Using Action Abstraction to Evolve Effective Controllers

Track: Artificial Life, Evolutionary Robotics, Adaptive Behavior, Evolvable Hardware

ABSTRACT

We propose that abstracting the actions of a behavior coordination mechanism promotes the faster development and higher fitness of an effective controller for complex, composite tasks. Various techniques are well suited for the development of controllers for individual simple tasks. However, as individual tasks are combined into complex, composite tasks, many of these techniques quickly become impractical. By reusing existing behaviors, the focus of development for a controller can be shifted from low-level control to high-level coordination of these existing behaviors. As a result, the development of an effective controller becomes far more practical. Experiments using a single-agent task in a continuous environment demonstrate that grammatical evolution is capable of discovering fuzzy rulesets which effectively coordinate existing behaviors in a controller in fewer generations and with higher fitness than a monolithic controllers.

Keywords

Artificial intelligence, Evolutionary robotics, Fuzzy systems, Modelling behaviours and ecosystems

1. INTRODUCTION

The development of controllers for intelligent agents given a simple task is relatively straightforward and even the most basic techniques are easily capable of developing such controllers. However, as agents are given more than one task, the development of effective controllers quickly becomes impractical. While some complex tasks are composed of a series of sequential simple tasks, this paper is concerned with complex tasks that are composed of a collection of simple tasks that may all be active at the same time and may not have termination criteria.

The naïve approach to such complex tasks is to combine the simple, or *primitive*, tasks into a single complex, or *composite*, task. A controller is then developed for the entire composite task, effectively developing a single, monolithic controller responsible for addressing each primitive task and the coordination between them. The problem with this monolithic approach is that development of even the simplest composite task can be impractical since each primitive task added makes the prospect of developing an effective controller exponentially more difficult. Furthermore, the controller is highly dependent on the current composite task and definition of fitness making its reuse limited. To make the development of controllers for composite tasks practical, a method for combating the effects of this curse of dimensionality must be developed. A solution that also promotes the development of higher quality controllers further improves their practicality.

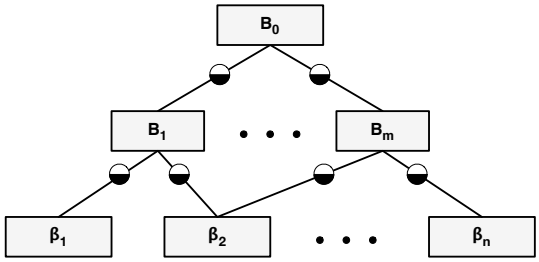


Figure 1: A set of primitive behaviors (denoted β_i) are organized into a hierarchy and adaptively weighted by composite behaviors (denoted B_m) as described by Tunstel [21] and redrawn here. The half-filled circles denote the weights used to compose behaviors.

One approach to mitigating the curse of dimensionality is to keep the primitive tasks separate and develop a controller for each separately. The individual controllers can then be combined in such a way as to act as a controller for the composite task as a whole. In behavior-based robotics, these individual controllers are known as *behaviors* and are responsible for accomplishing primitive tasks [2]. Coordination of these behaviors can be performed either by using an explicit coordination mechanism or within the behaviors themselves [11]. The option of controlling coordination within the behaviors themselves can lead to complexity problems in developing the behaviors, so an explicit coordination mechanism is often chosen. Since an explicit coordination is only responsible for effectively coordination each of the combating behaviors to accomplish the composite task, it does not need to produce low-level control actions directly. This change in focus allows for the use of techniques to address the curse of dimensionality which are not applicable to low-level control problems.

GECCO '09 Montreal, Quebec CA

Since composite tasks are, by definition, hierarchical in nature, a common approach is to use a hierarchical approach in developing associated controllers [1, 5, 9, 21]. Since the use of a hierarchy allows for a change in focus from low-level control to high-level coordination, abstraction can be used to simplify the coordination process. By changing the development focus from low-level control to high-level coordination, we are effectively abstracting the actions of the coordination mechanism. We propose that this abstraction of the action space will promote the faster development of an overall controller and result in higher overall fitness. Furthermore, we propose that this action abstraction insulates the controller from the negative effects of state abstraction and perceptual aliasing [20]. In this work, we use grammatical evolution to evolve fuzzy rulesets that are responsible for coordinating low-level behaviors to accomplish a given composite task (see Figure 1). Through a series of experiments, we demonstrate that this change in focus from low-level control to high-level coordination results in a significant improvement in the rate at which effective coordination can be evolved and the fitness of the overall controller when compared to a monolithic controller.

2. MOTIVATION

In this work, we consider the problem of learning a controller for a set of N simple, or *primitive*, tasks, where each task is a Markov decision problem (see [19] for a more in-depth discussion). All of the primitive tasks may be active at the same time and may not even have termination criteria. In general, the state space for each task is distinct and local to that specific task, while all the tasks share the same action space. For example, if both a CollisionAvoidance and GoalSeek task are active, only the state space for the CollisionAvoidance task contains information regarding potential collisions while only GoalSeek's state space contains information regarding the goal location. The combination of these primitive tasks forms a *composite* task. An important aspect to this combination is that the primitive tasks will often interfere with one another, necessitating an intelligent coordination mechanism.

While each primitive task has a (relatively) small and local state space, the state space for the composite task represents the cross product of the state space for each primitive task: $S = S_1 \times S_2 \times \ldots S_N$. When this joint state space is combined with the low-level action space, the resulting complexity can make the development of an effective controller impractical. This exponential increase in the size of the state-action space is the motivating factor for finding an alternative to the approach of developing a single, monolithic controller for the composite task. Without such an alternative, the practical ability to develop an effective controller for the composite task is in question.

3. FUZZY BEHAVIOR HIERARCHIES

In behavior-based controllers, command fusion is the act of combining the contributions of individual behaviors into a single action [11]. The behaviors are executed in parallel and their resulting actions are then combined into a single action. Since the behaviors frequently have individual goals that conflict with one another, the controller must effectively coordinate the behaviors and combine the actions to be effective. Fuzzy logic[1] can be used to simplify this process through the use of fuzzy inferencing techniques and has been frequently used with great success [4, 11, 14, 17, 21, 23]. In fuzzy command fusion, each behavior is implemented using a set of fuzzy rules whose output is the desired action [16]. However, fuzzy command fusion in and of itself is incapable of fully addressing the complexity of coordinating many behaviors.

As previously discussed, the use of a hierarchy can reduce the number of rules for a hierarchy to be a linear function of the state space as opposed to non-hierarchical approaches in which the number of fuzzy rules is exponential in the number of features [13]. An example of a hierarchical approach, and the one used for these experiments, is a fuzzy behavior hierarchy [21]. Behaviors responsible for accomplishing simple, primitive tasks are called *primitive behaviors* (see Figure 1). Primitive behaviors reside at the lowest level of the hierarchy and use low-level control actions to accomplish their task. In general, primitive behaviors are not required to use the same set of actions, but the primitive behaviors used in this work do. Individual rules within a primitive behavior's ruleset have the following form:

$$\text{IF } x \text{ is } \tilde{A}_i \text{ THEN } u \text{ is } \tilde{B}_i \tag{1}$$

where x and u represent the primitive task state information and motor command output fuzzy variables, respectively. $\tilde{A}_i$ and $\tilde{B}_i$ represent the fuzzy linguistic values corresponding to the variables x and u [21]. Examples of fuzzy linguistic values describing state information are `LEFT` and `CLOSE`, while examples fuzzy linguistic values describing motor commands are `FASTER` and `SMALL_RIGHT`.

High-level, or *composite*, behaviors reside in the higher levels of the hierarchy and modulate lower-level behaviors, which can be either primitive or other composite behaviors, through weights in order to accomplish a composite task. A composite behavior determines a lower-level behavior's weight, or *degree of applicability* (DOA), by evaluating the behavior's relevance given the agent's state with respect to the composite task. For example, the composite behavior for the CA-GS task could determine that since a collision is not imminent, the CollisionAvoidance behavior should have a `LOW` weight while the GoalSeek behavior should have a `HIGH` weight. Individual rules within a composite behavior's ruleset have the following form:

$$\text{IF } x \text{ is } \tilde{A}_i \text{ THEN } \alpha \text{ is } \tilde{D}_i \tag{2}$$

where $\tilde{A}_i$ is defined in Equation 1 and $\tilde{D}_i$ represents the fuzzy linguistic values (e.g., `LOW`, `MEDIUM`, `HIGH`, etc.), corresponding to a behavior's DOA [21].

Once this weighting action is performed, the behaviors for the lower level tasks are executed. The DOA value for each primitive behavior is used to weight its output and, therefore, allow some primitive behaviors to contribute more to the current action that other primitive behaviors. The output of the entire behavior hierarchy is calculated as follows:

$$\beta_H = \biguplus_{p \in P} \alpha_p \cdot \beta_p \tag{3}$$

where α_p and β_p are the DOA and output of a given primitive behavior p, respectively, β_H is the output of the entire

[1]See [6, 7] for a more detailed description on the use of fuzzy logic in control.

behavior hierarchy, and P is the set of all primitive behaviors. The fuzzy output values are then defuzzified using Center of Sums defuzzification as follows:

$$u^* = \frac{\int_{u \in U} u \cdot \mu_{\tilde{\beta}_H}(u)}{\int_{u \in U} \mu_{\tilde{\beta}_H}(u)} = \frac{\int_{u \in U} \sum_{p \in P} \alpha_p \cdot \mu_{\tilde{\beta}_P}(u)}{\int_{u \in U} \sum_{p \in P} \alpha_p \cdot \mu_{\tilde{\beta}_P}(u)} \quad (4)$$

where u is the motor command output fuzzy variable, α_p and β_p are the DOA and output of a given primitive behavior p, respectively, β_H is the output of the entire behavior hierarchy, and P is the set of all primitive behaviors. This is an instance of *weight-counting* defuzzification [22] and results in a crisp-valued output used for agent control

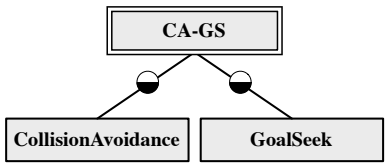
Since composite behaviors only weight lower-level behaviors using state information, composite behaviors do not require lower-level behaviors to provide any information to aid in the weighting process. This is in contrast to other behavior coordination mechanisms which, for example, may require low-level behaviors to indicate the utility of a specific action [12]. The only restriction on the lower-level behaviors is that primitive behaviors generate actions that can be process using fuzzy inferencing.

Since primitive behaviors are the only behaviors responsible for producing control actions, it is possible that composite behaviors may not require the full joint state space in order to effectively weight lower-level behaviors. Precisely how much of the joint state space is required, however, is unknown. For example, it is possible that the direction of the closest collision is irrelevant when weighting the CollisionAvoidance primitive behavior and only the estimated time until the collision is important. There are two ways in which the state space can be reduced. First, state information that is determined to be irrelevant can simply be removed from the state space. However, the difficulty is in determining which information is relevant and which is not. In the second option, state information is extracted into a more abstract form. Since primitive tasks are, by definition, simple and straightforward, the process by which information is abstracted can be easily defined and is, therefore, the method used in this work.

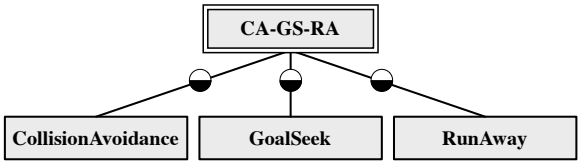
4. THE NAVIGATION PROBLEM

In this work, we first consider the conceptually simple composite task of navigating an agent towards a goal location while avoiding any obstacles in its path. This composite task, denoted CA-GS, is the combination of the CollisionAvoidance and GoalSeek primitive tasks (see Figure 2a). The state information local to the CollisionAvoidance task consists of the relative direction to the collision and the estimated time until collision. The state information local to the GoalSeek task consists of the relative direction to the goal location and the estimated time of arrival at the goal location given only the current speed of the agent. All state information is normalized and measured relative to the agent's position and orientation. This is done to decouple the learned behavior from the specifics of the environment.

We also consider a more complex composite task in which a third primitive task is added to the previous two. In this new primitive task, denoted RunAway, the agent must avoid approaching "hazardous" objects in the environment (see Figure 2b). The hazardous objects are not physical objects with which the agent can collide, but instead represent areas that could be dangerous to the agent like high-traffic areas or areas with difficult terrain. The state information local to the RunAway task consists of the relative direction and magnitude of a repulsive "force" which effectively acts to push the agent away from all the sensed hazardous objects. The new composite task is denoted CA-GS-RA.



(a) CA-GS Behavior Hierarchy



(b) CA-GS-RA Behavior Hierarchy

Figure 2: In the CA-GS behavior hierarchy, the CA-GS composite behavior weights the CollisionAvoidance and GoalSeek primitive behaviors. The CA-GS-RA behavior hierarchy adds the RunAway primitive behavior.

While these tasks are conceptually simple, their execution is not. The environments in which these tasks are performed are unbounded and continuous. This is in contrast to the grid worlds commonly used, but more accurately reflects the complex tasks which we wish to give agents. Furthermore, both two and three-dimensional environments are used to increase the complexity of the state-action space without modifying the composite task itself. Since fuzzy inferencing is used for control, the real-valued state information is discretized using relatively coarse fuzzy linguistic values.

5. EVOLVING FUZZY RULESETS

Tunstel originally used genetic programming to evolve the fuzzy rules for composite behaviors [21]. However, exploratory experiments have shown that the creation of invalid fuzzy rules can be quite common when using genetic programming. To avoid wasted computational effort in evaluating these invalid fuzzy rules, we use grammatical evolution [10, 15] to evolve the fuzzy rules instead of genetic programming.

Grammatical uses a variable-length bit-string as a genotype, just as genetic algorithms. This bit-string is then used to generate the phenotype using a grammar. Codons within the bit-string are used to select replacement symbols from within the grammar. Replacement stops when only terminal symbols remain or a maximum number of replacements has been made. If the end of the bit-string is reached before either of these conditions is met, replacement continues using the codons at the beginning of the bit-string. The resulting phenotype has the expressive power of individuals from genetic programming. However, since a grammar is used to generate the phenotype, the rules of the grammar can ensure

that the resulting phenotype is valid.

⟨*rule*⟩ ::= ⟨*antecedent*⟩ ⟨*consequent*⟩ ⟨*rule*⟩
| ⟨*antecedent*⟩ ⟨*consequent*⟩ ⟨*rule*⟩
| ⟨*antecedent*⟩ ⟨*consequent*⟩

⟨*antecedent*⟩ ::= ⟨*antecedent*⟩ ⟨*antecedent*⟩
| VERY (⟨*antecedent*⟩)
| NOT (⟨*antecedent*⟩)
| ⟨*collision-dir*⟩
| ⟨*time-till-collision*⟩
| ⟨*goal-dir*⟩
| ⟨*goal-arrival-time*⟩

⟨*consequent*⟩ ::= ⟨*consequent*⟩ ⟨*consequent*⟩
| VERY (⟨*consequent*⟩)
| ⟨*relative-steer-speed*⟩
| ⟨*steer-dir*⟩

⟨*collision-dir*⟩ ::= collision-dir(BACK_LEFT)
| collision-dir(LEFT)
| collision-dir(SMALL_LEFT)
| collision-dir(CENTER)
| collision-dir(SMALL_RIGHT)
| collision-dir(RIGHT)
| collision-dir(BACK_RIGHT)

⟨*time-till-collision*⟩ ::= time-till-collision(NOW)
| time-till-collision(REAL_SOON)
| time-till-collision(SOON)
| time-till-collision(LONG_TIME)
| time-till-collision(DISTANT)

⟨*goal-dir*⟩ ::= goal-dir(BACK_LEFT)
| goal-dir(LEFT)
| goal-dir(SMALL_LEFT)
| goal-dir(CENTER)
| goal-dir(SMALL_RIGHT)
| goal-dir(RIGHT)
| goal-dir(BACK_RIGHT)

⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(REAL_SOON)
| goal-arrival-time(SOON)
| goal-arrival-time(LONG_TIME)
| goal-arrival-time(DISTANT)

⟨*relative-steer-speed*⟩ ::= steer-speed(MUCH_SLOWER)
| steer-speed(SLOWER)
| steer-speed(SAME)
| steer-speed(FASTER)
| steer-speed(MUCH_FASTER)

⟨*steer-dir*⟩ ::= steer-dir(LEFT)
| steer-dir(SMALL_LEFT)
| steer-dir(CENTER)
| steer-dir(SMALL_RIGHT)
| steer-dir(RIGHT)

Figure 3: The grammar used for evolving a monolithic fuzzy ruleset for the two-dimensional CA-GS composite behavior is shown. The grammar's start symbol is ⟨*rule*⟩.

Figure 3 shows the grammar used to generate fuzzy rulesets for the two-dimensional CA-GS composite behavior. Replacement starts using the start symbol ⟨*rule*⟩. It is designed to not only ensure that at least one rule is generated, but also bias replacement towards the generation of multiple rules. Exploratory experiments demonstrated that without this bias, the generated rulesets had too few rules to provide effective control[2] Since grammatical evolution does not provide a means of giving weight to specific choices for replacement, this bias was manually implemented by specify multiple replacements which generate additional rules.

The replacement of symbols continues until only terminal symbols remain (i.e, antecedents or consequents), or the maximum number of rules have been generated. The resulting ruleset is guaranteed to be valid and contain complete antecedents and consequents. This guarantee can provide significant benefits as the search space is now restricted to all valid rulesets. Note that the grammar uses linguistic hedges such as `NOT` and `VERY` to allow broad or more specific rules respectively. The use of these linguistic hedges can result in fewer rules, and, therefore, a simpler ruleset.

6. EXPERIMENTS

To evaluate the relative performance of fuzzy behavior hierarchies, fuzzy rulesets for the CA-GS and CA-GS-RA composite behaviors and a monolithic behavior were evolved. Since it is possible that composite behaviors in fuzzy behavior hierarchies do not require the full state space for effective coordination of lower-level behaviors, grammars describing four different levels of abstraction of the agent's state space were used. These were used to evaluate how abstractions affected both the rate at which effective rulesets evolved and the quality of the evolved rulesets.

Full This state space represents the original, joint state space of all the primitive tasks without any abstraction and acts as a baseline for comparison (see Figure 4a).

Large In this state space, state information describing directions, such as `SMALL_LEFT` or `SMALL_RIGHT`, are abstracted away into variables which denote the absolute value of the angle, such as `SMALL` (see Figure 4b). State information not describing a direction remains unchanged.

Medium In this state space, state information describing three-dimensional directions are abstracted away into a single variable which denotes the absolute value of the largest angle (see Figure 4c). State information not describing a direction remains unchanged. Note that in two-dimensional environments, this state space is the same as the **Large** state space.

Small In this state space, state information is abstracted into a single dynamic priority which is calculated using all the relevant state information local to each primitive task (see Figure 4d). This dynamic priority represented the task's determination of its applicability

[2]While such a bias is not generally needed in a grammar, the fact that we are randomly generating rules using the grammar necessitates the bias. An example would be to give higher weight to the letter 'e' when randomly generating words for the English language since it is the most common letter.

⟨*goal-dir-theta*⟩ ::= goal-dir-theta(BACK_LEFT)
| goal-dir-theta(LEFT)
| goal-dir-theta(SMALL_LEFT)
| goal-dir-theta(CENTER)
| goal-dir-theta(SMALL_RIGHT)
| goal-dir-theta(RIGHT)
| goal-dir-theta(BACK_RIGHT)

⟨*goal-dir-phi*⟩ ::= goal-dir-phi(DOWN)
| goal-dir-phi(SMALL_DOWN)
| goal-dir-phi(CENTER)
| goal-dir-phi(SMALL_UP)
| goal-dir-phi(UP)

⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(REAL_SOON)
| goal-arrival-time(SOON)
| goal-arrival-time(LONG_TIME)
| goal-arrival-time(DISTANT)

(a) Full grammar

⟨*goal-dir-theta-abs*⟩ ::= goal-dir-theta-abs(ZERO)
| goal-dir-theta-abs(SMALL)
| goal-dir-theta-abs(MEDIUM)
| goal-dir-theta-abs(LARGE)

⟨*goal-dir-phi-abs*⟩ ::= goal-dir-phi-abs(ZERO)
| goal-dir-phi-abs(SMALL)
| goal-dir-phi-abs(MEDIUM)
| goal-dir-phi-abs(LARGE)

⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(REAL_SOON)
| goal-arrival-time(SOON)
| goal-arrival-time(LONG_TIME)
| goal-arrival-time(DISTANT)

(b) Large grammar

⟨*goal-dir-max-abs*⟩ ::= goal-dir-max-abs(ZERO)
| goal-dir-max-abs(SMALL)
| goal-dir-max-abs(MEDIUM)
| goal-dir-max-abs(LARGE)

⟨*goal-arrival-time*⟩ ::= goal-arrival-time(NOW)
| goal-arrival-time(REAL_SOON)
| goal-arrival-time(SOON)
| goal-arrival-time(LONG_TIME)
| goal-arrival-time(DISTANT)

(c) Medium grammar

⟨*goal-seek-priority*⟩ ::= goal-seek-priority(ZERO)
| goal-seek-priority(LOW)
| goal-seek-priority(MEDIUM)
| goal-seek-priority(HIGH)

(d) Small grammar

Figure 4: Samples of grammars for the three-dimensional GoalSeek primitive task are shown. Each uses a different level of abstraction of the state space.

to the agent's current state. For example, using this state space, the CA-GS-RA composite behavior would only use dynamic priorities for the primitive behaviors CollisionAvoidance, GoalSeek, and RunAway to determine how to weight its sub-behaviors.

Note that these abstractions of the state space are only used by composite behaviors. Since primitive behaviors are responsible for producing low-level control actions, they still require the unabstracted state space relevant to their primitive task since. Furthermore, since monolithic controllers are also responsible for producing low-level control actions, they require the unabstracted joint state space of the composite task.

The ECJ [8] library was used to perform evolutionary runs and was modified to allow for the use of grammatical evolution. The evolutionary parameters used in the experiments are shown in Table 1. The fitness functions for both composite tasks are shown in Table 2. The task-dependent values used to calculate fitness were capped at maximum values and normalized before calculations were performed. The reward function was designed to be intentionally dense to provide as much fitness information as possible [18].

Environments were randomly generated and contained a random number of obstacles and "hazardous" objects, when appropriate. The environments were organized into ten folds of four environments each for use in cross-validation [3]. Evolved rulesets were evaluated in the training set of environments to determine the ruleset's fitness. Solutions that had the highest fitness on the training set, referred to as the "Best of Generation", were also evaluated in the validation set of environments. The solution with the highest fitness on the validation set over the entire run, referred to as the "Best of Run", was evaluated in the testing set of environments. This cross-validation was performed to determine the ability of the evolved rulesets to generalize to environments that were not encountered during training.

Agents were given a maximum of 1500 time steps in each environment to reach the goal location. Episodes ended

Table 1: Grammatical evolution parameters

Parameter	Value
Population	50
Generations	50
Codon size (bits)	8
Minimum codons	50
Maximum codons	200
Tournament size	7
Crossover type	One-point
Crossover probability	100%
Elite cloning probability	0%
Mutation probability per bit	1%

Table 2: Fitness function

Event	Value	Task
Collision	-150	CA-GS
		CA-GS-RA
Goal reached	$+150$	CA-GS
		CA-GS-RA
Goal distance	$-0.03 \times Dist_{Goal}$	CA-GS
		CA-GS-RA
Runaway	$-0.06 \times Str_{Run}$	CA-GS-RA

early when an agent collided with an obstacle or reached the goal location. Since the environments are randomly generated, an exact optimal performance value can not be easily calculated. However, an effective controller should be able to have a fitness of approximately 140.

7. RESULTS

Figure 5 shows the validation results of evolving fuzzy rulesets for both the two and three-dimensional CA-GS composite task over 30 runs. In each case, the monolithic rulesets were unable to gain any traction in accomplishing the task, while rulesets for the composite behaviors succeeded almost immediately. In fact, composite behavior rulesets in the three-dimensional task were consistently found in the initial, random population. Despite the significant differences in state information present in each of the abstracted state spaces, there was no statistically significant difference between the mean testing fitness of rulesets using the different abstracted state spaces for either the two or three-dimensional tasks.

Figure 6 shows the validation results of evolving fuzzy rulesets for both the two and three-dimensional CA-GS-RA composite task over 30 runs. In each case, the monolithic rulesets were again unable to gain any traction in accomplishing the task, while rulesets for the composite behaviors were far more successful. Composite behavior rulesets had difficulty generalizing in the two-dimensional CA-GS-RA task, but not in the three-dimensional case. Further investigation reveals that a number of the "Best of Run" rulesets were unable to generalize to the validation set of environments. The reason for this loss in generalizability

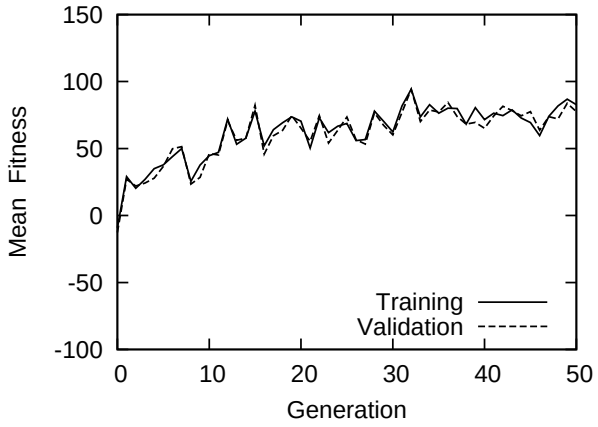


Figure 7: The training and validation fitness of "Best of Generation" fuzzy rulesets for the evolved GoalSeek primitive task fuzzy rulesets in two-dimensional environments are shown.

as compared to "Best of Run" rulesets from the other tasks is not known and merits further investigation. As with the CA-GS composite task, there was no statistically significant difference between the mean testing fitness of rulesets using the different abstracted state spaces in both the two and three-dimensional tasks.

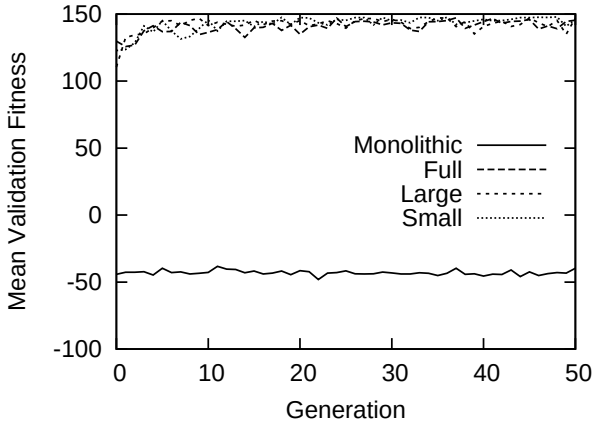
To demonstrate that effective rulesets using low-level control actions can be evolved for the GoalSeek primitive task, additional experiments were performed. Figure 7 shows the "Best of Generation" training and validation results of evolving fuzzy rulesets for the two-dimensional GoalSeek primitive task over 30 runs. While the fitness curves look erratic, the mean "Best of Generation" fitness on the training and validation sets were above 140 and the mean "Best of Run" fitness on the testing set was 139.

8. DISCUSSION

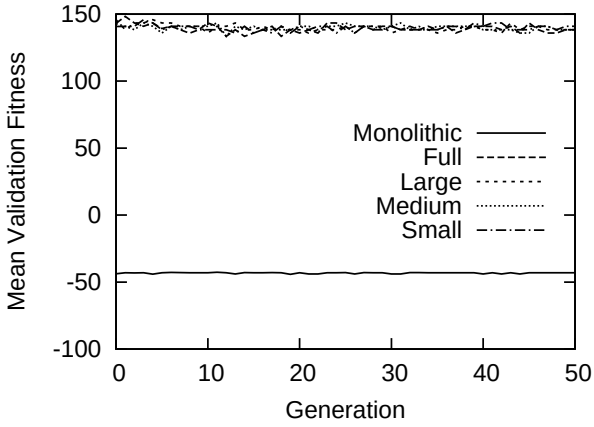
The success of the rulesets evolved for composite behaviors was entirely due to the fact that the rulesets were able to focus on the high-level coordination of effective primitive behaviors rather than low-level control. Even though the state-action space of for CA-GS-RA composite behaviors was far larger than that of the monolithic behavior, the evolutionary process quickly found rulesets that provided effective coordination while monolithic rulesets where unable to gain any traction on the problem.

An unexpected result was that there was no performance difference between the different abstractions of the state space for a given composite behavior. Although our initial hypothesis that state abstraction would not result in a loss in fitness held, we did not anticipate that there would be no difference in the rate at which effective rulesets were evolved. This discovery further lends credence to our assertion that the overall improved performance is due to the abstraction of the action space by focusing on high-level coordination rather than low-level control.

What is not reflected in Figures 5 and 6 is the computational effort required to develop the primitive behaviors used by the composite behaviors. The primitive behaviors used in these experiments were manually developed and were de-

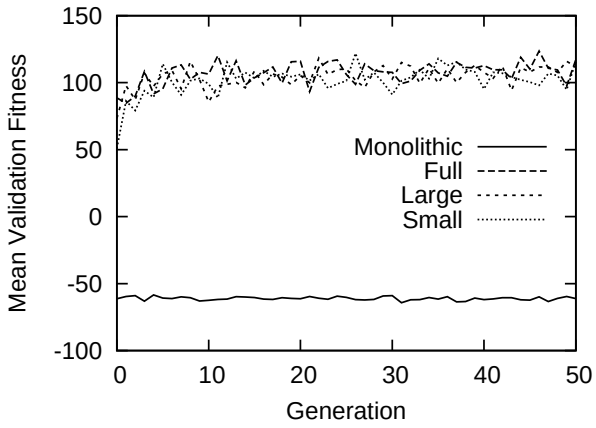


(a) 2-dimensional CA-GS

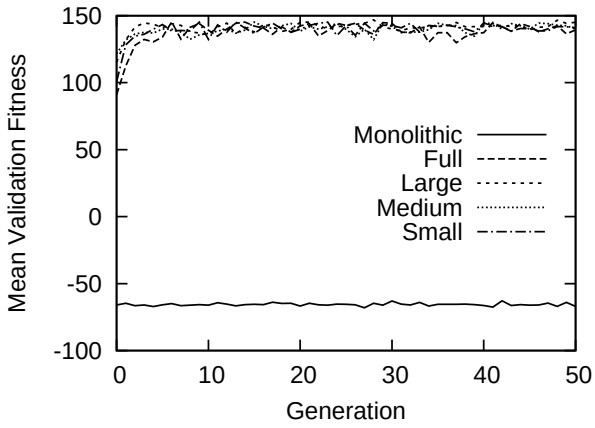


(b) 3-dimensional CA-GS

Figure 5: The fitness of "Best of Generation" fuzzy rulesets in the validation set of environments for the evolved CA-GS composite task fuzzy rulesets are shown.



(a) 2-dimensional CA-GS-RA



(b) 3-dimensional CA-GS-RA

Figure 6: The fitness of "Best of Generation" fuzzy rulesets in the validation set of environments for the evolved CA-GS-RA composite task fuzzy rulesets are shown.

signed to be effective, but not optimal. Since the primitive tasks associated with these behaviors are relatively simple, the process of manually creating these rulesets was straightforward. However, if manually creating the rulesets is impractical, the additional experiments in evolving a ruleset for the GoalSeek primitive behavior demonstrate that evolution can easily be applied to find effective rulesets. As a result, we believe that even when the computational effort of evolving rulesets from primitive behaviors is included, the evolution of rulesets for composite behaviors still has a clear advantage in the amount of computational effort required to create an effective controller.

While beneficial, there can be problems with using command fusion for agent control [12]. The most significant problem with command fusion is that the process of fusing two opposite actions can result in an action that is inappropriate for any primitive task. However, we were unable to find evidence of any such situations in the experiments described here. While it is possible that the implementation of the evaluation environment prevented such problems, we believe that the process of evolving a ruleset that weights the primitive behaviors was successful at avoiding such situations due to the low fitness of the weighting actions which can result in inappropriate actions.

9. CONCLUSIONS AND FURTHER WORK

The development of controllers for composite tasks is a difficult process. As composite tasks become more complex, the development such controllers quickly becomes impractical. A significant step in improving the practicality of developing controllers for those complex, composite tasks is to decompose the task into a collection of simple, primitive tasks. As a result, development process shifts from one of low-level control to one of high-level coordination. Our results demonstrate that this shift in focus promotes higher fitness and faster of evolution of fuzzy rulesets for the composite tasks used here. The added benefit of this approach is that existing behaviors can be reused for the development

of various composite tasks. In total, these results show that the change our shift in focus can make the development of controllers for complex, composite tasks farm more practical.

While existing primitive behaviors were able to be effectively reused by the evolved rulesets, the reused primitive behaviors were developed manually. The next step in this research is to evaluate and compare the reusability of rulesets for primitive behaviors developed through evolution with other approaches, such as manual development or other machine learning techniques. Furthermore, we would like to perform additional experiments on more complex, multi-agent composite tasks. We would also like to compare the overall fitness of evolved rulesets with those developed with reinforcement learning. Lastly, the unexplained loss of generalization for the two-dimensional CA-GS-RA task as compared to the other tasks requires more investigation.

10. REFERENCES

[1] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[2] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[3] P. R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1995.

[4] F. Hoffmann. Fuzzy behavior coordination for robot learning from demonstration. In *International Conference of the North American Fuzzy Information Processing Society*, pages 157–162, Banff, Canada, 2004.

[5] M. Humphrys. Action selection methods using reinforcement learning. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 135–144. MIT Press, Bradford Books, 1996.

[6] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller. I. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404–418, March 1990.

[7] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller. II. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):419–435, March 1990.

[8] S. Luke. ECJ 15: A Java evolutionary computation library. http://cs.gmu.edu/∼eclab/projects/ecj/, 2006.

[9] M. Nicolescu and M. J. Matarić. A hierarchical architecture for behavior-based robots. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 227–233, Bologna, Italy, July 2002. ACM Press.

[10] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[11] P. Pirjanian. Behavior coordination mechanisms – state-of-the-art. Technical Report IRIS-99-375, Institute for Robotics and Intelligent Systems, University of Southern California, October 1999.

[12] P. Pirjanian and M. J. Matarić. Multiple objective vs. fuzzy behavior coordination. In *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, volume 61 of *Studies in Fuzziness and Soft Computing*, chapter 10, pages 235–253. Springer-Phisica Verlag, 2001.

[13] G. Raju, J. Zhou, and R. A. Kisner. Hierarchical fuzzy control. *International Journal of Control*, 54(5):1201–1216, November 1991.

[14] N. Ramos, P. U. Lima, and J. M. Sousa. Robot behavior coordination based on fuzzy decision-making. In *ROBOTICA 2006 - 6th Portuguese Robotics Festival*, Guimarares, Portugal, 2006.

[15] C. Ryan, J. J. Collins, and M. O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, editors, *First European Workshop on Genetic Programming*, volume 1391, pages 83–95, Paris, 1998. Springer-Verlag.

[16] A. Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180–197, 1997.

[17] A. Saffiotti and Z. Wasik. Using hierarchical fuzzy behaviors in the robocup domain. In D. M. C. Zhou and D. Ruan, editors, *Autonomous Robotic Systems*, pages 235–262. Springer-Verlag, Berlin, DE, 2003.

[18] W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *International Conference on Robotics and Automation*, volume 4, pages 3404–3410. IEEE, 2002.

[19] N. Sprague and D. H. Ballard. Multiple-goal reinforcement learning with modular sarsa(0). In *International Joint Conference on Artificial Intelligence*, pages 1445–1447, 2003.

[20] S. Steven D. Whitehead. *Reinforcement Learning for the Adaptive Control of Perception and Action*. PhD thesis, University of Rochester, 1992.

[21] E. Tunstel. Fuzzy-behavior synthesis, coordination, and evolution in an adaptive behavior hierarchy. In *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, volume 61 of *Studies in Fuzziness and Soft Computing*, chapter 9, pages 205–234. Springer-Phisica Verlag, 2001.

[22] E. Tunstel, M. A. A. de Oliveira, and S. Berman. Fuzzy behavior hierarchies for multi-robot control. *International Journal on Intelligent Systems*, 17(5):449–470, 2002.

[23] P. Vadakkepat, O. C. Miin, X. Peng, and T. H. Lee. Fuzzy behavior-based control of mobile robots. *IEEE Transactions on Fuzzy Systems*, 12(4):559–565, 2004.