CM3020 Artificial Intelligence
Midterm Coursework

Chapter 1
Project Overview

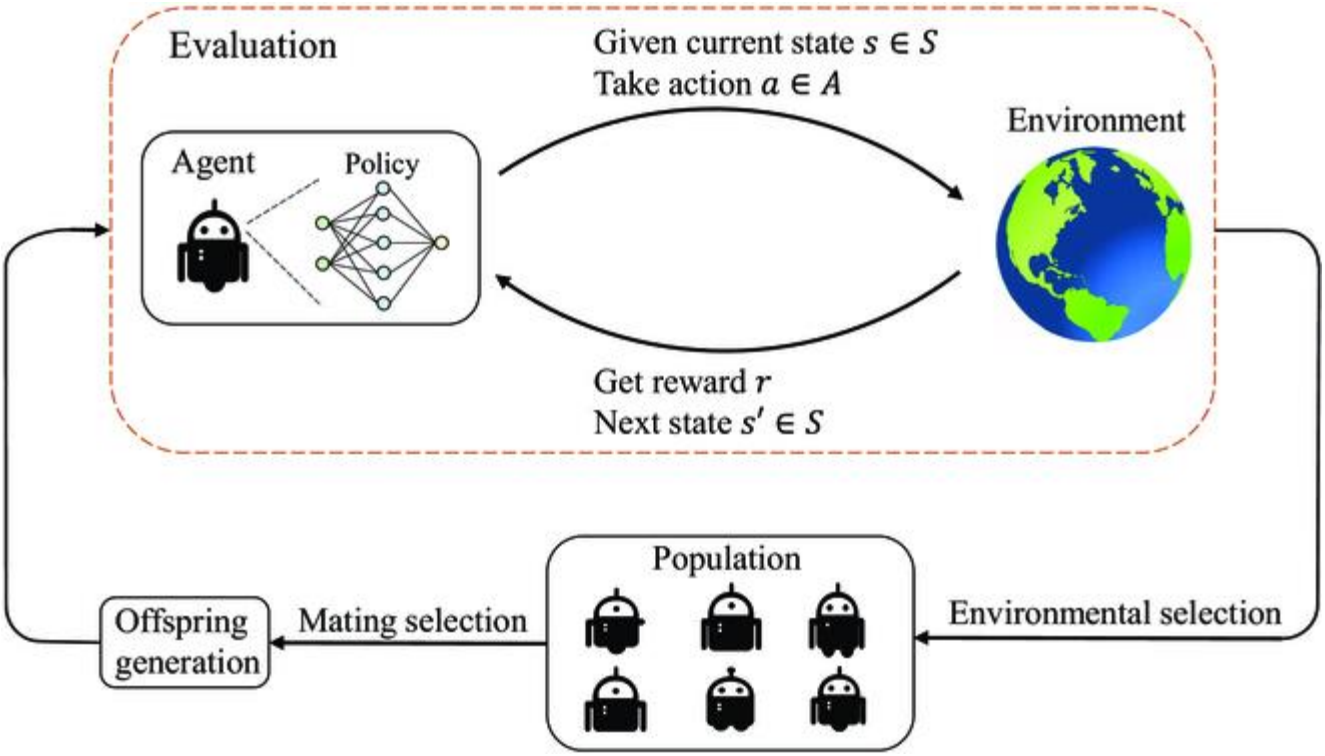
Understanding Reinforcement Learning and Its Uses

Reinforcement learning (RL) has made significant strides in recent years, particularly in solving intricate challenges found in gaming scenarios that demand adaptive decision-making. Video games provide ideal platforms for testing RL algorithms because they offer structured rules, consistent environments, and quantifiable goals. The field has progressed from handling basic tasks, like Atari games, to conquering far more advanced challenges, such as Dota 2 and StarCraft II. These achievements highlight RL's growing capacity to compete with—and even outperform—human players in complex, interactive settings.Brief Explanation of Evolutionary Reinforcement Learning and its Significance

Evolutionary Reinforcement Learning (ERL) is an advanced hybrid approach that merges the advantages of evolutionary algorithms (EAs) with conventional reinforcement learning (RL) techniques to tackle complex decision-making problems. While standard RL is effective at developing policies through environment interaction and gradient-based updates, it often faces limitations like sparse feedback and getting stuck in local optima. In contrast, EAs utilize population-driven search methods to explore the solution landscape more extensively, making them particularly effective in unpredictable or non-differentiable environments.

ERL leverages this complementary relationship by employing EAs to explore and optimize strategy parameters across a population, while RL entitys refine these policies through focused, gradient-based learning. This integration enables a balanced approach to exploration and exploitation. A major advantage of ERL is its effectiveness in handling problems that pose difficulties for standard RL, including high-dimensional, multi-modal, and non-stationary environments.

By utilizing the diverse strategy candidates generated by evolutionary processes, ERL promotes more resilient learning and improved generalization across varied tasks. As a result, ERL has proven particularly useful in areas like robotics, game AI, and real-world applications where uncertainty and complexity are prevalent. This hybrid paradigm is advancing the development of intelligent systems capable of solving challenges beyond the reach of standalone RL or EA techniques.



Summary of the Mountain Climb Environment and Its Difficulties

The Mountain Climb Environment is a specialized simulation designed to evaluate the effectiveness of cutting-edge reinforcement learning techniques integrated with evolutionary algorithms. It serves as a testbed to challenge and measure the capabilities of these hybrid approaches in a controlled, yet demanding, setting.

Why combine Reinforcement Learning with Evolutionary Algorithms?

Combining Reinforcement Learning (RL) with Evolutionary Algorithms (EAs) brings together the complementary strengths of both approaches, resulting in more effective and robust learning, especially for complex and challenging environments. Here's why this combination is valuable:

- EAs are excellent explorers. They maintain a diverse population of policies, encouraging broad exploration of the solution space.
- RL excels at exploitation, fine-tuning policies using gradient-based learning from environment feedback.
- RL methods often get trapped in local optima, especially in environments with sparse or deceptive rewards.
- EAs use stochastic mutation and crossover, making them less susceptible to local minima and more likely to discover better global solutions.
- RL requires differentiable environments to compute gradients.
- EAs work with non-differentiable, noisy, or black-box environments.

Combining both allows for broader exploration (via EAs) and precise optimization (via RL), leading to faster and more reliable convergence.

Project Goals

The goal of this project is to develop an agent capable of ascending a mountain to the greatest height achievable, utilizing an Evolutionary Reinforcement Learning (ERL) approach. This involves optimizing various parameters that influence the entity's climbing ability and overall performance.

By merging evolutionary computation with strategy-based learning, we bypass the need for traditional gradient descent and backpropagation techniques. Instead, a population of entitys is evolved using evolutionary strategies. Performance of every entity is evaluated through a fitness function, and the best-performing entity of the generation is selected. The reinforcement learning algorithm then fine-tunes its strategy using this top entity's parameters, resulting in a more efficient, adaptable, and robust training process.

Projected Achievements

This project primarily focuses on integrating evolutionary methods with reinforcement learning algorithms, with an emphasis on the Proximal Policy Optimization (PPO) technique. The training process begins with a diverse population of entitys, each equipped with its own neural network and a unique strategy for interacting with the environment. As these entitys engage with the environment, they receive rewards that reflect their performance.

Based on these rewards, each entity's fitness is evaluated, and the top performers are selected. These high-performing entitys are then used to develop the succeeding iteration through mutation, enabling the population to evolve and improve over time. Through repeated iterations of selection, mutation, and strategy refinement, the entitys gradually enhance their ability to perform the task.

The ultimate objective is for the entity to ascend the mountain as high as possible, showcasing its capacity to learn and adapt through this hybrid evolutionary-reinforcement learning framework.

Reason for Selecting Evolutionary Reinforcement Learning

This project aims to identify the best set of parameters for the agent to maximize its performance—specifically, reaching the highest point possible. The process begins by generating a population of agents with randomly assigned attributes. A fitness function is then used to evaluate their effectiveness and select the top-performing agents to progress into the next generation.

Once the optimal parameters are identified, the next step is to train the selected entity effectively to achieve the objective. Evolutionary algorithms are used to discover the best-performing entity, and then reinforcement learning is applied to train its strategy based on the parameters of that top entity. This combined approach ensures that the entity is not only well-optimized but also well-trained to perform its task successfully.

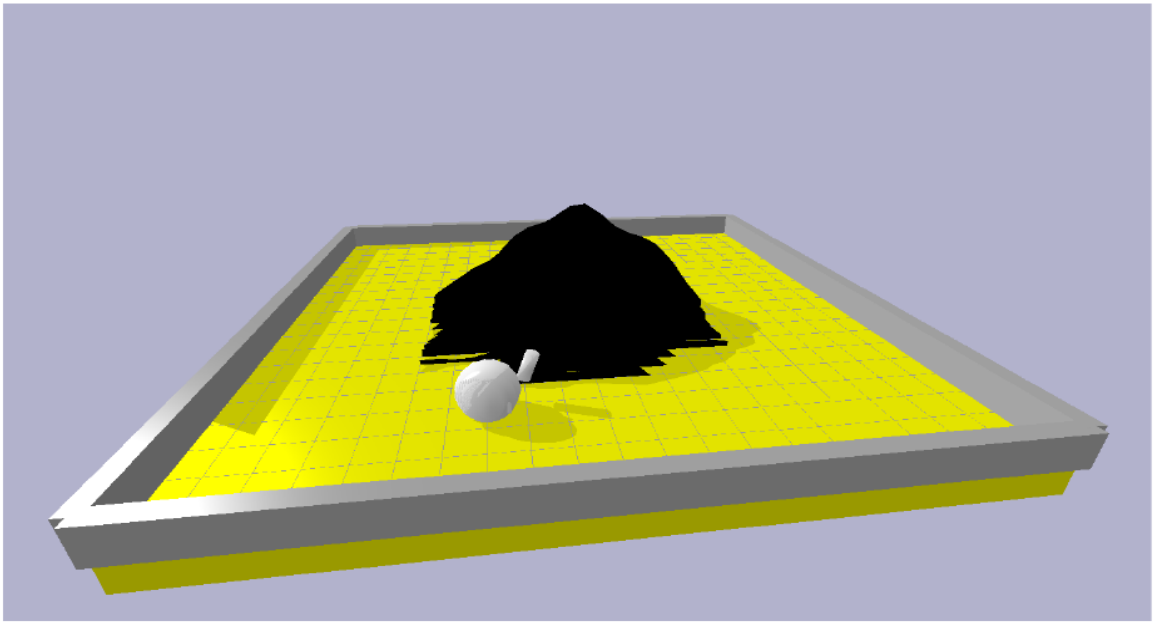
Coverage and Extent Table:

Comparison Criteria	Reinforcement Learning (RL)	Evolutionary Algorithm (EA)	Hybrid Evolutionary Reinforcement Learning Approach
Exploration	Limited: Biased by rewards and gradients	Global: Diversity driven exploration	Combines Diversity and efficiency
Exploitation	Strong: Focuses on local policy optimization	Weak: relies on population level refinement	RL fine tunes EA discovered solutions
Adaptability	Adapts locally but can overfit	Retains diversity across generations	Balances local adaption with global search
Optimization Landscape	Works best in smooth landscape	Excels in rugged, high dimensional spaces	Combines strength for diverse landscape
Optimization Strategy	Uses Gradient Descent to Reduce the Loss Function	Assesses Fitness Scores to Identify Top Performers	Selects the Optimal Agent via EA and Refines Strategy with RL
Catastrophic Failures	High Risk during Exploration	Low risk due to diverse population	Reduced risk, fallback from EA to RL
Sparse Rewards	Struggles without shaping	Excels; independent of reward density	EA handles exploration, while RL focuses on exploitation once rewards become available.

Chapter 2
Challenge Analysis

Summary of the Climbing Simulation

The Mountain Climb Environment is a tailor-made simulation developed to evaluate the performance of our Evolutionary Reinforcement Learning algorithm. It follows the design principles of the OpenAI Gym framework.



Simulation Dynamics & Key Issues

Within the Mountain Climb Environment, the agent—referred to here as Mario—navigates a side-scrolling landscape filled with obstacles such as pits, adversaries, and structural challenges. The key goals are:

- Achieve the greatest possible vertical ascent.
- Begin climbing the mountain as early in the episode as possible.

Entity and Simulation World

The Mountain Climb Environment features an entity, which is essentially a randomly generated creature composed of multiple links connected by joints. The main objective is to evolve this

creature over time to improve its ability to climb. Several physical parameters of the entity can be evolved, including:

- Segment lengths
- Segment geometries
- Segment thicknesses
- Total number of segments
- Speed of motorized joints

These configurable attributes are encapsulated within the Genome class, which plays a central role in representing the entity's structure and behavior.

Encoding of State and Action

The agent's objective is to monitor its position relative to the mountain's center, which represents the peak it can potentially reach. The state space is defined by the agent's current location within the environment.

State Variable	Description	Range	Shape
Position of the Agent	The horizontal position of the cart (agent)	$-4.8, 4.8$ (default)	1

The agent operates using a velocity-based control scheme, specifically a continuous velocity control mode.

Parameter	Description	Value
Motor Control Mode	Mode of controlling the motors (e.g., velocity, position)	Velocity
Action Space Dimension	Number of motors controlled by the agent	Equal to the number of motors

Challenge Definition

The agent (as shown in the figure) to successfully ascend to the mountain's peak, it must possess a suitable structure, effective movement dynamics, precise motor control, and optimized segment

dimensions. As a result, all physical characteristics that impact its climbing capability are treated as evolvable traits.

These parameters are evolved over multiple episodes. Each episode consists of 10,000 timesteps. If the entity fails to reach the top within these 10,000 timesteps, the episode is marked as a failure, and a new episode begins with updated parameters. The entity's overall performance is evaluated across multiple episodes using a fitness function. This performance score is determined as the average reward the entity receives per episode, where the reward is based on the entity's distance from the target position—the higher the climb, the greater the reward.

Optimization Goal

Our objective is to generate a population of agents, each engaging with the environment to accumulate rewards. These rewards are then used to evaluate the fitness of each individual agent, since the ultimate aim of an entity is to maximize its reward. The entitys that achieve the highest rewards are considered the fittest.

From the current generation, we select these fittest entitys and use them to produce offspring for the next generation. By repeatedly applying this evolutionary process, we aim to discover the physical parameters that lead to the highest possible rewards, effectively optimizing the entitys' performance over time.

Chapter 3
Analytical Framework

Performance Scoring

The entity's performance score is determined over multiple episodes based on its distance from the target position, which is the top of the mountain. Once all entitys in the population have been evaluated over these episodes, we compute the average reward to assess the overall performance.

After evaluating the current generation, a new generation is created through mutation and crossover of the fittest entitys from the previous generation. This process allows the parameters to evolve and improve over time.

$$r(s,a) = -\sqrt{\left\{\left(x_{\{\{agent\}\}} - x_{\{\{target\}\}}\right)^2 + \left(y_{\{\{agent\}\}} - y_{\{\{target\}\}}\right)^2\right\}}$$

Assessment of Fitness

The fitness of each strategy is calculated as:

$$F\left(\pi_{\{\theta_i\}}\right) = J\left(\pi_{\{\theta_i\}}\right) = \{E\}_{\left\{\tau\sim\backslash pi_{\{\theta_i\}}\right\}\left[\sum_{\{t=0\}}^{\{T\}}\gamma^t r(s_t,a_t)\right]}.$$

Selection

The policies are selected based on their fitness score.

$$P\left(\pi_{\{\theta_i\}}\right) = \left\{F\left(\pi_{\{\theta_i\}}\right)\right\}\left\{\sum_{\{j=1\}}^{\{N\}F\left(\pi_{\{\theta_j\}}\right)}\right\}.$$

Mutation

$$\theta_i' = \theta_i + \{N\}(0,\sigma^2),$$

Crossover

$$\theta_{child} = \alpha\theta_a + (1-\alpha)\theta_b,$$

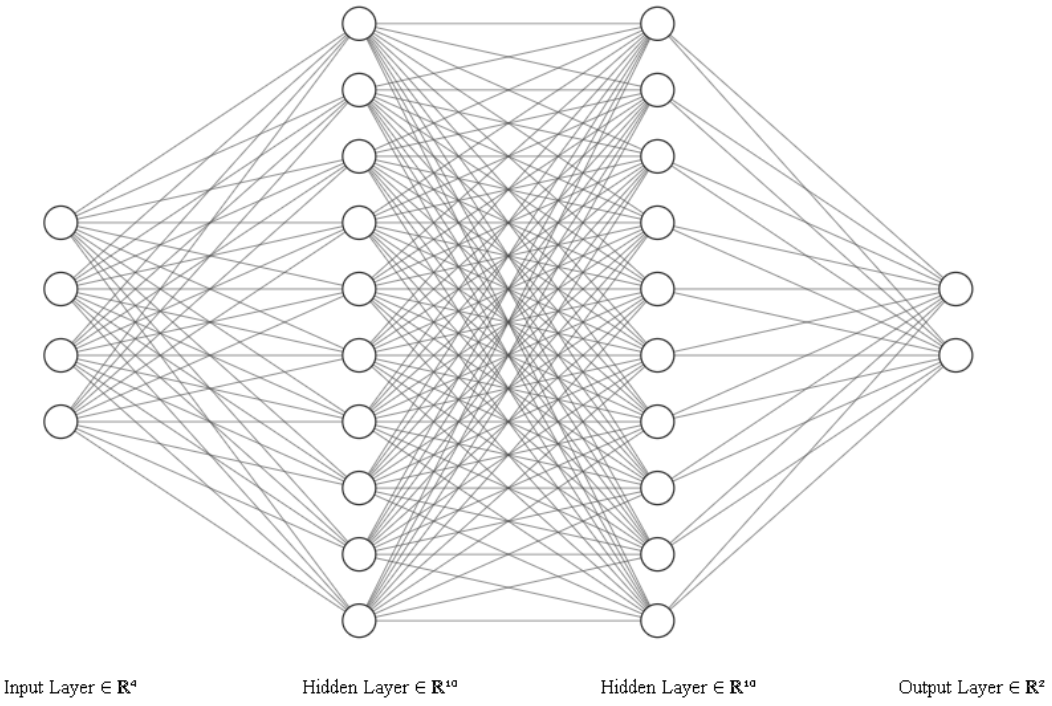
Chapter 4

Structural Design of the

Neural Strategy Modeling

The agent population is composed of multiple individuals, each driven by a distinct policy encoded within a neural network. The parameters of these networks are fine-tuned according to the outcomes of the fitness evaluation, which reflects each entity's performance.

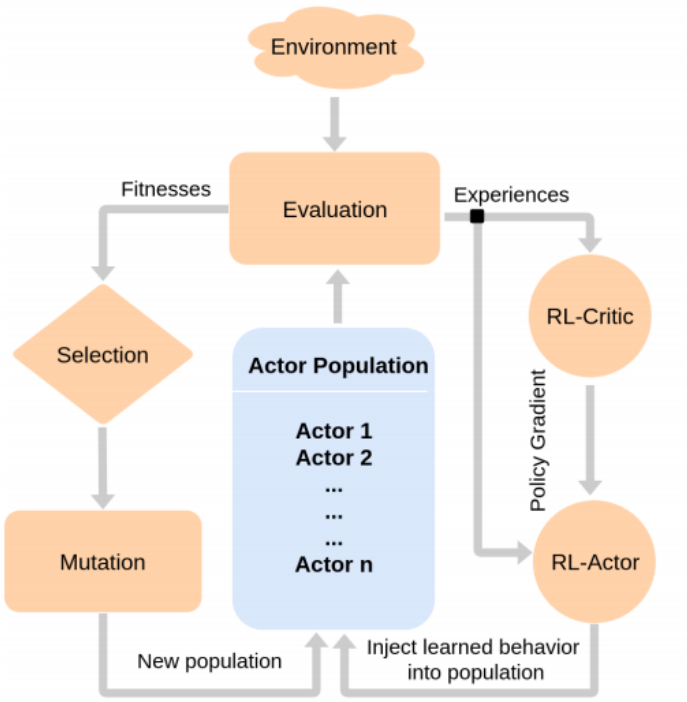
The structure of the strategy network is described as follows.



The input layer of the neural network receives the entity's current state (such as its position) and processes it to produce an output, which corresponds to motor velocities. Based on the network's output, the agent executes an action and receives corresponding feedback from the environment in the form of a reward. The fitness of each agent is then computed, followed by an evaluation of the

entire population. After this assessment, a new generation is formed by applying mutation and crossover to the top-performing agents from the current group.

End-to-End System Blueprint



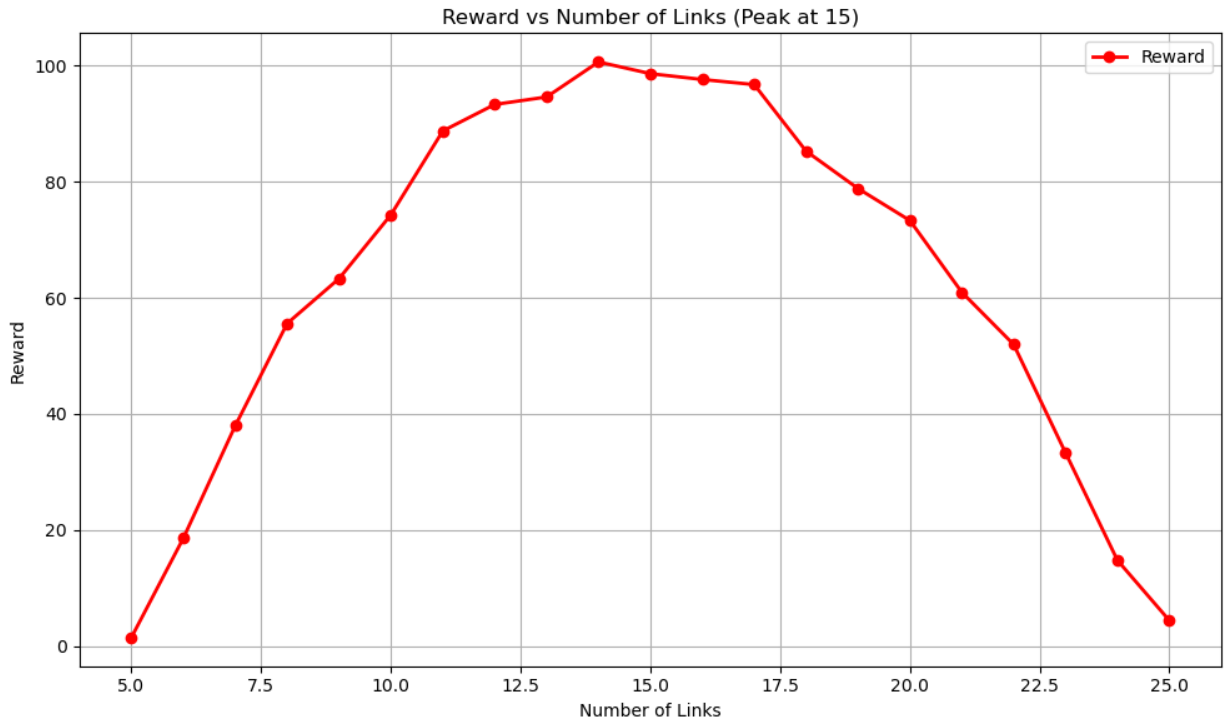
Chapter 5
Outcome Metrics and Interpretation

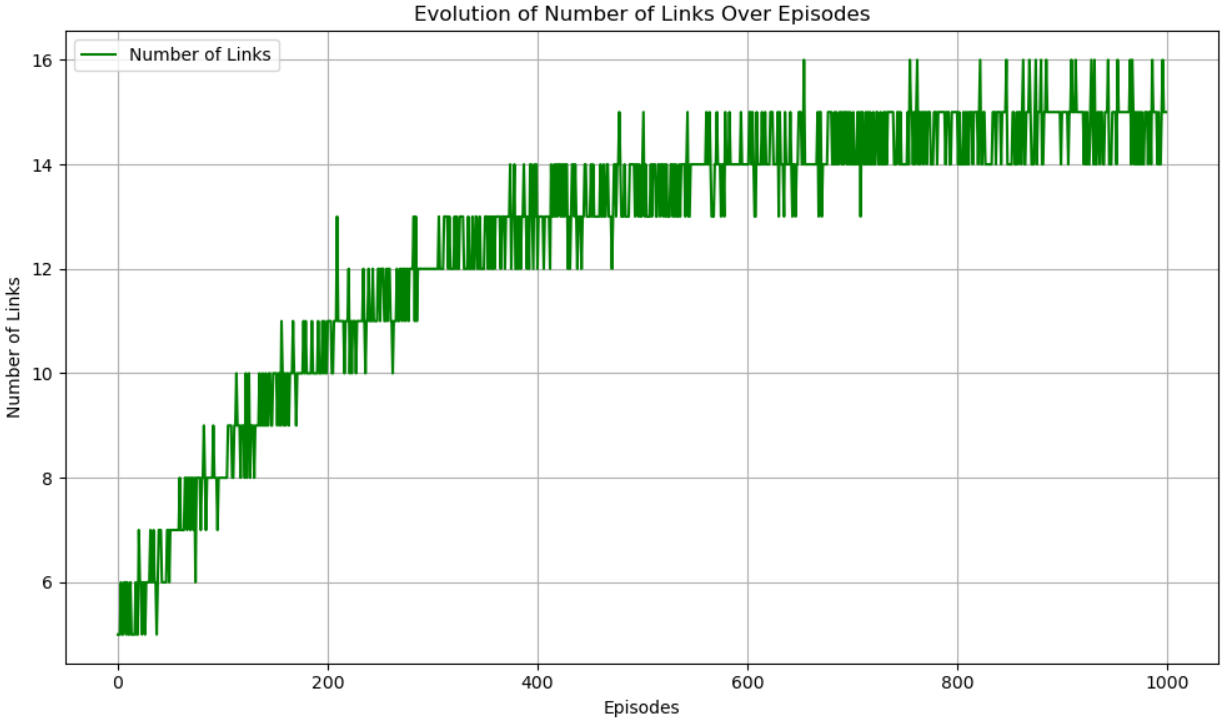
Evaluating Agent Performance Within the Population

The number of links is among the adaptive parameters, each agent is assigned a randomly determined link count. The fitness score is then computed in relation to this structural attribute.

Top Performer Incentive Measurement

The reward for the fittest entity is calculated based on its progress toward the target, evaluated through altitude gained. This reward guides the selection process, helping to identify entitys with the best performance.


Reward vs Number of Links (Peak at 15)
Reward
Reward
Number of Links
0
20
40
60
80
100
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0


Evolution of Number of Links Over Episodes
Number of Links
Number of Links
Episodes
6
8
10
12
14
16
0
200
400
600
800
1000