CS295 Introduction to Algorithmic Game Theory **November 8, 2022**

Lecture 8. Complexity Classes and AGT.

Instructor: Ioannis Panageas *Scribed by: Rose Zhang, Yiyang Min*

This lecture will discuss on the complexity of computing a Nash equilibrium, with review of basic and function complexity classes.

Contents

1 Standard Complexity Classes - Decision problem, P and NP

Definition 1.1 (Decision Problem)**.** A decision problem is a yes-or-no question on an infinite set of inputs. [1]

Example 1.1 (Decision Problem). Is n a prime number?

Remark. Notice that the input for example 1.1 is $\log_2 n$.

Definition 1.2 (Complexity Class **P**)**.** The set of decision problems for for which some algorithm can provide an answer in polynomial time.

Remark. Example 1.1 is a problem in **P**. This is non-trivial to prove: naive n^2 algorithms run in non-polynomial time with respect to input size, which is $\Theta(2^n)$.

Remark. For algorithms, ”in polynomial time” and ”efficiently” are interchangeable.

Definition 1.3 (Complexity Class **NP**)**.** The set of decision problems for which a ”yes” answer is verifiable in polynomial time.

Definition 1.4 (Complexity Class co-**NP**)**.** The set of decision problems for which a ”no” answer is verifiable in polynomial time.

Example 1.2 (Travelling salesman problem (TSP)). Given a complete weighted graph, find the lowest-weight cycle that visits each vertex at least once.

Remark. TSP is not a decision problem, but it can be translated to one.

Example 1.3 (Decision Version of TSP). Given a complete weighted graph, does there exist a cycle that visits each vertex at least once of weight less than w?

Remark. Decision Version of TSP is in **NP**: to verify, output an example of one such cycle.

Problems in **P** must be in **NP**, but whether vice versa remains one of the hardest problem nowadays which also provides one of the hardest way to earn one million dollars. [2]

2 Function Complexity Classes - function problem, FP, FNP and TFNP

Definitions of P and NP describe decision problems well, but many problems are not decision problems and are hard to transform into decision problem. E.g. finding a Nash equilibrium with given input. Therefore, new definitions are needed.

[1] `https://en.wikipedia.org/wiki/Decision_problem`
[2] `http://www.claymath.org/millennium-problems/p-vs-np-problem`

Definition 2.1 (Function Problem (non-rigorous))**.** A function problem is a computational problem defined by a function f where the input is some x and the expected output is a y such that $f(x) = y$. [3]

Definition 2.2 (Complexity Class **FP**)**.** The set of function problems for which some algorithm can provide an output/answer in polynomial time.

Definition 2.3 (Complexity Class **FNP**)**.** The set of all function problems for which the validity of an (input, output) pair can be verified in polynomial time (by some algorithm).

Definition 2.4 (Complexity Class **TFNP**)**.** Subclass of **FNP** for which existence of solution is guaranteed for every input.

We are more interested in class **TFNP** since finding Nash equilibria is **FNP** and the existance of Nash equilibria is proven.

2.1 Non-Constructive Arguments

Four basic common non-constructive arguments, under condition that input in finite:

- **Local Search** Every directed acyclic graph must have a sink (no outgoing edge).
  Imagine the vertices to be cups placed on different hight cylinders, connected with pipe to lower cups indicated by directional edge. Pour water to any cup and it would flow to the lowest cup with no lower cup connected.

- **Pigeonhole Principle** If a function maps n elements to $n-1$ elements, then there is at least one collision.
  The only way to avoid collision is to assign the $n-1$ images one element, which consumes $n-1$ elements in the domain. But one more element un-assigned in domain, assigning which causes the collision.

- **Handshaking lemma** If a graph has a node of odd degree, then it must have another.
  Total number of outgoing edge should be even, since each edge leads to two outgoing edge on two vertices.

- **End-of-line** If a directed path has an unbalanced node, then it must have another.
  Total number of incoming edge should be the same to the outgoing edge as each directed edge contribute a pair of incoming edge and outgoing edge.

2.2 Non-Constructive Classes

Based on these arguments, there are four complexity classes in **TFNP**:

- **PLS** All problems in **TFNP** whose existence proof is implied by Local Search argument.

[3] `https://en.wikipedia.org/wiki/Function_problem`

- **PPP** All problems in **TFNP** whose existence proof is implied by the Pigeonhole Principle.
- **PPA** All problems in **TFNP** whose existence proof is implied by the Handshaking lemma.
- **PPAD** All problems in **TFNP** whose existence proof is implied by the End-of-line argument.

3 Reduction, C-hard and C-complete

Before going detail to **TFNP**, based on the problems in this script, one can observe that there are some problems are intuitively harder then the others, e.g. example 1.2, can we find them out? Moreover, can we give a similar definition similar to the definition of P and NP are given in section 1?

Definition 3.1 (Reduction)**.** A reduction is an algorithm for transforming one problem into another problem. [4]

To the previous question, the answer is "yes" by using reduction algorithm, which link two problems by transform the input and output from one to another. Moreover, two problems have a difficulty comparison relation under reducible.

Definition 3.2 (Reducible)**.** Problem A is reducible to problem B, if an algorithm for solving problem B efficiently (if it existed) could also be used as a subroutine to solve problem A efficiently. Denoted as $A \leq_p B$, where subscript p means through a reduction algorithm in polynomial time. [5]

If A is reducible, it means A cannot be "harder" than B. Based on this, the hardest problems in problem class C is defined as follows.

Definition 3.3 (C-hard)**.** For problem class C, a problem Q is C-hard if $\forall P \in C, P \leq_p Q$.

Corollary 3.1. For problem class C, a problem Q is C-hard if Q can be turned into any other C problem in polynomial time.

Corollary 3.2. For problem class C, a problem Q is C-hard if Q is at least as hard as any C problem.

Definition 3.4 (C-complete)**.** For problem class C, a problem Q is C-complete if Q is C-hard and $Q \in C$.

Corollary 3.3. For problem class C, a problem Q is C-complete if Q is the hardest problem in C.

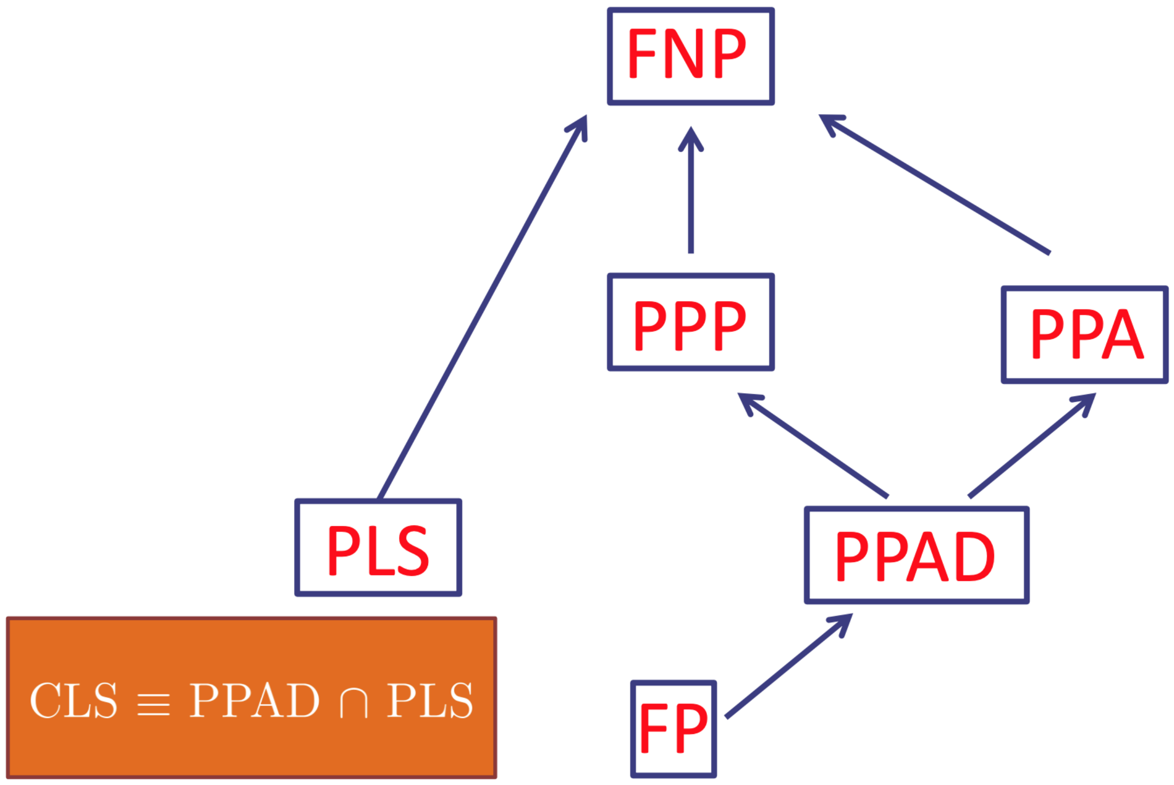
3.1 Proven C-complete problems

Showing that a problem is C-hard/complete is a difficult proof likely involving Turing Machines. Once a problem has been proved to be C-hard/complete, one can easily prove it for another problem using reductions. Luckily, there are many categories C for which problems have been proven to be C-hard/complete.

[4] `https://en.wikipedia.org/wiki/Reduction_(complexity)`
[5] `https://en.wikipedia.org/wiki/Reduction_(complexity)`

- SAT is an **NP**-complete problem.
- Nash equilibrium is **PPAD**-complete.
- Pure Nash equilibrium in congestion games is **PLS**-complete.

3.2 Relation Between Function Complexity Classes



FNP
PPP
PPA
PLS
PPAD
FP
CLS ≡ PPAD ∩ PLS