NUS | Computing
National University of Singapore

CS3230
eric_han@nus.edu.sg
https://eric-han.com
Computer Science

T11 – Week 12

Reductions and Computational Complexity

CS3230 – Design and Analysis of Algorithms

Admin

1. Tutorial scores will be computed weekly - `Tutorials - Attendance & Participation`, please check.
2. Assignment scores will be computed soon / weekly - `Assignments - Best Seven`, please check when it is ready.

Further Explanation

- **Exchange Argument** - Any optimal solution can be converted into greedy optimal solution.
  - Intuition is to make the optimal solution 'unique'.
  - For our party problem in the assignment, for any optimal party configuration, we can replace with the latest time b_i.
- **Optimal Substructure** - An optimal solution can be built from optimal solutions of its subproblems.
  - Intuition that solving the smaller problems, allows us to solve the any larger problem with the smaller problem.
  - For our party problem, the optimal solution to a sub-sequence of students would contribute directly to the optimal solution of a larger set.

Lecture Review

Revisiting Time Complexity

Time complexity is actually computed based on the input size.

- **Example 1: Sorting**
  Input: N (32-bit) Integers.
  Input Size: $O(32 \cdot N) = O(N)$.
  Merge sort algorithm runs in $O(N \log N)$
  - polynomial w.r.t. input size.
- **Example 2: Fibonacci**
  Input: One single Integer, which has value N.
  Input Size: $O(\log N)$ for just that one Integer.
  DP algorithm (that sums the last two Fibonacci values) runs in $O(N)$
  - this is **not** polynomial w.r.t. input size, as there is an exponential gap from $\log N$ to N
  - but it is **pseudopolynomial** considering the input is N and DP runtime as $O(N)$.

Reductions

Key Idea: To solve **A**, maybe we can translate/reduce problem **A** to **B**.

```
Solve_A(instance_of_A):
    instance_of_B = translate_A_to_B(instance_of_A)
    solution_of_B = Solve_B(instance_of_B)
    solution_of_A = translate_B_to_A(solution_of_B)
    return solution_of_A
```

We call this **polynomial time reduction** if both sub-functions `translate_A_to_B` and `translate_B_to_A` run in polynomial time. This process is denoted as $A \leq_p B$.

Decision vs Optimization Problems

- **Decision Problem**: A problem where the output is Boolean (YES/NO).
- **Optimization Problem**: A problem where we aim to optimize the output. Synonyms: maximize, minimize, most optimal, longest, shortest, etc.

Question 1 [G]/[P1]

GRAPH-COLORING is the problem of assigning colors to vertices of a graph such that no two adjacent vertices share the same color.

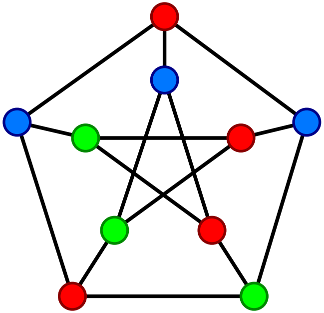

Figure 1: Graph Coloring

Which statement(s) is/are True?

a. If we can solve the **optimization** problem for GRAPH-COLORING in polynomial time, we can solve the **decision** problem in polynomial time.

b. If we can solve the **decision** problem for GRAPH-COLORING in polynomial time, we can solve the **optimization** problem in polynomial time.

c. If the **decision** problem for GRAPH-COLORING **cannot** be solved in polynomial time, the **optimization** problem **cannot** be solved in polynomial time.

d. If the **optimization** problem for GRAPH-COLORING **cannot** be solved in polynomial time, the **decision** problem **cannot** be solved in polynomial time.

Answer 1a

True: If we can solve the **optimization problem**, we can solve the **decision problem**.

- Simply determine the **minimum** number of colors required (chromatic number).
- If this minimum is $\leq k$, return **YES**; otherwise, return **NO**.

Answer 1a

True: If we can solve the **optimization problem**, we can solve the **decision problem**.

- Simply determine the **minimum** number of colors required (chromatic number).
- If this minimum is $\leq k$, return **YES**; otherwise, return **NO**.

Answer 1b

True: If we can solve the **decision problem**, we can solve the **optimization problem**.

- Test for increasing color counts until the smallest valid number is found.
- A more efficient approach is **binary search** on the number of colors.

[1] The contrapositive of $P \to Q$ is $\neg Q \to \neg P$.

Answer 1a

True: If we can solve the **optimization problem**, we can solve the **decision problem**.

- Simply determine the **minimum** number of colors required (chromatic number).
- If this minimum is $\leq k$, return **YES**; otherwise, return **NO**.

Answer 1b

True: If we can solve the **decision problem**, we can solve the **optimization problem**.

- Test for increasing color counts until the smallest valid number is found.
- A more efficient approach is **binary search** on the number of colors.

Answer 1c

True: This is the contrapositive[1] of (a).

[1] The contrapositive of $P \to Q$ is $\neg Q \to \neg P$.

Answer 1a

True: If we can solve the **optimization problem**, we can solve the **decision problem**.

- Simply determine the **minimum** number of colors required (chromatic number).
- If this minimum is $\leq k$, return **YES**; otherwise, return **NO**.

Answer 1b

True: If we can solve the **decision problem**, we can solve the **optimization problem**.

- Test for increasing color counts until the smallest valid number is found.
- A more efficient approach is **binary search** on the number of colors.

Answer 1c

True: This is the contrapositive[1] of (a).

Answer 1d

True: This is the contrapositive of (b).

[1] The contrapositive of $P \to Q$ is $\neg Q \to \neg P$.

Question 2 [G]/[P2]

PARTITION versus BALL-PARTITION:

- **Partition**: Given positive integers S, can it be split into two subsets with equal sum?
  - Eg. $S = \{18, 2, 8, 5, 7, 24\} \to S_1 = \{18, 2, 5, 7\}, S_2 = \{8, 24\}$ (sum = 32).
- **Ball-Partition**: Given k balls, can they be evenly split into two boxes? (is k even?)
  - Eg. $k = 4$, Partition as $\{2, 2\}$.

Show that PARTITION $\leq_p$ BALL-PARTITION using the following transformation A:

1. From the problem PARTITION, we are given a set of positive integers S.
2. Define k as the total sum of all integers in S.
3. Use this number k for the BALL-PARTITION problem.

What is wrong with this transformation?

a. The transformation does not run in polynomial time.

b. This transformation is correct.

c. A YES solution to $A(S)$ does not imply a YES solution to S.

d. A YES solution to S does not imply a YES solution to $A(S)$.

Answer 2a

False. Transformation A only sums the integers in S, so it runs in polynomial time.

Answer 2a

False. Transformation A only sums the integers in S, so it runs in polynomial time.

Answer 2b

False. Overall, it is not correct. See below for the argument.

Answer 2a

False. Transformation A only sums the integers in S, so it runs in polynomial time.

Answer 2b

False. Overall, it is not correct. See below for the argument.

Answer 2c

A YES instance of $A(S)$ does **not** imply a YES instance of S (**True**).
Counterexample:

- Instance α: $S = \{1, 7\}$ with sum $1 + 7 = 8$.
- Transformed into instance β: $A(S) = 8$.
- $A(S) = 8$ balls can be BALL-PARTITIONED into $\{4, 4\}$,
- but $S = \{1, 7\}$ is a NO instance of PARTITION.

Answer 2a

False. Transformation A only sums the integers in S, so it runs in polynomial time.

Answer 2b

False. Overall, it is not correct. See below for the argument.

Answer 2c

A YES instance of $A(S)$ does **not** imply a YES instance of S (**True**).
Counterexample:

- Instance α: $S = \{1, 7\}$ with sum $1 + 7 = 8$.
- Transformed into instance β: $A(S) = 8$.
- $A(S) = 8$ balls can be BALL-PARTITIONED into $\{4, 4\}$,
- but $S = \{1, 7\}$ is a NO instance of PARTITION.

Answer 2d

A YES instance of S does **not** imply a YES instance of $A(S)$ (**False**).
If PARTITION has a YES solution (i.e., two subsets sum to half of the total sum), we can always set the number of balls in each box in BALL-PARTITION to this half-sum.

Question 3 [G]/[P3]

Show $\textsc{Partition} \leq_p \textsc{Knapsack}$ *(as in Lecture)*, using transformation:

Given a $\textsc{Partition}$ instance $\{w_1, w_2, \ldots, w_n\}$ with total sum $S = \sum_{i=1}^{n} w_i$, construct a $\textsc{Knapsack}$ instance $\{(w_1, w_1), (w_2, w_2), \ldots, (w_n, w_n)\}$ with capacity $W = \frac{S}{2}$ and threshold $V = \frac{S}{2}$.

Which statement(s) is/are True?

a. The transformation runs in polynomial time.

b. A YES answer to the $\textsc{Partition} \implies$ a YES answer to the $\textsc{Knapsack}$.

c. A YES answer to the $\textsc{Knapsack} \implies$ a YES answer to the $\textsc{Partition}$.

d. [G] Is this transformation invertible – $\textsc{Knapsack} \leq_p \textsc{Partition}$?

Answer 3a

True.

- This reduction runs in poly-time, specifically $O(n \cdot \log(w_{\max}))$,
- as it simply copies n weights to n (weight, weight-as-value) pairs.

However,

- If the maximum weight $w_{\max} = \max\{w_1, w_2, \ldots, w_n\}$ fits in standard 32/64-bit signed integers,
- then $\log(w_{\max})$ is at most 32/64, making the reduction run in $O(n)$.

Answer 3b

True. YES-instance for PARTITION $\rightarrow$ YES-instance for KNAPSACK.

Proof

- Use one subset, e.g., S_1 (or S_2) from PARTITION for KNAPSACK.
- Subset S_1 has total weight $S/2$ and total value $S/2$ (same for S_2).
- Thus, it is a YES-instance for KNAPSACK.

Answer 3c

True. YES-instance for $\textsc{Knapsack} \rightarrow$ YES-instance for $\textsc{Partition}$.

Proof

- A YES-instance for $\textsc{Knapsack}$ means there exists a subset Z with weight $\leq S/2$ and value $\geq S/2$.
- Since weight equals value in the transformed instances from α to β, the only way this can happen is if both the weight and value of Z are exactly $S/2$.
- Thus, the same subset Z (and $T \setminus Z$) can be used as a YES-instance for $\textsc{Partition}$.

Question 4

Hamiltonian-Cycle (HC) vs Travelling-Salesperson-Problem (TSP) *(as in Lecture)*

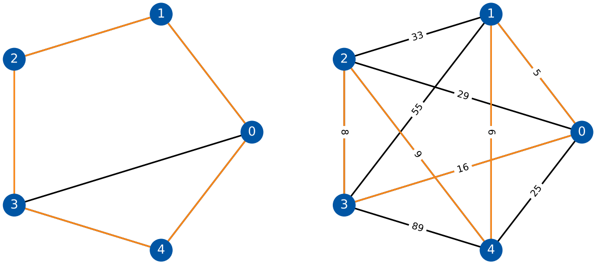

Figure 2: Illustration of Hamiltonian Cycle (left) and TSP Solution (right)

Show that $\text{HC} \leq_p \text{TSP}$!

1. **Show the transformation algorithm.**

   Let $G = (V, E)$ be an instance α of HC.
   Construct an instance β of TSP:

   - Create a complete graph G' on the same vertices V.
   - For each pair $u, v \in V$:
     - If $(u, v) \in E$, set $w(u, v) = 1$.
     - Else, set $w(u, v) = 2$ (or any value $>$1).

   Theorem: G has a Hamiltonian cycle $\iff$ G' has a TSP tour of cost at most n.

2. **Show the transformation algorithm runs in polynomial time.**

   - At most $\frac{n(n-1)}{2}$ edges are added from G to G',
   - This reduction runs in $O(n^2)$.

3 Show a YES answer to the HC $\implies$ a YES answer to the TSP.

Theorem (→): If G has a Hamiltonian cycle, then G' has a TSP tour of cost at most n.

Proof:

- Let C be a Hamiltonian cycle in G.
- Since G is a subgraph of complete graph G',
- C must exist in G'.
- C is a valid tour (each vertex appears exactly once).
- Each edge in C has cost 1 in G' (since it exists in G).
- So, the tour cost in G' is n.
- Hence, G' has a tour of cost at most n.

4 **Show a YES answer to the TSP $\implies$ a YES answer to the HC.**

Theorem (←): If G' has a TSP tour of cost at most n, then G has a Hamiltonian cycle.

Proof:

- Let C be a TSP tour of cost at most n in G'.
- Each edge in G' has cost ≥ 1.
- Since C has n edges,
- each edge must have cost exactly 1.
- Thus, each edge in C is present in G.
- As C visits each vertex exactly once, it is Hamiltonian.
- Hence, G has a Hamiltonian cycle.

Practical [Optional]

Practical repo: To help you further your understanding, not compulsory; Work for Snack!

1. Implement `partition_to_knapsack`.
2. Fill-in the missing parts for `knapsack_solver` – populate the DP table.
3. Check that you get this output:

```
Partition instance [3, 1, 1, 2, 2, 1] is solvable.
```