L08 Complexity Classes and AGT

CS 295 Introduction to Algorithmic Game Theory

Ioannis Panageas

Standard Complexity Classes

- P: Set of decision problems for which some algorithm can provide an answer in polynomial time.

Standard Complexity Classes

- P: Set of decision problems for which some algorithm can provide an answer in polynomial time.

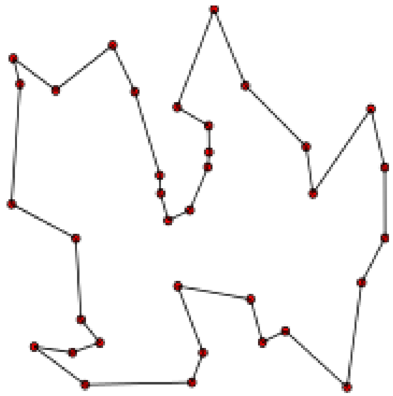
- NP: Set of all decision problems for which for the instances where the answer is "yes“, we can verify in polynomial time that the answer is indeed yes.
- co-NP: Same as above with yes->no.

Problems in **P**

- Problems that can be solved in **polynomial time**.
  - Decision version of shortest path (is the shortest path at most L (yes/no)?)
  - Decision version of finding the maximum number of a list (is the maximum at most M (yes/no)?)
  - Is n a prime number (yes/no)?

Problems in **NP**

- Problems that "yes" instance can be verified in **polynomial time**.
- e.g., The travelling salesman problem (TSP)
  - Given a complete weighted graph, find the shortest route that visits each vertex once and returns to origin (decision version).



Equilibrium Computation

- The goal is to find a function that maps games to (mixed strategy) Nash equilibria.
- This is NOT a decision problem ~~(yes/no)~~.

Equilibrium Computation

- The goal is to find a function that maps games to (mixed strategy) Nash equilibria.
- This is NOT a decision problem ~~(yes/no)~~.
- Examples:
  - Add two numbers and find the outcome
  - Is the sum of two numbers odd?

Function Complexity Classes

- FP: The set of function problems for which some algorithm can provide an output/answer in polynomial time.

- FNP: set of all function problems for which the validity of an (input, output) pair can be verified in polynomial time (by some algorithm).

Function Complexity Classes

- FP: The set of function problems for which some algorithm can provide an output/answer in polynomial time.

- FNP: set of all function problems for which the validity of an (input, output) pair can be verified in polynomial time (by some algorithm).
- TFNP: Subclass of FNP for which existence of solution is guaranteed for every input!

Non-constructive arguments

- Local Search: Every directed acyclic graph must have a sink.

Non-constructive arguments

- Local Search: Every directed acyclic graph must have a sink.
- Pigeonhole Principle: If a function maps n elements to n-1 elements, then there is a collision.

Non-constructive arguments

- Local Search: Every directed acyclic graph must have a sink.
- Pigeonhole Principle: If a function maps *n* elements to *n*-1 elements, then there is a collision.
- Handshaking lemma: If a graph has a node of **odd** degree, then it must have another.

Non-constructive arguments

- Local Search: Every directed acyclic graph must have a sink.
- Pigeonhole Principle: If a function maps n elements to n-1 elements, then there is a collision.
- Handshaking lemma: If a graph has a node of **odd** degree, then it must have another.
- End of line: If a **directed** path has an unbalanced node, then it must have another.

From non-constructive arguments to complexity classes in TFNP

- PLS: All problems in TFNP whose existence proof is implied by Local Search arg.
- 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.

Proving a negative result

Question: How to prove there is no polynomial time algorithm for a problem?

– i.e., show that there is no algorithm of time $O(n), O(n^2), O(n^3), \dots \text{ etc?}$

Answer: We don’t know how to do it. Instead, we do reductions!

The hardest problems in class C

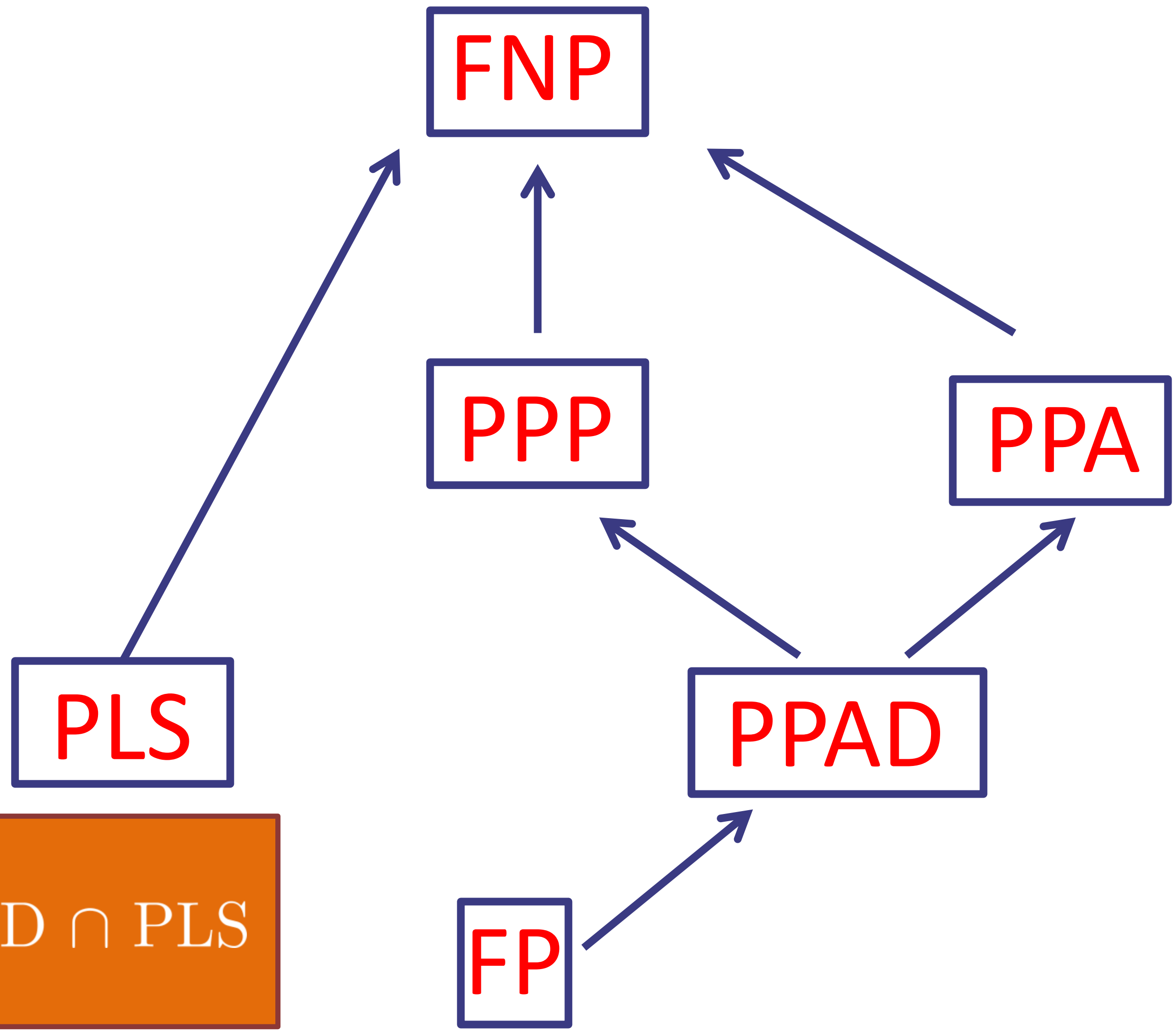
- A problem $\boldsymbol{Q}$ is **C-hard** if
  - all problems in C can be reduced to it: for all $\boldsymbol{P}$ in C, $\boldsymbol{P} \leq_P \boldsymbol{Q}$
  - $\boldsymbol{Q}$ can be turned into any other C problem, in poly time
  - $\boldsymbol{Q}$ is at least as hard as any C problem
- A problem $\boldsymbol{Q}$ is **C-complete** if it is in class C and C-hard
  - $\boldsymbol{Q}$ Is the hardest problem in C.
  - $\boldsymbol{Q}$ is in C, and for all $\boldsymbol{P}$ in C, $\boldsymbol{P} \leq_P \boldsymbol{Q}$

The hardest problems in class C

- A problem ***Q*** is **C-hard** if
  - all problems in C can be reduced to it: for all ***P*** in *C*, ***P*** $\leq_P$ ***Q***
  - ***Q*** can be turned into any other C problem, in poly time
  - ***Q*** is at least as hard as any C problem
- A problem ***Q*** is **C-complete** if it is in class C and C-hard
  - ***Q*** Is the hardest problem in C.
  - ***Q*** is in C, and for all ***P*** in *C*, ***P*** $\leq_P$ ***Q***

- SAT is an NP-complete problem! (Cook, 71').
- Find a Nash eq. is PPAD-complete! (Daskalakis et al 06')
- Find a pure Nash eq. in congestion games is PLS-complete! (Fabrikant et al 04).

AGT and complexity classes



$$\mathrm{CLS} \equiv \mathrm{PPAD} \cap \mathrm{PLS}$$