Algorithmic Game Theory Project: Network Security Games

Di Franco Federico
Serratore Francesco
Visciglia Domenico

December 28, 2025

Contents

1 Introduction

The primary objective of this project is to address the **Network Security problem**: securing a graph $G = (V, E)$ such that the traffic on every edge is controlled by at least one secure node. According to the project specification, a set of nodes $S \subseteq V$ is defined as a *Network Security Set* if, for every node $i \in V$, either $i \in S$ or all its neighbors $N(i)$ are in S ($N(i) \subseteq S$). We observe that this condition implies that every edge in the graph must be incident to at least one node in S, making the problem topologically equivalent to the **Minimum Vertex Cover** (MVC) problem, which is known to be NP-hard.

We approach this optimization problem through the lens of Algorithmic Game Theory, decomposing the analysis into four distinct computational tasks:

1. **Strategic Game Modeling:** Defining a non-cooperative game where Pure Nash Equilibria (PNE) correspond to valid security sets.
2. **Coalitional Analysis:** Using Cooperative Game Theory and Shapley Values to assess the marginal importance of each node in maintaining network security.
3. **Market Allocation:** Simulating an economic environment where secure nodes (Buyers) purchase services from Vendors subject to budget and capacity constraints.
4. **Mechanism Design:** Implementing a truthful VCG (Vickrey-Clarke-Groves) auction to procure secure paths within the network.

Exact Benchmark (Bonus Analysis). To rigorously validate the heuristic results obtained via Game Theory, we went beyond the core requirements and implemented an exact solver based on Boolean Satisfiability (SAT). While this approach is inherently limited by the NP-hard nature of the problem and does not scale to large networks, it provides a crucial **"Ground Truth"** for smaller instances. This allows us to benchmark the quality (minimality) of the Nash Equilibria found by our strategic algorithms against the mathematically optimal solution. All experiments are conducted on three distinct network topologies: Regular, Erdős-Rényi, and Barabasi-Albert graphs.

2 Task 1: Strategic Game Modeling

We modeled the Network Security problem as a strategic game $\Gamma = \langle N, \{S_i\}, \{u_i\} \rangle$ where nodes act as selfish agents aiming to minimize their cost while ensuring security.

2.1 Game Formulation

The game environment is implemented in the class `SecurityGame`.

- **Players (N):** The set of nodes V in the graph.
- **Strategies (S_i):** Each player i selects a strategy $s_i \in \{0, 1\}$.
  - $s_i = 1$: The node secures itself (incurring a cost c).
  - $s_i = 0$: The node does not secure itself (relying on neighbors).

- **Payoff Function (u_i):** The utility function logic, defined in the `get_payoff` method, incentivizes "Free-Riding" only if the node is safely covered by at least one neighbor. Let $N(i)$ be the set of neighbors of i. The payoff is:

$$u_i(s) = \begin{cases} \alpha - c & \text{if } s_i = 1 \\ \alpha & \text{if } s_i = 0 \land \exists j \in N(i) : s_j = 1 \\ 0 & \text{if } s_i = 0 \land \forall j \in N(i) : s_j = 0 \end{cases} \tag{1}$$

  We utilized parameters $\alpha = 10$ (Value of Security) and $c = 4$ (Cost of Implementation). Since $\alpha > c > 0$, the game structure is a *Game of Strategic Substitutes* (or local Public Good game), where a neighbor's investment reduces the incentive for a player to invest.

2.2 Iterative Equilibrium Finding Algorithms

To find Pure Nash Equilibria (PNE), which correspond to minimal security sets, we implemented three learning dynamics.

Best Response Dynamics (BRD) Implemented in `BestResponseDynamics.py`, this algorithm models myopic rational players.

- **Update Rule:** In each iteration, we iterate through all players sequentially. Player i calculates the hypothetical payoff of switching strategies given the *current* profile of opponents s_{-i}.
- **Strict Improvement:** The player switches strategy only if the new payoff is strictly greater than the current one ($u_{new} > u_{old}$). This strict inequality prevents cycles in indifferent states.
- **Convergence:** The algorithm terminates when a full round passes with no strategy changes, indicating a Nash Equilibrium has been reached.

Fictitious Play (FP) Implemented in `FictitiousPlay.py`. Unlike BRD, players respond to the *historical frequency* of opponents' play rather than the current state.

- **Belief System:** Each node i maintains `neighbor_counts`, tracking how often each neighbor j has played '1'. The empirical probability is estimated as $\hat{p}_j = \frac{\text{count}(j=1)}{\text{total\_rounds}}$.
- **Joint Probability:** Assuming independence between neighbors, the probability that *all* neighbors cover i is estimated as $P_{all} = \prod_{j \in N(i)} \hat{p}_j$.
- **Deterministic Threshold Response:** We implemented a deterministic variation where node i plays 0 if and only if the probability of being covered exceeds the cost-benefit ratio:

$$s_i^{(t+1)} = \begin{cases} 0 & \text{if } P_{all} > \frac{\alpha - c}{\alpha} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

  This approach provided robust convergence by filtering out noise from stochastic updates.

Regret Matching (RM) Implemented in `RegretMatching.py`. This algorithm seeks to minimize cumulative regret over time.

- **Regret Calculation:** At each step, player i compares the actual payoff $u(s_i, s_{-i})$ with the counterfactual payoff $u(s'_i, s_{-i})$ they would have received by playing the alternative strategy.
- **Cumulative Regret Update:** $R_i^{s'} \leftarrow R_i^{s'} + \max(0, u_{counterfactual} - u_{actual})$.
- **Action Selection:** Strategies are chosen probabilistically, with probability proportional to the positive cumulative regret (or uniform if no regret exists).
- **PNE Extraction:** Since RM converges to a Correlated Equilibrium (often mixed), we extract a pure strategy for the final configuration by applying a hard threshold ($p > 0.5 \implies 1$) to the final probability distribution.

3 Task 2: Coalitional Game Theory

In the second task, we shift our perspective from non-cooperative agents to a cooperative framework. Here, the focus is on assessing the "value" or importance of each node in forming a valid Network Security Set. To do this, we model the problem as a coalitional game and compute the **Shapley Value** for each node, which represents its average marginal contribution to the coalition's goal.

3.1 Shapley Value Approximation

The exact computation of Shapley values requires evaluating $2^{|N|}$ coalitions or summing over $|N|!$ permutations. Since this is computationally intractable for large networks, we implemented a **Monte Carlo approximation** in the class `CoalitionalSecurityGame`.

Marginal Contribution Logic We defined the marginal contribution of a node i to a coalition S based on the strict "ALL Rule" required by the Network Security Set definition. In our implementation, adding node i to a permutation generates a value of $+1$ if and only if it actively increases the security coverage of the network in one of two ways:

1. **Direct Contribution:** Node i itself becomes secured by entering the coalition (and it was not already secured by its neighbors).
2. **Indirect Contribution (Critical):** Node i completes the security requirement for a neighbor $j \notin S$.
   - According to the problem definition, a node j outside the set is secure only if *all* its neighbors are in the set.
   - Thus, node i generates value for j if and only if i is the **last missing neighbor** required to complete j's coverage.

Algorithm Implementation The approximation algorithm works as follows:

- We generate a fixed number of random permutations of the players (parameterizable in the code).
- For each permutation, we scan nodes sequentially.
- We maintain a counter `current_neighbors_in_S` for every node to track how many of its neighbors have already appeared in the permutation.
- If adding node i increments this counter for neighbor j to match its degree ($deg(j)$), and j is not in the set, we credit a contribution to i.

3.2 Set Construction: Reverse Greedy Heuristic

Once the Shapley values are computed, we use them as a ranking metric to construct a **Minimal Network Security Set**. Instead of a standard greedy approach (adding best nodes), which often fails to satisfy the strict "ALL neighbors" constraint efficiently, we implemented a **Reverse Greedy (Pruning)** approach:

1. **Initialization:** We start with the full set of nodes $S = V$ (which is trivially a secure set).
2. **Sorting:** We rank nodes by their Shapley values in **ascending order** (from least to most important).
3. **Pruning:** We iterate through the sorted list and attempt to remove each node i from S.
4. **Feasibility Check:** Node i can be removed if and only if its removal does not compromise the security of any neighbor.
   - Specifically, if a neighbor j is already excluded from the set ($j \notin S$), it relies entirely on its neighbors (including i) for security.
   - Therefore, if any neighbor of i is not in S, node i **cannot** be removed, as doing so would leave that neighbor uncovered.

This heuristic proved highly effective. In structured networks like Barabasi-Albert, the high-degree hubs naturally obtained high Shapley values (due to their potential to cover many leaves) and were correctly retained by the pruning process, while peripheral nodes were removed.

4 Task 3: Market Allocation

In this task, we simulate an economic environment where the nodes identified in the Security Set act as **Buyers** who must purchase security services from a set of **Vendors**. The goal is to maximize the aggregate Social Welfare of the system under different supply constraints.

4.1 Market Model

The market simulation is implemented in the class `SecurityMarketplace`.

- **Agents:**
  - **Buyers:** The subset of secure nodes found in previous tasks. Each buyer b is assigned a random budget $B_b \in [1, 100]$.
  - **Vendors:** A set of service providers, each characterized by a price P_v, a security level $Q_v \in [1, 10]$, and a capacity K_v (maximum number of clients).
- **Utility Function:** We defined a utility function $U(b, v)$ that captures the trade-off between the quality of the service (security level) and the economic savings (budget surplus). To prioritize security quality over mere cost savings, we applied a weight factor (multiplier of 10) to the security level. The utility is defined as:

$$U(b, v) = \begin{cases} (Q_v \times 10) + (B_b - P_v) & \text{if } B_b \geq P_v \\ -\infty & \text{if } B_b < P_v \quad \text{(Incompatible)} \end{cases} \tag{3}$$

4.2 Allocation Mechanisms

We analyzed two distinct market scenarios, implementing specific algorithms for each to maximize welfare.

Scenario 1: Infinite Capacity (Unbounded Supply) In this scenario, vendors have infinite items ($K_v = \infty$). Since there is no rivalry for goods, the global optimum coincides with the local optimum for each agent.

- **Algorithm:** We implemented a **Local Greedy Search**. Each buyer independently iterates through all available vendors and selects the one that maximizes their individual utility $U(b, v)$.
- **Outcome:** This mechanism naturally leads to a Pareto-optimal allocation where the "Best Value" vendors (high quality/price ratio) capture the entire market share compatible with their price point.

Scenario 2: Limited Capacity (Scarcity) In this scenario, vendors have a finite capacity K_v. Simple local maximization is no longer sufficient, as early movers could deplete high-value resources, leaving others unmatched. To address this, we implemented a **Global Greedy** allocation strategy aimed at maximizing Total Social Welfare:

1. **Feasible Set Generation:** We generate a list of all potential valid matches (b, v) where $B_b \geq P_v$.
2. **Global Sorting:** This list is sorted in descending order based on the utility value $U(b, v)$.
3. **Greedy Assignment:** We iterate through the sorted list. A match (b, v) is finalized if and only if:
   - The buyer b is not yet matched.

- The vendor v has remaining capacity ($Sales_v < K_v$).

Observation: This approach effectively illustrates the **"Crowding Out"** effect. High-budget buyers tend to generate higher utility (due to the surplus term) and thus secure the best vendors first. Once the high-quality stock is depleted, other buyers are forced to "second-best" options or remain unmatched, highlighting the inefficiencies introduced by scarcity.

5 Task 4: VCG Path Auction

In the final task, we address the problem of procuring a secure path from a source node s to a target node t. We model this as a truthful mechanism design problem where the agents are the nodes themselves, each "selling" the right to traverse them.

5.1 Mechanism Design

The auction logic is encapsulated in the class `VCGPathAuction`. Since standard shortest path algorithms operate on edge weights, while our costs are associated with nodes, we implemented a specific graph transformation.

Graph Transformation and Cost Function To model node-based costs using standard Dijkstra algorithms, we constructed a temporary directed graph `temp_G`. For every undirected edge (u, v) in the original graph, we created two directed edges:

- $(u \to v)$ with weight w_v.
- $(v \to u)$ with weight w_u.

The "Social Cost" w_i of traversing node i is defined as the sum of its private cost (bid) and a public penalty representing the insecurity externality:

$$w_i = c_i + \text{Penalty}(i) \tag{4}$$

where $c_i \in [1, 20]$ is the private random bid, and $\text{Penalty}(i) = 10$ if $i \notin$ Security Set (0 otherwise). This effectively internalizes the negative externality of using insecure nodes into the path cost.

5.2 VCG Payment Rule

We implemented the Vickrey-Clarke-Groves (VCG) mechanism, which ensures truthfulness (dominant strategy incentive compatibility). The payment P_i for a winning node i on the optimal path is calculated as the externality it imposes on the system (the "harm" to social welfare if it were absent).

The formula used is:

$$P_i = \text{SP}(G \setminus \{i\}) - (\text{SP}(G) - w_i) \tag{5}$$

where $\text{SP}(G)$ denotes the total cost of the shortest path from s to t in graph G.

Implementation Details The method `run_vcg_mechanism` executes the following steps:

1. **Optimal Allocation:** Compute the shortest path on the full graph to identify the winning coalition of nodes.
2. **Payment Calculation:** For each winning node i (excluding source and target):
   - We create a temporary graph copy removing node i ($G \setminus \{i\}$).
   - We re-run the shortest path algorithm to find the best alternative solution ($\text{SP}(G \setminus \{i\})$).
   - **Monopoly Handling:** If removing node i disconnects s and t (making the target unreachable), the node is a "Bridge" with infinite monopoly power. In this case, the algorithm correctly assigns an infinite payment (or a very large cap), reflecting the lack of alternatives.

5.3 Visual Analysis

The results are visualized by color-coding the optimal path: green nodes represent secure transit points (low social cost), while red nodes indicate insecure points where the high penalty was paid because avoiding them would have been even more expensive (e.g., bottlenecks).

6 Bonus Section: Exact Benchmark via SAT Solver

As an advanced validation step, we implemented an exact solver to determine the true Minimum Vertex Cover (MVC). Since the problem is NP-hard, heuristic algorithms (BRD, Shapley) provide approximations that are not guaranteed to be minimal. To benchmark their quality, we formulated the problem as a Boolean Satisfiability (SAT) instance.

6.1 Symbolic Encoding

The solver is implemented in `SatVertexCover.py` using the `SymPy` library for symbolic logic. We mapped the graph problem into a boolean formula Φ as follows:

- **Variables:** For each node $i \in V$, we defined a boolean variable x_i, where $x_i = \text{True}$ implies $i \in S$.
- **Edge Constraints:** To ensure validity, every edge $(u, v) \in E$ generates a clause $(x_u \vee x_v)$. The conjunction of all such clauses enforces the Vertex Cover property.

6.2 Cardinality Constraint: The Sequential Counter

The core challenge in SAT is enforcing the constraint "Choose at most k nodes" ($\sum x_i \leq k$), as boolean logic lacks arithmetic operators. Instead of naive encodings (which explode combinatorially), we implemented a **Sequential Counter** circuit.

We introduced auxiliary variables $S_{i,j}$ representing the state: "The sum of the first i variables is at least j". The logic is defined recursively:

$$S_{i,j} \iff S_{i-1,j} \vee (S_{i-1,j-1} \wedge x_i) \tag{6}$$

This circuit effectively "counts" the active variables as we traverse the list of nodes. The final constraint $\neg S_{N,k+1}$ ensures that the total count never exceeds k.

6.3 Optimization Strategy

To find the *minimum* k:

1. We perform a **Binary Search** on the integer $k \in [0, |V|]$.
2. For each step k, we construct the formula $\Phi_k = \Phi_{\text{Edges}} \wedge \Phi_{\text{Count}\leq k}$.
3. We query the SAT solver: if Φ_k is satisfiable, we try a smaller k; otherwise, we increase k.

Results. This exact approach provided a rigorous "Ground Truth". For small instances ($N \approx 50$), we confirmed that the Nash Equilibria found by our Strategic Games were often optimal or very close to optimal (size difference $\leq 1-2$ nodes), validating the effectiveness of the game-theoretic heuristics.

7 Conclusion and Comparative Analysis

This project successfully established a comprehensive framework to analyze Network Security through the lens of Algorithmic Game Theory. By integrating non-cooperative dynamics, cooperative stability analysis, and mechanism design, we moved from identifying theoretical security sets to simulating their economic implications.

Topological Insights. The experimental results, which can be explored interactively via the provided **HTML Dashboard**, highlighted significant differences in how network topology influences security strategies:

- **Regular Graphs:** Due to the structural homogeneity (constant degree k), we observed a uniform distribution of responsibilities. Shapley values appeared evenly distributed, indicating that no single node is critical. Strategic algorithms (BRD, FP) converged to equilibria where secure nodes were scattered in a pattern to cover the graph without clear "leaders".
- **Barabasi-Albert (Scale-Free) Graphs:** This topology provided the most distinct insights. The presence of **Hubs** (high-degree nodes) created a strong hierarchy.
  - *Coalitional:* The Monte Carlo Shapley approximation assigned disproportionately high values to hubs, correctly identifying them as critical for network coverage.
  - *Strategic:* The game dynamics rapidly converged to configurations where hubs entered the security set (Strategy 1) to cover their multitude of leaf neighbors (Strategy 0), effectively acting as "public good" providers.
  - *Comparison with SAT:* Our Exact SAT Solver confirmed that on Scale-Free networks, the heuristic sets dominated by hubs are often identical or extremely close to the true Minimum Vertex Cover.
- **Erdős-Rényi Graphs:** exhibited behavior intermediate between the two, with local clusters of importance emerging based on random degree variance.

Economic Implications. The **Market Simulation** demonstrated that while Infinite Capacity leads to efficient Pareto-optimal allocations, the introduction of scarcity (Limited Capacity) creates a "Crowding Out" effect, where wealthier agents monopolize high-quality vendors, potentially forcing others into an unmatched state despite a willingness to pay. Finally, the **VCG Auction** revealed how topology dictates market power: in Regular graphs, alternative paths kept VCG payments close to private costs, whereas in Barabasi-Albert graphs, hubs acting as "bridges" between clusters could extract significantly higher payments (or infinite, in case of disconnection), reflecting their monopoly on connectivity.

In summary, this work illustrates that while game-theoretic equilibria provide robust solutions for decentralized security, the underlying network topology is the primary determinant of both the cost of security and the distribution of market power.