To Appear In IEEE Transactions on Computational Intelligence and AI in Games (2015).

# Discovering Multimodal Behavior in Ms. Pac-Man through Evolution of Modular Neural Networks

Jacob Schrum and Risto Miikkulainen

***Abstract*—Ms. Pac-Man is a challenging video game in which multiple modes of behavior are required: Ms. Pac-Man must escape ghosts when they are threats and catch them when they are edible, in addition to eating all pills in each level. Past approaches to learning behavior in Ms. Pac-Man have treated the game as a single task to be learned using monolithic policy representations. In contrast, this paper uses a framework called Modular Multiobjective NEAT (MM-NEAT) to evolve modular neural networks. Each module defines a separate behavior. The modules are used at different times according to a policy that can be human-designed (i.e. Multitask) or discovered automatically by evolution. The appropriate number of modules can be fixed or discovered using a genetic operator called Module Mutation. Several versions of Module Mutation are evaluated in this paper. Both fixed modular networks and Module Mutation networks outperform monolithic networks and Multitask networks. Interestingly, the best networks dedicate modules to critical behaviors (such as escaping when surrounded after luring ghosts near a power pill) that do not follow the customary division of the game into chasing edible and escaping threat ghosts. The results demonstrate that MM-NEAT can discover interesting and effective behavior for agents in challenging games.**

***Index Terms*—Multiobjective Optimization, Multimodal Behavior, Neuroevolution, Ms. Pac-Man, Modularity**

## I. Introduction

MS. PAC-MAN is among the most popular video games of all time. This popularity extends to AI research, as evidenced by numerous papers and two different competitions. Ms. Pac-Man is interesting because simple rules give rise to a game in which complex strategies are needed to succeed.

Ms. Pac-Man is a predator-prey scenario, with a twist. Ms. Pac-Man is usually the prey of the ghosts, but if she eats a power pill, the situation is reversed: Ghosts temporarily become her prey. The switch in game dynamics requires a switch in play strategy. In other words, multiple distinct modes of behavior are required. Despite the need for multimodal behavior, most learning approaches to the game have focused on learning monolithic policies that control Ms. Pac-Man regardless of whether ghosts are threatening or edible. Although it is possible to represent multimodal behavior with such policies, it is difficult to do so.

In contrast, this paper evolves neural networks with multiple output modules using a framework called Modular Multiobjective NEAT (MM-NEAT). Each module represents a different policy, and the agent can use one at a time. Arbitration between modules (i.e. when to use which module) can be based on a human-specified task division similar to that used in Multitask Learning [1], or discovered automatically through the use of special neurons that indicate the network's preference for using each module. The number of preference neuron modules can be fixed, or discovered using Module Mutation (also called Mode Mutation [2]).

This paper builds on earlier results showing that modular neural networks can be successfully evolved for Ms. Pac-Man [3]. The earlier research focused on preference neuron networks with a fixed number of modules, and networks evolved using one form of Module Mutation. This paper evaluates various numbers of modules, and compares with two more forms of Module Mutation [2], a combination of all three forms, and the aforementioned Multitask Learning approach. Further, while all earlier results were based on sensors that do not distinguish between threat and edible ghosts, this paper also evaluates split sensors (Section VI-B), demonstrating how task divisions can be incorporated at the level of sensors. The results achieved with modular networks are the strongest learning results in Ms. Pac-Man to date.

The main conclusion is that learning a task division with preference neurons produces networks superior to non-modular and Multitask networks. The best module division using preference neurons is unexpected: One module handles the critical behavior of escaping when surrounded, often after luring threat ghosts near a power pill, which makes them easier to eat. Therefore, MM-NEAT is a promising approach for discovering behavior for game agents automatically.

The paper progresses as follows: Related work in multimodal behavior and Ms. Pac-Man is in Section II. The Ms. Pac-Man simulator is described in Section III, and the need for multimodal behavior in Ms. Pac-Man is motivated in Section IV. Section V describes evolutionary methods for discovering such behavior. Sections VI and VII describe experiments evaluating these methods, which are discussed in Section VIII.

## II. Related Work

This section first discusses related research in multimodal behavior, and then describes previous work in Ms. Pac-Man.

### A. Multimodal Behavior Research

Domains requiring multimodal behavior are common in both video game and robotics research, so various approaches have been implemented to deal with such domains.

For complex tasks, it is common to combine controllers into a hierarchy. The components of such hierarchies can be hand-designed [4] or learned. For example, Togelius's evolved subsumption architecture [5] was used in EvoTanks [6] and

Jacob Schrum is with the Department of Mathematics and Computer Science, Southwestern University, Georgetown, TX 78626 USA (e-mail: schrum2@southwestern.edu)

Risto Miikkulainen is with the Department of Computer Science, University of Texas, Austin, TX, 78712 USA (e-mail: risto@cs.utexas.edu)

Unreal Tournament [7], and Stone's Layered Learning [8] was applied to RoboCup Soccer. Recently, Lessin et al. used the principles of Encapsulation, Syllabus, and Pandemonium to learn complex behavior for virtual creatures [9]. These approaches still require a programmer to divide the domain into constituent tasks and develop effective training scenarios for each task.

Hierarchical Reinforcement Learning (HRL) also produces hierarchical controllers consisting of multiple sub-controllers. Early HRL research required the hierarchy to be human-specified [10]. Today, ways of learning the hierarchy in addition to all sub-controllers also exist [11]. Most HRL techniques are based on the formalism of Semi-Markov Decision Processes (SMDPs), which was first used to develop partial control policies called options [12]. Similar techniques, e.g. skills [13], activities [14], modes [15], and behaviors [16], 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.

A hierarchical control policy is also a modular policy, and some approaches to learning multimodal behavior, including those in this paper, simply focus on learning modular policies. The concept of modularity used is similar to that of Calabretta et al. [17]. They evolved modular 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, and the combination of an actuator neuron and its selector unit is a module.

A similar approach is Mode Mutation [2], whose modules define complete policies rather than the behavior of individual actuators. Each new policy has 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 used by Calabretta et al. Unlike duplication, however, Mode Mutation can be performed multiple times, with no bound on the number of new modules produced. The name Mode Mutation suggests that each module encapsulates a single mode of behavior, which is not necessarily true: One module may exhibit multiple modes of behavior, and the same behavior can be represented in multiple modules. So, it is more appropriate to rename this operation Module Mutation [3]. Since Module Mutation is used in this paper, it is discussed further in Section V-C3.

The modules discussed so far have only consisted of output neurons. A benefit of such modules is that it is clear when and how each module is being used. However, according to a more common and general definition of modularity [18], [19], a module is simply 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 [20], [21], [22]. These methods evolve modular neural networks, assuming that distributing a domain across modules makes optimization easier.

Modular policies have also been explored in Genetic Programming (GP). An early example is Koza's Automatically Defined Functions [23], which encapsulate portions of a program tree that can potentially be re-used. A similar GP technique is Adaptive Representation through Learning (ARL; [24]), which culls modules from program trees based on differential parent/child fitness. Interestingly, ARL has also been applied to Pac-Man, and is the only modular/multimodal approach that has been so applied. Even though Pac-Man is composed of multiple sub-tasks, the large body of research on Pac-Man has focused on monolithic control policies. This body of research is the focus of the next section.

## B. *Pac-Man Research*

Pac-Man (1980) and its sequel Ms. Pac-Man (1981) are among the most popular video games of all time. They feature gameplay that is simple, yet requires complex strategies for success. This combination has made the game appealing to computational intelligence researchers.

Until recently, individual researchers created their own simulators. This diversity was problematic because it made fair comparisons difficult, and because in some cases the custom simulators were less challenging than the original game. For example, Koza [25] used GP to learn Pac-Man behavior in a custom simulator, whose rules were then copied by others [24], [26]. However, this variant of the game is actually much easier than the arcade version [27], [28].

Even the original Pac-Man is a poor choice for AI research. Ghost behavior is deterministic, so it is possible to maximize the score by following memorized paths, without any strategic intelligence. For this reason, current research focuses on *Ms.* Pac-Man, which is non-deterministic. Non-determinism makes evaluations noisy, which in turn makes learning hard. Another difference is that Ms. Pac-Man has four mazes in comparison to Pac-Man's one. Because of these differences, success in Ms. Pac-Man depends more on generalization than memorization.

Microsoft's Revenge of Arcade port of this game was used in the Ms. Pac-Man screen-capture competition[1] at IEEE computational intelligence conferences from 2007 to 2011. Many approaches have been evaluated in this domain. Thawonmas and others constructed a rule-based system [29], and later used Evolution Strategies to optimize its parameters [30]. Handa and Isozaki evolved fuzzy logic systems [31], while Wirth and Gallagher created an influence map model for the game [32]. Robles and Lucas [33] adapted traditional game-tree search to work in Ms. Pac-Man, and in the most recent competition, Ikehata and Ito [34] used Monte-Carlo Tree Search (MCTS) in their winning entry. The competition has not been run since 2011, but in 2012 Foderaro et al. [35] painstakingly modeled the idiosyncratic details of the ghosts' behaviors[2] and decomposed the corridors and junctions of the mazes into cells in order to learn a decision-tree-based policy that outperformed MCTS (though this success is likely due to their detailed, human-supplied ghost model).

[1]http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html

[2]Based on http://home.comcast.net/~jpittman2/pacman/pacmandossier.html

A common conclusion throughout these papers is that the quality of any learning method is greatly affected by the quality of the screen-capture procedure used to assess the current game state. In order to separate issues of computer vision from issues of machine learning, Lucas [27] developed a Ms. Pac-Man simulator that has gradually become standard for research on Ms. Pac-Man.

This simulator has changed since it was introduced. Initially, it was designed to evolve after-state evaluating neural networks [27], but it improved as it was used for other research projects, such as showing how evolved multi-layer perceptrons (MLPs) outperformed temporal difference learning using both interpolated tables and MLPs [36], and showing how game-tree search could be applied to Ms. Pac-Man [33].

The most recent version of the simulator was used in the Ms. Pac-Man vs. Ghosts competitions[3] in 2011 [37] and 2012. The primary appeal of this simulator is that it allows controllers for both Ms. Pac-Man and the ghosts to be programmed. However, it also includes a standard `Legacy` team that is an approximation of the ghost team in the original commercial game. Several approaches have been evaluated against the `Legacy` team. GP was used with complex sensors and actions [38], [39], with simple sensors and primitive actions [40], and in conjunction with MCTS [41]. MCTS was also used on its own, though the best results occurred under different evaluation conditions (given an unfair amount of evaluation time to search the tree, performed in the first maze only [42], or evaluated against ghost teams other than the `Legacy` team [43]). Ant Colony Optimization (ACO) [28] was also evaluated. Scores from most of these methods are compared with the results of this paper in Section VII-C.

Although these common platforms are useful, other platforms are still used. Bom et al. [44] used a custom simulator to train Ms. Pac-Man using Q-Learning on neural networks. Subramanian et al. [45] automatically learned options (an HRL approach mentioned in Section II-A) based on game recordings of human subjects using another simulator [46]. Though these studies are interesting, it is difficult to compare these results with those obtained using the more common Ms. Pac-Man vs. Ghosts simulator.

Since the Ms. Pac-Man vs. Ghosts simulator is the most common, it will be used as a platform to learn multimodal behavior in this paper. Details of how it works are given next.

## III. Ms. Pac-Man Simulator

In Ms. Pac-Man, each maze contains several pills and four power pills. All pills and power pills must be eaten to clear a level. Each pill earns 10 points, and each power pill earns 50 points. To reduce learning time, Ms. Pac-Man visits each maze exactly once (mazes are repeatedly visited in the original game), and evaluation ends when the fourth maze is cleared. There are 932 pills across all mazes.

In each evaluation, four hostile ghosts start 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 loses a life. However, if Ms. Pac-Man eats a power pill, then for a limited time the game dynamics are reversed, and Ms. Pac-Man can eat the ghosts. The 1st, 2nd, 3rd, and 4th ghosts eaten in sequence are worth 200, 400, 800, and 1600 points, respectively. The maximum score is achieved by eating all four ghosts after eating each power pill in each level. This goal becomes more challenging in each subsequent level, because the edible time decreases as the level increases.

The highest score that can be achieved across four levels is 58,120. Though Ms. Pac-Man normally has multiple lives, experiments in this paper only allow her to have one, both to reduce evaluation time and to encourage consistently good behavior (since dying will have a large impact on fitness).

In the original game, the speed of all agents depends on various factors. The simulator simplifies movement by having agents usually move at the same speed. However, edible ghosts move at half speed, which is necessary for Ms. Pac-Man to have a chance at catching them.

Another change is the behavior of the ghosts. The `Legacy` team approximates the ghosts in the original game using different path metrics for each ghost: The red, blue, and pink ghosts pursue Ms. Pac-Man along paths minimizing distance according to shortest path, Manhattan distance, and Euclidean distance, respectively. The orange ghost makes uniformly random movement choices. These choices are one source 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 a power pill is eaten, so that edible ghosts flee Ms. Pac-Man.

This version of Ms. Pac-Man is challenging, has proven worthwhile as a benchmark (Section II-B), and does not require screen capture. Therefore, it was used to carry out the experiments in this paper. The next section explains why multimodal behavior is required to succeed in it.

## IV. Multimodal Behavior

The research surveyed in Section II-A presents many different perspectives on how best to learn multimodal behavior. Some approaches micromanage the behavioral hierarchy to the point where individual modes can be as simple as *turn left* or *move to point A*. If the hierarchy has several layers, then these low-level behaviors are often subsumed by more interesting behavioral modes, like *retreat* and *attack*.

The perspective taken in this paper counts only relatively high-level behaviors as behavioral modes. These modes are organized in a flat hierarchy, i.e. high-level behaviors are not explicitly sub-divided into low-level behaviors. The exact threshold between low- and high-level behaviors is subjective, but it is generally the case that a label for a high-level behavioral mode will describe what a behavior accomplishes without fully divulging the details of how it is executed.

In Ms. Pac-Man specifically, multimodal behavior is needed because she must respond differently to edible and threat ghosts. At the least, she must avoid threats and eat pills to clear levels. To maximize her score, she must also pursue edible ghosts, which requires a reversal of the typical behavior.

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

Limited computational resources are part of the challenge of learning different behavioral modes. Representing a complex policy as a massive collection of perfect memorized responses to every situation would make it easy to exhibit different modes of behavior, but such a policy is hard to learn in practice for any interesting domain. Therefore, function approximation is used to generalize across similar states, including many which have never been seen during learning.

This paper uses neural networks to represent Ms. Pac-Man policies, which determine behavior as described in Section VI-A. However, all forms of function approximation are limited in terms of what they can represent with limited structure. The difficulty in producing different behavioral modes also depends on the sensor information supplied to the policies, which is an issue explored in Section VI-B.

Developing separate behavioral modes is even more difficult when there is no clear boundary between tasks. Although the ghosts are usually all threats or all edible, there are also cases when both types of 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. A learned policy must therefore not only have behaviors against threat and edible ghosts, but also for the blended situations in between. It is difficult to supply a proper hand-designed task division in such blended tasks, as shall be demonstrated in Section VII.

Although the threat/edible split seems obvious, other task divisions also have merit. Dealing with threat ghosts is actually a collection of tasks, since Ms. Pac-Man must avoid threats, collect pills, and decide when to eat power pills so that she can eat all ghosts. This last behavior, a form of luring, will prove important in the experiments below: The best performing policies dedicate a network module to escaping when surrounded, which is required in order for luring to be effective rather than suicidal. The discovery of this escape module is a surprising and powerful result. The next section describes how these modular networks are evolved.

## V. Evolutionary Methods

Evolutionary multiobjective optimization is used to evolve controllers for Ms. Pac-Man. The evolved individuals are neural networks, and modular architectures are used to encourage multimodal behavior.

### A. *Evolutionary Multiobjective Optimization*

The research community has always treated Ms. Pac-Man as a single-objective problem, where the goal is to maximize game score. Even though all that matters is the score, pill and ghost eating contribute to this score in different ways. In this paper, results are evaluated according to the highest scoring individual in each population, but populations are evolved using multiobjective optimization to maximize pill and ghost eating scores separately. Optimizing with multiple objectives improves search by helping avoid local optima [47]. A principled way of dealing with multiple objectives is provided by the concepts of Pareto dominance and optimality:

**Pareto Dominance:** Vector $\vec{v} = (v_1, \ldots, v_n)$ dominates vector $\vec{u} = (u_1, \ldots, u_n)$ iff

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

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

The above definitions indicate that one solution is better than (i.e. 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 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. Layers of Pareto fronts can be defined by iteratively removing the front and recalculating it for the remaining individuals. Solving a multiobjective optimization problem involves approximating the *first* Pareto front as well as possible. This paper accomplishes this goal using the Non-Dominated Sorting Genetic Algorithm II (NSGA-II [48]). The approximation produced by NSGA-II potentially contains multiple solutions that must be analyzed in order to determine which fulfill the needs of the user. For Ms. Pac-Man, the notion of game score determines which solution is best.

NSGA-II is indifferent as to how these solutions are represented. This paper uses the standard NSGA-II algorithm [48], which is based on $(\mu + \lambda)$ elitist selection favoring individuals in higher Pareto fronts 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. However, instead of the usual bit-string representation, neural networks are evolved.

### B. *Neuroevolution*

Neuroevolution is the simulated evolution of neural networks. All behavior in this paper is learned using the network representation of NEAT (Neuro-Evolution of Augmenting Topologies [49]), a constructive neuroevolution method that starts with simple networks that become more complex from mutations across generations. The initial population of networks has no hidden neurons, only input and output neurons.

Whenever NSGA-II creates $\lambda$ new child networks from $\mu$ parents, offspring can be modified by three mutation operators. Weight mutation perturbs the weights of existing network connections, link mutation adds new connections between existing nodes, and node mutation splices new nodes along existing connections. New links can connect any node to any other node, which allows them to be recurrent or even self-recurrent. Another key innovation of NEAT is topological crossover based on historical markers. 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 NEAT method has been used to solve many challenging problems [50], [49], but the resulting networks only define single control policies. The next section describes methods for augmenting network architectures so that they possess multiple policies, making it easier to learn multimodal behavior.

### *C. Modular Networks*

The networks in this paper can have multiple output modules. Each such module defines a different control policy. These sub-policies correspond to options in the SMDP formalism (Section II-A). Arbitration between modules can be based on a human-specified division, which is done with the Multitask Learning approach, or can be discovered using preference neurons. In preference neuron networks where the modules are fixed, evolution must discover how to use the modules. With Module Mutation, evolution must also settle on an appropriate number of modules. More specifically:

*1) Multitask Learning:* Multitask networks were first proposed by Caruana [1] in the context of supervised learning using neural networks and backpropagation. One network has multiple modules, where each module corresponds to a different, yet related, task (Fig. 1b). Each module is trained on the data for the task to which it corresponds, but because hidden-layer neurons are shared by all outputs, knowledge common to all tasks can be stored in the weights of 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 and shared, rather than learned independently multiple times.

Although Multitask Learning is a powerful technique, there are known problems with it. The first one is that the individual tasks to learn need to be identified a priori. The appropriate task division is not always obvious, and obvious divisions may actually hurt learning. 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 [51], [52].

Multitask Learning with neuroevolution has been previously applied to domains with isolated tasks [2]. In such domains, the 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 can supply a learning system with a helpful bias, but this bias will only be useful if it is appropriate. When tasks are blended, as in Ms. Pac-Man, it is hard to provide an appropriate bias (i.e. division). In order to discover better task divisions, a means of learning how to arbitrate between tasks is needed.

*2) Preference Neurons:* Preference neurons make module arbitration without human-specified task divisions possible. Each module's preference neuron outputs the network's relative preference for using that module. Whenever inputs are presented to the network, the module whose preference neuron output is the highest is used to define the output of the network.

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.

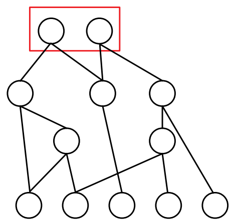
This architecture assumes that a designer specifies the number of modules. If a good guess at the number cannot be made, one option is to simply give a network lots of modules, and hope that it evolves to ignore those it does not need. However, adding extra modules needlessly increases the size of the search space, defeating some of the benefits of constructive neuroevolution. However, new modules can also be introduced gradually, using Module Mutation.

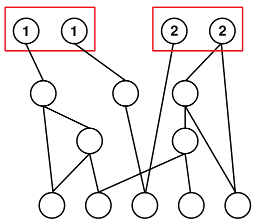
*3) Module Mutation:* Module Mutation is any structural mutation operator that adds a new output module to a neural network. An indefinite number of modules may be added in this way. Such networks depend on preference neurons for module arbitration. New populations start with a single module and a preference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

Different versions of Module Mutation were evaluated in prior research [2]. MM(P), for *previous*, creates modules with lateral inputs from a previous module, each with a connection weight of 1.0 (Fig. 1e). The new module is thus similar to the previous module, but not identical because the $\tanh$ activation function is applied at every node. In contrast, MM(R), for *random*, creates modules with random input link weights and sources (Fig. 1f). New MM(R) modules are often very different from existing modules, and explore the space of policies better, but they also have a higher chance of decreasing the fitness of a network. MM(R) was shown to be superior to MM(P) in two previous domains requiring multimodal behavior [2].

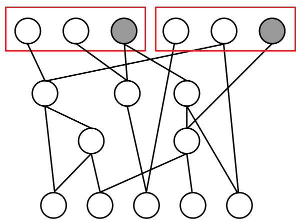
Another form of Module Mutation was introduced recently [3]: Module Mutation combined with the duplication operator (Section II-A). This operator is called MM(D), for *duplicate* (Fig. 1g), because the new module duplicates the behavior of an existing module. For every link into a policy neuron in the original module, a duplicate link into the corresponding policy neuron of the new module is created, and it has the same source neuron and link weight as the link being copied. A network that undergoes MM(D) will have the exact same behavior as the original network. MM(D) provides a network with new structure for evolution to explore without altering the network's fitness. 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.

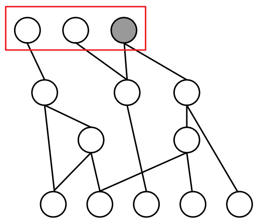
Support for all these types of modular networks, combined with NSGA-II, results in a software framework called Modular

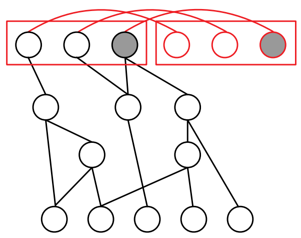

(a) Single-module Network

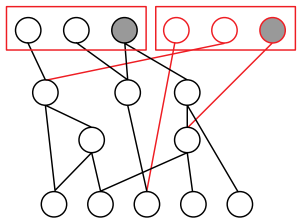


(b) Multitask Network


(c) Preference Neuron Network


(d) Before Module Mutation


(e) Module Mutation Previous


(f) Module Mutation Random

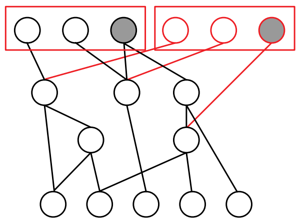

(g) Module Mutation Duplicate

Fig. 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(P), there is a new module connected to a previous module by lateral connections. (f) After MM(R), there is a new module with random source inputs and link weights. This new module will represent a random policy based on signals generated by pre-existing neurons in the network. (g) 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. Once any form of Module Mutation is performed, both the pre-existing and newly added preference neurons become relevant. Extra modules allow these networks to learn multimodal behavior more easily by making it possible to associate a different module with each behavioral mode.

Multiobjective NEAT (MM-NEAT)[4]. The core evolutionary algorithm of MM-NEAT is NSGA-II, but the evolved representations are neural networks. They can either have a single output module like networks in the original NEAT, or multiple modules as just described. Also, in addition to the standard mutation operators supported by NEAT, several forms of Module Mutation are available in MM-NEAT. MM-NEAT's ability to discover multimodal behavior is demonstrated next.

## VI. Experimental Setup

This section describes the policy representation, sensors, objectives, and specific network architectures used to evolve multimodal behavior in Ms. Pac-Man. Then the procedure for evaluating the final results is explained.

### A. *Direction-Evaluating Policy*

A learned policy can control Ms. Pac-Man in several different ways (Section II-B), regardless of the method used to represent the function approximator defining the policy.

This paper uses an approach introduced by Brandstetter and Ahmadi (BA [40]). First, direction-oriented sensors evaluate each available direction using a function approximator with a single output, then the direction with the highest output value is picked. This approach is similar to the common Reinforcement Learning (e.g. SARSA, Q-Learning [53]) approach of using Q-values (also known as state-action values) to pick the best action to use in each state. Q-Value learning using Reinforcement Learning approaches has not been attempted in this simulator, though Temporal Difference learning has been used to learn *state* values by evaluating afterstates [27], [36]. The BA approach has proven superior to these early afterstate-based approaches to Ms. Pac-Man. Because each network is evaluated once per direction, it is possible for networks with preference neurons to use a different module for each direction on the same time step. When discussing module usage in the results (Section VII), the chosen module for a time step is the one that the network used when fed inputs for the direction that Ms. Pac-Man ultimately chose to move in.

The BA approach was chosen because it uses primitive actions ($\uparrow, \downarrow, \leftarrow, \rightarrow$) and simple sensors. Other evolved Ms. Pac-Man agents [25], [24], [38], [39] use high-level actions that bias learning and impose an additional programming burden. Primitive actions assure that intelligent behavior discovered is due to evolution, rather than sophisticated high-level actions. Sensors also have an impact on how challenging it is to learn complex, multimodal policies. Therefore, two sensor configurations are evaluated: split sensors and conflict sensors.

### B. *Sensor Configurations*

Previous approaches to learning Ms. Pac-Man nearly always made a distinction between edible and threat ghosts. Any sensor dealing with threat ghosts, such as the distance to the nearest one, was accompanied by a similar sensor that only gave information about edible ghosts. This design choice

[4]Download at http://nn.cs.utexas.edu/?mm-neat

makes it clear how the sensor should be interpreted: Threat ghosts are always bad and edible ghosts are always good. In other words, sensors dealing with ghosts are split into two categories, and are therefore called split sensors. Such sensors are the norm in Ms. Pac-Man research, but they bias learning toward discovering a particular task division.

In contrast, if there was only one type of sensor for all ghosts, and sensors did not distinguish ghosts based on type, then there would be conflicting ways of interpreting these sensors. Such general sensors are called conflict sensors: They make learning harder, but remove an additional source of bias, and provide a better test for the evolution of modular networks. Methods that can learn from conflict sensors are important because it may not be obvious how to design appropriate split sensors for every domain.

This paper shows how modular networks can learn multimodal behavior even with unbiased conflict sensors that do not suggest how to break up the domain into sub-tasks. However, it also shows how split sensors using an appropriate task division can perform well even without modular networks. These approaches differ in how they handle ghosts, but they share several other sensors. These sensors can be divided into those that are direction-oriented, and those that are not. Recall that neural networks are evaluated for each direction in which Ms. Pac-Man can potentially move. The sensors that are not direction oriented will provide the same reading for each direction on any given time step. These sensors are listed in Table I. The direction-oriented sensors depend on the specific direction being evaluated, and are listed in Table II.

Most 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 threat presence and the power pill proximity [38], [39] are useful because they help optimize the timing of eating power pills.

Most directed sensors are from the BA approach [40]. The one exception is Options From Next Junction (OFNJ). OFNJ looks at the next junction in a given direction, and counts the number of subsequent junctions that can be safely reached from the first junction without reversing. The safety of a route can be determined by taking all agent distances into account and conservatively assuming ghosts will follow the shortest path to the target junction. No forward simulation is needed to calculate this value. BA has a weaker version of this sensor that merely detects whether an upcoming junction is blocked by a threat. OFNJ makes it easier to avoid ghosts. However, high scores depend on eating edible ghosts, which are only detectable by the ghost sensors.

Conflict sensors have 16 direction-oriented ghost sensors: distances to the 1st, 2nd, 3rd, and 4th closest ghosts, whether each ghost is approaching, whether a directional path to each ghost contains junctions, and whether each ghost is edible. Because ghosts are sorted by directional distances, a different sorting could apply to each direction. The sorting ignores whether each ghost is edible, but this information is provided via the additional sensor for each ghost.

In contrast, split sensors have 24 direction-oriented ghost sensors: distances to the 1st, 2nd, 3rd, and 4th closest threat ghosts, and 1st, 2nd, 3rd, and 4th closest edible ghosts, whether each threat ghost is approaching, whether each edible ghost is approaching, whether a directional path to each threat ghost contains junctions, and whether a directional path to each edible ghost contains junctions. At any given time, each ghost can only be edible or a threat, but not both. Therefore, there will not always be a 3rd closest threat ghost, or a 1st closest edible ghost, etc. For these missing ghosts, distance sensors return 1, because they are effectively infinitely distant, and the other sensors return 0, because an absent ghost is clearly not approaching, or at the end of a path with any junctions. Notice that sets of four conflict sensors correspond to groups of eight split sensors, except for edible conflict sensors; split sensors do not need this information because it is already provided in how the sensors are split.

Both sensor setups provide sufficient information to make intelligent decisions in Ms. Pac-Man, but the split sensors bias evolution towards a particular task division. In contrast, the conflict sensors must learn an appropriate task division in order to perform well in this domain.

Having explained how the evolving Ms. Pac-Man controllers sense their environment, it is now time to explain what they try to achieve and how they will be evaluated.

TABLE I
**Common 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 |

These sensors are shared by both the split and conflict sensor configurations. 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 II).

## C. Objectives and Performance

Populations are evolved with separate pill and ghost objectives. The Pill Score is simply the number of pills eaten. This count includes power pills, even though they are worth more than regular pills in terms of game score. Both types are treated the same since both need to be eaten to clear levels. Because there are 932 pills across the four mazes and four power pills per maze, this objective has a maximum of 948.

The Ghost Score is more complicated. Once a power pill is eaten, the value of the first eaten ghost is 200, and each subsequently eaten ghost value is doubled, so this objective gives higher rewards for ghosts that are worth more points. The 1st, 2nd, 3rd, and 4th ghosts are worth 1, 2, 4, and 8 points, respectively. Therefore, it is possible to earn 15 Ghost Score points per power pill, which adds up to 60 points per maze, and 240 points across all four mazes.

TABLE II
**Common 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 |

These sensors are shared by both the split and conflict sensor configurations. 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. When combined with the undirected sensors in Table I, Ms. Pac-Man can sense everything of importance except for ghosts, which are handled differently by the split and conflict sensor configurations.

Because evaluation in Ms. Pac-Man is noisy, each neural network is evaluated 10 times. Fitness scores are the average scores across evaluations. 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. This restriction discourages behaviors that allow staying alive a long time without making progress, such as moving in circles while the ghosts chase from behind. This time limit is high enough to not affect the champions by the end of evolution.

The game score is almost a weighted combination of the Pill Score and Ghost Score (different score values for regular and power pills cause a small discrepancy). Results in Section VII are given in terms of game scores. However, learning based only on game score would throw away valuable information about how these objectives interact; using both objectives along with NSGA-II allows evolution to explore different areas of the trade-off surface to find skilled, multimodal behavior.

## D. Evolving Networks

The experiments show the benefits of modular neural networks in a domain requiring multimodal behavior. Populations of networks with one module (1M), two modules (2M), and three modules (3M) are evolved. Modular networks include preference neurons to decide which module to use on each time step. If either two or three modules happens to be the ideal number of modules for this domain, then evolving to use these fixed modules should be easier than using Module Mutation (which must also discover how many modules to use). Populations of networks that start with one module, but can add more via MM(P), MM(R), or MM(D), are also evaluated. Additionally, the MM(D,P,R) approach allows all three forms of Module Mutation to be applied to networks. Finally, Multitask Learning using both two modules (MT2) and three modules (MT3) is evaluated. The different approaches are summarized in Table III.

Because Ms. Pac-Man blends the tasks of dealing with edible and threat ghosts, it is not obvious how to split the tasks across Multitask modules, so two approaches are evaluated. MT2 uses one module if any ghost is edible, and a different module otherwise. This second module must sometimes deal with threat and edible ghosts simultaneously. The MT3 approach uses an additional module for these circumstances: one module for all threats, one module for all edible, and one module for any combination of threat and edible ghosts.

TABLE III
**Evolutionary Approaches.**

| Label | Description |
|---|---|
| 1M | Networks with one module (control). |
| 2M | Two modules arbitrated by preference neurons. |
| 3M | Three modules arbitrated by preference neurons. |
| MM(D) | Networks can gain new modules via MM(D). |
| MM(P) | Networks can gain new modules via MM(P). |
| MM(R) | Networks can gain new modules via MM(R). |
| MM(D,P,R) | MM(D), MM(P), and MM(R) can all be performed. |
| MT2 | Two modules: one used if all ghosts are threats, the other if any is edible. |
| MT3 | Three modules: one used if all ghosts are threats, one if all are edible, and one if there is a mix of both types. |

All experimental approaches are summarized in this table. Each approach is evaluated using both conflict sensors and split sensors.

Since both split and conflict sensors are used, subscripts S and C are used to identify each approach. For example, $1M_S$ refers to 1M runs using split sensors, and $MM(D)_C$ refers to MM(D) results using conflict sensors.

Populations of each type are evolved 30 times for 200 generations, with a population size of $\mu = \lambda = 100$. When offspring are produced, each network link has a 5% chance of Gaussian perturbation. Additionally, each network has a 40% chance of having a new random link added between existing neurons, and a 20% chance of a new neuron being spliced along a randomly chosen link. In MM(P), MM(R), and MM(D) runs, Module Mutation has a 10% chance of being applied per offspring. In MM(D,P,R) runs, each type of Module Mutation occurs with an independent 3.3% chance. Finally, topological network crossover has a 50% chance of being applied when offspring are produced, with parents chosen via tournament selection, as normally done in NSGA-II.

Networks are initialized with a randomly weighted link from each input neuron to each policy neuron. In modular networks, each preference neuron begins with only a single incoming link from a randomly chosen input, to allow module arbitration to be swayed by mutations more easily. Evolving populations of networks under these conditions produces several evolved populations. The champion of each population, in terms of game score, is then submitted to post-evolution evaluations.

## E. Post-Evolution Evaluation

Averaging scores across 10 evaluations only mitigates some of the noise in evaluation. To get a more reliable evaluation of the final results, the champion of each run is evaluated an additional 1,000 times, post-evolution. These average scores are then compared across methods.

The distribution of these scores for each method often does not conform to a normal distribution, so non-parametric tests are used to compare methods. The Kruskal-Wallis test is used to compare all methods using a given sensor configuration (split or conflict). For each test, a $p$-value below 0.05% indicates a significant difference between at least two champions. In this case, it is appropriate to do additional post-hoc analysis using two-tailed Mann-Whitney $U$ tests. Conducting multiple comparisons in this fashion increases the chance of finding significant differences when there are none, so Bonferroni error correction is used to adjust $p$-values appropriately. Such analysis capabilities are built in to the `R` programming language, which is used to perform all statistical tests. The results of these tests, and all other outcomes from the experiments discussed in this section, are discussed next.

## VII. Results

First the results from experiments using split sensors are presented, then results from experiments using conflict sensors. Results with both types of sensors are compared against past results from the literature. Videos of behaviors are available online at http://nn.cs.utexas.edu/?ol-pm.

### A. Split Sensor Results

All nine methods using split sensors achieve close to the same level of performance (Fig. 2). By the end of 200 generations there are no significant differences between any of them, as indicated by the Kruskal-Wallis test ($H = 9.22, df = 8, N = 30, p \approx 0.32$).

Despite a lack of significant differences in performance, there are interesting differences in the behaviors of champions. Fig. 3 plots module usage vs. average game scores across 1,000 evaluations. Multitask champions are required by their human-specified task divisions to use certain modules at certain times, resulting in fairly consistent usage. Most other champions use one module 100% of the time, and ignore other modules, if they exist. However, the highest scoring champions—which are $\mathtt{2M}_\mathrm{S}$, $\mathtt{3M}_\mathrm{S}$, and $\mathtt{MM(D)}_\mathrm{S}$ networks—use one module only about 95% of the time. The remaining 5% is dedicated to an escape module that helps Ms. Pac-Man decide when to eat power pills after luring ghosts near by so as to maximize the number of ghosts that will be eaten (Fig. 4).

The division into an escape module and another module that handles everything else is unexpected. The expected module division would distinguish between situations with threat and edible ghosts, but this division does not emerge because the split sensors already divide the domain in this way at the level of sensors. As a result, networks that only use one module exhibit two distinct modes of behavior within that module. Ms. Pac-Man switches from fleeing to chasing ghosts when the edible ghost sensors take over the network.

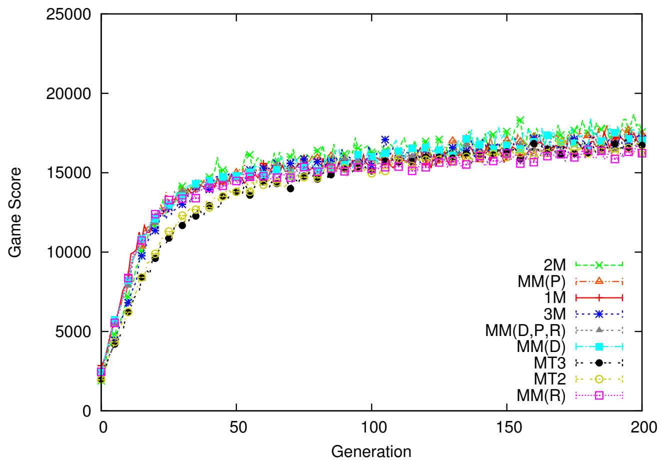


Fig. 2: **Median Champion Scores Over Evolution With Split Sensors:** Each line is the median champion score across 30 runs of evolution for a particular method, where an individual champion's score is in turn the average across 10 evaluations. Medians are compared instead of averages because the data is not normally distributed, but rather bimodal with medians close to the lower mode in each case. The order of methods in the key corresponds to the order of median scores in the final generation, but all methods are so tightly clustered that there are no significant differences between them. However, there are modular networks with effective behaviors that are difficult for $\mathtt{1M}_\mathrm{S}$ networks to discover.

When ghosts of both types are present, the threat and edible sensors compete with each other to settle on a behavior that is appropriate to the specific situation. As a result, networks that use only one module can exhibit a threat/edible split on par and sometimes better than Multitask Learning, which is explicitly programmed to treat threat and edible ghosts differently.

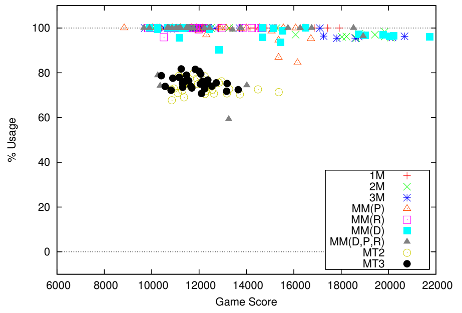
Using split sensors enabled most networks with preference neurons to settle on behaviors that used a single module and still achieve decent scores. However, the highest scores still depend on having a distinct module for escaping, because the split sensors do not divide the domain along this dimension. Therefore, these results serve as a good example of how modular networks can discover a useful task division not anticipated in how the sensors were designed.

The next set of results, using conflict sensors, show that the benefits of modular architectures become even more pronounced as sensors become more general. These sensors do not bias evolution towards the threat/edible division. Instead, evolution is forced to discover task divisions across modules.

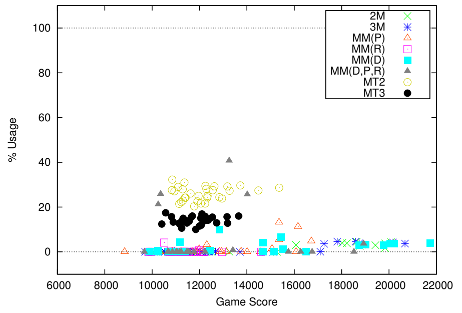
### B. Conflict Sensor Results

In contrast to split sensors, with conflict sensors, some methods perform significantly better than others. The order of performance in the final generation from best to worst is $\mathtt{2M}_\mathrm{C}$, $\mathtt{MM(P)}_\mathrm{C}$, $\mathtt{3M}_\mathrm{C}$, $\mathtt{MM(D)}_\mathrm{C}$, $\mathtt{MM(D,P,R)}_\mathrm{C}$, $\mathtt{MT3}_\mathrm{C}$, $\mathtt{MM(R)}_\mathrm{C}$, $\mathtt{MT2}_\mathrm{C}$, and finally $\mathtt{1M}_\mathrm{C}$ (Fig. 5). $\mathtt{1M}_\mathrm{C}$ is worse than all modular approaches.

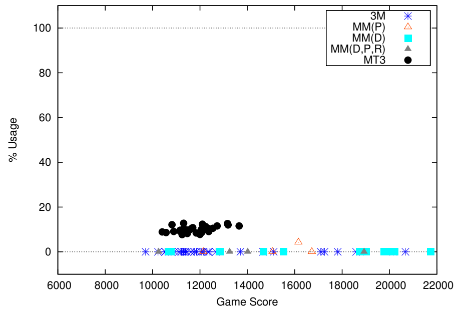
Applying the Kruskal-Wallis test to the results from post-evolution evaluations indicates that there is a significant difference between at least two of the nine methods ($H = 65.07, df = 8, N = 30, p \approx 4.7 \times 10^{-11}$). Table IV shows *adjusted* $p$-values for post-hoc Mann-Whitney $U$ tests. The results support the general conclusion that modular networks are



(a) Most Used Module

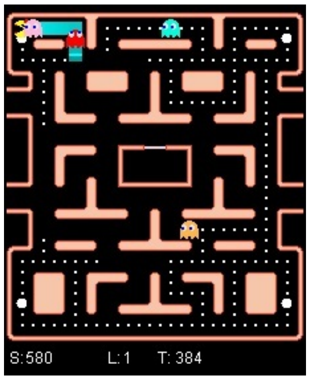


(b) Second Most Used Module

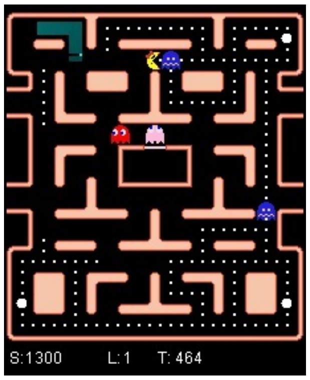


(c) Third Most Used Module

Fig. 3: **Average Post-Evolution Champion Scores vs. Module Usage With Split Sensors:** Each champion was evaluated 1,000 times, and the average scores plotted against the percentage of the time each chose a particular module. (a) The percentage of time steps the most used module is chosen to control Ms. Pac-Man is shown on the y-axis. Champions with scores below 17,000 nearly all use one module 100% of the time. However, both Multitask Learning approaches are required to use their favored module between 65% and 82% of the time, which is how often all ghosts are threats. A few MM(D,P,R)$_S$ champions have a similar usage percentage, but in general only Multitask networks use their primary module so seldom. However, most of the few champions with scores above 17,000 use their favored module more than 95% of the time, leaving just under 5% to dedicate to a useful escape behavior. (b) Usage of the second most used module is now shown on the y-axis. Champions with an escape module use it just under 5% of the time, often to signal when Ms. Pac-Man should move toward a nearby power pill after luring ghosts to the point where she is nearly surrounded (Fig. 4). If she is nearly surrounded, but no power pills are near, this module will sometimes activate to guide Ms. Pac-Man along the best escape route. This behavior is the reason that these champions receive higher scores than the many individuals that do not use their second module (points on the 0% line). The different Multitask usage patterns also result in lower scores. The second most used module for MT3$_S$ networks activates when all ghosts are edible, whereas the second most used MT2$_S$ module activates when any ghost is edible, hence the difference in usage between these two methods. (c) Usage of the third most used module is now shown on the y-axis. MT3$_S$ uses this module when there is a mixture of threat and edible ghosts. All 3M$_S$ champions and a few Module Mutation champions have a third module, but except for one MM(P)$_S$ champion, none of these networks actually use their third module. These results show that split sensors can help networks achieve good scores with just one module, but the best scores depend on two modules: one for escaping after luring, and the other for everything else.



(a) Escaping To Power Pill



(b) After Eating Power Pill

Fig. 4: **Luring Behavior With an Escape Module:** (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 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 was not used after the power pill was eaten. 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. An animation of this and other behaviors can be seen at `http://nn.cs.utexas.edu/?ol-pm`.

better than single-module networks. Additionally, preference neuron networks are generally better than Multitask networks.

As in Fig. 3, Fig. 6 plots the average scores from post-evolution evaluations against module usage percentages of each champion across 1,000 evaluations. Nearly all modular approaches use just two modules. The one major exception

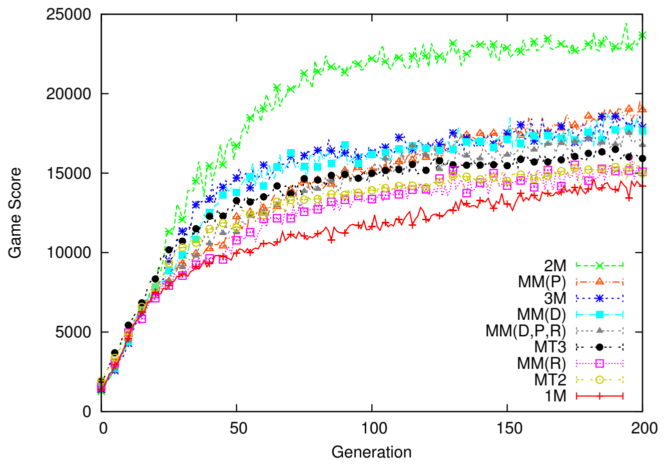


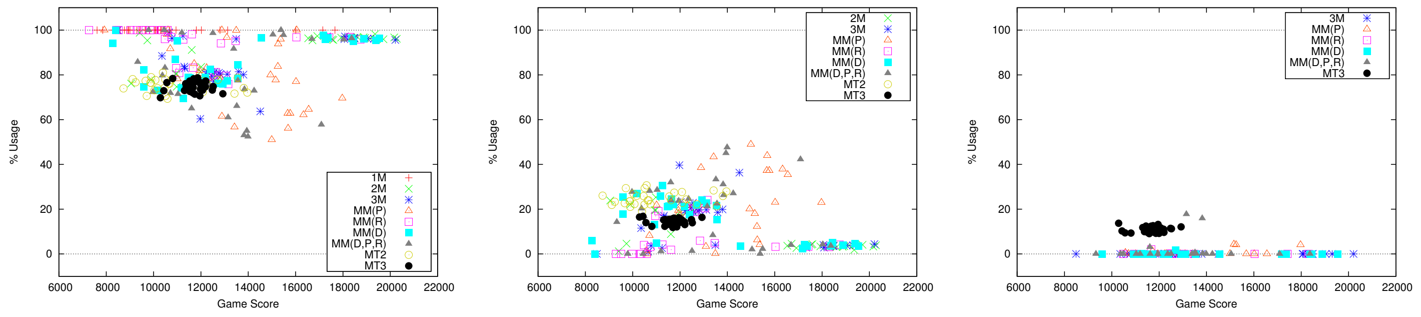
Fig. 5: **Median Champion Scores Over Evolution With Conflict Sensors:** In a plot of all methods (as in Fig. 2), 2M$_C$ greatly surpasses all the others. The remaining modular methods are all better than 1M$_C$, but by varying degrees. MM(P)$_C$, 3M$_C$, and MM(D)$_C$ are all far above 1M$_C$. MM(D,P,R)$_C$ is slightly lower. Then come MT3$_C$, MM(R)$_C$, and MT2$_C$, which are just barely above 1M$_C$. The different Module Mutation approaches are scattered between the other methods, but methods with a fixed number of modules and preference neurons generally perform well, and Multitask networks generally perform poorly.

is MT3$_C$, which is required to use three modules by design. There are also two MM(D,P,R)$_C$ champions whose third most used module is chosen 15%–17% of the time. A few more Module Mutation results have champions whose third most used module is chosen 1%–4% of the time: four MM(P)$_C$ runs, one MM(R)$_C$ run, one MM(D)$_C$ run, and one MM(D,P,R)$_C$ run. Such rare usage can still have a major impact, as has

TABLE IV
**Adjusted *p*-Values From Pairwise Mann-Whitney *U* Tests Comparing Post-Evolution Conflict Sensor Results.**

| | 1M$_C$ | MT2$_C$ | MM(R)$_C$ | MT3$_C$ | MM(D,P,R)$_C$ | MM(D)$_C$ | 3M$_C$ | MM(P)$_C$ |
|---|---|---|---|---|---|---|---|---|
| MT2$_C$ | 1.0 | - | - | - | - | - | - | - |
| MM(R)$_C$ | 1.0 | 1.0 | - | - | - | - | - | - |
| MT3$_C$ | **0.00589** | 0.58242 | 1.0 | - | - | - | - | - |
| MM(D,P,R)$_C$ | **0.00059** | **0.01355** | 0.60734 | 0.98573 | - | - | - | - |
| MM(D)$_C$ | **0.00394** | 0.07998 | 1.0 | 1.0 | 1.0 | - | - | - |
| 3M$_C$ | **0.000036** | **0.00051** | 0.07192 | **0.01629** | 1.0 | 1.0 | - | - |
| MM(P)$_C$ | **0.000088** | **0.000052** | 0.05196 | **0.00117** | 1.0 | 1.0 | 1.0 | - |
| 2M$_C$ | **0.000027** | **0.00094** | **0.02478** | **0.03931** | 1.0 | 1.0 | 1.0 | 1.0 |

The champion of each run of each method was evaluated further in 1,000 trials after evolution. Each number is a $p$-value resulting from a two-tailed Mann-Whitney $U$ test comparing two neuroevolution methods, that has been 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. 5, which results in most significant differences clustering near the lower-left of the table. Nearly all modular approaches are better than 1M$_C$. The only two preference neuron approaches that are significantly different from each other are 2M$_C$ and MM(R)$_C$. Each approach with preference neurons is also significantly better than at least one of the two Multitask Learning approaches, with the exceptions of MM(D)$_C$ and MM(R)$_C$. These results provide a more in-depth 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 typical single-module networks.



(a) Most Used Module (b) Second Most Used Module (c) Third Most Used Module

Fig. 6: **Average Post-Evolution Champion Scores vs. Module Usage With Conflict Sensors (plotted as in Fig. 3):** (a) All 1M$_C$ networks use their single module 100% of the time, and most score below 13,200. Both Multitask approaches are confined to using their primary module (for dealing with all threat ghosts) between 68% and 83% of the time, and always score less than 14,000. Behavior of networks with preference neurons is more varied: Many MM(R)$_C$ champions use only one module, and have scores similar to poor 1M$_C$ runs. Preference neuron champions near the Multitask cluster mostly split their modules into a threat/edible division as well, but their division is seldom as strict as the Multitask divisions. However, many MM(P)$_C$ and some MM(D,P,R)$_C$ runs have slightly higher scores (14,000–17,000) and use their most used module between 50% and 85% of the time, but modules in this cluster are used in erratic and confusing ways, and therefore the function of each module is not clear. However, the best modular approaches (scoring 16,000–20,500) use one module over 95% of the time because the other module, used for escaping, is only needed a small percentage of the time. (b) Most champions only use two modules, resulting in a plot that is nearly a vertical mirror of the previous plot. For a threat/edible split, the edible module is used second most. Networks with an escape module use it second most. Points that do not mirror the previous plot are for champions that use more than two modules, i.e. all MT3$_C$ and a few Module Mutation champions. The second most used module for MT3$_C$ networks activates when all ghosts are edible, and is thus chosen 10% to 18% of the time. (c) Most champions with a third module use it less than 0.7% of the time. However, MT3$_C$ champions use their module, for a mixture of threat and edible ghosts, 7% to 15% of the time. Module Mutation runs that use a third module more than a negligible amount are uncommon. These figures show that even though different module usage patterns are possible in Ms. Pac-Man, high scores can be achieved with just two modules. In particular, the best scorers primarily use just one module, however, the few time steps when they do use the other module end up being very important to the overall success of the agent.

already been demonstrated by the split sensor results that use an escape module. All other champions with a third module use it less than 0.7% of the time, which is too low to noticeably affect behavior. A few rare Module Mutation results have and use four or five modules to no noticeable effect (except for one exception discussed below).

Most champions fit into clusters corresponding to module usage patterns, which can be analyzed by observing agent behavior. As with split sensor results, the highest scoring champions are those that favor one module over 95% of the time and use the other module for escaping, often after luring ghosts near a power pill. In these cases, it is surprising that a single module can handle both threat and edible tasks, since the conflict sensors do not distinguish between threat and edible ghosts directly.

In fact, the highest scoring MM(P)$_C$ champion has five modules, and makes meaningful use of four of them: one module for escaping, one for eating edible ghosts, and two for avoiding threats. However, there are several 2M$_C$, 3M$_C$, MM(D)$_C$, and even MM(R)$_C$ champions that achieve higher scores by exhibiting luring behavior using only two modules (as in Fig. 4). Among the different Module Mutation approaches, MM(D)$_C$ produces the most champions that lure with an escape module, though its median champion score across 30 runs is between MM(D,P,R)$_C$ and MM(P)$_C$.

A major difference from results with split sensors is that networks with conflict sensors often use preference neurons to establish a threat/edible module division. Because conflict sensors do not distinguish between threat and edible ghosts directly, it is easier for modular networks to dedicate a separate module to handling each situation. This division is common across all modular networks, and leads to scores comparable to those achieved by Multitask Learning approaches. However, the division is seldom as stark as with Multitask networks. When ghosts of both types are present, these networks generally use the edible module until threat ghosts block the path to any remaining edible ghosts, which is a sensible strategy.

There is another common way of using two modules that is harder to interpret. It is exhibited by a few $\mathtt{MM(D,P,R)}_{\mathtt{C}}$ champions and many $\mathtt{MM(P)}_{\mathtt{C}}$ champions. There is usually one module that is mostly used when ghosts are edible, and another that is mostly used when ghosts are threats, but the modules are not as clearly divided between tasks as those learned by other methods. Ms. Pac-Man seems to arbitrarily switch between these modules at inappropriate times. Similar module switching was observed in previous research using MM(P) [2], and seems to result from the manner in which new modules are connected to previous modules. Because each new module is directly connected to a previous module, it is inclined to maintain similar behavior and usage, and it is therefore more difficult to specialize each module's behavior. Despite resulting in modules whose functional role is difficult to interpret, $\mathtt{MM(P)}_{\mathtt{C}}$ champions using this confused module division generally score higher than networks using a pure threat/edible split. Because this division was discovered consistently, $\mathtt{MM(P)}_{\mathtt{C}}$ has the highest median performance among the Module Mutation approaches. However, $\mathtt{MM(P)}_{\mathtt{C}}$ is the least likely modular approach to discover luring behavior via an escape module, which is responsible for the highest scores presented in this paper.

The worst individuals are $\mathtt{1M}_{\mathtt{C}}$ networks that do not learn what to do when ghosts are edible: Ms. Pac-Man jitters helplessly, and tends to only eat them if they wander directly into her path. The worst modular networks also exhibit this behavior, but these networks mostly use only one of their available modules. The $\mathtt{MM(R)}_{\mathtt{C}}$ method in particular produces champions that only use one module, likely because the completely random modules introduced by MM(R) are often quickly discarded by evolution. Learning to use multiple modules makes learning multimodal behavior more likely.

However, there are also three high performing $\mathtt{1M}_{\mathtt{C}}$ outliers. The best of these lures ghosts near power pills before successfully escaping. With only one module, improvements in one behavior often come at the expense of other behaviors, and in this case, the network's ability to chase ghosts suffers (as with all behaviors described, this one can be seen at `http://nn.cs.utexas.edu/?ol-pm`). This is why is is difficult for single-module networks to evolve effective multimodal behavior. In contrast, modular networks separate these behaviors into different modules, and are thereby able to achieve higher scores.

However it emerges, multimodal behavior leads to high scores in Ms. Pac-Man. This conclusion holds true in the full game as well, as shall be shown in the next section.

### C. Comparison Results

To compare performance with the literature, slight changes must be made in the game setup. In particular, scores discussed so far have been achieved using only one life, whereas results in the literature generally let Ms. Pac-Man start with three lives, and earn a fourth after achieving 10,000 points, as in the original game. This simple change leads to the `FourMaze` variant, so-called because evaluation remains restricted to a single visit to each maze.

However, much of the literature describes entrants in the Ms. Pac-Man vs. Ghosts competitions (`MPMvsG` variant), and these scores were achieved under the following additional rules: (1) clearing the fourth maze leads back to the first maze, until each maze is visited four times (16 levels); (2) the per-level time limit is 3,000 time steps, but running out of time advances Ms. Pac-Man to the next level instead of killing her; and (3) Ms. Pac-Man is awarded half the score from remaining pills in the level when time runs out.

Furthermore, evaluations in both variants are timed, meaning Ms. Pac-Man only has 40ms to decide on each action. This time limit is seldom a problem for the evolved networks, but whenever an action is not returned in time, the action made on the previous time step is repeated.

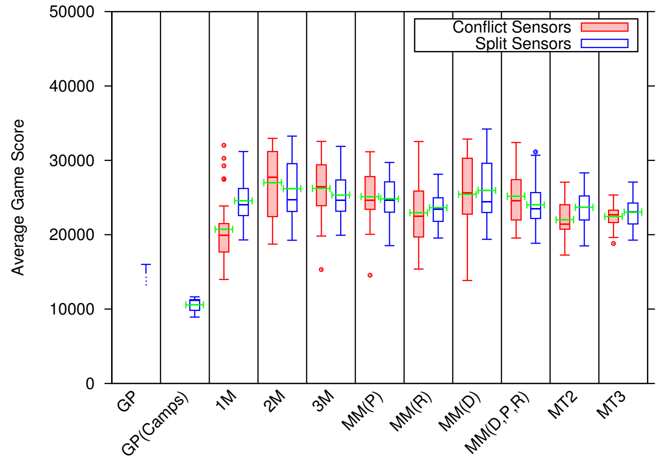
To compare scores achieved by modular networks with those in the literature, champions from each run were evaluated an additional 100 times in both the `FourMaze` (Fig. 7) and `MPMvsG` (Fig. 8) variants. Modular networks with preference neurons produce several champions with average scores far exceeding those achieved in previous work in both `FourMaze` and `MPMvsG`. Average scores of Multitask Learning champions are on par with previous work in `MPMvsG`, but are superior in `FourMaze`. When comparing maximum scores instead of averages, the best GP result in `FourMaze` (44,560) is slightly less than the best $\mathtt{MM(D)}_{\mathtt{S}}$ result (44,920), and comparable to the best results of other modular networks. The best maximum scores in `MPMvsG` evaluations are much higher than maximum scores from the literature. Even the best $\mathtt{1M}_{\mathtt{C}}$ results are superior to most previously published results, although typical $\mathtt{1M}_{\mathtt{C}}$ performance is lower.

The results demonstrate the success of MM-NEAT in Ms. Pac-Man. There are many promising directions for future research, as will be described next.

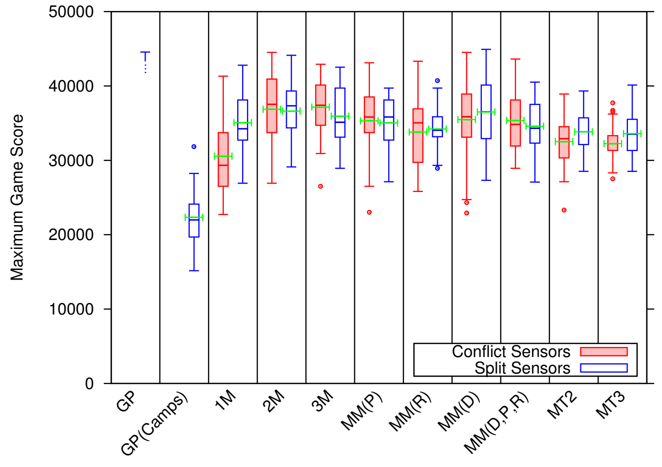
## VIII. Discussion and Future Work

The discovery of luring behavior via an escape module is an interesting and surprising result. In fact, all networks (whether with split or conflict sensors) that do not have a module for escaping are more likely to eat power pills at inopportune times because effective luring is too risky. Unfortunately, the edible time is so short that even if Ms. Pac-Man immediately switches to a module that pursues the ghosts, she has a hard time catching them before time runs out. Thus, while variants of threat/edible divisions are an intelligent way of splitting up the domain across modules, luring behavior still leads to the highest scores.

However, luring champions still need to have distinct behaviors for fleeing threats and chasing edible ghosts, and

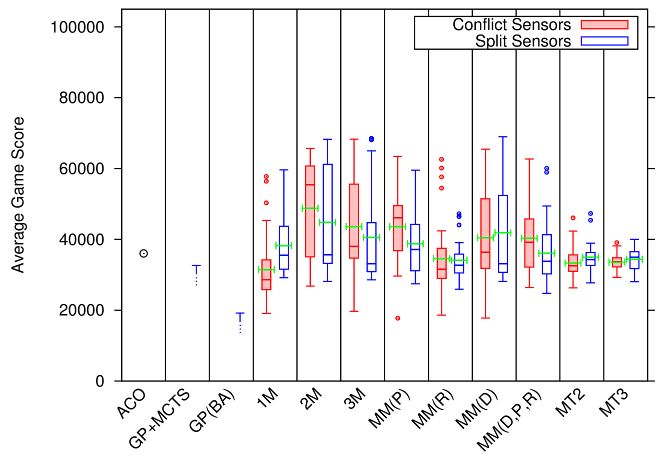


(a) Average FourMaze Scores

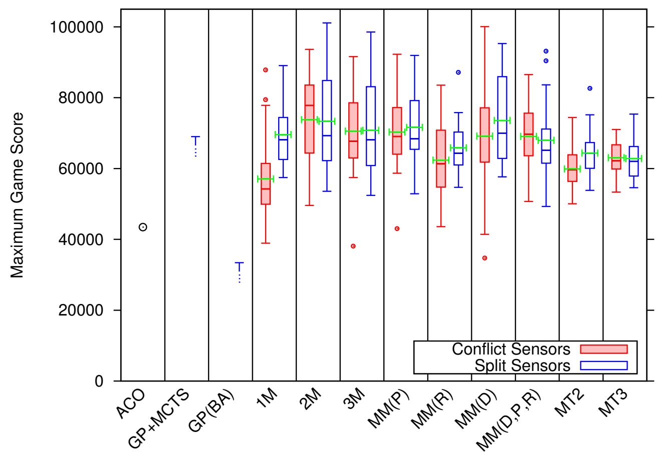


(b) Maximum FourMaze Scores

Fig. 7: **Comparison To Previous FourMaze Scores in the Literature:** The (a) average and (b) maximum scores of 30 champions from each method are compared with scores from the literature in FourMaze evaluations. Previous results include plain GP [38] and GP plus Training Camps [39]. For each column, the left box depicts performance with conflict sensors, and the right box shows performance with split sensors. These box-and-whisker plots show the lowest non-outlier, first quartile, median, third quartile, and highest non-outlier scores, with outliers defined as being over $1.5IQR$ (inter-quartile range) distance from the nearest quartile. The average across all champions is also shown as a green line intersecting each box. For each method, the majority of average champion scores are higher than those of previous methods. Notice that GP has only one result, i.e. one from the best run in that study. The trailing away of the whisker represents uncertainty about the distribution of the other scores. Interestingly, even 1M$_{C}$ outperforms previous work. There are many potential reasons for this result: evolution is driven by multiple objectives, the evaluation scheme is more demanding (only one life), direction-evaluating policies are used (the BA approach), and the sensors are different. With split sensors, even the worst champion of each run has a higher average score than those obtained in previous work. However, the best overall champions are modular networks that discovered an escape module, demonstrating the benefits of modular approaches. In terms of maximum champion scores, the best results of this paper are better than GP(Camps) and comparable to plain GP. These results show the power of modular neuroevolution in Ms. Pac-Man.



(a) Average MPMvsG Scores



(b) Maximum MPMvsG Scores

Fig. 8: **Comparison To Previous MPMvsG Scores in the Literature:** The (a) average and (b) maximum scores of 30 champions from each method are compared with scores from the literature in MPMvsG evaluations, and depicted as in Fig. 7. Previous results include GP used to learn a direction-evaluating policy (the BA method [40]), GP used to learn a default policy for MCTS [41], and ACO [28]. Scores with MPMvsG rules are much higher than with FourMaze rules because the evaluation scheme is more lenient, and Ms. Pac-Man can visit more levels. The best scores from this paper are again better than all previously published results. In terms of average champion scores, the best previous approach is ACO (which only produces a single result rather than a distribution), but all methods from this paper produce champions with higher average scores. In terms of maximum champion scores, the best previous approach is GP+MCTS, but once again the methods from this paper perform better. 1M$_{C}$ and 1M$_{S}$ also perform better than methods in previous work, but the best approaches are again modular networks with preference neurons. The neuroevolution approach of this paper is thus very strong, and is enhanced further by evolving modular architectures.

for networks using conflict sensors it is surprising that one module can handle both of these modes of behavior. The OFNJ sensor likely helps these behavioral modes coexist in the same module. If the urge to pursue ghosts is slightly overwhelmed by the pressure to go towards safety, then Ms. Pac-Man will head towards ghosts when they are edible, and run away from them when they are threats because the influence of OFNJ is stronger. However, escaping when surrounded is a different behavior from all of these, which is why it tends to need a dedicated module in order to execute correctly.

Networks evolved with split sensors perform well even with only one module, but the highest scoring results are still modular networks that have a separate escape module. Split sensor results would be better if an escape module were discovered reliably. In contrast, when conflict sensors are used, discovering multiple modules becomes more important: $\texttt{1M}_\texttt{C}$ networks have trouble learning multiple modes of behavior because they only have one module, which is why most modular approaches are significantly better. As with split sensors, conflict sensor champions would perform even better if the escape module were discovered reliably. Therefore, one way to improve the results is to develop methods that encourage the use of multiple modules more generally.

Multiobjective evolution should fill this role by encouraging exploration of trade-offs, but vanilla multiobjective evolution does not consistently capitalize on the potential of extra modules. One extension of multiobjective evolution that could do so is Targeting Unachieved Goals [50]. This fitness-based shaping technique turns off objectives in which the population is performing well so that evolution can focus on the objectives that need it most. Such a change in focus encourages modular networks to use additional modules better [54].

Another shaping technique that is easily combined with multiobjective evolution is Behavioral Diversity [55]: An extra objective rewards individuals that exhibit behaviors substantially different from the norm of the population, and thus may encourage use of additional modules when the majority of the population is ignoring them.

Additional objectives could also be used to reward multiple modules directly, but care must be taken in how such objectives are defined. For example, simply encouraging equal usage of all available modules would be detrimental in Ms. Pac-Man, because the high-scoring networks that have an escape module use it only a small portion of the time. Even the threat/edible division does not result in an even split, because ghosts are more often threats than edible. An alternative approach would be to reward diverse module usage, effectively applying Behavioral Diversity on module usage. This approach seems more promising, and is an interesting direction for future work.

Although the Multitask Learning approaches in this paper performed poorly, Multitask networks would presumably be successful if they were explicitly programmed to use an escape/edible/threat task division. In particular, the availability of a good escape module makes effective luring possible. Though such a task division was not considered in this paper, the rule-based agent that won the CEC 2009 Ms. Pac-Man screen-capture competition actually had explicit rules for encouraging luring [29]. Of course, it is difficult to optimize parameters to coordinate luring with other behaviors, which is why a later version of this agent used Evolution Strategies to do it [30].

Therefore, even if a Multitask division with escape and/or luring modules had been implemented, it would have been difficult to design rules about when to use such modules. In contrast, preference neurons are able to discover the escape module on their own, and also discover when to use it. The extra flexibility of preference neurons make them generally a better choice than Multitask Learning.

A different set of split sensors could also be designed to explicitly encode a distinction between escaping and not escaping, but once again, this is not an obvious a priori decision. Using split sensors to encode the obvious threat/edible division proved useful, but escaping was still a mode of behavior that required its own module, discovered by evolution. If the sensors could be configured by evolution, then perhaps sensors that encourage escaping behavior could be discovered without the need for multiple output modules. How to accomplish such discovery is yet another open question.

## IX. Conclusion

Ms. Pac-Man is a challenging game requiring multimodal behavior to succeed. Modular approaches are superior because they can dedicate separate modules to different modes of behavior. Some evolve to handle threat and edible ghosts with separate modules, which is a sensible division. This division can also be encoded with Multitask networks or split sensors, leading to similar performance. However, the best networks evolve an even better, unexpected division that focuses one module on the behavior of escaping ghosts after luring them near power pills, so that they can be easily eaten. This task division results in the best behavior, regardless of whether split or conflict sensors are used. Evolution of modular networks using MM-NEAT should be useful in other domains requiring multimodal behavior, especially if ways are found to evolve the best task divisions more reliably. Encouraging the evolution of these divisions via shaping is an interesting direction for future work.

## Acknowledgment

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

## References

[1] R. A. Caruana, "Multitask Learning: A Knowledge-based Source of Inductive Bias," in *ICML*, 1993, pp. 41–48.
[2] J. Schrum and R. Miikkulainen, "Evolving Multimodal Networks for Multitask Games," *TCIAIG*, vol. 4, no. 2, pp. 94–111, 2012.
[3] ——, "Evolving Multimodal Behavior With Modular Neural Networks in Ms. Pac-Man," in *GECCO*. ACM, 2014, pp. 325–332.
[4] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," *Robotics and Automation*, vol. 2, no. 10, 1986.
[5] J. Togelius, "Evolution of a Subsumption Architecture Neurocontroller," *Intelligent and Fuzzy Systems*, pp. 15–20, 2004.
[6] T. Thompson, F. Milne, A. Andrew, and J. Levine, "Improving Control Through Subsumption in the EvoTanks Domain," in *CIG*. IEEE, 2009, pp. 363–370.
[7] N. van Hoorn, J. Togelius, and J. Schmidhuber, "Hierarchical Controller Learning in a First-Person Shooter," in *CIG*. IEEE, 2009, pp. 294–301.
[8] P. Stone and M. Veloso, "Layered Learning," in *ECML*. Springer Verlag, 2000, pp. 369–381.
[9] D. Lessin, D. Fussell, and R. Miikkulainen, "Open-Ended Behavioral Complexity for Evolved Virtual Creatures," in *GECCO*. ACM, 2013, pp. 335–342.
[10] T. G. Dietterich, "The MAXQ Method for Hierarchical Reinforcement Learning," in *ICML*, 1998.
[11] B. Hengst, "Discovering Hierarchy in Reinforcement Learning with HEXQ," in *ICML*, 2002, pp. 243–250.

[12] R. S. Sutton, D. Precup, and S. P. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

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

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

[15] R. Alur, A. K. Das, J. M. Esposito, R. B. Fierro, G. Z. Grudic, Y. Hur, V. Kumar, I. Lee, J. P. Lee, J. P. Ostrowski, G. J. Pappas, B. Southall, J. R. Spletzer, and C. J. Taylor, "A Framework and Architecture for Multirobot Coordination," in *ISER*. Springer, 2000, pp. 303–312.

[16] M. Huber and R. A. Grupen, "A Feedback Control Structure for On-line Learning Tasks," *Robotics and Autonomous Systems*, pp. 22–23, 1997.

[17] R. Calabretta, S. Nolfi, D. Parisi, and G. Wagner, "Duplication of Modules Facilitates the Evolution of Functional Specialization," *ALife*, vol. 6, no. 1, pp. 69–84, 2000.

[18] J. Clune, J.-B. Mouret, and H. Lipson, "The Evolutionary Origins of Modularity," *Royal Society B*, pp. 20 122 863–20 122 863, 2013.

[19] N. Kashtan and U. Alon, "Spontaneous Evolution of Modularity and Network Motifs," *Natl. Acad. Sci. USA*, pp. 13 773–13 778, 2005.

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

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

[22] J. Huizinga, J.-B. Mouret, and J. Clune, "Evolving Neural Networks That Are Both Modular and Regular: HyperNeat Plus the Connection Cost Technique," in *GECCO*. ACM, 2014.

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

[24] J. P. Rosca, "Generality Versus Size in Genetic Programming," in *GP*. MIT Press, 1996, pp. 381–387.

[25] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[26] I. Szita and A. Lőrincz, "Learning to Play Using Low-Complexity Rule-Based Policies: Illustrations through Ms. Pac-Man," *JAIR*, pp. 659–684, 2007.

[27] S. M. Lucas, "Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man," in *CIG*. IEEE, 2005, pp. 203–210.

[28] G. Recio, E. Martín, C. Estébanez, and Y. Sáez, "AntBot: Ant Colonies for Video Games," *TCIAIG*, vol. 4, no. 4, pp. 295–308, 2012.

[29] R. Thawonmas and H. Matsumoto, "Automatic Controller of Ms. Pac-Man and its Performance: Winner of the IEEE CEC 2009 Software Agent Ms. Pac-Man Competition," in *JSST 2009*, Oct. 2009.

[30] R. Thawonmas and T. Ashida, "Evolution Strategy for Optimizing Parameters in Ms Pac-Man Controller ICE Pambush 3," in *CIG*. IEEE, 2010, pp. 235–240.

[31] H. Handa and M. Isozaki, "Evolutionary Fuzzy Systems for Generating Better Ms. PacMan Players," in *FUZZ-IEEE*, 2008, pp. 2182–2185.

[32] N. Wirth and M. Gallagher, "An Influence Map Model for Playing Ms. Pac-Man," in *CIG*. IEEE, 2008, pp. 228–233.

[33] D. Robles and S. M. Lucas, "A Simple Tree Search Method for Playing Ms. Pac-Man," in *CIG*. IEEE, 2009, pp. 249–255.

[34] N. Ikehata and T. Ito, "Monte-Carlo Tree Search in Ms. Pac-Man," in *CIG*. IEEE, 2011, pp. 39–46.

[35] G. Foderaro, A. Swingler, and S. Ferrari, "A Model-based Cell Decomposition Approach to On-line Pursuit-evasion Path Planning and the Video Game Ms. Pac-Man," in *CIG*. IEEE, 2012, pp. 281–287.

[36] P. Burrow and S. M. Lucas, "Evolution versus Temporal Difference Learning for Learning to Play Ms. Pac-Man," in *CIG*. IEEE, 2009, pp. 53–60.

[37] P. Rohlfshagen and S. M. Lucas, "Ms Pac-Man versus Ghost Team CEC 2011 Competition," in *CEC*, 2011, pp. 70–77.

[38] A. M. Alhejali and S. M. Lucas, "Evolving Diverse Ms. Pac-Man Playing Agents Using Genetic Programming," in *UKCI*, 2010, pp. 1–6.

[39] ——, "Using a Training Camp with Genetic Programming to Evolve Ms Pac-Man Agents," in *CIG*. IEEE, 2011, pp. 118–125.

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

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

[42] S. Samothrakis, D. Robles, and S. M. Lucas, "Fast Approximate Max-n Monte Carlo Tree Search for Ms Pac-Man," *TCIAIG*, vol. 3, no. 2, pp. 142–154, 2011.

[43] T. Pepels, M. H. M. Winands, and M. Lanctot, "Real-Time Monte Carlo Tree Search in Ms Pac-Man," in *TCIAIG*, vol. 6, no. 3. IEEE, 2014, pp. 245–257.

[44] L. Bom, R. Henken, and M. Wiering, "Reinforcement Learning to Train Ms. Pac-Man Using Higher-order Action-relative Inputs," in *ADPRL*, 2013.

[45] K. Subramanian, C. Isbell, and A. Thomaz, "Learning Options through Human Interaction," in *ALIHT*, 2011.

[46] J. DeNero and D. Klein, "Teaching Introductory Artificial Intelligence with Pac-Man," in *EAAI*, 2010.

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

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

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

[50] J. Schrum and R. Miikkulainen, "Evolving Agent Behavior In Multiobjective Domains Using Fitness-Based Shaping," in *GECCO*. ACM, 2010, pp. 439–446.

[51] S. Thrun and J. O'Sullivan, "Clustering Learning Tasks and the Selective Cross-Task Transfer of Knowledge," in *Learning to Learn*. Springer US, 1998, pp. 235–257.

[52] Z. Kang, K. Grauman, and F. Sha, "Learning with Whom to Share in Multi-task Feature Learning," in *ICML*, 2011, pp. 521–528.

[53] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998. [Online]. Available: http://www.cs.ualberta.ca/~sutton/book/the-book.html

[54] J. Schrum, "Evolving Multimodal Behavior Through Modular Multiobjective Neuroevolution," Ph.D. dissertation, University of Texas at Austin, 2014.

[55] J.-B. Mouret and S. Doncieux, "Using Behavioral Exploration Objectives to Solve Deceptive Problems in Neuro-evolution," in *GECCO*. ACM, 2009, pp. 627–634.

**Jacob Schrum** is an Assistant Professor of Computer Science at Southwestern University in Georgetown, Texas. He received his B.S. degree in 2006 from Southwestern University, where he triple-majored in Computer Science, Math and German, graduating with honors in both Computer Science and German. He received his M.S. degree in Computer Science from the University of Texas in 2009, and his Ph.D. in 2014 for his dissertation, "Evolving Multimodal Behavior Through Modular Multiobjective Neuroevolution." His research focuses on automatically learning intelligent agent behavior in complex domains such as video games, particularly using neuroevolution.

**Risto Miikkulainen** is a Professor of Computer Sciences at the University of Texas at Austin. He received an M.S. in Engineering from the Helsinki University of Technology, Finland, in 1986, and a Ph.D. in Computer Science from UCLA in 1990. His current research focuses on methods and applications of neuroevolution, as well as models of natural language processing, and self-organization of the visual cortex; he is an author of over 300 articles in these research areas. He is currently on the Board of Governors of the Neural Network Society, and an action editor of IEEE Transactions on Autonomous Mental Development, IEEE Transactions on Computational Intelligence and AI in Games, the Machine Learning Journal, Journal of Cognitive Systems Research, and Neural Networks.