Final Assignment:

Evolving Intrinsic Motivations For Altruistic Behaviour

Self-Organising Agent Systems

Master in Artificial Intelligence

Student: Mario Rosas

Professor: Dr. Maite Lopez-Sanchez

Academic Year: 2023-2024

May 12, 2024

Contents

Evolving Intrinsic Motivations for Altruistic Behaviours

1 | Introduction

The aim of this report is to understand and put into practice the theory behind the general functioning of Self-Organising Agent Systems, while reviewing one of the latest research findings in the field. The self-organisation paradigm is central to understanding collective behaviours in diverse social environments, where voluntary compliance of conventional norms keeps the environment functioning. By studying these phenomena, the field of self-organising multi-agent systems (SOAS) emerges as an effective tool for addressing agent coordination, cooperation and collaboration in decentralised computer systems. By embedding theories from other areas such as economics, politics and related fields into executable logical specifications of conventional rules, SOAS offers a systematic approach to designing systems in which agents self-manage in order to achieve one or more individual or common goals for the system.

Among the fundamental aspects in the study of SOAS is the concept of institutions: sets of rules that provide an algorithmic framework for fomenting sustainable, fair and legitimate interactions between agents. This factor creates links between computer science and social sciences and, at the same time, contributes to the creation of new knowledge on the development of complex systems and the modelling of social systems. As the Digital Age advances, it is important to understand how computational representations of qualitative values such as justice and democracy contribute to the stability and legitimacy of systems.

The study of cooperative behavior in multi-agent systems contains interesting challenges and opportunities within the field of artificial intelligence, particularly in the context of self-organizing agent systems. The main goal of the present project is to comprehend and validate the discoveries presented in the paper *"Evolving Intrinsic Motivations for Altruistic Behavior"*, published by Jane X. Wang and colleagues at DeepMind, in which they explore the emergence of cooperation among self-interested agents through an integration of multi-agent reinforcement learning (MARL) system and evolutionary theory. This paper addresses intertemporal social dilemmas (ISDs), where the interests of individual agents conflict with collective outcomes over varying timescales.

Intertemporal social dilemmas are the main framework for examining the complexities of agent interactions where immediate self-interest may affect long-term collective benefits, this

situations highlight the conflicts between selfish actions and altruistic outcomes overtime. The typical approach to resolving such dilemmas often relies on externally imposed strategies or pre-defined rule sets, which may not scale well or adapt dynamically as environments evolve. In contrast, the approach taken in the paper introduce evolutionary principles to evolve altruistic behaviors among agents, without the need for manual design of reward systems or strategies to solve the social dilemmas.

The paper introduces an modular architecture for MARL, where each agent has an intrinsic motivation function that evolves over time. This setup emphasizes autonomy and emergent complexity. In general, this paper contributes to our understanding of how complex cooperative behaviors can evolve in decentralized agent-based systems. This understanding is crucial for developing AI systems that are capable of sophisticated social interaction and cooperation.

2 | Paper: Methodology and Experiments

In the paper *"Evolving Intrinsic Motivations for Altruistic Behavior"*, the authors implement a complex MARL complemented with evolutionary strategies and variations of specific strategies to study the behaviour of agents in 2 different scenarios where they face an intertemporal social dilemma, and where the ideal outcome would be that agents learn to prioritise long-term social benefits instead of always going for actions that provide immediate rewards. Details of this work are presented below.

2.1 | Agent Architecture

The system employs a modular architecture for the agents involved in the simulation. Each agent is composed of two primary neural network modules:

- **Policy Network.** Responsible for deciding the actions of the agents based on the state of the environment and its own internal state. It operates on a faster timescale, adjusting to immediate circumstances in the environment through reinforcement learning.
- **Reward Network.** Modulates the intrinsic motivation of the agent. It operates on a slower evolutionary timescale, where the network's parameters evolve across generations of agents. The intrinsic reward function is formulated as an additional term in the reward structure of each agent, influencing decisions by integrating social preferences.

As there are two components computing the rewards, then the total reward is composed by them in the following way:

- **Total Reward:** The combination of *extrinsic* and *intrinsic* rewards, guiding the overall agent behaviour

$$r_i(s_i, a_i) = r_i^E(s_i, a_i) + u_i(f_i)$$

  - *Extrinsic Reward:* Feedback given by the environment for actions taken, reflecting immediate, tangible benefits.

$$r_i^E(s_i, a_i)$$

  - *Intrinsic Reward:* Additional rewards based on social features or the collective welfare, to promote cooperative behaviors.

$$u(f)$$

The computation of the intrinsic rewards are proposed to be computed using 2 different strategies, which are tested to see the pros and cons of each of them, these stategies are:

- **Prospective Method.** Intrinsic rewards are calculated based on expectations of future rewards, aiming to influence immediate decisions for long-term benefits.
- **Retrospective Method.** Intrinsic rewards are based on past actions' outcomes, focusing on historically received rewards, to encourage behaviors beneficial in similar future contexts

2.2 | Multi-Level Evolution

The system distinguishes between two timescales of optimization:

- **Learning Timescale.** Individual agents adjust their policy networks based on both extrinsic environmental rewards and intrinsic rewards shaped by the reward network.
- **Evolutionary Timescale.** The parameters of the reward networks are evolved based on the performance of the agents, specifically how well they contribute to the collective outcomes of their group.

2.3 | Training and Evolution Process

The training and evolution process involves the following steps:

- **Population-Based Training.** The system uses a population of agents where policy and reward networks are periodically sampled and reassigned to different agents, facilitating a diverse exploration of strategies and behaviors.

- **Matchmaking and Social Dynamics.** Agents are matched in each episode based on their past performance and cooperative behavior. This assortative matchmaking helps in forming groups where cooperative tendencies are either rewarded or penalized based on the collective outcomes, thus aligning individual incentives with group success.
  - *Random vs Assortative Matchmaking.* Agents can be matched in two different ways: uniformly at random or based on their level of cooperativeness.
- **Shared vs. Individual Reward Networks.** Exploration of different configurations where reward networks are either shared among all agents in an episode or assigned individually. This distinction tests how collective versus individual reward structures impact the evolution of cooperative behavior.

2.4 | Experiment Environments

The paper presents two main experimental environments, both structured as intertemporal social dilemmas within a MARL setting. These environments are designed to test the cooperative behaviors of agents. Here are the details of these environments

- **Cleanup Game.**
  - *Objective.* Agents must manage the cleanliness of a aquifer to ensure the continuous spawning of apples in a separate field. Apples are the primary reward source in this environment.
  - *Mechanics.* Apples spawn at a rate inversely proportional to the pollution level in the aquifer. Over time, pollution accumulates, reducing the apple spawn rate unless agents clean the aquifer.
  - *Dilemma.* Cleaning the aquifer does not provide direct rewards to the agents. Thus, agents face a choice between harvesting apples for immediate rewards and cleaning to ensure long-term apple availability for all. If all agents decide not no clean, the apple spawn rate declines to zero, hurting collective rewards.
- **Harvest Game.**
  - *Objective.* Agents collect apples from a shared environment where the growth rate of apples depends on the local density of remaining apples.
  - *Mechanics.* Apples regrow more quickly in areas with more surrounding apples. If too many apples are harvested too quickly from a specific area, the local apple population can be depleted, leading to longer-term scarcity.

– *Dilemma.* The temptation for agents to harvest apples aggressively for immediate gain versus the need to harvest them sustainably to maintain overall environmental productivity.

3 | Experimental Setup - Design

Given the limitations of both time and computing resources, the application approach aims to provide an overview of the hypotheses and conclusions outlined by the authors by establishing a limited framework and simplifications in the overall system design. Consequently, only a subset of variables was taken into account in both the implementation and experimental setup. Specifically, we focused on the Cleanup game and the exploration of the shared retrospective network reward strategy under both random and assortative matchmaking conditions. This selection was based on the authors' findings indicating superior performance in all experiments conducted in the referenced article.

3.1 | Implementation Consideratins

3.1.1 | Initialization of Policy and Reward Network Parameters

- **Policy Network:** Initialize each agent's policy network with random weights, which will adapt through reinforcement learning and evolutionary processes.
- **Reward Network:** Begin with random parameters for the reward network, set to evolve across agent generations, optimizing for cooperative behaviors.

3.1.2 | Learning Parameters

- **Mutation Rates:** Set initial mutation rates to explore parameter space effectively and allow adaptive evolution of strategies.
- **Selection Process:** Employ a fitness-proportionate selection or tournament selection strategy to evolve successful agents and reward parameters.
- **Learning Rates:** Optimize learning rates to ensure effective convergence and the ability to escape from local optima.

3.1.3 | Experimental Conditions

- **Shared Retrospective Reward Network + Random Matchmaking:** Agents share a reward network and are paired randomly, serving as a baseline for cooperation analysis.

3.2 | Performance Evaluation

- **Group Performance:** Measure total collective rewards to assess the efficacy of cooperative strategies.
- **Cooperation vs. Defection Rates:** Analyze behaviors to determine rates of cooperative versus selfish actions.
- **Long-term Sustainability:** Evaluate strategies effects on environmental sustainability and resource depletion.

3.3 | Expected Challenges

- **Complexity of Evolutionary Algorithms:** Handling the computational demands and complexity of evolutionary algorithms with large populations.
- **Balancing Exploration and Exploitation:** Fine-tuning mutation rates and selection pressures to optimize strategy exploration without reducing diversity.
- **Assortative Matching Efficacy:** Ensuring that assortative matchmaking leads to effective promotion of cooperation without reducing system diversity.
- **Parameter Tuning:** Determining optimal settings for learning rates, network architectures, and hyperparameters through extensive experimentation.
- **Measurement of Social Behaviors:** Accurately measuring and interpreting complex social interactions and emergent behaviors.

4 | Resources and Implementation Details

5 | Resources and Implementation Initial Guidelines

5.1 | Experimental Environment

- **Sequential Social Dilemma Games:** Check if is feasible to use the existing open-source repository by Eugene Vinitsky, which implements the Cleanup and other related environments suitable for the planned experiments. In case it not feasible to use it, implement a custom environment.

5.2 | Tools and Libraries

- **Python:** The primary programming language for implementation due to its broad support and wide use in the scientific computing and machine learning communities. Packages of this language are tailored to the implementation needs of the system, especially for ML algorithms and evolutionary strategies.
- **OpenAI Gym:** Use OpenAI Gym to create and manage the simulation environments. This library offers a standardized API and a variety of tools that facilitate the development and comparison of reinforcement learning algorithms.

5.3 | Alternative Plans

- **Plan B - Manual Configurations of Social Norms:** Should the evolutionary strategies fail to yield significant insights or results, I will explore manually configuring intrinsic rewards based on predefined social norms. This approach will allow us to observe how the manipulation of these norms influences altruistic behaviors over time, providing a comparative basis against the evolved strategies.

5.4 | Implementation Strategy

Detailed steps for setting up and running experiments include:

1. **Initial Setup:** Download and configure the Sequential Social Dilemma Games environment from the specified repository. Set up Python and OpenAI Gym on the computational platform.
2. **Parameter Initialization:** Define initial parameters for the policy and reward networks, learning rates, and evolutionary processes as per the experimental design.
3. **Running Simulations:** Execute simulations under different matchmaking conditions (random and assortative) to test the effectiveness of shared retrospective reward networks in promoting cooperation.
4. **Data Collection and Analysis:** Collect data on agent performance, cooperation rates, and environmental sustainability. Use statistical and machine learning tools to analyze the results and draw conclusions about the effectiveness of different strategies.
5. **Adjustments and Iterations:** Based on preliminary results, adjust parameters and strategies as needed to optimize outcomes and explore additional configurations if necessary.

5.5 | Final implementation

After a careful evaluation of several tools suitable for the implementation phase, a final decision was made to adopt a Python-based system. Python's versatility, its broad support in the scientific computing and reinforcement learning communities as it has a strong ecosystem of libraries made it the optimal choice for the requirements of this implementation. Within the implementation, three fundamental files form the backbone of the system, each playing a distinct but interconnected role. The following is a comprehensive breakdown of these files, clarifying their functionalities and contributions to the overall system architecture. It is important to mentio that in this final implementation it was not the best option to use the pre built environment implemented by Eugene Vinitsky, as there where constructed in a old version of an specific library, which was not prepared to connect with the more recent tools and packages that were used to build the agent architecture and training process. So a custom environment was created on top of the PyGame package.

- **environment.py**: Defines a custom environment within the gymnasium framework. It serves as the backbone for creating a reinforcement learning environment specifically designed for studying cooperative behaviors and decision-making strategies within a simulated Cleanup game scenario. The *CleanupEnv* class inherits from the *gym.Env* class, which facilitates the creation and management of customizable environments for RL experiments.

  - The *init method* initializes various attributes of the environment, including the action space (representing possible actions agents can take), the observation space (representing the environment's state), and the grid size defining the spatial dimensions of the game world. Additionally, it sets up the initial configuration of agents, apples, and waste within the environment, along with initializing the graphical interface using the pygame library.
  - The *reset method* resets the environment to its initial state, randomly placing agents within the grid and initializing apples and waste. It also determines the initial distribution of waste within the aquifer region, simulating varying levels of cleanliness.
  - The step method advances the environment by one time step, executing actions chosen by agents and updating the environment accordingly. It computes rewards based on agents' actions, updates their positions, collects apples, manages waste, and checks for episode termination conditions.
  - The *_update_grid*, *_spawn_apples*, and *_spawn_waste* methods handle the manipulation of the environment grid, spawning of apples and waste, and updating their

positions based on the current state of the environment and predefined spawning rates.

- The *_get_observation method* generates observations for each agent based on their current positions and the surrounding environment, providing input for agents' decision-making processes.
- The *render method* renders the current state of the environment graphically using the pygame library, displaying agents, apples, and waste within the grid.

- **agent_architecture**: defines a neural network architecture named "AgentNetwork" implemented using the PyTorch framework, designed to facilitate learning and decision-making processes for agents within a reinforcement learning environment.
  - The *init mehod* initializes the AgentNetwork module, setting up its components such as the visual encoder, LSTM, value heads, policy head, and intrinsic reward network. Parameters include the input shape and the number of actions.
  - The *forward method* handles the forward pass of data through the neural network. It takes input observations, the agent's previous action and rewards, as well as hidden and cell states of the LSTM. Processes involve visual encoding, combining input with previous action and rewards, passing through the LSTM, and computing policy probabilities, extrinsic, and intrinsic values.
  - The *get_intrinsic_reward method* computes intrinsic rewards based on input shared information from all the agents. It takes input, and then utilizes the intrinsic reward network to calculate the reward. This involves transforming features into a hidden representation through a learned linear layer with ReLU activation, followed by a dot product operation with a weight vector.

- *training.py*: Defines the training process for a population of agents within a reinforcement learning environment designed for the Cleanup game scenario.
  - First step is to import the libraries and environment. The environment is initialized, and various training parameters such as the number of episodes, steps per episode, learning rate, and discount factor are defined.
  - Second step, the population of agents is initialized, each equipped with an AgentNetwork architecture. The RMSprop optimizer is employed for each agent to optimize their neural network parameters.
  - Third step, several utility functions are defined, including random matchmaking for selecting agents for each episode, computing loss based on predicted and actual

rewards, computing fitnesses for the evolutionary process, and evolving the population through mutation and selection.

- Fourth step the training loop iterates through episodes, with each episode comprising several steps. During each step, agents select actions based on observations from the environment, interact with the environment to receive rewards, and update their neural network parameters using backpropagation. Additionally, an evolutionary process is employed periodically to evolve the population based on agents' performance. Fitnesses are computed, and agents are selected for reproduction and mutation based on their fitness scores.
- Fifth step, the trained agent models' parameters are saved to disk for future use and evaluation.

Additionally two secondary scripts are provided to execute and visualize the simulation process.

- **simulation.py**: script with random selection process of actions to test the correct functioning of the environment and serves as a baseline for the behavior of the agents.
- **simulation_trained.py**: script that loads previously trained agent architectures and runs an episode simulation to analyze agent behavior when using trained parameters.

6 | Experimental Results & Analysis

Due to the time requirements and complexity of the implementations, not all the experiments initially proposed were carried out. According to the results of the implementation, several runs of the simulation were made adjusting the parameters to verify that the agents were learning, first to collect apples according to the reward that this action gives them, then another round was made also making a fine tuning of the parameters to find a combination that allows the model to make the system intrinsic reward is significant in the decision making of the actions performed by the agents.

The experimental results demonstrate a significant difference in agent behavior when intrinsic motivation rewards are included versus when they are excluded. This variation in behavior shows the importance of intrinsic motivation in shaping agent actions within the Cleanup game environment.

When intrinsic motivation rewards were included, example ilustrated in the figure 1.1, agents exhibited a increased tendency to clean the waste at the aquifer. This behavior sug-

gests that intrinsic rewards effectively motivated agents to engage in environmentally beneficial actions beyond their immediate, extrinsic goals. Specifically, the agents were observed to prioritize cleaning the aquifer over merely collecting apples, indicating a broader consideration of long-term environmental health and sustainability.

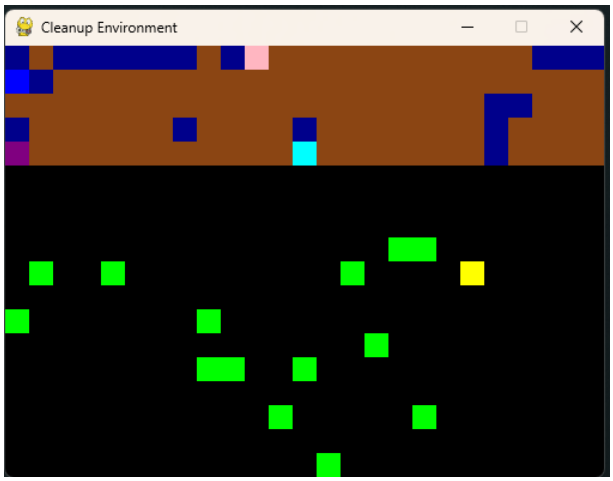

Figure 1.1: No use of intrinsic motivation reward

In contrast, as shown in the example of figure 1.2 in the absence of intrinsic motivation rewards, agents predominantly focused on the direct objective of collecting apples. This shift in behavior highlights the agents' tendency to favor immediate, extrinsic rewards over actions that contribute to the overall well-being of the environment. The lack of intrinsic motivation led to an approach, where agents were less inclined to engage in cleaning activities that did not provide immediate extrinsic benefits.

Figure 1.2: No use of intrinsic motivation reward

The inclusion of intrinsic motivation thus plays then an important role in fostering cooperative and altruistic behaviors among agents. By balancing intrinsic and extrinsic rewards, the agents were encouraged to act in ways that benefit both themselves and the collective environment. This dual-motivation system highlights the potential for designing more sophisticated and effective agent behaviors through the integration of intrinsic rewards.

These results can be taken as a indicator that the use of RL and evolutionary strategies to incorporate intrinsic motivation mechanisms in the development of self-organizing multi-agent systems is a good methodology. By doing so, it is possible to create agents that not only pursue individual goals but also contribute to the collective good, leading to more sustainable and cooperative outcomes in complex environments.

7 | Conclusions

Through the overall analysis of this work it is important to mention that positive results were obtained in the implementation phase. Despite working with a simplified version of the system, the results demonstrate the effectiveness of providing agents with information about their overall performance in the environment in conjunction with information about the performance of the other agents in the system to mitigate self-centred behaviour. This approach helps agents to effectively address the difficulties introduced by intertemporal social dilemma scenarios in achieving a common welfare state, thus showing the potential of integrating social norms automatically into self-organising agent systems through the use of reinforcement learning (RL) systems combined with evolutionary strategies. In particular, the evolutionary features used by these strategies allow agents to dynamically adapt to the proposed scenario, fostering innovative techniques for dealing with complex social dilemmas.

In addition, the development of this project contributed significantly to the positive results in understanding and learning the field of study derived from the process followed throughout the life cycle of this project. Starting with the review of the original scientific article and related literature, followed by the planning, execution and analysis during the implementation and experimentation phases. The main objective of improving the understanding of Self-Organising Multi-Agent Systems (SOAS) methodologies was successfully achieved. This iterative process not only facilitated the practical application of theoretical concepts, but also enhanced a deeper understanding of the dynamics inherent in self-organising systems.