Algorithmic Game Theory – Group 4 Report

1 Introduction

Nash equilibria (NE) come in two forms, pure Nash equilibria (PNE) and mixed Nash equilibria (MNE). In any NE, no agent wishes to unilaterally deviate from their chosen strategy, since doing so would not increase their expected payoff. While not every game possesses a PNE, every strategic game with finite players and actions possesses at least one MNE [4], where each player chooses some probability distribution over their set of actions. In this case, players' payoffs become expected values. As such, a strategy profile $p = (p_i)_{i \in N}$, where N is the set of all players, is a combination of strategies where the expected payoff of player i is $u_i(p) = \sum_{s_1 \in S_1} \cdots \sum_{s_n \in S_n} \prod_{j=1}^{n} p_j(s_j) u_i(s_1, \ldots, s_n)$. We may view PNE as a special case of MNE where each player assigns a probability of 1 to a single action.

An approximate Nash equilibrium (ANE) is some profile of mixed strategies where no player can improve their expected payoff by more than some $\varepsilon > 0$ by switching to another strategy. Given a strategic game, we define NASH as the problem of computing an ANE for a given ε, which should clearly be at least as tractable as obtaining an exact NE. NASH is *total* [3], that is, a search problem which has a guaranteed solution. As not all **NP** problems are total, it is unsuitable to place NASH in **NP**. We may however construct **NP**-complete variations of NASH by imposing restraints which remove the guarantee of a solution [1].

A congestion game is defined by some set of players N, a set of resources M, and, for each player $i \in N$, a set of pure strategies $\sum_i$ where each $A_i \in \sum_i$ is a non-empty subset of M. For each resource $j \in M$, we define a non-decreasing delay function $d_j : \{1, \ldots, |N|\} \to \mathbb{R}_+$. A player i's payoff is the negative of the cost $\lambda_i(A) = \sum_{j \in A_i} (d_j(\sigma_j(A)))$ where $\sigma_j(A)$ is the number of players using resource j. In a *network* congestion game, the resources are edges in a graph; each player has a start vertex and end vertex, and their actions are paths between these vertices. It was proven by [6] that congestion games must always have a PNE as a result of the existence of a potential function (more on this later).

2 Overview of Results

We demonstrate that in general, NASH is **PPAD**-complete. We show NASH to be in **PPAD** by reducing it to the problem BROUWER and from BROUWER to the known **PPAD**-complete problem END OF THE LINE (EOTL) and we show completeness via the reverse reduction.

We also show that, for symmetric congestion games, a PNE can be computed in polynomial time via a reduction to MIN-COST FLOW, and that the generic, symmetric, and asymmetric network cases for congestion games are **PLS**-complete. For general congestion games, we show this via a reduction from POSNAE3FLIP. For symmetric congestion games, we prove completeness via a reduction from generic congestion games. Finally, the completeness of asymmetric network games is shown through a reduction from WITNESSED EXTENDED POSNAE3FLIP (WXPNAE3FLIP) [4].

3 Derivation of Results

3.1 The Complexity Class PPAD

The class Polynomial Parity Argument for Directed Graphs (**PPAD**) consists of total search problems where a solution is guaranteed by the theorem "if a directed graph contains an unbalanced vertex (in-degree $\neq$ out-degree), then it must contain another". Equivalently, **PPAD** is the class of problems reducible to the problem EOTL, which, given a directed graph G of 2^n vertices and a specified unbalanced vertex v, is the problem of finding some other unbalanced vertex $u \neq v$. We assume that each vertex has at most one incoming and one outgoing edge and that we have access to two boolean circuits P and S both of size polynomial in n, where for any two distinct vertices u, v, $P(v) = u$ and $S(u) = v$ if there is a directed edge from u to v.

3.2 Proof that NASH is in PPAD

An instance of the problem BROUWER consists of an efficient algorithm Π_F for the evaluation of a function $F : [0,1]^m \to [0,1]^m$, satisfying the Lipschitz condition $\forall x_1, x_2 \in [0,1]^m : d(F(x_1), F(x_2)) \leq K \cdot d(x_1, x_2)$ for

some constant K, where $d(x, x')$ is the Euclidean distance between points x and x'. A solution is guaranteed by Brouwer's fixed point theorem [3] and is a point x such that $d(F(x), x) \leq \varepsilon$, for a desired accuracy ε.

Reduction from BROUWER to EOTL. We give the reduction in the 2D case, however this proof can be generalised to any dimension. Given an instance of BROUWER, we subdivide the unit square into smaller squares of size dependent on ε and K, then divide each of these squares into two right triangles, assigning one of three colours to each vertex x depending on the direction F maps it to. We assign a colouring that satisfies the following: no vertex on the bottom of the square is red, no vertex on the left is blue and no vertex on the other two sides is yellow. Sperner's Lemma [3] shows that in any colouring satisfying this property, at least one trichromatic triangle exists. Since a triangle's vertices have unique colours if and only if F displaces them in conflicting directions, then any vertex x of a trichromatic triangle is also an approximate fixed point.

We construct a directed graph G by considering each triangle T with at least one red and yellow vertex to be a node in G, with a directed edge between triangles T and T' if they share an edge that goes from red to yellow clockwise in T. We may assume that the left side of the unit square contains only one change from yellow to red (if it doesn't, we introduce a new column of red vertices to the left of the square and colour the bottom one yellow; this still satisfies Sperner's Lemma), and denote the triangle containing these transitioning vertices, T_{source}, as the source of a path in G. The path from T_{source} must end at some sink T_{sink}, since by our construction, no path can intersect itself, and no path can leave the unit square. Additionally, a triangle can only be a sink if it is trichromatic, meaning any vertex of T_{sink} will yield an approximate fixed point. It is clear to see that the discovery of T_{sink} yields our solution to EOTL, completing the reduction.

Reduction from NASH to BROUWER. Given an instance of NASH, we may define a preference function P that maps from a strategy profile s to a strategy profile s' such that $u_i(s) + \varepsilon \leq u_i(s')$ for any unsatisfied player i. We reason that a fixed point of P is any strategy profile mapped to itself, meaning no player wishes to deviate from their current strategy, that is, an ε-approximate MNE. [3] shows that such a function P is efficiently computable, satisfies the Lipschitz condition given in BROUWER for some K, and that the approximate fixed points of P do in fact correspond to ANE. Therefore, there exists an efficient reduction from NASH to BROUWER, completing our proof that NASH is in **PPAD**.

3.3 Proof that NASH is PPAD-complete

We demonstrate that NASH is **PPAD**-complete by outlining a reduction from EOTL. We do this in two steps: converting the graph into a Brouwer function; then converting the Brouwer function to a game and encoding this game in terms of another game with three players.

To encode a graph G as a continuous Brouwer function F, we use the three-dimensional unit cube as the domain of F. This creates a fine rectilinear mesh in the cube, where a point corresponds to the center of each cubelet. Away from the center of these cubelets, F is interpolated using the closest grid points. We then colour each point x one of four colours $\{0, 1, 2, 3\}$ such that $F(x) - x$ is approximately zero only in the vicinity of all four colours. Each vertex in G corresponds to two special sites in the cube. F will colour most points 0, but at these sites a sequence of the other colours appears. If G has an edge from u to v then F will colour a sequence of points with colours 1, 2, and 3 between the corresponding sites in the cube to connect them, where approximate fixed points only appear at sites that correspond to an "end of the line" of G. As we have shown BROUWER is in **PPAD**, this demonstrates that BROUWER is **PPAD**-*complete*.

From BROUWER to NASH. F can be efficiently computed using arithmetic circuits represented as "data flow graphs", or "gadgets". We introduce players for every node in the gadget to create a game with an ANE corresponding to a fixed point of F. In a simulating game, each player has a circuit-computed value and actions "stop" and "go". By choosing different payoffs, we can implement addition, multiplication, and comparison.

Computing Brouwer Function with Games. To compute a Brouwer function with games we include three players x_1, x_2, and x_3 whose "go" probabilities represent a point x in the cube. Using additional players to compute $F(x)$ with gadgets, we can end up with players y_1, y_2, and y_3 whose "go" probabilities represent $F(x)$. Then, we can give payoffs to the first three players that ensure in any NE, they agree with y_1, y_2, and y_3 and must be a fixed point. There is an issue where the comparator gadget is "brittle", meaning if the inputs are equal then it can output anything. To solve our issue, we average the results of F at a grid of many points near the point of interest. This makes the computation robust but introduces a small error to F. Therefore, it works approximately, and the three players play an approximate fixed point at equilibrium.

The final step is to convert this game of n players with 2^n strategies into a graphical game [3] of three players where each player's payoff depends only on their neighbours actions. In the graphical game, we colour each player one of three colours such that no two players who either play together or are involved with the same third player have the same colour. We introduce three "lawyers" that represent all nodes of a given colour, producing a game involving only lawyers. Due to the colouring, the lawyers will have no "conflict of interest" and so an MNE of the lawyers' game will correspond to a MNE of the original graphical game. The lawyers then assign equal probability to each player by playing a generalisation of rock-paper-scissors. This completes the reduction from graphical games to three-player games, proving that NASH is **PPAD**-complete.

3.4 The Complexity Class PLS

Polynomial Local Search (**PLS**) is the class of problems where each instance x has a set of feasible solutions F_x of polynomial length with a neighbourhood structure such that $N(s)$ is the set of neighbours of a solution s, and a function $d : F_x \to \mathbb{R}$ defining the "cost" of a solution. The goal of a **PLS** problem is to find a solution $s \in F_x$ which is a local optimum of d. Completeness in **PLS** is defined with respect to **PLS** reductions which map instances of some **PLS** problem L_1 to instances of L_2 such that the local optima are preserved.

Given that Rosenthal's potential function $\Phi(s) = \sum_e \sum_{j=1}^{f_s(e)} d_e(j)$ is an exact potential function for congestion games, i.e. $\Phi(x, s_{-i}) - \Phi(y, s_{-i}) = u_i(x, s_{-i}) - u_i(y, s_{-i})$, a local minimum of Φ is a Nash equilibrium. Therefore, finding a PNE of a congestion game is a **PLS** problem.

3.5 Finding PNE in Symmetric Network Congestion Games

In a symmetric network congestion game, all players have the same start (a) and goal (b) vertices. A polynomial algorithm for finding PNE in such games is shown via a reduction to MIN-COST FLOW [4]. Given the network $N = (V, E, a, b)$ with delay functions d_e, replace each edge e with n parallel edges, each with capacity 1, and costs $d_e(1), d_e(2), ..., d_e(n)$. Any min-cost flow over this network is a state of the network congestion game that minimises Rosenthal's potential function: If j players use an edge then j of the parallel edges must be used by the flow. A minimum flow of this form must use the edges with costs $d_e(1), ..., d_e(j)$. This means that for every edge in our original network, the flow adds cost $\sum_{k=1}^{j} d_e(k)$ for all the players using it. Summing over all edges in a flow, we get Rosenthal's potential function as the cost of the flow.

3.6 PLS-completeness of all Other Instances of Congestion Games

Besides Symmetric Network Congestion games, there are three more forms of congestion game to consider: general, symmetric, and asymmetric network. All of which we show to be **PLS**-complete [4].

PLS-completeness of General Congestion Games. This is shown via a reduction from a **PLS**-complete form of not-all-equal 3SAT [4] called POSNAE3FLIP (Positive Not-All-Equal 3 Flip). An instance of POSNAE3FLIP is a 3SAT formula containing positive literals only, a clause c has weight w_c and is "satisfied" when not all literals have equal value. An assignment's weight is the summed weight of satisfied clauses. Finally, two assignments are adjacent if one can be obtained from the other by changing the value of a single variable.

Given an instance f of POSNAE3FLIP, for each 3-clause c of weight w_c we produce two resources, e_c and e'_c, with delay 0 if there are two or fewer players, and w_c otherwise. Smaller clauses are defined similarly. Each variable has a representative player with two strategies: One containing all e_c and another containing all e'_c where the player's variable is in c. We interpret player x choosing their e_c (resp. e'_c) strategy as equivalent to x being set to true (resp. false). For a clause c, $\sum_{j \in (e_c \cup e'_c)} \sum_{k=1}^{f_j(A)} d_j(k) \leq w_c$ with equality if, and only if, c is unsatisfied. c is unsatisfied in two possible cases: if all literals are true or all literals are false. In either case, every $d_j(k)$ is 0 except $d_j(3) = w_c$. The value of Rosenthal's function on an action profile A is the sum of weights of clauses left unsatisfied by the assignment corresponding to A. So finding local minimum of Φ (a PNE) is the same as finding an assignment that maximises the weights of satisfied clauses.

PLS-completeness of Symmetric Congestion Games. This is shown via a reduction from General Congestion games. Given a congestion game with action sets $S_1, ..., S_n$, construct a symmetric game (a game where all action sets are equal) like so: Let $S'_i = \{s \cup \{e_i\} : s \in S_i\}$ for each i, where each e_i is distinct. Let each e_i have a delay function where $d_{e_i}(j) = 0$ if $j = 1$ and $d_{e_i}(j) = M$ (a very large number) if $j \geq 2$. Finally, let the action set for our symmetric game be $S = \bigcup_i S'_i$. For any equilibrium, only one player will use a strategy from S'_i, otherwise they induce a delay M. For a sufficiently large M, there is no way this could

be an equilibrium. Therefore, given an equilibrium for the symmetric game we can generate an equilibrium of the non-symmetric game by renaming players (so the player using strategy S'_i becomes player i).

PLS-completeness of Asymmetric Network Congestion Games. This is proven via a reduction from WXPNAE3FLIP. An extended NAE3 formula has three additional kinds of clauses, satisfied in different ways [4]. For a given WXPNAE3FLIP formula, we define a witness graph W in which each variable x has two nodes, (x, s) and (x, t). Furthermore, each clause has a number of nodes. For example, the NAE clauses have two nodes, where a variable passes through one if it's true and the other if it's false. We define the edges so that each variable x has two "standard paths" from (x, s) to (x, t), both corresponding to an assignment to the variable. A WXPNAE3FLIP instance is a formula F and a valid weighted witness graph W (weighted so that the standard paths are strictly the shortest paths from (x, s) to (x, t)), with the goal being to find a truth assignment whose total weight is a local maximum.

The reduction from WXPNAE3FLIP to asymmetric network congestion games expands each clause-related node into an edge between two nodes. The delays on these clause-edges are set to be the weight imposed by the clause given the number of variables using it. The delays on other edges are set to be incomparably large if too many variables use them (i.e. more variables than the number of standard paths the edge is on). The idea being that any action profile in which each variable uses only its standard path corresponds directly to a valid truth assignment. Therefore, in a PNE of this game, each variable must use one of its standard paths and no variable may decrease the weight induced by unsatisfied clauses by changing path.

4 Discussion and Further Research

Two key questions arise from these results: How applicable to other games is [4]'s method of finding PNE, and what does this (in light of [3]'s results) say about the feasibility of MNE as a prediction of player behaviour?

Basic utility games, network creation games, and congestion games are all cases in which the Nash dynamics converge and are proven to do so via a general potential function [4]. The union of two games is also proven to possess PNE (both with cross-monotonic payoffs), increasing the number of games for which this is a viable strategy. However in some cases, induction can also be used within polynomial time [4].

Since NASH is **PPAD**-complete [3] and **PPAD** is a subclass of **NP**, the existence of a polynomial-time algorithm to find MNE is unlikely, leading us to question the feasibility of using MNE as predictions of player behaviour. However, this concern may be somewhat counterbalanced by the discovery of polynomial-time algorithms for computing ANE in the case of bimatrix games [2].

In the case of calculating PNE, one limitation is that proofs using exact potential functions apply to congestion games only, since any exact potential game is isomorphic to a congestion game [5]. This paper also acknowledges that potential methods aren't always viable, for example in games with cyclic dynamics where a potential function exists but is not computable [4]. As we only give proof in one-direction, further research may include investigating whether there exist games with convergent Nash dynamics but no computable potential function.

References

[1] Vincent Conitzer and Tuomas Sandholm. "Complexity results about Nash equilibria". In: *arXiv preprint cs/0205074* (2002).

[2] Artur Czumaj et al. "Distributed methods for computing approximate equilibria". In: *Algorithmica* 81.3 (2019), pp. 1205–1231.

[3] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. "The complexity of computing a Nash equilibrium". In: *SIAM Journal on Computing* 39.1 (2009), pp. 195–259.

[4] Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. "The complexity of pure Nash equilibria". In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. 2004, pp. 604–612.

[5] Dov Monderer and Lloyd S Shapley. "Potential games". In: *Games and economic behavior* 14.1 (1996), pp. 124–143.

[6] Robert W Rosenthal. "A class of games possessing pure-strategy Nash equilibria". In: *International Journal of Game Theory* 2.1 (1973), pp. 65–67.