Computational Models — Lecture 13[1]

Handout Mode

Iftach Haitner.

Tel Aviv University.

January 23, 2017

[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

- SAT is NP-Complete (slides from lecture 12)
- CSAT $\in \mathcal{NPC}$
- 3SAT $\in \mathcal{NPC}$
- CLIQUE, IND-SET $\in \mathcal{NPC}$
- HAMPATH, HAMCYCLE $\in \mathcal{NPC}$

- Sipser, 7.4–7.5

Section 1

Reminder

Polynomial-time reducibility, reminder

Definition 1 (poly-time computable functions)

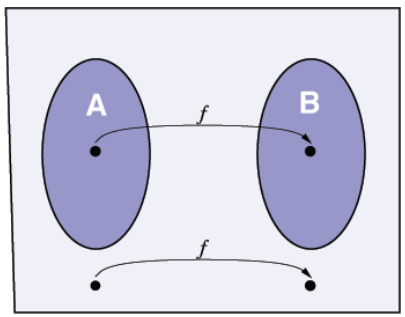
A function $f : \Sigma^* \mapsto \Sigma^*$ is polynomial-time computable, if there is a poly-time deterministic TM that

- starts with input w, and
- halts with $f(w)$ on tape.

Definition 2 (polynomial-time reduction)

A polynomial-time computable $f : \Sigma^* \mapsto \Sigma^*$ is a poly-time reduction from language A to B, if $x \in A \iff f(x) \in B$ for every $x \in \Sigma^*$.
Is such a reduction from A to B exists, we say that A is poly- time mapping reducible to B, denoted $A \leq_P B$.

The mapping f efficiently converts questions about membership in A to membership in B.



NP completeness, reminder

Definition 3 ($\mathcal{NP}$-complete)

A language B is NP-complete, if

- B $\in \mathcal{NP}$, and
- Every A $\in \mathcal{NP}$ is poly-time reducible to B (i.e., A $\leq_P$ B)

Let $\mathcal{NPC}$ denote the class of all $\mathcal{NP}$-complete languages.

Section 2

SAT is NP-Complete (slides in lecture 12)

Section 3

CSAT $\in \mathcal{NPC}$

CSAT

It is useful to consider a special version of SAT.

- A literal is a variable or negated variable: x or $\overline{x}$.
- A clause is several literals joined by $\vee$s: $(x_1 \vee \overline{x_2} \vee \overline{x_3})$
- A Boolean formula is in conjunctive normal form (CNF) if it consists of clauses, connected with $\wedge$s.
- For example: $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$
- The language of satisfied CNF formulas:

$$\text{CSAT} = \{\langle \phi \rangle : \ \phi \text{ is satisfiable CNF formula}\}$$

- Clearly, CSAT $\leq_P$ SAT (no need Cook-Levin for that).
- The *proof* of Cook-Levin can be modified to show that CSAT is NPC (take home exercise..)

Section 4

3SAT $\in \mathcal{NPC}$

3SAT

Definition 4

A Boolean formula is in k-CNF form, if it is a CNF formula, and all clauses have k literals.

Example of 3CNF: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

The language of satisfied 3CNF formulas:

$$\text{3SAT} = \{\langle \phi \rangle : \ \phi \text{ is satisfiable 3CNF formula}\}$$

- Clearly $\text{3SAT} \leq_P \text{SAT}$.
- and $\text{3SAT} \in \mathcal{NP}$.
- We show next that $\text{CSAT} \leq_P \text{3SAT}$, yielding that $\text{3SAT} \in \mathcal{NPC}$.
- Since 3SAT has more structure than CSAT, it is simpler to reduce it to other languages.

1SAT, 2SAT $\in \mathcal{P}$

Question 5

Is 1SAT $\in \mathcal{P}$?

Question 6

Is 2SAT $\in \mathcal{P}$?

Yes, see appendix...

CSAT $\leq_P$ 3SAT

- The reduction maps CNF formulae to 3CNF ones "clause by clause".
- A clause with $d \leq 3$ literals is mapped to equivalent clause with 3 literals.

  $(x_1 \vee x_2) \longmapsto (x_1 \vee x_2 \vee x_2)$
- A clause with $d > 3$ literals is mapped to $d - 2$ clauses, built on the original literals together with $(d - 3)$ new variables.

  $\phi = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee x_8) \longmapsto$
  $\phi_3 = (x_1 \vee \overline{x_2} \vee y_1) \wedge (\overline{y_1} \vee \overline{x_3} \vee y_2) \wedge (\overline{y_2} \vee x_4 \vee x_8)$

The reduction works!

Claim 7

ϕ has a satisfying assignment iff ϕ_3 does.

Proof's idea: (for case $d > 3$)
$\Leftarrow$ Given a satisfying assignment for ϕ_3, use the settings of the x_i's to get satisfying assignment for ϕ. Why does it work? For each set of clauses that came from a clause in ϕ, note that each y_i sets exactly one clause to true. There are $k-2$ clauses but only $k-3$ y_i's. So at least one clause needs to be satisfied by one of the x_j or $\overline{x_j}$ literals. Set this literal to true in ϕ.

$\Rightarrow$ An assignment satisfying ϕ, makes at least one literal per clause happy. In the "ϕ_3 clause" of this literal the new variables are under no constraints. Use the $d-2$ new variables to set the remaining $d-2$ clauses to "true".

CSAT $\leq_P$ 3SAT, cont.

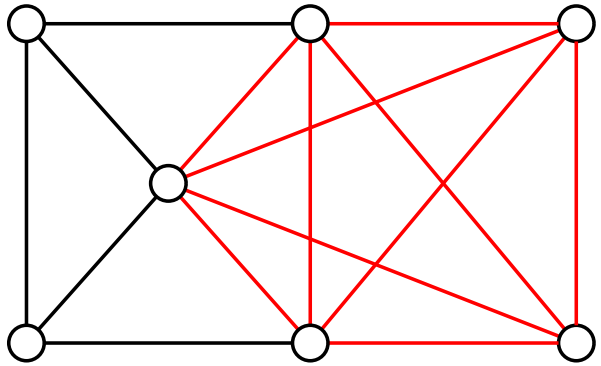
$\phi = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee x_8) \longmapsto$
$\phi_3 = (x_1 \vee \overline{x_2} \vee y_1) \wedge (\overline{y_1} \vee \overline{x_3} \vee y_2) \wedge (\overline{y_2} \vee x_4 \vee x_8)$

- Doing the above mapping to each clause of a CNF formula, we get a 3CNF that is satisfied iff the original one is.
- Since this mapping is polynomial time (?), we get CSAT $\leq_P$ 3SAT. ♣.

Section 5

Clique is NPC

3SAT $\leq_P$ CLIQUE



CLIQUE $= \{\langle G, k\rangle : G$ is an undirected graph with a k-clique$\}$

Claim 8

3SAT $\leq_P$ CLIQUE.

Since CLIQUE $\in \mathcal{NP}$, it follows that CLIQUE $\in \mathcal{NPC}$.

Since IND-SET $\in \mathcal{NP}$, and CLIQUE $\leq_P$ IND-SET, it follows that IND-SET $\in \mathcal{NPC}$.

Proof's idea: We'll construct a poly-time reduction f that maps 3*CNF* formulae ϕ to pairs $\langle G, k\rangle$ of graphs and numbers.

The function f will have the property that ϕ is satisfiable, iff G has a clique of size k.

Proving 3SAT $\leq_P$ CLIQUE

On input ϕ, a 3CNF formula, the mapping reduction is defined as follows:

let k be the number of clauses in ϕ.

We construct a graph $G = G(\phi)$, see below, and output (G, k).

The graph G is defined as follows:

- Nodes in G are organized into triples $t_1, \ldots, t_k$.
- Each triple corresponds to a clause of ϕ
- Each node in a triple corresponds to a literal in corresponding clause.

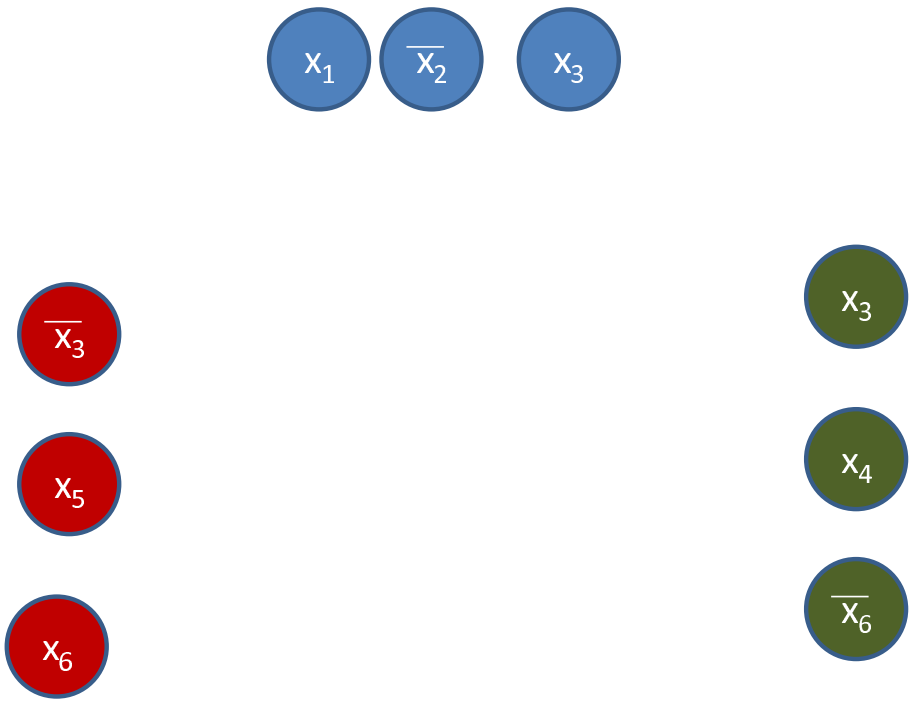
Ongoing example:

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

Nodes of G

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$

Add a node per literal

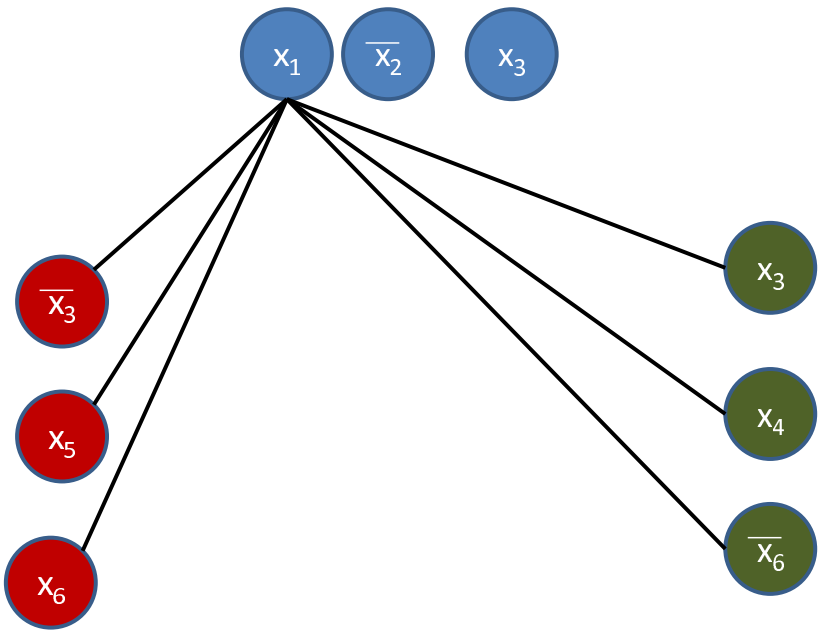


Edges of G

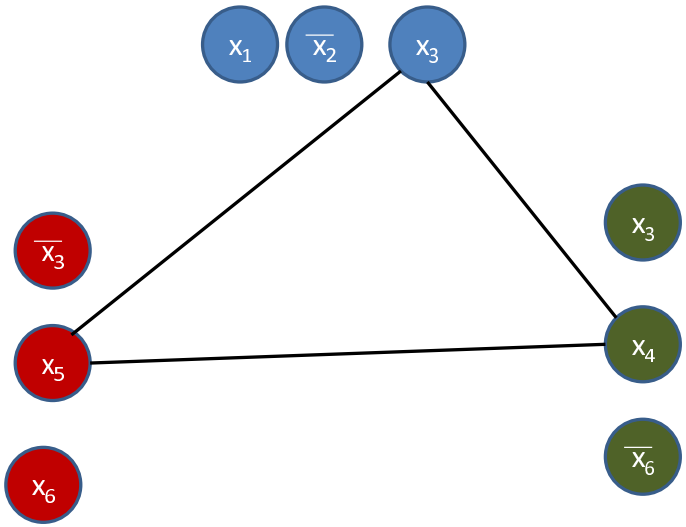
Add edges between all vertex pairs, except

- within same triple
- between contradictory literals (e.g., x_3 and $\overline{x_3}$)

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$



$\phi \in 3SAT \implies \langle G, k \rangle \in CLIQUE$

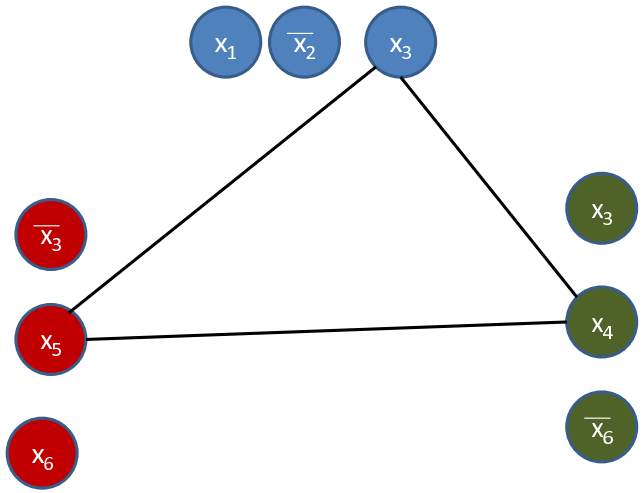


Proof: Suppose ϕ is satisfiable by an assignment ψ.
With respect to ψ:

- At least one literal is assigned to 1 in every clause (?)
- Select a 1-literal in every tuple;
- These literals can be joined by edges (?)

  Yielding a k-clique in G.

$\langle G, k \rangle \in$ CLIQUE $\Longrightarrow$ $\phi \in$ 3SAT



Proof: Suppose G has a k-clique.

- No two of the cliques nodes are in the same triple. (?)
- G has $3k$ vertexes and k clauses, so each triple has exactly one clique node.
- Assign 1 to each node in clique
- Assignment has no contradictions (?)

  Yielding a satisfying assignment to ϕ.

Recap

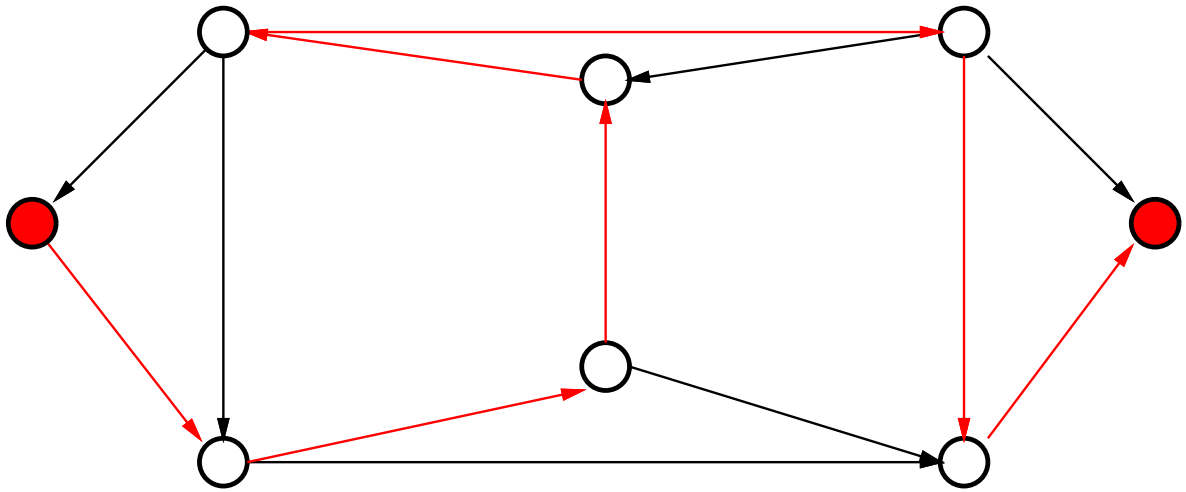
- We've constructed a poly-time computable function f.
- We saw that f has the property that $\phi \in 3SAT$ iff $f(\phi) \in CLIQUE$.
- Therefore, f is a poly-time reduction from 3SAT to CLIQUE

  $\Longrightarrow$ $3SAT \leq_P CLIQUE$. ♣

Section 6

Hamiltonian Paths and Hamiltonian Cycles are $\mathcal{NPC}$

Reminder – Hamiltonian Path



A Hamiltonian path in a directed G visits each node exactly once.

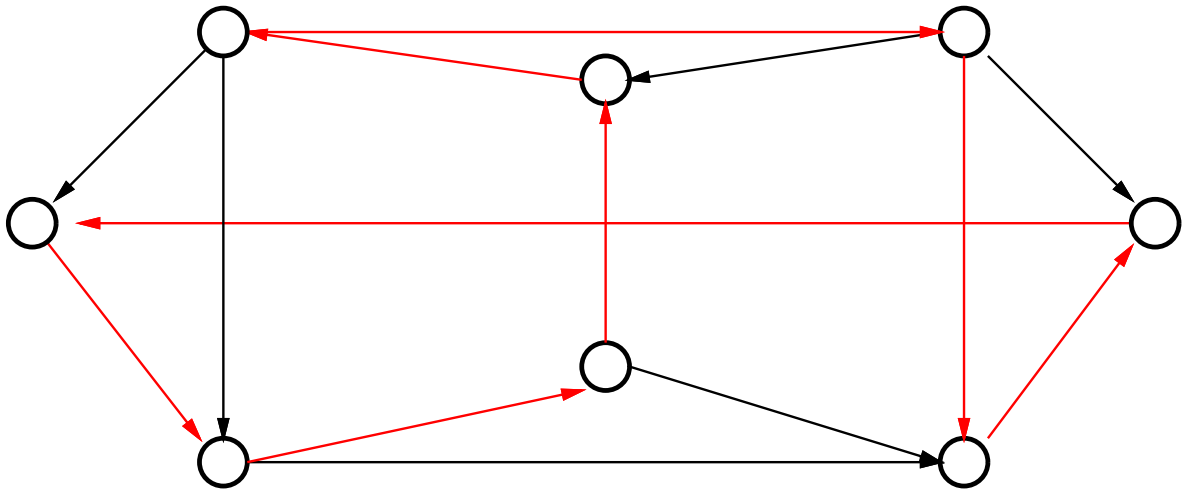
$$\text{HAMPATH} = \{\langle G, s, t\rangle : G \text{ has Hamiltonian path from } s \text{ to } t\}$$

Theorem 9

HAMPATH $\in \mathcal{NPC}$.

Soon

Hamiltonian Cycle



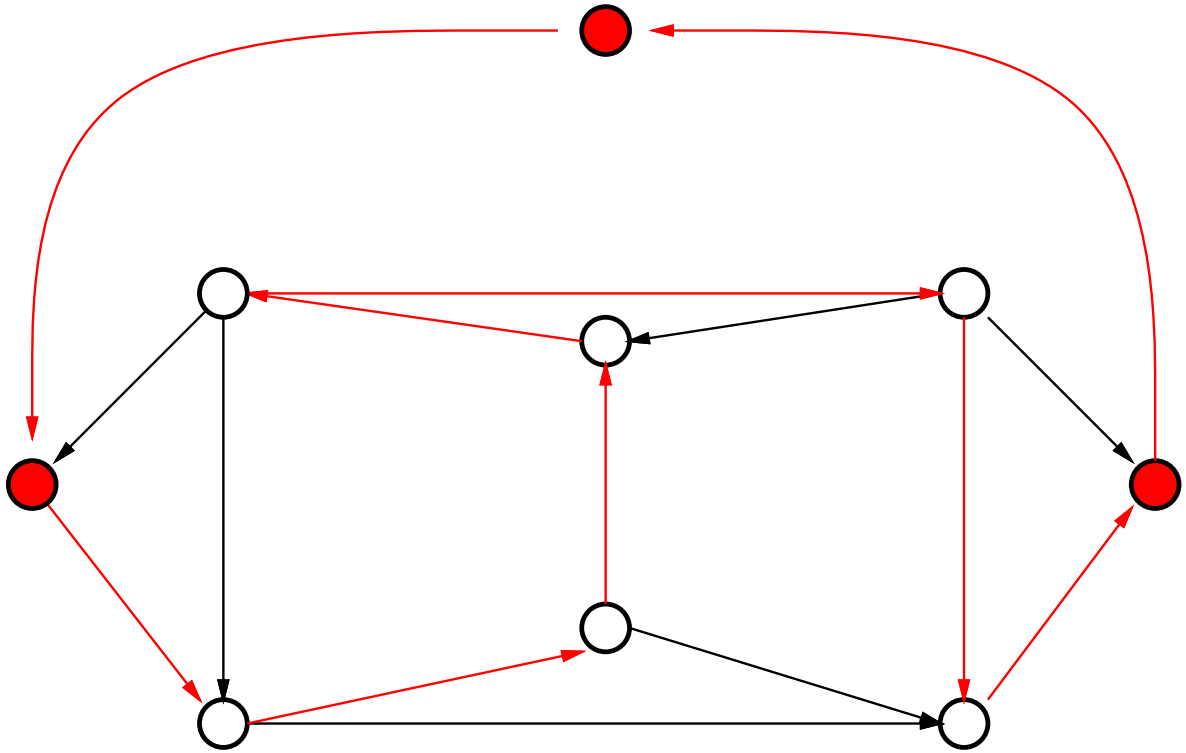
A Hamiltonian cycle in a graph is an Hamiltonian path that ends up where it starts (i.e., $s = t$).

$$\text{HAMCYCLE} = \{\langle G \rangle : G \text{ has Hamiltonian cycle}\}$$

Clearly HAMCYCLE $\in \mathcal{NP}$. We will show that HAMPATH $\leq_P$ HAMCYCLE, and deduce that HAMCYCLE $\in \mathcal{NPC}$

HAMPATH $\leq_P$ HAMCYCLE.

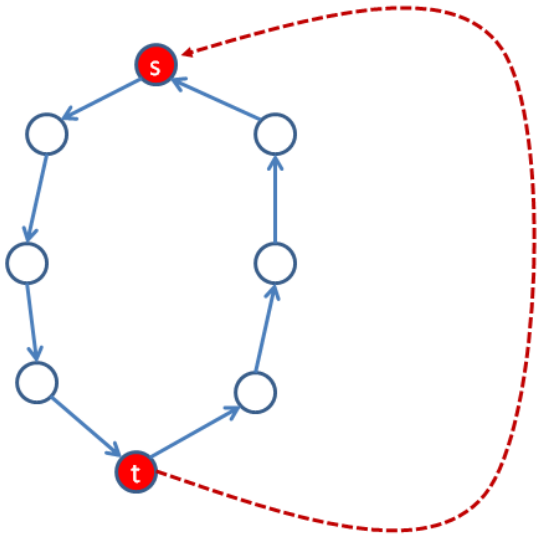
Proof's idea:



Hey, is the new vertex really needed? Why not just add an edge from t to s?

HAMPATH $\leq_P$ HAMCYCLE.

Why the new vertex really needed? Why not just add an edge from t to s?



HAMCYCLE $\leq_P$ HAMPATH.

Claim 10

HAMCYCLE $\leq_P$ HAMPATH.

Left as an easy (recommended) exercise.

Undirected Hamiltonian Circuit

$$\text{UHAMCYCLE} = \{\langle G \rangle : \ G \text{ is undirected } \& \text{ has Hamiltonian cycle}\}$$

where Hamiltonian cycle/path in an undirect graph, is defined analogously to the direct case.

Clearly $\text{UHAMCYCLE} \in \mathcal{NP}$.

It is not hard to show (see Sipser 7.55, for a similar proof) that $\text{HAMCYCLE} \leq_P \text{UHAMCYCLE}$, and deduce that $\text{UHAMCYCLE} \in \mathcal{NPC}$

CSAT $\leq_P$ HAMPATH

For any CNF formula ϕ with clauses $c_1, \ldots, c_\ell$ and variables $x_1, \ldots, x_k$, we construct a directed graph G with vertices s and t, such that

ϕ is satisfiable iff $\exists$ a directed Hamiltonian path from s to t.

thus establishing

Theorem 11

HAMPATH, HAMCYCLE $\in \mathcal{NPC}$

Turn to a separate pdf presentation:

tau-cm2017a.wdfiles.com/local--files/course-schedule/ham-reduction-new.pdf

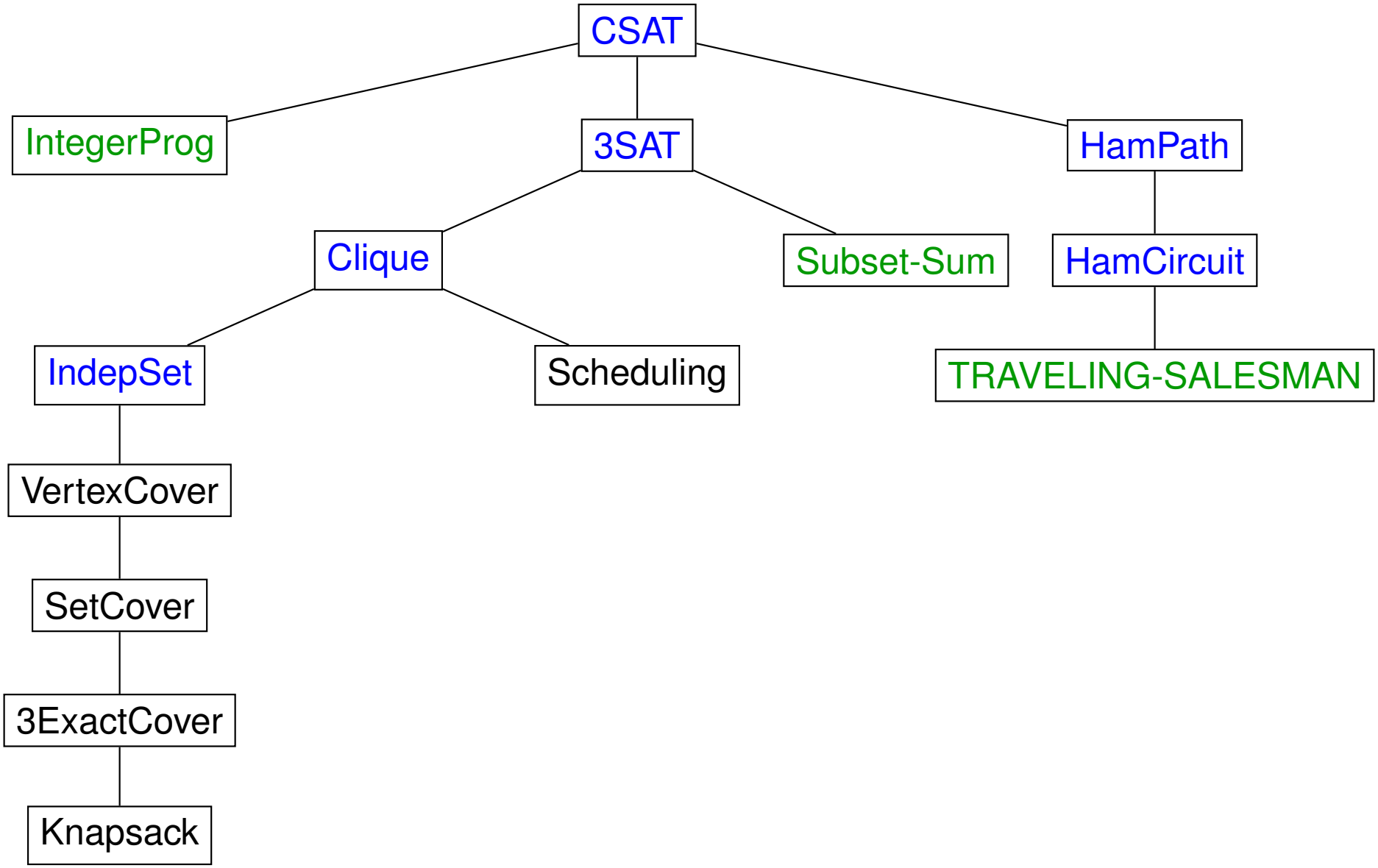
Section 7

Summary so far

Reductions summary

- We have seen that SAT and CSAT are NP-complete.
- CSAT $\leq_P$ 3SAT
- 3SAT $\leq_P$ CLIQUE
- CLIQUE $\leq_P$ IND-SET
- CSAT $\leq_P$ HAMPATH
- HAMPATH $\leq_P$ HAMCYCLE
- Hence, all of the above languages are NP-complete
- Full list of NP-complete languages contains thousands of known NP-complete problems (from combinatorics, operation research, VLSI design, computational geometry, bioinformatics, . . .).
- NP-completeness of new and of old problems is still established these days.

Chains of reductions: NPC languages



Why all the fuss about decision problems?

Don't we want to *find* the satisfying assignment?

Given an oracle for the decision problem, show that you can find an assignment in poly time!

Beyond NP-completeness?

Exciting ideas to come in Computational Complexity!!

e.g., you will learn to understand a statement like "Super-Mario Brothers is PSPACE-complete"!

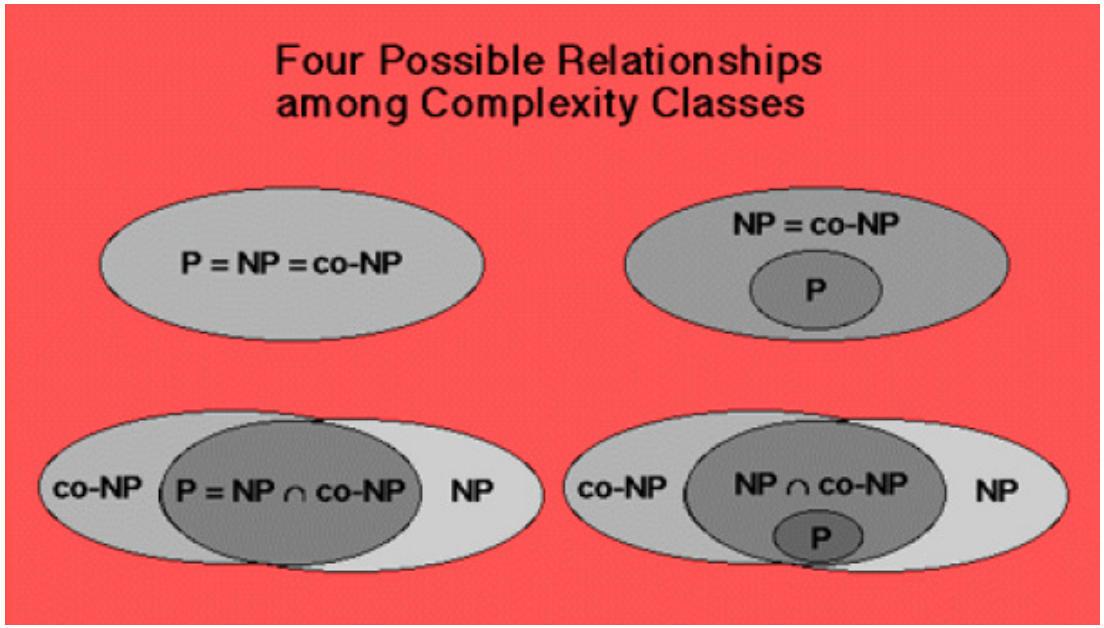
Section 8

Concluding Remarks

Ladies and Gentlemen, Boys and Girls

We have just ended the third part of the course:

Introduction to Computational Complexity (no more).



And with it, naturally, we've ended the whole course.

We hope you have enjoyed your flight, and look forward to meeting you in the future (but not in Moed B :-).

The dreaded exam

- All material covered in class and recitations, from all parts of course.
- Five “open” questions.
- You can bring and use two double sided A4 (normal size) pages marked with your name.
- See website for more info.
- Piece of cake.

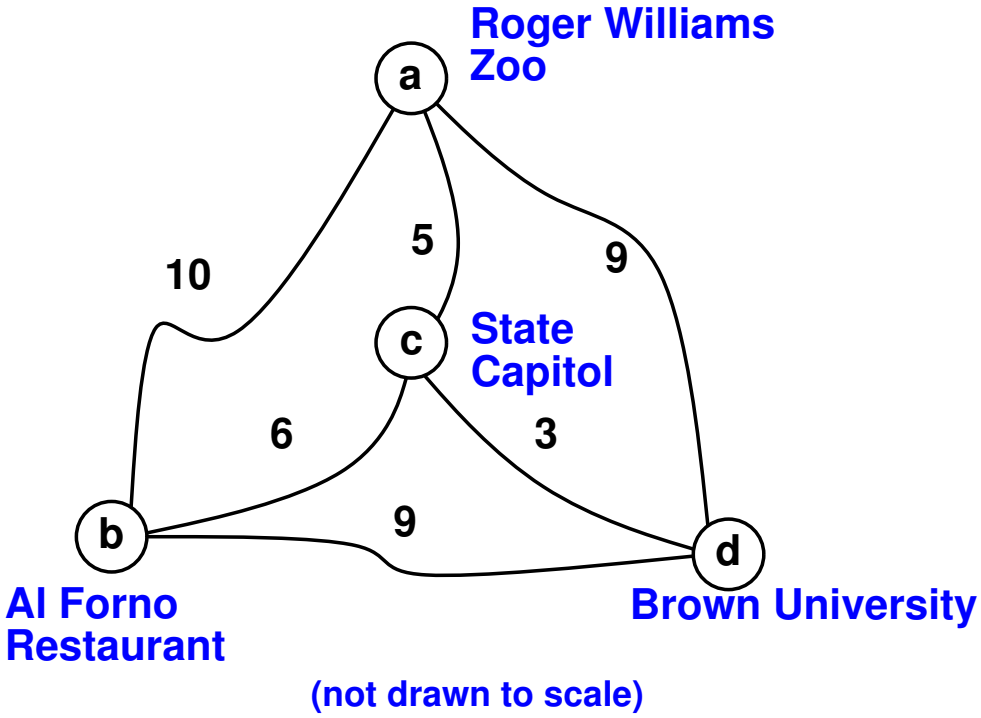
Part II

Appendix
Not Taught in Class

Section 9

Traveling Salesman is NPC

Traveling Salesman



Input: set of cities C and set of inter-city distances D

Goal: find a an hamiltonian cycle (TS tour) of total distance at most k

TRAVELING-SALESMAN $=$
$\{\langle C, D, k\rangle : (C, D)$ has a TS tour of total distance $\leq k\}$

UHAMCYCLE $\leq_P$ TRAVELING-SALESMAN

Proof: Given undirected graph $G = (V, E)$, construct traveling salesman instance.

- The cities are identical to the nodes of the original graph, $C = V$.
- The distance of going from v_1 to v_2 is 1 if $(v_1, v_2) \in E$, and 2 otherwise.
- The bound on the total distance of a tour is $k = |V|$.
- **correctness:** $\Longrightarrow$ Suppose G has a Hamiltonian cycle.
  - The distance assigned by reduction to all edges in this cycle is 1.
  - Thus in (C, D) there is a traveling salesman tour of total distance $|V| = k$ (namely $(C, D, k) \in$ TRAVELING-SALESMAN).

  $\Longleftarrow$ Suppose (C, D) has a traveling salesman tour of total distance $|V| = k$.
  - Tour cannot contain any edge of distance 2.
  - Therefore it gives a Hamiltonian cycle in G.
- **Efficiency:** reduction done in quadratic time (filling up distances for all edges of the complete graph).

Section 10

SUBSET-SUM Is NPC

SUBSET-SUM

- A collection of non-negative integers $x_1, \ldots, x_k$
- A target number t

Question 12

does some sub-collection add up to t?

Example 13

- $\langle \{4, 11, 16, 21, 27\}, 25 \rangle \in$ SUBSET-SUM because $4 + 21 = 25$.
- $\langle \{4, 11, 16, 21, 27\}, 26 \rangle \notin$ SUBSET-SUM (?)

SUBSET-SUM cont.

$$\text{SUBSET-SUM} = \{\langle S = \{x_1, \ldots, x_k\}, t\rangle : \exists\{y_1, \ldots, y_\ell\} \subseteq S: \ \sum_{j=1}^{\ell} y_j = t\}$$

Collections are sets: repetitions not allowed.

Theorem 14

SUBSET-SUM $\in \mathcal{NP}$

Proof's idea: The subset is the certificate.

Algorithm 15 (V)

On input $(\langle S = \{x_1, \ldots, x_k\}, t\rangle, c)$:
`Accept` if the following holds (otherwise `Reject`):

1. c is a collection of numbers summing to t.
2. c is a subset of S

Theorem 16

There is a pseudo-polynomial algorithm for SUBSET-SUM

Graph characterization of SUBSET-SUM

Definition 17

For $\langle S = \{x_1, \ldots, x_k\}, t\rangle$, let $G(S, t) = (V, E)$

- $V = V_0 \bigcup V_1 \ldots \bigcup V_k$, where $V_i = \{v_{i,0}, \ldots, v_{i,t}\}$
- $E = E_1 \bigcup \ldots \bigcup E_k$, where $E_i = \{(v_{i-1,j}, v_{i,j}), (v_{i-1,j}, v_{i,j+x_i}) : j \in \{0, \ldots, t\}\}$

Claim 18

$v_{i,j}$ is reachable from $v_{0,0}$ in $G(S, t)$, iff $\langle\{x_1, \ldots, x_i\}, j\rangle \in$ SUBSET-SUM.

Proof?

Pseudo-polynomial algorithm for SUBSET-SUM

Algorithm 19

Input: $\langle S, t \rangle$.

Return `TRUE` iff $v_{k,t}$ is reachable from $v_{0,0}$ in $G(S, t)$.

- Correctness
- Size of $G(S, t)$ is $O(|S| \cdot t)$, and therefore the resulting algorithm for SUBSET-SUM runs in time $\text{poly}(|S| \cdot t)$.

  Not $\text{poly}(\log_2 t)$ but $\text{poly}(t)$ (i.e., pseudo-polynomial)

3SAT $\leq_P$ SUBSET-SUM

Theorem 20

3SAT $\leq_P$ SUBSET-SUM.

Thus establishing

Theorem 21

SUBSET-SUM $\in \mathcal{NPC}$.

Proof's idea: Let ϕ be 3CNF a formula with

- variables $x_1, \ldots, x_k$
- clauses $C_1, \ldots, C_\ell$.

We "encode" ϕ into a set S containing $2k + 2\ell$ numbers in decimal notation, and a "target" number t, such that $\phi \in$ 3SAT $\iff$ $(S, t) \in$ SUBSET-SUM

Variable encoding

Variable x_i is encoded as two $(k+\ell)$-digit numbers: y_i and z_i.

Each decimal representation has two parts.

- Left-hand part, k digits: for both y_i and z_i, the i'th digit is 1, rest are 0s
- Right-hand part, ℓ digits: one digit for each clause
  - j^{th} digit of y_i is 1 if C_j contains x_i
  - j^{th} digit of z_i is 1 if C_j contains $\overline{x_i}$
  - rest are 0s

Example: $\phi = (x_1 \vee \overline{x_2} \vee \ldots) \wedge \ldots \wedge (\overline{x_1} \vee \overline{x_2} \vee \ldots)$

	1	2	...	k	C_1	...	C_ℓ
y_1	1	0	...	0	1	...	0
z_1	1	0	...	0	0	...	1
y_2	0	1	...	0	0	...	0
z_2	0	1	...	0	1	...	1

Clause encoding

Each clause C_i is encoded into two equal $(k+\ell)$-digit numbers: g_i and h_i.

In both numbers, the $(k+i)$'th digit is 1 and rest are 0.

	1	...	k	C_1	C_2	...	C_ℓ
$\vdots$							
g_1	0	...	0	1	0	...	0
h_1	0	...	0	1	0	...	0
g_2	0	...	0	0	1	...	0
h_2	0	...	0	0	1	...	0

Target encoding

The target t is assigned the following $(k + \ell)$-digit number

- first k digits are 1
- last ℓ digits are 3

	1	...	k	C_1	C_2	...	C_ℓ
$\vdots$							
t	1	...	1	3	3	...	3

Overall encoding

	1	2	...	k	C_1	C_2	...	C_ℓ
y_1	1	0	...	0	1	0	...	0
z_1	1	0	...	0	0	0	...	1
y_2	0	1	...	0	0	0	...	0
z_2	0	1	...	0	1	0	...	1
$\vdots$								
g_1	0	0	...	0	1	0	...	0
h_1	0	0	...	0	1	0	...	0
g_2	0	0	...	0	0	1	...	0
h_2	0	0	...	0	0	1	...	0
$\vdots$								
t	1	1	...	1	3	3	...	3

$S = \{y_1, z_1, y_2, z_2, \ldots, y_k, z_k, g_1, h_1, g_2, h_2 \ldots, g_\ell, h_\ell\}$.

Transformation takes $O(n^2)$ time.

ϕ is satisfiable $\implies$ $(S, t) \in$ SUBSET-SUM

Given satisfying assignment to ϕ, construct initial solution V to (S, t) as follows: If $x_i = 1$, add y_i to V; otherwise, add z_i to V.

- Each of the first k digits of $\sum_{v \in V} v$ is 1 (as in t)
- For $1 \leq j \leq \ell$, consider the $(k + j)$'th digit of $\sum_{v \in V} v$:
  - at least 1 and not more than 3 literals in C_j are 1 $\implies$ the $(k + j)$'th digit is between 1 and 3.
  - Add "enough" $\{h_i, g_i\}$ to V to bring this digit up to 3. (Doing this does not effect the other digits of $(\sum_{v \in V} v)$).

Hence, $\sum_{v \in V} v = t$. ♣

$(S, t) \in$ SUBSET-SUM $\implies$ ϕ is satisfiable

Given a solution V to (S, t).

1. Summation causes no carries
   - All digits in the numbers of S are either 0 or 1.
   - Since ϕ is in 3CNF, each "column" contains at most five 1s.
2. For each $i \in [k]$, the subset V contains one of y_i or z_i, but not both.
3. For each $i \in [\ell]$, the subset V contains at least one element of $\{y_1, z_1, \ldots, y_k, z_k\}$ whose $(k+i)$'th bit is 1:
   - Each of final ℓ "columns" sums to 3.
   - The set $\{g_i, h_i\}$ contributes at most 2
   - So at least one must comes from literal taking the value 1

The satisfying assignment.

For $i \in [k]$: if $y_i \in V$, set $x_i = 1$; otherwise, set $x_i = 0$.

The assignment satisfies ϕ because all clauses are satisfied (see Item 3)

Section 11

Integer Programming (IP) is NPC

Integer Programming (IP)

Definition 22

A linear inequality has the form

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \leq b$$

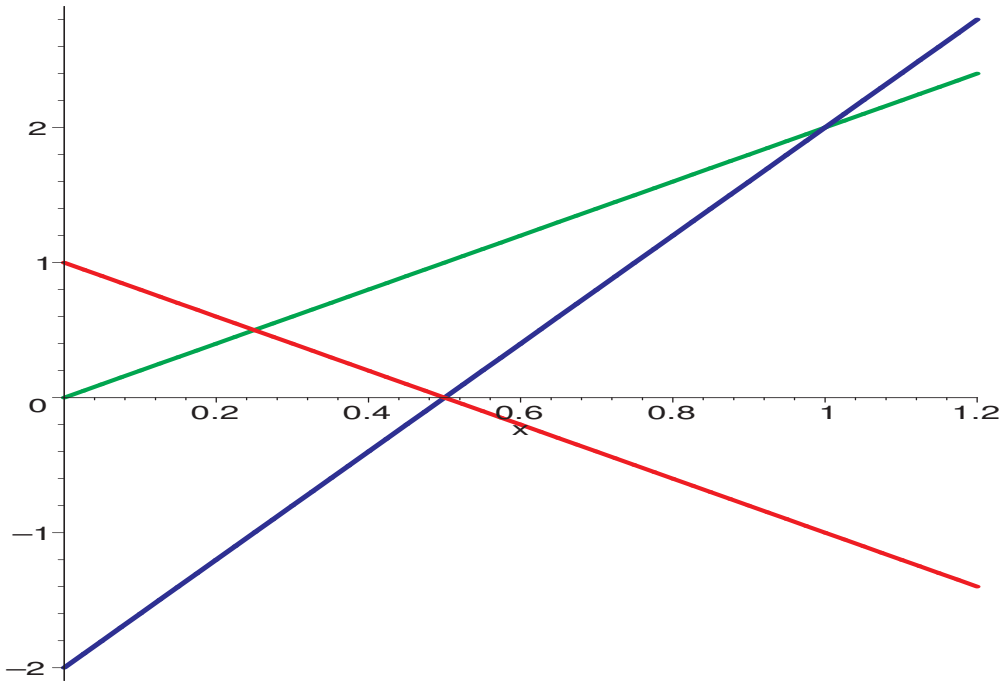
where $a_1, \ldots, a_n, b$ are real numbers, and $x_1, \ldots, x_n$ are real variables.

The Integer Programming (IP) problem]:
Input: A set of linear inequalities with integer coefficients in variables $x_1, x_2, \ldots, x_n$.
Question: Does this set has an integer solution – all x_i are integers?

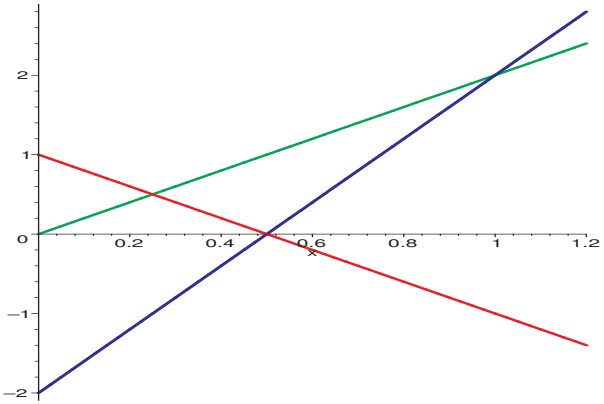
Integer Programming: Example

Consider the following system of linear inequalities

$$
\begin{aligned}
y &\leq 2x && \text{below green line} \\
-2x + 1 &\leq y && \text{above red line} \\
4x - 2 &\leq y && \text{above purple line} \\
0 \leq x &\leq 1 \\
0 \leq y &\leq 2
\end{aligned}
$$



Integer Programming: Example, 2



This set does have a unique (integer) solution: the right hand corner of the solid triangle, $(1, 2)$.

But if we change the constraint on y to $0 \leq y \leq 1$, then we'd have no solution with integer coordinates, even though there are many solutions with rational, or real, coordinates.

The Language IP

The language of integer linear inequalities with integer solution:

$$\text{IP} = \{\langle e_1, \ldots, e_m \rangle : \\ e_1, \ldots, e_m \text{ are integer linear inequalities with (joint) integer solution}\}$$

Theorem 23

$\text{IP} \in \mathcal{NPC}$.

IP $\in \mathcal{NP}$

Question 24

Do we always have a small enough certificate?

Consider the following equations

$$\begin{aligned} x_0 &= 1 \\ x_1 &= 2x_0 \\ &\vdots \\ x_m &= 2x_{m-1} \end{aligned}$$

The solution $x_m = 2^m$!

Question 25

Solving a linear set of m equations with m unknowns: $Ax = b$, where $|a_{i,j}|, |b_i| < M$. How large can $|x_i|$ get?

Answer: Recall Cramer's Rule, using determinants. $|A| < m! \cdot M^m$. (Same holds for inequalities.)
Hence, solution size is polynomial bounded.

CSAT $\leq_P$ IP

$$\text{CSAT} = \{\langle\phi\rangle : \ \phi \text{ is a satisfiable CNF Boolean formula}\}$$

Let ϕ be a CNF formula with ℓ clauses and k variables $x_1, \ldots, x_k$.

We reduce ϕ to an IP instance E with $2k$ variables $x_1, y_1, \ldots, x_k, y_k$, $\ell + 2k$ linear inequalities, and k linear equalities.

- Each x_i in ϕ corresponds to the variable x_i in E.
- Each $\overline{x_i}$ in ϕ corresponds to the variable y_i in E.
- For each i, we add E the inequalities $x_i \geq 0$, $y_i \geq 0$, and the equality $x_i + y_i = 1$ (what do these three express?)
- For each clause C of ϕ, we add E the inequality $\sum_{z \in C} z \geq 1$ (what does this inequality express?)

  For example, $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$ is translated to $x_1 + y_2 + y_3 + x_4 \geq 1$.

Reduction Example

$\phi = (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$ is translates to

$$
\begin{aligned}
x_1 + y_2 + y_3 + x_4 &\geq 1 \\
x_3 + y_5 + x_6 &\geq 1 \\
x_3 + y_6 &\geq 1 \\
x_1 \geq 0,\ y_1 \geq 0,\ x_1 + y_1 &= 1 \\
x_2 \geq 0,\ y_2 \geq 0,\ x_2 + y_2 &= 1 \\
x_3 \geq 0,\ y_3 \geq 0,\ x_3 + y_3 &= 1 \\
x_4 \geq 0,\ y_4 \geq 0,\ x_4 + y_4 &= 1 \\
x_5 \geq 0,\ y_5 \geq 0,\ x_5 + y_5 &= 1 \\
x_6 \geq 0,\ y_6 \geq 0,\ x_6 + y_6 &= 1
\end{aligned}
$$

CSAT $\leq_P$ IP: Validity (sketch)

Should show

1. Reduction, g, is poly-time computable
2. $\phi \in \text{CSAT} \implies g(\phi) \in \text{IP}$
3. $g(\phi) \in \text{IP} \implies \phi \in \text{CSAT}$.

1. Poly time: easy (verify details!).
2. Suppose $\phi \in \text{CSAT}$. Take a satisfying assignment.
   If $x_i = 1$, in $g(\phi)$ assign $x_i = 1, y_i = 0$.
   If $x_i = 0$, in $g(\phi)$ assign $x_i = 0, y_i = 1$.
   - "Sanity check" constraints (i.e., $x_i \geq 0$, $y_i \geq 0$, $x_i + y_i = 1$) are satisfied.
   - "Clause constraints" (i.e., $\sum_{z \in C} z \geq 1$) are satisfied due to at least one literal satisfied in each clause.
   - Implying $g(\phi) \in \text{IP}$.
3. $g(\phi) \in \text{IP} \implies \phi \in \text{CSAT}$, is similar.

Section 12

KNAPSACK

KNAPSACK

- A collection of non-negative integers $x_1, \ldots, x_k$ (size)
- A collection of non-negative integers $y_1, \ldots, y_k$ (benefit)
- A capacity B
- A target number t

Question: does some $S \subseteq \{1, \ldots, k\}$ has $\sum_{i \in S} x_i \leq B$ and $\sum_{i \in S} y_i \geq t$?

Example 26

- $(\{4, 11, 16, 21, 27\}, \{7, 3, 9, 5, 35\}, B = 20, t = 16) \in \mathit{KNAPSACK}$ because $S = \{1, 3\}$.

KNAPSACK cont.

$$\text{KNAPSACK} = \{\langle\{x_1,\ldots,x_k\},\{y_1,\ldots,y_k\},B,t\rangle : \exists S \subseteq \{1,\ldots,k\} : \sum_{i\in S} x_i \leq B \wedge \sum_{i\in S} y_i \geq t\}$$

Theorem 27

$\text{KNAPSACK} \in \mathcal{NP}$

Proof's idea: The subset is the certificate.

Algorithm 28 (V)

On input $(\langle\{x_1,\ldots,x_k\},\{y_1,\ldots,y_k\},B,t\rangle, c)$:
`Accept` if the following holds (otherwise `Reject`):

1. $\sum_{i\in c} x_i \leq B$
2. $\sum_{i\in c} y_i \geq t$.
3. c is a subset of $[1,\ldots,k]$

Pseudo-polynomial algorithm for KNAPSACK

Theorem 29

There is a pseudo-polynomial algorithm for KNAPSACK

Algorithm 30 (KNAPSACK)

Input: $\{x_1, \ldots, x_k\}, \{y_1, \ldots, y_k\}, B, t$:

1. Set $A(0,0) = 0$ and $A(0,p) = \infty$ for $p \neq 0$.
2. For $i = 1$ to k:

   For $p = 0$ to t:

   Set $A(i,p) = \min\{A(i-1,p), x_i + A(i-1,p-y_i)\}$.
3. `Accept` if $t \leq \max\{p\colon A(k,p) \leq B\}$.

- Running time $O(k \cdot t)$
- Not $\text{poly}(\log_2 t)$ but $\text{poly}(t)$ (i.e., pseudo-polynomial)

Section 13

2SAT $\in P$

A graph characterization for $2CNF$

Definition 31

For $2CNF$ formula ϕ with variables $x_1, \ldots, x_k$, define a directed graph $G(\phi) = (V, E)$ as

- $V = \{x_1, \overline{x_1}, \ldots, x_k, \overline{x_k}\}$
- $E = \{(\overline{\ell}, h), (\overline{h}, \ell) \colon (\ell \vee h) \in \phi\}$ (taking $\overline{\overline{x}} = x$).

Claim 32

$G(\phi)$ contains path from ℓ to $h \implies$ in any satisfying assignment of ϕ with $\ell = 1$, it holds that $h = 1$.

Proof?

Claim 33

$G(\phi)$ contains path from ℓ to $h \implies G(\phi)$ contains path from $\overline{h}$ to $\overline{\ell}$

Proof?

A graph characterization for $2CNF$, cont

Definition 34

A variable x in $2CNF$ ϕ is bad, if $\exists$ paths in $G(\phi)$ from x to $\overline{x}$ and from $\overline{x}$ to x.

Claim 35

A $2CNF$ formula is satisfiable iff it contains no bad variables.

- ϕ has bad variable $\implies$ ϕ is unsatisfiable. Proof? by Claim 32
- ϕ has no bad variables $\implies$ ϕ is satisfiable. Proof?

A graph characterization for $2CNF$, cont.

Definition 36

A variable x in $2CNF$ ϕ is bad, if $\exists$ paths in $G(\phi)$ from x to $\overline{x}$ and from $\overline{x}$ to x.

Claim 37

A $2CNF$ formula is satisfiable iff it contains no bad variables.

Algorithm 38

While ϕ has unassigned variables:

1. Pick unassigned literal $\ell \in V$ that has no path from ℓ to $\overline{\ell}$
2. Assign 1 to all literals reachable from ℓ, and 0 to their negations.
3. Recompute graph for "reduced formula"

Can there be conflicts (i.e., x and $\overline{x}$ assigned the same value)?

- Same iteration? Assume $\ell \rightsquigarrow h \rightsquigarrow \overline{h}$. $\implies$ $h \rightsquigarrow \overline{h} \rightsquigarrow \overline{\ell}$ $\implies$ $\ell \rightsquigarrow \overline{\ell}$.
- Different iterations? Removing edges cannot add bad variables.

Final assignment satisfies ϕ. Assume h is set to 1, then all clauses it participates in are satisfied.

Poly-time algorithm for 2SAT

Algorithm 39 (TwoSatSolver)

Input: *2CNF* ϕ
Return `TRUE` if there exist no bad variables in ϕ:

- Efficiency?
- Correctness?