random assignment.

- We let W denote the random variable for the number of satisfied clauses,
- and we let Wj denote whether clause Cj is satisfied or not.
- And then this simplified to a sum over j, from 1 to M, of whether the expectation of whether clause Cj is satisfied or not.
- And then we just proved that this expectation - the probability that clause Cj is satisfied - is at least 1/2.

This is at least 1/2 for each of these m clauses. Therefore, this whole sum is at least m/2.

So, what we've shown is a randomized algorithm which achieves a ½ - approximation factor, though it's simply the expected performance of this algorithm.

Now, it turns out we can de-randomize this algorithm. We can find a deterministic algorithm which is guaranteed to find a 1/2 – approximation for the number of satisfied causes - and this uses the method of conditional expectations. It's quite simple, so let me quickly give you the idea of how it works.

So we're going to run through the variables one by one, so let's let i go from 1 to N. Now we're going to try the two possible settings for xi - True or False - and we're going to compute the expected performance for each. So, given this setting for xi, and given a fixed setting for x1 thru x_{i-1} - so, the early ones are fixed, and now we're trying both possible settings for XI - we compute the expected performance for a random assignment for the remaining variables, x_{i+1} thru xn.

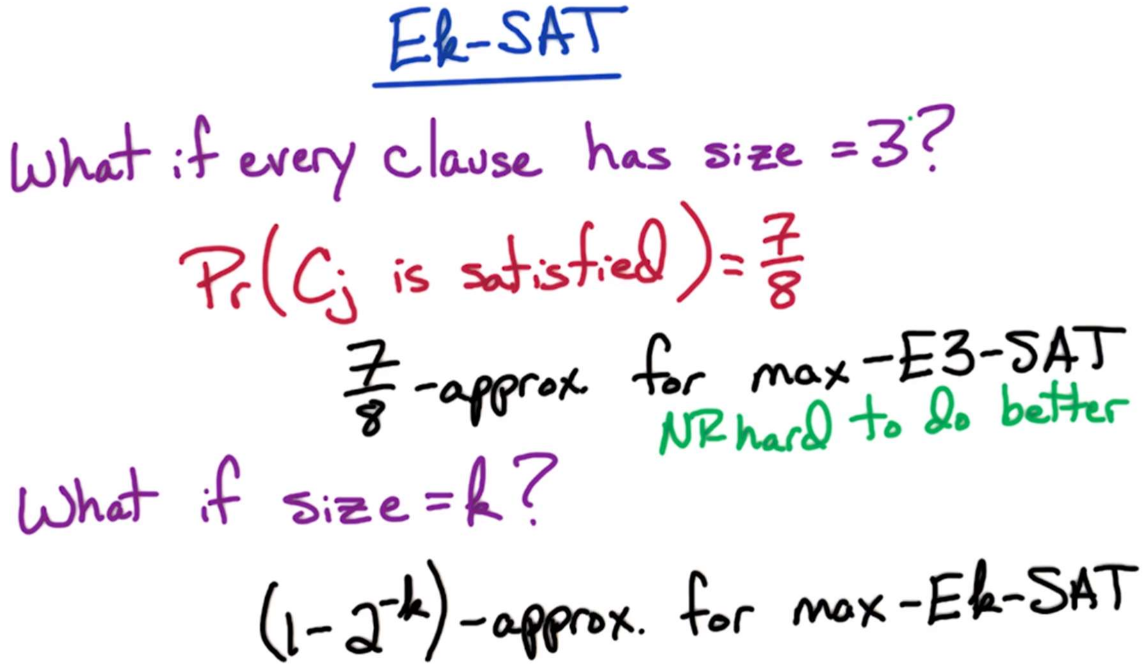
Now, this expected performance is quite easy to compute based on the analysis approach we just did. And, what we're going to do is we're going to take the better of these two assignments, and then we're going to fix it, and then we're going to move on to the next variable.

Now, one interesting feature is that we're not saying something about the optimal number of satisfied clauses. For instance, our formula f might have 12 clauses, and, maybe the maximum number of clauses that can be satisfied simultaneously is maybe 10. So m* in this instance would be 10, and m would be 12. Now, what we're proving is that a random assignment satisfies at least 6 of the clauses. Now, since that’s within a factor of 2 of the total number of clauses, therefore, we're within a half approximation of the maximum number of satisfied clauses.

And, actually, what was shown is that every formula has an assignment which satisfies at least half of the clauses. Intuitively, if the average is at least m/2, there must be at least one setting which achieves the average.

***

(Slide 8) Ek-SAT



Ek-SAT

What if every clause has size $=3$?

$\Pr(C_j \text{ is satisfied}) = \frac{7}{8}$

$\frac{7}{8}$-approx for max-E3-SAT

NP hard to do better

What if size $=k$?

$(1-2^{-k})$-approx. for max-Ek-SAT

There's one important point that we have to make about the performance of the algorithm which will be useful later.

Instead of Max-SAT, let's consider max-Ek-SAT. So, every clause has size exactly k. For instance, let's consider k=3, and suppose every clause in our input formula f had size exactly 3.

Now. in this case, what's the probability that a particular clause Cj is satisfied? Well, there's one setting of the three literals which appear in this clause so that this clause is not satisfied. The probability of that assignment is one-eighth. Therefore, the probability that it is satisfied is exactly seven-eighths; and, therefore for the special case of max-E3-SAT, we achieve a 7/8 - approximation algorithm.

And, what if, instead of size 3, every clause had size exactly k? Then, the probability that a specific clause is not satisfied is 2^{-k}. So, the probability that it is satisfied is $1 - 2^{-k}$, and, therefore, we achieve a $(1 - 2^{-k})$ - approximation algorithm for max-Ek-SAT. We're going to use later that this algorithm works well when all the clauses are large.

And, now, we're going to make a LP-based algorithm which works well when the clauses are small. Now, one interesting tidbit is that this seven-eighths approximation algorithm for max-

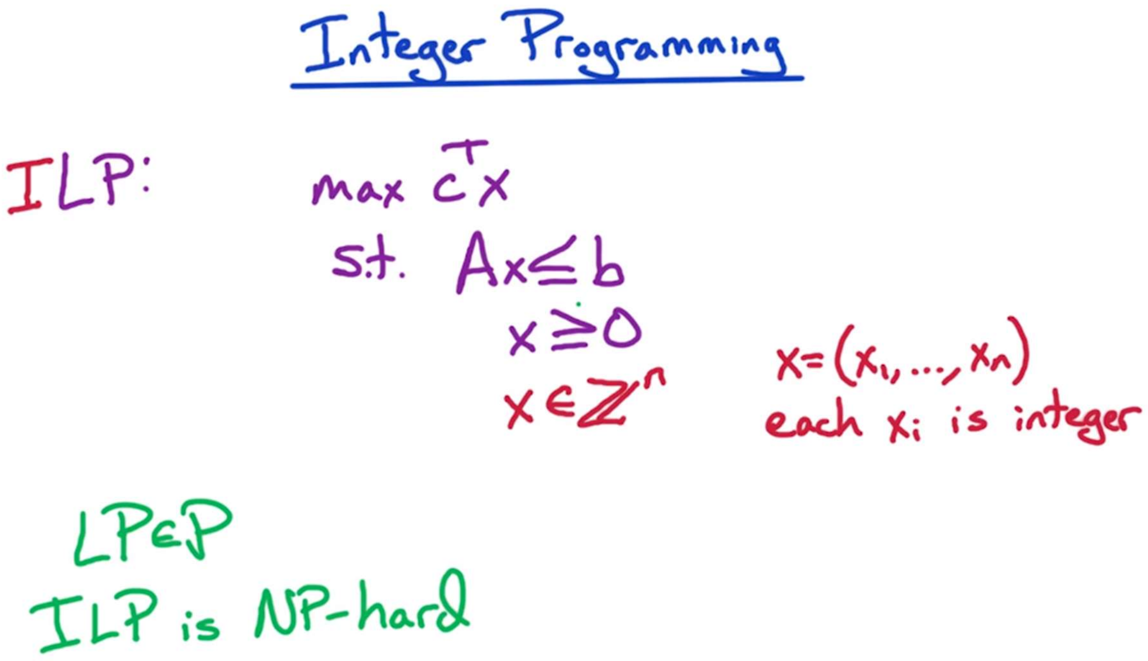
E3-SAT is the best possible. Hovstad proved that it's NP-hard to do any better than seven-eighths for this case.

If we achieve an algorithm which has guaranteed performance 7/8 + ε (epsilon), for any epsilon ε - for instance, if we achieve a 0.88 approximation algorithm, then that implies that P=NP. So, this very simple naive algorithm is the best possible when all the causes are of size exactly 3.

Thus, the hard case is when the formula has varying size clauses - has some unit clauses, some clause of size two, and some clauses of size three and so on. But, if all the clauses are of the same size and they happen to be of size three, then we can achieve the best possible algorithm by just a random assignment.

***

(Slide 9) Integer Programming



Integer Programming

ILP: $\max c^T x$

s.t. $Ax \leq b$

$x \geq 0$

$x \in \mathbb{Z}^n$

$x = (x_1, \dots, x_n)$

each x_i is integer

LP∈P

ILP is NP-hard

Our second approach: we'll use Linear programming to get an approximation algorithm for Max-SAT. To do that, we'll use a stronger form of Linear programming known as Integer Linear Programming.

Before we get into that, first, let's recall the general form of Linear Programming. It's a linear program in canonical form: For the variable vector x, we're maximizing c^T x, subject to the following constraints: Ax ≤ b, and x is non-negative. This is the canonical form for Linear program.

Now, in an Integer Linear Program, so in ILP, it has the same canonical form with one additional constraint, X is constrained to be $\in Z^n$, where Z are the integers. So, if you think of the vector x of size n, this constraint is saying that each xi is an integer value.

So, what's going on geometrically? In the Linear program, we had a feasible region - this convex set, defined by these constraints, and we're trying to find the best real numbered point x in that feasible set, so that we maximize this objective function – but, we consider all real numbered points in that set x.

Now, we're placing this grid - this n-dimensional grid - and we're only looking at the grid points contained in that feasible set. Those are the only feasible points now. We want to find the best grid point which maximizes this objective function.

Linear program had a nice property that there always is a vertex of the feasible region which is an optimal point which maximizes the objective function. We no longer have such a property for Integer Linear Programming. And, whereas linear programming is polynomial time solvable - so it lies in the class P - in contrast, integer linear programming is NP-hard, and we're going to see that right now.

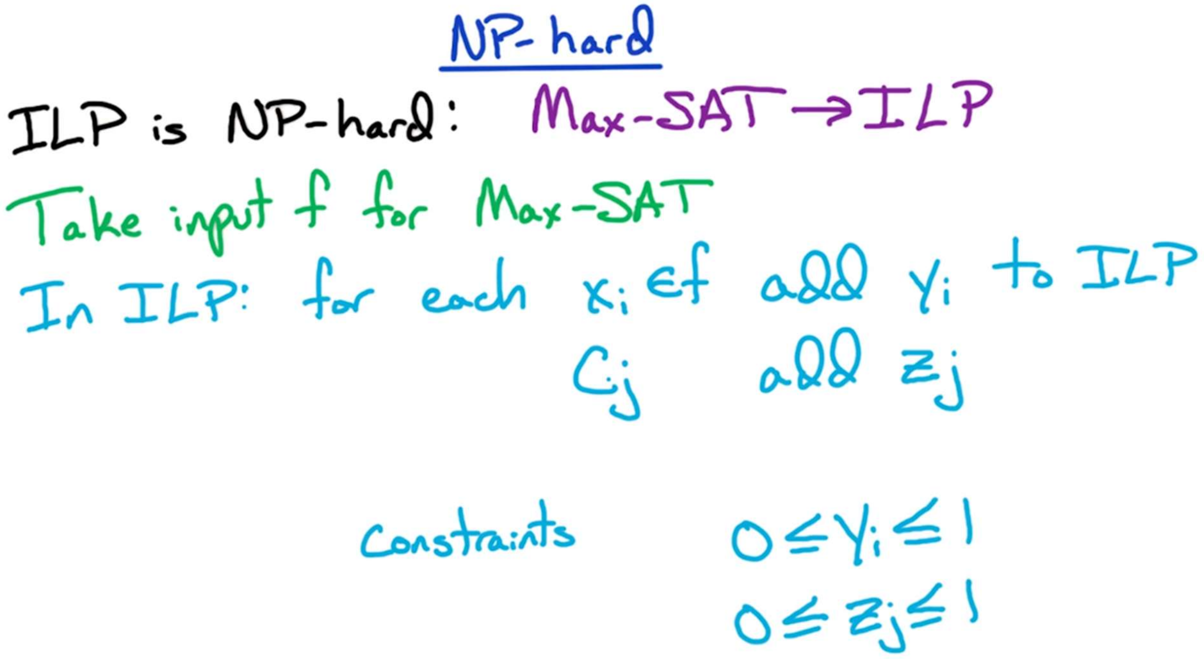
We're going to see how to reduce Max-SAT to Integer Linear Programming. In fact, many of the NP-complete problems that we've seen so far such as Vertex cover, Independent set, and so on, are easy to reduce to Integer Linear Programming. So, it's a quite powerful technique.

Well, since ILP, Integer Lienear Programming, is NP-hard, we can't expect to solve it in polynomial time. So, what's our game plan? Well, first, we're going to see how to reduce SAT or Max-SAT to Integer Linear Programming, then we're going to look at the Linear Programming relaxation – So, we're going to ignore this one constraint - this integer value constraint. So, we're going to look at the best real number point x. Of course, the objective function might go up. We're going to use this real number point x which is the optimal solution to the Linear Program to find an integer point which is nearby.

That's going to give us a feasible solution to the Integer Linear Program, and then we'll see how far away it is from the optimal solution. And that will give us our approximation algorithm to the Max-SAT

***

(Slide 10) NP-Hard



NP-hard

ILP is NP-hard: Max-SAT → ILP

Take input f for Max-SAT

In ILP: for each $x_i \in f$ add y_i to ILP

C_j add z_j

Constraints $0 \le y_i \le 1$

$0 \le z_j \le 1$

We're going to prove now that Integer Linear Programming is NP-hard. To do that we're going to reduce the Max-SAT problem to Integer Linear Programming.

Consider an input f for the Max-SAT problem and let's say that f has n variables and m constraints.

- In the integer linear program that we create, what are the variables going to be? Well, for each variable in the formula f, we can add a variable yi to the integer linear program that we create.
- And for each clause Cj, we're going to add a variable zj.
- So our initial linear program is going to have n + m variables and we're going to add the constraints that the yi's are between 0 and 1, and also each zj is between 0 and 1.
- And since these variables are constrained to be integer value, they'll either receive value 1 or 0. It can't receive any fractional value.

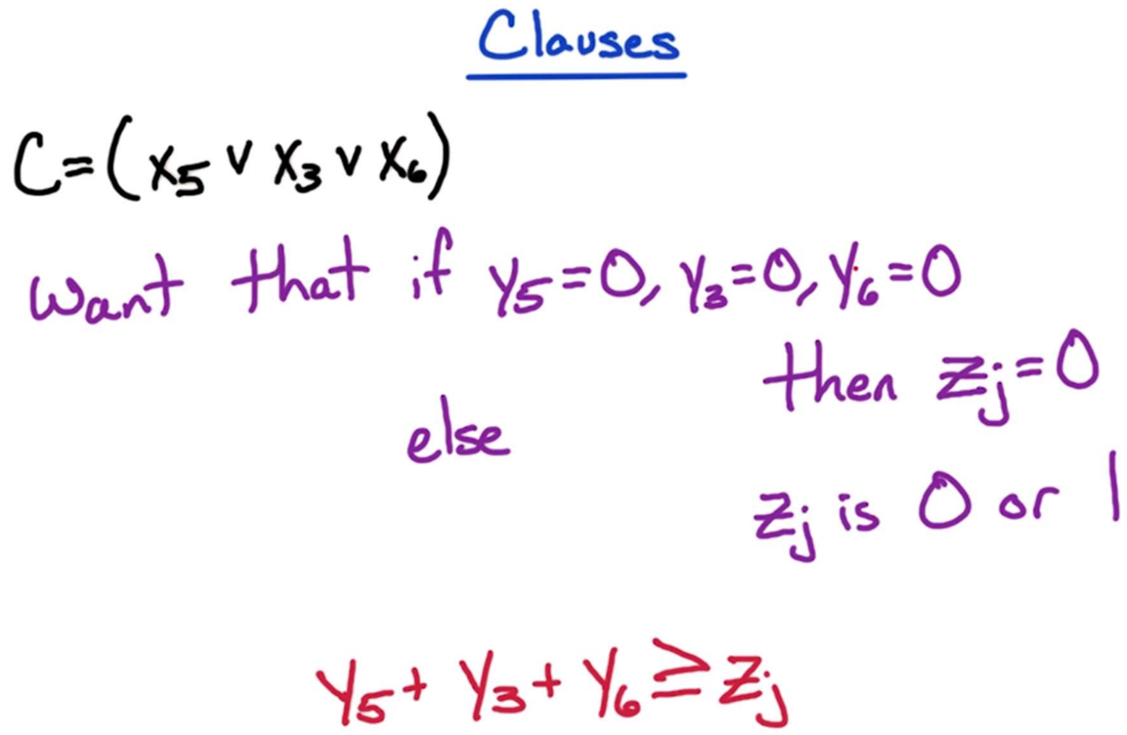
Now, intuitively what's going on?

- These y's are going to correspond to whether this variable xi is set to True or False. So, yi equals one corresponds to xi being True. xi being false corresponds to yi being 0.
- zj is going to correspond to whether this clause is satisfied or not. So, it's going to take value 1, if this clause is satisfied by the literals appearing in it and it's gonna take value 0, if this clause is not satisfied.

That's the intuition.

***

(Slide 11) Clauses



Clauses

$C = (x_5 \vee x_3 \vee x_6)$

want that if $y_5 = 0, y_3 = 0, y_6 = 0$

then $z_j = 0$

else

z_j is 0 or 1

$y_5 + y_3 + y_6 \geq z_j$

We're going to have a constraint for every clause. Let's take an example clause to get some idea of what we want to achieve.

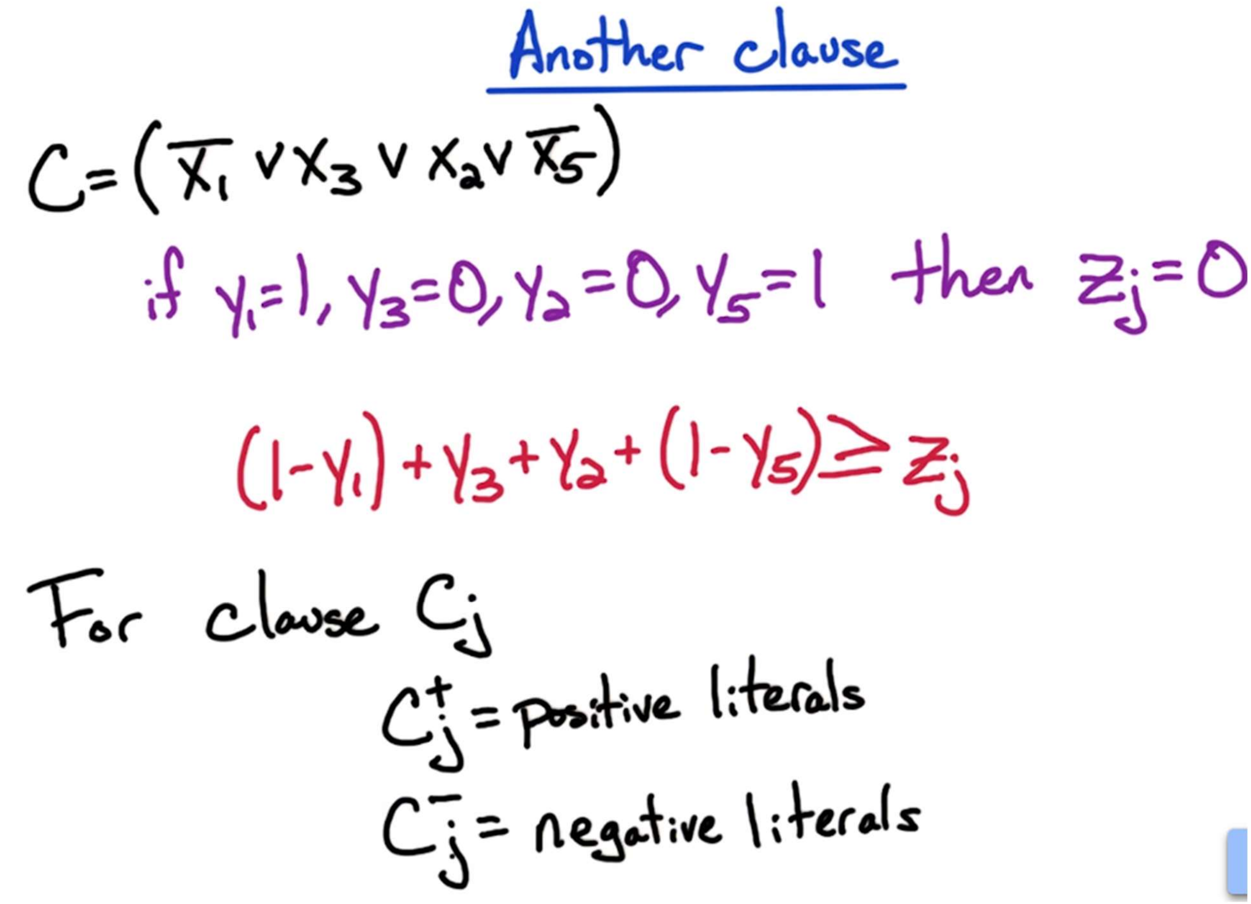
Consider this clause (x5 ∨ x3 ∨ x6). We made all the literals positive in order to make it a simple example. Now, what do we want from our constraint? What we want that if these variables are set to False - now, setting X5 to false is going to correspond to setting y5 to 0, and y3 to 0, and finally, y6 to 0. So, if these three variables are set to 0, which are going to correspond to setting these variables to False, then this clause is not satisfied. Then, we want the variable for this clause to be value 0. So, when this clause, this jth clause is not satisfied, we want these variables zj to be constrained to be value 0.

What if it's another setting? Well, then, this clause (x5 ∨ x3 ∨ x6) is satisfied. In this case, zj will be value 0 or 1 - We can't force it to be value 1 - but what we can do is we can try to maximize these zj's. that'll be our objective function, and then the optimal point will take value 1. So if it can achieve value 1, then we will.

In summary, the constraint we need to add is that if all three of these variables are 0, then zj is 0. How do we achieve that? We say that zj is at most y5 + y3 + y6. So if these three variables are 0, then zj must be 0. Now, if at least one of these is 1, then zj can take values 0 or 1. Recall that these variables are constrained between 0 and 1, and also these Y's are constrained to be 0 or 1.

***

(Slide 12) Another Clause



Another clause

$C = (\overline{X_1} \vee X_3 \vee X_2 \vee \overline{X_5})$

if $y_1=1, y_3=0, y_2=0, y_5=1$ then $z_j=0$

$(1-y_1)+y_3+y_2+(1-y_5) \geq z_j$

For clause C_j

C_j^+ = positive literals

C_j^- = negative literals

Take a look at another clause that has some positive and negative literals.

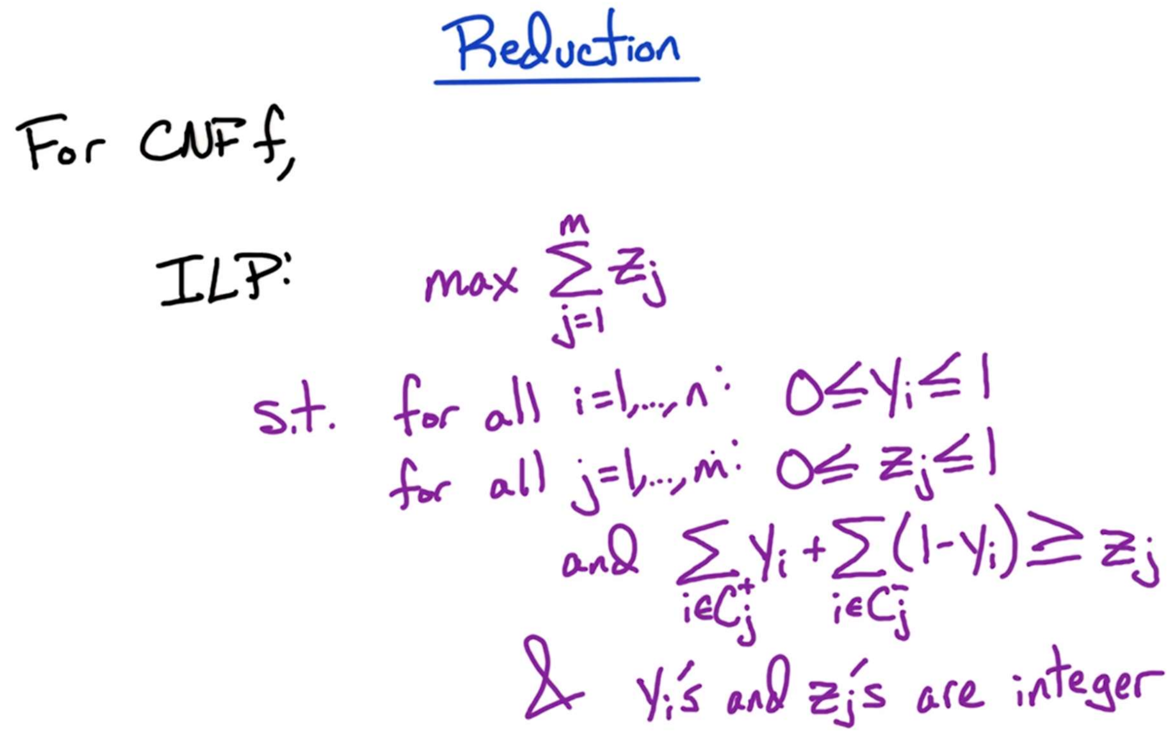
Consider this clause of size four: ($\overline{x1} \vee x3 \vee x2 \vee \overline{x5}$). Now, we want, that if x1 is set to True, then x3 is false, x2 is false, and x5 is True which is going to correspond to y1 being 1, y3 and y2 being value 0 and y5 being value 1. Then, we want that the variable z for this clause - let's say it's clause j - is forced to take value 0 which corresponds to this clause being unsatisfied.

Well, previously when we had just had the positive form of these literals, then we looked at the sum of the y's appearing. Now, we're going to look at the complement for these negative literals. So, we're going to look at (1 – y1) + y3 + y2 (because these appear in positive form) + (1 – y5) (because it appears as a negative form), and we're going to constrain zj to be at most this quantity. So, if this setting (y1 = 1, y3 = 0, y2 = 0, y5 =1) is true, then zj must be zero. For any other setting of these four y's, then zj can be 0 or 1.

And, in general, for clause Cj, we have to consider which literals are positive form and which literals are in negative form. So, let's use Cj^+ and Cj^- (superscript minus). Cj^+ will, of course, be the positive literals in the jth clause - in this case, it's x2 and x3. And, Cj^- will be the negative literals - in this case, x1 and x5.

***

(Slide 13) Reduction



Reduction

For CNF f,

ILP: $\max \sum_{j=1}^{m} z_j$

s.t. for all $i=1,\dots,n$: $0 \le y_i \le 1$

for all $j=1,\dots,m$: $0 \le z_j \le 1$

and $\sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1-y_i) \ge z_j$

& y_i's and z_j's are integer

Now, we can look at the general reduction.

Consider Max-SAT input as little f. Let's define the ILP.

- Our goal is to maximize the number of satisfied clauses, so we have a maximization problem.

  For each clause we have a variable zj, which denotes whether the clause is satisfied or not. It's going to take value 1 if the clause is satisfied, and 0 if it's unsatisfied.

  So, since we want to maximize the number of satisfied clauses, that means we want to maximize the number of clauses where the variables zj equals 1. This sum will be the number of satisfied clauses.

Now what are the constraints? Recall that we have a variable for each variable in the formula. And we also have a variable in the ILP for each clause in the formula.

- So, let's go over the variables in the formula first: So, for i going from 1 to n, we're going to constrain yi to be between 0 and 1. And since this is an integer value, it takes value 0 or 1.

- Now, for each clause, similar constraint - zj takes value 0 or 1

- And, then, we have the constraint for the clause that we just reviewed.
    - So, for Clause zj, we look at the positive literals. So, for each variable which appears in this clause in the positive form (the bad case is when that corresponding variable yi is set to 0; that means that literal is not satisfied).
    - Similarly, for each variable which appears in the negative literal, it's unsatisfied when yi takes value 1 - which means (1 – yi) has value 0. And we're going to impose this as an upper bound on zj.

      So if all the literals in this clause are unsatisfied then zj is forced to take value 0, so this clause is unsatisfied.

      And if at least one literal is satisfied and then Zj can take values 0 or 1. And since we're maximizing, zj will take value 1, if it can.

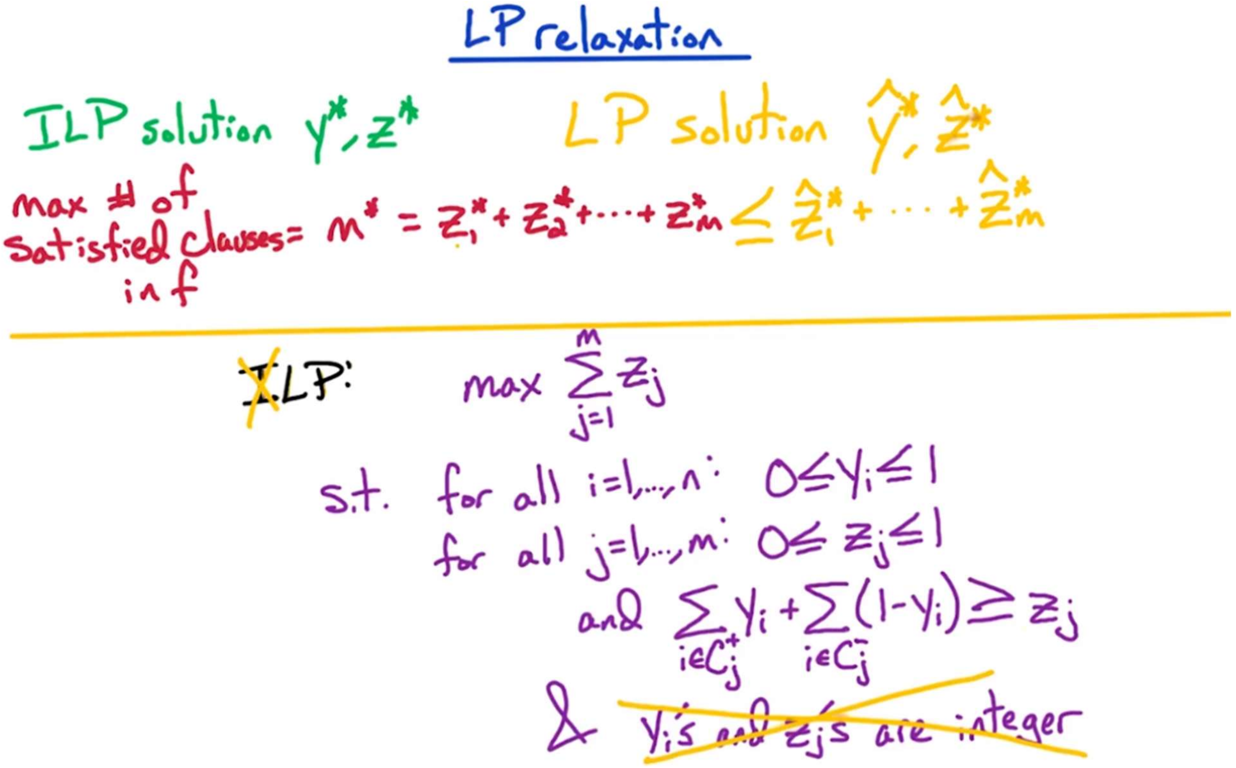
- Finally, we have the constraint that these variables, the yi's and the zj's are integer value. So they lie on this n-dimensional grid.

  Actually, in this case, we're in (n+m) dimensions.

And that defines our reduction from Max-SAT to integer linear programming. In fact, it's equivalent; so, this integer linear program is equivalent to the original Max-SAT problem on this in-

***

(Slide 14) LP Relaxation



LP relaxation

ILP solution y^*, z^* LP solution $\hat{y}^*, \hat{z}^*$

max # of satisfied clauses in f = $m^* = z_1^* + z_2^* + \cdots + z_m^* \leq \hat{z}_1^* + \cdots + \hat{z}_m^*$

~~I~~LP: max $\sum_{j=1}^{m} z_j$

s.t. for all $i=1,\ldots,n$: $0 \leq y_i \leq 1$

for all $j=1,\ldots,m$: $0 \leq z_j \leq 1$

and $\sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1-y_i) \geq z_j$

& ~~y_i's and z_j's are integer~~

Take this ILP - this integer linear program that we just defined - and consider an optimal point for this ILP. Of course, we don't know how to find this optimal point in polynomial time, and we're not in fact finding it. We're just considering this optimal point for the purposes of the proof.

So let y* and z* denote the optimal point for this ILP. Now, what's the value of the objective function at the optimal point? - where the value of the objective function is simply the sum of the z*'s? What do we know about the value of the objective function at this optimal point? Well, it equals m*.

What is m*? m* is the maximum number of satisfied clauses in the original formula f. m* is what we're trying to find and we can't solve this ILP in polynomial time, but we can solve any linear program in polynomial time.

Can we convert this ILP into a similar LP? Well, this last constraint is what makes this an ILP; so, let's drop this last constraint. So, now, the feasible points are no longer constrained to be an integer value and then this changes from an ILP to a linear program - this is simply a linear program now.

We can solve this linear program in polynomial time, and, let's denote the solution to this linear program instead of by y* and z* - let's put a hat there - so, $\hat{y}$* (y-hat-star) and $\hat{z}$* (z-hat-star) are the optimal solution to this linear program - these points, we can find.

What's the relation between the objective function for the linear program compared to the ILP objective function at the optimal point? Well, the value of the objective function for this LP at this optimal point is the sum of the z-hat-stars. How do these compare? Well, any feasible point for the ILP is also a feasible point for the LP. Because the ILP constrains it to lie in this grid; the LP considers any point. So, the LP has considered more points - so, since the ILP optimal point is also feasible for the LP, and the optimal solution for the LP is at least as good as the optimal solution for the ILP - So, we know that the objective function value is at least this objective function value for the ILP optimal solution.

So, we can find an upperbound on the maximum number of satisfied clauses in f. Does that do us any good? Well, not necessarily - that doesn't help us find m* in any way.

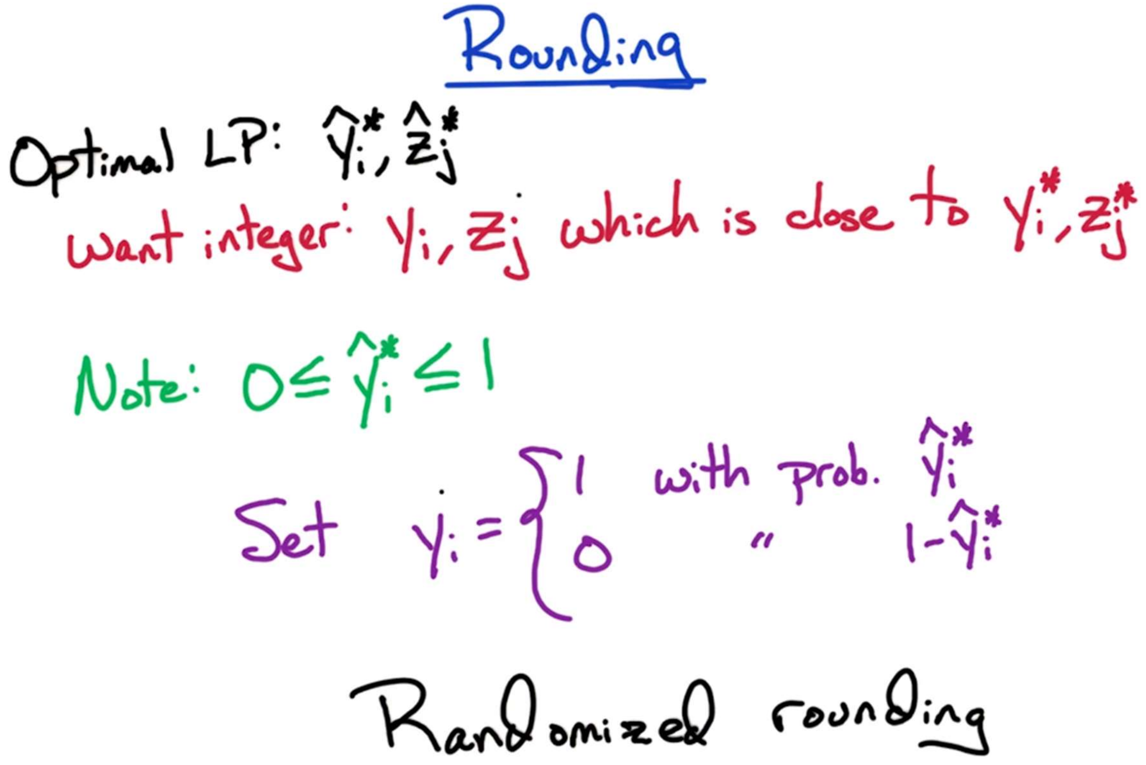
What we're going to do now, is we're going to convert this LP optimal solution into a feasible point for the ILP. It may not be an optimal solution to the ILP; but, it will be feasible. How do we do this? Well, we have to convert this LP solution into a point on the grid. The simplest way to do that is to convert these fractional points - these y-hat-stars to integer points - by rounding them to the nearest integer point. And, we're going to do this in a probabilistic way.

Now, once we round these to integer points, then we have a valid True/False assignment for our set input f. And, then, we're going to prove that this rounded integer point is not that far, not that much worse than the original LP solution (i.e., than the fractional solution). And since this fractional solution is at least as good as the optimal integer solution, then we know that the integer solution that we found is not that far off from the optimal integer solution.

So, we're going to take this LP solution and round it - that'll get it worse than this optimal solution. But, we will show that this integer solution that we find is not far off from this fractional solution.

***

(Slide 15) Rounding



Rounding

Optimal LP: $\hat{y}_i^*, \hat{z}_j^*$

want integer: y_i, z_j which is close to y_i^*, z_j^*

Note: $0 \le \hat{y}_i^* \le 1$

Set $y_i = \begin{cases} 1 & \text{with prob. } \hat{y}_i^* \\ 0 & \text{''} \quad 1-\hat{y}_i^* \end{cases}$

Randomized rounding

Once again, we take this LP, and we find the optimal point, and we denote the optimal point by this vector ($\hat{y}^*$, $\hat{z}^*$).

- Hats (^) correspond to LP's, so they might be fractional values.
- Without the hats, it corresponds to integers

So, our goal is to find an integer point - we'll denote it by yi and zj. We drop the stars because it might not be optimal any longer; but, we want this integer point (that we find by rounding this point) - we want this integer point (that we find) to be close to the optimal integer point.

How do we prove that this point (that we find) - this integer point (that we find) - is close to the optimal integer point because we don't know it at this point? Well, we will show that this rounding procedure doesn't change the objective function too much. So, this integer point (yi, zj) that we find is close to this optimal fractional point ($\hat{y}_i^*$, $\hat{z}_j^*$) - this optimal LP solution - and this ($\hat{y}_i^*$, $\hat{z}_j^*$) is at least as good as this (y_i^*, z_j^*) . Therefore, if this (yi, zj) is close to this one ($\hat{y}_i^*$, $\hat{z}_j^*$) , then it's also close to this one (y_i^*, z_j^*).

How do we round from this fractional point ($\hat{y}_i^*$, $\hat{z}_j^*$) to this integer point (yi, zj)? Recall that our LP and our integer linear program had the constraint that these variables $\hat{y}_i^*$ are constrained between 0 and 1. Therefore, this optimal solution also satisfies these constraints – so, $\hat{y}_i^*$ is between 0 and 1.

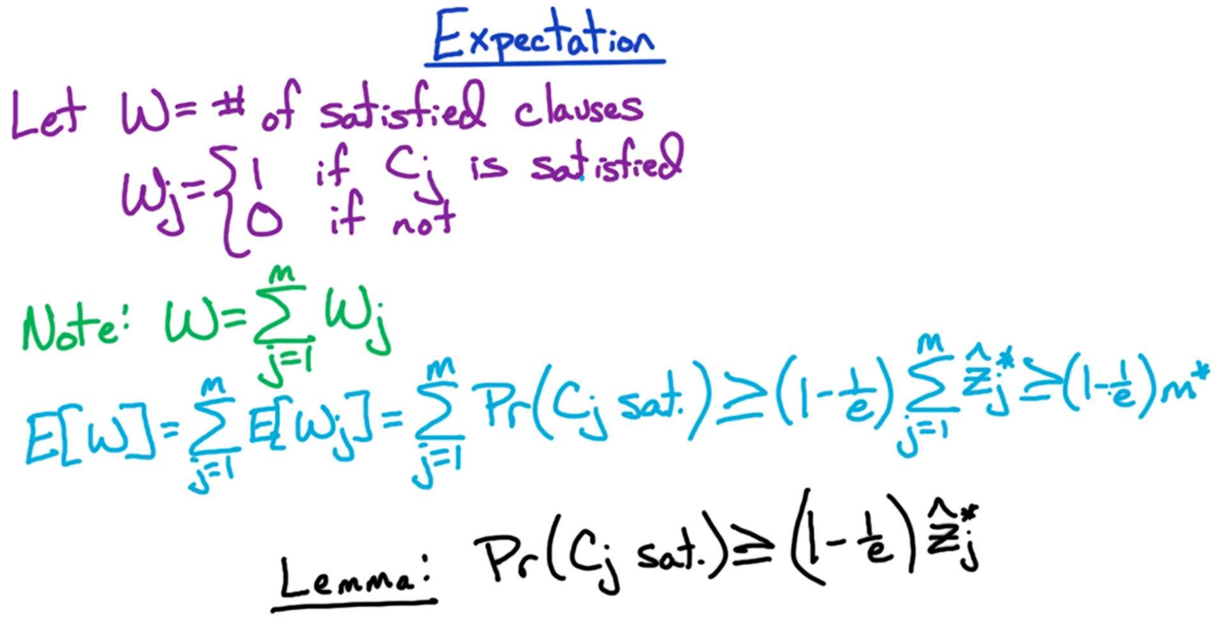
Therefore, we can think of it as like a probability. So, we're going to round this – so, we're going to set yi to be 1 or 0 with probability proportional to this - with probability $\hat{y}_i^*$ is between 0 and 1, and it's a real number. With that probability, we set yi to be 1. So if this is 3/4, then with probability 3/4, we set yi to be 1. And with probability 1/4, we set it to be 0. This is known as randomized rounding.

Now this actually completes our algorithm. We have an assignment now - we have a True/False assignment for the variables in the original formula. So we've taken this fractional point, and we rounded it to an integer point. Notice we don't have around these z's - we just have to round these y's. Now, if yi = 1, then we set the variable xi to be True. If yi = 0, we set the variable to be False. So we have a True/False assignment for the x's, and it's a randomized algorithm for setting these x's.

So, as we did before, we want to look at the Expected performance of this randomized algorithm.

***

(Slide 16) Expectation



Expectation

Let $W = \#$ of satisfied clauses

$$W_j = \begin{cases} 1 & \text{if } C_j \text{ is satisfied} \\ 0 & \text{if not} \end{cases}$$

Note: $W = \sum_{j=1}^{m} W_j$

$$E[W] = \sum_{j=1}^{m} E[W_j] = \sum_{j=1}^{m} \Pr(C_j \text{ sat.}) \geq \left(1-\frac{1}{e}\right) \sum_{j=1}^{m} \hat{z}_j^* = \left(1-\frac{1}{e}\right) m^*$$

Lemma: $\Pr(C_j \text{ sat.}) \geq \left(1-\frac{1}{e}\right) \hat{z}_j^*$

Once again, we want to look at the expected performance of this randomized algorithm that we just defined.

Therefore, we let capital W denote the number of satisfied clauses - this is in the random assignment - the assignment produced by solving the LP and doing randomized rounding.

Since the assignment is random, capital W is a random variable. So, we're going to look at its expectation. Now, to simplify the analysis of the expectation, we're going to do a clause-by-clause analysis.

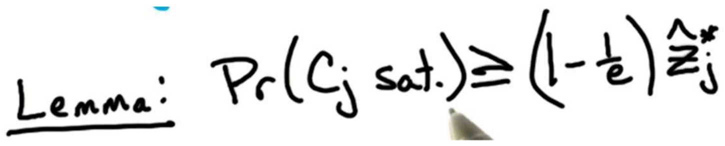
As before, we consider each clause. So, we consider the j clause, and we denote the random variable Wj - it takes value 1 if this clause Cj is satisfied, and 0 if it's not satisfied.

Now, if we sum Wj, where j going from 1 to m, then, for each satisfied clause, we get a count of 1 – so, the total count is going to be the total number of satisfied clauses - which equals capital W. We can analyze the expectation of capital W (E[W]) - the expected number of satisfied clauses - in a manner similar to what we did earlier in the lecture.

We can use this identity (W = ΣWj) to re-express expectation of W to be the expectation of this sum (E(W) = E(ΣWj). And then we can apply this linearity of expectation and take the sum from inside the expectation to outside. We get that the expectation of W equals the sum over j from 1 to m of the expectation of Wj (E(W) = Σ E(Wj)).

Now, the expectation of Wj (E(Wj)) is quite simple because it's a 0-1 random variable. So, the expectation of WJ in this case is just the probability that this clause Cj is satisfied.

Now, we're going to prove that the probability that this clause is satisfied is at least (1 – 1/e) times the LP value for this clause. The LP value for this clause is ẑj*.



Lemma: $\Pr(C_j \text{ sat.}) \geq \left(1-\frac{1}{e}\right)\hat{z}_j^*$

In some sense, this is like the probability that this LP satisfies this clause and this rounding procedure afterwards. This assignment that we end up with will satisfy this clause with probability at least 1 – 1/e times this original probability

So, we're going to prove this lemma momentarily; but, now, let's plug that back into the computation of E[W}. We can take this (1 – 1/e) outside of this sum. So, we have (1 – 1/e) times the sum from j going from 1 to m of ẑj* (Σ ẑj*). Now, what is this quantity Σ ẑj*? This is the value of the objective function for the linear program.

What do we know about the linear program versus the integer linear program? Well, the linear program is at least as great as the integer. So, this value of the objective function is going to be at least this - the optimal - for the integer linear program. This is going to be at least (1 – 1/e) times this value for the integer linear program. For the integer linear program, it's equal to the max sat. So, it's equal to the maximum number of clauses satisfied which we denoted by m*.

And, in conclusion, we can show that the expected performance of this algorithm - the number of satisfied clauses in expectation - is going to be at least the optimal number times (1 – 1/e).

Therefore, we have a (1 – 1/e) - approximation algorithm which improves upon our ½ - approximation algorithm. So we simply have to prove this lemma and then we have our algorithm.

***

(Slide 17) Lemma

Lemma

Look at $C_j = (x_1 \vee x_2 \vee \cdots \vee x_k)$

LP: $\hat{y}_1^* + \cdots + \hat{y}_k^* \geq \hat{z}_j^*$

$\Pr(C_j \text{ is sat.}) = 1 - \Pr(C_j \text{ is unsat.})$

$\Pr(C_j \text{ is unsat.}) = \prod_{i=1}^{k} (1 - \hat{y}_i^*)$

Take a look at this lemma that we're trying to prove.

Look at a clause Cj. And, let's say Cj is of size k and let's suppose the clause is (x1 ∨ x2 ∨ ··· ∨ xk) - so, we’re assuming all the literals are positive form. Let's just do the analysis for this simpler case and the same approach will apply to the general clause. And, in fact, since we're just looking at this clause in isolation and because of the symmetry, we can always convert this clause to appear in this form.

Now, what was the LP constraint for this clause? Let's look at it in terms of the optimal solution to the LP. So variable x1 has this variable in the LP $\widehat{y1}$*. We want that if this is 0, meaning if x1 is False, and, if that's the case for all k of these variables, then we want that this variable ẑj* is 0 in this case. So, if all of these literals are unsatisfying – meaning, all these variables are set to False - then all of these y's are set to 0 - then zj must be 0 - corresponding to this clause being unsatisfied. This is the LP constraint.

Look at $C_j = (x_1 \vee x_2 \vee \cdots \vee x_k)$

LP: $\hat{y}_1^* + \cdots + \hat{y}_k^* \geq \hat{z}_j^*$

Now, we're trying to analyze the probability that this clause is satisfied after the randomized rounding procedure. This is, of course, the same as 1 – Pr(this clause is unsatisfying). What's the probability that this clause is unsatisfied? Well, what's the probability that all these variables are set to false (Pr(Cj is unsatisfied) = Π (1 - ŷi*). So, we're looking at the product over these k variables. The ith variable is set to false with probably (1 - ŷi*).

$$\Pr(C_j \text{ is sat.}) = 1 - \Pr(C_j \text{ is unsat.})$$

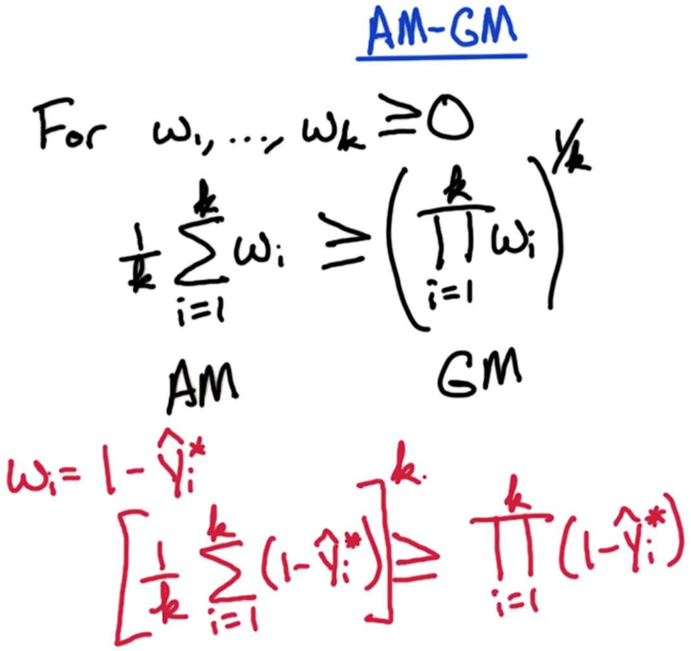
$$\Pr(C_j \text{ is unsat.}) = \prod_{i=1}^{k} (1 - \hat{y}_i^*)$$

Now, if you recall the lemma, we're trying to show that the probability that this clause is satisfied is at least (1 – 1/e) times this quantity ẑj*. What we know is that the probability it is satisfied equals 1 minus this product, of these Yi's. Now we know how to relate the sum of these Yi's to this Zj_star. How do we relate the product to the sum? So we want to relate this product to this sum and then we can relate it to ẑj*.

To relate this product to this sum, we're going to use the Arithmetic Mean (*points to* ŷ1* + ··· + ŷk*) - Geometric Mean (*points to* Π (1 - ŷi*)) inequality.

***

(Slide 18) AM-GM



AM-GM

For $w_1, \dots, w_k \geq 0$

$$\underbrace{\frac{1}{k}\sum_{i=1}^{k} w_i}_{\text{AM}} \geq \underbrace{\left(\prod_{i=1}^{k} w_i\right)^{1/k}}_{\text{GM}}$$

$w_i = 1 - \hat{y}_i^*$

$$\left[\frac{1}{k}\sum_{i=1}^{k}(1-\hat{y}_i^*)\right]^{k} \geq \prod_{i=1}^{k}(1-\hat{y}_i^*)$$

Let's take a look at the Arithmetic Mean - Geometric Mean and inequality. Let's take a look at its general form.

- We have k non-negative numbers, w1 through wk. Think of these as weights.

We want to look at the mean of these weights. So we can look at the arithmetic mean and the geometric mean.

- The arithmetic mean takes the sum and divides by the number of terms.
- The geometric mean takes the product and takes the kth root.

How do these two things relate?

- Well, the arithmetic mean is always at least the geometric mean (AM ≥ GM). And this holds for any non-negative weights.

AM-GM

For $w_1, \dots, w_k \geq 0$

$$\underset{\text{AM}}{\frac{1}{k}\sum_{i=1}^{k} w_i} \geq \underset{\text{GM}}{\left(\prod_{i=1}^{k} w_i\right)^{1/k}}$$

How do we relate this to our problem? Well, we're going to set wi to be (1 - ŷi*). And, then, the arithmetic mean is 1/k times Σ (1 - ŷi*). And, this is at least the geometric mean, which is Π (1 - ŷi*) and then we take the kth root of the right hand side.

$$w_i = 1 - \hat{y}_i^*$$

$$\frac{1}{k}\sum_{i=1}^{k}(1-\hat{y}_i^*) \geq \left[\prod_{i=1}^{k}(1-\hat{y}_i^*)\right]^{1/k}$$

Now, alternatively, instead of taking the kth root of the right hand side, we can take the kth power of both sides and we have the following inequality:

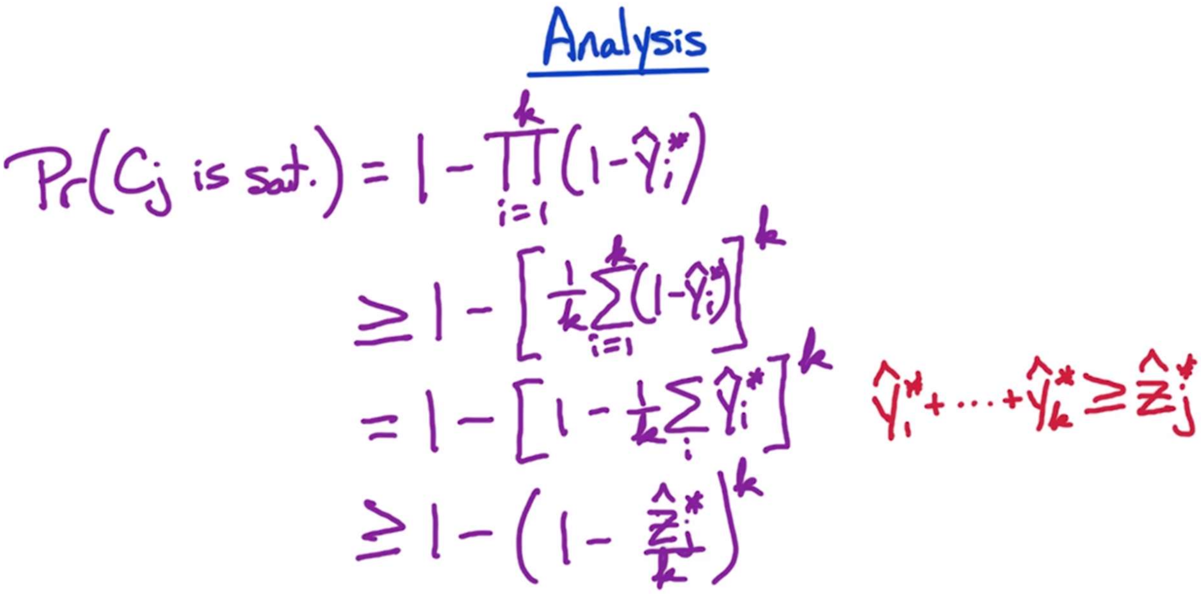
$$w_i = 1 - \hat{y}_i^*$$

$$\left[\frac{1}{k}\sum_{i=1}^{k}(1-\hat{y}_i^*)\right]^{k} \geq \prod_{i=1}^{k}(1-\hat{y}_i^*)$$

This is the one we're going to use.

***

(Slide 19) Analysis



Analysis

$$\Pr(C_j \text{ is sat.}) = 1 - \prod_{i=1}^{k}(1-\hat{y}_i^*)$$

$$\geq 1 - \left[\frac{1}{k}\sum_{i=1}^{k}(1-\hat{y}_i^*)\right]^k$$

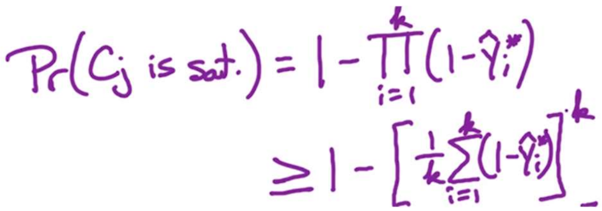
$$= 1 - \left[1 - \frac{1}{k}\sum_{i} \hat{y}_i^*\right]^k \qquad \hat{y}_1^* + \cdots + \hat{y}_k^* \geq \hat{z}_j^*$$

$$\geq 1 - \left(1 - \frac{\hat{z}_j^*}{k}\right)^k$$

Going back to our analysis from earlier, we're looking at the probability that this jth clause is satisfied.   This equals 1 - the probability that it's unsatisfied - which is the product over i, from 1 to k, of (1 - ŷi*).

Now, we can apply the Arithmetic Mean - Geometric Mean inequality. The arithmetic mean is 1/k times the sum of i, going from one to k of (1 - ŷi*), and, then we're raising it to the kth power since we dropped that 1/kth root of the geometric mean.

Now, the inequality said that the geometric mean is at most the arithmetic mean. But we're doing the negative of this, so we get that it's “at least”.



$$\Pr(C_j \text{ is sat.}) = 1 - \prod_{i=1}^{k}(1-\hat{y}_i^*)$$

$$\geq 1 - \left[\frac{1}{k}\sum_{i=1}^{k}(1-\hat{y}_i^*)\right]^k$$

Now, we can simplify this a bit. We can do the sum over each of these terms separately. And the first term is a sum over i from one to k of 1. That's simply k. So, that divides by this one over k.   So, we get one for the first term. So, we get a one for this first one, and the second term becomes 1/k times the sum over I, from one to k, of ŷi*. And then we get this whole quantity raised to the Kth power still.

$$\geq 1 - \left[\frac{1}{k}\sum_{i=1}^{k}(1-\hat{y}_i^*)\right]^k$$
$$= 1 - \left[1 - \frac{1}{k}\sum_i \hat{y}_i^*\right]^k$$

Now we're in business, because we have this sum over these yi-hat-stars. Now, if you recall before, we had this constraint from the linear program. It said that if each of these variables was set to False – so, each of these yi's was set to zero – then, the corresponding zj must be zero.

$$\hat{y}_1^* + \cdots + \hat{y}_k^* \geq \hat{z}_j^*$$

Now, this is actually a linear program, so we don't have this integral constraints. So, we simply know that the sum of these y's is at least this real number z. But, we still have this inequality, and our goal is to relate this probability that this clause is satisfied to this quantity ẑj* (zj-hat-star).

So, we can apply this inequality, plug it in, and we can drop these y's and get the z. Now, this is "at least".

Then, we're doing a negative, and then another negative. So, we get that this is at least one minus the quantity (1 - ẑj*/k), all raised to the kth power.

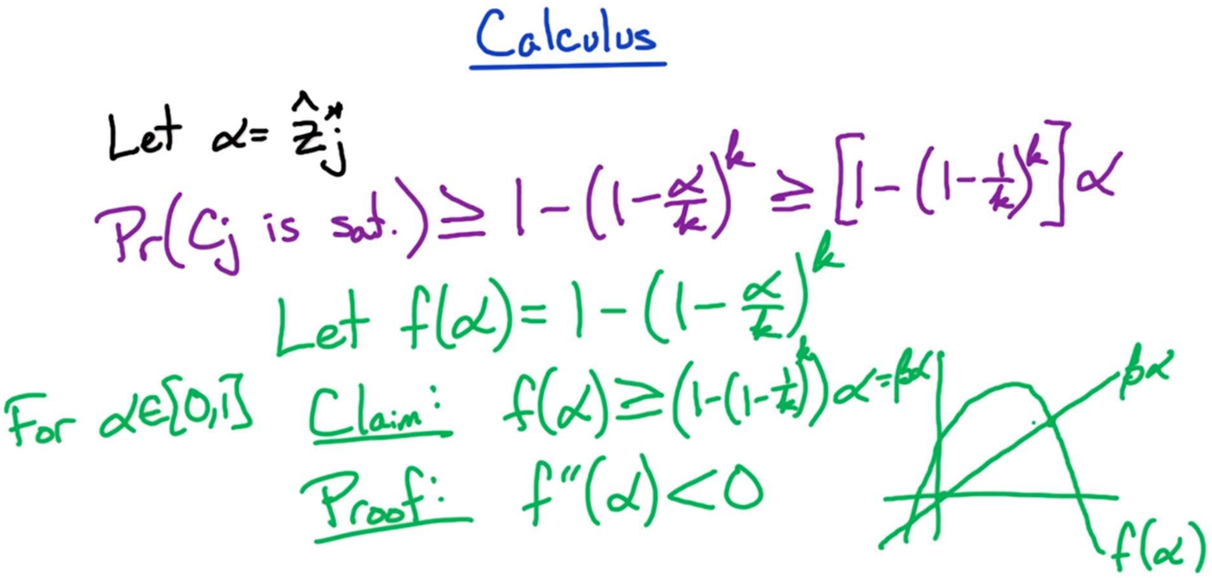
$$\geq 1 - \left(1 - \frac{\hat{z}_j^*}{k}\right)^k$$

So, we replaced the sum over y's by ẑj*).

Now, we have to do a bit of calculus to simplify this. What we want to do is we want to move this zj outside of this kth power.

***

(Slide 20) Calculus



Calculus

Let $\alpha = \hat{z}_j^*$

$Pr(C_j \text{ is sat.}) \ge 1-\left(1-\frac{\alpha}{k}\right)^k \ge \left[1-\left(1-\frac{1}{k}\right)^k\right]\alpha$

Let $f(\alpha) = 1-\left(1-\frac{\alpha}{k}\right)^k$

For $\alpha \in [0,1]$ Claim: $f(\alpha) \ge \left(1-\left(1-\frac{1}{k}\right)^k\right)\alpha = \beta\alpha$

Proof: $f''(\alpha) < 0$

Let α denote ẑj*. What we've shown so far is that the probability that this clause Cj satisfied is at least (1 – (1 - α/k)^{k}.

Now, once again, we want to get this α outside of this power. We want to make it a linear function of α. Let's look at this function of α. Let's define f(α) to be this right hand side.

And our claim is that f(α) – so, this right hand side here - is at least some constant times α. It's a constant in the sense it's independent of α - It does depend on k - the size of this clause.

Now, what is that constant there? Well, it's going to be similar to this without the alpha. It's going to be (1 – (1 – 1/k)^{k}). Once we prove that, then we can apply that here and we get that this is at least this quantity - which is this constant times α.

We got the alpha outside of the power so it's now linear in α. It's linear in this z. Now, how do we prove this claim?

First off, you take the second derivative and you prove that it's negative. Therefore, this function is concave. Since f(α) is a concave function, it's going to look something like this (see diagram in slide above).

We're trying to relate f(α), this curve, to this linear function.

Let's denote this linear function as $\beta\alpha$, so this quantity here $(1 – (1 – 1/k)^k)$ is β. Let's look at this line $\beta\alpha$. For large values of α, it might not be bigger. But notice, what values of α are we interested in? We're interested in α's between 0 and 1. So, we only need this claim for values of α between 0 and 1.

So, since it's a concave function it's got one peak. If we look at the end points that we're interested in, 1 and 0, and, if we prove that this curve is above this line $\beta\alpha$ at these two points, 0 and 1, then it's also above at every intermediate point.

So once we take the second derivative and prove it's concave then it suffices to check the points $\alpha=0$ and $\alpha=1$. And if you plug in $\alpha = 0$, we get 0 on both sides; and $\alpha=1$, it's straightforward to check that that's also true.

***

(Slide 21) Finishing Up

Finishing up

$$\Pr(C_j \text{ is sat.}) = 1 - \prod_i (1-\hat{y}_i^*)$$
$$\geq 1 - \left(1 - \frac{\hat{z}_j^*}{k}\right)^k$$
$$\geq \left[1 - \left(1 - \frac{1}{k}\right)^k\right]\hat{z}_j^*$$
$$\geq \left(1 - \frac{1}{e}\right)\hat{z}_j^*$$
$$e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{3!} \pm \cdots$$
$$e^{-x} \geq 1 - x$$

Okay, we're almost done. Actually, we really are done - we just have to summarize.

- We're looking at the probability that this clause, Cj, is satisfied. This is equal to $(1 – \Pi (1 - \hat{y}i^*))$.
- Then we apply the AM-GM inequality to convert this product into a sum.
- Then we apply the LP constraint to relate that sum over the yi's to zj.
- And then we got this inequality. We got that this is at least $1 – (1 - \hat{z}j^*/k)^k$.
- Now what we just proved is that this whole thing is at least this constant which is a function of k but independent of z: $(1 – (1 – 1/k)^k)\ \hat{z}j^*$.

Okay, now let's simplify this a little bit. If you recall, the Taylor series for e^{-x}:

$e^{-x} = 1 – x + x^2/2 – x^3/3! \pm$ (a plus-minus, alternating series).

Now, what if we want to relate e^{-x} to 1 - x? Well, the next term is positive. So e^{-x} is going to be bigger. So, 1 – 1/k is at most $(e^{-1/k})^k$ - We have 1/e. And, then take the minus and then we have it - it's "at least"! So, we have this quantity – it becomes (1 – 1/e) $\hat{z}j^*$, and that's what we wanted to prove. That was the lemma we claimed. And therefore, we have a (1 – 1/e) - approximation algorithm.

That completes the analysis of this LP-based algorithm.

***

(Slide 22) Max-SAT Approximation Summary

Summary

Take NP-hard Problem

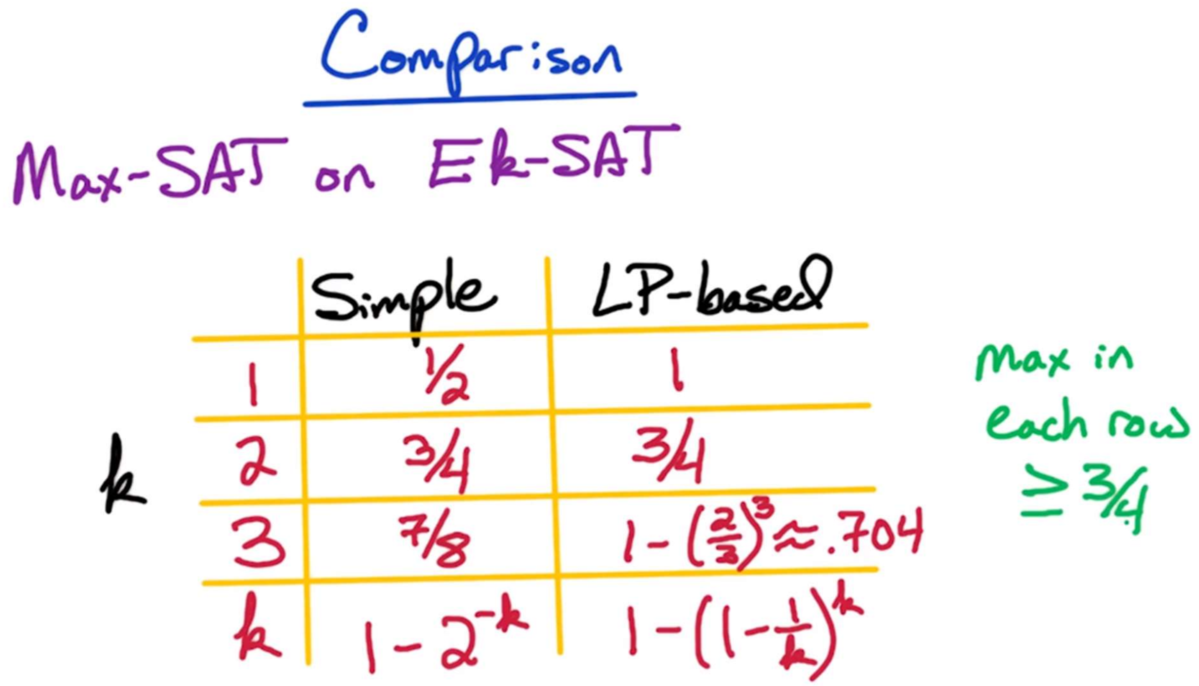
- reduce to ILP
- relax to LP & solve
- randomized round

Now, let's summarize some important points about this LP-based algorithm.

- It's a very general approach. We can take a hard problem and oftentimes we can reduce it to Integer Linear Programming. Integer linear programming is very general, very powerful technique. So, it's very common that we can reduce these NP-hard problems to Integer Linear Programming.
- Then we can relax it to a Linear Programming. We just drop this integrality constraint,
- and then we can solve it. There are polynomial time solvers. So, we can often use the simplex algorithm. It's also fast.
- Then we can take this LP solution, and we can round it in a similar randomized way as we just did for Max-SAT. And this gives us a feasible point for the integer linear program. And, hopefully, it's a reasonable heuristic for this solution - the optimal value of this ILP.

***

(Slide 23) Comparison



Comparison

Max-SAT on Ek-SAT

k		Simple	LP-based
	1	$1/2$	1
	2	$3/4$	$3/4$
	3	$7/8$	$1-(\frac{2}{3})^3 \approx .704$
	k	$1-2^{-k}$	$1-(1-\frac{1}{k})^k$

Max in each row $\geq 3/4$

Now, we've seen two algorithms for Max-SAT. Now, what if we look at Max-SAT on Exact k-SAT (Ek-SAT) formulas? So these are formulas where all clauses were size exactly k. And let's look at the performance of these two algorithms, that we've seen so far, as a function of k.

So we have

- the simple algorithm - that's the simple randomized assignment. So each variable is set to True with probability a half, and false with probably a half.
- and we have this LP based scheme. So, we write Max-SAT as an integer linear program, we relax it to a linear programming, and then we round it.

How do these two schemes compare for different values of k? Let's take a look:

- Let's look at the performance of these two algorithms for k=1. So it's all unit clauses, so it's quite trivial;
- k=2 and k=3, and in general k:
  - If you recall the simple algorithm, the probability of clauses not satisfied is 2^{-k}. So the probability this is satisfied is $1 - 2^{-k}$. So, in general for k, it achieves of $1 - 2^{-k}$ approximation factor. Plugging in k=1, and get one-half. k=2 - we get three-quarters, K=3, we get 7/8's.
  - Now the LP, the general form, was quite complicated. We proved for general k that it achieves : $(1 – (1 – 1/k)^k)$. Plugging in k=1, we get 1, which is quite good. Simple scheme was quite bad for k=1, but the LP based scheme is quite good.

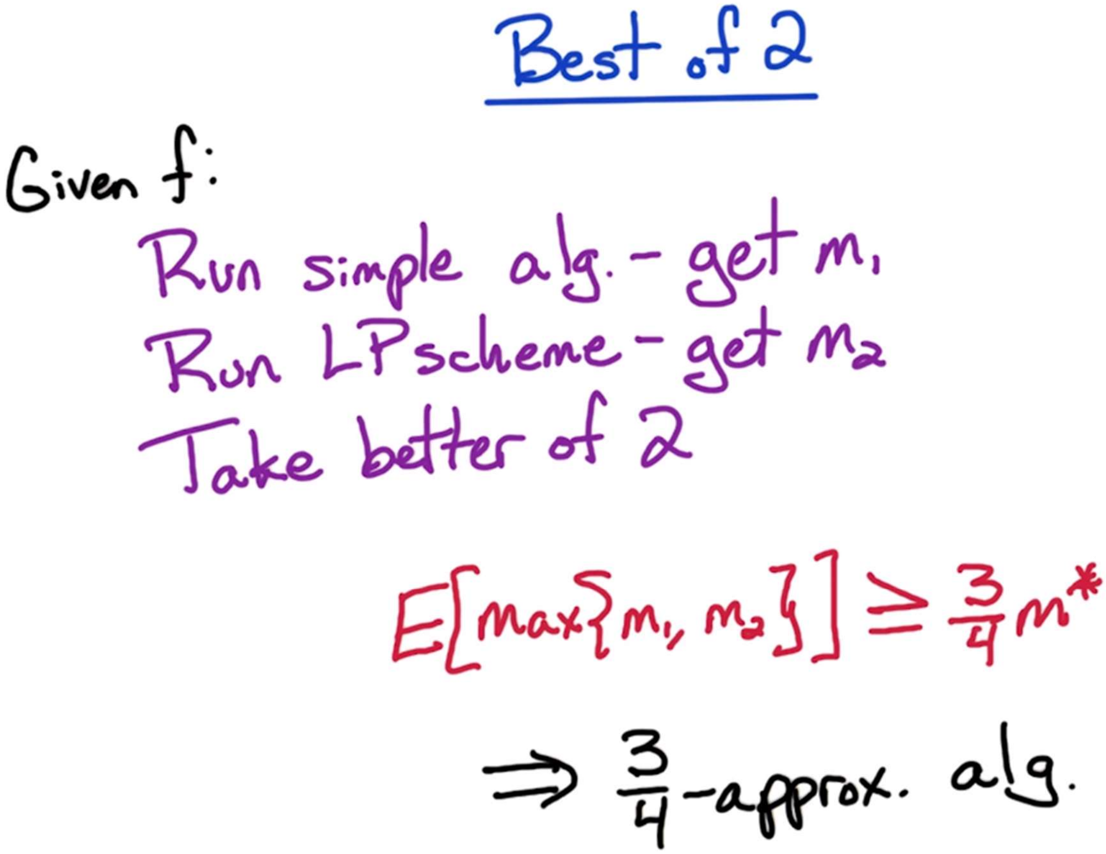
Now k=2, they match, both three-quarters. For k=3, we got 1 – $(2/3)^3$. If you plug that into a calculator, that's roughly 0.704. So for K, at least three, the simple scheme beats the LP scheme. But for small clauses, the LP scheme is at least as good or even better.

The key observation is that if you look at each row - the max in each row - the best of these two schemes for every k is at least three-quarters.

So, can we combine these two schemes to achieve a ¾ - approximation? Yes, we can!

***

(Slide 24) Best of 2



Best of 2

Given f:

Run simple alg. - get m_1

Run LPscheme - get m_2

Take better of 2

$$E[\max\{m_1, m_2\}] \geq \frac{3}{4} m^*$$

$\Rightarrow \frac{3}{4}$-approx. alg.

Here's our combined algorithm.

- Take our input formula f. We first run the simple randomized algorithm, which assigns each variable True or False independently with probably a half, and we get an assignment which satisfies, let's say, m1 clauses of f. So m1 is the number of clauses satisfied by this simple algorithm - this simple randomized algorithm.
- We also run the LP-based scheme, and we get an assignment. We look at how many clauses are satisfied by that assignment. Let's say it satisfies m2 clauses
- and whichever of these two is better, we take that assignment.

So we look at both of these assignments and we take the better of these two assignments, that's our algorithm.

Now, if we look at the expected performance of this algorithm, what we're going to achieve is the max of these two. So the performance, the expected performance of this best of two algorithms, is the expectation of the max of m1 and m2.

That's what our algorithm is going to achieve. And, what we can prove is that this max is at least three-fourths the optima value. Why is that?

Well, that follows from the fact that each of these algorithms - the best of these two algorithms for each row on the previous slide - is at least three quarters. For every specific k, we get at least three quarters, and then we can analyze this in a clause-by-clause manner so that we get at least three quarters of the optima value.

You can look at the online notes to see the calculus behind this. But the punch line is that this algorithm, this combined algorithm, gives a ¾ - approximation algorithm for max SAT, even when the formula has clauses where some are small and some are big. So even with formulas of varying length clauses, we achieve a ¾ - approximation.