CS 341 F20 Lecture 21: NP-completeness, continued A. Lubiw, U. Waterloo

Recall

Summary of Lecture 20

NP-completeness of Independent Set, Vertex Cover, Hamiltonian cycle, TSP

What you should know from Lecture 20:

- how to prove a problem is NP-complete using a polynomial time many-one reduction

Next:

Ind. Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover

$\leq_P$

Circuit SAT $\leq_P$ 3-SAT $\leq_P$ Ham.cycle $\leq_P$ TSP

$\leq_P$ Subset Sum

These are harder proofs.
Goal: appreciate trickier constructions; establish the results. this is a Math Faculty after all

Subset Sum.
Input: Numbers $w_1, \ldots, w_n, W$
Question: Is there a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} w_i = W$

Theorem. Subset Sum is NP-complete.
Proof.

1. Subset Sum is in NP. (done in previous lecture)

2. 3-SAT $\leq_P$ Subset Sum

   Assume we have a polynomial time algorithm for Subset Sum. Make a polynomial time algorithm for 3SAT.

   Input: A 3-SAT formula F with clauses $C_1 \ldots C_m$ on variables $x_1 \ldots x_n$

   Output: Is F satisfiable?
   - construct an instance of Subset Sum such that
     it has a solution iff F is satisfiable
   - run the Subset Sum algorithm
   - return its answer

   We've seen how to turn 3-SAT into a packing problem (Independent Set) and into a sequencing problem (Hamiltonian cycle) and now we must turn it into a number problem.

Input: A 3-SAT formula F with clauses $C_1 \ldots C_m$ on variables $x_1 \ldots x_n$
Construct an instance of Subset Sum such that it has a solution iff F is satisfiable

Idea: Choosing numbers in Subset Sum will be choosing True/False.
The bits of the numbers will encode information about the clauses.

Create a 0-1 matrix

	C_1	C_2	$\cdots$	C_m
x_1	1	0	$\cdots$	
$\neg x_1$	0	1	$\cdots$	
x_2	0	0		
$\neg x_2$	1	0		
x_3	1	0		
$\neg x_3$	0	0		
$\vdots$				

e.g. $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$
$C_2 = (\neg x_1 \vee x_4 \vee x_5)$

General rule

$M[x_i, C_j] = 1$ if x_i in C_j
$M[\neg x_i, C_j] = 1$ if $\neg x_i$ in C_j
$= 0$ otherwise

We assume no clause contains the same literal twice.

Regard the rows as binary (or other base) numbers.
Choosing a number = choosing a row. Adding numbers = adding up rows.

Input: A 3-SAT formula F with clauses $C_1 \dots C_m$ on variables $x_1 \dots x_n$
Construct an instance of Subset Sum such that it has a solution iff F is satisfiable

Idea: Choosing numbers in Subset Sum will be choosing True/False.
The bits of the numbers will encode information about the clauses.

Create a 0-1 matrix

	C_1	C_2	$\dots$	C_m
x_1	1	0	$\dots$	
$\neg x_1$	0	1	$\dots$	
x_2	0	0		
$\neg x_2$	1	0		
x_3	1	0		
$\neg x_3$	0	0		
$\vdots$				
target sum	≥ 1	≥ 1	$\dots$	

e.g. $C_1 = (x_1 \vee \neg x_2 \vee x_3)$
$C_2 = (\neg x_1 \vee x_4 \vee x_5)$

General rule
$M[x_i, C_j] = 1$ if x_i in C_j
$M[\neg x_i, C_j] = 1$ if $\neg x_i$ in C_j

We assume no clause contains the same literal twice.

to ensure we pick ≥ 1 literal in each clause

Regard the rows as binary (or other base) numbers.
Choosing a number = choosing a row. Adding numbers = adding up rows.

Input: A 3-SAT formula F with clauses $C_1 \dots C_m$ on variables $x_1 \dots x_n$
Construct an instance of Subset Sum such that it has a solution iff F is satisfiable

Idea: Choosing numbers in Subset Sum will be choosing True/False.
The bits of the numbers will encode information about the clauses.

Create a 0-1 matrix

	C_1	C_2	$\dots C_m$
x_1	1	0	
$\neg x_1$	0	1	
x_2	0	0	
$\neg x_2$	1	0	
x_3	1	0	
$\neg x_3$	0	0	
$\vdots$			
target sum	≥ 1	≥ 1	$\dots$

e.g. $C_1 = (x_1 \vee \neg x_2 \vee x_3)$

$C_2 = (\neg x_1 \vee x_4 \vee x_5)$

General rule

$M[x_i, C_j] = 1$ if x_i in C_j

$M[\neg x_i, C_j] = 1$ if $\neg x_i$ in C_j

We assume no clause contains the same literal twice.

to ensure we pick ≥ 1 literal in each clause

Regard the rows as binary (or other base) numbers.
Choosing a number = choosing a row. Adding numbers = adding up rows.

Issues: (1) ensure we don't choose row x_i and row $\neg x_i$

(2) how can we ensure sum ≥ 1? What can the sum be? 1 or 2 or 3.
Add slack rows of 1 and 2 so sum can always be 4.

	C_1	C_2	$\cdots$	C_m	x_1	x_2	$\cdots$	x_n
x_1					1	0	$\cdots$	0
$\neg x_1$					1	0	$\cdots$	0
x_2		as above			0	1	$\cdots$	0
$\neg x_2$					0	1	$\cdots$	0
$\vdots$								
x_n								
$\neg x_n$								
$S_{1,1}$	1							
$S_{1,2}$	2							
$S_{2,1}$		1						
$S_{2,2}$		2						
$\vdots$								
$S_{m,1}$				1				
$S_{m,2}$				2				
target	4	4	$\cdots$	4	1	1	$\cdots$	1

for (1)

slack rows for (2)

must pick ONE of $x_i, \neg x_i$

Finally:

W = interpret last row in base 10

numbers = one for each row, interpreting the row in base 10

What is the size of the Subset Sum Problem?

$2n + 2m$ numbers each with $n+m$ base 10 digits.

Why base 10?

Large enough to avoid carries. And familiar.

Claim. Polynomial time.

Claim. *F* is satisfiable iff there is a subset of the numbers with sum W.
Proof.

$\Rightarrow$ Suppose *F* is satisfiable. If x_i is True, pick row x_i. If x_i is False, pick row $\neg x_i$.
Then column x_i adds up to its target 1, and column C_j adds to 1, 2, or 3.
Next we choose some slack rows $s_{j,1}$ and $s_{j,2}$ to increase the sum to 4:

$$1 + s_{j,1} + s_{j,2} = 4$$

$$2 + s_{j,2} = 4$$

$$3 + s_{j,1} = 4$$

This gives a set of rows (i.e. numbers) that sum to W.

$\Leftarrow$ Suppose there is a subset with sum W.

Note that any whole column sum is ≤ 6, so no carries occur, and column sums must give the target digits.

Because x_i column sum is 1, we must have chosen row x_i or row $\neg x_i$ (not both) — set the variable accordingly.

Because column C_j sum is 4 and slacks sum to ≤ 3, we must have chosen a literal to satisfy clause C_j. Thus *F* is satisfiable.

Summary of Lecture 21, Part 1

Subset Sum is NP-complete

What you should know from Lecture 21, Part 1:

- appreciate that NP-hardness proofs can be tricky, and that we can use numbers to encode things

this should be second nature to you as a CS student!

Next:

Ind. Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover

$\leq_P$

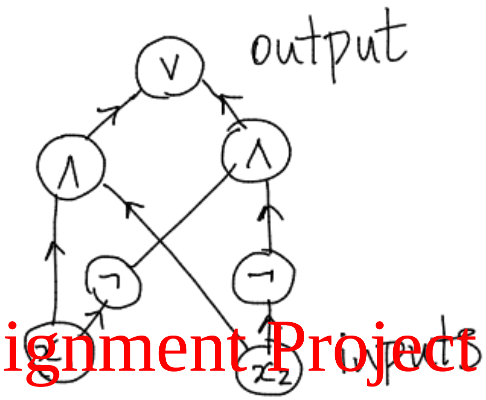
Circuit SAT $\leq_P$ 3-SAT $\leq_P$ Ham.cycle $\leq_P$ TSP

$\leq_P$

Subset Sum

The first NP-completeness proofs.

Circuit Satisfiability



$(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$

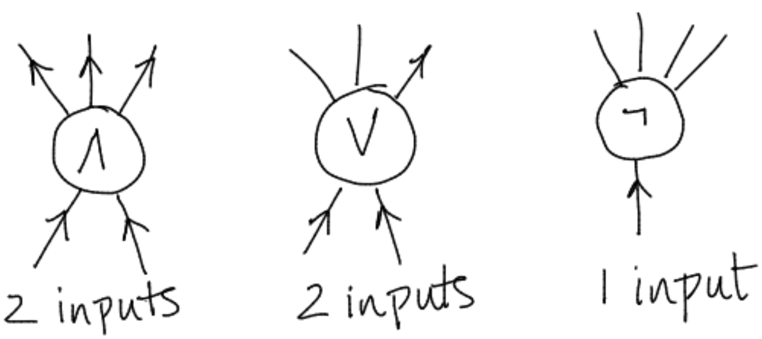
$x_1 \equiv x_2$

x_1 is the same (value as) x_2

example $x_1 = 0$ $x_2 = 1$ output 0

A ***circuit*** is a directed acyclic graph with:

- ***sources*** (no edge entering), labelled with variables or 0 or 1 — inputs
- one ***sink*** (no edge leaving) — output
- ***internal nodes***



A circuit ***computes an output*** (in the obvious way) when values are given for the input variables.

Circuit Satisfiability
Input: A circuit C
Question: Is there an assignment of values to inputs such that the output is 1?
i.e., is C satisfiable?

Theorem. Circuit SAT is NP-complete.
Proof.

1. Circuit SAT is in NP. (easy, details omitted)

2. this is the first NP-completeness proof so we must prove that

for every Y in NP, $Y \leq_P$ Circuit SAT

i.e. for every Y in NP, there is an algorithm that maps any input y for Y to a circuit C s.t. y is a YES input iff C is satisfiable.

High level idea only.

What can we use? Just that $Y \in$ NP,
i.e., there is a poly time verification algorithm A for Y. A takes two inputs y, g, (g = certificate or "guess") and outputs YES/NO. Property of A:

y is a YES instance for Y iff $\exists$ g (of poly size) s.t. A(y,g) outputs YES

2. this is the first NP-completeness proof so we must prove that

for every Y in NP, $Y \leq_P$ Circuit SAT

What can we use? Just that $Y \in$ NP,
i.e., there is a poly time verification algorithm A for Y. A takes two inputs y, g,
(g = certificate or "guess") and outputs YES/NO. Property of A:

y is a YES instance for Y iff $\exists$ g (of poly size) s.t. A(y,g) outputs YES

Idea: Convert algorithm A with known input y and unknown input g
to a circuit C with input variables = bits of g
such that C is satisfiable iff $\exists$ g s.t. A(y,g) outputs YES

Write a program for algorithm A. Compile it. Assemble . . .
At the hardware level, A is implemented by $\wedge$, $\vee$, $\neg$ gates
We get a circuit C.

Lots of hand-waving here. Relying on your intuition as CS students.

Inputs to C : bits of y (known), bits of g (variables)
Internal nodes of circuit: memory locations after each time step of algorithm A.

Because size(g) is polynomial and A runs in polynomial time, the circuit has polynomial size.

Is there an algorithm to convert A, y to C? Yes: compiler, assembler, etc.
and this takes polynomial time.

Summary of Lecture 21, Part 2

Circuit SAT is NP-complete — the first NP-completeness proof (at least the idea)

Next:

Ind. Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover

$\leq_P$

Circuit SAT $\leq_P$ 3-SAT $\leq_P$ Ham.cycle $\leq_P$ TSP

$\leq_P$

Subset Sum

Theorem. 3-SAT is NP-complete.
Proof.

1. 3-SAT is in NP. (easy, details omitted)

2. Circuit SAT $\leq_P$ 3-SAT

   Assume we have a polynomial time algorithm for 3-SAT. Make a polynomial time algorithm for Circuit SAT.

   Input: A circuit *C*
   Output: Is *C* satisfiable?

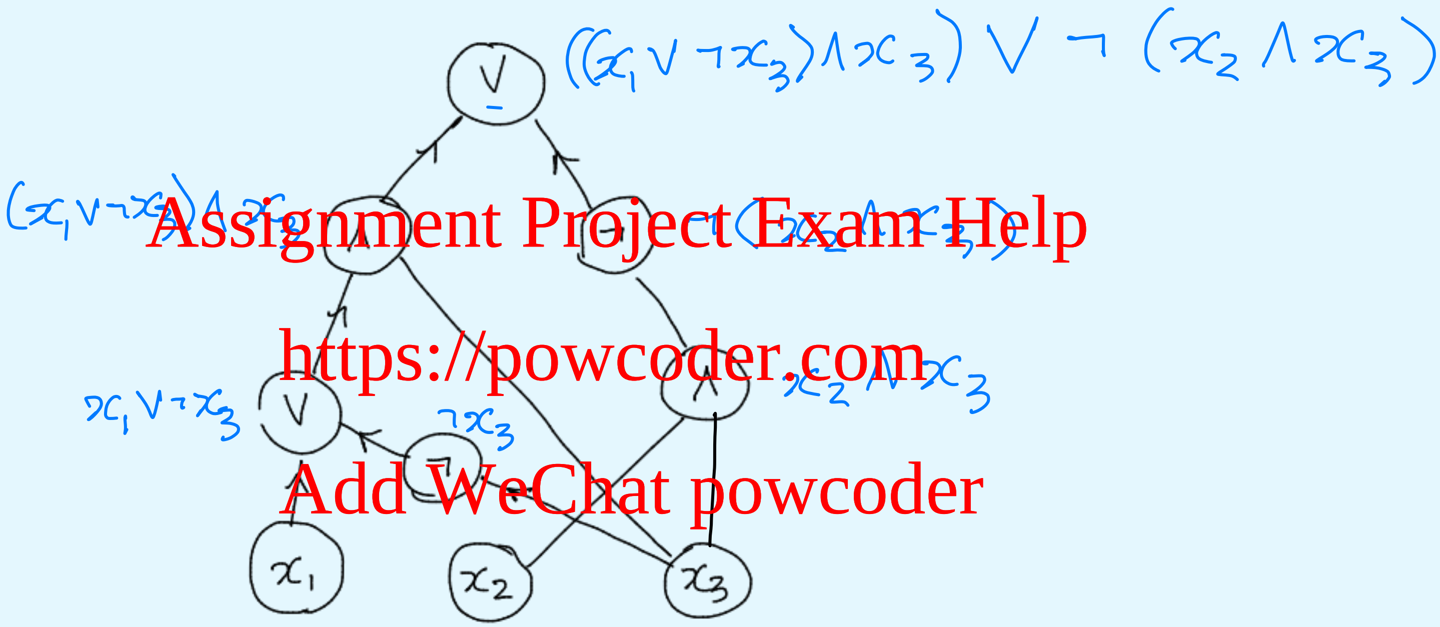
   - construct a 3-SAT formula *F* such that
     C is satisfiable iff *F* is satisfiable
   - run the 3-SAT algorithm

   - return its answer

   Intuitively (or from CS 245), circuits and formulas are equivalent. Just convert circuit *C* to formula *F*.

Convert circuit C to formula F.

the obvious way:

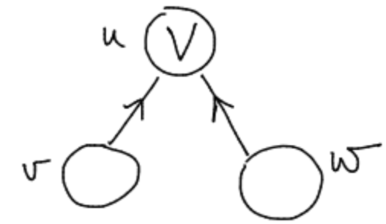


Caution: Is this polynomial size?

NO!

Convert circuit *C* to formula *F*.

the better way: make a variable x_u for each node *u* in the circuit



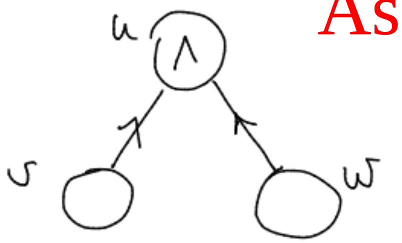
$x_u \equiv x_v \vee x_w$

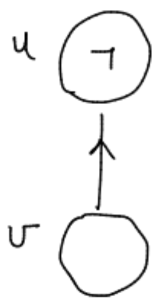
as 3-SAT clauses

$(\neg x_u \vee x_v \vee x_w) \wedge (x_u \vee \neg x_v) \wedge (x_u \vee \neg x_w)$

$x_u \vee \neg(x_v \vee x_w)$

$x_u \vee (\neg x_v \wedge \neg x_w)$





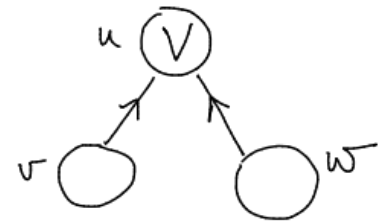
Note: $a \equiv b$ means $(\neg a \vee b) \wedge (a \vee \neg b)$

Claim. We can turn clauses of 2 literals into clauses of 3 literals.

Final formula: F = $\bigwedge$ of all clauses $\wedge$ x_{output}

Convert circuit C to formula F.

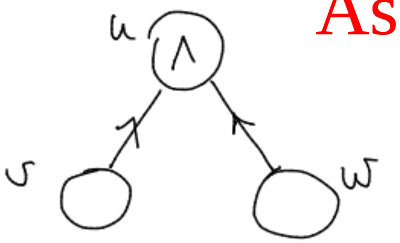
the better way: make a variable x_u for each node u in the circuit



$x_u \equiv x_v \vee x_w$

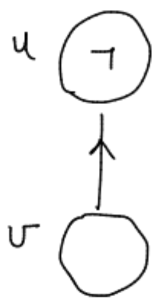
as clauses:

$(\neg x_u \vee x_v \vee x_w) \wedge (x_u \vee \neg x_v) \wedge (x_u \vee \neg x_w)$



$x_u \equiv x_v \wedge x_w$

$(\neg x_u \vee x_v) \wedge (\neg x_u \vee x_w) \wedge (x_u \vee \neg x_v \vee \neg x_w)$



$x_u \equiv \neg x_v$

$(x_u \vee x_v) \wedge (\neg x_u \vee \neg x_v)$

Note: $a \equiv b$ means $(\neg a \vee b) \wedge (a \vee \neg b)$

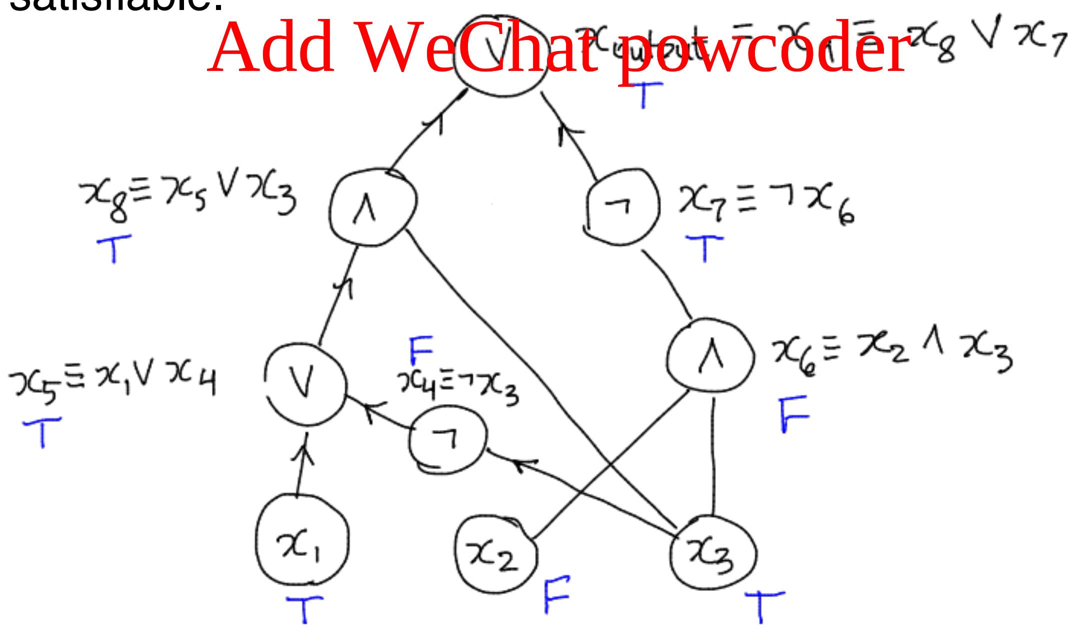
Claim. We can turn clauses of 2 literals into clauses of 3 literals.

Final formula: $F = \bigwedge$ of all clauses $\wedge\ x_{output}$

Claim 1. *F* has polynomial size and can be computed in polynomial time.

Claim 2. *F* is satisfiable iff *C* is satisfiable.
Proof.
⇐ Suppose *C* is satisfiable. Then assigning True/False to variables of *F* according to *C*'s computation will satisfy *F*.

⇒ Suppose *F* is satisfiable. Then there is an assignment of True/False to the variables (original inputs + new variables for circuit nodes) that makes *F* True. For circuit *C*, use the same values for the input variables. By construction, the variables for the circuit nodes capture the evaluation of *C*. And $x_{\text{output}} = 1$ (True). Therefore *C* is satisfiable.



Summary of Lecture 21

Ind. Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover

$\leq_P$

Circuit SAT $\leq_P$ 3-SAT $\leq_P$ Ham.cycle $\leq_P$ TSP

$\leq_P$

Subset Sum

What you should know from Lecture 21.

Appreciate NP-completeness proofs. Know some basic NP-complete problems.

Next:

A glimpse of more recent results on NP-completeness.