CS 295 — Introduction to Algorithmic Game Theory October 20, 2021

Lecture 8 and 9

Lecturer: Ioannis Panageas *Scribe(s): (Freddy Reiber and Ofek Gila)*

In this lecture we will look at the complexity of general games, more specifically the computational difficulty of finding Nash equilibria. Before that, a quick review on basic/common complexity classes.

Contents

1 Standard Complexity Classes **1**
- 1.1 Problems in P . . . 2
- 1.2 Problems in NP . . . 2
- 1.3 C-hard and C-complete . . . 2
- 1.4 Equilibria Computation . . . 2

2 Reductions in NP **3**
- 2.1 3-SAT . . . 3
- 2.2 Vertex Cover . . . 3

3 Function Complexity Classes **4**
- 3.1 F, FNP, $TFNP$ . . . 5
- 3.2 Non-Constructive Arguments and Classes . . . 5

4 The Complexity of Pure Nash Equilibria **6**
- 4.1 The Class PLS . . . 6
- 4.2 Finding Pure Nash Eq. in Congestion Games $\in$ PLS . . . 6
- 4.3 Finding Pure Nash Eq. in Congestion Games is PLS-Hard . . . 7

5 The Complexity of Finding Nash Equilibria **8**
- 5.1 2D Sperner's Lemma . . . 8
- 5.2 Brouwer Fixed-Points . . . 10
- 5.3 Nash reduction to Brouwer . . . 10
- 5.4 The Class PPAD . . . 10

1 Standard Complexity Classes

Definition 1 Class P is a set of decision problems in which you can compute an answer in polynomial time. $\diamondsuit$

Class P is one of the most common and fundamental complexity classes that is studied. Generally, we consider problems that are $\in P$ to be "easy". In theoretical Computer Science we consider finding a problem that is $\in P$ to be a positive result, as for any given input we can calculate an answer efficiently.

1.1 Problems in P

An example of this would be determining if a sequence is non-decreasing. This is a decision problem as we return TRUE or FALSE. To decide this problem, simply iterate through the sequence making sure that no element is smaller than the last, i.e., $n_{i+1} \geq n_i$. If such an element exists, return FALSE otherwise return TRUE.

1.2 Problems in NP

Definition 2 Class NP is a set of decision problems in which you can verify an answer in polynomial time. $\diamondsuit$

A second commonly discussed complexity class, is the class NP. For a problem to be in NP you must be able to verify the answer in polynomial time. However, we do not know if problems $\in NP$ have polynomial time solvers, although we suspect they do not ($P = NP$ problem). For a more formal definition, and introductory text, look at Sipser - Introduction to the Theory of Computation. An example of this would be the Hamiltonian Path, as one can verify it by simply iterating through the path.

1.3 C-hard and C-complete

Before discussing the relevant AGT problems, there are two more important definitions.

Definition 3 A problem Q is C-hard where C is some complexity class, if all problems in C can be reduced to Q. $\diamondsuit$

The main idea in this definition is that of a reduction. Simply put, a reduction takes an instance of one problem and turns it into an instance of another problem, in polynomial time. This technique allows us to say that the second problem is as hard as the first problem, a solution the first problem can be used to solve the second problem. Therefore if Q can be reduced to all problems in C it must be at least as hard as the hardest problems in C. For an example of a reduction, see section 2.

Definition 4 A problem Q is C-complete if it is C-hard and $\in C$ $\diamondsuit$

Simply put, this means that Q is the hardest problem in C (note that there can be many hardest problems). This is because we know that Q is at least as hard as problems in C, from our earlier definition, but it could still be harder. However, if we show that the problem is $\in C$, then we know it can't be any harder, as it is also in the same class. That would prove that it is C-complete.

1.4 Equilibria Computation

The main problem we care about in AGT, that of calculating equilibria. Moreover, our goal is to find some algorithmic procedure that is able to take any given game and calculate it. Notice that this problem is not a decision problem, i.e. what we return is not a Boolean. Thus our problem does not fall into any of these classes. Therefore we must define other complexity classes that properly capture the complexity of this problem. We will show this in section 3.

2 Reductions in NP

It is important to know which problems can be solved efficiently in AGT. In general, efficient algorithms are those that can be solved in polynomial time. Conversely, it is important to identify which algorithms *cannot* be solved efficiently, i.e., are NP-hard. It is often preferable to prove the stronger statement, however, that problems are NP-complete. The easiest way to show that a problem is NP-complete is to first show that it belongs to NP by showing how it can be verified in polynomial time, and then show that it is NP-hard by reducing a known NP-complete problem to it.

2.1 3-SAT

Definition 5 The 3-SAT problem is the problem of determining whether a Boolean expression E, defined by a conjunction of clauses where each clause is a disjunction of exactly 3 literals, is satisfiable. ♢

For example, consider the following Boolean expression:

$$E = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \tag{1}$$

One possible assignment of variables is: $x_1 = 1, x_2 = 1, x_3 = 0$, so this expression is satisfiable. On the other hand, consider:

$$E = (x_1 \vee x_2 \vee x_2) \wedge (\bar{x}_1 \vee x_2 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_2) \tag{2}$$

This equation cannot be satisfied by any assignment of Booleans, therefore it is *not* satisfiable. The 3-SAT problem is known to be NP-complete.[1]

2.2 Vertex Cover

Definition 6 The vertex cover problem is the problem of determining whether k vertices of a graph G can be selected such that every edge in G has at least one endpoint in one of the k vertices. ♢

First, we must show that this problem is in NP. This is trivial, since given a set V of k vertices, one can simply iterate over all the edges and verify that one of their endpoints is in V. This can be done naively in polynomial time. Next, we show that an NP-complete problem, we'll be using 3-SAT, can be reduced to the vertex cover problem. This will show that it is at least NP-hard, and in this case NP-complete.

The second part is a bit trickier. Given a specific instance of a vertex cover problem with Boolean expression E consisting of v variables and c clauses we construct a graph G as follows:

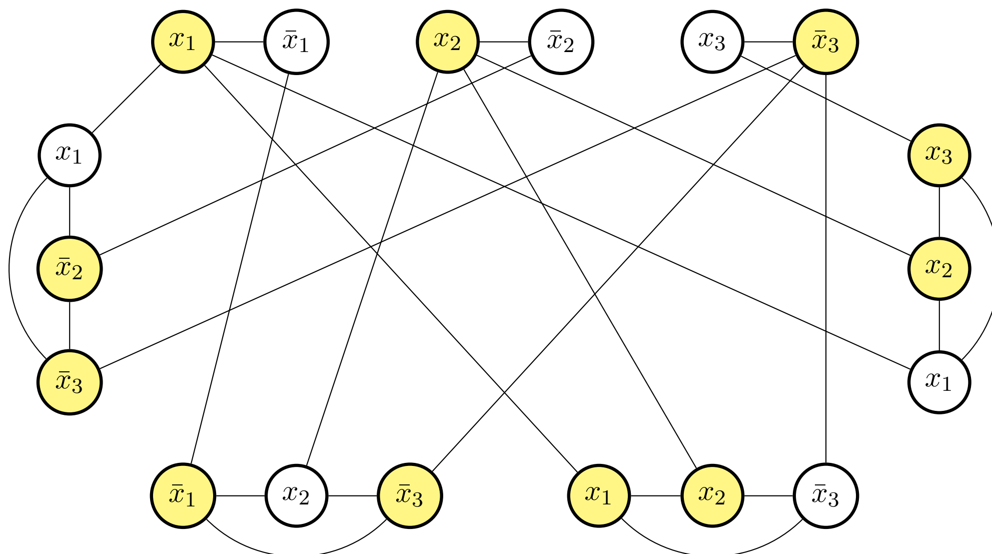
1. Create a pair of vertices $(x_i, \bar{x}_i)$ for every variable.

[1] One way of showing that the 3-SAT problem is NP-complete is by reducing the general SAT problem to it. The SAT problem can be shown to be NP-complete using the Cook—Levin theorem.

2. For every clause $(\ell_1, \ell_2, \ell_3) \in E$, create a 'gadget', i.e. a clique consisting of ℓ_1, ℓ_2, and ℓ_3.

3. Add an edge between every literal in every gadget to the same literal created in step 1.

This transformation can be done in polynomial time. Iff a vertex cover of size $v + 2c$ exists, the 3-SAT problem is satisfiable. To prove this, consider how at least one vertex in every pair of vertices created in step 1 must be selected in a valid vertex cover, which is already at least v vertices. This selection corresponds to which literals are true. Next, consider how at least two vertices per gadget must be selected since they form a clique. This would result in a vertex cover with $v + 2c$ vertices. If a gadget, however, does not contain any literals that are true, then all three vertices must be selected, to account for the edges between each vertex and the vertices in step 1. A single false gadget will result in a greater minimum vertex cover than the above, and therefore no vertex cover with $v + 2c$ vertices would be possible. See the below figure for an example of eq. (1):



The vertices created in step 1 are the top row, while the gadgets representing the clauses are shown in the left-hand column, the bottom row, and the right-hand column. Since a vertex covering (shown above) with $v + 2c$ vertices exists, the original Boolean expression E is satisfiable, showing that a known NP-complete problem (3-SAT) can be reduced to vertex cover, meaning that vertex cover is NP-hard. Combined with the original finding that vertex cover is in NP, we have shown vertex cover to be NP-complete.

3 Function Complexity Classes

In this section we will look at functional complexity classes.

3.1 F, FNP, $TFNP$

First we look at three complexity classes that are a natural extension of our original commonly used complexity classes.

Definition 7 FP: The set of function problems for which some algorithm can provide an output/answer in polynomial time. $\diamondsuit$

Definition 8 FNP: The set of all function problems for which the validity of an (input, output) pair can be verified in polynomial time. $\diamondsuit$

Based on intuition, calculating the Nash of games is likely difficult, as there seems to be no way to infer something about the solution based on looking at other possible solutions. However, we do know from Nash's theorem that the solution must exist, thus it can't be NP-Hard as it those problems may not have a solution. Another natural extension of FNP would be the following:

Definition 9 $TFNP$: A subclass of FNP for which existence of a solution is guaranteed for every input! $\diamondsuit$

3.2 Non-Constructive Arguments and Classes

Within $TFNP$ we also have number of subclass complexity classes. For all the ones discussed here, they are based off of non-constructive arguments. This is a type of mathematical argument that shows that something (in this case a solution) must exist, without actually showing how to find/construct it. We then can use those arguments to further divide our complexity class, and more accurately find the complexity of a given problem. Here are 4 common non-constructive arguments:

- **Local Search**: Every directed acyclic graph must have a sink (a node with no outgoing edges).
- **Pigeonhole Principle**: If a function maps n elements to $n - 1$ elements then their 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 these arguments we can define new complexity classes that are defined by these non-constructive arguments, and further define the complexity of problems in $TFNP$. Note that all of these classes are subclasses of $TFNP$:

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

4 The Complexity of Pure Nash Equilibria

In this section we fully identify the complexity of finding pure Nash equilibria in congestion games. Specifically, we show that computing pure Nash eq. is PLS-complete.

4.1 The Class PLS

In the previous section, we looked some general definitions. We will know look at the full definition of one specific class, **PLS**. We will also give examples of these with the local max-cut problem.

Definition 10 PLS (Polynomial-time Local Search) is a complexity class intended to exemplify local search problems. For a problem given problem, we can define it by three polynomial-time algorithms: $\diamondsuit$

1. The algorithm takes as input an instance and outputs an arbitrary feasible solution.
2. The algorithm takes as input an instance and a feasible solution, and returns the objective function value of the solution.
3. The algorithm takes as input an instance and a feasible solution and either reports that the solution is "locally optimal" or produces a better solution.

We will now give an example of this to help build intuition. One common PLS-complete. problem is the local max-cut problem. The goal of the local max-cut problem is to divide the graph into two sets, such that if you were to cut along all edges between the two groups the cut weight would be a local maximum. More specifically, you cannot increase the cut weight by switching which set a single vertex is in. We can see this can be solved using our three algorithms. For the first algorithm, we produce an arbitrary cut/partition. For the second, we can sum the total weight of all edges along the cut. For the third we check all possible $|V|$ moves. If one improves the objective we choose that move. Thus we an say that local max-cut is $\in$ PLS. It also turns out that local max-cut is PLS-hard, thus we can say:

Theorem 1 *The Local Max-Cut problem is PLS-Complete.*

4.2 Finding Pure Nash Eq. in Congestion Games $\in$ PLS

With our class rigorously defined we can know look at the complexity of computing pure Nash equilibria in congestion games.

Theorem 2 *The problem of computing pure Nash equilibria in congestion games is PLS-complete.*

The first step in showing that this theorem is true is to show that PNE of congestion games $\in$ PLS. We will do show using the same three algorithm procedure as we did for local max-cut.

1. Take as input a congestion game and return an arbitrary strategy profile.
2. Take as input a congestion game and strategy profile s, and return the value of the potential function $\Phi(s) = \sum_e \sum_{j=1}^{l_e(s)} c_e(j)$.
3. Check if the given strategy profile s is a PNE. If it is not, we find an agent i that can change their strategy from s_i to s_i' such that $\Phi(s_i', s_{-1} < \Phi(s_i, s_{-i})$.

As all of these can be done in polynomial time in respect to the representation of the game, we can say the PNE congestion games $\in$ PLS.

4.3 Finding Pure Nash Eq. in Congestion Games is PLS-Hard

We will now show that finding PNE in Congestion Games is PLS-Hard. To do so we use a reduction from Local Max-Cut. The reduction works as follows:

Given a weighted graph (G, E) we define the following congestion game,

- Agents are the vertices V.
- For each edge $e \in E$ we have the resources $r_e, \overline{r_e}$.
- Each player v has then two strategies, $s_v = \{r_e : e \text{ is incident to v}\}$ and $\overline{s_v} = \{\overline{r_e} : e \text{ is incident to v}\}$.
- The cost of using a resource r_e or $\overline{r_e}$ is 0 if one agent uses it and w_e if two players use it.

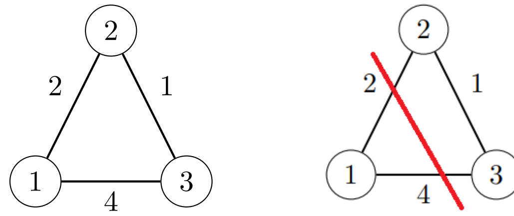
Notice that nothing in this reduction is computationally hard, and can all be done in polynomial time. We will now show that this reduction gives a bijection between strategy profiles of this congestion game and possible cuts of the graph G.

Notice that for a given cut $(S, \overline{S})$ where agents in S chose black(s_v) and agents in $\overline{S}$ choose purple($\overline{s_v}$), we have:

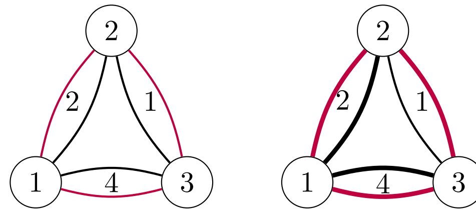
$$w(S, \overline{S}) = \sum_{e=(u,v):u \in S, v \in \overline{S}} w_e = \sum_{e \in E} w_e - \Phi(s, \overline{s})$$

Moreover, we can say that cuts with larger weight have a strategy profile with smaller potential. Therefore, local maximal of cuts of G correspond to the local minima of the potential function. Thus an optimal solution of local max-cut is a PNE of a congestion game. An example of the reduction can be seen below:

Let's start with the graph G below as our graph with its max cut shown,



From this, if we apply the reduction we get the following congestion game, with the cost if both edges are chosen in between. In the image next to it, we have the optimal solution, with the bolded edges representing which strategies are played.



Looking at this game, we can see that the best strategy is for players 2 and 3 to take the purple paths, and for player 1 to use the black paths. This means that only the edge between 2 and 3 has its weight counted, which is a PNE. Notice that this is the same result we would get if we based our answer off the reduction. With the cut making the following two sets $1, 2, 3$ we have player 1 play black, and players 2, 3 play purple.

5 The Complexity of Finding Nash Equilibria

In the previous section we identified that the complexity of finding pure Nash equilibria in congestion games is PLS-complete. Similarly, in this section we will show that the complexity of of finding Nash equilibria in general is PPAD-complete.

5.1 2D Sperner's Lemma

We start our discussion with a classic example of a PPAD-complete problem, Sperner's lemma:

Theorem 3 *Consider a triangulation of a 2d simplex Δ and a proper 3-coloring where each vertex is assigned a different color, and where vertices on each edge of Δ only use the two colors from the edge's respective endpoints. There always exists an odd number of trichromatic triangles.*

A random simplex with side length 10 was generated following the above conditions, and the colors red, green, and blue. Then, it was augmented by a new edge on the red-yellow side, which contains a red vertex wherever possible. This new simplex cannot contain any new trichromatic triangles, since no blue vertices are added, and no added vertex can be adjacent to a blue vertex. Now we remove every red-yellow edge from the simplex. Figure 1 illustrates the aforementioned steps.

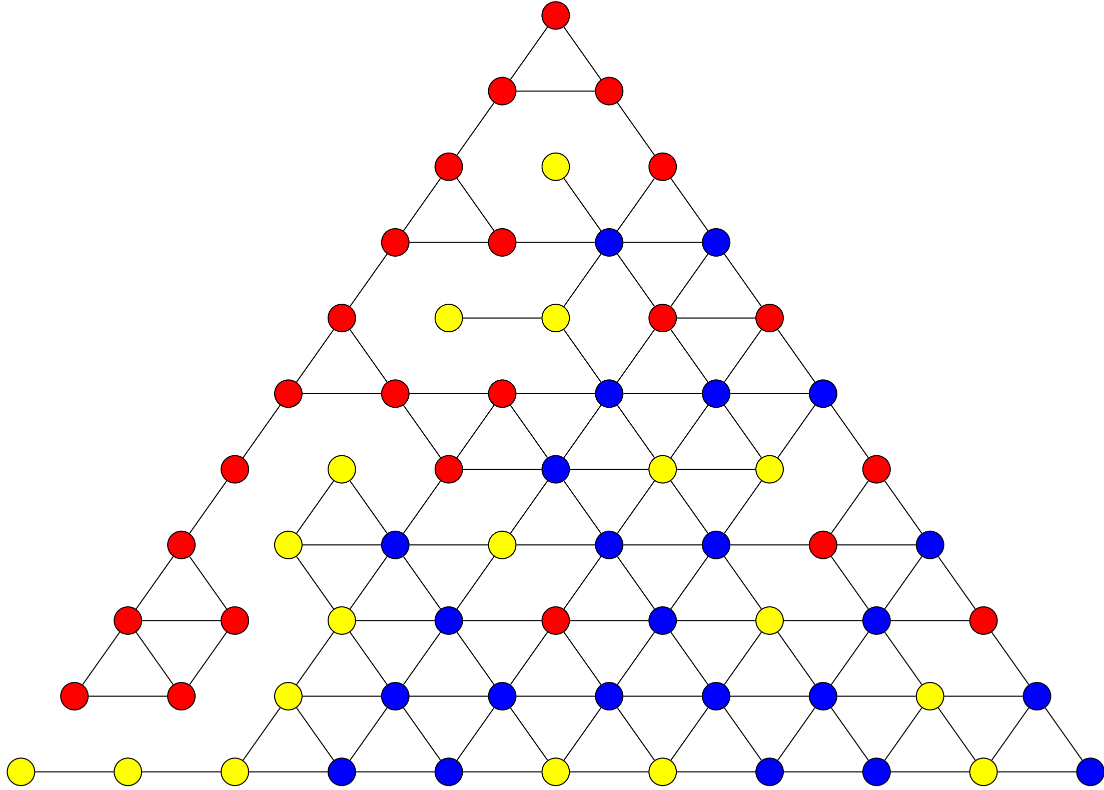


Figure 1: A simplex following theorem 3, augmented by a red-dominated leftern edge, with all red-yellow edges removed

Observe how this creates a series of paths through the simplex. Further, observe how any dead end either ends in a trichromatic triangle, or in the bottom-left corner. The proof for this is rather straightforward:

Proof Let's assume that we are at a dead end of a path which is not on an edge. There was a path that lead here, meaning that one vertex of the triangle must be red, and another vertex must be yellow. If the third vertex is either red or yellow, then there exists a second red-yellow edge, meaning that there is also a way out of this triangle, contradicting the assumption that this is a dead end. Therefore, the third vertex must be blue, and this triangle is trichromatic. Furthermore, the only red-yellow edge possible on the outside of the simplex is in the bottom-left corner, due to the nature of the augmented red-dominated leftern edge. ■

Trivially, each trichromatic triangle has exactly one red-yellow edge, meaning that they appear in exactly one path each. Each path has two ends, meaning that there are an even number of dead ends. From before, we know that one of these dead ends are in the bottom-left corner, while all the others are trichromatic triangles. Therefore, there must be an odd number of trichromatic triangles in this simplex. Since the original augmentation did not change the number of trichromatic triangles, there were an odd number of triangles in the

original simplex.

Sperner's lemma can be applied to higher dimensional simplexes using similar reasoning.

5.2 Brouwer Fixed-Points

The problem of computing fixed points in Brouwer functions is defined as:

Definition 11 Brouwer Fixed-Points: Given a polynomial-time algorithm Π_F for the evaluation of a K-Lipschitz function $F : [0,1]^m \to [0,1]^m$ and accuracy ε, output a rational point x such that $||F(x) - x||_\infty \leq \varepsilon$. $\diamondsuit$

In the 2D case, i.e., where $m = 2$, we can embed a Sperner's grid over the domain of the function, where the function changes by at most $\frac{\varepsilon}{2}$ between each two vertices of the grid (due to the K-Lipschitz nature of the function). We can then color the vertices on the grid either red, yellow, or blue, depending on the direction of $f(x) - x$, respecting boundary conditions in case of ties at the boundaries. And finally, we can use Sperner's lemma to prove that at least one trichromatic triangle exists here, where any one of its vertices is within ε of being a fixed-point. Therefore, the 2D Brouwer fixed-point problem has been reduced to Sperner's lemma and is actually in the same complexity class.

5.3 Nash reduction to Brouwer

Just like we reduced the problem of Brouwer's fixed points to Sperner's lemma, we can reduce Nash to Brouwer's fixed points. We do this by using the function f we created from the proof of the existence of Nash equilibria:

$$f_{is_i}(x) \equiv \frac{x_i(s_i) + \max\{u_i(s_i; x_{-i}) - u_i(x), 0\}}{1 + \sum_{s' \in S_i} \max\{u_i(s_i; x_{-i}) - u_i(x), 0\}} \tag{3}$$

We already know that the fixed points of this function are Nash equilibria. A vector field of $f(x) - x$ can be created, and Brouwer's fixed-point can be applied from the previous section to find Nash equilibria.

5.4 The Class PPAD

Recall how in section 3.2 we looked at general definitions of several complexity classes. This class can be formally defined using the End-Of-The-Line problem:

Definition 12 PPAD: Given a directed graph G where every vertex has at most one incoming and one outgoing edge, and given a source vertex (one with no incoming edges) s, find another vertex t which is either a source or a sink (one with no outgoing edges). $\diamondsuit$

A vertex t must exist due to the nature of graph G, but in general it may take exponential time to find it. It has been shown in 2006 that finding two-player Nash equilbiria is PPAD-complete[2], making both finding a trichromatic triangle in Sperner's lemma and computing fixed points of Brouwer functions PPAD-complete.

[2]Chen and Deng, Settling the Complexity of Two-Player Nash Equilibrium