Solutions to Nash Equilibrium Problems

Zhaohui Wang

March 14, 2025

Solution to Problem 1.1

To find a Nash equilibrium for a two-player game defined by a game matrix, we present a finite algorithm:

1. **Enumerate all strategy combinations:** Suppose player 1 has m strategies and player 2 has n strategies. There are $m \times n$ possible pure strategy combinations.
2. **Check the Nash equilibrium condition:** For each combination (s_1, s_2), compute:
   - The payoff $U_1(s_1, s_2)$ for player 1 choosing s_1.
   - The payoff $U_2(s_1, s_2)$ for player 2 choosing s_2.

   Verify that neither player benefits from unilateral deviation:

   - If there exists an s_1' such that $U_1(s_1', s_2) > U_1(s_1, s_2)$, then (s_1, s_2) is not a Nash equilibrium.
   - If there exists an s_2' such that $U_2(s_1, s_2') > U_2(s_1, s_2)$, then (s_1, s_2) is not a Nash equilibrium.
3. **Complexity:** This brute-force approach has a complexity of $O(mn)$. Finding mixed strategy equilibria generally requires algorithms such as Lemke-Howson or linear programming, which may run in exponential time.

Solution to Problem 1.2

Consider a two-player game where each player has n strategies. The payoffs are selected independently and uniformly at random from $[0, 1]$. We show that the probability of a pure Nash equilibrium approaches $1 - \frac{1}{e}$ as $n \to \infty$.

1. **A strategy combination (i, j) is a pure Nash equilibrium if:**
   - $U_1(i, j)$ is the maximum in row i.

- $U_2(i,j)$ is the maximum in column j.

2. The probability that $U_1(i,j)$ is the maximum in row i is $\frac{1}{n}$, and similarly, the probability that $U_2(i,j)$ is the maximum in column j is $\frac{1}{n}$. Since these are independent,

$$P((i,j) \text{ is a Nash equilibrium}) = \frac{1}{n} \times \frac{1}{n} = \frac{1}{n^2}.$$

3. There are n^2 strategy pairs, so the probability that at least one is a Nash equilibrium is:

$$P(\text{at least one Nash equilibrium}) = 1 - \left(1 - \frac{1}{n^2}\right)^{n^2}.$$

4. Taking the limit as $n \to \infty$:

$$\lim_{n\to\infty} \left(1 - \frac{1}{n^2}\right)^{n^2} = \frac{1}{e}.$$

5. Thus, the probability that a Nash equilibrium exists is approximately $1 - \frac{1}{e}$ for large n.

Solution to Problem 1.3

We examine why finding a Nash equilibrium in a three-person zero-sum game is at least as hard as in general two-player games.

- **Two-player zero-sum games:** Nash equilibria can be computed efficiently using linear programming (polynomial time).

- **General two-player games:** Computing a Nash equilibrium is PPAD-complete, which is believed to be harder than polynomial-time solvable problems.

- **Three-player zero-sum games:** The total sum of payoffs remains zero, but each player's best response depends on the other two players, making the structure more complex.

- **Computational hardness:** Finding a Nash equilibrium in three-player zero-sum games requires solving general non-zero-sum games, which are PPAD-complete. Thus, the problem is at least as hard as finding a Nash equilibrium in a general two-player game.

Solution to Problem 1.4

Consider an n-player game where each player has only two pure strategies, and the game's structure is represented as a tree graph G with a maximum degree of 3. We need to design a polynomial-time algorithm to determine whether a pure Nash equilibrium exists in this game.

Algorithm Design

The problem can be efficiently solved using a **tree-based dynamic programming (Tree DP)** approach. Since G is a tree, it has no cycles, allowing us to use depth-first search (DFS) to process the tree efficiently.

Key Idea: Each node (player) in the tree has a payoff that depends on its strategy and the strategies of its neighbors. We traverse the tree in a bottom-up manner, ensuring that each player's best response is stable given its neighbors' choices.

Algorithm Steps: 1. Select an arbitrary node as the root and traverse the tree using DFS. 2. For each node i with parent p, check the best strategy responses based on the payoffs. 3. Propagate valid strategy choices upward to ensure that each subtree remains stable. 4. If a valid assignment of strategies exists such that no player wants to deviate, then a pure Nash equilibrium exists.

Time and Space Complexity Analysis

- Since G is a tree with n nodes, DFS runs in $O(n)$ time. - Each node has at most two possible strategy choices, leading to a constant factor check for each node. - The overall time complexity is $O(n)$. - Space complexity is also $O(n)$, as we store player strategies and the dependency graph.

Thus, the algorithm runs efficiently in polynomial time.

—

Solution to Problem 1.5

Consider an n-player symmetric game where each player has two strategies: "on" and "off." The game's payoff is determined by a function dependent only on the number of other players choosing "on." We aim to design a polynomial-time algorithm to compute a **correlated equilibrium** (CE) for this game.

Mathematical Formulation

A correlated equilibrium is a probability distribution over strategy profiles such that no player has an incentive to deviate unilaterally. Given:

- $u_{on}(k)$: Payoff for playing "on" when k other players also play "on."
- $u_{off}(k)$: Payoff for playing "off" when k other players also play "on."

The goal is to find a probability distribution $P(s)$ over strategy profiles such that for all players:

$$\mathbb{E}[U_i|s_i] \geq \mathbb{E}[U_i|s_i']$$

where U_i is the expected utility given a recommended strategy s_i.

Algorithm Using Linear Programming

1. Define a probability variable $P(s)$ for each possible strategy profile. 2. Set up linear constraints ensuring that each player's expected utility from following their recommended strategy is at least as high as any unilateral deviation. 3. Solve the resulting linear program (LP) to find an optimal probability distribution.

Complexity Analysis

- The number of constraints in the LP is proportional to $O(n2^n)$ due to the number of strategy profiles. - Solving an LP takes at most polynomial time in the number of constraints, leading to an overall complexity of $O(n^2)$ for practical input sizes. - Space complexity is $O(2^n)$ due to storage of probability distributions.

Thus, the approach efficiently finds a correlated equilibrium in polynomial time.

Solution to Problem 1.6

Problem Analysis

Consider a 2-player game in matrix form where each player has n pure strategies. In a Nash equilibrium, players may use mixed strategies with nonzero probability assigned to many pure strategies. The objective is to show that an ϵ-approximate Nash equilibrium can be constructed where each player plays a strategy with a support of size at most $O(\log n)$.

Model Assumptions

Let p^j represent the mixed strategy of player j, where p_i^j is the probability of using the i-th pure strategy. The support of p^j is defined as:

$$S^j = \{i : p_i^j > 0\}$$

An ϵ-approximate Nash equilibrium satisfies that for both players, no alternative strategy $\tilde{p}^j$ improves expected payoff by more than ϵM, where M is the maximum possible payoff.

Mathematical Derivation

1. Start with any Nash equilibrium where players use mixed strategies. 2. Identify the most significant $O(\log n)$ strategies contributing to the expected payoff. 3. Construct a new mixed strategy by choosing from these $O(\log n)$ strategies with uniform probability. 4. Show that this approximation introduces at most an ϵM loss in expected payoff.

Algorithm Construction

- Extract a Nash equilibrium mixed strategy. - Reduce the support to the most influential $O(\log n)$ pure strategies. - Randomly select one of these strategies with uniform probability.

Time and Space Complexity Analysis

- Finding a Nash equilibrium is PPAD-complete, but reducing the support has complexity $O(n)$. - Constructing the approximate strategy takes $O(\log n)$ operations. - Overall complexity is $O(n)$.

Solution to Problem 1.7

Problem Analysis

The classical Bertrand competition involves n firms producing the same product and competing in quantity. The price $p(q)$ follows a demand-price function, which is monotonically decreasing and differentiable.

Model Assumptions

- Each firm i produces quantity q_i, and total supply is $q = \sum q_i$. - The price adjusts according to a function $p(d)$, e.g., $p(d) = 1 - d$. - Each firm maximizes its revenue $q_i p(q)$.

Mathematical Derivation

(a) Monopolist vs. Competition: - A single firm chooses q_m to maximize $q_m p(q_m)$. - With n firms, each firm optimizes $q_i p(q)$. - Compare revenue under both cases to show monopolist advantage.

(b) Second derivative condition: - Assume $p''(d) \leq 0$. - Show that monopolistic income is at most n times the competitive income.

Algorithm Construction

- Compute competitive equilibrium quantities for all firms. - Compute monopoly optimal production. - Compare revenues.

Time and Space Complexity Analysis

- Finding competitive equilibrium takes $O(n)$. - Computing monopoly outcome is $O(1)$. - Overall complexity: $O(n)$.

Solution to Problem 1.8

Problem Analysis

A minimum spanning tree (MST) game involves n agents and a graph $G = (V, E)$ with edge costs. The goal is to find a budget-balanced cost-sharing method for constructing an MST that lies in the core.

Model Assumptions

- Graph G is complete with edge costs satisfying the triangle inequality. - MST T is rooted at node 0. - The cost of an agent is the first edge in the path to 0 in T.

Mathematical Derivation

- Construct T using Prim's or Kruskal's algorithm. - Assign costs based on the first edge in the unique path to the root. - Show that the sum of assigned costs equals the MST cost. - Prove that any subset S of agents is charged at most $c(S)$.

Algorithm Construction

- Compute MST using Kruskal's or Prim's algorithm. - Assign costs to agents based on their first connecting edge. - Verify core constraints.

Time and Space Complexity Analysis

- MST computation: $O(m \log n)$. - Cost assignment: $O(n)$. - Overall complexity: $O(m \log n)$.