NEUROPLASTICITY IN GENETIC PROGRAMMING AGENTS FOR ADAPTIVE AND CONTINUAL DECISION MAKING

NEUROPLASTICITY IN GENETIC PROGRAMMING AGENTS FOR ADAPTIVE AND CONTINUAL DECISION MAKING

By ALI NAQVI, BSc

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the Requirements for the Degree Master of Science

McMaster University

MASTER OF SCIENCE (2025)

Hamilton, Ontario, Canada (Computing and Software)

TITLE:	Neuroplasticity in Genetic Programming Agents for Adaptive and Continual Decision Making
AUTHOR:	Ali Naqvi BSc (Computer Engineering), McMaster University, Hamilton, Ontario, Canada
SUPERVISOR:	Dr. Stephen Kelly
NUMBER OF PAGES:	xiii, 61

Abstract

Dynamically decomposing complex tasks into reusable sub-policies remains a core challenge in Reinforcement Learning. Tangled Program Graphs, a genetic-programming framework for general-purpose machine learning (applied here to reinforcement learning), addresses this by evolving connections between different agents in order to break down complex problems into manageable sub-problems.

Inspired by memetic algorithms, which accelerate evolutionary search through agentic refinement, we introduce Neuro-Tangled Program Graphs. This biologically grounded extension utilizes hierarchical plasticity within the structure of an agent, applying a homeostatic rule at the initial decision edges and a competitive Oja-style update in each subsequent decision edge.

Evaluated on both a static and dynamic variant of the MuJoCo Ant environment, this approach yields higher peak returns and evolves with 59-88% fewer mean effective instructions used per step, demonstrating stronger performance and a more compact search.

Next, we add an TD-style online value baseline and eligibility traces to stabilize and distribute dense step-wise rewards over time, sharpening temporal updates

within each agent. We then examine how trace length and a per-team plasticity decay factor shape learning dynamics. To set these, we compare end-to-end evolutionary tuning with MAP-Elites using a multi-archive that explores (trace length x decay).

The benefits of reward modulation are then tested for with TPG and NeuroTPG variants on a customized static and dynamic maze environment. This addition show a consistently better performance across all seeds and also a more interpretable final structure.

Overall, our findings highlight the vital role of a local search within population search algorithms. Our studies hope to open a new avenue to gradient-free memetic algorithms which offer many benefits and opportunities from various already developed field of studies.

Acknowledgements

"Think where man's glory most begins and ends, and say my glory was I had such friends."

— W.B. Yeats

I would like to express my gratitude to the following people:

- My supervisor, Dr. Stephen Kelly, for his patience, guidance, and invaluable advice throughout my graduate studies.
- My classmates and the many others who made this journey enjoyable.
- My close friends, whose constant support and encouragement would merit a thesis of their own.
- My siblings, Hiba, Batul, and Wasay, for always being there through the ups and downs—even if that meant bothering me exactly when I needed it.
- And finally, my parents, whose countless sacrifices have made all of this possible. This thesis is dedicated to you, Bushra and Shabab.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Statement of Problem

Adaptive agents in non-stationary environments must continually integrate new experience without costly retraining [31]. Biology achieves such rapid, within-lifetime adjustment through neuroplasticity: synapses and circuits change their influence as a function of recent activity and feedback. A parallel goal in artificial life is to provide learning systems with similar within-trial adaptability such that behaviour can track shifting contingencies.

Tangled Program Graphs (TPGs) are an evolutionary control framework that encode behaviour as a directed graph of teams (decision nodes) connected by edges labeled with programs (<state,action> value functions). At run time, the agent starts at the root team and executes each outgoing edge's program. Each program, implemented here as a linear genetic program [5]—a simple register machine—produces

a scalar bid (for routing) and a candidate action, stored in its action register. The edge with the highest bid is chosen (winner-takes-all), and this routing repeats until a leaf (action) is reached, at which point the winning program's action is used [16].

In the conventional stateless TPG baseline, program parameters are fixed after evolution and the registers of all programs are reset after each decision. Program selection depends on the current observation only; TPG retains no explicit memory of the paths it has taken. In contrast, a *stateful* variant preserves selected registers across decisions to provide short-horizon memory along the active path, which has been shown to help on partially observable tasks [19].

Both variants offer no explicit memory of paths chosen, limiting an agent's ability to adapt its exploration of a changing environment post-training.

A second limitation comes from TPG following conventional evolutionary-computation algorithms: the emphasis on population-level optimization gives individual programs little scope on how each of their actions can be meaningful and instead focuses on total behavior. The two primary research objectives in this thesis address these challenges, outlined in Section 1.2.

1.2 Research Objectives

1.2.1 Objective 1: Path Memory

To enhance an agent's ability to adapt within its structure after each decision, we introduce a biologically inspired two-level plasticity pipeline in TPG:

- **Root team homeostasis.** The root team maintains a smoothed estimate

of each edge's effective utility by homeostatically tracking normalized bids (Sec. 3.1.2), biasing toward consistent use of useful edges while damping transient spikes.

- **Sub-team Oja adaptation.** When traversal flows from a root team to a sub-team, the sub-team's member programs undergo a normalized Hebbian update using an Oja-style rule [12] (Sec. 3.1.3). This ensures competitive learning within the sub-team that correlates with the activating root program.

Crucially, these weight updates persist across episodes (trials) within a generation, and reset between generations. This mechanism allows the graph to accumulate and refine structural knowledge over many interactions with the environment, aiming to strike a balance between stability and adaptability.

We evaluate this approach on both static and dynamic variants of the MuJoCo Ant environment [6]. Our results demonstrate that the biologically plausible modifications—homeostatic adjustment at the root and normalized Hebbian refinement in following connections—not only improve performance but also significantly reduces the complexity of the solutions in each generation, mitigating the bloating problem commonly observed in genetic programming [27].

1.2.2 Objective 2: Temporal Credit Assignment

While NeuroTPG improves on vanilla TPG, it struggles to assign credit to decisions that influence future rewards (see Sec. 6.1.3). We therefore add temporal-difference (TD) learning [40] to NeuroTPG and pair it with a Quality Diversity (QD) method

[33] that searches the TD parameter space, providing insight into which parameter values are required for TD learning yield the best behavior. The QD method does not raise performance compared to the absence of it but offers interpretability on which parameter niches improve results.

Each agent maintains its own reward-modulated decision structure during an episode, which begins with random initial conditions; all path-specific credit is reset between episodes. Experiments on static and dynamic maze tasks confirm that TD-augmented NeuroTPG is effective across seeds in a dense-reward task relative to TPG and plain NeuroTPG.

1.3 Contributions

Our main contributions according to each objective are as follows.

Objective 1: Path Memory

- A novel *two-level plasticity* framework in TPG that operates hierarchically: root-level homeostatic weight adjustment for global optimization, and sub-team normalized Hebbian learning (Oja's rule) for local specialization.

- *Efficiency without compromise*: Across static and dynamic variants of MuJoCo Ant tasks, the NeuroTPG design reduces complexity, requiring 59–88% fewer mean effective instructions per step. At the same time, it promotes stronger and more consistent high-performing tails, achieving efficiency without loss of robustness.

Objective 2: Temporal Credit Assignment

- *TD-augmented NeuroTPG:* Integrating TD learning enables the agent to stabilize and distribute frequent, small rewards across recent decisions, improving robustness in the dense-feedback maze.
- *MAP-Elites insight:* A MAP-Elites archive used for keeping diversity in the TD learning variables, providing an interpretable view of temporal-credit dynamics.

Together, these contributions establish NeuroTPG and its variants as a biologically inspired framework that unifies plasticity and temporal credit assignment, yielding agents that are both efficient and adaptive across diverse sequential decision tasks.

Chapter 2

Literature Review

For the foundation of this work, we first introduce Evolutionary Computation. In the following section, we position our two-level plasticity pipeline within four strands of prior work: Tangled Program Graphs, Lifetime Learning in Evolutionary Computation, Hebbian Plasticity, and Homeostatic Plasticity and Synaptic Scaling. The prior work for the addition of TD-learning is then followed in the subsequent sections: Temporal-Difference Learning and Quality-Diversity.

2.1 Evolutionary Computation

Evolutionary Computation refers to a family of population-based stochastic search methods inspired by natural selection. A candidate solution (the genotype) expresses a behavior (the phenotype) that is scored by a task-specific objective (fitness). As shown in Figure 2.1, each generation applies selection operators that bias reproduction toward higher-fitness individuals, and variation operators (mutation and

crossover) that introduce diversity. The resulting search dynamics balance exploitation of discovered structure with exploration of novel behaviors [22].

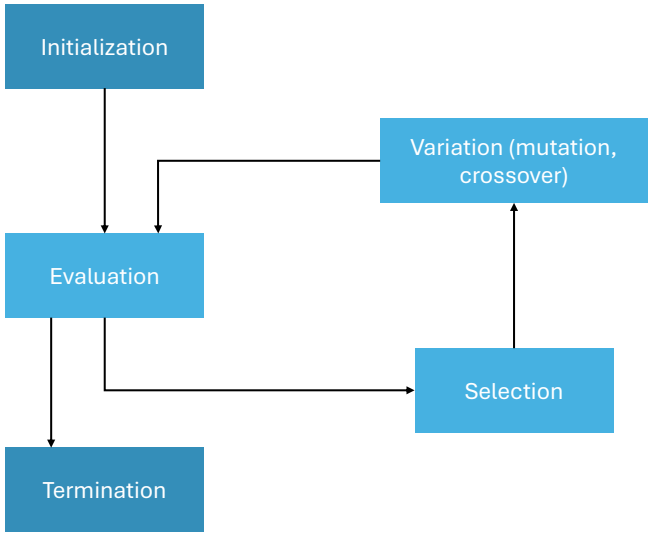


Figure 2.1: An abstract representation of the digital evolution process. It begins with the random initialization of individuals, followed by evaluation. The best-performing individuals are then selected, and variation operators such as mutation and crossover are applied. This cycle of evaluation, selection, and variation continues until a termination condition is met, such as a wall-clock time limit or a specified generation count.

2.1.1 Genetic Programming

Genetic Programming (GP) evolves *computer programs* as individuals [23]. A program maps inputs to outputs (e.g., actions) and is evaluated on task-specific fitness; selection and variation then act on its code. Main forms include (i) *tree-based GP*: programs as expression trees (ii) *Linear GP*: instruction sequences over registers,

(iii) *Graph-based GP*: reusable subgraphs/modules; includes team–program structures as in Tangled Program Graphs (Sec. 2.2)), and (iv): grammar-guided forms that constrain semantics.

Variation operators act by mutation functions, registers, constants, or control flow, and by recombining sub-trees, or blocks depending on the representation. Operator rates are tuned to balance exploration (new code) with exploitation (refining current code).

State and temporal credit. In sequential decision problems, Evolutionary Computation faces a well-known *temporal credit assignment* challenge: fitness aggregates returns over long horizons, providing weak guidance about which micro-decisions improve performance. Classic strategies to mitigate this include shaping objectives, hierarchical decomposition, and hybridization with within-lifetime learning (e.g., reinforcement learning) so that individuals can adapt during an episode while evolution searches over the structure that provides such adaptation [14, 25].

Bloat and introns. Another practical consideration is *bloat*: inactive or weakly contributing code (introns) tends to accumulate under neutral drift, sometimes improving robustness to mutation but at the cost of increasing evaluation time. Common countermeasures include parsimony pressure [32] or structural limits.

2.2 Tangled Program Graphs

The Tangled Program Graph (TPG) framework has emerged as a promising approach for decomposing tasks and building composite agents from a set of previously discovered behaviours [18, 16]. This framework is particularly useful in settings that benefit from automatic problem decomposition and temporal memory [10]. TPGs treat policy learning as the evolution of a directed graph whose vertices are **teams** (decision routing nodes) and edges/leaves are **programs** (computing nodes) (Fig. 2.3).

2.2.1 Program Representation

Each program assumes a Linear Genetic Program (LGP) representation [5]: where their first scalar register supplies a bid, and their second register encodes the action. In these programs, TPGs employ a built-in temporal memory by preserving register values from the previous step, which accumulate to make the programs *stateful*.

Recently, a new variant which adds *vector* and *matrix* registers to each program has enabled the possibility of having a high-dimensional action space, with the scalar LGP still used to produce the scalar bid [10].

An abstract version of a program's register memory representation can be seen in Figure 2.2, where programs are represented as a linear sequence of instructions which operate on this memory, and state variables are drawn from the observation space (see Alg. 1).

In the stateful variant, the registers retain their values across time steps, resetting only at the end of an evaluation. This allows the registers to accumulate and encode

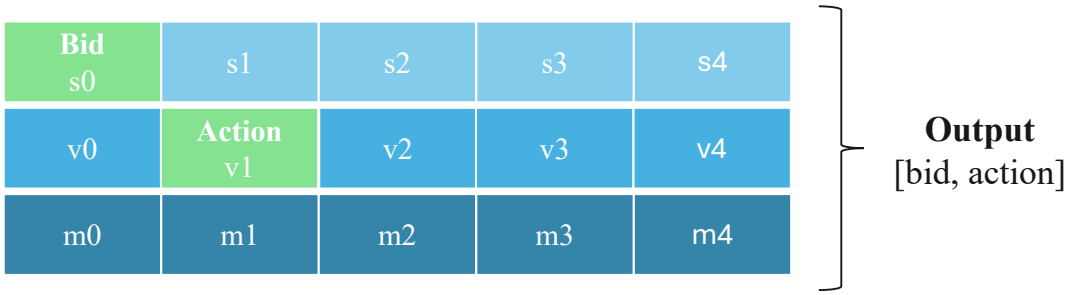


Figure 2.2: Abstract representation of program memory used in this paper. A program has three types of memory registers (scalar, vector, and matrix) that can interact with the observation and each other. This enables an output of high-dimensional actions. More information can be found in [10].

a rough *mental model* of the environment, capturing temporal patterns that guide future decisions. In contrast, in the stateless variant the registers are re-initialized at each step, so this internal temporal memory is not available.

Algorithm 1 Illustrative register-machine program used by a Tangled Program Graph (TPG) agent. Each program contains eight scalar (`s`), vector (`v`), and matrix (`m`) memory instances. Two evolved constants—m_w (vector/matrix width) and o_i (observation offset)—control how the current observation $\vec{obs}(t)$ is copied (Lines 1–3). Memory is reset at the start of each episode; the program returns a bid and continuous action (Line 7).

1: `v0 = roll(`$\vec{obs}(t)$`, -`o_i`)[:`m_w`]` ▷ Copy observation to vector memory
2: `vi = roll(`$\vec{obs}(t)$`, -`o_i`)[:`m_w`*`m_w`]` ▷ Copy observation to temporary vector `vi`
3: `m0 = vi.reshape(`m_w`,`m_w`)` ▷ Copy observation to matrix memory
4: `v3 = s0*vi` ▷ Program execution begins
5: `v1 = s4*v3`
6: `s0 = mean(v3)`
7: **return** `s0,v1` ▷ bid, continuous action vector

Teams

An agent's decision begins at a single root team. The root's member programs execute in series, produce bids, and the program with the highest bid is chosen, with its corresponding action used to update the environment. Through evolutionary selection, mutation, and crossover, highly adapted teams of programs gradually emerge. Mutations can also allow a program to reference another team in the population rather than directly selecting an action (Fig. 2.3). This encourages problem decomposition, breaking down a complex problem into smaller, more manageable parts.

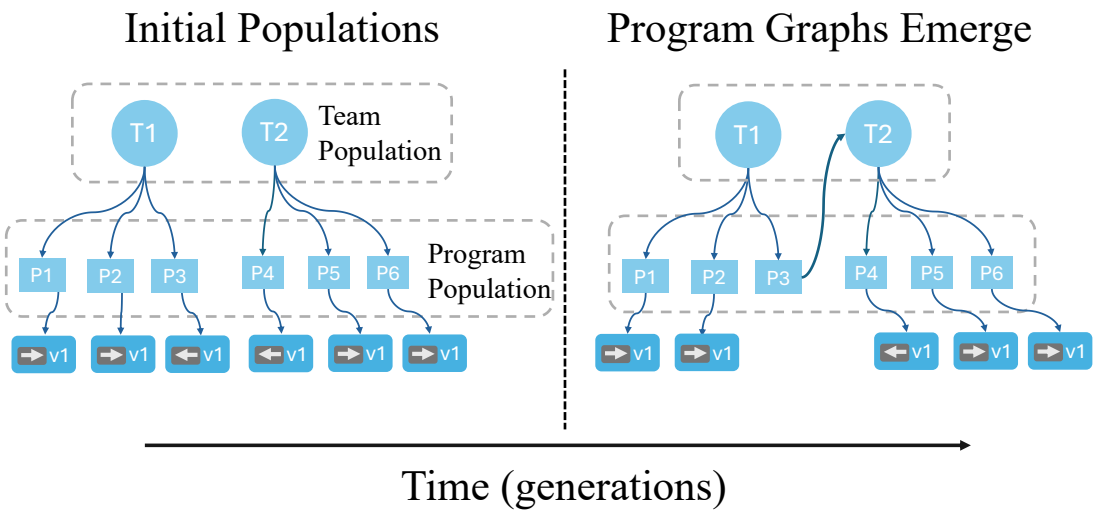


Figure 2.3: Illustration on how emergence of program graphs can occur through evolution. Through the mutation process, solutions can connect with one another, encouraging problem decomposition. A node (T) represents a team, and an edge (P) is a program.

As new interconnections between teams are established, agents consisting of multiple teams may arise. All evolutionary modifications (mutation and crossover) continue to occur exclusively at the agent's root team. This ensures that agents are

constructed from the bottom-up, and lower-level structures are protected from variation as long as the graph as a whole is performing well. However, this property also implies that components deeper in the hierarchy change more slowly over evolutionary time, thus further motivating additional mechanisms which support rapid lifetime adaptation.

This framework has shown notable success in various applications, including Atari game playing agents [16], visual reinforcement learning in ViZDoom [17, 20], and multi-task learning [18]. In these applications, the TPG framework has been competitive with deep learning approaches while producing agents that are several orders of magnitude less computationally complex.

2.3 Lifetime Learning in Evolutionary Computation

An intra-generational plasticity mechanism known as the Baldwin effect, which accelerates convergence through individual learning, has long been explored in neuroevolution [1]. Under the Baldwin effect, individual learning improves an agent's fitness without directly transmitting acquired knowledge to offspring. In Genetic Programming (GP), this often leads to the learned traits during the genotypes lifetime; the *ability to learn* is inherited, closely aligning with biological learning and the Baldwin effect. However, much existing research focuses on embedding gradient-based local search within an agent's structure [42] and follows Lamarckian evolution

in which these changes are inherited by offspring. Although not a biological mechanism, Lamarckian updates are popular because they can speed convergence and improve sample efficiency; however, they may reduce evolvability or encourage premature convergence [7]. Our pipeline follows the Baldwin effect while remaining entirely gradient-free, hence maintaining GP's compatibility with non-differentiable programs.

2.4 Hebbian Plasticity

Hebbian plasticity refers to the broader class of biologically inspired mechanisms that adjust synaptic strengths based on neural co-activity [15]. Hebbian Learning is a biologically inspired unsupervised learning strategy that has gained significant attention in the field of deep learning. An extension of this, Oja's rule [30], a stabilized variant of Hebbian Learning, has been central to unsupervised learning, particularly for Principal Component Analysis (PCA) and dynamic weight stabilization in neural networks [38]. PCA is an unsupervised dimensionality-reduction technique that finds the orthogonal directions (principal components) which captures the greatest variance in the data. It projects high-dimensional inputs onto a lower-dimensional space, aiming to retain as much of the original variability as possible. Recent work demonstrates that Oja's rule can replace biologically implausible engineering tricks (e.g., batch normalization) in deep networks, enabling robust learning under constraints like online training and sub-optimal initialization [37]. Traditional Hebbian Learning strengthens connections between simultaneously activated neurons, while

Competitive Hebbian Learning introduces a selection mechanism in which neurons compete for activation, allowing only the strongest connections to emerge [28]. This competitive principle has proven valuable across various neural architectures; when applied to train early convolutional layers, Competitive Hebbian Learning produced features that rivaled or surpassed backpropagation in downstream classification accuracy [24].

Position in this work: While Hebbian Learning is usually applied inside neural networks to directly adjust synaptic weights, we embed Hebbian updates in a TPG agent such that the updates steer which programs (edges) are followed during an episode, without those Hebbian-modified values themselves being parts of the genome or evolutionary search (Fig. 3.1). This allows the agent to adapt its decision-making within a lifetime, complementing the slower process of evolution with rapid, experience-driven adjustments.

2.5 Homeostatic Plasticity and Synaptic Scaling

Homeostatic plasticity (HP) provides essential negative feedback to counteract the unbounded growth induced by Hebbian Learning: when a neuron's firing rate drifts from its target, all its synapses undergo synaptic scaling, where synapses are multiplicatively scaled to restore activity to a stable setpoint [41]. Unlike correlation-based Hebbian updates which are fast, local, and destabilizing, HP acts on a slower timescale and preserves the *relative* strengths of incoming weights while normalizing their sum, attaining information retention and network stability [43].

Position in this work: We implement synaptic scaling at the TPG's *root team*, the only team guaranteed to be visited at every decision. This makes homeostatic plasticity a *global* gain control on routing: it preserves relative edge strengths while normalizing their sum, preventing over-potentiated edges from monopolizing traversal. Because each individual is a TPG rooted at a single team, stabilizing bids at the root regularizes the structure that mutation and crossover operate on (Sec. 3.1.2).

2.6 Temporal Credit Assignment in Evolutionary Computation

A core difficulty in sequential tasks is *temporal credit assignment*: fitness aggregates many-step outcomes while genomes encode local structures whose effects are delayed and irregular. With sparse or deceptive rewards, purely episodic signals are not robust and can induce short-horizon overfitting.

Two broad responses recur in the literature. (i) Reshape evaluation to expose intermediate structure—potential-based shaping, curricula, or sub-goal decompositions—acknowledging bias and optimality tradeoffs, and the need for expert domain knowledge [2]. (ii) Combine evolution with within-episode adaptation, where evolution searches structure/hyperparameters and short-lag credit comes from reinforcement-learning updates during evaluation [8]. In such hybrids, one-step temporal-difference learning, TD(0), often supplies low-variance credit signals; optional eligibility traces bias updates toward recent selections without constructing multi-step targets [40, 21].

We next review TD learning as the canonical short-horizon credit mechanism

used in these hybrids (Sec. 2.6.1).

2.6.1 Temporal-Difference Learning

A central challenge in reinforcement learning is learning the value of states from experience. Temporal-Difference (TD) learning provides an elegant solution by combining the bootstrapping of dynamic programming with the sampling-based approach of Monte Carlo methods [40]. Instead of waiting for a final outcome, a TD agent updates its value estimate $\mathrm{V}(s_t)$ using the currently observed reward r_{t+1} and its own estimate of the next state's value, $\mathrm{V}(s_{t+1})$. This process is driven by the one-step TD error (TD(0)), δ_t, which measures the discrepancy in a single-step prediction:

$$\delta_t \;=\; r_{t+1} \;+\; \gamma\, V(s_{t+1}) \;-\; V(s_t), \tag{2.6.1}$$

where $\gamma \in [0, 1]$ is the discount factor controlling the effective planning horizon (larger γ places more weight on distant rewards, smaller γ on near-term rewards).

Eligibility Traces

Eligibility traces provide a backward-view mechanism that assigns decaying credit to recent features or parameters. In TD(λ), the forward view mixes n-step returns; the equivalent online backward view maintains traces:

$$e_t \leftarrow \gamma\lambda\, e_{t-1} + \nabla_\theta V_\theta(s_t), \tag{2.6.2}$$

$$\Delta\theta \propto \alpha\, \delta_t\, e_t. \tag{2.6.3}$$

where the eligibility trace is given with e_t, and carries decaying credit; γ is the same discount factor weighting future outcomes in Equation 2.6.1; λ, the trace-decay parameter controlling the backward credit span; θ are the parameters being updated; α is the learning rate.

Equation 2.6.2 is used to update the traces and Equation 2.6.3 is to update the parameters.

Eligibility-like mechanisms also underlie *three-factor* plasticity rules in computational neuroscience, where a Hebbian term is gated by a delayed neuromodulatory signal; these provide a biological lens on TD-style credit assignment at synapses [44].

TD in evolutionary systems. Early Learning Classifier Systems (LCS) established that evolutionary search over structures and TD updates over values can coexist: evolution discovers rule structures while TD(0) updates their "strengths" [14, 25]. This EvoRL pattern where we establish structure by evolution and value by learning, motivates our integration for TPGs.

Position in this work. We adopt a scalar running baseline at the agent level in a TD-style, and maintain eligibility traces on program–program transitions to gate Hebbian plasticity. The trace parameter λ and a decay rate γ are set per team (evolved across generations), so a single agent may express different credit spans and forgetting rates as it routes through different teams. This mitigates classical TPG's local credit-assignment limitation, enabling rapid within-structure adaptation without genome changes. We analyze the roles of γ (discount) and λ (backward credit span), and use MAP-Elites to explore the per-team (λ, decay) landscape across tasks (Sec. 2.7).

2.7 Quality-Diversity

Quality-Diversity (QD) algorithms aim to produce a diverse set of solutions that cover a defined feature space, while simultaneously maximizing performance across this space. These algorithms assess solution performance using standard fitness evaluations and ensure diversity through an n-dimensional descriptor. In this study, we utilize Map-Elites (ME) [29], a QD algorithm that maintains an archive of solutions organized by their respective descriptors. The ME process begins with the initialization phase, where n_{pop} random individuals are generated, evaluated, and placed into the archive based on their descriptors. The algorithm then iteratively progresses by uniformly selecting n_{batch} individuals from the archive, applying variation operators to produce new offspring, and evaluating these new solutions. Each newly generated solution is subsequently considered for insertion into the archive based on its descriptor and fitness. If an archive cell corresponding to a solution's descriptor is identified and empty, the solution is directly inserted; if the cell already contains a solution, a local fitness-based competition determines which individual occupies the cell, with the superior individual retained.

Position in this work: In the work, we incorporate ME into the NeuroTPG + TD learning variant to track and analyze the dynamics of γ and λ in relation to the problem domain. While ME did not lead to a performance improvement and introduced additional variance across final fitness scores, the interpretability it provides justifies its inclusion.

Chapter 3

Methodology

This chapter is organized into two sections. The first examines how to incorporate neuroplasticity into TPG agents (Sec 3.1); the second outlines how to add reward signals to the rules (Sec 3.2). In both cases, what is inherited is the capacity to learn—architectural biases and plasticity mechanisms—not the learned parameters.

3.1 Hierarchal Neuroplasticity in Agents

We enhance TPG with a hierarchal neuroplasticity scheme. By first stabilizing the raw bids using shifted softmax [4], we can apply our learning rules. A *homeostatic synaptic-scaling* rule nudges pointer weights toward normalized bids in [0,1] at the root team, while *competitive Hebbian* (Oja) updates correlation weights inside each child team visited (Fig. 3.1). This division reflects biological observations: global activity is stabilized, whereas local synapses compete on short time-scales. The goal is for NeuroTPG to achieve both long-term robustness and in-episode adaptation. The

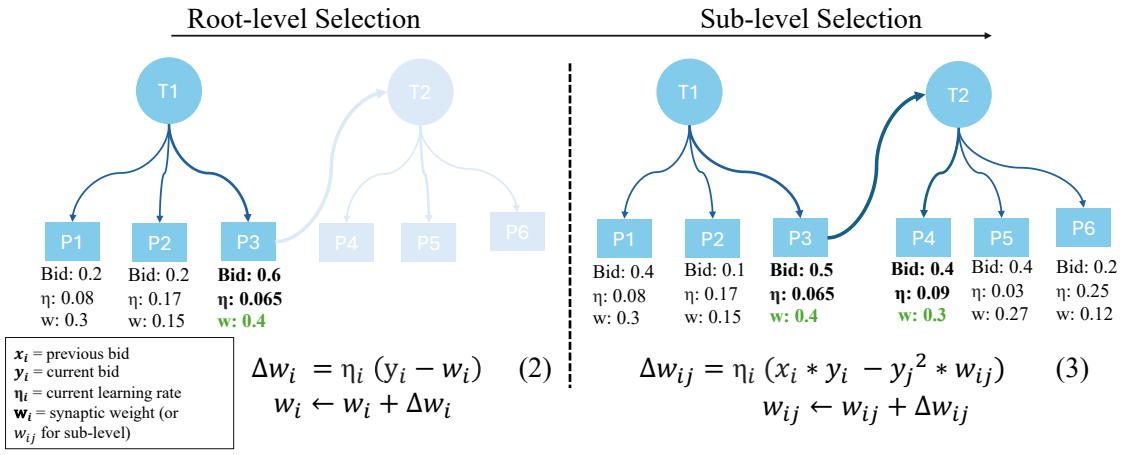


Figure 3.1: Graph showing the transition from a root-level selection to a sub-level selection of a program. Each program is initialized with a noise-based weight, which is thereafter an evolvable attribute (Sec. Evolved Program Parameters). The appropriate program-specific learning rule is then utilized where the highest-weight is selected.

following subsections detail the two learning rules, the program hyper-parameters which allow these learning rules to be more effective, and how they are embedded in the TPG architecture.

3.1.1 Softmax Bid Normalization

Raw bid values produced by each program can vary significantly. Before any learning rule is used, we pass the vector of bids of the current team, $\mathbf{x}$ through a shifted soft-max:

$$y_i = \frac{\mathbf{e}^{x_{j-a}}}{\sum_{i=1}^{n} \mathbf{e}^{x_{i-a}}} \quad j = 1 : n. \tag{1}$$

where n is the number of programs; $a = \max_{k=1,\ldots,n} x_k$, i.e., the maximum bid value among the n programs, y_i is the scaled bid for program i. This maps bids to value in $[0, 1]$ while preserving relative magnitudes and avoiding overflow which can occur due to the unbounded range of bid values.

3.1.2 Root-level Homeostatic Synaptic Scaling

In cortex, homeostatic synaptic scaling multiplicatively normalizes a neuron's inputs to stabilize firing rates [41]. Here, we apply an analogous mechanism to the root team's member program's pointer weights. After each decision step, we update the pointer weight w_i of the program i with

$$\Delta w_i = \eta_i(y_i - w_i) \tag{2}$$

where $y_i \in [0, 1]$ is the program i's normalized bid and η_i is its learning rate (discussed in Sec. Evolved Program Parameters). Equation (2) is a commonly used delta rule [9] that nudges every weight towards its average bid, all while preserving their relative ordering. In practice this means the root's traversal concentrates on the high-bidders, while low-bidding paths are gradually de-emphasized, focusing the search on the best candidates. Because the rule is only applied at the root, it guides the global decision-making without interfering with the competitive specialization that occurs deeper in the graph.

3.1.3 Competitive Local Learning inside Child Teams

The outgoing pointers to non-root teams form a fully connected, directed graph. Each edge is dynamically weighted by Oja's stabilized Hebbian update:

$$\Delta w_{ij} = \eta_i(x_i y_j - y_j^2 w_{ij}) \tag{3}$$

where x_i is the mapped bid of the preceding program, y_j is the bid of the candidate program, and η_i is the *post-synaptic* learning rate carried by program i (an evolved parameter discussed in Sec. Evolved Program Parameters). Oja's negative normalization term ($-y_j^2 w_{ij}$) prevents runaway growth, ensuring that useful pathways are amplified while the total input to each program remains bounded. Typically, Oja's rule leverages the PCA advantage by adaptively identifying directions of maximum variance in multi-dimensional inputs, however, since we apply this learning rule to scalar values, we lose this benefit, reducing its role primarily to a normalization method [3].

To prevent the possibility of a positive-feedback lock-in where the same program repeatedly wins, we add an upper bound of 5 on reinforcement (i.e., on the effective strengthening of an edge). In vanilla TPG, selection ultimately penalizes overbidding programs; the cap simply accelerates stabilization without changing that long-run pressure.

3.1.4 Evolved Program Parameters

With the addition of the local learning rules, each program maintains two scalar parameters: the learning rate η and the initial noise ν. The purpose of these parameters is to break the bid symmetry which is established and prevent the selection of the same program every time. The initialization of the learning rate is a random value taken from the log normal distribution between 0.00 and 0.25. The initialization of the noise is the same, instead between 0.00 and 0.50. We use this distribution due to its biological plausibility [35]. At the beginning of every new generation, we mutate both values so there is always a log-normal self-adaptation. With probability p, we draw a Gaussian perturbation $\delta \sim \mathcal{N}(0, \sigma)$, multiply the current value by e^{δ}, and clamp the result to the interval $[0, 1]$. Each register also maintains constants as the starting values. These evolved constants are passed down through generations and adjusted via mutation to initialize registers. Initially, they are assigned at random, but evolution refines them over time alongside other mutation and crossover operations in LGPs and TPGs. Following the method in [34], each constant can be mutated by scaling it with a random value drawn uniformly from [0.5,2.0], with a 50% chance of mutation. Additionally, constants have a 10% chance of having their sign flipped.

3.1.5 NeuroTPG Execution with Learning Rules

Algorithm 2 performs a single decision step in NeuroTPG. At each time-step, bids are normalized, adjusted according to the team type, and the program with the highest adjusted weight is executed. Weights are cleared at the end of each generation so

Algorithm 2 NeuroTPG selection (single decision step). Raw bids are normalized by a shifted softmax (Eq. (1)); these p_i modulate plasticity: at the root, connections are rescaled homeostatically (Eq. (2)); in sub-teams, an Oja-style update (Eq. (3)) uses the previous-edge regularized weight *prev*. The program with the largest *post-update* weight w_i is selected. Connection-specific weights reset at the end of each generation.

procedure SELECTPROGRAM(*bids*, *team*, *prev*)
  $p \leftarrow$ SHIFTEDSOFTMAX(*bids*) ▷ p_i used to modulate updates
  for each program i **in** *team* **do**
    $w_i \leftarrow$ CURRENTWEIGHT(i)
    if TEAM.IS_ROOT **then**
      $w_i \leftarrow$ HOMEOSTATICSCALE(w_i, p_i)
    else
      $w_i \leftarrow$ OJAUPDATE($w_i, i, prev, p_i$)
    end if
  end for
  return $\arg\max_i w_i$
end procedure

that lifetime adaptations are not inherited.

This dual-scale plasticity lets NeuroTPG discover task-specific decompositions rapidly (via Oja) while maintaining long-term stability and search diversity (root-level homeostasis), yielding lower complexity through all agents and producing a stronger solution (Sec. Results and Discussion). Here, long-term stability refers to the homeostatic scaling at the root team, which prevents any single program from dominating the population and thereby avoids runaway growth or collapse of candidate solutions over many generations. Search diversity refers to the preservation of alternative pathways that remain viable, keeping the evolutionary search open to novel adaptations.

3.2 Graph Path Refinement with Reward Signals

Learning in maze environments is dominated by the *credit-assignment* problem: the agent must infer which decisions along a trajectory were responsible for the eventual outcome. NeuroTPG + TD tackles this challenge by embedding a lightweight temporal-difference (TD) learner inside each agent and by augmenting the bidding mechanism with eligibility-traced probabilities. This section details the TD component and its integration with program selection.

3.2.1 Temporal-Difference Credit Assignment

We follow the ideas in Sutton and Barto [40] to propagate reward information across time. Rather than training a separate parametric critic, we keep a scalar running baseline V (per team) and use an online TD-style update. Approximating $V(s_{t+1}) \approx V(s_t)$ from Equation 2.6.1 yields the practical update used in our code: Let $v_t \in \mathbb{R}$ denote the current value baseline (a scalar maintained online). After executing action a_t in state s_t and receiving the current step reward r_t, the one-step TD error is computed with a state independent version and the baseline is updated with:

$$\delta_t \;=\; r_t \;-\; (1-\gamma)\, V_t, \tag{3.2.1}$$

$$V_{t+1} \;=\; V_t \;+\; \alpha\, \delta_t, \tag{3.2.2}$$

where $\gamma \in [0, 1]$ is the discount factor, $\alpha > 0$ is the baseline learning rate (for all experiments, $\alpha = 0.02$).

3.2.2 Eligibility-Traced Bids

To propagate credit across a chain of calls, each program maintains a scalar bid-eligibility trace e_i. At every decision step, traces decay and the executed program receives an additive boost proportional to the product of consecutive bids along the call chain. Concretely, let `prevbid` be initialized to 1.0 at the root team and set to the winning program's (current) bid thereafter. If program i executes at time t with (current) bid `currbid`, then

$$e_i \leftarrow \gamma \lambda e_i \qquad \text{for all } i, \tag{3.2.3}$$

$$e_i \leftarrow e_i + \texttt{prevbid} \cdot \texttt{currbid} \quad \text{for the executed program } i, \tag{3.2.4}$$

and we update the program preferences in the active team by

$$u_i \leftarrow u_i + w_i \delta_t \, e_i, \tag{3.2.5}$$

where the weight w_i is the pointer weight of the program i changed through the learning rules (Sec. 2.5) and $\eta > 0$ is the bid learning rate. After the update, preferences are re-normalized into probabilities via the softmax and sampled for the next decision. Traces are reset to zero after the completion of an episode.

The error δ_t is a one-step prediction error that supplies the signed learning signal. Eligibility traces e_t are not used to form a TD(λ) target, instead, they distribute δ_t over recently co-active program to program transitions to gate Hebbian updates. The multiplicative co-activity term `prevbid`·`currbid` emphasizes strongly expressed

choices, while the trace decay $\gamma\lambda$ controls how quickly influence from previous selections fades:

$$e_t(i \to j) \;=\; \gamma\lambda\, e_{t-1}(i \to j) \;+\; \underbrace{\mathrm{pre}_t(i) \times \mathrm{post}_t(j)}_{\text{bid co-activity}}, \tag{3.2.6}$$

where $e_t(i \to j)$ is the eligibility trace for the transition from program i to j; $\mathrm{pre}_t(i)$ is the normalized bid of the presynaptic program i at time t; and $\mathrm{post}_t(j)$ is the bid of the postsynaptic program j. This equation uses uses δ_t to modulate Hebbian plasticity; for the gated update mechanism, we apply a target-tracking update:

$$\Delta w_{ij} \;\propto\; \big(\mathrm{post}_t(j) - w_{ij}\big) \times \delta_t\, e_t(i \to j) \tag{3.2.7}$$

where w_{ij} is the synaptic weight between programs i and j. The difference term $\mathrm{post}_t(j) - w_{ij}$ pulls the weight toward the current postsynaptic activation. This gated mechanism has updates more dependent on the reward signal; we leave an additive reward signal for future work.

Evolved parameters. Within a team, TD hyper-parameters are shared: the discount γ and the bid-trace decay λ (Sec. 3.2.2). Because agents are graphs of teams (via team pointers), a single agent may implicitly contain multiple (γ, λ) settings; The team which contains the final action program executed has the agent use its parameters. In the Map-Elites variant, (γ, λ) serve as genotypic descriptors (Sec. 3.2.3), encouraging diversity in credit-assignment dynamics. In the non–Map-Elites variant, both are evolved directly, constrained to $[0, 1]$. Mutation follows the methodology

presented for program parameters (Sec. 3.1.4). Crossover is done by choosing either parent's parameter with equal probability.

3.2.3 Map-Elites

Building upon TD-learning hyperparameters and their critical roles, we leverage QD algorithms to ensure a diverse set of solutions effectively spanning the feature space defined by these parameters. The solutions' structural characteristics, represented by an n-dimensional descriptor, form the basis for maintaining this diversity in ME. Specifically, each bin in ME corresponds to a unique combination of λ and γ, where each parameter is discretized into 10 sequential intervals ranging between 0 and 1. This discretization results in a total of 100 bins, each uniquely characterized by distinct parameter combinations.

Chapter 4

Environments

Modern reinforcement-learning studies typically evaluate algorithms on a *suite* of environments rather than a single task. This thesis follows that convention but focuses on two complementary benchmarks:

1. The continuous-control **MuJoCo Ant** locomotion task, whose high-dimensional dynamics test scalability.

2. A custom **Maze Navigation** task with dense, step-wise shaping rewards that evaluates how TD stabilizes and distributes temporal updates, with the dynamic variant further testing rapid adaptation via hidden action remapping.

Together they expose both the symbolic search component of NeuroTPG and the TD-style baseline to qualitatively different challenges.

4.1 MuJoCo-Ant

Figure 4.1 depicts the quadruped robot used in the standard MuJoCo Ant task introduced by Schulman *et al.* [36]. The agent observes a 27-dimensional state vector and applies torques in an 8-dimensional action space. Its high-dimensional, continuous dynamics and requirement for coordinated limb motion make it a challenging benchmark for testing the capabilities of our neuroplasticity enhanced TPG (**NeuroTPG**).

The reward (R) at each time step is

$$R = healthy + forward - control\_cost \tag{4}$$

where *healthy* is a constant bonus for maintaining an upright posture, *forward* is proportional to forward displacement, and *control_cost* penalizes large torques.

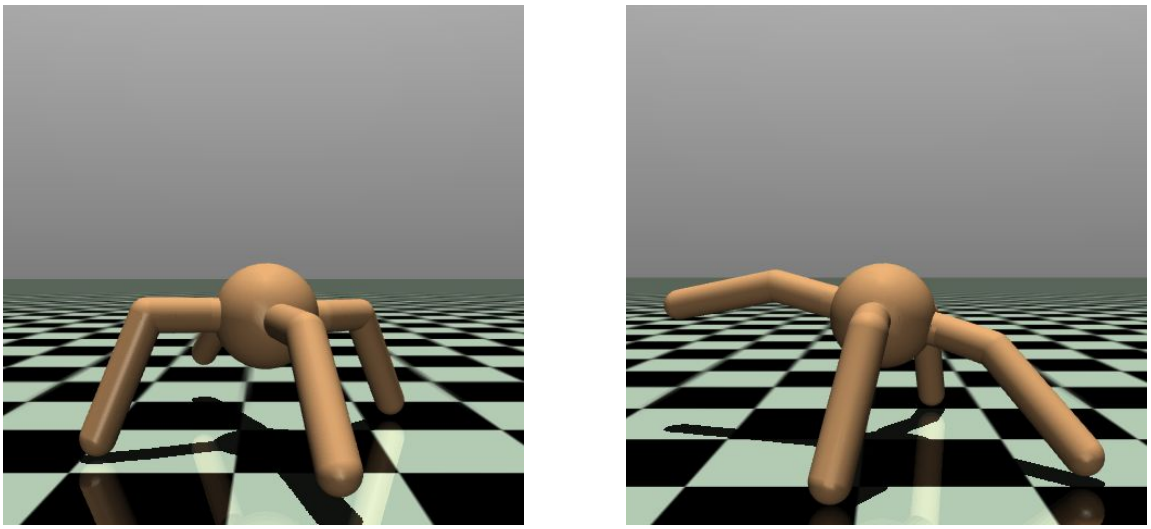


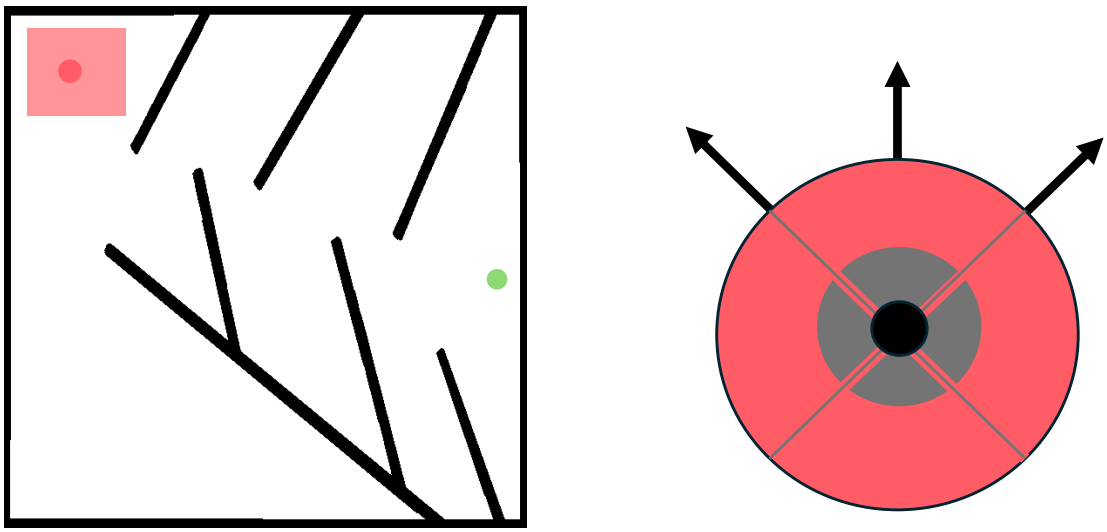
Figure 4.1: The MuJoCo Ant robot used as our benchmark environment.

4.1.1 Damaged Morphology Variant

In the *Immediate-Break* (IB) variant, one randomly chosen leg is disabled at the start of each episode: its two actuator commands are clamped to zero for all timesteps. This contrasts with the standard Ant (Baseline), where all four legs remain functional. Both share identical state, action, and reward definitions, differing only in the disability of a leg. By disabling a different leg in each episode, we introduce hidden variability in the transition dynamics, effectively creating a dynamic environment [31], enabling a direct comparison of performance under a *static* morphology (Baseline) versus a *dynamic* morphology that demands rapid policy adjustment (IB).

4.2 Maze Navigation

To assess TD learning under dense, stepwise feedback, we use a 2D maze in which progress toward a fixed goal requires sustained sequences of moves. This setting emphasizes stabilizing and distributing frequent small updates over extended horizons, rather than bridging sparse rewards.



(a) (b)

Figure 4.2: (a) Maze used in this work. The red rectangle indicates the region from which start positions are sampled for each episode (trial); the red circle within the region marks a particular sampled start; green circle marks the stationary goal. (b) is the robot and its sensors. An agent has three distance-to-wall sensors (arrows) and four sensors (gray wedges) used as a "compass" towards the goal irrespective to walls.

Figure 4.2 shows the 600 × 600 pixel maze used throughout, inspired by [13]. Unlike classic mazes that feature deep local minima, this layout is closer to a path navigation task. Agent observations follow [26] (see Fig. 4.2b for sensor geometry). The distance-to-wall sensors are normalized to a range of [0,1] where the maximum distance is 100 pixels. The four "compass" values are one-hotted where values are either 0 or 1 if the goal is in the direction. The action space is discrete with unit-pixel moves: up, down, left, and right. At each timestep t, the agent receives a shaping

reward proportional to its reduction in distance to the goal:

$$R_t \;=\; \frac{d_{t-1} - d_t}{D_{\max}}, \tag{4.2.1}$$

where d_t is the agent–goal distance at time t and $D_{\max}$ is a fixed normalizer (we use the maze diagonal, $D_{\max} = \sqrt{W^2 + H^2}$ for a $W \times H$ maze). Upon reaching the goal at time t, a one-time terminal bonus is awarded,

$$\text{Goal}_t \;=\; 100 \cdot \frac{T_{\max} - t}{T_{\max}}, \tag{4.2.2}$$

which scales by completion earliness; otherwise $\text{Goal}_t = 0$.Because the shaping term $-(d_t - d_{t-1})/D_{\max}$ is typically $\mathcal{O}(10^{-2})$, we multiply all rewards by 100 before passing them to the TD-style reward baseline. This rescaling stabilizes TD errors without changing optimal policies.

4.2.1 Dynamic Variant: Hidden Action Permutation

To probe rapid adaptation, we introduce a *hidden* non-stationarity. At the start of each episode, we sample a switch time in the first 20 timesteps and a permutation over the four actions. For the remaining timesteps, the environment applies the remapped action while observations remain unchanged. Thus the transition dynamics shift mid-episode without any explicit cue. NeuroTPG's eligibility-traced bids (Sec. 3.2.2) allow value information to propagate across this switch, enabling recovery without explicit system identification.

Chapter 5

Experimental Setup

5.1 Architecture Setup

Table 5.1: Hyper-parameters used by all evolutionary seeds (unless specified in subsequent paragraph). Structural limits, mutation/crossover settings, population-generation parameters, action-pointer mutation.

Parameter	**Value (p = probability)**
Team max size	∞
Team mutation rate	0.6 (p)
Team crossover rate	0.5 (p)
Program max size	∞
Program mutation rate	0.175 (p)
Instruction mutation rate	0.1 (p)
Root teams kept per generation	1 000
Root teams generated per generation	500
Action-pointer mutation rate	0.125 (p)

Table 5.1 lists the values shared by *all* seeds in this study. By leaving both

team size and program size unconstrained (set to ∞), we allow maximal structural flexibility, with consideration of the risk of unbounded bloat. In each generation, 1000 root teams are retained, of which 500 are preserved as elites from the previous generation and the remaining 500 are generated. Parents are selected from this elite set using tournament selection with size 3, and the 500 new root teams are generated from them via crossover and/or mutation. These genetic operators are applied only at the root team level and include independent events—addition and deletion, which ensures balanced structural variation. The probability of an action pointer switching to a team pointer is set to 0.125.

SF denotes the stateful variant and SL denotes the stateless variant. Both variants were evaluated in single-task static and dynamic environments, comparing fitness scores and agent complexity over 250 generations.

TD-learning and Map-Elites With the addition of TD-learning and Map-Elites, there is an increase in hyper-parameters. Here we discuss the values changed/added from Table 5.2. The root teams kept in the ME variant is set to 100 due to the total bins in the archive we preserve being 100 (see Sec. 3.2.3).

Parameter	**Value (p = probability)**
λ mutation rate	0.5 (p)
γ mutation rate	0.5 (p)
Root teams kept per generation (ME)	100
Root teams generated per generation (ME)	600

Table 5.2: Hyper-parameters used by TD-learning and MAP-Elites. (ME) indicates parameter settings specific only to Map-Elites. Values are regarding mutation/crossover settings and population-generation parameters.

5.2 Environmental Setup

Unlike most prior work, each episode is 500 steps (vs. the usual 1,000) per simulation. This shorter duration focuses our testing on the initial behaviour following leg damage. We maintain this as the baseline across all test cases to ensure consistent measurement metrics. An episode here refers to one complete run of the Ant agent, starting from its initial state and ending after the specified number of time-steps. Therefore, the evaluation of each agent is over 40 independent episodes—preserving only program weights between episodes—and the average results are reported.

For the maze we use a default value of 1000 time-steps per simulation with a total of 20 independent episodes. These episodes would always have a random starting position for the agent where orientation of the agent will always face west, towards the origin of the map. Furthermore, all non-TD learning variants are run with stateful programs, as credit assignment requires a dependence on memory (further explained in Sec. 6.2.1). In contrast, TD-embedded agents use stateless programs to isolate the contribution of the TD-style baseline to decision quality.

The memory register size (vector and matrix) of all variants are changed depending on the environment where the Mujoco-Ant tests are kept at 27 to match the observation dimension and the Maze environment uses a size of 7.

Chapter 6

Results and Discussion

6.1 Neuroplasticity (Ant)

In this section, we compare the performance of NeuroTPG and TPG across static and dynamic environments using both SF and SL variants. Performance is assessed in terms of fitness and complexity, with additional analysis of variability across runs.

6.1.1 Performance

Both the SF and SL NeuroTPG variants produce higher-fitness agents at the end of evolution and lower complexity—measured as mean effective instructions per step—relative to vanilla TPG in both environments (Tab. 6.1). Convergence trends are broadly similar (Fig. 6.1f,g). NeuroTPG's within-generation plasticity, however, makes outcomes more sensitive to initial conditions than vanilla TPG: fitness variance across seeds is higher, as reflected by larger standard deviations (Tab. 6.1). The

Table 6.1: Best agent performance and median performance across seeds (± standard deviations) on two MuJoCo Ant variants. Abbrev.: SF = stateful; SL = stateless. Fitness (Fit) is the mean episodic return; Complexity (Comp) is the mean effective instructions per step.

Variant	**Ant**		**Ant Immediate-Break**	
	Best Agent: Fit/Comp	Median Fitness	Best Agent: Fit/Comp	Median Fitness
TPG:SF	592.78 / 7198	531.546±12.05	532.94 / 2367	525.52±2.55
TPG:SL	583.66 / 5485	533.24±23.09	549.07 / 1304	522.29±10.05
NeuroTPG:SF	**884.70 / 79**	533.43±89.28	**598.98 / 1264**	526.45±11.76
NeuroTPG:SL	815.22 / 342	531.51±67.10	560.56 / 288	522.63±11.01

Bold: best agent produced in respective environment.

agents were run for an additional 60 post-evolution test episodes, which still showed a wide spread in the box plots (Fig. 6.1a,b). Despite this variability, NeuroTPG achieves substantially lower complexity than vanilla TPG (Fig. 6.1d,e), especially in the SL variant.

Quantitatively, relative to vanilla TPG, NeuroTPG reduces complexity by

- **Ant**: SF (66.2%↓), SL (82.4%↓)
- **Ant IB**: SF (59.3%↓), SL (88.5%↓)

This indicates a more parameter-efficient search: reduced complexity without sacrificing performance.

Statistical Significance To assess robustness, we launched 20 seeds at five team-connection probabilities, 25%, 20%, 15%, 12.5% (the setting used elsewhere in this paper (Tab. 5.1)), and 5%, for a total of 100 seeds. Per-seed final fitness did not show a significant central-tendency advantage for NeuroTPG (Wilcoxon signed-rank

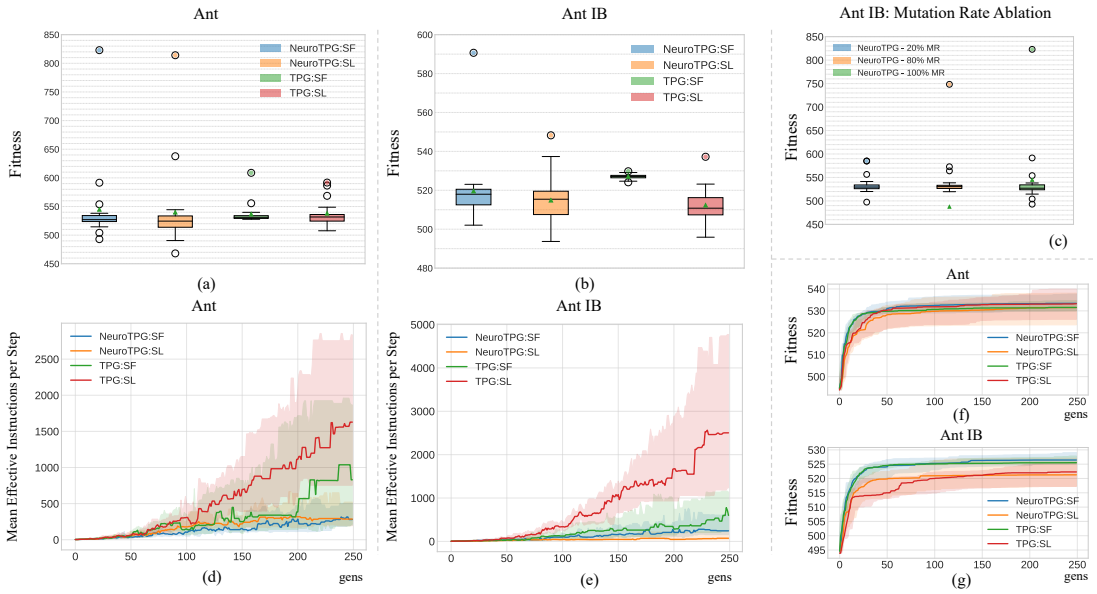


Figure 6.1: Performance of TPG variants on the Ant tasks where Ant is the static and Ant Immediate-Break (IB) is the dynamic variant. Abbrev.: SF = stateful; SL = stateless. Solid lines denote the median, and shaded regions indicate the standard deviation across 20 runs. (a) Ant test set: best agent per seed (60 episodes/seed). (b) Ant IB test set: best agent per seed (same protocol). (c) Mutation-rate ablation on the held-out test set. (d,e) *Evolution-time complexity*: mean effective instructions per step across generations. (f,g) *Evolution-time fitness*: mean episodic return across generations in each environment.

test). In contrast, a heavy-tail analysis using the per-config TPG $q_{0.95}$ threshold found consistent advantages: NeuroTPG produced $14/100 = 14.0\%$ outlier seeds [95% CI $0.085, 0.221$] vs. TPG's $5/100 = 5.0\%$ [95% CI $0.022, 0.112$]; McNemar's test [39] favored NeuroTPG ($b = 10$, $c = 1$), $p = 0.00586$. The magnitude of above-threshold exceedance, combined across configs via Fisher's method [11], was significant ($p = 0.0228$). Hence, the chance of observing ≥ 1 outlier in a 20-seed batch was 0.951 for NeuroTPG [95% CI $0.832, 0.993$] vs. 0.642 for TPG [95% CI $0.353, 0.907$]. We restrict inferential claims to the SF variant.

6.1.2 Rule Ablation

Effect of rule placement

To isolate the contribution and examine the effects of each biological mechanism, we retrain with different components in an ablation study on the Ant IB: SF variant:

Table 6.2: Learning-rule sequence for each TPG variant. Step 1 applies the rule at the root team; Step 2 applies it within each sub-team. Abbrev.: H = Homeostasis (Eq. 2); O = Oja (Eq. 3).

Variant	Step 1	Step 2
TPG – Homeostasis	H	–
TPG – Oja	–	O
TPG – Global Homeostasis	H	H

Table 6.2 summarizes the variants. In *Global Homeostasis*, the homeostatic update (Eq. 2) is applied at both the root and all sub-teams (replacing Oja), providing a correlation-agnostic control. This isolates the specific contribution of Oja's correlation term by testing whether cross-program correlation is necessary for performance. The rule-ablation results in Table 6.3 highlight distinct and complementary roles for the two plasticity mechanisms. Removing Oja's rule caused the largest performance drop, underscoring its role in rapid, state-dependent adaptation within an episode. Homeostasis alone retains some robustness but cannot prevent erratic routing from transient high-bidders, while even simple local updates reduce complexity relative to vanilla TPG (Tab. 6.3). These results suggest the two mechanisms are complementary, homeostasis stabilizes global routing, and Oja promotes local adaptability, together achieving higher performance with more compact policies.

Variant	**Best Agent: Fitness / Complexity**
TPG – Homeostasis Rule	533.20/1606
TPG – Oja's Rule	586.76/1140
TPG – Global Homeostasis	541.92/289
NeuroTPG	**598.98/1264**

Table 6.3: Ablation study: removal of steps from the proposed pipeline. The best agent's fitness and its complexity are reported.

Impact of evolved parameters

In this study, we evolved two key program parameters—learning rate and initial noise level—and then trained with different mutation probabilities $p \in \{20, 80, 100\}\%$ to assess their impact (Fig. 6.1 c). Although both median fitness and policy complexity remained largely unchanged across values of p, the maximum fitness achieved by the top-performing agents increased consistently with higher mutation rates. This finding suggests that more aggressive variation of these parameters during evolution produces stronger agents.

6.1.3 Transfer test

The decision not to reset the synaptic weights aligns with biological plausibility in that organisms continually learn with experience from many evaluations. However, this raises the question of whether it enables genuine learning from experience and,

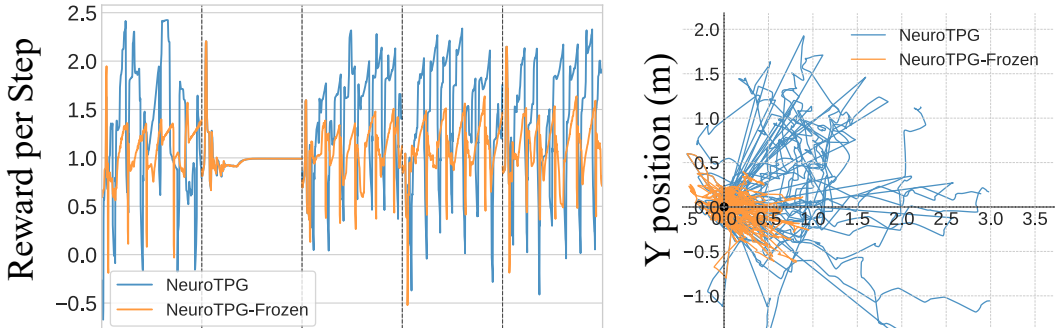


Figure 6.2: Replay of the best NeuroTPG agent trained in the standard Ant environment, now tested on the Dynamic variant. Performance with active neuroplasticity weights is compared to a frozen version in which raw bids determine program selection. (a) Reward per step for the first five episodes (500 steps each). A negative reward means the Ant moves backwards. (b) XY trajectory of the ant at every step; each line represents one episode. NeuroTPG agents with plastic weights consistently traverse farther distances, highlighting the benefit of adaptive program selection.

if so, how that experience can be better utilized for dynamic environments. To test this, we take the best NeuroTPG SF agent trained on the static Ant environment and run it on the dynamic variant for 40 episodes. In this experiment, recovery from a broken leg is compared with a "frozen" NeuroTPG agent which relies on its bids rather than its weights. As shown in Figure 6.2a, the plastic NeuroTPG achieves higher rewards on average than the frozen agent but also exhibits substantial variance, including episodes with negative rewards—indicating that the Ant moves backwards. This suggests that, while plasticity improves adaptability, it does not yet guarantee stability or consistent forward progress. Figure 6.2b further shows that the plastic agents generally travel farther, but sometimes follow inefficient or reversed paths. These behaviours motivate potential improvements such as reward modulation or enhanced local credit assignment to reduce undesirable movement and

stabilize performance without sacrificing adaptability.

6.2 TD-style Reward Modulation (Maze)

We compare **TPG**, **NeuroTPG**, and **NeuroTPG + TD** with and without **MAP-Elites (ME)** on a Maze environment under a static and dynamic variant. We report fitness (mean episodic return) and complexity (average effective instructions per step).

6.2.1 Stateless vs. Stateful Programs

To evaluate the necessity of memory in TPG and NeuroTPG (without TD-learning), we first tested agents with stateless programs. Across all seeds, both variants converged to local minima, achieving median scores of 14.08 ± 0.01 in the static maze and 8.70 ± 0.01 in the dynamic maze. These results highlight that memory is essential for solving this maze setup. For all subsequent experiments, we therefore utilize stateful programs for both TPG and NeuroTPG to ensure agents retain the memory necessary for effective decision-making in the maze tasks. In contrast, all TD variants use stateless programs to effectively credit programs without having their internal structure accumulating.

6.2.2 Performance

Table 6.4: Best agent performance and population means on maze variants. Fitness (Fit) is the mean episodic return; Complexity (Comp) is the average effective instructions per decision.

Variant	**Static Maze**		**Dynamic Maze**	
	Best Agent: Fit/Comp	Median Fit	Best Agent: Fit/Comp	Median Fit
TPG	91.17/285	33.75±20.88	19.87/48	17.68±1.37
NeuroTPG	79.10/180	29.28±17.48	20.44/5	18.12±1.46
NeuroTPG+TD	**98.99/1**	96.80±1.28	**42.68/2**	31.90±3.64
NeuroTPG+TD+ME	95.05/0	92.47±7.94	38.51/3	30.42±3.38

Bold: best agent produced in respective environment.

Table 6.5: Best agent and median test-set pass rates (goal reached). Values are all divided by the total number of episodes, 100.

Variant	**Static Maze (N=100)**		**Dynamic Maze (N=100)**	
	Best Agent	Median	Best Agent	Median
TPG	0.87	0.00	0.00	0.00
NeuroTPG	**0.98**	0.00	0.00	0.00
NeuroTPG+TD	0.86	0.77	0.02	0.01
NeuroTPG+TD+ME	0.83	<u>0.79</u>	**0.04**	<u>0.01</u>

Bold: best agent per environment. <u>Underline</u>: highest median per environment.

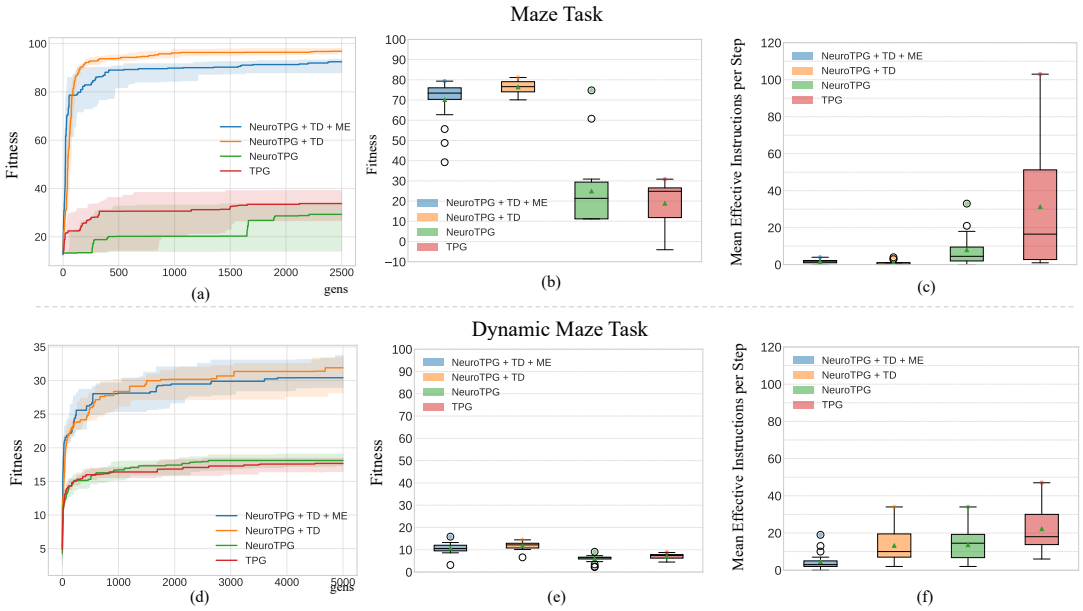


Figure 6.3: Row 1: static, Row 2: dynamic Maze. Solid lines denote the median, and shaded regions indicate the standard deviation across 20 runs.(a,d) fitness across generations for each variant; (b,e) Variance in final performance of NeuroTPG+TD variants shown as box plots; (c,f) Box plots of effective instructions per step (complexity) for each variant in the final generation.

Across seeds, both TD-enabled variants outperform their non-TD counterparts on fitness in *both* static and dynamic mazes (Tab. 6.4). The ME variant shows larger across-seed variance—as expected from quality-diversity search—while its medians are comparable to NeuroTPG+TD without ME. Complexity decreases monotonically from TPG → NeuroTPG → NeuroTPG+TD variants (Fig. 6.3c) in the static maze environment. However, due to all variants' low performance in the dynamic environment, the complexity across all variants remain relatively similar.

In Table 6.5, the test case results (out of 100) show that the best agents across

all variants can consistently solve the static maze, with the NeuroTPG variant performing the strongest. While other agents achieve higher final fitness by completing successful runs more quickly (Tab. 6.4), NeuroTPG yields the most reliable success overall. The median results highlight that non-TD variants achieve no successful runs, underscoring the necessity of outliers for performance. For the dynamic maze, both the best-agent and median success rates remain low across all variants.

Final-position plots indicate that failures under TD variants are sparse and not spatially clustered but are still in the trajectory to the goal (Fig. **??**c,d,g,h). In contrast, non-TD variants in the static maze achieve higher pass rates (Fig. **??**a,b) though inspection of their policies in the next paragraph reveals overfitting. Motivated by this visual pattern, we further investigate the effect of longer test horizons in Section 6.2.2.

From the final positions of TPG and NeuroTPG (Fig. **??**a,b), it can observed that there is an extreme cluster of the best agent's final position. These two best-performing agents are outliers which learn to exploit a specific sequence whereas TD-embedded agents are reactive. In the static maze, NeuroTPG (without TD) often exploits a high-return but brittle route (Fig. 6.4a); TD-enabled agents take shorter, more direct trajectories (Fig. 6.4b), yielding higher pass rates at similar or lower complexity.

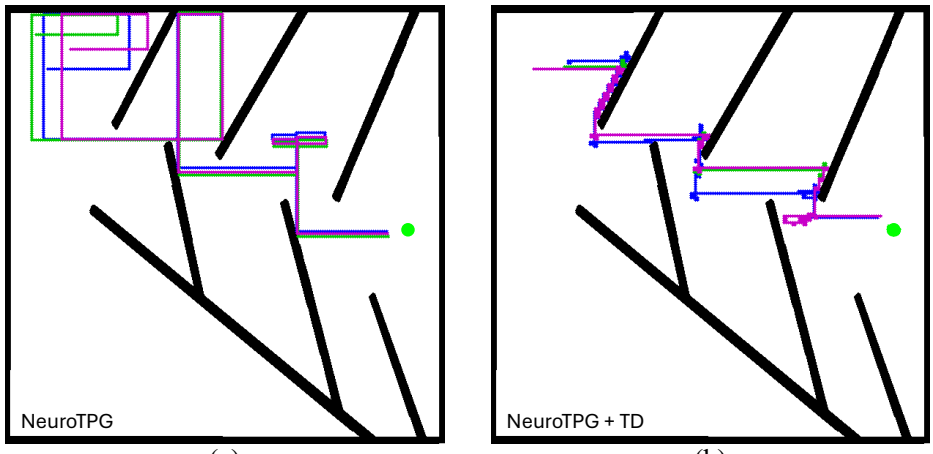


Figure 6.4: Trajectory of three distinct starting positioning successful episodes (trials). a) NeuroTPG and b) NeuroTPG + TD.

Learning curves do not appear to have plateaued by the end of the budget (Fig. 6.3a,d), additional generations may yield further gains.

Evaluation–horizon sensitivity To test whether TD-learning helps agents exploit longer time budgets at *test time*, we increased the max episode length from 1,000 to 5,000 steps (training unchanged). We report median [success rate/return] over all 20 seeds (Tab. 6.6).

Across both tasks, TD-enabled variants exhibit larger gains with longer horizons than non-TD baselines (i.e., a positive horizon–method interaction), indicating that TD-learning can improve the agent's ability to capitalize on extended time budgets rather than merely needing more generations.

Method	1k steps	5k steps	Δ (5k−1k)
TPG	0/50	0/50	0
NeuroTPG	0/50	0/50	+0
NeuroTPG + TD	0/50	6/50	+6
NeuroTPG + TD + ME	1/50	7/50	+6

Table 6.6: Evaluation–horizon sensitivity with max episode (trial) length increased at *test time* from 1,000 to 5,000 steps (training unchanged). Values reported are median pass rates across all seeds on 50 episodes.

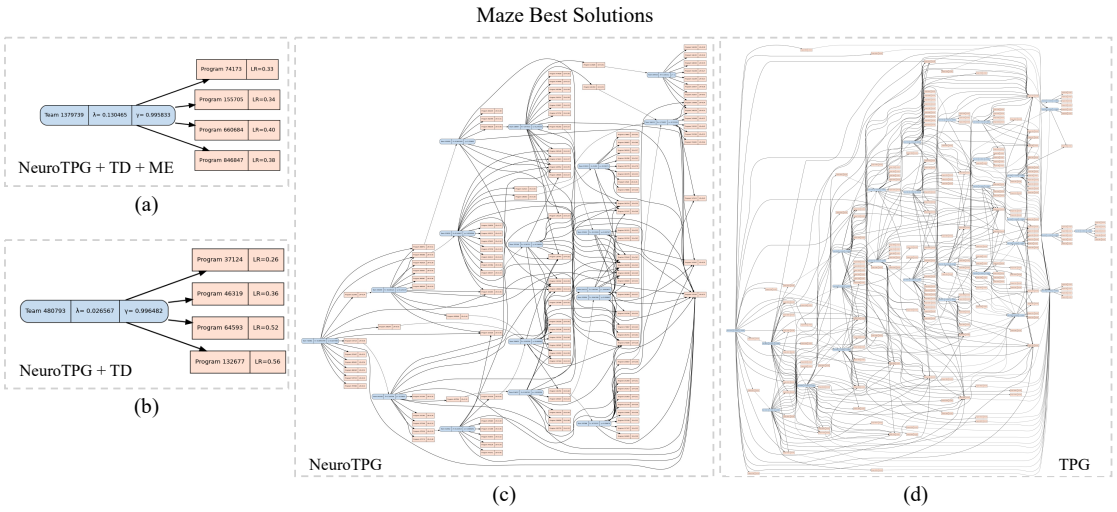


Figure 6.5: Evolved graphs for best fitness agents in all variants for static Maze environment. Blue nodes: teams (labels show ID, λ and γ (these two parameters are ignored and not evolved in TPG and NeuroTPG variants)); Edges: programs (labels show ID, learning rate (LR)) where the LR is not applicable to the TPG variant.

Complexity. As shown in Fig. 6.5 and Tab. 6.4, NeuroTPG reduces structural complexity relative to TPG, and NeuroTPG+TD is more compact still. The best agents from each NeuroTPG+TD variant use four programs that map one-to-one to

the discrete actions (Fig. 6.6). The hierarchical reduction in structural complexity, progressing from d to a, highlights the effectiveness of incorporating both TD and the learning rules, each contributing to complexity reduction through their respective dynamics.

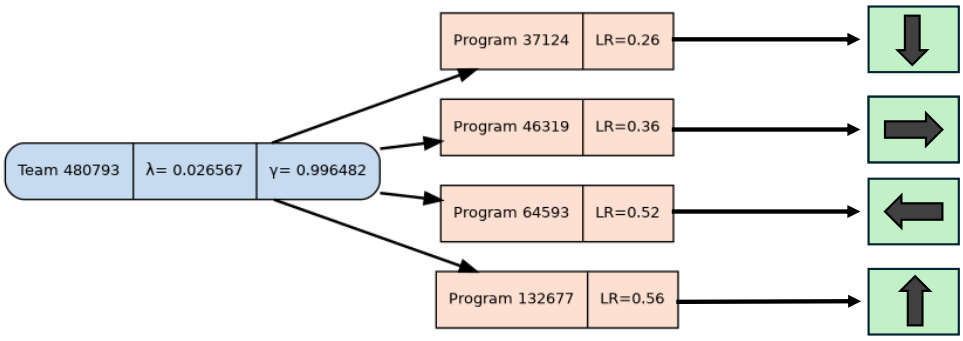


Figure 6.6: The best NeuroTPG + TD agent of the static environment. The programs map to their specific individual action.

The low mean effective instructions per step in these agents (see Tab. 6.4) is also due to evolution learning to rely on the evolved constants rather than operations in the program 3.1.4. Since evolutionary changes adddress *how much* a program gets updated if it was used, the starting time steps make every program selectable, which discourages permanently unused programs that can persist in vanilla TPG without immediate performance penalties.

6.2.3 Archive Structure

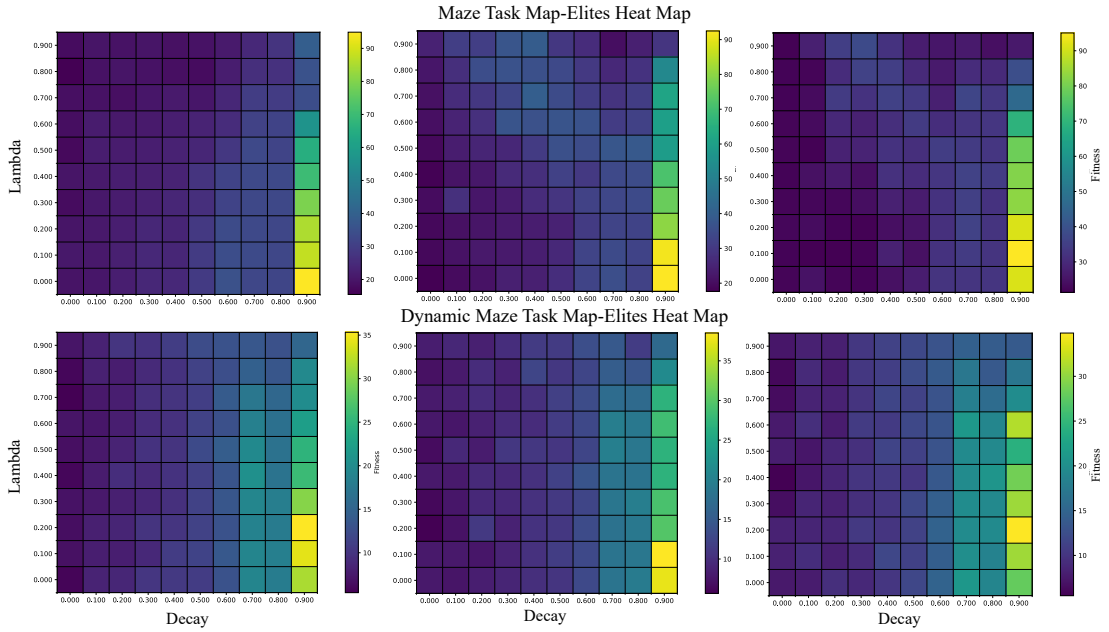


Figure 6.7: ME archive heatmaps (top: static; bottom: dynamic maze). for the top three seeds per environment. Vertical axis: λ, horizontal axis: γ; cell intensity: fitness.

The MAP-Elites archive in Fig. 6.7 concentrates elites in bins with $\gamma \geq 0.9$ and $\lambda < 0.1$. This indicates that, for this maze task, rapid local adaptation is favored over long-range temporal assignment. The observation aligns with TD(0): the TD error δ_t reflects near-term outcomes, and short-lived traces reduce interference from weakly correlated distant events in a dense-reward setting. Because γ also acts as the discount for the critic (here, a running baseline), this reflects a coupled timescale, moderate critic horizon (e.g., $\sim$10 steps when $\gamma \approx 0.9$) paired with near-instantaneous within-episode plasticity. Finally, since TPG permits team-to-team calls, these bins report the root team's parameters and thus are most interpretable as

the *initial* decision dynamics, not a full accounting of downstream teams (although final structures of TD-variants have only one team). We hypothesize that, across tasks with differing temporal credit demands, MAP-Elites will partition elites along these bins, making the archive a useful lens on task-specific timescales; exploring this in multi-task settings is left for future work.

Chapter 7

Conclusion

This thesis has advanced Tangled Program Graphs (TPGs) along two complementary dimensions to address the challenges of lifelong adaptation and delayed reward in non-stationary environments.

First, we introduced a biologically inspired **two-level plasticity** pipeline (Objective 1). At the root team, a homeostatic weight-scaling mechanism gradually biases selection toward consistently high-bidding programs. At the sub-team level, we applied normalized Hebbian updates (Oja's rule) to reinforce correlations between activating root programs and their successors. Across both static and dynamic MuJoCo-Ant tasks, the best NeuroTPG agent with two-level plasticity achieved higher final episodic returns than vanilla TPG while using **55–88 % fewer effective instructions per action**, demonstrating more compact and efficient evolutionary searches.

Second, we addressed temporal credit assignment (Objective 2) by integrating a lightweight TD(0)-style baseline with eligibility traces into NeuroTPG and coupling it

with a MAP-Elites archive over the TD hyperparameter space. In the dense-feedback maze, TD primarily stabilizes and distributes frequent stepwise rewards across recent decisions, improving within-episode adaptation in both static and dynamic variants. MAP-Elites provides an interpretable landscape of (γ, λ) niches, revealing which temporal-difference settings yield the best behavioral adaptation.

Together, these contributions show that embedding biologically plausible plasticity and reinforcement-learning mechanisms directly within the evolutionary search framework substantially improves adaptability, performance, and efficiency of TPG-based agents.

Bibliography

[1] L. Aguilar, S. Bennati, and D. Helbing. How learning can change the course of evolution. *PLOS ONE*, 14:e0219502, 09 2019. doi: 10.1371/journal.pone.0219502.

[2] H. Bai, R. Cheng, and Y. Jin. Evolutionary reinforcement learning: A survey. *Intelligent Computing*, 2:0025, 2023. doi: 10.34133/icomputing.0025. URL `https://spj.science.org/doi/abs/10.34133/icomputing.0025`.

[3] P. Baldi and P. J. Sadowski. The ebb and flow of deep learning: a theory of local learning. *CoRR*, abs/1506.06472, 2015. URL `http://arxiv.org/abs/1506.06472`.

[4] P. Blanchard, D. J. Higham, and N. J. Higham. Accurate computation of the log-sum-exp and softmax functions, 2019. URL `https://arxiv.org/abs/1909.03469`.

[5] M. F. Brameier and W. Banzhaf. *Linear Genetic Programming*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 1441940480.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv*, 1606.01540, 2016.

[7] B. Burlacu, S. M. Winkler, and M. Affenzeller. *Revisiting Gradient-Based Local Search in Symbolic Regression*, pages 259–273. Springer Nature Singapore, Singapore, 2025. ISBN 978-981-96-0077-9. doi: 10.1007/978-981-96-0077-9_13. URL https://doi.org/10.1007/978-981-96-0077-9_13.

[8] J. D. Co-Reyes, Y. Miao, D. Peng, E. Real, S. Levine, Q. V. Le, H. Lee, and A. Faust. Evolving reinforcement learning algorithms, 2022. URL https://arxiv.org/abs/2101.03958.

[9] M. Dawson. Connectionism and classical conditioning. *Comparative Cognition and Behavior Reviews*, 3 (Monograph):1–115, 01 2008. doi: 10.3819/ccbr.2008.30008.

[10] T. Djavaherpour, A. Naqvi, E. Zhuang, and S. Kelly. Evolving Many-Model Agents with Vector and Matrix Operations in Tangled Program Graphs. In *Genetic Programming Theory and Practice XXI*. Springer (AD), 2024.

[11] R. A. Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, Edinburgh, 1925.

[12] D. O. Hebb. (1949) donald o. hebb, the organization of behavior, new york: Wiley, introduction and chapter 4, "the first stage of perception: growth of the

assembly," pp. xi- xix, 60-78. In *Neurocomputing, Volume 1: Foundations of Research*. The MIT Press, 04 1988. ISBN 9780262267137. doi: 10.7551/mitpress/4943.003.0006. URL https://doi.org/10.7551/mitpress/4943.003.0006.

[13] D. Herel, D. Zogatova, M. Kripner, and T. Mikolov. Emergence of novelty in evolutionary algorithms. In *The 2022 Conference on Artificial Life*, ALIFE 2022. MIT Press, 2022. doi: 10.1162/isal_a_00501. URL http://dx.doi.org/10.1162/isal_a_00501.

[14] J. H. Holland. *Genetic Algorithms and Adaptation*, pages 317–333. Springer US, Boston, MA, 1984. ISBN 978-1-4684-8941-5. doi: 10.1007/978-1-4684-8941-5_21. URL https://doi.org/10.1007/978-1-4684-8941-5_21.

[15] T. Keck, T. Toyoizumi, L. Chen, B. Doiron, D. Feldman, K. Fox, W. Gerstner, P. Haydon, M. Hübener, H.-K. Lee, J. Lisman, T. Rose, F. Sengpiel, D. Stellwagen, M. Stryker, G. Turrigiano, and M. van Rossum. Integrating hebbian and homeostatic plasticity: The current state of the field and future research directions. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 372:20160158, 03 2017. doi: 10.1098/rstb.2016.0158.

[16] S. Kelly and M. I. Heywood. Emergent tangled graph representations for atari game playing agents. 2017. URL https://www.paperdigest.org/paper/?paper_id=doi.org_10.1007_978-3-319-55696-3_5.

[17] S. Kelly and M. I. Heywood. Emergent solutions to high-dimensional multitask

reinforcement learning. *Evol. Comput.*, 26(3):347–380, Sept. 2018. ISSN 1063-6560. doi: 10.1162/evco_a_00232. URL https://doi.org/10.1162/evco_a_00232.

[18] S. Kelly, R. J. Smith, and M. I. Heywood. Emergent policy discovery for visual reinforcement learning through tangled program graphs: A tutorial. In *Genetic Programming Theory and Practice*, 2018. URL https://api.semanticscholar.org/CorpusID:59413962.

[19] S. Kelly, R. J. Smith, M. I. Heywood, and W. Banzhaf. Emergent tangled program graphs in partially observable recursive forecasting and vizdoom navigation tasks. *ACM Trans. Evol. Learn. Optim.*, 1(3), Aug. 2021. doi: 10.1145/3468857. URL https://doi.org/10.1145/3468857.

[20] S. Kelly, T. Voegerl, W. Banzhaf, and C. Gondro. Evolving hierarchical memory-prediction machines in multi-task reinforcement learning, 2021. URL https://arxiv.org/abs/2106.12659.

[21] H. Kimura and S. Kobayashi. An analysis of actor/critic algorithms using eligibility traces reinforcement learning with imperfect alue functions.

[22] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5.

[23] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[24] G. Lagani, G. Amato, F. Falchi, and C. Gennaro. Training convolutional neural networks with hebbian principal component analysis, 2020. URL https://arxiv.org/abs/2012.12229.

[25] P. L. Lanzi. *Classifier Systems*, pages 172–178. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_115. URL https://doi.org/10.1007/978-0-387-30164-8_115.

[26] J. Li, J. Storie, and J. Clune. Encouraging creative thinking in robots improves their ability to solve challenging problems. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 193–200, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326629. doi: 10.1145/2576768.2598222. URL https://doi.org/10.1145/2576768.2598222.

[27] S. Luke and L. Panait. A comparison of bloat control methods for genetic programming. *Evolutionary computation*, 14:309–44, 02 2006. doi: 10.1162/evco.2006.14.3.309.

[28] T. Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. 1993. URL https://api.semanticscholar.org/CorpusID:123120074.

[29] J. Mouret and J. Clune. Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909, 2015. URL http://arxiv.org/abs/1504.04909.

[30] E. Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[31] S. Padakandla. A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Computing Surveys*, 54(6):1–25, July 2021. ISSN 1557-7341. doi: 10.1145/3459991. URL `http://dx.doi.org/10.1145/3459991`.

[32] R. Poli and N. F. McPhee. Parsimony pressure made easy. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, page 1267–1274, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605581309. doi: 10.1145/1389095.1389340. URL `https://doi.org/10.1145/1389095.1389340`.

[33] J. Pugh, L. Soros, and K. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3, 07 2016. doi: 10.3389/frobt.2016.00040.

[34] E. Real, C. Liang, D. R. So, and Q. V. Le. Automl-zero: evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[35] G. Scheler. Logarithmic distributions prove that intrinsic learning is hebbian. *F1000Research*, 6:1222, 10 2017. doi: 10.12688/f1000research.12130.2.

[36] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional

continuous control using generalized advantage estimation, 2018. URL https://arxiv.org/abs/1506.02438.

[37] N. Shervani-Tabar and R. Rosenbaum. Meta-learning biologically plausible plasticity rules with random feedback pathways, 2023. URL https://arxiv.org/abs/2210.16414.

[38] N. Shervani-Tabar, J. Eshraghian, A. Lindsey, and T. Lillicrap. Oja's plasticity rule overcomes challenges of training neural networks under biological constraints. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2406.06669.

[39] M. Smith and G. Ruxton. Effective use of the mcnemar test. *Behavioral Ecology and Sociobiology*, 74:133, 10 2020. doi: 10.1007/s00265-020-02916-y.

[40] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[41] G. G. Turrigiano. The self-tuning neuron: Synaptic scaling of excitatory synapses. *Cell*, 135(3):422–435, 2008. ISSN 0092-8674.

[42] L. D. Whitley, V. S. Gordon, and K. E. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, PPSN III, page 6–15, Berlin, Heidelberg, 1994. Springer-Verlag. ISBN 3540584846.

[43] F. Zenke, W. Gerstner, and S. Ganguli. The temporal paradox of hebbian learning and homeostatic plasticity. *Current Opinion in Neurobiology*, 43:166–176, 2017. ISSN 0959-4388. doi: https://doi.org/10.1016/j.conb.2017.03.015. URL `https://www.sciencedirect.com/science/article/pii/S0959438817300910`. Neurobiology of Learning and Plasticity.

[44] Łukasz Kuśmierz, T. Isomura, and T. Toyoizumi. Learning with three factors: modulating hebbian plasticity with errors. *Current Opinion in Neurobiology*, 46:170–177, 2017. ISSN 0959-4388. doi: https://doi.org/10.1016/j.conb.2017.08.020. URL `https://www.sciencedirect.com/science/article/pii/S0959438817300612`. Computational Neuroscience.