In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015), pp. 345--352, Madrid, Spain, July 2015. Best Paper: Digital Entertainment and Arts.

Solving Interleaved and Blended Sequential Decision-Making Problems through Modular Neuroevolution

Jacob Schrum
Dept. of Mathematics and Computer Science
Southwestern University
Georgetown, TX 78626 USA
schrum2@southwestern.edu

Risto Miikkulainen
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
risto@cs.utexas.edu

ABSTRACT

Many challenging sequential decision-making problems require agents to master multiple tasks, such as defense and offense in many games. Learning algorithms thus benefit from having separate policies for these tasks, and from knowing when each one is appropriate. How well the methods work depends on the nature of the tasks: Interleaved tasks are disjoint and have different semantics, whereas blended tasks have regions where semantics from different tasks overlap. While many methods work well in interleaved tasks, blended tasks are difficult for methods with strict, human-specified task divisions, such as Multitask Learning. In such problems, task divisions should be discovered automatically. To demonstrate the power of this approach, the MM-NEAT neuroevolution framework is applied in this paper to two variants of the challenging video game of Ms. Pac-Man. In the simplified interleaved version of the game, the results demonstrate when and why such machine-discovered task divisions are useful. In the standard blended version of the game, a surprising, highly effective machine-discovered task division surpasses human-specified divisions, achieving the best scores to date in this game. Modular neuroevolution is thus a promising technique for discovering multimodal behavior for challenging real-world tasks.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*; I.2.6 [**Artificial Intelligence**]: Learning—*Connectionism and neural nets*

Keywords

Games, Neural networks, Multi-objective optimization

GECCO '15, July 11 - 16, 2015, Madrid, Spain

DOI: http://dx.doi.org/10.1145/2739480.2754744

1. INTRODUCTION

Discovering intelligent agent behavior automatically for complex environments is an important goal for Artificial Intelligence. Such behavior is needed in both real-world robots and virtual environments. However, most interesting domains consist of multiple tasks, each requiring different behavior. Such behavior is multimodal, because distinct behavior is required in each task. Learning such behavior is difficult, and even harder when individual tasks overlap.

This paper presents a new way of identifying tasks for complex domains, and applies several methods of constructing modular policies for such domains using evolved neural networks (i.e. neuroevolution). Tasks are defined by partitioning the state space such that the frequency of transitions between tasks is low. This approach is then complimented by using domain semantics to define predicates (based on high-level domain features) that identify each task.

When distinctions between tasks are clear, the tasks are interleaved. A learning agent can then split up a domain by dedicating separate policies to each task. When distinctions are not clear, the tasks are blended. In such domains, it is sensible to identify the prominent tasks, and treat the domain as a blend of these representative tasks. However, to succeed in such domains, agents need the freedom to discover their own task divisions.

This paper builds on recent research in Ms. Pac-Man using evolved neural networks [24]. To support multiple tasks, networks have separate output modules to represent different behaviors. Previous research required evolution to discover how to use these modules, but this paper also allows a human designer to specify how modules are used in each task, in a style similar to Multitask Learning [7]. This extension makes it possible to identify the costs and benefits of discovering task divisions automatically. Another new contribution is that these methods are compared not only in the original Ms. Pac-Man, where the tasks are blended, but also in an interleaved version of the game. This extension makes it possible to determine where each approach is most effective.

The conclusion is that while a variety of approaches work well in the interleaved version, agents in the original blended version need the freedom to discover their own task division in order to succeed. When given such freedom, the most effective task divisions discovered are sometimes surprising: A luring strategy is learned, which uses one module to gather all ghosts close together, so that another module can quickly escape to a power pill, allowing the first module to easily capture the ghosts. Such a task division is not obvious to a human designer a priori, demonstrating the power of the modular neuroevolution approach.

2. RELATED WORK

Domains requiring multimodal behavior are common in both robotics and simulated environments, and various approaches have been implemented to deal with such domains.

For complex tasks, it is common to combine controllers into a hierarchy. Such hierarchies can be hand-designed [5] or learned. For example, Togelius's [31] evolved subsumption architecture was used in EvoTanks [30] and Unreal Tournament [32], and Stone's [27] Layered Learning was applied to RoboCup Soccer. Recently, Lessin et al. [18] used a human-designed hierarchical syllabus to evolve complex behavior for virtual creatures. Though effective, these approaches require a programmer to divide the domain into constituent tasks and develop effective training scenarios for each.

Hierarchical Reinforcement Learning (HRL) also produces hierarchical controllers consisting of multiple sub-controllers. Early HRL research required the hierarchy to be human-specified [10], but methods now exist for learning it as well [11]. Most HRL techniques are based on the formalism of Semi-Markov Decision Processes (SMDPs), which was first used to develop partial control policies called options [29]. Similar techniques, e.g. skills [16] and activities [3], also fit this formalism. The methods in this paper can also be cast in the SMDP formalism, but they do not depend on it. Therefore, a simpler formalization is presented in the next section.

Several evolutionary approaches to learning multimodal behavior simply focus on discovering modular policies. One approach is to associate components of the architecture automatically with specific functionality. For instance, Calabretta et al. [6] evolved neural networks to control robots using a duplication operator, which copies one output neuron with all of its connections and weights. The network then has two outputs for the same actuator, and needs to arbitrate between them. Such arbitration is performed by selector units: For each actuator, the output neuron with the highest corresponding selector unit activation controls the actuator for that time step.

A similar approach is Module Mutation [23, 24], which introduces groups of neurons rather than individual neurons. A single Module Mutation adds enough output neurons to define a new policy, plus an additional neuron to arbitrate between modules. The behavior-defining neurons are called policy neurons, and the one arbitration neuron per module is called a preference neuron. Preference neurons are similar to the selector units of Calabretta et al. [6]. Since Module Mutation is used in this paper, it is discussed further in Section 4.3 on learning methods.

Other researchers have developed modular networks with a more general concept of a module [8, 13], i.e. a cluster of interconnected neurons with few connections to neurons in other clusters. Such modular networks can also be created using generative and developmental methods [19, 33, 12]. These methods evolve modular neural networks assuming that having different modules handle different parts of the task makes optimization easier. Compared to these approaches, the modular networks of this paper create clear task divisions in which it is always known which module is responsible for an agent's actions.

Modular policies have also been explored in Genetic Programming (GP). An early example is Koza's Automatically Defined Functions [17], which encapsulate portions of a program tree that can potentially be reused. A similar GP technique is Adaptive Representation through Learning [22], which culls modules from program trees based on differential parent/child fitness.

All of these techniques face the same challenge of having to break a domain into separate tasks. The next section presents a new way of identifying tasks within a domain.

3. TASK DIVISIONS

This section first describes the Markov Decision Process (MDP) formalism that is the basis of Reinforcement Learning (RL) problems [28]. This formalism is then used to specify a way of identifying individual tasks. The nature of the task division is used to distinguish between domains where tasks are interleaved vs. blended.

3.1 Markov Decision Process

An MDP is a formal description of a domain in terms of the results of taking certain actions in certain states. Specifically, an MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition function, and $\mathcal{R}$ is the reward function. The transition function is defined as $\mathcal{P} : (\mathcal{S} \times \mathcal{A} \times \mathcal{S}) \rightarrow [0, 1]$ where $\mathcal{P}(s, a, s')$ returns the probability of reaching state s' immediately after performing action a in state s. The reward function provides the agent with feedback on how well it is performing. Typical RL formulations use scalar rewards, but $\mathcal{R}$ can be extended to multiple objectives: $\mathcal{R} : (\mathcal{S} \times \mathcal{A}) \rightarrow \mathbb{R}^N$ is defined such that $\mathcal{R}(s, a)$ returns a tuple of the expected immediate rewards in each objective for performing action a in state s. This tuple has length N, the number of objectives.

The commonly used discount factor γ is excluded from this definition because only episodic domains are considered in this paper. As a result, the goal of an agent is to maximize the sum of rewards in each objective throughout the course of an episode. An episode is an evaluation period with definite start and end points, which allows rewards to be safely weighted equally without emphasizing early rewards in favor of later rewards [28]. For use in an evolutionary algorithm, the sum of rewards in each objective is treated as a different fitness function subject to multiobjective optimization.

The next sections use this definition of MDPs to explain the concepts of interleaved and blended tasks.

3.2 Interleaved Tasks

A collection of T interleaved tasks for an MDP is defined as a partition of its state space $\{\mathcal{S}_1, \ldots, \mathcal{S}_T\}$. By the definition of a partition, $\mathcal{S}$ is the union of these sets, which are pairwise disjoint:

$$\mathcal{S} = \bigcup_{i=1}^{T} \mathcal{S}_i \text{ and} \tag{1}$$

$$\forall i, j \in \{1, \ldots, T\} (\mathcal{S}_i = \mathcal{S}_j \vee \mathcal{S}_i \cap \mathcal{S}_j = \varnothing). \tag{2}$$

Importantly, each state is in exactly one task because all tasks cover $\mathcal{S}$, but are disjoint from each other. However, not all partitions are meaningful. For the concept of interleaved tasks to be useful, an agent should mostly remain in each task for an extended period before switching to another, so that its actions can have a useful impact in each task. In other words, the proportion of task switches to total time spent in the domain should be small. A low rate of thrashing back and forth between tasks indicates that a domain consists of a few important tasks that partition the state space. Of course, frequency of task switches also depends on a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (assuming the policy is deterministic), but because state transitions are stochastic, even the policy does not completely dictate which states are visited. The sequence of states visited within an episode determines the number of task transitions it contains. Therefore, an episode e consisting of m states can be defined as the sequence of visited states $s_1, s_2, \ldots, s_m$. Given this definition of an episode, the thrashing rate τ_e of episode e is defined as

$$\tau_e = x_e / m, \tag{3}$$

where x_e is the number of task transitions in e. The number of task transitions can be formally defined as the cardinality, or size, of the

set of states in the episode where the next state is in a different task:

$$x_e = |\{s_i | s_i \in \mathcal{S}_j \wedge s_{i+1} \in \mathcal{S}_k \wedge j \neq k\}|, \quad (4)$$

where $i \in \{1, \ldots, m-1\}$, and therefore refers to any state before the last state, and $j, k \in \{1, \ldots, T\}$, so that each refers to one of the T tasks in the MDP. In other words, each time adjacent states of the episode are in different tasks, one task transition is tallied. From τ_e, the *domain's* thrashing rate $\bar{\tau}$ is defined as the average τ_e across all possible e. For a domain to have distinct interleaved tasks, $\bar{\tau}$ must be low.

Calculating $\bar{\tau}$ may not be possible, but a rough estimate is enough to determine whether a partition of the state space creates interleaved tasks. The lower $\bar{\tau}$ is, the more likely a domain can be usefully labelled as having interleaved tasks. The interleaved domain used in this paper has a thrashing rate less than 0.01, as will be explained in Section 5.1.

Although the thrashing rate provides a way of identifying the states in each task, the intuitive notion of a task is that each one has some distinct semantic properties that are clear to someone who understands the domain. In a domain with interleaved tasks, some obvious set of semantic properties should correspond to the structurally defined tasks that result in a low $\bar{\tau}$. Each task should have a predicate that is true for all states in that task, and false for all other states. Formally, $\psi_1, \ldots, \psi_T : \mathcal{S} \rightarrow \{\top, \bot\}$ are semantic predicates for the T tasks in the MDP for which each state satisfies one and only one semantic predicate:

$$\forall i \in \{1, \ldots, T\} \forall s \in \mathcal{S}_i (\psi_i(s) = \top) \text{ and} \quad (5)$$

$$\forall i, j \in \{1, \ldots, T\} \forall s \in \mathcal{S}_i (i \neq j \rightarrow \psi_j(s) = \bot). \quad (6)$$

These predicates allow any task i to be defined as the subset of $\mathcal{S}$ that satisfies ψ_i: For any predicate on states $\psi : \mathcal{S} \rightarrow \{\top, \bot\}$, define that predicate's task as the set of states that satisfy it:

$$\Phi(\psi) = \{s \in \mathcal{S} | \psi(s) = \top\}. \quad (7)$$

For the set of predicates above that satisfy properties 5 and 6, $\Phi(\psi_i) = \mathcal{S}_i$ for all $i \in \{1, \ldots, T\}$. Generally, an initial guess at suitable predicates should lead to the desired low thrashing rate.

However, predicates with properties 5 and 6 are not easy to define in domains with blended tasks, described next.

3.3 Blended Tasks

Interleaved tasks are defined in terms of the structure of the state space, which in turn depends on the transition function. However, it is also desirable for states in the same task to share high-level properties linked to semantic predicates. Unfortunately, it is sometimes hard to partition the state space of a domain with respect to such properties. In fact, sets of states in which different semantic properties hold may not be disjoint. The region where such properties overlap blends the properties of different tasks.

Assume that m distinct semantic predicates $\psi_1, \ldots, \psi_m$ are identified in some domain, but that

$$\exists i, j \in \{1, \ldots, m\} (i \neq j \wedge \Phi(\psi_i) \cap \Phi(\psi_j) \neq \varnothing). \quad (8)$$

That is, the set of m predicates does not satisfy properties 5 and 6 because the resulting tasks are not disjoint. It is possible to choose new predicates specifically designed to address the overlap, by replacing every two predicates ψ_x, ψ_y whose tasks overlap with three new predicates $\phi_x, \phi_y, \phi_{xy}$:

$$\phi_x(s) = \psi_x(s) \wedge \neg\psi_y(s), \quad (9)$$

$$\phi_y(s) = \psi_y(s) \wedge \neg\psi_x(s), \quad (10)$$

$$\phi_{xy}(s) = \psi_x(s) \wedge \psi_y(s). \quad (11)$$

These new predicates split up states into those that only satisfy ψ_x, those that only satisfy ψ_y, and those that satisfy both.

If every state satisfies at least one of the initial m predicates, iterating this process creates a set of predicates that generate disjoint tasks. If the thrashing rate between the resulting tasks is sufficiently low, then they are interleaved.

However, such a proliferation of predicates ignores the fact that some states have properties from multiple tasks. An alternative is to treat regions of intersection as blended regions between tasks. This view allows a learning agent to focus on major distinctions between tasks rather than micromanage different behaviors for many small but similar tasks. At least two semantic predicates ψ_p, ψ_q that satisfy the following properties are required:

$$\exists s \in \mathcal{S} (\psi_p(s) \wedge \neg\psi_q(s)), \quad (12)$$

$$\exists s \in \mathcal{S} (\psi_q(s) \wedge \neg\psi_p(s)), \quad (13)$$

$$\exists s \in \mathcal{S} (\psi_q(s) \wedge \psi_p(s)). \quad (14)$$

In other words, the tasks defined by ψ_p and ψ_q are not disjoint, but neither is a subset of the other. As a result, the states not in the intersection correspond to specific tasks, and the states in the intersection are in the blended region between those two tasks.

In domains with blended tasks, it becomes important for an agent to decide what role to take on at any given time. There may be clear goals in regions of a specific task, but in the blended region there are often multiple competing goals. If an agent can discover its own task division, it can more easily learn which goals to pursue in such regions. However, learning methods that easily allow for different modes of behavior in different tasks are needed in both blended and interleaved domains. Such methods are described next.

4. LEARNING METHODS

Evolutionary multiobjective optimization is used to evolve controllers for MDPs with multiple tasks. The evolved individuals are neural networks, and modular architectures are used to encourage multimodal behavior.

4.1 Evolutionary Multiobjective Optimization

Because different tasks can have different goals, domains requiring multimodal behavior may have multiple objectives. Therefore, a principled way of dealing with multiple objectives is needed. Such a framework is provided by the concepts of Pareto dominance and optimality:

Pareto Dominance: Vector $\vec{v} = (v_1, \ldots, v_N)$ dominates $\vec{u} = (u_1, \ldots, u_N)$ iff

$$\forall i \in \{1, \ldots, N\} (v_i \geq u_i) \text{ and} \quad (15)$$

$$\exists i \in \{1, \ldots, N\} (v_i > u_i). \quad (16)$$

Pareto Optimality: A set of points $\mathcal{A} \subseteq \mathcal{F}$ is Pareto optimal iff it contains all points such that $\forall \vec{x} \in \mathcal{A} \neg\exists \vec{y} \in \mathcal{F}$ such that $\vec{y}$ dominates $\vec{x}$. The points in $\mathcal{A}$ are non-dominated, and make up the non-dominated Pareto front of $\mathcal{F}$.

These definitions indicate that one solution dominates another if it is strictly better in at least one objective and no worse in the others. The best solutions are not dominated by any other solutions, and make up the Pareto front of the search space. The next best individuals are those that would be in a recalculated Pareto front if the actual Pareto front were removed first. Layers of Pareto fronts can be defined by successively removing the front and recalculating

it for the remaining individuals. Solving a multiobjective optimization problem involves approximating the first Pareto front as best as possible. This paper accomplishes this goal using Non-dominated Sorting Genetic Algorithm II (NSGA-II [9]).

NSGA-II uses ($\mu + \lambda$) elitist selection favoring individuals in higher Pareto fronts of the current population over those in lower fronts. Within a given front, individuals that are more distant from others in objective space are favored by selection so that the algorithm explores diverse trade-offs.

Applying NSGA-II to a problem produces a population containing an approximation to the Pareto front. This approximation set potentially contains multiple solutions. Usually, this set must be analyzed to determine which solutions fulfill the needs of the user, but for the domains in this paper (whose objectives are described in Section 5.2), a specific objective weighting (based on Ms. Pac-Man game score) is available which reduces two objectives to a single score. Evolution on this single objective could also be used, but optimizing with multiple objectives instead of one is appealing because it can improve search by helping avoid local optima [14].

NSGA-II is indifferent as to how solutions are represented. This paper uses NSGA-II to evolve artificial neural networks.

4.2 Neuroevolution

Neuroevolution is the simulated evolution of artificial neural networks. All behavior in this paper is learned via a version of NEAT (Neuro-Evolution of Augmenting Topologies [26]), a constructive neuroevolution method that has been successful in many RL domains [15, 25].

NEAT networks start with no hidden neurons, and are modified by three mutation operators during evolution. Weight mutation perturbs the weights of existing network connections, link mutation adds new connections between existing nodes, and node mutation splices new nodes along existing connections. NEAT also features an efficient method of topological crossover between networks.

The variant of NEAT used in this paper is Modular Multiobjective NEAT[1] (MM-NEAT [24]), which distinguishes itself from NEAT by incorporating NSGA-II, and providing several methods for creating networks with multiple output modules, described next.

4.3 Modular Networks

MM-NEAT networks allow for multiple output modules. Each such module defines a different control policy. A new feature of MM-NEAT in this paper is the ability to arbitrate between modules based on a human-specified division, i.e. via Multitask networks. As in previous research, arbitration can also be based on preference neurons. In preference neuron networks, evolution must discover how to use the modules. Using Module Mutation, evolution must also settle on an appropriate number of modules. Each of these approaches is evaluated in this paper.

Multitask networks were first proposed by Caruana [7] for supervised learning using neural networks and backpropagation. One network has multiple modules, and each module corresponds to a different task (Fig. 1b). Each module is trained on data for its corresponding task, but because hidden-layer neurons are shared by all outputs, knowledge common to all tasks can be stored in the hidden layer. This approach speeds up supervised learning of multiple tasks because knowledge shared across tasks is only learned once, rather than learned independently multiple times.

Multitask Learning is a powerful technique, but the individual tasks need to be identified a priori (the specific divisions used in Ms. Pac-Man are discussed in Section 5.2). Appropriate task divisions are not always obvious, and obvious divisions may actually hurt learning. Therefore, Multitask Learning will only be useful if its task division is appropriate. When tasks are blended, it is hard to provide an appropriate division. To discover better task divisions, a means of learning how to arbitrate between tasks is needed.

Preference neurons make module arbitration without a human-specified task division possible. Each module has policy neurons for defining behavior and a preference neuron that outputs the network's preference for using that module's policy neurons (Fig. 1c). Whenever inputs are presented to the network, the module whose preference neuron output is highest specifies the network's output.

Preference neurons partition the state space into one set per output module, though there is no requirement that these sets are interleaved tasks in terms of the thrashing rate $\bar{\tau}$ between them. However, preference neuron networks have the capacity to evolve such a task division if it is useful.

This architecture assumes that a designer specifies the number of modules. However, evolution can also discover how many modules are needed by adding modules gradually, using Module Mutation.

Module Mutation is any structural mutation operator that adds a new output module to a neural network. An indefinite number of modules may be added in this way. Such networks depend on preference neurons for module arbitration. New populations start with a single module and a preference neuron that only becomes relevant after more modules are added (Fig. 1d). Each Module Mutation adds a new set of policy neurons and a new preference neuron.

Different versions of Module Mutation exist [23, 24]. This paper uses MM(D), for *duplicate*, because each new module duplicates the behavior of an existing module. For every link into a policy neuron in the original module, a duplicate link into the corresponding policy neuron of the new module is created, and it has the same source neuron and link weight as the link being copied (Fig. 1e). After MM(D), a network will behave the same as before, but evolution can then modify the new structure, so that module behavior diverges. However, links to the new module's preference neuron are not copied from the parent module. Rather, the new preference neuron has a single link with a random source and weight, to encourage the module to be used in different circumstances.

MM(D) and the other modular approaches described so far are evaluated in domains with interleaved tasks and blended tasks, as described next.

5. EXPERIMENTS

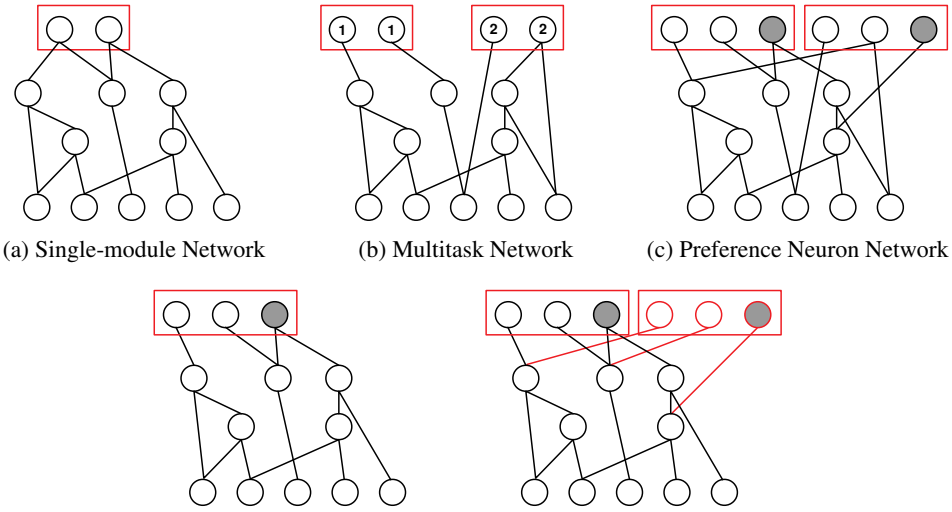
The domains used for evaluation are two variants of the classic video game Ms. Pac-Man. This game requires multimodal behavior because enemy agents are sometimes threats, and sometimes sources of points. Most of the modular architectures described above were applied to the original, blended version of Ms. Pac-Man [24], in which a mixture of edible and threat ghosts can be present at the same time. However, the interleaved version of the domain, in which such mixtures never occur, is new, and seeing these results side-by-side gives new insight into when and why certain types of modular networks are successful. This section describes both blended and interleaved variants of Ms. Pac-Man, then describes experiments in both domains.

5.1 Ms. Pac-Man

Ms. Pac-Man (1981) is among the most popular video games of all time. Its gameplay is simple, yet requires complex strategies for success. A popular platform for Ms. Pac-Man research is the simulator for the Ms. Pac-Man vs. Ghosts competitions[2] [21],

[1]Download at http://nn.cs.utexas.edu/?mm-neat. Download includes source code for all experiments presented in this paper.

[2]http://www.pacman-vs-ghosts.net/



(a) Single-module Network (b) Multitask Network (c) Preference Neuron Network

(d) Before Module Mutation (e) Module Mutation Duplicate

Figure 1: Modular Networks: These example networks are designed for a domain where two policy neurons define the behavior of an agent. Inputs are at the bottom, and each output module is contained in its own red box. (a) Standard neural network with just one module. (b) Multitask network with two modules, each consisting of two policy neurons. A human-specified task division indicates when to use Policy 1 vs. Policy 2. (c) A fixed network with two modules that uses preference neurons (colored gray) to determine which module to use. (d) A starting network in a population where Module Mutation is enabled. It has one module, and an irrelevant preference neuron. (e) After MM(D), the network gains a new module with policy neurons linked to the same neuron sources with the same link weights as policy neurons in the module that was duplicated. However, the new preference neuron is linked to a random source with a random weight so that the new module is used in different situations. After MM(D), both the pre-existing and newly added preference neurons become relevant. Extra modules allow these networks to learn multimodal behavior more easily by associating a different module with each behavioral mode.

which includes a standard `Legacy` ghost team that approximates the original ghosts.

In Ms. Pac-Man, each of four mazes contains several pills and four power pills. Ms. Pac-Man moves around, eating pills she comes in contact with, all of which must be eaten to clear a level. Each pill earns 10 points, and each power pill 50 points.

Each maze starts with four hostile ghosts in a lair near the center of the maze. They come out one by one and pursue Ms. Pac-Man according to different algorithms. If a ghost touches Ms. Pac-Man, she dies and the episode ends. However, if Ms. Pac-Man eats a power pill, then for a limited time she can eat the ghosts. The 1^{st}, 2^{nd}, 3^{rd}, and 4^{th} ghosts eaten in sequence are worth 200, 400, 800, and 1600 points, respectively. The maximum score is achieved by eating all four ghosts after each power pill. This goal becomes more challenging in each subsequent level, because edible time decreases as the level increases, but it is still achievable because edible ghosts move at half speed.

If ghosts were always all edible or all threatening, this domain would provide a clear example of interleaved tasks: one task predicate satisfied when any ghost was edible, and the other when any ghost was a threat. An upper bound on $\bar{\tau}$ for this interleaved domain can be calculated by realizing that each power pill can cause two task transitions: ghosts become edible, then go back to being threats. Some amount of time has to pass before Ms. Pac-Man can reach a power pill, and once she eats one she is guaranteed to survive for the entire edible time, which is 200 in the first maze. Therefore, for every two task transitions, over 200 time steps must pass. Thus, an upper bound for the thrashing rate is $2/200 = 0.01$. This bound will hold despite the fact that the edible time decreases to 145 by the fourth level. It takes over 1,000 time steps simply to eat all pills in one of the mazes, and dodging ghosts causes further delays. Therefore, the extra time steps required to beat preceding levels more than cancels out the decrease in edible time.

However, sometimes ghosts of both types are in the maze at the same time. After a ghost is eaten it returns to the lair for a short time before reemerging as a threat, which can happen before the edible time has expired for the other ghosts. Therefore, there are situations with both threat and edible ghosts, each at different positions relative to Ms. Pac-Man, which put the competing goals of survival and ghost eating at odds with each other. This set of states represents the blended region between the threat and edible tasks. Thus, Ms. Pac-Man is a game with blended tasks.

However, the game can be modified to have strictly interleaved tasks by confining eaten ghosts to the lair until either all ghosts have been eaten, or the edible time has expired. This change creates a new domain that shows the relative benefits of different types of modular policies in domains with interleaved vs. blended tasks. Although the two domains are slightly different, each requires multimodal behavior to succeed. The next section explains how experiments in these domains are conducted.

5.2 Experimental Setup

Neural networks are evolved in the manner described by Schrum and Miikkulainen [24], details of which are provided below. Networks are evaluated with sensor information corresponding to each available movement direction on each time step in order to produce a single output value for each direction. Ms. Pac-Man moves in the direction with the highest network output.

There are both direction-oriented sensors and sensors that are not direction-oriented, i.e. ones providing the same reading for each direction. Direction-oriented distances measure the shortest path to objects of interest starting in a particular direction and continuing without reversing. Distances have a maximum of 200, and all sensors are scaled to $[0, 1]$.

The direction-oriented sensors are: distances to the nearest pill, power pill, and junction, the maximum numbers of pills and junc-

tions within 30 steps, distances to the 1st, 2nd, 3rd, and 4th closest ghosts, whether each ghost is approaching, whether a directional path to each ghost contains junctions, and the Options From Next Junction (OFNJ). OFNJ looks at the next junction in a given direction, and counts the number of subsequent junctions that can be safely reached from the first junction without reversing. The safety of a route can be determined by taking all agent distances into account and conservatively assuming ghosts will follow the shortest path to the target junction. Additionally, the blended game has a sensor for whether each ghost is edible, which is not needed in the interleaved version because a single sensor provides this information about all ghosts.

The remaining sensors are not direction-oriented: a constant bias, the proportions of remaining pills, power pills, edible ghosts, and edible time, and Boolean sensors indicating whether any ghost is edible, whether there are four threats outside the lair, and whether Ms. Pac-Man is within 10 steps of a power pill.

These sensors are used to evolve Ms. Pac-Man controllers with separate pill and ghost objectives. The pill score is the number of pills eaten, including power pills. The ghost score gives points proportional to the score that would be received for each ghost eaten: the 1st, 2nd, 3rd, and 4th ghosts are worth 1, 2, 4, and 8 points, respectively. Evaluation in Ms. Pac-Man is noisy because the ghost movement is non-deterministic, so each neural network is evaluated 10 times; fitness scores are the average scores across evaluations. Although these objectives are used during evolution, the final results are evaluated in terms of game score.

Because 10 episodes per network are costly, an episode will end after the fourth maze is cleared, and an 8,000 time step per level limit is imposed, after which point Ms. Pac-Man dies.

In both the interleaved and blended versions of the game, populations of networks with one module (`1M`), two modules (`2M`), and three modules (`3M`) are evolved. Modular networks use preference neurons to decide which module to use on each time step. Populations that start with one module, but can add more via `MM(D)`, are also evaluated. These approaches were previously evaluated in the blended Ms. Pac-Man game, but not in the interleaved version.

Multitask networks require human-specified task divisions. In the interleaved game, the Multitask (`MT`) approach has one module each for the edible and threat tasks. In the blended game, two Multitask approaches are used. The two module (`MT2`) approach uses one module when any ghost is edible, and another module otherwise, and the three module (`MT3`) approach uses separate modules when all ghosts are either threats or edible, and the third module when there is a mix of both types. `MT3` treats the blended region between tasks as its own task.

Populations of each type are evolved 30 times for 200 generations each, with a population size of 100. The mutation rates are: 5% chance of weight perturbation per link, 40% chance of a new link, and 20% chance of a new node. In MM(D) runs, Module Mutation is applied to 10% of offspring. The crossover rate is 50%. These settings lead to the results described in the next section.

5.3 Results

In general, modular networks perform well in both the interleaved and blended domains, while `1M` performs the worst. Multitask approaches handle interleaved tasks well, but perform poorly when tasks are blended.

Figure 2a shows how in the interleaved domain, `2M`, `3M`, `MM(D)` and `MT` all quickly achieve higher scores than `1M`. In the final generation, each of these modular approaches has much higher scores than `1M`. Figure 2b shows how in the blended domain, methods using preference neurons (`MM(D)`, `3M` and especially `2M`) reach scores much higher than those achieved by the Multitask methods (`MT2` and `MT3`) and `1M`. The modular approaches that use preference neurons are all better than `1M` and the Multitask approaches. The high performance of preference neuron approaches is consistent with previous work, but the poor performance of Multitask networks is an interesting new result.

These results indicate that while human-specified and machine-learned task divisions can both handle interleaved tasks well, human-specified task divisions have difficulty with blended tasks. This result holds when the blended region is treated as its own task, as with `MT3`.

These results are statistically verified by post-learning evaluations of the champions from each run. Each was evaluated 1,000 times in the game, and the resulting average scores were compared. In each domain, Kruskal-Wallis tests verify that the learning algorithms perform differently, and pairwise Mann-Whitney U post tests with Bonferroni error correction identify significant differences between specific pairs. Specifically, there are differences between methods in the interleaved domain ($H = 37.02, df = 4, N = 30, p \approx 1.8 \times 10^{-7}$), and post tests reveal that all modular approaches are significantly better than `1M` ($p < 0.05$), but not different from each other. There were also significant differences between methods in the blended domain ($H = 49.69, df = 5, N = 30, p \approx 1.6 \times 10^{-9}$), but these post tests indicate that while `2M`, `3M`, and `MM(D)` are significantly better than `1M` and `MT2`, only `2M` and `3M` are significantly better than `MT3` ($p < 0.05$).

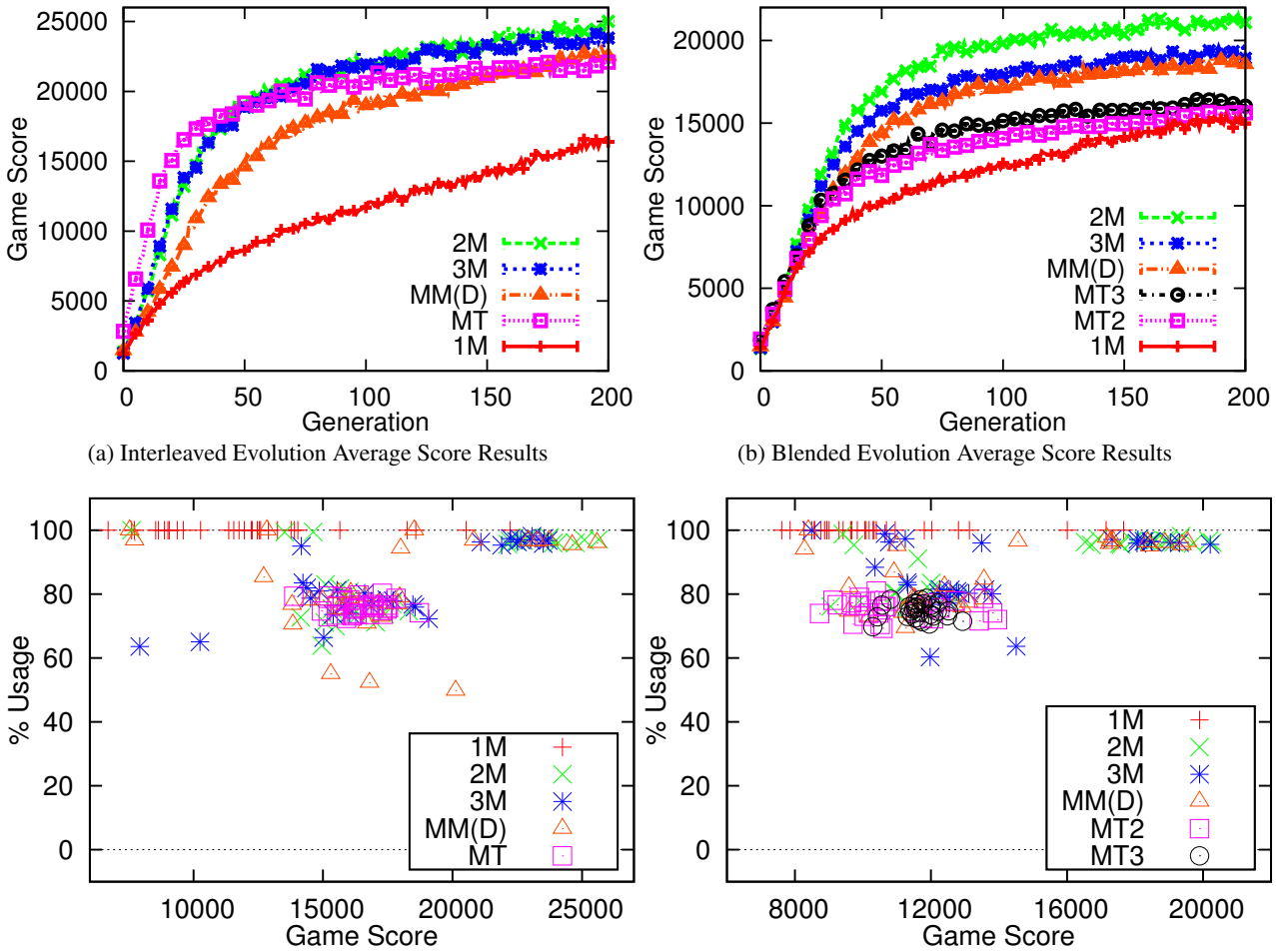
These post-learning evaluations also reveal the types of task divisions learned by each champion. Average scores are plotted against the usage of the most used module in Figures 2c and 2d. Color-coded movies[3] of the module usage are observed to determine the role of each module. Similar divisions emerge in both versions of the game. However, certain divisions do not work as well in blended tasks as they do in interleaved tasks.

A common strategy in both domains is to dedicate one module to dealing with edible ghosts (roughly 20% of the time) and another to threat ghosts (roughly 80%). This strategy is mandated by `MT` in the interleaved domain, and approximated in different ways by `MT2` and `MT3` in the blended domain (the 20% that `MT3` dedicates to edible ghosts is split across two modules). The `2M`, `3M`, and `MM(D)` networks also learn a pure threat/edible strategy in the interleaved domain, and an approximation of it in the blended domain. In particular, preference neurons in the blended domain do not switch modules as soon as ghost states change, but switch modules strategically based on the number and proximity of ghosts of different types. Sometimes, these networks will even thrash back and forth between modules as they dodge threat ghosts until they find an opening to chase an edible ghost. Despite the thrashing, these strategies are still a form of threat/edible strategy.

However, threat/edible strategies earn higher scores in the interleaved domain than in the blended domain. More importantly, their scores are noticeably higher than most `1M` scores in the interleaved domain, whereas threat/edible and `1M` scores mostly overlap in the blended domain.

There is another, surprising strategy that emerges in both domains. Only networks using preference neurons can develop it, because it involves a task division that is not obvious: Networks use one module about 95% of the time to deal with threat and edible ghosts, but the remaining 5% dictates what Ms. Pac-Man does when surrounded by threat ghosts. Specifically, the most-used module eats pills while threat ghosts chase her, but sometimes the ghosts surround Ms. Pac-Man. At this point, the least-used module

[3] Available at http://nn.cs.utexas.edu/?interleaved-pm and http://nn.cs.utexas.edu/?blended-pm



(a) Interleaved Evolution Average Score Results

(b) Blended Evolution Average Score Results

(c) Interleaved Primary Module Usage Post Results

(d) Blended Primary Module Usage Post Results

Figure 2: Results During Evolution and Post-Learning Evaluations For Both Interleaved and Blended Domains. Average champion scores across 30 runs of evolution are shown for (a) interleaved and (b) blended domains. Interleaved scores are generally higher because this version of the game is easier. Most modular approaches perform well in both domains, but Multitask networks only perform well in the interleaved domain. Multitask performance in the blended domain is close to that of `1M`, which performs poorly in both domains. Post-learning evaluations in both the (c) interleaved and (d) blended domains clarify the difference. Average scores of each champion across 1,000 evaluations are plotted on the x-axis, and the percentage of time steps that each champion used its most-used module across all evaluations is plotted on the y-axis. The middle cluster in each figure corresponds to champions using a threat/edible task division, while the cluster in the top-right of each figure corresponds to a luring strategy. The gap between threat/edible and luring scores in the blended domain is larger than the corresponding gap in the interleaved domain, thus highlighting the importance of being able to discover custom task divisions in a domain with blended tasks. Videos of each evolved behavior are available in the interleaved (http://nn.cs.utexas.edu/?interleaved-pm) and blended (http://nn.cs.utexas.edu/?blended-pm) domains.

kicks in to help Ms. Pac-Man escape. Often this behavior happens near a power pill, which then makes it easy for Ms. Pac-Man to first eat the power pill, and then all ghosts. The overall behavior learned is thus a *luring* behavior, and it greatly increases the ghost score. Although champions in both domains evolve luring behavior, it is more vital in the domain with blended tasks, where the score gap between luring behavior and threat/edible behavior is larger.

A final curious result is that preference neuron networks rarely make use of three or more modules. In particular, it is not clear why `3M` or `MM(D)` networks do not develop a luring strategy with the most-used module split up into one module for edible ghosts and one for threat ghosts. The `3M` networks evolve to ignore one module, and `MM(D)` networks either stop adding modules beyond the second, or mostly ignore their few additional modules. Of course, even though a three-module division seems to make sense from a human perspective, the successful two-module luring networks show that it is not necessary. The luring networks with just two modules are apparently easier for evolution to discover than networks that behave similarly, but have more modules, possibly because of the extra coordination effort required.

6. DISCUSSION

Similar behaviors were developed in the interleaved and blended domains, but the resulting scores are different. A threat/edible division makes sense in the interleaved domain, because the two tasks are sharply divided. Luring behavior earns higher scores in this domain, but the difference is small because a simple threat/edible division already makes it easy to eat a lot of ghosts. Because all ghosts are edible at the same time, advanced planning is not needed to assure that ghosts are eaten before threats return, which would disrupt the simple strategy of pursuing the edible ghosts wherever they are.

In contrast, luring behavior is strongly needed in the blended domain. In this domain, the threat/edible split is better than using a single module, but this strategy will often fail to eat many ghosts because the return of threat ghosts disrupts it. Luring behavior allows ghosts to be quickly eaten once they become edible, which means that a mix of threat and edible ghosts are encountered less often; luring agents spend less time in the blended region between tasks. In other words, evolution discovered a task division that makes the blended domain behave more like an interleaved domain, which makes Ms. Pac-Man more successful.

In the blended domain—the standard Ms. Pac-Man—the scores achieved by the luring strategy are better than those achieved by other techniques, such as Genetic Programming [1, 4], Monte-Carlo Tree Search [2], and Ant Colony Optimization [20]; see Schrum and Miikkulainen [24] for a detailed comparison.

The results in this paper show how the choice of a learning technique should fit the type of task division. Human-specified task divisions, such as those employed by Multitask networks, work fine in an interleaved domain, but poorly in one with blended tasks. If Multitask networks were explicitly programmed to dedicate a module to luring, then they could perhaps achieve similar success. Note, however, that this task division was not obvious a priori, nor is it clear how to implement it a posteriori.

Though preference neuron networks automatically evolve good task divisions, it could be useful in future work to incorporate the formalism from Section 3 into MM-NEAT to discover appropriate task divisions more quickly and/or reliably.

Regardless, this paper not only reaffirms that letting evolution discover its own task divisions is a promising technique for domains with multiple tasks, but also identifies properties of domains that make different modular architectures more likely to succeed.

7. CONCLUSION

This paper demonstrates how behavior can be divided effectively into separate tasks in sequential decision-making problems. Two kinds of domains were identified—interleaved and blended—and modular neuroevolution techniques were used to solve them. The results in two variants of Ms. Pac-Man showed that while human-specified task divisions work well in interleaved domains, they are difficult to properly specify for blended domains. Machine discovery of multimodal behavior is therefore a promising approach to challenging sequential decision-making problems.

8. ACKNOWLEDGMENTS

This research was supported in part by NSF grants DBI-0939454, IIS-0915038, and SBE-0914796, and by NIH grant R01-GM105042.

9. REFERENCES

[1] A. M. Alhejali and S. M. Lucas. Evolving Diverse Ms. Pac-Man Playing Agents Using Genetic Programming. In *UKCI*, 2010.

[2] A. M. Alhejali and S. M. Lucas. Using Genetic Programming to Evolve Heuristics for a Monte Carlo Tree Search Ms Pac-Man Agent. In *CIG*, pages 65–72. IEEE, 2013.

[3] A. G. Barto and S. Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, pages 41–77, 2003.

[4] M. F. Brandstetter and S. Ahmadi. Reactive Control of Ms. Pac Man Using Information Retrieval Based on Genetic Programming. In *CIG*, pages 250–256. IEEE, 2012.

[5] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *Robotics and Automation*, 2(10), 1986.

[6] R. Calabretta, S. Nolfi, D. Parisi, and G. Wagner. Duplication of Modules Facilitates the Evolution of Functional Specialization. *ALife*, 6(1):69–84, 2000.

[7] R. A. Caruana. Multitask Learning: A Knowledge-based Source of Inductive Bias. In *ICML*, pages 41–48, 1993.

[8] J. Clune, J.-B. Mouret, and H. Lipson. The Evolutionary Origins of Modularity. *Royal Society B*, pages 20122863–20122863, 2013.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *Evolutionary Computation*, 6:182–197, 2002.

[10] T. G. Dietterich. The MAXQ Method for Hierarchical Reinforcement Learning. In *ICML*, 1998.

[11] B. Hengst. Discovering Hierarchy in Reinforcement Learning with HEXQ. In *ICML*, pages 243–250, 2002.

[12] J. Huizinga, J.-B. Mouret, and J. Clune. Evolving Neural Networks That Are Both Modular and Regular: HyperNeat Plus the Connection Cost Technique. In *GECCO*, pages 697–704. ACM, 2014.

[13] N. Kashtan and U. Alon. Spontaneous Evolution of Modularity and Network Motifs. *National Academy of Sciences*, 102(39):13773–13778, 2005.

[14] J. D. Knowles, R. A. Watson, and D. Corne. Reducing Local Optima in Single-Objective Problems by Multi-objectivization. In *EMO*, pages 269–283. Springer, 2001.

[15] N. Kohl and R. Miikkulainen. Evolving Neural Networks for Strategic Decision-Making Problems. *Neural Networks, Special issue on Goal-Directed Neural Systems*, 2009.

[16] G. Konidaris and A. Barto. Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining. In *NIPS*, pages 1015–1023, 2009.

[17] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[18] D. Lessin, D. Fussell, and R. Miikkulainen. Open-Ended Behavioral Complexity for Evolved Virtual Creatures. In *GECCO*, pages 335–342. ACM, 2013.

[19] J.-B. Mouret and S. Doncieux. MENNAG: A Modular, Regular and Hierarchical Encoding for Neural-networks Based on Attribute Grammars. *Evolutionary Intelligence*, 2008.

[20] G. Recio, E. Martín, C. Estébanez, and Y. Sáez. AntBot: Ant Colonies for Video Games. *TCIAIG*, 4(4):295–308, 2012.

[21] P. Rohlfshagen and S. M. Lucas. Ms Pac-Man versus Ghost Team CEC 2011 Competition. In *CEC*, pages 70–77. IEEE, 2011.

[22] J. P. Rosca. Generality Versus Size in Genetic Programming. In *GP*, pages 381–387. MIT Press, 1996.

[23] J. Schrum and R. Miikkulainen. Evolving Multimodal Networks for Multitask Games. *TCIAIG*, 4(2):94–111, 2012.

[24] J. Schrum and R. Miikkulainen. Evolving Multimodal Behavior With Modular Neural Networks in Ms. Pac-Man. In *GECCO*, pages 325–332. ACM, 2014.

[25] K. O. Stanley, B. D. Bryant, I. Karpov, and R. Miikkulainen. Real-Time Evolution of Neural Networks in the NERO Video Game. In *National Conference on Artificial Intelligence*, 2006.

[26] K. O. Stanley and R. Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation*, pages 99–127, 2002.

[27] P. Stone and M. Veloso. Layered Learning. In *ECML*, pages 369–381. Springer Verlag, 2000.

[28] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[29] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[30] T. Thompson, F. Milne, A. Andrew, and J. Levine. Improving Control Through Subsumption in the EvoTanks Domain. In *CIG*, pages 363–370. IEEE, 2009.

[31] J. Togelius. Evolution of a Subsumption Architecture Neurocontroller. *Intelligent and Fuzzy Systems*, pages 15–20, 2004.

[32] N. van Hoorn, J. Togelius, and J. Schmidhuber. Hierarchical Controller Learning in a First-Person Shooter. In *CIG*, pages 294–301. IEEE, 2009.

[33] P. Verbancsics and K. O. Stanley. Constraining Connectivity to Encourage Modularity in HyperNEAT. In *GECCO*, pages 1483–1490. ACM, 2011.