CS3230

eric_han@nus.edu.sg
https://eric-han.com

Computer Science

T12 – Week 13

Reductions and Com. Complexity (cont.)

CS3230 – Design and Analysis of Algorithms

Admin

1. Please check:
   - `Tutorials - Attendance & Participation`
   - `Assignments - Best Seven`
2. Last lecture this week – Revision & Gradient Descent.
3. Some parting advice: $ is important but not everything.
   - GES 2016: $3500, $4000, $5000
   - CS3230 skills can help you land a good technical job.
4. FYP - AI & Machine Learning
5. CS3230 Practical Exercises

GES 2024

NUS GES 2024 Numbers

- Bachelor of Computing (Computer Science)
  - 25/50/75 percentile: $6500, $5600, $7500
  - Mean: $6788
  - Employed: 89.1%

Student Feedback on Teaching (SFT)

NUS Student Feedback https://blue.nus.edu.sg/blue/:

- Don't Mix module/grading/project feedback - **feedback only for teaching**.
- Feedback is confidential to university and anonymous to us.
- Feedback is optional but highly encouraged.
- Past student feedback improves teaching; see https://www.eric-han.com/teaching
  - ie. Telegram access, More interactivity.
- Your feedback is important to me, and will be used to improve my teaching.
  - Good > Positive feedback > Encouragement
    - Teaching Awards (nominate)
    - Steer my career path
  - Bad > Negative feedback (nicely pls) > Learning
    - Improvement
    - Better learning experience

Lecture Review

- **P**: solvable in polynomial time
- **NP**: verifiable in polynomial time
  Prove `SOMETHING` is in NP:
  1. Provide a certificate for 'Yes' instance.
  2. Verify certificate in polynomial time.
- **NP-hard**: polynomial-time reducible from all NP problems
  Prove `SOMETHING` is in NP-hard:
  1. Show it's at least as hard as a known NP-hard problem.
  2. Show reduction: `A-PROVEN-NP-HARD-PROBLEM` $\leq_p$ `SOMETHING`.
- **NP-complete**: both in NP and NP-hard
  Prove `SOMETHING` is in NP-complete:
  1. Prove it's in NP.
  2. Prove it's NP-hard.

Question 1 [P1]/[G]

Which of the following imply P = NP?

a. There is a problem in P that is also NP-complete

b. There is a problem in P that is also in NP

c. There is a problem in NP that is also NP-hard

Answer 1a

- Suppose there exists a problem in P that is also in NP-complete
- Since all NP-complete problems reduce to each other,
- **every** NP problem would also be solvable in polynomial time.
- So P = NP.

However, as of 2024, nobody has proven this yet!

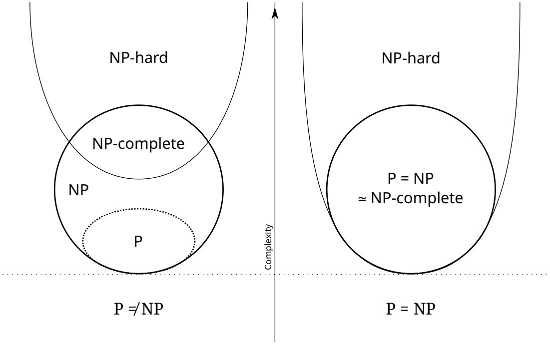

Figure 1: NP-complete with different assumptions (Wiki)

Answer 1b

- Every problem in P is also in NP (i.e., $\mathsf{P} \subseteq \mathsf{NP}$).
- $\mathsf{P} \subseteq \mathsf{NP}$ does **not** imply $\mathsf{P} = \mathsf{NP}$.

Answer 1b

- Every problem in P is also in NP (i.e., $\mathsf{P} \subseteq \mathsf{NP}$).
- $\mathsf{P} \subseteq \mathsf{NP}$ does **not** imply $\mathsf{P} = \mathsf{NP}$.

Answer 1c

- A problem in NP and NP-hard is a NP-complete problem.
- Does **not** imply $\mathsf{P} = \mathsf{NP}$.

Subset Sum

Given a multiset S of n integers (usually non-negative), $S = \{S_1, S_2, \ldots, S_n\}$, and a target integer W. Is there exists a subset $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} S_i = W$?

(See Subset Sum on Visualgo)

Example

- Given $n = 5$, $S = \{5, 1, 5, 1, 4\}$, and $W = 7$
- YES-instance, with certificate indices $\{0, 1, 3\}$ (values $\{5, 1, 1\}$), summing to 7.

We want to prove `SUBSET-SUM` is NP-complete.

Question 2 [P2]

Prove that `SUBSET-SUM` is in NP.

Question 2 [P2]

Prove that `SUBSET-SUM` is in NP.

Answer

- **Certificate**: Subset I itself (the indices of S summing to W).
- **Verification**: Check if $\sum_{i \in I} S_i = W$ in $O(n)$ time.

Question 2 [P2]

Prove that `SUBSET-SUM` is in NP.

Answer

- **Certificate**: Subset I itself (the indices of S summing to W).
- **Verification**: Check if $\sum_{i \in I} S_i = W$ in $O(n)$ time.

Pro-tips

1. Do **NOT** leave this question blank in the final.
2. Your verifier only needs to run in polynomial time w.r.t. input size—it does **NOT** need to be the fastest possible.

Question 3 [G]

Prove that `SUBSET-SUM` is NP-hard.

Hint: Reduce from `3-SAT` (see on Visualgo).

Question 3 [G]

Prove that `SUBSET-SUM` is NP-hard.

Hint: Reduce from `3-SAT` (see on Visualgo).

Answer

Given a `3-SAT` instance, we can reduce it to a `SUBSET-SUM` instance.
We build this reduction incrementally, with different attempts:

1. Modelling just variables.
2. Modelling Variables and Constraints.
3. Modelling Variables, Constraints, and Literals

Attempt 1: Modelling just variables

Given a `3-SAT` instance with variables $\{x_1, \ldots, x_N\}$, we use symbols (literals) v_i and v_i':

- v_i: assign $x_i =$ **TRUE**
- v_i': assign $x_i =$ **FALSE**

To ensure single assignment per variable, we use a base-10 bitmask:

- Set target sum W as a bitmask $1 \ldots 1_{10}$ (length N).
- Each literal corresponds to an integer in set S, with the appropriate bit set to 1.

This guarantees each variable is assigned exactly once but doesn't yet consider clause constraints.

We need to consider constraints.

Example

	x_1	x_2	x_3	Value in S	Meaning
$v_1 =$	1	0	0	100_{10}	x_1 = TRUE
$v_1' =$	1	0	0	100_{10}	x_1 = FALSE
$v_2 =$	0	1	0	10_{10}	x_2 = TRUE
$v_2' =$	0	1	0	10_{10}	x_2 = FALSE
$v_3 =$	0	0	1	1_{10}	x_3 = TRUE
$v_3' =$	0	0	1	1_{10}	x_3 = FALSE
$W =$	1	1	1	111_{10}	

Table 1: Example with three variables $\{x_1, x_2, x_3\}$: Integers $S = \{100_{10}, 100_{10}, 10_{10}, 10_{10}, 1_{10}, 1_{10}\}$ with target sum $W = 111_{10}$.

Attempt 2: Modelling Variables and Constraints

Now, include the constraints (clauses) in the `3-SAT` formula Φ:

- When assigning TRUE/FALSE for x_i,
- we can know which clause will be true.

Use the bitmask approach again,

- setting a bit to 1 if literal satisfies clause C_j.
- In polynomial time, we can verify all M clauses have at least one TRUE literal.

However, determining a suitable W for clauses $(C_1, \dots, C_M)$ isn't straightforward, as the number of satisfied literals varies (1–3 per clause).

We can't test multiple values of W within `SUBSET-SUM` (eg. Testing 1231 vs. 3212).

Example

	x_1	x_2	x_3	C_1	C_2	C_3	C_4	Value in S	Meaning
$v_1 =$	1	0	0	1	0	0	1	1001001_{10}	
$v_1' =$	1	0	0	0	1	1	0	1000110_{10}	$x_1 =$ FALSE satisfy C_2 and C_3
$v_2 =$	0	1	0	0	0	0	1	100001_{10}	
$v_2' =$	0	1	0	1	1	1	0	101110_{10}	$x_2 =$ FALSE satisfy C_1, C_2, and C_3
$v_3 =$	0	0	1	0	0	1	1	10011_{10}	$x_3 =$ TRUE satisfy C_3 and C_4
$v_3' =$	0	0	1	1	1	0	0	11100_{10}	
$W =$	1	1	1	?	?	?	?	$111????_{10}$	$C_1/C_2/C_3/C_4$ has $1/2/3/1$ satisfied literals

Table 2: Example of `3-SAT` from CLRS:
$\Phi = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$. This is a YES-instance with certificate $x_1 = x_2 =$ FALSE and $x_3 =$ TRUE.

Attempt 3: Modelling Variables, Constraints, and Literals

For each of the M clauses in the `3-SAT` instance, introduce two slack symbols:

- s_j: adds slack $+1$ to clause C_j
- s_j': adds slack $+2$ to clause C_j

This ensures variables, constraints, and literals are fully modelled, and sets a clear target sum for each clause, completing the reduction to `SUBSET-SUM`.

Example

	x_1	x_2	x_3	C_1	C_2	C_3	C_4	Value in S	Meaning
$v_1 =$	1	0	0	1	0	0	1	1001001_{10}	
$v_1' =$	1	0	0	0	1	1	0	1000110_{10}	$x_1 =$ FALSE satisfy C_2 and C_3
$v_2 =$	0	1	0	0	0	0	1	100001_{10}	
$v_2' =$	0	1	0	1	1	1	0	101110_{10}	$x_2 =$ FALSE satisfy C_1, C_2, and C_3
$v_3 =$	0	0	1	0	0	1	1	10011_{10}	$x_3 =$ TRUE satisfy C_3 and C_4
$v_3' =$	0	0	1	1	1	0	0	11100_{10}	
$s_1 =$	0	0	0	1	0	0	0	1000_{10}	Take both +1 slack
$s_1' =$	0	0	0	2	0	0	0	2000_{10}	and +2 slacks for C_1
$s_2 =$	0	0	0	0	1	0	0	100_{10}	
$s_2' =$	0	0	0	0	2	0	0	200_{10}	Take only +2 slacks for C_2
$s_3 =$	0	0	0	0	0	1	0	10_{10}	Take only +1 slack for C_3
$s_3' =$	0	0	0	0	0	2	0	20_{10}	
$s_4 =$	0	0	0	0	0	0	1	1_{10}	Take both +1 slack
$s_4' =$	0	0	0	0	0	0	2	2_{10}	and +2 slacks for C_4
$W =$	1	1	1	4	4	4	4	1114444_{10}	$C_1/C_2/C_3/C_4$ has target $4/4/4/4$

Table 3: Example of `3-SAT` from CLRS.

Summary

We've reduced (any) Φ from `3-SAT` to a corresponding `SUBSET-SUM` instance. Some technical details (omitted, see CLRS 34.5.5) include:

- Reduction runs in polynomial time.
- YES-instance of `3-SAT` $\implies$ YES-instance of `SUBSET-SUM`.
- YES-instance of `SUBSET-SUM` $\implies$ YES-instance of `3-SAT`.

Pro-tip

This detailed proof seems long, but NP-completeness proofs can be short—don't skip such questions! (See next question.)

Find Family

- **Undirected Bipartite Graph**: $G = (L \cup R, E)$
  Bipartite graph has disjoint vertex sets L, R with edges between L and R.
- **Siblings**: $u, v \in L$
  If there exists a vertex $r \in R$ such that edges (u, r) and (v, r) both exist.
- **Family**: $F \subseteq L$
  A subset is a **family** if for all distinct vertices in $u, v \in F$ are siblings.
- **Decision Problem (`FIND-FAMILY`)**:
  Given a bipartite graph $G = (L \cup R, E)$ and integer k, does there exist a **family** of size $\geq k$?

Example

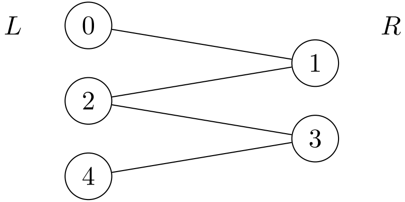

Figure 2: Example bipartite graph with $L = \{0, 2, 4\}$ and $R = \{1, 3\}$. Here, 0 and 2 are siblings, 2 and 4 are siblings, but $\{0, 2, 4\}$ is not a family since 0 and 4 are not siblings.

Question 4

Prove that `FIND-FAMILY` is in NP.

Question 4

Prove that `FIND-FAMILY` is in NP.

Answer

- **Certificate**: The family set F itself.
- **Verification**: For each pair $u, v \in F$, check if there exists $r \in R$ adjacent to both. Runs in $O(|F| \cdot |R|) = O(|L|^2 \cdot |R|)$ — polynomial in input size.

Question 5 [P3]/[G]

Prove that `FIND-FAMILY` is NP-hard.

Question 5 [P3]/[G]

Prove that `FIND-FAMILY` is NP-hard.

Answer

1. Decide which ... NP-hard problem to use.
2. Given ..., we transform to `FIND-FAMILY`

Decide which ... NP-hard problem to use

We aim to solve an instance of another NP-hard problem using `FIND-FAMILY`.
Family definition: A subset $F \subseteq L$ is a **family** if every pair $u, v \in F$ are siblings.

Problem	Condition on subset F
`FIND-FAMILY`	'... for every pair of vertices in F, they are *siblings*'
`CLIQUE`	'... for every pair of vertices in F, they are *adjacent*'

Try reduction: `CLIQUE` $\leq_p$ `FIND-FAMILY`

Given ..., we transform to `FIND-FAMILY`

Polynomial time reduction

1. Given a `CLIQUE` instance $G = (V, E)$: is there a clique of size $\geq k$?
2. Construct a `FIND-FAMILY` instance with $L = V$ and $R = E$.
3. For each edge $(u, v) \in E$, connect u and v in L to node (u, v) in R.

Total time to build the bipartite graph is polynomial $O(|V| + |E|)$.

Proof `CLIQUE` $\iff$ `FIND-FAMILY`

Straightforward to argue YES-instance of `CLIQUE` corresponds ($\iff$) to a YES-instance of `FIND-FAMILY`:

- ($\Rightarrow$) Suppose G has a clique of size k. $\cdots$
- ($\Leftarrow$) Suppose there is a family of size k in the bipartite graph. $\cdots$

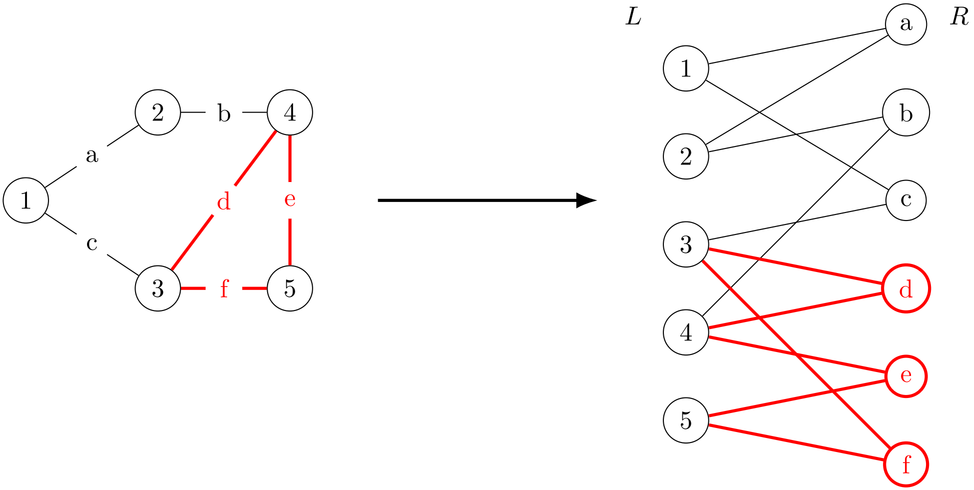

Figure 3: Edges d, e, f (in red) form a size-3 clique $\{3, 4, 5\}$ in G, which corresponds to a size-3 family $F = \{3, 4, 5\}$ in the `FIND-FAMILY` instance.

Congratulations

Give yourself a pat on the back for completing CS3230:

- It has been a privilege teaching you this semester!
- Where to go from here:
  - Hate theory: possibly in your life, most difficult theory course
  - Dislike theory, love concepts: CS3233 - Competitive Programming, ICPC
  - Love theory:
    - CS5339 - Theory and Algorithms for Machine Learning (scarlett)
    - Algorithms & Theory focus area