Project Milestone 2: Finding Optimal Taxation Policy With RL

Ardhito Nurhadyansah[1] and Ian Suryadi Timothy H[2]

[1]2106750206
[2]2106750875

Abstract

In this project, we simulate the economic output, given a tax policy. This project is adapted from the work of (Zheng et al., 2020) , where the model is comprises of two interconnected Markov Decision Process. The inner-loop MDP is the agents MDP, which have partial observations and interact to each other in a gather-and-build simulation. The outer-loop MDP models a social planner that find the optimal tax policy to maximize a social welfare function, which is formulated by a product of equality and economic productivity.

1 Problem Definition

Tax is one tool for a country to reduce inequality. Traditional tax systems are often derived from static models or rely on simplifying assumptions. However, real-world economies are dynamic and complex, featuring uncertain behaviors, heterogeneous agents, and evolving macroeconomic conditions. In this project, we are experimenting with tax policies and observe its effect on equality and productivity. This project is adapted from the work of (Zheng et al., 2020) , with some adjustments regarding compute power availability. The problem is formulated as two interconnected Markov Decision Processes (MDPs):

- Agents' MDP (inner loop): models the individual behaviors of economic agents.
- Planner's MDP (outer loop): models the tax-setting problem to optimize a social welfare function.

1.1 Agents' MDP (Inner Loop)

Each agent operates in a partially observable environment defined by:

- **State Space (S):**
  - Local Spatial Information: Agent's position in a 2D grid and nearby environmental features (e.g., resource tiles, obstacles).
  - Resource Availability: Status of resource regeneration (e.g., wood and stone).
  - Agent Attributes: Individual coin endowment, resource inventory, accumulated labor, and skill level (which affects income generation, e.g., coins earned per house built).
  - Market State: Public trading offers (bids and asks).
  - Tax Policy: The current tax schedule applied to income.
- **Action Space (A):**

$$a \in \{\text{Movement, Resource Collection, Building, Trading}\}$$

- **Transition Function (T):** The dynamics update the state based on:

  - Movements and interactions (e.g., collisions, blocked paths).
  - Stochastic resource regeneration.
  - Market execution and trade outcomes.
  - Periodic taxation that applies a bracketed tax schedule and redistributes income.
- **Reward Function (r):** The instantaneous reward for agent i at time t is defined as the change in its utility:

$$r_{i,t} = u(x_{i,t}, l_{i,t}) - u(x_{i,t-1}, l_{i,t-1}),$$

  where the utility function is defined as

$$u(x, l) = \frac{x^{1-\eta} - 1}{1 - \eta} - l, \quad \eta > 0,$$

  with x representing coins and l the cumulative labor.
- **Discount Factor (γ):** Each agent maximizes its expected discounted return:

$$\max_{\pi_i} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t \, r_{i,t}\right].$$

1.2 Planner's MDP (Outer Loop)

The planner sets tax policies with the goal of maximizing overall social welfare, based on aggregated economic outcomes.

- **State Space (S_p):** The planner observes aggregated indicators:
  - Aggregate Wealth Distribution: Summary statistics (e.g., mean, variance, Gini coefficient) of agents' incomes.
  - Productivity: Total income generated by the agents.
  - Market and Resource Summaries: Global resource availability and trading volumes.
  - Tax History: Current and previous tax schedules and tax revenue collections.
- **Action Space (A_p):** The planner's decision is to set a tax schedule:

$$a_p = [\tau_0, \tau_1, \ldots, \tau_{B-1}],$$

  where each $\tau_b \in [0, 1]$ is the marginal tax rate for bracket b.
- **Transition Function (T_p):** The state evolves based on the effect of the tax policy on the agents' behaviors during a tax period. The agents' adaptive responses determine the new aggregate economic state.
- **Reward Function (r_p):** The planner receives a reward based on the improvement in a social welfare function. For example:

$$r_p = \text{swf}(s'_p) - \text{swf}(s_p),$$

  where a candidate social welfare function is:

$$\text{swf} = \text{Equality}(x_c) \times \text{Productivity}(x_c),$$

  or alternatively a weighted sum of individual utilities:

$$\text{swf} = \sum_i \omega_i \, u(x_{c,i}, l_i).$$

- **Discount Factor (γ_p):** The planner maximizes:

$$\max_{\pi_p} \mathbb{E}\left[\sum_{p=0}^{P} \gamma_p^p \, r_p\right].$$

2 Solution Design

A two-level deep reinforcement learning framework will be used, integrating the agents and planner's MDP.

2.1 Inner Loop (Agents' Learning)

- Agents interact in the environment and learn policies π_i using deep RL algorithms (e.g., PPO).
- Their objective is to maximize the expected discounted return based on their MDP, with rewards that reflect changes in utility (coins versus labor cost).

2.2 Outer Loop (Planner's Learning)

- The planner observes an aggregate state s_p (wealth distribution, productivity, etc.) and sets a tax schedule a_p for the upcoming tax period.
- Its policy π_p is trained with deep RL methods to maximize improvements in social welfare:
$$r_p = \text{swf}(s'_p) - \text{swf}(s_p).$$

2.3 Interaction Dynamics

- Influence of Planner on Agents: The tax schedule (action a_p) directly affects agents' rewards by modifying post-tax incomes.
- Influence of Agents on Planner: The agents' adaptive responses under the new tax policy determine the next aggregate state s'_p that the planner observes.

3 Preliminary Analysis

Our preliminary analysis builds upon the findings reported by (Zheng et al., 2020) , that the two-level reinforcement learning framework can capture complex dynamics. Specifically, the following observations were made:

- **Agents' Adaptive Behavior:**
  - Agents naturally exhibit specialization, where lower-skilled agents tend to focus on resource collection while higher-skilled agents emphasize building. This division of labor is a key emergent property observed in the simulation.
  - Agents dynamically adjust their strategies in response to the imposed tax schedule, demonstrating the non-stationarity of the inner-loop MDP and confirming that individual behaviors are influenced by external economic incentives.
- **Impact of Tax Policies on Social Welfare:**
  - The RL-based planner is capable of setting tax policies that strike an effective balance between income equality and overall productivity. They report that the AI-driven tax policy achieves approximately a 16% improvement in the product of equality and productivity relative to baseline methods.
  - The simulation reveals that agents sometimes engage in tax-gaming strategies, such as alternating between high and low income periods, in order to reduce their effective tax burdens.
- **Convergence and Stability:**
  - Curriculum learning and entropy regularization was crucial in stabilizing the training process. These techniques help both the agents and the planner converge to stable policies despite the inherent non-stationarity.
  - The planner's ability to adjust tax policies based on aggregated economic indicators (e.g., wealth distribution, overall productivity) ensures that the outer-loop MDP effectively guides the inner-loop dynamics towards improved social welfare outcomes.

Building on the concepts given in the work of (Zheng et al., 2020) , our project aims to adapt the methodology for doing economic simulations, with some modifications on the training process regarding compute power availability.

4 Modeling

In this section, we describe the overall system design with a focus on:

1. The policy network architectures used by agents and the planner.

2. The key hyperparameters chosen and their rationale (e.g., learning rates, network sizes).
3. References to how the environment's code and classes will be organized in our project.

4.1 Architecture of the Policy Networks

Agent Policy Network. Our agents each implement a policy $\pi(a \mid o)$ via a neural network that processes an observation o and outputs a distribution over possible actions a. Observations include:

- *Spatial data*: local $N \times N$ grid cells (near the agent) indicating terrain, resources, or other agents.
- *Non-spatial data*: agent inventory (e.g., wood, stone, coin), skill level, and market/-trading information.
- *Planner signals*: the bracketed tax schedule or agent memories, used especially in phase 2 when the planner's tax is being activated.

We will feed the spatial data into a small convolutional stack (2–3 conv layers), flatten it, then concatenate with the non-spatial vector. After fully connected (FC) layers, an output layer produces action logits. We include an LSTM or GRU layer to help with partial observability and history.

Planner Policy Network. If the simulation includes a tax planner (Phase 2), it observes aggregated economic metrics (wealth distribution, Gini, average labor, etc.) and outputs a set of discrete bracketed tax rates $\{\tau_b\}$. We will use a simple FC architecture with a final multi-branch head (one branch per bracket).

4.2 Key Hyperparameters and Rationale

- **Batch Size & Rollouts**: According to the paper, we run multiple environment replicas in parallel (e.g., 16–30) for sample efficiency. Each replica collects around 200 steps before an update. But in regard of the compute availability, this might be adjusted.
- **Discount Factor** (γ): Close to 1 (e.g., >= 0.98–0.999) to handle multi-step horizons.
- **Entropy Regularization**: Encourages exploration; a small coefficient (0.01–0.05) for agents, and often a slightly larger one for the planner to make the agents not become too "excited" when the tax applied.

5 Algorithm

In our training algorithm, we emphasize several key aspects: sample collection and PPO (or another RL method) updates, two-phase training (initially training the agent alone in a free market, followed by a phase where the planner's policy is included), as well as entropy schedules and parallel rollouts.

5.1 Overall Procedure

First, we initialize the agent and planner policy parameters (denoted by θ and ϕ, respectively) using, for example, random network weights. Two separate buffers are also reset to record transitions: one for agent experiences and another for planner experiences. The training continues until a predefined stopping criterion is met (such as no further improvement in rewards).

For each training iteration, the environment is reset to produce an initial state, along with observations and hidden states for both the agent and the planner. The procedure then proceeds over a specified sampling horizon $\mathcal{H}$. At each timestep t within this horizon, the following steps occur:

Step 1: The agent observes its state o and, using its current policy π_θ, selects an action a while updating its hidden state h.

Step 2: At intervals defined by the tax period T (i.e., when $t \mod T = 0$), the planner, based on its observation o_p and hidden state h_p, selects a new tax schedule τ by invoking its policy π_ϕ. At other timesteps, the planner performs a no-op (but its hidden state is still updated).

Step 3: The environment then takes a step using the chosen action and tax schedule, resulting in a new state s', updated observations o' for the agent and o'_p for the planner, and corresponding pre-tax rewards r (for the agent) and r_p (for the planner).

Step 4: If the timestep is the last in the period (i.e., $t \mod T = T-1$), the environment applies taxes by updating the state and rewards accordingly.

Step 5: The agent's transition—comprising the observation, action, reward, and next observation—is logged to its buffer, and the planner's transition (the planner's observation, chosen tax schedule, planner reward, and next planner observation) is similarly stored.

Step 6: The state variables s, o, and o_p are updated to their new values for the next iteration step.

After completing the $\mathscr{H}$-step rollout, both policies are updated using an on-policy algorithm such as Proximal Policy Optimization (PPO) on the transitions stored in their respective buffers. The transition buffers are then cleared, and the next iteration begins.

5.2 Two-Phase Training

The training process is divided into two distinct phases:

Phase 1: Agent-only Training (Free Market)

In this phase, only the agent is trained. Rollouts are collected in parallel from M replicas of the environment—each initialized in a no-tax configuration. For each replica, the agent executes T timesteps using policy π_θ while storing the resulting experiences (observations, actions, and rewards). These experiences are then used to update the agent policy via PPO over one or more training epochs.

Phase 2: Agents + Planner Co-adaptation

Here, both the agent and the planner are trained concurrently. Environment replicas are now reset with tax actions enabled so that the planner can choose tax schedules. Each replica generates rollouts over T timesteps with both the agent policy π_θ and the planner policy π_ϕ active, and the collected experiences for both are stored. The agent policy is updated with PPO using its specific experiences, and similarly, the planner policy is updated using the planner's transitions. Additionally, parameters such as the maximum allowable tax rates or the entropy coefficient (which encourages exploration) may be gradually annealed during training.

Why Two Phases?

The two-phase approach addresses key challenges in the joint optimization of agent and planner policies:

1. **Learning Stability:** During early training, an untrained planner may impose arbitrary high tax rates, creating noisy reward signals that make it difficult for agents to learn effective behaviors. By first training agents in a tax-free environment (Phase 1), they can learn basic economic strategies without this interference.
2. **Smooth Tax Introduction:** When transitioning to Phase 2, we gradually introduce taxation through an annealing schedule. The maximum allowable tax rate starts at 10% and is linearly increased to 100% over the training period, allowing agents to adapt gradually to the tax environment.
3. **Policy Exploration:** To handle the complex joint learning dynamics, we employ entropy regularization for the planner's policy:

$$\text{entropy}(\pi) = -\mathbb{E}_{a\sim\pi(.|s)}[\log \pi(a|s)]$$

This promotes exploration during the critical co-adaptation period.

This approach ensures that agents first develop competent economic behaviors before introducing the complexity of tax policy optimization.

5.3 PPO and Entropy Regularization

For the policy updates, we use the standard PPO objective function:

$$L(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)A_t,\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)A_t\Big)\Big],$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid o_t)}{\pi_{\theta_{\text{old}}}(a_t \mid o_t)}$$

is the probability ratio, and A_t denotes the advantage estimate. An entropy bonus term, $\alpha_H\,\mathscr{H}[\pi]$, is also added to promote exploration, with the coefficient α_H gradually reduced over time.

5.4 Sample Efficiency and Rollouts

In each training iteration, we can collect $M \times T$ transitions. For example, if we use $M = 30$ parallel environment replicas and $T = 200$ timesteps per replica, approximately 6000 agent transitions are collected each iteration. Although the planner's actions might occur less frequently (possibly once every k timesteps), its transitions are captured each time it acts.

6 Experiment

Here, we document our *free-market* baseline experiment and illustrate how we collect data such as social welfare, as well as show how skill affects utility. The code below references Python snippets from `agent.py`, `utils.py`, etc.

6.1 Free Market Environment Setup

We create an environment where no tax component is activated. The config snippet is:

```
env_config = {
    'scenario_name': 'layout_from_file/simple_wood_and_stone',
    'components': [
        ('Build', dict(skill_dist="pareto", payment_max_skill_multiplier=3)),
        ('ContinuousDoubleAuction', dict(max_num_orders=5)),
        ('Gather', dict()),
        # No tax component here
    ],
    'env_layout_file': "quadrant_25x25_20each_30clump.txt",
    'fixed_four_skill_and_loc': True,
    'n_agents': 4,
    'world_size': [25, 25],
    'episode_length': 1000,
    'multi_action_mode_agents': False,
    'multi_action_mode_planner': True,
    'flatten_observations': False,
    'flatten_masks': True,
}
```

We then run multiple episodes with random agent actions (for demonstration). We measure social welfare using `coin_eq_times_productivity`, stored in the environment's `previous_episode_metrics`.

6.2 Code for Running the Free Market Simulation

```
def run_free_market_simulation(num_episodes=50):
    social_welfare_history = []
    for ep in range(num_episodes):
        obs = env.reset()
        done = {'__all__': False}

        while not done['__all__']:
            # random actions for now
            actions = sample_random_actions(env, obs)
            obs, rewards, done, info = env.step(actions)

        metrics = env.previous_episode_metrics
        swf = metrics.get('social_welfare/coin_eq_times_productivity', np.nan)
        social_welfare_history.append(swf)
        print(f"Episode {ep+1}: SW = {swf}")
    return social_welfare_history
```

Using this code, we plot social welfare vs. episode index. Figure below shows a typical run of 50 episodes with random policies. The social welfare metric remains relatively unstable due to random play.

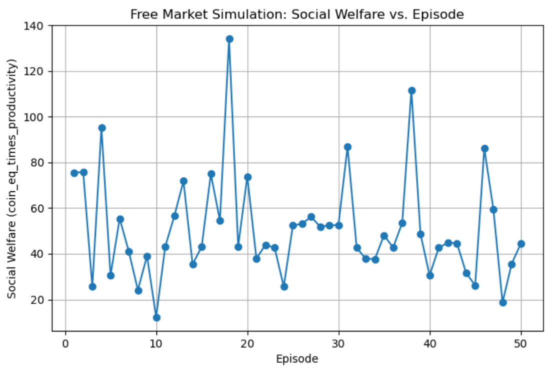

Figure 1: Free Market Simulation: Social Welfare Over 50 Episodes.

6.3 Utility vs. Skill

Additionally, we analyze how an agent's skill level (10, 15, 20, 30) interacts with labor performed to yield different utility values, using an isoelastic function. The script `plot_and_save_utility_curves` scans labor from 0 to 1000 and plots

$$u(\text{skill}, \text{labor}) = \frac{(\text{income}(\text{skill}, \text{labor}))^{1-\eta} - 1}{1-\eta} - \text{labor}.$$

Figure below shows that higher skill yields greater maximum utility at higher labor, with diminishing returns.

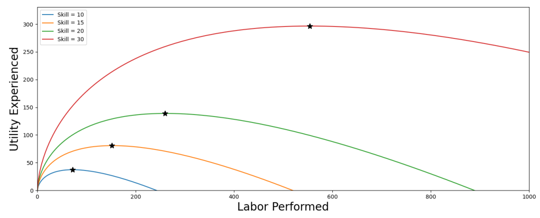

Figure 2: Isoelastic Utility Curves for Different Skill Levels. Markers indicate the labor level that maximizes the utility at each curve.

6.4 Summary of Findings

- Even in a free-market scenario, we see that higher-skill agents can outperform (e.g. building) if they invest more labor.
- Random actions are not beneficial for social welfare. This underscores the need for a trained policy or carefully guided search.

Future experiments will introduce taxes via the `PeriodicBracketTax` component and compare free-market vs. taxed outcomes.

7 Citations and References

Zheng, S., Trott, A., Srinivasa, S., Naik, N., Gruesbeck, M., Parkes, D. C., Socher, R. (2020). The AI Economist: Improving equality and productivity with AI-driven tax policies. arXiv. https://arxiv.org/abs/2004.13332

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv. https://arxiv.org/abs/1707.06347