UNIVERSITY OF CAPE TOWN

DOCTORAL THESIS

Neuro-Evolution Behavior Transfer for Collective Behavior Tasks

Author:
SABRE Z. DIDI

Supervisor:
Dr. GEOFF S. NITSCHKE

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science

on

January 22, 2018

Declaration of Authorship

I, SABRE Z. DIDI, declare that this thesis titled, "Neuro-Evolution Behavior Transfer for Collective Behavior Tasks " and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Signed by candidate

Date: 22/01/2018

Abstract

The design of effective, robust and autonomous controllers for multi-agent and multi-robot systems is a long-standing problem in the fields of computational intelligence and robotics. Whilst nature-inspired problem-solving techniques such as reinforcement learning (RL) and evolutionary algorithms (EA) are often used to adapt controllers for solving such tasks, the complexity of such tasks increases with the addition of more agents (or robots) in difficult environments. This is due to specific issues related to task complexity, such as the curse of dimensionality and bootstrapping problems.

Despite an increasing attempt over the last decade to incorporate behavior (knowledge) transfer in machine learning so that relevant behavior acquired in previous learning experiences can be used to boost task performance in complex tasks, using evolutionary algorithms to facilitate behavior transfer (especially multi-agent behavior transfer) has received little attention. It remains unclear how behavior transfer addresses issues such as the bootstrapping problem in complex multi-agent tasks (for example, RoboCup soccer).

This thesis seeks to investigate and establish the essential features constituting effective and efficient evolutionary search to augment behavior transfer for boosting the quality of evolved behaviors across increasingly complex tasks. Experimental results indicate a hybrid of objective-based search and behavioral diversity maintenance in evolutionary controller design coupled with behavior transfer yields evolved behaviors of significantly high quality across increasingly complex multi-agent tasks. The evolutionary controller design method thus addresses the bootstrapping task for the given range of multi-agent tasks, whilst comparative controller design methods yield scant performance results.

Acknowledgements

My advisor, Dr. Geoff S. Nitschke, is an excellent role model, and has been a choice guide to the success of this thesis. I extend him my sincere gratitude and thanks for countless hours of discussion, draft reviews and insightful comments that shaped the outcome of this thesis. None of the research outputs nor this thesis would have existed without his kind efforts to help shape the ideas and several drafts.

I am also eternally grateful to my wife, Nothabo, for her incredible patience, love and support during the trying times of study and thesis write-up.

Many thanks to the anonymous reviewers that provided invaluable insights and through review feedback and comments. I am also most grateful for the financial support jointly provided by the Center for Artificial Intelligence Research (CAIR) and the Faculty of Science at the University of Cape Town.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multi-agent[1] systems (MAS) are inspired by social systems in nature, where collective behavior of the system is a result of behavioral interactions among different species or organizations within an environment (Weiss, 1999). MAS is an example of a social system where by multiple autonomous agents interact in a dynamic environment resulting in the emergence of complex group properties. Due to its concurrent and distributed nature, group behaviors produced by a MAS can be highly effective, more robust, and adaptable than that produced by a single agent (Yong and Miikkulainen, 2007). These properties of MAS promote its usefulness in real-world problem solving and in the study of behavioral properties of a range of diverse systems such as, biology (Amigoni and Schiaffonati, 2007), economics (Antona et al., 1998; Nishizaki et al., 2009) and artificial life (Klein, 2003; Klein and Spector, 2009).

Evolutionary Algorithms (EAs) are inspired by evolution and developmental biology and have emerged as a powerful tool to synthesize controllers for multi-agent tasks (Moriarty and Miikkulainen, 1995; Bryant and Miikkulainen, 2003; Miikkulainen, 2010; Miikkulainen et al., 2012). A focus of this thesis is on Neuro-Evolution (NE), a technique that combines EAs and artificial neural networks (ANNs) (Yao, 1999; Floreano et al., 2008). NE is inspired by the evolution of biological nervous systems and applies abstractions of natural evolution (EAs) to adapt ANNs to solve computational problems. Recently, there has been an increasing interest in multi-agent learning (Weiss, 1999) and evolutionary robotics (Nolfi and Floreano, 2000), scaling up evolutionary solutions for increasingly complex problems (Doncieux et al., 2011; Verbancsics, 2011; Didi and Nitschke, 2016b). That is, complex problems for which it is difficult to directly evolve solutions. This is because randomly initialized populations of solutions are generally not able to solve non-trivial tasks and certain level of adaptation is required to solve increasingly complex tasks (Silva et al., 2016). This is known as the *bootstrap problem* (Kawai et al., 2001) that has become of wider concern and is aggravated by growing complexity in multi-agent problems and agent interaction dynamics (Doncieux et al., 2011).

[1] The terms *multi-agent* and *collective behavior* (David, 2002) are used interchangeably throughout the thesis

Natural learning has an intrinsic ability to solve complex tasks as humans (and other animals) have the ability to recognize and apply relevant knowledge from past learning experiences. The transfer of experiences from learning one task into learning another task, is termed Transfer Learning[2] (TL) (Klahr and Carver, 1988). TL is inspired by how humans apply prior conceived knowledge to solve complex tasks. Most traditional machine learning algorithms address multi-agent problems in isolation (Torrey and Shavlik, 2009). This hinders the progress of machine learning approaches to attain the level of complexity that is observed in human counterparts. TL has been a key approach to counter the bootstrap problem, where trained controllers are transferred from one source task to a desired target task where further adaptation takes place (Didi and Nitschke, 2016b; Taylor et al., 2006b; Torrey and Shavlik, 2009; Degrave et al., 2015; Knudson and Tumer, 2012; Moshaiov and Tal, 2014). This TL approach reuses knowledge learned from controllers trained in one or more tasks to bootstrap learning in more complex task. Transfer learning research has demonstrated that transferring knowledge learned on a source task accelerates learning and increases solution quality in target tasks by exploiting relevant prior knowledge (Verbancsics, 2011; Didi and Nitschke, 2016b). This transfer of controllers can happen between different tasks (that is, different fitness or reward functions) or between different domains (that is, environment or robot feature).

TL has been widely studied in the context of reinforcement learning for single-agent tasks such as robot navigation (Lutz, 2007; Taylor and Stone, 2009; Fernando and Manuela, 2006) and some multi-agent tasks (Taylor et al., 2013; Taylor et al., 2006b; Taylor et al., 2007a), where behavior transfer is between the same task with varying complexity. A popular, well established and complex multi-agent test-bed *RoboCup Keep-away Soccer* (2002), has received a lot of research attention (Taylor and Stone, 2007; Taylor et al., 2006b; Taylor et al., 2007a; Verbancsics, 2011; Didi and Nitschke, 2016b) and is thus used as the experimental case study in this thesis. Much of the work done on TL has been for single-agent tasks using Reinforcement Learning (RL), and there are relatively few examples of work done using EAs and complex multi-agent (collective behavior) tasks, such as RoboCup. In such cases it is usually RL that is used for TL (Taylor et al., 2006b; Taylor et al., 2007a), and most of the TL work with EAs has been done with single agent tasks (Bahceci and Miikkulainen, 2008; Knudson and Tumer, 2012). There is not much that applies EAs for TL in complex multi-agent tasks. Thus, an unexplored research area is how to best evolve behavioral solutions that facilitate behavior transfer between source and target tasks.

There is increasing empirical evidence that in order to boost the efficacy of evolutionary search, maintaining genotypic and phenotypic diversity in automated controller design improves the quality of evolved behaviors (Mouret and Doncieux, 2012; Cully and

[2]The terms *Transfer Learning*, *Policy Transfer* and *Behavior Transfer* are used interchangeably throughout the thesis.

Mouret, 2016; Gomes et al., 2016b; Cully et al., 2015a). Many methods of maintaining diversity have been suggested and two main competing methods are: behavioral diversity maintenance (Lehman and Stanley, 2011a; Mouret and Doncieux, 2012; Gomes et al., 2016b; Cully et al., 2015a; Cully and Mouret, 2016) and genotypic diversity maintenance (Crepinsek et al., 2013; Mouret and Doncieux, 2012; Nitschke et al., 2012; Gomes et al., 2016b; Gomes and Christensen, 2013b). However, current empirical data indicates that for controller evolution to solve complex collective behavior tasks, neither objective or non-objective based search performs well. Rather, recent research results indicate that hybridizing objective and non-objective based search facilitates the evolution of the high quality behaviors (Gomes and Christensen, 2013b; Gomes et al., 2016b).

Whilst the benefits of non-objective (behavioral and genotypic diversity maintenance) and hybrid evolutionary search (Gomes and Christensen, 2013b; Mouret and Doncieux, 2012) and policy transfer (Taylor and Stone, 2009) methods have been separately demonstrated for increasing behavior quality in various tasks, the impact of using non-objective and hybrid evolutionary search in the context of policy transfer remains unknown. Given this research gap, it is the focus of this thesis to elucidate the essential features constituting effective and efficient evolutionary search to augment policy transfer for boosting the quality of evolved behaviors in collective behavior tasks.

1.1 Motivation

The fundamental motivation for this research is the necessity of deriving solutions that integrate knowledge from prior learning to speed up adaptation in increasingly complex tasks. Uncovering effective ways to reuse knowledge is imperative to lifelong learning and capability to scale up adaptation to complex tasks.

Most of the real world problem domains involve complex distributed dynamics and high degree of uncertainty. In general, multi-agent systems are applicable to solve such complex problems, where such systems have many autonomous components with adaptive capabilities and exhibit emergent phenomena that cannot be derived by individual system components. Systems of this nature include autonomous vehicle control (Hoffmann and Pfister, 1996; Hoffmann, 2001), distributed traffic light control (Kuyer et al., 2008; Mckenney and White, 2013), robot soccer (Stone and Veloso, 1998), coordination of large swarm of robots (Bonabeau et al., 1994; Floreano et al., 2007), and models of natural and social interaction (for example, animals herding and birds flocking) (Reynolds, 1987; Spector et al., 2005). These systems feature agents with incomplete information about the environment, distributed control units and information, and complex interaction dynamics. Problem solving techniques mimicking social behaviors observed in nature (for example, ant colonies), that exhibit inherent distributed and concurrent processes, provide the flexibility to solve such problems with

distributed units. In nature (for example, in animal societies), multiple agents work together to collectively solve complex tasks (that is, tasks that are difficult or impossible for a single agent to solve). In the same way, problem solving strategies that lead to collective behaviors have emerged and shown to be successful in solving problems with complex dynamics and high degree of uncertainty (Ward et al., 2001; Baldassarre et al., 2003; Nitschke et al., 2007).

The motivation of this research thus stems from the fact that there is little research on evolutionary transfer learning methods for bootstrapping collective behavior adaptation. Specifically, there is little research that has focused on collective behavior adaptation for the purpose of investigating how transfer learning can be best used to increase the effectiveness of different evolutionary search methods (Taylor et al., 2006b; Verbancsics, 2011; Didi and Nitschke, 2016b). Given this relatively unexplored area of research, the core motivation of this thesis is to investigate various evolution (neuro-evolution) methods to ascertain the most appropriate method for facilitating behavior transfer across a range of increasingly complex collective behavior tasks.

1.2 Research Question

The main research question is how to derive the most appropriate controller evolution method that couples with transfer learning for improving adaptation in complex collective behavior tasks. Various controller evolution methods will be comparatively tested against RL methods usually used for TL in collective behavior tasks. This thesis thus seeks to ascertain the impact of using evolutionary adaptation to derive collective behaviors (in multi-agent tasks) and then transfer evolved solutions to collective behavior tasks of increasing complexity. Specifically, the key research question that this thesis seeks to answer is:

- What is the most appropriate artificial evolution method to evolve collective behaviors that are transferable, and with further evolution, effective in collective behavior tasks of increasing complexity?

Addressing this research question requires investigating a number of related issues, as given below:

1. Investigate the efficacy of neuro-evolution methods for collective behavior adaptation and transfer across collective behavior tasks compared to traditional well-established RL methods.

2. Investigate the impact of *phenotypic* and *genotypic* diversity maintenance for directing the neuro-evolution search process compared to traditional *objective-based* approaches.

1.3 Contributions and Impact

The main contribution of this thesis is a novel method for facilitating collective behavior transfer in complex collective behavior tasks with supporting theoretical and empirical analysis to address the bootstrap problem. Specifically, this thesis contributes the following:

- A novel method for adapting and transferring collective behaviors in increasingly complex multi-agent tasks, where such behavior adaptation and transfer boosts the quality of collective behavior solutions.
- A comprehensive empirical study and analysis of the effectiveness and efficiency of each evolutionary search method, and comparative traditional RL methods, with and without policy transfer across increasingly complex collective behavior tasks.

1.4 Overview of the Dissertation

The following work is divided into 6 chapters: Background (Chapter 2), Methodology (Chapter 3), Evolutionary Algorithms (Chapter 4), Reinforcement Learning (Chapter 5), Comparative Analysis (Chapter 6) and Discussion and Conclusion (Chapter 7).

Chapter 2 provides the review of prior work in transfer leaning, focusing on three aspects of the study: search methods directing evolution, diversity maintenance in evolutionary algorithms and in reinforcement learning and methods to implement transfer learning.

Chapter 3 presents the approach that has been taken to carry out the evaluation of our methods. Methods discussed include evolutionary algorithms used for evaluation, evolutionary search methods, reinforcement learning methods, diversity maintenance methods and method for performing transfer learning.

Chapter 4 focuses on collective behavior transfer learning based on evolutionary algorithms. The algorithms considered are Neuro-Evolution of Augmenting Topology (NEAT), a direct encoding neuro-evolution method and a variant of NEAT called HyperCube-Based Neuro-evolution of Augmenting Topology (HyperNEAT), an indirect encoded method. This chapter's main goal is to make behavior performance comparisons of different methods and search methods to establish the most appropriate methods for behavior transfer. A series of experiments conducted are discussed to establish the best search method in terms of performance and quality of solutions.

Chapter 5 focuses on collective behavior transfer learning in reinforcement learning. Several methods of transfer learning are explored and comparative results presented. This is to establish the most appropriate method of implementing collective behavior transfer in reinforcement learning.

Chapter 6 presents a comparative analysis that looks at the difference between the performance of reinforcement learning and evolutionary algorithms in collective behavior transfer. This is to establish how the performance results of reinforcement learning compares to evolutionary algorithms.

Chapter 7 is the conclusion chapter and as such draws conclusions from the discussion and analysis of the previous chapter. It reviews the major contributions of this study and suggests the future direction of this work.

Chapter 2

Background and Related Work

This chapter reviews the research in collective behavior transfer, behavior diversity, evolutionary algorithms and reinforcement learning in collective behavior tasks. The goal is to understand the foundational research that this work is based on and how it differs from the previous research.

2.1 Learning in Collective behavior tasks

Multi-agent systems (MAS) (Weiss, 1999) are systems composed of multiple autonomous interacting agents, sharing a common environment, which they perceive with sensors and control with actuators (Sycara, 1998; Vlassis, 2007; Shoham and Leyton-Brown, 2008). MAS can either be competitive (Weiss, 1999; Aditya Rawal, 2010) or cooperative with stigmergetic control or direct communication (Stone and Veloso, 2002; Panait and Luke, 2005a; Panait and Luke, 2005b; Haynes and Sen, 1995; Matsubara et al., 1996; Aditya Rawal, 2010).

Learning in a multi-agent system is challenging because of several issues to do with the nature of multi-agent tasks or learning dynamics. First, learning involves multiple agents concurrently learning in a non-stationary environment (Boutsioukis et al., 2012; D'Ambrosio and Stanley, 2013). Therefore, the outcome of the agents actions varies with the changing policies of other agents in the environment resulting in non-guaranteed convergence.

In MAS, the global states are derived from a combination of local states from the perception of each agent in the environment. This leads to an exponential increase in the state space as the number of agents in the system increases (Guestrin et al., 2002) and consequently the complexity in the interaction dynamics increases with the number of agents. Also, agents do not usually have a full view of the environment, either due to noisy sensors and actuators, concurrent changes in internal states of other agents or locality and proximity issues and that results in partial environment observability (Chang et al., 2004). The curse of dimensionality of the state space, non-stationary

Multi-Agent Task	Related Work	Search method	Genotype Diversity	Behavior Diversity	Behavior Transfer
Prey-Capture	(Boutsioukis et al., 2012)	MARL	No	No	Yes
	(Nolfi and Floreano, 1999)	CCEA	No	No	No
	(Panait and Luke, 2005b)	CCEA	No	No	No
	(Nitschke, 2005)	CCEA	No	No	No
	(Miikkulainen et al., 2012)	CCEA	No	No	No
	(Yong and Miikkulainen, 2007)	CCEA	No	No	No
	(Yong and Miikkulainen, 2010)	CCEA	No	No	No
	(Gomes et al., 2014a)	CCEA	Yes	Yes	No
	(D'Ambroisio et al., 2010)	GDS	No	No	No
	(D'Ambrosio and Stanley, 2013)	GDS	No	No	No
Room Cleaning	(D'Ambroisio et al., 2010)	GDS	Yes	No	No
Keep-away	(Fachantidis et al., 2011)	MARL	No	No	Yes
	(Taylor and Stone, 2007)	MARL	No	No	Yes
	(Taylor et al., 2013)	MARL	No	No	Yes
	(Gomes et al., 2014a)	CCEA	Yes	Yes	No
	(Taylor et al., 2006b)	TWEANN	No	No	Yes
	(Whiteson et al., 2010)	TWEANN	No	No	No
	(Verbancsics and Stanley, 2010)	TWEANN	No	No	Yes
	(Didi and Nitschke, 2016b)	GDS	Yes	Yes	Yes

TABLE 2.1: Comparative analysis of major research work done on evolutionary algorithms and reinforcement learning to do with application of diversity maintenance mechanism and policy transfer in multi-agent tasks.

environment, partial environment observability and complex collective behavior interaction dynamics makes the task of finding global optimum solutions intractable (Taylor et al., 2013; Verbancsics, 2011; D'Ambrosio and Stanley, 2013).

There are four major approaches to multi-agent learning as shown in table 2.1, namely: Multi-Agent Reinforcement Learning (MARL) (Panait and Luke, 2005b), Cooperative Co-evolution algorithm (CCEA) (Dreżewski, 2003; Yong and Miikkulainen, 2010; Yong and Miikkulainen, 2007), Topology and weight evolving artificial neural network (TWEANN) (Stanley and Miikkulainen, 2002; Stanley, 2004) and Generative and Developmental system (GDS) (D'Ambrosio and Stanley, 2008; Stanley et al., 2009). The first two approaches are similar in that they decompose a learning problem into roles that are then learned independently, and they both suffer from the curse of dimensionality (Yong and Miikkulainen, 2007). GDS is an indirect encoding neuro-evolution method that enables effective evolution of complex tasks (Stanley, 2016). The following sections provide a detailed discussion of these approaches, challenges associated with them, and how they have been used in multi-agent learning.

2.1.1 Reinforcement Learning

Reinforcement learning (RL) is one type of machine learning that learns by trial and error interaction with its environment (Kaelbling et al., 1996; Sutton and Barto, 1998; Stone et al., 2005). In this learning framework the agents perceive the state of the environment and execute an action that causes a change in the state and receives a

reward representing the quality of the transition. An agent learns to perform well for a given state-action pair by repeating this process several times as defined in the algorithm. The goal of the agent is to maximize the total reward it receives over time. This is inspired by how children or animals develop motor skills, without any direct communication. Various movements are tried in the process, successful trials are rewarded for moving closer to the goal and conversely penalized for stumbling and falling.

In single-agent reinforcement learning, the problem space and environment is modeled as a *Markov Decision Process* (MDP) (Howard, 1960), a technique for modeling sequential decision-making tasks where a learning agent interacts with a system in a sequential pattern.

Definition 1: A Markov Decision Process (MDP) is a tuple $\langle S, A, T, \rho \rangle$ where S is the finite set of environment states, A is the finite set of agents actions, $f : S \times A \times S \rightarrow [0, 1]$ is the state transition probability function, and $\rho : S \times A \times S \rightarrow \Re$ is the reward function. The state signal $s_t \in S$ represents the state of an environment at discrete time-step t, $a_t \in A$ represents an action taken in time-step t that cause a transition to time-step $t + 1$ and $\Re$ denotes a set of real numbers. The agents then receive a scalar reward r_{t+1} for taking action a_t leading to state s_{t+1}. The behavior of the agent is defined by its policy π, which specifies how the agent select an action a_t given a state s_t . The policy may be either stochastic, $\pi : S \times A \rightarrow [0, 1]$, or deterministic, $\pi : S \rightarrow A$. The agent's goal is to maximize the expected return, $E\{R_t\}$, for each time-step t. For each episodic task the expected return is given by equation 2.1:

$$R_t = \sum_{t=0}^{T} r_t \tag{2.1}$$

where T is the final time-step corresponding to the terminal state.

However if the task has a continuous state, the expected return is given by equation 2.2:

$$R_t = \sum_{k=0}^{\infty} \gamma r_{t+k+1} \tag{2.2}$$

where $\gamma \rightarrow [0, 1]$ is the discount rate. The majority of reinforcement learning algorithms are based on estimating the value functions, which gives an estimate of how good it is to be in a given state or execute a given action in a given state under a certain policy. The value of executing a certain action in a given state under policy π is given by an action-value function (Q-value function), $Q^\pi : S \times A \rightarrow \Re$, in equation 2.3:

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi\{\sum_{k=0}^{\infty} \gamma t_{t+k+1}^k | s_t = s, a_t = a\} \tag{2.3}$$

The optimal action-value (Q-value) function is defined by Bellman's optimality equation 2.4:

$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \underset{\pi}{max} Q^*(s',a'), \forall s \in S, \forall a \in A, \tag{2.4}$$

where next state $s' = s + 1$, and the next action $a' = a + 1$. The greedy policy is deterministic, and selects at all times the action with the highest Q-value, given by equation 2.5:

$$\pi(s) = \underset{\pi}{argmax} Q^{\pi}(s,a) \tag{2.5}$$

Conversely the ε-greedy policy is stochastic and it selects an action based on equation 2.6:

$$\pi(s) = \begin{cases} \underset{\pi}{argmax} Q^{\pi}(s,a), & P(1-\varepsilon) \\ \text{random action}, & P(\varepsilon) \end{cases} \tag{2.6}$$

There are several RL approaches, namely: direct policy [1] search (Peters and Schaal, 2008), value-function based (Sutton, 1998) and model based (Bertsekas, 2001). Most MARL algorithms are derived from value-function based approaches, mainly Temporal Difference (TD) methods such as Q-Learning (Watkins, 1989) and SARSA (Sutton, 1998). SARSA uses a stochastic policy, utilizing equation 2.6 for selecting the next action to execute whilst Q-Learning is deterministic and utilizes equation 2.5 for choosing the next action to execute.

The simplicity and generality of RL algorithms has made it an attractive method for multi-agent control tasks. *Multi-agent reinforcement learning* (MARL) is an extension of RL to multi-agent environments, where a number of players simultaneously choose actions from a given set of actions. Usually in MARL, the multi-agent problem is modeled by a *stochastic game* (SG), which is an extension of MDP and game theory. A RL framework of SG is given by the following formal definition (Littman, 1994; Hu and Wellman, 1998).

Definition 2: A stochastic game is a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}_{i...\mathcal{N}}, \mathcal{T}, \mathcal{R}_{1...\mathcal{N}} \rangle$, where $\mathcal{N}$ is the number of players, $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of joint actions ($\mathcal{A}_i$ is the set of actions available to the i^{th} player), $\mathcal{T}$ is the transition function $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ and $\mathcal{R}$ is the reward function ($\mathcal{R}_i$ is the reward for the i^{th} player, $\mathcal{S} \times \mathcal{A} \rightarrow \Re$). In this SG model, all players select their actions simultaneously, and the reward each player receives and the next state depends on the joint actions of all players according to Markov property.

The stochastic games are classified into three main groups, namely: fully cooperative (that is, where all agents have the same reward function), fully competitive (where one player's reward is always the negative of the others) and mixed games (Busoniu et al., 2008). The MARL algorithms are organized based on this classification of tasks (that is, fully cooperative, fully competitive and mixed task). Under competitive SG, Littman

[1] The terms *policy* and *behavior* are equivalent and will be used interchangeably throughout the thesis

(1994) proposed a *Minimax-Q* learning algorithm for zero-sum games in which a player maximizes its reward in the worst situation. This algorithm was applied to a two-player zero-sum multi-agent SG environment and it was found to be very slow to learn an optimal behavior, as it requires a linear programming in each episode and in each state (thus increasing computational cost) (Littman, 2001a). Hu and Wellman (1998), extended the zero-sum game framework to general-sum and proposed a Nash-Q learning algorithm. However, this Nash-Q learning algorithm comes with strict conditions and additional assumptions (to guarantee convergence to Nash Equilibrium) that are difficult to satisfy in multi-agent setting. To address this difficulty, Littman (2001) proposed *Friend-or-Foe Q-learning* (FFQ) algorithm. The FFQ-learning classifies equilibrium as either coordinated or adversarial. Compared to Nash-Q, FFQ-learning provided a strong convergence guarantee, but the FFQ-learning still require a very strong condition for application. Therefore, like Nash-Q, the FFQ-learning was found to be inappropriate for a system where neither coordination nor adversarial equilibrium exist (Suematsu and Hayashi, 2002).

Considering the drawbacks of Nash-Q and FFQ learning algorithms, Suematsu and Hayashia (2002) introduced a SARSA based multi-agent algorithm called *Extended Optimal Response Learning* (EXORL). The idea of EXORL algorithm was to derive a policy which is an optimal response to that of the opponent and reach a Nash equilibrium when the opponent is adaptable. As in Nash-Q algorithm, EXORL encountered difficulties when there exists multiple equilibria. The other drawback of EXORL is that one player is assumed to be able to observe opponent's actions and rewards. In a multi-agent setting where agents learn simultaneously, this may be very difficult. There are many other MARL algorithms that were proposed to address some of these challenges, such as WoLF (Suematsu and Hayashi, 2002) and JAL (Cao et al., 1997).

The MARL algorithms discussed above have been designed for static stochastic games (Busoniu et al., 2008) and are not suitable for a special case of multi-agent sequential control tasks with continuous state and action spaces, where actions can last more than one time step (for example, keep-away soccer (Stone et al., 2005)). In the keep-away soccer simulation, decisions are made only when the actions terminate and to handle such situation, it is convenient to model the problem as semi-Markov decision process (SMDP) (Bradtke and Duff, 1994; Stone and Veloso, 2002; Stone and Sutton, 2002; Whiteson et al., 2003; Taylor and Stone, 2005; Taylor and Stone, 2009). With multiple agents and concurrent updates, the state of the environment may change continually between decisions, unlike in MDP model where state changes occur only due to an agent's actions. In the SMDP model, time between transitions can be split into several units and this can vary from one transition to another. A multi-agent SMDP is described by a tuple $\langle N, S, A, T, R, F \rangle$ where N is finite set of n agents, with each agent $i \in N$, having a finite set A^i of individual actions. The joint action space $A = \prod_{i=1}^{n} A_i$ represents the parallel execution of actions a_i by each agent $i, i = 1, ..., n$.

However, several challenges have emerged for multi-agent reinforcement learning (MARL). First, joint control paradigm (Busoniu et al., 2008), the difficult in defining the learning goal for multiple RL agents. Second, non-stationary environment (Ghavamzadeh et al., 2006), learning in a dynamic environment makes convergence unpredictable. Third, non or partial observability (Chang et al., 2004; Busoniu et al., 2008), states and actions of other agents required to make decisions are not fully visible. Fourth, curse of dimensionality (Bellman, 1961; Kaelbling et al., 1996; Ghavamzadeh et al., 2006), the exponential growth in the number of state-action space with respect to an increase in the number of agents. Fifth, scalability issues (Busoniu et al., 2008), failure to scale up algorithms to realistic problem sizes. Several methods have been devised to address these problems such as transfer learning in reinforcement learning (discussed in section 2.2.1) and neuro-evolution methods discussed in the following subsection.

2.1.2 Neuro-Evolution (NE)

Evolutionary algorithms (EA) are optimization methods that use abstractions of biological evolution to adapt encoded representations of solutions in a survival of the fittest process (Goldberg and Richardson, 1987). EAs are inspired by evolution and developmental biology (Goldberg and Holland, 1988), and are stochastic, population based methods that have gained a lot of attention recently seeking to develop robust solutions for multi-agent learning (Hornby et al., 2003).

There are many variants of EAs, this study reviews variants of Neuro-Evolution (NE) methods as a type of EA. Basically in most of NE methods the researcher defines an objective function, $f(s)|s \in P, f : P \rightarrow \Re$, maps P the set of all candidate solutions, s to a real-value, v a measure of fitness. The objective is to rank each element of P based on the evaluation results. All candidate solutions are evaluated and a subset of that population is selected for reproduction. The evolutionary process continues for a number of generations until an optimum solution is found or a predefined number of generations are evaluated.

A driving issue behind EAs has always been to scale up complexity in phenotypes to a level of complexity observed in natural organism to generate solutions for complex tasks. There are many categories of NEs and in this study there are three classifications, namely: Conventional Neuro-Evolution (CNE), Topology and weight evolving artificial neural networks (TWEANNs) and Generative encoding.

Conventional Neuro-Evolution and Coevolution

Conventional Neuro-evolution (CNE) is a class of NE methods that includes all methods that evolves weights of a topologically fixed neural network. A technique called

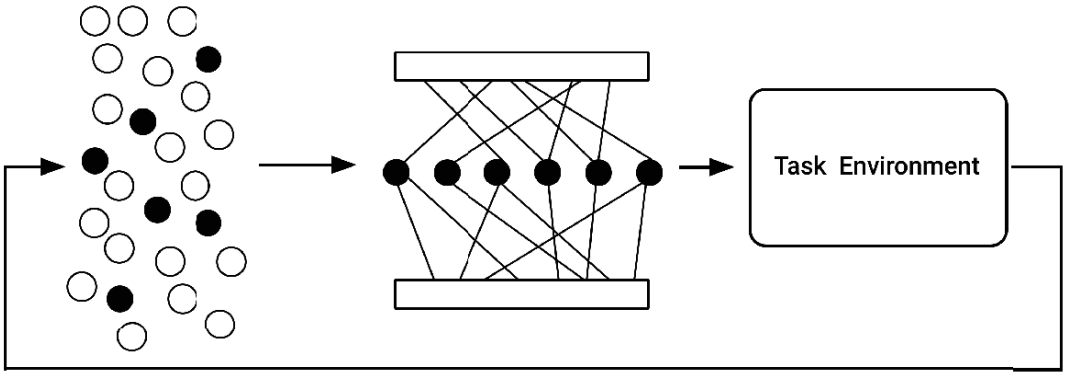


FIGURE 2.1: The Symbiotic Adaptive Neuro-Evolution (SANE) encoding. Left side shows a population of neurons that are drawn to construct the hidden layer of a neural network. The network is then applied to a task for fitness evaluation. Figure adapted from Moriarty (1997).

Symbiotic, Adaptive Neuro-Evolution (SANE) (Moriarty, 1997) was proposed. Figure 2.1 shows the architecture where representation is on neuronal level. This approach was inspired by cooperative social systems in nature, such as antibodies in the immune system. The body maintains a pool of antibodies, each specializing on defence against a particular antigen, to survive any infection. Therefore a task in this approach is decomposed into components and each individual represents a partial solution and a complete solution is formed by combining all individuals in the population (Moriarty and Miikkulainen, 1996). The fitness is shared among individuals that participated in the network. The standard SANE was then extended to Hierarchical SANE that includes hierarchical modularity (Moriarty and Miikkulainen, 1998), where two populations are created and evolved simultaneously, where one population is of neurons and another of ANNs blueprints (templates to specify how the neurons are combined to generate networks). On each fitness evaluation, the fitness assigned to each neuron is computed as average performance from all the networks it participated in. This method was tested on various benchmark tasks such as robot control and pole balancing, and it outperformed standard SANE (neuron-level EA without blueprints) and temporal-difference methods (that is, SARSA and Q-Learning) (Moriarty and Miikkulainen, 1998).

An extension to SANE resulted in a new approach called Enforced Sub-Population (ESP) (Gomez and Miikkulainen, 1997). Figure 2.2 shows a design of this architecture that decomposes a population into subpopulations where each subpopulation contributes a neuron to construct a network hidden layer. Unlike SANE, there are no blueprints and each network hidden layer is constructed by neurons from different subpopulations. The number of subpopulations are predefined and are equivalent to the number of network hidden neurons. At random a member of each population is picked to construct a network for each fitness evaluation. This method encourages specialization

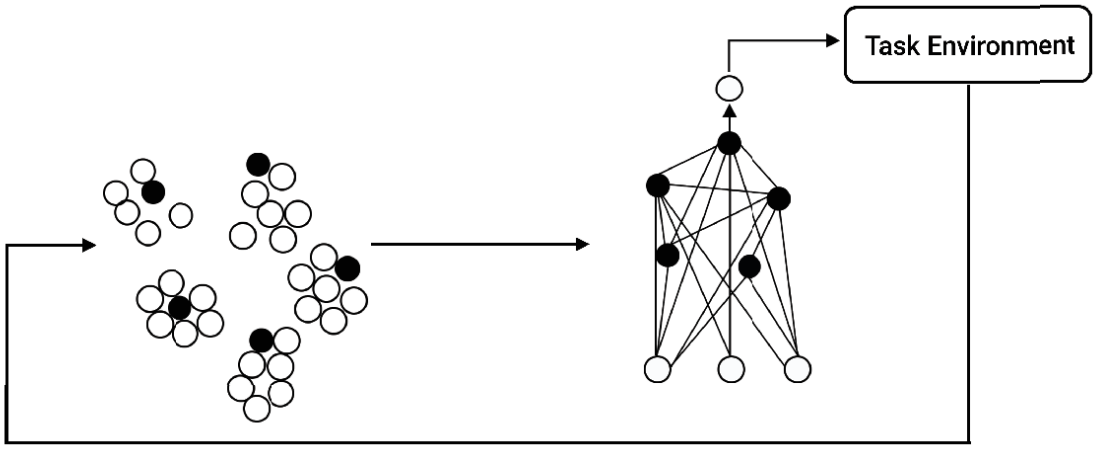


FIGURE 2.2: Enforced Sub-Population method (ESP). The method maintains a sub-population of neurons, each contributing a neuron for the construction of the network hidden layer. The constructed network is then applied to a task for fitness evaluation. Figure adapted from Gomez (1997).

at neuron level and was tested on a reinforcement learning single agent benchmark task called double pole-balancing (Geva and Sitte, 1993; Gomez and Miikkulainen, 1997). The pole-balancing (especially the version where only partial sensor information is given) is a surrogate for complex robot control tasks (Geva and Sitte, 1993).

This method was further extended to Multi-Agent ESP - as exhibited in Figure 2.3 which was then tested on prey-capture (or pursuit and evasion) task (Yong and Miikkulainen, 2007; Yong and Miikkulainen, 2010). This task is inspired by the hunting behaviors of predator animals, and seeks to mimic as well as to reproduce their behaviors so as to study multi-agent emergent behaviors such as cooperation, competition, communication and controller design. In this task, a team of autonomous agents must cooperate to capture a fast moving prey. Multi-agent ESP was first applied on this prey-capture domain (Miller and Cliff, 1996) to demonstrate how different problem representations, evolving, and coordinating a team of autonomous agents affect performance (Yong and Miikkulainen, 2007). The configuration shown in Figure 2.3, three predator agents chasing one prey, is based on stigmergic control of agents rather than direct communication between agents. The results of Yong and Miikkulainen indicated that a homogeneous network that controls the whole team performs worse than a set of heterogeneous controllers, evolved cooperatively to control a single agent. These results were assumed to be caused by niching in co-evolution, which is a distinct part of ESP. The set of simpler subtasks are identified and each agent evolved separately and in parallel for one such subtask, instead of searching the entire solution space. In the same study it was also found that cooperation is most efficient through stigmergic interaction instead of direct communication between agents. There are several follow-up results that show different results, depending on the context, type of domains and the investigation being carried out (Haynes and Sen, 1996b; Haynes and

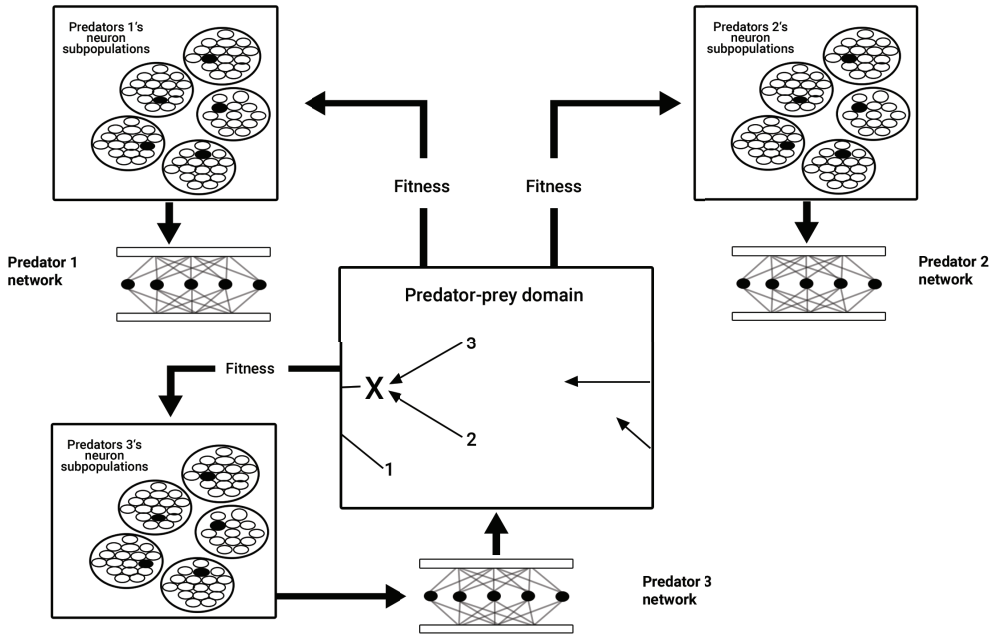


FIGURE 2.3: Multi-Agent ESP architecture. It shows the configuration of a predator-prey task of three predator agents capturing a single prey. Each agent is controlled by its own network constructed from a set of its subpopulations. All three controllers are evolved simultaneously and rewards shared equally among neurons that participated in the network evaluation. Figure adapted from Yong and Miikkulainen (2007).

Sen, 1996a).

Cooperation and competition in agent teams have been studied in Multi-Agent ESP, but the Multi-Agent ESP architecture was found to be unable to sustain arms-race of multiple competing and cooperating agents (Aditya Rawal, 2010). This limitation led to further ESP extension, Multi-Component-ESP (Aditya Rawal, 2010) with architecture presented in Figure 2.4. Each predator agent has a set of $n_p + 1$ networks that controls its movement, where n_p is the number of prey agents perceived. For n_d number of predator agents, the architecture will have a total of $(n_p + 1) * n_d$ networks. Figure 2.4(a) shows four networks, each constructed from its own pool of enforced sub-populations, where a single network represents a single prey sensor. The fifth is an integrator network that combines output of all four networks and specifies the next move of the predator. This method was developed to test the possibility of sustaining both competitive and cooperative co-evolution in multi-agent tasks (Aditya Rawal, 2010). It was also tested on the same predator-prey domain to study how reward structure and coordination mechanisms affect multi-agent evolution (Rajagopalan et al., 2011; Miikkulainen et al., 2012). The results highlighted that predator agents learned effectively through stigmergic interaction to switch roles dynamically and to head off the prey agents before capturing them. Simultaneously the prey agents developed complex high-level behaviors to evade the opponents such as baiting, scattering, direction reversal and

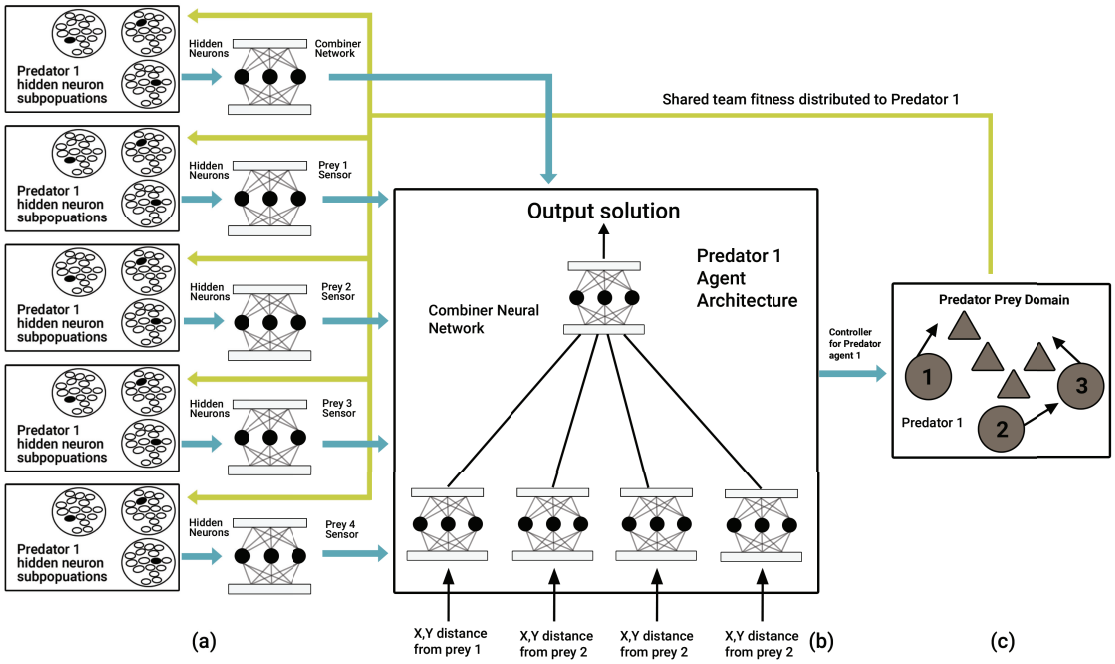


FIGURE 2.4: Multi-Component ESP architecture. It shows the configuration of a predator-prey task of three predator agents capturing four prey agents. Each predator agent is controlled by its own set of five networks. Four of which corresponds the number of prey and each collects sensory inputs of each prey agent. The fifth controller integrates sensory outputs from those four controller networks and determines the next predator agent. Figure adapted from Rawal (2010).

sidestepping. It is interesting to note that co-evolution does not search the entire solution space instead a set of simpler subtasks are identified and optimized separately and in parallel by each agent for each subtask. In this way each agent adapts to the behavior of other agents, and explicit communication is not necessary.

Co-evolution in EAs refers to maintaining and evolving individuals for different roles in the same task, either in a single population or in multiple populations. This is in contrast to a monolithic technique which evolves a population where each individual specifies a controller for every agent on the team. This rapidly becomes intractable, as the search space exponentially grows with the increase in the number of agents. In Cooperative Co-evolution in EAs (CCEA) the agents share the fitness values and the best setup that can fully utilize CCEA is that which is decomposable into modules that can interact and cooperate to solve a task. Each module can be evolved in its population and each population contributes its fittest candidate to the solution. Several experiments were conducted with ESP to test if co-evolution is necessary in evolution, it was observed individuals trained separately that do not cooperate are weaker than those that use co-evolution to evolve cooperative behavior (Yong and Miikkulainen, 2010; Gomes et al., 2014a).

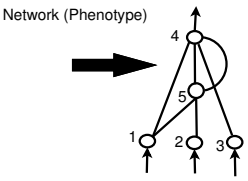
One of the problems associated with neuro-evolution methods discussed in this subsection is *Competing Conventions*, a term introduced by Schaffer et al. (1992), for a

problem previously identified by RadCliffe (1993) as *Structural Mapping Problem* and by Whitley et al. (1990) as the *Permutation Problem*. This problem refers to a situation whereby there is many-to-one mapping between genotype and phenotype, resulting in a single solution being expressed by two genotypes with different encodings. The presence of competing conventions problem on evolution results in crossover producing damaged off-springs. This problem is also described as a form of deception that renders crossover to be ineffective (Goldberg, 1989). A method discussed in the following subsection was developed to address *Competing Conventions* problem and this method evolves both the weights and topology of the artificial neural network (Stanley and Miikkulainen, 2002).

Topology and Weight Evolving Artificial Neural Networks

The CNE methods discussed (including ESP multi-agent variants) all use a fixed topology, meaning that a complete network topology is specified prior to evolution and fixed during evolution. However *Neuro-Evolution of Augmenting Topologies* (NEAT) (Stanley and Miikkulainen, 2002) evolves both the topology and weights of the network, starting off with a topology of only input and output nodes. Over generations of evolution, new nodes and connections are added and removed, eliminating a burden of applying human knowledge on network design. Past empirical evidence has demonstrated the superiority of NEAT over most fixed-topology methods such as ESP (and its variants) and temporary-difference methods such as SARSA and Q-Learning on challenging benchmark reinforcement learning tasks such as pole balancing (Stanley and Miikkulainen, 2002), robot control (Stanley and Miikkulainen, 2004a), Server Job Scheduling (Whiteson and Stone, 2006) and the well-known complex multi-agent benchmark task, keep-away soccer (Taylor et al., 2006b; Verbancsics and Stanley, 2010). The success of NEAT has often been attributed to three inherent properties discussed below, namely Complexification, Historical marking and Speciation. NEAT is a general purpose NE search method that can be applied to a wide variety of problems including multi-agent problems (Taylor et al., 2006b; Miikkulainen et al., 2012; Zhao and Peng, 2012).

The mutation in standard NEAT is carried out in two ways: add connection which creates a new connection between existing nodes or add node which split the existing connection and add a new node in-between the two new connections. This has an effect of disabling the old connection and adds the two new connections to the genotype. NEAT is based on three key ideas: complexification, gene matching using historical markings and speciation. Firstly, in complexification, the approach begins with a simple network with no hidden nodes, differing only in their weights and gradually increases in complexity over evolution. Secondly, it is necessary to provide a way of identifying matching genes for crossover; the approach assigns a unique historical marking to every new piece of network structure expressed by the innovation number as shown in figure



Genome (Genotype)

Nodes Genes	Node 1 Sensor	Node 2 Sensor	Node 3 Sensor	Node 4 Sensor	Node 5 Sensor		
Connect Genes	In 1 Out 4 Weight 0.7 Enabled Innov 1	In 2 Out 4 Weight 0.5 Disabled Innov 2	In 3 Out 4 Weight 0.5 Enabled Innov 3	In 2 Out 5 Weight 0.2 Enabled Innov 4	In 5 Out 4 Weight 0.4 Enabled Innov 5	In 1 Out 5 Weight 0.5 Enabled Innov 6	In 4 Out 5 Weight 0.7 Enabled Innov 11

FIGURE 2.5: NEAT Representation. Shown on the left is a genotype representation, that directly encodes to the phenotype on the right. In the genotype description, each connection gene specifies the "in" and "out" node, weight scalar value, connection expression indicator and innovation number respectively. In the given example, the second gene connection expression indicator has a disabled value, which means the connection between node two and four is not expressed. Figure adapted from Stanley (2002).

2.5. Thirdly, the approach provides speciation with intention of preserving diversity and to allow competition to take place within their niches instead of within the population at large. This gives the topology the chance of being optimized without the need of competing with all the other networks, and fitness evaluation is explicitly shared among the networks in the same niche. This has an advantage of minimizing dimensionality of the search space and emergent properties can be obtained early in evolution, elaborated and refined as new genes are added (Altenberg, 1994). This algorithm through speciation employs categorical modularity (Darwen and Yao, 1996) in the genotype space. Standard NEAT does not exploit or use regularity at either the genotype or phenotype level for the benefits of improving problem solving behavior. NEAT has been used to evolve solutions to a number of multi-agent task such as Keep-away soccer (Taylor et al., 2006b; Taylor et al., 2007a; Taylor and Stone, 2009; Verbancsics and Stanley, 2010) and also as a function approximator in reinforcement learning (Whiteson and Stone, 2006).

All of the EA methods considered this far rely on direct encodings to represent policies (mapping between genotype and phenotype is one-to-one). Evolving policies, especially in multi-agent learning requires searching a high dimensional space. Hence these methods suffer from the curse of dimensionality. The main limitation of such encoding is that even if the different parts of the phenotype are similar they must be distinctly encoded and hence discovered separately which makes the search process less efficient, especially for complex tasks.

Generative Encoding in Neuro-Evolution

Generative encoding is motivated by generative development in biology, in which the genotype maps to the phenotype indirectly through the process of growth. A gene in the genotype can be used multiple times during the process of phenotype development, resulting in a compressed genotype for complex phenotypes (D'Ambrosio and Stanley, 2013). Since most of the elements in nature have regular or symmetric patterns, the

generative encoding provides an opportunity to explore patterns and regularities by encoding the genotype as a description that maps indirectly to the target structure (Stanley et al., 2009).
There are generally two forms of gene reuse: First, code reuse through repeated patterns with or without variations in size. If a pattern has to be repeated at a certain point of the phenotypic structure the same gene group expresses its specification. Second, occurrence of reuse is when a single unique group of genes in the genotype is capable of initiating alternative developmental pathways (Hornby, 2005; Lipson, 2004). This has been a research focus for the past three decades, where Lewis (1978) discovered that there are Hox genes responsible for regulating body morphology and specifies development of different segments in the anterior-posterior axis of the fly. Later on, Cohn et al. (1997) found that a fibroblast growth factors (FGFs) of a chick embryo induces development of a limb such as a wing or leg depending on the locality of expression.

There are a number of approaches to indirect encoding as demonstrated by some original work by Kitano (1990) and Gruau (1996). Kitano introduced an indirect encoding that utilizes genotype rewriting rules called grammar-based encoding (Kitano, 1990). Gruau introduced a cellular encoding approach that extensively contributed to the introduction of developmental or generative encoding approaches (Gruau and Quatramaran, 1996). Stanley (2007) then developed a *Compositional Pattern Producing Network* (CPPN) encoding that encodes functions within nodes and exploits regularities of the problem. This encoding led to an extension of NEAT called *Hypercube-based NEAT* (HyperNEAT) (Stanley et al., 2009) which uses NEAT to evolve the CPPN network.

As an extension of HyperNEAT method a new approach was developed called Developmental Symbolic Encoding (DSE) (Suchorzewski, 2011). This approach, as shown in figure 2.6, utilises concepts of both Cellular encoding and CPPN (hybridizing cellular encoded method with HyperNEAT method). Similar to cellular encoding, the network is developed through cell division, and compared to CPPN, it forms connectivity patterns between groups of nodes and it is claimed to exploit geometric patterns in the domain space. This approach was tested for the bit mirroring problem (Suchorzewski, 2011) and on visual discrimination tasks (Suchorzewski and Clune, 2011). The results of both experiments demonstrated that the approach is able to produce regular patterns and scalable networks. To the authors knowledge, this approach has not been tested in any multi-agent setting.

HyperNEAT, a method of interest to this study, combines two networks in the encoding, namely: CPPN and the Substrate (Stanley, 2007). CPPN is a network of functions of geometry that generate connectivity patterns with nodes situated in multiple dimensions of the Cartesian space, where each node is itself a mathematical function. Each function (for example: Sine, Gaussian, Sigmoid and Linear) expresses a certain

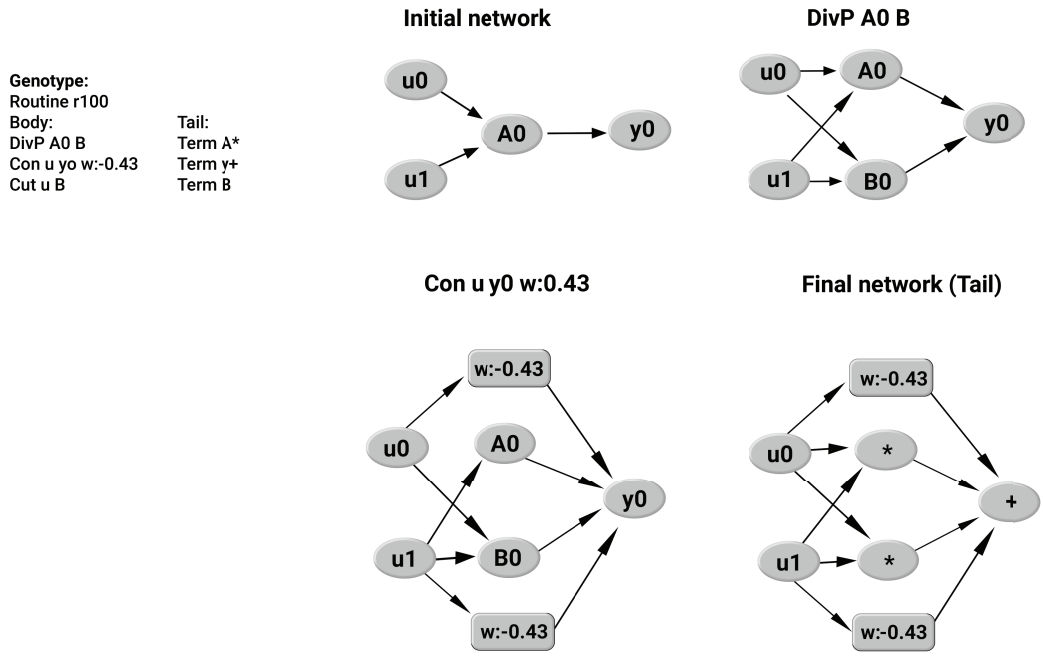


FIGURE 2.6: DSE representation. Top-left : shows the Developmental Symbolic Encoding (DSE) genotype encoding, a tree of routines that comprising of a list instructions (in this example three instructions for the body and three instructions for the tail) that defines how to develop the network. The instructions are applied to an initial network, and through cell division the network is progressively developed to a final network (bottom-right). Figure adapted from Suchorzewski (2011).

desirable property of the task domain. For example, the Gaussian function $(G(x))$ represents bilateral symmetry and the Sine function $(Sine(x))$ represents repeating motifs in the task. The CPPN encoding is then represented as a directed graph that determines which functions are connected. Generally, the CPPN is a variation of ANN, in that instead of using a single activation function (for example, Sigmoid function), CPPN use several sets of activation functions. The output of each node is computed as a sum of weighted inputs, which is applied to the activation function. Each node i of a hidden layer, has activation level, y_i that is computed based on the input signals x_j and corresponding connection weights w_{ij} as given in equation 2.7:

$$y_i = \sigma \sum_{j}^{N} w_{ij} x_j \tag{2.7}$$

where σ is the activation function (for example, Sigmoid function). Each node of a CPPN hidden layer can have its own different activation function, representing a certain property of the task domain.

The aim of HyperNEAT is to extend CPPNs, which encode spatial patterns, to also represent connectivity patterns (D'Ambrosio and Stanley, 2013; Stanley et al., 2009; Altenberg, 1994). NEAT can then evolve CPPNs to exploit the regularity in the geometry

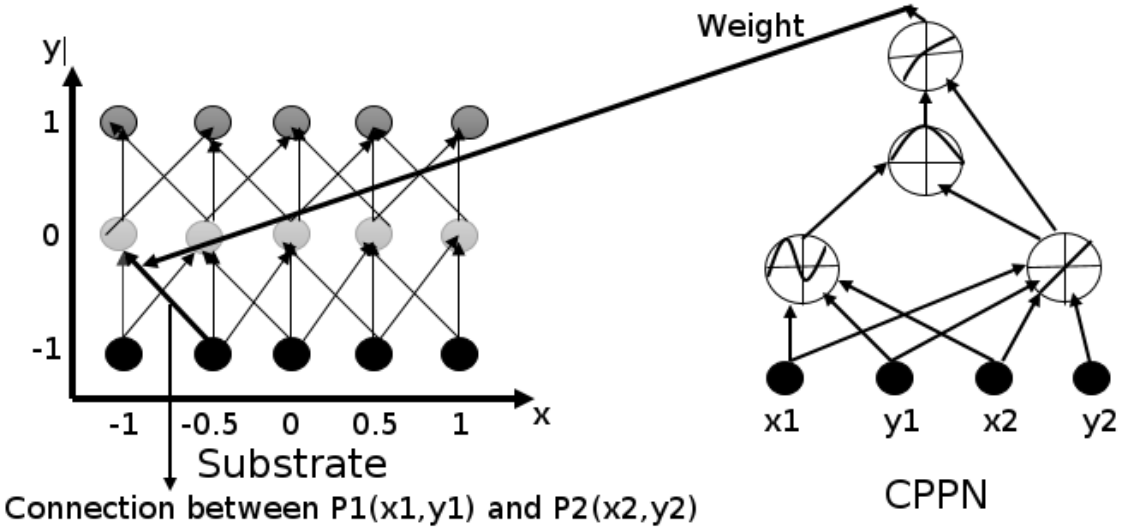


FIGURE 2.7: HyperNEAT representation. The encoding has two networks: Substrate (left-side) and a CPPN (right-side). Substrate encodes the geometry of a given task. CPPN inputs are coordinates of each queried connection, for example, $P1(x1, x2)$ and $P2(x2, y2)$ and compute the weight of each queried connection.

of the problem space and produce regular patterns in the solution space. This produces compressed genotype representations that aids easy scalability of the solution. The nodes are situated in the substrate to reflect the geometry of the task (Stanley et al., 2009; Clune et al., 2009; Gauci and Stanley, 2008; Risi and Stanley, 2012). As a result, the connectivity of the substrate is a function of the task geometry. For example, consider nodes in a two dimensional grid space with values ranging from -1 to 1 in both axes. CPPN inputs are obtained from querying each connection in the substrate (that is, taking values of two end points of a connection), for example $P1(x1, y1)$ and $P2(x2, y2)$. Those inputs labeled $x1, y1, x2, y2$ represent a point in four-dimensional space, and the output represents the weight of the connection as shown in figure 2.7 .

Regular HyperNEAT was found to face a great challenge in accounting for irregularities in various problem domains (Clune et al., 2008). In an attempt to extend HyperNEAT to domains that are highly irregular, Clune et al. (2009), proposed the Hybridizing Indirect and Direct encoding (HybriID). The HybriID method integrates the ability of direct encoding to optimize the parameters and indirect encoding ability to discover and exploit regularities in the problem domain. It runs HyperNEAT for a fixed number of generations and then switches to NEAT for weight optimization without topological mutation for the remaining evolutionary runs.

The other drawback with regular HyperNEAT is that the experimenter has to decide on the topology of the substrate (that is, the number of hidden nodes and their distribution in the geometry). Risi and Stanley (2012) provided a solution that extended this work to

evolvable substrates within HyperNEAT. Instead of prior placement, this method of evolvable substrate discovers the best topology of the substrate and, as a result, that expands the scope of neural structures that can be evolved. HyperNEAT was used to successfully discover a controller for food-gathering (Stanley et al., 2009), line-following robots (Drchal et al., 2009), evolved walking behavior of a quadruped robot (Clune et al., 2009), and controllers for robot keep-away (Verbancsics and Stanley, 2010), and produced an evaluation function for checkers (Abul et al., 2000).

HyperNEAT has successfully generated regularities in the solution space through exploiting regularities in the problem space. However Clune et al. (2010) discovered that HyperNEAT has difficulties in generating modular solutions (that is, phenotype modularity). Regular HyperNEAT exploits emergent regularity and emergent modularity in the genotype (this is because some regularities are repeated motifs which are encoded in the genotype), but does not discover emergent modularity in the phenotype (that is, does not produce modular solutions). As a result, Verbancsics and Stanley (2011) extended this approach to enable phenotype modularity through setting a bias towards local connectivity, an approach called HyperNEAT with Link Expression Output (HyperNEAT-LEO). Clune et al. (2013), demonstrated that phenotypic modularity is a by-product of selection to minimize connection costs. That is, evolving controllers with selection pressures to optimize network performance and minimize costs yields networks that are significantly more modular and adaptable than those that only select for performance. However, this approach has not been tested on complex collective behavior tasks and it remains unknown how such an approach can be effectively utilized in evolution of such tasks.

The HyperNEAT-LEO approach allows standard HyperNEAT to evolve the pattern of weights independently from the pattern of connection expression. Link Expression Output (LEO) is represented as independent output to the CPPN that specifies if a connection can be expressed or not (Verbancsics and Stanley, 2011). The locality is expressed through a Gaussian function which gets as input the difference between coordinates (that is Δx, Δy and Δz in a $3D$ space). The Gaussian function has a peak value when the Δx fed as input is zero. The connection is expressed only if the link expression output magnitude is greater than a LEO threshold (by default it is zero). LEO attains a value greater than zero when Δx, Δy and Δz approaches zero. The Gaussian function provides the principle of locality since as the difference between coordinates approaches zero, the greater is the output of the Gaussian function. The output of Gaussian function is combined with bias towards locality in a step function to generate the LEO output which specifies if the connection could be expressed or not.

The concept of seeding LEO with principles of locality enables HyperNEAT-LEO to exploit emergent regularity and discover emergent modularity in the phenotype. The

HyperNEAT-LEO outperformed standard HyperNEAT on a Retina Problem (Verbancsics and Stanley, 2011). The Retina problem is a problem domain where modularity is known to be helpful and has been used to test if a particular NE method can produce modular ANNs solutions. The ANN was evolved to recognize and classify objects on the left and right side of an artificial retina (Verbancsics and Stanley, 2011). The artificial retina has eight pixels, half on the left and the half on the right, and all representing an input layer of the network (each pixel containing a binary value, to give a maximum of 16 patterns, thus 0000 to 1111). The goal was to have an output indicating either [L AND R] or [L OR R] (that is, true if there is either an object on both sides or true if there is either an object on the left or right side of an artificial retina respectively). In another variant of the retina problem, the test on the left was independent from the test of validity of a pattern on the right. In such a case for a modular ANN solution the input-output connections on the left side are detached from those on the right side of an artificial retina. HyperNEAT-LEO performed well in finding a solution to a modular retina problem and as a result it captures the key features of natural organisms: phenotypic modularity and regularity (Verbancsics and Stanley, 2011).

2.2 Collective Behavior Transfer

This section discusses various types of behavior transfer methods available in literature that have been used in machine learning to leverage learning in target task by transferring behaviors from the source task.

2.2.1 Reinforcement Learning Behavior Transfer

For RL and EA to be useful in real world and simulated multi-agent systems, the learning process needs to be accelerated so that optimal solutions can be obtained with minimal and reasonable computational effort. There have been many attempts to accelerate the learning process both in RL and EA using Transfer learning (Taylor et al., 2006b). Transfer Learning (TL) was first introduced in RL to leverage learning of one task by utilizing policies acquired from prior learning of a related but simple task. This concept was borrowed from psychology (Klahr and Carver, 1988), where learning how to perform a task is supported by the experience gained from performing a related but relatively simple task. TL is similar to a broad concept of incremental evolution (Christensen and Dorigo, 2006), where evolution starts on a simple task and then transfer evolved controllers to a complex task. However, unlike incremental evolution, where a goal-task has to be organized into a number of subtasks of increasing complexity and evolution parameters are progressively altered during evolution and in phases (gradually) (Gomez and Miikkulainen, 1997), TL transfers controllers between two different tasks across domains or within the same domain (Taylor and Stone, 2009). TL is often viewed as a radical version of incremental evolution and the goal of TL is to

accelerate evolution of behaviors in the target task by utilizing evolved controllers from a source task.

There are two main requirements for TL to be effective: First, if there is a suitable mapping technique between the source state-action space $S_s \times A_s$ and the target state-action space $S_t \times A_t$ and, Second, if there is a common state-action representation or shared features between the tasks (Taylor and Stone, 2009). The current research of TL in RL to meet this requirement has focused on shared representation and mapping from source to the target task, which provides a common understanding between the two tasks and avoids negative transfer (Woltz et al., 2000; Taylor and Stone, 2009). Woltz et al. (2000) defines negative transfer as a behavioral psychology term that refers to the interference of the prior acquired knowledge with the learning process on a new task. This relates to experience with one task that could cripple performance on a different but related task. Taylor and Stone (2009), suggested that negative transfer can be avoided by selecting a source task closely related to the target task. Carroll and Seppi (2005) motivated the need for general similarity metrics that could enable robust transfer.

This study tested four similarity metrics: policy overlap, transfer time, Q-Values and reward structure. The policy overlaps distance measure search for the number of states with identical policy between source and the target task. Transfer time is the measure of similarity that seeks to quantify task performance advantage gained from transfer. Using TD methods in RL, the similarity measure can be a root mean squared error between the Q-Values, $Q(s, a)$ or immediate expected reward $R(s, a)$ for choosing an action, a in state s of source and target task. Carroll and Seppi demonstrated that there is no best similarity measure for mapping between all source and target tasks, in all situations and all tested metrics were beneficial in different situations.

Furthermore, determining how best to translate policies between the source task and target task in TL is an open problem (Pan and Yang, 2010). A mapping function is required, formally represented as a pair of functions (ρ_s, ρ_a) where $\rho_s(S) : S_{source} \rightarrow S_{target}$ and $\rho_a(A) : A_{source} \rightarrow A_{target}$ (Taylor et al., 2007b). In the previous work, Taylor et al. (2007) proposed a transfer mapping function, *Transfer via Inter-Task Mapping* (TVITM), which was further adapted for policy search methods to Transfer via inter-task Mapping for policy search methods (TVITM-PS) (Taylor et al., 2007a).

This transfer mapping function is a static heuristic method that alters a policy learned from the source task to suit the new task before training takes place (Taylor et al., 2007b) (this transfer approach was adopted in this thesis, section 3.3.4). This method uses human knowledge on how the source task is related to the target task, and then the mapping function is formulated on the analogy of the relationship between the two tasks. Hence, it is not an adaptable and scalable solution and if TL is to be applied to real-life autonomous systems there has to be a way to determine how to translate representation between tasks

independently (Taylor et al., 2013). An attempt to address this problem was made by Ammar et al. (2012) using a shared subspace. The subspace will be populated by shared features between the source and the target task. The projection of each state from source and target to the subspace enables the mapping of representations from the source task to target task.

Taylor et al. (2009) proposed a classification of transfer learning approaches into five categories: lifelong learning, imitation learning, human advice, shaping and representation transfer. Lifelong learning as suggested by Thrun (1998), is sequential learning where an agent learns domains of tasks consecutively over an extensive period and with knowledge sharing across multiple tasks. This concept was further extended to RL by Sutton (2007), who suggested that continuous agent learning in an environment is necessary, since the environment may change overtime, resulting in knowledge acquired from prior learning becoming insufficient. In Sutton (2007), an agent was made to learn several local environments over a period, subsequently tracking changes in parts of each environment. The empirical results suggested an improvement in performance compared to a method that depends on a once off transfer from prior learning. Several other authors have explored this technique, with some measure of success (Ruvolo and Eaton, 2013; Ammar et al., 2014), and challenges were addressed such as the need to measure the relevance and accuracy in retained knowledge alongside that of training models for a new task (Silver et al., 2013) were addressed. The lifelong learning approach is suitable for complex multi-agent problems where knowledge transfer happens online and multiple tasks are learned simultaneously during the adaptation of agent behavior.

Imitation learning is a transfer learning technique, in which the learning agent adapts its behavior by observing the behavior of an expert mentor. A number of authors have used this method to transfer knowledge from an expert to a learner, such as an implicit imitation model (Price and Boutilier, 1999; Price and Boutilier, 2003), a method whereby an agent observes the state transitions generated by mentors actions and uses this information to update estimated values of its own states and actions.

Human advice is a technique where human knowledge is used to guide the learning through passing some suggestions to the learning agents (Maclin et al., 2005; Torrey et al., 2010) or providing on-line feedback. These techniques require human intervention, and as such, human participation through on-line feedback becomes an integral part of learning. Torrey et al. (2010) suggested a human advice approach applied in RL multi-agent contests, thus, transfer between keep-away (Stone and Sutton, 2002) and Break-away (Torrey et al., 2005). A Break-away task is a subtask of RoboCup soccer. Unlike keep-away, the objective of break-away learners (that is, a team of *attackers*) is to score a goal against an opponent team. The opponent team consists of *defenders* and a *goalie* controlled by a hand-coded policy. The action choices available to a learner are to keep the ball, pass to a teammate, or score a goal. The simulation ends

when an opponent gains ball possession, ball goes out of bounds, or threshold time is exceeded.

In the case of behavior transfer between keep-away and break-away task, the human advice was encoded in logical rules from prior learning and applied to a new task. In this way, the advice provides a mapping function between tasks and identifies the differences between tasks. The other example of human advice method, was suggested by Yong et al. (2006), where human knowledge is incorporated into neuro-evolution in real-time. In this work, by Yong et al. (2006), the advice is expressed in a formal language and converted into nodes and connections of a network, using *Knowledge Based Artificial Neural Network* (KBANN) algorithm (Maclin and Shavlik, 1996). In our study, we are interested in methods where agents learn to adapt behavior autonomously without human intervention.

Reward Shaping is one of the popular approaches used with policy-based techniques to transfer policies across tasks. Reward shaping provides an additional reward acquired from prior learning to reduce the number of suboptimal actions made and the learning time (Randlov and Alstrom, 1998; Ng et al., 1999). This approach modifies the basic reward with that additional reward to bias the agent's exploration as shown by equation 2.8 (the formula for the off-policy TD method):

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + F(s,s') + \gamma \underset{\pi}{max} Q(s',a') - Q(s,a)] \tag{2.8}$$

where $F(s,s')$ is the shaping reward. It was then demonstrated that if reward shaping is used improperly, it can alter the agent's goal and problem (Randlov and Alstrom, 1998). An attempt to solve this problem led to a proposal of potential-based reward shaping (Ng et al., 1999) given by equation 2.9:

$$F(s,s') = \gamma\Phi(s') - \Phi(s) \tag{2.9}$$

where Φ is the potential function defined over a source and target state s, and γ is the discount factor. Reward shaping on multi-agent learning has received relatively little research attention. The challenge with this approach as with human advice is that human knowledge has to be injected into the task definition to improve learning performance. The interest we have in this study is on those methods that enable transfer of knowledge without human modifications.

Representation transfer is the approach where basis functions are learned in one task and applied in another without modification. This approach was first used on task sequences where reward functions differ but the state space remains the same (Ferguson and Mahadevan, 2006). If the state spaces differ significantly some mapping functions are required to transform basis functions from one state space to another (Ferguson and Mahadevan, 2008). However such mapping requires prior knowledge of both representations of the state spaces to be successful. Taylor and Stone proposed another

method with a name "Elaboration" where the function approximator (FA) is modified to allow for a change in representation to account for a change that occurs from one task to another (Taylor and Stone, 2007).

The above classification captures many transfer learning methods. However, it is no longer a sufficient classification, since there are overlaps and gaps that exclude new approaches such as shared features (Konidaris et al., 2012). Konidaris et al. (2012) presents a method that uses shared features, and its definition is similar to a method introduced by Taylor et al. (2007) which uses a hand-coded mapping function that maps weights of the function approximators of each task to another. Konidaris suggests that their work has two advantages, first being that the mapping constructed between each task and the agent space grows linearly instead of quadratic or exponentially. The second advantage being that, since the method utilizes a shaping function, the mappings can be constructed between state descriptors, rather than between function approximation terms. This allows a blackbox transfer in terms of function approximations and transfer can be done to very different function approximators where a direct mapping would be difficult to obtain (Konidaris et al., 2012). There are other methods of transfer that are not easily classified, such as policy re-use (Fernando and Manuela, 2006), policy fragments learned in a symbolic form using inductive logic programming (Torrey et al., 2006) and sample transfer (Lazaric et al., 2008). By contrast, few methods have been proposed for multi-agent transfer learning in NE, using either direct or indirect encoding (as shown in table 2.1).

2.2.2 Neuro-Evolution for Behavior Transfer

Transfer learning for multi-agent task remains a relatively unexplored research area, with notable exceptions including, Didi and Nitschke (2016), Taylor et al. (2007) and Verbancsics (2011). No other work has used neuro-evolution methods for transferring policies between multi-agent tasks (as shown in table 2.1). Taylor et al. (2007) introduced a method called "Transfer via Inter-Task Mapping for Policy Search methods (TVITM-PS) an extension of a method named: Transfer via inter-task mapping, discussed in section 2.1.1. TVITM-PS transfers policies between tasks with different state-action spaces (that is, multiple representations). The method defines a mapping function $\rho(\pi_{source}) \rightarrow \pi_{target}$ to successfully transfer policies between a pair of tasks. This method relies on human knowledge of the two tasks to be able to design how the two tasks are related. It is the only method used with NEAT that transfers between multi-agent tasks. If there are topological differences in task representation (that is, number of neurons on the input-output layers differ), TVITM-PS provides incomplete mapping that needs further training to optimize the policies for the new task. This enables transfer to happen in multiple representations. However in this method the state-action variables will always increase with the number of agents, which causes this approach to suffer from the curse of dimensionality. This approach was selected in this

thesis for transferring controllers (evolved with NEAT) between collective behavior tasks of increasing complexity (section 3.1.1).

A new approach termed "Bird's eye view" (BEV) was then introduced by Verbancsics and Stanley (2011), to address this particular problem. This approach demonstrated that generative encoding (that is, indirect encoding) can allow transfer between two tasks with different state-action spaces without a change in representation. This approach was selected in this thesis for transferring controllers (evolved with HyperNEAT) between collective behavior tasks (section 3.2.1). The work by Verbancsics and Stanley successfully tested and presented empirical results for different keep-away task configurations and dimensionality. However, different evolutionary search methods for neuro-evolution have not been tested with transfer learning. Verbancsiscs and Stanley (2011) compared direct to indirect encoding search methods - however, the test was restricted to objective-based search methods only. The interest of this research is to extend this work to other neuro-evolution search methods such as non-objective search (diversity maintenance) methods.

2.3 Diversity Maintenance Methods

Traditional EAs with elitist selection are suitable for locating an optimal solution in a unimodal problem domain, which converges to a single optimal solution (Sareni and Krahenbuhl, 1998). Problems in which NE methods are applied are typically multimodal domains, in which multiple peaks (global or local) exist. For optimal global solutions to be identified, a method that sustains population diversity has to be used, otherwise the search can easily become trapped in local optima before discovering global optimum. This problem is referred to as premature convergence and is caused by genetic drift in evolutionary search (Eiben and Smith, 2003). Niching schemes are the primary technique for sustaining population diversity in EAs and have been extensively studied as a potential solution to premature convergence (Sareni and Krahenbuhl, 1998; Stanley and Miikkulainen, 2002; Mouret and Doncleux, 2009; Moriguchi and Honiden, 2010a). Niching techniques have introduced a mechanism for evolutionary algorithms to explore multiple fitness landscapes and avoiding getting trapped in the local optima before reaching a global optimum.

Many niching mechanism have been studied that seek to maintain population diversity in the genotype and phenotype search spaces. Examples are: speciation (Goldberg and Richardson, 1987), fitness sharing (Sareni and Krahenbuhl, 1998), crowding (Sareni and Krahenbuhl, 1998), and implicit fitness sharing (Smith et al., 1993). Each of these methods is applied at the genotype level. Lehman and Stanley (2008) introduced a technique called novelty search, which measures diversity at the behavioral space or

based on the phenotypes. This method differentiates individuals in the population based on behavioral properties and aims to abandon objective based search for novelty search.

Novelty search is a special case of behavior diversity maintenance (Mouret and Doncleux, 2009) that has become a popular method for directing evolutionary search and boosting quality of evolved solutions in a range of applications (Lehman and Stanley, 2008; Doncieux and Mouret, 2010; Doncieux et al., 2011; Lehman and Stanley, 2011a). Whilst, general behavioral diversity maintenance selects diverse behaviors with respect to the current population, novelty search takes into account an archive of previously evolved behaviors and behaviors in the current population. That is, the novelty is the average distance in behavior space of one individual to its closest neighbors in the current population and in an archive of the behaviors recorded before. In this way, a search for novel phenotypes (behaviors) replaces the fitness function traditionally used to direct evolutionary search (Eiben and Smith, 2003).

Previous work has demonstrated the efficacy of novelty for evolving controllers in a range of tasks of varying complexity (Velez and Clune, 2014) and such controllers out-performed controllers evolved with objective-based search in a range of evolutionary robotics tasks (Doncieux and Mouret, 2010; Gomes et al., 2016a; Gomes et al., 2016b). Novelty search has been exploited in neuro-evolution, especially methods that are more susceptible to premature convergence (such as CCEA shown in table 2.1) and domains that are more prone to deception (such as maze-nagivation tasks) (Mouret and Doncleux, 2009; Silva et al., 2016). The novelty metric measures how different an individual is from the rest of the population (that is, current population and an archive of past novel individuals) with respect to behavior. This evolutionary process creates selection pressure towards behavioral innovation. Developing a novelty search strategy involves identifying a task-specific behavior function that maps each genotype to a behavior (Kistemaker and Whiteson, 2011). Therefore, novelty search is performed in the phenotype instead of genotype space.

However, related work suggests that complex tasks such as collective behavior tasks associated with swarm robotics (Gomes and Christensen, 2013b) that are characterized by high dimensional and deceptive fitness landscapes (Cuccu and Gomez., 2011), novelty alone does not offer advantage over objective based evolutionary search. Hybridizing objective and novelty search tend to direct evolution towards effective exploration and exploitation of the search space and evolve effective high-quality solutions overall (Cuccu and Gomez., 2011; Gomes et al., 2014a; Gomes et al., 2015; Silva et al., 2016). Different approaches have been introduced for hybridizing objective (fitness function) and novelty search so as to evolve high-quality solutions. One common approach is linear combination of novelty score and fitness score using a weighted sum (section 3.4.2).

An alternative approach is the minimal criteria novelty search (MCNS) introduced by Lehman and Stanley (2010). MCNS defines a fitness criteria in which individuals must meet to be selected for reproduction and effectively reduces the dimensionality of the behavior space. The downside to using MCNS is that a proper care has to be taken when selecting the minimal criteria due to the restrictions that are imposed by each criteria on the search space, and if there are no individuals in the population that meets the selected criteria, there is no selection pressure and evolution enters a random drift (Lehman and Stanley, 2010; Silva et al., 2016). A third approach is to use Pareto-based multi-objective EAs (MOEAs) that simultaneously optimizes novelty and objective during evolution (Mouret and Doncleux, 2009; Mouret et al., 2012). Linear combination of objective and novelty search approach has been adopted in this thesis because of the simplicity of implementation and that previous research demonstrated efficacy of this approach in collective behavior evolution (Gomes et al., 2014a; Didi and Nitschke, 2016b).

The behavior characterization is a critical stage in the application of novelty search strategy and requires prior knowledge about which behavior features affect performance. The behavior of an individual x, having a set of N behavioral vectors, is determined by a behavior function, $F : x \rightarrow \Re^N$. Typically, the novelty of each individual, is the mean distance to the k-nearest neighbors of that point, where k is a constant that is experimentally determined. The novelty metric is given by equation 2.10:

$$nov_x = \frac{1}{k} \sum_{i=1}^{k} \delta(x, y_i) \tag{2.10}$$

where y_i is the i^{th}-nearest neighbor of x with respect to the behavior space and δ is the distance metric.

Gomez (2009) compared several distance metrics based on both genotype and phenotype space, namely: Euclidean distance, Hamming distance, relative entropy and *normalized compression distance* (NCD). In the work of Gomez (2009), the agents behaviors were defined as their action sequences and the results demonstrated that similarity between individuals measured by NCD outperformed the rest. Trujillo et al. (2008), used a NCD method to substitute a topology-based distance metric with a behavior-based one. The empirical results show that NCD method outperforms the original NEAT that utilizes topological-based diversity. Despite all the work in this research area, there is no common agreement on the best way to compute behavior similarity. Apart from the work of Gomez (2009) and Trujillo et al. (2008), there is no other work that has used NCD as a distance metric for novelty search in neuro-evolution. However, recent work in neuro-evolution (Lehman and Stanley, 2010; Kistemaker and Whiteson, 2011; Gomes et al., 2014a; Didi and Nitschke, 2016b; Silva et al., 2016), has seen an increase in the use of Euclidean distance metric to compute behavior similarity.

Furthermore, previous work by Wineberg and Oppacher (2003), suggested that Hamming distance is effective where similarity measures use binary or symbolic genotypes and that Euclidean distance is effective for genotypes with real value genes. The behavioral vectors in this work are real values and considering previous work by Gomes (2014), that used a Euclidean distance metric for novelty search in collective behavior tasks, this work adopts a Euclidean distance metric for computing behavior similarity.

Novelty search has been found to be unreliable when the size of the behavior space is unlimited (Lehman and Stanley, 2010). Lehman and Stanley (2010), addressed this problem by proposing a *minimal criteria novelty search* in which a minimal performance criteria is set - where, if an individual does not meet that particular value, its novelty is set to zero. Applying a reconfigurable behavioral space that limits the space of viable behaviors, Lehman and Stanley demonstrated the superiority of this strategy over searching for novelty only. Kistemaker and Whiteson extended this work by investigating the critical factors that affect the performance of novelty search and they suggested that the design of a good behavior function is essential to the success of novelty search (Kistemaker and Whiteson, 2011). However, finding a good behavior function (characterization) is not an easy task (Kistemaker and Whiteson, 2011), hence further research is necessary in this aspect of novelty search.

Doncieux and Mouret (2014) demonstrated that selective pressure have a critical role in evolutionary robotics (ER). That is, exploration of the behavior space is as equally important as exploiting the highly performing behaviors in increasing search efficiency and effectiveness. Furthermore, exploiting prior knowledge in form of evolved controllers is beneficial to evolution of behaviors, but proper care is needed in selecting the type of knowledge to transfer and balancing a trade-off between behavior space exploration and exploitation of behaviors.

Previous work by Kelly and Heywood (2014) on comparing genotype to behavior diversity for keepaway teams using genetic programming (GP) and further work by Gomes et al. (2014), inspired the use of behavioral diversity maintenance in this thesis. Gomes et al.(2014) made an empirical study comparing behavioral diversity performance at the team versus individual level using CCEA. The study was applied on collective behavior tasks (Keep-away soccer (Stone and Sutton, 2002) and predator-prey domain (Haynes and Sen, 1996b)), and is the only study that applies behavior diversity and explicit genotypic diversity on collective behavior tasks (as shown in the table 2.1). However, this CCEA study does not test the impact of maintaining population diversity on collective behavior tasks to be transferred to tasks with increasing complexity. It is against this background that our research seeks to investigate the impact of incorporating diversity maintenance mechanisms in adapting neuro-evolution methods to derive collective behavior that will be transfered to boost learning in collective behavior tasks with increasing complexity. In this thesis, we compare our NE methods for transfer learning with RL, as RL is a well established method for transfer learning

and thus forms a task performance benchmark with which to compare our various NE approaches to collective behavior transfer learning.

2.4 Discussion and Summary

This chapter reviews the important existing work related to this thesis. It also provides a background on significant techniques previously used by others and that have been employed in this thesis. The review has focused on the current state of the art in collective behavior transfer, with interest on neuro-evolution and reinforcement learning as adaptation methods. It has been indicated in the review, that there are several successful approaches to collective behavior transfer (with limited complexity) based on reinforcement learning. However, there is very limited research work on collective behavior transfer based on neuro-evolution as adaptation method, where neuro-evolution is applied to collective behavior tasks to derive a collective behavior that will be transfered to improve learning in tasks with increasing complexity. The work in this thesis differs from the other work in many aspects. First, while other researchers have demonstrated the importance of transfer learning in behavior adaptation, no work has focused on the impact of different evolutionary search methods on collective behavior transfer in neuro-evolution. Secondly, this literature review shows, that to the best of the authors knowledge, no previous work has tested the impact of maintaining behavior diversity during collective behavior adaptation, where collective behaviors are transferred between tasks increasing complexity. Finally, we are also interested in investigating the impact of using non-objective versus objective based search methods to adapt collective behavior, before being transferred for further adaptation to collective behavior tasks with increasing complexity. The following chapter will address this current gap in the state of the art and describe the neuro-evolution and collective behavior transfer learning methods used in the experiments of this thesis.

Chapter 3

Methodology

In the experiments[1] described in this thesis, behaviors are evolved for collective behavior tasks that are controlled by ANNs, and neuro-evolution is required for evolving these controllers. The behavior transfer framework which our work is based on, is given in figure 3.1, where behavior adaptation begins in the source task, to derive behaviors that will be transferred to bootstrap learning in collective behavior tasks with increasing complexity (section 4.4.1).

NEAT has been selected to represent direct encoding and HyperNEAT to represent indirect encoding neuro-evolution methods. NEAT is appropriate for this study because firstly, it has been widely applied to collective behavior tasks (Taylor et al., 2006b; Torrey and Shavlik, 2009; Degrave et al., 2015; Knudson and Tumer, 2012; Moshaiov and Tal, 2014), and secondly, it outperformed many direct encoding methods for evolving controllers to solve control and sequential tasks (Stanley and Miikkulainen, 2002; Stanley and Miikkulainen, 2004c; Stanley and Miikkulainen, 2004b). HyperNEAT is a generative encoding that extends NEAT and has been successfully used in previous studies for evolving controllers for collective behavior tasks (Verbancsics and Stanley, 2010; D'Ambrosio and Stanley, 2013). These neuro-evolution methods are compared with reinforcement learning (a traditional policy search method for control and sequential tasks).

3.1 Neuro-Evolution of Augmenting Topology (NEAT)

The NEAT method, introduced by Stanley and Miikkulainen (2002) evolves artificial neural networks to solve difficult control and sequential tasks. Empirically, NEAT has shown success in a wide variety of tasks, that includes a broad range of collective behavior tasks and robotics (Stanley, 2004; Stanley et al., 2005; Taylor et al., 2006b; Torrey and Shavlik, 2009; Degrave et al., 2015; Knudson and Tumer, 2012; Moshaiov and Tal, 2014). However, there has been relatively little research as to the efficacy of NEAT as a

[1] Source code and executables found here: https://github.com/sdidi/KeepawaySim

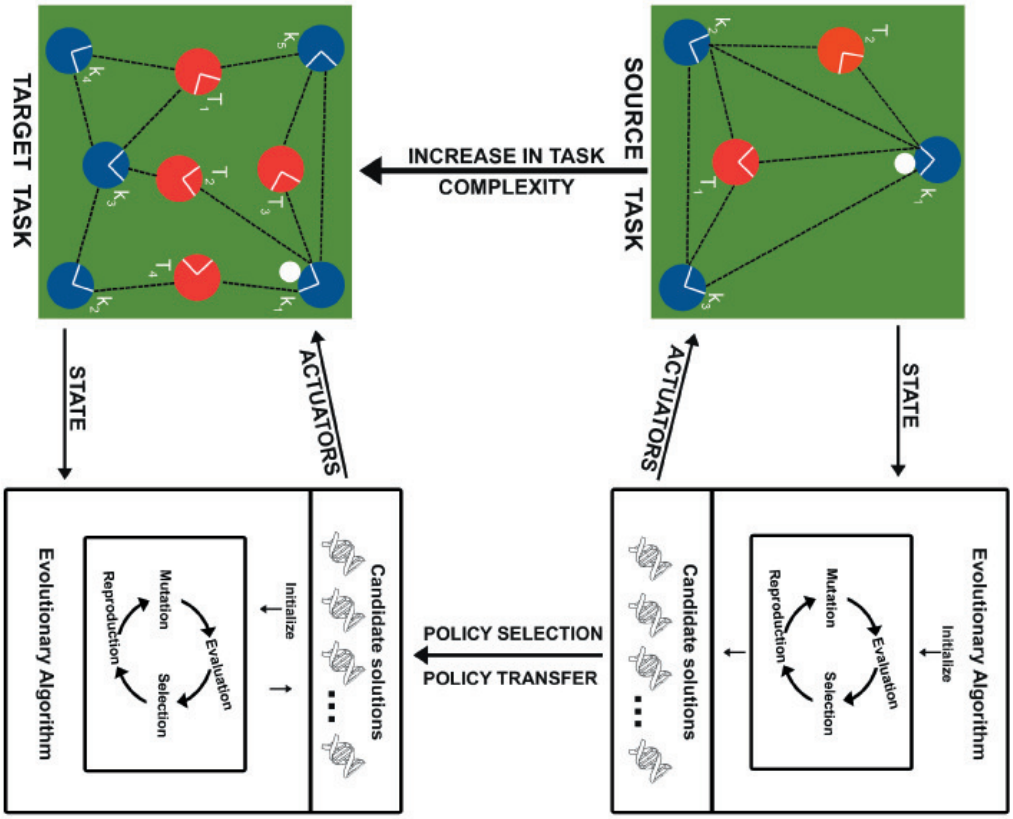


FIGURE 3.1: Collective behavior transfer framework. The figure shows behavior adaptation in a collective behavior task and behavior (policy) transfer to tasks of increasing complexity. Neuro-evolution is used for behavior adaptation in both the source and target tasks.

policy search method for collective behavior tasks, where behaviors derived are transferred to tasks of increasing complexity for further adaptation.

NEAT, unlike many previous neuro-evolution methods, evolves both connection weights and ANN topologies, and applies three key techniques to maintain a balance between performance and diversity of solutions. These are: gene tracking through historical markings, protecting innovation through speciation, and complexification (that is, starting small and gradually increasing ANN topological complexity).

NEAT assigns a unique historical marking to every new gene so that crossover can only be performed between pairs of matching genes. Using historical markings, it is possible to trace the origin of all genes through the ancestral lines and if two genes share an ancestral gene, they must represent the same structure. As new genes emerge through mutation, a global innovation number is assigned to a new gene and is incremented chronologically each time a new gene is created. When mating occurs, the resultant offspring inherit the same innovation numbers on each gene, which are maintained throughout evolution. In this way, there is a possibility of explosion of innovation numbers from a situation where the same structure appears twice or more in the same generation and receiving different innovation numbers. To avoid that problem, a list of

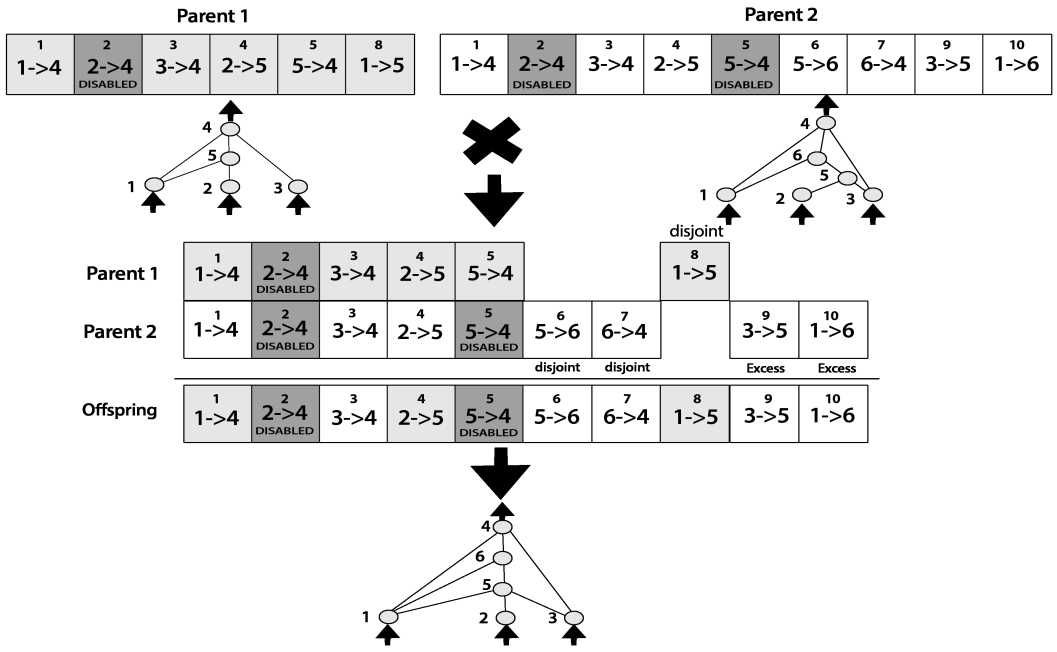


FIGURE 3.2: Cross-over based on historical innovation numbers (Stanley and Miikkulainen, 2002). The figure shows matching genes between Parent 1 and 2, as well as disjoint and excess genes. The matching genes are inherited arbitrarily, whereas excess and disjoint genes are inherited only from the more fit parent. If the parents have the same fitness value, then disjoint and excess genes are inherited arbitrarily from parents as well.

the innovation numbers is kept and any identified identical structures are assigned the same number. Using this approach, crossover only happens between pairs of genes with the same innovation numbers (that is, matching genes). Genes that do not match are either *disjoint* or *excess* and represent a structure that is not present in the other genotype. The disjoint genes are those that appear in the range of other parent innovation numbers and excess gene are those beyond other parent innovation numbers (figure 3.2).

NEAT speciates the population so that ANNs (genotypes) compete primarily within their own niches (identified by historical markings) instead of competing with the population at large. This approach protects topological innovation and creates groupings in the population such that similar topological structures are in the same species. The genotype similarity is measured using compatibility distance metric which depends on the number of excess and disjoint genes between a pair of genotypes. The more disjoint or excess genes are between a pair of genotypes the less compatible they are. A linear combination of excess E and disjoint D genes with the average weight differences of matching genes $\overline{W}$ (including disabled genes), gives a compatibility distance δ presented in equation 3.1:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W} \tag{3.1}$$

where c_1, c_2 and c_3 are constants used to adjust the weight of each factor. N, is the number of genes of the genotype with largest number of genes, used for normalization.

The compatibility measure δ is evaluated with a compatibility threshold δ_{th} to determine if genotypes are of the same species or not. If $\delta < \delta_{th}$ genotypes are considered similar. New species are gradually identified as evolution progress, if a genotype is not compatible with a list of existing species, a new species is produced with that genotype as its representative. Prior to artificial evolution, a species size limit is set and if that limit is exceeded, instead of creating a new species, the compatibility threshold is adjusted. In the experiments of this thesis, the compatibility threshold is made to be dynamic δ_{th}. If the maximum number of species is exceeded, the δ_{th} is adjusted upwards if there are too many species or downwards if there are too few species. This requires re-ordering existing genotypes and helps to limit the number of species in a population. While speciation protects topological innovation, a population is generally not safe from dominance of a single species (especially one with many individuals that perform well). This has adverse effects on the population diversity. NEAT uses explicit fitness sharing to prevent overly large species from dominating the whole population. Explicit fitness sharing imposes a penalty on overly large species by dividing the fitness by the size of the species (Goldberg and Richardson, 1987). Therefore, the size of species has an adverse effect on individual fitness. The adjusted fitness f_i^1 of an individual i is computed using equation 3.2:

$$f_i^1 = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j)} \tag{3.2}$$

where $\delta(i,j)$ is the compatibility distance of individual i from j and sh is a sharing function computed by an equation 3.3:

$$sh(\delta(i,j)) = \begin{cases} 0, & \delta(i,j) > \delta_T \\ 1, & \text{Otherwise} \end{cases} \tag{3.3}$$

where δ_T is the compatibility threshold. This approach provides a natural way of managing the size of species. Species then reproduce by first eliminating under performing individuals from the population, and after mating the remaining individuals in the population, all parents are replaced by their offspring.

The original NEAT approach, starts off evolution with a population of simple ANNs, where only input and output nodes are specified for the ANNs. The appropriate topology (that is, hidden nodes and their connections) and weights are discovered through evolution. This process, that gradually builds topological structures using two special mutation operators: *add hidden node* and *add connection*, is called complexification. The network complexity (measured by the total number of nodes and connections) is gradually adjusted over generations through evolution to match the level of complexity in the problem domain.

In this thesis, experiments were conducted using a C# implementation of NEAT called SharpNEAT 2.0[2], developed by Green (2004). Unlike original NEAT, that utilizes complexification as a primary topology search method, SharpNEAT adds another process that is called pruning (Green, 2004; Lockett et al., 2007). Pruning simplifies network topologies, a process required when the task performance stagnates for a given period (that is, no improvement in the fitness of individuals). This process removes the excess structures by using additional mutation operators: *delete connection* and *delete node*. A connection for deletion is selected randomly, there by simplifying the implementation of this process and a node with only one incoming and outgoing connection is the best candidate for deletion (Green, 2004; Jorgensen et al., 2008).

3.1.1 NEAT Behavior Transfer

There are many ways of transferring behaviors from the source to target task. In this thesis, three methods were tested to ascertain which method best facilitates transfer of behaviors from a source task to initialize evolutionary adaptation in a target task.

- First, the entire evolved population was transferred from the source task (at the final generation of neuro-evolution) and set as the initial population for neuro-evolution in the target task.
- Second, the target population was seeded with the fittest genotype in the source task and used as a bias for initializing the remainder of the target population.
- Third, the fittest 50% of the population evolved for the source task was selected to seed and bias initialization of the rest of the starting population in the target task.

The first approach was found to be the most effective for all methods and tasks tested in this thesis. This approach is presented in algorithm 1 and was thus used in company with the selected mapping function for policy transfer (algorithm 2). Algorithm 2 is applied to ensure that evolved networks are fully connected before behavior transfer occurs.

Algorithm 1 Behavior Transfer

1: *Initialize* s, a
2: **foreach** *genotype* g_i $(i = 1, ..., N)$ *in* P' **do**
3: *Generate a network with number of inputs and outputs corresponding to a* Π_{target} *configuration*
4: *Add the same number of hidden nodes to* Π_{target} *as in* Π_{source}
5: **foreach** *pair of nodes*(n_i, n_j) *in* Π_{target} **do**
6: **if** $\exists$ *link* $L_{i,j} \in \Pi_{source}$ **then**
7: *add link* $L_{i,j}$ *to* Π_{target} *with* $w^t{}_{i,j} = w^s{}_{i,j}$ *in* Π_{source}

[2] http://sharpneat.sourceforge.net/

Algorithm 2 is a transfer mapping function that is an extension of that proposed by Taylor et al. (2007). This transfer mapping function, identifies input and output nodes that are not connected to the network. The nodes identified are then connected directly to the end points (that is, input nodes to connects to the output nodes) with random weights assigned to their respective connections.

Algorithm 2 Transfer Mapping Function

1: *Generate a network* Π_{target} *with nodes*(n_i, n_j) *as in specified configuration*
2: *add the same number of hidden nodes to* Π_{target} *as in* Π_{source}
3: *Initialize* s, a
4: **foreach** *pair of nodes*(n_i, n_j) *in* Π_{target} **do**
5: **if** $\exists$ *link* $L_{i,j} \in \Pi_{source}$ **then**
6: *add link* $L_{i,j}$ *to* Π_{target} *with* $w^t{}_{i,j} = w^s{}_{i,j}$ *in* Π_{source}
7: **else**
8: **if** $\nexists$ *nodes*$(n_i, n_j) \in \Pi_{source}$ **then**
9: *add link* $L_{i,j}$ *to* Π_{target} *with* $w^t{}_{i,j} = random\ weights$

3.2 HyperNEAT: Hypercube-based NEAT

Hypercube-based NEAT (HyperNEAT) (Stanley et al., 2009) is a generative encoding neuro-evolution method that extends NEAT and uses two networks, a *Composite Pattern Producing Network* (CPPN) (Stanley, 2007) and a *substrate* (ANN). This encoding enables evolution to explore patterns and regularities by encoding the genotype as a description that maps indirectly to the phenotype. The main benefit of HyperNEAT is that it exploits task geometry regularity and thus effectively represents complex solutions with minimal genotype structure (Stanley et al., 2009). Through CPPN, encoding patterns are described at a higher level as composition of functions where each function represents certain motif in a pattern. Examples of such motifs are symmetry (for example, with a Gaussian function), imperfect symmetry (by summing up symmetric functions and non symmetric functions), and repetition (with periodic functions such as sine), repetition with variation (by summing up periodic and non-periodic functions).

HyperNEAT uses the evolutionary process of NEAT to evolve the CPPN, in the same way that standard NEAT evolves ANNs. The regularities encoded on CPPN are computed directly from the geometry of the task inputs. The input to the CPPN is derived from the coordinates of the two endpoints in the n dimensional substrate network. Each end point of the sampled connection is represented by the n parameters (that is, equivalent to n dimensions of the substrate hyperplane).

To apply HyperNEAT to a collective behavior task (section 4.4.1), the substrate input and output layer are arranged in two-dimensions to match a two dimensional geometry of the visual field. The input layer is directly connected to the output layer (that is, there are no hidden layers). Each connection between input and output nodes is iteratively queried in the substrate network and the coordinates of the two endpoints (that is, x_1, y_1, x_2 and

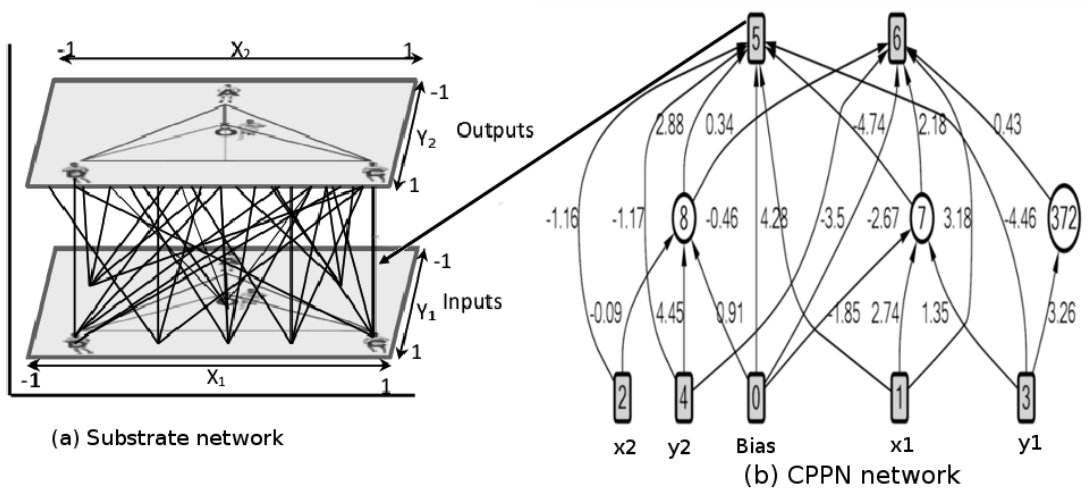


FIGURE 3.3: HyperNEAT configuration for a collective behavior task. Shown in (a) is a substrate network that is applied to a task and measures fitness. Shown in (b) is the CPPN that is evolved using NEAT operators to generate candidate solutions. The CPPN takes coordinates of sampled substrate network connection as inputs and outputs the weight of the connection and LEO expression value.

y_2) are collected and used as input parameters in the CPPN network. This way, CPPN encoding indirectly maps genotypes to ANNs and encodes pattern regularities (such as symmetries, repetitions) of the geometry of a task in the form of the substrate. The output of the CPPN network is the synaptic weight of that connection and the Link Expression Out (LEO) that determines if the connection should be expressed or not. This process is repeated for all connections in the substrate network. An example of this configuration, as used in this thesis is given in figure 3.3.

HyperNEAT was selected as the indirect encoding neuro-evolution method for this thesis, since previous work indicated that transferring the *connectivity patterns* (Gauci and Stanley, 2008) of evolved behaviors is an effective way of facilitating transfer learning in collective behavior tasks (Bahceci and Miikkulainen, 2008; Verbancsics and Stanley, 2010). Further work demonstrated the efficacy of this method in synthesizing controllers to solve collective behavior tasks (D'Ambrosio and Stanley, 2013; Didi and Nitschke, 2016b). HyperNEAT's capability to evolve controllers that account for task geometry makes it an appropriate evolutionary method for deriving controllers that elicit behaviors robust to variations in state and action spaces (Risi and Stanley, 2013) as well as noisy, partially observable environments of multi-agent tasks. Also, it has been demonstrated that HyperNEAT evolved multi-agent policies can be effectively transferred to increasingly complex versions of keep-away soccer (Stone et al., 2006) without further adaptation (Verbancsics and Stanley, 2010) and that transferred behaviors often yield comparable task performance to specially designed learning algorithms (Stone et al., 2005).

3.2.1 HyperNEAT Behavior Transfer

For all HyperNEAT variants the entire evolved population was transferred from the source task (at the final generation) and set as the initial population for evolution in the target task. A discussion provided in section 3.1.1 supports the choice of this approach. It is based on previous work (Didi and Nitschke, 2016b) that indicates this method is most effective for HyperNEAT and various keep-away tasks. The behavior transfer approach applied for transferring the behaviors between the source and target tasks for HyperNEAT is the same as used for NEAT (presented in algorithm 1). Using *Bird Eye View* (Verbancsics and Stanley, 2010) representation, the substrate network encodes the collective behavior task environment state to directly reflect the task geometry. This way the CPPN evolves the solution as a direct function of the task geometry, exploiting the regularities in the task geometry.

The source task and a target task can be represented with the same substrate network on a two dimensional space, where changes in task configuration are depicted by the values assigned to each coordinate of the substrate network. The substrate geometric representation does not necessarily need to change with addition of new agents onto a simulation. Instead, to reflect a change in the world state (visual space), new agents are drawn onto the existing representation (substrate input layer) by adjusting the input values for the coordinates corresponding to locations of new agents in the visual space (for example, an input value changes from a 0 to either $+1$ or -1 to indicate the presence of a teammate or an opponent, respectively). Thus, the substrate network size or resolution (input and output layer configuration) remains the same across different tasks but each change in task configuration is accounted for on the existing representation through input values. Using this approach, the behavior transfer from a source to a target task can be performed without the need of a transfer mapping function.

3.3 Reinforcement Learning

Our method is compared to reinforcement learning (RL), a well established traditional method for adapting controllers for sequential and control tasks. Temporal difference (TD) methods (Sutton, 1998) are popular for learning behaviors in reinforcement learning tasks (Stone and Sutton, 2002; Stone et al., 2005). The TD methods have been chosen in the previous work for RL due to their ability to handle delayed reward, by constructing an internal reward signal that is less delayed than the original reward (Singh and Sutton, 1996).

The previous research has shown that collective behavior tasks, such as keep-away (section 4.4.1), are affected by the delay (in terms of efficiency) between an action and its effective reward using TD methods (Stone et al., 2005). For effective learning there has to be another mechanism used with TD methods to eliminate the effect of the delay.

Eligibility trace, introduced by Klopf (1972), has been widely used to support TD methods for handling delayed reward (Stone and Sutton, 2002; Stone et al., 2005). The eligibility trace keeps a temporary record of visited states and actions taken. Each time a state is visited, a short-term memory process, a *trace* (that marks a state as eligible for learning) is initiated which then decays gradually over time, controlled by a parameter λ. This way, only the eligible states or actions receives credit or penalty when a TD error occurs and thereby improving efficiency. SARSA(λ) and Q-Learning(λ), representing SARSA with eligibility trace and Q-Learning with eligibility trace, respectively are used in this thesis.

3.3.1 SARSA

SARSA is an acronym for State Action Reward State Action and is defined by a 5-tuple $\langle s_t, a_t, r, s_{t+1}, a_{t+1} \rangle$, where (s_t, a_t) represents a current state-action pair and (s_{t+1}, a_{t+1}) a subsequent state-action pair. The value r is the immediate reward from taking action a_t at state s_t. SARSA learns to estimate the action-value function, $Q(s, a)$, which computes the long term reward of performing action a in a state s. It is called an on-policy method because it estimates the action-value function that guides learning and simultaneously updates the policy with respect to the changing action-value function as shown in equation 3.4:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3.4}$$

where, α is the learning rate, γ is a discount factor that is used to scale the immediate reward with respect to future rewards.

SARSA(λ) a TD method combined with eligibility traces is used in this thesis (algorithm 3). The value of eligibility trace $e(s, a)$ is increased or replaced each time a state is visited, based on a type of eligibility configuration set. There are two configurations of eligibility traces. First, the *accumulative eligibility trace* that decreases for every step the agent takes after visiting that state. The value of λ, in the range $[0, 1]$ shown in algorithm 3, dictates the pace of declining towards zero. This is depicted with equation 3.5:

$$e(s) = \begin{cases} \gamma\lambda e_{t-1}(s), & if s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1, & \text{Otherwise} \end{cases} \tag{3.5}$$

Second, *replacing eligibility* traces that decay the eligibility trace value each time step when the state is not visited and replace the trace value with a value one otherwise. This is

depicted with equation 3.6:

$$e(s) = \begin{cases} \gamma\lambda e_{t-1}(s), & if s \neq s_t \\ 1, & \text{Otherwise} \end{cases} \tag{3.6}$$

The experiments in reinforcement learning adopted replacing eligibility trace as previous research indicated success of the later compared to the former (Sutton, 1998).

Algorithm 3 SARSA(λ) based on tile coding

Construct tiles based on the number of states and actions of the task
$\theta,\ \tilde{e} \leftarrow 0; \forall\, s, a$
repeat ▷ for each episode
  if *first step* **then** ▷ first time step
    foreach $a \in \mathcal{A}_a$ **do**
      $\mathcal{F}_a \leftarrow \{set\ of\ tiles\ for\ a, s\}$
      $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
    $LastAction \leftarrow \begin{cases} \underset{a}{argmax} Q(a), & P(1-\varepsilon) \\ \text{random action}, & P(\varepsilon) \end{cases}$
    $LastActionTime \leftarrow CurrentTime$
    foreach $i \in \mathcal{F}_a$ **do**
      $e(i) \leftarrow 1$
  else *step* $\neq$ *first step*
    $r \leftarrow CurrentTime - LastActionTime$
    $\delta \leftarrow r - Q(LastAction)$
    foreach $a \in \mathcal{A}_s$ **do**
      $\mathcal{F}_a \leftarrow \{set\ of\ tiles\ for\ a, s\}$
      $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
    $LastAction \leftarrow \begin{cases} \underset{a}{argmax} Q(a), & P(1-\varepsilon) \\ \text{random action}, & P(\varepsilon) \end{cases}$
    $LastActionTime \leftarrow CurrentTime$
    $\delta \leftarrow \delta + Q(LastAction)$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    $Q(LastAction) \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
    $\vec{e} \leftarrow \gamma\lambda\vec{e}$ ▷ Decay the eligibility trace
    foreach $a \in \mathcal{A}_s\ a \neq LastAction$ **do**
      foreach $i \in \mathcal{F}_a$ **do**
        $e(i) \leftarrow 0$
    foreach $i \in \mathcal{F}_{LastAction}$ **do**
      $e(i) \leftarrow 1$ ▷ replacing eligibility trace
until *episode*, *e is terminal*

3.3.2 Q-Learning

Q-Learning is a policy search method where the learned Q function directly approximates the optimal action-value function, independent of the policy that directs learning as in equation 3.7:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \underset{\pi}{max} Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3.7}$$

where $\underset{\pi}{max}$ represents the optimal value, and a greedy policy is used that always select the action corresponding to the optimal value.

Q(λ) a second TD method used in the RL experiments of this thesis. This version of Q-learning replaces eligibility traces and together with SARSA(λ) provides comparative benchmark algorithms to contrast with the neuro-evolution methods (section 3.1 and 3.2). The design of Q(λ) method is given in algorithm 4:

These two TD methods are applied to collective behavior tasks with very large and continuous state spaces, which this work explores. It is not possible for the agents to have a direct experience of all possible points in the state space (thus some states never recur). Rather, agents need to be exposed to a limited state space and then generalize the learning experience to other points in the wider state space, not yet visited. To be able to generalize that way, a table of Q-values must be approximated using some representation that limits the state space. This technique is called *function approximation* and there are many examples of function approximation that have been used in reinforcement learning (Watkins, 1989; Stone et al., 2005). The following subsection discusses, the function approximator applied for this work.

3.3.3 Function Approximation

Tile coding (1998) was adopted as a function approximation technique, where piecewise functions are used to approximate a value function. These functions are a discretization of the state space, creating partitions that maps to tilings. In this way the tiles and width of the tilings are specified prior to learning and this provides the mapping between state values and tiles. In line with previous work (Stone et al. 2005), given 13 state variables in a source task, 32 tilings per variable were selected, so for each step 416 integers would be stored in order to execute an update. The tiles encoding the current state in each tiling makes up a feature set, $\mathcal{F}_a$, with each action, a indexing the tilings. The number of possible tiles is large and relatively few are visited in practice.

A current state is represented by a single tile in each tiling, hence only one tile is active per tiling. Therefore, in each tiling there is only one tile for each variable. Given 13 state variables, there are $13 * 32 = 416$ tiles, and giving 416 tiles in each $\mathcal{F}_a$. The approximate value function, $Q(a)$, conventionally represented as a table, is then presented in parameterized form with parameter vector θ_t. $Q_a(s)$ is computed by summing the

Algorithm 4 Q(λ) based on tile coding

Construct tiles based on the number of states and actions of the task
$\theta,\ \tilde{e} \leftarrow 0; \forall\, s, a$
repeat ▷ for each episode
  if *first step* **then** ▷ first time step
    foreach $a \in \mathcal{A}_a$ **do**
      $\mathcal{F}_a \leftarrow \{set\ of\ tiles\ for\ a, s\}$
      $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
    $LastAction \leftarrow \underset{a}{argmax} Q(a)$
    $LastActionTime \leftarrow CurrentTime$
    foreach $i \in \mathcal{F}_a$ **do**
      $e(i) \leftarrow 1$
  else *step* $\neq$ *first step*
    $r \leftarrow CurrentTime - LastActionTime$
    $\delta \leftarrow r - Q(LastAction)$
    foreach $a \in \mathcal{A}_s$ **do**
      $\mathcal{F}_a \leftarrow \{set\ of\ tiles\ for\ a, s\}$
      $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
    $LastAction \leftarrow \underset{a}{argmax} Q(a)$
    $LastActionTime \leftarrow CurrentTime$
    $\delta \leftarrow \delta + Q(LastAction)$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    $Q(LastAction) \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
    $\vec{e} \leftarrow \gamma\lambda\vec{e}$ ▷ Decay the eligibility trace
    foreach $a \in \mathcal{A}_s\ a \neq LastAction$ **do**
      foreach $i \in \mathcal{F}_a$ **do**
        $e(i) \leftarrow 0$
    foreach $i \in \mathcal{F}_{LastAction}$ **do**
      $e(i) \leftarrow 1$ ▷ replacing eligibility trace
until *episode*, *e is terminal*

weights of all the tiles in $\mathcal{F}_a$, as in equation 3.8:

$$Q_a(s) = \theta^T \phi = \sum_{i=0}^{n} \theta(i)\phi(i) \tag{3.8}$$

Where, θ is a weight associated with each tile, i is the index of tiles in the feature set $\mathcal{F}_a$ and ϕ is the value indicating if the corresponding tile in $\mathcal{F}_a$ is active (for example, a value one shows its active and zero otherwise) and n the total number of tiles.

The algorithms 3 and 4 illustrates the application of the function of tile coding in approximating the value function and how it is applied in TD methods. The main difference between the two methods is how to select an action to execute from the set of available macro-actions. SARSA(λ) uses an $\epsilon-$greedy policy whereas Q(λ) takes the greedy policy, and follows the action corresponding to the highest $Q_a(s)$. The goal of selecting both of these methods is to enhance our study so that we could easily generalize the findings when we have carried out extensive tests. The following section presents the behavior transfer method implemented to transfer controllers between collective behavior tasks.

3.3.4 TD Behavior Transfer

To implement behavior transfer from a source to target collective behavior task for a TD method, there is need for mapping between states and actions in the two tasks. The choice of mapping function used in this thesis, is supported by empirical evidence from previous work by Taylor et al. (2005), which demonstrated specific domains and mappings functions, used successfully to increase the efficacy of RL learning. Using tile coding, the weights for activated tiles in target tasks can be initialized by assigning weights from the final episodes of the source task. In this work, we copy the weights learned in the source task into weights in the target task using the algorithm 5, derived from TVITM (Taylor et al., 2007b) discussed in section 2.2.1.

Algorithm 5 TD-BehaviorTransfer

Construct tiles based on the number of states and actions of the target task
foreach *non − zero weights* θ_i^s *in the source tiles* **do**
  $S_{source} \leftarrow \{value\ of\ state\ variable\ corresponding\ to\ tiles\}$
  $A_{source} \leftarrow \{action\ corresponding\ to\ i\}$
  foreach *value of* S_{target} *corresponding to value in* S_{source} **do**
    foreach *value of* A_{target} *corresponding to* A_{source} **do**
      $\theta_i^t \leftarrow \theta_i^s$
compute type relationships between S_{source} *and* S_{target} *states variables*
foreach *state value* j *in* S_{target} *not present in* S_{source} **do**
  $\theta_j^t \leftarrow mapped\ values\ of\ S_{source}$

In step 8, of algorithm 5 we test two configurations. First, initializing the rest of the remaining weights in the target pool to zero and secondly, weights from source are copied to the remaining target weights based on relationships. For example, weights for the tiles that correspond to *distance to teammate 2* state variable in the source are copied to initialize the weights of tiles corresponding to *distance to teammate 3* in the target state representation. Notably, HyperNEAT behavior transfer approach (section 3.2.1) has relatively less complex implementation compared to that of NEAT (section 3.1.1) and TD methods which are of similar complexity. As evolved controllers (behaviors) are transferred using HyperNEAT without alteration compared to a hand coded approach using NEAT and TD methods.

The subsection that follows provides the discussion of five variants of the behavior adaption techniques used with NEAT and HyperNEAT methods. These techniques will be investigated in this thesis to ascertain the method that best adapts behaviors between the source and target collective behavior task (that is, significantly improved task performance).

3.4 Behavior Adaptation Methods

This section presents a description of five different types of evolutionary search methods used in this thesis. Each of these approaches were used to direct the search process in neuro-evolution methods (both NEAT and HyperNEAT). These approaches are objective-based search, behavior diversity (novelty) search, genotypic diversity search, hybrid behavior diversity and objective-based search and hybrid genotypic diversity and objective-based search.

3.4.1 Objective Based Search (OS)

OS is a conventional method for behavior adaptation in neuro-evolution guided by an objective function which evaluates the performance of each individual against a fitness metric. In this thesis, experiments are conducted on simulated keep-away soccer domain, and the objective function computes mean episodic length using equation 3.9:

$$fit_x = \frac{1}{N}\sum_{j=1}^{N} T_j \tag{3.9}$$

Where, the length of an episode j is T_j, N is the number of task trials (simulation length), and T_j is the length of trial j. In this method the objective is to maximize the mean episodic length and the search process is directly guided by this performance metric. OS is the first of the search variants used together with NEAT and HyperNEAT evolutionary adaptation.

3.4.2 Behavioral Diversity and Objective-based Search

Many NE methods have incorporated behavioral diversity maintenance into their search processes as a means of discovering novel and higher quality solutions, compared to the same methods using objective based search (Mouret and Doncieux, 2012).

Novelty Search (NS)

Novelty search (Lehman and Stanley, 2011a) is a search process that rewards evolved behaviors based on their novelty. Thus, a genotype is more likely to be selected for reproduction if its encoded behavior is sufficiently different from all other behaviors produced thus far in artificial evolution. NS has been demonstrated as yielding solutions that out-perform objective based search in various multi-agent tasks (Gomes et al., 2013; Gomes et al., 2016b). Given this, NS was selected as the behavioral diversity mechanism for controller evolution in this thesis.

The function of NS is to consistently generate novel team (keep-away) behaviors. Hence, we define team behavior in terms that potentially influence team behavior but are not directly used for task performance evaluation. That is, we use the behavioral properties: *average number of passes*, *dispersion of team members*, and *distance of the ball to the center of the field*. To measure novelty we normalize each of these properties as task specific behavioral vectors, where the addition of these vectors is always in the range: $[0, 1]$. This team level behavioral characterization has been used previously (Gomes et al., 2014a) and out-performs individual behavioral characterizations. Behavioral distance is computed using equation 3.10:

$$\delta_i(x, y) = \|x_i - y_{ij}\| \tag{3.10}$$

Where, $x_i$and y_{ij} are normalized behavioral characterization vectors of two genotypes. Novelty is then quantified by equation 3.11, which replaces the fitness function of NEAT and HyperNEAT.

$$nov_x = \frac{1}{3k} \sum_{i=1}^{k} \sum_{j=1}^{3} \delta(x_j, y_{ij}) \tag{3.11}$$

Where, x_j is the j^{th} behavioral property of genotype x, y_{ij} is the j^{th} behavioral property of the i^{th} nearest neighbor of genotype x and δ is the behavioral distance between two genotypes x and y computed in equation 3.10. The nov_x then is derived from the mean behavioral distance of an individual with k nearest neighbors. The parameter k (number of nearest neighbors) is user specified. Related work used $k = 20$ (Liapis et al., 2015) and $k = [3, 10]$ (Gomes et al., 2015) with varying results. Gomes et al. (2015) found k values

are highly dependent on the type of novelty archive, where $k = 15$ yielded relatively good performance across all tested archives. Hence this study uses $k = 15$.

As in related work (Lehman and Stanley, 2011a), the novelty of newly generated genotypes is calculated with respect to previously novel behaviors stored in the novelty archive, where archived behaviors are ranked by diversity. The maximum archive size is 1000 and the number of behaviors added to the archive after each generation is limited to 10 (given results of related work (Gomes et al., 2015; Meyerson et al., 2016)).

Hybrid Objective-Novelty Search (ONS)

In line with previous hybrid NS research (Gomes et al., 2014a), we use a metric that linearly combines NS with the objective-based search of NEAT and HyperNEAT (equation 3.12):

$$score_i = \rho \cdot \overline{fit_i} + (1 - \rho) \cdot \overline{nov_i} \tag{3.12}$$

Where, $\overline{fit_i}$ and $\overline{nov_i}$ are normalized fitness and novelty of i^{th} genotype respectively, $\rho \in [0, 1]$ is user selected to control the relative contribution of each metric to selection pressure. Previous work demonstrated that a medium to high novelty weight 50-80% yields the best results (Gomes and Christensen, 2013b). We found that a novelty weight of 40% yielded the best results in this case study. All other novelty search parameters are the same as used for the NS variant (subsection 3.4.2).

3.4.3 Genotype Diversity and Objective Based

Genotype Novel Search (GNS)

Similar to novelty search (section 3.4.2), GNS is the genotype diversity search for novel genotypes (controller encodings). The genotypic distance between two genotypes is measured using linear combination of *Excess* (E) and *Disjoint* (D) genes (Stanley and Miikkulainen, 2002), and a mean weight difference of matching genes $\overline{W}$ (Risi et al., 2010) (equation 3.13). Genes that do not match are either *disjoint* or *excess* depending on whether they occur within or outside the range of parent innovation numbers (Stanley and Miikkulainen, 2002).

$$\delta_g(a, b) = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W} \tag{3.13}$$

Where, N is the number of genes in the longest genotype of the population, and coefficients c_1, c_2 and c_3 are parameters used to adjust the weighting of the three factors

E, D and $\overline{W}$ respectively. The sparseness (S_g) of genotype x in population evolution is computed by equation 3.14.

$$S_g(x) = \frac{1}{k}\sum_{i=1}^{k}\delta_g(x, y_i) \tag{3.14}$$

Where, y_i is the i^{th} nearest neighbor of x, k is the number of nearest neighbors of x and δ_g is the compatibility distance measure (equation 3.13). The generation n exploration metric is then the population's mean sparseness (equation 3.15):

$$E_g(n) = \frac{1}{N}\sum_{x=1}^{N} S_g(x) \tag{3.15}$$

Where, N is the population size parameter and $S_g(x)$ is the sparseness of individual x in generation n computed in equation 3.14. If exploration measure $E_g(n)$ is high it means the population has genetically diverse genotypes. The *GNS* variant thus uses equation 3.14 in place of NEAT or HyperNEAT's fitness function, as genetically diverse genotypes are selected. The same nearest neighbor and archive parameters are used for *GNS* as used for the behavior novelty search *(NS)* variant (subsection 3.4.2).

Hybrid Objective-GNS (OGN)

OGN uses equation 3.12, except that $\overline{nov_i}$ now represents the genotype diversity metric (equation 3.14). That is, equation 3.14 specifying the genotype sparseness in the population (normalized into the range $[0, 1]$) replaces the normalized novelty function value $\overline{nov_i}$ in equation 3.12. Similarly, $\rho \in [0, 1]$ controls the relative contribution of fitness versus genotypic diversity directed search. Previous work by Didi and Nitschke (2016), indicated that a genotypic diversity weight of 40% was appropriate for the experiments in this thesis (discussed in chapter 4). The rest of other parameters are the same as used for the GNS variant (subsection 3.4.3).

3.5 Discussion and Summary

This chapter discussed novel methods that combined existing NE methods (NEAT and HyperNEAT) with various search approaches, integrating them with methods for collective behavior transfer between tasks of increasing complexity. The behavior transfer method for each technique was discussed and adopted to transfer behaviors between tasks. This thesis utilizes two neuro-evolution (that is, NEAT and HyperNEAT) methods with five variants of each (that is, five adaptation techniques applicable to each method). These methods were both direct and indirect encoding NE methods and

essential for ascertaining the best behavior adaptation method that yields best performance benefits when transferred to tasks with increasing complexity.

This chapter also discussed RL methods that will be used as a task-performance benchmark against which the neuro-evolution transfer learning methods will be compared. These RL methods included SARSA and Q-Learning (TD methods). For TD methods tile coding was adopted as a function approximation technique to discretize the state representation, as the behavioral search space of keep-away increases exponentially with task complexity.

Chapter 4

Behavior Transfer and Neuro-Evolution Experiments

This chapter evaluates the performance and appropriateness of the neuro-evolution methods for collective behavior evolution and transfer of behaviors between collective behavior tasks of increasing complexity. This chapter discusses experiments that compares five variants of NEAT and HyperNEAT (table 4.1) in a keep-away soccer domain (Taylor et al., 2007b). In particular, to address the thesis research question (discussed in section 1.2), the experiments investigate what is the best NE method for collective behavior evolution to use in company with policy transfer and thus boost task performance and efficiency. Specifically, there are three main features of the methods being tested in this case study, to ascertain which combination yields the best results (in terms of task performance and efficiency). These methodological features are:

1. Direct (NEAT) versus indirect (HyperNEAT) encoding
2. Evolutionary search variant to direct NE (behavioral versus genotypic diversity as well as a hybrid approach)
3. Use of policy transfer versus no policy transfer

First, we test and evaluate the efficacy of NEAT and HyperNEAT (that is, direct and indirect encoding NE methods, respectively) as appropriate methods for yielding task performance and efficiency boosts after policy transfer. Second, we test and evaluate the impact of using a non-objective (that is, behavior diversity and genotype diversity) versus objective (fitness) based search approach for two given policy search methods (NEAT and HyperNEAT). Third, we compare results of evolution of controllers with and without behavior transfer across a range of keep-away tasks.

In this thesis the appropriateness of the methods is measured in terms of effectiveness, solution complexity and efficiency of collective behavior across task of increasing complexity. Effectiveness is improved average task performance between tasks with and without behavior transfer. This measure indicates performance gained through the use of behavior transfer. Efficiency refers to average number of generations to reach a task performance threshold, a comparison of collective behavior adaptation before and after

Variant Name	Variant Description
OS	Objective-based Search
NS	Novelty Search
ONS	Hybrid Novelty-Objective based Search
GNS	Genotypic Novelty Search
ONS	Hybrid Genotypic Novelty-Objective based Search

TABLE 4.1: The five variants (NEAT and HyperNEAT) evaluated for collective behavior adaptation and transfer across keep-away tasks of increasing complexity

behavior transfer. The average maximum task performance attained by each method, in each task, without behavior transfer is the performance threshold that will be used as a benchmark. Then the average number of generations to attain this threshold is used as an indicator of efficiency. The third quality variable measured in these experiments is topological complexity (that is, sum of nodes and connections in the topological structure) of the solution. A method that evolves simple (minimal topological complexity) solutions that efficient and effective compared to a method that evolves complex solutions that are relatively ineffective and inefficient would be preferable.

Effectiveness and efficiency is also used to measure the efficacy of five variants of NEAT and HyperNEAT for collective behavior adaptation and transfer across keep-away tasks of increasing complexity. Table 4.1 shows the five variants which differ in the way they guide evolutionary search.

4.1 Collective behavior Task and Performance Specification

Keep-away[1] is a domain introduced by Stone and Sutton (2002) and is a subtask of *RoboCup soccer*[2] (Noda et al., 1998), that was developed as a benchmark task for complex multi-agent learning. In keep-away, one team, the *keepers*, attempts through learning to maintain possession of the ball as long as possible in a fixed bounded region (20×20 square grid), while an opponent team, the *takers* attempts to gain control of the ball. In the keep-away simulator, an episode is started by setting the position of the ball, keeper and taker agents in a visual space. Three keeper agents are selected at random, and placed close to the three corners of the field and other agents are placed at random positions close to the center of the field. The taker agents are spaced and located close to the fourth left bottom corner of the field. The ball is placed in possession of the keeper located at the top left corner of the field. The episode runs for the duration equivalent to the number of cycles the keeper team is able to maintain possession of the ball and episode ends either when the ball goes out of bounds or the taker agent gains control of the ball. When the episode ends the players positions are reset for another episode and ball position is given to the keeper's team.

[1] All experiments were run in *RoboCup keep-away version 0.6*
[2] https://sourceforge.net/projects/sserver/

The keeper agents learn to make high-level decisions such as to pass the ball or hold the ball. The low-level actions are handled by the simulation framework, that allows players to execute low level instructions such as dash to the ball, kick the ball or turn. Some high-level decisions such as *getOpen* (Stone et al., 2005), to move to a space free of opponent players so as to receive a pass, are handled by a heuristic method provided by the keep-away framework. Actions are selected for execution every single simulation cycle that is equivalent to 100 ms. In each simulation cycle, a learning agent receives sensory inputs and acts asynchronously. Keep-away simulator introduces some random noise to the sensors and actions to make the environment realistic. Each agent learns independently (that is, there is distributed control of agent behavior) and there is no communication between agents. The takers follow a heuristic approach exhibited by algorithm 6, that selects two agents arbitrarily so as to estimate the next location of the ball and dash to that position, while the rest of the agents, estimate open positions between the opponents and block passes.

Algorithm 6 Taker agent Behavior

```
1: Initialize position of agents, assigns id's
2: read taker agent's Id
3: repeat
4:   foreach time_step ∈ episode_duration do
5:     if agent_Id <= 2 then
6:       next_position ← predict(next_ball_position)
7:       policy ← moveTo(next_ball_position)
8:     else agent_Id > 2
9:       next_position ← predict(most_Open_Space)
10:      policy ← moveTo(most_Open_Space) + Intercept(Ball)
11: until Terminal_State
```

The objective of the keeper team in the keep-away task is to maximize the expected length of the episode (that is, *episodic hold time*). Episodic hold time is then used as a performance measure of the keep-away task. Since a keep-away can be played effectively with homogeneous teams (Whiteson et al., 2003), in this thesis neuro-evolution evolves teams of homogeneous controllers, with shared fitness between players. The fitness function that measures the task performance is given in equation 3.9 in section 3.4.1. Task performance is measured over 30 trials and averaged over 20 simulation runs for five variants of NEAT and HyperNEAT (see table 4.1). The role of NEAT and HyperNEAT methods in this task, is to synthesis controllers for the keepers team so as to maximize the episodic hold time.

4.1.1 Complexity in keep-away task

Empirical evidence from previous research by Stone et al. (2005) and Taylor et al. (2005) indicated that increasing number of agents in the keep-away task, leads to an increase

in the task complexity, regardless of which agent is added (that is, keeper or a taker agent) to the simulation. Increasing the number of takers increases the likelihood of the ball interception because of two major reasons: first, as more players will be available to block passes, second, the angle of passes will be small. On the other hand, an increase in keeper agents in a fixed bounded region, results in a crowded field and that potentially increases the complexity of the task. More keeper agents in the field, means more passing options but leads to reduction in the average pass distance. This leads to more potential errors and interference between the players. Therefore, task complexity equates to the ratio of taker to keeper agent plus the total number of agents.

The keep-away players sensors and actuators are noisy, players have a partial view of the environment and the simulation environment is highly stochastic. The keep-away soccer adds evenly distributed probabilistic noise to all objects movements, where noise is a random number whose distribution is uniform that is added to the parameters of a moving object such as velocity and turning angle of a moving player. Noise is mainly added in activation parameters, objects motion and visual perceptions (Stone, 2000). Due to noisy sensors and actuators and as well as the hidden state enforced, the agents only have a partial world view at any given time. These listed properties adds to the complexity of the keep-away task. Also leads to noisy fitness functions and for that reason this task is considered to be deceptive (Gomes et al., 2013), though not perversely deceptive as in deceptive maze navigation (Lehman et al., 2013).

Objective-based search (OS), a popular evolutionary search method, selects behaviors that produce highest average hold time (that is, time in possession of the ball). This method does not consider and exploit the behavioral properties that influence that behavioral outcome. Hence, even though the action of moving with the ball (that is, hold the ball) maintains ball possession, the field position of this keeper can be a disadvantage if the keeper is surrounded by opponent team players. Thus, such behaviors that satisfy the fitness objective are deceptive in that evolutionary stepping stones (such as, evaluating the position of players before taking an action) are not rewarded and not selected. Behaviors that could likely benefit the team such as maximizing the number of passes, and limiting the passing distance, ideally need to be identified and exploited as they contribute to task performance.

Having more players in the task, such as *6vs5* keep-away task is considered to be the most complex of the five tasks, with many opponent players blocking the passing lane and advancing towards the ball from many directions increases the likelihood of making errors. Therefore, this particular task is classified as the most appropriate to train with behavior transfer to jump-start the evolutionary process.

Furthermore, previous research has indicated that as the task complexity increases, it becomes more difficult to design an appropriate fitness function and objective-based evolution becomes more vulnerable to deception (Zaera et al., 1996; Gomes et al., 2013). In support of that argument, Whitley et al. (1991) suggested that high level of

complexity has tendency to generate deceptive fitness landscapes. The deception then becomes a limiting factor for success. Comparing the performance of novelty search and objective-based search on deceptive domains, Lehman and Miikkulainen (2013) demonstrated that novelty search is capable to produce better solutions than objective-based search (that is, in specific domains deemed to be highly deceptive). However, it is difficult to quantify and establish deception in keep-away tasks, thus we consider keep-away soccer to be non-deceptive domain.

In summary, introducing more agents to the field and noise in sensors and actuators, relates to an increase in task complexity. This slows down evolutionary progress in keep-away tasks of increasing complexity and leads to poor overall task performance. In this thesis, five NE variants are evaluated on keep-away tasks with increasing complexity to ascertain the appropriate method for adapting collective behavior controllers.

4.2 NEAT for Collective Behavior Evolution

NEAT (Stanley and Miikkulainen, 2002) is a direct encoding NE method, used in this case for behavior evolution, evolves ANN to control the collective behaviors of the keep-away task. Figure 4.1 shows an example of an ANN evolved by NEAT for a $3vs2$ keep-away task configuration, that has 13 sensory inputs, a bias node and three motor controllers. The description of each input and the output node is given in table 4.2. Mapping between the genotype and the phenotype representation for NEAT is one-to-one. Generally, the encoding of sensory-motor of a keep-away team controller needs to change as task complexity increases. The input-output configurations for each keep-away task is illustrated in table 4.3, for example a $3vs2$ keep-away task has 13 input and 3 output nodes.

As task complexity increases, from *3vs2* to *4vs3 keep-away*, an ANN topology with 19 input nodes and 4 output nodes is required. The additional output node represents the decision of *keeper 1* to *pass to keeper 4*. The extra six input nodes represent:

1. distance of *keeper 4* from the field's center,
2. distance of *taker 3* from the field's center,
3. distance of *keeper 1* from *taker 3*,
4. distance between *keeper 4* and the closest taker,
5. angle formed between *keeper 1* and the closest keeper and taker,
6. distance of *keeper 1* to *keeper 4*.

Similarly, for the *5vs3 task* an ANN with 23 inputs and five outputs is needed.

However, for all keep-away tasks tested in this chapter (*3vs2*, *4vs3*, *5vs3*, *5vs4*, *6vs4* and *6vs5*) the ANN sensory-motor layer topology was kept static (13 sensory inputs and

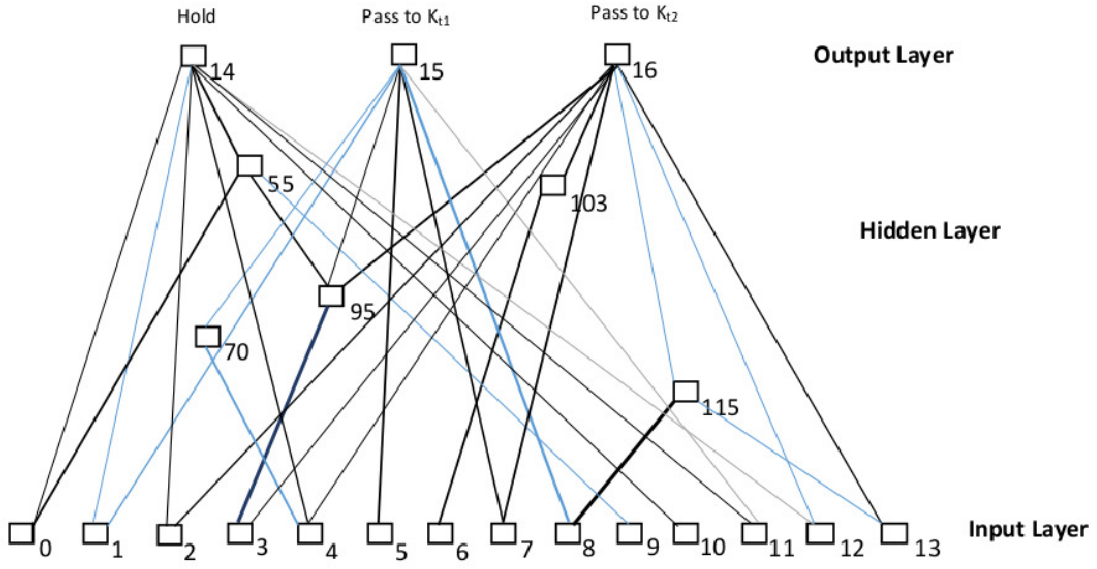


FIGURE 4.1: Example of network evolved with NEAT for a $3vs2$ keep-away task. The network has thirteen sensory inputs and three outputs and a bias node. NEAT evolves the hidden layer topology and connectivity. The network configuration parameters are described in table 4.2.

Sensory Inputs	**Description**
dist(K_b, C), dist(K_{t1}, C), dist(K_{t2}, C)	Distance of each keeper to field center
dist$(T1, C)$, dist$(T2, C)$	Distance of each taker to field center
dist(K_b, K_{t1}), dist(K_b, K_{t2})	Distance of each teammate to keeper 1
dist$(K_b, T1)$, dist$(K_b, T2)$	Distance of each taker to keeper 1
$min_{j\in1,2}dist(K_{t1}, T_j)$, $min_{j\in1,2}dist(K_{t2}, T_j)$	Distance of closest taker to keeper 1
$min_{j\in1,2}angle(K_{t1}, T_j)$, $min_{j\in1,2}angle(K_{t1}, T_j)$	Angle of closest keeper, taker, keeper 1
Motor Outputs	
Hold Pass to K_{t1}, Pass to K_{t2}	Do not pass ball Pass to keeper 2, keeper 3

TABLE 4.2: Sensory inputs (13 input nodes) and motor outputs (three outputs) for a team's ANN controller in the *3vs2 keep-away task*. Keeper 1 is the agent with the ball.

three motor outputs) in order to facilitate behavior transfer across tasks of increasing complexity. In the RoboCup soccer simulator, due to noisy sensors each player can see objects within $90°$ field of view and sensory vision precision deteriorates with distance. In the simplified keep-away versions, players are given $360°$ field view and location precision fades with distance (Stone and Sutton, 2002). In our method, players are prioritized by sensory stimulations based on distance and strength of sensory vision. Thus, as the number of agents increased with task complexity, a heuristic selected which agents in the environment would be processed by the ANN's 13 sensory input nodes. At each sensory-motor cycle (task trial iteration), the heuristic selected the closest two keeper and taker agents to be processed by the ANN, but had the potential to process any agent as sensory input.

Task Configuration	ANN Inputs/Outputs
$3vs2$	$13/3$
$4vs3$	$19/4$
$5vs3$	$23/5$
$5vs4$	$25/5$
$6vs4$	$29/6$
$6vs5$	$31/6$

TABLE 4.3: The number of sensory inputs and motor outputs for ANN keeper team controllers applied each keep-away task. For this work a topology of thirteen inputs and three outputs, is always used to facilitate policy transfer. A heuristic method is used to select players that will participate in the configuration for each task.

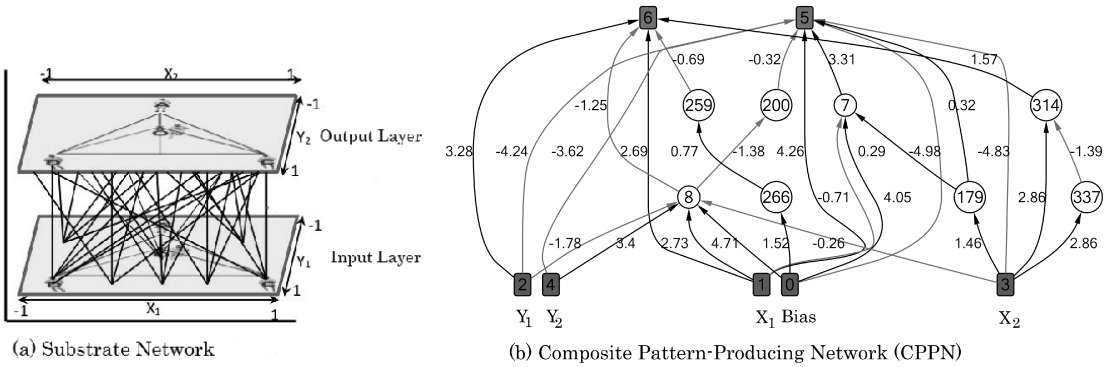


FIGURE 4.2: Example of network evolved with HyperNEAT. **Left:** Substrate encoding the virtual field (20 x 20 grid of inputs and outputs). **Right:** CPPN takes as inputs coordinates of two-endpoints of a connection on a substrate network and gives weight of that connection and a connection expression value as output.

4.3 HyperNEAT for Collective Behavior Evolution

HyperNEAT (Stanley et al., 2009) uses indirect encoding and can thus represent changes in task complexity without changing genotype representation (Verbancsics and Stanley, 2010). Verbansics and Stanley (2010) introduced *Bird's Eye View* (BEV) representation, which is used in this thesis to encode keep-away's visual state (layout of the field and locality of agents) and actions onto a substrate network. The virtual space for a keep-away soccer simulation field is represented by a 20 x 20 grid space, where each agent is modeled to occupy a single grid cell per simulation. HyperNEAT encoding has two networks, one that encodes the field properties (that is, substrate) and the other one that encodes the regularity of the field properties as composition of functions (that is, CPPN).

The substrate network is multi-layered comprising of input and output layers with each layer being a two-dimensional space, with coordinates in the x, y plane in the range of $[-1.0, 1.0]$. Each grid cell in the keep-away soccer simulation virtual field is represented by a node in each substrate network layer. Then a 20 x 20 grid input space (and output space) is represented by 400 nodes in the substrate network layer. A 400×400

input-output space yields $160,000$ possible connections (direct connections in our method) in the substrate network. Therefore, the substrate network geometry (that is, the layout of nodes in the substrate network) directly maps to the tasks geometry and this enables HyperNEAT to exploit the task's geometric regularities and relationships (Stanley et al., 2009). The position of each agent is marked on the substrate input layer, where each position of the keeper and taker are marked with 1.0, -1.0, respectively. The cells following in direct paths between agents are also marked, each cell in the path from a keeper with the ball to a teammate, is marked with a value 0.3. A cell in the path to an opponent agent (that is, taker) is marked with a value -0.3 and other paths (without agents) are marked with a value 0. On the substrate output layer, the position to pass the ball to, is indicated by activating the node with the highest output. If the hold position action has been selected, the position where the ball is will be activated and the ball remains in that position.

The synaptic weights of the connections between the input and output layer of the substrate network are computed by the CPPN. Each potential connection in the substrate network is queried and the coordinates of the two end points ($(x1, y1)$ and $(x2, y2)$) applied as input to the CPPN. NEAT evolves the CPPN network and the two outputs represents the weight of the queried connection and a link-expression output (LEO) value. The LEO value indicates if the connection should be expressed or discounted from the network. In this way, the connection weights are then produced as a function of their endpoints. The functions used in this thesis work are listed in table 4.4 (right), where each function represents a particular type of regularity in the domain space. For example, a Sine function represents repetition (that is, repeating motifs) and Gaussian function reflects symmetric motifs. Since inputs into the CPPN are locations in the substrate network (that in keep-away, reflect the field), functions compute elements of locality on the substrate plane.

4.4 NEAT and HyperNEAT Experiments Setup

Experiments are run in a source keep-away task, where NEAT evolves a population of 150 genotypes (from *scratch*) for 30 generations. The evolved population, in its entirety, is then transferred to a target task, and further evolved for 100 generations (table 4.4). The choice of parameters is based on the previous successful empirical evidence (Verbancsics and Stanley, 2010) and preliminary empirical tests conducted (Didi and Nitschke, 2016b; Didi and Nitschke, 2016a). Since keep-away task has noisy sensors, a way of managing noisy fitness evaluations in neuro-evolution is desired. Beyer (2000) suggested three techniques for coping with noisy fitness evaluations in evolutionary algorithms, which are: increasing the candidate solution population size, increasing evaluation sample size and taking average fitness to direct evolution. Following that suggestion and previous successful work by others (Verbancsics and Stanley, 2010; Didi and Nitschke, 2016b), in this thesis all candidates solutions are re-evaluated and fitness

NE Parameters	**Setting**
Population Size	150
Generations (Source task)	30
Generations (Target task)	100
Maximum number of species	10
Maximum species population	30
Mutation type	Gaussian
Weight value range	[-5.0, 5.0]
Mutation rate	0.05
Survival threshold	0.2
NS / GNS Parameters	**Setting**
NS nearest neighbor k	15
Maximum archive size	1000
Compatibility threshold	3
Behavioral threshold	0.03

HyperNEAT CPPN	**Functions**
Identity	x
Gaussian	$e^{-2.5x^2}$
Bipolar Sigmoid	$\frac{2}{1+e^{-4.9x}} - 1$
Absolute value	$\|x\|$
Sine	sine(x)
Simulation Parameters	**Setting**
Number of Runs	20
Iterations per task trial	4500
Trials per generation	30
Agent positions	Random
Environment size	20x20 grid
Agent speed (per iteration)	1 grid cell
Ball speed (per iteration)	2 grid cells

TABLE 4.4: **Left:** *Neuro-Evolution* (NE), *Novelty Search* (NS) parameters (final three rows). **Right:** CPPN (HyperNEAT) activation Functions and simulation parameters.

is sampled 30 times. An average fitness is computed to constitute a task performance metric. The performance of each variant of neuro-evolution method is computed from 20 independent simulation runs. This is to ensure the robustness of the results and to eliminate the elements of chance in performance gains or deficiency. The obtained results are compared to those where no policy transfer takes place, that is where NEAT is used to evolve keep-away behaviors from *scratch* in the target tasks. A population evolved from scratch starts off with a population of randomly initialized weights and a bias.

Evolution runs for 100 generations in instances where the population was evolved from scratch and after behavior transfer to a target task. This is to ascertain the performance gain realized through transfer of evolved behaviors. For both NEAT and HyperNEAT experiments, each genotype is evaluated over 30 task trials per generation, where each task trial tests different randomly determined agent positions. Consequently, this eliminates the possibility of erratic performance due to arbitrary selection of starting positions. Table 4.4 specifies the neuro-evolution and simulation parameters for these experiments.

4.4.1 Behavior Transfer Experiments

Collective behavior transfer was applied between the keep-away task configuration *3vs2* (*source* task) and one of five more complex target tasks (that is, task configurations *4vs3*, *5vs3*, *5vs4*, *6vs4*, *6vs5*). The keeper team behavior is evolved in the source task for 30 generations. These evolved controllers are then transferred to each target task and further evolved for 100 generations (table 4.4). This number of generations was selected

because of the previous study in Keep-away task, where 57 generations corresponded to between 800 and 1000 hours of training time in standard keep-away simulation (Taylor et al., 2006a). The behaviors in both source and target tasks are evolved with one of the five NE variants shown in table 4.1. The entire evolved population at the end of evolution in the source task, is transferred to be the initial population of the target task.

The efficacy of behavior transfer is evaluated in terms of evolution time (genotype evaluations) taken to attain a policy transfer threshold, with and without policy transfer. The threshold was the *average maximum fitness* attained after applying NEAT and HyperNEAT to evolve behaviors *from scratch* in each target task. The performance evaluation between different NE variants is measured in terms of task performance (solution fitness) and efficiency (time to reach task performance threshold).

4.5 Results and Discussion

Mann-Whitney u statistical tests (p<0.05) (Flannery et al., 1986) were applied in pair-wise comparisons between average task performance, efficiency and complexity yielded by NE variants to ascertain if there was statistical significance between different results (Appendix B and C). Furthermore, we applied pair-wise practical significance *t-test*, with *effect size* (Cohen, 1988) treatment, between task performance and efficiency results (Appendix D).

Behavior transfer is applied between the source *3vs2* keep-away task and one of the five more complex target tasks (that is, *4vs3*, *5vs3*, *5vs4*, *6vs4* and *6vs5*) for each of the five NE variants. Keep-away behavior is evolved in the source task for 30 generations and then transferred to a target task where behavior is further evolved for 100 generations (see table 4.4). Entire evolved populations are transferred from the source task and used as the initial population for evolution in the target task.

Figure 4.3 and 4.4 presents a plotting of normalized average task performance for NEAT and HyperNEAT side-by-side showing results for the five NE variants (OS, NS, ONS, GNS, OGN). The results are obtained from evolution with and without transfer of evolved behaviors from the source task, averaged over 30 trials and taking an average of maximum task performance from 20 independent runs. The fitness values are normalized to $[0, 1]$ using equation 4.1:

$$\overline{fit_x} = \frac{fit_x - fit_{min}}{fit_{max} - fit_{min}} \tag{4.1}$$

where a value of fit_{max} is the highest fitness score and fit_{min} is the lowest fitness score and must be set by the experimenter.

HyperNEAT outperforms NEAT in almost all task configurations as shown on table A.1 (appendix A). This is statistically significant ($p < 0.05$, Mann-Whitney test) and indicates the appropriateness of HyperNEAT for searching for optimal behaviors in collective

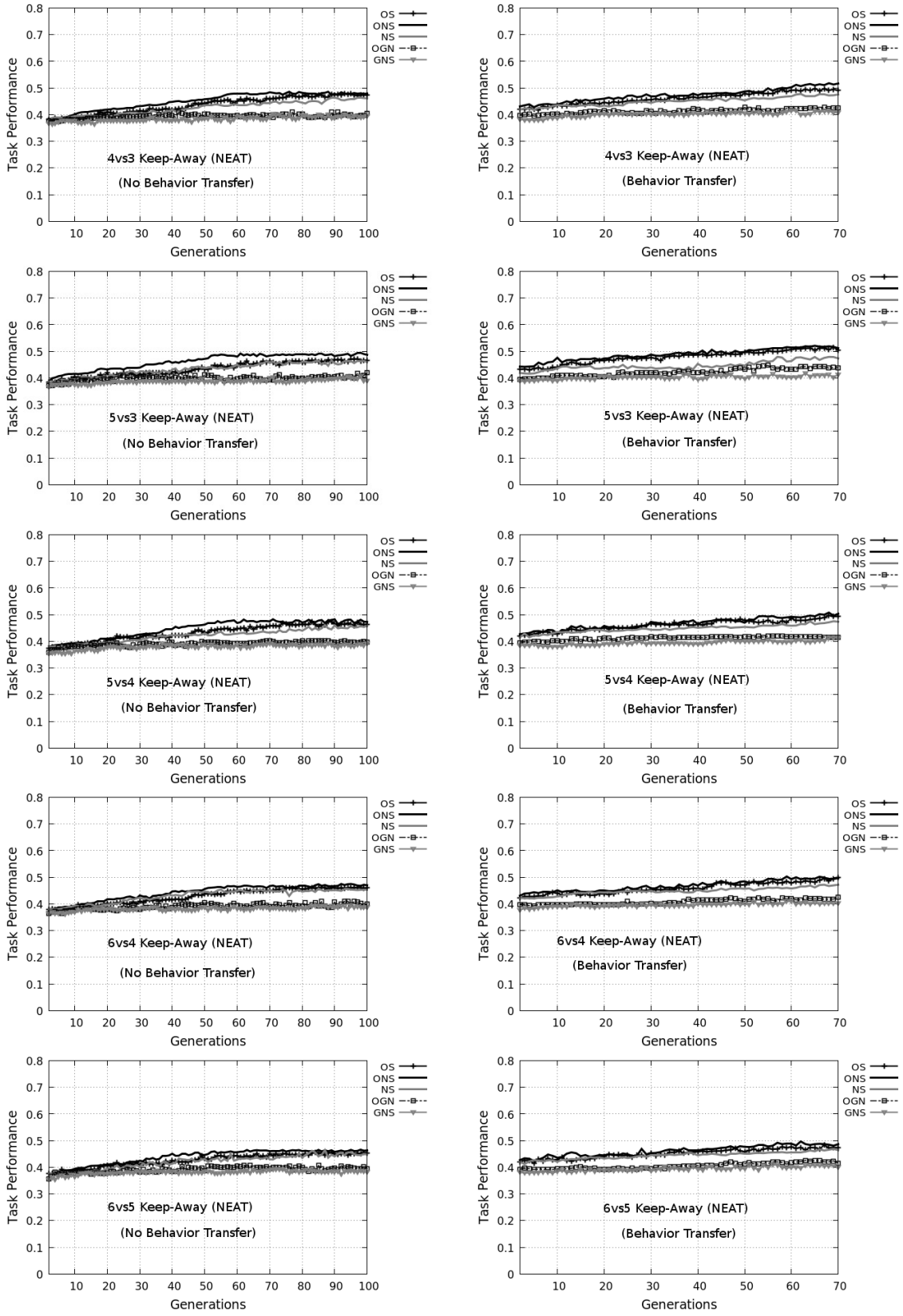


FIGURE 4.3: Task performance progression graph. The graph shows progression of mean of normalized maximum task performance for all variants of NEAT with and without behavior transfer. Averages are calculated over 20 runs and for each target keep-away task. Shown are 6vs4 and 6vs5 keep-away task performance.

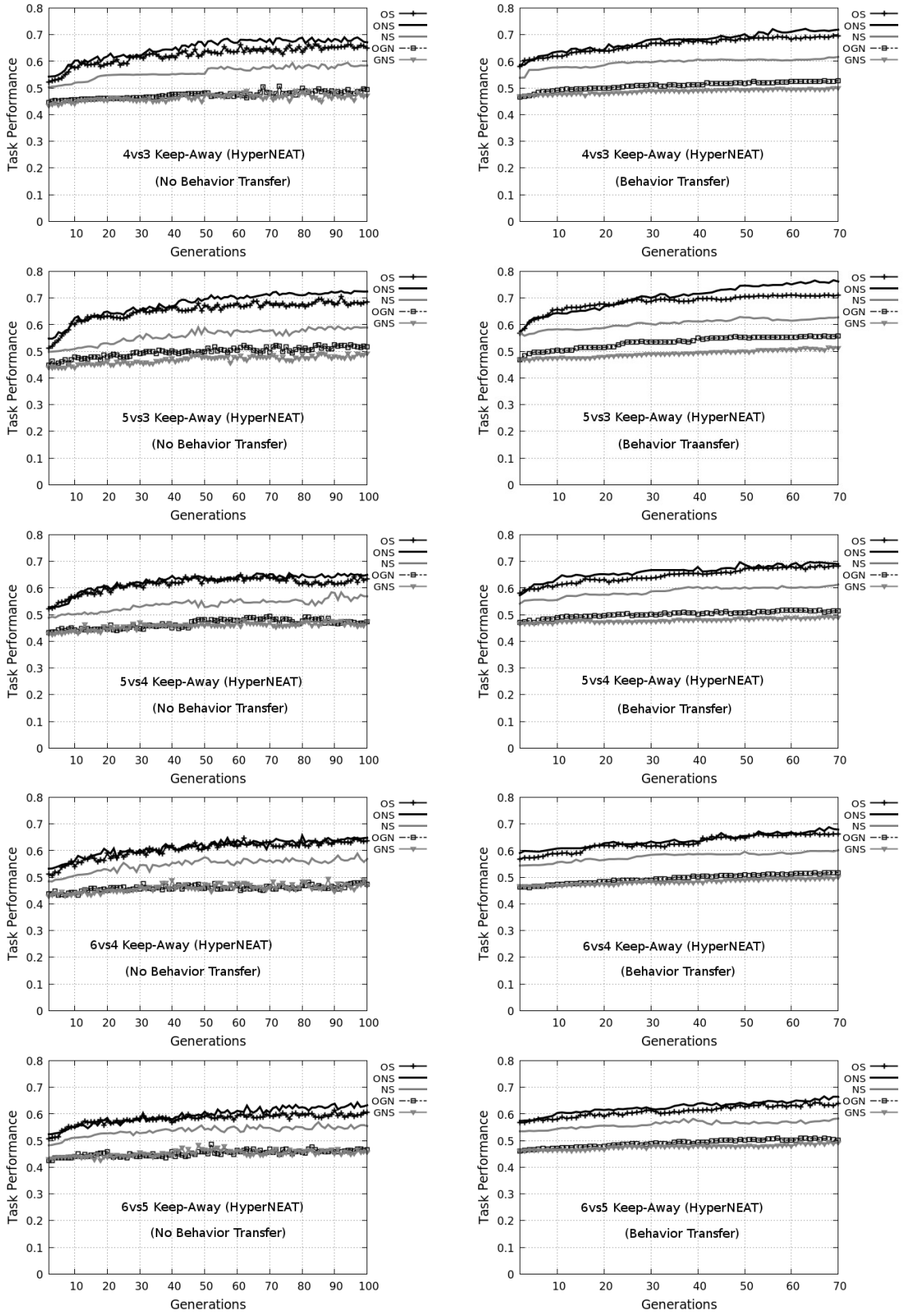


FIGURE 4.4: Task performance progression graph. The graph shows progression of mean of normalized maximum task performance for all variants of HyperNEAT with and without behavior transfer. Averages are calculated over 20 runs and for each target keep-away task. Shown are 6vs4 and 6vs5 keep-away task performance.

behavior tasks. The highest value recorded with a particular case of HyperNEAT is 0.748 with a standard deviation of 0.038, which is obtained with behavior transfer on ONS variant. This value equates to episodic hold time of $0.748 \times 18 = 13.5$ seconds compared to $0.705 \times 18 = 12.6$ seconds obtained without behavior transfer.

A relatively poor performance was recorded for GNS variant of NEAT (0.381 ± 0.040), which had the lowest task performance of all NE variants (with statistical significance). The task performance decreases with the number of agents in the task, with the exception of 5vs3 where the task performance is comparable to that yielded for 4vs3 or slightly high in most of NE variants. Comparing between NE variants in each task, it is important to note that ONS, has the highest task performance, followed by OS, NS, OGN and GNS in that order. ONS consistently obtained the highest task performance in all keep-away tasks, with a mean normalized task performance of 0.72 across all tasks which when equated to the actual time yields hold time of $0.72 \times 18 = 13$ seconds. This is compared to other HyperNEAT variants, OS variant with 12.3, NS with 11, OGN with 9.5 and lastly, GNS with 9 seconds.

The performance gain by each method and variant from behavior transfer is computed using equation 4.2:

$$Performance_Gain = \frac{\overline{Perf_{BT}} - \overline{Perf_{NoBT}}}{\overline{Perf_{BT}}} \tag{4.2}$$

where $\overline{Perf_{BT}}$ and $\overline{Perf_{NoBT}}$ are the normalized mean task performance for a method evolved with behavior transfer and that without behavior transfer, respectively.

The task performance gain metric, evaluates the effectiveness of behavior transfer on collective behavior tasks. The results are shown in table 4.5, where the average task performance gain is above 7% and the highest recorded is for the OS variant at 10.86%. Considering task performance across all tasks and all NE variants (for both NEAT and HyperNEAT), the highest task performance gain is obtained by NS and ONS variants with an average of 7.87% and 7.28%, respectively. The lowest task performance gain across all tasks is observed in GNS for both NEAT and HyperNEAT with an average of 4.59% and 5.56%, respectively. These results demonstrate that transfer of evolved behaviors from a source task to a relatively complex target task for further evolution, significantly improves task performance. This attests to the effectiveness of behavior transfer in bootstrapping evolution for collective behavior tasks of increasing complexity.

To further analyze the efficacy of behavior transfer for collective behavior tasks of increasing complexity, figure 4.3 and figure 4.3 provides a comparison of task performance results for all NE variants with and without behavior transfer. Keep-away behaviors were evolved in the source task for 30 generations, in order to evolve behaviors for bootstrapping evolution in the target task. Then the derived behaviors are transferred and used as the initial population for one of the five target keep-away tasks.

NE Variant	**Keep-Away Task (Percentage Performance Gain)**				
	4vs3	**5vs3**	**5vs4**	**6vs4**	**6vs5**
OS					
NEAT	6.59%	9.48%	10.48%	7.51%	10.86%
HyperNEAT	7.01%	7.63%	7.84%	5.92%	6.25%
NS					
NEAT	8.53%	6.69%	9.67%	7.93%	6.72%
HyperNEAT	6.53%	9.10%	7.74%	7.76%	8.00%
ONS					
NEAT	7.17%	7.06%	7.79%	6.81%	6.49%
HyperNEAT	8.70%	8.20%	8.30%	6.76%	7.72%
GNS					
NEAT	4.60%	4.56%	3.69%	4.68%	5.41%
HyperNEAT	6.73%	5.03%	5.21%	5.00%	5.85%
OGN					
NEAT	5.36%	4.34%	5.91%	5.88%	4.13%
HyperNEAT	8.20%	8.35%	8.86%	8.48%	8.82%

TABLE 4.5: Behavior transfer performance gain. Average percentage gain for five NE variants (NEAT and HyperNEAT) obtained using equation 4.2.

Figure 4.4 exhibits a jump-start on target task behavior evolution, derived from evolved behaviors from a source task which boost tasks performance in collective behavior task of increasing complexity. Task performance for a method that is initialized with behaviors evolved from a source task, starts off with high task performance values and records high fitness gradients for some generations. This indicated improved quality of solutions in populations initialized with transferred behaviors compared to those evolved from scratch. That is, population initialized with random weights. This is evident in both methods (NEAT and HyperNEAT) but more pronounced with HyperNEAT where representation of source and target tasks is static.

Figures 4.5 and 4.6 present the average maximum task performance of each NEAT and HyperNEAT variant at the final generation of evolution, in each target task. To highlight the benefits of using NEAT and HyperNEAT to facilitate behavior transfer, average task performance results of behaviors evolved without behavior transfer from previous evolution are included. These task performance values are obtained from 20 independent runs for each of the five NE variants and in each of the five tasks of increasing complexity (that is, *4vs3*, *5vs3*, *5vs4*, *6vs4* and *6vs5* keep-away). The results indicate that the task performance distributions in runs with behavior transfer is relatively higher than that of its counterparts, were evolution always started from scratch. All methods that use behavior transfer to bootstrap evolution in collective behavior tasks, have relatively high fitness values compared to those evolved without behavior transfer. The highest task performance is observed in ONS, where normalized fitness values are distributed above 0.6 for HyperNEAT and 0.5 for NEAT. HyperNEAT outperforms NEAT in all tasks which shows the appropriateness of HyperNEAT for evolving behaviors for collective behavior tasks.

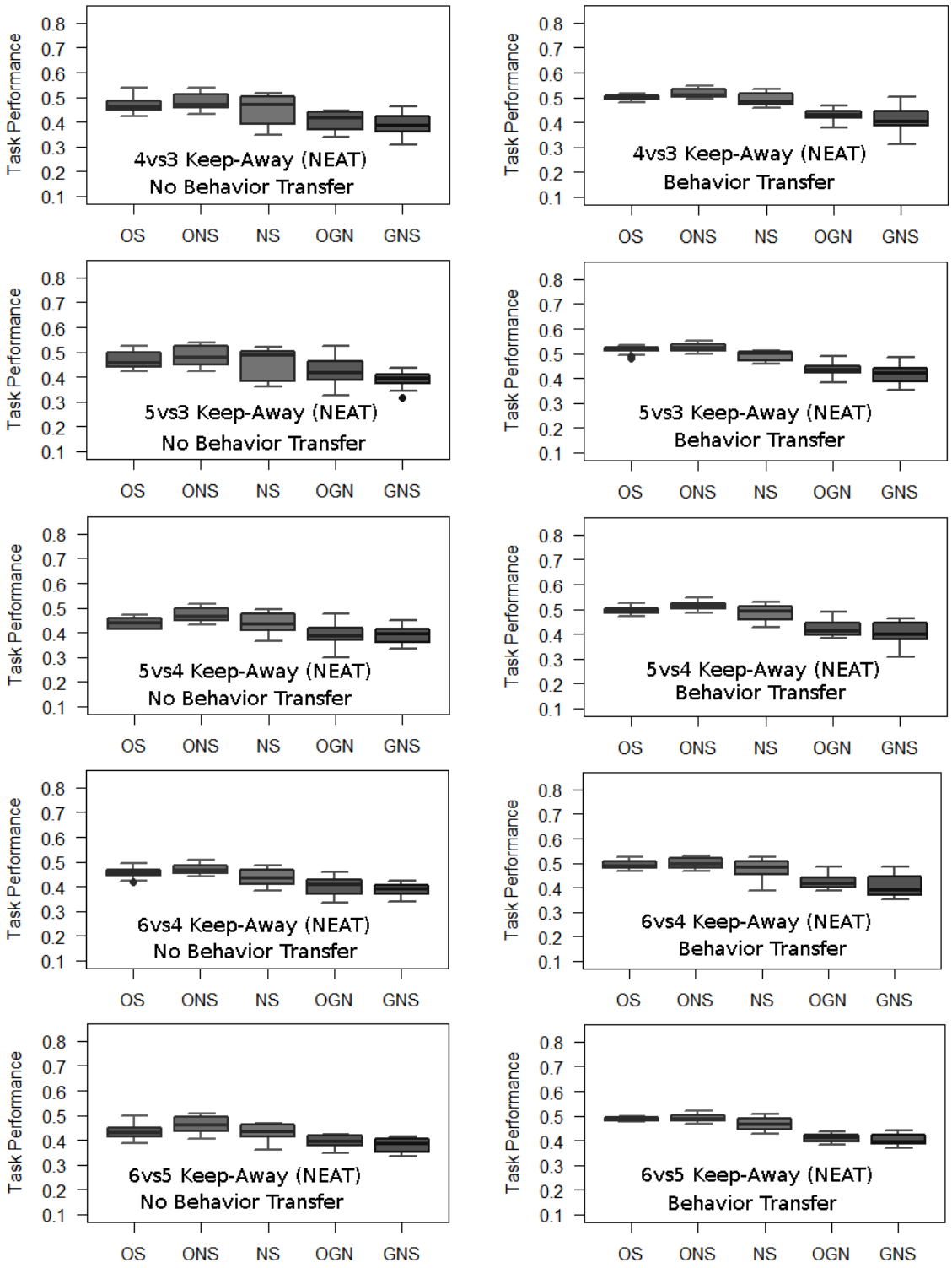


FIGURE 4.5: Average task performance distribution of genotypes. The box plot reflects the quartile distribution of actual task performance for 20 independent runs for all keep-away tasks, comparing performance with behavior transfer (right) and without behavior transfer (left) for NEAT.

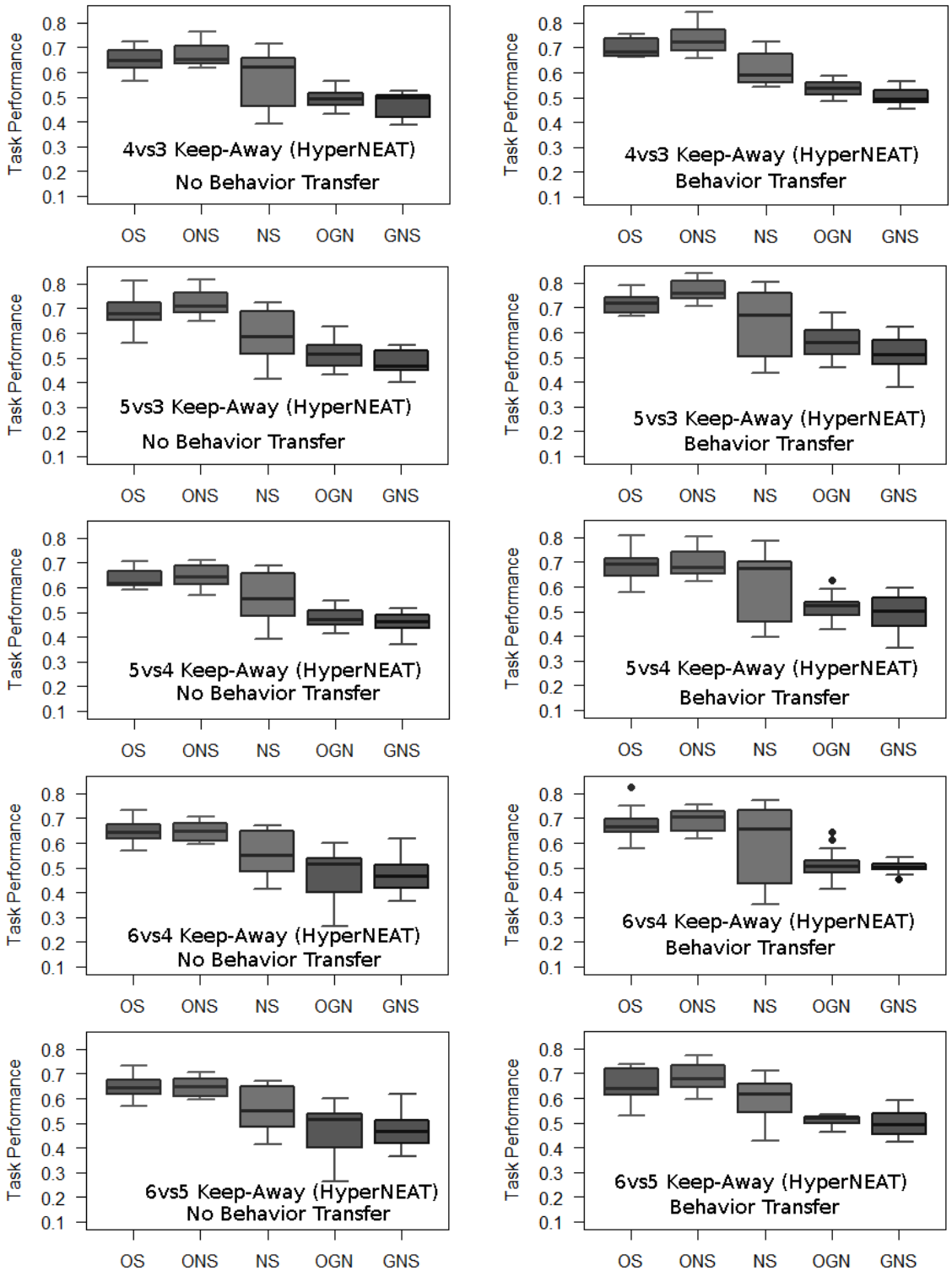


FIGURE 4.6: Average task performance distribution. The box plot reflects the quartile distribution of actual task performance for all keep-away tasks, comparing performance with behavior transfer (right) and without behavior transfer (left) for HyperNEAT.

The effectiveness of each NE variant is further investigated using the heat-maps (figure 4.8) produced from the stored population of candidate solutions at the final generation of evolution. Figure 4.7 and 4.8 shows heat-maps that presents portions of genotypes evolved by each variant of HyperNEAT and NEAT, that falls within each 20 percentile of normalized task performance. The mapping highlights the explorative capabilities of the comparative methods by showing the fitness diversity of the fittest evolved populations. Each NE variant has $150 \times 20 = 3000$ points on the heat-map, the darker the region, the higher the concentration of genotypes in that region. Hence, if darker regions fall in high performance regions, suggests that evolution was able to discover regions with high quality of solutions.

4.5.1 Task Performance Comparisons

The task performance comparison is conducted based on the statistical analysis results obtained from comparing all variants evolved with and without behavior transfer ($p < 0.05$, Mann-Whitney test). In the given experiments 4vs3 and 5vs3 keep-away tasks attained high performance values compared to other tasks. The task performance degrades as the complexity of the task increases for most of the tasks (with exception of *5vs3* that is more comparable to *4vs3* keep-away task performance) and the lowest performance of the five tasks is 6vs5. GNS task performance is the lowest for all tasks among all the NE variants. The following sections will discuss task performance with respect to each NE variant (that is, OS, NS, ONS, GNS and OGN).

4.5.2 Objective-based Search (OS variant)

Experimental results demonstrate the appropriateness of objective-based search for behavior evolution in keep-away, for all keep-away tasks tested with NEAT and HyperNEAT. The OS variant comes is the second highest in task performance and in some instances as noted in 5vs4 is comparable to ONS. The results of behaviors evolved with behavior transfer for OS variant performed well compared to those evolved from scratch. In the case of 4vs3 keep-away, 5% of OS evolved genotypes were in the performance range of $[0.6, 0.8]$ compared to the same task with 6% where evolution starts with evolved behaviors transferred from a source task. However, the performance gain is not that much compared to 25% of ONS evolved (with behavior transfer) genotypes that were in the performance range of $[0.6, 0.8]$ and 8% of genotypes in the same performance range evolved without behavior transfer. This particular case (given behavior transfer and behavior evolution with the ONS variant of HyperNEAT) exhibits a huge gain in average task performance.

Task performance results also indicate relative increase in task performance when keep-away team behaviors are evolved by NEAT and HyperNEAT OS variant with behavior transfer. Figures 4.3 and 4.6 attests to the effectiveness of behavior transfer in

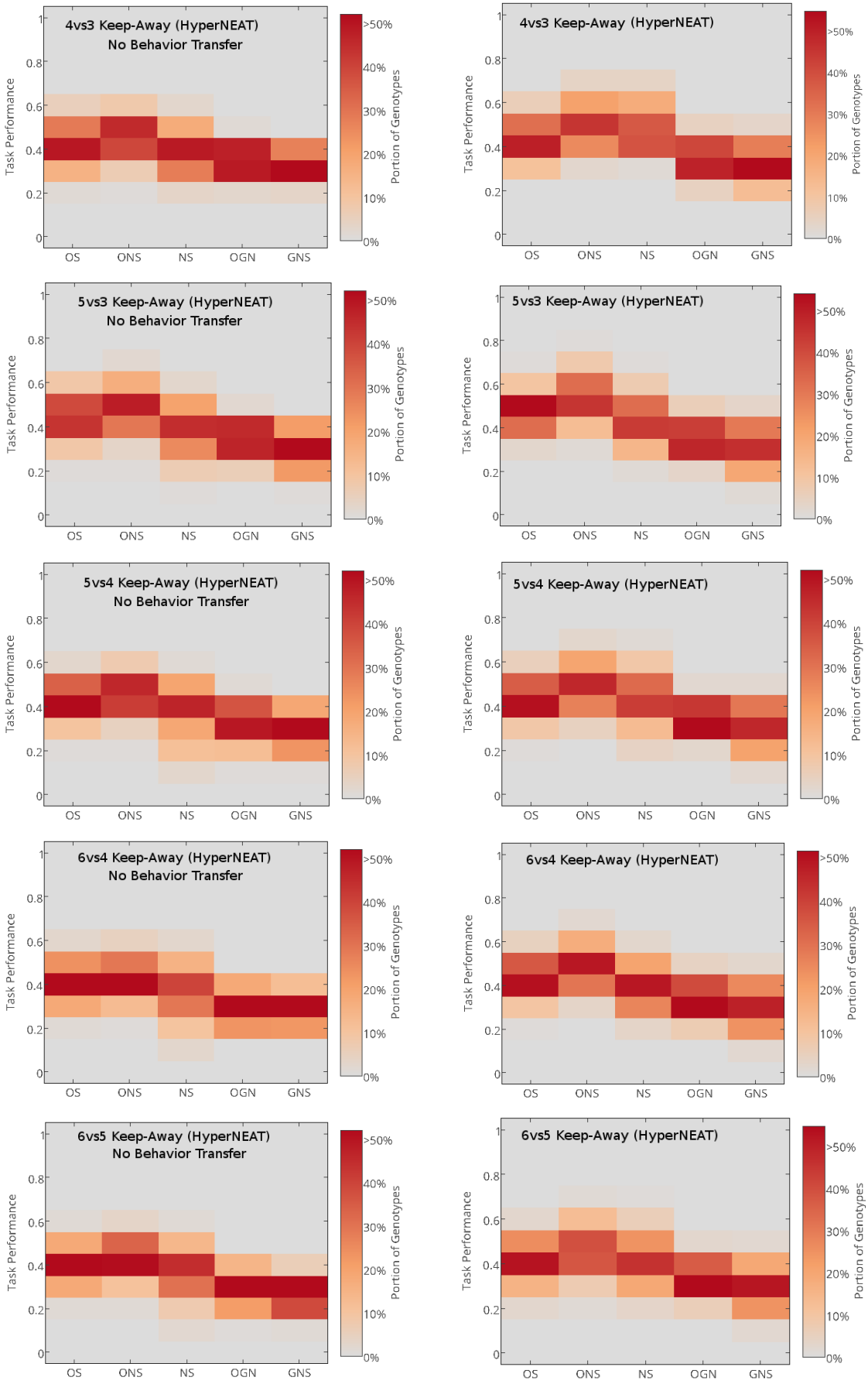


FIGURE 4.7: Heat-maps presenting the portion of genotypes, in the final generation of evolution. Heat-map for all keep-away tasks, with genotypes that falls within each 20 percentile of normalized task performance [0.0, 1.0] for five HyperNEAT variants evolved with and without behavior transfer.

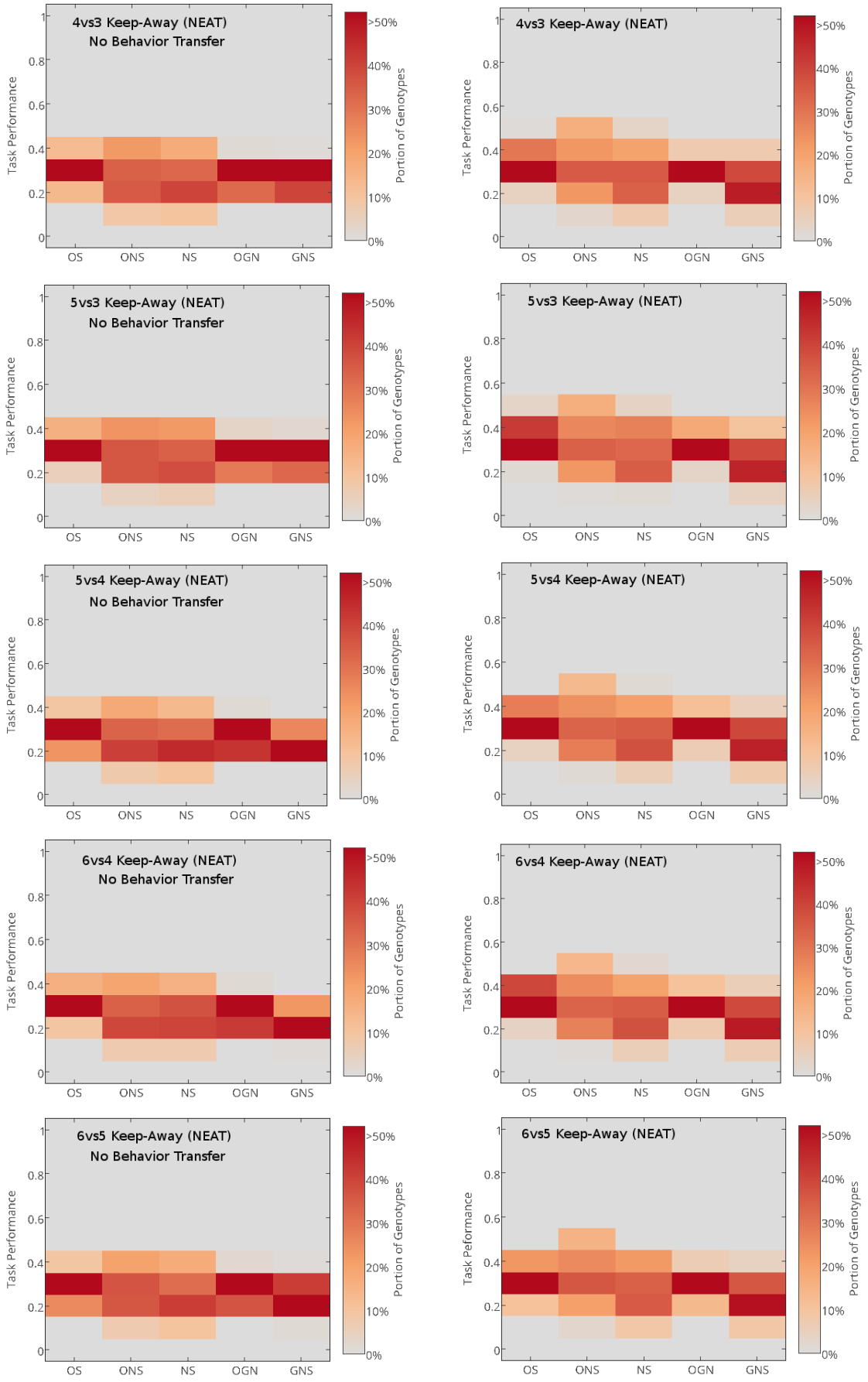


FIGURE 4.8: Heat-maps presenting the portion of genotypes, in the final generation of evolution in a given target task. Heat-map shows genotypes that falls within each 20 percentile of normalized task performance [0.0, 1.0] for five NEAT variants evolved with and without behavior transfer.

boosting evolution, especially at early generations of evolution. Apart from the task performance levels, figure 4.4 also shows the fitness progression that reflect quality of adaptation in each NE variant.

Task performance yielded by the OS variant increases at a slower rate with the number of generations compared to the task performance of the ONS variant. For example, consider 4vs3 keep-away, where OS with behavior transfer performs relatively well from generation 1 to generation 60, and there is then a gradual fall in the gradient of the task performance increase. Conversely, ONS maintains a constantly increasing fitness gradient which indicates the potential of the ONS variant for evolving effective behaviors in complex behavior tasks. In the case of 6vs5 keep-away, the OS variant of NEAT with behavior transfer has comparable performance to ONS and due to relatively high task complexity, we observe a low fitness gradient of task performance increase that flatness just after 60 generations of evolution (figure 4.3). However, the OS variant of HyperNEAT with behavior transfer (figure 4.4), shows a much higher gradient of task performance increase compared to the same variant without behavior transfer. Though this remains lower than the average task performance yielded by the ONS variant. The results demonstrates that the OS is better suited compared to GNS and OGN for evolving effective behaviors in increasingly complex collective behavior tasks, but not as appropriate as the ONS variant for evolving effective behaviors in such tasks.

4.5.3 Genotype Diversity Maintenance (GNS, OGN variants)

In terms of the effectiveness of genotypic diversity maintenance variants (GNS and OGN), in complex and deceptive tasks, results indicate that genotypic diversity maintenance approaches perform relatively poorly compared to objective based and behavioral diversity variants. Average task performance progression for keep-away tasks (given behavior transfer and no behavior transfer) is shown in figure 4.4. These results indicate the average of highest normalized task performance for five tasks tested in this thesis with both OGN and GNS does not exceed the 0.5 average performance level. This is very low compared to that of ONS which exceeds the 0.6 performance level for most tasks.

That is, figures 4.3 and 4.6 indicates that for all tasks, average tasks performance progressions of the GNS and OGN variants (NEAT and HyperNEAT) are consistently the lowest. For example, in 6vs5 keep-away, the GNS and OGN variants of both NEAT and HyperNEAT attain their maximum average task performance after approximately 50 generations, where as the average task performances of all other NE variants continue to increase beyond 50 generations.

These results demonstrate that the genotype diversity maintenance search approach, fails to build a selection gradient towards behaviors that lead to a high task performance that is comparable to the behavioral diversity and objective based approaches.

The topological diversity mechanism, does not incorporate or exploit behavioral properties that could directly search in the behavioral space. As topology refers to the structure of the controller and genotype encode controllers, hence the GNS and OGN variants explicitly select for genotypes (team controllers) that are dissimilar from each other. This way, selection pressure is towards genotypes with topological diversity instead of optimizing task performance. Figure 4.7 indicates that for all tasks, the largest portions of populations at final generation of evolution, GNS and OGN normalized task performance were in the range of $[0.3 - 0.4]$ which is lowest performing part of the behavioral space. For the HyperNEAT variants applied to *4vs3* keep-away, there are 0% of GNS and OGN genotypes in the task performance range of $[0.6 - 0.8]$, compared to 6% of OS, 25% of ONS and 22% of NS genotypes. Where as, there are 83% of OS, 72% of ONS, 76% of NS, 46% of OGN and only 33% of GNS evolved genotypes in the range of $[0.4 - 0.6]$. The rest of evolved genotypes are below the 0.4 task performance level. For *6vs5* keep-away, there are 0% of GNS and OGN genotypes in the task performance range of $[0.6 - 0.8$, compared to 3% of OS, 14% of ONS and 31% of NS genotypes. Where as, there are 78% of OS, 72% of ONS, 46% of NS, 39% of OGN and only 21% of GNS evolved genotypes in the range of $[0.4 - 0.6]$. Compared to OS, ONS and NS, these results indicate that the GNS and OGN variants of NEAT and HyperNEAT are not appropriate for evolving collective behaviors with relatively high task performance.

The effort to explicitly guide evolutionary search to high task performing behavioral space using OGN variant (a hybrid of OS and GNS), produced relatively poor task performance results compared to the other three variants (OS, ONS and NS). This is due to the fact that GNS selects for genetic diversity, but not necessarily the best controllers (that is, novel genotypes do not always translate into good behaviors). This is mitigated in the OGN approach, evidenced by the task performance and fitness diversity graphs (figure 4.3, 4.4, 4.8 and 4.7). The OGN includes objective-based search (fitness function) that selects for fitter solutions though this selection for genotypes with increased fitness is compromised by the selection only occurring within relatively poor regions of the genotype space.

The genotypic diversity mechanisms used in this thesis are methodologically most similar to that used by Kelly and Heywood (2014), who tested both genotypic and behavioral diversity maintenance in 4vs3 keep-away behavior evolution with genetic programming methods. Kelly and Heywood found behavior diversity maintenance yielded the greatest task performance benefits and genotype diversity was relatively ineffective but still preferable to no diversity maintenance (Kelly and Heywood, 2014). The results obtained in this thesis support the previous work, that confirms that behavioral diversity performs relatively well compared to genotypic diversity in directing neuro-evolution to finding the fittest genotypes (controllers) in the population (Mouret and Doncieux, 2009; Moriguchi and Honiden, 2010b; Gomez, 2009; Doncieux and Mouret, 2010).

To the author's knowledge, apart from Kelly and Heywood (2014), genotypic diversity maintenance mechanism such as GNS and OGN have not been evaluated in previous work investigating the task performance (in terms of effectiveness and efficiency) of genotypic versus behavioral diversity maintenance to direct evolutionary search and boost evolved collective behavior evolution. Overall, these results highlight that, compared to behavior diversity maintenance mechanism (ONS and NS) and objective-based search (OS), the genotypic diversity maintenance mechanisms (GNS and OGN) were relatively ineffective in collective behavior evolution.

4.5.4 Behavioral Diversity Maintenance (NS, ONS variants)

Novelty search (NS) is one type of behavioral diversity maintenance method that can be applied to direct any evolutionary search process (Mouret and Doncleux, 2009). NS effectively explores the behavioral space to discover solutions with varying behaviors and performance. However, because it ignores an objective function, there is no bias towards building a fitness gradient. A method that exploits the strength of both approaches is desirable, since as regions of highly fit genotypes are discovered, fitness optimization directs evolution to improve task performance within such high fitness regions.

The HyperNEAT ONS variant demonstrated its superiority in terms of task performance for all five tasks and behavior effectiveness for all five tasks (figure 4.4). For all keep-away tasks, the ONS variant evolved effective behaviors in terms of search space exploitation and discovered widely distributed behaviors with higher task performance ($p < 0.05$, Mann-Whitney test and Cohen's effect size > 0.6). The efficacy of the ONS variant for boosting evolved collective behavior task performance is supported by previous work that highlighted the benefits of hybrid objective-novelty search approaches for balancing exploration and exploitation of the behavior space for evolution of high quality solutions to single (Cuccu and Gomez., 2011) and multi-agent tasks (Gomes et al., 2014b). Exploration of the behavior space is necessary to discover a wide range of behaviors with varying qualities. Once such a high task performance region of the behavior space is discovered, then objective based search exploits the region via selecting the highest performing behaviors.

Comparative to other approaches, relatively few genotypes of the fittest ONS (NEAT and HyperNEAT) evolved populations were in the lowest task performance range (figure 4.8 and 4.7). However, statistical values indicate the effectiveness of NS variant of HyperNEAT in terms of evolving genotypes with relatively high fitness than other variants, in some collective behavior tasks. For example, for *6vs5* keep-away, NS variant of HyperNEAT has 31% of evolved genotypes in the task performance range of $[0.6, 0.8]$, compared to 14% of ONS, 3% of OS and 0% of GNS and OGN. Where as, there are 22% of NS variant of NEAT evolved genotypes in the task performance range of $[0.4, 0.6]$,

40% of ONS, 23% of OS, 7% of OGN and 5% of GNS (a comparable average performance of NS to OS variants of NEAT).

The efficacy of the ONS variant, resulting from its ability to balance behavior space exploration and search space exploitation, is demonstrated in figures 4.8 and 4.7 for NEAT and HyperNEAT behavior distribution, respectively. ONS evolved a greater portion of effective behaviors (in terms of task performance and fitness diversity in fittest evolved population), compared to all other variants in all the tasks. Figures 4.8 and 4.7 presents portions of genotypes with behaviors yielding task performance in the given regions. Apart from 6vs5 keep-away, the ONS variant of HyperNEAT records the highest percentage of genotypes in the fittest region of the search space. For example, in 5vs4 keep-away, 22% of genotypes are in the range of $[0.6, 0.8]$ for ONS compared to 6% and 10% for OS and NS variants, respectively. The NS variant of NEAT has consistently lower task performance than the OS and ONS variants in terms of the percentage of genotypes in the fittest region of the search space. For example, in 5vs3 keep-away, 31% of the NS NEAT variant evolved genotypes in the task performance range of $[0.4, 0.6]$, compared to 43% for ONS, 45% for OS, 18% for OGN and 10% for GNS.

The task performance progression graph in figure 4.4 and 4.3 demonstrates the capacity of the ONS variant to balance exploration versus exploitation for all keep-away tasks. That is, for all tasks, the task performance gradient of ONS evolved behaviors keeps going up, compared to other NE variants for both NEAT and HyperNEAT. This is especially the case of the ONS variant of HyperNEAT, where there was a statistically significant difference compared to other variants ($p < 0.05$, Mann-Whitney test). Comparing the task performance of NS and ONS variants, the results obtained suggest that behavioral diversity encourages the discovery of quality solutions, however for sustaining the task performance a hybrid method that combines objective-based search and behavioral diversity is an appropriate method for evolving behaviors in collective behavior tasks. Similarly, related research work showed hybrid behavioral diversity and objective based search methods worked well in evolving behaviors in swarm robotics task (Cuccu and Gomez., 2011; Gomes et al., 2013).

4.5.5 Efficiency Comparison

In ascertaining the quality of solutions produced by each evolutionary search variant and NE method for each task, we measure efficiency. Efficiency in this thesis is defined as the number of generations required to attain a task performance threshold, averaged over 30 trials and 20 evolutionary runs. The task performance threshold value is obtained experimentally, and is a steady maximum task performance for each method without behavior transfer. Efficiency is considered high, if the number of generations required for each method to reach a threshold is relatively low.

Task	OS Threshold	NEAT (N) Search Efficiency		HyperNEAT (HN) Search Efficiency	
		No BT	BT	No BT	BT
4 vs 3	**N**: 0.467 / **HN** : 0.662	89	43	61	30
5 vs 3	**N**: 0.469 / **HN** : 0.687	88	23	65	25
5 vs 4	**N**: 0.454 / **HN** : 0.650	83	26	61	34
6 vs 4	**N**: 0.457 / **HN** : 0.645	76	36	62	41
6 vs 5	**N**: 0.438 / **HN** : 0.614	84	19	90	29

Task	ONS Threshold	NEAT (N) Search Efficiency		HyperNEAT (HN) Search Efficiency	
		No BT	BT	No BT	BT
4 vs 3	**N**: 0.486 / **HN** : 0.689	76	50	80	44
5 vs 3	**N**: 0.492 / **HN** : 0.721	61	35	71	43
5 vs 4	**N**: 0.482 / **HN** : 0.654	62	47	66	22
6 vs 4	**N**: 0.474 / **HN** : 0.648	85	39	88	44
6 vs 5	**N**: 0.465 / **HN** : 0.632	63	28	63	36

Task	NS Threshold	NEAT (N) Search Efficiency		HyperNEAT (HN) Search Efficiency	
		No BT	BT	No BT	BT
4 vs 3	**N**: 0.451 / **HN** : 0.580	90	32	77	19
5 vs 3	**N**: 0.462 / **HN** : 0.591	69	54	82	21
5 vs 4	**N**: 0.440 / **HN** : 0.574	87	17	89	16
6 vs 4	**N**: 0.441 / **HN** : 0.560	55	15	77	15
6 vs 5	**N**: 0.442 / **HN** : 0.559	83	30	85	30

Task	GNS Threshold	NEAT (N) Search Efficiency		HyperNEAT (HN) Search Efficiency	
		No BT	BT	No BT	BT
4 vs 3	**N**: 0.402 / **HN** : 0.487	71	22	63	29
5 vs 3	**N**: 0.405 / **HN** : 0.491	94	37	86	35
5 vs 4	**N**: 0.392 / **HN** : 0.474	70	42	81	15
6 vs 4	**N**: 0.392 / **HN** : 0.481	85	13	80	25
6 vs 5	**N**: 0.389 / **HN** : 0.473	65	15	54	21

Task	OGN Threshold	NEAT (N) Search Efficiency		HyperNEAT (HN) Search Efficiency	
		No BT	BT	No BT	BT
4 vs 3	**N**: 0.430 / **HN** : 0.500	73	65	68	21
5 vs 3	**N**: 0.417 / **HN** : 0.521	97	23	62	23
5 vs 4	**N**: 0.402 / **HN** : 0.494	70	13	70	16
6 vs 4	**N**: 0.404 / **HN** : 0.481	50	34	82	19
6 vs 5	**N**: 0.404 / **HN** : 0.480	77	38	52	21

TABLE 4.6: Efficiency comparison of NEAT (N) versus HyperNEAT (HN) variants with Behavior Transfer (BT) and No Behavior Transfer (No BT). Search Efficiency: Average number of generations to reach the task performance threshold for the given variant.

The statistical tests were applied in pair-wise comparisons between average efficiency results of all NE variants with and without behavior transfer in each task (BT and No BT in table 4.6). That is, for each task, average NEAT and HyperNEAT efficiency for each NE variant with behavior transfer was compared to each NE variant without behavior

transfer. The results were statistically significant ($p < 0.05$, Mann-Whitney test), a higher efficiency was observed for all NE variants evolved with behavior transfer compared to the same NE variants without behavior transfer. This demonstrates the efficacy of using behavior transfer for bootstrapping evolution in collective behavior tasks of increasing complexity. This was especially the case for HyperNEAT variants in 6vs5 keep-away, where results indicates a relatively large efficiency gains versus other NE variants. That is, when behavior transfer is used to boost evolution in a complex collective behavior task. For example, OS variant results show that 29 generations are required to reach a threshold of 0.614 with behavior transfer compared to 90 generations of evolution without behavior transfer. However, the ONS variant applied to the same task, requires 36 generations to reach a threshold value of 0.632 with behavior transfer, compared to 63 generations without behavior transfer. The HyperNEAT NS variant, requires 30 generations of evolution to reach a threshold value of 0.559 with behavior transfer, compared to 85 generations without behavior transfer. The HyperNEAT GNS variant, requires 21 generations to reach a threshold value of 0.473 with behavior transfer, compared to 54 generations without behavior transfer. Lastly, the OGN variant, requires 21 generations of evolution to reach a threshold value of 0.480 with behavior transfer, compared to 52 generations without behavior transfer. Similar results are observed in all other NE variants, that shows a significant efficiency gains where behaviors were evolved with behavior transfer, compared to evolving genotypes without behavior transfer.

4.5.6 Solution Complexity

To understand why hybrid objective-based and novelty search evolved behaviors with higher task task performance than objective-based search for all keep-away tasks in this case study. We compared solution (topological) complexity of evolved networks by all NEAT and HyperNEAT search variants. Solution complexity is obtained from the ANN topological complexity for each method. Previous work demonstrated that NEAT novelty search explores comparatively simple solutions before it explores complex solutions and hence, evolves behaviors with lower topological complexity compared to NEAT objective-based search (Lehman and Stanley, 2011a; Gomes et al., 2013). However, there has been little work investigating topological complexity of HyperNEAT evolved behaviors given novelty search and objective-based search (Morse et al., 2013). To ascertain if that advantage holds in behaviors evolved with HyperNEAT in keep-away task, we compare complexity of solutions evolved with all five NE variants. Similar to the previous research (Lehman and Stanley, 2011a; Gomes et al., 2013), topological complexity is given by the number of connections and neurons of the fittest solutions averaged over 20 runs (equation 4.3).

$$E_x = \frac{1}{N}\sum_{i=1}^{N}(n_c + n_n) \tag{4.3}$$

where N is the number of runs, n_c and n_n, is the number of ANN connections and hidden nodes, respectively. For clarity, network x complexity (E) is normalized to the range: [0.0,1.0]. A 1.0 value indicates maximum network complexity as observed for behaviors evolved with each NE method (NEAT or HyperNEAT).

Figure 4.9 presents a comparison between complexity of solutions evolved with each variant of NEAT and HyperNEAT. These results indicate that GNS evolve solutions with highest average topological complexity than the rest, since the approach optimizes for variations in genotypic space ($p < 0.05$, Mann-Whitney u test) (Appendix C). A method that attains a good solution with relatively low topological complexity is preferred as it allows evolution to run faster and much efficient. Figure 4.9 indicates that NS and ONS variants for all tasks (both NEAT and HyperNEAT) have lower topological complexity at the maximum generation of evolution. The NS and ONS variants have significantly lower network complexity compared to OS, OGN and GNS. For example, in keep-away 4vs3, NS variant has the lowest normalized network complexity of 0.477, compared to ONS with 0.499, OS with 0.516, OGN with 0.544 and GNS with normalized network complexity of 0.795. Comparable results are observed across all other keep-away tasks, where networks evolved with NS variant had the lowest average network complexity and networks evolved with GNS variants had the highest average network complexity. These network complexity results confirm that novelty search can find solutions with relatively lower topological complexity than either objective-based search or genotypic diversity maintenance (Lehman and Stanley, 2011a; Gomes et al., 2013).

Topology complexity results show an increase in network complexity with an increase in task complexity for most of keep-away tasks in this case study. For example, in keep-away 4vs3 OS the mean normalized network complexity was 0.516, compared to 0.539 for keep-away 6vs5, 0.499 for ONS evolved keep-away 4vs3 compared to 0.504 for 6vs5. NS variant had 0.477 compared to 0.506, OGN with 0.544 compared to 0.560 and the GNS with 0.795 of normalized network complexity for keep-away 4vs3 compared to 6vs5 with complexity of 0.861.

Table 4.9 shows the average minimum and maximum topological complexity of each HyperNEAT variant at the final generation of evolution. The minimum and maximum complexity values are computed by taking the average minimum and maximum network complexity at generation 100 across 20 runs, respectively. These results show that NS variant consistently, had the lowest minimum network complexity across all keep-away tasks. For example, keep-away 4vs3, NS variant had mean normalized complexity of 0.446, compared with 0.460 of ONS, 0.472 of OS, 0.474 of OGN and 0.611 of GNS. Conversely, GNS variant had the highest minimum network complexity than all HyperNEAT variants. In this case study, the highest mean network complexity was

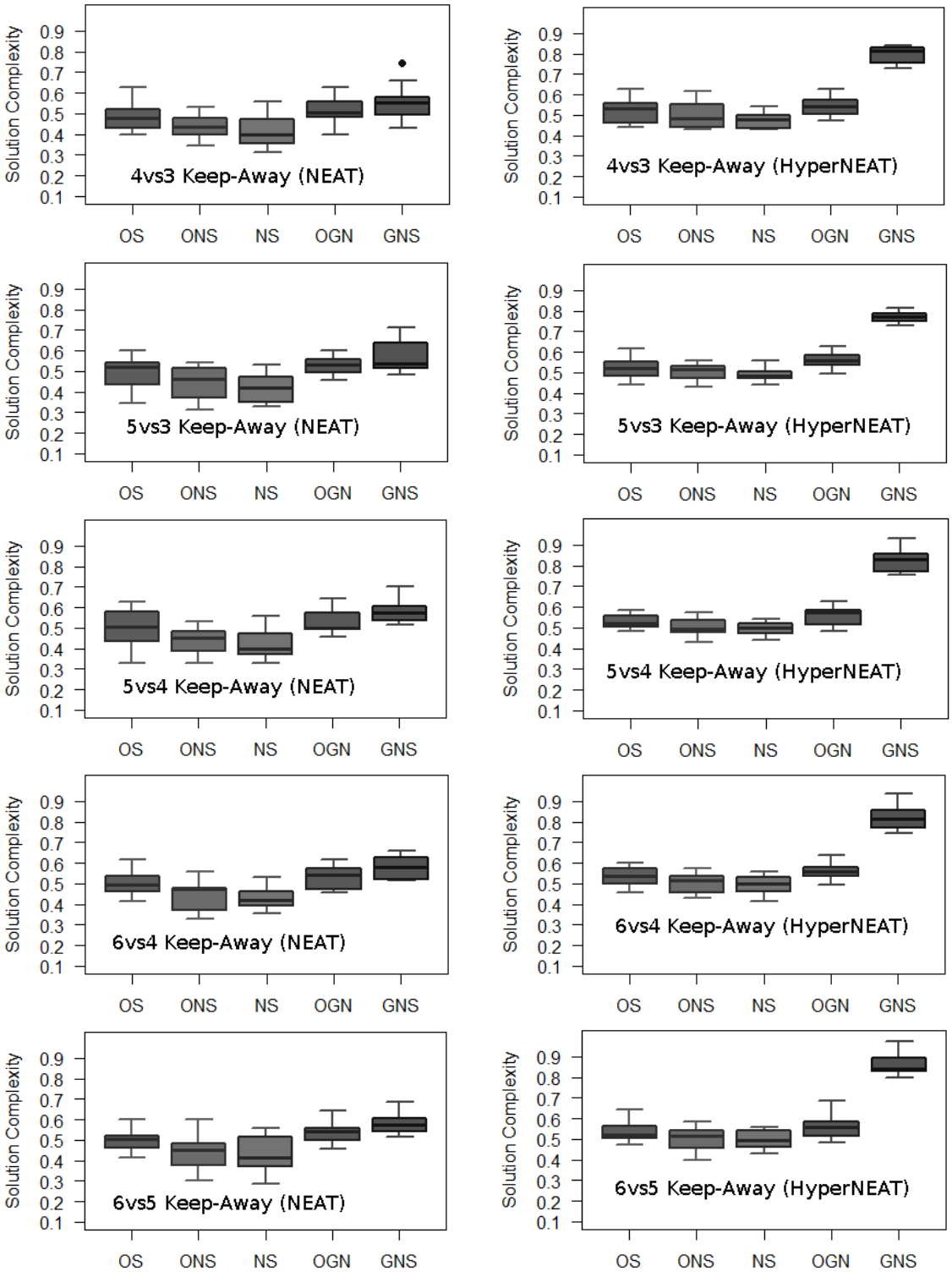


FIGURE 4.9: Topological complexity distribution of best-of-generation genotypes. The box plot reflects the quartile distribution of topological (solution) complexity from the 20 independent runs for all keep-away tasks (NEAT and HyperNEAT).

Task Performance	**Keep-Away 4vs3**									
	Generations					**Complexity**				
	OS	ONS	NS	OGN	GNS	OS	ONS	NS	OGN	GNS
0.45	-	-	-	1	5	-	-	-	0.404	0.439
0.50	-	-	1	21	72	-	-	0.380	0.436	0.726
0.55	1	1	6	-	-	0.386	0.379	0.386	-	-
0.60	4	5	34	-	-	0.411	0.408	0.414	-	-
0.65	25	21	-	-	-	0.446	0.424	-	-	-
0.70	88	53	-	-	-	0.476	0.450	-	-	-

Task Performance	**Keep-Away 5vs3**									
	Generations					**Complexity**				
	OS	ONS	NS	OGN	GNS	OS	ONS	NS	OGN	GNS
0.45	-	-	-	1	11	-	-	-	0.414	0.466
0.50	-	-	1	10	72	-	-	0.397	0.433	0.740
0.55	2	1	3	48	-	0.403	0.399	0.402	0.455	-
0.60	4	4	32	-	-	0.419	0.406	0.424	-	-
0.65	9	14	-	-	-	0.421	0.409	-	-	-
0.70	48	27	-	-	-	0.424	0.414	-	-	-

Task Performance	**Keep-Away 5vs4**									
	Generations					**Complexity**				
	OS	ONS	NS	OGN	GNS	OS	ONS	NS	OGN	GNS
0.45	-	-	-	1	5	-	-	-	0.425	0.484
0.50	-	-	1	23	90	-	-	0.399	0.440	0.799
0.55	1	1	2	-	-	0.417	0.402	0.401	-	-
0.60	7	5	37	-	-	0.419	0.407	0.424	-	-
0.65	34	18	-	-	-	0.433	0.418	-	-	-
0.70	-	78	-	-	-	0.457	0.443	-	-	-

Task Performance	**Keep-Away 6vs4**									
	Generations					**Complexity**				
	OS	ONS	NS	OGN	GNS	OS	ONS	NS	OGN	GNS
0.45	-	-	-	1	1	-	-	-	0.447	0.457
0.50	-	-	1	35	75	-	-	0.408	0.485	0.755
0.55	1	1	10	-	-	0.405	0.401	0.413	-	-
0.60	16	8	70	-	-	0.423	0.421	0.435	-	-
0.65	46	42	-	-	-	0.441	0.435	-	-	-
0.70	-	-	-	-	-	-	-	-	-	-

Task Performance	**Keep-Away 6vs5**									
	Generations					**Complexity**				
	OS	ONS	NS	OGN	GNS	OS	ONS	NS	OGN	GNS
0.45	-	-	-	1	1	-	-	-	0.457	0.464
0.50	-	-	1	45	91	-	-	0.402	0.498	0.814
0.55	1	1	17	-	-	0.413	0.410	0.416	-	-
0.60	17	10	83	-	-	0.432	0.419	0.430	-	-
0.65	86	61	-	-	-	0.480	0.449	-	-	-
0.70	-	-	-	-	-	-	-	-	-	-

TABLE 4.7: Average normalized CPPN complexity (neurons and connections, over 20 runs) for fittest behaviors evolved by each HyperNEAT variant for each keep-away task. The Task Performance column indicates which 5 percentile group these fittest behaviors are in and the Generations column indicates the average number of generations taken to evolve the corresponding best performing behaviors and network complexity (given behavior transfer).

Task Performance	**Average Evolved Network Complexity**				
	OS	ONS	NS	OGN	GNS
4vs3	0.516 ± 0.051	0.499 ± 0.055	0.477 ± 0.037	0.544 ± 0.041	0.795 ± 0.039
5vs3	0.521 ± 0.042	0.501 ± 0.035	0.493 ± 0.030	0.556 ± 0.041	0.771 ± 0.42
5vs4	0.531 ± 0.046	0.501 ± 0.030	0.495 ± 0.026	0.559 ± 0.053	0.829 ± 0.053
6vs4	0.538 ± 0.042	0.502 ± 0.041	0.497 ± 0.041	0.556 ± 0.049	0.832 ± 0.066
6vs5	0.539 ± 0.046	0.504 ± 0.051	0.506 ± 0.051	0.560 ± 0.051	0.861 ± 0.048

TABLE 4.8: Solution complexity comparison for HyperNEAT variants with Behavior Transfer (BT) at the final generation of evolution.

Task Performance	**Min and Max Network Complexity**				
	OS	ONS	NS	OGN	GNS
	[Min,Max]	[Min,Max]	[Min,Max]	[Min,Max]	[Min,Max]
4vs3	[0.472,0.526]	[0.460,0.519]	[0.446,0.519]	[0.474,0.587]	[0.611,0.874]
5vs3	[0.483,0.556]	[0.466,0.548]	[0.447,0.539]	[0.476,0.659]	[0.626,0.853]
5vs4	[0.482,0.564]	[0.463,0.551]	[0.457,0.527]	[0.486,0.628]	[0.663,0.907]
6vs4	[0.488,0.577]	[0.468,0.556]	[0.452,0.539]	[0.489,0.614]	[0.622,0.928]
6vs5	[0.494,0.580]	[0.462,0.549]	[0.460,0.540]	[0.496,0.659]	[0.674,0.942]
	Mean Network Complexity				
	[0.484,0.561]	[0.464,0.545]	[0.452,0.533]	[0.484,0.629]	[0.639,0.901]

TABLE 4.9: Average minimum and maximum network complexity for all HyperNEAT variants with behavior transfer at the final generation of evolution.

attained by the GNS with value of 0.874, compared to OGN with 0.587, OS with 0.526, NS and ONS with lowest mean normalized maximum network complexity of 0.519.

Similar observations were noted with other keep-away tasks. For example, keep-away 5vs3 network complexity analysis results show NS variant with the lowest mean normalized network complexity range of [0.447, 0.538], ONS with [0.466, 0.538], OS with [0.483, 0.556], OGN with [0.476, 0.659] and, lastly, GNS with network complexity range of [0.626, 0.853]. Keep-away 5vs4, shows the NS variant with lowest mean complexity range of [0.457, 0.527] and GNS variant with the highest mean network complexity range of [0.457, 0.527]. Similarly, keep-away 6vs4 and 6vs5 tasks shows NS variant with the lowest complexity values, subsequently followed by ONS, OS, OGN and GNS has the highest values, with network complexity of 0.928 and 0.942, respectively.

The difference in solution complexity levels between OS and NS is ascribed to the convergent nature of objective-based evolution. However, novelty search explores simple solutions before exploiting higher levels of topological complexity, there-by delaying convergence. NS and ONS as a result, are capable of finding solutions with lower topological complexity compared to OS. GNS and OGN on the other hand evolve solutions that search for solution directly in the topological space and as a result, finds solutions that have higher topological complexity than OS, ONS and NS variants of NEAT and HyperNEAT.

Further analysis of behaviors evolved by each NE variant, table 4.7 presents network complexity and efficiency values corresponding to the fittest behaviors evolved by each variant in each keep-away task. To show how topological complexity relates to the effectiveness (task performance) of evolved behaviors, the left column of table 4.7 shows task performance level attained by each NE variant. To show how complexity relates to efficiency (speed of adaptation) of each NE variant evolved behaviors, the generations column represent the average number of generations each variant takes to reach the given task performance level.

Table 4.7 supports the previous work and shows that behavior diversity maintenance methods evolve simple and high quality controllers, for all keep-away tasks (Gomes et al., 2013; Nitschke and Didi, 2017). ONS evolved minimal average network complexity and the highest average task performance, compared to other NE variants. In some tasks a significantly lower network complexity was evolved by the NS variant, but in these tasks NS yielded a lower average task performance. For example, 6vs5 keep-away (the most complex task), the fittest ONS evolved behaviors corresponded to average network complexity: 0.449 and task performance range of $[0.65, 0.7]$ in 61 generations (on average), where the fittest NS corresponded to average network complexity: 0.430 and task performance range: $[0.60, 0.65]$. Where as the fittest OS corresponded to average network complexity: 0.480 and task performance range of $[0.65, 0.70]$ in 80 generations (on average). Similar results were observed with all other keep-away tasks (4vs3, 5vs3, 5vs4 and 6vs4).

GNS variant has the evolved networks with the highest solution complexity but lowest tasks performance. For example, for 6vs5 keep-away, GNS variant has solution complexity of 0.814, compared to 0.498 of OGN and 0.402 at task performace 0.50 (generation 91, 45 and 1, respectively). The significantly higher average network complexity of the fittest GNS evolved behaviors is attributed to the genotypic diversity maintenance mechanism of GNS (section 3.4.3).

OS and OGN evolved behaviors with relatively high complexity compared to ONS and NS (but lower than GNS) but with lower quality than ONS. OS had higher complexity due to exploitative nature of objective-based search. ONS overall, yielded the most benefits, in terms of evolving highest quality behaviors encoded by significantly simple networks than all the other variants across all keep-away tasks (table 4.9, 4.7, 4.8). These results demonstrate that the ONS variant is the most appropriate NE evolutionary search method that addresses the bootstrapping problem for increasingly complex keep-away tasks, given behavior transfer. Furthermore, these results lend support to the previous research that demonstrated superiority of hybrid objective-behavior diversity search for complex multi-robotic tasks (Gomes et al., 2013).

The following section presents behavior search space analysis, showing the behavior space exploration and visualization.

4.5.7 Behavior Search Space Analysis

To elucidate each NE search variant's capacity to explore behavior spaces, we applied Kohonen self-organizing maps (SOMs) (Kohonen, 1997) to the final generation of behaviors evolved by each variant to visualize the contribution of various behavioral components to the fittest behaviors types. SOMs are artificial neural networks trained using unsupervised learning to create a mapping that transforms multiple dimensional space into a low dimensional input space, while preserving topological relations of the original input pattern. These SOMs were selected as previous work (Gomes et al., 2013) indicated their suitability for evolved behavior analysis.

For all keep-away tasks, the final generation behavior population evolved by each NE variant was used as input data for training SOMs. The behavior characterization vectors characterizing keep-away behavior include three variables: *average number of passes*, *dispersion of team members*, and *distance of the ball to the center of the field* (section 3.4.2). The objective of this behavior analysis was to visualize different behavioral patterns created through collective behavior adaptation based on NEAT and HyperNEAT variants. Hence, draw some synergies between each pattern produced and task performance (indicated by episodic length).

For clarity and thorough analysis, we selected to only visualize behavior types for HyperNEAT variants. Figures 4.10, 4.11, 4.12, 4.13 and 4.14 shows two types of behavior visualization, Left: code-book vector representation and right: unified distance matrix indicating behavior exploration in each HyperNEAT variant across the given keep-away tasks of increasing complexity. In this way, code-book vector behavior visualization on the left complements the unified distance matrix on the right to show keep-away behavior vector distribution on the lattice. Furthermore, we do not discuss behavior maps for OGN and GNS variants in this chapter, since these variants yielded significantly lower average task performances compared to ONS, NS and OS, for all tasks (section 4.5). For reference, appendix E presents behavior type visualizations of GNS and OGN search variants.

Code-book vector representation of the given input pattern is a visualization of behavior exploration based on weight vectors, where each segment represents the magnitude of each behavior variable in each node. That is, the size of the each variable segment represents the weight of genotypes with such as a behavior in that region. For example, for 4vs3 keep-away, ONS variant the episodic length is spread out throughout the lattice but highly concentrated on the left side of the lattice (figure 4.10). Furthermore, this map shows relationships between behavioral vectors, for example episodic length correlates to number of passes. The the top left corner of each map (left side), represents the highly fit genotypes, ONS has a higher number of genotypes in that region than the rest of other HyperNEAT variants (figure 4.10).

Unified distance matrix (u-matrix) exhibited in figures 4.10, 4.11, 4.12, 4.13 and 4.14 on the right, is one method of visualizing clusters with a 10×10 SOM map. This metric is obtained by computing the Euclidean distance in the input space of units that are neighbors within the SOM map. Each u-matrix illustrates a representation of how the final generation of behaviors evolved by each variant are topologically related to each other in the behavior space. This provides an indication of the diversity of evolved behavior types and thus provide a tool for analyzing behavior exploration. U-matrices are presented on a gray scale map, where a light color represents a large distance and that indicate sparse regions of the input space. On the other hand, a dark color indicate a cluster of similar data in that region of the input space. For example, there is a cluster for ONS in the top left corner region which represents highly fit genotypes that suggesting a significantly higher concentration of genotypes in that area that close to each other in a cluster formation (figure 4.10). Such a cluster is also observed in NS variant on the left-middle side of figure 4.10 (right), a region with fairly distributed behavioral vectors when related to the code-book vector representation (left). The differing u-matrix y-axis values indicates normalized behavior distances.

Observing the ONS behavior visualization map in figures 4.10, 4.11, 4.12, 4.13 and 4.14 (left), the fittest behavior types are characterized by the following components. First, high number of passes with little team dispersion and distance to the field's center. Second, maximizing keeper distance from the field's center and team dispersion with relatively few passes. Comparatively, the fittest OS evolved behavior types similarly had a large episodic length with low team dispersion and average distance from the center of the field. ONS compared to OS shows widely spaced clusters in the lattice (figure 4.10 right) and dark patches on the highly fit regions of the behavioral search space (4.10 right). This equates to the ONS variant's capability to adequately explore the behavioral search space and discover regions of highly fit behaviors. This support the superiority of ONS over OS variants in evolved high quality behaviors in keep-away tasks of increasing complexity (section 4.5.4).

Observing OS u-matrix (figure 4.14 right) by comparison to ONS, the overall behavior distribution in the lattice is dark with few light patches, meaning that OS evolved behavior types were relatively close together in the behavior space and OS variant was operating in a comparatively small and less fit region of the behavior space. The relative lack of diversity between these behavior types is indicated by the similarity in dark regions of most OS u-matrix cells, with few exceptions such as the left side of the bottom row of the OS u-matrix (figure 4.14 right). The behavior explored by OS at the final generation of evolution can thus be viewed as one large cluster of behavior types that were in close proximity in the behavior space but did not share biases for specific behavior components. However, compared to other variants, for example NS and OS has a relatively darker region towards the fittest regions of the behavioral space (u-matrix top-left corner) (figure 4.14 right). This indicate that OS evolves behaviors with higher task performance that the other three variants (NS, OGN and GNS).

Observing NS u-matrix (figure 4.14 right), overall light color in the SOM map lattice indicates slightly more pronounced behavior type diversity at the final generation of evolution. The significantly lower average task performance of NS versus ONS and OS evolved behaviors indicates that while wide behavior search space exploration is necessary for discovering highly fit behaviors, the lack of an exploitative objective-based search causes NS to be ineffective across keep-away tasks of increasing complexity.

GNS and OGN variants have significant low task performance due to the fact that the genotypic diversity maintenance mechanism has no bias towards behavior exploration in the behavior space and also fails to build a selection gradient towards behaviors that lead to a high task performance that is comparable to the behavioral diversity and objective based approaches (section 4.5.3).

The following section elucidate the impact of behavior transfer on effectiveness and efficiency of evolved behaviors in keep-away task of increasing complexity.

4.5.8 Behavior Transfer Results

The *Mann-Whitney* test (Flannery et al., 1986) was applied in a series of pair-wise comparisons to gauge if there is a statistically significant difference between corresponding result sets of the five method variants of NEAT and HyperNEAT. Pair-wise comparisons were conducted between average results data for NEAT or HyperNEAT (for a given experiment). The null hypothesis stated that two comparative data sets were not significantly different, and $p = 0.05$ was selected as the significance threshold.

The performance of each NE variant for both NEAT and HyperNEAT with behavior transfer is contrasted with the same variant evolved from scratch. Figure 4.4 shows average task performance progression, highlighting the performance difference between further evolution after behavior transfer and evolution from scratch (for each method variant). Statistical tests indicated that for all keep-away tasks, HyperNEAT variants with behavior transfer yielded significantly higher average performance based on effectiveness and efficiency of solutions ($p < 0.05$, Mann-Whitney test). Specifically, the highest average task performance was observed when behaviors were evolved with ONS variants of HyperNEAT with behavior transfer for all keep-away tasks.

Figures 4.5 and 4.6 further demonstrate that there is a clear difference between NEAT and HyperNEAT results with and without behavior transfer. The results are further presented in table 4.5 showing task performance gain, given behavior transfer. The results for all keep-away tasks and NE method (NEAT and HyperNEAT) variants shows a significant percentage gain in task performance when behaviors are further evolved after policy transfer compared to behavior evolution from scratch ($p < 0.05$). That is, average task performance comparisons indicated that, for all keep-away tasks, the majority of methods that evolved keep-away behaviors with behavior transfer (477 out

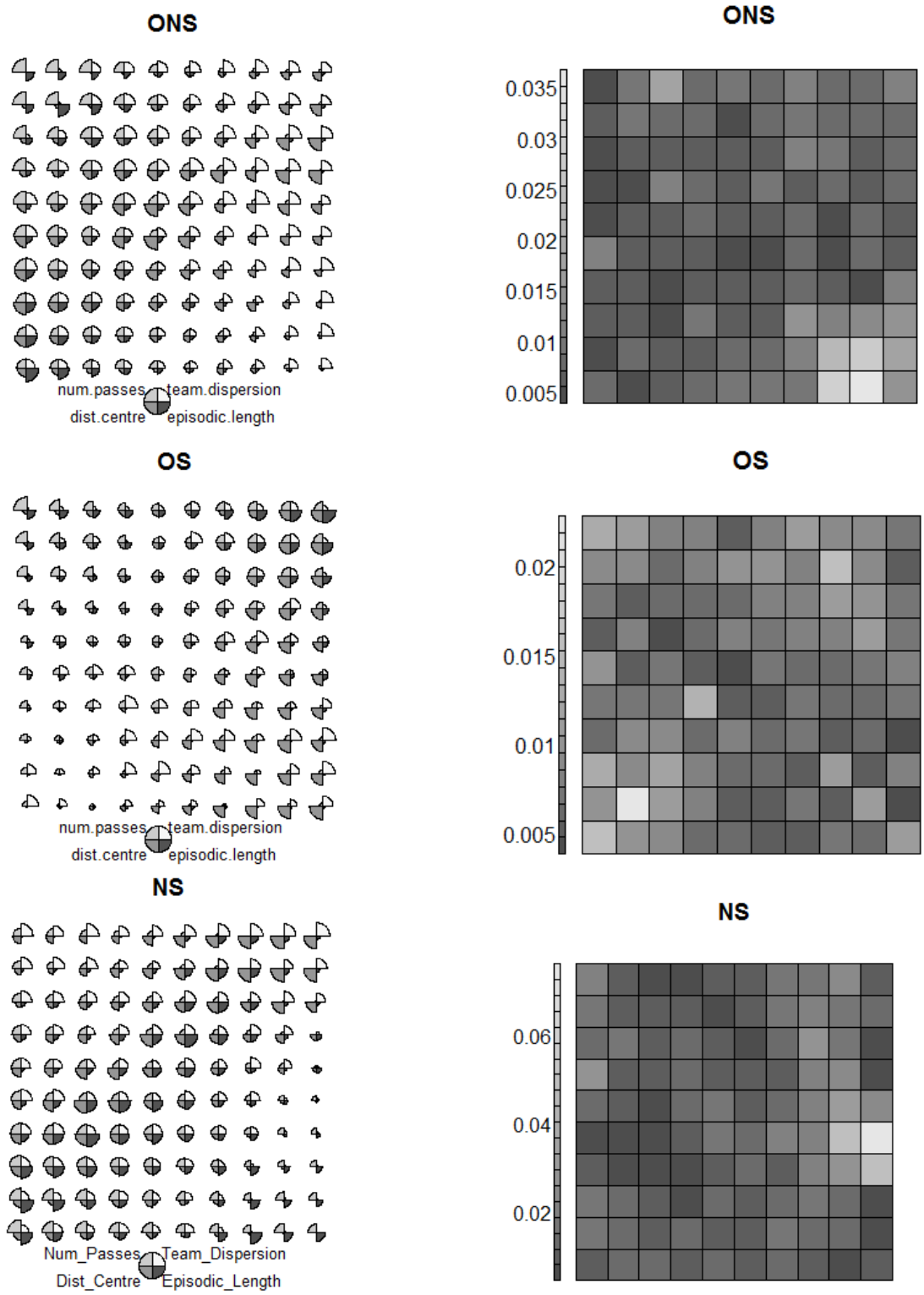


FIGURE 4.10: Kohonen Self-Organizing Maps (SOMs) representing the explored behavior search space in Keep-away 4vs3 for given HyperNEAT variants with behavior transfer. In each NE variant, left: shows codebook weight vector for data visualization and right: shows u-matrix representing boundaries between clusters. Each cirle in the codebook weight vector represents a behavior type (four slices). Each slice represents one component of the behavior characterization vector. The differing u-matrix Y-axis values indicates normalized behavior distances.

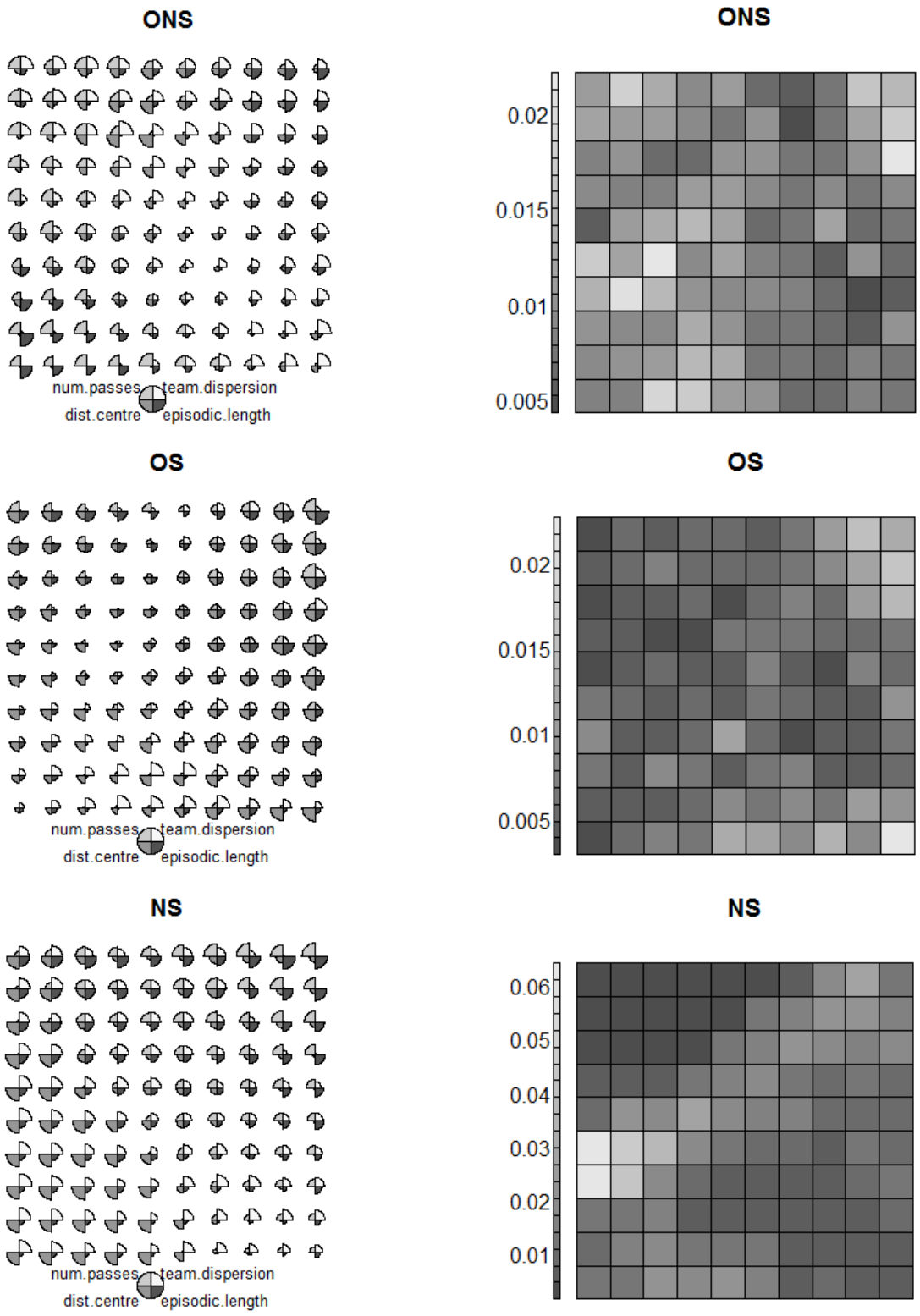


FIGURE 4.11: Kohonen Self-Organizing Maps (SOMs) representing the explored behavior search space in Keep-away 5vs3 for given HyperNEAT variants with behavior transfer. In each NE variant, left: shows codebook weight vector for data visualization and right: shows u-matrix representing boundaries between clusters. Each cirle in the codebook weight vector represents a behavior type (four slices). Each slice represents one component of the behavior characterization vector. The differing u-matrix Y-axis values indicates normalized behavior distances.

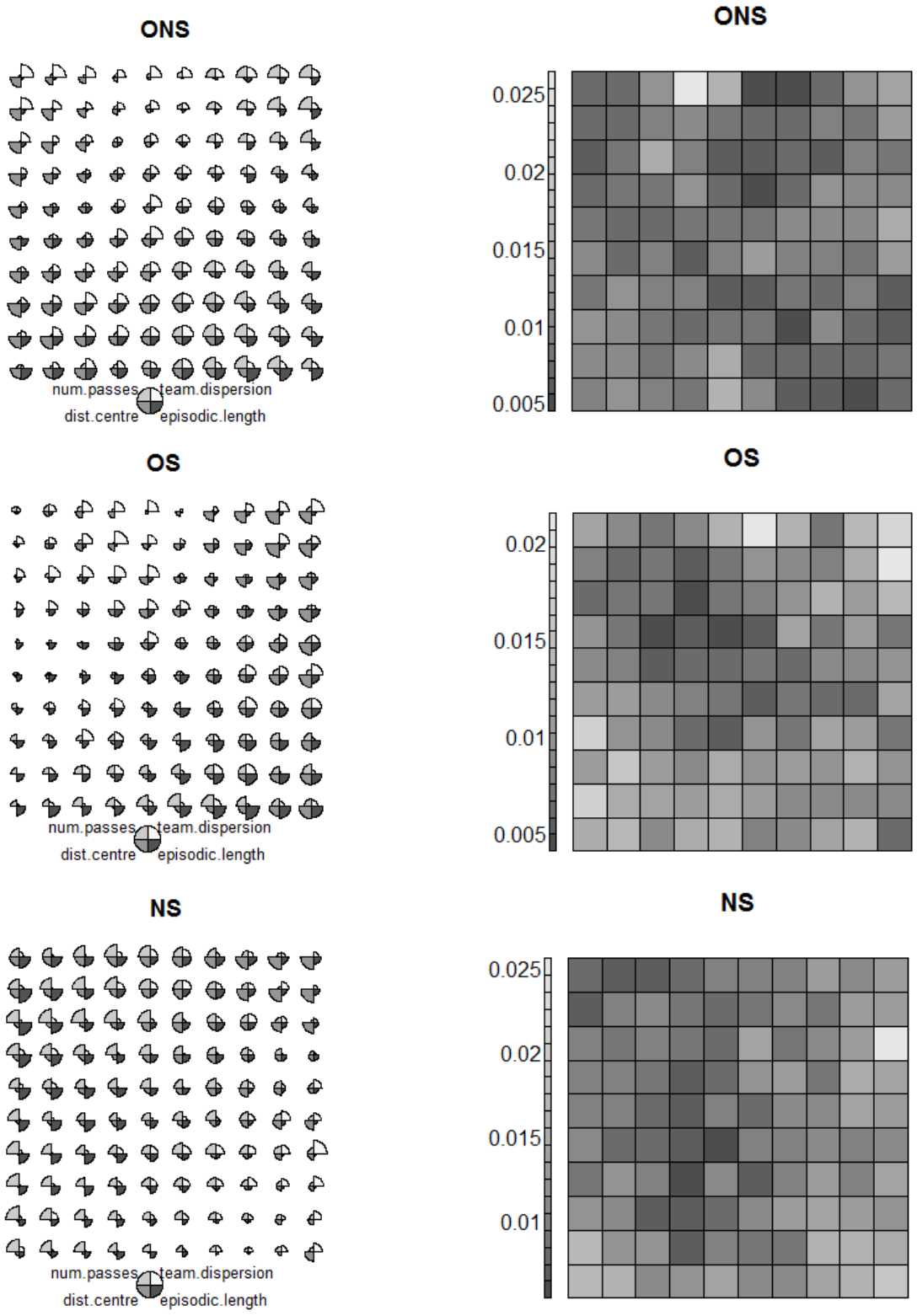


FIGURE 4.12: Kohonen Self-Organizing Maps (SOMs) representing the explored behavior search space in Keep-away 5vs4 for given HyperNEAT variants with behavior transfer. In each NE variant, left: shows codebook weight vector for data visualization and right: shows u-matrix representing boundaries between clusters. Each cirle in the codebook weight vector represents a behavior type (four slices). Each slice represents one component of the behavior characterization vector. The differing u-matrix Y-axis values indicates normalized behavior distances.

of 500 statistical comparisons) yielded a significantly higher task performance over the same methods with behavior transfer. Average efficiency comparisons yielded similar results with 488 out of 500 statistical comparisons indicating that methods that evolved collective behaviors with behavior transfer yielded a significantly higher efficiency compared to the same methods without behavior transfer. However, for all method variant comparisons, HyperNEAT overall evolved significantly higher quality collective behaviors. That is, 242 out of 250 method comparisons indicated that HyperNEAT yielded a significant higher average maximum task performance. Similarly, 248 out of 250 method comparisons indicated HyperNEAT yielded significantly higher average efficiency in evolved behaviors.

HyperNEAT superiority over NEAT is ascribed to the fact that a policy (encoded by CPPN) evolved in the source task is not altered when transferred to the target task for further evolution (Verbancsics and Stanley, 2010). Given the source task is similar to the target task, and that transfer is between task of increasing complexity, based on HyperNEAT-BEV is able to represent tasks in the same substrate network with additional agent positions drawn on the substrate geometry (section 3.2.1). The policy encoded by CPPN, expressed as a function of the task geometry does not change when transferred to another task. However, for NEAT the policy is altered to reflect the new task information (section 3.1.1). Some of the information useful for evolution is lost in the process, as shown in the previous work by Verbancsics and Stanley (2010) which attributes the poor performance of NEAT compared to HyperNEAT for a keep-away task.

This is the first time that collective behaviors evolved with other search variants besides objective-based search (for example, ONS, NS, GNS and OGN) have been used in company with policy (behavior) transfer. The key result is that most task performance and efficiency benefits are gained by using ONS together with policy transfer.

4.6 Summary and Conclusion

In this chapter, we demonstrated behavioral diversity maintenance (given behavior transfer) is an effective means to evolve collective (keep-away) behaviors that perform well given increasing task complexity. The key result was that the HyperNEAT ONS variant consistently outperformed traditional objective-based search (OS) and genotypic diversity (GNS and OGN) variants across task of increasing complexity (that is, 4vs3, 5vs3, 5vs4, 6vs4 and 6vs5). These comparisons also highlighted the overall effectiveness and efficiency of behavior transfer, where evolution starts with a source task to derive behaviors that are transferred to tasks of increasing complexity for further evolution.

In the following chapter, these NE approaches are compared with reinforcement learning, a traditional method used for evolving controllers for collective behavior tasks.

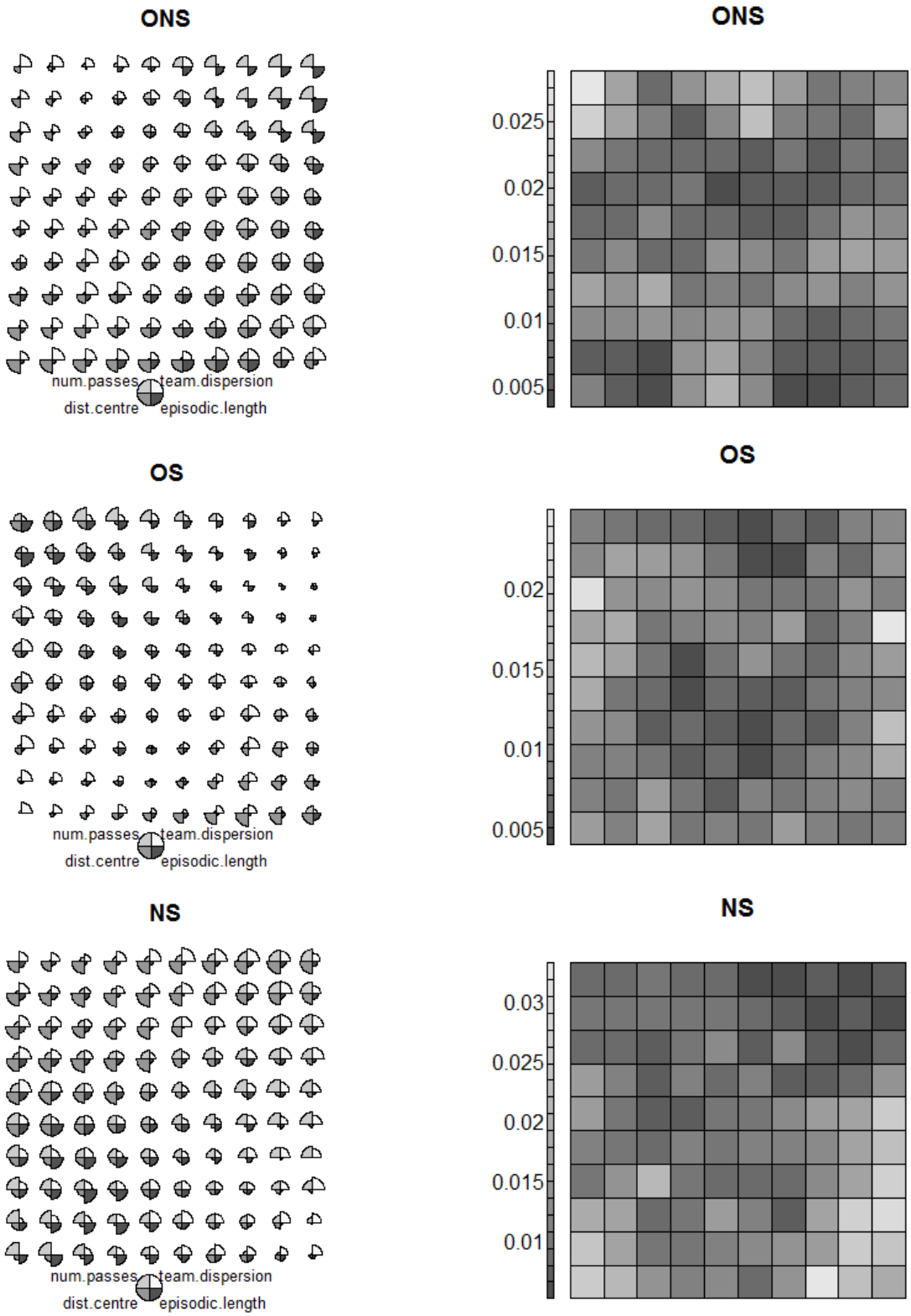


FIGURE 4.13: Kohonen Self-Organizing Maps (SOMs) representing the explored behavior search space in Keep-away 6vs4 for given HyperNEAT variants with behavior transfer. In each NE variant, left: shows codebook weight vector for data visualization and right: shows u-matrix representing boundaries between clusters. Each cirle in the codebook weight vector represents a behavior type (four slices). Each slice represents one component of the behavior characterization vector. The differing u-matrix Y-axis values indicates normalized behavior distances.

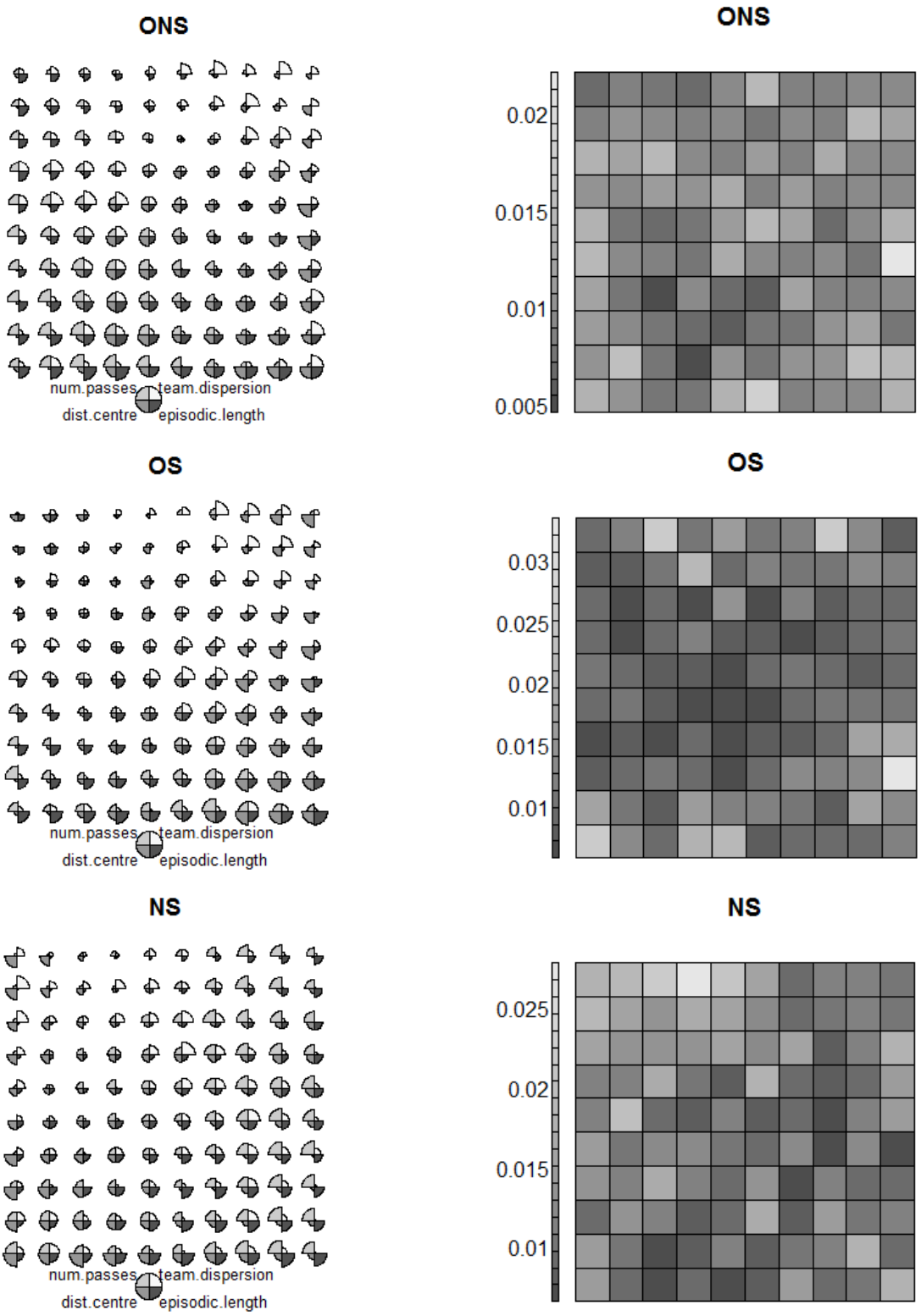


FIGURE 4.14: Kohonen Self-Organizing Maps (SOMs) representing the explored behavior search space in Keep-away 6vs5 for given HyperNEAT variants with behavior transfer. In each NE variant, left: shows codebook weight vector for data visualization and right: shows u-matrix representing boundaries between clusters. Each cirle in the codebook weight vector represents a behavior type (four slices). Each slice represents one component of the behavior characterization vector. The differing u-matrix Y-axis values indicates normalized behavior distances.

Chapter 5

Reinforcement Learning Experiments

This chapter evaluates the performance of reinforcement learning methods compared to neuro-evolution methods for policy (collective behavior adaptation) and transfer learning in collective behavior tasks. In particular, the chapter investigates the task performance of TD methods, that is SARSA (Sutton, 1998) and Q-Learning (Watkins, 1989), for collective behavior adaptation and policy transfer. The performance of TD methods (section 2.1.1) are compared with NEAT and HyperNEAT variants for five keep-away tasks (that is, *4vs3, 5vs3, 5vs4, 6vs4* and *6vs5*). This is to address this thesis research question (section 1.2):

- Investigate the efficacy of neuro-evolution methods for policy (collective behavior adaptation) and transfer learning in collective behavior systems compared to traditional well established Reinforcement Learning (RL) methods

The experiments in this chapter will first look at the performance (effectiveness and efficiency) comparison of SARSA (section 3.3.1) and Q-Learning (section 3.3.2) with and without behavior transfer for keep-away tasks of increasing complexity. Effectiveness is improved average task performance between tasks with and without behavior transfer. This measure indicates performance gained through the use of behavior transfer. Efficiency refers to average number of episodes to reach a task performance threshold, a comparison of collective behavior adaptation before and after behavior transfer. Secondly, the comparison of both TD methods to the neuro-evolution methods (section 3.1 and 3.2) collective behavior adaptation, given behavior transfer across tasks of increasing complexity.

5.1 Collective Behavior adaptation in Reinforcement Learning

Keep-away a subtask of RoboCup Soccer[1] (discussed in section 4.1), is used as a benchmark task in this chapter, to evaluate the performance of RL methods compared to NE methods discussed in section 3.1 and 3.2. The RoboCup soccer simulator operates in discrete time steps $(t = 0, 1, 2, ..., n)$, each time step representing $100ms$ of simulation

[1] https://sourceforge.net/projects/sserver/

time. Each player receives a signal every $150ms$ of simulation time that contains visual sensory information about the relative distances and angles of other objects in the environment. The players must sense and act asynchronously, and thus the action executed by each player is not a direct response to the visual perception. Every player is controlled by a separate process, keep-away provides heuristic method for controlling the taker players (algorithm 6 in section 4.1) and a RL method controls the keeper players, specifically, what action to take when the player is with the ball (section 4.1). If the keeper player is without the ball, a heuristic method is executed to move to an open position or dash for the ball (section 4.1) (Stone et al., 2005). This RL case study uses Keep-away soccer version 0.6[2].

5.2 Mapping Keep-Away soccer to Reinforcement learning

In this case study, the keep-away task is modeled as a Semi-Markov Decision Process (SMDP) (section 2.1.1). Since actions in the keep-away task can last more than one simulation time step and decisions are made only when the pending action terminates, it is convenient to model the problem as SMDP. The keeper agents make decisions at discrete SMDP time steps, and this model maps well to RL problem as it deals with partially observable environments (Yang and Francesca, 2014). In the SMDP model, the episode consists of a sequence of states, actions, and rewards as: $s_0, a_0, r_1, s_1, ..., a_n, r_n, s_n$, where s_i is the state at time t_i, a_i is the action executed by a player at time t_i, s_{i+1} is the resultant state after execution of action a_i (selected from a finite set of actions). Every such state transition yields an immediate reward $r_{i+1} = R(s_{t+1}|s_t, a_t)$. The state, s_n is the terminal state of each episode, attained when a player has lost ball possession to an opponent team or when the ball has gone out of region bounds.

Previous work (Whiteson and Stone, 2006) has indicated that there are various ways of calculating an immediate reward, r_i. The three most popular approaches used with keep-away are as follows. First, the immediate reward value of 1 was assigned for each time step t_i (that is, where keeper players have ball possession) and a value 0 for a time step, t_n where players lost possession. Whiteson and Stone (2006) suggested that the immediate reward obtained this way leads to the increase in exploitation. Second, a reward value of 0 for time step, t_i and -1 otherwise (Stone et al., 2001; Whiteson and Stone, 2006). Third, a reward, r_i is calculated as the difference between the current time step, t_i and the previous time step t_{i-1} (simulation time when last action was carried out) as depicted by equation 5.1 (Stone et al., 2005). Since keep-away is episodic, there is no discounting required to express the objective, the immediate reward is required to ensure that accumulatively the episode length could be maximized.

[2]https://github.com/sdidi/KeepawaySim

State Variables	Description
$dist(K_i, C), i \in [1, n_k]$	Distance of each keeper and the center of the field
$dist(T_i, C), i \in [1, n_t]$	Distance of each taker and the center of the field
$dist(K_b, K_i), i \in [2, n_k]$	Distance of Keeper 1 to other keepers
$dist(K_b, T_i), i \in [1, n_t]$	Distance of each taker to keeper 1
$min_{j \in [1,n_t]} dist(K_i, T_j), i \in [2, n_k]$	Distance of closest taker to keeper 1
$min_{j \in [1,n_t]} angle(K_{t1}, T_j), i \in [2, n_k]$	Smallest angle between a each keeper and the takers with vertex at keeper 1.
Actions	**Description**
Hold	Do not pass ball
Pass to $K_i, i \in [2, n_k]$	Pass to a teammate ($K_i, i \in [2, n_k]$)

TABLE 5.1: Keep-away state description and the finite set of actions, where n_t and n_k are the number of taker and keeper players, respectively.

$$r_i = \begin{cases} t_i - t_{i-1}, & s_i \neq s_n \\ t_{i-1} - t_i, & s_i = s_n \end{cases} \tag{5.1}$$

where s_i and s_n are the current state and the final state of an episode, respectively. This translates to a value of 1 and a value of -1 if the difference between subsequent time steps is equals 1. The second and third approach was shown to increase exploration, hence the third approach was chosen as it is supported by previous work on keep-away soccer (Stone et al., 2005; Whiteson and Stone, 2006).

The ultimate goal of the team (keepers) is to select a sequence of actions that could maximize the total long term reward, thus maximizing the episodic length. Keeper players can learn different policies or the same policy during training. Previous work (Whiteson et al., 2003) has demonstrated that keep-away can be played effectively with homogeneous teams (section 4.1), hence we chose to learn same policy for all keeper players. As in the previous case study, here we also consider five keep-away tasks (4vs3, 5vs3, 5vs4, 6vs4 and 6vs5) of increasing complexity. The purpose of the experiments in this chapter is to compare the performance of TD methods and NE methods (already discussed in chapter 4) with and without behavior transfer. Table 5.1 provides the description of state information used by the agents, for example, relative distances and angles to visible objects in the environment. The state space dimensionality of each keep-away task is shown in table 5.2, where features represents the number of continuous state variables.

5.2.1 Keep-away Task Complexity

The state space size of a keep-away task increases with the number of keeper and taker agents (table 5.2). Previous work indicated that the task complexity of the keep-away increases with the size of the state space (Stone et al., 2005; Taylor et al., 2005; Didi and

Task Configuration	State Features/Actions
$3vs2$	$13/3$
$4vs3$	$19/4$
$5vs3$	$23/5$
$5vs4$	$25/5$
$6vs4$	$29/6$
$6vs5$	$31/6$

TABLE 5.2: All features and actions available for players to process during training.

Nitschke, 2016a). This increase in task complexity is due to the following complex interaction dynamics. First, an increase in taker agents correlates to the difficulty of making successful passes. Second, an increase in keeper agents, given a fixed field size results in a crowded field and high probability for interference between keeper agents. Third, an increase in state space dimensionality necessitates increased computational complexity as more sensory inputs are processed.

Consider, KvsT keep-away task ($K \geq 3$, $T \geq 2$, where K and T denotes the number of keepers and takers, respectively), the complexity of a given keep-away task, x is given by equation 5.2.

$$Complexity(x) = \frac{T}{K} * OBJ \tag{5.2}$$

where, OBJ is the total number of dynamic objects in the field, K and T are the total numbers of keeper and taker agents, respectively. Given the collective behavior tasks in this thesis (table 5.2, 6vs5 keep-away, with complexity of 9.16 was considered to be more complex than the rest of the keep-away tasks. Thus, the keep-away tasks in order of complexity were: 4vs3, 5vs3, 5vs4, 6vs4 and 6vs5.

However, as discussed in section 4.1.1, complexity of a keep-away task, was not only attributed to the number of agents in the field (equation 5.2) but was also due to other properties. These include noisy sensors and actuators, players with a partial view and a highly stochastic environment (Ghavamzadeh et al., 2006). Previous work has suggested that these properties lead to an increase in task complexity (Ghavamzadeh et al., 2006; Busoniu et al., 2008) and deceptive fitness landscape (Gomes and Christensen, 2013a).

The following sections of this chapter will test the TD methods on tasks of increasing complexity (that is, 4vs3, 5vs3, 5vs4, 6vs4 and 6vs5) and compare the performance of TD methods to NEAT and HyperNEAT variants with and without behavior transfer.

5.2.2 Function Approximation and Keep-Away

The Keep-away domain is defined by a large continuous state space, that is represented by a Cartesian product of n state variables (that is, $S = S_1 \times S_2 \times ... \times S_n, S \subseteq \mathbb{R}^n$). A

value function approximation is used with RL to address learning in continuous state space and to manage the high state dimensionality as discussed in section 3.3.3. In the RL experiments in this thesis, we use TD methods with replacing eligibility traces (Stone et al., 2005) for learning and a single dimensional tile coding (Sutton, 1998) as a function approximator (section 3.3.3).

Linear tile coding with a single dimensional tiling is used to discretize the continuous state space. Supported by previous work (Stone et al., 2005), each state variable was covered by 32 tilings and each of the tilings with a tile width of $1/32$. Each tiling has a specified number of tiles (table 5.3), and a single tile represents a single feature of the state space. Therefore, only a single tile per tiling will be activated, that corresponds to a feature containing the current state in each tiling. A feature set, $\mathcal{F}_a$, consists of 32 active tiles per state variable and hence, a total of $13 \times 32 = 416$ active tiles for 13 state variables describing a 3vs2 keep-away task. The width of each tile was specified based on the level of state generalization required and supported by previous work (Stone et al., 2005; Fachantidis et al., 2011). The interval of state variables for a 3vs2 keep-away task for a 20×20 region is given in table 5.3. For example, state variables measuring distances were given a width of three meters and a width of 10 degrees for state variables measuring angles. In the case of 3vs2 keep-away, the state variable x_1 corresponded to the distance of a keeper (with a ball) to the center of the field, which was five tiles per tiling, and the distance of one player to another which had 10 tiles per tiling and angles between players had 18 tiles per tiling. Since each state variable has 32 tilings, the total number of tiles for each keep-away task (available to each action) are given by equation 5.3:

$$Sum_{tiles} = \sum_{i=1}^{n} N(x_i) \times 32 \tag{5.3}$$

where $N(x_i)$ is the number of tiles per tiling for the i^{th} state variable x_i and n is the total number of state variables. For example, for a *3vs2* keep-away, that has 13 state variables, there are 3872 tiles available for each action.

The current state value is computed from a feature set, $\mathcal{F}_a$, that consist of a list of active tiles, indicating which features are currently active in the state space. Each element of $\mathcal{F}_a$, is associated with a weight value, θ. An action value for action a, Q_a, in the current state is computed as the sum of weight vectors, θ^T, associated with active tiles, ϕ^T, in $\mathcal{F}_a$ (section 3.3.3 and equation 3.8).

5.3 Experimental Setup

The keep-away simulation runs on a 20×20 region. Each experiment[3] run was repeated 20 times and the task performance results of each method was averaged over 20

[3]Source code and executables found here: https://github.com/sdidi/KeepawaySim

Features	Interval	Tile width	Tiles per Tiling
dist(K_b, C)	[0,14.14]	3	5
dist(K_{t1}, C)	[0,14.14]	3	5
dist(K_{t2}, C)	[0,14.14]	3	5
dist$(T1, C)$	[0,14.14]	3	5
dist$(T2, C)$	[0,14.14]	3	5
dist(K_b, K_{t1})	[0,28.28]	3	10
dist(K_b, K_{t2})	[0,28.28]	3	10
dist$(K_b, T1)$	[0,28.28]	3	10
dist$(K_b, T2)$	[0,28.28]	3	10
$min_{j\in1,2} dist(K_{t1}, T_j)$	[0,28.28]	3	10
$min_{j\in1,2} dist(K_{t2}, T_j)$	[0,28.28]	3	10
$min_{j\in1,2} angle(K_{t1}, T_j)$	[0,180]	10	18
$min_{j\in1,2} angle(K_{t1}, T_j)$	[0,180]	10	18

TABLE 5.3: The number of tiles per Tiling for a *3vs2 keep-away task*.

independent runs. Each run ends after 15000 of episodes that lasted for about $15 - 20$ hours of simulation time. To be able to compare RL to NE methods, we had to compute the maximum episodes as: the number of maximum generations (that is, 100) multiplied by the size of the population (that is, 150), thus equivalent to 15000. The task performance plotted shows the mean episodic length averaged over 150 episodes (figure 5.1).

The RL experiments described in this chapter are using the scalar parameters shown in table 5.4, $\alpha = 0.125$, $\epsilon = 0.01$, $\gamma = 0.85$ and $\lambda = 0$ (exception of SMDP episode step that has $\lambda = 1$). Previous work (Stone et al., 2005) indicated that the fastest learning rate is obtained at $\alpha = 0.125$ and that $\epsilon = 0.01$ provides sufficient exploration without significantly affecting task performance. Previous work also demonstrated that the keep-away task performance results are not sensitive to varying λ and hence, the adoption of $\lambda = 0$ (Stone et al., 2005; Fachantidis et al., 2011).

5.4 Reinforcement learning and Collective behavior Transfer

To facilitate transfer of behavior from the source task to target task, a vector of weights θ_i^T, associated with each feature set is periodically stored in memory at the interval of 150 episodes (a number equivalent to a single neuro-evolution run). Then the weights obtained at the final episode of the source task are extracted to be transferred to boost learning in the target task. The approximate value function, $Q_a(s)$ is computed based on the sum of weights of all tiles in $\mathcal{F}_a$, as shown in equation 3.8 in section 3.3.3.

The transfer of behaviors from the source task (3vs2) to a target task (that is, 4vs3, 5vs3, 5vs4, 6vs4 and 6vs5), is handled by a policy that transforms behaviors (set of weights, θ_i) from one task to map to another task (section 3.3.4 and algorithm 5). This method transforms the final weights obtained from collective behavior adaptation in a source

Parameter	Symbol	Value	Simulation Parameters	Setting
Discount factor	γ	0.85	Number of Runs	20
Epsilon	ϵ	0.01	Maximum episodes per run	15 000
Learning rate	α	0.125	Maximum episodic length	18
Eligibility Trace	λ	0	Agent positions	Random
			Environment size	20 x 20 grid
			Agent speed (per iteration)	1 grid cell
			Ball speed (per iteration)	2 grid cells

TABLE 5.4: The values of the experiment and simulation parameters

task, so that the adapted behavior can be used to initialize learning in the target task. Previous research has applied two approaches for behavior transfer. The first approach, initializes learning in the target task by copying weight vector of state features that correspond to those the source task. Zero weight values are then given to additional features that are not present in the source task.

The second approach copies the weights that corresponds to the source task to initialize learning in the target task with no modification. Based on functional relationship between additional state features and state features in the source task, weights are copied to the additional state features that are functionally related to those in the source task. For example, weights from the source task for $dist(K_3, K_1)$ and action a_i, can be copied to a new state feature that corresponds to $dist(K_4, K_1)$ and action a_i, in the target task. Similarly, weights for a state feature that corresponds to an angle $min_{j\in[1,2]}angle(K_3, T_j)$ in the source task, was copied to that of an angle $min_{j\in[1,3]}angle(K_4, T_j)$ in the target task. Supported by previous work (Stone et al., 2005) that indicated a relatively high task performance was obtained when policy transfer was based on the second method compared the first method. Hence, adoption of the second method of behavior transfer mapping for the experiments in this case study.

The source task was trained for 4500 episodes, a value equivalent to the 30 generations of evolution of the source task for NE methods (NEAT and HyperNEAT). That is, $30 \times 150 = 4500$, a product of a number of generations and population size, respectively. The policy obtained (that is, a vector of weights) after training in the source task initializes the behavior adaptation (reinforcement learning) in the target task. The transfer mapping is based on a policy that is provided in section 3.3.4, so that the transferred behaviors can be beneficial to the new task with a different state representation.

5.5 Results Discussion

To detect significant differences in task performance of RL methods with and without behavior transfer, we used Mann-Whitney test with 95% confidence. We performed a pairwise test between SARSA and Q-Learning at the early stages of training and at the

terminal point (that is, episode number 15000). Overall results show that the task performance is significantly better when trained with behavior transfer ($p < 0.05$) than training from scratch. However, this benefit gradually decreases at the later stages of training were the behavior of each method with and without behavior transfer is not significantly different due to early convergence.

The efficacy of each method is measured according to effectiveness and efficiency. The effectiveness of each TD method is measured as the average task performance of evolved behaviors, evaluated progressively throughout behavior adaptation and at the final episode. Efficiency is the average number of episodes taken by evolved behaviors to reach minimum threshold. The equivalent definitions of effectiveness and efficiency were used in the NE case study. The following section discusses the performance of each method based on Effectiveness.

5.5.1 Effectiveness

To evaluate the effectiveness of each TD method we compared the task performance of each method in collective behavior task of increasing complexity with and without behavior transfer. In this thesis, task performance is a measure of keep-away team's capability to maintain the possession of the ball away from the taker agents. That is, task performance was calculated as the total time where the keeper managed to maintain possession of the ball, normalized into a range: [0,1] and averaged over all 20 independent runs. Normalization was done with respect to the average maximum episodic length (table 5.4), that is applied to each task.

Figure 5.1 shows the mean normalized task performance progression averaged over 20 independent runs for SARSA(λ) versus Q-Learning(λ) with and without behavior transfer for all keep-away tasks. The y-axis represents the normalized average time that the keepers are able to maintain control of the ball (that is, episodic length). The x-axis is the number of episodes with a scale of $1 : 150$ (that is, each unit represents 150 episodes and with maximum episode number of 15000).

The task performance progression results demonstrate the superiority of SARSA over the Q-Learning reinforcement method for all keep-away tasks. For example, for 5vs4 keep-away, SARSA(λ) teams yielded a task performance that increased steadily over the first 4500 episodes and attained the highest task overall performance. Whereas, on a comparative rate of task performance increases was observed for Q-Learning between episodes 6000 and 7500. Similar results are observed across all tested keep-away tasks where SARSA shows high task performance than Q-Learning. This confirms the previous work (Stone et al., 2005) that indicated that SARSA converges faster than Q-Learning and that SARSA attains higher task performance compared to Q-Learning given the same training duration. For example, in 4vs3 keep-away, for SARSA(λ) without behavior transfer there is a steep gradient of task performance increase in the

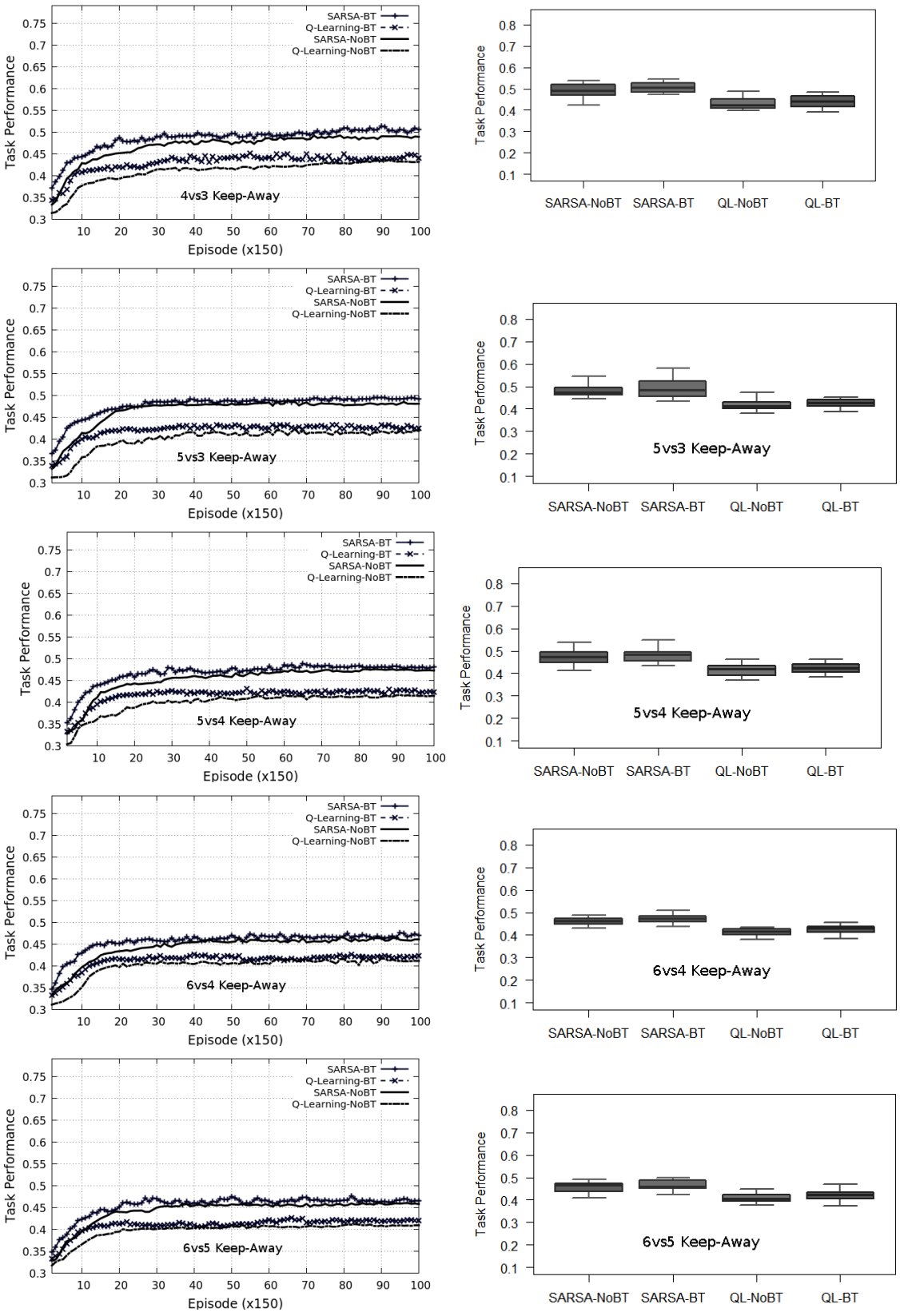


FIGURE 5.1: Task performance progression graphs and boxplots. Left: Average (over 20 runs) task performance progression for each target keep-away task. Right: Boxplot showing the average task performance for comparative methods at the final episode of each of the keep-away task.

first 3000 episodes, followed by a gentle gradient of task performance increase from there on until 6000 episodes where task performance plateaus until the last episode. This was compared to that of Q-Learning(λ) where relatively low gradient of task performance increase was observed, with exception of the first 1500 episodes that recorded a fairly high gradient. Interestingly, for 5vs4 keep-away as with other keep-away tasks, Q-Learning(λ) without behavior transfer takes more time (based on number of RL episodes) to converge compared to SARSA(λ). These observations support the previous results that indicated that while Q-Learning converges to optimal policy under restrictive conditions (Watkins, 1989), it can be unstable with linear function approximation (Stone et al., 2005).

Figure 5.1 also shows a graph for each method with behavior transfer starting at a higher task performance value with behavior transfer compared to that yielded for each TD method without behavior transfer. This confirms that each TD method benefits from behavior transfer between the source task (Keep-away 3 vs 2) and each of the five target tasks (4vs3, 5vs3, 5vs4, 6vs4 and 6vs5). However, the task performance benefit is observed in the early stages of adaptation, task performance difference between adaptation with and without behavior transfer is observed to gradually decrease after about 7500 episodes of learning, whereby the task performance with behavior transfer is comparable to that without behavior transfer.

5.5.2 Efficiency of TD methods

To evaluate efficiency of the two TD methods we compared the time (based on number of episodes) to reach threshold task performance. The comparison of search efficiency of SARSA(λ) and Q-Learning(λ) with and without behavior transfer is exhibited in figure 5.5, given a task performance threshold value. For example, 4vs3 keep-away, SARSA(λ) reaches a task performance threshold value of 0.418 at 900 and 1500 episodes of evaluation with and without behavior transfer, compared to 2850 and 5250 episodes for Q-Learning(λ) with and without behavior transfer, respectively. Whereas, 6vs5 keep-away, SARSA(λ) reaches a task performance threshold value of 0.405 at 1200 and 1800 episodes of evaluation with and without behavior transfer, compared to 2100 and 6300 episodes for Q-Learning(λ) with and without behavior transfer, respectively. These results show that SARSA(λ) behavior adaptation method is more efficient for adapting behaviors for keep-away tasks compared to Q-Learning(λ) method. These results support the previous work by Taylor et al. (Taylor et al., 2007b), that demonstrated relative superiority of SARSA(λ) for non-deterministic keep-away task compared to Q-Learning(λ).

Figure 5.1 shows that task performance for each TD method degrades with an increase in the keep-away task complexity. For example, 4vs3 keep-away has relatively higher task performance for both TD methods compared to that of 6vs5 keep-away (where, 6vs5 is considered to be more complex than 4vs3). Previous work (Zaera et al., 1996),

Task	Performance Threshold	Q-Learning Search Efficiency Episodes x 150		SARSA Search Efficiency Episodes x 150	
		No BT	BT	No BT	BT
4 vs 3	0.418	35	19	10	6
5 vs 3	0.415	38	16	12	6
5 vs 4	0.412	43	16	11	7
6 vs 4	0.409	42	15	12	8
6 vs 5	0.405	42	14	12	8

TABLE 5.5: Efficiency comparison of SARSA versus Q-Learning variants with Behavior Transfer (BT) and No Behavior Transfer (No BT). Search Efficiency: Average number of episodes to reach the task performance threshold for a given a keep-away task.

indicated that task complexity correlates with deception in task domain. Some RL methods have mechanism for balancing a trade-off between exploration and exploitation which is necessary to overcome the search difficulty induced by task complexity.

Two commonly used strategies that provide a balance between exploration of action space and exploitation of known actions are ε-greedy (Whitehead and Ballard, 1991) and Boltzmann policy (Watkins, 1989). A ε-greedy method has been used with SARSA in this thesis (section 2.1.1 and equation 2.6). That is, based on ε-greedy method, the best action is selected with some probability of ε and with probability of $1 - \varepsilon$ a random action is selected. Supported by previous work (Stone et al., 2005), a low value of ε value 0.01 was set to encourage exploration (table 5.4). Q-Learning always select an optimal action with the highest Q-value and hence does not explicitly support action space exploration in the action space. Taylor et al. (2007), suggested that the relatively poor task performance of Q-Learning compared to SARSA for non-deterministic tasks was partly due to the fact that it does not explicitly encourage action space exploration.

In summary, the results obtained from this case study and supported by previous work, suggests that task complexity does affect the performance (in terms efficiency and effectiveness) of TD methods, as much as it had an impact on NE objective-based search methods (Stone et al., 2005; Taylor et al., 2007b; Gomes and Christensen, 2013a; Didi and Nitschke, 2016a). Hence, this indicates that TD methods are not appropriate as task complexity increases in keep-away task.

5.5.3 Reinforcement Learning Behavior Transfer

To analyze the benefits of behavior transfer based on the results shown in figure 5.1 and table 5.5 we adopt some of the behavior transfer measuring metrics from previous work (Taylor et al., 2007b; Torrey and Shavlik, 2009). Taylor et al. (2007) suggested five possible ways of measuring the benefits of behavior transfer:

- *The asymptotic performance* - the difference in performance at the final episode.

- *The total reward* - the area under a learning curve.
- *The reward area ratio* - the ratio of the area under a curve with behavior transfer to the area under a curve without behavior transfer.
- *The time to threshold* - the average time to reach the performance threshold.
- *The jump-start* - the initial task performance.

Related to this, Torrey and Shavlik (2009) suggested: *higher slope* (the gradient of the task performance curve), *higher asymptote* (similar to asymptotic performance) and *higher start* (similar to jump-start). Since these behavior transfer measuring metrics are similar we adopt both sets of metrics for our results analysis.

Figure 5.1 shows the behavior transfer significantly improves performance in terms of the jump-start metric, that is, time to the task performance threshold given a higher gradient of task performance increase for SARSA(λ) and Q-Learning(λ) across all tasks. However, there is less of total reward and asymptotic performance. This is partly due to convergence and that at the later stages of learning in complex environments learning slows down. This is evidenced by the fall in the gradient of the learning curve that is observed in SARSA(λ) and Q-Learning(λ) task performance progression graphs (figure 5.1).

RL Methods	**Keep-Away Task (Percentage Performance Gain)**				
	4vs3	**5vs3**	**5vs4**	**6vs4**	**6vs5**
SARSA(λ)	9.17%	8.85%	7.71%	7.21%	7.07%
Q-Learning(λ)	10.68%	8.67%	9.21%	7.35%	7.69%

TABLE 5.6: Reinforcement Learning behavior transfer performance gain.

The *jump-start* is the average initial task performance observed on the target task progression graphs, when behavior transferred is used to initialize collective behavior adaptation (left side of figure 5.1). For example keep-away *4vs3*, shows SARSA(λ) with behavior transfer starting at a normalized task performance of 0.351 and Q-Learning(λ) at 0.338, compared to 0.322 and 0.306 of SARSA(λ) and Q-Learning(λ) without behavior transfer, respectively. The average percentage of this task performance benefit (equation 5.4) is 9.2% and 10.7% for SARSA(λ) and Q-Learning(λ), respectively. For 6vs5 keep-away, the average percentage benefit of behavior transfer in terms of jump-start is 7.1% and 7.7% for SARSA(λ) and Q-Learning(λ), respectively. Similar results were observed for all other keep-away tasks (table 5.6). This benefit is due to the effective transfer of adapted behaviors (that is, the set of weights θ^T) from previous training. It is interesting to observe that even though SARSA(λ) has a higher task performance value compared to Q-Learning(λ), both methods attained comparable benefit from behavior transfer, based on the jump-start metric.

$$Benefit_{jumpstart} = \frac{TP_{BT} - TP_{NoBT}}{TP_{NoBT}} \tag{5.4}$$

where TP_{BT} and TP_{NoBT} are the normalized task performance of each method with and without behavior transfer, respectively.

The *time to threshold* metric (termed efficiency in section 5.5.3 and table 5.5) is the average number of episodes to reach a minimum performance threshold. This threshold value is the maximum task performance (that is, the peak value of the task performance curve) obtained by Q-Learning(λ) without behavior transfer. There is a significant improvement in the efficiency based on this metric (table 5.5), for example for keep-away 4vs3, for Q-Learning(λ) without behavior transfer it takes 5250 episodes to reach the performance threshold of 0.418, compared to 2850 episodes with behavior transfer, 1500 and 900 for SARSA(λ) with and without behavior transfer, respectively.

5.5.4 Neuro-Evolution versus Reinforcement Learning

Figures 5.2 and 5.3 shows the comparison of TD methods to NE methods (NEAT and HyperNEAT) for all variants and for all keep-away tasks. The keep-away tasks tested are of high dimensionality where the number of dimensions increases with the keep-away tasks (table 5.2). HyperNEAT indirect encoding with built in capability to handle high dimensional state space gives it ability to learn large networks. Overall HyperNEAT variants yield the highest task performance, followed by NEAT, SARSA(λ) and then Q-Learning(λ) (figure 5.2 and 5.3). Neuro-evolution methods, specifically HyperNEAT obtained the highest task performance for all keep-away tasks. ONS variant of HyperNEAT recorded the highest task performance with and without behavior transfer. Most interestingly, in the comparison between the NEAT, HyperNEAT and RL methods (after 100 generations) the ONS variant of HyperNEAT in all the tasks (4vs3, 5vs3, 5vs4, 6vs4 and 6vs5) has relatively steep gradient of task performance increase than the rest of the other methods. We postulate that the good task performance is due to the ability of ONS to balance the trade off between exploration of the behavioral space and exploitation of behaviors that leads to high average fitness (section 4.5.4). As an example of this average task performance, the ONS variant of HyperNEAT produced average normalized task performance of 0.736 with behavior transfer compared to 0.516 for NEAT and 0.5 with SARSA(λ), given behavior transfer.

In addition to the discussion in section 4.5.4, previous work demonstrated that keep-away is an example of a *fractured* task domain as it possess a *fractured* decision space (Kohl and Miikkulainen, 2008; Grabkovsky et al., 2011). A fractured task domain is described as a domain whereby the optimal action for the agent to perform differs abruptly, rather than slowly and gradually between neighboring states in the state space. The same research demonstrated that it is difficult for most neuro-evolutionary methods to deal with rapid discontinuity in the decision space (Kohl and Miikkulainen, 2008). This suggests the relatively poor task performance of NEAT as a method for keep-away task adaptation compared to HyperNEAT, is due to the fact that NEAT uses a direct encoding representation that makes it difficult to represent abrupt decision

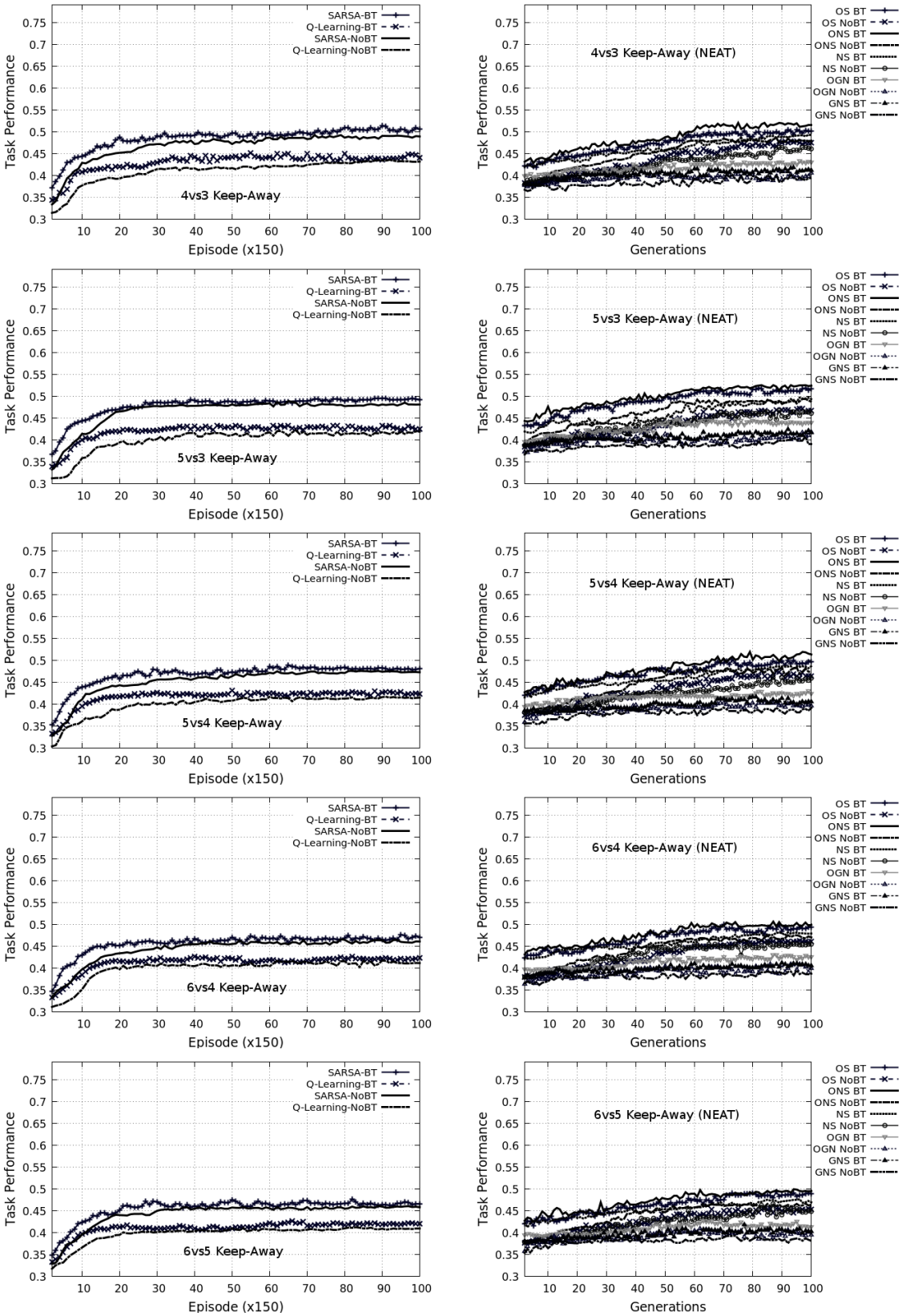


FIGURE 5.2: RL versus NE method average task performance comparison for all keep-away tasks. Left: SARSA and Q-Learning (TD methods), Right: NEAT (NE method).

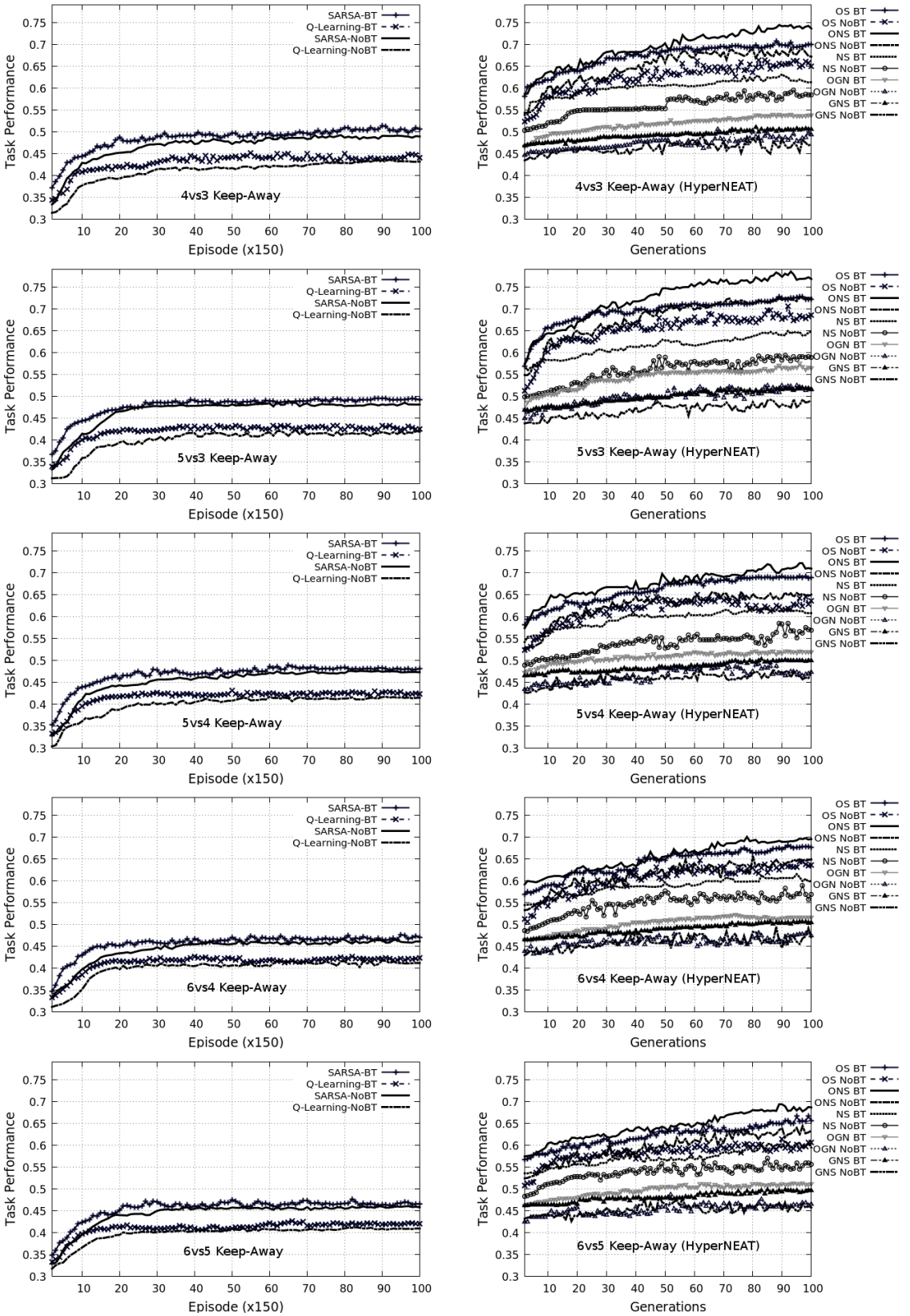


FIGURE 5.3: RL versus NE method average task performance comparison for all keep-away tasks. Left: SARSA and Q-Learning (TD methods), Right: HyperNEAT (NE method).

boundaries. However, HyperNEAT that evolves connectivity related to task geometry has the ability to directly exploit the task geometry. Grabkovsky et al. (2011), suggested that fractured problems are better solved by a method that can map the genotype representation directly to the domain space, so that the geometry of the task domain can be exploited. This suggests that HyperNEAT has the capacity to capture and exploit geometric information and thus improve the task performance for fractured domains (such as keep-away task).

Another explanation of relatively poor performance of NEAT compared to HyperNEAT (section 5.2) for all its variants across all tasks, is that NEAT needs more evaluations for evolving optimal behaviors for keep-away tasks of increasing complexity (Taylor et al., 2006a). Thus, NEAT requires more generations to reach an optimal solution and is unable to exploit regularities in the domain space. HyperNEAT adapts agent behavior especially well in large domains where there is regularity in the domain space, however its performance decreases on irregular problems due to its bias towards producing regularities (Clune et al., 2011). Therefore relatively good solutions are found with fewer generations than NEAT for high-dimensional problems with high degree of regularity, such as the keep-away (Verbancsics and Stanley, 2010).

The type of keep-away task we adopted for these experiments is the standard benchmark task, that is partially observable (section 4.1). The ONS and OS variants of NEAT have a comparable performance to SARSA(λ) at the early stages of evaluation and outperforms SARSA(λ) at the later stages of evolution. SARSA(λ) converges faster than NEAT but with relatively low average task performance across all task (figure 5.2). For example, for 4vs3 keep-away, SARSA(λ) reaches a maximum steady performance after an average of 6000 evaluations (equivalent to 40 generations of the ONS variant of NEAT) and it takes ONS variant of NEAT 60 generations to attain the same level of task performance. However, the ONS and OS variant of NEAT shows a steady growth in performance across the generations compared to SARSA(λ) that reaches a task performance plateau rapidly, across all the keep-away tasks. These results indicate that even though SARSA(λ) converges to a solution in fewer episodes, NEAT finds better (higher average task performance) with more genotype evaluations.

Thus, the comparison of NE methods to TD methods (given behavior transfer), show that both TD and NE methods (all variants) have a relatively good jump-start and time to threshold. TD methods have a relatively higher slope of task performance increase at the early stages of behavior adaptation compared to all NE method variants. However, TD methods have less total reward and asymptotic performance compared to NE methods (specifically, ONS, OS and NS variants). The highest benefits (in terms of average task performance) yielded by the ONS variant of HyperNEAT method. The relatively poor average task performance of RL methods compared to HyperNEAT (given the behavior transfer), is attributed to the the fact that TD methods learns well when the domain is fully observable and have difficulty learning in domains that are partially observable such as the keep-away (Taylor and Stone, 2009).

5.6 Summary and Conclusion

In this chapter we have compared the task performance of neuro-evolution methods with the TD methods for adapting controllers for keep-away tasks of increasing complexity. First, the TD methods (SARSA(λ) and Q-Learning(λ)), were compared based on effectiveness and efficiency of adapted behaviors with and without behavior transfer. The statistical analysis results indicated that SARSA(λ) outperforms Q-Learning(λ) with and without behavior transfer ($p < 0.05$, Mann-Whitney test) based on both efficiency and effectiveness. These results supported previous work, and indicated that Q-Learning(λ) is unstable with linear functional approximation (Stone et al., 2005) and that SARSA(λ) learns more effective (higher task performance) policies than Q-Learning(λ) in non-deterministic tasks (Taylor et al., 2007a).

Second, the task performance for each TD method was compared across keep-away task of increasing complexity. The results show that as complexity increases, task performance for both SARSA(λ) and Q-Learning(λ) degrades. This indicates that TD methods are not appropriate as task complexity increases.

The TD methods, were further compared to variants of NEAT and HyperNEAT (that is, OS, ONS, NS, OGN and GNS) with and without behavior transfer. The results demonstrated that the ONS variant of HyperNEAT performs much better (with respect to effectiveness and efficiency) than both TD methods and all variants of NEAT. Specific evolutionary search methods to direct NE such as behavior diversity maintenance and the hybrid approach, work most effectively at balancing exploration versus exploitation in the search space, more so than TD methods.

Comparing the best performing evolutionary search variant of NEAT and SARSA(λ) with behavior transfer it has been noted SARSA(λ) has relative higher task performance at early training stages than NEAT, and that NEAT outperforms SARSA(λ) at the later stages of evolution. This indicates that even though SARSA(λ) learns faster, given more evaluations NEAT is able to discover much better (higher task performance) solutions than SARSA.

Overall comparisons of each method with and without behavior transfer indicated that behavior transfer boosted collective behavior task performance for both NE and TD methods in task of increasing complexity. Evaluating the benefits of behavior transfer in both TD methods and NE variants, it shows there is task performance gain realized by each method through the transfer of learned and evolved behaviors, respectively. However, results showed that the behavior transfer benefit (task performance gain) in TD methods is high at the early stages of training but gradually decreases as the learning progresses and adaptation benefits of NE methods is realized for longer during artificial evolution. Specifically, the highest task performance solutions (for all tasks) was achieved by indirect encoding NE methods directed by behavior diversity maintenance.

To the best of our knowledge this thesis was the first work to test RL versus NE methods in tasks of increasing complexity. The following chapter presents the overall results discussion and possible future extensions to this thesis work.

Chapter 6

Discussion

The thesis research objective was to elucidate the essential features constituting effective and efficient evolutionary search that when coupled with behavior transfer are most suited to solving increasingly complex collective behavior (keep-away) tasks (section 1.2). We investigated five evolutionary search methods to ascertain the appropriate approach to evolve controllers for collective behavior (keep-away) tasks of increasing complexity. Experimental comparisons included behavior evolution with and without behavior transfer, where behavior transfer involved the transfer of behaviors evolved in a source task for further evolution in relatively complex target tasks.

Evolutionary search approaches investigated were objective-based search (OS), novelty search (NS), genotypic diversity search (GNS), hybrid of objective and novelty search and hybrid of objective based and genotypic diversity maintenance search (ONS and OGN, respectively). In this thesis, three methodological features were explored to ascertain an appropriate combination that enables the evolution of high quality solutions based on effectiveness (task performance) and efficiency (speed of adaptation) of evolved behaviors. These features are as follows: First, direct versus indirect encoding neuro-evolution methods for collective behavior evolution (that is, NEAT and HyperNEAT, respectively). Second, non-objective evolutionary search versus objective based search approach for guiding collective behavior evolution. Third, neuro-evolution with collective behavior transfer. Evolutionary methods derived from various combinations of these methodological features were compared to reinforcement learning methods. RL methods were selected since they are well-established and traditionally used for behavior transfer, and thus constitute a task performance benchmark. Each of these methodological features and RL methods were evaluated on collective behavior (keep-away) tasks of increasing complexity.

The main goal of this thesis was to establish the most appropriate neuro-evolution search method to evolve effective and efficient behaviors coupled with behavior transfer to mitigate the *bootstrap* problem (Kawai et al., 2001) for collective behavior tasks of increasing complexity. Collective behavior case study in this thesis was RoboCup keep-away. Results analysis indicated that a hybrid approach that directs evolutionary

search with respect to an objective based search and novelty search coupled with behavior transfer evolved relatively high-quality solutions.

The following sections discuss the impact of each methodological feature in defining evolutionary methods, where the effectiveness and efficiency of such evolutionary methods are compared to benchmark RL methods.

6.1 Benefits of Neuro-evolution and Behavior transfer

As an attempt to evaluate the efficacy of a proposed approach, five variants of NEAT and HyperNEAT (with and without behavior transfer) were applied to keep-away tasks of varying complexity. The results indicated relatively high task performance was obtained with HyperNEAT, especially when this method was used in company with behavior transfer. The results also support related work that demonstrated the efficacy of HyperNEAT when used with behavior transfer compared to NEAT in predator-prey domain and less complex keep-away task (Verbancsics and Stanley, 2010; D'Ambrosio et al., 2011). As in related work by Verbancsics and Stanley (2010), this thesis used HyperNEAT-BEV and similar benefits are highlighted for evolved behaviors transferred across tasks. The key advantage of HyperNEAT-BEV was that the geometric relationships encoded in evolved CPPNs are extrapolated for varying task complexity. Thus, evolved behaviors from the source task in the form of connectivity patterns encoded in evolved CPPNs are readily transferable across task without modifications.

In the case of NEAT, such behavioral extrapolations are not possible due to a need of scaling up sensory inputs to account for an increase in the number of agents. That is, to effectively transfer evolved behaviors, a hand-coded mapping function was required to transform the evolved behaviors to enable transfer between a source and a target task. Furthermore, previous work by Grabkovsky et al. (2011) suggested that the relatively low task performance of NEAT compared to HyperNEAT for a keep-away task was due to its inability to deal with *fractured* domains (Kohl and Miikkulainen, 2008). In this thesis, we used a heuristic method together with a transfer function to keep the number of sensory inputs low and to simplify the transfer of controllers between tasks (section 4.2).

Keep-away soccer has been shown to possess a fractured decision space, whereby the optimal action an agent selects changes abruptly as the agent moves from one state to a neighboring state. Some direct encoding neuro-evolution methods (such as NEAT using objective-based search), have difficulty representing such abrupt decision boundaries and dealing with rapid discontinuity in the decision space. Kohl and Miikkulainen (2009) suggested that, in order to perform well in a task with a fractured decision space, an evolutionary search method must be able to generate representations that capture local features of a task. Grabkovsky et al. (2011) indicated that a method that can map the genotype representation directly to the task geometry (such as observed in HyperNEAT) improves evolution in fractured domains.

It is also important to note that due to the nature of its algorithm, NEAT needs more evaluations for evolving optimal behaviors for keep-away tasks of increasing complexity (Taylor et al., 2006a). In comparison, it has been shown that HyperNEAT was able to generate optimal behaviors given relatively few generations (section 5.5.4). However, fracture and dimensionality are just two types of complexity inherent to complex multi-agent tasks (such as keep-away soccer). The following section discusses issues leading to significantly different evolved behavior performance given various versions of NEAT and HyperNEAT applied in keep-away soccer.

6.2 Benefits of Objective versus Non-Objective-based Search

There are two main properties of evolutionary search approaches that drives the search process (Eiben and Schippers, 1998). That is, exploration of the search space and exploitation of highly fit regions. Exploration of the search space was provided by either genotype diversity or behavior diversity, or a combination of both of these approaches. Based on task performance for all variants of NEAT and HyperNEAT, it was observed that the task performance of behaviors evolved with the objective-based search was relatively low compared to the task performance of behaviors evolved with the hybrid of objective-based search and behavioral diversity maintenance search. Task performance results indicated that there were significant differences between the performance of behaviors evolved with a hybrid of behavioral diversity maintenance and objective-based search, compared with other evolutionary search methods (section 4.5).

Overall, the task performance results (section 4.5) indicated that genotypic diversity maintenance technique for NEAT and HyperNEAT was not adequate to address the *bootstrap* problem in collective behavior tasks such as keep-away. Abandoning fitness based search for novelty only has been shown to be not effective for evolving effective behaviors for keep-away tasks of increasing complexity. As demonstrated in section 4.5, maintaining a good balance between behavior diversity maintenance in evolved solutions and providing a bias towards highly fit solutions (through objective based search) was beneficial to the task performance for each evolutionary search method, given collective behavior tasks with increasing complexity. Objective-based search fails to evolve high quality solutions as the task complexity increases as shown by each version of the keep-away task and statistical tests.

Previous work has demonstrated that novelty search has the ability to leverage artificial evolution via broader exploration of the search space when the domain has deceptive fitness landscapes (Lehman and Stanley, 2008). While behavior diversity maintenance methods have been tested on deceptive tasks (Lehman and Stanley, 2008). There has been little research that comprehensively evaluates novelty search on increasingly complex (but not deceptive) multi-agent tasks such as keep-away (Gomes and

Christensen, 2013a). Unlike, the previous novelty search work deception is not the focus of this thesis. An idea behind using the hybrid approach for this thesis was to compliment the effort of objective based search by integrating a mechanism that promote solution diversity based on domain specific behavioral properties. However, since novelty search is not directed by a fitness objective, there is no bias towards fitness optimization, which may result in poor task performance (section 4.5). Thus, there is more to discovering high-quality solutions than just exploration, some previous research (Cuccu and Gomez., 2011; Gomes et al., 2015), has shown that NS performs poorly on very high dimensional and fractured search spaces (section 4.5). Hence, it is necessary to provide a balance between solution space exploration induced by novelty search and exploitation directed by fitness based search. This hybrid method (behavior diversity maintenance and objective based search, ONS) evolves collective behavior that yields relatively high performance (in terms of behavior effectiveness and efficiency) across all keep-away tasks tested in this thesis (section 4.5). Comparative results from collective behavior evolution experiments (section 4.5) demonstrated that for some keep-away tasks, ONS yields a significantly higher task performance compared to NS, indicating that NS is not appropriate for complex multi-agent tasks such as RoboCup keep-away. Thus, results indicate that a combination of objective and novelty based search is needed as multi-agent task complexity increases. The following section looks at the impact of reinforcement learning versus neuro-evolution on collective behavior evolution.

6.3 Reinforcement Learning versus Neuro-evolution

Most work on behavior transfer for multi-agent control tasks in machine learning has used reinforcement learning (RL), with few exceptions (Verbancsics and Stanley, 2010). The evolutionary search method variants of NEAT and HyperNEAT proposed in this thesis, were compared to temporal difference (TD) methods. The statistical analysis results given in section 5.5, demonstrated that the hybrid novelty search and objective based search (ONS) variant of HyperNEAT performs much better than both TD methods and all other variants of NEAT and HyperNEAT. SARSA(λ) significantly outperformed Q-Learning(λ) across all tasks. These results support previous work indicating that Q-Learning(λ) is unstable with linear function approximation (such as used for keep-away) (Stone et al., 2005) and that SARSA(λ) learns better than Q-Learning(λ) in non-deterministic tasks (Taylor et al., 2007a).

At the final generation of evolution, some variants of NEAT were observed to have comparable performance to SARSA(λ) across all keep-away tasks (for example, NS). However, OS and ONS variants of NEAT had higher task performance than SARSA(λ) with behavior transfer (section 5.5). Figure 5.2 indicated a relatively higher task performance at early behavior adaptation stages for SARSA(λ) compared to NEAT, and that NEAT outperforms SARSA(λ) at the later stages of evolution. These task

performance results suggested that even though SARSA(λ) converges relatively faster, given more evaluations NEAT evolves relatively effective behaviors compared to SARSA(λ). However, across all tasks, HyperNEAT ONS variant significantly outperforms, NEAT, SARSA(λ) and Q-Learning(λ).

Evaluating the benefits of behavior transfer in both TD methods and NE variants, the results shown in figure 5.2 and discussed in section 5.5.4 indicated that there was a task performance gain realized by each method through the transfer of learned and evolved behaviors, respectively. However, comparative task performance results (that is, between NE and RL with behavior transfer) indicated the benefit of behavior transfer task performance in TD methods was relatively high at the early stages of behavior adaptation but gradually decreased as the adaptation progressed. Conversely, NE continues to show more benefits even after the bootstrapping of increased task performance elicited by behavior transfer. Specifically, relatively high task performance with behavior transfer was obtained when evolving behaviors with a method that uses a hybrid of objective-based search and novelty search approaches. This thesis postulates that the relatively poor average task performance of RL methods compared to NE, is attributed to the fact that TD methods learn well when the domain is fully observable, and have difficulty learning in domains that are partially observable such as the keep-away (Taylor et al., 2006a; Taylor and Stone, 2009).

6.4 Behavior Transfer versus No Behavior Transfer

We tested the relative efficacy of behaviors evolved with and without behavior transfer of RL and NE across a range of keep-away tasks of increasing complexity. Results analysis highlighted the benefits of behavior transfer versus evolving behaviors from scratch with respect to effectiveness (task performance) and efficiency (speed of adaptation) of evolved behaviors. This outcome elucidates the benefits of behavior transfer for dealing with the *bootstrap* problem in complex tasks such as keep-away.

Most importantly, the comparative task performance results indicate that behavior transfer coupled with evolutionary search is a consistently suitable method for boosting the effectiveness and efficiency of evolved solution quality across increasingly complex tasks. Behavior transfer allows for the derivation of behaviors that could not otherwise be produced by RL or NE, given that either method is run from scratch. Furthermore, the experiment results highlighted that behavior transfer with NE yields significantly higher quality solutions than behavior transfer with RL, in terms of effectiveness and efficiency of evolved behaviors in keep-away tasks of increasing complexity (section 5.5.4). We postulate that the difference in task performance between RL and NE is due to the fact that RL methods, in particular TD methods, have difficulty learning in partially observable tasks (Taylor et al., 2006a). The defining feature of RoboCup keep-away was partial observability (section 2.1).

In terms of NE comparisons in these experiments, HyperNEAT consistently outperformed NEAT, and one reason for this is as follows: unlike NEAT and RL that use incomplete mapping for transfer in between tasks (section 3.1.1 and 3.3.4, respectively), HyperNEAT does not change representation between tasks during behavior transfer and has relatively higher task performance gain, given behavior transfer (Verbancsics and Stanley, 2010; Didi and Nitschke, 2016b).

6.5 Summary

This chapter discussed the comparative results of RL and NE methods applied in company with behavior transfer in increasingly complex keep-away tasks. The key methodological features that led to the highest quality solutions (behaviors) across all tasks are ONS and behavior transfer used by HyperNEAT. That is, evolutionary search in solution space for some tasks such as in keep-away based on objective-based alone leads to poor performance due to inadequate behavior exploration. On the other hand, evolutionary search based on behavior diversity maintenance mechanism alone was not adequate as it lacked pressure towards exploitation. Therefore, a method that provides adequate behavior space exploration (induced by behavioral diversity maintenance mechanism) and exploitation (induced by objective based search) yielded better results for collective behavior evolution, given behavior transfer in keep-away tasks.

The following chapter provides the thesis conclusion, highlighting the contributions of this work and future work.

Chapter 7

Conclusion

In this thesis, we have addressed two research objectives (that relates to the research questions in section 1.2). The first research objective was to ascertain the appropriateness of neuro-evolution methods for evolving effective behaviors and behavior transfer in collective behavior tasks of increasing complexity. These NE results (with behavior transfer) were compared to reinforcement learning methods, where RL was selected because it is a well-established method for behavior transfer in multi-agent systems (and thus constitutes an acceptable task performance benchmark). The second research objective was to investigate the impact of non-objective based search (genotypic and novelty search) for directing the neuro-evolution search process compared to traditional objective based techniques.

A comprehensive set of experiments allowed us to deduce the features necessary for producing a method that yields efficient and effective solutions for increasingly complex collective behavior (keep-away) tasks. That is, HyperNEAT ONS with behavior transfer was significantly more efficient than all other tested NE variants and RL methods with behavior transfer. The following sections give a summary of the contributions of this work and future direction.

7.1 Contributions

The major contributions of this thesis are as follows:

First, this thesis proposed an indirect encoded hybrid objective-behavior diversity search approach for evolving increasingly effective and complex collective behaviors. This approach integrates two methodological features: the type of neuro-evolution encoding (direct vs indirect) and the evolutionary search approach (a hybrid of objective-based and novelty search vs objective-based, novelty search and genotypic diversity).

Second, the proposed search approach was coupled with behavior transfer to boost evolution of effective collective behaviors across a range of increasingly complex keep-away tasks. The results indicated this approach evolved behaviors with

significantly higher task performance with behavior transfer than without behavior transfer. Also, this hybrid search approach out-performed all other evolutionary search approaches (with and without behavior transfer).

Third, comprehensive empirical evidence highlighting the efficacy of hybrid method that combines novelty and objective-based search, given behavior transfer, for evolving effective and efficient behaviors in keep-away task of increasing complexity. This hybrid method consistently yielded the highest task performance compared to other evolutionary search approaches such as objective-based search, pure novelty search, genotype diversity maintenance search and a hybrid of genotype diversity and objective-based search. In the context of behavior transfer, the hybrid novelty and objective-based search method yielded the most task performance benefits in comparison to traditional behavior transfer methods (SARSA and Q-Learning). The performance benefits included the effectiveness, efficiency and behavior transfer metrics (such as, jump-start, overall asymptotic performance, total reward, the reward area ratio and time to threshold).

Thus, this thesis highlighted that integrating particular methodological features into an evolutionary search method coupled with behavior transfer, resulted in an effective balance of search space exploration and exploitation across collective behavior tasks of increasing complexity. Specifically, a hybrid behavioral diversity maintenance and objective-based search approach is best suited for directing evolutionary search of an indirect encoding neuro-evolution method (HyperNEAT), given behavior transfer. Results indicated the efficacy of the NE approach was further boosted by behavior transfer where such behavior transfer was necessary for bootstrapping of solutions needed to solve complex collective behavior tasks. The success of this method in RoboCup keep-away soccer, is supported by related work that similarly demonstrated the benefits of combining behavioral diversity maintenance and objective-based search for evolving solutions for various tasks (Gomes and Christensen, 2013a). The success of the hybrid method is supported by separate lines of research: first, by related work in RL multi-agent behavior transfer (Taylor et al., 2007b), and second, by related NE work that combines behavior diversity maintenance and objective-based search (Gomes and Christensen, 2013a).

7.2 Future Possibilities

The development of a hybrid method that combines objective based and novelty search technique was inspired by the goal of widening exploration of the search space during the process of discovering regions of highly performing solutions. However, this approach used a user defined parameter ρ, that controls the level of contribution between novelty search and objective-based search (section 3.4.2). We propose investigation of dynamic adaptation of ρ parameter, where ρ parameter is adjusted

based on fitness landscape in a hybrid method. To further investigate this idea we propose an extension of this evolutionary search method to explore search spaces demonstrated in other domains, where such a method produces large archives of diverse and highly performing solutions (Pugh et al., 2015). Thus, such an approach provide both quality and diverse solutions. Similar work has already been done (Lehman and Stanley, 2011b; Mouret and Clune, 2015; Cully et al., 2015b). In particular, *Novelty Search with Local Competition* (NSLC) (Lehman and Stanley, 2011b) and *Multi-dimensional Archive of Phenotypic Elites* (MAP-Elites) (Mouret and Clune, 2015; Cully et al., 2015b). For example, the MAP-Elites technique divides the behavior space based on a domain specific behavior characterization (BC) into discrete bins. Each bin stores the latest elite genotype in a multi-dimensional archive and a single genotype occupying any given bin at any time. Thus, the elite genotype within a bin captures quality and the whole set of bins in the archive captures diversity. Specifically, such techniques eliminate the issue of integrating behavioral diversity and objective search as both quality and diversity are addressed by an evolutionary search. However, the notable drawback of this current approach is the assumption that a user has knowledge of the behavior (feature) space bounds. Thus, behavior space exploration is limited by the user assumed knowledge of behavior space.

The goal of the future research is to extend these methods to explore high dimensional and deceptive search spaces associated with complex collective behavior tasks. Specifically, exploring dynamic adaptation of exploration versus exploitation in a hybrid method, that adjusts the ρ parameter automatically.

7.3 Summary

This thesis proposed a hybrid approach that couples three methodological features (HyperNEAT with evolutionary search directed by hybrid objective-behavioral diversity search coupled with behavior transfer) to bootstrap evolution of collective behaviors in keep-away tasks of increasing complexity. This method effectively addressed the bootstrapping issue for evolving high quality behaviors for increasingly complex collective behavior tasks.

Appendix A

Task Performance

Method	4vs3 Keep-Away	5vs3 Keep-Away	5vs4 Keep-Away	6vs4 Keep-Away	6vs5 Keep-Away
OS					
NEAT	0.501 ± 0.019	0.517 ± 0.028	0.496± 0.023	0.493 ± 0.017	0.488± 0.018
HyperNEAT	0.699 ± 0.051	0.721 ± 0.045	0.689 ± 0.054	0.676 ± 0.050	0.656 ± 0.060
NEAT (NBT)	0.475 ± 0.027	0.467 ± 0.029	0.464± 0.021	0.460 ± 0.020	0.453± 0.027
HyperNEAT (NBT)	0.650 ± 0.046	0.666 ± 0.056	0.635 ± 0.034	0.636 ± 0.044	0.615 ± 0.041
NS					
NEAT	0.492 ± 0.050	0.493 ± 0.030	0.486± 0.041	0.479 ± 0.032	0.469± 0.029
HyperNEAT	0.613 ± 0.114	0.648 ± 0.128	0.607± 0.125	0.606 ± 0.151	0.594± 0.083
NEAT (NBT)	0.461 ± 0.037	0.460 ± 0.039	0.459± 0.028	0.454 ± 0.027	0.448± 0.032
HyperNEAT (NBT)	0.573 ± 0.138	0.589 ± 0.094	0.560 ± 0.082	0.559 ± 0.140	0.546 ± 0.114
ONS					
NEAT	0.516 ± 0.028	0.524 ± 0.024	0.513± 0.020	0.499 ± 0.026	0.491± 0.025
HyperNEAT	0.736 ± 0.070	0.748 ± 0.038	0.709 ± 0.052	0.695 ± 0.042	0.686 ± 0.050
NEAT (NBT)	0.479 ± 0.025	0.487 ± 0.023	0.473± 0.019	0.470 ± 0.022	0.464± 0.023
HyperNEAT (NBT)	0.672 ± 0.036	0.705 ± 0.069	0.650 ± 0.031	0.648 ± 0.031	0.633 ± 0.027
GNS					
NEAT	0.413 ± 0.045	0.417 ± 0.028	0.406± 0.047	0.406 ± 0.043	0.403± 0.040
HyperNEAT	0.505 ± 0.046	0.517 ± 0.063	0.499 ± 0.073	0.500 ± 0.021	0.496 ± 0.050
NEAT (NBT)	0.394 ± 0.040	0.398 ± 0.028	0.391± 0.043	0.387 ± 0.048	0.381± 0.046
HyperNEAT (NBT)	0.471 ± 0.048	0.491 ± 0.056	0.473 ± 0.071	0.475 ± 0.061	0.467 ± 0.087
OGN					
NEAT	0.429 ± 0.022	0.438 ± 0.044	0.423± 0.049	0.425 ± 0.027	0.412± 0.024
HyperNEAT	0.537 ± 0.050	0.563 ± 0.061	0.519 ± 0.058	0.519 ± 0.056	0.510 ± 0.024
NEAT (NBT)	0.406 ± 0.043	0.419 ± 0.052	0.396± 0.042	0.400 ± 0.049	0.395± 0.005
HyperNEAT (NBT)	0.493 ± 0.043	0.516 ± 0.075	0.473 ± 0.075	0.475 ± 0.084	0.465 ± 0.063

TABLE A.1: Normalized Task Performance. Average normalized maximum task performance for the five variants (NEAT and HyperNEAT): OS, NS, ONS, GNS and OGN. Task performance results of evolving in each task with No Behavior Transfer (NBT) are included as a benchmark comparison.

Appendix B

Effectiveness vs Efficiency - Statistical Tests

B.1 Efficiency - Statistical Test Comparison (Behavior Transfer)

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	$--$	1.5×10^{-3}	8.39×10^{-3}	1.06×10^{-7}	1.40×10^{-2}	9.50×10^{-8}	2.91×10^{-5}	3.78×10^{-2}
HN(NS)	1.5×10^{-3}	$--$	1.10×10^{-5}	5.60×10^{-3}	3.47×10^{-6}	1.80×10^{-3}	1.23×10^{-7}	2.30×10^{-3}
N(OS)	8.39×10^{-3}	1.10×10^{-5}	$--$	6.76×10^{-8}	3.14×10^{-2}	6.77×10^{-8}	1.19×10^{-6}	4.40×10^{-2}
HN(OS)	1.06×10^{-7}	5.60×10^{-3}	6.76×10^{-8}	$--$	6.73×10^{-8}	$5.99 \times 10^{-2}\varnothing$	6.67×10^{-8}	1.86×10^{-7}
N(ONS)	1.40×10^{-2}	3.47×10^{-6}	3.14×10^{-2}	6.73×10^{-8}	$--$	6.74×10^{-8}	1.29×10^{-4}	9.70×10^{-3}
HN(ONS)	9.14×10^{-8}	1.80×10^{-3}	6.67×10^{-8}	$5.99 \times 10^{-2}\varnothing$	6.74×10^{-8}	$--$	6.78×10^{-8}	3.32×10^{-7}
N(GNS)	2.91×10^{-5}	1.23×10^{-7}	1.19×10^{-6}	6.77×10^{-8}	1.29×10^{-4}	6.78×10^{-8}	$--$	7.72×10^{-7}
HN(GNS)	3.79×10^{-2}	2.30×10^{-3}	4.40×10^{-2}	1.86×10^{-7}	9.70×10^{-3}	3.32×10^{-7}	7.72×10^{-7}	$--$
N(OGN)	3.04×10^{-4}	2.20×10^{-7}	5.88×10^{-5}	6.78×10^{-8}	5.50×10^{-3}	6.79×10^{-8}	$--$	2.25×10^{-5}
HN(OGN)	5.29×10^{-2}	8.50×10^{-3}	2.50×10^{-3}	5.11×10^{-7}	1.02×10^{-4}	2.17×10^{-7}	2.17×10^{-7}	$1.98 \times 10^{-1}\varnothing$

TABLE B.1: Efficiency Statistical Tests for 4vs3 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants efficiency comparisons. Where, N and HN represents NEAT and HyperNEAT, respectively and a symbol $\varnothing$, indicates not significantly different

This section presents a set of statistical tests results of method efficiency comparison between behaviors evolved with NEAT and HyperNEAT for all given search variants (NS, OS, ONS, OGN and GNS) and for keep-away tasks of increasing complexity (that is, 4vs3, 5vs3, 5vs4, 6vs4 and 6vs5). To ascertain if there was a statistically significant difference between efficiency results from the given NE variants (given behavior transfer), a pair-wise Mann-Whitney u test with 95% confidence interval (p-values < 0.05) was performed between the method efficiency data sets. The null hypothesis states that efficiency results do not significantly differ and p-values of less than a threshold of 0.05 (Mann-Whitney u test), rejects this hypothesis. The values indicated with symbol $\varnothing$ having p-value more than 0.05 accepted this null hypothesis, suggesting that there is no statistical significance between the given method efficiency results.

Table B.1 shows 43 different statistical test results comparing efficiency for NEAT and HyperNEAT variants evolved with behavior transfer for 4vs3 keep-away task. The null hypothesis states that there is no significant difference between efficiency values for

behaviors evolved with NEAT vs HyperNEAT for different search variants. The statistical test results shows that 41 out of 43 tests rejected this null hypothesis and 2 tests (HyperNEAT OS vs ONS variant and HyperNEAT GNS vs OGN) accepted this hypothesis. Further investigation revealed that HyperNEAT OS vs ONS and GNS vs OGN variants are statistical different based on effectiveness results and Cohen's *d* effect size is 0.64 and 1.05 respectively, suggesting that there is practical significant difference between their results (table B.11 and D.1).

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	−−	3.60×10^{-2}	2.29×10^{-2}	6.76×10^{-8}	$7.97 \times 10^{-2}\varnothing$	6.79×10^{-8}	1.1×10^{-5}	1.9×10^{-2}
HN(NS)	3.60×10^{-3}	−−	3.14×10^{-2}	5.60×10^{-3}	3.60×10^{-2}	$5.07 \times 10^{-2}\varnothing$	8.52×10^{-6}	6.77×10^{-3}
N(OS)	2.29×10^{-2}	3.14×10^{-2}	−−	6.65×10^{-8}	6.37×10^{-3}	6.68×10^{-8}	6.80×10^{-7}	1.54×10^{-3}
HN(OS)	6.76×10^{-8}	5.60×10^{-3}	6.65×10^{-8}	−−	6.73×10^{-8}	$9.67 \times 10^{-1}\varnothing$	6.72×10^{-8}	9.06×10^{-8}
N(ONS)	$7.97 \times 10^{-2}\varnothing$	3.60×10^{-2}	6.37×10^{-3}	6.73×10^{-8}	−−	6.76×10^{-8}	3.05×10^{-6}	7.63×10^{-3}
HN(ONS)	6.78×10^{-8}	5.70×10^{-2}	6.68×10^{-8}	$9.67 \times 10^{-2}\varnothing$	6.76×10^{-8}	−−	6.76×10^{-8}	1.42×10^{-7}
N(GNS)	1.10×10^{-5}	8.52×10^{-6}	6.80×10^{-7}	6.73×10^{-8}	3.04×10^{-6}	6.76×10^{-8}	−−	7.52×10^{-6}
HN(GNS)	1.89×10^{-2}	6.77×10^{-3}	1.54×10^{-3}	9.06×10^{-8}	7.63×10^{-3}	1.42×10^{-7}	7.52×10^{-6}	−−
N(OGN)	2.30×10^{-3}	1.03×10^{-4}	3.60×10^{-3}	6.76×10^{-8}	2.00×10^{-3}	6.79×10^{-8}	$7.56 \times 10^{-2}\varnothing$	2.21×10^{-4}
HN(OGN)	1.14×10^{-2}	1.80×10^{-2}	1.77×10^{-6}	6.69×10^{-8}	3.36×10^{-4}	9.07×10^{-8}	7.79×10^{-8}	$2.39 \times 10^{-1}\varnothing$

TABLE B.2: Efficiency Statistical Tests for 5vs3 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants efficiency comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol ∅, indicates not significantly different

Table B.2 shows statistical tests comparing the efficiency results for behaviors evolved with NEAT and HyperNEAT variants for keep-away 5vs3. Statistical difference results indicates 40 out of 43 tests rejected the hypothesis, that stated there is no significant difference between compared efficiency results. In addition to the observations presented in table B.1, NEAT NS vs ONS variants accepted the null hypothesis, suggesting that efficiency results for behaviors evolved with NS and ONS variants do not differ significantly (p <0.05, Mann-Whitney u test for 5vs3 keep-away).

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	−−	2.43×10^{-4}	3.22×10^{-5}	6.56×10^{-8}	1.80×10^{-3}	6.56×10^{-8}	2.31×10^{-7}	$5.40 \times 10^{-2}\varnothing$
HN(NS)	2.43×10^{-4}	−−	3.27×10^{-5}	4.11×10^{-2}	5.22×10^{-5}	4.57×10^{-2}	3.75×10^{-6}	2.50×10^{-3}
N(OS)	3.22×10^{-4}	3.27×10^{-5}	−−	6.72×10^{-8}	2.08×10^{-2}	6.72×10^{-8}	6.72×10^{-8}	6.29×10^{-5}
HN(OS)	6.56×10^{-8}	4.11×10^{-2}	6.72×10^{-8}	−−	6.71×10^{-8}	7.10×10^{-3}	6.24×10^{-8}	1.77×10^{-6}
N(ONS)	1.80×10^{-3}	5.22×10^{-5}	2.08×10^{-2}	6.71×10^{-8}	−−	6.71×10^{-8}	1.04×10^{-5}	2.97×10^{-2}
HN(ONS)	6.56×10^{-8}	4.57×10^{-2}	6.72×10^{-8}	7.10×10^{-3}	6.71×10^{-8}	−−	6.74×10^{-8}	3.43×10^{-7}
N(GNS)	2.31×10^{-7}	3.75×10^{-6}	6.29×10^{-5}	6.24×10^{-8}	1.04×10^{-5}	6.24×10^{-8}	−−	3.24×10^{-6}
HN(GNS)	$5.42 \times 10^{-2}\varnothing$	2.50×10^{-3}	8.07×10^{-3}	1.77×10^{-6}	2.97×10^{-2}	3.43×10^{-6}	3.24×10^{-6}	−−
N(OGN)	1.96×10^{-5}	9.51×10^{-6}	1.50×10^{-3}	6.54×10^{-8}	6.75×10^{-4}	6.54×10^{-8}	$4.89 \times 10^{-1}\varnothing$	3.30×10^{-4}
HN(OGN)	1.53×10^{-2}	1.80×10^{-3}	8.88×10^{-7}	1.20×10^{-7}	1.56×10^{-5}	8.91×10^{-8}	6.11×10^{-8}	4.08×10^{-2}

TABLE B.3: Efficiency Statistical Tests for 5vs4 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants efficiency comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol ∅, indicates not significantly different

Table B.3, B.4 and B.5 shows the statistical test comparing efficiency results for behaviors evolved with HyperNEAT and NEAT across all variants in 5vs4, 6vs4 and 6vs5

keep-away tasks, respectively. Statistical results indicate that there was statistical difference between behavior pairs under comparison (with few observed exceptions).

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	−−	2.45×10^{-4}	1.70×10^{-2}	5.36×10^{-8}	3.4×10^{-3}	5.38×10^{-8}	3.31×10^{-6}	2.12×10^{-2}
HN(NS)	2.45×10^{-4}	−−	1.29×10^{-4}	8.30×10^{-3}	1.27×10^{-4}	1.14×10^{-2}	1.78×10^{-5}	5.10×10^{-3}
N(OS)	1.70×10^{-2}	1.29×10^{-4}	−−	6.77×10^{-8}	$6.75 \times 10^{-2}\varnothing$	6.73×10^{-8}	2.22×10^{-2}	4.00×10^{-3}
HN(OS)	5.36×10^{-8}	8.30×10^{-3}	6.77×10^{-8}	−−	6.59×10^{-8}	4.73×10^{-2}	6.68×10^{-8}	6.77×10^{-8}
N(ONS)	3.40×10^{-3}	1.27×10^{-4}	$6.75 \times 10^{-2}\varnothing$	6.59×10^{-8}	−−	6.55×10^{-8}	8.90×10^{-3}	1.50×10^{-3}
HN(ONS)	5.33×10^{-8}	1.14×10^{-2}	6.73×10^{-8}	4.73×10^{-2}	6.55×10^{-8}	−−	6.64×10^{-8}	6.72×10^{-8}
N(GNS)	3.31×10^{-6}	1.79×10^{-5}	2.22×10^{-2}	6.68×10^{-8}	8.90×10^{-3}	6.64×10^{-8}	−−	6.54×10^{-5}
HN(GNS)	2.12×10^{-2}	5.10×10^{-3}	4.00×10^{-3}	6.77×10^{-8}	1.50×10^{-3}	6.73×10^{-8}	6.54×10^{-5}	−−
N(OGN)	1.09×10^{-7}	1.37×10^{-5}	5.55×10^{-3}	6.46×10^{-8}	1.55×10^{-4}	6.42×10^{-8}	$5.42 \times 10^{-2}\varnothing$	3.23×10^{-5}
HN(OGN)	7.87×10^{-3}	3.30×10^{-3}	9.19×10^{-4}	7.88×10^{-8}	2.19×10^{-4}	6.74×10^{-8}	4.84×10^{-6}	$5.79 \times 10^{-1}\varnothing$

TABLE B.4: Efficiency Statistical Tests for 6vs4 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants efficiency comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol ∅, indicates not significantly different

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	−−	1.10×10^{-3}	4.59×10^{-4}	1.23×10^{-7}	2.30×10^{-3}	1.22×10^{-7}	1.59×10^{-5}	$5.79 \times 10^{-2}\varnothing$
HN(NS)	1.10×10^{-3}	−−	1.10×10^{-5}	1.30×10^{-2}	3.29×10^{-5}	2.98×10^{-2}	2.05×10^{-6}	7.70×10^{-3}
N(OS)	4.59×10^{-4}	1.10×10^{-5}	−−	7.85×10^{-8}	$6.55 \times 10^{-1}\varnothing$	9.08×10^{-8}	8.32×10^{-4}	3.84×10^{-2}
HN(OS)	1.23×10^{-7}	1.33×10^{-2}	7.85×10^{-8}	−−	1.06×10^{-7}	4.41×10^{-2}	6.73×10^{-8}	1.15×10^{-4}
N(ONS)	2.30×10^{-3}	2.29×10^{-5}	$6.55 \times 10^{-1}\varnothing$	1.06×10^{-7}	−−	1.22×10^{-7}	7.55×10^{-4}	7.63×10^{-3}
HN(ONS)	1.22×10^{-7}	2.98×10^{-2}	9.08×10^{-8}	4.41×10^{-2}	1.22×10^{-7}	−−	6.70×10^{-8}	2.20×10^{-4}
N(GNS)	1.59×10^{-5}	2.05×10^{-6}	8.32×10^{-4}	6.73×10^{-8}	7.55×10^{-4}	6.70×10^{-8}	−−	7.10×10^{-3}
HN(GNS)	$5.79 \times 10^{-2}\varnothing$	7.70×10^{-3}	3.84×10^{-2}	1.15×10^{-4}	7.63×10^{-3}	2.20×10^{-4}	7.10×10^{-3}	−−
N(OGN)	2.58×10^{-5}	3.97×10^{-6}	2.39×10^{-2}	6.75×10^{-8}	2.58×10^{-5}	3.97×10^{-6}	$6.00 \times 10^{-2}\varnothing$	2.22×10^{-2}
HN(OGN)	8.56×10^{-3}	4.30×10^{-3}	4.12×10^{-5}	1.21×10^{-7}	2.45×10^{-4}	1.21×10^{-7}	2.03×10^{-6}	$1.80 \times 10^{-1}\varnothing$

TABLE B.5: Efficiency Statistical Tests for 6vs5 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants efficiency comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol ∅, indicates not significantly different

B.2 Efficiency - Behavior Transfer vs No Behavior Transfer

This section presents the statistical tests results that assess the statistical difference between NE variants efficiency results (behavior transfer versus no behavior transfer) based on Mann-Whitney u test with 95% confidence interval (p-values < 0.05). The null hypothesis states that there was no significant difference between the efficiency results of each method evolved with behavior transfer versus that evolved without behavior transfer. Statistical test p-values of less than a threshold of 0.05 (Mann-Whitney u test), rejects this hypothesis. The values indicated with symbol ∅ having p-value more than 0.05 accepted this null hypothesis, suggesting that there was no statistical significance between the given method efficiency results.

Table B.6, B.7, B.8, B.9 and B.10, shows statistical test p-values comparing efficiency results of behaviors evolved with NEAT and HyperNEAT with vs without behavior

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	1.93×10^{-2}	1.25×10^{-5}	5.60×10^{-3}	1.06×10^{-7}	6.38×10^{-3}	1.60×10^{-7}	3.65×10^{-2}	1.61×10^{-3}
HN(NS)	9.09×10^{-3}	9.89×10^{-3}	1.07×10^{-2}	2.75×10^{-2}	7.20×10^{-3}	1.23×10^{-2}	1.79×10^{-2}	1.81×10^{-2}
N(OS)	2.98×10^{-2}	1.41×10^{-5}	2.29×10^{-2}	6.77×10^{-8}	4.09×10^{-2}	6.78×10^{-8}	9.72×10^{-6}	6.78×10^{-3}
HN(OS)	7.83×10^{-8}	3.37×10^{-2}	6.73×10^{-8}	1.14×10^{-2}	6.70×10^{-8}	1.55×10^{-2}	6.74×10^{-8}	1.38×10^{-7}
N(ONS)	4.41×10^{-2}	5.88×10^{-5}	$9.46 \times 10^{-2}\varnothing$	6.79×10^{-8}	2.29×10^{-2}	6.80×10^{-8}	2.21×10^{-3}	3.28×10^{-2}
HN(ONS)	6.75×10^{-8}	1.00×10^{-3}	6.74×10^{-8}	2.62×10^{-2}	6.71×10^{-8}	8.18×10^{-3}	6.75×10^{-8}	8.85×10^{-8}
N(GNS)	5.88×10^{-5}	1.23×10^{-7}	2.34×10^{-6}	6.78×10^{-8}	1.78×10^{-4}	6.79×10^{-8}	$4.73 \times 10^{-1}\varnothing$	2.01×10^{-6}
HN(GNS)	2.30×10^{-2}	1.03×10^{-4}	2.98×10^{-2}	6.77×10^{-8}	6.55×10^{-3}	6.78×10^{-8}	1.50×10^{-3}	7.17×10^{-3}
N(OGN)	3.04×10^{-4}	2.21×10^{-7}	5.88×10^{-5}	6.78×10^{-8}	5.50×10^{-3}	6.79×10^{-8}	2.10×10^{-3}	7.35×10^{-3}
HN(OGN)	9.89×10^{-3}	1.20×10^{-3}	6.95×10^{-3}	9.13×10^{-8}	9.61×10^{-3}	6.78×10^{-8}	1.59×10^{-5}	4.89×10^{-2}

TABLE B.6: Efficiency Statistical Tests for 4vs3 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer

transfer. The statistical test results indicates an average of 78 out 80 statistical tests rejects the null hypothesis.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	6.39×10^{-3}	1.8×10^{-3}	4.73×10^{-2}	6.74×10^{-8}	9.08×10^{-3}	6.77×10^{-8}	4.14×10^{-4}	1.56×10^{-2}
HN(NS)	3.60×10^{-3}	2.29×10^{-2}	3.56×10^{-3}	8.27×10^{-5}	9.80×10^{-3}	9.27×10^{-5}	2.05×10^{-6}	8.42×10^{-3}
N(OS)	3.37×10^{-2}	1.32×10^{-2}	$9.24 \times 10^{-2}\varnothing$	6.65×10^{-8}	3.94×10^{-2}	6.65×10^{-8}	7.82×10^{-7}	2.97×10^{-2}
HN(OS)	9.13×10^{-8}	9.68×10^{-3}	6.66×10^{-8}	1.72×10^{-2}	6.74×10^{-8}	1.89×10^{-2}	6.74×10^{-8}	2.55×10^{-7}
N(ONS)	2.98×10^{-2}	1.43×10^{-2}	1.55×10^{-2}	6.75×10^{-8}	7.35×10^{-3}	6.71×10^{-8}	1.59×10^{-5}	3.79×10^{-2}
HN(ONS)	6.79×10^{-8}	1.10×10^{-2}	6.68×10^{-8}	9.00×10^{-3}	6.76×10^{-8}	2.07×10^{-2}	6.76×10^{-8}	1.65×10^{-7}
N(GNS)	9.09×10^{-7}	9.03×10^{-7}	1.04×10^{-7}	6.73×10^{-8}	2.94×10^{-7}	6.76×10^{-8}	2.74×10^{-2}	7.87×10^{-7}
HN(GNS)	7.15×10^{-3}	1.14×10^{-2}	3.50×10^{-2}	6.75×10^{-8}	5.61×10^{-3}	9.15×10^{-8}	1.90×10^{-3}	4.98×10^{-2}
N(OGN)	9.73×10^{-6}	3.95×10^{-6}	5.92×10^{-7}	6.74×10^{-8}	2.05×10^{-6}	6.77×10^{-8}	$6.75 \times 10^{-2}\varnothing$	5.83×10^{-6}
HN(OGN)	7.97×10^{-3}	1.33×10^{-2}	1.55×10^{-2}	6.75×10^{-8}	4.57×10^{-2}	1.06×10^{-7}	7.1×10^{-3}	3.94×10^{-2}

TABLE B.7: Efficiency Statistical Tests for 5vs3 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	2.23×10^{-3}	9.19×10^{-5}	9.00×10^{-3}	2.18×10^{-7}	7.70×10^{-3}	6.67×10^{-8}	7.12×10^{-3}	3.90×10^{-3}
HN(NS)	7.48×10^{-4}	4.68×10^{-2}	4.65×10^{-5}	1.37×10^{-6}	7.36×10^{-5}	2.03×10^{-5}	3.29×10^{-6}	1.32×10^{-2}
N(OS)	1.37×10^{-5}	2.90×10^{-5}	4.09×10^{-2}	6.68×10^{-8}	5.42×10^{-3}	6.68×10^{-8}	2.45×10^{-5}	1.01×10^{-2}
HN(OS)	1.38×10^{-7}	5.79×10^{-3}	9.05×10^{-8}	3.36×10^{-4}	9.03×10^{-8}	5.50×10^{-3}	6.22×10^{-8}	2.55×10^{-5}
N(ONS)	9.40×10^{-3}	1.36×10^{-4}	2.13×10^{-6}	5.60×10^{-8}	1.29×10^{-2}	5.60×10^{-8}	5.55×10^{-8}	3.21×10^{-2}
HN(ONS)	4.36×10^{-7}	5.49×10^{-3}	1.62×10^{-7}	1.28×10^{-4}	1.40×10^{-7}	4.38×10^{-2}	6.18×10^{-8}	5.15×10^{-5}
N(GNS)	1.85×10^{-7}	5.13×10^{-6}	6.61×10^{-6}	6.72×10^{-8}	3.94×10^{-6}	6.71×10^{-8}	8.38×10^{-2}	4.44×10^{-6}
HN(GNS)	5.79×10^{-3}	5.07×10^{-4}	1.79×10^{-2}	9.08×10^{-8}	1.20×10^{-2}	1.22×10^{-7}	7.06×10^{-5}	$9.46 \times 10^{-2}\varnothing$
N(OGN)	1.17×10^{-6}	9.69×10^{-6}	1.98×10^{-4}	6.72×10^{-8}	3.67×10^{-5}	6.72×10^{-8}	4.23×10^{-2}	1.08×10^{-7}
HN(OGN)	9.46×10^{-3}	1.30×10^{-3}	9.08×10^{-3}	1.57×10^{-6}	2.39×10^{-2}	2.05×10^{-6}	1.60×10^{-3}	4.79×10^{-2}

TABLE B.8: Efficiency Statistical Tests for 5vs4 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	4.13×10^{-4}	1.28×10^{-4}	4.15×10^{-2}	6.71×10^{-8}	2.28×10^{-2}	6.67×10^{-8}	1.33×10^{-2}	1.60×10^{-3}
HN(NS)	1.47×10^{-2}	2.18×10^{-2}	9.19×10^{-4}	6.21×10^{-4}	6.78×10^{-4}	2.60×10^{-3}	5.83×10^{-5}	4.25×10^{-2}
N(OS)	2.36×10^{-2}	1.60×10^{-4}	1.99×10^{-2}	6.74×10^{-8}	1.80×10^{-2}	6.70×10^{-8}	3.35×10^{-4}	4.70×10^{-3}
HN(OS)	5.36×10^{-8}	8.58×10^{-3}	6.77×10^{-8}	2.07×10^{-2}	6.59×10^{-8}	3.85×10^{-2}	6.68×10^{-8}	1.43×10^{-7}
N(ONS)	2.14×10^{-2}	1.58×10^{-4}	3.13×10^{-2}	6.57×10^{-8}	2.70×10^{-3}	6.53×10^{-8}	2.27×10^{-6}	4.30×10^{-3}
HN(ONS)	5.32×10^{-8}	1.02×10^{-2}	6.72×10^{-8}	7.10×10^{-3}	6.54×10^{-8}	1.92×10^{-2}	6.63×10^{-8}	2.93×10^{-7}
N(GNS)	7.49×10^{-7}	3.97×10^{-6}	5.08×10^{-4}	6.78×10^{-8}	1.27×10^{-4}	6.74×10^{-8}	1.02×10^{-2}	8.57×10^{-6}
HN(GNS)	9.24×10^{-3}	1.50×10^{-3}	5.25×10^{-3}	6.00×10^{-7}	4.24×10^{-2}	6.87×10^{-7}	6.00×10^{-3}	1.71×10^{-2}
N(OGN)	9.85×10^{-7}	4.56×10^{-6}	1.50×10^{-3}	6.74×10^{-8}	4.53×10^{-4}	6.70×10^{-8}	1.48×10^{-2}	1.10×10^{-5}
HN(OGN)	2.36×10^{-2}	7.56×10^{-4}	4.09×10^{-2}	1.06×10^{-7}	3.94×10^{-2}	7.82×10^{-8}	9.00×10^{-3}	6.00×10^{-3}

TABLE B.9: Efficiency Statistical Tests for 6vs4 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	5.84×10^{-6}	3.48×10^{-6}	1.26×10^{-2}	6.75×10^{-8}	5.65×10^{-3}	6.77×10^{-8}	7.19×10^{-3}	1.05×10^{-2}
HN(NS)	1.93×10^{-2}	3.65×10^{-2}	7.70×10^{-3}	1.44×10^{-2}	7.70×10^{-3}	4.38×10^{-2}	1.29×10^{-4}	1.33×10^{-2}
N(OS)	4.14×10^{-4}	1.59×10^{-5}	1.56×10^{-2}	6.77×10^{-8}	$5.08 \times 10^{-2}\varnothing$	6.74×10^{-8}	2.46×10^{-4}	1.20×10^{-2}
HN(OS)	2.20×10^{-7}	9.03×10^{-3}	9.12×10^{-8}	3.60×10^{-3}	9.13×10^{-8}	6.50×10^{-3}	6.73×10^{-8}	5.10×10^{-3}
N(ONS)	9.00×10^{-3}	6.59×10^{-5}	9.18×10^{-4}	6.75×10^{-8}	1.93×10^{-2}	6.72×10^{-8}	1.29×10^{-4}	5.61×10^{-3}
HN(ONS)	2.93×10^{-7}	$8.82 \times 10^{-2}\varnothing$	9.07×10^{-8}	3.30×10^{-3}	1.42×10^{-7}	1.43×10^{-2}	7.78×10^{-8}	1.50×10^{-3}
N(GNS)	1.04×10^{-6}	6.90×10^{-7}	5.60×10^{-3}	6.77×10^{-8}	3.30×10^{-3}	6.74×10^{-8}	$6.17 \times 10^{-2}\varnothing$	2.10×10^{-3}
HN(GNS)	2.18×10^{-2}	5.08×10^{-4}	4.91×10^{-2}	3.92×10^{-7}	6.75×10^{-3}	6.86×10^{-7}	2.56×10^{-2}	$5.61 \times 10^{-2}\varnothing$
N(OGN)	3.97×10^{-6}	9.11×10^{-7}	1.44×10^{-2}	6.77×10^{-8}	8.30×10^{-3}	6.74×10^{-8}	4.90×10^{-2}	9.00×10^{-3}
HN(OGN)	2.85×10^{-2}	4.15×10^{-4}	4.41×10^{-2}	2.95×10^{-7}	4.55×10^{-2}	3.91×10^{-7}	2.56×10^{-2}	8.39×10^{-3}

TABLE B.10: Efficiency Statistical Tests for 6vs5 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

B.3 Effectiveness - Statistical Test Comparison

This section presents a set of statistical tests results of task performance comparison (in terms of method effectiveness) between NE variants of NEAT and HyperNEAT (NS, OS, ONS, OGN and GNS) for keep-away tasks of increasing complexity (that is, 4vs3, 5vs3, 5vs4, 6vs4 and 6vs5). To ascertain if there was a statistically significant difference between NE variants task performance results, a pair-wise Mann-Whitney u test with 95% confidence interval (p-values < 0.05) was performed between task performance data sets. As in section B.1, the null hypothesis states that task performance results do not significantly differ in terms of effectiveness and p-values less than a threshold of 0.05 (Mann-Whitney u test), rejects this hypothesis. The values indicated with symbol $\varnothing$ having p-value more than 0.05 accepted this null hypothesis, suggesting that there is no statistical significance between the given task performance values results (given effectiveness).

Figures B.11, B.12, B.13, B.14 and B.15 shows the task performance statistical test comparison for five given NEAT and HyperNEAT variants in keep-away tasks of increasing complexity (4vs3, 5vs3, 5vs4, 6vs4 and 6vs5). The null hypothesis was that

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	–	6.80×10^{-8}	8.20×10^{-3}	5.55×10^{-10}	7.25×10^{-4}	5.55×10^{-10}	1.53×10^{-6}	4.97×10^{-2}
HN(NS)	5.55×10^{-10}	–	5.55×10^{-10}	8.42×10^{-6}	4.74×10^{-9}	8.42×10^{-6}	5.55×10^{-10}	2.49×10^{-7}
N(OS)	8.20×10^{-3}	5.55×10^{-10}	–	5.55×10^{-10}	2.32×10^{-2}	5.55×10^{-10}	3.63×10^{-8}	1.36×10^{-2}
HN(OS)	5.55×10^{-10}	8.20×10^{-6}	5.55×10^{-10}	–	5.55×10^{-10}	$7.20 \times 10^{-2}\varnothing$	5.55×10^{-10}	5.55×10^{-10}
N(ONS)	7.25×10^{-4}	4.74×10^{-9}	2.32×10^{-2}	5.55×10^{-10}	–	5.55×10^{-10}	4.74×10^{-9}	8.20×10^{-3}
HN(ONS)	5.55×10^{-10}	8.41×10^{-6}	5.55×10^{-10}	$7.20 \times 10^{-2}\varnothing$	5.55×10^{-10}	–	5.55×10^{-10}	5.55×10^{-10}
N(GNS)	1.53×10^{-6}	5.55×10^{-10}	3.63×10^{-8}	5.55×10^{-10}	4.74×10^{-9}	5.55×10^{-10}	–	2.49×10^{-7}
HN(GNS)	4.97×10^{-2}	2.49×10^{-7}	1.35×10^{-2}	5.55×10^{-10}	8.20×10^{-3}	5.55×10^{-10}	2.49×10^{-7}	–
N(OGN)	4.74×10^{-9}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	2.32×10^{-2}	4.74×10^{-9}
HN(OGN)	1.83×10^{-4}	7.25×10^{-4}	4.15×10^{-5}	5.55×10^{-10}	2.32×10^{-2}	5.55×10^{-10}	3.63×10^{-8}	2.32×10^{-2}

TABLE B.11: Task performance Statistical Tests for 4vs3 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants method effectiveness comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol $\varnothing$, indicates not significantly different

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	–	8.42×10^{-6}	1.53×10^{-6}	5.55×10^{-10}	4.15×10^{-5}	5.55×10^{-10}	3.63×10^{-8}	8.20×10^{-3}
HN(NS)	8.42×10^{-6}	–	4.15×10^{-5}	8.20×10^{-3}	4.15×10^{-5}	2.6×10^{-3}	1.53×10^{-6}	1.83×10^{-4}
N(OS)	1.53×10^{-6}	4.15×10^{-5}	–	5.55×10^{-10}	4.91×10^{-2}	5.55×10^{-10}	4.74×10^{-9}	2.32×10^{-2}
HN(OS)	5.55×10^{-10}	8.20×10^{-3}	5.55×10^{-10}	–	5.55×10^{-10}	7.25×10^{-4}	5.55×10^{-10}	5.55×10^{-10}
N(ONS)	4.15×10^{-5}	4.15×10^{-5}	4.91×10^{-2}	5.55×10^{-10}	–	5.55×10^{-10}	5.55×10^{-10}	4.91×10^{-2}
HN(ONS)	5.55×10^{-10}	2.60×10^{-3}	5.55×10^{-10}	7.25×10^{-4}	5.55×10^{-10}	–	5.55×10^{-10}	5.55×10^{-10}
N(GNS)	3.63×10^{-8}	1.53×10^{-6}	4.74×10^{-9}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	–	
HN(GNS)	8.20×10^{-3}	1.83×10^{-4}	2.32×10^{-2}	5.55×10^{-10}	4.91×10^{-2}	5.55×10^{-10}	8.42×10^{-6}	–
N(OGN)	1.53×10^{-6}	8.14×10^{-6}	3.63×10^{-8}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	$2.73 \times 10^{-1}\varnothing$	4.15×10^{-5}
HN(OGN)	8.42×10^{-6}	2.6×10^{-3}	4.15×10^{-5}	3.63×10^{-8}	7.25×10^{-4}	5.55×10^{-10}	2.49×10^{-7}	$1.35 \times 10^{-1}\varnothing$

TABLE B.12: Task performance Statistical Tests for 5vs3 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants method effectiveness comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol $\varnothing$, indicates not significantly different

task performance results obtained from keep-away behaviors evolved with NEAT and HyperNEAT do not differ between different given variants. The statistical test results shows an average of 41 out of 44 tests, rejects this null hypothesis. Then 3 out of 44 marginally accepts the hypothesis, indicating that task performance results significantly differ between NEAT and HyperNEAT variants. Further analysis, indicated that 45 out of 50 statistical tests had Cohen's d effect size greater than 0.6, suggesting that the task performance results are practically different (table D.1.

B.4 Effectiveness - Behavior Transfer vs No Behavior Transfer

Similar to the statistical test results presented in appendix B.3, this section presents the statistical tests results that assess the statistical difference between NE variants task performance results (behavior transfer versus no behavior transfer) based on Mann-Whitney u test with 95% confidence interval (p-values < 0.05). The null hypothesis states that there is no significant difference between the task performance results of each method evolved with behavior transfer versus the same evolved without behavior transfer. Statistical test p-values of less than a threshold of 0.05

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	–	4.15×10^{-5}	$1.35 \times 10^{-1}\varnothing$	5.55×10^{-10}	2.32×10^{-2}	5.55×10^{-10}	3.63×10^{-8}	8.20×10^{-3}
HN(NS)	4.15×10^{-5}	–	4.15×10^{-5}	$7.20 \times 10^{-2}\varnothing$	4.15×10^{-5}	4.91×10^{-2}	1.53×10^{-6}	7.25×10^{-4}
N(OS)	$1.35 \times 10^{-1}\varnothing$	4.15×10^{-5}	–	5.55×10^{-10}	8.2×10^{-3}	5.54×10^{-10}	5.55×10^{-10}	8.20×10^{-3}
HN(OS)	5.55×10^{-10}	$7.20 \times 10^{-2}\varnothing$	5.55×10^{-10}	–	5.55×10^{-10}	4.91×10^{-2}	5.55×10^{-10}	4.74×10^{-9}
N(ONS)	2.32×10^{-2}	4.15×10^{-5}	8.20×10^{-3}	5.55×10^{-10}	–	5.55×10^{-10}	5.55×10^{-10}	8.20×10^{-3}
HN(ONS)	5.55×10^{-10}	4.91×10^{-2}	5.55×10^{-10}	4.97×10^{-2}	5.55×10^{-10}	–	5.55×10^{-10}	5.55×10^{-10}
N(GNS)	3.63×10^{-8}	1.53×10^{-6}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	–	1.83×10^{-4}
HN(GNS)	8.20×10^{-3}	7.25×10^{-4}	8.20×10^{-3}	4.74×10^{-9}	8.20×10^{-3}	5.55×10^{-10}	1.83×10^{-4}	–
N(OGN)	4.15×10^{-5}	4.15×10^{-5}	2.49×10^{-7}	5.55×10^{-10}	4.74×10^{-9}	5.55×10^{-10}	1.34×10^{-1}	1.83×10^{-4}
HN(OGN)	8.20×10^{-3}	7.25×10^{-4}	8.20×10^{-3}	3.63×10^{-8}	2.32×10^{-2}	3.63×10^{-8}	2.49×10^{-7}	$2.75 \times 10^{-1}\varnothing$

TABLE B.13: Task performance Statistical Tests for 5vs4 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants method effectiveness comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol ∅, indicates not significantly different

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	–	7.25×10^{-4}	$1.35 \times 10^{-1}\varnothing$	5.55×10^{-10}	7.19×10^{-2}	5.55×10^{-10}	4.15×10^{-5}	1.33×10^{-2}
HN(NS)	7.25×10^{-4}	–	7.25×10^{-4}	2.32×10^{-2}	7.25×10^{-4}	2.32×10^{-2}	1.83×10^{-4}	7.25×10^{-4}
N(OS)	$1.35 \times 10^{-1}\varnothing$	7.25×10^{-4}	–	5.55×10^{-10}	$4.97 \times 10^{-1}\varnothing$	5.54×10^{-10}	3.63×10^{-8}	4.91×10^{-2}
HN(OS)	5.55×10^{-10}	2.32×10^{-2}	5.55×10^{-10}	–	5.55×10^{-10}	2.32×10^{-2}	5.55×10^{-10}	5.55×10^{-10}
N(ONS)	$7.19 \times 10^{-2}\varnothing$	7.25×10^{-4}	$4.97 \times 10^{-1}\varnothing$	5.55×10^{-10}	–	5.55×10^{-10}	3.63×10^{-8}	7.71×10^{-2}
HN(ONS)	5.55×10^{-10}	2.32×10^{-2}	5.55×10^{-10}	2.32×10^{-2}	5.55×10^{-10}	–	5.55×10^{-10}	5.55×10^{-10}
N(GNS)	4.15×10^{-5}	1.83×10^{-4}	3.63×10^{-8}	5.55×10^{-10}	3.63×10^{-8}	5.55×10^{-10}	–	2.49×10^{-7}
HN(GNS)	1.33×10^{-2}	7.25×10^{-4}	4.91×10^{-2}	5.55×10^{-10}	$7.71 \times 10^{-2}\varnothing$	5.55×10^{-10}	2.49×10^{-7}	–
N(OGN)	1.53×10^{-6}	1.83×10^{-4}	3.63×10^{-8}	5.55×10^{-10}	3.63×10^{-8}	5.55×10^{-10}	2.60×10^{-3}	2.49×10^{-7}
HN(OGN)	2.32×10^{-2}	8.20×10^{-3}	$1.71 \times 10^{-1}\varnothing$	3.63×10^{-8}	$4.97 \times 10^{-1}\varnothing$	4.74×10^{-9}	8.42×10^{-6}	$7.71 \times 10^{-2}\varnothing$

TABLE B.14: Task performance Statistical Tests for 6vs4 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants method effectiveness comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol ∅, indicates not significantly different

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	–	2.49×10^{-7}	1.83×10^{-4}	5.55×10^{-10}	2.60×10^{-3}	5.55×10^{-10}	3.63×10^{-8}	$5.92 \times 10^{-2}\varnothing$
HN(NS)	2.49×10^{-7}	–	2.49×10^{-7}	2.07×10^{-2}	2.49×10^{-7}	2.60×10^{-3}	4.74×10^{-9}	7.25×10^{-4}
N(OS)	1.83×10^{-4}	2.49×10^{-7}	–	5.55×10^{-10}	$4.97 \times 10^{-1}\varnothing$	5.54×10^{-10}	5.55×10^{-10}	8.20×10^{-3}
HN(OS)	5.55×10^{-10}	2.07×10^{-2}	5.55×10^{-10}	–	5.55×10^{-10}	4.91×10^{-2}	5.55×10^{-10}	3.63×10^{-8}
N(ONS)	2.60×10^{-3}	2.49×10^{-7}	$4.97 \times 10^{-1}\varnothing$	5.55×10^{-10}	–	5.55×10^{-10}	5.55×10^{-10}	4.91×10^{-2}
HN(ONS)	5.55×10^{-10}	2.60×10^{-3}	5.55×10^{-10}	$5.91 \times 10^{-2}\varnothing$	5.55×10^{-10}	–	5.55×10^{-10}	5.55×10^{-10}
N(GNS)	3.63×10^{-8}	4.74×10^{-9}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	–	1.53×10^{-6}
HN(GNS)	5.92×10^{-2}	7.25×10^{-4}	8.20×10^{-3}	3.63×10^{-8}	4.91×10^{-2}	5.55×10^{-10}	1.53×10^{-6}	–
N(OGN)	3.63×10^{-8}	4.74×10^{-9}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	5.55×10^{-10}	2.32×10^{-3}	1.53×10^{-6}
HN(OGN)	4.15×10^{-5}	1.53×10^{-6}	8.14×10^{-6}	4.74×10^{-9}	2.60×10^{-3}	5.55×10^{-10}	5.55×10^{-10}	$4.56 \times 10^{-1}\varnothing$

TABLE B.15: Task performance Statistical Tests for 6vs5 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants method effectiveness comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol ∅, indicates not significantly different

(Mann-Whitney u test), rejects this hypothesis. The values indicated with symbol ∅ having p-value more than 0.05 accepted this null hypothesis, suggesting that there was no statistical significance between the given method effectiveness results.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	3.85×10^{-2}	6.79×10^{-8}	2.10×10^{-3}	6.77×10^{-8}	3.28×10^{-5}	6.78×10^{-8}	7.20×10^{-3}	7.10×10^{-3}
HN(NS)	5.79×10^{-3}	2.39×10^{-2}	3.37×10^{-2}	7.56×10^{-6}	5.64×10^{-3}	3.56×10^{-6}	5.26×10^{-5}	4.11×10^{-2}
N(OS)	6.78×10^{-5}	6.75×10^{-8}	6.58×10^{-5}	6.73×10^{-8}	3.95×10^{-6}	6.74×10^{-8}	3.34×10^{-4}	6.19×10^{-4}
HN(OS)	6.80×10^{-8}	3.85×10^{-2}	6.77×10^{-8}	2.80×10^{-3}	6.74×10^{-8}	1.04×10^{-4}	6.80×10^{-8}	6.76×10^{-8}
N(ONS)	1.60×10^{-2}	6.78×10^{-3}	2.56×10^{-2}	6.76×10^{-8}	5.61×10^{-4}	6.78×10^{-8}	5.89×10^{-5}	1.23×10^{-2}
HN(ONS)	6.79×10^{-8}	2.00×10^{-3}	6.77×10^{-8}	1.93×10^{-2}	6.74×10^{-8}	8.35×10^{-4}	6.80×10^{-8}	6.76×10^{-8}
N(GNS)	1.42×10^{-7}	6.74×10^{-8}	6.72×10^{-8}	6.72×10^{-8}	6.68×10^{-8}	6.73×10^{-8}	$2.18 \times 10^{-1}\varnothing$	1.05×10^{-7}
HN(GNS)	5.97×10^{-1}	6.36×10^{-8}	1.54×10^{-2}	6.33×10^{-8}	3.00×10^{-3}	6.35×10^{-8}	1.60×10^{-3}	$1.89 \times 10^{-1}\varnothing$
N(OGN)	6.79×10^{-8}	6.79×10^{-8}	6.77×10^{-8}	6.77×10^{-8}	6.72×10^{-8}	6.78×10^{-8}	$7.77 \times 10^{-1}\varnothing$	6.75×10^{-8}
HN(OGN)	4.67×10^{-2}	3.40×10^{-7}	4.9×10^{-2}	6.75×10^{-8}	2.22×10^{-2}	6.76×10^{-8}	1.41×10^{-5}	$4.41 \times 10^{-1}\varnothing$

TABLE B.16: Task performance Statistical Tests for 4vs3 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	5.30×10^{-3}	1.61×10^{-4}	5.25×10^{-5}	6.78×10^{-8}	9.72×10^{-6}	6.74×10^{-8}	2.94×10^{-2}	1.54×10^{-2}
HN(NS)	1.79×10^{-4}	1.08×10^{-2}	1.67×10^{-2}	5.24×10^{-5}	2.07×10^{-2}	1.22×10^{-7}	7.93×10^{-7}	2.38×10^{-2}
N(OS)	1.55×10^{-2}	7.39×10^{-5}	9.71×10^{-6}	6.76×10^{-8}	5.14×10^{-6}	6.72×10^{-8}	1.79×10^{-4}	6.00×10^{-3}
HN(OS)	6.78×10^{-8}	4.73×10^{-2}	6.77×10^{-8}	6.78×10^{-3}	6.78×10^{-8}	2.03×10^{-5}	6.78×10^{-8}	1.89×10^{-7}
N(ONS)	4.74×10^{-2}	4.60×10^{-4}	2.56×10^{-2}	6.78×10^{-8}	1.30×10^{-3}	6.74×10^{-8}	1.10×10^{-5}	4.80×10^{-2}
HN(ONS)	6.77×10^{-8}	1.20×10^{-2}	6.77×10^{-8}	$8.20 \times 10^{-2}\varnothing$	6.76×10^{-8}	3.30×10^{-3}	6.78×10^{-8}	6.68×10^{-8}
N(GNS)	6.74×10^{-8}	6.75×10^{-8}	6.73×10^{-8}	6.73×10^{-8}	6.72×10^{-8}	6.69×10^{-8}	1.79×10^{-2}	7.80×10^{-7}
HN(GNS)	4.70×10^{-2}	2.47×10^{-4}	6.78×10^{-3}	6.77×10^{-8}	8.30×10^{-3}	6.73×10^{-8}	1.16×10^{-4}	3.84×10^{-2}
N(OGN)	1.80×10^{-5}	1.20×10^{-6}	5.20×10^{-7}	6.76×10^{-8}	2.94×10^{-7}	6.72×10^{-8}	$6.80 \times 10^{-2}\varnothing$	4.63×10^{-5}
HN(OGN)	2.08×10^{-2}	2.10×10^{-3}	4.90×10^{-2}	6.78×10^{-8}	4.60×10^{-2}	6.74×10^{-8}	2.69×10^{-6}	$8.60 \times 10^{-1}\varnothing$

TABLE B.17: Task performance Statistical Tests for 5vs3 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	1.15×10^{-4}	2.74×10^{-4}	3.48×10^{-6}	6.77×10^{-8}	1.43×10^{-7}	6.78×10^{-8}	3.08×10^{-2}	7.10×10^{-3}
HN(NS)	1.06×10^{-2}	4.96×10^{-2}	4.38×10^{-2}	2.92×10^{-5}	4.1×10^{-2}	4.16×10^{-5}	1.25×10^{-6}	4.37×10^{-2}
N(OS)	1.97×10^{-5}	9.03×10^{-4}	7.48×10^{-8}	6.45×10^{-8}	6.45×10^{-8}	6.45×10^{-8}	1.01×10^{-2}	2.50×10^{-3}
HN(OS)	6.69×10^{-8}	4.97×10^{-2}	6.7×10^{-8}	9.17×10^{-4}	6.72×10^{-8}	6.84×10^{-4}	5.97×10^{-8}	1.60×10^{-7}
N(ONS)	$1.13 \times 10^{-1}\varnothing$	7.10×10^{-3}	9.00×10^{-3}	6.78×10^{-8}	1.80×10^{-5}	6.79×10^{-8}	4.73×10^{-6}	4.80×10^{-2}
HN(ONS)	6.75×10^{-8}	4.86×10^{-2}	6.76×10^{-8}	1.93×10^{-2}	6.78×10^{-8}	2.23×10^{-2}	6.02×10^{-8}	1.62×10^{-7}
N(GNS)	7.80×10^{-8}	2.05×10^{-6}	6.72×10^{-8}	6.74×10^{-8}	6.74×10^{-8}	6.75×10^{-8}	$2.40 \times 10^{-1}\varnothing$	1.22×10^{-2}
HN(GNS)	1.43×10^{-2}	2.10×10^{-3}	4.59×10^{-4}	6.77×10^{-8}	2.68×10^{-6}	6.78×10^{-8}	1.10×10^{-3}	$5.39 \times 10^{-2}\varnothing$
N(OGN)	7.54×10^{-7}	5.64×10^{-6}	7.44×10^{-8}	6.41×10^{-8}	6.44×10^{-8}	6.42×10^{-8}	$3.76 \times 10^{-1}\varnothing$	6.31×10^{-5}
HN(OGN)	4.90×10^{-2}	5.60×10^{-3}	3.15×10^{-2}	6.78×10^{-8}	6.21×10^{-4}	6.78×10^{-8}	1.48×10^{-5}	$1.89 \times 10^{-1}\varnothing$

TABLE B.18: Task performance Statistical Tests for 5vs4 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	6.18×10^{-4}	4.30×10^{-3}	3.45×10^{-6}	6.77×10^{-8}	2.06×10^{-6}	6.77×10^{-8}	1.54×10^{-2}	7.92×10^{-7}
HN(NS)	3.90×10^{-3}	$3.23 \times 10^{-1}\varnothing$	1.54×10^{-2}	6.59×10^{-5}	1.93×10^{-2}	7.55×10^{-6}	7.75×10^{-7}	4.68×10^{-2}
N(OS)	9.00×10^{-3}	5.3×10^{-3}	3.41×10^{-6}	6.66×10^{-8}	2.32×10^{-6}	6.67×10^{-8}	9.97×10^{-4}	2.03×10^{-6}
HN(OS)	6.69×10^{-8}	7.80×10^{-1}	6.66×10^{-8}	4.11×10^{-2}	6.77×10^{-8}	2.10×10^{-3}	6.60×10^{-8}	6.76×10^{-8}
N(ONS)	1.55×10^{-2}	1.08×10^{-2}	1.30×10^{-3}	6.77×10^{-8}	4.59×10^{-4}	6.78×10^{-8}	3.24×10^{-5}	8.27×10^{-5}
HN(ONS)	6.70×10^{-8}	$8.60 \times 10^{-1}\varnothing$	6.67×10^{-8}	1.02×10^{-2}	6.77×10^{-8}	1.80×10^{-3}	6.61×10^{-8}	6.77×10^{-8}
N(GNS)	3.37×10^{-7}	3.98×10^{-6}	6.67×10^{-8}	6.77×10^{-8}	6.78×10^{-8}	6.78×10^{-8}	4.73×10^{-2}	6.77×10^{-8}
HN(GNS)	4.45×10^{-2}	1.93×10^{-2}	1.40×10^{-2}	9.12×10^{-8}	4.60×10^{-2}	7.87×10^{-8}	3.30×10^{-3}	2.94×10^{-2}
N(OGN)	8.76×10^{-7}	6.47×10^{-5}	6.44×10^{-8}	6.53×10^{-8}	6.54×10^{-8}	6.54×10^{-8}	$9.24 \times 10^{-2}\varnothing$	7.59×10^{-8}
HN(OGN)	3.6×10^{-2}	2.23×10^{-2}	4.56×10^{-2}	9.12×10^{-8}	4.74×10^{-2}	6.77×10^{-8}	1.32×10^{-2}	$7.46 \times 10^{-2}\varnothing$

TABLE B.19: Task performance Statistical Tests for 6vs4 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(NS)	1.10×10^{-3}	2.35×10^{-6}	6.78×10^{-8}	6.78×10^{-8}	9.15×10^{-8}	6.77×10^{-8}	2.80×10^{-3}	5.61×10^{-4}
HN(NS)	6.21×10^{-4}	2.08×10^{-2}	1.06×10^{-2}	1.30×10^{-3}	1.67×10^{-2}	2.91×10^{-5}	1.89×10^{-7}	2.74×10^{-2}
N(OS)	5.62×10^{-4}	1.37×10^{-6}	2.05×10^{-6}	6.75×10^{-8}	1.80×10^{-6}	6.75×10^{-8}	3.02×10^{-4}	1.78×10^{-4}
HN(OS)	6.78×10^{-8}	6.78×10^{-3}	6.78×10^{-8}	6.36×10^{-3}	6.77×10^{-8}	1.55×10^{-2}	6.69×10^{-8}	1.05×10^{-7}
N(ONS)	$5.98 \times 10^{-2}\varnothing$	2.04×10^{-5}	2.39×10^{-2}	6.77×10^{-8}	4.70×10^{-3}	6.77×10^{-8}	6.83×10^{-7}	5.30×10^{-3}
HN(ONS)	6.79×10^{-8}	5.31×10^{-3}	6.77×10^{-8}	$5.97 \times 10^{-1}\varnothing$	6.78×10^{-8}	3.15×10^{-2}	6.70×10^{-8}	6.73×10^{-8}
N(GNS)	6.78×10^{-8}	6.78×10^{-8}	6.77×10^{-8}	6.76×10^{-8}	6.77×10^{-8}	6.76×10^{-8}	8.57×10^{-3}	6.72×10^{-8}
HN(GNS)	4.86×10^{-2}	5.88×10^{-5}	2.62×10^{-2}	2.21×10^{-7}	1.64×10^{-2}	9.12×10^{-8}	1.10×10^{-3}	1.07×10^{-2}
N(OGN)	6.79×10^{-8}	6.79×10^{-8}	6.78×10^{-8}	6.77×10^{-8}	6.78×10^{-8}	6.77×10^{-8}	$2.85 \times 10^{-1}\varnothing$	9.08×10^{-8}
HN(OGN)	2.50×10^{-2}	2.47×10^{-4}	3.90×10^{-2}	4.52×10^{-7}	4.51×10^{-2}	9.12×10^{-8}	8.30×10^{-3}	$8.60 \times 10^{-1}\varnothing$

TABLE B.20: Task performance Statistical Tests for 6vs5 Keep-Away. Statistical significance test comparisons of NE method variant evolved with behavior transfer versus the same without behavior transfer. Shown across y-axis are the NE variants with behavior transfer and x-axis are the NE variants without behavior transfer.

Appendix C

Solution Complexity -Statistical Tests

C.1 Pair-wise Statistical Test Comparisons

This section presents a set of statistical tests results of solution complexity comparison between behaviors evolved with NEAT and HyperNEAT for all given search variants (NS, OS, ONS, OGN and GNS) and for keep-away tasks of increasing complexity (that is, 4vs3, 5vs3, 5vs4, 6vs4 and 6vs5). This tests if there was a statistically significant difference between solution complexity results from the given NE variants (given behavior transfer). A pair-wise Mann-Whitney u test with 95% confidence interval (p-values < 0.05) was performed between the solution complexity results. The null hypothesis states that topological complexity do not significantly differ between the given method variants. Thus, p-values of less than a threshold of 0.05 (Mann-Whitney u test), rejects this hypothesis. The values indicated with symbol $\varnothing$ having p-value more than 0.05 accepted this null hypothesis, suggesting that there is no statistical significance between the given topological complexity results.

Table C.1, C.2, C.3, C.4 and C.5 shows 43 different statistical test results comparing topological complexity for given method variants evolved with behavior transfer for a range of given keep-away tasks. The null hypothesis states that there is no significant difference between topological complexity results for behaviors evolved each of the compared variants.

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(OS)	2.6×10^{-3}	$9.20 \times 10^{-1}\varnothing$	–	$5.56 \times 10^{-2}\varnothing$	2.51×10^{-2}	3.67×10^{-2}	6.30×10^{-3}	6.24×10^{-8}
HN(OS)	1.16×10^{-4}	1.56×10^{-2}	$5.56 \times 10^{-2}\varnothing$	––	1.16×10^{-4}	1.56×10^{-2}	1.23×10^{-8}	6.19×10^{-8}
N(ONS)	$1.16 \times 10^{-1}\varnothing$	1.61×10^{-2}	2.51×10^{-2}	1.31×10^{-4}	––	2.50×10^{-3}	1.19×10^{-5}	6.30×10^{-8}
HN(ONS)	4.68×10^{-4}	$2.60 \times 10^{-1}\varnothing$	3.70×10^{-2}	$3.60 \times 10^{-1}\varnothing$	2.50×10^{-3}	––	3.64×10^{-2}	6.10×10^{-8}
N(GNS)	2.93×10^{-5}	6.68×10^{-4}	6.30×10^{-3}	1.20×10^{-2}	1.19×10^{-5}	3.64×10^{-2}	––	9.69×10^{-8}
HN(GNS)	6.21×10^{-8}	6.08×10^{-4}	6.24×10^{-8}	6.19×10^{-8}	6.32×10^{-8}	6.10×10^{-8}	9.69×10^{-8}	––
N(OGN)	3.22×10^{-4}	9.70×10^{-3}	5.71×10^{-3}	$7.54 \times 10^{-2}\varnothing$	9.94×10^{-5}	$1.62 \times 10^{-1}\varnothing$	$2.76 \times 10^{-1}\varnothing$	6.18×10^{-8}
HN(OGN)	1.58×10^{-5}	4.79×10^{-5}	2.10×10^{-3}	9.13×10^{-3}	1.29×10^{-6}	1.02×10^{-2}	$8.92 \times 10^{-1}\varnothing$	6.12×10^{-8}

TABLE C.1: Solution Complexity for 4vs3 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants solution (topological) complexity comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol $\varnothing$, indicates not significantly different

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(OS)	1.50×10^{-3}	$5.20 \times 10^{-2}\varnothing$	–	$4.21 \times 10^{-1}\varnothing$	3.68×10^{-2}	$7.33 \times 10^{-2}\varnothing$	2.26×10^{-2}	5.65×10^{-8}
HN(OS)	2.72×10^{-5}	3.11×10^{-2}	$4.21 \times 10^{-1}\varnothing$	$--$	4.00×10^{-3}	$1.99 \times 10^{-1}\varnothing$	5.81×10^{-3}	6.25×10^{-8}
N(ONS)	$1.63 \times 10^{-1}\varnothing$	$1.54 \times 10^{-1}\varnothing$	3.68×10^{-2}	4.00×10^{-3}	$--$	3.51×10^{-2}	1.10×10^{-4}	6.36×10^{-8}
HN(ONS)	2.21×10^{-4}	$3.33 \times 10^{-1}\varnothing$	$7.33 \times 10^{-2}\varnothing$	$1.99 \times 10^{-1}\varnothing$	3.51×10^{-2}	$--$	3.70×10^{-3}	6.19×10^{-8}
N(GNS)	2.15×10^{-6}	3.35×10^{-4}	2.26×10^{-2}	5.81×10^{-3}	1.10×10^{-4}	3.70×10^{-3}	$--$	6.21×10^{-8}
HN(GNS)	6.33×10^{-8}	6.10×10^{-8}	5.65×10^{-8}	6.25×10^{-8}	6.35×10^{-8}	6.19×10^{-8}	9.21×10^{-8}	$--$
N(OGN)	1.53×10^{-5}	2.38×10^{-2}	$1.49 \times 10^{-1}\varnothing$	$7.24 \times 10^{-1}\varnothing$	1.80×10^{-3}	$1.08 \times 10^{-1}\varnothing$	$1.18 \times 10^{-1}\varnothing$	6.22×10^{-8}
HN(OGN)	3.84×10^{-7}	2.42×10^{-5}	8.07×10^{-4}	1.04×10^{-2}	8.27×10^{-6}	1.24×10^{-4}	$7.66 \times 10^{-1}\varnothing$	6.33×10^{-8}

TABLE C.2: Solution Complexity for 5vs3 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants solution (topological) complexity comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol $\varnothing$, indicates not significantly different

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(OS)	3.79×10^{-2}	$7.75 \times 10^{-1}\varnothing$	–	$3.10 \times 10^{-1}\varnothing$	4.06×10^{-2}	$8.60 \times 10^{-1}\varnothing$	9.20×10^{-3}	6.35×10^{-8}
HN(OS)	5.56×10^{-5}	1.80×10^{-3}	$3.10 \times 10^{-1}\varnothing$	$--$	5.76×10^{-6}	1.00×10^{-2}	2.50×10^{-3}	6.21×10^{-8}
N(ONS)	$6.15 \times 10^{-1}\varnothing$	1.20×10^{-3}	4.06×10^{-2}	5.67×10^{-6}	$--$	9.93×10^{-4}	2.61×10^{-7}	6.35×10^{-8}
HN(ONS)	7.19×10^{-4}	$7.42 \times 10^{-1}\varnothing$	$8.64 \times 10^{-1}\varnothing$	1.00×10^{-2}	9.93×10^{-4}	$--$	2.43×10^{-5}	6.14×10^{-8}
N(GNS)	1.48×10^{-6}	1.74×10^{-6}	9.20×10^{-3}	2.50×10^{-3}	2.61×10^{-7}	2.43×10^{-5}	$--$	6.36×10^{-8}
HN(GNS)	6.23×10^{-8}	6.08×10^{-8}	6.35×10^{-8}	6.21×10^{-8}	6.35×10^{-8}	6.14×10^{-8}	6.36×10^{-8}	$--$
N(OGN)	1.35×10^{-4}	6.55×10^{-3}	$3.84 \times 10^{-1}\varnothing$	$4.21 \times 10^{-1}\varnothing$	4.54×10^{-5}	$1.09 \times 10^{-1}\varnothing$	6.10×10^{-3}	6.07×10^{-8}
HN(OGN)	3.21×10^{-6}	7.66×10^{-8}	6.07×10^{-3}	4.65×10^{-2}	9.59×10^{-7}	3.52×10^{-4}	$4.29 \times 10^{-1}\varnothing$	6.09×10^{-8}

TABLE C.3: Solution Complexity for 5vs4 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants solution (topological) complexity comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol $\varnothing$, indicates not significantly different

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(OS)	7.87×10^{-5}	$7.75 \times 10^{-1}\varnothing$	–	2.02×10^{-2}	4.06×10^{-2}	8.70×10^{-3}	8.49×10^{-3}	6.58×10^{-8}
HN(OS)	5.70×10^{-7}	$1.43 \times 10^{-1}\varnothing$	2.02×10^{-2}	$--$	1.89×10^{-5}	2.74×10^{-2}	3.16×10^{-2}	6.49×10^{-8}
N(ONS)	$4.21 \times 10^{-1}\varnothing$	6.30×10^{-3}	8.70×10^{-3}	1.89×10^{-5}	$--$	5.60×10^{-3}	8.68×10^{-7}	6.32×10^{-8}
HN(ONS)	5.54×10^{-5}	$7.44 \times 10^{-1}\varnothing$	8.64×10^{-3}	2.74×10^{-2}	5.60×10^{-3}	$--$	2.39×10^{-4}	6.39×10^{-8}
N(GNS)	1.77×10^{-7}	1.44×10^{-4}	2.87×10^{-4}	3.16×10^{-2}	8.68×10^{-7}	2.39×10^{-4}	$--$	6.35×10^{-8}
HN(GNS)	6.57×10^{-8}	6.48×10^{-8}	6.58×10^{-8}	6.49×10^{-8}	6.32×10^{-8}	6.39×10^{-8}	6.36×10^{-8}	$--$
N(OGN)	2.56×10^{-7}	4.29×10^{-2}	7.94×10^{-3}	$7.14 \times 10^{-1}\varnothing$	2.09×10^{-4}	6.24×10^{-3}	3.52×10^{-2}	6.41×10^{-8}
HN(OGN)	1.28×10^{-7}	2.59×10^{-4}	7.29×10^{-4}	2.37×10^{-2}	1.37×10^{-6}	7.58×10^{-4}	3.15×10^{-2}	6.55×10^{-8}

TABLE C.4: Solution Complexity for 6vs4 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants solution (topological) complexity comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol $\varnothing$, indicates not significantly different

Method	N(NS)	HN(NS)	N(OS)	HN(OS)	N(ONS)	HN(ONS)	N(GNS)	HN(GNS)
N(OS)	1.56×10^{-2}	$9.78 \times 10^{-1}\varnothing$	–	3.19×10^{-2}	7.20×10^{-3}	6.30×10^{-3}	4.56×10^{-5}	6.34×10^{-8}
HN(OS)	2.70×10^{-4}	1.23×10^{-2}	3.19×10^{-2}	$--$	2.43×10^{-5}	$1.34 \times 10^{-1}\varnothing$	3.90×10^{-3}	6.33×10^{-8}
N(ONS)	$7.86 \times 10^{-1}\varnothing$	5.30×10^{-3}	7.20×10^{-3}	2.43×10^{-5}	$--$	5.20×10^{-3}	7.49×10^{-7}	6.49×10^{-8}
HN(ONS)	3.90×10^{-3}	$6.14 \times 10^{-1}\varnothing$	$6.35 \times 10^{-1}\varnothing$	$1.34 \times 10^{-1}\varnothing$	5.20×10^{-3}	$--$	8.49×10^{-5}	6.42×10^{-8}
N(GNS)	6.65×10^{-7}	6.81×10^{-6}	4.56×10^{-5}	3.90×10^{-3}	7.49×10^{-7}	8.49×10^{-5}	$--$	6.35×10^{-8}
HN(GNS)	6.42×10^{-8}	6.26×10^{-8}	6.34×10^{-8}	6.34×10^{-8}	6.49×10^{-8}	6.42×10^{-8}	6.36×10^{-8}	$--$
N(OGN)	7.70×10^{-5}	1.15×10^{-2}	2.34×10^{-2}	$8.49 \times 10^{-1}\varnothing$	3.51×10^{-5}	7.67×10^{-3}	7.40×10^{-3}	6.42×10^{-8}
HN(OGN)	1.79×10^{-5}	3.68×10^{-4}	2.60×10^{-3}	$2.05 \times 10^{-1}\varnothing$	4.90×10^{-6}	8.4×10^{-3}	9.42×10^{-3}	6.44×10^{-8}

TABLE C.5: Solution Complexity for 6vs5 Keep-Away. Statistical significance test with 95% confidence interval (p <0.05, Mann-Whitney u test), for NE method variants solution (topological) complexity comparisons. Where, N and HN represents NEAT and HyperNEAT methods, respectively and a symbol $\varnothing$, indicates not significantly different

Appendix D

Cohen's Effect Size - Practical Tests

D.1 Pairwise Practical Tests Comparisons

Cohen (1988) suggested a measure for practical significance between a pair of samples, called the Cohen's *d* effect size. The probability of null hypothesis testing positive by chance (that is, evaluating p-values alone) decreases with increase in effect size. According to Cohen (1988), a Cohen's d effect score of 0.2 represented small effect size, 0.5 represented medium effect size and 0.8 represented large effect size. This means if the pair-wise Cohen's d effect size was 0.2, the difference was considered trivial despite the p-value score less than the set threshold (Cohen, 1988). Cohen's d effect size of 0.8 indicates a nonoverlap of 47% in the two distributions and that more than 1.7 indicates a nonoverlap of more than 75% in the two distributions. Cohen's d effect size score of 0 indicates a complete overlap (0% nonoverlap) of scores between the two distributions, suggesting the two distributions are practically not different.

Pair	**HyperNEAT**					**NEAT**				
Comparison	4vs3	5vs3	5vs4	6vs4	6vs5	4vs3	5vs3	5vs4	6vs4	6vs5
NS-OS	1.64	0.71	0.83	0.67	0.88	0.50	1.37	0.39	0.54	0.99
NS-ONS	1.94	1.25	0.88	0.85	1.31	1.17	1.74	1.11	0.72	1.09
NS-GNS	2.16	1.26	1.03	0.71	1.40	2.08	2.54	2.21	1.90	2.95
NS-OGN	1.52	0.83	0.88	0.86	1.35	2.66	2.11	2.01	1.76	2.64
OS-ONS	0.64	1.33	0.1	0.40	0.48	1.09	0.52	1.15	0.30	0.34
OS-GNS	5.74	2.96	2.89	2.93	2.88	2.54	3.49	2.80	2.65	5.52
OS-OGN	4.76	3.78	2.97	4.38	3.18	2.07	2.77	2.29	2.06	4.81
ONS-GNS	4.98	4.71	2.99	3.49	3.71	2.87	3.69	3.28	2.72	4.98
ONS-OGN	4.26	3.94	3.09	5.55	4.39	4.32	3.42	3.31	2.94	4.90
GNS-OGN	1.05	0.71	0.29	0.29	0.33	0.43	0.59	0.42	0.53	0.57

TABLE D.1: NEAT and HyperNEAT variants task performance pair-wise practical significance comparisons based on Cohen's effect size (d values).

Table D.1 shows a comparative analysis of pair-wise Cohen's d effect size scores for different HyperNEAT variants task performance in a given range of keep-away tasks. Indicated on the left column are the methods compared, for example, NS-ONS compute a pair-wise comparison of the practical statistical difference between NS and ONS

variants (NEAT and HyperNEAT). The pair-wise comparison of OS and ONS variants shows an average Cohen's d effect size of 0.59, suggesting a moderate practical significance with 33% standard nonoverlap (Cohen, 1988). A significantly high average effect size (Cohen's d > 4.25) observed between ONS and OGN variants, indicating a significantly high task performance difference. An relative low average Cohen's d effect score (Cohen's d < 0.33) was observed between GNS and OGN variants of HyperNEAT, indicating that the task performance difference is less significant.

	HyperNEAT					**NEAT**				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.43	1.50	1.76	0.83	0.44	0.99	1.25	1.56	0.60	0.36
OS	0.66	1.16	1.54	3.68	2.86	0.99	1.25	1.56	0.60	0.36
ONS	1.01	0.64	1.11	4.06	3.27	0.91	1.57	2.08	1.40	1.52
GNS	2.40	6.12	4.73	0.76	1.50	2.64	10.63	3.49	0.40	0.95
OGN	2.27	5.51	4.94	0.32	1.26	2.65	3.38	3.70	0.16	0.69

TABLE D.2: HyperNEAT variants task performance pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 4vs3 keep-away.

Table D.2, D.3, D.4, D.5 and D.6 shows pairwise Cohen's d effect size results for NEAT and HyperNEAT variants, based on task performance results. Results analysis indicate significantly higher values comparing the task performance yielded by NE methods evolved with and without behavior transfer, for example, NEAT and HyperNEAT ONS variants in 6vs5 keep-away task shows significantly high effect size (Cohen's d=0.83 and d=1.15, respectively).

	HyperNEAT					**NEAT**				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.51	1.74	2.45	0.88	0.32	0.84	1.38	1.55	0.77	0.38
OS	0.37	0.64	1.69	2.76	2.04	0.91	1.88	2.13	1.46	0.96
ONS	0.76	0.14	0.96	3.48	2.76	0.26	1.09	1.33	1.81	1.37
GNS	1.68	6.01	6.57	0.63	1.45	4.22	9.09	5.72	0.82	1.58
OGN	1.32	4.23	5.32	0.15	0.82	1.85	2.49	2.65	0.13	0.43

TABLE D.3: NEAT and HyperNEAT variants task performance pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 5vs3 keep-away.

Table D.7, D.7, D.7, D.7 and D.7 shows pairwise Cohen's d effect size results for NEAT and HyperNEAT variants, based on efficiency values. Results analysis indicate significantly higher values comparing method efficiency yielded by NE methods evolved with and without behavior transfer, for example, NEAT and HyperNEAT ONS variants in 6vs5 keep-away task shows significantly high effect size (Cohen's d=0.87 and d=0.40, respectively).

	HyperNEAT					NEAT				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.43	1.73	1.80	0.72	0.52	1.41	1.93	2.44	0.75	0.40
OS	0.30	1.17	1.29	2.32	2.38	1.90	3.15	3.95	0.97	0.57
ONS	0.45	0.80	0.91	2.47	2.52	0.52	1.09	1.84	1.87	1.62
GNS	1.58	4.20	4.98	0.70	1.21	3.28	7.74	5.09	0.40	0.96
OGN	1.41	4.56	4.75	0.42	0.90	2.50	3.15	3.63	0.22	0.66

TABLE D.4: HyperNEAT variants task performance pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 5vs4 keep-away.

	HyperNEAT					NEAT				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.34	1.72	2.07	0.86	0.56	1.27	2.12	2.21	0.81	0.38
OS	0.41	0.62	1.11	4.04	2.51	0.85	1.99	2.06	1.48	1.26
ONS	0.44	0.61	1.14	4.65	2.70	0.33	1.24	1.40	1.90	1.84
GNS	1.09	4.06	3.94	0.69	0.75	3.23	6.07	4.98	0.54	1.49
OGN	0.97	2.65	3.00	0.42	0.53	2.44	3.61	3.61	0.17	0.85

TABLE D.5: HyperNEAT variants task performance pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 6vs4 keep-away.

	HyperNEAT					NEAT				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.44	1.35	1.86	0.87	0.76	1.22	2.26	2.27	1.16	0.80
OS	0.77	0.19	0.83	3.13	3.78	1.27	2.57	2.52	1.39	1.00
ONS	0.83	0.15	0.83	3.39	4.33	0.19	1.07	1.15	2.39	2.08
GNS	1.62	3.03	3.59	0.46	0.79	3.34	14.22	4.98	0.81	1.34
OGN	1.33	2.27	2.78	0.28	0.50	3.06	5.36	4.93	0.27	0.83

TABLE D.6: HyperNEAT variants task performance pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 6vs5 keep-away.

Table D.12 shows pair-wise Cohen's d effect size comparing method efficiency yielded by NE variants evolved with behavior transfer. Results indicate that most of method efficiency between NE variants are significantly different, with exception of a few, such as GNS vs OGN in keep-away 6vs4 and 6vs5 (NEAT) with limited effect size (Cohen's d=0.13 and d=0.12, respectively), and ONS vs OS in keep-away 5vs3 (NEAT) and ONS vs OS in 6vs4 keep-away (HyperNEAT) with (Cohen's d=0.12 and d=0.14, respectively).

	HyperNEAT					NEAT				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.13	0.81	0.95	0.47	0.32	0.71	0.70	0.32	0.53	0.49
OS	0.84	0.54	0.85	3.21	2.94	0.41	0.37	0.25	1.83	1.36
ONS	1.32	0.37	0.33	3.23	3.01	0.42	0.32	0.15	1.01	0.90
GNS	1.46	3.28	3.64	0.70	1.11	1.56	4.07	1.40	0.24	0.27
OGN	1.25	3.43	3.45	0.36	0.78	1.79	2.42	1.69	0.67	0.53

TABLE D.7: HyperNEAT variants efficiency pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 4vs3 keep-away.

	HyperNEAT					NEAT				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.36	1.50	1.46	0.78	0.57	0.59	0.38	0.56	0.88	0.50
OS	0.40	0.63	0.58	2.91	2.93	0.37	0.10	0.30	1.73	0.95
ONS						0.40	0.18	0.34	1.46	0.84
GNS	1.11	4.59	3.06	0.27	0.60	2.16	6.57	2.51	0.74	0.77
OGN	1.08	2.71	2.65	0.29	0.57	1.82	2.49	2.20	0.11	0.33

TABLE D.8: HyperNEAT variants efficiency pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 5vs3 keep-away.

	HyperNEAT					NEAT				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.81	2.19	1.62	0.61	0.47	1.02	0.27	0.44	0.69	0.32
OS	0.15	1.11	0.47	1.38	1.48	1.62	0.16	0.42	1.89	0.89
ONS	0.19	1.46	0.72	1.54	1.92	0.55	1.38	0.82	3.57	1.86
GNS	1.47	3.34	2.63	0.22	0.68	2.97	1.90	1.94	0.17	0.48
OGN	1.29	2.45	1.99	0.17	0.49	2.44	1.32	1.54	0.27	0.29

TABLE D.9: HyperNEAT variants efficiency pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 5vs4 keep-away.

	HyperNEAT					NEAT				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.34	1.35	1.27	0.41	0.45	1.03	0.24	0.18	0.63	0.92
OS	0.57	0.85	0.73	2.35	2.87	0.61	0.32	0.43	1.36	1.79
ONS	0.51	0.99	0.92	2.26	2.78	0.41	0.67	0.93	1.96	2.60
GNS	1.09	4.02	2.27	0.49	0.49	2.55	1.32	1.49	0.52	0.26
OGN	1.28	3.25	3.42	0.59	0.63	2.30	1.24	1.33	0.43	0.18

TABLE D.10: HyperNEAT variants efficiency pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 6vs4 keep-away.

	HyperNEAT					**NEAT**				
Method	**NS**	**OS**	**ONS**	**GNS**	**OGN**	**NS**	**OS**	**ONS**	**GNS**	**OGN**
NS	0.42	0.90	0.82	0.40	0.41	1.86	0.55	0.66	0.46	0.15
OS	0.10	1.10	0.96	1.21	2.94	1.22	0.15	0.13	1.19	0.74
ONS	0.17	1.01	0.87	1.23	2.84	0.90	0.67	0.40	1.80	1.42
GNS	1.32	2.99	2.49	0.31	0.65	2.23	1.14	1.17	0.29	0.69
OGN	1.37	2.41	2.36	0.41	0.74	2.01	0.83	0.92	0.20	0.40

TABLE D.11: HyperNEAT variants efficiency pair-wise practical significance comparisons based on Cohen's effect size (d values), behavior transfer vs no behavior transfer in 6vs5 keep-away.

Efficiency	**HyperNEAT**					**NEAT**				
Comparison	4vs3	5vs3	5vs4	6vs4	6vs5	4vs3	5vs3	5vs4	6vs4	6vs5
NS-OS	1.05	0.67	0.76	0.97	0.46	0.25	0.44	1.41	0.83	1.13
NS-ONS	1.24	0.65	0.22	0.87	0.36	0.57	0.24	1.01	1.06	0.88
NS-GNS	0.98	1.00	1.26	0.85	0.92	1.51	1.76	3.07	1.97	2.01
NS-OGN	0.75	0.84	1.23	0.92	1.11	1.34	1.13	1.91	2.49	1.69
OS-ONS	0.30	0.12	0.90	0.41	0.39	0.62	0.42	0.33	0.14	0.25
OS-GNS	2.86	3.53	2.68	2.68	1.70	2.59	2.46	1.77	0.86	0.89
OS-OGN	2.53	3.15	2.39	2.71	1.59	2.64	1.94	1.14	1.13	0.43
ONS-GNS	2.93	3.37	2.19	2.80	1.62	1.40	2.14	1.99	0.94	0.98
ONS-OGN	2.71	3.44	3.15	3.41	3.05	1.09	1.19	1.10	1.32	0.59
GNS-OGN	0.36	0.38	0.35	0.13	0.12	0.15	0.30	0.47	0.29	0.48

TABLE D.12: HyperNEAT variants efficiency pair-wise practical significance comparisons based on Cohen's effect size (d values).

Appendix E

Task Performance Comparison

E.1 Task Performance Comparisons

Average normalized maximum task performance progression for HyperNEAT vs NEAT (given policy transfer), over 20 runs in each task. Evolution starts with a source task for 30 generations and then evolved behaviors are transfered to initiate evolution at the target task and evolution runs for 70 generations in the target task.

Comparison of TD methods (SARSA and QL-Learning) with NE methods (NEAT and HyperNEAT) for all keep-away tasks.

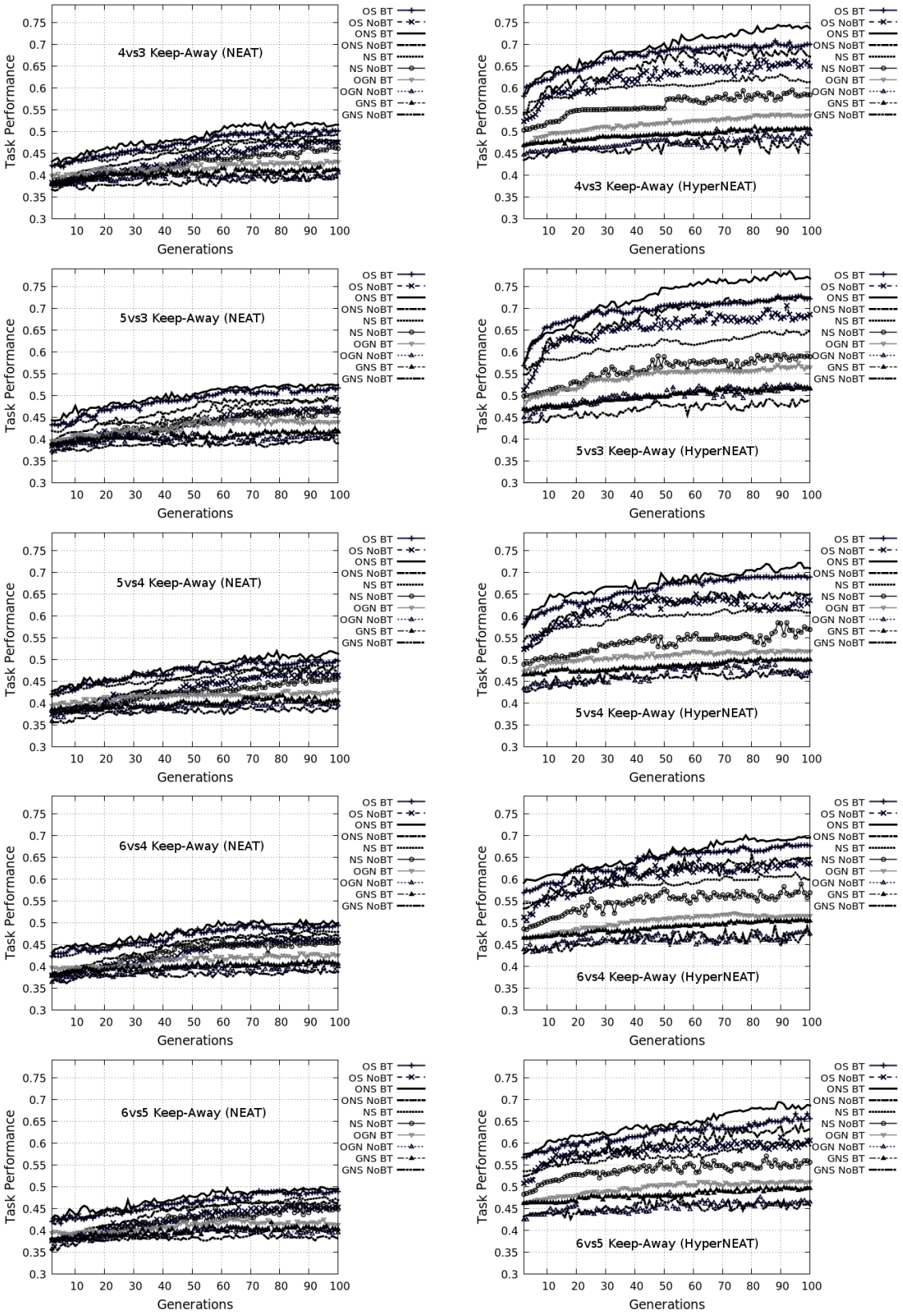


FIGURE E.1: Task performance progression graph. The graph shows progression of mean of normalized maximum task performance for all variants of NEAT vs HyperNEAT. Averages are calculated over 20 runs and for each target keep-away task.

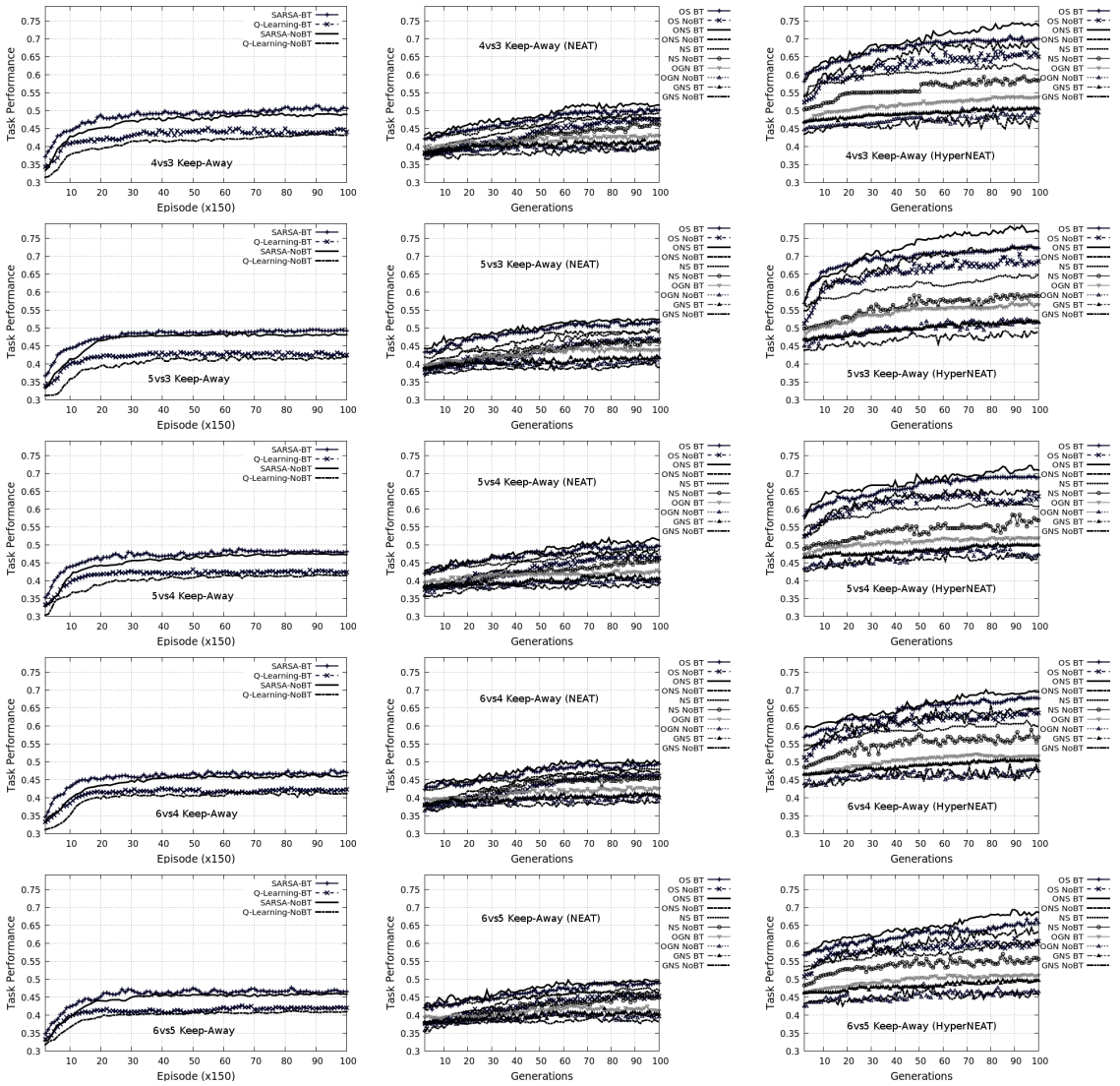


FIGURE E.2: RL versus NE method average task performance comparison for all keep-away tasks. Left: SARSA and Q-Learning (TD methods), Right: NEAT and HyperNEAT (NE methods).

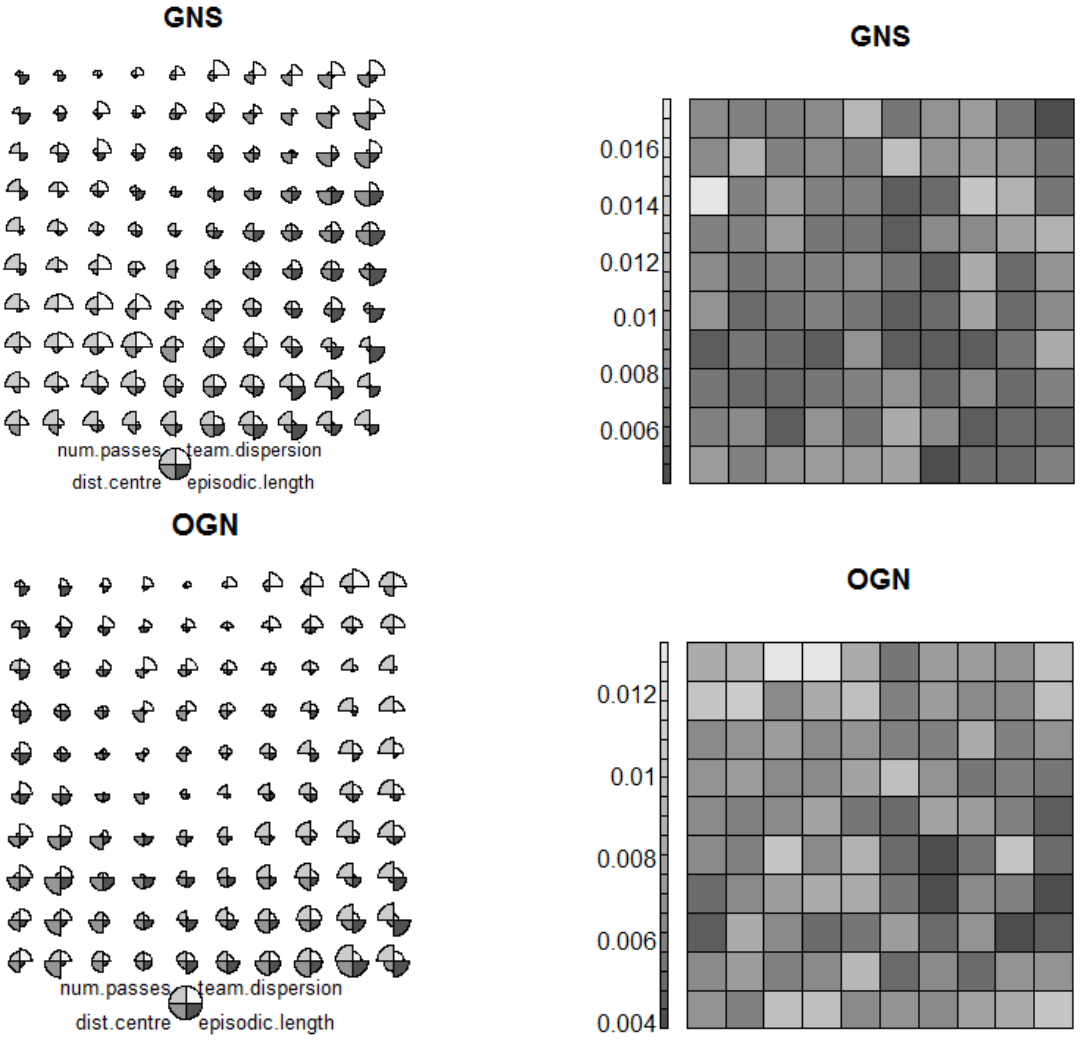


FIGURE E.3: Kohonen Self-Organizing Maps (SOMs) representing the explored behavior search space in Keep-away 4vs3 for given HyperNEAT variants (GNS and OGN) with behavior transfer. In each NE variant, left: shows codebook weight vector for data visualization and right: shows u-matrix representing boundaries between clusters.

Bibliography

1. Abul O, Polat F, and Alhajj RS. Multiagent reinforcement learning using function approximation. IEEE Transactions on Systems, Man, and Cybernetics.Part C: Applications and Reviews 2000;30:485–497.
2. Aditya Rawal Padmini Rajagopalan RM. Constructing Competitive and Cooperative Agent Behavior Using Coevolution. In: *IEEE conference on computational Intelligence and Games*. IEEE Press, 2010:107–114.
3. Altenberg L. The evolution of evolvability. In: *Advances in Genetic Programming*. Ed. by Kinnear K. Cambridge, USA: MIT Press, 1994:47–74.
4. Amigoni F and Schiaffonati V. Multiagent-Based Simulation in Biology. In: *Model-Based Reasoning in Science, Technology, and Medicine*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007:179–191.
5. Ammar H, Tuyls K, Taylor M, Driessens K, and Weiss G. Reinforcement Learning Transfer via Sparse Coding. In: *Proceedings of the eleventh international conference on autonomous agents and multiagent systems*. Valencia, Spain: AAAI, 2012:4–8.
6. Ammar H, Eaton E, Ruvolo P, and Taylor M. Online Multi-Task Learning for Policy Gradient Methods. In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. Beijing, China: JMLR Workshop and Conference Proceedings, 2014:1206–1214.
7. Antona M, Bousquet F, LePage C, Weber J, Karsenty A, and Guizol P. Economic Theory of Renewable Resource Management: A Multi-agent System Approach. In: *Multi-Agent Systems and Agent-Based Simulation: First International Workshop, MABS '98, Paris, France, July 4-6, 1998. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998:61–78.
8. Bahceci E and Miikkulainen R. Transfer of Evolved Pattern-Based Heuristics in Games. In: *Proceedings of the IEEE Symposium On Computational Intelligence and Games*. Perth, Australia: Morgan Kaufmann, 2008:220–227.
9. Baldassarre G, Nolfi S, and Parisi D. Evolving mobile robots able to display collective behavior. Artificial Life 2003;9:255–267.
10. Bellman R. Adaptive Control Processes. New Jersey, USA: Princeton University Press, 1961.
11. Bertsekas D. Dynamic Programming and Optimal Control. MA:Athena Scientific: Belmont, 2001.

12. Beyer HG. Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. Computer Methods in Applied Mechanics and Engineering 2000;186:239–267.
13. Bonabeau E, Theraulaz G, Arpin E, and Sardet E. The building behavior of lattice swarms. In: *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. Cambridge, USA: MIT Press, 1994:307–312.
14. Boutsioukis G, Partalas I, and Vlahavas I. Transfer Learning in Multi-Agent Reinforcement Learning Domains. In: *Recent Advances in Reinforcement Learning*. Springer, 2012:249–260.
15. Bradtke SJ and Duff MO. Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In: *Advances in Neural Information Processing Systems 7 Conference*. Denver, Colorado, USA, 1994:393–400.
16. Bryant B and Miikkulainen R. Neuro-evolution for Adaptive Teams. In: *Proceedings of the Congress on Evolutionary Computation*. Canberra, Australia: IEEE Press, 2003:2194–2201.
17. Busoniu L, Babuska R, and DeSchutter B. A Comprehensive Survey of Multiagent Reinforcement Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 2008;38:156–172.
18. Cao YU, Fukunaga AS, and Kahng AB. Cooperative mobile robotics: antecedents and directions. Autonomous Robotics 1997;4:1–23.
19. Carroll J and Seppi K. Task similarity measures for transfer in reinforcement learning task libraries. In: *Proceedings of 2005 IEEE International Joint Conference on Neural Networks*. QC, Canada: IEEE Press, 2005:803–808.
20. Chang Y han, Ho T, and Kaelbling LP. All learning is Local: Multi-agent Learning in Global Reward Games. In: *Advances in Neural Information Processing Systems 16*. MIT Press, 2004:807–814.
21. Christensen A and Dorigo M. Incremental Evolution of Robot Controllers for a Highly Integrated Task. In: *From Animals to Animats 9*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006:473–484.
22. Clune J, Ofria C, and Pennock R. How a generative encoding fares as problem-regularity decreases. Parallel Problem Solving from Nature 2008;5199:358–367.
23. Clune J, Beckmann B, Ofria C, and Pennock R. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In: *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*. Trondheim, Norway: IEEE, 2009:2764–2771.
24. Clune J, Beckmann B, McKinley P, and Ofria C. Investigating whether Hyperneat produces modular neural networks. In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. Portland, USA: ACM Press, 2010:635–642.
25. Clune J, Mouret JB, and Lipson H. The evolutionary origins of modularity. Proceedings of Royal Society B 2013;280.

26. Clune J, Stanley K, Pennock RT, and Ofria C. On the performance of indirect encoding across the continuum of regularity. IEEE Transactions on Evolutionary Computation 2011;15:346–367.
27. Cohen J. Statistical power analysis for the behavioral sciences (2nd ed.) Hillsdale, N.J: Lawrence Earlbaum Associates, 1988.
28. Cohn MJ, Patel K, Krumlauf R, Wilkinsont DG, Clarke JD, and Tickle C. HOX9 genes and vertebrate limb specification. Nature 1997;387:97–101.
29. Crepinsek M, Liu S, and Mernik M. Exploration and Exploitation in Evolutionary Algorithms. ACM Computing Surveys 2013;45(3):1–33.
30. Cuccu G and Gomez. F. When novelty is not enough. In: *Proceedings of the European Conference on the Applications of Evolutionary Computation (EvoApplications)*. Springer, 2011:234–243.
31. Cully A and Mouret JB. Evolving a Behavioral Repertoire for a Walking Robot. Evolutionary Computation 2016;24(1):59–88.
32. Cully A, Clune J, Tarapore D, and Mouret JB. Robots that can adapt like animals. Nature 2015;527:503–507.
33. Cully A, Clune J, Tarapore D, and Mouret JB. Robots that can adapt like animals. Nature 2015;521(1):503–507.
34. D'Ambroisio D, Lehman J, Risi S, and Stanley K. Evolving Policy Geometry for Scalable multi-agent learning. In: *Proceeding of 9th International Conference on Autonomous Agents and Multiagents Systems*. Richland, USA: International Foundation for Autonomous Agents Systems, 2010:731–738.
35. D'Ambrosio D and Stanley K. Generative Encoding for Multiagent Learning. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Atlanta, USA: ACM Press, 2008:819–826.
36. D'Ambrosio D and Stanley K. Scalable Multiagent Learning through Indirect Encoding of Policy Geometry. Evolutionary Intelligence Journal 2013;6:1–26.
37. D'Ambrosio D, Lehman J, Risi S, and Stanley K. Task switching in Multirobot Learning through Indirect Encoding. In: *Proceedings of the International Conference on intelligent Robots and Systems*. San Francisco, USA: IEEE, 2011:2802–2809.
38. Darwen P and Yao X. Automatic modularization by speciation. In: *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*. Piscataway, USA: IEEE Computer Society Press, 1996:88—93.
39. David L. Collective Behavior. Upper Saddle River, NJ: Prentice Hall, 2002.
40. Degrave J, Burm M, Kindermans P, Dambre J, and Wyffels F. Transfer learning of gaits on a quadrupedal robot. Adaptive Behavior 2015;23:9–19.
41. Didi S and Nitschke G. Hybridizing Novelty Search for Transfer Learning. In: *Proceedings of the IEEE Conference on Computational Intelligence and Robotics*. Greece: IEEE Press, 2016:1–8.
42. Didi S and Nitschke G. Multi-Agent Behavior-Based Policy Transfer. In: *Proceedings of the European Conference on the Applications of Evolutionary Computation*. Springer, 2016:181–197.

43. Didi S and Nitschke G. Neuro-Evolution for Multi-Agent Policy Transfer in RoboCup Keep-Away. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. Singapore: ACM, 2016:1281–1282.
44. Doncieux S and Mouret JB. Behavioral diversity measures for Evolutionary robotics. In: *In IEEE Congress on Evolutionary Computation (CEC)*. Barcelona, Spain: IEEE, 2010:1303–1310.
45. Doncieux S and Mouret JB. Beyond black-box optimization: a review of selective pressures for evolutionary robotics. Evolutionary Intelligence 2014;7(2):71–93.
46. Doncieux S, Mouret JB, Bredeche N, and Padois V. Evolutionary Robotics: Exploring New Horizons. In: *New Horizons in Evolutionary Robotics*. Berlin, Germany: Springer, 2011:3–25.
47. Drchal J, Koutnik J, and Snorek M. HyperNEAT controlled robots learn how to drive on roads in simulated environment. In: *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*. IEEE, 2009:1087–1092.
48. Dreżewski R. A model of co-evolution in multi-agent system. In: *Multi-Agent Systems and Applications III*. Berlin, Germany: Springer-Verlag, 2003:314–323.
49. Eiben A and Schippers C. On Evolutionary Exploration and Exploitation. Fundamental Informaticae 1998;35:35–50.
50. Eiben A and Smith J. Introduction to Evolutionary Computing. Berlin, Germany: Springer, 2003.
51. Fachantidis A, Partalas I, Taylor M, and Vlahavas I. Transfer Learning via Multiple Inter-Task Mappings. In: *Recent Advances in Reinforcement Learning*. Ed. by Sanner S and Hutter M. Berlin, Germany: Springer, 2011:225–236.
52. Ferguson K and Mahadevan S. Proto-transfer learning in markov decision processes using spectral methods. In: *International Conference on machine learning (ICML), Workshop on Transfer Learning*. Pittsburgh, 2006.
53. Ferguson K and Mahadevan S. Proto-transfer learning in markov decision processes using spectral methods. Tech. rep. Technical Report TR-08-23. University of Massachusetts-Amherst, 2008.
54. Fernando F and Manuela V. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. Hakodate, Japan: ACM, 2006:720–727.
55. Flannery B, Teukolsky S, and Vetterling W. Numerical Recipes. Cambridge, UK: Cambridge University Press, 1986.
56. Floreano D, Hauert S, Leven S, and Zufferey J. Evolutionary Swarms of Flying Robots. In: *International Symposium on Flying Insects and Robots*. Cambridge, USA: MIT Press, 2007:35–36.
57. Floreano D, Dürr P, and Mattiussi C. Neuroevolution: from architectures to learning. Evolutionary Intelligence 2008;1:47–62.
58. Gauci J and Stanley K. A case study on the critical role of geometric regularity in machine learning. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 2008:628–633.

59. Geva S and Sitte J. A cart-pole experiment benchmarking for trainable controllers. IEEE Control Systems Magazine. 1993;13:40–51.
60. Ghavamzadeh M, Mahadevan S, and Makar R. Hierarchical multi-agent reinforcement learning. Autonomous Agents and Multi-Agent Systems 2006;13(2):197–229.
61. Goldberg D and Richardson J. Genetic algorithms with sharing for multimodal function optimization. In: *Proceedings of the Second International Conference on Genetic Algorithms*. San Francisco, USA: Morgan Kaufmann, 1987:148—154.
62. Goldberg DE. Genetic algorithms and Walsh functions: Part 2, deception and its analysis. Complex Systems 1989;3:153–171.
63. Goldberg D and Holland J. Genetic Algorithms and Machine Learning. Machine Learning 1988;3:95–99.
64. Gomes J and Christensen A. Generic behavior similarity measures for evolutionary swarm robotics. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Amsterdam, the Netherlands: ACM Press, 2013:199–206.
65. Gomes J and Christensen A. Generic Behaviour Similarity Measures for Evolutionary Swarm Robotics. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2013:199–206.
66. Gomes J, Urbano P, and Christensen A. Evolution of swarm robotics systems with novelty search. Swarm Intelligence 2013;7:115–144.
67. Gomes J, Mariano P, and Christensen A. Avoiding convergence in cooperative coevolution with novelty search. In: *Proceedings of the International conference on Autonomous Agents and Multi-Agent Systems*. Paris, France: ACM, 2014:1149–1156.
68. Gomes J, Mariano P, and Christensen A. Novelty Search in Competitive Coevolution. In: *Proceedings of the International Conference on Parallel Problem Solving from Nature*. Ljubljana, Slovenia: Springer, 2014:233–242.
69. Gomes J, Mariano P, and Christensen A. Devising Effective Novelty Search Algorithms: A Comprehensive Empirical Study. In: *Proceedings of the Genetic Evolutionary Computation Conference*. 2015:943–950.
70. Gomes J, Duarte M, Mariano P, and Christensen A. Cooperative Coevolution of Control for a Real Multirobot System. In: *Proceedings of the 14th International Conference on Parallel Problem Solving from Nature (PPSN XIV)*. Edinburgh, UK: Springer, 2016:In press.
71. Gomes J, Mariano P, and Christensen A. Novelty-driven Cooperative Coevolution. Evolutionary Computation 2016;In Press.
72. Gomez F and Miikkulainen R. Incremental evolution of complex general behavior. Adaptive Behavior 1997;5:317–342.
73. Gomez F. Sustaining diversity using behavioral information distance. In: *Proceedings of the Genetic Evolutionary Computation Conference*. ACM, 2009:113–120.
74. Grabkovsky S, Birger K, and Lowell J. Comparison of NEAT and HyperNEAT Performance on a Strategic Decision-Making Problem. Genetic and Evolutionary Computing, International Conference on 2011:102–105.

75. Green C. Phased Searching with NEAT: Alternating Between Complexification and Simplification. Unpublished Manuscript. 2004.
76. Gruau F and Quatramaran K. Cellular encoding for interactive evolutionary robotics. In: *Proceeding of Fourth European conference on artificial life*. Cambridge: MIT Press, 1996:368–377.
77. Guestrin C, Lagoudakis M, and Parr R. Coordinated Reinforcement Learning. In: *Proceedings of the Nineteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002:227–234.
78. Haynes T and Sen S. Evolving Cooperation Strategies. In: *Proceedings of the First International Conference on Multi-Agent Systems*. Cambridge, USA: MIT Press, 1995:450–459.
79. Haynes T and Sen S. Co-adaptation in a team. International Journal of Computational Intelligence and Organizations 1996;1:1–20.
80. Haynes T and Sen S. Evolving behavioral strategies in predators and prey. In: *Adaptation and Learning in Multi-Agent Systems: Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 1996:113–126.
81. Hoffmann F. Fuzzy Logic Techniques for Autonomous Vehicle Navigation. In: *Studies in Fuzziness and Soft Computing Volume: The Role of Fuzzy Logic Control in Evolutionary Robotics*. Ed. by Driankov D and Saffiotti A. Heidelberg, Germany: Physica-Verlag HD, 2001:119–147.
82. Hoffmann F and Pfister G. Evolutionary learning of a fuzzy control rule base for an autonomous vehicle. In: *Proceedings of the Fifth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Granada, Spain: MIT Press, 1996:659–664.
83. Hornby G. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In: *Proceedings of the Conference on Genetic and Evolutionary Computation*. Washington DC, USA: ACM Press, 2005:1729–1736.
84. Hornby G, Lipson H, and Pollack J. Generative Representations for the Automated Design of Modular Physical Robots. IEEE Transactions on Robotics and Automation 2003;19(4):703–719.
85. Howard RA. Dynamic Programming and Markov Processes. Cambridge, MA: MIT Press, 1960.
86. Hu J and Wellman MP. Multiagent reinforcement learning:theoretical framework and an algorithm. In: *In Proceedings of the Fifteenth International Conference on Machine Learning*. San Francisco, California: ACM Press, 1998:242–250.
87. Jorgensen TG, Haynes BP, and Norlund C. Pruning Artificial Neural Networks Using Neural Complexity Measures. International Journal of Neural Systems 2008;18:389–403.
88. Kaelbling L, Littman M, and Moore A. Reinforcement learning: A survey. Journal of Artificial Intelligence 1996;4:237–285.
89. Kawai K, Ishiguro A, and Eggenberger P. Incremental Evolution of Neurocontrollers with a Diffusion-Reaction Mechanism of Neuromodulators. In:

Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Maui, Hawaii, USA: IEEE, 2001:2384–2391.

90. Kelly S and Heywood M. Genotype versus Behavioural Diversity for Teams of Programs under the 4-v-3 Keepaway Soccer Task. In: *Proceedings of Association for the Advancement of Artificial Intelligence*. n: AAAI, 2014:3110–3111.
91. Kistemaker S and Whiteson S. Critical factors in the performance of novelty search. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2011:965–972.
92. Kitano H. Designing neural networks using genetic algorithms with graph generation systems. Complex Systems Journal 1990;4:461–476.
93. Klahr D and Carver S. Cognitive objectives in a LOGO debugging curriculum: Instruction, learning and transfer. Cognitive Psychology 1988;20:362–404.
94. Klein J. Breve: A 3D Environment for the Simulation of Decentralized Systems and Artificial Life. In: *Proceedings of the Eighth International Conference on Artificial Life*. Cambridge, MA, USA: MIT Press, 2003:329–334.
95. Klein J and Spector L. 3D Multi-Agent Simulations in the breve Simulation Environment. In: *Artificial Life Models in Software*. London: Springer London, 2009:79–106.
96. Klopf AH. Brain function and adaptive systems. A Heterostatic theory. Tech. rep. Air Force Cambridge Research Laboratories, Bedford USA, 1972.
97. Knudson M and Tumer K. Policy Transfer in Mobile Robots using Neuro-Evolutionary Navigation. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Philadelphia, USA: ACM Press, 2012:1411–1412.
98. Kohl N and Miikkulainen R. Evolving Neural Networks for Fractured Domains. In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2008:1405–1412.
99. Kohl N and Miikkulainen R. Evolving Neural Networks for Strategic Decision-making Problems. Neural Networks 2009;22:326–337.
100. Kohonen T. Self-organizing Maps. Secaucus, NJ, USA: Springer-Verlag New York, Inc, 1997.
101. Konidaris G, Scheidwasser I, and Barto A. Transfer in Reinforcement Learning via Shared Features. Journal of Machine Learning Research 2012;13:1333–1371.
102. Kuyer L, Whiteson S, Bakker B, and Vlassis N. Multiagent Reinforcement Learning for Urban Traffic Control Using Coordination Graphs. Machine Learning and knowledge Discovery in Databases 2008;5211:656–671.
103. Lazaric A, Restelli M, and Bonarini A. Transfer of samples in batch reinforcement learning via clusterring. In: *Proceedings of the Twenty-Fifth International Conference on machine Learning*. Helsinki, Finland: ACM, 2008:544–551.
104. Lehman J and Stanley K. Exploiting open-endedness to solve problems through the search for novelty. In: *Proceedings of the International Conference on Artificial Life*. Winchester, UK: MIT Press, 2008:329–336.

105. Lehman J and Stanley K. Revising the evolutionary computation abstraction: minimal criteria novelty search. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2010:103–110.
106. Lehman J and Stanley K. Abandoning objectives: Evolution through the search for novelty alone. Evolutionary computation 2011;19(2):189–223.
107. Lehman J and Stanley K. Novelty search and the problem with objectives. In: *Genetic Programming in Theory and Practice IX*. Berlin, Germany: Springer, 2011:37–56.
108. Lehman J, Stanley K, and Miikkulainen R. Effective Diversity Maintenance in Deceptive Domains. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2013:215–222.
109. Lewis EB. A gene complex controlling segmentation in Drosophila. Nature 1978;276:565–570.
110. Liapis A, Yannakakis G, and Togelius J. Constrained novelty search: A study on game content generation. Evolutionary Computation 2015;23(1):101–129.
111. Lipson H. Principles of modularity, regularity, and hierarchy for scalable systems. In: *Proceedings of Genetic and Evolutionary Computation Conference Workshop on Modularity, Regularity, and Hierarchy in Evolutionary Computation*. Washington DC, USA, 2004:125–128.
112. Littman ML. Markov games as a framework for multi-agent learning. In: *In Proceedings of the Eleventh International Conference on Machine Learning*. San Francisco, California: ACM Press, 1994:157–163.
113. Littman ML. Value-function reinforcement learning in Markov games. Journal of Cognitive Systems Research 2001;2:55–66.
114. Littman M. Friend-or-foe Q-learning in general-sum games. In: *Proceeding of the 18th International Conference in Machine Learning*. San Francisco, USA: Morgan Kaufmann, 2001:322–328.
115. Lockett AJ, Chen CL, and Miikkulainen R. Evolving explicit opponent models in game playing. In: *Genetic and Evolutionary Computation Conference, GECCO 2007*. London,UK: ACM, 2007:2106–2113.
116. Lutz F. Generalization and Transfer Learning in Noise-affected Robot Navigation Tasks. In: *Proceedings of the Aritficial Intelligence 13th Portuguese Conference on Progress in Artificial Intelligence*. Berlin, Heidelberg: Springer-Verlag, 2007:508–519.
117. Maclin R and Shavlik JW. Creating advice-taking reinforcement learners. Journal of Machine Learning 1996;22:251–281.
118. Maclin R, Shavlik J, Torrey L, Walker T, and Wild E. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In: *Proceedings of the 20th National Conference on Artificial Intelligence*. Pennysylvania, USA: The MIT Press, 2005:819–824.
119. Matsubara H, Noda I, and Hiraki K. Learning of cooperative actions in multi-agent systems: a case study of pass and play in soccer. In: *Adaptation,*

Co-evolution, and Learning in Multi-agent Systems: Papers from the 1996 AAAI Spring Symposium. Boston, USA: AAAI Press, 1996:63–67.

120. Mckenney D and White T. Distributed and adaptive traffic signal control within a realistic traffic simulation. Journal of Engineering Applications of Artificial Intelligence 2013;26:574–583.
121. Meyerson E, Lehman J, and Miikkulainen R. Learning Behavior Characterization for Novelty Search. In: *Proceedings of the Genetic Evolutionary Computation Conference*. New York, NY, USA: ACM, 2016:149–156.
122. Miikkulainen R. Neuroevolution. Encyclopedia of Machine Learning 2010:716–720.
123. Miikkulainen R, Feasley E, Johnson L, et al. Multiagent Learning through Neuroevolution. In: *Advances in Computational Intelligence*. Ed. by Liu J, Alippi C, Bouchon-Meunier B, Greenwood G, and Abbass H. Berlin, Germany: Springer, 2012:24–46.
124. Miller G and Cliff D. Co-Evolution of Pursuit and Evasion I: Biological and Game-Theoretic Foundations (Tech. Rep. CSRP311). Brighton, England: University of Sussex: School of Cognitive and Computing Sciences, 1996.
125. Moriarty D. Symbiotic Evolution of Neural Networks in Sequential Decision Tasks. PhD thesis. Austin, Texas: Department of Computer Sciences, The University of Texas, 1997.
126. Moriarty D and Miikkulainen R. Learning Sequential Decision Tasks. In: *Advances in the Evolutionary Synthesis of Intelligent Agents*. Cambridge, USA: MIT Press, 1995:367–382.
127. Moriarty D and Miikkulainen R. Efficient Reinforcement Learning Through Symbiotic Evolution. Machine Learning 1996;22:11–33.
128. Moriarty D and Miikkulainen R. Hierarchical evolution of neural networks. In: *Proceedings of the 1998 IEEE Conference on Evolutionary Computation*. NJ, USA: IEEE Press Piscataway, 1998:428–433.
129. Moriguchi H and Honiden S. Sustaining Behavioral Diversity in NEAT. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, USA: ACM, 2010:611–618.
130. Moriguchi H and Honiden S. Sustaining Behavioral Diversity in NEAT. In: *Proceedings of the 12th annual conference on Genetic and Evolutionary Computation*. Portland, USA: ACM, 2010:611–618.
131. Morse G, Risi S, Snyder C, and Stanley K. Single-unit Pattern Generators for Quadruped Locomotion. In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2013:719–726.
132. Moshaiov A and Tal A. Family Bootstrapping: A Genetic Transfer Learning Approach for Onsetting the evolution for a Set of Realated Robotic Tasks. In: *Proceedings of the Congress on Evolutionary Computation*. IEEE Press, 2014:2801–2808.
133. Mouret JB and Clune J. Illuminating search spaces by mapping elites. arXiv preprint 2015.

134. Mouret JB and Doncieux S. Evolving modular neural-networks through exaptation. In: *Proceedings of IEEE Congress on Evolutionary Computation*. IEEE, 2009:1570–1577.
135. Mouret JB and Doncieux S. Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study. Evolutionary Computation 2012;20(1):91–133.
136. Mouret JB and Doncleux S. Using Behavioral Exploration Objectives to Solve Deceptive Problems in Neuro-evolution. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Montreal, Canada: ACM Press, 2009:627–634.
137. Mouret JB, Koos S, and Doncieux S. Crossing the Reality Gap: A Short Introduction to the Transferability Approach. In: *Proceedings of the International Conference on the Simulation and Synthesis of Living Systems (Artificial Life XIII)*. East Lansing, USA: MIT Press, 2012.
138. Ng A, Harada D, and Russell S. Policy invariance under reward transformations: Theory and application to reawrd shaping. In: *Proceedings of the 16th International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann, 1999:278–287.
139. Nishizaki I, Katagiri H, and Toshihisa T. Simulation Analysis Using Multi-Agent Systems for Social Norms. Computational Economics 2009;34(1):37–65.
140. Nitschke G. Designing emergent cooperation: A pursuit-evasion game case study. Artificial Life and Robotics 2005;9(4):222–233.
141. Nitschke G and Didi S. Evolutionary Policy Transfer and Search Methods for Boosting Behavior Quality: RoboCup Keep-Away Case Study. Frontiers in Robotics and AI 2017;4(62):1–25.
142. Nitschke G, Schut M, and Eiben A. Emergent Specialization in Biologically Inspired Collective Behavior Systems. In: *Intelligent Complex Adaptive Systems*. New York, USA: IGI, 2007:100–140.
143. Nitschke G, Eiben A, and Schut M. Evolving Team Behaviors with Specialization. Genetic Programming and Evolvable Machines 2012;13(4):493–536.
144. Noda I, Matsubara H, Hiraki K, and Frank I. Soccer server: A tool for research on multiagent systems. Applied Artificial Intelligence 1998;12:233–250.
145. Nolfi S and Floreano D. Co-evolving predator and prey robots: Do arm races arise in artificial evolution. Artificial Life 1999;4:311–335.
146. Nolfi S and Floreano D. Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press, 2000.
147. Pan S and Yang Q. A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering 2010;22(10):1345–1359.
148. Panait L and Luke S. Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems 2005;3:387–434.
149. Panait L and Luke S. Cooperative multi-agent Learning: The state of the art. Autonomous Agents and Multi-Agent Systems 2005;11:387–434.
150. Peters J and Schaal S. Reinforcement learning of motor skills with policy gradients. Neural Networks 2008;21:682–697.

151. Price B and Boutilier C. Impicit Imitation in Multiagent reinforcement learning. In: *Proceeding of the sixteeenth Conference on machine Learning*. San Francisco, USA: Morgan Kaufmann, 1999:325–334.
152. Price B and Boutilier C. Accelerating reinforcement learning through implicit imitation. Journal of Artificial intelligence Research 2003;19:569–629.
153. Pugh J, Soros L, Szerlip P, Paul A, and Stanley K. Confronting the Challenge of Quality Diversity. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2015:967–974.
154. R. Sutton A. Koop DS. On the role of tracking in stationary environments. In: *Proceedings of the 24th International Conference on Machine Learning*. New York, USA: ACM, 2007:871–878.
155. Radcliffe NJ. Genetic set recombination and its application to neural network topology optimisation. Neural Computing & Applications 1993;1:67–90.
156. Rajagopalan P, Rawal A, Miikkulainen R, Wiseman M, and Holekamp K. The Role of Reward Structure, Coordination Mechanism and Net Return in the Evolution of Cooperation. In: *Proceedings of the IEEE Conference on Computational Intelligence and Games*. 258-265: IEEE Press, 2011.
157. Randlov J and Alstrom P. Learning to drive a bicycle using reinforcement learning and shaping. In: *Proceedings of the 15th International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann, 1998:463–471.
158. Reynolds CW. Flocks, herds, and schools: a distributed behavioral model. Computer Graphics 1987;21:25–34.
159. Risi S and Stanley K. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. Artificial Life 2012;18(4):331–363.
160. Risi S and Stanley K. Confronting the challenge of learning a flexible neural controller for a diversity of morphologies. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2013:255–261.
161. Risi S, Hughes C, and Stanley K. Evolving Plastic Neural Networks with Novelty Search. Adaptive Behavior 2010;18:470–491.
162. Ruvolo P and Eaton E. An efficient lifelong learning algorithm. In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Atlanta, USA: JMLR Workshop and Conference Proceedings, 2013:507–515.
163. Sareni B and Krahenbuhl L. Fitness Sharing and Niching Methods Revisited. IEEE Transactions on Evolutionary Computation 1998;2(3):97–106.
164. Schaffer J, Whitley D, and Eshelman L. Combinations of genetic algorithms and neural networks: a survey of the state of the art. Combinations of Genetic Algorithms and Neural Networks 1992;2:1–37.
165. Shoham Y and Leyton-Brown K. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. New York, USA: Cambridge University Press, 2008.
166. Silva F, Duarte M, Correia L, Oliveira S, and Christensen A. Open Issues in Evolutionary Robotics. Evolutionary Computation 2016;In press.

167. Silver D, Yang Q, and Li L. Lifelong machine learning systems: beyond learning algorithms. Lifelong Machine Learning, Papers from the 2013 AAAI Spring Symposium 2013:25–27.
168. Singh SP and Sutton RS. Reinforcement learning with replacing eligibility traces. Machine Learning 1996;22:123–158.
169. Smith R, Forrest S, and Perelson A. Searching for diverse, cooperative populations with genetic algorithms. Evolutionary Computation 1993;1:127—149.
170. Spector L, Klein J, Perry C, and Feinstein M. Emergence of collective behavior in evolving populations of flying agents. Genetic Programming and Evolvable Machines 2005;6:111–125.
171. Stanley K. Efficient Evolution of Neural Networks Through Complexification. Ph. D. Dissertation. Austin, USA: Department of Computer Sciences, The University of Texas, 2004.
172. Stanley K. Compositional pattern producing networks: A novel abstraction of development. Genetic Programming and Evolvable Machines 2007;8:131–162.
173. Stanley K and Miikkulainen R. Evolving neural networks through augmenting topologies. Evolutionary Computation 2002;10:99—127.
174. Stanley K and Miikkulainen R. Competitive Coevolution Through Evolutionary Complexification. Journal of Artificial Intelligence Research 2004;21:63–100.
175. Stanley K, Bryant B, and Miikkulainen R. Evolving Neural Network Agents in the NERO Video Game. In: *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games*. Piscataway, USA: IEEE Press, 2005:182–189.
176. Stanley K, D'Ambrosio D, and Gauci J. A Hypercube-Based Indirect Encoding for evolving large-scale neural networks. Artificial Life 2009;15:185–212.
177. Stanley KO and Miikkulainen R. Competitive Coevolution Through Evolutionary Complexification. Journal of Artificial Intelligence Research 2004;21(1):63–100.
178. Stanley KO and Miikkulainen R. Evolving a Roving Eye for Go. In: *Genetic and Evolutionary Computation Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004:1226–1238.
179. Stanley K. Generative and Developmental Systems Tutorial. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: ACM, 2016:609–638.
180. Stone P. Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer. Cambridge, MA, USA: MIT Press, 2000.
181. Stone P and Sutton R. Keepaway Soccer: A Machine Learning Testbed. In: *RoboCup-2001: Robot Soccer World Cup V*. Berlin, Germany: Springer-Verlag, 2002:214–223.
182. Stone P and Veloso M. The CMUNITED-97 Simulator Team. In: *RoboCup-97: Robot Soccer World Cup*. Berlin, Germany: Springer Verlag, 1998:389–397.
183. Stone P and Veloso M. Towards Collaborative and Adversarial Learning: A Case Study in Robotic Soccer. Evolution and learning in multi-agent systems 2002;48:83–104.

184. Stone P, Sutton RS, and Singh S. Reinforcement Learning for 3 vs. 2 Keepaway. In: *RoboCup 2000: Robot Soccer World Cup IV*. London, UK, UK: Springer-Verlag, 2001:249–258.
185. Stone P, Sutton R, and Kuhlmann G. Reinforcement learning for RoboCup-soccer keepaway. Adaptive Behavior 2005;13(3):165–188.
186. Stone P, Kuhlmann G, Taylor M, and Liu Y. Keepaway Soccer: From machine learning testbed to benchmark. In: *Proceedings of RoboCup-2005: Robot Soccer World Cup IX*. Springer, 2006:93–105.
187. Suchorzewski M. Evolving Scalable and Modular Adaptive Networks with Developmental Symbolic Encoding. Evolutionary intelligence 2011;4:145–163.
188. Suchorzewski M and Clune J. A novel generative encoding for evolving modular, regular and scalable networks. In: *Proceedings of the Congress on Evolutionary Computation*. New Orleans, USA: IEEE Press, 2011:1523–1530.
189. Suematsu N and Hayashi A. A multiagent reinforcement learning algorithm using extended optimal response. In: *Proceeding of the 1st International Joint Conference in Autonomous Agents and Multiagent Systems*. Bologna, Italy, 2002:370–377.
190. Sutton R. Learning to predict by the methods of temporal difference. Machine Learning 1998;3:9–44.
191. Sutton R and Barto A. An Introduction to Reinforcement Learning. Cambridge, USA: John Wiley and Sons, 1998.
192. Sycara K. Multiagent systems. AI Magazine. 1998;19:79–92.
193. Taylor A, Dusparic I, Galvan-Lopez E, Clarke S, and Cahill V. Transfer Learning in Multi-Agent Systems through Parallel Transfer. In: *Proceedings of the 30th International Conference on Machine Learning*. Omnipress, 2013:28–37.
194. Taylor M and Stone P. Behavior Transfer for Value-Function-Based Reinforcement Learning. In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. New York, USA: ACM Press, 2005:53–59.
195. Taylor M and Stone P. Towards reinforcement learning representation transfer. In: *Proceedings of the 6th International Joint Conference on autonomous agents and multi-agent systems*. New York, USA: ACM, 2007:101–103.
196. Taylor M and Stone P. Transfer Learning for Reinforcement Learning Domains: A survey. Journal of Machine Learning Research 2009;10(1):1633–1685.
197. Taylor M, Stone P, and Liu Y. Value Functions for RL-based Behavior Transfer: A Comparative Study. In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*. Pittsburgh, Pennsylvania: AAAI Press, 2005:880–885.
198. Taylor M, Whiteson S, and Stone P. Comparing Evolutionary and Temporal Difference Methods in a Reinforcement Learning Domain. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2006:1321–1328.
199. Taylor M, Whiteson S, and Stone P. Transfer Learning for Policy Search Methods. In: *ICML 2006: Proceedings of the Twenty-Third International Conference on Machine Learning Transfer Learning Workshop*. Pittsburgh, USA: ACM, 2006:1–4.

200. Taylor M, Whiteson S, and Stone P. Temporal Difference and Policy Search Methods for Reinforcement Learning: An Empirical Comparison. In: *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*. AAAI Press, 2007:1675–1678.
201. Taylor M, Stone P, and Liu Y. Transfer learning via inter-task mappings for temporal difference learning. Journal of Machine Learning 2007;8(1):2125–2167.
202. Thrun S. Lifelong Learning Algorithms. Boston, MA: Springer, 1998.
203. Torrey L and Shavlik J. Transfer Learning. In: *Handbook of Research on Machine Learning Applications*. Hershey, USA: IGI Global, 2009:17–23.
204. Torrey L, Walker T, Shavlik JW, and Maclin R. Using Advice to Transfer Knowledge Acquired in One Reinforcement Learning Task to Another. In: *In Proceeding of the 16th European Conference on Machine Learning*. Springer, 2005:412–424.
205. Torrey L, Shavlik J, Walker T, and Maclin R. Skill acquisition via transfer learning and advise taking. In: *Proceeding of the 17th European Conference on Machine Learning*. Berlin, Germany: Springer-Verlag, 2006:425–436.
206. Torrey L, Maclin R, Shavlik J, Walker T, and Wild E. Transfer Learning via Advice Taking. In: *Advances in Machine Learning*. Berlin, Germany: Springer, 2010:147–170.
207. Trujillo L, Olague G, Lutton E, and de Vega F. Discovering Several Robot Behaviors through Speciation. In: *Applications of Evolutionary Computing*. Berlin, Germany: Springer, 2008:164–174.
208. Velez R and Clune J. Novelty Search Creates Robots with General Skills for Exploration. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Vancouver, Canada: ACM, 2014:737–744.
209. Verbancsics P. Effective Task Transfer through Indirect Encoding. PhD Thesis. Orlando, USA: Department of Electrical Engineering and Computer Science, University of Central Florida, 2011.
210. Verbancsics P and Stanley K. Evolving static representations for task transfer. Journal of Machine Learning Research 2010;11:1737–1763.
211. Verbancsics P and Stanley K. Constraining connectivity to encourage modularity in HyperNEAT. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2011:1483–1490.
212. Vlassis N. A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. Morgan and Claypool Publishers, 2007.
213. Ward C, Gobet F, and Kendall G. Evolving Collective Behavior in an Artificial Ecology. Artificial Life: Special issue on Evolution of Sensors in Nature, Hardware and Simulation 2001;7:191–209.
214. Watkins C. Learning from delayed rewards. England: PhD Thesis, University of Cambridge, 1989.
215. Weiss G. Multiagent Systems. Cambridge, USA: MIT Press, 1999.
216. Whitehead S and Ballard D. Learning to percieve and act by trial and error. Machine Learning 1991;7(1):45–83.

217. Whiteson S and Stone P. Evolutionary function approximation for reinforcement learning. Journal of Machine Learning Research 2006;7(1):877–917.
218. Whiteson S, Kohl N, Miikkulainen R, and Stone P. Evolving Keep-away Soccer Players through Task Decomposition. In: *Proceeding of the Genetic and Evolutionary Computation Conference*. Chicago: AAAI Press, 2003:356–368.
219. Whiteson S, Taylor M, and Stone P. Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. Autonomous Agent Multi-Agent Systems 2010;21(1):1–25.
220. Whitley D, Starkweather T, and Bogart C. Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Computing 1990;14:347–361.
221. Whitley D, Mathias K, and Fitzhorn P. Delta-Coding: An iterative search strategy for genetic algorithms. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. Montreal, Canada: Morgan Kaufmann, 1991:77–84.
222. Wineberg M and Oppacher F. The Underlying Similarity of Diversity Measures Used in Evolutionary Computation. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Springer, 2003:1493–1504.
223. Woltz D, Gardner M, and Bell B. Negative transfer errors in sequential skills: Strong-but-wrong sequence application. Journal of Experimental Psychology: Learning, Memory and Cognition 2000;26(3):601–635.
224. Yang G and Francesca T. Argumentation Accelerated Reinforcement Learning for RoboCup Keepaway-Takeaway. In: *Theory and Applications of Formal Argumentation: Second International Workshop, TAFA 2013, Beijing, China, August 3-5, 2013, Revised Selected papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014:79–94.
225. Yao X. Evolutionary Artificial Neural Networks. Journal of Neural Systems 1999;4:203–222.
226. Yong C and Miikkulainen R. Cooperation Coevolution of Multi-Agent Systems. Technical Report AI01-287. Austin, USA: Department of Computer Sciences, The University of Texas, 2007.
227. Yong C and Miikkulainen R. Co-evolution of role-based cooperation in Multi-Agent systems. IEEE Transactions on Autonomous Mental Development 2010;1:170–186.
228. Yong C, Stanley KO, Miikkulainen R, and Karpov I. Incorporating advice into neuroevolution of adaptive agents. In: *Proceedings of the Second AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. ACM, 2006:98–104.
229. Zaera N, Cliff D, and Bruten J. (Not) Evolving Collective Behaviors in Synthetic Fish (Tech. Rep.) Bristol, England: Hewlett-Packard Laboratories, 1996.
230. Zhao J and Peng G. NEAT versus PSO for Evolving Autonomous Multi-agents Coordination on Pursuit-Evasion Problem. In: *Informatics in Control, Automation and Robotics: Volume 2*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012:711–717.