Solving Multiple Isolated, Interleaved, and Blended Tasks through Modular Neuroevolution

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

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

Abstract

Many challenging sequential decision-making problems require agents to master multiple tasks. For instance, game agents may need to gather resources, attack opponents, and defend against attacks. Learning algorithms can thus benefit from having separate policies for these tasks, and from knowing when each one is appropriate. How well this approach works depends on how tightly coupled the tasks are. Three cases are identified: Isolated tasks have distinct semantics and do not interact, interleaved tasks have distinct semantics but do interact, and blended tasks have regions where semantics from multiple tasks overlap. Learning across multiple tasks is studied in this article with Modular Multiobjective NEAT, a neuroevolution framework applied to three variants of the challenging Ms. Pac-Man video game. 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. In isolated and interleaved versions of the game, human-specified task divisions are also successful, though the best scores are surprisingly still achieved by machine discovery. Modular neuroevolution is thus shown to be capable of finding useful, unexpected task divisions better than those apparent to a human designer.

Keywords

Neuroevolution, multimodal behavior, modularity, video games, Ms. Pac-Man.

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 a distinct mode of behavior is required in each task. Learning such behavior is difficult, and even harder when semantics from individual tasks overlap.

This article presents a theoretical framework for identifying tasks in complex domains, and applies several methods of constructing modular policies for such domains using Modular Multiobjective Neuroevolution of Augmenting Topologies (MM-NEAT; Schrum and Miikkulainen, 2014), a variant of the popular NEAT algorithm (Stanley and Miikkulainen, 2002) that supports networks with several types of output modules. These modules can be associated with individual tasks. The framework of this article indicates how tasks in a domain are related to each other. 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.

A means of classifying domains with multiple tasks is presented. There are three categories, which differ based on how tightly coupled the tasks are. First, when tasks are isolated, nothing that occurs in one task can influence another task. The evaluation order does not matter, and there is a single transition from one sequentially ordered task to the next. Second, when transitions occur back and forth between tasks, but the distinctions between tasks are still clear, then the tasks are interleaved. In both isolated and interleaved cases a learning agent can easily split up a domain by dedicating separate policies to each task. In contrast, in the third case distinctions are not clear. In such domains, it is sensible to identify the prominent tasks, and treat the domain as a blend of these representative tasks. Success in these blended domains depends on an agent having the freedom to discover its own task division, but such freedom actually leads to the best performance in interleaved and isolated tasks as well.

Many domains have multiple tasks, such as robotics, sports, and video games. A robot could be deployed in multiple environments, and each deployment is a different isolated task. In baseball, the tasks of batting and fielding are isolated, since both teams stop play to switch between these roles. In sports like American football, the offensive and defensive aspects of the game are different interleaved tasks. In a sport like dodgeball, the offensive and defensive roles are blended because there are multiple balls in play, and one could be struck by a ball in the middle of attempting to throw one. Video games share many features in common with these domains, since agents must often take on offensive and defensive roles. An agent in a shooter game will generally need to avoid damage while also attacking, and also needs to forage for items, and perform other high-level strategic actions. Some games have power-ups that cause stark changes in the game dynamics, which often correspond to a task switch.

This paper builds on recent research in Ms. Pac-Man using evolved neural networks (Schrum and Miikkulainen, 2014, 2015). To support multiple tasks, networks have separate output modules to represent different behaviors. This architecture can implement multimodal behavior in two ways: A human designer can specify how modules are to be used in each task, in a style similar to Multitask Learning (Caruana, 1993), or evolution can discover the task division automatically using special preference neurons. This article gives an in-depth analysis of previous results in the blended and interleaved variants of the game, and includes new experiments in a version with isolated tasks.

The main conclusion is that approaches that discover a task division using preference neurons produce the best scores in all variants. This conclusion is true even in the new domain with isolated tasks, which would seem most likely to benefit from a clean multitask division between separate tasks. Although multitask networks can handle the domains with isolated and interleaved tasks, they are ineffective in the blended version of the game: Preference neurons are vital in this version. In all domains, the most effective task division turned out to be an unexpected one: Evolution discovers an escape module that handles situations in which Ms. Pac-Man is nearly surrounded, which only comprises 5% of the game, and a general pill and ghost-eating module that handles the remaining 95%. In addition to helping Ms. Pac-Man escape tough situations, this division implements a luring behavior. The general module gathers ghosts close together, so that the escape module can lead Ms. Pac-Man to a power pill, making it easy to capture the nearby ghosts. This highly asymmetrical task division was not obvious to a human designer a priori, demonstrating the power of the modular neuroevolution approach.

This paper proceeds by first discussing related work learning in domains with multiple tasks. Then ways of identifying domains with isolated, interleaved and blended tasks are discussed. Next, the learning methods used in these domains are explained, followed by experiments showing which methods work best in each type of domain.

2 Related Work

Multimodal behavior can often be generated with the use of modular control policies. This section presents approaches that involve separately learned controllers, Hierarchical Reinforcement Learning approaches, and single controllers with modular architectures.

2.1 Separately Learned Controllers

For complex tasks, it is common to combine controllers into a hierarchy. Such hierarchies can be hand-designed (Brooks, 1986) or learned piece by piece. For example, Togelius's (2004) evolved subsumption architecture was used in EvoTanks (Thompson et al., 2009) and Unreal Tournament (van Hoorn et al., 2009), and Stone and Veloso's (2000) Layered Learning was applied to simulated RoboCup Soccer. Recently, Lessin et al. (2013) 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.

2.2 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) also produces hierarchical controllers consisting of multiple sub-controllers. In general, HRL methods require the hierarchy to be human-specified (Dietterich, 1998), though methods for learning the hierarchy have also been developed (Hengst, 2002). However, these methods generally apply only to limited situations or produce highly constrained results. 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 (Sutton et al., 1999). Similar techniques, e.g. skills (Konidaris and Barto, 2009) and activities (Barto and Mahadevan, 2003), also fit this formalism. The methods developed 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 Section 3.

2.3 Modular Architectures

A hierarchical control policy is also a modular policy, and 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. (2000) evolved neural networks to control robots using a duplication operator, which copies one output neuron with all of its connections and weights (duplication can only be performed once per output neuron). 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 (Schrum and Miikkulainen, 2012, 2014), which introduces groups of neurons rather than individual neurons. Unlike duplication, however, Module Mutation can be performed multiple times, with no bound on the number of new modules produced. 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. (2000). Since Module Mutation is used in this paper, it is discussed further in Section 4.3.3.

Other researchers have developed modular networks with a more general concept of a module (Clune et al., 2013; Kashtan and Alon, 2005), 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 (Mouret and Doncieux, 2008; Verbancsics and Stanley,

2011; Huizinga et al., 2014; Kodjabachian and Meyer, 1998; Gruau, 1994; Suchorzewski and Clune, 2011). 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.

Another approach is to create a modular policy with a population of distinct neural networks. An example of this approach is Neural Learning Classifier Systems (Howard et al., 2010; Hurst and Bull, 2006; Dam et al., 2008), in which a single agent is controlled by a population of neural networks, subsets of which activate to handle particular situations. During learning, activated networks are generally modified according to a rule similar to that used in temporal difference learning (Sutton and Barto, 1998). Individual networks also accrue fitness whenever activated, and a genetic algorithm is periodically or probabilistically used to allow offspring of fitter networks to replace less fit networks.

Modular policies have also been explored in Genetic Programming (GP). An early example is Koza's (1994) Automatically Defined Functions, which encapsulate portions of a program tree that can potentially be reused. A similar GP technique is Adaptive Representation through Learning (Rosca, 1996; Rosca and Ballard, 1996) 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 (Sutton and Barto, 1998). 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 isolated, interleaved, or 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: $\mathcal{R} : (\mathcal{S} \times \mathcal{A} \times \mathcal{S}) \rightarrow \mathbb{R}^N$ is defined such that $\mathcal{R}(s, a, s')$ returns a tuple of the expected immediate rewards in each objective for performing action a in state s and ending up in state s'. This tuple has length N, which is the number of objectives.

The commonly used discount factor γ is excluded from this definition because only episodic domains are considered in this article. Therefore $\gamma = 1$ in all cases. 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 (Sutton and Barto, 1998). For use in an evolutionary algorithm, the sum of rewards in each objective is treated as a different fitness function subject to multiobjective optimization.

Because every episode has a start and end point, specific start and end states can be identified in $\mathcal{S}$. Strictly speaking, there can be multiple start and end states, but these can be abstracted away by using a single master start state that randomly transitions to one of the actual start states, and a master end state to which all actual end states involuntarily transition. These master start and end states will simplify the discussion of how tasks are divided in the isolated case. How-

ever, in general it is possible for task transitions to occur in the middle of an episode. Whereas episodes have definite start and end states, an agent may switch back and forth between several tasks in the course of an episode.

It is worth pointing out that many interesting domains are actually Partially Observable MDPs (POMDPs; Sutton and Barto, 1998), meaning that states cannot be distinguished without a perfect memory of all past states visited. The extra challenges of POMDPs slightly complicate the formalism provided so far, but these complications can be safely ignored in the discussion below, because the observability of states does not directly impact whether or not the domain consists of separate tasks. All POMDPs have underlying MDPs resulting from granting agents an unbounded memory of the history of observations they have made. Therefore, when a domain is actually a POMDP, it can still be categorized according to the following definitions if they apply to the domain's underlying MDP. Whether or not the agent is aware of the fact that it is in a POMDP is irrelevant.

The next three sections use the definition of an MDP to explain the concepts of isolated, interleaved, and blended tasks. This formalism provides a framework for thinking about the types of multimodal behavior needed in different domains, and how to achieve it.

3.2 Isolated Tasks

Isolated tasks do not interact with each other at all. Each such task is essentially a different problem, even when a single agent needs to be able to perform all of them. Humans are capable of performing many tasks that have nothing to do with each other. It is not uncommon for a person's leisure activities, e.g. playing basketball, drawing, or mountain climbing, to be completely unrelated to one's job, which could involve reading reports, construction, or managing employees. It is therefore clear that any fully general, artificially intelligent agent must be able to handle multiple tasks to stand on even footing with humans.

Because a domain with isolated tasks can be viewed as a collection of separate problems, its structure is the same as if it were constructed from separate MDPs. Domains with isolated tasks can be constructed by taking any set of T distinct domains $(\mathcal{S}_1, \mathcal{A}_1, \mathcal{P}_1, \mathcal{R}_1), \ldots, (\mathcal{S}_T, \mathcal{A}_T, \mathcal{P}_T, \mathcal{R}_T)$ and combining them into one MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. In this combined MDP,

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

Note that $\{\mathcal{S}_1, \ldots, \mathcal{S}_T\}$ is a partition of $\mathcal{S}$. The definition of a partition means that $\mathcal{S}$ is the union of these sets (equation 1 above), which are pairwise disjoint:

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

Importantly, each state is in exactly one task because all tasks cover $\mathcal{S}$, but are disjoint from each other. This requirement is imposed even if states from different MDPs look identical to an agent in those MDPs, since the source MDP of a given state is part of what defines that state: Two states from different MDPs are different by virtue of being from different MDPs. Of course, an agent may not know which MDP a state came from. This issue relates to POMDPs: States from different tasks may look identical to an agent, but are distinct because the tasks are distinct.

The action space is defined similarly as

$$\mathcal{A} = \bigcup_{i=1}^{T} \mathcal{A}_i, \tag{3}$$

but in this case there is no strict partition requirement. In fact, in a situation where one would actually want to have one agent learn multiple isolated tasks, it makes sense for the action spaces

of each task to be the same. Such is the case for the isolated Ms. Pac-Man domain of this article, but is not a requirement of the definition.

The transition function for the combined MDP is defined as

$$\mathcal{P}(s, a, s') = \mathcal{P}_i(s, a, s') \text{ if } s, s' \in \mathcal{S}_i \text{ and } a \in \mathcal{A}_i. \quad (4)$$

However, if $s \in \mathcal{S}_i$ and $s' \in \mathcal{S}_j$ for $i \neq j$, then $\mathcal{P}(s, a, s') = 0$ regardless of a, except in a few special cases. According to the formalism defined above, each of the constituent task MDPs has exactly one start state and one end state. So, if s^i_{start} is the start state for Task i, and s^i_{end} is the end state for Task i, then

$$\mathcal{P}(s^i_{\text{end}}, a, s^{i+1}_{\text{start}}) = 1 \text{ for all } a \in \mathcal{A} \text{ and } 1 \leq i < T. \quad (5)$$

Therefore, a single episode in the combined MDP consists of one episode in each task, in sequence. However, this construction is simply a means of fitting isolated tasks into a single formally defined MDP. In practice, the order of evaluation does not matter, and could even be done in parallel without consequence. In fact, given an MDP, one can determine if it consists of isolated tasks by checking to see if its state space can be partitioned into tasks whose evaluation order does not affect the outcome of evaluation.

The reward function $\mathcal{R}$ is the last component that needs to be defined. Because the state space is partitioned across tasks, the reward function for the whole domain is simply the reward function for the given task in which the states are. Therefore, $\mathcal{R}(s, a, s') = \mathcal{R}_i(s, a, s')$ for $s, s' \in \mathcal{S}_i$, and $\mathcal{R}(s, a, s') = 0$ for $s \in \mathcal{S}_i$, $s' \in \mathcal{S}_j$, and $i \neq j$.

It is not uncommon for isolated tasks to each have different measures of performance, which makes the use of multiobjective optimization in such domains appropriate. However, separate objectives are not a requirement of isolated tasks. For example, a person might work two separate jobs: store clerk by day, and musician by night. These jobs are isolated tasks, but performance in both could be measured by the amount of money earned.

The above discussion constructs a domain with isolated tasks from several individual tasks, but these definitions also reveal how a domain with isolated tasks can be identified. It must be possible to partition the state space into tasks whose evaluation order does not affect the outcomes within each individual task. In particular, final rewards should be the same regardless of execution order. Such an outcome is only possible if the transition function in the original domain only transitions from one task to the next in sequence, without ever returning to a previous task.

If an agent can transition back and forth between tasks, then those tasks are interleaved, as will be described next.

3.3 Interleaved Tasks

Interleaved tasks are separate and distinct, but an agent can transition back and forth between them in the course of a single episode. It is always clear which task an agent is in, but the agent can now have some control over when task transitions occur. As a result, an agent may be preparing for one task while still in another.

Isolated tasks can be thought of as an extreme example of interleaving: Each task is visited only once before shifting to the next. However, the definition of a domain with interleaved tasks is more permissive; some of the requirements of isolated tasks must be relaxed to arrive at the definition of interleaved tasks.

First, the requirement that the state spaces for the individual tasks create a partition of the combined state space will be retained. Every state is a member of exactly one task. However, more ways of transitioning between different tasks are now allowed. The state space $\mathcal{S}_i$ for any given task can have transitions to any other task $\mathcal{S}_j$ where $i \neq j$.

However, 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

$$\tau_e = x_e/m, \tag{6}$$

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:

$$x_e = |\{s_i \in e | s_i \in \mathcal{S}_j \wedge s_{i+1} \in \mathcal{S}_k \wedge j \neq k\}|, \tag{7}$$

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 5.1.2.

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. This concept applies to isolated tasks as well, but is especially important in conceptualizing interleaved tasks. 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, $\psi_1, \ldots, \psi_T : \mathcal{S} \rightarrow \{\text{True}, \text{False}\}$ are semantic predicates for the T tasks in the MDP for which each state satisfies one and only one semantic predicate:

$$\forall i \in \{1, \ldots, T\} \forall s \in \mathcal{S}_i : \psi_i(s) \text{ and} \tag{8}$$

$$\forall i, j \in \{1, \ldots, T\} \forall s \in \mathcal{S}_i : (i \neq j \rightarrow \neg\psi_j(s)). \tag{9}$$

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

$$\Phi(\psi) = \{s \in \mathcal{S} | \psi(s)\}. \tag{10}$$

For the set of predicates above that satisfy properties 8 and 9, $\Phi(\psi_i) = \mathcal{S}_i$ for all $i \in \{1, \ldots, T\}$. Generally, an initial guess at suitable predicates should lead to the desired low thrashing rate. Such predicates should also exist for any set of isolated tasks.

However, predicates with properties 8 and 9 are not easy to define in domains with blended tasks, described next.

3.4 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

$$\exists i, j \in \{1, \ldots, M\} : (i \neq j \wedge \Phi(\psi_i) \cap \Phi(\psi_j) \neq \varnothing). \tag{11}$$

That is, the set of M predicates does not satisfy property 9 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 $\phi_{(x,\neg y)}, \phi_{(\neg x,y)}, \phi_{(x,y)}$:

$$\phi_{(x,\neg y)}(s) = \psi_x(s) \wedge \neg\psi_y(s), \tag{12}$$

$$\phi_{(\neg x,y)}(s) = \psi_y(s) \wedge \neg\psi_x(s), \tag{13}$$

$$\phi_{(x,y)}(s) = \psi_x(s) \wedge \psi_y(s). \tag{14}$$

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:

$$\exists s \in \mathcal{S} : (\psi_p(s) \wedge \neg\psi_q(s)), \tag{15}$$

$$\exists s \in \mathcal{S} : (\psi_q(s) \wedge \neg\psi_p(s)), \tag{16}$$

$$\exists s \in \mathcal{S} : (\psi_q(s) \wedge \psi_p(s)). \tag{17}$$

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 is 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 blended, interleaved, and isolated 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[1]:

Definition 1 (Pareto Dominance) *Vector* $\vec{v} = (v_1, \ldots, v_N)$ *dominates* $\vec{u} = (u_1, \ldots, u_N)$ *if and only if the following conditions hold:*

1. $\forall i \in \{1, \ldots, N\} : v_i \geq u_i$, *and*
2. $\exists i \in \{1, \ldots, N\} : v_i > u_i$.

Definition 2 (Pareto Optimal) *A set of points* $\mathcal{S} \subseteq \mathcal{F}$ *is Pareto optimal if and only if it contains all points such that* $\forall \vec{x} \in \mathcal{S}$: $\neg\exists \vec{y} \in \mathcal{F}$ *such that* $\vec{y}$ *dominates* $\vec{x}$. *The points in* $\mathcal{S}$ *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; Deb et al., 2002).

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 5.2.2), 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 (Knowles et al., 2001).

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; Stanley and Miikkulainen, 2002), a constructive neuroevolution method that has been successful in many RL domains (Kohl and Miikkulainen, 2009; Stanley et al., 2006).

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 (potentially recurrent) connections between existing nodes, and node mutation splices new nodes along existing connections. NEAT also features an efficient method of topological crossover between networks.

[1] These definitions assume a maximization problem

Every new link and neuron introduced by mutation is given a unique innovation number to identify it. The genotype that encodes each neural network stores these innovations linearly in a consistent order across all members of the population. This representation makes it easy to align components with a shared origin within different genotypes, thus making crossover between networks computationally efficient.

The variant of NEAT used in this paper is Modular Multiobjective NEAT[2] (MM-NEAT; Schrum and Miikkulainen, 2014), 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. Arbitration can be accomplished with a human-specified policy via Multitask networks, or with a machine-discovered policy using preference neurons. When preference neurons are combined with Module Mutation, evolution must also settle on an appropriate number of modules. Each of these approaches is evaluated in this article.

4.3.1 Multitask Learning

Multitask networks were first proposed by Caruana (1993) 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 (or even just a single task of interest) because knowledge shared across tasks is only learned once, rather than learned independently multiple times.

In the supervised learning contexts where Multitask Learning is commonly applied, even when it is clear how to divide the tasks, it may be unclear which tasks are related enough to benefit from sharing information. For this reason, methods have been developed to learn how tasks should share information (Thrun and O'Sullivan, 1998; Kang et al., 2011).

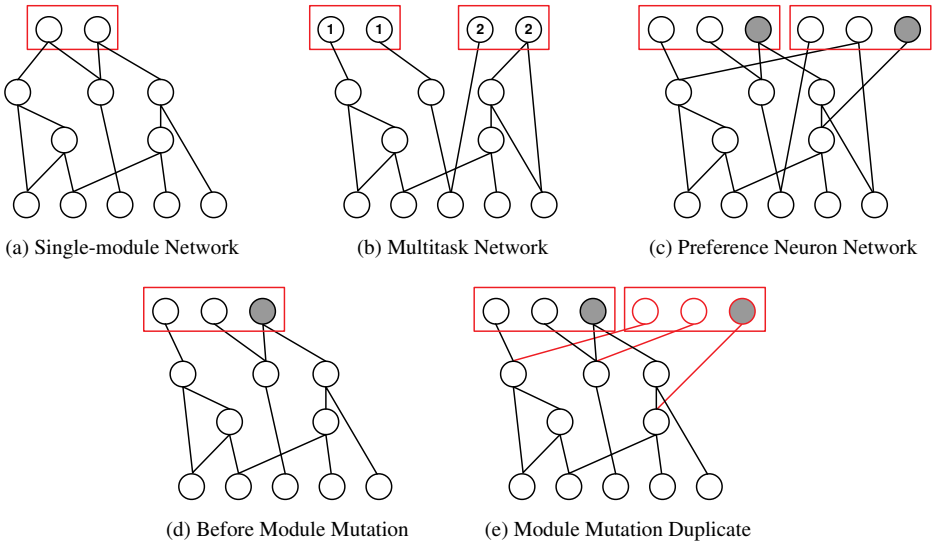
Multitask Learning with neuroevolution has been previously applied to domains with isolated tasks (Schrum and Miikkulainen, 2012). In such tasks, the evolving agents are always aware of the task they currently face. Each network has a module for each task, and these modules are initially connected only to input neurons; the modules can share information if they evolve to share hidden neurons.

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 5.2.3). 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.

4.3.2 Preference Neurons

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.

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



(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.

For example, assume a domain requires two outputs to designate the behavior of an agent, and a network has two modules (Fig. 1c). Then the network has six outputs: two policy neurons and one preference neuron for Module 1, and two policy neurons and one preference neuron for Module 2. Whenever the output of Preference Neuron 1 is higher than the output of Preference Neuron 2, the two policy neurons of Module 1 define the behavior of the agent. Otherwise, the policy neurons of Module 2 are used.

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.

4.3.3 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 (Schrum and Miikkulainen, 2012, 2014). 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 randomly chosen 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 isolated, interleaved, and blended tasks, as described next.

5 Experiments

The domains used for evaluation are three versions of the challenging 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 (Schrum and Miikkulainen, 2014), in which a mixture of edible and threat ghosts can be present at the same time. Later, an interleaved version of the domain was introduced (Schrum and Miikkulainen, 2015), in which such mixtures never occur. In this article, a new isolated version of the domain is introduced in which ghost eating is completely isolated from pill eating. The results from the three versions provide insight into when and why certain types of modular networks are successful. This section describes blended, interleaved, and isolated variants of Ms. Pac-Man, then describes experiments in each version.

5.1 Ms. Pac-Man

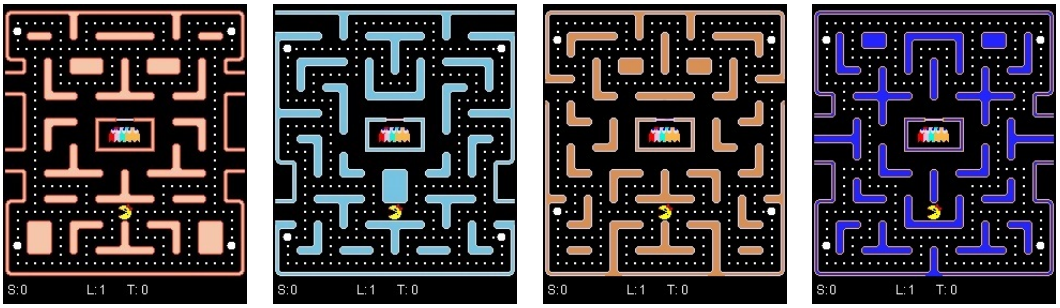
Pac-Man (1980) and its sequel Ms. Pac-Man (1981) are among the most popular video games of all time. Their gameplay is simple, yet requires complex strategies for success. A popular platform for Ms. Pac-Man research is the simulator for the Ms. Pac-Man vs. Ghosts competitions (Rohlfshagen and Lucas, 2011), which includes a standard `Legacy` ghost team that approximates the original ghosts. Many forms of Artificial Intelligence have been applied to versions of this simulator, including Neuroevolution (Lucas, 2005; Burrow and Lucas, 2009), Temporal-Difference Learning (Burrow and Lucas, 2009), Game-Tree Search (Robles and Lucas, 2009), Genetic Programming (Alhejali and Lucas, 2010, 2011; Brandstetter and Ahmadi, 2012), Monte-Carlo Tree Search (Samothrakis et al., 2011; Alhejali and Lucas, 2013; Pepels et al., 2014), and Ant Colony Optimization (Recio et al., 2012).

The original version of Ms. Pac-Man has blended tasks because Ms. Pac-Man must deal with being the predator and the prey at the same time. However, versions of the game with interleaved and isolated tasks are developed for this article. This section first describes the original, blended version of the game, and then describes the interleaved and isolated variants.

5.1.1 Blended Version

In Ms. Pac-Man, each of four mazes (Fig. 2) 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. One ghost moves randomly, which is one source



(a) Maze 0 (b) Maze 1 (c) Maze 2 (d) Maze 3

Figure 2: **Ms. Pac-Man Mazes in the Interleaved and Blended Versions.** The game of Ms. Pac-Man consists of four mazes. Ms. Pac-Man starts in the lower center of each maze, and the ghosts start in the lair, which is slightly above the center. At the start of each maze, all pills are present and available to be eaten by Ms. Pac-Man. Power pills are located near the four corners of each maze. A maze is beaten once all pills and power pills in a maze have been eaten. Though the general goals in each maze are the same, the unique structure of each maze presents a unique challenge. Despite the different structures in each maze, Ms. Pac-Man must learn behavior that generalizes across all four in order to succeed.

of non-determinism in the game. The other source applies to all ghosts: Normally, ghosts can only go forward or turn left or right, but every time step there is a 0.15% chance that all ghosts will randomly reverse direction. Such random reversals are unpredictable events that can either help or harm Ms. Pac-Man.

Reversals also occur deterministically whenever Ms. Pac-Man eats a power pill, causing ghosts to flee Ms. Pac-Man, because this event reverses the game dynamics for a limited time: Ms. Pac-Man 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.

The game has blended tasks because sometimes both threat and edible ghosts 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.

In order to create an interleaved variant of this game, the blended region between tasks must be eliminated.

5.1.2 Interleaved Version

If ghosts were always all edible or all threatening, this scenario 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 version 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 in later levels.

The game can be forced to have 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 game that is simpler than the original blended version, yet still an example of a challenging domain with interleaved tasks.

This version is also similar enough to the original game that observing the comparative performance of different types of modular policies can give insight into which methods are appropriate for which types of task divisions. Each version requires multimodal behavior to succeed, but the nature of the task divisions should be different.

Another way to evaluate and compare different modular policies is to change the nature of the task division yet again, and create a version of the game with isolated tasks, described next.

5.1.3 Isolated Version

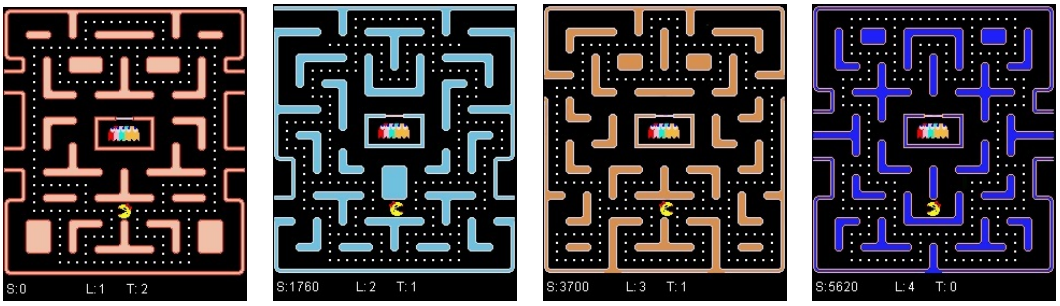
The interleaved version described above separates the domain into threat and edible tasks. However, both tasks still influence each other, and pills are present in both tasks. Because ghost score can only be increased while ghosts are edible, this objective is associated with that task. In contrast, pills are always present and can always be eaten, though more are eaten during the threat task, partly because ghosts are threats for the majority of evaluation.

When deriving isolated tasks from the original game, it is therefore reasonable to have one task for eating ghosts and one for eating pills. Additionally, the ghost-eating task should only feature edible ghosts, and the pill-eating task should only feature threat ghosts. These considerations lead to the following isolated tasks.

The pill-eating task is like the original game, but without power pills. It is thus impossible for threat ghosts to become edible, so Ms. Pac-Man can always treat the ghosts as threats. In other words, within the pill-eating task, Ms. Pac-Man does not need a distinct mode of behavior that handles edible ghosts.

This modification makes the game much more challenging, because there are particular pills in each maze that are difficult to eat without the protection offered by a power pill. The most prominent of these pills are those near power pills, because power pills are always located near the corners of the maze. Without power pills it is dangerous for Ms. Pac-Man to trap herself in a corner. Therefore, all pills on the same C-path (i.e. a path that connects two junctions and contains no junctions; Ikehata and Ito, 2011) as a power pill are also removed from each maze (Fig. 3). Removing this small number of pills slightly reduces the maximum possible pill score, but also makes the task a more reasonable test of performance; without power pills, some of the most dangerous-to-eat pills can only be safely eaten by going in circles and waiting for a random ghost reversal to create an opportunity. The isolated pill-eating task removes many of these pills. However, there are still several dangerous pills left, particularly in the 3rd maze (Fig. 3c), so this task is very challenging.

In contrast, the ghost-eating task is now fairly easy. It is designed to faithfully capture the focus of the ghost-eating task in the blended and interleaved versions. First all pills are removed. Then, for each of the four power pills in each of the four mazes, Ms. Pac-Man starts at the location of the power pill, as if she had just eaten it (Fig. 4). The ghosts are edible, and Ms. Pac-Man has the usual amount of edible time available. As with the interleaved version, eaten ghosts cannot exit the lair while other ghosts are still edible. However, the evaluation ends as soon as the edible time runs out or all ghosts are eaten, which means Ms. Pac-Man will never encounter a threat ghost in this task. Because this type of evaluation is repeated for each of four power pill locations and each of four mazes, a single evaluation actually consists of 16



(a) Maze 0 (b) Maze 1 (c) Maze 2 (d) Maze 3

Figure 3: **Ms. Pac-Man Mazes in the Isolated Pill-Eating Task.** The pill-eating task of the isolated version is more challenging than the standard Ms. Pac-Man game because there are no power pills. Without the temporary invulnerability and ability to eat ghosts offered by power pills, Ms. Pac-Man must be very careful about when she approaches certain areas in each maze, because it is easy to be trapped by the ghosts. To make the challenge level more reasonable, pills that were originally on the same C-path as a power pill are removed. However, this simple fix still leaves some dangerous areas in certain mazes. In particular, the unusual power pill placement in Maze 2 (Fig. 3c) means that there are still pills in the lower corners. This maze also has dangerous pills in an area directly below Ms. Pac-Man's starting point. Therefore, very skilled behavior is required to clear all levels in the pill-eating task.

very quick evaluations whose scores add up. In fact, these 16 evaluations can be seen as 16 very similar isolated tasks, but are grouped together and labelled as the ghost-eating task.

To make the isolated ghost-eating task as similar to the blended ghost-eating task as possible, the initial locations of the ghosts in each evaluation are chosen based on recordings of skilled play in the blended version. Specifically, the best Ms. Pac-Man agent from the experiments in the blended game described below (Section 5.3.1) was subjected to enough games to eat each power pill at least 100 times. At each such event, the locations of the ghosts were saved, creating a database of 100 entries for each maze and power pill. In the isolated ghost-eating task, each time Ms. Pac-Man starts an evaluation, the ghost locations are initialized by randomly picking one of the 100 possibilities corresponding to the current maze and starting power pill location.

Experiments are conducted in each of the three versions described to demonstrate how different approaches to modular neuroevolution deal with different types of task divisions.

5.2 Experimental Setup

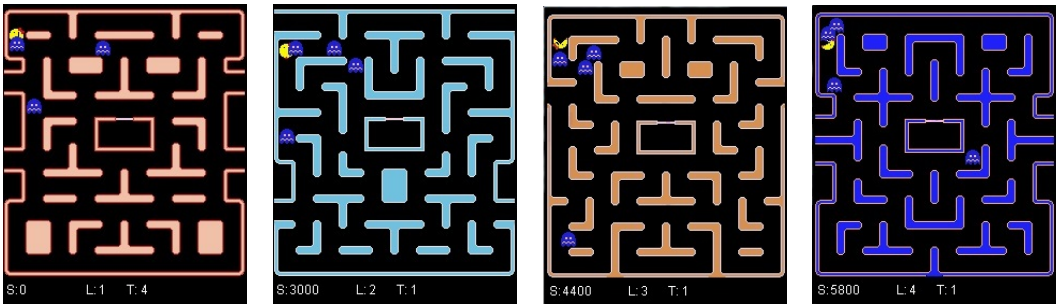
Neural networks are evolved in the manner described by Schrum and Miikkulainen (2014, 2015), details of which are provided below.

5.2.1 Sensors

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 (Table 1) and sensors that are not direction-oriented (Table 2), 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 value of 200, and all sensors are scaled to $[0, 1]$.

Most direction-oriented sensors (Table 1) are similar to those of Brandstetter and Ahmadi (2012), who used Genetic Programming to evolve Ms. Pac-Man agents. A notable exception is the Options From Next Junction (OFNJ) sensor. OFNJ looks at the next junction in a given



(a) Maze 0 (b) Maze 1 (c) Maze 2 (d) Maze 3

Figure 4: **Ms. Pac-Man Mazes in the Isolated Ghost-Eating Task.** Capturing edible ghosts is easy in the isolated ghost-eating task. All ghosts start out edible, and Ms. Pac-Man is allotted the standard amount of edible time to eat them. Ms. Pac-Man does not need to worry about dodging threat ghosts because ghosts remain imprisoned in the lair after being eaten, and she does not need to lure ghosts near power pills because ghosts start at locations taken from a database of situations experienced by a skilled Ms. Pac-Man agent that does know how to lure ghosts. The database contains entries for each maze corresponding to the four power pills in each maze, which are the locations that Ms. Pac-Man starts at in each evaluation. These figures show her at the location of the upper left power pill of each maze. Notice that because of the skill of the agent that established the database of ghost start locations, Ms. Pac-Man often begins evaluation extremely close to edible ghosts, which makes eating them very easy. However, there are also situations where a ghost starts far away from Ms. Pac-Man (Fig. 4c). These far ghosts can prevent an optimal agent from attaining a perfect score, but very good scores are attainable despite this small amount of randomness.

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. Brandstetter and Ahmadi used a weaker version of this sensor that merely detects whether an upcoming junction is blocked by a threat. OFNJ goes further by indicating whether a ghost could reach the junction in question, which does a better job of helping Ms. Pac-Man avoid death.

The remaining sensors are not direction-oriented (Table 2). Most of these undirected sensors measure useful proportions. Sensing whether any ghost is edible indicates when some are vulnerable, but does not provide information about any specific ghost. Awareness of threats and nearby power pills (Alhejali and Lucas, 2010, 2011) is useful because it helps Ms. Pac-Man optimize the timing of eating a power pill in order to maximize the number of ghosts eaten.

5.2.2 Evaluation

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 1^{st}, 2^{nd}, 3^{rd}, and 4^{th} 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 evaluations take a long time to carry out, a limit of 8,000 time steps is imposed for each maze, after which Ms. Pac-Man is killed (in the isolated version, this restriction applies only to the pill-eating task). Furthermore, Ms. Pac-Man is given only one life, rather than the usual three. The time limit discourages behaviors that stay alive a long time without making progress, such as moving in circles while the ghosts chase from behind. This time limit is high

Table 1: **Directed Sensors in Ms. Pac-Man.**

Sensor Name	Description
Nearest Pill Distance	Distance to nearest regular pill in given direction
Nearest Power Pill Distance	Distance to nearest power pill in given direction
Nearest Junction Distance	Distance to nearest maze junction in given direction
Max Pills in 30 Steps	Number of pills on the path in the given direction that has the most pills
Max Junctions in 30 Steps	Number of junctions on the path in the given direction that has the most junctions
Options From Next Junction	Number of junctions reachable from next nearest junction that Ms. Pac-Man is closer to than a threat ghost
n^{th} Nearest Ghost Distance	Distance to n^{th} nearest ghost in given direction for $n = 1, \ldots, 4$
n^{th} Nearest Ghost Approaching?	1 if n^{th} nearest ghost in given direction is approaching, 0 otherwise, for $n = 1, \ldots, 4$
n^{th} Nearest Ghost Path Has Junctions?	1 if directional path to n^{th} nearest ghost contains any junctions, 0 otherwise, for $n = 1, \ldots, 4$
n^{th} Nearest Ghost Edible?	(blended version only) 1 if n^{th} nearest ghost in given direction is edible, 0 otherwise, for $n = 1, \ldots, 4$

The maximum distance that can be sensed is 200. Higher distances, and distances to objects that are no longer in the maze, are reduced to 200. All such distance sensor values are divided by 200 so that they are confined to the range $[0, 1]$. The remaining sensors are similarly scaled to the range $[0, 1]$ according to their maximum values. These sensors are direction oriented, meaning that they can compute different values for each direction. Distance measurements and object counts are made along the shortest path in the given direction without reversing. Sensors referring to the n^{th} Nearest Ghost are actually sets of four sensors, one for each of the four ghosts sorted by their distance from Ms. Pac-Man along a directional path. The last of these, n^{th} Nearest Ghost Edible?, is only used in the blended version because it is unnecessary in the interleaved and isolated versions: the undirected sensor Any Ghosts Edible? (Table 2) provides complete information about edible ghosts instead, because if any ghost is edible in those versions, then all of them are. Overall, the directed and undirected sensors provide a comprehensive view of the maze that makes it possible for Ms. Pac-Man to make intelligent decisions.

Table 2: **Undirected Sensors in Ms. Pac-Man.**

Sensor Name	Description
Bias	Constant value of 1
Proportion Pills	Number of regular pills left in maze
Proportion Power Pills	Number of power pills left in maze
Proportion Edible Ghosts	Number of edible ghosts
Proportion Edible Time	Remaining ghost edible time
Any Ghosts Edible?	1 if any ghost is edible, 0 otherwise
All Threat Ghosts Present?	1 if four threats are outside the lair, 0 otherwise
Close to Power Pill?	1 if Ms. Pac-Man is within 10 steps of a power pill, 0 otherwise

All sensors that measure a proportion are scaled to the range $[0, 1]$. These sensors do not depend on direction, so the same values will be returned for each potential movement direction on each time step. They can only meaningfully influence direction preference when combined with direction-oriented sensors (Table 1).

enough to not affect the champions by the end of evolution. Having a single life makes good performance depend more strongly on skill than luck. Additionally, Ms. Pac-Man progresses through each of the four mazes only once before evaluation is finished. Ms. Pac-Man revisits mazes repeatedly in the commercial game, but this four-maze evaluation approach is less time

consuming and has been used by others (Alhejali and Lucas, 2010, 2011).

5.2.3 Evolved Networks

In the blended, interleaved, and isolated 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.

Multitask networks require human-specified task divisions. In the isolated version, the Multitask (MT) approach has one module each for the ghost-eating and pill-eating tasks. The MT approach also has one module per task in the interleaved game, in which the tasks are to deal with edible and threat ghosts. 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 $\mu = \lambda = 100$. 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 were chosen after initial experimentation in the blended version of Ms. Pac-Man (Schrum and Miikkulainen, 2014), and are used in all three versions of the game in this article. These settings lead to the results described in the next section.

5.3 Results

In general, modular networks perform well in all three versions, particularly those with preference neurons, while 1M performs the worst. The degree to which the modular approaches outperform 1M depends on the version of the game.

5.3.1 Blended Version Results

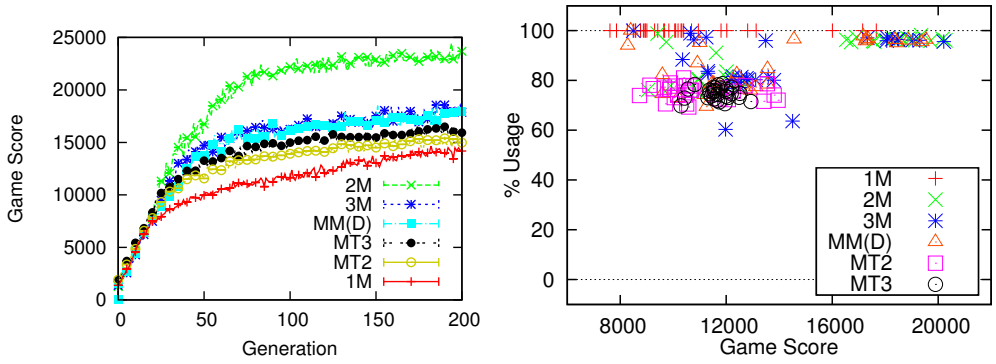
Fig. 5a shows that in the blended version, 2M networks are far superior to all other approaches. The next best approaches are 3M and MM(D), which also use preference neurons. Both Multitask methods (MT3 and MT2) are worse than the preference neuron approaches, and 1M is the worst of all.

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. Applying the Kruskal-Wallis test to the results from post-evolution evaluations indicates that there is a significant difference between at least two of the six methods ($H = 49.694, df = 5, N = 30, p \approx 1.6 \times 10^{-9}$). Table 3 shows *adjusted* p-values for post-hoc Mann-Whitney U tests. The results support the general conclusion that modular networks are better than single-module networks. Additionally, preference neuron networks are better than Multitask networks.

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 Fig. 5b. Color-coded movies[3] of the module usage are observed to determine the role of each module.

A common strategy is to dedicate one module to dealing with edible ghosts (roughly 25% of the time) and another to threat ghosts (roughly 75%). The MT2 and MT3 task divisions approximate this division in different ways (the 25% that MT3 dedicates to edible ghosts is split across two modules). Some 2M, 3M, and MM(D) networks also learn a threat/edible strategy, but

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



(a) Blended Median Score Evolution Results

(b) Blended Primary Module Usage Post Results

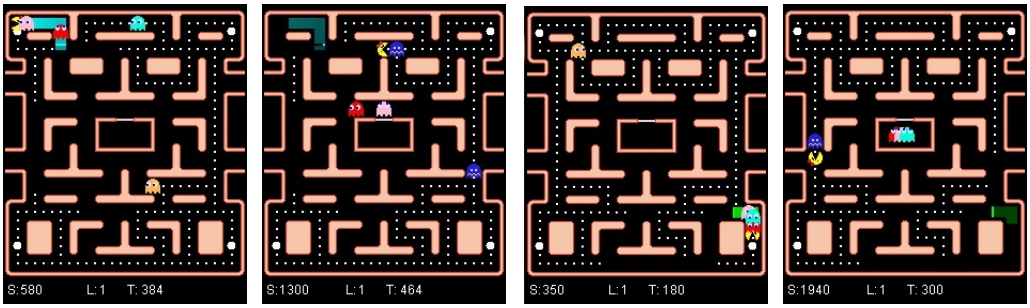
Figure 5: **Results During Evolution and Post-Learning Evaluations For Blended Version.** Median champion scores across 30 runs of evolution are shown for the blended version in Fig. 5a. The `2M` networks are vastly superior to all others, but all modular approaches are superior to `1M`. Post-learning evaluations in the blended version are shown in Fig. 5b. Average scores of each champion across 1,000 evaluations are plotted on the x-axis. The fact that these scores are lower than those achieved during evolution demonstrates the impact of noisy evaluations. Although champions may achieve very high scores across ten evaluations during evolution, evaluating the same champions 1,000 times provides a more accurate and slightly lower score. The percentage of time steps that each champion used its most-used module across all evaluations is plotted on the y-axis. The resulting clusters give insight into different levels of performance achieved during evolution. The middle cluster (scoring between 8,000 and 15,000) corresponds to champions using a threat/edible task division (threat module used roughly 75% of the time), while the cluster in the top-right (scoring 16,000 to 20,000) corresponds to a luring strategy that depends on an escape module (other module used over 95% of the time). The gap between threat/edible and luring scores in the blended version is large, because of how important it is to discover custom task divisions in a domain with blended tasks. Videos of each evolved behavior are available in the blended version at `http://nn.cs.utexas.edu/?blended-pm`.

Table 3: **Adjusted *p*-Values From Pairwise Mann-Whitney *U* Tests Comparing Post-Evolution Blended Version Results.**

	1M	MT2	MT3	MM(D)	3M
MT2	0.57789	-	-	-	-
MT3	**0.00245**	0.24268	-	-	-
MM(D)	**0.00164**	**0.03332**	1.0	-	-
3M	**0.000015**	**0.00021**	**0.00679**	1.0	-
2M	**0.000011**	**0.00039**	**0.01638**	1.0	1.0

The champion of each run of each method was evaluated after evolution in 1,000 further trials. Each number is a p-value resulting from a two-tailed Mann-Whitney U test comparing two neuroevolution methods, adjusted according to Bonferroni correction as performed by `R`. Values below 0.05 (**bold**) indicate statistically significant differences. Methods are sorted from worst to best according to the order established in Fig. 5a, which results in most significant differences clustering near the lower-left of the table. Except for `MT2`, all modular approaches are better than `1M`. All preference neuron approaches are significantly better than `MT2`, and `2M` and `3M` are both significantly better than `MT3`. These results provide a deeper analysis of champion performance than the results during evolution. The main result is that modular networks in general and preference neuron networks in particular have an advantage over single-module networks.

preference neurons in the blended version 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.



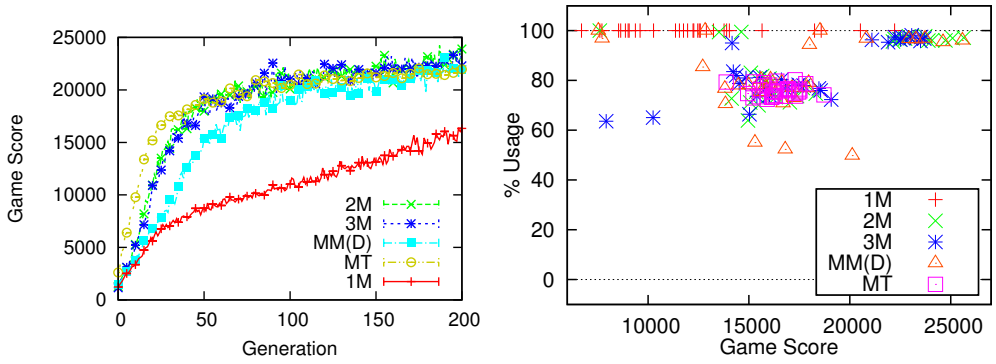
(a) Blended: Escaping To Power Pill (b) Blended: After Eating Power Pill (c) Interleaved: Escaping to Power Pill (d) Interleaved: After Eating Power Pill

Figure 6: **Luring Behavior With an Escape Module in Blended and Interleaved Versions.** (a) Ms. Pac-Man waits at the junction for the ghosts to get close, then activates the escape module, which leads her and the ghosts to the power pill. The maze cells in the upper left are shaded blue to indicate locations in which the escape module was used. (b) After eating the power pill, Ms. Pac-Man quickly eats the ghosts that were chasing her, then chases the remaining ghosts. The escape module is not used after the power pill is eaten. An animation of this and other behaviors in the blended version can be seen at `http://nn.cs.utexas.edu/?blended-pm`. (c) In the interleaved version, Ms. Pac-Man lures and escapes in a similar fashion. (d) However, because eaten ghosts remain imprisoned while free ghosts are edible, interleaved champions do not need to be as skilled at luring to get high ghost-eating scores. In fact, even champions that do not lure at all can achieve high ghost scores, as seen in the videos at `http://nn.cs.utexas.edu/?interleaved-pm`. In summary, the very best champions in both versions use an escape module for luring. Though it is rarely activated, half of the network's neural resources are dedicated to this module because luring is only effective if Ms. Pac-Man can successfully escape the surrounding threats to reach a power pill, which in turn leads to the highest scores.

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.

There is another, surprising strategy that proves vital in the blended version (Figs. 6a and 6b). 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 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. This behavior is vital in the blended version, as demonstrated by the large score gap between luring networks and threat/edible networks.

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. In fact, the ability of a single module to both dodge threats and eat edible ghosts indicates the presence two separate modes of behavior, even though there are no distinct modules for each mode. Multiple



(a) Interleaved Median Score Evolution Results

(b) Interleaved Primary Module Usage Post Results

Figure 7: **Results During Evolution and Post-Learning Evaluations For Interleaved Version.** Median champion scores across 30 runs of evolution are shown for the interleaved version in Fig. 7a. Interleaved scores are generally higher than in the blended version because the interleaved version is easier. However, all modular approaches achieve scores much higher than `1M`. There is no longer a large distinction between Multitask and preference neuron approaches; all do well in the interleaved game. However, distinct behaviors are still learned by different approaches, as shown by the analysis of post-learning evaluations shown in Fig. 7b. The middle cluster (scoring 13,000 to 19,000) still corresponds to champions using a threat/edible task division, and the top-right cluster (scoring 20,000 to 26,000) still corresponds to a luring strategy with an escape module. However, the gap between threat/edible and luring scores in the interleaved version is smaller than the corresponding gap in the blended version, which reveals that discovering a custom task division with preference neurons is not as important in the interleaved version. Videos of each evolved behavior are available in the interleaved version at `http://nn.cs.utexas.edu/?interleaved-pm`.

behavioral modes can exist in a single module.

The results above show why modular networks with preference neurons are so important for domains with blended tasks, but the comparative performance of these methods in the interleaved version are slightly different, as described next.

5.3.2 Interleaved Version Results

Fig. 7a shows how in the interleaved version, `2M`, `3M`, `MM(D)` and `MT` all quickly achieve higher scores than `1M`. In the final generation, each of these modular approaches still has much higher scores than `1M`.

As with the blended results, post-learning evaluations show that these differences are statistically significant. After 1,000 evaluations for each champion, the average scores were compared with the Kruskal-Wallis test, showing that there is a significant difference between at least two of the five methods ($H = 37.0228, df = 4, N = 30, p \approx 1.78 \times 10^{-7}$). Table 4 shows *adjusted* p-values for post-hoc Mann-Whitney U tests. The results make it clear that in the interleaved version, all modular approaches are better than `1M`, but none are different from each other.

Analyzing the module usage of evolved champions in post-evaluations (Fig.7b) reveals that similar behaviors emerge, but the relative effectiveness of certain strategies is different. The highest scores are still earned by networks that use preference neurons in conjunction with an escape module to exhibit luring behavior. Having distinctive modules for edible and threat tasks is also common, and is achieved by representatives of all modular approaches. However, the interleaved version differs from the blended version in that threat/edible strategies earn higher scores. Specifically, the gap between threat/edible and luring champion scores is smaller in the interleaved version, and the gap between threat/edible strategies and most `1M` champions is larger. In the blended version, threat/edible scores mostly overlap with `1M` scores, which is

Table 4: **Adjusted *p*-Values From Pairwise Mann-Whitney *U* Tests Comparing Post-Evolution Interleaved Version Results.**

	1M	MT	MM(D)	3M
MT	**0.0000035**	-	-	-
MM(D)	**0.00011**	1.0	-	-
3M	**0.0000031**	1.0	1.0	-
2M	**0.00000092**	1.0	1.0	1.0

Interleaved champions were also evaluated in 1,000 additional trials after evolution had finished, and the results of pairwise Mann-Whitney U tests are shown in this table as in Table 3. In the interleaved version, all modular approaches are significantly better than 1M, but none are significantly different from each other. This contrast with the blended results demonstrates how a domain with a clear, interleaved task division can be handled with a wider variety of modular approaches.

why luring is so important in that version. However, all modular approaches do well in the interleaved version because the clear task boundaries make it easier to apply multiple modules effectively.

Movies of behaviors in the interleaved version[4] demonstrate that multiple modules can easily be used effectively without needing any complicated reasoning about mixtures of threat and edible ghosts. Networks using a threat/edible strategy simply use the module that matches the current state of the ghosts, without needing to have special reasoning to handle a blended region between tasks. It is also easier to have an effective luring strategy, because threat ghosts do not re-emerge from their lair to harass Ms. Pac-Man as she tries to eat the last edible ghost(s) (Figs. 6c and 6d).

Overall, these results indicate that while human-specified and machine-learned task divisions can both handle the interleaved version well, human-specified task divisions have difficulty with blended tasks.

Because isolated tasks are more clearly divided than interleaved tasks, all modular approaches, including human-specified ones, generally work well in the isolated version too, as explained next.

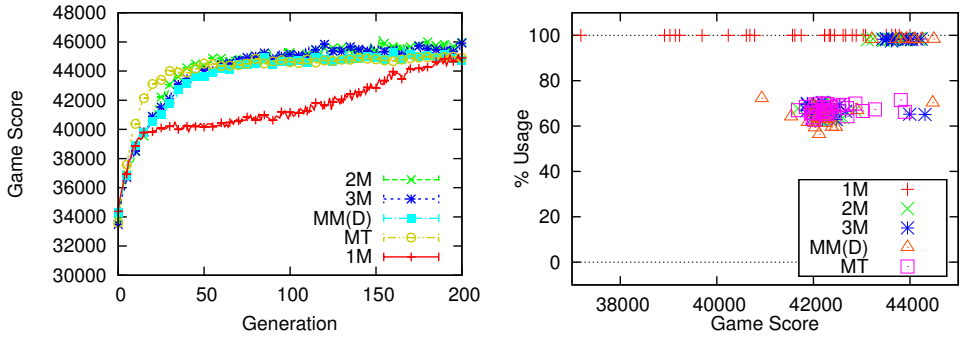
5.3.3 Isolated Version Results

Fig. 8a shows how the median level of performance for each method reaches roughly the same level by 200 generations. However, all modular approaches are superior to 1M early in evolution. All scores are in a high but narrow score range because all methods quickly master the relatively easy ghost-eating task. The methods differ in the pill-eating task, which is harder than the original blended version of the game.

The plot of median scores throughout evolution indicates that, in the end at least, there is not much difference between methods. However, when analyzing post-learning evaluations of the champions (Fig. 8b), the Kruskal-Wallis test shows that there is a significant difference between at least two of the five approaches ($H = 12.5529, df = 4, N = 30, p \approx 0.01368$). Yet again, no specific difference between methods could be confirmed with post-hoc Mann-Whitney U tests (Table 5).

However, despite the similar scores, champions with multiple modules are still the best. This result is confirmed in videos of the evolved behaviors[5]. Moreover, even without power pills to lure ghosts towards, networks with an escape module are still the most successful. The escape module is rarely used in the first maze, because the removal of pills that were near a power pill make this maze easy to clear. The escape module is more useful in the second maze,

[4] Available at http://nn.cs.utexas.edu/?interleaved-pm
[5] Available at http://nn.cs.utexas.edu/?isolated-pm



(a) Isolated Median Score Evolution Results

(b) Isolated Primary Module Usage Post Results

Figure 8: **Results During Evolution and Post-Learning Evaluations For Isolated Version.** Median champion scores across 30 runs of evolution are shown for the isolated version in Fig. 8a. The range of scores is very narrow because eating ghosts is much easier in the isolated version. The methods primarily differ in the pill-eating task, which has less of an impact on game score. Nevertheless, all modular approaches have a clear advantage over `1M` early in evolution, because they can easily dedicate a separate module to each task. However, most `1M` runs do eventually find a way to achieve performance comparable to the modular methods (note that only the median performance is plotted). Post-learning evaluations of the champions (Fig. 8b) indicate that there are still several low-scoring `1M` networks, and that modular champions form the usual clusters. There is a large cluster (scoring 41,000 to 43,500) that uses its primary module around 70% of the time to handle the pill-eating task, and its second module in the ghost-eating task. There is also a cluster of modular champions in the upper right (scoring 43,000 to 45,000) that uses its primary module over 95% of the time. This primary module is used in the ghost-eating task, and in the majority of the pill-eating task. The second module is an escape module, as in the blended and interleaved versions. However, since there are no power pills, the escape module does not help Ms. Pac-Man set up a chance to eat ghosts. The escape module is used purely to escape dangerous situations, but this ability ends up being very important in the challenging pill-eating task. There are also a small number of preference neuron networks with three modules (scoring 43,500 to 45,000) that have an escape module, as well as separate pill-eating (primary module used around 70% of the time) and ghost-eating modules. Intelligent use of three modules is impressive, but ultimately unnecessary, as these scores are not better than champions with just an escape module and one other module. Videos of each evolved behavior are available in the isolated version at `http://nn.cs.utexas.edu/?isolated-pm`.

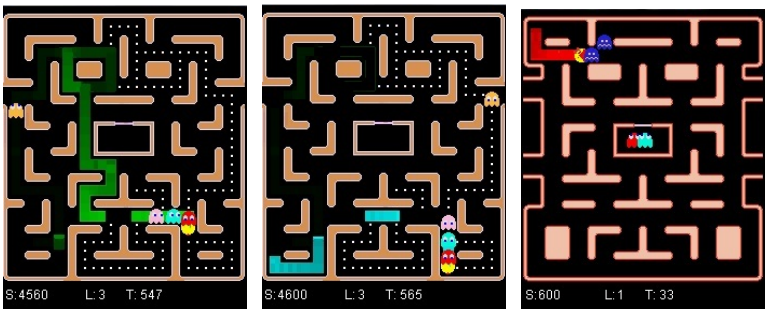
Table 5: **Adjusted *p*-Values From Pairwise Mann-Whitney *U* Tests Comparing Post-Evolution Isolated Version Results.**

	1M	MT	MM(D)	3M
MT	1.0	-	-	-
MM(D)	0.699	1.0	-	-
3M	0.110	0.571	1.0	-
2M	0.080	0.055	1.0	1.0

Isolated champions were also put through 1,000 post evaluations, and the results of pairwise Mann-Whitney U tests are shown in this table as in Tables 3 and 4. In the isolated version, no significant differences are detected via Mann-Whitney U tests, despite the significant difference detected by the Kruskal-Wallis test. However, even without a significant difference in scores, there is a difference in how champions of each method behave and use modules.

and vital in the third maze, which contains three extremely dangerous areas near the bottom of the maze. The fourth maze is also challenging, but not as much as the third maze.

Fig. 9 shows one of the rare champions that make intelligent use of three modules: one for escape in the pill-eating task, one for all other situations in the pill-eating task, and one for the



(a) Isolated: Primary Pill Task Module (b) Isolated: Escape Module (c) Isolated: Ghost Task Module

Figure 9: **Intelligent Use of Three Modules in the Isolated Version of Ms. Pac-Man.** These figures show the behavior of an MM(D) champion with three modules. Each module has a different color trail associated with it. (a) For the majority of the pill task, the green module controls Ms. Pac-Man and leads her to navigate the maze and eat pills. (b) However, there are dangerous situations where the default behavior is not good enough. The blue escape module activates to help Ms. Pac-Man make decisions when she is nearly trapped. (c) A separate red module is used to eat ghosts in the ghost task. This champion is a rare instance of a network that intelligently uses three modules. It earns high scores, though not any better than two-module champions that use an escape module. The key to high scores is the escape module. Learning to use the other module for both pill and ghost eating is apparently not difficult. A video of this behavior and others in the isolated version can be seen at http://nn.cs.utexas.edu/?isolated-pm.

ghost-eating task. Only one MM(D) champion and two 3M champions use three modules in this manner, but they are among the best networks.

Learning a module division that splits up the pill-eating and ghost-eating tasks is also common, though not as effective despite the fact that the tasks are completely isolated. The reason that one module per task it not enough in most cases is that the pill-eating task is so much harder than the ghost-eating task. Dedicating half of the network's neural resources to it is generally not effective. However, there are exceptions. All MT networks are forced to use a human-specified task division, yet the two highest scoring MT champions have scores comparable to modular approaches using an escape module. The best 1M champions also have scores in this range, though the majority of 1M champions score much lower. Discovering multimodal behavior is thus possible within a single module, but is much easier to do with multiple modules.

6 Discussion and Future Work

Similar behaviors were developed in all three versions of Ms. Pac-Man, but the performance of the methods differs.

Using an escape module leads to the best scores in all versions. In the interleaved and blended versions, the escape module helps Ms. Pac-Man avoid death, but it is also vital in allowing her to eat power pills at the best possible time so that she can eat as many ghosts as possible. In the isolated version, the escape module is still important, even without power pills: It helps Ms. Pac-Man survive the very difficult pill-eating task. Only networks with preference neurons could use an escape module, because such a task division was not human-specified, but rather discovered by evolution.

A threat/edible division was commonly discovered, but its effectiveness varied across versions. It is always less effective than using an escape module. In the blended version, threat/edible divisions overlap substantially with poor 1M results. Threat/edible divisions are more effective in the interleaved version, and are clearly superior to poor 1M results. Because all ghosts are edible at the same time, advanced planning is not needed to assure that ghosts are

eaten before threats return. Threat/edible divisions are also effective and superior to poor `1M` results in the isolated version. However, there are fewer poor `1M` results in the isolated version than in the other versions, because the ghost-eating task is so easy.

Threat/edible divisions can be discovered by evolution via preference neurons, but are also used by Multitask networks. This human-specified task division is a natural choice for the isolated version, because one module is dedicated to each of the isolated tasks, which are defined according to Section 3.2. This division also fits well in the interleaved version, whose tasks fit the definition in Section 3.3, because the decision of which module to use can be simply based on whether the ghosts are edible: Ms. Pac-Man pursues edible ghosts and flees threat ghosts. The threat/edible division is less well suited to the blended version, because it is not obvious which states in the blended region between tasks should be associated with each module. The three-module Multitask approach assigns a module to the blended region between tasks, as suggested by the predicate of equation 14 in Section 3.4, but this approach is inferior to the custom division discovered by evolution. With both threat and edible ghosts, it is more intelligent to make a decision about which module to use based on how close the different types of ghosts are in each direction. Regardless of which module is chosen, Ms. Pac-Man still needs to deal with ghosts of both types.

These results show that learning multimodal behavior is easier when multiple modules are available. However, a few `1M` champions did exhibit some form of multimodal behavior as well. In many cases, the powerful OFNJ sensor makes this possible. Avoiding a particular action based solely on this sensor allows Ms. Pac-Man to have a mode of behavior that keeps her alive. If a preference for pill eating is learned on top of this basic skill, then Ms. Pac-Man has at least a decent chance of clearing several mazes. In other words, good sensors can make it possible to learn multiple modes of behavior within a single module, but the specific behavioral modes that can be learned depend on the specific sensors available. In addition, modular networks that in principle can include three or more modules settle on two modules because one module can both avoid ghosts and eat pills. Some modules contain multiple modes of behavior while others contain just one.

Both of these basic behaviors are required in the pill-eating task of the isolated version. This task is completely isolated from the ghost-eating task, yet pill-eating and ghost-avoidance behaviors are actually blended inside the pill-eating task. One of the isolated tasks is itself a blended domain, despite having only one fitness objective (pill score). However, the boundary between these pill-eating and ghost-avoidance tasks is even less clear than the boundary between threat and edible tasks in the blended version of the game. As a result, evolution must discover when Ms. Pac-Man is threatened enough to use a different mode of behavior. The threat ghost subtasks of the interleaved and blended versions also consist of these blended tasks. This observation implies that the formalism described earlier in the article is in fact hierarchical: isolated tasks can be interleaved or blended domains, and interleaved tasks can be blended domains.

This hierarchy is likely the reason why allowing evolution to discover its own task division using preference neurons was the most successful in each version of the game. Only preference neurons could assign an escape module to the escape task that was blended into each version. This ability imparts a slight but noticeable score boost in the isolated and interleaved versions, but is essential in the blended version.

In fact, the luring behavior that results from having an escape module allows ghosts to be quickly eaten once they become edible, which means that a mix of threat and edible ghosts is 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 version behave more like an interleaved domain, which makes Ms. Pac-Man more successful. In the blended version—the standard Ms. Pac-Man—the scores achieved by the luring strategy are better than

those achieved by other techniques, such as Genetic Programming (Alhejali and Lucas, 2010; Brandstetter and Ahmadi, 2012), Monte-Carlo Tree Search (Alhejali and Lucas, 2013), and Ant Colony Optimization (Recio et al., 2012); see Schrum and Miikkulainen (2014) 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, are good at splitting up isolated and interleaved domains, but do not handle blended tasks well. Furthermore, if blended subtasks are embedded within an isolated or interleaved task, a Multitask division will have trouble recognizing and handling those subtasks. Of course, if Multitask networks were explicitly programmed to dedicate a module to escaping, then they could perhaps achieve similar success. Note, however, that use of an escape module is not immediately obvious, nor is it clear how to implement precisely when it should activate: the factors that influence it include how close the ghosts are, as well as the number and safety of available junctions in all four directions.

It might be possible to exploit the hierarchical nature of task divisions with a hybrid method that involves both human encoding and machine discovery. One could evolve networks that have distinct groups of modules arbitrated by preference neurons, but that arbitrate between groups through a human-specified task division. Once the human-specified division had chosen a group of sub-modules, preference neurons would determine which specific module is used. This approach would likely be most effective if the human-specified division functioned at a high, clearly defined level, such as between isolated or interleaved tasks, and preference neurons were allowed to handle the blended tasks. However, even splitting up different types of blended tasks with a mixture of human-specified and machine-discovered divisions could be useful. For example, a human could split the blended version based on threat and edible ghosts, and then leave discovery of an escape module within the task with threat ghosts to preference neurons.

Such an approach may be necessary for tasks that are more complex than Ms. Pac-Man. In fact, although many domains can be described with the formalism of Section 3, it may need to be extended to describe more complex domains. For example, a first-person shooter game like Unreal Tournament 2004 clearly has blended tasks, but the level of blending is so intense—an agent may shoot while retreating and also seeking a health item—that the state space is like a mosaic of tasks with different levels of blending. The methods of this article may well work in such a domain, but one could imagine that the hierarchical extensions described thus far, and others not yet considered, may be necessary to achieve good performance in such domains.

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. Three kinds of domains were identified—isolated, interleaved, and blended—and modular neuroevolution techniques were used to solve them. The results in three versions of Ms. Pac-Man showed that while human-specified task divisions work well in isolated and interleaved domains, they are difficult to properly specify for blended domains. Evolution discovered the best behavior in all three domains with the use of preference neurons, which makes machine discovery of multimodal behavior 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.

References

Alhejali, A. M. and Lucas, S. M. (2010). Evolving Diverse Ms. Pac-Man Playing Agents Using Genetic Programming. In *UK Workshop on Computational Intelligence*, pages 1–6.

Alhejali, A. M. and Lucas, S. M. (2011). Using a Training Camp with Genetic Programming to Evolve Ms Pac-Man Agents. In *Conference on Computational Intelligence and Games*, pages 118–125. IEEE.

Alhejali, A. M. and Lucas, S. M. (2013). Using Genetic Programming to Evolve Heuristics for a Monte Carlo Tree Search Ms Pac-Man Agent. In *Conference on Computational Intelligence and Games*, pages 65–72. IEEE.

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

Brandstetter, M. F. and Ahmadi, S. (2012). Reactive Control of Ms. Pac Man Using Information Retrieval Based on Genetic Programming. In *Conference on Computational Intelligence and Games*, pages 250–256. IEEE.

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

Burrow, P. and Lucas, S. M. (2009). Evolution versus Temporal Difference Learning for Learning to Play Ms. Pac-Man. In *Conference on Computational Intelligence and Games*, pages 53–60. IEEE.

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

Caruana, R. A. (1993). Multitask Learning: A Knowledge-based Source of Inductive Bias. In *International Conference on Machine Learning*, pages 41–48.

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

Dam, H. H., Abbass, H. A., and Lokan, C. (2008). Neural-based learning classifier systems. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):26–39.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197.

Dietterich, T. G. (1998). The MAXQ Method for Hierarchical Reinforcement Learning. In *International Conference on Machine Learning*.

Gruau, F. (1994). Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–183.

Hengst, B. (2002). Discovering Hierarchy in Reinforcement Learning with HEXQ. In *International Conference on Machine Learning*, pages 243–250.

Howard, G., Bull, L., and Lanzi, P.-L. (2010). A Spiking Neural Representation for XCSF. In *Congress on Evolutionary Computation*, pages 1–8. IEEE.

Huizinga, J., Mouret, J.-B., and Clune, J. (2014). Evolving Neural Networks That Are Both Modular and Regular: HyperNeat Plus the Connection Cost Technique. In *Genetic and Evolutionary Computation Conference*, pages 697–704. ACM.

Hurst, J. and Bull, L. (2006). A Neural Learning Classifier System with Self-Adaptive Constructivism for Mobile Robot Control. *Artificial Life*, 12(3):353–380.

Ikehata, N. and Ito, T. (2011). Monte-Carlo Tree Search in Ms. Pac-Man. In *Conference on Computational Intelligence and Games*, pages 39–46. IEEE.

Kang, Z., Grauman, K., and Sha, F. (2011). Learning with Whom to Share in Multi-task Feature Learning. In *International Conference on Machine Learning*, pages 521–528.

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

Knowles, J. D., Watson, R. A., and Corne, D. (2001). Reducing Local Optima in Single-Objective Problems by Multi-objectivization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 269–283. Springer.

Kodjabachian, J. and Meyer, J.-A. (1998). Evolution and Development of Modular Control Architectures for 1D Locomotion in Six-legged Animats. *Connection Science*, 10(3–4):211–237.

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

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

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Lessin, D., Fussell, D., and Miikkulainen, R. (2013). Open-Ended Behavioral Complexity for Evolved Virtual Creatures. In *Genetic and Evolutionary Computation Conference*, pages 335–342. ACM.

Lucas, S. M. (2005). Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. In *Conference on Computational Intelligence and Games*, pages 203–210. IEEE.

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

Pepels, T., Winands, M. H. M., and Lanctot, M. (2014). Real-Time Monte Carlo Tree Search in Ms Pac-Man. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 6, pages 245–257. IEEE.

Recio, G., Martín, E., Estébanez, C., and Sáez, Y. (2012). AntBot: Ant Colonies for Video Games. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):295–308.

Robles, D. and Lucas, S. M. (2009). A Simple Tree Search Method for Playing Ms. Pac-Man. In *Conference on Computational Intelligence and Games*, pages 249–255. IEEE.

Rohlfshagen, P. and Lucas, S. M. (2011). Ms Pac-Man versus Ghost Team Congress on Evolutionary Computation 2011 Competition. In *Congress on Evolutionary Computation*, pages 70–77. IEEE.

Rosca, J. P. (1996). Generality Versus Size in Genetic Programming. In *Conference on Genetic Programming*, pages 381–387. MIT Press.

Rosca, J. P. and Ballard, D. H. (1996). Discovery of Subroutines in Genetic Programming. In *Advances in Genetic Programming*, pages 177–202. MIT Press.

Samothrakis, S., Robles, D., and Lucas, S. M. (2011). Fast Approximate Max-n Monte Carlo Tree Search for Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):142–154.

Schrum, J. and Miikkulainen, R. (2012). Evolving Multimodal Networks for Multitask Games. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):94–111.

Schrum, J. and Miikkulainen, R. (2014). Evolving Multimodal Behavior With Modular Neural Networks in Ms. Pac-Man. In *Genetic and Evolutionary Computation Conference*, pages 325–332. ACM.

Schrum, J. and Miikkulainen, R. (2015). Solving Interleaved and Blended Sequential Decision-Making Problems through Modular Neuroevolution. In *Genetic and Evolutionary Computation Conference*, pages 345–352. ACM.

Stanley, K. O., Bryant, B. D., Karpov, I., and Miikkulainen, R. (2006). Real-Time Evolution of Neural Networks in the NERO Video Game. In *National Conference on Artificial Intelligence*, pages 1671–1674. AAAI Press.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation Journal*, pages 99–127.

Stone, P. and Veloso, M. (2000). Layered Learning. In *European Conference on Machine Learning*, pages 369–381. Springer Verlag.

Suchorzewski, M. and Clune, J. (2011). A Novel Generative Encoding for Evolving Modular, Regular and Scalable Networks. In *Genetic and Evolutionary Computation Conference*, pages 1523–1530. ACM.

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

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

Thompson, T., Milne, F., Andrew, A., and Levine, J. (2009). Improving Control Through Subsumption in the EvoTanks Domain. In *Conference on Computational Intelligence and Games*, pages 363–370. IEEE.

Thrun, S. and O'Sullivan, J. (1998). Clustering Learning Tasks and the Selective Cross-Task Transfer of Knowledge. In *Learning to Learn*, pages 235–257. Springer US.

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

van Hoorn, N., Togelius, J., and Schmidhuber, J. (2009). Hierarchical Controller Learning in a First-Person Shooter. In *Conference on Computational Intelligence and Games*, pages 294–301. IEEE.

Verbancsics, P. and Stanley, K. O. (2011). Constraining Connectivity to Encourage Modularity in HyperNEAT. In *Genetic and Evolutionary Computation Conference*, pages 1483–1490. ACM.