In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014), pp. 325--332, Vancouver, BC, Canada, July 2014. Best Paper: Digital Entertainment and Arts.

Evolving Multimodal Behavior With Modular Neural Networks in Ms. Pac-Man

Jacob Schrum
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
schrum2@cs.utexas.edu

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

ABSTRACT

Ms. Pac-Man is a challenging video game in which multiple modes of behavior are required to succeed: 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 to evolve modular neural networks. Each module defines a separate policy; evolution discovers these policies and when to use them. The number of modules can be fixed or learned using a new version of a genetic operator, called Module Mutation, which duplicates an existing module that can then evolve to take on a distinct behavioral identity. Both the fixed modular networks and Module Mutation networks outperform traditional monolithic networks. More interestingly, the best modular networks dedicate modules to critical behaviors that do not follow the customary division of the game into chasing edible and escaping threatening ghosts.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation

Keywords

Games, Neural networks, Multi-objective optimization

1. 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.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2576768.2598234.

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, multimodal behavior is required to succeed. Despite the game's multimodal nature, most learning approaches to the game have focused on learning monolithic policies that control Ms. Pac-Man regardless of whether predator or prey ghosts are present.

In contrast, this paper evolves neural networks with multiple output modules using a new framework called Modular Multiobjective NEAT. Each module represents a different policy the agent can use, and special preference neurons are used to arbitrate between available modules. The number of modules can be fixed, or evolved using Module Mutation (also called Mode Mutation [21]). A new version of this structural mutation operator—called Module Mutation Duplicate because new modules copy the behavior of a previous module—is developed and evaluated in this paper.

The results show that modular networks are more likely to learn multimodal behaviors suited to this challenging game. Moreover, the best module division is unexpected: Modules are developed to handle critical behaviors, such as luring threat ghosts near a power pill in advance of eating them, and choosing how to escape when nearly surrounded.

The paper progresses as follows: Related work in multimodal behavior and Ms. Pac-Man is discussed in Section 2. The Ms. Pac-Man simulator is described in Section 3, and the need for multimodal behavior in Ms. Pac-Man is motivated in Section 4. Section 5 describes the evolutionary methods used to discover this behavior. Sections 6 and 7 describe experiments evaluating these methods, which are discussed in Section 8 before concluding in Section 9.

2. RELATED WORK

This section first discusses related research into multimodal behavior and modular neural network architectures, and then continues with previous work in Ms. Pac-Man.

2.1 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. This section presents approaches that involve separately learned controllers, and single controllers with modular architectures.

2.1.1 Separately Learned Controllers

For complex tasks, it is common to combine controllers into a hierarchy. Such hierarchical controllers can be hand-

designed [5], or learned: Togelius's evolved subsumption architecture [25] has been used in EvoTanks [24] and Unreal Tournament [26], and Stone's Layered Learning [23] was applied to simulated RoboCup Soccer. These approaches still require a programmer to divide the domain into constituent tasks and develop effective training scenarios for each task.

These hierarchical approaches are appealing because each component has a clear purpose. However, properly dividing a domain into subtasks requires significant human expertise. Therefore, instead of manually combining learned components, methods have been developed in which the learned controllers have built-in capacity to split up into separate modules, as will be discussed next.

2.1.2 Modular Architectures

Modular approaches automatically associate components of the architecture with specific functionality. For instance, Calabretta et al. [7] evolved neural networks to control robots using a duplication operator, which copies one output neuron with all of its connections and weights. The network then has two outputs for the same actuator, and needs to arbitrate between them. This is accomplished via selector units: The output neuron with the highest corresponding selector unit activation is chosen. Duplication can only be performed once per output.

A similar approach is Mode Mutation [21], which introduces complete modules rather than individual neurons; a single Mode Mutation adds enough output neurons to define a new policy, plus an additional neuron to arbitrate between modules. The behavior-defining neurons are called policy neurons, and the one arbitration neuron per module is called a preference neuron. Preference neurons are similar to the selector units used by Calabretta et al. Unlike the duplication operator, 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. So, it is more appropriate to rename this operation Module Mutation. Previous versions of Module Mutation initialized new modules either with random connections, or links to a previous module. This paper introduces Module Mutation Duplicate (MM(D)), which creates new modules via duplicating an old module (Section 5.3.2).

Other researchers have developed modular networks using generative and developmental methods [14, 27]. These approaches evolve modular neural networks assuming that distributing aspects of a problem across modules makes optimization of task-specific behavior easier. The concept of a module is less strictly defined in these contexts (they need not consist exclusively of output neurons). A module is simply a cluster of interconnected neurons with few connections to neurons in other clusters.

With so many approaches to learning multimodal behavior, it is surprising that none have been applied to Ms. Pac-Man, which is clearly composed of multiple sub-tasks. Regardless, there has been a great deal of research in the domain of Ms. Pac-Man, which is discussed next.

2.2 Ms. 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 similar 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 [12] used Genetic Programming (GP) to learn Pac-Man behavior in a custom simulator, but his variant of the game turned out to be much easier than the arcade version [13, 16].

Even the original Pac-Man is a poor choice for AI research, because ghost behavior is deterministic. Therefore, it is possible to maximize one's score by following memorized paths, without any strategic intelligence. For this reason, current research focuses on the non-deterministic Ms. Pac-Man. Its 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. These differences make success in Ms. Pac-Man depend more on generalization than memorization.

Microsoft's Revenge of Arcade port of this game was used in the annual Ms. Pac-Man screen-capture competition[1] at IEEE conferences from 2007 to 2011. Many approaches have been evaluated in this domain: influence maps [28], game-tree search [17], decision trees [9], and Monte-Carlo Tree Search (MCTS) [10]. A common conclusion is that the quality of any method is greatly affected by the quality of the screen-capture procedure used to assess the game state.

A simulator by Lucas and colleagues [13, 6, 17] avoids this complication. It has become a standard research platform since it was used in the Ms. Pac-Man vs. Ghosts competitions[2] in 2011 and 2012. This simulator includes a `Legacy` team that approximates the original ghosts. Several approaches have been evaluated against the `Legacy` team. GP was used with complex sensors and actions [1, 2], with simple sensors and primitive actions [4], and in conjunction with MCTS [3]. MCTS was also used on its own, though the best results occurred under unusual evaluation conditions (first maze only [18]). Ant Colony Optimization (ACO) [16] was also evaluated. Scores from these methods are compared with the results of this paper in Section 7.3.

Since this 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.

3. 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 50 points. To reduce learning time, 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 1^{st}, 2^{nd}, 3^{rd}, and 4^{th} ghosts eaten in sequence are worth 200, 400, 800, and 1600 points, respectively. The maximum score is achieved by eating all four ghosts after

[1] `http://tinyurl.com/mqcfzxf`
[2] `http://www.pacman-vs-ghosts.net/`

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 make the behavior more consistent (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 `Legacy` ghost team. This team roughly approximates the behavior of 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, which is one source of non-determinism in the game. The other source applies to all ghosts: Normally, ghosts cannot reverse direction, 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 2.2), 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.

4. MULTIMODAL BEHAVIOR

Ms. Pac-Man requires multimodal behavior because she must respond differently to edible and threat ghosts. Although this distinction alone imposes a challenge, the game is actually more complicated than that. Usually, all ghosts are either threats or edible, but there are cases when ghosts of both types are in the maze at the same time. After a ghost is eaten it returns to the lair for a short time before reemerging as a threat, which can happen before the edible time has expired for the other ghosts. A learned policy must therefore not only have behaviors against threatening and edible ghosts, but also for the blended situations in between.

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 eating edible ghosts will be easy. This last behavior, a form of luring, will prove important in the experiments below: The best performing policies dedicate a network module almost completely to luring, which is a surprising and powerful result. The next section describes how these modular networks are evolved.

5. 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.

5.1 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 in Ms. Pac-Man 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 instead of one improves search by helping avoid local optima [11]. 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 $\vec{u} = (u_1, \ldots, u_n)$, i.e. $\vec{v} \succ \vec{u}$, 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 $\vec{y} \succ \vec{x}$. 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 solution if it is strictly better in at least one objective and no worse in the others. The best solutions are not dominated by any other solutions, and make up the Pareto front of the search space. The next best individuals are those that would be in a recalculated Pareto front if the actual Pareto front were removed first. Layers of Pareto fronts can be defined by successively removing the front and recalculating it for the remaining individuals. Solving a multiobjective optimization problem involves approximating the *first* Pareto front as best as possible. In this paper this goal is accomplished using the Non-Dominated Sorting Genetic Algorithm II (NSGA-II [8]).

NSGA-II uses $(\mu + \lambda)$ 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.

Applying NSGA-II to a problem produces a population containing an approximation to the Pareto front. This approximation set potentially contains multiple solutions. Usually, this set must be analyzed in order to determine which solutions fulfill the needs of the user, but 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. In this paper, NSGA-II is used to evolve artificial neural networks.

5.2 Neuroevolution

Neuroevolution is the simulated evolution of neural networks. All behavior in this paper is learned via a version of NEAT (Neuro-Evolution of Augmenting Topologies [22]), a constructive neuroevolution method that starts with simple networks that become more complex from mutations across several generations. The initial population of networks has no hidden neurons, only input and output neurons.

Networks are modified by the usual three mutation operators in NEAT. Weight mutation perturbs the weights of existing network connections, link mutation adds new (potentially recurrent) connections between existing nodes, and node mutation splices new nodes along existing connections. 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 basic NEAT method described so far has been used to solve many challenging problems [20, 22], but the resulting networks only define a single control policy. The next section describes new methods for augmenting network architectures so that they possess multiple policies, making it easier to learn multimodal behavior.

5.3 Modular Networks

The modular networks of this paper can have multiple output modules. Each output module defines a different control policy. In fixed networks, evolution must discover how to use the existing modules, and when Module Mutation is used, evolution must also settle on an appropriate number of modules. Both approaches depend on preference neurons

5.3.1 *Preference Neurons*

Preference neurons make module arbitration possible. Each module's preference neuron outputs the network's relative preference for using that module. On each time step, the network module whose preference neuron output is highest is used to define the output of the network.

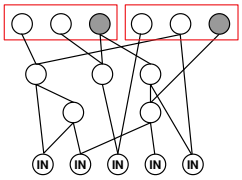
For example, assume a domain requires two outputs to designate the behavior of an agent, and a network has two modules (Fig. 1a). Then the network has six outputs: two policy neurons and one preference neuron for Module 1, and two policy neurons and one preference neuron for Module 2. Whenever the output 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 learns 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.

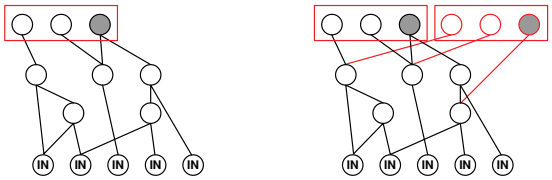
5.3.2 *Module Mutation*

Module Mutation is any structural mutation operator that adds a new output module to a neural network. An unknown 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 have been evaluated in prior research [21]. MM(P) initializes new modules with lateral inputs from a previous module. The new module is thus similar to the previous module, but not identical because of the tanh activation function. In contrast, MM(R) initializes new modules with random input link weights and sources. New MM(R) modules are often very different from existing modules, and explore the space of policies better,



(a) Two-module network



(b) Network before MM(D) (c) Network after MM(D)

Figure 1: **Modular Networks:** These example networks are designed for a domain where two policy neurons define the behavior of an agent. Each output module is contained in its own red box. (a) A fixed network with two modules that uses preference neurons to determine which module to use. (b) A starting network in a population where Module Mutation is enabled. It has one module, and an irrelevant preference neuron. (c) After MM(D), the network gains a new module, and both the pre-existing and newly added preference neurons are relevant. The policy neurons in the new module are 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. Extra modules allow these networks to learn multimodal behavior more easily by associating a different module with each behavioral mode.

but they also have a higher chance of decreasing fitness. MM(R) was shown to be superior to MM(P) in two previous domains requiring multimodal behavior [21].

This paper presents a new approach combining Module Mutation with the idea of the duplication operator (Section 2.1.2). This new operator is called Module Mutation Duplicate (MM(D); Fig. 1c), 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 that 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.

It turns out that MM(D) is the most powerful Module Mutation approach in Ms. Pac-Man [19]. New MM(D) modules behave like previous modules (an improvement on MM(P)), but can diverge from previous modules because new links are created to define each new module (as with MM(R)). It is therefore the method of choice in this paper. Support for all varieties of modular networks, combined with the integration of NSGA-II, results in a new software framework called Modular Multiobjective NEAT (MM-NEAT)[3]. MM-NEAT's ability to discover multimodal behavior is demonstrated in the experiments described next.

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

6. EXPERIMENTAL SETUP

This section describes the policy representation, sensors, objectives, and specific network architectures used to evolve multimodal behavior in Ms. Pac-Man.

6.1 Direction-Evaluating Policy

A learned policy can control Ms. Pac-Man in several different ways (see Section 2.2), regardless of the method used to represent the function approximator defining the policy.

This paper uses an approach introduced by Brandstetter and Ahmadi (BA [4]). Direction-oriented sensors evaluate each available direction using a function approximator with a single output, and then the direction with the highest output value is picked. This approach is similar to the common Reinforcement Learning approach of evaluating afterstates to pick an action, but does not require a model, and has proven superior to previously attempted afterstate-based approaches to Ms. Pac-Man [13, 6].

The BA approach was chosen because it uses primitive actions and simple sensors. Other evolved Ms. Pac-Man agents [12, 1, 2] 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. The sensors in this paper are similarly simple, as explained next.

6.2 Sensor Configuration

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 makes it clear how the sensor should be interpreted: Threat ghosts are always bad and edible ghosts are always good. In contrast, if there were simply a generic ghost sensor, there would be conflicting ways of interpreting it depending on the type of ghost. Thus splitting up the sensors into threat and edible sensors is another source of bias in learning. Such a bias makes learning easier [19], but requires knowledge that is not available in all domains.

This paper shows how modular networks can learn multimodal behavior even with unbiased sensors that do not suggest how to break up the domain into separate tasks. There are three sets of such sensors: direction-oriented ghost sensors, other direction-oriented sensors, and sensors that are not direction-oriented. Direction-oriented distances measure the shortest path starting in a particular direction and continuing without reversing. Distances have a maximum of 200, and all sensors are scaled to $[0, 1]$.

There are 16 direction-oriented ghost sensors: distances to the 1^{st}, 2^{nd}, 3^{rd}, and 4^{th} 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.

There are six remaining direction-oriented sensors: distances to the nearest pill, power pill, and junction, the maximum numbers of pills and junctions within 30 steps, and a special sensor called 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. OFNJ is the one sensor from this group that is not part of the BA approach. 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 death, but high scores depend on eating edible ghosts, which are only detectable by the ghost sensors above.

The remaining eight sensors are not direction-oriented, i.e. they provide the same reading for each direction checked. They are a constant bias, the proportions of remaining pills, power pills, edible ghosts, and edible time, and Boolean sensors indicating whether any ghost is edible, all ghosts and threats are outside the lair, and Ms. Pac-Man is within 10 steps of a power pill. This last sensor is useful because it warns Ms. Pac-Man that a power pill is about to be eaten, which helps optimize the timing of this important event.

These sensors are sufficient to make intelligent decisions, but are challenging to use because they are so general. However, the results show that modular networks are able to make effective use of them.

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

6.3 Objectives and Performance

Populations are evolved with separate pill and ghost objectives. The `Pill Score` is simply the number of pills eaten. This fitness function treats power pills as regular pills, 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 tricky to define. For each power pill eaten, the point value of each subsequently eaten ghost doubles, so this objective gives higher rewards for the ghosts that are worth more points. The 1^{st}, 2^{nd}, 3^{rd}, and 4^{th} 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.

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 stay 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` (special handling of power pills causes a small discrepancy). Results in Section 7 are 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 tradeoff surface to find skilled, multimodal behavior.

6.4 Evolving Networks

These experiments show the benefits of modular neural networks in a domain requiring multimodal behavior. Populations of networks with `One Module`, `Two Modules`, and `Three Modules` are evolved. Modular networks use prefer-

ence 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 learning 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(D)`, are also evaluated.

Populations of each type are evolved 20 times for 200 generations each, 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(D) runs, Module Mutation is applied to 10% of offspring. Topological network crossover is applied 50% of the time offspring are produced, with parents chosen via tournament selection.

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 more easily allow module arbitration to be swayed by mutations. Evolving populations of networks under these conditions leads to the results in the next section.

7. RESULTS

This section analyzes the results of learning, describes the behaviors exhibited by champion networks, and compares these champions to previous results in the literature.

7.1 Learning

The main result is that modular networks achieve better game scores than `One Module` networks throughout evolution (Fig. 2). `Two Modules` performs the best, then `Three Modules` followed by `MM(D)`. Learning curves show average scores across 20 champions for each method. Because these scores were achieved during evolution, each champion score is in turn an average across 10 game evaluations.

Such averaging mitigates some of the noise in evaluation, but to get a more reliable evaluation of the final results, they were also compared post-learning. The champion of each run was evaluated 100 times, and their average scores were compared for each method. The non-parametric Kruskal-Wallis test confirmed that there is a significant difference between at least two of the four methods ($H = 13.55, p \approx 0.0036$). Mann-Whitney U tests were then used to compare each modular approach to `One Module`: `Two Modules` ($U = 313, p \approx 0.0017$), `Three Modules` ($U = 316, p \approx 0.0013$), and `MM(D)` ($U = 283, p \approx 0.025$) are all significantly better than `One Module`. However, none of the modular approaches are significantly different from each other.

The average scores from post-learning evaluations are plotted in Fig. 3 against the percentage of the time that the most used module was chosen by that champion across all 100 evaluations. This figure is sufficient to understand the module usage behavior of all champions because, surprisingly, no modular champion uses more than two modules, including `Three Modules` and `MM(D)` champions. `MM(D)` champions all had two or three modules, but networks with as many as five modules also emerged in some runs. Extra modules are simply ignored (with usage less than 0.2%). For the modular methods, there are two large clusters, which can be analyzed by observing the behavior of networks in these clusters.

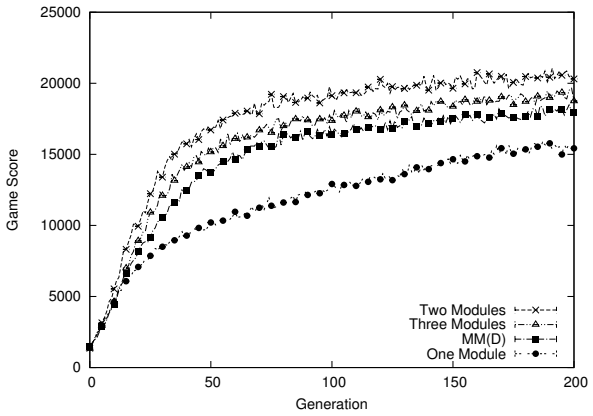


Figure 2: **Average Champion Scores Over Evolution:** Average scores across 20 runs of each method are plotted by generation. The modular approaches quickly split off from `One Module` and maintain superiority until the end. Therefore, modular networks attain better scores quicker, and overall.

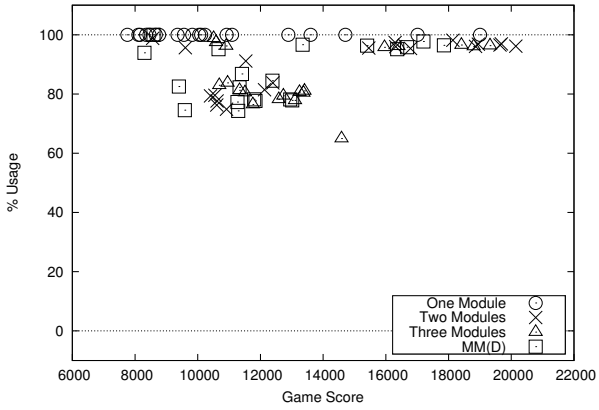


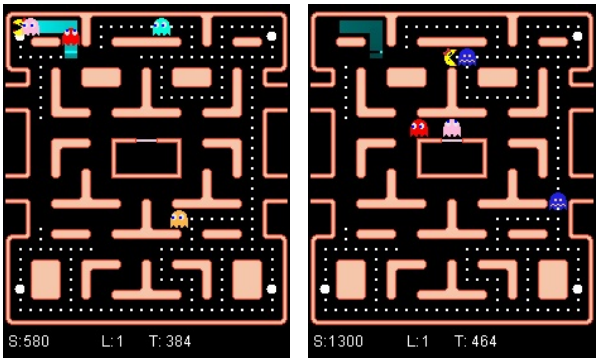
Figure 3: **Average Post-Learning Champion Scores vs. Primary Module Usage:** Each champion is evaluated 100 times, and the average scores are plotted against the percentage of the time each choses its most used module. Recall that networks process inputs for each direction in which Ms. Pac-Man can move; the "chosen" module is the one that the network used when fed inputs for the direction that Ms. Pac-Man ultimately chose to move in. All `One Module` networks use their single module 100% of the time, and most score below 11,000. Modular approaches that use one module around 80% of the time (scoring 9,000–14,000) generally use a threat/edible split. However, the best modular approaches (scoring 15,000–21,000) use one module over 95% of the time, because the other module, used for luring, is only needed a small percentage of the time when it really counts.

7.2 Behavior

Observing evolved behaviors reveals how multiple modules are used to implement successful behaviors. Videos of behaviors can be seen at `http://nn.cs.utexas.edu/?ol-pm`. There are two common module usage patterns: threat/edible split and luring.

The threat/edible split is the obvious way of splitting the domain: Ms. Pac-Man should behave differently around edible ghosts than around threat ghosts. However, ghosts of both types are sometimes present simultaneously. In these situations, Ms. Pac-Man generally uses the edible module until threat ghosts block the path to any remaining edible ghosts, which is a sensible strategy for staying alive.

This strategy achieves good scores (9,000–14,000), but occasionally has trouble eating ghosts. Sometimes it leads Ms. Pac-Man to eat power pills at inopportune times. The edible



(a) Luring (b) After Luring

Figure 4: **Behavior With a Luring Module:** (a) Ms. Pac-Man waits at the junction for the ghosts to get close, then activates the luring module, which leads her and the ghosts to the power pill. The cells in the upper left shaded blue indicate locations in which the luring 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 luring 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 it leads to such high scores. An animation of this behavior is at `http://nn.cs.utexas.edu/?ol-pm`.

time is so short that even though she immediately switches to a module that pursues the ghosts, she has a hard time catching them before time runs out. To eat more ghosts, Ms. Pac-Man needs to lure them close to power pills before eating them, so that the edible ghosts are easy to catch.

This is the strategy used by the best modular networks (those scoring 15,000–21,000; Fig. 4). Ms. Pac-Man uses one module for navigating the maze and eating pills while ghosts chase her, but sometimes she stops at a junction and waits for the ghosts to get closer. When she is almost captured, a luring module takes over and leads the pursuing ghosts toward a power pill, which she then eats. At this point, control returns to the first module, and Ms. Pac-Man quickly eats the now vulnerable ghosts. The luring module also helps Ms. Pac-Man escape dangerous situations even when no power pill is present. The basic behavior is the same: Lure the ghosts to a junction, then escape via the best route.

Networks that discover a luring module use it a small portion of the time (under 5%), but it is vital for increasing `Ghost Score`. However, it is surprising that the other module can support the different behavioral modes of running from threat ghosts and chasing edible ghosts. `Three Modules` and `MM(D)` runs that learn this behavior also use only two modules, even though other modules are available to take on these extra behavioral modes. In fact, the lower performance of these methods indicates that the extra modules are a harmful distraction in this domain.

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, luring is very different from these behaviors, which is why it tends to need a dedicated module in order to execute correctly.

The worst individuals are `One Module` 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 in these rare cases the network in question is mostly using only one of its available modules.

There are also five high performing `One Module` outliers. The best exhibits luring behavior, but its ability to chase ghosts suffers as a result, which is why modular luring networks achieve better scores. With only one module, improvements in one behavior often come at the expense of other behaviors, which is why even the best `One Module` champions have trouble exhibiting multimodal behavior.

Such multimodal behavior leads to high scores in the full game as well, as shall be shown in the next section.

7.3 Comparison

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 variant is called `FourMaze`, because evaluation is 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` and `MPMvsG` variants (Table 1). The average scores of the modular networks are superior to average scores of all previous works in both `FourMaze` and `MPMvsG`. The maximum scores are also better in most cases. Even the best `One Module` champion beats previously published results, although typical `One Module` performance is lower.

The results demonstrate the success of multiobjective neuroevolution in general, and modular networks in particular, but there are many promising directions for future research.

8. DISCUSSION AND FUTURE WORK

Modular networks discover multimodal behavior more often than networks with one module. The worst modular networks are those that fail to use multiple modules, indicating that the potential of modular architectures is not always realized.

Therefore, one direction for future work is to investigate methods of encouraging use of modules that remain untapped. Multiobjective evolution should fill this role by encouraging exploration of trade-offs in objective space, 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 [20]. This fitness-based shaping technique turns off objectives in which the population is performing well so that evolution can focus on the objec-

Method	FourMaze		MPMvsG	
	AVG	MAX	AVG	MAX
GP [1]	16,014	**44,560**	N/A	N/A
GP+Training Camps [2]	11,413	31,850	N/A	N/A
GP (BA) [4]	N/A	N/A	19,198	33,420
ACO [16]	N/A	N/A	36,031	43,467
GP+MCTS [3]	N/A	N/A	32,641	69,010
`One Module`	32,009	41,320	57,767	87,840
`Two Modules`	**32,959**	42,120	65,323	88,430
`Three Modules`	32,039	40,720	63,701	82,590
`MM(D)`	32,647	44,520	**65,447**	**100,070**

Table 1: **Post-Learning Scores in Full Ms. Pac-Man:** Champions from this paper are compared with results from the literature in two variants of the full game. Only results for champion networks with the highest average scores are reported. The best `One Module` scores are better than previous work as well, hinting that there is a benefit from using multiobjective neuroevolution with a challenging evaluation scheme (just one life), but keep in mind that the best `One Module` result is an outlier. Average modular network scores are higher than previously reported scores and `One Module` scores, especially in the `MPMvsG` variant.

tives that need it most. Such a change in focus encourages modular networks to use additional modules better [19].

Another shaping technique that is easily combined with multiobjective evolution is Behavioral Diversity [15]: 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 a luring 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. Figuring out how to reward multiple module use without punishing corner cases like these is a challenging open question.

9. CONCLUSION

Ms. Pac-Man is a challenging game that requires multimodal behavior to succeed. Modular approaches are superior because they can dedicate separate modules to different modes of behavior. Some learn to handle threat and edible ghosts with separate modules, which is a sensible division, but others learn an even better, unexpected division that focuses one module on the behavior of luring ghosts near power pills, so that they can be easily eaten. 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.

10. ACKNOWLEDGMENTS

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

11. REFERENCES

[1] A. M. Alhejali and S. M. Lucas. Evolving diverse Ms. Pac-Man playing agents using Genetic Programming. In *UKCI*, 2010.

[2] A. M. Alhejali and S. M. Lucas. Using a Training Camp with Genetic Programming to evolve Ms Pac-Man agents. In *CIG*, pages 118–125, 2011.

[3] 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*, 2013.

[4] M. F. Brandstetter and S. Ahmadi. Reactive control of Ms. Pac Man using information retrieval based on Genetic Programming. In *CIG*, pages 250–256, 2012.

[5] R. A. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation*, 2(10), 1986.

[6] P. Burrow and S. M. Lucas. Evolution versus Temporal Difference Learning for learning to play Ms. Pac-Man. In *CIG*, pages 53–60, 2009.

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

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

[9] 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*, 2012.

[10] N. Ikehata and T. Ito. Monte-Carlo Tree Search in Ms. Pac-Man. In *CIG*, pages 39–46, 2011.

[11] J. D. Knowles, R. A. Watson, and D. Corne. Reducing Local Optima in Single-Objective Problems by Multi-objectivization. In *EMO*, 2001.

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

[13] S. M. Lucas. Evolving a neural network location evaluator to play Ms. Pac-Man. In *CIG*, 2005.

[14] J.-B. Mouret and S. Doncieux. MENNAG: a modular, regular and hierarchical encoding for neural-networks based on attribute grammars. *Evolutionary Intelligence*, 2008.

[15] J.-B. Mouret and S. Doncieux. Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In *GECCO*, pages 627–634, 2009.

[16] G. Recio, E. Martín, C. Estébanez, and Y. Sáez. AntBot: Ant colonies for video games. *TCIAIG*, 2012.

[17] D. Robles and S. M. Lucas. A simple tree search method for playing Ms. Pac-Man. In *CIG*, 2009.

[18] S. Samothrakis, D. Robles, and S. M. Lucas. Fast Approximate Max-n Monte Carlo Tree Search for Ms Pac-Man. *TCIAIG*, 3(2):142–154, 2011.

[19] J. Schrum. *Evolving Multimodal Behavior Through Modular Multiobjective Neuroevolution*. PhD thesis, University of Texas at Austin, 2014.

[20] J. Schrum and R. Miikkulainen. Evolving agent behavior in multiobjective domains using fitness-based shaping. In *GECCO*, 2010.

[21] J. Schrum and R. Miikkulainen. Evolving multimodal networks for multitask games. *TCIAIG*, 4(2), 2012.

[22] K. O. Stanley and R. Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation*, 10:99–127, 2002.

[23] P. Stone and M. Veloso. Layered Learning. In *ECML*, 2000.

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

[25] J. Togelius. Evolution of a subsumption architecture neurocontroller. *Intelligent and Fuzzy Systems*, 2004.

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

[27] P. Verbancsics and K. O. Stanley. Constraining connectivity to encourage modularity in HyperNEAT. In *GECCO*, pages 1483–1490, 2011.

[28] N. Wirth and M. Gallagher. An Influence Map Model for Playing Ms. Pac-Man. In *CIG*, 2008.