CSC373 Worksheet 7 Solution

August 15, 2020

1. **My Work**

   The longest simple cycle problem is the problem of finding a cycle of maximum length in a graph [5].

The decision problem is, given k, to determine whether or not the instance graph has a simple cycle of length at least k. If yes, output 1. Otherwise, output 0.

My Work

The language corresponding to the decision problem is as follows:

$$
\begin{aligned}
\text{LONGEST-SIMPLE-CYCLE} = \{\langle G, v_0, v_1, ..., v_k, k\rangle :\ & G = (V, E) \text{ is an undirected graph} \\
& k \geq 3 \text{ is an integer,} \\
& v_0, v_1, ..., v_k \in V \text{ are distinct,} \\
& v_0 = v_k, \\
& \text{There should exist a simple cycle in G} \\
& \text{with at least } k \text{ edges}\}
\end{aligned}
$$

Correct Solution:

The problem LONGEST-SIMPLE-CYCLE is a relation that associates each instacne of a graph with the longest simple cycle in that graph .

The decision problem is, given k, to determine whether or not the instance graph has a simple cycle of length at least k. If yes, output 1. Otherwise, output 0.

The language corresponding to the decision problem is as follows:

$$\begin{aligned}\text{LONGEST-SIMPLE-CYCLE} = \{\langle G, k\rangle : & G = (V, E) \text{ is an undirected graph} \\ & k \geq 0 \text{ is an integer,} \\ & \text{There should exist a simple cycle in G} \\ & \text{with at least } k \text{ edges}\}\end{aligned}$$

__Notes__

- **A Cycle in an Undirected Graph**
  - A path $\langle v_0, v_1, ..., v_k$ forms a cycle if $k \geq 3$, and $v_0 = v_k$.
- **Simple Cycle**
  - A cycle is simple if $v_1, v2, ..., v_k$ are distinct
- **Decision Problem**
  - Is the problem with yes/no solution

- **Alphabet**
  - Is a finite set of symbols
  - Is denoted Σ

    __**Example:**__

    For decision problem, its alphabet is: $\Sigma = \{0, 1\}$
    * 1 means 'yes'
    * 0 means 'no'
- **Language**
  - Is any set of strings made of symbols from Σ
  - Is denoted L

    __**Example:**__

    $L = \{10, 11, 101, 111, 1011, 1101, 10001\}$

  - Is denoted Σ^* for language of all strings over Σ plus empty string ϵ.

    __**Example:**__

    $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 11, 000, ...\}$

    __**Example 2:**__

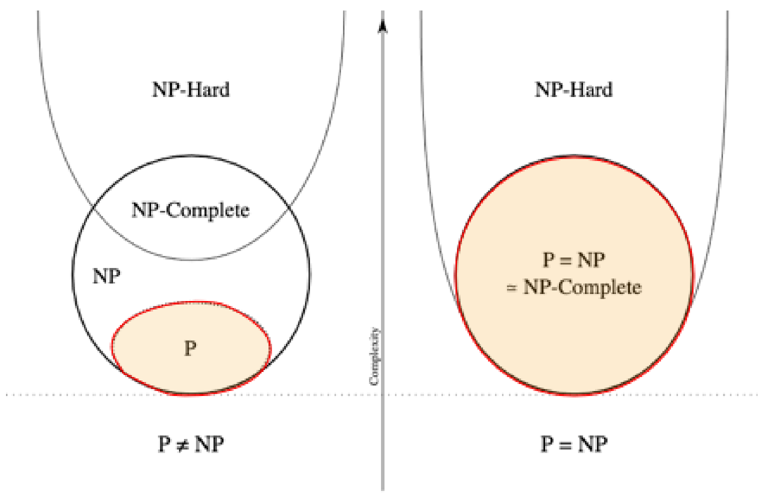
The decision problem PATH has the corresponding language

$$\begin{aligned}\text{PATH} = \{\langle G, U, v, k\rangle : & G = (V, E) \text{ is an undirected graph,} \\ & u, v \in V, \\ & k \geq 0 \text{ is an integer, and} \\ & \text{tere exists a path from } u \text{ to } v \text{ in } G \\ & \text{consisting of at most } k \text{ edges}\}\end{aligned}$$

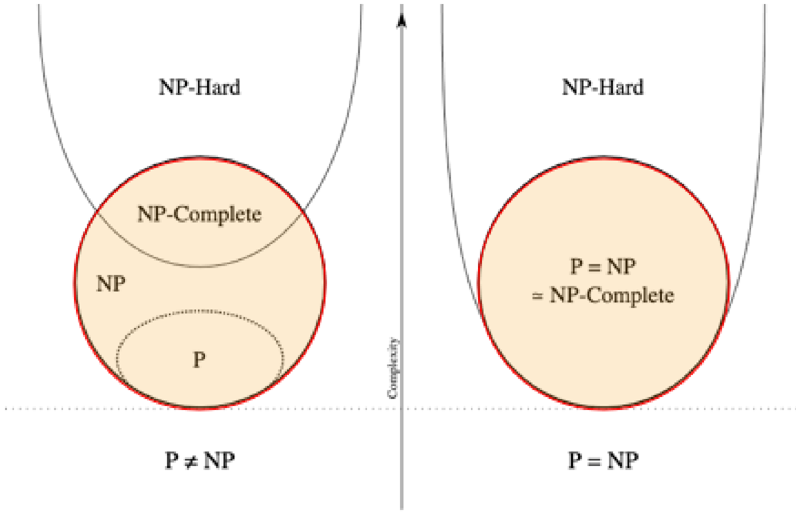
- **P**
  - Is set of problems that can be solved by a deterministic Turing machine in Polynomial time (i.e. $\mathcal{O}(n^k)$) [2].

    Example:

    1) Shortest path problems
    2) Calculating the greatest common divisor
    3) Finding maximum bipartite matching



- **NP (Non-deterministic Polynominal):**

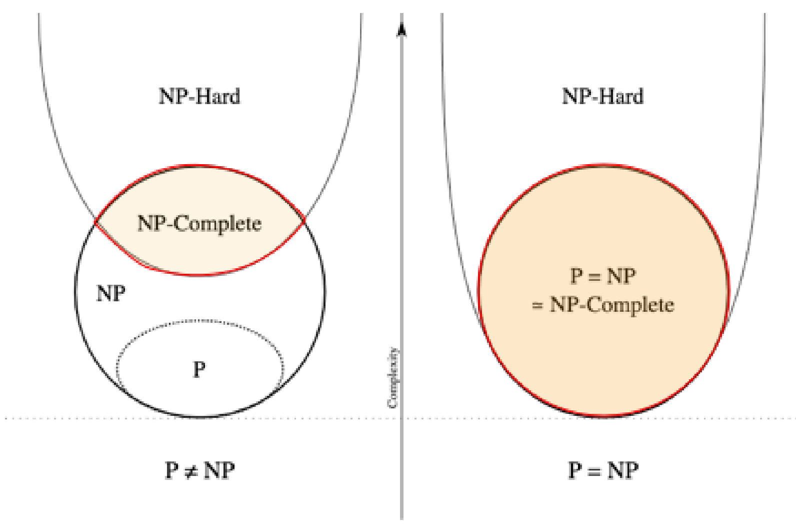


- Is set of decision problems that can be solved by a Non-deterministic Turing Machine in Polynomial time.[2]
- Has no particular rule is followed to make a guess [1].
- Can be solved in polynominal time via a "lucky algorithm", a magical algorithm that always make a right guess [2]
- $P \subseteq NP$

Examples:

- Longest-path problems
- Hamiltonian Cycle
- Graph coloring

- **NP-Complete Problems:**

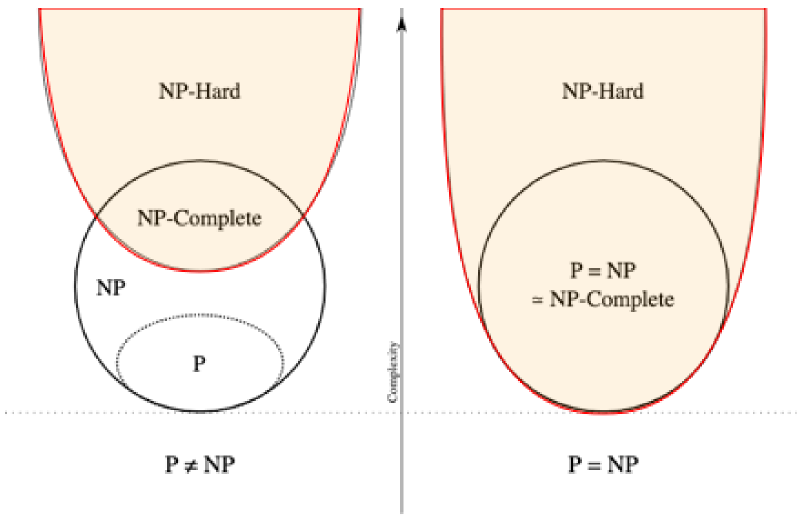


- A decision problem A is NP-complete (NPC) if

1) $A \in NP$ and
2) Every (other) problems A' in NP is reducible to A

- Has no efficient solution in polynominal number of steps (not yet) [3]
- Is not likely that there is an algorithm to make it efficient [3]

- **NP-Hard:**



- A decision problem A is NP-hard if
  1) $A \in NP$ (Not necessarily) and
  2) Every (other) problems A' in NP is reducible to A
- NP-Hard means "at least as hard as any problems in NP"
- Does not have to be about decision problems

Example:

1) Alan Turing's Halting Problem

References

1) Encyclopedia Britannica, NP-Complete Problem, link
2) Geeks for Geeks, NP-Completeness, link
3) Wikipedia, NP-complete, link
4) UCLA UC-Davis, ECS122A Handout on NP-Completeness, link

2. **Notes**

- I need help from professors on the meanings behind formalization, and how its done :(.
  - I am having a lot of difficulty answering this question.
  - Some of the questions that comes to my mind are
    1. What does it mean when asked to give a formal x of something?
    2. How can I make x formal? What are some thought processes involved in making it formal?
    3. What is the end the first two parts of the problem are looking for?
    4. What does it mean when two are polynominally related?
- **Encoding**
  - Represents problem instances in a way that the program understands
  - Encoding of a set S is a mapping e from S to the set of binary strings.

    Example

    Given natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4\}$,

    it's encoding is $\{0, 1, 10, 11, 100, ...\}$.

    Using this encoding, $e(17) = 10001$.

3. I need to show whether the algorithm for 0-1 knapsack problem in exercise 16.2-2 is a polynominal time algorithm.

   Exercise 16.2-2 states that the algorithm has running time of $\mathcal{O}(nW)$

   The definition of polynominals tells us it is of the form

   $$a_n x^n + a_{n-1}x^{n-1} + ... + a_2x^2 + a_1x + a_0 \tag{1}$$

   Since we know W represents the total weight of knapsack and is a variable in the form of an integer, we can write W is a polynominal.

   The rule of polynominals tells us product of two polynominals are a polynominal.

   Since we know n is a polynominal and W is a polynominal, we can conclude $\mathcal{O}(nW)$ is a polynominal.

   Thus, we can conclude the 0-1 knapsack problem has a polynominal running time.

   Notes

- I understand this proof is wrong.
- I feel the need to understand the language L in terms of 0-1 fractional knapsack problem
- How can I give language to this problem?
- What is the formal definition of 0-1 knapsack problem?
- How can I show that this algorithm is accepted?
- How can I show that there exists a constant k such that for any length-n string $x \in L$, algorithm A accepts x in time $\mathcal{O}(n^k)$.
- **Polynominal Time**
  - Is expressed in the format $\mathcal{O}(n^k)$.
  - A language L is **polynominal**
- **Language (Cont')**
  - A **language is accepted in polynominal time** by an algorithm A, if it's accepted by A, and if in addition there exists a constant k such that for any length-n string $x \in L$, algorithm A accepts x in time $O(n^k)$.
  - I feel this statement is saying "the algorith runs in polynominal time if the algorithm is correct and is expressed in $\mathcal{O}(n^k)$ for some constant k".

4. **<u>References</u>**

1) Universkity of Helsinki, Design and Analysis of Algorithms, Fall 2014 Exercise III: Solutions, link