# GENETIC ENCODING FOR EVOLVING INSTINCTS AND LEARNING IN NON-AXIOMATIC REASONING SYSTEMS

---

A Dissertation
Submitted to
the Temple University Graduate Board

---

In Partial Fulfillment
of the Requirements for the Degree
DOCTOR OF PHILOSOPHY

---

by
Christian Geoffrey Hahm
May 2026

Examining Committee Members:

Justin Yuan Shi, Advisory Chair, Computer and Information Sciences
Pei Wang, Computer and Information Sciences
Yan Wang, Computer and Information Sciences
Claes Strannegård, External Reader, University of Gothenburg

## ABSTRACT

The Non-Axiomatic Reasoning System (NARS) is a logic-based AI architecture that supports explainable reasoning, explicit knowledge representation, general-purpose processing, and online learning. Despite these advantages, scaling NARS up to complex application domains remains difficult because strong autonomous performance typically depends on extensive manual design. Often in practice, one or more human designers must study the target domain, craft background knowledge, tune numeric system parameters, and sometimes modify the system architecture to obtain a high-performing system configuration. While pre-training is standard in neural approaches, no general method exists to automatically pre-optimize NARS systems for learning and performing in sensorimotor domains.

This dissertation introduces a methodology and genetic encoding to optimize NARS configurations using evolutionary algorithms. In contrast to neural models optimized via gradients, NARS configuration spans symbolic knowledge structures, discrete mechanism choices, and continuous parameters, yielding objectives that are not readily differentiable; this motivates the use of evolutionary search. The proposed evolutionary approach serves a role analogous to pre-training in neural systems: it produces a task-adapted initialization of background knowledge and system configuration prior to deployment to achieve better performance and learning abilities. Given an agent's sensorimotor vocabulary (i.e., the sets of its sensors and motors), the encoding generates and mutates background knowledge ("instincts") intended to bootstrap useful behavior. In addition, the encoding optimizes numeric meta-parameters that influence NARS architecture and learning dynamics. To increase expressive power, the encoding was extended with support for the implantation and learning of variables and compound structures, improving performance in scenarios that benefit from more expressive knowledge patterns.

The proposed method was evaluated across four domains spanning perception, control, multi-agent interaction, and planning: (1) a multi-agent wheeled-robot ecosystem environment, (2) a 2D multi-agent grid-world survival environment, (3) a BlocksWorld planning

puzzle scenario, and (4) 3D soft voxel robot locomotion (including both pure NARS and a neuro-symbolic variant). Experiments were designed to assess the effects of evolved instincts alone, runtime ("lifelong"/"online") learning alone, and the combination of both, establishing the contribution of each method to task performance. Across these domains, the evolutionary process consistently discovered NARS agents that achieved substantial improvements in task performance. Agents with configurations evolved using the basic encoding outperformed fixed-configuration baselines, and the extended encodings with variables and compounds provided additional gains on select tasks. Because NARS maintains explicit internal knowledge structures, evolved behavior was able to be further analyzed through inspection of the system's memory contents to connect agent actions to the learned or implanted reasons that produced them.

Overall, this dissertation presents the first genetic encoding for NARS that can jointly evolve meta-parameters and structured symbolic knowledge for sensorimotor domains. It empirically shows that evolutionary optimization with the proposed genetic encoding yields substantial performance gains over fixed-configuration baselines, and that extensions to the encoding can improve performance over the basic encoding in certain scenarios benefiting from generalization and compounds. Evolved agents can be deployed in application settings and also serve as analyzable artifacts for research and development, providing a path toward more robust general-purpose logical reasoning systems.

Dedicated to my parents,

Dr. Geoffrey Hahm and Melissa Zapp Hahm,

for inspiring me as a student

## ACKNOWLEDGEMENTS

I would like to extend my gratitude to the following individuals for their encouragement, inspiration, and support throughout the development of this dissertation.

First and foremost, I thank the members of my dissertation committee, Professor Pei Wang, Professor Justin Yuan Shi, Professor Yan Wang, and Professor Claes Strannegård, for their unwavering support, encouraging spirit, and thoughtful feedback.

Professor Justin Yuan Shi and Professor Yan Wang have taken a genuine interest in this work, and I deeply appreciate their thoughtfulness, understanding, and guidance. Their excellent feedback greatly improved the quality of this dissertation.

Professor Claes Strannegård has been an inspiration to me throughout the years as one of the few researchers working on the vitally important topics of AI evolution, logic, and ecosystems. Through his unique expertise, he provided valuable feedback on this work and has inspired both this dissertation and my other work over the years. I am grateful for his long-standing dedication to these important topics and for his commitment to examining this dissertation.

I would like to express my deepest gratitude to my academic advisor and mentor, Professor Pei Wang. In addition to inventing the AI system NARS, which serves as the foundation of this work, he has profoundly shaped this project and all of my academic work through his unique and thoughtful perspectives on cognition and artificial intelligence. He has generously shared these ideas with me and with the other members of our lab, the Temple AGI Team (TAGIT). I will always treasure his mentorship.

Professor Daniel Strongin deserves especially heartfelt thanks. Without his support, this dissertation would not have been possible. I also thank Joshua Yancy for recommending me to the graduate school program.

I thank the members of the Temple AGI Team for their experiments, dedication to NARS and AGI, and the valuable results and discussions they have shared, which have advanced the study of both AGI and NARS. In particular, I acknowledge Patrick Hammer, Xiang

Li, Peter Isaev, Hongzheng Wang, Mina Gabriel, Tangrui Li, Bowen Xu, Edward Sharick, Boyang Xu, and Maxim Tarasov for their efforts. I also wish to acknowledge the members of the AGI Society for hosting the AGI conference over many years and for bringing together a community of thoughtful, passionate, and remarkable people to share important ideas.

I especially want to thank my wife, Katarina Pokrajac Hahm, for her support throughout this time. She worked tirelessly to provide for our family while I pursued this degree, and I am deeply proud of her accomplishments and dedication. Her hard work enabled me to focus my efforts on learning and research so as to produce my best work. Our loyal cockapoo dogs, Jake and Oliver, brought me limitless joy and companionship during this time, and I thank them from the bottom of my heart.

Finally, I praise God for the many blessings given to me during this time and throughout my life.

## TABLE OF CONTENTS

Page

ABSTRACT ..... ii

DEDICATION ..... iv

ACKNOWLEDGEMENTS ..... v

LIST OF TABLES ..... x

LIST OF FIGURES ..... xi

CHAPTER

1 INTRODUCTION ..... 1

1.1 Thesis Statement and Approach ..... 5

1.2 Main Contributions ..... 6

1.3 Scope and Generality ..... 6

1.4 Dissertation Outline ..... 7

2 LITERATURE REVIEW ..... 8

2.1 AI Genetic Encodings and Evolutionary Algorithms ..... 8

2.2 Multi-Agent Ecosystems Combining Evolution and Learning ..... 11

2.3 Non-Axiomatic Reasoning System (NARS) ..... 14

2.3.1 NARS as a Next-Generation Reasoning System ..... 14

2.3.2 The Future of NARS ..... 20

3 *NARS-UNITY* SYSTEM OVERVIEW ..... 24

3.1 Architecture ..... 25

3.2 Control Mechanism ..... 28

3.2.1 Task Processing of Judgment ..... 28

3.2.2 Task Processing of Goal ..... 30

3.2.3 Sentence Table ..... 31

3.3 Temporal Learning at Runtime ..... 32

3.4 Personality Parameters (Meta-Level Parameters) ..... 34

4 ENCODING NARS PERSONALITY AND BASIC INSTINCTS ..... 36
4.1 Basic NARS Genome ..... 36
4.1.1 Layout of a NARS Genome ..... 37
4.2 Experiment Domain #1: Wheeled Robot Animat Ecosystem ..... 46
4.2.1 Experiment Setup ..... 46
4.2.2 Evolutionary Algorithm ..... 49
4.2.3 Experiment 1.1: Evolution of Sensorimotor Contingencies ..... 53
4.2.4 Experiment 1.2: Evolution of Learning ..... 56
5 EVOLVING GENERALIZATION LEARNING AND INSTINCTS CONTAINING VARIABLES ..... 63
5.1 Extending the Basic Encoding with Variables ..... 63
5.2 Experiment Domain #2: Grid World ..... 66
5.2.1 Experiment Setup ..... 66
5.2.2 Evolutionary Algorithm ..... 68
5.2.3 Experiment 2.1: Evolution of Instincts Containing Variables ..... 77
5.2.4 Experiment 2.2: Evolution of Generalization Learning ..... 81
6 EVOLVING COMPOUND LEARNING AND INSTINCTS CONTAINING COMPOUNDS ..... 87
6.1 Extending the Basic Encoding with Compounds ..... 87
6.2 Experiment Domain #3: BlocksWorld ..... 89
6.2.1 Experiment Setup ..... 89
6.2.2 Evolutionary Algorithm ..... 91
6.2.3 Experiment 3.1: Evolution of Instincts Containing Compounds ..... 97
6.2.4 Experiment 3.2: Evolution of Compound Learning ..... 110
7 EVOLVING NARS CONTROL OF A COMPLEX ROBOT ..... 120
7.1 Experiment Domain #4: Soft Voxel Robot Locomotion ..... 121
7.1.1 Experiment Setup ..... 121

7.1.2 Evolutionary Algorithm……122
7.1.3 Experiment 4.1: Evolved Pure NARS Locomotion ……126
7.1.4 Experiment 4.2: Evolved Neuro-Symbolic NARS Locomotion……128
8 CONCLUSION ……133
BIBLIOGRAPHY ……139

## LIST OF TABLES


| Table | | Page |
|---|---|---|
| 4.1 | Wheeled Robot Ecosystem Sensory Set $\mathbb{S}$ | 49 |
| 4.2 | Wheeled Robot Ecosystem Motor Set $\mathbb{M}$ | 49 |
| 4.3 | NARS Personality Parameter Ranges in Wheeled Robot Ecosystem | 51 |
| 5.1 | Compact Grid World Sensory Set $\mathbb{S}$ | 69 |
| 5.2 | Compact Grid World Motor Set $\mathbb{M}$ | 70 |
| 5.3 | Expanded Grid World Sensory Set $\mathbb{S}$ for $5 \times 5$ Perception | 71 |
| 5.4 | Expanded Grid World Motor Set $\mathbb{M}$ for $5 \times 5$ Perception | 72 |
| 5.5 | NARS Personality Parameter Ranges in Grid World | 77 |
| 6.1 | Compact BlocksWorld Sensory Set $\mathbb{S}$ | 93 |
| 6.2 | Compact BlocksWorld Motor Set $\mathbb{M}$ | 93 |
| 6.3 | Full BlocksWorld Sensory Set $\mathbb{S}$ for 3-Block Puzzle | 94 |
| 6.4 | Full Blocksworld Motor Set $\mathbb{M}$ for 3-Block Puzzle | 94 |
| 6.5 | NARS Personality Parameter Ranges in BlocksWorld | 95 |
| 7.1 | Compact Soft Voxel Robot Locomotion Sensory Set $\mathbb{S}$ | 123 |
| 7.2 | Compact Soft Voxel Robot Locomotion Motor Set $\mathbb{M}$ | 124 |
| 7.3 | NARS Personality Parameter Ranges in Soft Voxel Robot Locomotion | 125 |
| 7.4 | Soft Voxel Robot Locomotion Neuro-Symbolic Sensory Set $\mathbb{S}$ | 130 |
| 7.5 | Soft Voxel Robot Locomotion Neuro-Symbolic Motor Set $\mathbb{M}$ | 130 |

## LIST OF FIGURES


Figure Page

2.1 NARS MNIST Experiment Spotlight of Attention ..... 21

3.1 NARS Overall System Architecture ..... 26

3.2 NARS Concept ..... 27

3.3 NARS Temporal Module ..... 33

4.1 Generic NARS Genome Example ..... 38

4.2 Generic NARS Genome Example Showing Sensorimotor Contingencies ..... 40

4.3 Conceptual Graphic of NARS Genome ..... 41

4.4 Tree Version of NARS Statement ..... 42

4.5 Wheeled Robot Screenshot ..... 46

4.6 Wheeled Robot Vision Sensor Screenshot ..... 47

4.7 Concrete NARS Genome Example ..... 50

4.8 Animat Reproductive Pools ..... 52

4.9 NARS Self-Sustained Population Metrics (Percent of Population Born By NARS) ..... 54

4.10 Wheeled Robot Evolved Instincts Metrics ..... 55

4.11 Wheeled Robot Ecosystem Instinct-Enabled Evolved Personality Parameters ..... 56

4.12 Wheeled Robot Ecosystem Fitness Scores under Different Experimental Conditions ..... 58

4.13 Wheeled Robot Ecosystem Best Learning-Enabled Agent Evolved Personality Parameters ..... 60

4.14 Wheeled Robot Ecosystem Screenshot ..... 61

5.1 Size of the Search Space of Basic Sensorimotor Contingencies ($|\mathbb{S}|^2|\mathbb{M}|$) ..... 63

5.2 Grid World Screenshot ..... 66

5.3 Grid World Screenshot of Various-Sized Perceptual Fields ..... 67

5.4 Grid World Variable Encoding vs. Basic Encoding Performance Comparison $(3 \times 3)$ ..... 78

5.5 Grid World Variable Encoding vs. Basic Encoding Performance Comparison $(5 \times 5)$ ..... 79

5.6 Grid World Genome Screenshot ..... 80

5.7 Grid World Variable Encoding vs. Basic Encoding Performance Comparison $(7 \times 7)$ ..... 81

5.8 Grid World Generalization vs. No-Generalization Performance Comparison $(3 \times 3)$ ..... 82

5.9 Grid World Generalization vs. No-Generalization Performance Comparison $(5 \times 5)$ ..... 83

5.10 Grid World Evolved Generalization Confidence Values ..... 84

5.11 Grid World Generalization vs. No-Generalization Performance Comparison $(7 \times 7)$ ..... 85

6.1 BlocksWorld Puzzle Sizes ..... 90

6.2 BlocksWorld Screenshot ..... 91

6.3 BlocksWorld Compound Encoding vs. Basic Encoding Performance Comparison (2-Block Puzzle) ..... 98

6.4 BlocksWorld Compound Encoding vs. Basic Encoding Performance Comparison (3-Block Puzzle) ..... 99

6.5 BlocksWorld Compound Encoding vs. Basic Encoding Performance Comparison (4-Block Puzzle) ..... 101

6.6 Size of the Search Space of Order-Independent Compound Terms Containing up to 3 Subterms ..... 102

6.7 Size of the Search Space of Order-Independent Compound Terms ..... 102

6.8 Size of the Search Space of Order-Dependent Compound Terms ..... 103

6.9 BlocksWorld Compound-and-Variable-Encoding vs. Base Encodings Performance Comparison (2-Block Puzzle) ..... 105

6.10 BlocksWorld Compound-and-Variable-Encoding vs. Base Encodings Performance Comparison (3-Block Puzzle) ..... 106

6.11 BlocksWorld Compound-and-Variable-Encoding vs. Base Encodings Performance Comparison (4-Block Puzzle) ....108

6.12 BlocksWorld Screenshot of Solved 4-Block Puzzle ....111

6.13 BlocksWorld Compound Learning vs. Basic Learning Performance Comparison (2-Block Puzzle) ....112

6.14 BlocksWorld Compound Learning vs. Basic Learning Performance Comparison (3-Block Puzzle) ....113

6.15 BlocksWorld Compound Learning vs. Basic Learning Performance Comparison (4-Block Puzzle) ....114

6.16 BlocksWorld Evolved Compound Learning vs. Evolved Compound Instincts Performance Comparison ....115

6.17 BlocksWorld Temporal Induction Screenshot ....117

6.18 BlocksWorld $N$-Compound Toggle Evolution ....118

7.1 Soft Voxel Robot Locomotion Screenshot ....121

7.2 Soft Voxel Robot Quadruped Screenshot ....122

7.3 Soft Voxel Robot Locomotion Extended-Encoding Comparison ....127

7.4 Evolved Neuro-Symbolic Architecture ....128

7.5 Soft Voxel Robot Locomotion Neuro-Symbolic vs. Extended Encoding Comparison ....131

# CHAPTER 1
# INTRODUCTION

Artificial intelligence has advanced rapidly through several distinct methodological traditions since its inception in the mid-20th century. Data-driven neural networks excel at high-dimensional perception and continuous control, symbolic reasoning systems provide explicit representations that support structured logic-based inference and human inspection. Reinforcement learning (RL) provides a general framework for learning behavior through interaction to maximize expected cumulative reward, and deep reinforcement learning uses deep neural networks as function approximators within RL (e.g., for policies and value functions). More recently, large language models (LLMs) have demonstrated impressive generality in natural-language interaction.

Despite these successes, building agents that are simultaneously general, effective, adaptive at runtime, strongly sample efficient, and transparent to human inspection remains an open challenge, especially in settings where knowledge and computational resources are limited. This dissertation explores an alternative AI optimization route: evolving explicit initial knowledge ("instincts") and meta-parameters in symbolic reasoning-system agents to improve their learning and performance. Here, instincts are the agent's evolved initial symbolic sensorimotor knowledge (seed beliefs/rules defined over a user-specified sensor–motor vocabulary), while meta-parameters are configurable control settings that shape architecture, inference, learning, and action selection during deployment. In this dissertation, a reasoning-system agent is an agent whose internal state is maintained as explicit symbolic beliefs and goals, and whose behavior is produced by auditable inference and decision procedures over that state.

A core trade-off appears repeatedly across modern AI: high performance from large-scale offline optimization versus continual online learning and inspectable decision-making under resource constraints. Many state-of-the-art neural systems can achieve high performance, but typically only through lengthy pre-training and large-scale optimization that rely

on extensive datasets and substantial compute. In addition, many deployed neural models are effectively "frozen" after training: their parameters do not change online, which limits their ability to learn continuously from new experience. This is especially problematic in open-ended environments where the agent must revise its beliefs and learn new strategies in real time to act according to new information, rather than only using knowledge obtained through offline pre-training. These systems also tend to be difficult to interpret: internal representations are distributed across weights and node activations, and explaining why a particular action was chosen often requires indirect analysis rather than direct inspection of explicit reasoning steps. On the other hand, traditional symbolic systems are explainable but can be brittle, expensive to knowledge-engineer, and difficult to scale to tasks requiring open-ended perception and adaptation.

For the purposes of this dissertation, pre-training refers to optimizing an agent's initial conditions prior to deployment, while runtime learning refers to within-lifetime online adaptation. Neural systems are typically strong in pre-training but limited in runtime learning, whereas NARS supports effective runtime learning but lacks an automatic pre-training methodology. This dissertation targets that pre-training gap by evolving initial symbolic "instincts" and meta-parameters that improve both immediate performance and subsequent learning at runtime, especially in sensorimotor settings such as multi-purpose robots operating in novel environments.

Reinforcement learning partially addresses adaptivity by updating the agent's policies according to the agent's experience. However, practical RL systems frequently struggle with fast, stable online learning in complex environments. Many RL successes rely on extensive interaction data, careful reward and hyperparameter shaping, and training procedures that are costly to run and fragile to distribution shifts (i.e., changing environment/data). This difficulty is compounded by commonly reported deep-RL failure modes such as unstable learning curves, low learning efficiency in sparse-reward settings, loss of

plasticity during continual learning, and catastrophic interference/forgetting of old knowledge when learning new knowledge [Ding and Dong, 2020, Dohare et al., 2024, Sutton, 2024, van de Ven et al., 2024, Qiu et al., 2025]. When task constraints require learning quickly from little data, or learning efficiently while deployed, purely reward-driven adaptation can be difficult to engineer and challenging to establish as robust under environment changes.

LLMs provide general-purpose abilities, though introduce a different set of tradeoffs. They can be valuable as interfaces, planners, or code and text generators, but they are not necessarily ideal as standalone controllers for grounded sensorimotor behaviors such as in robots, since high-quality behavioral descriptions, sensorimotor contingencies, and embodiment-specific dynamics are not richly represented in natural-language corpora. LLMs can help to develop robotics code itself, due to the extensive robotics code available online from which it learns. But for generating grounded, inspectable sensorimotor behavior under uncertainty, limited task-specific knowledge, and limited computational resources, particularly for embodied agents acting in open environments, LLMs may struggle. In their base form, LLMs are not optimized for continual online adaptation, persistent long-term memory, or real-time feedback control loops. They can produce ungrounded statements ("hallucinations"), struggle with logical guarantees [Arkoudas, 2023, Nezhurina et al., 2024], and do not inherently represent confidence/uncertainty calculations in their conclusions.

These limitations point to a gap in the state of the art: there is no standard pre-optimization methodology for optimizing AGI logical-reasoning-system agents that support both initialization of background knowledge and continual runtime learning, while still being general enough to apply across varied tasks and environments. This dissertation argues that a promising path toward closing this gap is to revisit a class of cognitive architectures with explicit internal representations, "logical reasoning systems", and to combine them with evolutionary optimization methods that can automatically discover

effective background knowledge and system configurations (e.g., thresholds, architectural components, control, and learning mechanisms) for logical reasoning agents.

This work focuses on the Non-Axiomatic Reasoning System (NARS), a logic-based reasoning architecture designed to operate under constraints of limited knowledge and limited computational resources by allowing real-time learning and incremental reasoning steps. The system operates under the Assumption of Insufficient Knowledge and Resources (AIKR): at any given time the agent's knowledge is incomplete and uncertain, and its computation and memory are bounded. Consequently, NARS is designed for incremental online revision of beliefs, uncertainty-aware inference, and resource allocation in reasoning and decision-making, enabling action even when conclusions are tentative.

NARS maintains explicit symbolic structures for concepts, beliefs, and goals, performs logical inference via auditable general-purpose reasoning rules, and represents uncertainty directly in its knowledge structures. These properties support AI interpretability and debugging, and they allow a human designer to inject goals and knowledge in a direct way. Importantly, NARS is also designed to learn at runtime by revising and extending its internal memory as it interacts with the world. It uses concept-centered memory representation [Wang and Thórisson, 2024], whereby different concepts are separated in different nodes yet linked to each other via connections, similar to the structure of a neural network but containing explicitly symbolic content, which can help mitigate the problem of catastrophic forgetting [Xu, 2025].

A central idea of this dissertation is that while learning adapts an individual agent's knowledge during its lifetime, evolution can be used to optimize agents' initial knowledge and learning configurations across a population. Evolution becomes especially useful when the design space is too large to be tractable for human consideration and the best initial structure for the agent is not obvious: rather than manually engineering instincts and meta-parameters, an evolutionary process can search for combinations that lead to agents that learn quickly and behave effectively after deployment. In this dissertation, NARS is treated

as an evolvable substrate, and a custom genetic encoding was developed to evolve initial knowledge ("instincts") and key meta-level parameters that shape learning and decision-making. This enables systematic experimentation on how different initial conditions of NARS influence its performance and runtime learning abilities across various environments.

Evolutionary algorithms (EAs) are general-purpose optimization methods which operate over populations of agents. Their fundamental mechanism is to generate populations of agent phenotypes from genomes (as specified by a genetic encoding), then to measure agent performance using a fitness function, and finally to generate new agent genomes, where contributions to the new population from the old population are proportional to the fitness scores achieved by the previous population of genomes/agents. Unlike gradient-based search methods, evolutionary search can work on discrete and symbolic problems, where gradients are not available. EAs can enable broader global exploration in some search landscapes than local gradient-based methods, which may be sensitive to local optima [Mustafaoglu et al., 2025, p.735], though often at the cost of higher evaluation and sample requirements. It has been shown that in some cases, EAs are more reliable in their ability to find good solutions and also capable of producing more robust solutions than reinforcement learning, even at a massive scale such as in fine-tuning billions of parameters in LLMs [Qiu et al., 2025].

## 1.1 Thesis Statement and Approach

This dissertation develops the first genetic encoding for NARS and demonstrates that evolutionary optimization of its belief priors and learning dynamics can improve domain-specific performance in sensorimotor applications. Concretely, it introduces a genetic encoding that enables an evolutionary algorithm to discover effective "genomes" composed of belief priors and meta-parameters governing the system's initial knowledge and runtime learning behavior. The resulting agents are evaluated empirically in 4 domains with respect to (1) task

performance arising from evolved runtime learning behavior, (2) task performance arising from evolved instincts, and (3) transparency of operation, assessed through inspection of reasoning traces and the explicit beliefs and goals underlying action selection.

## 1.2 Main Contributions

The contributions of this dissertation are:

- A novel genetic encoding and evolutionary methodology for NARS that makes it possible to evolve structured symbolic sensorimotor knowledge (instincts) and meta-level control parameters within a unified genome representation.
- Empirical evaluation across four case studies testing both optimization of instinct-based behavior and learning-based behavior, demonstrating how the proposed approach behaves in multiple domains, and clarifying what aspects generalize beyond any single environment.
- A transparency-oriented analysis showing how explicit symbolic knowledge representations, auditable inference steps, and uncertainty-aware reasoning support explainability, steering, and controlled adaptation, compared with common opaque alternatives.

## 1.3 Scope and Generality

Although the experiments in this dissertation use NARS as the reasoning system being explored, the broader aim is to inform the design of evolutionary methods used to optimize adaptive AI systems that 1.) maintain explicit internal representations, 2.) support online updates under resource constraints, 3.) expose controllable interfaces for goals and knowledge, and 4.) represent uncertainty explicitly. In this sense, NARS serves as a concrete

testbed for investigating how explicit representations and resource-bounded logical reasoning combined with evolution can improve system performance and continual adaptation that may also be sought in other symbolic or neuro-symbolic architectures.

A key scope constraint is that the knowledge-evolution component of the encoding is designed for sensorimotor (procedural) applications: it constructs candidate instincts by drawing from an explicit set of sensor and motor symbols. As a result, this part of the method is not intended to directly generate broad encyclopedic (declarative) world knowledge. However, the evolution of meta-level parameters that shape inference, learning dynamics, and decision thresholds is not inherently restricted to sensorimotor domains, and may be applicable to a wider range of NARS-based tasks.

## 1.4 Dissertation Outline

*Chapter 2* reviews relevant background in evolutionary algorithms, genetic encodings, multi-agent evolutionary ecosystems, and NARS. *Chapter 3* details the system architecture of the specific NARS system used in the experiments. *Chapter 4* introduces the proposed NARS genome representation and the evolutionary procedures used to generate and evaluate agents. *Chapters 4 through 7* present four case studies that test the approach across multiple domains, including environments that emphasize survival-like behaviors, planning and structured reasoning, and robotics-oriented control. Finally, *Chapter 8* summarizes findings, discusses limitations, and outlines directions for future work.

# CHAPTER 2
# LITERATURE REVIEW

## 2.1 AI Genetic Encodings and Evolutionary Algorithms

The evolutionary algorithm is a general optimization method for optimizing a parameterizable system by reproducing the system proportionally according to a fitness score. It has been argued that evolutionary algorithms can be a more robust and scalable alternative to reinforcement learning [Salimans et al., 2017] even for massive models like LLMs [Qiu et al., 2025]. The evolutionary algorithm is general in that it can optimize any type of parameterizable system, so long as the parameters of the system's phenotype can be "genetically encoded", i.e., encoded in a data structure that can undergo mutation (and optionally recombination) of its encoded values. An array of numbers is a simple example of a genetic encoding; numbers (corresponding to system parameters) are placed into the array, then "mutated" by perturbing and/or replacing the numbers to achieve an array containing new values (clamped to their respective valid ranges).

In order to enable the evolution of NARS, a genetic encoding had to be developed. The general idea was inspired by the following evolutionary methods and genetic encodings.

John Holland [Holland, 1975] introduced the traditional genetic algorithm in his book *Adaptation in Natural and Artificial Systems*. In it, he provided a formalization for any given *problem of adaptation* and its assigned *adaptive system* (evolutionary system): $\mathcal{A}, \Omega, \mathcal{T}, \mathcal{E}, \chi$ [Holland, 1975, p.28,35]. $\mathcal{A} = \{A_1, A_2, ...\}$ denotes the *set of all possible genomes* that can be discovered by an evolutionary algorithm. $\Omega = \{\omega_1, \omega_2, ...\}$ denotes the *set of genetic operators* used to modify genomes in mutation and recombination. $\mathcal{T} = \{\tau_1, \tau_2, ...\}$ denotes the *set of possible reproductive plans*, where each plan is a sequence of operators selected from $\Omega$ that traverse $\mathcal{A}$. $\mathcal{E} = \{E_1, E_2, ...\}$ denotes the *set of possible environments*. $\chi$ denotes a *criterion* to compare the possible plans from $\mathcal{T}$. In a genetic algorithm, for a given environment $E \in \mathcal{E}$, a reproductive plan $\tau \in \mathcal{T}$ made of genetic operators $\omega \in \Omega$ are used to modify a population of genomes from $\mathcal{A}$, where the

members of the population are ranked and reproduced according to $\chi$ (a fitness function). This results in a search process, where the goal is to find genomes from $\mathcal{A}$ which perform more optimally, as measured by $\chi$.

While Holland's standard genetic algorithm was defined on fixed-size strings of bits, the idea of artificial evolution was extended to other domains, such as variable-sized strings, fixed-sized and variable-sized neural networks, and computer programs.

[Hornby and Pollack, 2001] for example developed an encoding of variable-sized strings, where each string represented an ontogenetic program, and the characters of the string represented individual commands. Their evolved programs specified the ontogenesis (i.e., construction process) of both the body (morphogenesis) and the brain (neurogenesis). The characters of the string corresponded to commands executed in "turtle space", by which a robotic body made of rigid girders was constructed, and neurons were placed in the corresponding robotic parts according to the executed program command. This allowed them to perform AI robot brain-body evolution, for the locomotion task.

John Koza famously found that LISP-style computer programs could be evolved in the form of trees or "S-expressions" [Koza, 1992, Koza, 1994]. This can be done by representing programs as trees, where the root of any given tree (or subtree) is a computer operation (e.g., sum, multiply, IF statement, etc.) and the branches of the root correspond to the operation's argument. For example, the root of a tree might be "+", while the left subtree is a leaf containing "1" and the right subtree is a leaf containing "2", such that the program executes "1+2" and outputs "3". Such programs can be evolved by replacing a leaf or subtree with a new node or subtree, and by mutating the existing nodes of the tree to contain different values and operations, thus impacting the computation performed.

[Gruau, 1994] took a similar idea to Koza and implemented a special tree-style evolution called "cellular encoding". The tree represents a genetic program that is executed by "pre-neuronal cells", to perform neurogenesis. These cells can divide and perform various operations related to neurogenesis, until they reach the end of their program, resulting in

a finalized neural network. This is done by starting with a single "pre-neuronal cell" and having it execute the cellular program by traversing the tree data structure. By mutating the cellular program, the final neural network becomes modified, sometimes in complex ways depending on how high up on the tree the program was mutated. This algorithm was explored recently by evolving Hebbian neural networks for articulated robot locomotion in [Hahm, 2024b].

[de Garis, 1995] took an interesting approach to neural networks by implementing neurogenesis and neural processing in a 2D cellular automaton framework (the CAM-Brain Project). Thus the neural network and its signals are not stored in an abstract graph data structure like usual, but actually are set in a physical space. In this approach, a 3-cell wide trail exists in a cellular automaton grid, with some predefined handcoded cellular automaton rules. The trail is two sheaths encasing a signal pathway. To develop the network, the pathway and sheaths are grown according to growth signals sent down the pathway; once the brain is finished developing, neural signals are sent down this pathway instead. The sequence of growth signals for neurogenesis is evolved, and so the corresponding developed neural pathways become evolved too.

[Najarro and Risi, 2020] used evolutionary algorithms to evolve the Hebbian learning coefficients of a neural network for robot locomotion. The idea was that it is possible to evolve learning parameters that govern the runtime adaptation of connection weights, rather than to merely evolve the static connection weights themselves. It was found that robots with evolved Hebbian plasticity were able to adapt to novel situations at runtime. Specifically, their results showed that evolved Hebbian networks could overcome damage to the robot's leg and still succeed in locomotion, whereas the static-weighted networks could not.

One of the most powerful modern genetic encodings for neural networks is NeuroEvolution through Augmenting Topologies (aka NEAT) [Stanley and Miikkulainen, 2002]. Unlike traditional neural network genetic encodings of the past, which evolved neural connection weights but required fixed neural architectures, NEAT allows the neural architecture itself to evolve, in addition to the connection weights. NEAT does this by using mutations to add new connections and neurons to the network, as well as using mutations to delete them. This genetic encoding is powerful since the topology of a neural network greatly influences its computational abilities, so the ability to evolve and optimize the network architecture in addition to weights is a critical feature. NEAT is a "direct encoding", meaning the connections, neurons, and weights are all directly specified in the genome. This method was tested for soft voxel locomotion and object-seeking in [Hahm, 2025b], as well as for robot ecosystem survival and reproduction in [Hahm, 2025a, Hahm and Jay, 2026].

[Hahm and Wang, 2025] first specified the new genetic encoding of NARS. The encoding is a direct encoding, evolving Narsese statements corresponding to sensorimotor contingencies which allow the agent to navigate the world. The encoding also has a second, more traditional, part by which a vector of numbers corresponding to system metaparameters are encoded. Both the encoded statements and metaparameters can be recombinated and mutated. The idea to develop this encoding was directly inspired by symbolic string evolution [Hornby and Pollack, 2001], evolution of structures starting from sensor-motor neural sets [Stanley and Miikkulainen, 2002], and the evolution of learning parameters [Najarro and Risi, 2020].

## 2.2 Multi-Agent Ecosystems Combining Evolution and Learning

In order to tackle both milestones of "evolution" and, in a more limited way, "socialization", of NARS, evolutionary ecosystems containing multiple interacting agents were used in parts of this dissertation.

The field of artificial life (AL) attempts to digitally simulate certain aspects of natural lifeforms. In this way, AI is part of the AL field, since intelligence is observed in natural lifeforms, particularly animals, and AI attempts to simulate intelligence. While most AI projects approach intelligence in a substrate-agnostic way, focusing purely on the cognitive aspect itself, an AL approach to AI is more holistic, stressing substrate-specific factors like the agent's physical embodiment, its environment, its metabolism, and its social interactions. The ideal of a project combining AI and AL ideas simulates natural animals, such that the simulated animals evolve, are embodied with rich sensorymotor capabilities, and interact together in a shared ever-changing challenging environment; a full-blown artificial animal evolutionary ecosystem.

Full-fledged evolutionary AI ecosystems are somewhat rare projects, even moreso for those that evolve lifelong/runtime learning in addition to fixed behavior, but they are very important since they allow direct modeling of embodied cognition, agent interactions, goal-oriented behavior, adaptive learning, and evolution.

*PolyWorld* [Yaeger et al., 1994] was one of the first 3D simulations combining evolution and learning in an AI evolutionary ecosystem. Lifelong learning was permitted in agents in the form of the classic Hebb rule, though the learning rule itself was not evolved. The neural topologies of the brain, including connection weights, were evolved for fixed-morphology wheeled robots. The robots could reproduce and consume food resources scattered in the ecosystem. The robots had RGB vision and metabolism, could reproduce and fight with each other, and evolved various species-specific survival strategies such as cannibalism, patrolling the edges of the environment, and more. The early concept, creation, and visual presentation of this tool make it a landmark project in this research field.

[Weel et al., 2014] is a more modern example that evolved both AI brain and robotic body in a shared ecosystem environment. They allowed lifelong learning in the form of reinforcement learning (RL) with the RL PoWER algorithm. The learning abilities were made evolvable, such that the RL policy was evolved. The robots learned locomotion gaits

during their lifetime. The researchers analyzed learning ability by comparing an organism's locomotion speed at the beginning of its life to the locomotion speed at the end of its life. Similarly, they analyzed the evolution of learning ability by comparing locomotion speed at the beginning of the evolutionary process to the locomotion speed at the end of the process. While their experiments did find that learning took place during a given organism's lifetime, such that locomotion improved, the learning ability itself did not appear to improve over the course of evolution.

The idea of using evolutionary ecosystems in the context of artificial general intelligence (AGI) specifically has had some preliminary explorations and considerations by [Strannegård et al., 2021, Hahm, 2022].

*EcoTwin* [Strannegård et al., 2020, Strannegård et al., 2021] is a modern AI ecosystem combining evolution and learning. Lifelong learning in these works was tested in various forms. In [Strannegård et al., 2021], reinforcement learning was used for training a policy neural network in real-time according to reward signals. The reward signals were evolved, generated by an evolved "happiness" network, allowing the evolution of motivation and the adaptive learning process. In [Strannegård et al., 2020], symbolic patterns of Boolean variables were tested instead of regular neural networks. For example, in taste sensation, pattern $[\{sweet\}]$ represented sweet taste, and $[\{sweet, sour\}]$ was a compound representing sweet-sour taste. Various learning rules modified the symbolic patterns in the agent's memory set during its lifetime; some rules included, "pattern merge" which combined multiple patterns into one, "pattern concatenate" which appended one pattern to another (e.g., for sequence learning), and "pattern forget" which removed patterns from the memory set. The resulting evolved agents were able to learn locomotion, foraging, and even forms of language and arithmetic.

*ERAIASON* [Hahm, 2025a] is the simulator used in the first set of experiments in this dissertation (*Chapter 4*), developed specifically for 3D artificial animal research. It permits

a wide variety of configurations for automating 3D AI robot ecosystem research, by allowing the user to select (1) environment, (2) robotic embodiment, and (3) AI controller. Environment types that can be selected include flat worlds and cellular automaton voxel worlds [Hahm, 2023, Hahm, 2025a]. Robot types that can be selected include wheeled robots, articulated robots, and soft voxel robots. AI types that can be selected and evolved include regular sum-and-squash neural networks, continuous-time neural networks, neural networks with central pattern generators, and NARS. There are options to enable/disable real-time learning in the simulation. When enabled, real-time learning is permitted in the neural networks in the form of Hebbian ABCD learning (with the ABCD coefficients evolved), whereas learning in NARS works by permitting the formation of new concepts in its memory, such as those formed by temporal induction and syllogistic reasoning processes.

## 2.3 Non-Axiomatic Reasoning System (NARS)

### *2.3.1 NARS as a Next-Generation Reasoning System*

Non-axiomatic reasoning system, or NARS, was invented by Pei Wang in the mid-1980's as a general-purpose reasoning system that works under an assumption of insufficient knowledge and resources (AIKR) [Wang, 1986, Wang, 1995]. It is an adaptive reasoning system that grounds its conclusions on evidence obtained from its experience. The system employs logical principles to capture many fundamental properties of human cognition within a single unified model [Wang, 2019, Wang et al., 2022]. Its development was a response to the limitations of the existing computational models of reasoning at the time, as the traditional reasoning systems were deemed insufficient, by both Wang and the AI community at large, for modeling important properties of human cognition.

The limitations of traditional symbolic and logical reasoning systems, and the uncertainty in how to overcome those limitations, subsequently led to a shift in the overall field of AI from focusing mostly on symbolic systems to focusing mostly on neural networks

[Russell and Norvig, 2016, p.24]. In the common modern view, symbolic methods and neural networks are seen as complementary rather than competing [Russell and Norvig, 2016, p.25], though it has been argued by some (e.g., [Smolensky, 1988, Fodor and Pylyshyn, 1988]) that one or the other is the "proper" approach to computational cognition. Proponents of neural networks such as [Smolensky, 1988] contended that the *explicit* modeling of logic and compositional syntax is not necessary, and that neural networks process "subsymbols" which can theoretically approximate those desirable properties. Proponents of symbolic processing such as [Fodor and Pylyshyn, 1988] vehemently disagreed, countering that if the ultimate goal of the new neural network movement was to approximate functions of classical AI systems, such as compositionality and logic, then it is roundabout to use neural networks, which do not natively exhibit those properties. Though progress has been made in both domains, and research in the two schools occurs concurrently, their unification remains incomplete. The field of neuro-symbolic computing combines the two types of systems, but the different "languages" used by each type of system (i.e., node activations vs. compositional symbolic statements) can make it difficult for the two to interface with each other smoothly in a unified system.

NARS has many similarities and differences with other AI frameworks. For example, inductive logic programming (ILP) [Russell and Norvig, 2016, p.788] is a machine learning method that, like NARS, learns a logic program consisting of generalizations from positive examples, negative examples, and pre-existing knowledge. Unlike NARS, ILP systems typically use first-order logic, whereas NARS uses an evidence-grounded logic with truth values derived from accumulated experience. ILP is often treated as a "one-and-done" offline learning task where the method is run until a proper logic program is found, whereas NARS continuously incorporates new knowledge and performs reasoning without a fixed stopping condition, maintaining and revising multiple competing beliefs over time.

Bayesian reasoning and Bayesian networks [Russell and Norvig, 2016, p.806] are another class of methods that resemble NARS in that they support graded beliefs and update

them when new evidence arrives. However, NARS differs from Bayesian methods [Wang, 1993, Wang, 2001, Wang, 2004] in how it represents and revises uncertainty: NARS uses a two-dimensional truth value ⟨ frequency, confidence ⟩ intended to measure evidential support (with frequency reflecting the ratio of positive evidence to negative evidence, and confidence reflecting the total amount of evidence), rather than a single value representing probability. Consequently, NARS emphasizes evidence combination and belief revision under limited resources, while Bayesian networks rely on probabilistic inference that can become computationally expensive depending on the network structure and query.

**2.3.1.1 Non-Axiomatic Reasoning.** The NARS project directly answered some difficult problems in classical AI that were not overcome in the traditional systems, or at least not all in one single system. Some of these problems included ampliative (non-deductive) reasoning, handling uncertainty in truth value, accounting for the symbol grounding problem, handling limited resources and processing time, etc. Some traditional reasoning systems were able to handle these problems in isolation, but not altogether. The NARS approach says that many of these problems can be summed up as having one underlying cause: the systems in question are *axiomatic*.

In an axiomatic system, all the reasoning rules are truth-preserving, solutions to problems are derived usually through deduction on a set of axioms (i.e., statements considered as absolutely true), and the system usually cannot handle conflicts between truth values. In this sense, axiomatic logic is only useful in highly idealized (i.e., unrealistic) situations, where it is assumed the truth-value of any given statement is absolutely determinable, and that any potential conflicts between statements can be resolved by deriving statements which perfectly complement each other. In reality, no such guarantees can be made; often when we think we know something is absolutely true, we only find out later when presented with new evidence that it is not true in some special situations, or not really universally true at all.

In comparison, non-axiomatic logic (NAL), the evidence-based logic of NARS, is specially designed to be used in realistic situations. This is because in NAL, no knowledge is regarded as having a permanent, guaranteed, or final truth value, but rather all knowledge in the system can be challenged at any moment by new evidence. The amount of evidence can also be used to measure uncertainty, such that less evidence makes the system less certain and more evidence makes the system more certain. New evidence can be either derived by the system at runtime, or found incoming from the system's input channels. Furthermore, the system can use ampliative (i.e., non-deductive) reasoning, such as inductive and abductive reasoning, to derive knowledge outside the scope of the existing knowledge base.

The "evidence" component of NARS allows the system to derive conclusions based on subsets of relevant knowledge, rather than requiring the system to take all reasoning paths into account, which can strain system resources [Wang, 2025, p.7]. This allows the system to make partial (yet valid) inferences, as well as to handle conflicting beliefs, which allows it to perform logical inference in real-time with a constant time working cycle. This usefulness of evidence extends into the management of conflicting goals, such that evidence for one goal may be stronger than the evidence of a conflicting goal [Hahm et al., 2021]. Compare these features to traditional probabilistic reasoning systems, which often require recomputing (or incrementally updating) beliefs when new evidence is introduced. For example, in Bayesian networks, evidence updates can trigger belief/message propagation through affected portions of the graph (not in constant-time), consuming system resources and potentially inhibiting real-time responsiveness.

Due to the flexible logic of NAL, NARS systems can operate in application domains that traditional reasoning systems cannot, including real-time applications. This ability for real-time processing is important, since the semantics or meaning of the symbols/concepts of NARS are *grounded* in its sensorymotor experience. Therefore, a real-time robotics application with rich sensorymotor embodiment is highly desirable for testing NARS, since

rich embodiment provides it with a rich grounding for its concepts and perceptual experience.

These factors give NARS some benefits over neural networks, such as explainability, logic-based cognition, and progammability [Hahm and Suereth, 2025]. Still, despite NARS' improvements over traditional reasoning systems, there are some limitations. For complex applications, especially those involving sensory-motor processing, neural networks are the current preferred method, since they can learn arbitrary highly complex input-output mappings using backpropagation, whereas the theories for full-fledged self-organization of NARS is still under research and development. Nonetheless, if we can develop insights into NARS which improve its abilities in real-time and complex applications, eventually NARS could be a major contender against even state-of-the-art neural systems for cognitive tasks such as general-purpose reasoning (though NARS, being more about adaptability and general-purpose ability, will probably never be able to outperform special-purpose systems in their relevant domains).

[Hammer, 2021] worked directly in this direction, using insights from cognitive science to improve the NARS architecture and processing so as to increase the system's performance in event-based applications. While the original NARS system *OpenNARS* [Hammer et al., 2016] was designed mostly for processing declarative (encyclopedic) knowledge, the *OpenNARS-for-Applications* (ONA) system [Hammer and Lofthouse, 2020, Hammer, 2021] included various changes which made it better-suited for real-time applications and temporal learning. Some of these changes include using priority queue instead of Bag for task buffer, limiting the types of concepts the system stores, parallel processing of beliefs, and others [Hammer, 2021, p.28,52-57].

#### 2.3.1.2 The Symbol Grounding Problem.

A famous issue in traditional symbolic AI is the *symbol grounding problem* (or the problem of *intrinsic meaning*) [Searle, 1980, Harnad, 1990]: why should a formal system's internal symbols mean anything to the system itself?

In classical symbolic AI, inference operates over the syntax of expressions. The same rules are executed whether an atomic symbol is spelled "cat", 'dog", "apple", or "asdf", so long as they fit the syntax of the rules containing them. This makes it unclear how the system could be said to *understand* "cat", since any symbol could be placed as long as it fits the inference rule syntax. The apparent semantics seems to only come from the human reader of the symbol, who applies their own personal interpretation to the symbol, such as their personal understanding of "cat", "dog", or "apple".

One response has been to say that purely syntactic symbol manipulation is not enough, and that some symbols have to be grounded in the system's own interactions with the world, such as by linking atomic symbols to nonsymbolic (connectionist) representations, as in neurosymbolic AI [Harnad, 1990]. Under NARS' *experience-grounded semantics* [Wang, 1998, Wang, 2005], the system's "world" is given through its own experience stream (inputs) and its internally generated conclusions. Terms are not assumed to denote observer-independent, fixed concepts in the system; instead, a term's meaning is defined within the system by its role in the system's experience. Experience in NARS can indeed be sensorimotor, but also purely symbolic such as through language interaction.

Concretely, NARS represents its memory as a network of terms connected by learned and experienced relations. The meaning of a term is determined by its relations to other terms, characterized by the term's *extension* and *intension* in the system's memory [Wang, 1998, Wang, 2005]; the extension of a term $T$ includes the known specializations of $T$ and the intension includes the known generalizations of $T$ [Wang, 2005, p.10]. As new experience arrives and new conclusions are derived, the network changes, and so does the extension and intension of terms; therefore the meanings of terms are dynamic and history-dependent, rather than fixed once-and-for-all. Thus NARS solves grounding by not requiring an external interpretation for "cat", as in traditional systems; instead it treats "cat" (or whatever string is used) as meaningful due to the term's structured place in the system's experience. Importantly, this subjectivity does not make meaning arbitrary. Meanings of

terms can stabilize in the system as many regularities accumulate over time, and different NARS systems can partially converge towards a shared understanding of terms when they share the same environments and communicate with each other [Wang, 1998, p.23-24].

Finally, note that while the name of a term is arbitrary, its meaning to the system is not. NARS does not process terms merely as uninterpreted syntactical shapes; it processes them accordining to their learned relations and the evidence accumulated from the system's experience [Wang, 2005]. In this sense, a symbol's meaning is not a fixed interpretation supplied by an outside observer, but rather the symbol is grounded by its role within the system's interactions with its environment.

### *2.3.2 The Future of NARS*

The NARS project is still being explored to this day [Wang, 2006, Wang, 2013, Wang, 2025], with exciting new developments such as explorations in natural language processing [Kilic, 2015, Ireland, 2024, Isaev and Hammer, 2025, Li et al., 2025], machine emotion [Li, 2021], mathematics [Wang, 2022], computer vision [Wang et al., 2022, Hahm, 2024a], machine psychology [Johansson et al., 2024], real-time robotics [Hammer et al., 2024, Isaev, 2025], and more. The framework is gaining increasing attention for its unique and favorable properties such as explainability, runtime learning, resource-restricted processing abilities, and logic-based cognition.

While it is clear that certain aspects of NARS must be hardcoded by the system designer, such as parts of the logic and architecture, many aspects of the NARS system are configurable and parameterizable. For example, the size limit of the system's event buffer and the size of its long-term memory structures (see *Chapter 3*), the numeric values of the system's "personality parameters" (variables which modify the system's control process), the relevant background knowledge implanted in the system to bootstrap its runtime operation, etc. In practical NARS design and deployment, there has always been the questions

of “To which values should we set the meta-parameters for our application?”, “What architecture configuration should we set our NARS system in to optimize its abilities?”, and “What background knowledge should we provide the system to help it succeed?”. Normally, these factors are hand-picked by the human designer, but this can be an arduous manual guess-and-checking process. In fact, we could theoretically automatically optimize NARS configurations by automating the guess-and-check process, and we can do that by taking advantage of general-purpose optimization methods such as evolutionary algorithms.

Evolution of NARS has been considered as a future research endeavor, such as in the *Roadmap of NARS* [Wang, 2007], though to date the idea has not yet been tried comprehensively in multiple experimental conditions: “*In principle, a change in meta-level knowledge, that is, the system design, has the possibility of increasing the system's intelligence, though it is usually a dangerous experiment. For NARS, such changes are left for a separate evolution process, which is not carried out by the reasoning/learning mechanisms in the current design. The evolution of NARS will follow ideas similar to genetic algorithm...*” [Wang, 2007, p.13].

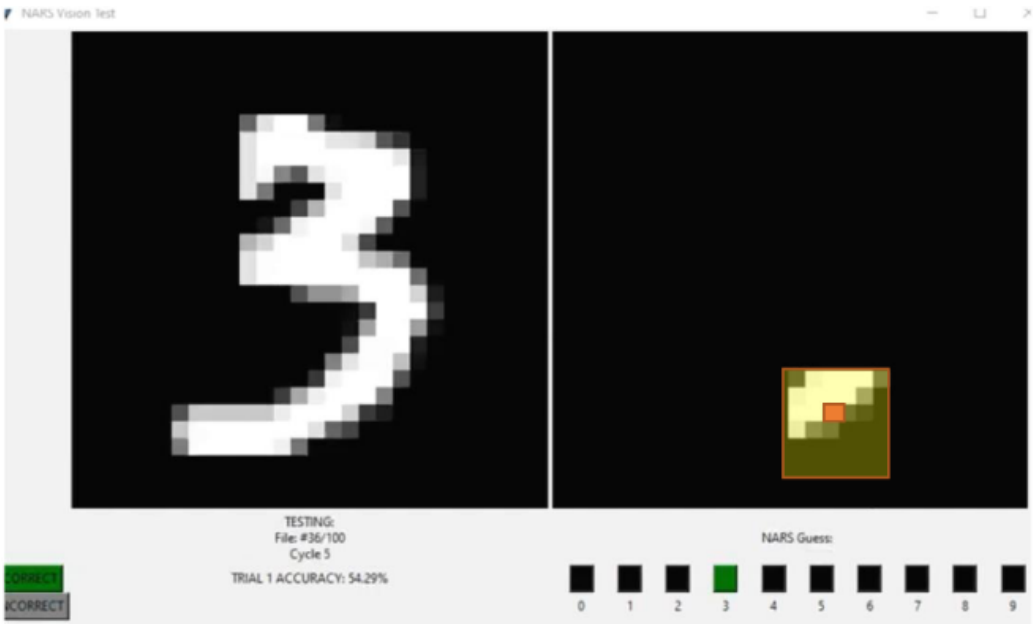


Figure 2.1: NARS MNIST Experiment Spotlight of Attention

There has been at least one preliminary usage of NARS evolution. In the experiments of [Wang et al., 2022, p.9–10], I used stochastic hill climbing (one parent, one mutated offspring, keep the better) to optimize NARS numeric meta-parameters for MNIST digit classification. There was no population or recombination, though there was random mutation. In that method, a single "best" NARS agent was kept, and a random mutation (perturbation) of that best/base agent was evaluated; if the mutated agent scored better than the best-so-far agent, it was used as the agent to mutate in future evaluations. A given NARS agent was trained on a training set of MNIST digit images, then evaluated against a test set. The NARS system was specially constructed to contain a "vision buffer" containing an image, and to use a "spotlight of attention" (see *Figure 2.1*) which created conjunctions and disjunctions from groups of pixel activations at runtime. The groups of pixels were formed into temporal associations with the digit label (e.g., $<pixel_1 \wedge pixel_2 \not\Rightarrow digit1>$) during the training phase, thus allowing NARS to predict digit labels from pixel activations during the testing phase. 5 metaparameters were evolved: $k$, $T$, and some metaparameters related to confidence values. NARS was tested on two tasks: memorization (to memorize exact images) and classification (to generalize from training set to test set of images). In the memorization tasks, NARS was able to perfectly memorize and recall 10 images of binary digits (0,1), and 10 images of digits (0 through 9). In the binary classification task, NARS was able to classify images of binary digits with an average of 97.78% accuracy (3 trials, with 30 training images and 90 test images per trial). In the digit classification task, NARS was able to classify images of digits with an average of 43.66% accuracy (3 trials, with 300 training images and 100 test images per trial).

A 2nd relevant milestone from the roadmap is "socialization", to test interactions of multiple NARS agents embodied in a shared environment, like a society, ecosystem, or culture of NARS systems: "*If multiple copies of NARS are implemented with different system parameters, hardware/software, and experience, they will form different beliefs and goals, and behave differently... These systems may cooperate in the solving of certain complicated*

*problems, as well as compete for certain resources*" [Wang, 2007, p.11]. Both milestones of evolving NARS and simulating embodied agents in multi-agent environments with resource competition are addressed in this dissertation.

# CHAPTER 3
# *NARS-UNITY* SYSTEM OVERVIEW

In order to evolve NARS, a configurable NARS system had to be developed which was capable of basic real-time sensorimotor processing. A basic NARS system has 3 high-level components: (1) a Buffer, a short-term memory structure where sensory events enter and derived tasks wait for processing, (2) a Memory, where data items are stored for long-term processing, and (3) an Inference Engine, which carries out the logical reasoning rules. To do sensorimotor processing, the NARS system has to be able to store information about transient events, and to process goals using knowledge ("sensorimotor contingencies") which describes how different sensory contexts relate to each other with respect to motor outputs. Here, my *NARS-Unity* project in C# was used, based on my *NARS-Python v0.3* architecture (originally written in Python) [Hahm, 2021b]. It is a basic NARS system which can perform simple sensorimotor processing, while leaving out many of the complex reasoning processes of NARS such as syllogistic reasoning. The purpose of using a simplified system was to have a straightforward NARS system on which to test the primary effects of evolution without confounding factors, such as syllogistic reasoning, which are not required for the real-time applications examined in the experiments.

The *NARS-Unity* C# implementation was created specifically for usage in the Unity simulator [Juliani et al., 2018], the simulator in which all the experiments of this dissertation were built. The *NARS-Unity* system is based on certain aspects of the overall abstract NARS theory, and takes inspiration from the other NARS implementations such as *OpenNARS* (written in Java) [Hammer et al., 2016] and *OpenNARS-for-Applications* (written in C) [Hammer and Lofthouse, 2020, Hammer, 2021]. This C# *NARS-Unity* system does not cover all components of the theoretical NARS model from [Wang, 2025], nor is it exactly the same as the other NARS implementations in Java and C, but instead it was built specifically with basic sensorimotor and goal processing as the focus, for use in real-time applications

and experiments. The specifics of the system architecture and control mechanism are as follows.

Depending on the experimental condition, learning could be enabled or disabled, to allow testing the effects of real-time learning on behavior vs. evolved-only behavior vs. their combination. Note that learning and evolution are different processes. Evolution occurs in populations of agents over multiple generations, while learning occurs in a single agent over its lifetime. Learning in NARS concretely refers to the use of logical reasoning to derive new concepts and concept links, especially ampliative reasoning (i.e., reasoning which "guesses" new concepts from existing knowledge like inductive and abductive reasoning, though the conclusions are not guaranteed to be valid as in deductive reasoning). When "learning was disabled", concretely this meant new concepts were not permitted to be formed in the agent's memory during the agent's lifetime. Instead, only backwards deduction and decomposition inference rules were permitted, for deriving goals using only existing knowledge with incoming sensory events.

### 3.1 Architecture

The architecture of the NARS system used in the experiments can be seen in *Figure 3.1*. NARS runs in an infinite loop, where each loop iteration is called a "working cycle". First, sensory tasks enter the system's Global Buffer. Then, NARS processes all the Tasks present in the Buffer at the beginning of the working cycle. If the Task is a Judgment (belief), then NARS simply stores it in the long-term Memory. If the Judgment happens to be a sensory event, then a reference is also passed to the Temporal Module, where it undergoes temporal induction to create new knowledge (aka learning). If the Task is a Goal, then it is either executed immediately if it is a motor operation, or otherwise a new Goal is derived that is expected to lead to the achieving of the original Goal.

The Global Buffer is an attention structure and short-term memory structure. In this implementation, it was a queue of Tasks/Sentences, which NARS read at each working

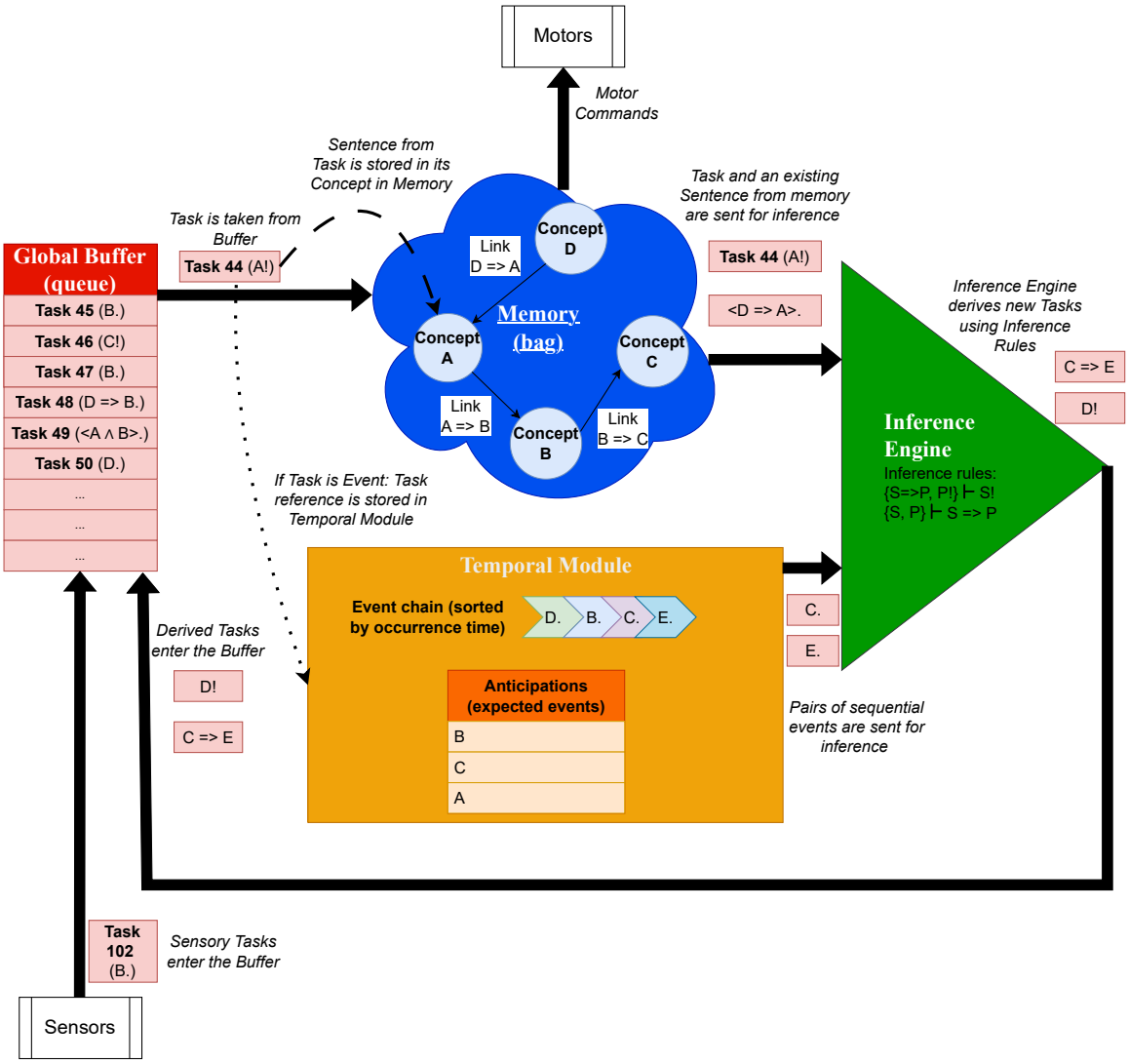


Figure 3.1: NARS Overall System Architecture

cycle. The Tasks were picked up by NARS and passed to the Temporal Module, for temporal processing, and the Memory, for storage and inference.

The Temporal Module stored events in order sorted by time. It performed temporal learning, by which new sensorimotor contingencies were learned from the events observed so far. For example, if an event $A$ occurred, after which NARS executed a motor operation $\Uparrow M$, then an event $B$ occurred, NARS combined them together to form a new statement $<A, \Uparrow M \Rightarrow B>$, which was then placed in the Buffer to be processed in the next working cycle.

The Temporal Module also held a list of Anticipations, which are events that NARS anticipates will happen in the near future. The Anticipations process allows NARS to "unlearn" incorrect statements; when an anticipated event failed to occur, a Judgment containing negative evidence is generated. For example, if NARS has a statement $<A \not\Rightarrow B>$, then when $A$ happens, NARS anticipates $B$ will happen within $W$ working cycles. If $B$ does occur, then the anticipation is removed from the list; if $B$ does not occur, then negative evidence (i.e., a statement with frequency zero) for $<A \not\Rightarrow B>$ is generated.

The Memory was a Bag (which is a probabilistic priority queue), which kept track of the Concepts and Sentences learned so far. It can be viewed as a network structure consisting of Concepts (nodes) and relations between concepts (links). Sentences were stored inside Concepts (see *Figure 3.2*, where they were stored for later processing.

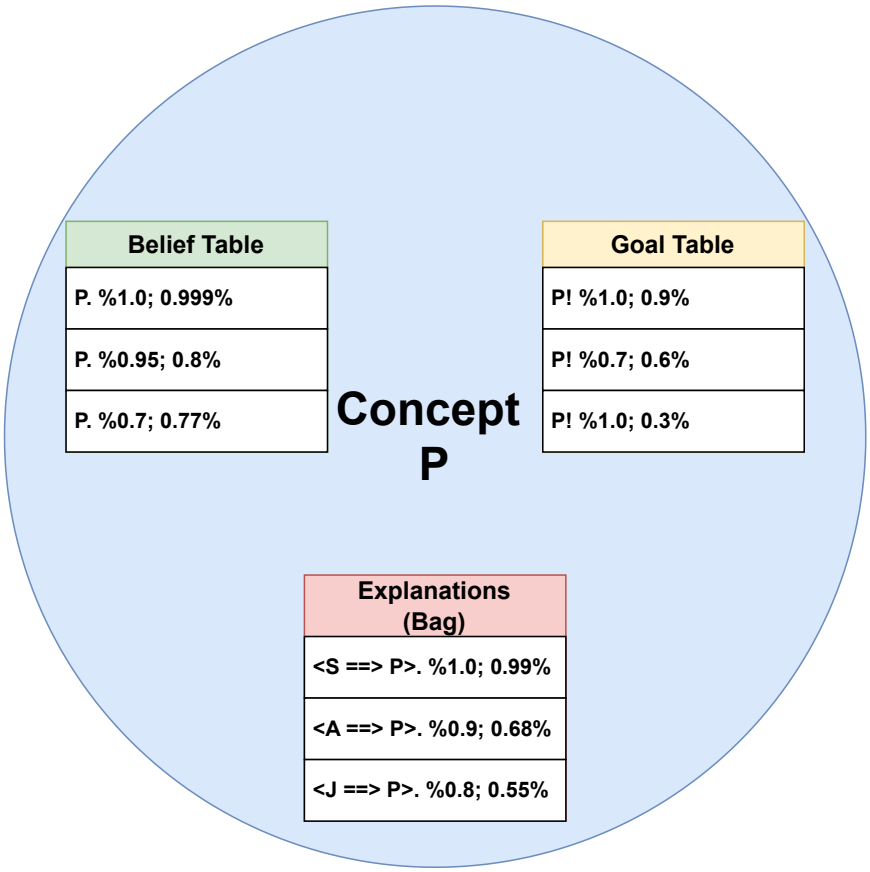


Figure 3.2: NARS Concept

The Inference Engine is a component which took in two Tasks/Sentences as its input, and output the derived results of reasoning on those two Tasks. There can be zero results, one result, or multiple results, depending on the content of the Tasks passed to the Inference Engine, and thus the corresponding reasoning rules used to process the Tasks. The derived Tasks were then put into the Buffer for processing in the next working cycle.

## 3.2 Control Mechanism

The control mechanism manages the actual operation of the NARS system over its architectural components. In this system, the control mechanism consisted of 3 high-level steps:

1. **Observe:** Process all Tasks present in the Global Buffer at the beginning of the working cycle

2. **Anticipate:** Update all Anticipations in the Temporal Module, by decreasing the countdown to the anticipated events

3. **Forget:** slightly decay the confidence values of all eternal beliefs in Memory. This was to allow the system to prune non-useful beliefs automatically, whereas the useful beliefs would continue to be reinforced if they were continually learned/experienced.

The Forget step used parallel processing (multithreading, 1 thread per concept) for a speed boost. During the Observe step, each Task in the Global Buffer was processed as follows.

### *3.2.1 Task Processing of Judgment*

NARS processed Tasks from the Global Buffer one at a time. A Task could contain either a judgment or goal. For example, there may be a judgment $J = <\{food\} \rightarrow [near]>.$, representing a sensory event that food is near, or $J$ can be another statement.

When a judgment $J.$ was processed (with its corresponding concept $C_J$):

1. If $J$. is an event, then put it in Temporal Module
2. If $J$. is an anticipated event, remove all Anticipations of $J$ (i.e., the event was "confirmed")
3. Place $J$. in the Belief Table of $C_J$
4. Probabilistically select an explanatory concept/belief related to $C_J$ according to its priority, and perform belief or goal inference with $J$. (e.g., if $J$ is an event, a concept $<E \Rightarrow J>$ would be selected allowing NARS to derive explanation $E$.)
5. Place the results of inference into the buffer for the next working cycle

The priority of explanatory belief was set to its "Expectation" $e$, a single value summarizing both the belief's truth and reliability computed using the confidence and frequency values of the belief:

$$e = c(f - 0.5) + 0.5$$

As $c$ approaches zero, the expectation of the belief becomes more uncertain, thus expectation approaches $0.5$:

$$\lim_{c \to 0} c(f - 0.5) + 0.5 = 0.5$$

Which means the system cannot be certain about the explanation one way or another.

As $c$ approaches one, the expectation of the belief becomes more certain, thus expectation approaches $f$:

$$\lim_{c \to 1} c(f - 0.5) + 0.5 = f$$

As events enter the system, references to the events were placed into the Temporal Moudle. Whenever the Temporal Module received a new event reference, it generated new statements via "temporal chaining", by which it performed temporal induction between each set of events to derive sensorimotor contingencies. Given an ordered list of events observed so far, the Temporal Module processed all such combinations of events that resulted in sentences of the form:

$$<S, \Uparrow M \Rightarrow P>.$$

This allows the system to learn, on the fly, which motor actions ($\Uparrow M$) to perform under which sensory contexts ($S$) so as to achieve certain results ($P$). A statement which supplies this sort of information is more generally referred to as a "sensorimotor contingency" [O'Neill and Ryan, 2001, p.491] or a "schema" [Drescher, 1993] in psychology.

### *3.2.2 Task Processing of Goal*

A Goal is an event which NARS desires to occur. It has a "desire value" indicating how strongly NARS desires the event to occur. In this version of NARS, Goals themselves were also treated as events, such that goals were temporary and only had high desire-values at specific times. For instinctual goals, they were fed into NARS at every working cycle to ensure their activation and processing.

For example, there may be an instinctual goal $G = <\{ENERGY\} \rightarrow [FULL]>!$, representing a desire to keep energy levels full, and this was given to NARS at each working cycle to maintain high activation. Non-instinctual goals had to be continuously derived during goal inference in order to maintain high activation.

When a goal $G!$ was processed (with its corresponding concept $C_G$):

1. Place $G!$ in Desire Table of $C_G$.

2. Take highest value desire $D!$ from Desire Table of $C_G$.

3. *Decision-making rule*: stop pursuing $D$! (go to last step) if the Expectation of its desire value is below decision-making threshold $T$

4. If the most confident belief in the Belief Table of $C_G$ says $D$ is already achieved, go to last step

5. If $D$ is an Operation, execute $D$! and go to last step.

6. If $D$ is a conjunction $(A, B)$!:

   (a) simplify with belief $A$ if its Expectation is "true" (above 0.6) to get $B$!, otherwise if $A$ is not true, derive a goal for $A$!

   (b) go to last step

7. If $D$ is a first-order statement, process with an explanation $E \models\!\Rightarrow D$ using backwards inference to get a precondition goal $E$!. If no such explanatory belief exists, motor babble.

8. Put desire $D$ back into Desire Table

### *3.2.3 Sentence Table*

Each concept in NARS contained 2 instances of a data structure called Sentence Table or simply Table, for storing judgments ("Belief Table") and goals ("Desire Table"). A Table was a sorted list of sentences, where the sentences were sorted by their confidence value, such that the highest-confidence sentence was available for quick taking. A Sentence Table's capacity was restricted to a certain number, which was configurable by the user or, in this case, evolution.

Normally, each sentence has an "evidential basis", which is a list of the IDs of the sentences that were used to derive the sentence. When two sentences have an overlap in their evidential basis, they cannot be processed together. This prevents circular reasoning, which occurs when derived events are processed with the premises used to derive it, resulting in

the same evidence being counted multiple times (which can result in redundant and invalid inference results). Here, evidential basis was ignored for events specifically since they were very short-lived and were meant to act more like transient "causal activations" rather than persistent/eternal beliefs which required stricter validation. The purpose of this design decision was to allow events derived from the same sensory event but different schema beliefs to be merged together, such that if multiple schemas suggested the same derived event then the truth-value of the event came out stronger.

If a concept represented an event, the sentence's confidence value decayed over time, and only 1 sentence was permitted in the Table. In such a case, if the Table was empty, the event being processed was simply placed in the Table. If the Table had already contained one event, then when a new event was processed, the two were merged together using the Revision Rule.

The other case of putting a sentence in a Table was when the sentence was "eternal" (i.e., not an event), meaning its confidence value did not change based on the current time. In that case, firstly, the sentence was added directly to the Table. Then it was Revised with all sentences in the Table for which there was no evidential overlap. Rather than to add all results of these Revisions to the Buffer, all results were directly placed into the Table. Afterwards, the lowest-confidence results were pruned to ensure the Table stayed within its capacity.

### 3.3 Temporal Learning at Runtime

NARS was able to learn at runtime via temporal learning. When an event was processed from the Global Buffer, a reference to the event was also passed into a data structure called the Temporal Module (see *Figure 3.3*). Only first-order events (non-compound) were permitted into the Temporal Module.

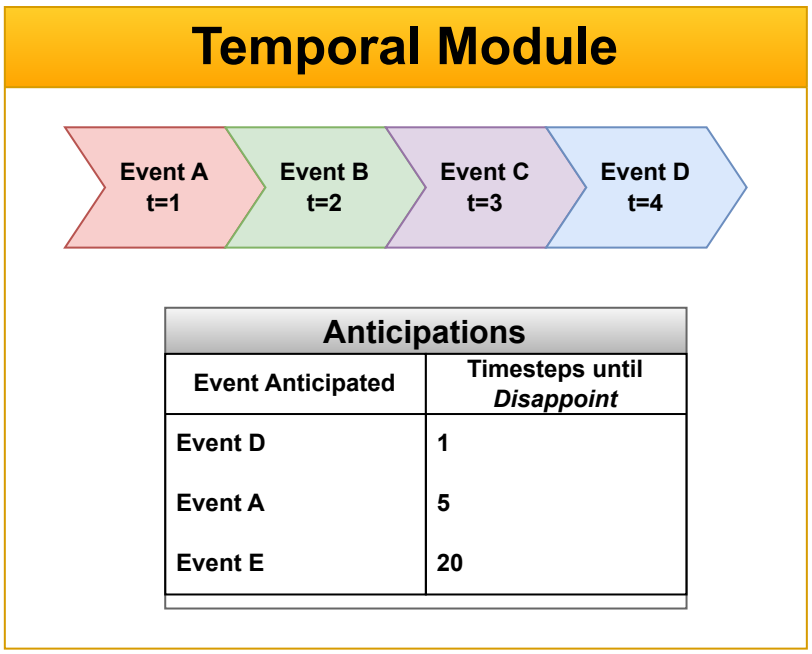


Figure 3.3: NARS Temporal Module

NARS learned new sensorimotor contingencies by using the temporal induction rule on sets of 3 sequentially-occuring events $S$, $\Uparrow M$, and $P$, to form a new sensorimotor contingency $<S, \Uparrow M \ /\!\!\Rightarrow \ P>$. Judgments formed this way were then placed into the Global Buffer to be absorbed into the system's memory.

An Anticipation was simply an event that NARS expected to happen, according to a belief formed by the system's experience during temporal learning; if it did not occur, then we say NARS was "Disappointed". For example, using $<S, \Uparrow M \ /\!\!\Rightarrow \ P>$, if event $S$ was true and NARS executed $\Uparrow M$, then NARS would anticipate event $P$ to occur in a certain number of working cycles (here, this number was evolved). When an anticipation failed to occur, NARS generated negative evidence for the explanatory belief used to derive the Anticipated event, thus decreasing the statement's truth-value and making it less likely to be used in future goal derivations and Anticipations. In other words, whereas temporal induction allowed the system to form new beliefs and strengthen existing beliefs by introducing *positive evidence*, anticipations allowed for optimizing the system's learned beliefs by introducing *negative evidence*.

## 3.4 Personality Parameters (Meta-Level Parameters)

Though the overall structure of the architecture and control mechanism above were held fixed, importantly there were configurable meta-level parameters which impacted the fixed structure of the control mechanism and architecture before NARS deployment. These meta-level parameters could not necessarily be learned by the system at runtime from its reasoning rules and experience, since they were either used to direct the reasoning process itself or determine aspects of the system architecture (thus, they are called "meta-level"). Modifying these values at runtime can make the system unstable and produce inconsistencies in its behavior, thus they are held fixed once the system is deployed.

The meta-level parameters (aka "personality parameters") were evolved in the experiments. There were 9 evolvable base personality parameters:

- **Evidential horizon ($k$):** determined the weight of a unit of evidence in NAL, and was used in truth value calculations.
- **Decision-Making threshold ($T$):** determined how strong a goal must be for NARS to pursue it.
- **Anticipation window size ($W$):** determined how many working cycles NARS will wait before becoming Disappointed at an Anticipation.
- **Forgetting rate ($F$):** determined the half-life, in working cycles, of eternal beliefs.
- **Temporal Module capacity ($C_{temp}$):** determined how many events were stored in the Temporal Module for temporal chaining
- **Table capacity ($C_{table}$):** determined how many eternal sentences were allowed in a Table.
- **Evidential base length ($E$):** determined the maximum length of evidence tracked in the evidential basis.

- **Time projection for belief events ($\mathcal{T}_J$):** determined the half-life of the confidence value of judgment events.
- **Time projection for goal events ($\mathcal{T}_G$):** determined the half-life of the confidence value of goal events.

# CHAPTER 4
# ENCODING NARS PERSONALITY AND BASIC INSTINCTS

## 4.1 Basic NARS Genome

In order to evolve NARS, a genetic encoding must be developed that enables evolution. Concretely, this means a genome representation that can be used to generate a NARS "phenotype", as well as perform genome mutation and (optionally) recombination.

There are three main aspects to any given NARS system at the moment of its deployment that we must consider for encoding.

The first aspect is the NARS system itself, including its architecture, logical inference rules, control mechanism, and configurable meta-level parameters (sometimes called NARS' "personality parameters"). This encompasses a large number of items, and modifying these may have a major impact on the system's capabilities and behavior, potentially resulting in unstable NARS systems, so it is important to choose these carefully to minimize the potential of genomes which produce invalid NARS systems.

The second aspect is the background knowledge inside the system's memory. Although NARS systems are designed to learn at runtime, from previous experiments (e.g., [Hahm, 2021a, Hahm et al., 2023] and others) we have found it useful to implant NARS with some background knowledge to bootstrap desirable behavior. Traditionally, when working with NARS in a specific problem domain, the human designer would determine this background knowledge manually by considering the problem domain and how NARS is being deployed. However, by using evolution and a genome, we can automate the production, discovery, implantation, and testing of background knowledge aka "instincts", and allow the evolutionary algorithm to find optimal knowledge for us. The reasoning/learning process can revise this knowledge at runtime according to the system's lived experience, in case the situation changes at runtime.

Finally, the third aspect of NARS to consider is the system's seed goals. The system directs it operations according to its goals; the goals directly drive behavior [Wang, 2025,

p.145] [Hahm et al., 2021] . The system uses its goals paired with its knowledge in logical inference to derive various goals which themselves can be used in later goal inference. These goal derivations eventually result in a causal chain of actionable steps, starting from a realizable goal all the way to the final "seed" goal to be completed. This chain provides NARS with a sequence of steps or operations, a plan, to execute in order to achieve its goals. In practice, the initial goals of the system are often set by the human so as to control the agent for our needs. In theory, goals, too, can be evolved like "instincts", by pulling from the sensory set $\mathbb{S}$, though this was not experimentally tested in this dissertation work.

### *4.1.1 Layout of a NARS Genome*

In order to cover all three main components, a genome for NARS (in this encoding) has 3 high-level components: $\beta$, $\Gamma$, and $\Pi$.

- $\beta$, a set of beliefs implanted at birth, providing instinctual procedural knowledge.
- $\Gamma$, a set of goals fed to the system during its lifetime, driving behavior.
- $\Pi$, a fixed-sized array of floats for the Personality Parameters, modulating the Control Mechanism and Architecture of NARS.

$\beta$ holds procedural knowledge that gives NARS a head start in knowing how to navigate the world. It is a variable-sized set that can be empty, or grow to contain many pieces of knowledge. Each element of $\beta$ is a Narsese *judgment*, a type of Narsese sentence representing world knowledge. The judgment has a dual-valued *truth-value* consisting of frequency ($f$), essentially the judgment's "truthiness", and confidence ($c$), representing by how much evidence the system weighs the judgment.

$\Gamma$ holds the seed goals that are fed into the system throughout its lifetime to drive the system's behavior. It is like the system's "lifeforce", as the goals and derived goals directly drive the system's actions. Each element of $\Gamma$ is a Narsese *goal*, a type of Narsese sentence representing motivation and desired states. The statement of a goal describes an

event that NARS will attempt to realize. Each goal has a dual-valued *desire-value* consisting of frequency ($f$), essentially describing how desirable the goal is, and confidence ($c$), representing by how much evidence the system weighs the goal. A goal with a high frequency is highly desired by NARS, and if it has high confidence then it means NARS is quite certain it wants to achieve the goal. Conversely, if a goal has low frequency, then NARS does not desire it very much, and a low confidence means NARS is unsure about whether the goal is desired or not. $\Gamma$ is also variable-sized and can be evolved similarly to $\beta$, though for application purposes it may be beneficial to keep $\Gamma$ unchanging and populated with the end-goals desired for the application.

$\Pi$ modulates the system's control mechanism and architecture. In this case, it is very simple, an array of numbers governing architecture capacities, forgetting rates, learning mechanism, etc. Therefore, each element of $\Pi$ is a number (float or integer) corresponding to some aspect of the system's design. The value of the number directly influences its corresponding design component, e.g., a buffer capacity determines the maximum number of sentences allowed in the buffer.

A generic version of a NARS genome can be seen in *Figure 4.1*:

$\beta$:
$J_1$. $\langle f_1, c_1 \rangle$
$J_2$. $\langle f_2, c_2 \rangle$
$J_3$. $\langle f_3, c_3 \rangle$
$J_4$. $\langle f_4, c_4 \rangle$

$\Gamma$:
$G_1$! $\langle f_5, c_5 \rangle$
$G_2$! $\langle f_6, c_6 \rangle$

$\Pi$:
$k$
$T$

Figure 4.1: Generic NARS Genome Example

Theoretically, the simplest possible NARS genetic encoding could leave out $\beta$ entirely, which would give NARS a blank initial memory. $\Gamma$ is a set providing goals, which are needed to drive the NARS' system's motor actions, though $\Gamma$ can also theoretically be omitted as long as the user supplies goals manually or at runtime. Only $\Pi$ is technically absolutely necessary to specify a NARS agent, since $\Pi$ sets the values determining the NARS architecture and control mechanism. A NARS agent specified by $\Pi$ alone would start with a completely empty memory.

However, without any built-in knowledge from $\beta$, every NARS system will be forced to learn everything about the world from scratch, which is more time-consuming, resource-intensive, error-prone, etc. than using instinctual knowledge acquired at "birth" which bootstraps good behavior. Therefore, it is preferable to include $\beta$ in any NARS agent, since it might evolve to contain useful knowledge that gives the NARS agent a head-start in its life.

**4.1.1.1 Evolving Sensorimotor Contingencies.** NARS, in its current stage, primarily operates in sensorimotor applications using 3-component "sensorimotor contingencies" or "schemas". These are structured rules that describe how the sensory situation changes according to the execution of motor actions [O'Regan and Noë, 2001, p.941]. In other words, they describe what actions to take under which contexts so as to achieve certain results [Drescher, 1993, p.129].

Sensorimotor contingencies or schemas in NARS take the form of:

$$<S, \Uparrow M \not\Rightarrow P>.$$

where $\{S, P\}$ are sensations and $\Uparrow M$ is a motor operation.

NARS can create statements of this form at runtime using temporal learning, or these statements can be implanted beforehand by a human user or algorithm.

In order to evolve rules of this form, the evolutionary algorithm needs access to the set of possible sensations $\mathbb{S}$ that NARS can experience, and also to the set of possible motor

operations $\mathbb{M}$ that NARS can execute. That way, the algorithm can pull $\{S, P\} \in \mathbb{S}$ and $\Uparrow M \in \mathbb{M}$ during the process of mutation. These sets can be pulled from to swap out terms within existing knowledge in the genome, and also to generate brand new sentences in the genome.

$\beta$:
$J_1 = <S_6, \Uparrow M_1 \not\Rightarrow S_2>. \langle f_1, c_1 \rangle$
$J_2 = <S_1, \Uparrow M_2 \not\Rightarrow S_3>. \langle f_2, c_2 \rangle$
$J_3 = <S_4, \Uparrow M_3 \not\Rightarrow S_3>. \langle f_3, c_3 \rangle$
$J_4 = <S_1, \Uparrow M_3 \not\Rightarrow S_7>. \langle f_4, c_4 \rangle$

$\Gamma$:
$G_1 = S_3! \langle f_5, c_5 \rangle$
$G_2 = S_7! \langle f_6, c_6 \rangle$

$\Pi$:
$k$
$T$

Figure 4.2: Generic NARS Genome Example Showing Sensorimotor Contingencies

An example of a NARS genome with sensorimotor contingencies can be seen in *Figure 4.2*. In this example, $S_1$ (and $S_2$, $S_3$, etc.) is a statement from $\mathbb{S}$, representing a possible sensory event that NARS could detect. For example, let's consider a simple scenario where NARS is an environment containing a door, and next to the door is a red button that opens the door when pressed. The statement $S_1$ might represent a visual event such as "*a red button is seen*", whereas $S_7$ might represent a visual event "*a door was seen opening*". The NARS here has a goal $G_2 = S_7!$ from its genome ($\Gamma$), which in this case would mean "*NARS wants to see a door opening*". Using the evolved knowledge set from $\beta$, the NARS can derive a (precondition) goal $G_d$ for goal $G_2$ using a backward deduction rule:

$$\{G_2, J_4\} \vdash$$
$$G_d = <S_1, \Uparrow M_3>!$$

Consequently, when NARS processes this derived goal $G_d$ while detecting sensory event $S_1$ ("*red button is seen*"), it will "instinctually" believe that it should activate motor operation $\Uparrow M_3$. The choice of motor operation is evolved, so it could be any operation from $\mathbb{M}$. Consequently, it might correspond to a motor operation that actually works in the situation, e.g., if there is an operation available like "*press button*", which will in fact cause the door to open. Or, conversely, the motor operation might correspond to something entirely wrong, like "*spin in a circle*", which will not cause the door to open. Over time due to the evolutionary process, those beliefs which are "correct" or useful for reproducing in the given environment will persist in the NARS population, whereas those beliefs which are "incorrect" or not useful for reproducing in the given environment will not.

A graphical visualization of an example genome can be seen in *Figure 4.3*.

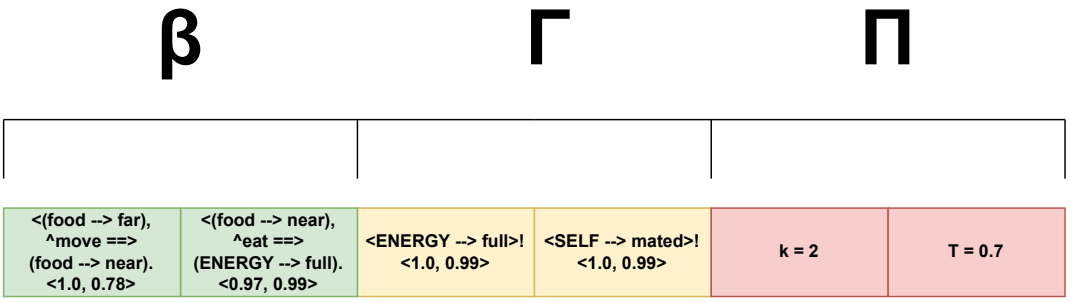


Figure 4.3: Conceptual Graphic of NARS Genome

Notably, the method can be viewed as similar to tree-based genetic programming with trees, since NARS statements can be represented in the form of trees, e.g., with the subject term in the left subtree, the predicate term in the right subtree, and the statement copula in the root node (see *Figure 4.4*).

Therefore, $\beta$ can be interpreted as a forest of trees, where each tree represents a sensorimotor contingency, and a sensorimotor contingency is essentially a program that influences how NARS will behave. In this work, however, the method was not implemented explicitly in terms of trees; instead, for simplicity, sensorimotor contingencies were encoded as tuples (i.e., a linearized representation).

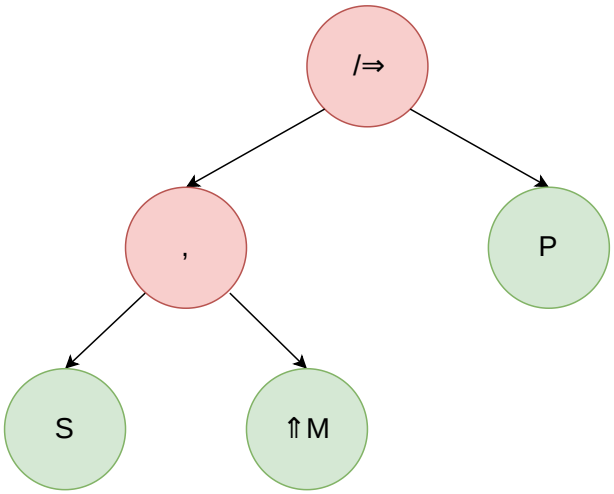


Figure 4.4: Tree Version of NARS Statement

**4.1.1.2 Genetic operators.** A set of genetic operators $\Omega$ was defined for mutation and crossover as:

**MUTATION**:

- *Add sentence*:
  - **Add belief**: Randomly select two statements $S$ and $P$ from $\mathbb{S}$, and a motor op $M$ from $\mathbb{M}$. Then, add a belief $<S, \Uparrow M \not\Rightarrow P>$. to $\beta$.
  - **Add goal**: Randomly select a statement $G$ from $\mathbb{S}$, and add a goal $G!$ to $\Gamma$.
- *Modify sentence*
  - **Modify belief**: Randomly select a belief $<S, \Uparrow M \not\Rightarrow P>$. from $\beta$. Then, randomly select $S$, $M$, or $P$. If $S$ or $P$ was chosen, replace it with a statement randomly selected from $\mathbb{S}$. Otherwise, if $M$ was selected, replace it with a statement randomly selected from $\mathbb{M}$.
  - **Modify goal**: Randomly select a goal $G!$ from $\Gamma$. Then, replace $G$ with a statement randomly selected from $\mathbb{S}$.
- *Remove sentence*

- **Remove belief**: Delete a random element from $\beta$.
- **Remove goal**: Delete a random element from $\Gamma$.

- *Modify evidential value*: When modifying $f$, the chance to add a small value randomly selected from $[-0.1, 0.1]$ is $X_e$, and the chance to set it as a random value from $[0, 1]$ is 1-$X_e$. Alternatively to selecting uniform randomly from the fixed range, select from a Gaussian distribution with an appropriate standard deviation (e.g., 0.1 or 0.2). Finally, clamp the value to its valid range ($[0, 1]$ for $f$, and $(0, 1)$ for $c$). This method which perturbs the value by a small amount or replaces it with a new value is inspired by established neural network evolution methods [Stanley and Miikkulainen, 2002, p.111].

- *Modify Personality Parameter*: When modifying a parameter $p \in \Pi$, the chance to add a small value randomly selected from a Gaussian distribution with mean zero and standard deviation $0.1(R_H - R_L)$ is $X_p$, and the chance to set it as a random value from the parameter's range $[R_L, R_H]$ is 1-$X_p$.

Note that the mutation operators for goals ($\Gamma$) are included for completeness, but they were not used in the experiments.

**CROSSOVER**:

The genome crossover was component-wise (i.e., crossover $\beta_1$ with $\beta_2$, $\Gamma_1$ with $\Gamma_2$, and $\Pi_1$ with $\Pi_2$).

- *Uniform Set Selection*: Align the two sets as though they are arrays/vectors, and iterate over them at the same time. The genomes may differ in size, in which case the end of the shorter genome is padded with empty slots. For each index, randomly select 1 parent and choose the element from their set. If only 1 parent has an element while the other parent has an "empty" slot, have a 50% chance to keep the non-empty element. This is similar to uniform crossover except that the elements of the vectors

are aligned. With this method, the number of elements in the offspring genome will be roughly equivalent to the average length of the parents'.

While the above crossover operator is the specific one used in these experiments, it is not the only possible way. There are many other possibilities, which were not explored here but could easily be explored in the future. For example, rather than using a variable-sized set for $\beta$ and $\Gamma$ and padding the shorter genome with empty slots, one could use a fixed-length array as the data structure of the genome. In that case, for crossover, the empty slots would be interspersed, rather than all at the end. From there, you would perform a traditional array crossover. Such a method may result in different recombination dynamics, and a different search trajectory.

Alternatively to the component-wise crossover version ($\beta$, $\Gamma$, etc.) stated above, the whole genome could be treated as a single long vector. This would allow a more traditional genetic algorithm vector crossover like single-point crossover, double-point crossover, etc. on the entire genome. Since each component of the genome ($\beta$, $\Gamma$, etc.) can be made to have a maximum fixed size, the genome can be treated as a single long fixed-size vector concatenating $\beta$, $\Gamma$, and $\Pi$, potentially containing empty elements. In this way, if $\beta$ can have up to $N_\beta$ elements, $\Gamma$ can have up to size $N_\Gamma$ elements, and $\Pi$ has $N_\Pi$ elements, then the resulting genome vector would be of size $N_\beta + N_\Gamma + N_\Pi$ (where some elements of $\beta$ and $\Gamma$ may be empty).

There is also the potential to try crossover at the level of individual beliefs, such as swapping subjects/predicates between 2 beliefs, though these more complicated situations are only mentioned here for consideration, and left for future research endeavors.

#### 4.1.1.3 Evolving Metaparameters ("Personality").

The values of the evolved personality parameters, such as evidential horizon $k$ and desire threshold $T$ (aka "Cautiousness") affect the system's control operations. They

$k$ impacts system weighting of confidence and evidence. If there is a certain amount of evidence $w$ for a statement, the confidence $c$ of the statement's truth-value is calculated by:

$$c = \frac{w}{w + k}$$

$k$ is in the denominator, meaning that confidence $c$ decreases as evidential horizon $k$ increases.

Thus, a low $k$ means the system believes things very quickly (i.e., it needs less evidence $w$ to achieve a high confidence $c$), whereas a high $k$ means it will take much longer to convince the system of something (i.e., it needs more evidence $w$ to achieve a high confidence $c$).

$T$ is a threshold used by the system to decide whether or not to pursue a goal. If a goal's desirability $d$ is greater than $T$ (or equal to $T$), then the system will pursue it. This is known as the Decision-Making Rule:

$$T \geq d$$

A low value of $T$ means the system is very impulsive and not cautious (i.e., a goal only needs relatively low desirability for the system to pursue it), whereas a high value of $T$ means the system is very timid and cautious (i.e., a goal needs to have a very high desirability for the system to pursue it).

A goal's desirability $d$ is simply a number which combines frequency and confidence. It is computed by the "expectation" formula:

$$d = c(f - 0.5) + 0.5$$

As $c$ approaches zero, desirability of the goal becomes more uncertain, thus desirability approaches $0.5$:

$$\lim_{c \to 0} c(f - 0.5) + 0.5 = 0.5$$

Which means "*whether the goal is desirable or not is uncertain*".

As $c$ approaches one, desirability of the goal becomes more certain, thus desirability approaches $f$:

$$\lim_{c \to 1} c(f - 0.5) + 0.5 = f$$

Which means "*it is certain the goal has desirability $f$*".

With high confidence, as $f$ approaches one desirability approaches one (meaning "*the event is definitely desirable*"), and as $f$ approaches zero desirability approaches zero (meaning "*the event is definitely not desirable*"). Thus a goal with both high frequency and confidence (closer to 1) is required to pass the Decision-Making Rule; how high the desirability must be depends on the value of the desire threshold $T$, an evolved meta-parameter.

## 4.2 Experiment Domain #1: Wheeled Robot Animat Ecosystem

### *4.2.1 Experiment Setup*

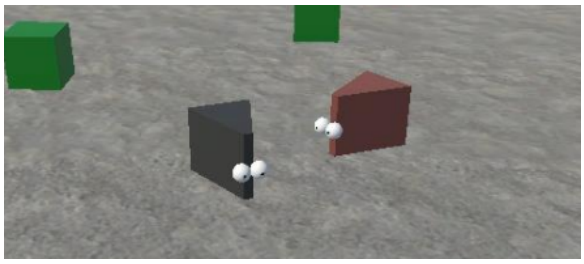

Figure 4.5: Wheeled Robot Screenshot

**4.2.1.1 Robot and Vision.** In these experiments, NARS was in control of a wheeled robot. Like all robots in the current version of ERAIASON, the robot had a set of universal robot actions: eating, fighting, cloning, and mating. The wheeled robot had two motor actions unique to it: moving forward and turning.

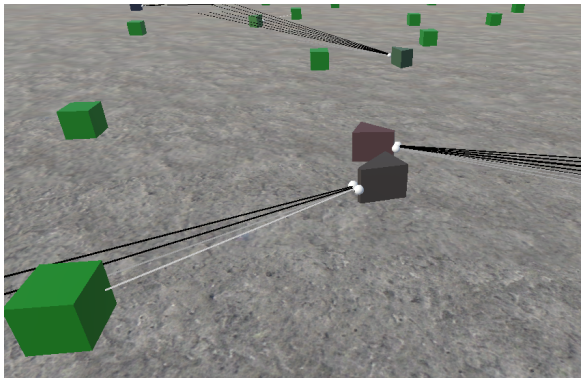

Figure 4.6: Wheeled Robot Vision Sensor Screenshot

The vision sensor of the robot was used to detect other robots and food. To simulate vision sensation, 9 raycasts were shot out from the robot's front in an outward circular pattern (see *Figure 4.6*. A vision ray shot 40 meters and could detect the ground, food blocks, and other robots.

NARS received vision sensation judgments for food ($\{food\}$) and other robots ($\{animat\}$) only (not the ground, for simplicity). It did not receive statements from all 9 rays, instead it received information from the ray with the highest activation aka the ray detecting the closest object of each type. Since discretized statements are easier to handle in NARS, the vision sensor did not change truth-value based on distance, but instead categorized the seen object into 1 of 4 properties: $[far]$, $[medium]$, $[near]$, and $[unseen]$. The truth value for visual sensations was fixed to $\langle 1.0, 0.99 \rangle$.

For example, let's say 3 rays hit 3 different objects: a food block nearby, a food block far away, and a robot at medium distance. Since there is a food nearby, the food far away is ignored. Then, 2 input judgments are sent to NARS: $<\{food\} \rightarrow [near]> \langle 1.0, 0.99 \rangle$ and $<\{animat\} \rightarrow [medium]> \langle 1.0, 0.99 \rangle$. If the rays did not hit anything, instead the judgments would be $<\{food\} \rightarrow [unseen]> \langle 1.0, 0.99 \rangle$ and $<\{animat\} \rightarrow [unseen]> \langle 1.0, 0.99 \rangle$.

An object was considered $[near]$ if it was detected in the action range (2 meters). An object was considered $[medium]$ distance if it was detected between 2 and 20 meters. An

object was considered $[far]$ if it was detected past 20 meters (up to the max of 40 meters). If no object of the type was detected by the vision sensor, it was considered $[unseen]$.

There was also a "mouth" or "energy" sensation, where NARS could sense its internal energy stores increasing whenever it ate. In this case, when NARS ate a portion of a food block, it received a judgment $<\{ENERGY\} \rightarrow [full]>$. $\langle f, 0.99 \rangle$, where $f = \frac{E}{E_{max}}$ was how filled NARS energy was, the # of stored energy $E$ divided by the maximum storage capacity $E_{max} = 500$. Similarly, the act of successful reproduction (asexually or sexually) was represented by the statement $<\{SELF\} \rightarrow [mated]>$.

An animat could only successfully execute motor operations on nearby objects. Concretely, actions (e.g., eat food, mate, or fight) only affected objects if they were visually detected in the action range (less than 2 meters).

Vision rays could detect robots, thus one robot could block another's vision rays. However, they could not physically block each other. The robot bodies had their collisions disabled; they moved through each other rather than colliding.

The NARS system executed 1 working cycle every other physics frame (i.e. 1 working cycle per $0.04$ seconds). Goals and sensory beliefs were fed every working cycle.

**4.2.1.2 Environment.** The environment was a fixed size arena of area $256 \times 256$ square meters. It consisted of a floor, which was a 2D flat plane, and 150 food blocks. The food blocks were spawned in random locations around the arena; starting with 1 energy, they grew at a rate of $0.01$ energy every $0.02$ seconds (i.e., $0.5$ energy per second) up until containing their maximum capacity of 100 energy. They grew in size depending on the amount of energy they contained. If a food block was completely consumed by dropping below zero energy, a new one was spawned in a random location.

The placement algorithm for robots sampled 1000 random locations, attempting to spawn them far away from food blocks. The first location 8 meters away from a food

block was used, otherwise if no such location was found, then the next best location was used. The same logic was used for food, but to spawn them away from robots.

### *4.2.2 Evolutionary Algorithm*

**4.2.2.1 Genetic Encoding.** The sensorymotor sets were as follows.

The sensory set contained 10 sensory events related to vision and physiology. The vision sensory events represented the detected presence and distance of food and other animats, while the physiology sensory events represented the internal energy stores and successful reproduction (see *Table 4.1*).

The motor set contained 6 motor operations which the agent used to control its robotic body and interact with the world. The agent could $\Uparrow move$ forward, $\Uparrow turn$ clockwise, $\Uparrow eat$ nearby food blocks to gain more energy, $\Uparrow fight$ with nearby agents to reduce their health, $\Uparrow clone$ itself to reproduce asexually, and $\Uparrow mate$ with nearby agents to reproduce sexually (see *Table 4.2*).

Table 4.1: Wheeled Robot Ecosystem Sensory Set $\mathbb{S}$

| | |
|---|---|
| $\mathbb{S}$ | $<\{food\} \rightarrow [near]>$ |
| | $<\{food\} \rightarrow [medium]>$ |
| | $<\{food\} \rightarrow [far]>$ |
| | $<\{food\} \rightarrow [unseen]>$ |
| | $<\{animat\} \rightarrow [near]>$ |
| | $<\{animat\} \rightarrow [medium]>$ |
| | $<\{animat\} \rightarrow [far]>$ |
| | $<\{animat\} \rightarrow [unseen]>$ |
| | $<\{ENERGY\} \rightarrow [full]>$ |
| | $<\{SELF\} \rightarrow [mated]>$ |

Table 4.2: Wheeled Robot Ecosystem Motor Set $\mathbb{M}$

| | |
|---|---|
| $\mathbb{M}$ | $\Uparrow move$ |
| | $\Uparrow turn$ |
| | $\Uparrow eat$ |
| | $\Uparrow fight$ |
| | $\Uparrow clone$ |
| | $\Uparrow mate$ |

Thus, a genome in this environment could look like the one in *Figure 4.7*.

This genome contains 2 instinctual goals: to eat food and to reproduce. It also contains 5 instinctual beliefs, which form a deductive chain or plan, telling NARS how to eat food in any given sensory situation:

$\beta$:
$J_1 = <<\{food\} \rightarrow [far]>, \Uparrow move \Rightarrow <\{food\} \rightarrow [near]>>. \langle 0.99, 0.81 \rangle$
$J_2 = <<\{food\} \rightarrow [near]>, \Uparrow eat \Rightarrow <\{ENERGY\} \rightarrow [full]>)>. \langle 1.0, 0.9 \rangle$
$J_3 = <<\{animat\} \rightarrow [far]>, \Uparrow move \Rightarrow <\{animat\} \rightarrow [near]>>. \langle 0.6, 0.64 \rangle$
$J_4 = <<\{food\} \rightarrow [unseen]>, \Uparrow turn \Rightarrow <\{food\} \rightarrow [far]>>. \langle 0.72, 0.88 \rangle$
$J_5 = <<\{animat\} \rightarrow [unseen]>, \Uparrow turn \Rightarrow <\{animat\} \rightarrow [far]>>. \langle 0.7, 0.99 \rangle$

$\Gamma$:
$G_1 = <\{ENERGY\} \rightarrow [full]>! \langle 1.0, 0.99 \rangle$
$G_2 = <\{SELF\} \rightarrow [mated]>! \langle 1.0, 0.99 \rangle$

$\Pi$:
$k = 1$
$T = 0.5$
etc.

Figure 4.7: Concrete NARS Genome Example

$$(<\{food\} \rightarrow [unseen]>, \Uparrow turn) \Rightarrow (<\{food\} \rightarrow [far]>, \Uparrow move) \Rightarrow (<\{food\} \rightarrow [near]>, \Uparrow eat) \Rightarrow <\{ENERGY\} \rightarrow [full]>$$

For recombination between 2 parents, the $\beta$ sets were lined up, the $\Gamma$ sets were lined up, and the $\Pi$ sets were lined up. Then, for each set, there was a 50/50 chance to select an element from one or the other parent. If the set from one parent was longer than the set from the other parent, then there was a 50/50 chance to take the gene or leave it. A second offspring was also produced, which took the other option of the 50/50 gene selection at each crossover step. Thus, this process produced offsprings which mixed the belief/goal sets and parameter values of the parents.

For mutation of 1 parent, there was a 1 in 3 chance to either: add a new belief to $\beta$, remove a random belief from $\beta$, or to change the term in a belief from $\beta$.

- *Add new belief:* Generate a new belief $<S, \Uparrow M \Rightarrow P>$. using randomly selected $\{S, P\} \in \mathbb{S}$ and $\Uparrow M \in \mathbb{M}$ with random truth value $\langle f, c \rangle$

- *Remove existing belief:* Uniform random select a belief from $\beta$. Remove it.
- *Modify existing belief:* Uniform random select a belief from $\beta$. Randomly replace $S$, $P$, or $\Uparrow M$.

Then, there was an 80% chance to mutate the truth values of sentences in $\beta$. In doing so, both frequency and confidence each had a 60% chance to be mutated. If they were chosen for mutation, there was a 5% chance to replace the value with a value from range $(0, 1)$, and a 95% chance to perturb the value by a number selected from a normal distribution with mean of zero and standard deviation of 0.1 (i.e., 68% of perturbation values will be in range [-0.1,0.1], 98% of values will be in range [-0.2,0.2], etc.).

Similarly, there was an 80% chance to modify the personality parameters $\Pi$. If so, each parameter had a 50% chance to be mutated. If so, there was a 5% chance to replace the value with a value from its range $[R_L, R_H]$, and a 95% chance to perturb the value by a number selected from a normal distribution with mean of zero and standard deviation of $0.1(R_H - R_L)$.

The allowable range that was permitted for each personality parameter can be seen in *Table 4.3*.

Table 4.3: NARS Personality Parameter Ranges in Wheeled Robot Ecosystem

| Parameter Name | Parameter Range |
|---|---|
| $k$ | $[1, 10]$ |
| $T$ | $[0, 1]$ |
| $W$ | $[1, 30]$ |
| $F$ | $[1, 250]$ |
| $C_{temp}$ | $[3, 20]$ |
| $C_{table}$ | $[1, 20]$ |
| $E$ | $[1, 50]$ |
| $\mathcal{T}_J$ | $[0.0000001, 10]$ |
| $\mathcal{T}_G$ | $[0.0000001, 10]$ |

**4.2.2.2 Evolutionary Search.** Although the robots could reproduce autonomously, it was a relatively rare occurrence, especially early in the evolutionary process before useful

behaviors had evolved. Therefore, when the population dropped below a minimal threshold, a new agent had to be spawned from some source. Thus, 3 reproductive pools of size 100 were kept which stored animat genomes along with their scores: a fitness hall-of-fame pool, a novelty hall-of-fame pool, and a recent population pool (see *Figure 4.8*). The fitness and novelty hall-of-fames kept most-fit and most-novel animats (for the novelty reproductive pool, their scores were recomputed whenever the archive changed). The recent population reproductive pool simply held the most recently dead agents, with their associated fitness scores.

| Fitness Hall of Fame | |
|---|---|
| Animat 25 | *score = 3.10* |
| Animat 12 | *score = 1.45* |
| Animat 30 | *score = 0.98* |

| Novelty Hall of Fame | |
|---|---|
| Animat 30 | *score = 203.50* |
| Animat 9 | *score = 104.10* |
| Animat 22 | *score = 55.13* |

| Recent Population | |
|---|---|
| Animat 31 | *score = 0.03* |
| Animat 30 | *score = 0.98* |
| Animat 33 | *score = 0.22* |

Figure 4.8: Animat Reproductive Pools

From these tables, animats could be selected for mutation or recombination. There was a 50/50 chance to perform mutation vs. recombination. To select an animat, 1 of 3 reproductive pools was randomly selected. Then, there was a 50/50 chance to perform Roulette Wheel (probabilistic) selection or Tournament selection ($k = 7$).

The equation computing the fitness score $\mathcal{F}$ was:

$$\mathcal{F} = (d + dl + lf + f)(1 + R)$$

In this equation, $d \in [0, 1]$ represented the robot's final displacement from its birthplace, $l \in [0, 1]$ represented the percent of lifetime the robot spent looking at food scaled by the robot's visual distance to the food, $f \in [0, \infty)$ was the total amount of food blocks the robot ate, and $R \in [0, \infty)$ was the total number of times the robot reproduced.

This fitness function encouraged robots to move far, to look closely at food, to eat food, and to reproduce.

The novelty function computing behavioral difference $\mathcal{D}_t$ between 2 robots $i$ and $j$ at a given moment in time $t$ from birth, was:

$$\mathcal{D}_t = |food_seen_i - food_seen_j| + ||pos_j - pos_i||$$

in this equation, $food_seen$ was the maximum activation of food visual sensor (at time $t$), $pos$ was a 2D location of the offset of the robot from its birthplace. Then, the behavioral difference scores $D_t$ were summed and averaged to get a final novelty score between 2 robots. The final novelty score overall for a robot was the average of the novelty between it and the $k-$nearest neighbors ($k = 15$) in the novelty archive, to which each robot had a 10% chance to be added when it died.

This novelty function encouraged robots to look at food from various distances, and to move around the map in various ways.

### *4.2.3 Experiment 1.1: Evolution of Sensorimotor Contingencies*

**4.2.3.1 Setup.** This first set of experiments was used to test the NARS genetic encoding evolution of sensorimotor contingencies. The NARS system was allowed to derive new goals from its existing knowledge, but was not permitted to learn new knowledge. Therefore its behavior was entirely evolved and instinctual.

The agent had 2 goals, which did not evolve: keep energy full ($<\{ENERGY\} \rightarrow [full]>! \langle 1.0, 0.99\rangle$) and reproduce ($<\{SELF\} \rightarrow [mated]>! \langle 1.0, 0.99\rangle$).

This 1-condition experiment had a minimum population of 50 agents. It was run for 3 trials, 480 time indices per trial (1 time index = 8 seconds, about 1 hour total per trial), and the results were averaged.

**4.2.3.2 Results.** The first major result shows the evolution of NARS' increased ability to autonomously upkeep its own population levels. NARS was able to sustain up to 30% of its population autonomously, on average around 20% (see *Figure 4.9*). With 50 agents

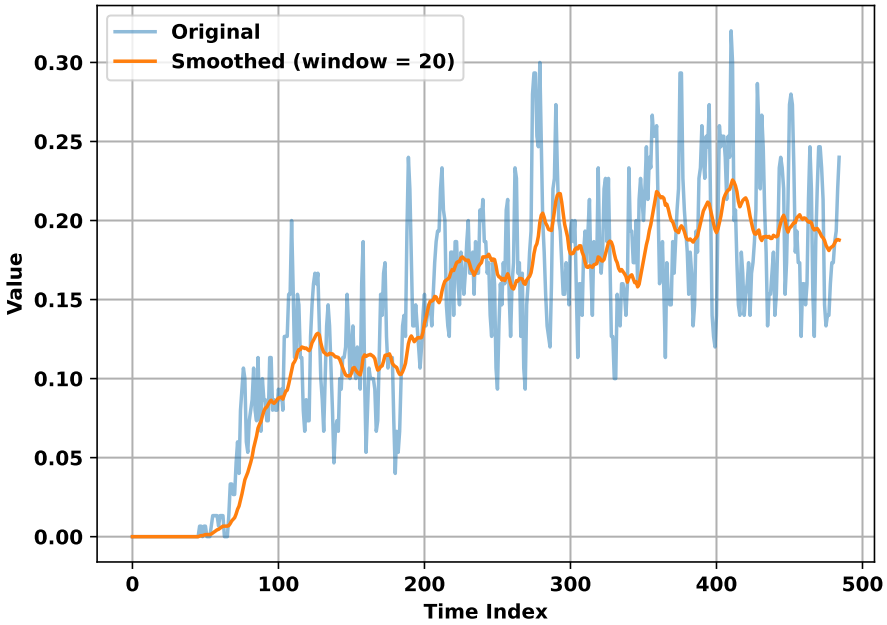


Figure 4.9: NARS Self-Sustained Population Metrics (Percent of Population Born By NARS)

at any given time and 20% produced by NARS, that means around 10 agents in the world were produced by NARS agents themselves, rather than the evolutionary algorithm, at any given time.

Other stats show (see *Figure 4.10*) that the best NARS agents were able to consume 3.5 food blocks, and asexually reproduce (aka "clone") a corresponding 3.5 times. There was also, rarely, some cases of sexual reproduction (aka "mating"), which could only happen in cases where both NARS agents had sufficient energy, one NARS approached another, and both were expressing their mating behavior. Though simple, it demonstrates the ability of NARS agents to interact in a shared ecosystem/society and do something useful (in this case, propagating more NARS agents).

We can examine the trends of the NARS personality parameters (see *Figure 4.11*) in the 100 best agents found so far (averaged over the 3 trials) to determine what factors may have caused NARS to perform better.

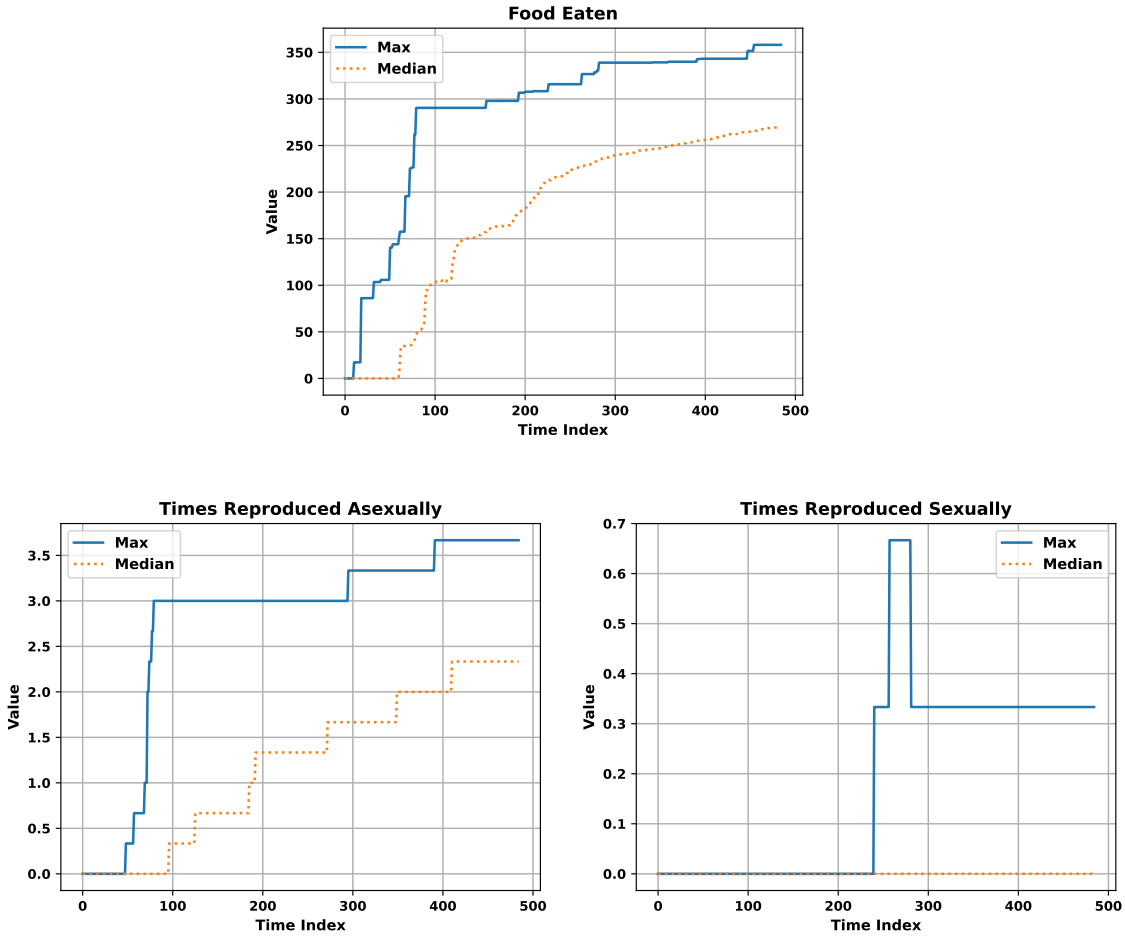


Figure 4.10: Wheeled Robot Evolved Instincts Metrics

Firstly, we can see that over time, the median number of beliefs increased in the top 100 agents, from 5 to about 8. This makes sense, since more information about the world should, in theory, make NARS more effective.

Secondly, we can see that the median evidential horizon $k$ remained approximately the same throughout the simulation. This suggests that changing the $k$ value away from its default of 1 did not have a positive impact on behavior.

Thirdly, we can see that the median decision-making threshold $T$ went down over time. This suggests that early NARS agents may have been too inhibited. It is likely that lowering the inhibition resulted in more motor operations, which resulted in higher fitness scores.

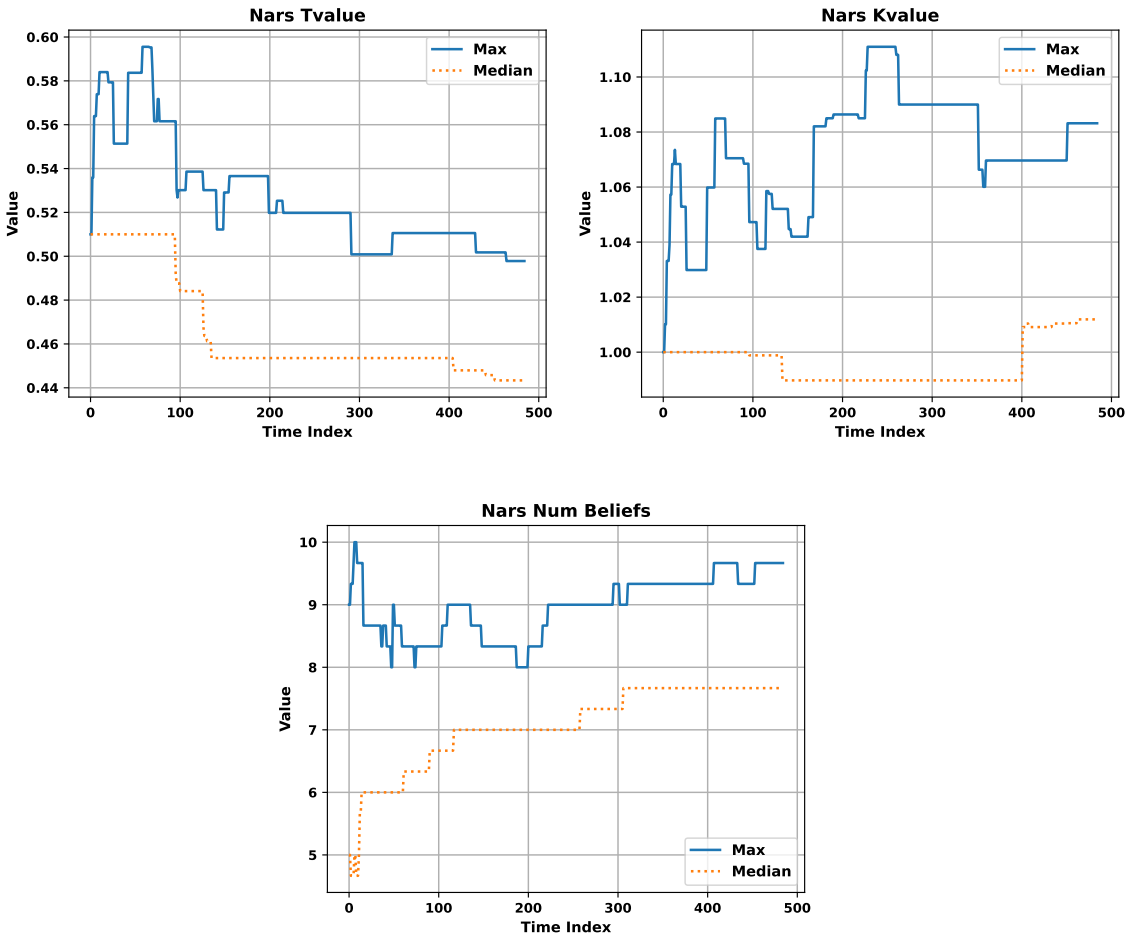


Figure 4.11: Wheeled Robot Ecosystem Instinct-Enabled Evolved Personality Parameters

Overall, the results indicate that it is possible and effective to evolve sensorimotor contingencies. From the results, we are also able to see which values made for a more effective NARS system in this environment.

### *4.2.4 Experiment 1.2: Evolution of Learning*

**4.2.4.1 Setup.** This second set of experiments was used to test the NARS genetic encoding evolution of sensorimotor contingencies *and* personality (learning) parameters. 8 different conditions were tested, testing various combinations of: learning enabled/disabled, evolution of instincts enabled/disabled, and fixed/random/evolved personality parameters.

The conditions were:

1. `NARS_EVOLVE_CONTINGENCIES_FIXED_PERSONALITY_LEARNING:` Learning enabled, personality parameters did not change, instincts were evolved.

2. `NARS_EVOLVE_CONTINGENCIES_FIXED_PERSONALITY_NO_LEARNING:` Learning disabled, personality parameters did not change, instincts were evolved.

3. `NARS_EVOLVE_CONTINGENCIES_RANDOM_PERSONALITY_LEARNING:` Learning enabled, personality parameters were randomized, instincts were evolved.

4. `NARS_EVOLVE_CONTINGENCIES_RANDOM_PERSONALITY_NO_LEARNING:` Learning disabled, personality parameters were randomized, instincts were evolved.

5. `NARS_EVOLVE_PERSONALITY_AND_CONTINGENCIES_LEARNING:` Learning enabled, personality parameters were evolved, instincts were evolved.

6. `NARS_EVOLVE_PERSONALITY_AND_CONTINGENCIES_NO_LEARNING:` Learning disabled, personality parameters were evolved, instincts were evolved.

7. `NARS_EVOLVE_PERSONALITY_LEARNING:` Learning enabled, personality parameters were evolved, blank memory.

8. `NARS_NO_CONTINGENCY_FIXED_PERSONALITY_LEARNING:` Learning enabled, personality parameters did not change, blank memory.

9. `NARS_NO_CONTINGENCY_RANDOM_PERSONALITY_LEARNING:` Learning enabled, personality parameters were randomized, blank memory.

"Fixed personality parameters" were selected randomly (once) at the beginning of the trial and held fixed for all agents. This was to provide some baselines, representing the usual method of configuring NARS.

"Random personality parameters" randomized the personality of every genome before deployment. This was to have a comparison with evolutionary search, which is itself a random-based search method guided by a fitness heuristic.

The agent had only 1 goal for simplicity, which did not evolve: keep energy full ($<\{ENERGY\} \rightarrow [full]>! \langle 1.0, 0.99 \rangle$).

**4.2.4.2 Results.** This 8-condition experiment had a minimum population of 30 agents. Each condition was run for 10 trials, 480 time indices per trial (1 time index = 8 seconds, about 1 hour per trial), and the results were averaged.

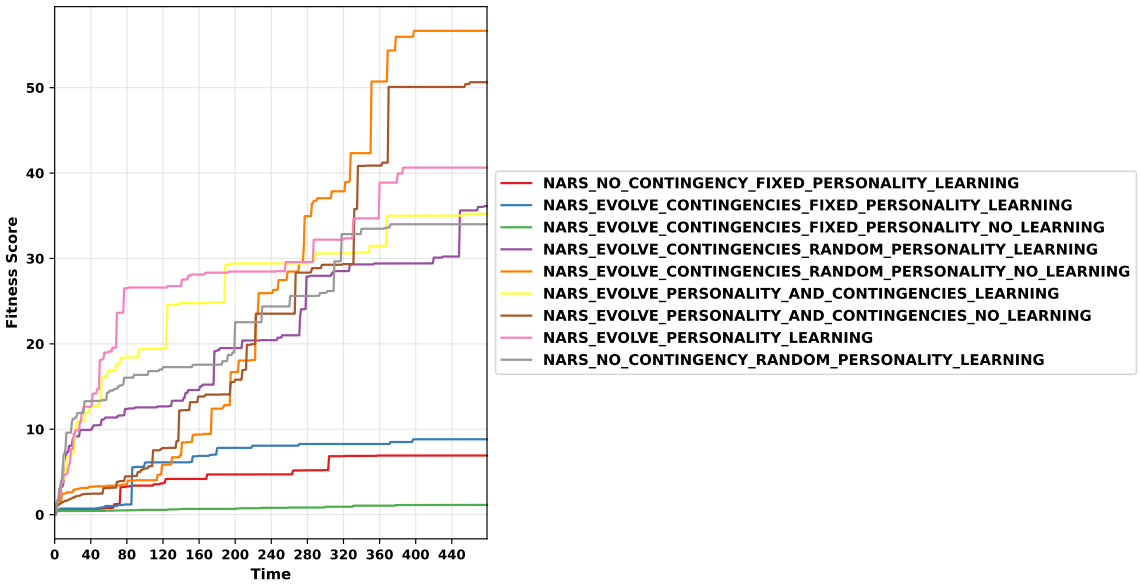


Figure 4.12: Wheeled Robot Ecosystem Fitness Scores under Different Experimental Conditions

First, we can examine the fitness scores in *Figure 4.12*. Immediately we can see that the best-scoring NARS by the end (orange line, 55) was produced by evolving instincts, random-searching personality parameters, and learning disabled. The second best method by the end (brown line, 50) was produced by evolving instincts, evolving personality parameters, and learning disabled. The third best method by the end (pink line, 40) was produced by evolving personality with learning enabled, and no instincts. The fourth best method by the end (purple line, 36) was produced by evolving instincts, random-searching personality parameters, and learning enabled. The fifth best method by the end (yellow line, 35) was produced by evolving instincts, evolving personality parameters, and learning enabled. The sixth best method by the end (gray line, 34) had random-searched personality parameters

and learning enabled, with no instincts. The seventh best method by the end (blue line, 9) had fixed personality, evolved instincts, and learning enabled. The eighth best method (red line, 7) had fixed personality, no instincts, and learning enabled. The worst-performing method (green line, 1.5) had fixed personality, evolved instincts, and learning disabled.

Already, we can see the huge benefit of testing various meta-level NARS "personality" configurations over the baselines. The three worst-performing methods all used fixed personality parameters, representing the baseline of how NARS systems are traditionally deployed, whereas the top six methods all tested varying personality parameters. The three methods with fixed personality parameters did far worse than the methods which tried various personality parameters.

The top 2 methods utilized the new genetic encoding's ability to bootstrap instinctual behaviors. The method in 3rd place used evolved learning capabilities to learn behaviors from scratch, and still performed very well. Somewhat surprisingly even random-searched personality parameters were able to produce agents with excellent performance, though it makes sense given the relatively small dimensionality (9 dimensions) of the personality parameter vector. Evolution would likely be preferred over random search if many more dimensions were added to the vector, such as in a future experiment optimizing many different meta-level aspects of NARS.

Although by the end of the evolutionary search process (time index 480), the methods which performed the best utilized only contingencies (orange line and red line), with learning disabled, it can be noted that the methods which had learning enabled (pink line, yellow line) reached higher scores earlier in the evolutionary search process. The methods with learning disabled achieved higher scores by the end but required more search time to achieve their optimal performance, while the methods with learning enabled achieved higher scores *earlier* in the search process, with the crossover point between the methods occurring approximately halfway through the trial (time index 240).

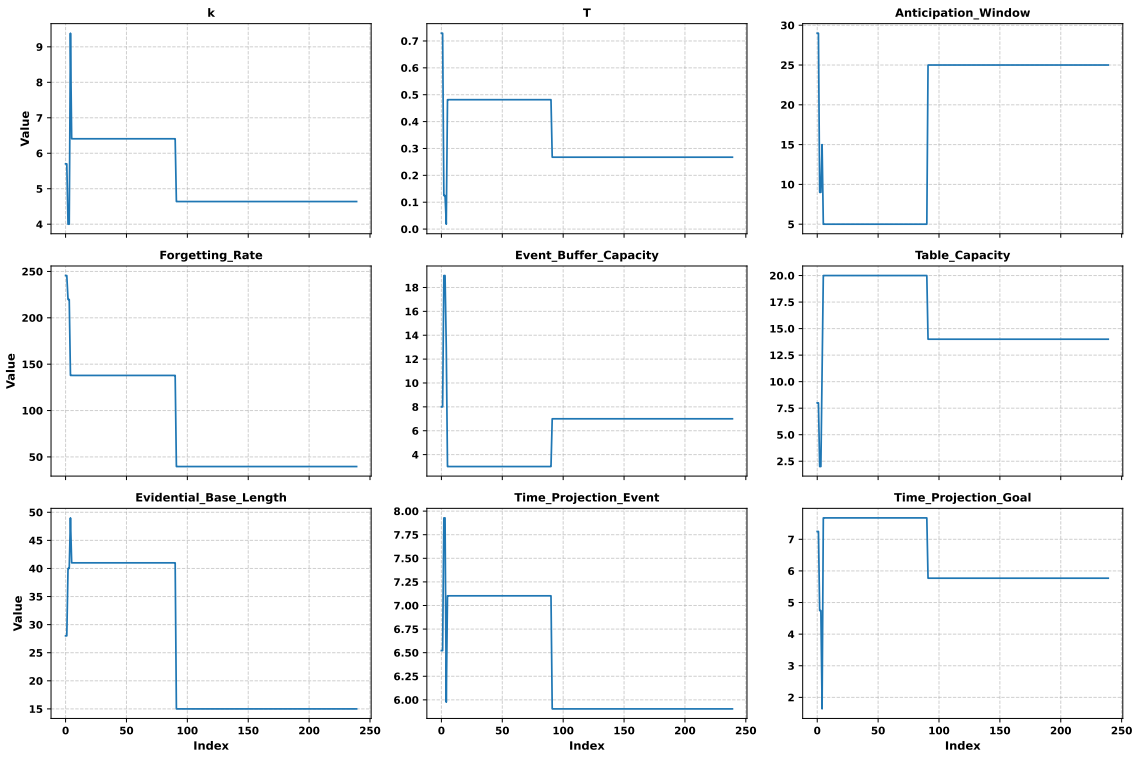


Figure 4.13: Wheeled Robot Ecosystem Best Learning-Enabled Agent Evolved Personality Parameters

Finally, we can examine how the personality parameters of the best NARS system changed over the course of a simulation (see *Figure 4.13*). We can examine the best evolved learning agent who started with a blank memory (pink line in *Figure 4.12*).

In *Figure 4.13*, we can see that the evidential horizon $k$ value of the best agent spiked at first to 9, but gradually reduced down to 5. The decision-making thresold $T$ value of the best agent remained below $0.5$ for most of the simulation, dipping down to $0.3$, making NARS very prone to action. The anticipation window also increased greatly, from 5 to 25, meaning NARS was waiting longer to become "disappointed" in its anticipations. The Forgetting Rate, a half life, dropped over the course of the simulation, making NARS forget faster. The length of the Temporal Module (labeled Event_Buffer here) stayed relatively small, from 3 to 7, meaning NARS was paying attention to a smaller window of events and producing fewer sensorimotor contingencies. The half life (time projection) of goal and

event stayed relatively in the middle of its range, around 5 or 6, meaning the confidence of a goal or event halved every 5 working cycles (here, every 0.2 seconds).

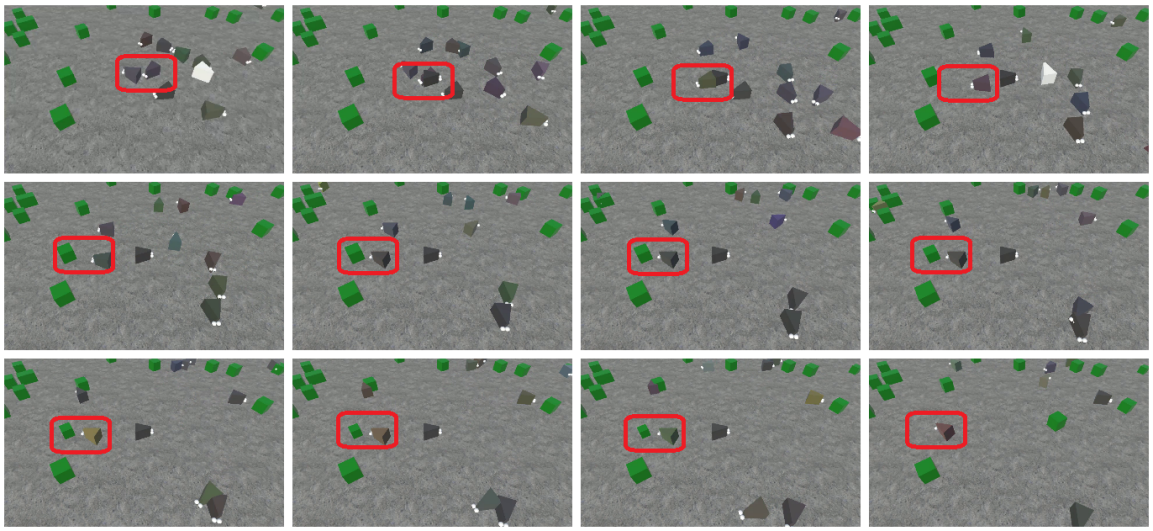

Figure 4.14: Wheeled Robot Ecosystem Screenshot

The learning ability of agents was spotted while viewing the simulation. In the sequence of frames shown in *Figure 4.14*, the agent started with a blank memory and was motor babbling and wandering around. When it incidentally came across a food block, it flashed its $\Uparrow eat$ motor operation and ate some food. This caused the following sequence of events to enter its Temporal Module:

1. $<\{food\} \rightarrow [near] >$
2. ... (potentially other events)
3. $\Uparrow eat$
4. ... (potentially other events)
5. $<\{ENERGY\} \rightarrow [full] >$

Using temporal induction, the system derived a hypothesis:

$$< <\{food\} \rightarrow [near] >, \Uparrow eat \Rightarrow <\{ENERGY\} \rightarrow [full] >>$$

Consequently, upon entering the system's memory, the system tested this hypothesis, which resulted in more food being eaten. This, in turn, further strengthened the hypothesis, which become more reinforced as the agent ate. This continued until the agent finished eating the food block, after which it moved on to a new location.

This experiment demonstrated the ability of evolved NARS agents to perform temporal (inductive) reasoning and build effective world models at runtime. It also demonstrated the improvement of evolved-configuration agents over baseline fixed-configuration agents. The next few chapters experimentally test extensions to this basic form of the genetic encoding.

# CHAPTER 5
# EVOLVING GENERALIZATION LEARNING AND INSTINCTS CONTAINING VARIABLES

## 5.1 Extending the Basic Encoding with Variables

The basic NARS genetic encoding can encode background knowledge or "instincts" in the form of sensorimotor contingencies $<S, \Uparrow M \models\!\!\!\Rightarrow P>$. A sensorimotor contingency consists of a combination of 2 specific sensory events ($S, P$, from a sensory event set $\mathbb{S}$) and 1 specific motor operation ($M$, from a motor operation set $\mathbb{M}$). Evolution of NARS instincts therefore involves a search process through the space of sensorimotor contingencies, which has a size of $|\mathbb{S}|^2|\mathbb{M}|$ (see *Figure 5.1*). Consequently, the search space for genomes which contain up to $N$ unique sensorimotor contingencies has a size of $\sum_{k=0}^{min(N,|\mathbb{S}|^2|\mathbb{M}|)} \binom{|\mathbb{S}|^2|\mathbb{M}|}{k}$.

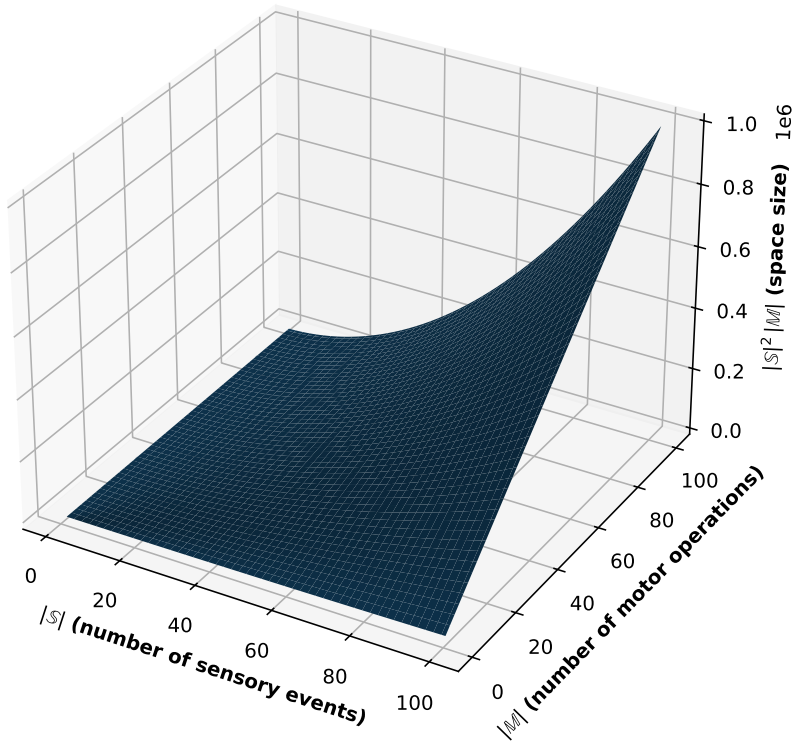


Figure 5.1: Size of the Search Space of Basic Sensorimotor Contingencies ($|\mathbb{S}|^2|\mathbb{M}|$)

Some tasks for NARS may require only a small sensor and motor set, such as demonstrated in the experiments of *Chapter 4*. This results in a rather tractable search space.

However, we can imagine scenarios where the sensor and/or motor set are much larger. We can also imagine cases where many redundant statements are generated for overly specialized situations, where a single more general statement would be more appropriate. In such cases, the size of the search space explodes.

In light of this, the basic genetic encoding of NARS was extended to include NAL *independent variables*, allowing it to create generalized statements, in addition to the specific statements allowed by the basic encoding. New mutation operators (added to $\Omega$) were created whose function was to add (or remove) variables from existing statements in the genome.

For example, a new *Variable-Introduction* mutation operator works as follows. For a sensorimotor contingency $<S, \Uparrow M \Rightarrow P>$ randomly selected from the existing genome, the mutation operator first selects a random element of the sensorimotor contingency ($S$, $M$, or $P$) and then replaces a random term within that element with a variable $\$x$. Then, the same process could occur in one (or both) of the remaining elements, resulting in one, two, or three variables within the overall sensorimotor contingency.

For example, for a sentence:

$$<<food \rightarrow [left]> \wedge \Uparrow eat(left) \Rightarrow <\{ENERGY\} \rightarrow [increasing]>>$$

The Variable-Introduction mutation operator would convert it into a sentence containing variables:

$$<<food \rightarrow [\$x]> \wedge \Uparrow eat(\$x) \Rightarrow <\{ENERGY\} \rightarrow [increasing]>>$$

which says "*when food has some property $x$ and NARS executes $\Uparrow eat$ with $x$ as an argument, then energy increases*".

A second mutation operator, the *Variable-Remove* mutation, reverses this process, replacing one or more variables in a statement with randomly selected concrete terms pulled from $\mathbb{S}$ and $\mathbb{M}$. For example, the statement containing variables above might turn into:

$$<<food \rightarrow [right]> \wedge \Uparrow eat(right) \not\Rightarrow <\{ENERGY\} \rightarrow [increasing]>>$$

Finally, the *Modify Belief* mutation operator was changed to account for variable terms. When the old statement $S$ contained a variable, the new statement to replace it, selected from $\mathbb{S}$, was also modified to contain a variable. In other words, only the *Variable-Remove* mutation operator could remove a variable. Modify Belief could only change the statement to a new statement, keeping the variable if it existed. For example, if this belief was modified:

$$<<food \rightarrow [\$x]> \wedge \Uparrow eat(\$x) \not\Rightarrow <\{ENERGY\} \rightarrow [increasing]>>$$

And $S$ was selected for modification, the statement might turn into:

$$<<water \rightarrow [\$x]> \wedge \Uparrow eat(\$x) \not\Rightarrow <\{ENERGY\} \rightarrow [increasing]>>$$

such that $S$ mutated from a statement about the $food$ sensor into a statement about the $water$ sensor.

The next section provides a hypothetical analysis of using such genetic operators in a grid world environment. The subsequent section details the results of actual experiments in the grid world, showcasing the efficacy of the variable-extended encoding over the basic encoding.

## 5.2 Experiment Domain #2: Grid World

### *5.2.1 Experiment Setup*

**5.2.1.1 Environment.** The environment was a 2D grid world representing a natural region containing goats (agents, which acted as moving obstacles and sensory distractors), grass (food/energy the agents wanted to consume), and water (stationary obstacles and sensory distractors). See Figure 5.2 for a screenshot of the simulation.

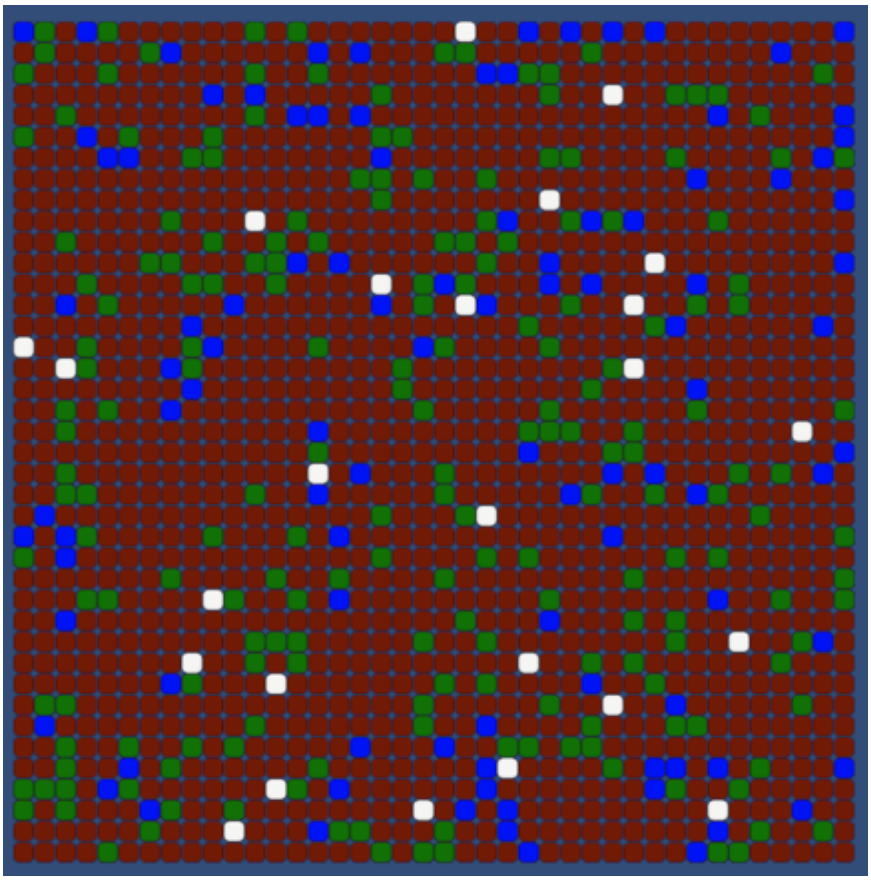

Figure 5.2: Grid World Screenshot

Agents could move into empty grid tiles, but could not move onto other tiles containing an agent, grass, or water.

The population size was 25 (i.e., 25 agents were in the world at any given time).

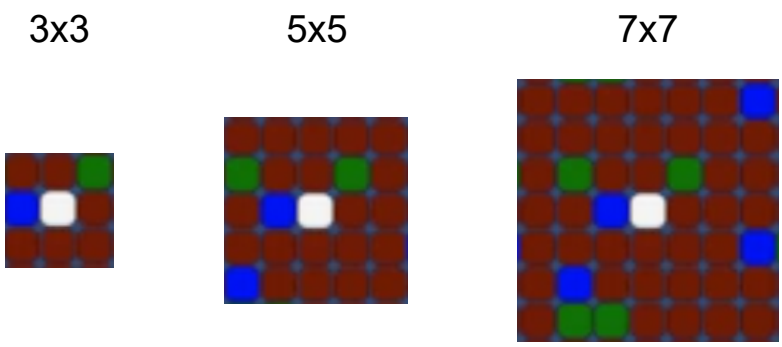


Figure 5.3: Grid World Screenshot of Various-Sized Perceptual Fields

**5.2.1.2 Agents.** Each agent had a "perceptual field" around them determining how many tiles they could sense and act upon. Three different-sized perceptual fields were tested: $3 \times 3$, where the agent could perceive only the tiles immediately surrounding it; $5 \times 5$, where the agent could perceive up to 2 tiles out; and $7 \times 7$, where the agent could perceive up to 3 tiles out (see *Figure 5.3*). Each increase in the perceptual field introduced more sensory/motor possibilities, and thus the size of the evolutionary search space.

The goal of the agents was to increase their energy stores, by consuming grass in the environment. More specifically, each genome was hardcoded to $\Gamma = \{<\{ENERGY\} \rightarrow [increasing]>! \langle 1.0, 0.99 \rangle\}$.

An agent in the grid world simulation was born with $10$ units of energy. Each simulation tick, the agent lost $1$ energy. If an agent's energy reached zero, the agent died, meaning it was removed from the simulation. However, by consuming a grass tile, the agent's energy was restored to 10. Regardless, an agent could only survive for a maximum of 100 ticks, after which it was culled, to allow for fitness evaluation and to make room for instantiating new agents with further evolved genomes.

### *5.2.2 Evolutionary Algorithm*

In the grid world experiments, there was only one reproductive pool, a Fitness Hall of Fame (size 100), where the genomes of the best-performing animats were stored, sorted according to their fitness.

When an agent died, its fitness was scored and the agent was added to the reproductive pool if it was in the Top 100 scorers found so far. The fitness function computing the fitness score $\mathcal{F}$ was:

$$\mathcal{F} = \begin{cases} f, & \text{if } f > 0, \\ \dfrac{m}{t}, & \text{otherwise.} \end{cases}$$

where $m$ was the number of times a successful $\Uparrow move$ was executed, $t$ was the number of timesteps the agent was alive, and $f$ was the number of food tiles eaten. This fitness function rewarded agents who ate more food; if the agent was unable to eat any food, then it incentivized agents to move.

Whenever the number of living agents dropped below 25, new agents were generated using the reproductive pool to restore the population levels. There was a 50/50 chance for asexual/sexual reproduction. A parent was chosen from the reproductive pool probabilistically, with a chance proportionate to its fitness.

**5.2.2.1 Genetic Encoding.** The sensorymotor sets for the grid world were as follows.

For the sensors: NARS was able to detect 3 types of objects (grass, goat, and water) in its $N \times N$ (where $N$ is odd, e.g., $5 \times 5$) cellular neighborhood, excluding the agent's current location. Thus, there were $N^2 - 1$ tiles that a NARS agent sensed at a given timestep.

Table 5.1: Compact Grid World Sensory Set $\mathbb{S}$

| | |
|---|---|
| $\mathbb{S}$ | $<grass \rightarrow [dir]>$ |
| | $<goat \rightarrow [dir]>$ |
| | $<water \rightarrow [dir]>$ |
| | $<\{ENERGY\} \rightarrow [increasing]>$ |

$dir$ was the relative grid cell location in which the object was detected.

For an $N \times N$ cellular neighborhood (where $N$ is odd)

$$m = \frac{N-1}{2}$$

$$dir \in \left\{ d_{i,j} \;\middle|\; i,j \in \{-m,\ldots,m\},\ (i,j) \neq (0,0) \right\}$$

$$dir \in \left\{ \begin{array}{ccccccc} d_{-m,m} & \cdots & d_{-1,m} & d_{0,m} & d_{1,m} & \cdots & d_{m,m} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ d_{-m,1} & \cdots & d_{-1,1} & d_{0,1} & d_{1,1} & \cdots & d_{m,1} \\ d_{-m,0} & \cdots & d_{-1,0} & & d_{1,0} & \cdots & d_{m,0} \\ d_{-m,-1} & \cdots & d_{-1,-1} & d_{0,-1} & d_{1,-1} & \cdots & d_{m,-1} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ d_{-m,-m} & \cdots & d_{-1,-m} & d_{0,-m} & d_{1,-m} & \cdots & d_{m,-m} \end{array} \right\}$$

with $d_{i,j}$ being the tile at relative 2D position $(i,j)$.

Therefore, with 3 possible objects to detect in $N^2-1$ possible places, there were $3N^2-3$ total possible sensory events that NARS could experience (e.g., for a $5 \times 5$ neighborhood, 72 events). When including the goal statement $<\{ENERGY\} \rightarrow [increasing]>$, this meant $3N^2 - 2$ events available for the genetic encoding from $\mathbb{S}$ to use when producing sensorimotor contingencies.

For the motors: NARS was able to move ($\Uparrow move$) anywhere in its $N \times N$ cellular neighborhood. It was also able to try $\Uparrow eat$ from a cell in its cellular neighborhood.

Table 5.2: Compact Grid World Motor Set $\mathbb{M}$

| $\mathbb{M}$ | $\Uparrow move(dir)$ |
|---|---|
| | $\Uparrow eat(dir)$ |

$dir$ was the relative grid cell location in which NARS wanted to execute its action. With 2 possible motor operations, each able to have 1 of $N^2 - 1$ possible arguments (the relative cell location), there were $2N^2 - 2$ total possible motor commands which could be generated by the encoding and used by NARS. Therefore, the total possible number of sensorimotor contingencies that the genetic encoding could generate was $(3N^2-2)^2(2N^2-2) = 18N^6 - 42N^4 + 32N^2 - 8$.

To illustrate the large search space, the fully expanded sensorymotor sets for $5 \times 5$ are shown in *Table 5.3* and *Table 5.4*. The sensorymotor sets for $7 \times 7$ were even larger.

Table 5.3: Expanded Grid World Sensory Set $\mathbb{S}$ for $5 \times 5$ Perception

| $\mathbb{S}$ | **goat** | **grass** | **water** |
|---|---|---|---|
| | <goat → $[d_{-2,2}]$> | <grass → $[d_{-2,2}]$> | <water → $[d_{-2,2}]$> |
| | <goat → $[d_{-1,2}]$> | <grass → $[d_{-1,2}]$> | <water → $[d_{-1,2}]$> |
| | <goat → $[d_{0,2}]$> | <grass → $[d_{0,2}]$> | <water → $[d_{0,2}]$> |
| | <goat → $[d_{1,2}]$> | <grass → $[d_{1,2}]$> | <water → $[d_{1,2}]$> |
| | <goat → $[d_{2,2}]$> | <grass → $[d_{2,2}]$> | <water → $[d_{2,2}]$> |
| | <goat → $[d_{-2,1}]$> | <grass → $[d_{-2,1}]$> | <water → $[d_{-2,1}]$> |
| | <goat → $[d_{-1,1}]$> | <grass → $[d_{-1,1}]$> | <water → $[d_{-1,1}]$> |
| | <goat → $[d_{0,1}]$> | <grass → $[d_{0,1}]$> | <water → $[d_{0,1}]$> |
| | <goat → $[d_{1,1}]$> | <grass → $[d_{1,1}]$> | <water → $[d_{1,1}]$> |
| | <goat → $[d_{2,1}]$> | <grass → $[d_{2,1}]$> | <water → $[d_{2,1}]$> |
| | <goat → $[d_{-2,0}]$> | <grass → $[d_{-2,0}]$> | <water → $[d_{-2,0}]$> |
| | <goat → $[d_{-1,0}]$> | <grass → $[d_{-1,0}]$> | <water → $[d_{-1,0}]$> |
| | <goat → $[d_{1,0}]$> | <grass → $[d_{1,0}]$> | <water → $[d_{1,0}]$> |
| | <goat → $[d_{2,0}]$> | <grass → $[d_{2,0}]$> | <water → $[d_{2,0}]$> |
| | <goat → $[d_{-2,-1}]$> | <grass → $[d_{-2,-1}]$> | <water → $[d_{-2,-1}]$> |
| | <goat → $[d_{-1,-1}]$> | <grass → $[d_{-1,-1}]$> | <water → $[d_{-1,-1}]$> |
| | <goat → $[d_{0,-1}]$> | <grass → $[d_{0,-1}]$> | <water → $[d_{0,-1}]$> |
| | <goat → $[d_{1,-1}]$> | <grass → $[d_{1,-1}]$> | <water → $[d_{1,-1}]$> |
| | <goat → $[d_{2,-1}]$> | <grass → $[d_{2,-1}]$> | <water → $[d_{2,-1}]$> |
| | <goat → $[d_{-2,-2}]$> | <grass → $[d_{-2,-2}]$> | <water → $[d_{-2,-2}]$> |
| | <goat → $[d_{-1,-2}]$> | <grass → $[d_{-1,-2}]$> | <water → $[d_{-1,-2}]$> |
| | <goat → $[d_{0,-2}]$> | <grass → $[d_{0,-2}]$> | <water → $[d_{0,-2}]$> |
| | <goat → $[d_{1,-2}]$> | <grass → $[d_{1,-2}]$> | <water → $[d_{1,-2}]$> |
| | <goat → $[d_{2,-2}]$> | <grass → $[d_{2,-2}]$> | <water → $[d_{2,-2}]$> |

Table 5.4: Expanded Grid World Motor Set $\mathbb{M}$ for $5 \times 5$ Perception

| | **move** | **eat** |
|---|---|---|
| $\mathbb{M}$ | $< \Uparrow\text{move}(d_{-2,2})>$ | $< \Uparrow\text{eat}(d_{-2,2})>$ |
| | $< \Uparrow\text{move}(d_{-1,2})>$ | $< \Uparrow\text{eat}(d_{-1,2})>$ |
| | $< \Uparrow\text{move}(d_{0,2})>$ | $< \Uparrow\text{eat}(d_{0,2})>$ |
| | $< \Uparrow\text{move}(d_{1,2})>$ | $< \Uparrow\text{eat}(d_{1,2})>$ |
| | $< \Uparrow\text{move}(d_{2,2})>$ | $< \Uparrow\text{eat}(d_{2,2})>$ |
| | $< \Uparrow\text{move}(d_{-2,1})>$ | $< \Uparrow\text{eat}(d_{-2,1})>$ |
| | $< \Uparrow\text{move}(d_{-1,1})>$ | $< \Uparrow\text{eat}(d_{-1,1})>$ |
| | $< \Uparrow\text{move}(d_{0,1})>$ | $< \Uparrow\text{eat}(d_{0,1})>$ |
| | $< \Uparrow\text{move}(d_{1,1})>$ | $< \Uparrow\text{eat}(d_{1,1})>$ |
| | $< \Uparrow\text{move}(d_{2,1})>$ | $< \Uparrow\text{eat}(d_{2,1})>$ |
| | $< \Uparrow\text{move}(d_{-2,0})>$ | $< \Uparrow\text{eat}(d_{-2,0})>$ |
| | $< \Uparrow\text{move}(d_{-1,0})>$ | $< \Uparrow\text{eat}(d_{-1,0})>$ |
| | $< \Uparrow\text{move}(d_{1,0})>$ | $< \Uparrow\text{eat}(d_{1,0})>$ |
| | $< \Uparrow\text{move}(d_{2,0})>$ | $< \Uparrow\text{eat}(d_{2,0})>$ |
| | $< \Uparrow\text{move}(d_{-2,-1})>$ | $< \Uparrow\text{eat}(d_{-2,-1})>$ |
| | $< \Uparrow\text{move}(d_{-1,-1})>$ | $< \Uparrow\text{eat}(d_{-1,-1})>$ |
| | $< \Uparrow\text{move}(d_{0,-1})>$ | $< \Uparrow\text{eat}(d_{0,-1})>$ |
| | $< \Uparrow\text{move}(d_{1,-1})>$ | $< \Uparrow\text{eat}(d_{1,-1})>$ |
| | $< \Uparrow\text{move}(d_{2,-1})>$ | $< \Uparrow\text{eat}(d_{2,-1})>$ |
| | $< \Uparrow\text{move}(d_{-2,-2})>$ | $< \Uparrow\text{eat}(d_{-2,-2})>$ |
| | $< \Uparrow\text{move}(d_{-1,-2})>$ | $< \Uparrow\text{eat}(d_{-1,-2})>$ |
| | $< \Uparrow\text{move}(d_{0,-2})>$ | $< \Uparrow\text{eat}(d_{0,-2})>$ |
| | $< \Uparrow\text{move}(d_{1,-2})>$ | $< \Uparrow\text{eat}(d_{1,-2})>$ |
| | $< \Uparrow\text{move}(d_{2,-2})>$ | $< \Uparrow\text{eat}(d_{2,-2})>$ |

**5.2.2.2 Using NAL Variables in the Grid World.** Consider a $5 \times 5$ cellular neighborhood, or perceptual field. The standard NARS genetic encoding technically supports all 24 cellular neighbors in the Narsese statements, as sensor properties for the 3 object types ($<grass \rightarrow [d_{1,0}]>, <water \rightarrow [d_{1,1}]>, <goat \rightarrow [d_{-1,1}]>$, etc.). By enumerating all $3 * 24 = 72$ possibilities in $\mathbb{S}$, we allow the encoding to select from any of these sentences to use for $S$ and $P$.

Function arguments (such as the argument $dir$ in $\Uparrow move(dir)$) can be handled similarly by enumerating all possibilities of motor commands ($\Uparrow eat$ and $\Uparrow move$) plus all possible cellular neighbors as arguments, e.g., $\Uparrow move(d_{1,-1})$ and $\Uparrow move(d_{1,0})$ and $\Uparrow move(d_{1,1})$, etc. In this case that amounts to $2 * 24 = 48$ possibilities which the encoding can select for $\Uparrow M$ in the sensorimotor contingency.

However, enumerating so many different objects and directions explicitly presents a struggle for the genetic algorithm. To encode a piece of generalized knowledge, it would have to generate explicit and redundant rules for how to act in every single location/direction. For example the encoding has to generate knowledge telling how to eat grass in all 24 places, e.g., $<grass \rightarrow [d_{1,-1}]> \wedge \Uparrow eat(d_{1,-1})$ (*eat at cell (1,-1) when grass is at cell (1,-1)*) plus $<grass \rightarrow [d_{1,1}]> \wedge \Uparrow eat(d_{1,1})$ (*eat at cell (1,1) when grass is (1,1)*), plus another for the (1,0), and for (-1,0), and (-1,-1), (-1,1), (-2,0), (2,0), (2,1), and so on. Despite the fact there is a clear underlying general pattern, the basic encoding has to generate each statement separately. While this may be tractable for a small neighborhood, the enumeration of all possibilities could become too extensive, say if we expanded the cellular neighborhood of directions to an even wider area, or if we decided to add more sensors (e.g., to detect trees, fruits, etc.) and more motor operations. In that case, the evolutionary search space for the optimal genome becomes extremely large, since many redundant rules would have to evolve in the same genome to achieve the universal behavior of eating food wherever it is present. The search process for the optimal genome becomes much longer as the number of redundant rules increases.

Is there a way to reduce the evolutionary search space in problems involving redundant beliefs? Luckily, non-axiomatic logic supports *variables*. Rather than explicitly requiring a concrete term in a non-axiomatic logic statement, a variable term can be used to represent *any arbitrary* object or term that fits the statement (i.e., it complies with the syntax of the statement). For example, rather than to use 24 redundant sentences describing how to eat grass in each and every one of the 24 directions, we can collapse every case down to 1 single generalized statement: $<grass \rightarrow [\$x]> \wedge \Uparrow eat(\$x)$ where $\$x$ is a variable term representing 1 of the 24 locations. This statement says "eat at location $x$ when grass is at location $x$".

For a concrete example on how the variable would be used in NARS, let's say NARS has a goal to fill its energy stores $G = <\{ENERGY\} \rightarrow [increasing]>!$. The genetic encoding may generate a sensorimotor contingency containing a variable:

$J_V = <<grass \rightarrow [\$x]> \wedge \Uparrow eat(\$x) \Rightarrow <\{ENERGY\} \rightarrow [increasing]>>.$

When the goal $G$ is processed with an explanation $J_V$, the following subgoal is produced: $\{G, J_V\} \vdash G_V = <<grass \rightarrow [\$x]> \wedge \Uparrow eat(\$x)>!$

If a sensory event fulfilling the variable exists $E_V = <grass \rightarrow [d_{1,0}]>$, then the variable $\$x = d_{1,0}$ can be instantiated in the rest of the statement, such that $G_V$ is interpreted as $<<\{grass\} \rightarrow [d_{1,0}]> \wedge \Uparrow eat(d_{1,0})>$ and, since $E_V$ is true, consequently $\{E_V, G_V\} \vdash \Uparrow eat(d_{1,0})$.

Thus, finally, NARS will execute $\Uparrow eat(d_{1,0})$, a motor operation with a specific argument, using only a single sensorimotor contingency ($J_V$) containing a variable and the current sensory context ($E_V$). If the event $E_V$ was instead that grass was sensed on the western tile, $\$x = d_{-1,0}$ so NARS would instead execute $\Uparrow eat(d_{-1,0})$; if the grass was detected to the south $\$x = d_{0,-1}$, NARS would execute $\Uparrow eat(d_{0,-1})$, and so on. In this way, the variable is very powerful since it allows generalizations, and can condense countless redundant statements down into 1 single generalized statement.

**5.2.2.3 Evolving Instincts Containing Variables in Grid World.** In the case of this experiment, for simplicity, the variable mutation operators were restricted to only selecting $S$ and $M$ for adding/removing variables. Remember that the cardinal direction $dir$ we want to turn into a variable is the property of statement $S$ and the function argument of operation $M$; therefore, the mutation operator was only permitted to replace the atomic terms in those two places. Consequently, applying the mutation operator to a statement like:

$<<grass \rightarrow [d_{1,0}]> \wedge \Uparrow eat(d_{-1,-1}) \Rightarrow <\{ENERGY\} \rightarrow [increasing]>>$

would turn it into:

$<<grass \rightarrow [\$x]> \wedge \Uparrow eat(\$x) \Rightarrow <\{ENERGY\} \rightarrow [increasing]>>$

Where concrete atomic terms $d_{1,0}$ and $d_{-1,1}$ were replaced with a variable $\$x$. Note that, as in the above example, even terms which are different can be replaced with a common variable. This makes the mutation operator different from the variable-introduction NAL reasoning rules, which (for purposes of logical reasoning) require both replaced terms to be the same. Indeed, if both terms were the same, NARS would be able to handle the variable generalization itself using logical inference rules, not necessarily needing evolution to come to that conclusion. Since we are performing evolution, not logical reasoning, there is no need to introduce such a restriction, as the conclusions of evolution do not necessarily need to be "justified" but are meant to be part of an experimental exploration of populations of agents. Evolution does not necessarily care whether or not the individual will survive, only to test various possibilities to improve the fitness of the entire species. In contrast, an individual agent *does* need some logical justification for coming to its conclusions, since the cost of producing invalid conclusions could be devastating for the agent, including task failure and potentially death.

**5.2.2.4 Mutation.** The new Mutation Operators $\Omega$ were as follows:

- *Add new belief:* generate a new belief $<S, \Uparrow M \Rightarrow P>$. using randomly selected $\{S, P\} \in \mathbb{S}$ and $\Uparrow M \in \mathbb{M}$ with random truth value $\langle f, c \rangle$

- *Remove existing belief:* Uniform random select a belief from $\beta$. Remove it.
- *Modify existing belief statement:* Uniform random select a belief from $\beta$. Randomly replace $S$, $P$, or $\Uparrow M$. ***(New)***. If $S/M$ is selected for replacement and contains a variable, make the replacement term also contain a variable.
- ***(New)*** *Toggle variables in existing belief statement:* Uniform random select a belief from $\beta$. If $S$ and $\Uparrow M$ *do not* contain variables, the relevant terms in both are swapped with variables. If $S$ and $\Uparrow M$ contain variables, the relevant terms are swapped with random concrete terms in place of the variables.

During the mutation step, there was an 80% chance to mutate $\beta$.

There was an 80% chance to mutate the content of sentences in $\beta$. In doing so, there was a 1 in 4 chance to either: add a new belief to $\beta$, remove a random belief from $\beta$, to change the term in a belief from $\beta$ with a concrete term, or to toggle (i.e., insert/remove) variable terms from a belief from $\beta$.

When a sensory term was randomly selected from $\mathbb{S}$, there was a 95% chance to uniform-randomly select a term, and 5% chance to select the goal statement ($<\{ENERGY\} \rightarrow [increasing]>$). This was meant to help the system evolve beliefs relevant to its goal, especially in cases where $\mathbb{S}$ was very large and the goal statement had a low chance of being selected.

Then, there was an 80% chance to mutate the truth values of sentences in $\beta$. In doing so, both frequency and confidence each had a 50% chance to be mutated. If they were chosen for mutation, there was a 5% chance to replace the value with a value from range $(0, 1)$, and a 95% chance to perturb the value by a number selected from a normal distribution with mean of zero and standard deviation of 0.1 (i.e., 68% of perturbation values will be in range [-0.1,0.1], 98% of values will be in range [-0.2,0.2], etc.).

After the mutation of $\beta$ was completed, there was an 80% chance to modify the personality parameters $\Pi$. If so, each parameter had a 60% chance to be mutated. If so, there was

a 5% chance to replace the value with a value from its range $[R_L, R_H]$, and a 95% chance to perturb the value by a number selected from a normal distribution with mean of zero and standard deviation of $0.1(R_H - R_L)$.

The allowable range that was permitted for each personality parameter can be see in *Table 4.3*. A new parameter called "generalization confidence" $G_c$ was added, which allowed evolving the confidence value assigned to generalizations made at runtime by NARS.

Table 5.5: NARS Personality Parameter Ranges in Grid World

| Parameter Name | Parameter Range |
|---|---|
| $k$ | $[1, 10]$ |
| $T$ | $[0.51, 1]$ |
| $W$ | $[1, 30]$ |
| $F$ | $[1, 250]$ |
| $C_{temp}$ | $[3, 20]$ |
| $C_{table}$ | $[1, 20]$ |
| $E$ | $[3, 50]$ |
| $\mathcal{T}_J$ | $[0.0000001, 10]$ |
| $\mathcal{T}_G$ | $[0.0000001, 10]$ |
| $G_c$ | $[0.0000001, 0.99999]$ |

### *5.2.3 Experiment 2.1: Evolution of Instincts Containing Variables*

**5.2.3.1 Setup.** This first set of experiments was used to test whether the new variable-introduction mutation operators improved the genetic encoding's effectiveness or not. These experiments tested the evolution of sensorimotor contingencies only, comparing the basic encoding to the encoding extended with the new mutation operators. Runtime learning was disabled, but personality parameters were allowed to evolve.

**5.2.3.2 Results.** For each of the 3 perceptual fields tested (i.e., $3 \times 3$, $5 \times 5$,$7 \times 7$). and 2 experimental conditions (with and without variables), 10 trials were run for 2500 timesteps, and the results were averaged.

The results for the $3 \times 3$ perceptual field experiments can be seen in *Figure 5.4*. Here, the basic encoding and variable-extended encoding achieved very similar performance, with

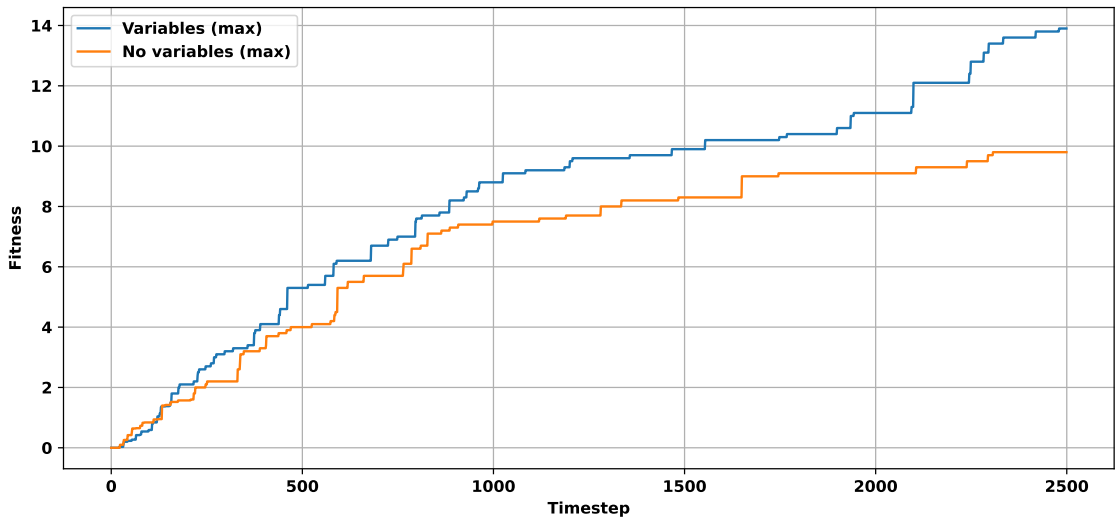


Figure 5.4: Grid World Variable Encoding vs. Basic Encoding Performance Comparison $(3 \times 3)$

the extended encoding slightly outperforming the basic encoding. The average final best agent resulting from the basic encoding scored a fitness of $9.8$, whereas the best agents from the extended encoding scored a fitness of $13.9$. It is not too surprising that the basic encoding was able to keep up with the extended encoding, since the search space of the $3 \times 3$ scenario is relatively small, thus evolution did not need much time to find effective sensorimotor contingencies.

The methods were compared using an exact two-sided permutation test on the difference in performance across independent trials. Across 10 independent trials per method, the genetic encoding extended with instinct-variable mutation operators achieved higher final best-so-far fitness than the basic encoding (mean $13.9$ vs. $9.8$; $\Delta = 4.1$; 95% bootstrap CI $[1.6,\ 6.7]$; exact permutation test $p = 0.00824872$). The area under curve (AUC) was also higher (mean $20{,}974.6$ vs. $17{,}126.2$; $\Delta = 3{,}848.38$; 95% CI $[-498.943,\ 7{,}934.69]$; $p = 0.111163$). With a significance level of $\alpha = 0.05$, the improvement of the extended encoding over the basic encoding in the $3 \times 3$ scenario was statistically significant for final best-so-far fitness, since $p < \alpha$; however, the AUC difference was not statistically significant ($p > \alpha$).

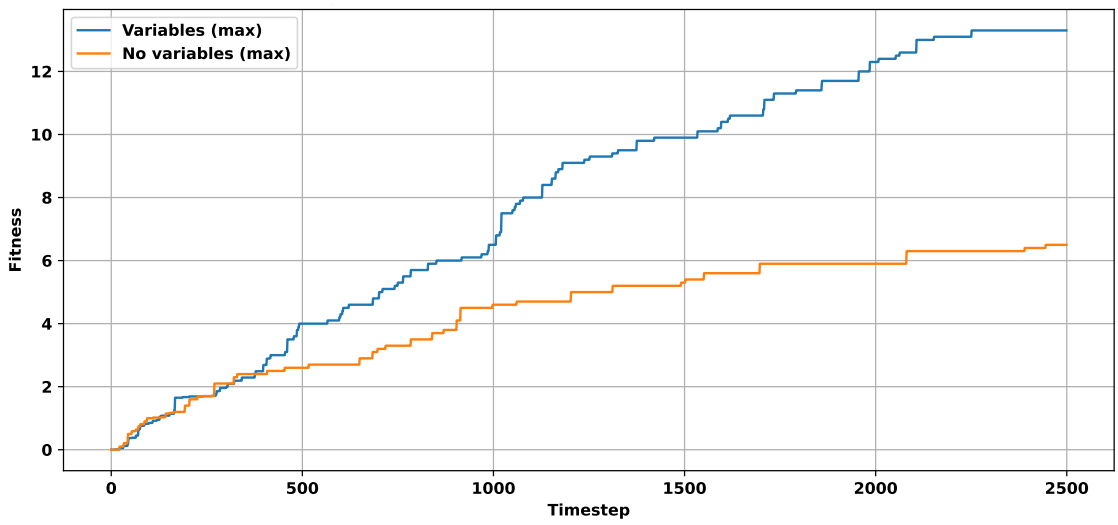


Figure 5.5: Grid World Variable Encoding vs. Basic Encoding Performance Comparison $(5 \times 5)$

The results for the $5 \times 5$ perceptual field experiments can be seen in *Figure 5.5*. Here, the improved effectiveness of the variable-extended encoding over the basic encoding becomes quite evident. The final best agent produced by the variable-extended encoding scored a little more than twice as high on average $(13.3)$ as the best agent produced by the basic encoding (with average score of $6.5$).

The methods were compared using an exact two-sided permutation test on the difference in performance across independent trials. Across 10 independent trials per method, the genetic encoding extended with instinct-variable mutation operators achieved higher final best-so-far fitness than the basic encoding (mean $13.3$ vs. $6.5$; $\Delta = 6.8$; 95% bootstrap CI $[4.4,\ 9.2]$; exact permutation test $p = 0.000108251$). The area under curve (AUC) was also higher (mean $19{,}993.7$ vs. $10{,}996$; $\Delta = 8{,}997.78$; 95% CI $[4{,}086.7,\ 14{,}621]$; $p = 0.00243564$). With a significance level of $\alpha = 0.05$, the improvement of the extended encoding over the basic encoding in the $5 \times 5$ scenario was statistically significant for both final value and AUC, since $p < \alpha$ in both cases.

When moving from the $3 \times 3$ perceptual field to $5 \times 5$, the number of instinct possibilities increases from $10,000$ to $255,792$. This massive increase in the search space increased the

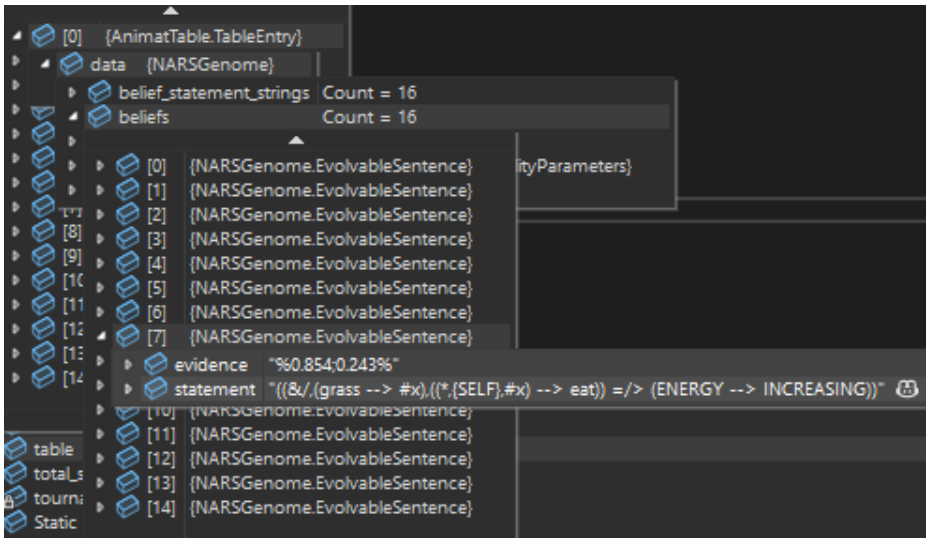


Figure 5.6: Grid World Genome Screenshot

difficulty of the search for the basic genetic encoding. Though it was able to discover *some* useful rules, it was not able to generalize using variables, and so was burdened with the task of generating many redundant pieces of knowledge (e.g., "eat grass here when it's here, and there when it's there, and over there when it's over there"), while avoiding the harmful or useless pieces of knowledge. On the other hand, the genetic encoding that was extended to introduce variables was able to discover both specialized *and* generalized rules. Thus, the best agents were those who found a generalized rule for eating (e.g., "eat grass wherever it is") (see *Figure 5.6*), and could consequently survive in a greater number of situations than agents who could only evolve specialized rules.

The results for the $7 \times 7$ perceptual field experiments can be seen in *Figure 5.7*.

When moving from the $5 \times 5$ perceptual field to $7 \times 7$, the number of possibilities for sensorimotor contingencies increases from $255,792$ to $2,018,400$ possibilities. Because of this increase in the search space, we would expect the basic encoding to perform worse than in scenarios with the smaller perceptual fields. Indeed, the basic encoding struggled greatly, with the final best agent achieving an average of only $4.1$ fitness score. Agents produced by the encoding extended with variables, however, had no trouble at all. Using their generalized rules, they were able to take advantage of the increase in sensory information,

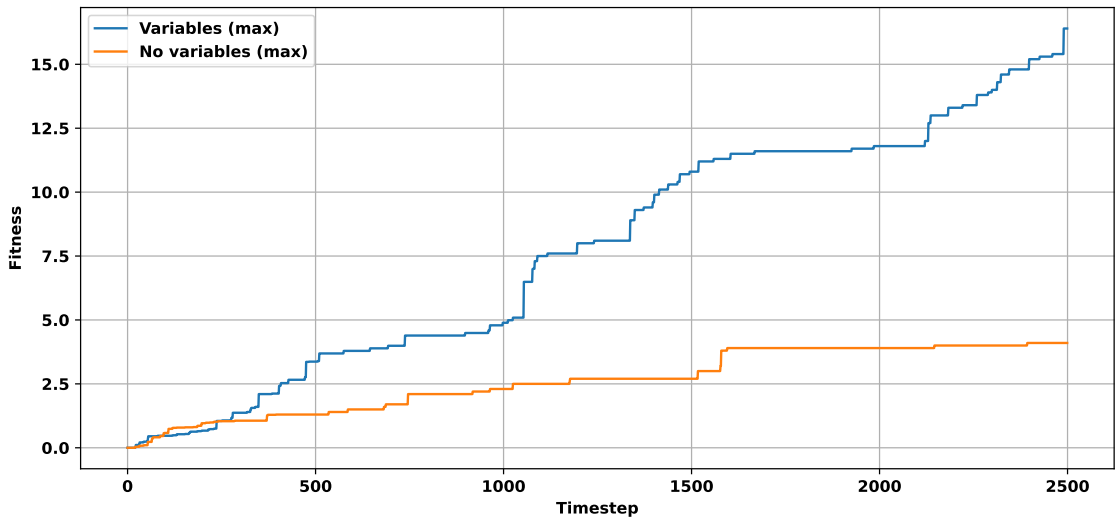


Figure 5.7: Grid World Variable Encoding vs. Basic Encoding Performance Comparison $(7 \times 7)$

with the final best agent scoring an average of $16.4$ fitness score, which is four times better than the basic encoding.

The methods were compared using an exact two-sided permutation test on the difference in performance across independent trials. Across 10 independent trials per method, the genetic encoding extended with instinct-variable mutation operators achieved higher final best-so-far fitness than the basic encoding (mean $16.4$ vs. $4.1$; $\Delta = 12.3$; 95% bootstrap CI $[6.4,\ 18.3]$; exact permutation test $p = 0.000833532$). The area under curve (AUC) was also higher (mean $19{,}344.7$ vs. $6{,}534.07$; $\Delta = 12{,}810.6$; 95% CI $[2{,}846.66,\ 24{,}793.4]$; $p = 0.0264024$). With a significance level of $\alpha = 0.05$, the improvement of the extended encoding over the basic encoding in the $7 \times 7$ scenario was statistically significant for both final value and AUC, since $p < \alpha$ in both cases.

### *5.2.4 Experiment 2.2: Evolution of Generalization Learning*

**5.2.4.1 Setup.** NARS itself can generate statements containing variables on-the-fly, rather than by the evolution of sensorimotor contingencies. The introduction of variables

is called “generalization” or “variable-introduction”, and the successful learning of such generalizations is a critically important function of NARS’ operation.

Generalizations have two benefits. Firstly, a generalization simplifies NARS’ memory contents by condensing many redundant implication statements or “rules” down into one rule/statement. Secondly, a generalization allows NARS to learn and apply a general principle in new situations in which the specific rule was not explicitly learned. This is part of what allows NARS to be a “general-purpose reasoning system”.

Here, NARS was modified to make generalizations during the temporal induction process. For each statement formed by temporal induction, variables were substituted in the relevant places to produce a “generalized” statement. The confidence assigned to this generalized statement was an evolved parameter (in range $(0, 1)$).

**5.2.4.2 Results.** For each of the 3 perceptual fields tested (i.e., $3 \times 3$, $5 \times 5$, $7 \times 7$) and 2 experimental conditions (with and without generalization), 10 trials were run for 1000 timesteps, and the results were averaged.

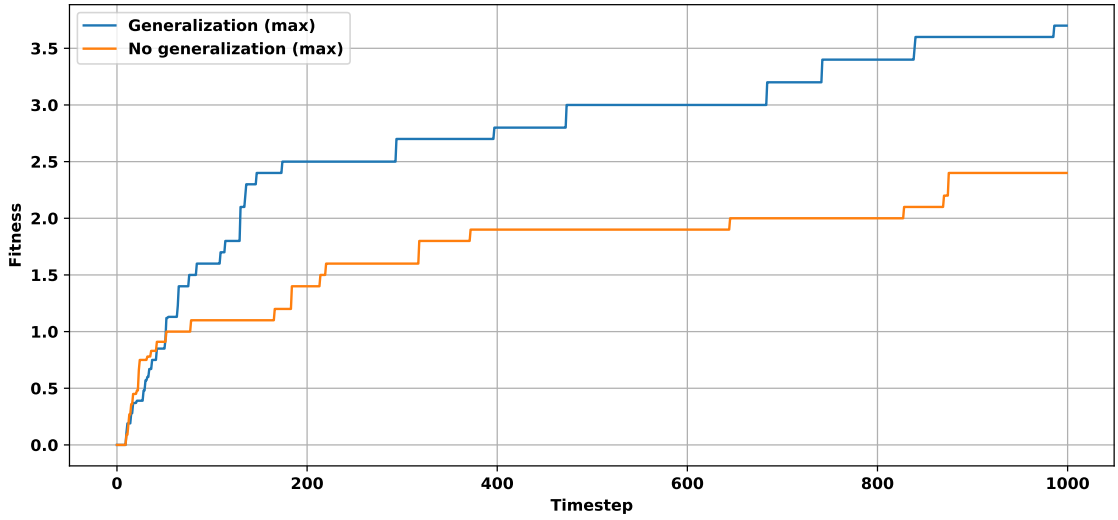


Figure 5.8: Grid World Generalization vs. No-Generalization Performance Comparison $(3 \times 3)$

*Figure 5.8* shows the performance comparison between agents with generalization abilities and agents incapable of generalization in the $3 \times 3$ scenario. Similarly to the results of

the genetic encoding variable experiments, agents with generalization learning performed better (3.7 vs. 2.4).

The methods were compared using an exact two-sided permutation test on the difference in performance across independent trials. Across 10 independent trials per method, the agents learning with generalization abilities achieved higher final best-so-far fitness than the agents learning without generalization abilities (mean 3.7 vs. 2.4; $\Delta = 1.3$; 95% bootstrap CI $[0.5,\ 2.1]$; exact permutation test $p = 0.0120267$). The area under curve (AUC) was also higher (mean 2,763.9 vs. 1,760.34; $\Delta = 1{,}003.56$; 95% CI $[483.337,\ 1{,}545.65]$; $p = 0.00211089$). With a significance level of $\alpha = 0.05$, the improvement of agents with generalization learning over agents without generalization learning in the $3 \times 3$ scenario was statistically significant for both final value and AUC, since $p < \alpha$ in both cases.

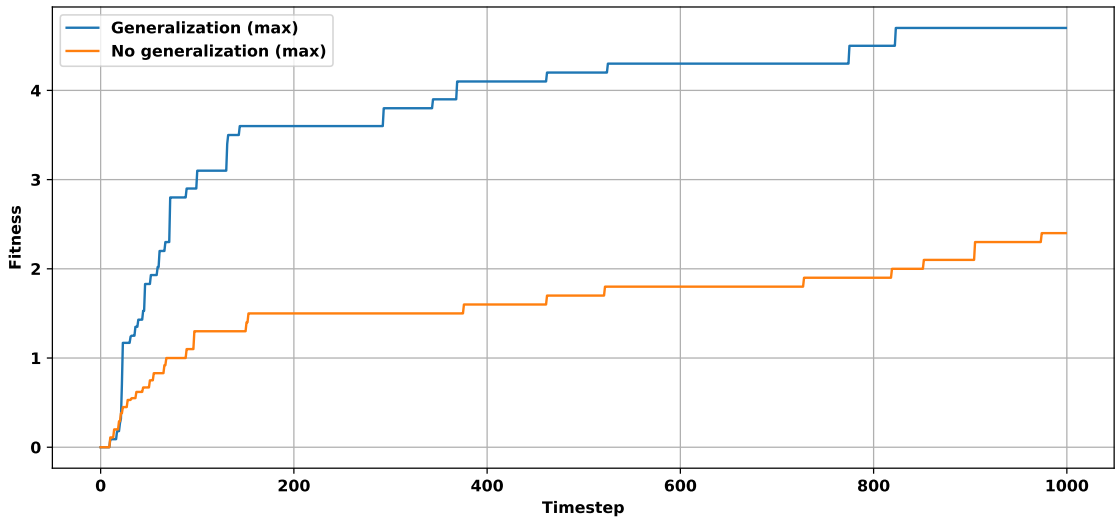


Figure 5.9: Grid World Generalization vs. No-Generalization Performance Comparison $(5 \times 5)$

*Figure 5.9* shows the performance comparison between agents with generalization abilities and agents incapable of generalization in the $5 \times 5$ scenario. Similarly to the results of the variable instinct experiments, there was a stark difference between the performance of agents with and without variables in the $5 \times 5$ scenario. Agents who could learn general rules scored up to 4.7 fitness, whereas agents who could only learn specific rules scored 2.4

fitness. Here, the benefit of allowing generalizations in the learning process becomes more evident.

The methods were compared using an exact two-sided permutation test on the difference in performance across independent trials. Across 10 independent trials per method, the agents learning with generalization abilities achieved higher final best-so-far fitness than the agents learning without generalization abilities (mean $4.7$ vs. $2.4$; $\Delta = 2.3$; 95% bootstrap CI $[1.4,\ 3.3]$; exact permutation test $p = 0.000389703$). The area under curve (AUC) was also higher (mean $3{,}900.7$ vs. $1{,}650.7$; $\Delta = 2{,}250$; 95% CI $[1{,}352.33,\ 3{,}136.88]$; $p = 0.000205677$). With a significance level of $\alpha = 0.05$, the improvement of agents with generalization learning over agents without generalization learning in the $5 \times 5$ scenario was statistically significant for both final value and AUC, since $p < \alpha$ in both cases.

The confidence value assigned to a NARS generalization was encoded in the genome as a personality parameter (in $\Pi$). We can see how this confidence value evolved over the course of the average of three simulations in *Figure 5.10*, along with the $k$-value which modulates the relative strength of confidence values.

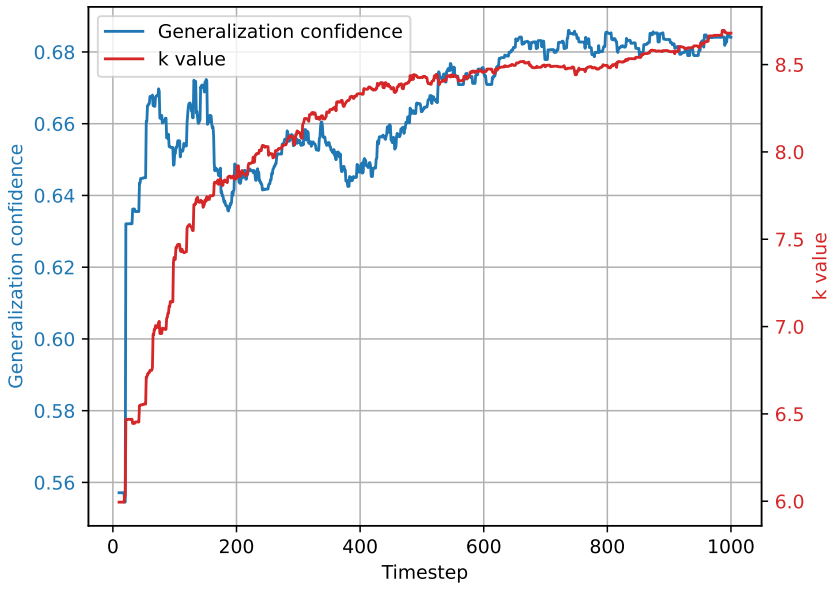


Figure 5.10: Grid World Evolved Generalization Confidence Values

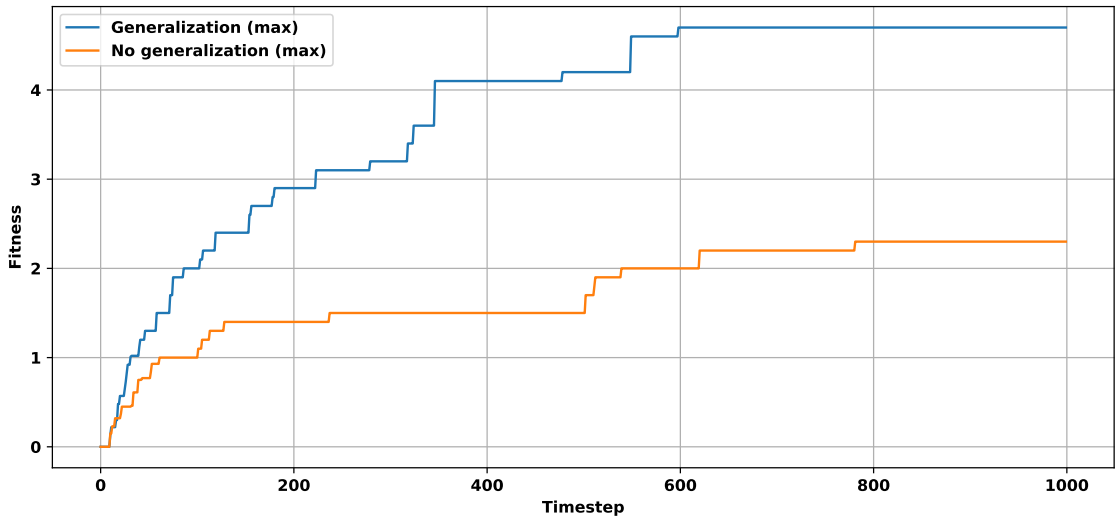


Figure 5.11: Grid World Generalization vs. No-Generalization Performance Comparison $(7 \times 7)$

*Figure 5.11* shows the performance comparison between agents with generalization abilities and agents incapable of generalization in the $7 \times 7$ scenario. The best agents with generalization learning scored an average of $4.7$ fitness by the end, beating the best agents without generalization learning who scored an average of $2.3$ fitness by the end.

The methods were compared using an exact two-sided permutation test on the difference in performance across independent trials. Across 10 independent trials per method, the agents learning with generalization abilities achieved higher final best-so-far fitness than the agents learning without generalization abilities (mean $4.7$ vs. $2.3$; $\Delta = 2.4$; 95% bootstrap CI $[1.5,\ 3.4]$; exact permutation test $p = 0.000248977$). The area under curve (AUC) was also higher (mean $3{,}787.15$ vs. $1{,}740.71$; $\Delta = 2{,}046.44$; 95% CI $[1{,}399.5,\ 2{,}729.27]$; $p = 3.24753 \times 10^{-5}$). With a significance level of $\alpha = 0.05$, the improvement of agents with generalization learning over agents without generalization learning in the $7 \times 7$ scenario was statistically significant for both final value and AUC, since $p < \alpha$ in both cases.

From these results, we can also see that agents who *learned* generalizations did not perform as well as agents who *evolved* with generalizations. This is likely because the learning process introduces many different beliefs at runtime, some of which may conflict

with the "optimal" belief in processing. This means each individual NARS has to spend time testing each of these beliefs for validity, and may not catch on to the optimal strategy in time to survive and perform extremely well in the grid world, especially without a training phase. Evolution, on the other hand, was able to simply prune non-useful beliefs from the genome such that they did not conflict with the more useful beliefs. Agents with evolved beliefs only, and no learning ability, would then utilize those more useful beliefs which have already been battle-tested at some point in evolutionary history by other agents.

Though this explains why the evolved beliefs performed better than learned beliefs, a key challenge in this area of research is still to optimize the learning process of individual NARS agents. In mission-critical situations, it seems inevitable that agents cannot rely solely on runtime learning if they want high performance, since the learning process can be dangerous, time-consuming, and mistake-prone for the individual. Instead, agents can benefit from evolutionary history, using the knowledge passed down from their ancestors to achieve greater success. Alternatively, as we will see in *Chapter 6*, giving the agent a training phase before the testing phase can greatly increase the effectiveness of agents operating purely on learning ability. A "safe" training phase, performed in a copy of the environment, allows the agent to explore, learn, and compare different potential strategies before attempting the actual task in the real environment.

# CHAPTER 6
# EVOLVING COMPOUND LEARNING AND INSTINCTS CONTAINING COMPOUNDS

## 6.1 Extending the Basic Encoding with Compounds

One of the critical features in NARS is the *compound term*. A term is a compound term when it has multiple subterms associated by a *connector*.

For example, consider two first-order statements: $<sky \rightarrow cloudy>$ (i.e., "*the sky is cloudy*") and $<weather \rightarrow rainy>$ (i.e., "*the weather is rainy*"). Without compound terms, NARS is only able to understand each of these concepts separately. NARS can understand what a situation is like when the sky is cloudy, or when the weather is rainy, but not both. By connecting the two statements with a connector, such as conjunction ($AND$, denoted by connector symbol $\wedge$), the two statements can be combined into one: $<<sky \rightarrow cloudy> \wedge <weather \rightarrow rainy>>$, which says "the sky is cloudy *and* the weather is rainy". Upon forming this compound concept, NARS can begin to understand the specific situation where it is *both* cloudy and rainy, i.e., a situation of inclement weather. Furthermore, by using the negation connector ($\neg$), NARS can also understand a similar but different situation where it is cloudy and *not* rainy: $<<sky \rightarrow cloudy> \wedge \neg <weather \rightarrow rainy>>$.

The conjunction term connector allows more detailed "compound" statements, which are useful to help NARS identify different specific situations. Perhaps a NARS agent can only do certain outdoor tasks when the sky is clear without rain (i.e., $<\neg <sky \rightarrow cloudy> \wedge \neg <weather \rightarrow rainy>>$), such as gathering light with solar panels. The agent may prefer other outdoor tasks when the sky is cloudy without rain (i.e., $<<sky \rightarrow cloudy> \wedge \neg <weather \rightarrow rainy>>$), and perhaps the agent would stay inside when the sky is cloudy with rain (i.e., $<<sky \rightarrow cloudy> \wedge <weather \rightarrow rainy>>$). The agent can begin to understand the conditions under which a rainbow forms by observing the rare situation where the sky is sunny *and* it is also raining (i.e., $<\neg <sky \rightarrow cloudy> \wedge <weather \rightarrow rainy>>$).

We have already seen a concrete useful example of compounds in the sensorimotor contingency. The subject term of the contingency statement $<(S, \Uparrow M) \not\Rightarrow P>$ is $(S, \Uparrow M)$, which is a compound term containing two subterms ($S$ and $\Uparrow M$) connected by temporal sequential conjunction (which uses connector symbol ",", rather than "$\wedge$"). The sequential conjunction $(S, \Uparrow M)$ indicates that "first event $S$ occurs, *and then* event $\Uparrow M$ occurs afterwards".

To extend the genetic encoding with compound terms, we will introduce two new genetic operators to $\Omega$ which mutate compound terms. For simplicity, we will ignore compound motor operators (i.e., we will ignore $\Uparrow M$ during mutation) and focus solely on the sensory aspects (i.e., $S$ and $P$). We will connect events using *parallel conjunction* (denoted by connector symbol ";") which indicates that two or more events are happening at the same moment in time.

Concretely, the first new mutation operator is called *Compound*, and functions by selecting a term from the sensorimotor contingency statement (e.g. $S$) and adds a term to produce a larger compound. For example, the Compound mutation takes a statement term $S = A$ and turns it into a compound statement $S = (A; B)$. If $S = (A; B)$, the mutation operator would turn it into $S = (A; B; C)$, and so on to create compounds of higher complexity (since we are using *parallel* conjunction, order of $A$,$B$,$C$, etc. does not matter here).

The second mutation operator, *Decompound*, has the reverse effect, by selecting a term from the sensorimotor contingency statement (e.g. $S$) and removing one element from the compound. For example, Decompound mutation would turn $S = (A; B)$ into either $S = A$ or $S = B$, depending on which term it removed. If there is only one element, such that $S$ is not a compound term, then the mutation operator has no effect.

Finally, the *Modify Belief* mutation operator from the basic encoding has to be changed to account for compound terms. When $S$ is selected for modification, rather than replacing $S$ entirely with a new sensory statement from $\mathbb{S}$, instead the mutation operator randomly

selects one of the subterms from $S$ and replaces that subterm with a new sensory statement from $\mathbb{S}$.

To evolve the agent's runtime compound learning ability, we can add additional values to $\Pi$ corresponding to the agent's Personality Parameters that dictate compound learning. For example, we can evolve the system's ability to create compound events of different sizes; a mutation can flip a Boolean switch on/off that permits the system to form compound events of size 2, 3, 4, etc.

The next section explores how these new genetic operators could be used in BlocksWorld, a classic AI logical planning puzzle which involves compound world states.

## 6.2 Experiment Domain #3: BlocksWorld

### *6.2.1 Experiment Setup*

**6.2.1.1 Environment.** BlocksWorld is a puzzle game in which, given labeled blocks stacked on a table, the agent has to figure out how to transform a configuration of blocks into a goal configuration. The environment consisted of certain labeled blocks (e.g., "Block A", "Block B', "Block C", etc.) in a certain configuration (e.g., "A is on B, B is on the table") (see *Figure 6.1*). The agent had two possible actions: it could *stack* one block on top of another, and it could *unstack* one block and place it on the table.

This problem is a classical AI planning logic problem, since the agent has to determine a chained plan to get from one state to the next, and the plan involves multiple incremental steps. Most often this problem is addressed with a version of first-order predicate logic that can handle actions and changing world states. Here, the problem is addressed with non-axiomatic logic and reasoning.

The world was divided into a $4 \times 4$ grid (see *Figure 6.2*, successfully solved puzzles are circled in red and highlighted green). Each grid cell contained a single BlocksWorld instance and a NARS agent assigned to that instance. All agents had the same amount

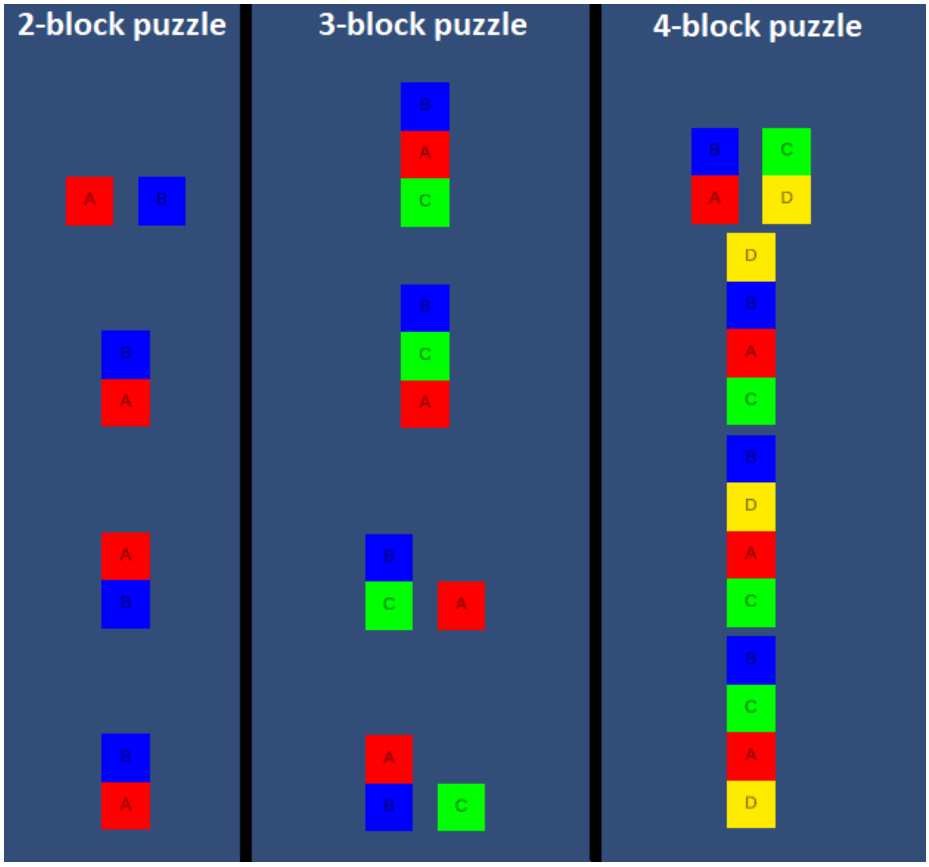


Figure 6.1: BlocksWorld Puzzle Sizes

of time to attempt a problem, after which they would either move onto the next problem instance (here called "episode") or, once all the agents had completed all their assigned problems/episodes, the "generation" was ended, and the next generation was spawned.

Each episode contained 1 problem instance. A problem instance was a randomly generated goal state, which was put into the NARS genome as $\Gamma$, and the environment started in its worst-case state, such that all the blocks were in an incorrect configuration. The agent then had to move one block at a time to transform the world into the correct configuration. The more blocks in the problem, the more difficult it was to solve, since more blocks meant more possible world states through which the agent had to navigate.

**6.2.1.2 Agents.** At each simulation tick, NARS was run for 4 working cycles with no inputs, to help forget the previous world state observed. Then, the goals of $\Gamma$ were fed into

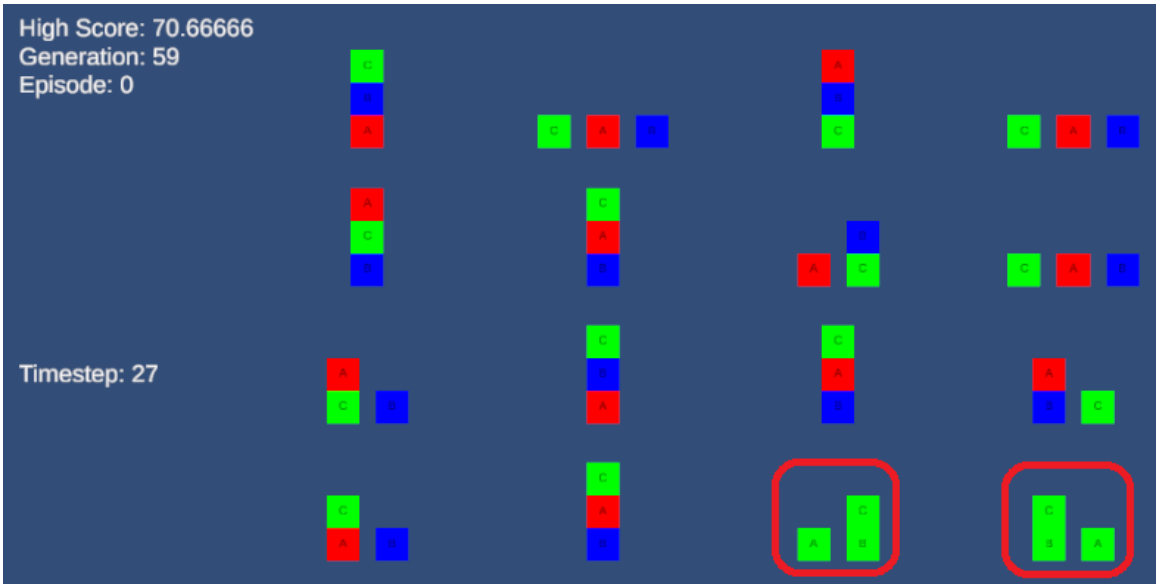


Figure 6.2: BlocksWorld Screenshot

the system, as well as their compounds (parallel conjunction) up to size 3. Then, NARS was run for 4 more working cycles; at each cycle, the sensory statements making up the world state were fed one-by-one into NARS, as well as their compounds (parallel conjunction) up to size 3.

The agent was allowed to execute 1 motor operation per simulation tick. The motor operation was determined by evaluating the desire-values of all the system's motor operations, and selecting the one with the highest expected desire-value. A motor operation only succeeded, changing the world state, for valid moves; if the agent attempted to execute an invalid move, such as trying to unstack a block which is covered by another block, then the move failed and the world state remained unchanged for the simulation tick. If the agent reached the goal state during an episode, the agent could no longer execute any moves.

### *6.2.2 Evolutionary Algorithm*

Two reproductive pools were utilized in the BlocksWorld experiments: a Hall of Fame pool (size 100), containing the top 100 scoring agents of all time, and a Recent Population pool (size 100), containing the 100 most recently evaluated agents. To generate a new agent,

there was a 50/50 chance to generate it from either the Hall of Fame or Recent Population reproductive pool. There was a 50/50 chance to perform sexual reproduction (whereby 2 agents were selected and recombinated, and one offspring was chosen as the new agent) or asexual reproduction (whereby 1 agent was selected and mutated to generate the new agent).

After each "generation", the $4 \times 4$ population was evaluated for fitness, added to the reproductive pools, and then a new $4 \times 4$ population was instantiated from the reproductive pools.

Within a generation, each NARS genome was tested to solve 3 separate problem instances (episodes). At each episode, a new NARS agent was spawned using the NARS genome. Then, it was given a certain number of simulation ticks to solve the problem instance. After an episode $i$, the agent received an episode fitness score $f_i$ for the problem.

$$f_i = UW$$

where $U$ was the number of unique BlocksWorld states the agent had reached. If the goal state had not been reached, $W$ was the number of blocks in the correct configuration; if the goal state had been reached, $W$ was the number of blocks in the correct configuration times 10.

This fitness function was meant to encourage the creation of agents who (1) could navigate to many world states ($U$), so as to give a better chance of navigating to the goal state, and (2) tended to end up with their blocks in their correct configuration ($W$), near or at the goal state.

The final fitness score $\mathcal{F}$ for the genome was the average $f_i$ score. For $E = 3$ episodes:

$$\mathcal{F} = \frac{1}{E} \sum_{i=1}^{E} f_i$$

**6.2.2.1 Genetic Encoding.** The sensorymotor sets for the BlocksWorld puzzles were as follows.

The world state was described by detecting the state of each block according to the relevant sensations in $\mathbb{S}$. The sensory set can be seen in *Table 6.1*. The first two sensations described what was *underneath* each block: a block could be on sitting on top of another block ($<(*, blockLabel1, blockLabel2) \rightarrow On>$), or sitting on the table ($<blockLabel \rightarrow OnTable>$). The third sensation described when the block had nothing *on top* of it ($<blockLabel \rightarrow Clear>$).

Table 6.1: Compact BlocksWorld Sensory Set $\mathbb{S}$

| | |
|---|---|
| $\mathbb{S}$ | $<blockLabel \rightarrow OnTable>$ |
| | $<(*, blockLabel1, blockLabel2) \rightarrow On>$ |
| | $<blockLabel \rightarrow Clear>$ |

There were two motor operations NARS could perform: $\Uparrow STACK$ and $\Uparrow UNSTACK$ (see *Table 6.2*). The $\Uparrow STACK(blockLabel1, blockLabel2)$ operation took two arguments, the two blocks to stack. The operation would result in $blockLabel1$ being put onto $blockLabel2$, as long as $blockLabel1$ could be picked up and $blockLabel2$ could be placed onto. In other words, $\Uparrow STACK(blockLabel1, blockLabel2)$ only succeeded when *both* $<blockLabel1 \rightarrow Clear>$ *and* $<blockLabel2 \rightarrow Clear>$. Since it requires 2 conditions, this is where a conjunction compound would help. The $\Uparrow UNSTACK(blockLabel)$ operation took only one argument, the block to unstack. The operation would result in the block being placed onto the table (i.e., $<blockLabel \rightarrow OnTable>$), as long as it was possible to unstack the block, meaning it had no block on top (i.e., $<blockLabel \rightarrow Clear>$) *and* it was stacked on another block (i.e., $<(*, blockLabel, x) \rightarrow On>$ for any other block $x$).

Table 6.2: Compact BlocksWorld Motor Set $\mathbb{M}$

| | |
|---|---|
| $\mathbb{M}$ | $\Uparrow STACK(blockLabel1, blockLabel2)$ |
| | $\Uparrow UNSTACK(blockLabel)$ |

For the 3-block puzzle, the fully expanded sensor set can be seen in *Table 6.3* and the fully expanded motor set can be seen in *Table 6.4*.

Table 6.3: Full BlocksWorld Sensory Set $\mathbb{S}$ for 3-Block Puzzle

| | |
|---|---|
| $\mathbb{S}$ | $<A \rightarrow OnTable>$ |
| | $<B \rightarrow OnTable>$ |
| | $<C \rightarrow OnTable>$ |
| | $<A \rightarrow Clear>$ |
| | $<B \rightarrow Clear>$ |
| | $<C \rightarrow Clear>$ |
| | $<(*, A, B) \rightarrow On>$ |
| | $<(*, A, C) \rightarrow On>$ |
| | $<(*, B, A) \rightarrow On>$ |
| | $<(*, B, C) \rightarrow On>$ |
| | $<(*, C, A) \rightarrow On>$ |
| | $<(*, C, B) \rightarrow On>$ |

Table 6.4: Full Blocksworld Motor Set $\mathbb{M}$ for 3-Block Puzzle

| | |
|---|---|
| $\mathbb{M}$ | $< \Uparrow STACK(A, B)>$ |
| | $< \Uparrow STACK(A, C)>$ |
| | $< \Uparrow STACK(B, A)>$ |
| | $< \Uparrow STACK(B, C)>$ |
| | $< \Uparrow STACK(C, A)>$ |
| | $< \Uparrow STACK(C, B)>$ |
| | $< \Uparrow UNSTACK(A)>$ |
| | $< \Uparrow UNSTACK(B)>$ |
| | $< \Uparrow UNSTACK(C)>$ |

Four (4) new Personality Parameters were added to $\Pi$ to permit the evolution of NARS runtime compounding. First was a float $C_c \in (0, 1)$ which determined the confidence value of the contingency formed by NARS. Next were three Booleans, $C_1$, $C_2$, and $C_3$, which, when $true$, allowed NARS to form contingencies where $S$ had 1 component (i.e., the basic contingency), 2 components (e.g., $(A; B)$), and 3 components (e.g., $A; B; C$) respectively.

The allowable range that was permitted for each personality parameter can be see in *Table 6.5*.

Table 6.5: NARS Personality Parameter Ranges in BlocksWorld

| Parameter Name | Parameter Range |
|---|---|
| $k$ | $[1, 10]$ |
| $T$ | $[0, 1]$ |
| $W$ | $[1, 30]$ |
| $C_{temp}$ | $[3, 20]$ |
| $C_{table}$ | $[1, 20]$ |
| $E$ | $[1, 50]$ |
| $\mathcal{T}_J$ | $[0.0000001, 10]$ |
| $\mathcal{T}_G$ | $[0.0000001, 10]$ |
| $\mathcal{C}_c$ | $[0.0000001, 0.99999]$ |
| $\mathcal{C}_1$ | $\{false,\ true\}$ |
| $\mathcal{C}_2$ | $\{false,\ true\}$ |
| $\mathcal{C}_3$ | $\{false,\ true\}$ |

**6.2.2.2 Using NAL Compounds in BlocksWorld.** Consider a situation where NARS wants to use an operation $\Uparrow STACK(A, B)$ to stack Block A onto Block B. In order for this operation to succeed, *both* Block A must be clear, so that NARS can successfully pick it up, and Block B must be clear, so that NARS can successfully place Block A on top of Block B. Here, we see a use for a conjunction compound term: both event $<A \rightarrow Clear>$ must be true and event $<B \rightarrow Clear>$ must be true at the same time, which can be represented by a parallel conjunction $<<A \rightarrow Clear>;<B \rightarrow Clear>>$. With compounds, NARS can have a sensorimotor contingency like:

$$<(<A \rightarrow Clear>;<B \rightarrow Clear>), \Uparrow STACK(A, B) \Rightarrow <(*, A, B) \rightarrow On>>$$

In such a case, the $S$ part of the sensorimotor contingency $<S, \Uparrow M \Rightarrow P>$ is a conjunction, a compound term, of multiple events pulled from the sensory set $\mathbb{S}$.

Without compounds, NARS can only understand or process one of these conditions at a time. It will never know the full condition required for stacking one block on another,

though it might still be able to succeed in the operation, either by blind luck, or by trying to make both conditions true but following separate reasoning paths, e.g. if it derives conclusions from both contingencies:

$$<<A \rightarrow Clear>, \Uparrow STACK(A,B) \Rightarrow <(*,A,B) \rightarrow On>>$$

and

$$<<B \rightarrow Clear>, \Uparrow STACK(A,B) \Rightarrow <(*,A,B) \rightarrow On>>$$

However, the compound term simplifies the situation and makes the sensory contexts tethered to each other. It allows NARS to view the "fuller picture" of its current sensory context, and allows NARS to require multiple conditions all hold true simultaneously (in the case of conjunction).

**6.2.2.3 Evolving Instincts Containing Compounds in BlocksWorld.** In the first BlocksWorld experiment, the evolution of sensorimotor contingencies containing compounds was tested. The basic genetic encoding was extended by implementing *Compound* and *Decompound* mutation operators. Both $S$ and $P$ (of sensorimotor contingency $<(S, \Uparrow M) \Rightarrow P>$) were permitted to be compounded in mutation using parallel conjunction; $\Uparrow M$ was left as a single motor operation which could not be compounded. The compounds were limited to contain at maximum 3 subterms, and at minimum 1 subterm.

In the second BlocksWorld experiment, sensorimotor contingencies were not evolved, instead NARS was permitted to form its own contingencies at runtime using temporal induction. In this scenario, the system was formed compounds (of parallel conjunction) during the temporal induction process. The length of the compound was determined by the evolved parameters $C_1$, $C_2$, and $C_3$ in $\Pi$.

**6.2.2.4 Mutation.** The new Mutation Operators $\Omega$ were as follows:

- *Add new belief:* generate a new belief $<S, \Uparrow M \Rightarrow P>$. using randomly selected $\{S, P\} \in \mathbb{S}$ and $\Uparrow M \in \mathbb{M}$ with random truth value $\langle f, c \rangle$

- *Remove existing belief:* Uniform random select a belief from $\beta$. Remove it.

- *Modify existing belief statement:* Uniform random select a belief from $\beta$. Randomly select $S$, $P$, or $\Uparrow M$ for replacement. If $S/P$ is selected for replacement and is a single statement, replace it with random sensory statement from $\mathbb{S}$. ***(New)*** If $S/P$ is selected for replacement and is a compound term, replace one of the subterms (randomly selected) with a random sensory statement from $\mathbb{S}$.

- ***(New)*** *(De)Compound sensory context in existing belief statement:* Uniform random select a belief from $\beta$. Randomly select $S$ or $P$ for (de)compounding. If the selected term is a single statement, turn it into a 2-component compound (parallel conjunction) using a randomly selected sensory statement from $\mathbb{S}$. If the selected term is the maximum-size compound (here, 3), remove one random subterm from the compound (i.e., "decompound"). If the selected term contains more than 1 subterm and less than the maximum, there is a 50/50 chance to add a new randomly selected sensory statement from $\mathbb{S}$ into the compound, or to remove a randomly selected subterm from the compound.

There was an 80% chance to mutate $\beta$. In doing so, there was a 1 in 4 chance to either: add a new belief to $\beta$, remove a random belief from $\beta$, to change the term in a belief from $\beta$ with a concrete term, or to perform (de)compounding on a belief from $\beta$.

### *6.2.3 Experiment 3.1: Evolution of Instincts Containing Compounds*

**6.2.3.1 Setup.** This first set of experiments tested the evolution of instinctual sensorimotor contingencies containing compounds.

For each BlocksWorld puzzle size (i.e., 2-block, 3-block, 4-block), 10 trials were run under 2 conditions: the first condition was using the basic encoding, and the second condition

was using the encoding extended with compound terms in $S$ and $P$. Evolved compounds could contain at maximum three (3) subterms and at minimum one (1) subterm, which were sized during mutation using the new Compound and Decompound mutation operators.

Each trial was run for 100 generations/populations, with 3 episodes per generation (300 total episodes). Each 2-block episode ran for 35 simulation ticks, each 3-block episode ran for 100 simulation ticks, and each 4-block episode ran for 200 simulation ticks.

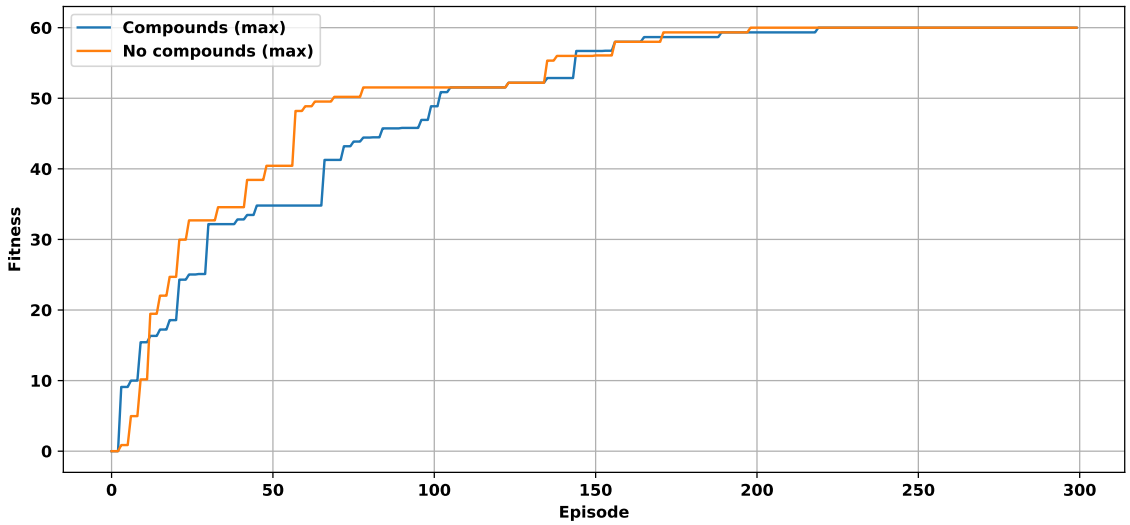


Figure 6.3: BlocksWorld Compound Encoding vs. Basic Encoding Performance Comparison (2-Block Puzzle)

**6.2.3.2 Results.** A comparison of the two genetic encoding methods on the 2-block puzzle can be seen in *Figure 6.3*. Ultimately both the basic encoding and the extended encoding achieved the maximum possible fitness for this puzzle, with a final score of 60 (since there were only 3 possible states in the 2-block puzzle, and 2 possible goal configurations, thus the maximum fitness for an episode was $f_i = 3 * 2 * 10 = 60$). Since fitness $\mathcal{F}$ for a genome was calculated using the average of all episodes, this result indicates that the best agent from both encodings managed to solve all the puzzles in 3 consecutive episodes.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents with compound instinct evolution achieved the same final best-so-far fitness as

the agents without compound instinct evolution (mean $60.0$ vs. $60.0$; $\Delta = 0$; 95% bootstrap CI $[0,\ 0]$; exact permutation test $p = 1.0$). The area under the curve (AUC) was slightly lower with compound instinct evolution (mean $14{,}717.5$ vs. $15{,}258.4$; $\Delta = -540.9$; 95% bootstrap CI $[-2{,}830.78,\ 1{,}672.52]$; exact permutation test $p = 0.660$). With a significance level of $\alpha = 0.05$, no statistically significant difference was detected in the 2-block scenario for either the final value or AUC because $p > \alpha$ in both cases.

The lack of difference in outcomes between the extended encoding and basic encoding in the 2-block puzzle makes sense, since the puzzle configurations of the 2-block puzzle are simple enough that there is no need to use compounds to determine a valid move. The fact that the compound-extended encoding struggled ever-so-slightly more than the basic encoding also makes sense, since the genome search space was larger yet compounds did not provide a tangible benefit.

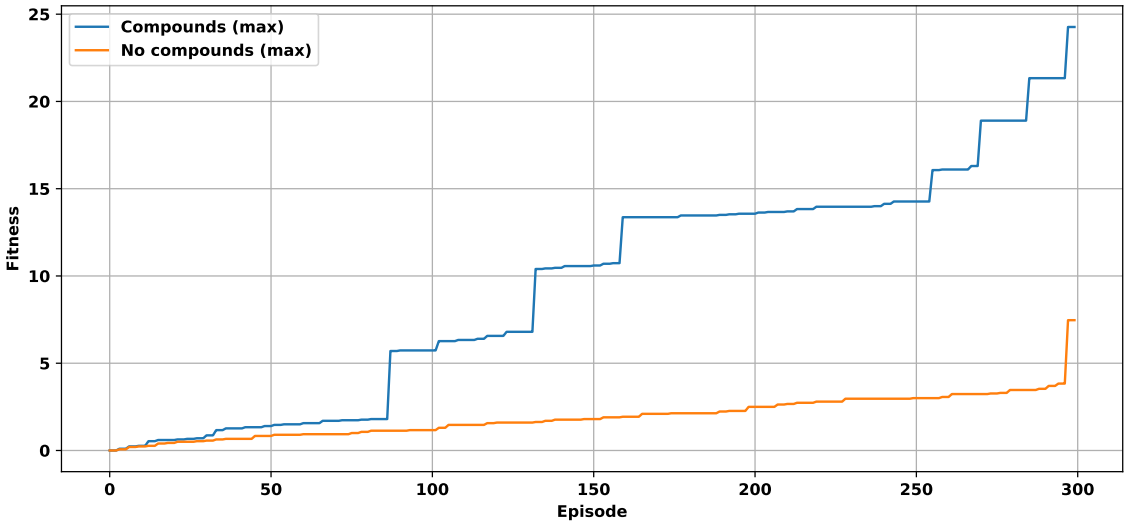


Figure 6.4: BlocksWorld Compound Encoding vs. Basic Encoding Performance Comparison (3-Block Puzzle)

A comparison of the two genetic encoding methods on the 3-block puzzle can be seen in *Figure 6.4*. The 3-block puzzle contained 13 possible states. Here we can see that the extended encoding greatly outperformed the basic encoding ($24.27$ vs. $7.47$). Since the fitness score of the extended encoding was much greater than 10, we know that the best

agents produced by the extended encoding were able to solve a 3-block puzzle. On the other hand, the best agents produced by the basic encoding in all but one trial were unable to solve the 3-block puzzles. This result indicates that allowing compounds during evolution can provide some benefit, allowing agents to solve the 3-block puzzle faster than the basic encoding.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents with compound instinct evolution achieved higher final best-so-far fitness than the agents without compound instinct evolution (mean $24.27$ vs. $7.47$; $\Delta = 16.80$; 95% bootstrap CI $[-0.67,\ 35.4]$; exact permutation test $p = 0.1185$). The area under the curve (AUC) was also higher (mean $2{,}825.67$ vs. $572.77$; $\Delta = 2{,}252.9$; 95% bootstrap CI $[382.50,\ 4{,}492.12]$; exact permutation test $p = 0.0213$). With a significance level of $\alpha = 0.05$, the improvement was statistically significant for AUC ($p < \alpha$) though no statistically significant difference was detected for the final value ($p > \alpha$) in the 3-block scenario. This means on average, the compound instinct evolution spent more generations at higher best-so-far fitness than the basic instinct evolution. Therefore, though we cannot conclude compound evolution reliably produced better solutions than the basic encoding by the end of the search process, compound evolution did achieve good solutions earlier and more consistently throughout the evolutionary process for the 3-block scenario.

A comparison of the two genetic encoding methods on the 4-block puzzle can be seen in *Figure 6.5*. The 4-block puzzle contained 73 possible states. Neither encoding was able to produce an agent capable of solving the 4-block puzzle, though the extended encoding still performed better than the basic encoding in the trials ($3.0$ vs. $1.77$).

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents with compound instinct evolution achieved higher final best-so-far fitness than the agents without compound instinct evolution (mean $3.0$ vs. $1.77$; $\Delta = 1.23333$; 95%

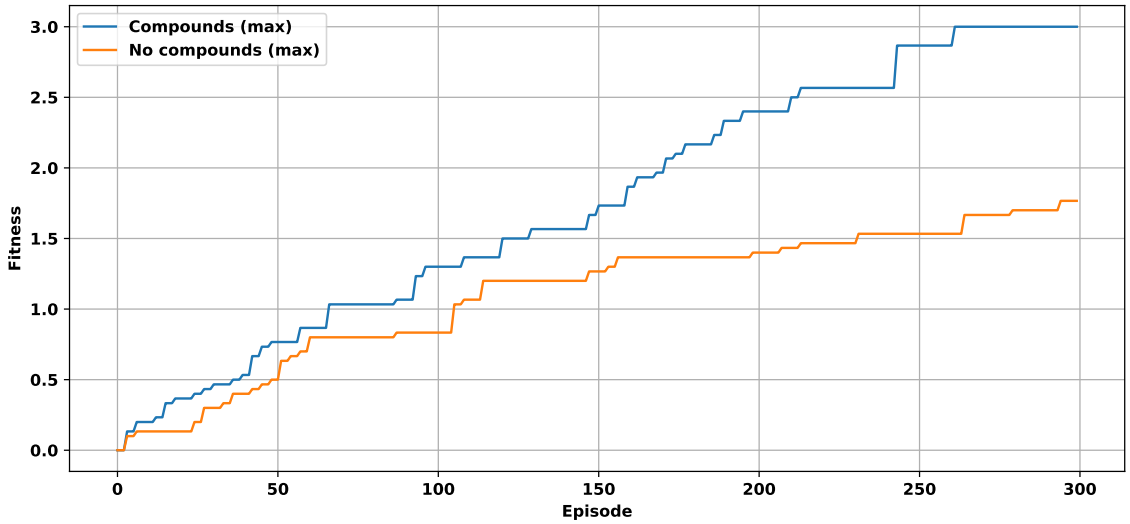


Figure 6.5: BlocksWorld Compound Encoding vs. Basic Encoding Performance Comparison (4-Block Puzzle)

bootstrap CI $[-0.27,\ 3.00]$; exact permutation test $p = 0.182792$). The area under the curve (AUC) was also higher (mean $522.3$ vs. $331.7$; $\Delta = 190.583$; 95% bootstrap CI $[-38.17,\ 447.57]$; exact permutation test $p = 0.155$). With a significance level of $\alpha = 0.05$, no statistically significant difference was detected in the 4-block scenario for either the final value or AUC. This means we cannot say that there was a reliable positive effect from compound encoding over basic encoding in the 4-block scenario.

Now let us consider why the extended encoding begins to struggle as the problem size increases to 4 blocks. A major issue in the compound instinct evolution is "combinatorial explosion". As $\mathbb{S}$ increases in size, the number of possible sensory combinations/compounds that can be formed increases dramatically. More combinations means a larger search space.

The extended encoding forms compounds of statements from $\mathbb{S}$. Consider the size of the sensory set $\mathbb{S}$. The 2-block puzzle has 6 statements in $\mathbb{S}$, the 3-block has 12 statements in $\mathbb{S}$, and the 4-block has 20 statements in $\mathbb{S}$. Since only parallel conjunctions are considered in this experiment, the order of statements in the combination/compound does not matter

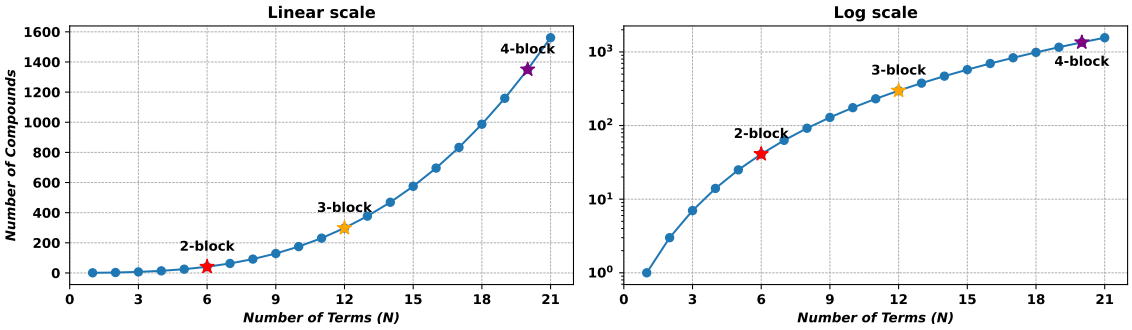


Figure 6.6: Size of the Search Space of Order-Independent Compound Terms Containing up to 3 Subterms

(i.e., the compounds are order-independent). The number of all possible order-independent compounds, up to size $L$, can be calculated using a summation of combinations:

$$\sum_{k=1}^{L} \binom{|\mathbb{S}|}{k}$$

In other words, it is the sum of all combinations of every size compound from 1 to $L$ (here $L = 3$). The relationship between number of elements and number of possible order-independent compound terms up to size 3 can be viewed in *Figure 6.6*, with the N-block puzzles denoted by stars on the logarithm-scale graph. For $L = 3$, the number of possible compounds is $41$ for the 2-block puzzle, $298$ for the 3-block puzzle, and $1,350$ for the 4-block puzzle.

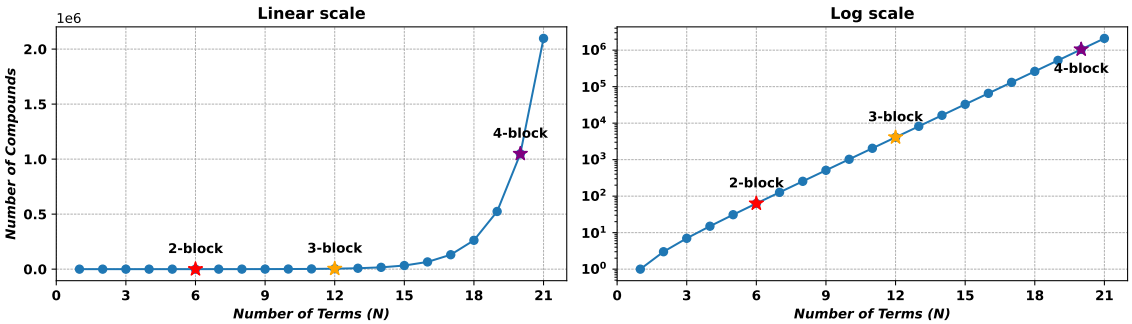


Figure 6.7: Size of the Search Space of Order-Independent Compound Terms

Although in this experiment the maximum compound size was restricted to 3, consider if compounds had been allowed to grow to any size unrestricted, up to a maximum-size compound $L = |\mathbb{S}|$ containing all sensory statements. The number of possible order-independent compounds up to size $|\mathbb{S}|$ can be calculated using a summation of combinations:

$$\sum_{k=1}^{|\mathbb{S}|} \binom{|\mathbb{S}|}{k} = 2^{|\mathbb{S}|} - 1$$

The relationship between number of elements and number of possible order-independent compound terms up to size $|\mathbb{S}|$ can be viewed in *Figure 6.7*, with the N-block puzzles denoted by stars on the logarithm-scale graph. In the case that we had allowed compounds to grow in size unconstrained, the 2-block puzzle would have had $63$ possible compounds, the 3-block puzzle would have had $4095$ possible compounds, and the 4-block puzzle would have a massive $1,048,575$ possible compounds.

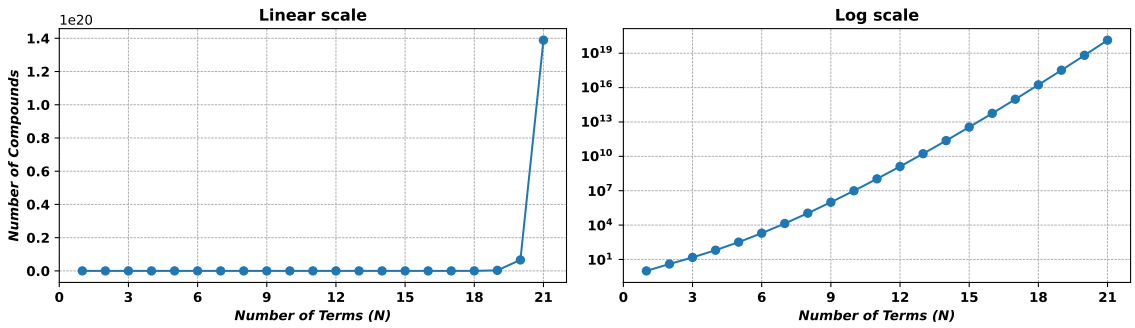


Figure 6.8: Size of the Search Space of Order-Dependent Compound Terms)

Finally, let us consider what happens to the number of possible compounds if we use an order-dependent term connector (like sequential conjunction), in which the order of components in the compound *does* matter. Assuming again that we allow compounds to grow up to the maximum size $L$, we can calculate the number of possible order-dependent compounds using a summation of permutations:

$$\sum_{k=1}^{L} \frac{|\mathbb{S}|!}{(|\mathbb{S}| - k)!}$$

Consider if we had allowed sequential conjunction of unconstrained size ($L = |\mathbb{S}|$) in BlocksWorld; in the 4-block puzzle for example, the number of possible order-dependent compounds would be a gargantuan $6.61 \times 10^{19}$, for the 3-block puzzle $1.30 \times 10^{9}$, and even the 2-block puzzle would have a large number of possibilities, $1,956$ possible order-dependent compounds. The relationship between number of elements and number of possible order-dependent compound terms up to size $L = |\mathbb{S}|$ can be viewed in *Figure 6.8*. The number of possible compounds explodes. In light of this, for practical purposes it is a good idea to limit the maximum size of compound that can be evolved by the genetic encoding. With $L = 3$, the 4-block puzzle would only have $7,240$ possible order-dependent compounds, the 3-block puzzle would only have $1,464$, and the 2-block puzzle would only have $156$.

We can increase the efficiency of forming useful sensorimotor contingencies containing compounds in two ways: allowing variables in addition to compounds during instinct evolution (collapsing redundancies), and/or by allowing the agent to form contingencies containing compounds at runtime (limiting the search space to compound states which are *actually experienced* by the agent).

**6.2.3.3 Putting It All Together: Evolving Instincts Containing Both Compounds and Variables.** The results indicate that the evolutionary algorithm struggled as more blocks were added to the BlocksWorld puzzle. Similarly to the GridWorld problem, this BlocksWorld problem requires redundant statements where general knowledge would suffice. Therefore, without variables, as the number of blocks increased, more redundant statements had to be evolved in order to cover the more possible scenarios (e.g., instincts describing how to put A on B, and B on C, and A on C, and A on D, etc.). As explored in *Chapter 5*, we can use a variable-introduction mutation operator to evolve generalizations, which cover multiple redundant statements using one single "general" statement through the use of variables.

The following graphs show the results of the genetic encoding that was extended to allow *both* variables *and* compounds in the NARS instincts. The results show the average of 10 trials under each experimental condition.

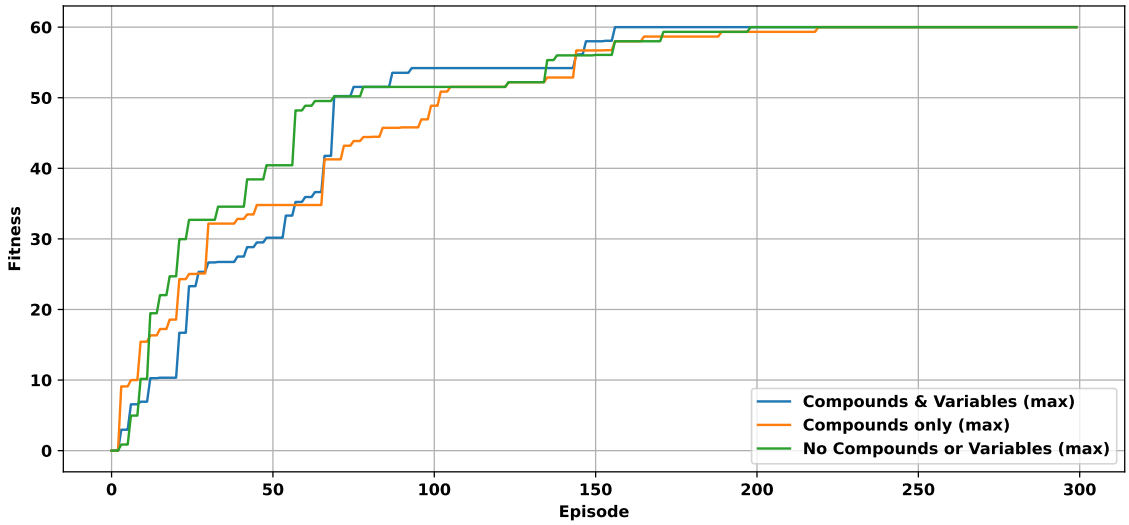


Figure 6.9: BlocksWorld Compound-and-Variable-Encoding vs. Base Encodings Performance Comparison (2-Block Puzzle)

In the 2-block puzzle, there was no major difference between the encoding using regular compounds and the encoding using both compounds and variables (see *Figure 6.9*). Both encodings produced agents able to solve the puzzles in three consecutive episodes, and there was no significant difference between them. Therefore, in the 2-block scenario, evolving compounds with variables did not provide a measurable advantage over the compound-only encoding or basic encoding.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents with compound + variable instinct evolution achieved the same final best-so-far fitness as the agents with compound-only instinct evolution (mean $60.0$ vs. $60.0$; $\Delta = 0$; 95% bootstrap CI $[0,\ 0]$; exact permutation test $p = 1.0$). The area under the curve (AUC) was slightly higher with compound + variable instinct evolution (mean $14{,}871.5$ vs. $14{,}717.5$; $\Delta = 154$; 95% bootstrap CI $[-2{,}052.2,\ 2{,}428.21]$; exact permutation test

$p = 0.901$). However, with a significance level of $\alpha = 0.05$, no statistically significant difference was detected in the 2-block scenario for either the final value or AUC because $p > \alpha$ in both cases.

The compound + variable method was also compared with the basic encoding (no compounds or variables) method in the 2-block scenario. The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method in the 2-block scenario, the agents with variable + compound instinct evolution achieved the same final best-so-far fitness as the agents with no-compound instinct evolution (mean $60.00$ vs. $60.00$; $\Delta = 0.00$; 95% bootstrap CI $[0.00,\ 0.00]$; exact permutation test $p = 1.00$). The area under the curve (AUC) was slightly lower with variable + compound instinct evolution (mean $14{,}871.50$ vs. $15{,}258.40$; $\Delta = -386.90$; 95% bootstrap CI $[-2{,}364.62,\ 1{,}582.44]$; exact permutation test $p = 0.7149$). However, with a significance level of $\alpha = 0.05$, no statistically significant difference was detected in the 2-block scenario for either the final value or AUC because $p > \alpha$ in both cases.

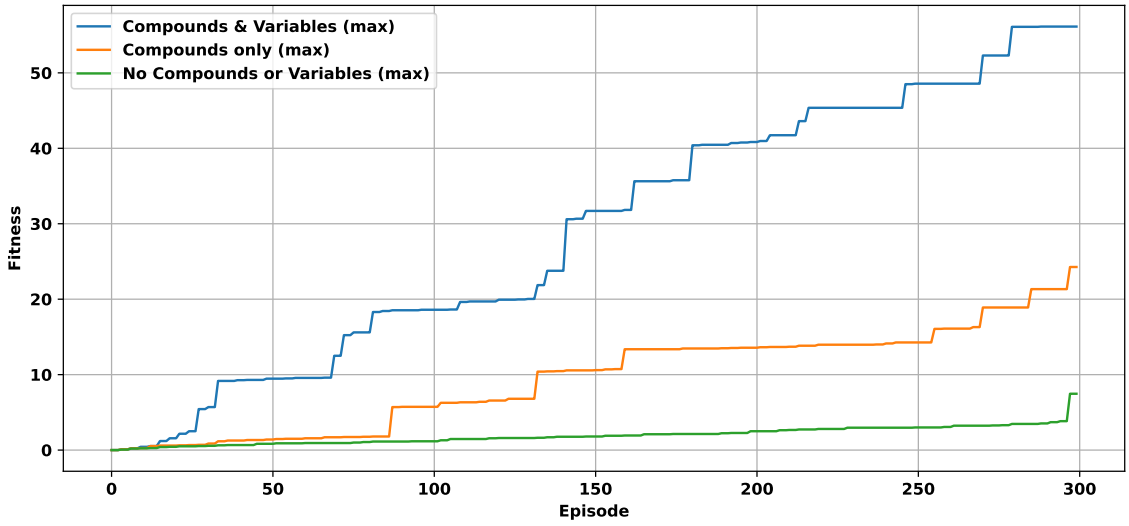


Figure 6.10: BlocksWorld Compound-and-Variable-Encoding vs. Base Encodings Performance Comparison (3-Block Puzzle)

*Figure 6.10* shows the comparison between allowing variables in compounds vs. using compounds alone, in the 3-block puzzle. The encoding using both variables and compounds scored higher overall than compounds alone by the end ($56.13$ vs. $24.27$).

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents with compound + variable instinct evolution achieved higher final best-so-far fitness than the agents with compound-only instinct evolution (mean $56.13$ vs. $24.27$; $\Delta = 31.87$; 95% bootstrap CI $[-5.43,\ 68.80]$; exact permutation test $p = 0.133$). The area under the curve (AUC) was also higher (mean $8{,}690.53$ vs. $2{,}825.67$; $\Delta = 5{,}864.87$; 95% bootstrap CI $[185.38,\ 11{,}882.5]$; exact permutation test $p = 0.0831$). However, despite the variable+compound encoding scoring higher on average in the trials, with a significance level of $\alpha = 0.05$, no statistically significant difference was detected in the 3-block scenario for either the final value or AUC because $p > \alpha$ in both cases.

The compound + variable method was also compared with the basic encoding (no compounds or variables) method in the 3-block scenario. The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method in the 3-block scenario, the agents with variable + compound instinct evolution achieved higher final best-so-far fitness than the agents with no-compound instinct evolution (mean $56.13$ vs. $7.47$; $\Delta = 48.67$; 95% bootstrap CI $[15.17,\ 82.77]$; exact permutation test $p = 0.0216$). The area under the curve (AUC) was also higher (mean $8{,}690.53$ vs. $572.77$; $\Delta = 8{,}117.77$; 95% bootstrap CI $[2{,}956.77,\ 13{,}644.10]$; exact permutation test $p = 0.00313$). With a significance level of $\alpha = 0.05$, the improvement gained by the variables + compounds encoding compared to the basic encoding was statistically significant for both final value and AUC in the 3-block scenario because $p < \alpha$ in both cases. Therefore, evolving compounds with variables resulted in better agents by the end, and evolved better agents earlier and throughout the simulation, compared to the basic encoding baseline, in the 3-block scenario.

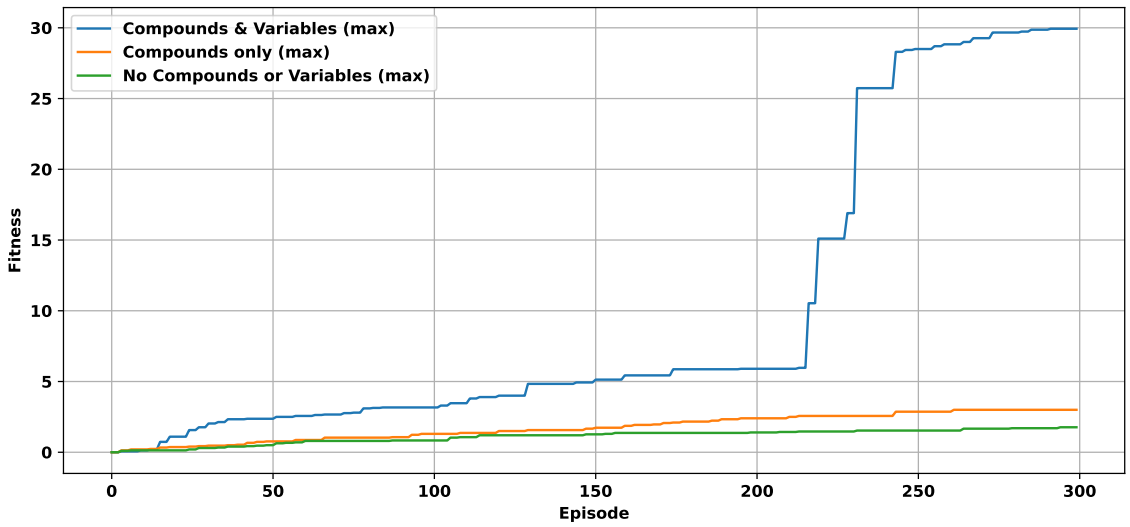


Figure 6.11: BlocksWorld Compound-and-Variable-Encoding vs. Base Encodings Performance Comparison (4-Block Puzzle)

In the 4-block puzzle, *Figure 6.11* shows that the encoding using both compounds and variables scored much higher than the encoding using compounds only (29.93 vs. 3.00), and in fact the encoding with both variables and compounds was able to produce agents capable of solving 4-block puzzles, whereas the encoding with compounds only (no variables) was unable to produce an agent capable of solving 4-block puzzles.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents with variable+compound instinct evolution achieved higher final best-so-far fitness than the agents with compound-only instinct evolution (mean 29.93 vs. 3.00; $\Delta = 26.93$; 95% bootstrap CI $[2.63,\ 59.50]$; exact permutation test $p = 0.0106$). The area under the curve (AUC) was also higher (mean 2,973.03 vs. 522.30; $\Delta = 2{,}450.73$; 95% bootstrap CI $[276.26,\ 5{,}091.67]$; exact permutation test $p = 0.0441$). With a significance level of $\alpha = 0.05$, the improvement gained by evolving variables with compounds were statistically significant for both final value and AUC in the 4-block scenario because $p < \alpha$ in both cases. Therefore, evolving compounds with variables resulted in better agents by the

end, and evolved better agents earlier and throughout the simulation, compared to evolving compounds without variables, in the 4-block scenario.

The compound + variable method was also compared with the basic encoding (no compounds or variables) method in the 4-block scenario. The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method in the 4-block scenario, the agents with variable + compound instinct evolution achieved higher final best-so-far fitness than the agents with no-compound instinct evolution (mean $29.93$ vs. $1.77$; $\Delta = 28.17$; 95% bootstrap CI $[4.00,\ 60.97]$; exact permutation test $p = 0.000920$). The area under the curve (AUC) was also higher (mean $2{,}973.03$ vs. $331.72$; $\Delta = 2{,}641.32$; 95% bootstrap CI $[483.11,\ 5{,}300.00]$; exact permutation test $p = 0.00694$). With a significance level of $\alpha = 0.05$, the improvement gained by the variables + compounds encoding compared to the basic encoding was statistically significant for both final value and AUC in the 4-block scenario because $p < \alpha$ in both cases. Therefore, evolving compounds with variables resulted in better agents by the end, and evolved better agents earlier and throughout the simulation, compared to the basic encoding baseline, in the 4-block scenario.

From these results, we can see that although the compounds-only approach only achieved statistically significant improvement over the basic encoding of AUC in the 3-block scenario, the compounds-and-variables approach achieved statistically significant improvement over the basic encoding for both final score and AUC in the 3-block and 4-block scenarios.

Putting aside variables and simply considering "raw" compound instincts only (without variables) a major problem in compound instinct evolution is that evolution blindly searches the space for combinations of sensory events even if the sensory events are not likely to be related at all in a realistic situation; for example, in BlocksWorld, forming compounds of two events which are impossible to co-occur in an actual situation (e.g., A is on B and B is on A). This results in compounds which are practically useless, since they will never be

encountered in a real situation. We can constrain the search for compounds/combinations in a practical way by only combining events which are certainly related to each other in some aspect, such as space or time. In the next experiment (*Section 6.2.4*), we will see the benefit of evolving the NARS agent itself to create and *learn* compounds at runtime, such that only events which occur next to each other in time are considered for compounding. In this way, only events which are actually and certainly temporally related to each other are compounded, which constrains the search space radically and guarantees more useful and practical sensorimotor contingencies.

### *6.2.4 Experiment 3.2: Evolution of Compound Learning*

**6.2.4.1 Setup.** This second set of experiments tested the evolution of NARS runtime compounding (compound learning) compared to basic learning (without compound $S$). In these experiments, agents with compound learning could form sensorimotor contingencies containing compound sensory preconditions (i.e., compound $S$ from $<(S, \Uparrow M) \not\Rightarrow P>$). Evolution modulated the confidence value of the formed contingencies, and could also switch on/off the ability for NARS to produce $S$ containing 1 statement, 2 statements, and 3 statements.

The agents operated purely on runtime learning, with no evolved instincts. Consequently, before each episode or problem instance, a "training phase" of 200 timesteps was allowed where NARS would motor babble and form temporal associations which were then integrated into the memory; the agent reaching the goal state did not block additional movements during the training phase, as it was purely meant to be an exploratory and learning period. After the training phase, the blocks were reset to their initial position and then "testing phase" proceeded as normal, for 150 timesteps. The agents could no longer motor babble, but instead used the knowledge acquired during the training phase to approach the problem. Since there was a separate training phase, the forgetting mechanism ($F$) was not used. The results show the average of 10 trials under each experimental condition.

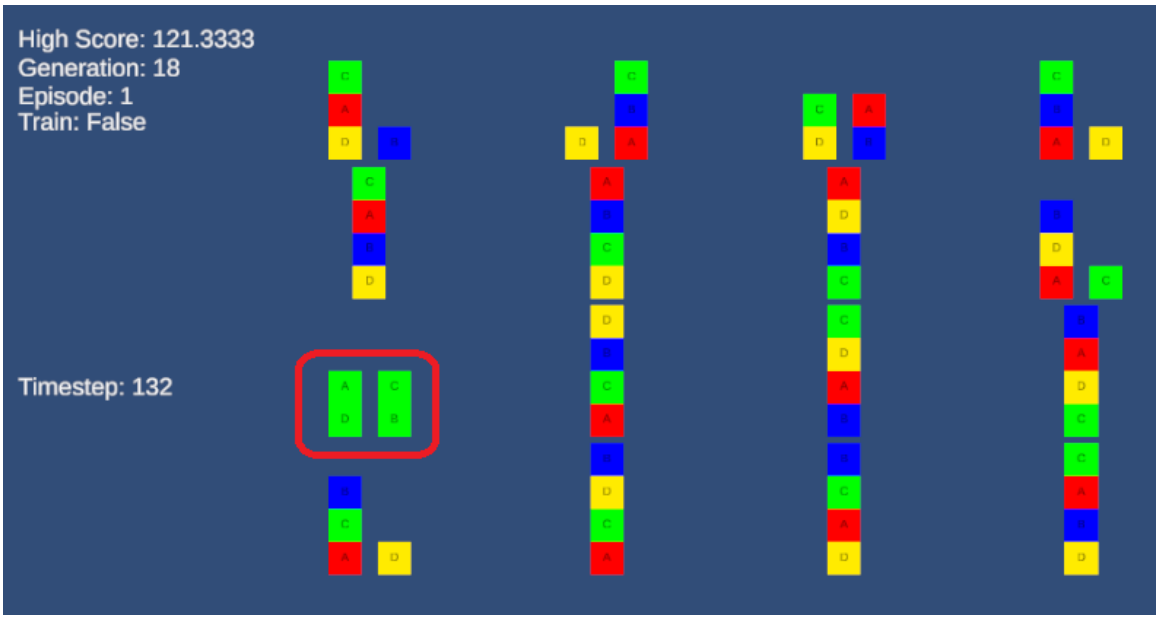


Figure 6.12: BlocksWorld Screenshot of Solved 4-Block Puzzle

#### 6.2.4.2 Results.

##### *6.2.4.2.1 Evolved Compound Learning vs. Evolved Basic Learning.*

The results indicate that agents using evolved compound learning capabilities greatly outperformed agents using evolved basic learning. Specifically, agents with learning capabilities (1) scored higher overall, (2) evolved high capabilities faster, (3) could solve 2-block, 3-block, *and* 4-block puzzles (see *Figure 6.12*), whereas agents with basic learning could not solve the 4-block puzzle.

In the 2-block puzzle (see *Figure 6.13*), compound learning agents maxed out at score 60 within the first $\sim 50$ episodes (i.e., 17 generations), whereas basic learning agents took $\sim 280$ episodes (i.e., 93 generations) to reach that same score. Both methods were able to produce agents who solved all their puzzles and achieved the maximum possible score, though the compound learning method was able to achieve top performance significantly faster.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method,

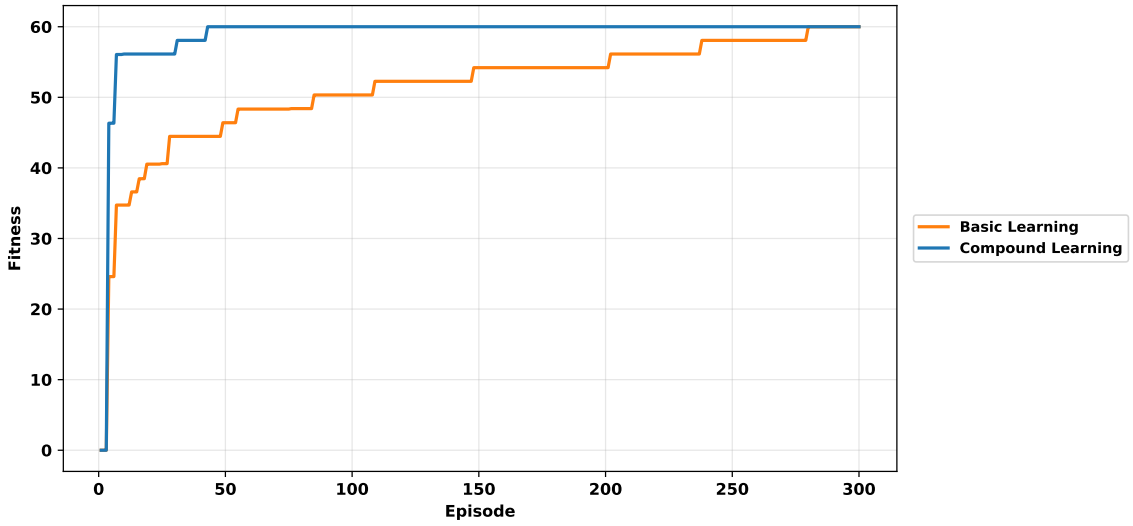


Figure 6.13: BlocksWorld Compound Learning vs. Basic Learning Performance Comparison (2-Block Puzzle)

the agents with evolved compound learning achieved the same final best-so-far fitness as the agents with evolve basic learning (mean $60.0$ vs. $60.0$; $\Delta = 0$; 95% bootstrap CI $[0,\ 0]$; exact permutation test $p = 1.0$). The area under the curve (AUC) was higher for the agents with compound learning vs. agents with basic learning (mean $17{,}632.8$ vs. $15{,}398$; $\Delta = 2{,}234.8$; 95% bootstrap CI $[1{,}272.19,\ 3{,}304.41]$; exact permutation test $p = 0.000151551$). With a significance level of $\alpha = 0.05$, the improvement was statistically significant for AUC ($p < \alpha$) but not for the final value ($p = 1.0$) in the 2-block scenario.

In the 3-block puzzle (see *Figure 6.14*), learning agents evolved to achieve a fitness of $213.1$ by episode $300$ (generation 100), whereas instinctual agents scored a lower maximum fitness of $90.3$ by episode 300 (generation 100). Both methods produced agents capable of solving 3-block puzzles, though the learning agents achieved much higher scores and also much earlier in the evolutionary process.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents with evolved compound learning achieved higher final best-so-far fitness than the agents with evolved basic learning (mean $213.1$ vs. $90.3$; $\Delta = 122.8$; 95% bootstrap CI

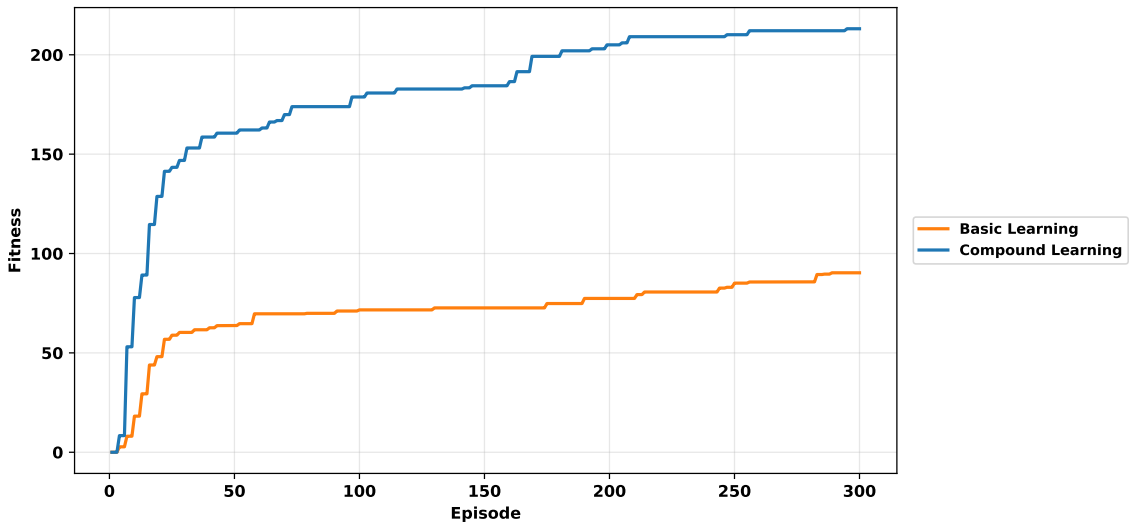


Figure 6.14: BlocksWorld Compound Learning vs. Basic Learning Performance Comparison (3-Block Puzzle)

$[108.9,\ 136.333]$; exact permutation test $p = 1.08251 \times 10^{-5}$). The area under the curve (AUC) was also higher (mean $54{,}228.1$ vs. $21{,}340.4$; $\Delta = 32{,}887.7$; 95% bootstrap CI $[27{,}241.3,\ 37{,}364.1]$; exact permutation test $p = 1.08251 \times 10^{-5}$). With a significance level of $\alpha = 0.05$, the improvements were statistically significant in the 3-block scenario for both final value and AUC because $p < \alpha$ in both cases. Therefore, evolved compound learning capabilities resulted in better agents by the end of evolution as well as earlier and throughout the evolutionary process, compared to evolved basic learning, in the 3-block scenario.

In the 4-block puzzle (see *Figure 6.15*), learning agents evolved to achieve a fitness greater than $219.97$ by episode $300$ (generation 100), whereas instinctual agents scored a maximum fitness of $0$ by episode 300 (generation 100). Only the agents capable of learning were capable of solving 4-block puzzles, whereas the basic learning agents were unable to make any progress at all on the 4-block puzzles.

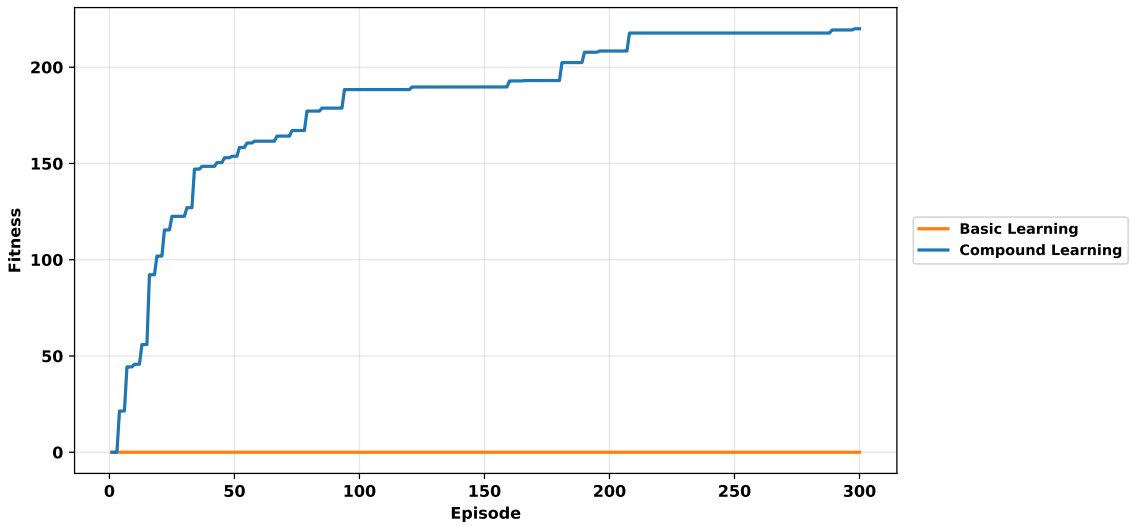


Figure 6.15: BlocksWorld Compound Learning vs. Basic Learning Performance Comparison (4-Block Puzzle)

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents with evolved compound learning achieved higher final best-so-far fitness than the agents with evolved basic learning (mean $219.967$ vs. $0$; $\Delta = 219.967$; 95% bootstrap CI $[201.933,\ 239.968]$; exact permutation test $p = 1.08251 \times 10^{-5}$). The area under the curve (AUC) was also higher (mean $54{,}589.1$ vs. $0$; $\Delta = 54{,}589.1$; 95% bootstrap CI $[50{,}799.4,\ 58{,}345.4]$; exact permutation test $p = 1.08251 \times 10^{-5}$). With a significance level of $\alpha = 0.05$, the improvements were statistically significant in the 4-block scenario for both final value and AUC because $p < \alpha$ in both cases. Therefore, evolved compound learning capabilities resulted in better agents by the end of evolution as well as earlier and throughout the evolutionary process, compared to evolved basic learning, in the 4-block scenario.

#### *6.2.4.2.2 Evolved Compound Learning vs. Evolved Compound Instincts.*

We can also compare the results of evolved compound learning to evolved compound instincts. The results indicate that agents using evolved compound learning capabilities greatly outperformed agents using evolved compound instincts. Specifically, agents with

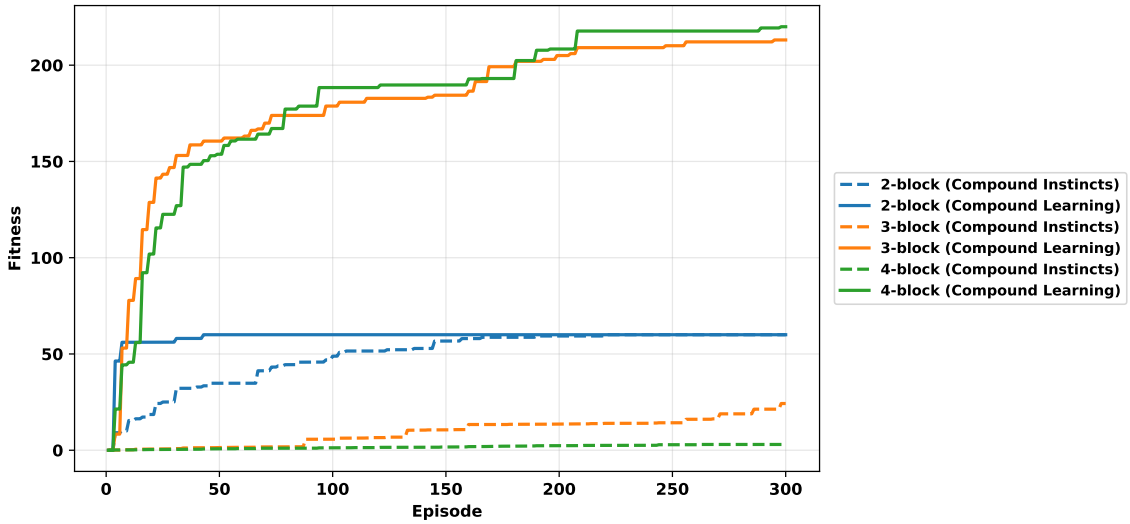


Figure 6.16: BlocksWorld Evolved Compound Learning vs. Evolved Compound Instincts Performance Comparison

learning capabilities (1) scored higher overall, (2) evolved high capabilities faster, (3) could solve 2-block, 3-block, *and* 4-block puzzles, whereas pure-instinct-based agents (at least those without variables) could not solve the 4-block puzzle.

*Figure 6.16* shows the results of the *learning-only* experiments vs. the *instinct-only* (no variable) experiments.

In the 2-block puzzle, learning agents maxed out at score 60 within the first $\sim 50$ episodes (i.e., 17 generations), whereas instinctual agents took $\sim$220 episodes (i.e., 73 generations) to reach that same score. Both methods were able to produce agents who solved all their puzzles and achieved the maximum possible score, though the learning method was able to achieve top performance significantly faster.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents learning compounds achieved the same final best-so-far fitness as the agents evolving compound instincts (mean 60.0 vs. 60.0; $\Delta = 0$; 95% bootstrap CI $[0,\ 0]$; exact permutation test $p = 1.0$). The area under the curve (AUC) was higher for the agents learning compounds vs. evolving compound instincts (mean 17,632.8 vs. 14,717.5; $\Delta =$

2,915.3; 95% bootstrap CI $[1{,}276.5,\ 4{,}829.02]$; exact permutation test $p = 0.000844357$). With a significance level of $\alpha = 0.05$, the improvement was statistically significant for AUC ($p < \alpha$) but not for the final value ($p = 1.0$) in the 2-block scenario.

In the 3-block puzzle, learning agents evolved to achieve a fitness of $213.1$ by episode $300$ (generation 100), whereas instinctual agents scored a much lower maximum fitness of $24.27$ by episode 300 (generation 100). Both methods produced agents capable of solving 3-block puzzles, though the learning agents achieved much higher scores and also much earlier in the evolutionary process.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the agents learning compounds achieved higher final best-so-far fitness than the agents evolving compound instincts (mean $213.1$ vs. $24.27$; $\Delta = 188.833$; 95% bootstrap CI $[168.633,\ 207.567]$; exact permutation test $p = 1.08251 \times 10^{-5}$). The area under the curve (AUC) was also higher (mean $54{,}228.1$ vs. $2{,}825.67$; $\Delta = 51{,}402.4$; 95% bootstrap CI $[45{,}783.1,\ 55{,}687.2]$; exact permutation test $p = 1.08251 \times 10^{-5}$). With a significance level of $\alpha = 0.05$, the improvements were statistically significant in the 3-block scenario for both final value and AUC because $p < \alpha$ in both cases. Therefore, evolved compound learning capabilities resulted in better agents by the end of evolution as well as earlier and throughout the evolutionary process, compared to evolved compound instincts, in the 3-block scenario.

In the 4-block puzzle, learning agents rapidly evolved to achieve a fitness greater than $219.97$ by episode $300$ (generation 100), whereas instinctual agents scored a maximum fitness of $3.00$ by episode 300 (generation 100). Only the agents capable of learning were capable of solving 4-block puzzles, whereas the instinctual agents were unable to solve the 4-block puzzles.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per

method, the agents learning compounds achieved higher final best-so-far fitness than the agents evolving compound instincts (mean 219.97 vs. 3.00; $\Delta = 216.967$; 95% bootstrap CI $[198.9,\ 237]$; exact permutation test $p = 1.08251 \times 10^{-5}$). The area under the curve (AUC) was also higher (mean 54,589.1 vs. 522.3; $\Delta = 54{,}066.8$; 95% bootstrap CI $[50{,}267.1,\ 57{,}825]$; exact permutation test $p = 1.08251 \times 10^{-5}$). With a significance level of $\alpha = 0.05$, the improvements were statistically significant in the 4-block scenario for both final value and AUC because $p < \alpha$ in both cases. Therefore, evolved compound learning capabilities resulted in better agents by the end of evolution as well as earlier and throughout the evolutionary process, compared to evolved compound instincts, in the 4-block scenario.

An example of a sensorimotor contingency, containing a size-3 compound, learned by NARS at runtime can be seen in *Figure 6.17*.

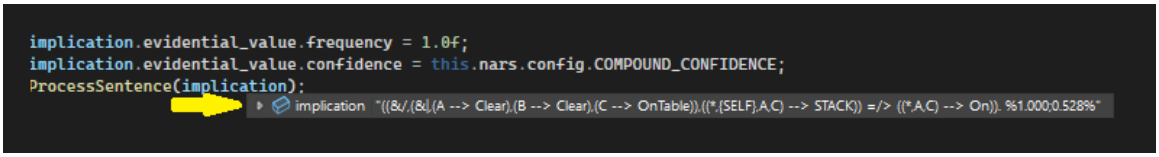


Figure 6.17: BlocksWorld Temporal Induction Screenshot

*Figure 6.18* shows how the best NARS agent (in 1 trial) toggled on/off the ability to create certain-sized compound during temporal learning. For example, we can see that the best NARS agents in the 2-block puzzles turned on the ability to learn 2-event and 3-event compounds for $S$, while leaving the ability off for learning $S$ with 1-event. In the 3-block puzzles, the best NARS evolved to learn $S$ with 2-event compounds, but not 3-event compounds or 1-event $S$. Finally, in the 4-block puzzle, the best NARS evolved to learn $S$ with 2-event compounds, but not 3-event compounds or 1-event $S$. Overall, we can see that NARS tended to perform better by learning sensorimotor contingencies containing compounds with 2 events, while refusing to learn sensorimotor contingencies with only 1 event in the precondition. In other words, better NARS agents preferred medium-specificity sensory preconditions, over less informative preconditions.

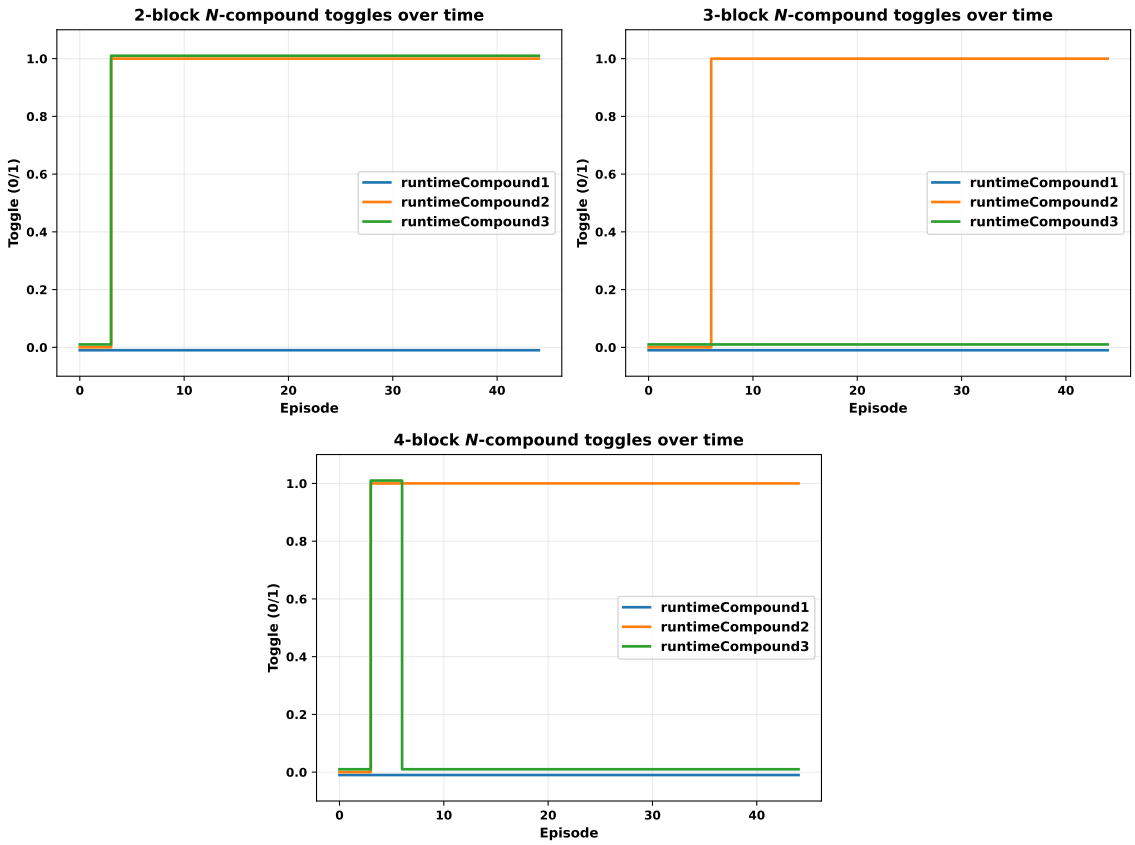


Figure 6.18: BlocksWorld $N$-Compound Toggle Evolution

As discussed in the compound-instinct experiments (*Section 6.2.3*), the combinatorial explosion that occurs by combining random events drastically increases the search space, and can hinder the speed and efficacy of instinct evolution. However, as has been demonstrated in this runtime compounding experiment, the search explosion can be combated by grouping events heuristically, such as by time, so that the grouped events are related in some way, and therefore the resulting compounds and statements are more likely to be relevant and/or useful in a realistic situation.

For example, agents who evolved runtime compounding only grouped together events which were valid states (since the agent only observes valid states, invalid world states cannot exist or be observed). In *Figure 6.17* for example, we can see that the agent formed a 3-component compound $(A \rightarrow Clear; B \rightarrow Clear; C \rightarrow OnTable)$ during temporal induction, a potentially useful sensorimotor contingency that said: "*when $A$ is clear and $B$*

*is clear and $C$ is on the table, then* $(*, \{SELF\}, A, C \rightarrow STACK)$ *results in $A$ sitting on $C$*". Of course, we can think of certain situations where this contingency is definitely true or useful, such as when $B$ is on the table and $C$ is clear. Yet also we can think of situations where this contingency is incomplete or not useful, such as when $B$ is clear but sitting on $C$, thus causing $(*, \{SELF\}, A, C \rightarrow STACK)$ to be an invalid move. Nonetheless, the state represented by the parallel conjunction $(A \rightarrow Clear; B \rightarrow Clear; C \rightarrow OnTable)$ is a completely valid state, just incomplete or missing some information that would make it more useful (e.g., $C \rightarrow Clear$), which could be remedied by allowing a larger, more complete, compound.

On the other hand, evolution of compound instincts was able to produce invalid world states in the parallel conjunctions. For example, *Block $A$ is on Block $B$ and Block $B$ is on Block $A$* could have formed, which is an impossible state to occur in reality, yet it was possible for this compound to form during instinct evolution. Thus, we can see the benefit of runtime learning and temporal heuristic grouping for events, such that evolved *learning* agents can outperform evolved *fixed* or *instinctual* agents in scenarios involving compound sensory states.

# CHAPTER 7
# EVOLVING NARS CONTROL OF A COMPLEX ROBOT

At this point, the efficacy of a basic genetic encoding for NARS has been demonstrated, as well as the usefulness of extended versions of the encoding allowing variables, compounds, and compounds with variables. The genetic encoding was shown to be useful in both the evolution of instinctual knowledge and the evolution of runtime learning, in 3 domains so far: (1) a classical wheeled-robot domain, (2) a grid-world domain, and (3) a BlocksWorld puzzle domain. Though it should be noted that each of these domains is relatively well-suited to being handled by symbolic/discrete AI systems, to the extent that exploration of these domains using symbolic AI systems has been demonstrated in various capacities in the past by other researchers (albeit, not usually with evolution or NARS).

The major issue with the traditional symbolic and reasoning AI systems is that they are often not able to scale to the same tasks as neural network systems, particularly tasks that involve complex sensory-motor setups, such as computer vision and complicated robots. Thus, in the modern research paradigms, the neural network AI approach is often preferred over the symbolic AI approach. Of course, the choice to use standard neural networks for these tasks leads to multiple problems, namely "un-explainability" and non-logical operation. Therefore, it is desirable to test whether a modern reasoning system like NARS is capable of accomplishing tasks often delegated to neural networks. In this section, the NARS encoding is tested in a domain usually delegated to neural networks: complex robot locomotion. The robot is "complex" in the sense that it contains multiple limbs with sensorymotor components, and must move its limbs to walk rather than simply executing a "*walk*" or "*move*" operation that is guaranteed to move the agent forward, such as is the case in wheeled robots. By demonstrating the ability for the NARS genetic encoding to produce NARS agents capable of locomoting complex robots, a new pathway is opened in which we can achieve explainable logic-based complex-robot control using NARS and evolutionary algorithms.

## 7.1 Experiment Domain #4: Soft Voxel Robot Locomotion

### *7.1.1 Experiment Setup*

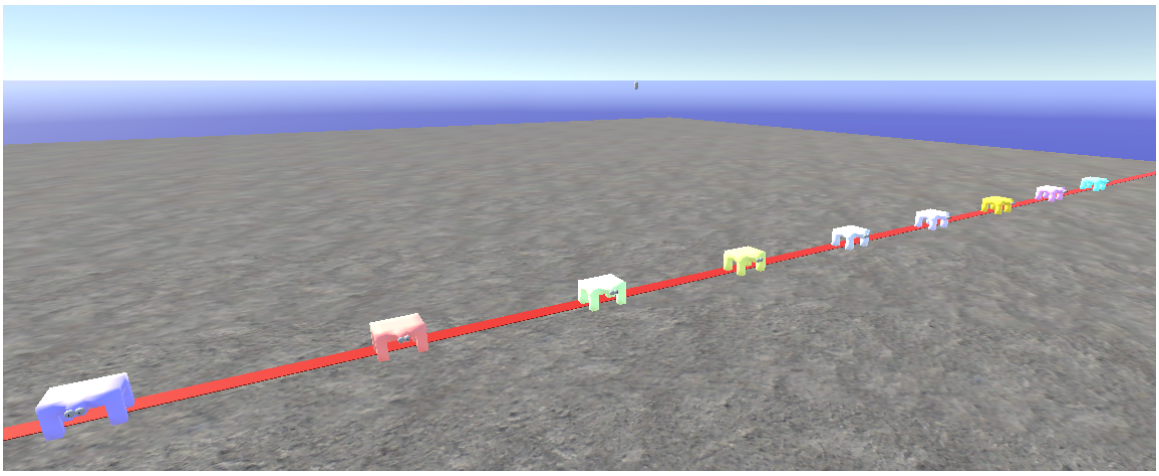

Figure 7.1: Soft Voxel Robot Locomotion Screenshot and Video

**7.1.1.1 Environment.** The environment was a classic robot locomotion environment, with robots tasked to walk on a flat plane. In this case, a wave of robots was lined up at starting position $z = 0$ (see *Figure 7.1*), and their task was to locomote as far as possible in the positive $z$-direction. There were 25 agents per wave.

**7.1.1.2 Agents.** Each NARS agent controlled a soft voxel robot. The robot consisted of 29 voxels laid out in a quadruped shape (see *Figure 7.2*; a $5 \times 4$ flat voxel body, 2 voxels per leg, and a "head" voxel). Each agent could control and sense with the bottom two voxels of each leg, making for a total of 8 voxels with sensor and motor components.

The agent could also sense which direction it was oriented towards: north, northeast, east, southeast, south, southwest, west, and northwest. This was to allow the agent to know its orientation in case it needed to re-orient itself to the positive $z$ direction, by changing motor activations, depending on the direction it was facing.

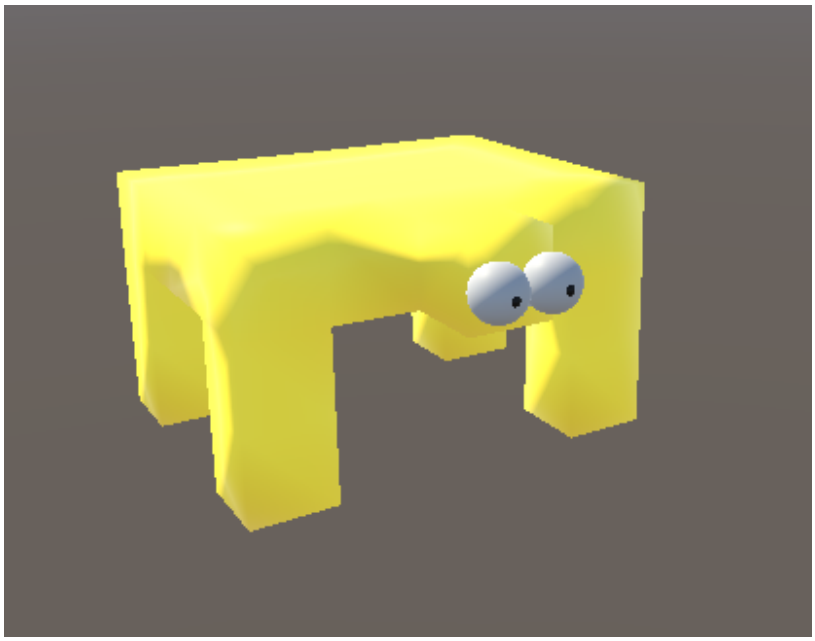

Figure 7.2: Soft Voxel Robot Quadruped Screenshot

### *7.1.2 Evolutionary Algorithm*

Three reproductive pools were used in the soft voxel robot locomotion experiments: a Fitness Hall of Fame pool (size 100), containing the top 100 scoring agents of all time, a Recent Population pool (size 100), containing the 100 most recently evaluated agents, and a Novelty Hall of Fame pool (size 100), containing the top 100 most novel agents at the current moment. To generate a new agent, there was a 1 in 3 chance to generate it from a given reproductive pool. There was a 50/50 chance to perform sexual reproduction (whereby 2 agents were selected and recombinated, and one offspring was chosen as the new agent) or asexual reproduction (whereby 1 agent was selected and mutated to generate the new agent).

After each wave, the 25-agent population was evaluated for fitness, added to the reproductive pools, and then a new 25-agent population was instantiated, with the new agents being formed from the reproductive pools.

The fitness score $\mathcal{F}$ was determined simply by:

$$\mathcal{F} = d$$

where $d$ was the distance travelled by the agent in the $z$-direction. This fitness function was meant to encourage the creation of agents who could travel further distances (in the $z$-direction).

The novelty function computing behavioral difference $\mathcal{D}_t$ between 2 soft voxel robots $i$ and $j$ at a given moment in time $t$ from birth, was:

$$\mathcal{D}_t = |pos_j^z - pos_i^z|$$

where $pos_i^z \in [0, \infty)$ was the $z$-location of agent $i$ at a given moment in time. This novelty function was meant to discover robots who could travel to various distances, both near and far, of course with the end goal of keeping those agents which traveled further (those who traveled further were placed in the Fitness Hall of Fame and consequently had more chances for reproduction).

**7.1.2.1 Genetic Encoding.** The sensorymotor sets for the soft voxel robot locomotion task were as follows.

Table 7.1: Compact Soft Voxel Robot Locomotion Sensory Set $\mathbb{S}$

| | |
|---|---|
| $\mathbb{S}$ | $<voxel_i \rightarrow Touch>$ |
| | $<voxel_i \rightarrow NoTouch>$ |
| | $<voxel_i \rightarrow Contracting>$ |
| | $<voxel_i \rightarrow Resting>$ |
| | $<voxel_i \rightarrow Relaxing>$ |
| | $<facing \rightarrow N>$ |
| | $<facing \rightarrow NE>$ |
| | $<facing \rightarrow E>$ |
| | $<facing \rightarrow SE>$ |
| | $<facing \rightarrow S>$ |
| | $<facing \rightarrow SW>$ |
| | $<facing \rightarrow W>$ |
| | $<facing \rightarrow NW>$ |
| | $<WALK \rightarrow FAR>$ |

For the sensors, the voxels on the legs had touch sensation and voxel contraction sensation. If a voxel $i \in [1, 8]$ was touching the ground the sensation was $<voxel_i \rightarrow Touch>$, otherwise if the voxel was not touching the ground the sensation was $<voxel_i \rightarrow NoTouch>$. A voxel could be in one of three activated states: $contracting$ (small and tensed), $resting$ (at its default size), or $relaxing$ (full sized and floppy). The agent was also given a sensation describing which direction the agent was primarily facing, corresponding to the cardinal and intercardinal directions ($N, NE, E, S, SW, W,$ and $NW$). For example, the agent's starting position and orientation was looking straight ahead, so the sensation at that time was $<facing \rightarrow N>$. However if an agent veered slightly right, the sensation changed to $<facing \rightarrow NE>$.

The sensory set is shown in *Table 7.1*. The sensation $<WALK \rightarrow FAR>$ was not actually possible for the agent to experience. Instead, it was the agent's goal, such that $\Gamma = <WALK \rightarrow FAR>$, and this goal drove other actions. The sensation was included so that it could be made as the predicate ($P$) of the instinctual sensorimotor contingencies, which thus allowed NARS to derive the subject ($(S, \Uparrow M)$) as a subgoal and consequently perform motor actions using valid sensory contexts.

Table 7.2: Compact Soft Voxel Robot Locomotion Motor Set $\mathbb{M}$

| | |
|---|---|
| $\mathbb{M}$ | $\Uparrow Contract(voxel_i)$ |
| | $\Uparrow Rest(voxel_i)$ |
| | $\Uparrow Relax(voxel_i)$ |

For the motors, NARS was able to actuate each individual voxel in order to set it to a specific state of contraction, like a muscle contraction. To fully contract a voxel $i$, making it tensed and at a small size, the agent would execute operation $\Uparrow Contract(voxel_i)$. To fully relax a voxel $i$, increasing it to a large size, the agent would execute operation $\Uparrow Relax(voxel_i)$. To restore a voxel to its original tenseness and size, the agent would execute operation $\Uparrow Rest(voxel_i)$. All voxels started in the $Resting$ state, after which the agent

could change their contraction states using the operations from $\mathbb{M}$. The motor set is shown in *Table 7.2*.

The personality parameter value ranges which could be reached by evolutionary search can be seen in *Table 7.3*.

Table 7.3: NARS Personality Parameter Ranges in Soft Voxel Robot Locomotion

| Parameter Name | Parameter Range |
|---|---|
| $k$ | $[1, 10]$ |
| $T$ | $[0, 1]$ |
| $W$ | $[1, 30]$ |
| $F$ | $[1, 250]$ |
| $C_{temp}$ | $[3, 20]$ |
| $C_{table}$ | $[1, 20]$ |
| $E$ | $[1, 50]$ |
| $\mathcal{T}_J$ | $[0.0000001, 10]$ |
| $\mathcal{T}_G$ | $[0.0000001, 10]$ |

**7.1.2.2 Using NAL Sensorimotor Instincts for Quadruped Robot Control.** The purpose of this experiment was to see if the evolved NARS can control a multi-component robot to make it walk forward using its limbs, even if in a simple way, rather than relying a pre-programmed motor operation that causes the robot to move forward automatically such as by driving on wheels. This type of control is a first step towards NARS agents which can move complex limbs for tasks other than locomotion, and towards NARS agents with multiple embodied components that enrich NARS sensor and motor concept bases.

The robots themselves were stable by default, in their resting state. If there was too much erratic muscle/voxel contraction however, it was possible for the robot to fall over. Therefore, NARS had to carefully choose which voxels to contract and under which circumstances so as the locomote the robot forward quickly, while also not being so erratic as to tip over the robot.

The soft voxel robot could be locomoted by "pumping" the voxels rhythmically, such that the robot leans forward or backwards at one state of voxel contraction, and then leans

the other way at the opposite state of voxel contraction. The agent could then rhythmically pump the voxel to create a "rocking" motion which causes the robot to move forward incrementally, after each rocking motion. For example, the agent could contract one of its front legs (i.e., $\Uparrow Contract(voxel_i)$), causing the robot to lean forward. Then, when the contracted muscle hits the floor, the robot could quickly relax its front leg, $\Uparrow Relax(voxel_i)$, increasing its size and creating an explosive movement that propels the robot forward. Thus, one pair of instincts that could be conducive to locomotion might be:

$$J_1 = <(<voxel_1 \rightarrow NoTouch>, \Uparrow Contract(voxel_1)) \not\Rightarrow <WALK \rightarrow FAR>>.$$

$$J_2 = <(<voxel_1 \rightarrow Touch>, \Uparrow Relax(voxel_1)) \not\Rightarrow <WALK \rightarrow FAR>>.$$

Through backwards deduction:

$\{J_1, G\} \vdash G_1 = (<voxel_1 \rightarrow NoTouch>, \Uparrow Contract(voxel_1))!$

and when $voxel_1$ is not touching the ground, $E_1 = <voxel_1 \rightarrow NoTouch>.$ is sensed, resulting in the deductive inference:

$\{E_1, G_1\} \vdash \Uparrow Contract(voxel_1)!$

a goal for a motor operation which causes $voxel_1$ to contract. A similar process would occur for $\{J_2, G\}$ when the contracted voxel hit the ground. Ultimately $\Uparrow Relax(voxel_1)$ would be derived, increasing the voxel's size rapidly and dragging the robot forward.

### *7.1.3 Experiment 4.1: Evolved Pure NARS Locomotion*

**7.1.3.1 Setup.** The robot locomotion experiment was run under 4 experimental conditions, each using a different genetic encoding, corresponding to the 4 encodings introduced previously. The first condition was the basic encoding from *Chapter 4*. The second condition was the encoding extended with the variable mutation operators from *Chapter 5*. The third condition was the encoding extended with the compound mutation operators from

*Chapter 6*. The fourth condition was the encoding extended with both compound and variable mutation operators, as demonstrated in *Chapter 6*. The beliefs ($\beta$) and meta-parameters ($\Pi$) were evolved, with learning disabled.

For each condition, 10 trials were run and the results were averaged. Each trial ran 100 waves of robots, where the robots of each wave had 40 seconds to travel as far as possible in the $z-$direction.

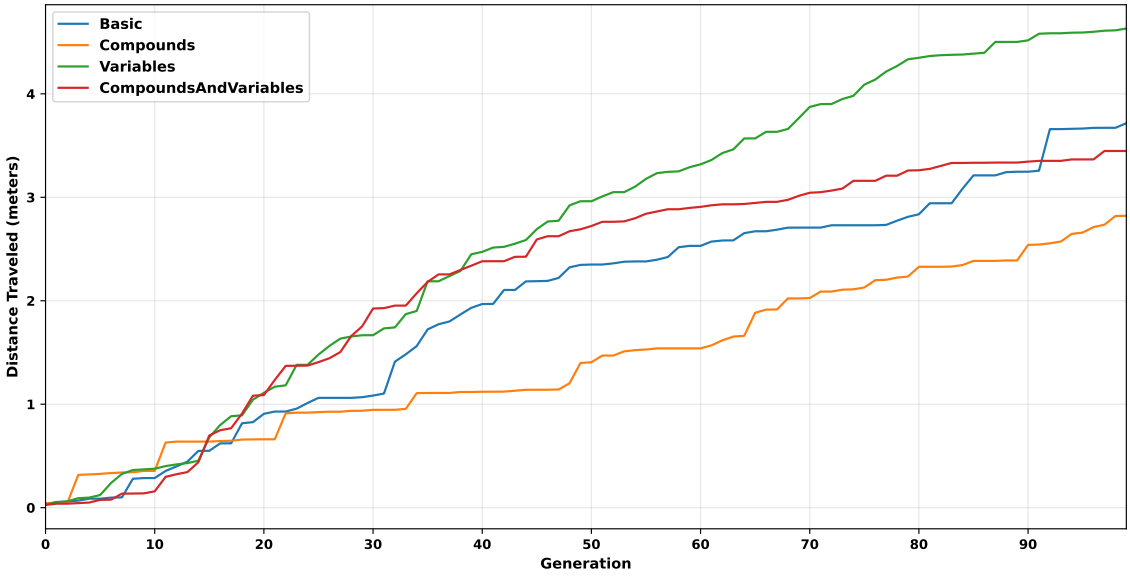


Figure 7.3: Soft Voxel Robot Locomotion Comparison of Extended and Basic Encodings

**7.1.3.2 Results.** The results of the soft voxel robot locomotion experiments can be seen in the graph of *Figure 7.3*. The variable-extended encoding achieved the best performance, traveling on average $\sim 4.6$ meters. The basic encoding ($\sim 3.75$ meters) and compounds + variables encoding ($\sim 3.4$ meters) achieved similar results throughout by the end. The compounds-only encoding (no variables) did the worst, achieving $\sim 2.8$ meters by the end, and performed lower than the other encodings throughout the evolutionary run.

The results show that all encodings were able to improve locomotion ability in NARS robots, though some versions produced better agents more quickly than others. It is interesting to see that the compounds-only encoding was not able to perform as well as the

others, since one might assume that controlling multiple voxels at the same time would be beneficial.

It may be that the larger search space introduced by compounds slowed down the evolutionary search process, making the explored agents less effective, though this effect was apparently somewhat mitigated by allowing variables in addition to compounds.

### *7.1.4 Experiment 4.2: Evolved Neuro-Symbolic NARS Locomotion*

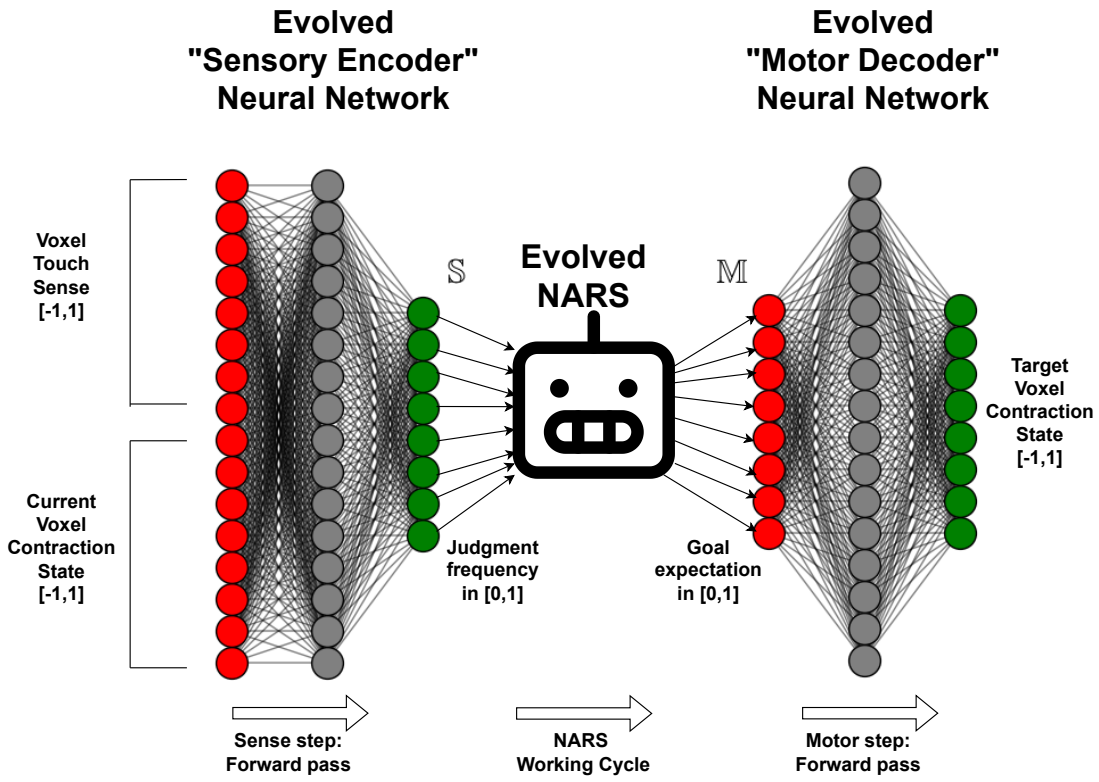


Figure 7.4: Evolved Neuro-Symbolic Architecture

**7.1.4.1 Setup.** Since terms in NARS can correspond to any arbitrary sensors and motors, not only physical sensors and motors, we can evolve a neuro-symbolic NARS system where the NARS sensor and motor terms correspond to neurons in a neural network. One way to do this is by attaching two neural networks to NARS: a "sensory encoder" and "motor decoder". The encoder network takes in sensory activations and performs a forward pass, and

the outputs of the encoder network are given to NARS as judgments, where the frequency (in $[0, 1]$) of each judgment is the activation of the corresponding encoder output neuron. Then, NARS is run for one working cycle. The decoder network takes in NARS motor activations and performs a forward pass; each decoder input neuron activation is set to the expectation (in $[0, 1]$) of NARS' corresponding motor goal.

It is possible to use NARS with a sensory encoder network, a motor encoder network, or both (as done here). Putting a sensory encoder neural network before NARS allows NARS to process the neural outputs using logical reasoning rules. Conversely, by putting a motor decoder neural network after NARS, the neural network can operate on logical outputs of NARS to transform them into complex continuous motor responses, which a reasoning system by itself often struggles to handle.

The neuro-symbolic architecture used in the experiment can be seen in *Figure 7.4*. The hidden and motor neurons used a hypertangent ($tanh$) activation function, which is in $[-1, 1]$. The outputs of the encoder were normalized to $[0, 1]$, the range of frequency values allowed in NARS. The encoder network consisted of 3 fully-connected layers: an input layer (size 16), a hidden layer (size 16), and an output layer (size 8). The decoder network also consisted of 3 fully-connected layers: an input layer (size 8), a hidden layer (size 16), and an output layer (size 8). The only requirement is that the size of the encoder's input layer must correspond to the number of physical robot sensors, and the size of the decoder's output layer must correspond to the number of physical robot motors. The other layers can be of any arbitrary size, and, if desired, additional layers could be added to the encoder or decoder or both; of course, changing the number of layers and layer sizes could impact the performance of the neuro-symbolic system.

Since the encoder network was chosen to have 8 output neurons, the sensory set $\mathbb{S}$ of NARS corresponding to those neurons is shown in *Table 7.4*:

Table 7.4: Soft Voxel Robot Locomotion Neuro-Symbolic Sensory Set $\mathbb{S}$

| | |
|---|---|
| $\mathbb{S}$ | $<encoderNeuron1 \rightarrow Activated>$ |
| | $<encoderNeuron2 \rightarrow Activated>$ |
| | $<encoderNeuron3 \rightarrow Activated>$ |
| | $<encoderNeuron4 \rightarrow Activated>$ |
| | $<encoderNeuron5 \rightarrow Activated>$ |
| | $<encoderNeuron6 \rightarrow Activated>$ |
| | $<encoderNeuron7 \rightarrow Activated>$ |
| | $<encoderNeuron8 \rightarrow Activated>$ |
| | $<WALK \rightarrow FAR>$ |

Since the decoder network was chosen to have 8 input neurons, the motor set $\mathbb{M}$ of NARS corresponding to those neurons *Table 7.5*:

Table 7.5: Soft Voxel Robot Locomotion Neuro-Symbolic Motor Set $\mathbb{M}$

| | |
|---|---|
| $\mathbb{M}$ | $\uparrow decoderNeuron1$ |
| | $\uparrow decoderNeuron2$ |
| | $\uparrow decoderNeuron3$ |
| | $\uparrow decoderNeuron4$ |
| | $\uparrow decoderNeuron5$ |
| | $\uparrow decoderNeuron6$ |
| | $\uparrow decoderNeuron7$ |
| | $\uparrow decoderNeuron8$ |

There was an 80% chance to mutate the encoder-decoder neural networks during the mutate step. If the mutation check was passed, each connection and neuron bias had a 5% chance to be perturbed by a value pulled from a normal (Gaussian) distribution with standard deviation 0.1.

The experiment was run using the standard Basic Encoding for NARS (no compounds or variables) for 10 trials, with the results averaged.

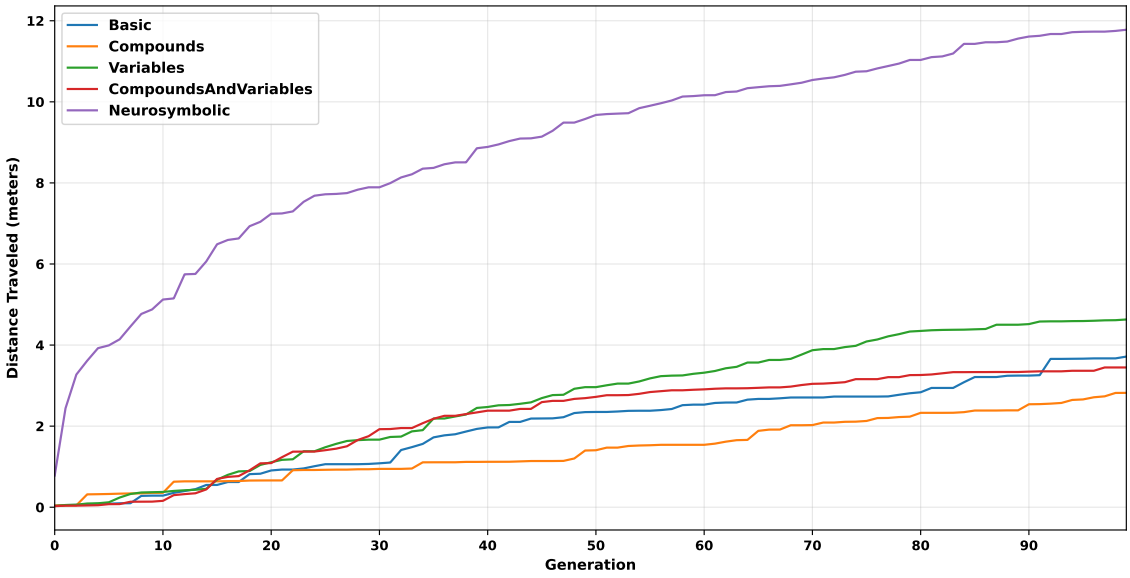


Figure 7.5: Soft Voxel Robot Locomotion Neuro-Symbolic vs. Extended Encoding Comparison

**7.1.4.2 Results.** The results indicate that the evolved NARS plus neural network (neuro-symbolic) approach greatly outperformed the evolved NARS-only approaches, where by the end the neuro-symbolic approach traveled an average of $11.7782$ meters and the NARS-only approaches traveled an average of less than $4.75$ meters (see *Figure 7.5*). Since the variable-extended encoding performed the best out of all the NARS-only approaches by the end of the evolution run, it was compared to the neuro-symbolic approach.

The methods were compared using an exact two-sided permutation test on the difference in mean performance across independent trials. Across 10 independent trials per method, the neuro-symbolic agents achieved higher final best-so-far fitness than the agents with compound-only instinct evolution (mean $11.7782$ vs. $4.631$; $\Delta = 7.14722$; 95% bootstrap CI $[5.80157,\ 8.50042]$; exact permutation test $p = 1.08251 \times 10^{-5}$). The area under the curve (AUC) was also higher (mean $882.662$ vs. $264.513$; $\Delta = 618.149$; 95% bootstrap CI $[522.061,\ 713.647]$; exact permutation test $p = 1.08251 \times 10^{-5}$). With a significance level of $\alpha = 0.05$, the improvement of the neuro-symbolic approach over the NARS-only

approach was statistically significant for both final value and AUC because $p < \alpha$ in both cases.

While combining neural networks with NARS provided an increase in performance, the tradeoff is that the raw sensors and motors are no longer directly manipulated by NARS. Nonetheless, combining NARS with neural architecture inserts logical reasoning into the agent's decision-making process, which has been shown as a useful property in previous experiments especially for "mission-critical" contexts. Such an experiment where NARS logically processed convolutional network outputs was performed in the TruePAL project where NARS generated hazard warnings for human drivers [Hahm, 2021a, Hahm et al., 2023]. The convolutional network provided object detections to NARS as "higher-level" atomics to process, which relieves NARS from needing to process the individual pixels itself, which reasoning systems tend to struggle with. In that experiment, the NARS system was manually configured by human designers and the convolutional "sensory encoder" network was pre-trained. The NARS evolutionary method could be used in the future to optimize NARS' operation on a pre-trained convolutional network's outputs, retaining explainability in its motor outputs. The neuro-symbolic setup in this dissertation allows evolution to automatically optimize *both* the neural and NARS systems in tandem, which may improve performance as they adapt to each other. By including a motor output neural network, the system's motor outputs become less interpretable, but nonetheless the system contains a logic component doing processing in-the-middle, and its internal logical reasoning process on the sensory-and-motor neural abstractions can still be investigated if desired.

# CHAPTER 8
# CONCLUSION

In *Chapter 1*, it was argued that contemporary AI methods, while highly successful in their respective domains, still leave an important gap for embodied agents operating under limited knowledge and limited computational resources. Neural approaches often achieve strong performance only through massive pre-training and costly optimization, and they typically remain "frozen" after training, limiting continual adaptation at runtime. Reinforcement learning provides a principled framework for learning from experience, yet practical deep-RL systems often struggle with fast and stable online learning, can be brittle under distribution shift, and can suffer from catastrophic forgetting and loss of plasticity when learning continually. Large language models demonstrate impressive generality in language interaction, but they are not designed for grounded closed-loop robot control, reliable uncertainty handling, or faithful embodied behavioral modeling in novel environments. Furthermore, the decisions made by pure neural methods are not transparent or easily "human-explainable", meaning it is difficult or impossible to explain the reasons for their actions, though lack of explainability has been taken as the price to pay for high performance [Gunning and Aha, 2019]. Taken together, these limitations motivate the central gap identified in *Chapter 1*: the lack of a methodology for pre-optimizing AI approaches that simultaneously support logical coherence, explainability, and continual runtime learning from scarce data, while remaining general enough to transfer across tasks and environments.

This dissertation's central contribution is a concrete step toward closing that gap by combining an explicit-representation logical reasoning system, the Non-Axiomatic Reasoning System (NARS), with evolutionary optimization. NARS provides the transparency and inspectability that many opaque models lack, because its beliefs and goals are explicit symbolic structures and its inferences are produced via auditable reasoning rules. The system can learn at runtime, computing truth values based on the available evidence, which

allows it to represent uncertainty using a "confidence" calculation. Evolution, in turn, provides an automated mechanism for optimizing the agent's starting conditions and learning configuration before deployment, thereby reducing reliance on hand-crafted knowledge engineering and manual system tuning, enabling more effective performance and adaptation in settings where data is scarce or learning must occur online.

To realize this approach, the dissertation developed a custom genetic encoding for NARS and used it with evolutionary search across multiple environments. The encoding supports two complementary pathways to effective behavior, reducing reliance on hand-crafted knowledge engineering and enabling faster and more effective runtime adaptation in settings where data is scarce:

- **Evolving object-level knowledge ("instincts"):** The encoding can generate background knowledge in the form of sensorimotor contingencies, providing agents with actionable priors that reduce the amount of runtime trial-and-error required. This is particularly valuable for bootstrapping useful starting behavior, and is needed in situations where trial-and-error can be dangerous to the agent. Across the case studies, evolved instincts supplied NARS with structured behavioral priors that enabled competent behavior early in deployment.

- **Evolving meta-level parameters and learning configuration:** The encoding can modify numeric meta-parameters that shape control, learning dynamics, and architectural preferences, and it can toggle specific learning mechanisms. By allowing evolution to tune learning-related parameters and selectively enable learning capabilities, this dissertation demonstrates a way to adapt the learning process itself to the demands of specific domains.

The four case studies provide empirical evidence that these two evolutionary pathways of instinct evolution and learning-configuration evolution can be used to produce NARS agents that are effective in sensorimotor contexts.

In *Chapter 4*, evolved instincts and learning supported survival- and reproduction-oriented behavior in wheeled robots equipped with a vision sensor, demonstrating that evolution can automate knowledge engineering and learning-mechanism optimization for embodied agents. It was found that NARS agents evolved with the new genetic encoding performed better than fixed-configuration agents, with enhanced survival and reproduction abilities in a multi-agent ecosystem environment.

In *Chapter 5*, the encoding was extended to support NAL variables, enabling generalization of knowledge beyond specific sensory contexts, which showed how both evolved priors and runtime learning with variables can promote broader competence by reducing the need for redundant knowledge and applying a single piece of knowledge to multiple similar situations. It was found that the variable-extended encoding, compared to the basic encoding, can be used to improve performance in domains with many different terms that fit the same patterns, like numerous grid tile locations, via generalization in learning and evolved instincts.

In *Chapter 6*, the encoding was extended to support NAL compounds, allowing evolved preconditions and temporal learning mechanisms to capture multi-event sensory contexts. It was found that the compound-extended encoding, compared to the basic encoding, can be used to improve performance in domains requiring multi-state sensory conditions. This was particularly effective when the compounds were combined with variables or used in runtime learning, whereby agents were able to solve the more complicated block planning puzzles.

In *Chapter 7*, the various NARS genetic encodings and an evolved NARS neuro-symbolic system were tested in the complex robot locomotion domain, which is usually reserved for neural networks due to the many components of limbed robots. It was found

that all the encodings could improve NARS performance in the domain, though some encodings such as the variable-extended encoding performed better than others. It was also found that a neuro-symbolic NARS system could be evolved, with an evolved NARS taking in information from an evolved "sensory-encoder" neural network (which processed raw sensory data) and outputting NARS signals into an evolved "motor-decoder" neural network (which translated NARS signals into continuous motor outputs). The evolved neuro-symbolic systems greatly outperformed NARS-only systems in achieving complex robot locomotion.

Across these studies, evolution tuned learning-, control-, and architecture-related meta-parameters such as desire-threshold ($T$) and evidential horizon ($k$), and it toggled learning mechanisms to improve performance (in *Chapter 6*), highlighting that not only what the agent knows initially at deployment, but also how it learns online, can be meaningfully optimized.

Although NARS served as the primary testbed in this dissertation, the broader contribution is not NARS-specific. The general result is that explicit-representation agents which learn at runtime can be treated as evolvable substrates, where evolution searches over (1) object-level structured priors ("instincts") and (2) meta-level configurations that govern the agent's architecture and control, modulating the mechanisms for its runtime updates. The specific genome syntax (NAL statements), inference rules, and meta-parameters such as $T$ and $k$ are particular to NARS; however, the evolutionary design pattern of encoding inspectable knowledge structures alongside learning-related meta-parameters applies to other symbolic and neuro-symbolic architectures that maintain explicit internal state and support incremental updates at runtime. In this sense, NARS is a concrete instantiation of a broader methodology for building adaptive, inspectable, logic-based agents, rather than the sole target of the approach.

The dissertation's results reinforce the potential for transparency and logical inspectability in AI to be practically achievable in various applications. Because NARS represents

beliefs and goals explicitly, the evolved agents' behavior remains amenable to analysis: researchers can inspect evolved sensorimotor contingencies from the genome, examine learned knowledge, and audit action selection by following the inference steps used to derive actions.

This research also situates NARS-based systems within the contemporary landscape shaped by deep learning and LLMs. While LLMs have advanced the state of the art in general language processing and can serve as powerful assistants for code and language generation and interface-level planning, their limitations for grounded sensorimotor control, uncertainty-aware decision-making, logical inference, and transparent behavioral modeling motivate continued development of explicit-representation (symbolic-logical) architectures for mission-critical settings. In such settings, hallucinations, uncalibrated confidence, and opaque decision pathways can be unacceptable. NARS does not aim to replace neural methods across all tasks; instead, it provides a complementary approach where logical coherence, explainability, and runtime adaptation are central requirements.

Looking forward, several directions follow naturally. First, further work is needed to characterize the tradeoff between evolved instincts and runtime learning across domains. Second, the evolutionary framework can be further expanded to optimize richer forms of representation and learning for other tasks, including richer sensorimotor interfaces, larger-scale tasks, cross-domain testing, and safety-relevant constraints. This includes the possibility to further explore evolved neuro-symbolic hybrid NARS systems, to examine the depth of their explainability and potential for enhanced performance. Third, it is possible to reframe the belief statements in NARS genome using tree data structures, and explore this direction using Koza-style genetic programming. The idea would be to treat $\beta$ as a forest of statement trees plus allowing subtree crossover and mutation, to potentially improve search efficiency of evolved sensorimotor contingencies. Finally, the broader paradigm of evolving explicit-representation reasoning systems to improve agent performance while preserving transparency may extend beyond NARS to other symbolic or neuro-symbolic

architectures, offering a general route to creating logic-based agents that learn online yet remain open to human inspection and intervention.

Overall, this dissertation demonstrates that evolution of NARS enables a new pathway for reasoning systems to achieve stronger sensorimotor performance while retaining the qualities that motivated this work: explicit logical inference, transparency to human inspection, and continual adaptation under AIKR constraints. By automating the discovery of both initial knowledge and learning configurations, the proposed genetic encoding reduces the need for manual knowledge engineering and creates a systematic way to study which architectural and learning features yield robust behavior in different environments. In doing so, it provides a practical contribution toward the broader goal of improving the effectiveness of agents that can adapt at runtime, learn from scarce data, and self-organize, while remaining logical and transparent.

---

Generative AI tools were used as writing assistance in certain sections for language polishing and summarization, as well as for parts of coding. All technical content, experimental design, implementation, and final interpretations are the author's.

The raw results data can be downloaded from:
`https://doi.org/10.5281/zenodo.18452743`

## BIBLIOGRAPHY


[Arkoudas, 2023] Arkoudas, K. (2023). GPT-4 Can't Reason. *arXiv preprint arXiv:2308.03762*.

[de Garis, 1995] de Garis, H. (1995). CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speeds Inside a Cellular Automata Machine (CAM). In *The Second International Symposium on Neuroinformatics and Neurocomputers*, pages 62–69. IEEE.

[Ding and Dong, 2020] Ding, Z. and Dong, H. (2020). Challenges of Reinforcement Learning. In *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 249–272. Springer.

[Dohare et al., 2024] Dohare, S., Hernandez-Garcia, J. F., Lan, Q., Rahman, P., Mahmood, A. R., and Sutton, R. S. (2024). Loss of Plasticity in Deep Continual Learning. *Nature*, 632(8026):768–774.

[Drescher, 1993] Drescher, G. (1993). The Schema Mechanism. *Machine Learning: From Theory to Applications*, pages 125–138.

[Fodor and Pylyshyn, 1988] Fodor, J. A. and Pylyshyn, Z. W. (1988). Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition*, 28(1-2):3–71.

[Gruau, 1994] Gruau, F. (1994). *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, l'École Normale Superieure de Lyon.

[Gunning and Aha, 2019] Gunning, D. and Aha, D. (2019). DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Magazine*, 40(2):44–58.

[Hahm, 2022] Hahm, C. (2022). Designing Naturalistic Simulations for Evolving AGI Species. In *International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH-2022)*, pages 296–303.

[Hahm, 2024a] Hahm, C. (2024a). Considering Simple Triangle Vision Perception with Eye Movements in NARS. In *NARS Workshop at AGI-24*.

[Hahm, 2025a] Hahm, C. (2025a). Experiments in Advancing the Evolution of Artificial Animals. Technical Report 24, Temple University AGI Team.

[Hahm, 2025b] Hahm, C. (2025b). Neuroevolution for Vision-based Seeking Behavior in 3D Soft Voxel Robots. *Artificial Life and Robotics*, pages 1–10.

[Hahm et al., 2023] Hahm, C., Gabriel, M., Hammer, P., Isaev, P., and Wang, P. (2023). NARS in TruePAL: a Trusted and Explainable AGI Partner for First Responders. Technical Report 19, Temple University AGI Team.

[Hahm and Jay, 2026] Hahm, C. and Jay, R. (2026). Testing the Effects of Evolved Central Pattern Generators in an Articulated Robot Animat Ecosystem. *Journal of Climbing and Walking Robots*.

[Hahm and Suereth, 2025] Hahm, C. and Suereth, R. (2025). Resolving Human Concerns about AI and Technology with Non-Axiomatic Reasoning Systems. *European Journal of Science, Innovation and Technology*, 5(1).

[Hahm and Wang, 2025] Hahm, C. and Wang, P. (2025). NARS Genetic Encoding. Technical Report 23, Temple University AGI Team.

[Hahm et al., 2021] Hahm, C., Xu, B., and Wang, P. (2021). Goal Generation and Management in NARS. In *Artificial General Intelligence: 14th International Conference, AGI 2021, Palo Alto, CA, USA, October 15–18, 2021, Proceedings 14*, pages 96–105. Springer.

[Hahm, 2021a] Hahm, C. G. (2021a). Integrating and Evaluating the TruePAL AI Assistant for First Responders. Technical Report 12, Temple University AGI Team; NASA Jet Propulsion Laboratory Summer Internship Program Final Report.

[Hahm, 2021b] Hahm, C. G. (2021b). NARS-Python v0.3 — Technical Overview. In *NARS Workshop at AGI-21*.

[Hahm, 2023] Hahm, C. G. (2023). A Framework of Artificial Matter. Technical Report 18, Temple AGI Team.

[Hahm, 2024b] Hahm, C. G. (2024b). Evolving Hebbian Neural Networks for Articulated Robot Locomotion. Technical Report 20, Temple University AGI Team.

[Hammer, 2021] Hammer, P. (2021). *Autonomy through Real-Time Learning and OpenNARS for Applications*. PhD thesis, Temple University.

[Hammer et al., 2024] Hammer, P., Isaev, P., Feng, L., Johansson, R., and Tumova, J. (2024). Non-Axiomatic Reasoning for an Autonomous Mobile Robot. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 17079–17085. IEEE.

[Hammer and Lofthouse, 2020] Hammer, P. and Lofthouse, T. (2020). 'OpenNARS for Applications': Architecture and Control. In *International Conference on Artificial General Intelligence*, pages 193–204. Springer.

[Hammer et al., 2016] Hammer, P., Lofthouse, T., and Wang, P. (2016). The OpenNARS implementation of the Non-Axiomatic Reasoning System. In *International Conference on Artificial General Intelligence*, pages 160–170. Springer.

[Harnad, 1990] Harnad, S. (1990). The Symbol Grounding Problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The University of Michigan Press, Ann Arbor.

[Hornby and Pollack, 2001] Hornby, G. S. and Pollack, J. B. (2001). Body-Brain Co-Evolution Using L-Systems as a Generative Encoding. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 868–875.

[Ireland, 2024] Ireland, D. (2024). Mirabile Dictu: Language Acquisition in the Non-Axiomatic Reasoning System. In *International Conference on Artificial General Intelligence*, pages 99–108. Springer.

[Isaev, 2025] Isaev, P. (2025). *Non-Axiomatic Reasoning and Causal Explorative System (NARCES), a Hybrid Symbolic Reasoning with Sub-Symbolic Learning for Autonomous Intelligent Agents*. PhD thesis, Temple University.

[Isaev and Hammer, 2025] Isaev, P. and Hammer, P. (2025). NARS-GPT: An Integrated Reasoning System for Natural Language Interactions. In *Intelligent Systems Conference*, pages 404–420. Springer.

[Johansson et al., 2024] Johansson, R., Hammer, P., and Lofthouse, T. (2024). Functional Equivalence with NARS. In *NARS Workshop at AGI-24*.

[Juliani et al., 2018] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al. (2018). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

[Kilic, 2015] Kilic, O. (2015). Intelligent Reasoning on Natural Language Data: a Non-Axiomatic Reasoning System Approach. Master's thesis, Temple University.

[Koza, 1992] Koza, J. (1992). On the Programming of Computers by Means of Natural Selection. *Genetic programming*.

[Koza, 1994] Koza, J. R. (1994). Genetic Programming as a Means for Programming Computers by Natural Selection. *Statistics and Computing*, 4:87–112.

[Li et al., 2025] Li, T., Wang, P., Wang, H., Hahm, C., Spatola, M., and Shi, J. (2025). PCRLLM: Proof-Carrying Reasoning with Large Language Models under Stepwise Logical Constraints. In *1st International Conference on Large Language Models (LLM 2025)*.

[Li, 2021] Li, X. (2021). *Functionalist Emotion Model in Artificial General Intelligence*. PhD thesis, Temple University.

[Mustafaoglu et al., 2025] Mustafaoglu, Z. S. L., Pingali, K., and Miikkulainen, R. (2025). Evolutionary Policy Optimization. *arXiv preprint arXiv:2504.12568*.

[Najarro and Risi, 2020] Najarro, E. and Risi, S. (2020). Meta-Learning Through Hebbian Plasticity in Random Networks. *Advances in Neural Information Processing Systems*, 33:20719–20731.

[Nezhurina et al., 2024] Nezhurina, M., Cipolina-Kun, L., Cherti, M., and Jitsev, J. (2024). Alice in Wonderland: Simple Tasks Showing Complete Reasoning Breakdown in State-of-the-Art Large Language Models. *arXiv preprint arXiv:2406.02061*.

[O'Neill and Ryan, 2001] O'Neill, M. and Ryan, C. (2001). Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358.

[O'Regan and Noë, 2001] O'Regan, J. K. and Noë, A. (2001). A Sensorimotor Account of Vision and Visual Consciousness. *Behavioral and brain sciences*, 24(5):939–973.

[Qiu et al., 2025] Qiu, X., Gan, Y., Hayes, C. F., Liang, Q., Meyerson, E., Hodjat, B., and Miikkulainen, R. (2025). Evolution Strategies at Scale: LLM Fine-tuning Beyond Reinforcement Learning. *arXiv preprint arXiv:2509.24372*.

[Russell and Norvig, 2016] Russell, S. J. and Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Pearson.

[Salimans et al., 2017] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv preprint arXiv:1703.03864*.

[Searle, 1980] Searle, J. R. (1980). Minds, Brains, and Programs. *Behavioral and brain sciences*, 3(3):417–424.

[Smolensky, 1988] Smolensky, P. (1988). On the Proper Treatment of Connectionism. *Behavioral and brain sciences*, 11(1):1–23.

[Stanley and Miikkulainen, 2002] Stanley, K. O. and Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary computation*, 10(2):99–127.

[Strannegård et al., 2021] Strannegård, C., Engsner, N., Ferrari, P., Glimmerfors, H., Södergren, M. H., Karlsson, T., Kleve, B., and Skoglund, V. (2021). The Ecosystem Path to AGI. In *International Conference on Artificial General Intelligence*, pages 269–278. Springer.

[Strannegård et al., 2020] Strannegård, C., Xu, W., Engsner, N., and Endler, J. A. (2020). Combining Evolution and Learning in Computational Ecosystems. *Journal of Artificial General Intelligence*, 11(1):1–37.

[Sutton, 2024] Sutton, R. S. (2024). Dynamic Deep Learning. `https://www.youtube.com/watch?v=75jr5E4OzEE&t=2548s`. Imperial College Autonomous Reasoning & Learning Group (ICARL) Seminar Series.

[van de Ven et al., 2024] van de Ven, G. M., Soures, N., and Kudithipudi, D. (2024). Continual Learning and Catastrophic Forgetting. *arXiv preprint arXiv:2403.05175*.

[Wang, 1986] Wang, P. (1986). A Reasoning System Capable of Handling Uncertainty. Master's thesis, Peking University.

[Wang, 1993] Wang, P. (1993). Belief Revision in Probability Theory. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence (UAI-93)*, pages 519–526, Washington, DC, USA. Morgan Kaufmann.

[Wang, 1995] Wang, P. (1995). *Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence*. PhD thesis, Indiana University.

[Wang, 1998] Wang, P. (1998). Grounding the Meaning of Symbols on the System's Experience. In *Working Notes of the AAAI Workshop on the Grounding of Word Meaning: Data and Models*, pages 23–24.

[Wang, 2001] Wang, P. (2001). Confidence as Higher-Order Uncertainty. In *Proceedings of the Second International Symposium on Imprecise Probabilities and Their Applications (ISIPTA 2001)*, pages 352–361, Ithaca, New York, USA.

[Wang, 2004] Wang, P. (2004). The Limitation of Bayesianism. *Artificial Intelligence*, 158(1):97–106.

[Wang, 2005] Wang, P. (2005). Experience-Grounded Semantics: A Theory for Intelligent Systems. *Cognitive Systems Research*, 6(4):282–302.

[Wang, 2006] Wang, P. (2006). *Rigid Flexibility: The Logic of Intelligence*. Springer, Dordrecht.

[Wang, 2007] Wang, P. (2007). From NARS to a Thinking Machine. In *Advances in Artificial General Intelligence*, pages 75–93. IOS Press, Amsterdam.

[Wang, 2013] Wang, P. (2013). *Non-Axiomatic Logic: A Model of Intelligent Reasoning*. World Scientific, Singapore.

[Wang, 2019] Wang, P. (2019). On Defining Artificial Intelligence. *Journal of Artificial General Intelligence*, 10(2):1–37.

[Wang, 2022] Wang, P. (2022). Axiomatic Reasoning in NARS. Technical Report 15, Temple University AGI Team.

[Wang, 2025] Wang, P. (2025). *Non-Axiomatic Logic: A Model of Intelligent Reasoning*. World Scientific Publishing, Singapore, second edition.

[Wang et al., 2022] Wang, P., Hahm, C., and Hammer, P. (2022). A Model of Unified Perception and Cognition. *Frontiers in Artificial Intelligence*, 5:806403.

[Wang and Thórisson, 2024] Wang, P. and Thórisson, K. R. (2024). Concept-Centered Knowledge Representation: A 'Middle-Out' Approach Fusing the Symbolic-Subsymbolic Divide. In *International Workshop on Situated Self-Guided Learning*, pages 10–52. Springer.

[Weel et al., 2014] Weel, B., Crosato, E., Heinerman, J., Haasdijk, E., and Eiben, A. (2014). A Robotic Ecosystem with Evolvable Minds and Bodies. In *2014 IEEE International Conference on Evolvable Systems*, pages 165–172. IEEE.

[Xu, 2025] Xu, B. (2025). A Brain-Inspired Sequence Learning Model based on a Logic. *Scientific Reports*, 15(1):13573.

[Yaeger et al., 1994] Yaeger, L. et al. (1994). Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context. In *Santa Fe Institute Studies in the Sciences Of Complexity*, volume 17, pages 263–263. Addison-Wesley Publishing Co.