1

State and Action Abstraction in the Development of Agent Controllers

Brent E. Eskridge
Southern Nazarene University and University of Oklahoma
USA

Dean F. Hougen
University of Oklahoma
USA

1. Introduction

The development of controllers for intelligent agents given a simple task is relatively straightforward and even basic techniques can be used to develop such controllers. However, as agents are given multiple tasks, using basic techniques for developing effective controllers quickly becomes impractical. Since each task may require distinct state information local to that task only, the resulting state space for the agent overall is simply too large to effectively cover. Furthermore, since each task places different demands on the agent, an effective controller must find the correct balance to achieve it's overall task. While each of these difficulties presents significant problems individually, their combination can make the development of agent controllers for these complex tasks impractical.
The potential avenues of investigation for this problem are vast and the literature on the subject covers the spectrum of algorithms and perspectives. This chapter's focus is on *composite* tasks that are the result of a combination of, in general, **C**oncurrent, **I**nterfering, and **N**on-**E**pisodic (CINE) simple, or *primitive*, tasks. An example of a primitive task is COLLISIONAVOIDANCE. For our agents, COLLISIONAVOIDANCE is an abstract task that takes two inputs (relative direction to nearest obstacle and estimated time to contact with that obstacle assuming current speed) and provides two outputs (desired direction and desired speed). As such, it is simple enough that subdividing it further would not produce coherent subtasks, hence it is considered primitive. Note, however, that other formulations of COLLISIONAVOIDANCE that include lower-level information (such as the data from individual sensors or regarding all obstacles sensed) or actions (such as motor control values for individual components of the system) could be usefully subdivided into more primitive tasks such as TRACKOBSTACLE or PANCAMERA. If we combine COLLISIONAVOIDANCE with another simple task such as GOALSEEK, we arrive at a composite task. It is a composite task because the component tasks are coherent in and of themselves. Composite tasks composed of CINE primitive tasks have received comparatively little attention and are, for reasons discussed below, potentially one of the more difficult areas of focus. In contrast, many approaches focus on complex tasks that are composed of a series of sequential primitive tasks.

The challenge of correctly balancing CINE tasks, in addition to the complexity of the state and action spaces, must be addressed for the development of effective controllers for combinations of CINE tasks to be practical. Due to the variety of challenges, it is possible that a number of techniques must be combined to find an acceptable solution. As the complexity of the combined state and action space is a major challenge, it makes sense to consider state and/or action abstraction. For example, to coordinate the combination of COLLISIONAVOIDANCE and GOALSEEK it is important to know if a collision is imminent. If it is, the agent should probably try to avoid it.[1] If it isn't, the agent should head for the goal. If the answer is somewhere in between, the agent might give similar importance to both COLLISIONAVOIDANCE and GOALSEEK. Further, to determine if a collision is likely to occur soon, it isn't necessary to know whether the nearest obstacle is on the right or the left, only whether it is in front of the agent or off to either side. So state abstraction (in the form of ignoring right/left information) could reduce the amount of information used to decide the relative importance of COLLISIONAVOIDANCE and GOALSEEK. However, even if not all information is needed for all decisions, information cannot be simply discarded.[2] To continue the example, if the decision is that COLLISIONAVOIDANCE is at least somewhat important on this timestep, the agent will need to know which way to turn to avoid the obstacle.

The preceding example contained an explicit instance of state abstraction (using only the magnitude of the angle to the obstacle while ignoring its sign) but also an implicit instance of action abstraction. Note that the decision as to which way to turn was abstracted into determining how imminent a collision appears, then deciding how much importance to assign to each of the subtasks, then determining which way to turn. Such action abstraction is not necessary, however. In contrast, it would be possible to directly calculate turning angle as a function of obstacle and goal directions and distances.

Temporal abstraction, in the form of creating sequences of tasks, is also used in some systems to deal with composite tasks (Rohanimanesh & Mahadevan, 2002). However, such temporal abstraction will not work when the tasks are interfering as in our present research.

One approach that promotes the use of state and action abstraction while still allowing access to the unabstracted states and actions is the use of a hierarchical controller. A hierarchical controller leverages the hierarchical nature of the composite task by using smaller controllers responsible for each primitive task in the lowest level of the hierarchical controller and meta-controllers in the higher levels of the hierarchical controller to coordinate the lower-level controllers. Since low-level controllers are only responsible for a single primitive task, they do not need access to the full state space of the composite task, thus avoiding the combinatorial complexity of the composite task's state space. Furthermore, high-level meta-controllers are able to use state and action abstraction to simplify the state space since they merely coordinate the lower-level controllers that produce control actions instead of producing control actions themselves.

2. Adaptive Fuzzy Behavior Hierarchies

An example of a hierarchical approach to agent control, and the one used for the work described here, is an adaptive fuzzy behavior hierarchy (Tunstel, 2001). The hierarchy is orga-

[1] Only "probably" because if the goal is extremely close, the agent may prefer to reach the goal rather than avoid the obstacle.

[2] Some authors do deal with cases in which some variables are irrelevant and may be simply ignored by the controller with no loss of performance. We are assuming that all variables are relevant at least on some timesteps.

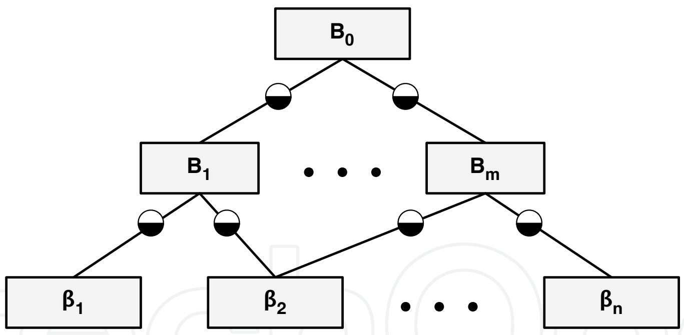


Fig. 1. Primitive behaviors (denoted β_i) are organized into a hierarchy and adaptively weighted by composite behaviors (denoted B_m) as described by Tunstel (2001) and redrawn here. The half-filled circles denote the weights and threshold values used to modulate behaviors.

nized using two types of behaviors. 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 are responsible for producing low-level control actions for the agent. Since each primitive behavior is responsible for a single primitive task, inputs to the behavior consist of only state information relevant to the associated primitive task.
Following the formulation of Tunstel (2001), let X and U be, respectively, the sets of all possible input and output values, or universes of discourse, for a primitive behavior with a ruleset of size M. Individual rules within the 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 represents the linguistic variables describing primitive task state information, such as direction or distance, and u represents linguistic variables describing motor command actions, such as steering direction and speed, $\tilde{A}_i$ and $\tilde{B}_i$ represent the fuzzy linguistic values corresponding to the variables x and u. The antecedent proposition "x is $\tilde{A}_i$" can be replaced with a compound antecedent using a conjunction or disjunction of propositions. The consequent "u is $\tilde{B}_i$" could also be replaced with a compound consequent. An an example of a low-level rule in a primitive behavior responsible for steering towards a goal could be:

$$\text{IF } \textit{goalDir} \text{ is } \texttt{LEFT} \text{ THEN } \textit{steerDir} \text{ is } \texttt{LEFT} \tag{2}$$

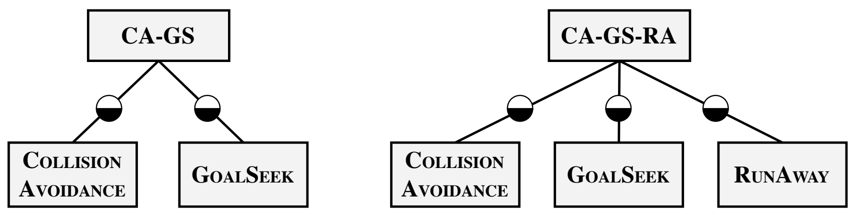
The output fuzzy set of each primitive behavior can be combined in a similar manner to produce a single output. However, since primitive behaviors often have conflicting goals, their actions often also conflict. A method of assigning different activation levels to different primitive behaviors could address these conflicts and allow an agent to accomplish its overall composite task. In an adaptive fuzzy behavior hierarchy, this is accomplished by means of *behavior modulation* in which the activation levels of primitive behaviors are adjusted, or adapted, based on the current overall state of the agent. These activation levels are referred to as degrees of applicability (DOA) and are assigned to primitive behaviors by a high-level *composite behavior*. Composite behaviors are only responsible for modulating other behaviors, either primitive or composite, and do not produce low-level control commands. For example, a composite behavior that is responsible for modulating a COLLISIONAVOIDANCE primitive behavior and a GOALSEEK primitive behavior could determine that since a collision is not imminent, the GOALSEEK behavior is more applicable and should have a `HIGH` activation, while the COLLISIONAVOIDANCE behavior should have a `LOW` activation. Composite behaviors are also implemented using fuzzy rulesets, but, since they produce outputs specifying activation levels and use different output fuzzy linguistic variables and values, their consequents differ from those found in primitive behaviors. Fuzzy rules within a composite behavior have the basic form:

$$\text{IF } x \text{ is } \tilde{A}_i \text{ THEN } \alpha \text{ is } \tilde{D}_i \tag{3}$$

where $\tilde{A}_i$ is defined as in Equation 1, α is the scalar activation level of a given behavior, and $\tilde{D}_i$ represents the fuzzy linguistic values (e.g. `LOW`, `MEDIUM`, `HIGH`) corresponding to the activation levels which are used to modulate a behavior. If a behavior is not explicitly given an activation level, it is automatically given a default activation of 0 and does not contribute to the overall output of the controller. Furthermore, threshold values can be used to provide cutoff points for a modulated behavior's activation (Tunstel, 1999). Just as with primitive behaviors, the output of a composite behavior is a fuzzy set. However, when defuzzified, the crisp values provide the current activation levels of lower-level behaviors, and not motor control commands. Using fuzzy rulesets to produce activation levels results in smooth transitions between different sets of activation levels in response to the changing state of the agent.
The activation level α_p of a modulated behavior p is used to calculate the weighted contribution of the behavior to the overall controller's output. The output of each primitive behavior can now be combined using their respective activation levels to weight their overall contribution to the action generated by the controller. The output of the entire behavior hierarchy is calculated as follows:

$$\tilde{\beta}_H = \biguplus_{p \in P} \alpha_p \cdot \tilde{\beta}_p \tag{4}$$

where $\tilde{\beta}_H$ is the output of the entire behavior hierarchy, P is the set of all primitive behaviors, $\tilde{\beta_p}$ is the output of the behavior p, and $\biguplus$ is the arithmetic sum of the fuzzy sets over all the primitive behaviors. The fuzzy output values are then defuzzified using the discrete form of Center-of-Sums defuzzification (Driankov et al., 1996).
Since composite behaviors only modulate lower-level behaviors using state information, composite behaviors do not require lower-level behaviors to provide any information to aid in the modulation 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 (Pirjanian & Matarić, 2001). The only restriction that an adaptive fuzzy behavior hierarchy places on modulated behaviors is that primitive behaviors produce a fuzzy set as output since fuzzy inferencing is used to combine their outputs into a single action.
It is important to note that since a composite behavior does not produce low-level control actions, it may not need the full joint state space of the composite task to provide effective behavior modulation. For example, it is possible that the direction of the closest collision is irrelevant when determining the modulation for a COLLISIONAVOIDANCE primitive behavior. It may be that only the estimated time until the collision is important. As a result, it may be possible to reduce the state information used by the modulation process in a composite behavior that provides comparable performance. A reduced state set such as this would provide significant benefits not only in reducing the complexity of the composite behavior's ruleset, but also in the effort required to develop the ruleset itself.
Although there are a number of ways to reduce the agent's state space (de Oliveira et al., 2003; Guyon & Elisseeff, 2003; Guyon et al., 2006; Raymer et al., 2000; Yang & Honavar, 1998), in the work presented here we use the approach where state information is converted into a more abstract form. This approach can result in either (1) fewer state variables or (2) no change in



(a) CA-GS behavior hierarchy (b) CA-GS-RA behavior hierarchy

Fig. 2. The behavior hierarchies for the CA-GS and CA-GS-RA single-agent composite tasks are shown.

the number of state variables but simpler extracted variable sets. For example, a composite behavior may not need to know the exact relative direction to an object and only requires the magnitude of the direction for effective control. In this example, the original *direction* state variable is extracted to a more simple representation where both `SMALL_LEFT` and `SMALL_-RIGHT` are abstracted to the same `SMALL` value. Since primitive tasks are, by definition, simple and straightforward, one can easily determine abstractions that may be beneficial
Although adaptive fuzzy behavior hierarchies have been shown to provide effective control, their implementation, as described by Tunstel, limits their application to two-level hierarchies. We have addressed this limitation by extending the architecture to properly function with hierarchies of arbitrary size (Eskridge & Hougen, 2009).

3. Problem Domains

For this work, a number of autonomous agent navigation problem domains were used. In each domain, an agent was given a complex task composed of N primitive tasks. In general, the primitive tasks used were active concurrently, interfered with one another, and were non-episodic (CINE). The only exception was the GOALSEEK task which terminated when an agent reached the goal location.
The composition of these N primitive tasks forms a composite task for which the agent should take an action at each timestep that maximizes the summed expected reward of each primitive task. An important aspect of this combination of primitive tasks is that an action that maximizes the reward for one primitive task could result in a penalty for another primitive task and, therefore, cause interference between the primitive tasks. While each primitive task had a (relatively) small state space, the state space for the composite task was the cross product of the state space for each primitive task: $S = S_1 \times S_2 \times \ldots \times S_N$. When this combined state space, referred to as the *joint state space*, was combined with the low-level action space, the resulting complexity made the traditional development of an effective controller impractical in some instances.

3.1 Single Agent Problem Domains

In the first single-agent composite task, an agent navigated towards a goal location while avoiding any obstacles in its path. This composite task, denoted CA-GS, was the combination of the COLLISIONAVOIDANCE and GOALSEEK primitive tasks. In the fuzzy behavior hierarchy for the CA-GS composite task, primitive behaviors were created at the lowest level for the COLLISIONAVOIDANCE and GOALSEEK primitive tasks (see Figure 2(a)). A composite

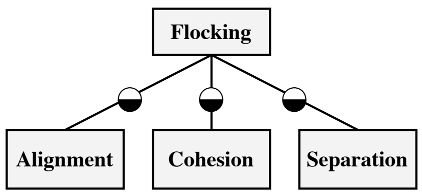


Fig. 3. The FLOCKING composite behavior composes the ALIGNMENT, COHESION, and SEPARATION primitive behaviors by using weights, denoted as half-filled circles.

behavior for the CA-GS composite task was then created at the level above the primitive behaviors and was responsible for weighting the primitive behaviors properly, given the current state of the agent with respect to the composite task as a whole.
In the second single-agent composite task, a third primitive task was added to the previous two. In this new primitive task, denoted RUNAWAY, the agent must avoid approaching too close to "hazardous" objects in the environment. The hazardous objects were not physical objects like obstacles with which the agent could collide, but instead represented areas that could be dangerous to the agent like areas of high-traffic or with difficult terrain. In COLLISIONAVOIDANCE, a penalty was only assessed if the agent actually collided with an obstacle. However, in RUNAWAY, an agent was penalized for simply being near hazards. The exact value of the penalty was dependent on the the distance to and strength of each hazard in the environment (a higher strength indicated a greater hazard). The new composite task was denoted CA-GS-RA. The fuzzy behavior hierarchy for the CA-GS-RA composite task was similar to that of the CA-GS hierarchy with the addition of the RUNAWAY primitive behavior (see Figure 2(b)).

3.2 Multi-Agent Problem Domains

In the first multi-agent composite task, a team of homogeneous agents must move together as a single unit, or *flock*, without explicit communication. This composite task, denoted FLOCKING, approximated the movement of flocks of birds or schools of fish (Reynolds, 1987; 1999) and was a combination of the ALIGNMENT, COHESION, and SEPARATION primitive tasks. In the ALIGNMENT primitive task, the agents were given the task of steering in the same direction and at the same speed as the rest of the team. In the COHESION primitive task, the agents were given the task of steering towards the other agents in the team in an effort to remain close to the team. Lastly, in the SEPARATION primitive task, the agents were given the task of steering away from other agents on the team which were "too close" in an effort to maintain a safe, minimum separation and prevent crowding. Agents relied only on the state information provided by sensors and did not communicate. Note that the goals of the ALIGNMENT and SEPARATION primitive tasks are diametrically opposed. Therefore, for FLOCKING to be successful, a policy that was able to effectively balance the two was necessary.
In the fuzzy behavior hierarchy for the FLOCKING composite task, primitive behaviors were created at the lowest level for the ALIGNMENT, COHESION, and SEPARATION primitive tasks (see Figure 3). A composite behavior for the FLOCKING composite task was then created at the level above the primitive behaviors and was responsible for weighting ALIGNMENT, COHESION, and SEPARATION properly, given the current state of the composite task.
In the next multi-agent composite task, we added the primitive task of COLLISIONAVOIDANCE to the FLOCKING composite task. In this task, each agent was tasked with avoiding collisions with other agents and with obstacles in the environment in addition to performing

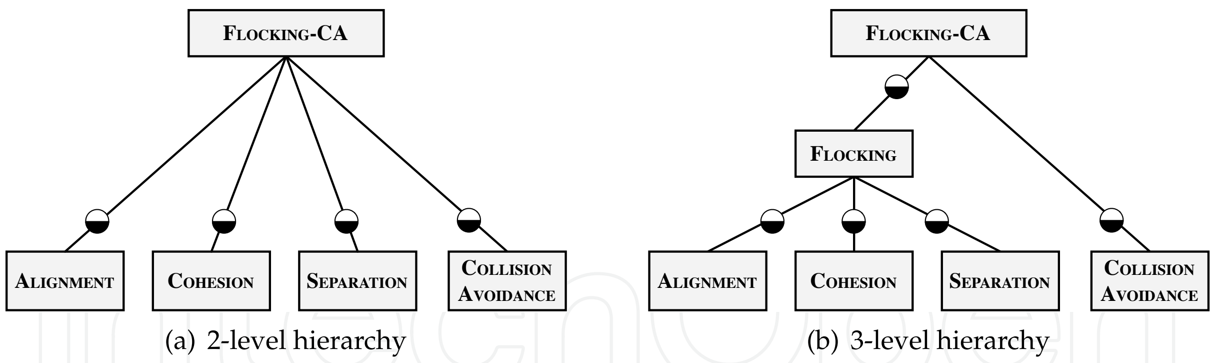
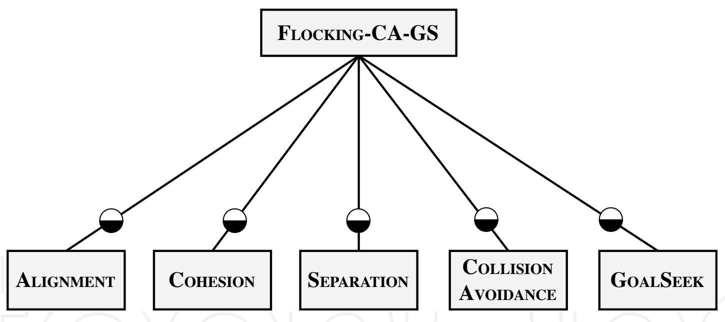


Fig. 4. The two alternatives for implementing the FLOCKING-CA composite behavior are shown. In the first, the FLOCKING-CA composite behavior composes the ALIGNMENT, COHESION, SEPARATION, and COLLISIONAVOIDANCE primitive behaviors. In the second, FLOCKING-CA composes the FLOCKING composite behavior with the COLLISIONAVOIDANCE primitive behavior.

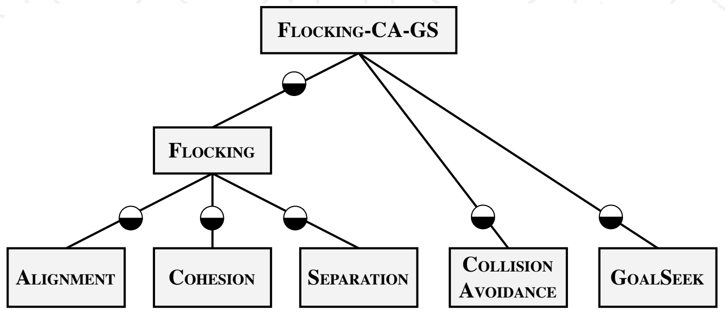
FLOCKING. The COLLISIONAVOIDANCE primitive task did not differentiate between collisions with other agents or obstacles. This new composite task, denoted FLOCKING-CA, presented the option of adding a new composite behavior, and, therefore, another level to the hierarchy (see Figure 4). Since the COLLISIONAVOIDANCE behavior was ignorant of the concept of a team and teammates, an argument can be made that it should be considered separately from the FLOCKING primitive behaviors. The use of an additional composite behavior at a higher level in the hierarchy not only simplified the action space of the FLOCKING-CA composite task, but it also had the potential to simplify the state space if the full joint state space of the composite task was not necessary for effective control. Furthermore, this hierarchical decomposition enabled existing policies for the FLOCKING task to be reused for the FLOCKING-CA task.

To further increase the complexity, the GOALSEEK primitive task was added to the previous composite task to create the FLOCKING-CA-GS composite task. While the COLLISIONAVOIDANCE primitive task could have actually assisted the task of FLOCKING by providing another means of avoiding collisions between members of the team in addition to the SEPARATION task, the addition of the GOALSEEK complicated the FLOCKING task. As with the FLOCKING-CA task, the clear separation between the FLOCKING composite task and the COLLISIONAVOIDANCE and GOALSEEK primitive tasks offered the potential for creating a separate composite behavior for coordinating the respective behaviors.

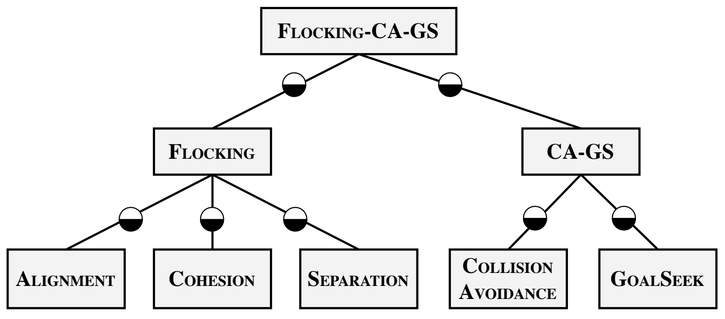
In the last multi-agent composite task, the RUNAWAY primitive task was added to the FLOCKING-CA-GS composite task to create the FLOCKING-CA-GS-RA composite task. As with the previous two composite tasks, a separate composite behavior which coordinated the FLOCKING composite task and the COLLISIONAVOIDANCE, GOALSEEK, and RUNAWAY primitive behaviors was possible. While the same potential benefits existed, the large number of primitive tasks which must be coordinated had the potential to exaggerate the results. As is described in Section 6, it was at this point that the complexity of the composite task's joint state space became too large for traditional methods of developing effective controllers to be practical and alternative methods were required. The use of this composite task represented the upper limit of complexity used in this work.



(a) 2-level hierarchy



(b) 3-level hierarchy reusing one composite behavior



(c) 3-level hierarchy reusing two composite behaviors

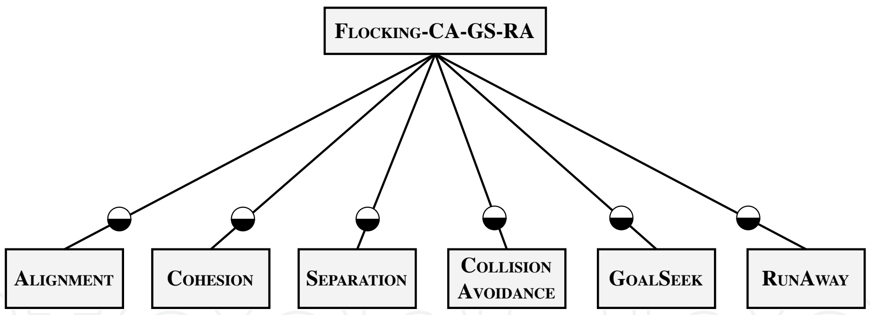
Fig. 5. Three alternatives for implementing the FLOCKING-CA-GS composite behavior are shown. Each is similar to the corresponding hierarchy for the FLOCKING-CA composite task with the addition of the GOALSEEK primitive behavior

4. Development of Controllers

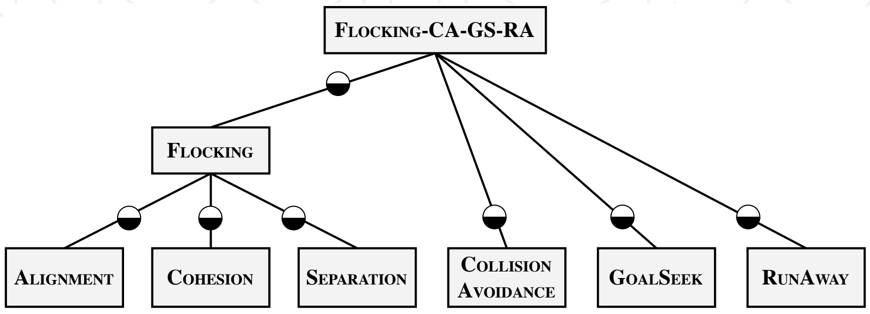
A standard approach to such complex tasks is to combine primitive tasks into a single composite task. A policy is then developed for the entire composite task, effectively developing a single policy responsible for addressing each primitive task and the coordination between them (see Figure 7(a)). The problem with this monolithic approach is that development of even the simplest composite task can be impractical due to the curse of dimensionality.

4.1 Modular Reinforcement Learning

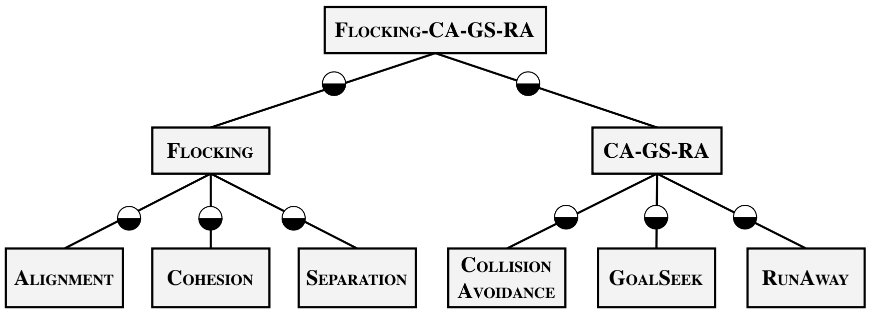
Humphrys (1996) and Karlsson (1997) independently describe a reinforcement learning algorithm that is appropriate for the CINE types of problems under study here. In this algorithm, commonly referred to as *modular* reinforcement learning (Bhat et al., 2006; Sprague & Ballard,



(a) 2-level hierarchy



(b) 3-level hierarchy reusing one composite behavior

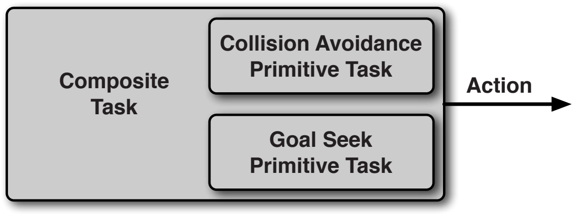


(c) 3-level hierarchy reusing two composite behaviors

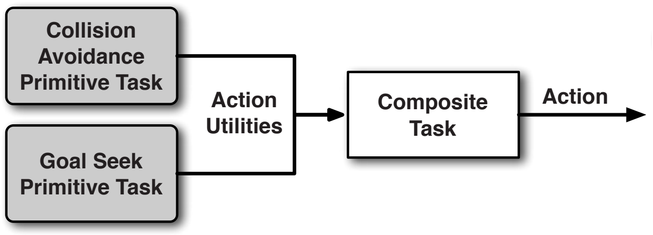
Fig. 6. Three alternatives for implementing the FLOCKING-CA-GS-RA composite behavior are shown. Each is similar to the corresponding hierarchy for the FLOCKING-CA-GS composite task with the addition of the RUNAWAY primitive behavior

2003), a policy for each active primitive task is learned simultaneously using the state information and rewards local only to the task (see Figure 7(b)). At each time step, the policy for each subtask provides the action selection mechanism with a utility value for each possible action. This utility value is calculated using the value of taking a particular action from a given state, referred to as a *Q-value*, and is often simply the Q-value itself. These utilities are then used by the action selection mechanism to choose the action that the agent will take. The approach used in this work to choose the action, called the "greatest mass," simply chooses the action with the highest utility, or sum of Q-values, across all the primitive task policies (Karlsson, 1997).

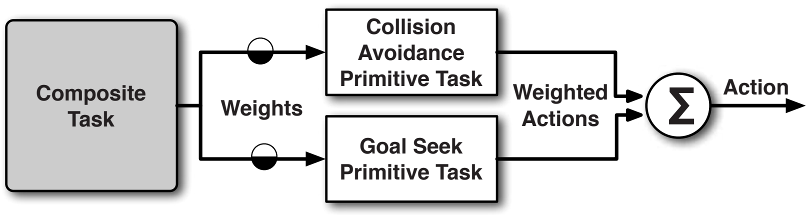
Q-learning should not be used in learning the primitive task policies since it is off-policy and assumes the optimal policy will be followed (Watkins & Dayan, 1992). One cannot assume that the optimal policy will be followed for modular reinforcement learning since primitive



(a) Monolithic Reinforcement Learning



(b) Modular Reinforcement Learning



(c) Composite Reinforcement Learning

Fig. 7. A comparison of the different algorithms discussed in this work is shown using the COLLISIONAVOIDANCE-GOALSEEK composite task. The shaded tasks are where policy learning occurs in each algorithm. The half-filled circles denote the weights used to compose actions from primitive task policies for composite reinforcement learning.

task policies must share control of the agent (Russell & Zimdars, 2003; Sprague & Ballard, 2003). As a result, an on-policy learning method, such as the Sarsa algorithm, should be used (Rummery & Niranjan, 1994).

4.2 Composite Reinforcement Learning

As will be discussed in Section 6.4, experiments demonstrate that modular reinforcement learning does not perform well in the CINE tasks under study in this chapter. In light of this result, we introduce a modified reinforcement learning approach called *composite reinforcement learning* (see Figure 7(c)) which can be used to learn effective control policies for composite tasks built using CINE primitive tasks. Composite reinforcement learning leverages the architecture of the adaptive fuzzy behavior hierarchy to significantly improve the rate at which effective control policies are learned. Unlike modular reinforcement learning, composite reinforcement learning does not attempt to learn policies for the primitive tasks simultaneously. Instead, composite reinforcement learning learns an effective control policy for a given composite behavior only and reuses existing implementations of lower-level behaviors. These reused lower-level behaviors are viewed as black boxes and are modulated by the policy being learned. Therefore, instead of learning low-level motor control actions, composite reinforcement learning learns high-level modulation (i.e., weighting) actions on the lower-level behaviors. The reinforcement learning algorithm itself is largely unmodified except that the concept of an action has changed. The policy's actions are now weighting actions and after the policy's action has been taken, the lower-level behaviors are executed and the overall action of the agent is computed. The composite task policy being learned then determines the total reward and updates the relevant Q-value.

Note that the Q-values used by the learned policy are associated only with the modulation actions and not with the actions taken by the lower-level behaviors. While this means that the maximum performance of the learned policy is dependent on the performance of the lower-level behaviors in their associated tasks, in practice this appears to not present problems (see Section 6) and offers many benefits over other approaches, such as modular reinforcement learning.

One of the most significant benefits is the abstraction of the action space into high-level "meta-actions." As a result, the reinforcement learner is not required to learn the entire composite task from scratch. Rather, it only needs to learn how to best coordinate lower-level behaviors to accomplish the composite task. In a related benefit, existing behaviors can potentially be reused *without modification* by the learned policy and without specific requirements on their implementation method. Composite reinforcement learning does not require reused behaviors provide any information to aid in the learning or control process (e.g., Q or utility values). As a result, individual behaviors can be developed in isolation, simplifying the development process, using the method most appropriate for the task.

Furthermore, since the action space has been abstracted away from low-level motor control actions, it may be possible to aggressively abstract the agent's state for use in the composite behavior without the corresponding performance penalties commonly associated with perceptual aliasing (Whitehead, 1992). This is especially significant in light of our interest in directly comparing the effects of state and action abstraction on a controller's performance and learning rate. Note that this abstraction of the state only occurs in the composite behaviors; primitive behaviors still access the unabstracted state associated with the relevant primitive task to produce control actions.

While the idea of abstracting the action space into meta-actions is not novel and many hierarchical reinforcement learning approaches use it extensively (Dietterich, 2000; Konidaris & Barto, 2007; Rohanimanesh et al., 2004), our formulation of an action is novel. Since most approaches focus on episodic and non-interfering tasks, meta-actions in these approaches represent temporally extended sequences of actions. When a meta-action is executed, the meta-action assumes control of the agent either for the entire sequence of actions or until an event causes the high-level policy to re-examine the agent's state. In contrast, the meta-actions used by composite reinforcement learning are taken every timestep and represent the coordination of lower-level behaviors for that timestep only. In general, no one behavior is given complete control of the agent's actions.

5. Experiments

To evaluate the effects of state and action abstraction on the process of developing controllers for composite tasks, a series of experiments were performed in which controllers were automatically developed for each primitive and composite task using reinforcement learning and grammatical evolution (see Section 3). Experimental runs were evaluated using two different metrics. The first metric used was the best generalized performance of the agent controllers. This generalized performance was determined by executing agent controllers in environments that were different than the ones used in their development. The controller's performance was the mean undiscounted, total reward of the agent in all the environments. The second metric used was the computational effort used to develop the controllers and used the number of updates to the Q-values as the measure.

5.1 Evaluation Environments

For each composite task, forty environments were randomly generated. Agents were given random positions and orientations within a specified region of the environment. If a goal location was required, it was randomly placed within a specified distance interval from the agent(s). If obstacles were required, a random number of obstacles were generated and given random positions in an area surrounding the agent(s) and goal location. The same procedure was followed for hazardous objects, if required. These forty environments were organized into ten folds of four environments each for use in cross-validation (Cohen, 1995). Eight folds were used as a training set, one fold was used as a validation set, and a final fold was used as a testing set. Both validation and testing sets were used to evaluate the generalizability of the learned controller.

Each experiment consisted for forty individual runs initialized with a different random seed. Four experimental runs for each of the ten folds were performed. The same set of environments was shared between all experiments for a given primitive or composite task, while folds had different sets of training, validation, and testing environments. For example, all experiments using the two-dimensional CA-GS composite task shared the same set of environments, regardless of how the controller was developed or the architecture it used.

Agents were given a maximum of 1,500 time steps in each environment which constituted a single training episode. This was ample time for even the most risk-averse agent(s) to reach the goal location, if applicable, or to gain sufficient experience in the environment. While most of the primitive tasks used are non-episodic (the exception being the GOALSEEK primitive task), training was broken into episodes as a consequence of the nature of the evaluation environments and the primitive tasks themselves. Since the environments were unbounded, it was possible that in exploring the state space, agents could wander away from the finite number of obstacles or the other agents on the team and never have a realistic opportunity to return to the more "interesting" states of the environment. Furthermore, since much of this work is intended to operate on real robots, agent collisions warranted early termination of a training episode. Training episodes also ended early when an agent or the team of agents reached the goal location since the GOALSEEK task is inherently episodic.

5.2 State Space Abstraction

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, four different levels of abstraction of the agent's state space were used when learning composite behaviors. These were used to evaluate how abstractions affected both the rate at which effective composite behaviors were learned and their quality. Table 1 details the effects of each abstraction level on the state information for a composite behavior using the GOALSEEK primitive task.

Full This state space represents the original, joint state space of all the primitive tasks used in the composite task without any abstraction and acts as a baseline for comparison.

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

Abstraction Level	State Information	States	Total States
Full	Goal arrival time	5	175
	Goal direction Θ	7	
	Goal direction Φ	5	
Large	Goal arrival time	5	125
	Goal direction $\|\Theta\|$	5	
	Goal direction $\|\Phi\|$	5	
Small	Goal seek priority	5	5

Table 1. The different state abstractions used in the development of a composite behavior using the GOALSEEK primitive task are shown.

of the angle, such as SMALL. State information not describing a direction remains unchanged.

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. This dynamic priority represented the task's determination of its applicability 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. While this level of abstraction may appear to be too extreme, we have previously shown that rule-sets using dynamic priorities can be developed which have similar performance to those using the **Full** state space (Eskridge & Hougen, 2006).

Minimal In this state space, the dynamic priorities from the **Small** state space were again used. However, instead of using the dynamic priorities for every primitive behavior, only the priorities of behaviors directly weighted by a composite behavior were used. For example, the FLOCKING-CA composite behavior would only use the dynamic priorities of the FLOCKING and COLLISIONAVOIDANCE sub-behaviors.

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

5.3 Reward Functions

The reward functions for each primitive task as used in the development of composite tasks are shown in Table 2. Except for the terminal events of a collision or reaching the goal location, each reward was given per timestep. The reward values were developed with the maximum number of timesteps in mind and ensured that the total undiscounted reward did not create a bias towards controllers which "minimized the pain" by causing a collision as quickly as possible. While some portions of the reward function were not required (e.g., the goal distance penalty), they make the reward function more "dense" and act as progress estimators by allowing learning to make the most of each experience (Matarić, 1997; Smart & Kaelbling, 2002). While the addition of these unrequired components may have biased policies away

Primitive Task	Description	Value
CollisionAvoidance	Collision event	-150
GoalSeek	Goal reached event	150
	Goal distance penalty	$-0.03 \times Dist$
RunAway	RunAway strength penalty	$-0.06 \times Str$
Alignment	Velocity heading difference penalty	$-0.02 \times \Delta Dir$
Cohesion	Position error penalty	$-0.04 \times Dist$
Separation	Separation strength penalty	$-0.02 \times Str$

Table 2. The reward functions used in developing controllers for composite tasks for each primitive task are shown. Note that while most rewards were given at each timestep, rewards for the terminal events of a collision and reaching the goal location are one-time rewards.

from the best solution, the benefit of allowing learning to make the most of each learning experience outweighed the potential problems in complex tasks such as the ones discussed in this chapter.

For the Flocking composite task, a survival reward of 0.09 was given for each timestep in which the agents were active, in addition to the rewards given by the primitive tasks themselves. This served to explicitly reward the agents for avoiding collisions with other agents and continuing to flock. In single agent environments, the agent merely needed to reach the goal for the reward to be given. In multi-agent environments, the reward for reaching the goal location was only awarded if the mean position of the team was within a specified distance to the goal. This served to explicitly reward agents that reached the goal with the other agents in the team and not agents that left their team to reach the goal faster.

Due to the complexity of the tasks and the randomness of the environments, an optimal performance value for each task would be prohibitive to calculate. As a result, the performance of learned controllers cannot be compared to the performance of an optimal controller. However, based on an understanding of the experimental configuration and experience with the tasks themselves, we can identify the approximate mean performance values one would expect from an effective controller.

5.4 Reinforcement Learning Configuration

The Sarsa reinforcement learning algorithm was used to learn policies for primitive and composite tasks. To speed learning, the replacing eligibility traces version of Sarsa, referred to as Sarsa(λ), was used (Sutton & Barto, 1998). The parameters used are shown in Table 3. Since the state-action space for many of the experiments performed precluded tabular storage of the state-action values, or Q-values, neural networks were used to approximate Q-values. The neural networks consisted of a single hidden layer in which the number of nodes was a function of the number of input nodes. Unlike previous work, we found that a relatively large number of hidden nodes (1.5 times the number of input nodes) were required for policies operating in our environments (Rummery & Niranjan, 1994). Previous work in the field has concluded that using a single network to approximate all the Q-values can result in unintentional modifications of the Q-values for actions other than the one chosen by the learning algorithm (Lin, 1993; Rummery & Niranjan, 1994). As a result, a separate network for each action was used in an effort to isolate the Q-values of each action.

Parameter	Value
Learning rate (α)	0.01
Discount factor (γ)	0.99
TD decay (λ)	0.25
Exploration (ϵ)	0.01
NN weight range	$[-0.25 : 0.25]$
NN momentum	0.01
NN hidden nodes	$1.5 \times N_{input}$

Table 3. Reinforcement learning parameters

The effects of a multi-agent environment further complicates learning as it makes the environment non-stationary (Claus & Boutilier, 1998). To simplify the process as much as possible, we chose to use the naïve approach in which all agents used and updated the same set of Q-values, or, more specifically, the same set of neural networks approximating Q-values. Experiments have shown that this form of cooperation does not impede learning and can even improve the learning rate (Crites & Barto, 1998; Tan, 1993).
While there are techniques for using fuzzy logic with reinforcement learning (Berenji, 1992; Er & Zhou, 2006; Glorennec & Jouffe, 1997; Jouffe, 1998), experience has shown that such modifications can increase the time needed to learn effective policies. Since fuzzy logic is not required to implement the behaviors (Tunstel, 2001), it was not used for behaviors developed using reinforcement learning. However, primitive behaviors that were developed manually and reused for development of composite behaviors did use fuzzy logic. While there are plans to use fuzzy logic with reinforcement learning (see Section 7), we were able to gather conclusive results without its use.

6. Results and Analysis

Figures showing the results of experimental runs reflect the mean performance of controllers on the validation set of environments. As stated in Section 5.1, environments were organized into ten folds for use with cross-validation. Each experiment consisted of four runs for each of the ten fold combinations for a total of 40 runs.

6.1 Developing Single-Agent, Composite Task Controllers

Figures 8 and 9 depict the results of learning for the CA-GS and CA-GS-RA composite tasks, respectively. For controllers developed using the **Full** and **Large** abstraction levels, composite reinforcement learning was able to achieve high performance within just a few updates for both tasks. However, the **Small** abstraction level was unable to converge to an effective policy in either task, although effective policies were learned. While monolithic reinforcement learning was able to learn an effective policy, results of a randomized two-way ANOVA test demonstrates that there was a statistically significantly difference between the rate at which effective controllers were learned between monolithic and composite reinforcement learning at the 95% confidence level. Modular reinforcement learning was unable to converge to an effective policy. It was able to achieve moderate success early and learned a number of effective policies, but was unable to converge to one.

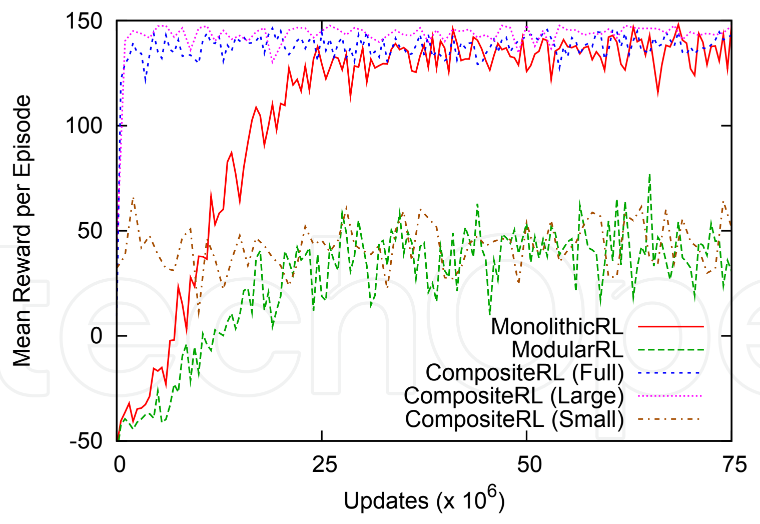


Fig. 8. Reinforcement learning results on the validation set environments for the CA-GS composite task are shown.

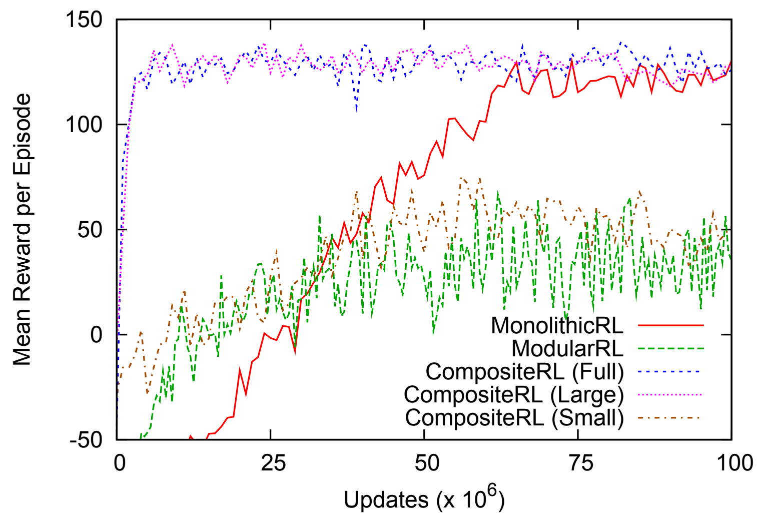


Fig. 9. Reinforcement learning results on the validation set environments for the CA-GS-RA composite task are shown.

6.2 Developing Multi-Agent, Composite Task Controllers

Figure 10 depicts the results of using reinforcement learning to learn a policy for the FLOCKING composite task. Controllers developed using the adaptive fuzzy behavior hierarchy and composite reinforcement learning had statistically significantly higher performance than those developed using either monolithic or modular reinforcement learning at the 95% confidence level. Furthermore, controllers using the **Small** abstraction level did not exhibit the poor performance previously seen in the single agent tasks. In fact, controllers using the **Small** abstraction level had statistically significantly better performance than all other controllers in the testing set environments at the 99% confidence level as determined by the paired Student's t-test using the Bonferroni adjustment.

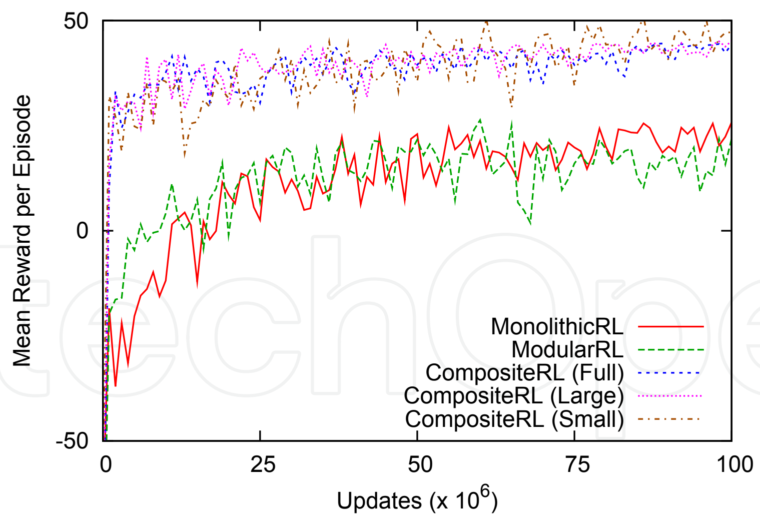


Fig. 10. Reinforcement learning results on the validation set environments for the FLOCKING composite task are shown.

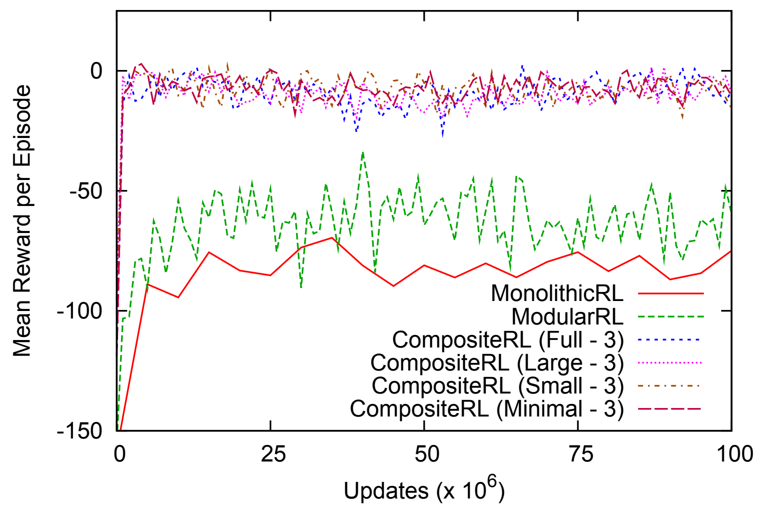


Fig. 11. Reinforcement learning results on the validation set environments for the FLOCKING-CA composite task are shown. Due to storage constraints, results for monolithic reinforcement learning experimental runs used fewer checkpoints than other experimental runs.

Figure 11 depicts the results of using reinforcement learning to learn a policy for the FLOCKING-CA composite task. Again, controllers that used the adaptive fuzzy behavior hierarchy were learned faster than those that didn't use the hierarchy and had higher performance. Note, that these controllers used the three-level hierarchy depicted in Figure 5(c). Controllers developed using modular reinforcement learning performed statistically significantly better than those developed using monolithic reinforcement learning at the 99% confidence level, but were unable to generalize to the full range of environments present in the validation or testing sets.
Not shown in the figure are the results for controllers using a two-level adaptive fuzzy behavior hierarchy. Even with the use of the adaptive fuzzy behavior hierarchy, the significantly

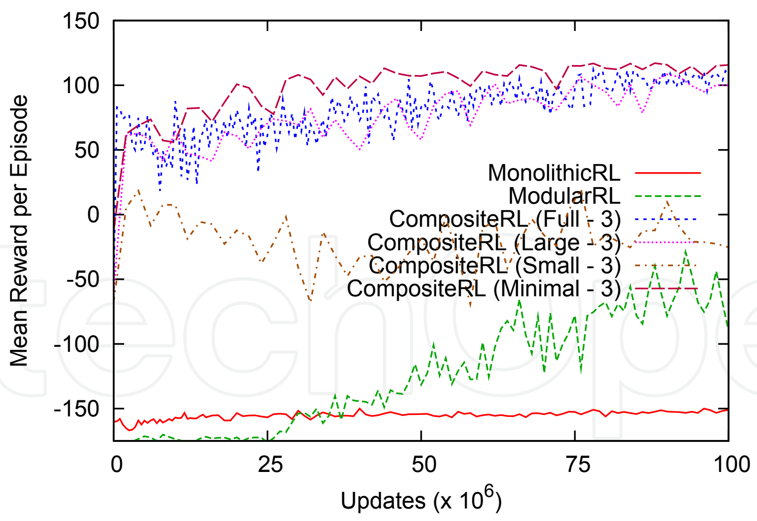


Fig. 12. Reinforcement learning results on the validation set environments for the FLOCKING-CA-GS composite task in two dimensions are shown. The "3" denotes controllers using a three-level behavior hierarchy. The results of controllers using two-level hierarchies are not shown to improve clarity.

larger action space of the 2-level hierarchy negated any benefits that the hierarchy offered. While the hierarchy and the reuse of existing primitive behaviors meant that it did not need to learn the low-level actions needed to accomplish FLOCKING-CA, the size of the state-action space was simply too large to quickly find an effective policy. Furthermore, the increased complexity resulted in almost a four-fold increase in the wall-clock time required to learn and update Q-values for the two-level hierarchies over that of the three-level hierarchies.

Figure 12 depicts the results of using reinforcement learning to learn a policy for the FLOCKING-CA-GS composite task. Just as in the previous experiments, controllers learned using composite reinforcement learning and using the adaptive fuzzy behavior hierarchy had a statistically significantly higher "best-of-run" performance than monolithic reinforcement learning at the 99% confidence level as monolithic reinforcement learning was even unable to learn how to simply avoid a collision. Controllers developed using modular reinforcement learning were able to gain some traction in learning an effective controller, but were unable to converge to an effective policy. Note that just as in the single-agent composite tasks, controllers developed using the **Small** abstraction level performed statistically significantly worse than other controllers developed using the adaptive fuzzy behavior hierarchy, including those using the **Minimal** abstraction level which also uses adaptive priorities.

This is the first set of results in which a difference between controllers using the different abstraction levels can be observed. Results of two-way randomized ANOVA tests show that controllers using the **Minimal** abstraction level were able to more statistically significantly achieve consistently higher performance than controllers using the other abstraction levels at the 95% confidence level. However, controllers using the **Full** abstraction level were able to learn a policy at some point during learning that had statistically significantly better performance than controllers using the **Minimal** abstraction level at the 99% confidence level.

Figure 13 depicts the results of using reinforcement learning to learn a policy for the FLOCKING-CA-GS-RA composite task. These results detail, for the first time, a clear separation in the performance of controllers using the various abstraction levels. Randomized two-

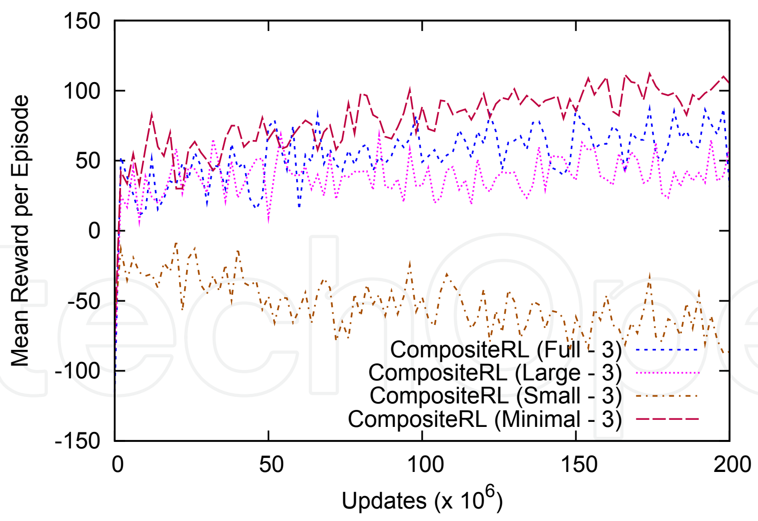


Fig. 13. Reinforcement learning results on the validation set environments for the FLOCKING-CA-GS-RA composite task in two dimensions are shown.

way ANOVA tests show that, like the FLOCKING-CA-GS composite task, controllers using the **Minimal** abstraction level were able to more statistically significantly achieve consistently higher performance than controllers using the other abstraction levels at the 95% confidence level, but there was no statistically significant difference in the "best-of-run" testing fitness between controllers using the different abstraction levels. Controllers using the **Full** abstraction level had a higher mean reward per episode than controllers using the **Small** abstraction level at the 95% confidence level. Monolithic and modular reinforcement learning were not used to learn controllers due to their consistent poor performance in the simpler, multi-agent composite tasks.

6.3 Analysis

These results demonstrate that controllers using adaptive fuzzy behavior hierarchies significantly outperformed controllers with other architectures in terms of performance, rate of development, or both. While there are many reasons for this improvement, we believe that the central reason for this improvement is the action abstraction that adaptive fuzzy behavior hierarchies provide. Note that in the CA-GS-RA task, the action space for monolithic controllers consisted of only two variables: the change in speed and the change of direction. However, for controllers using composite behaviors, the action space of the composite behavior consisted of three variables: the weights for the COLLISIONAVOIDANCE, GOALSEEK, and RUNAWAY primitive behaviors. Despite this increased action space, controllers using composite behaviors were developed significantly faster and with significantly better performance. This is due to the fact that although the action space of the composite behavior was larger, it consisted of high-level, abstracted "meta-actions" instead of the more complex, low-level control actions. As has been previously discussed, although other approaches also use the concept of "meta-actions," the action abstraction used in this chapter is fundamentally different since the primitive tasks used were, in general, concurrent, interfering, and non-episodic.
We can conclude that action abstraction is more useful than state abstraction by comparing the performance of controllers used for the complex, multi-agent composite tasks. In these tasks, the controller could be designed with a hierarchy that used a single composite behavior or a

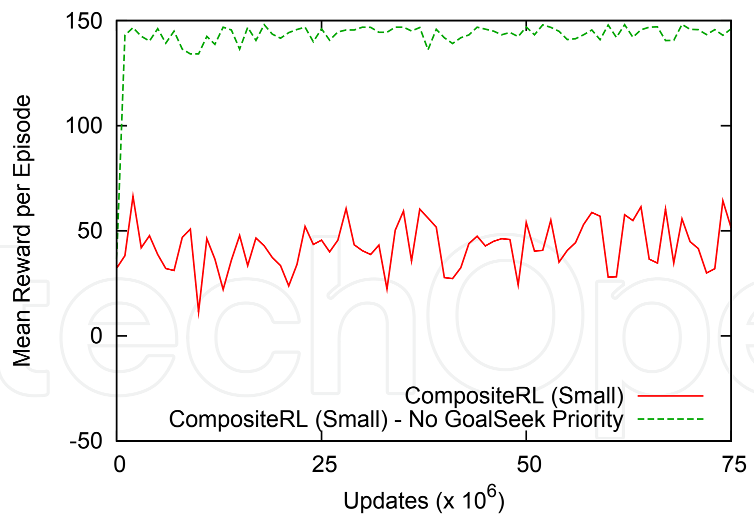
hierarchy that made use of multiple composite behaviors in different hierarchical levels. An example is the three alternatives for implementing the FLOCKING-CA-GS composite behavior shown in Figure 5. Results show that even with significant state abstraction, controllers using the two-level hierarchy shown in Figure 5(a) were only able to achieve mediocre performance due to the sheer size of the action space. However, effective controllers using the three-level hierarchy shown in Figure 5(c) were able to be developed without any abstraction of the agent's state. It is important to also note that using reinforcement learning to learn controllers using the two-level hierarchy in the FLOCKING-CA-GS task took significantly more computational time than those using the three-level hierarchy. This is due to the fact that in the Sarsa algorithm, the Q-value for each of the 3,125 possible actions must be calculated at each time step. Over the course of the entire experimental run, this resulted in almost a four fold increase in the wall clock time required to develop controllers using a two-level hierarchy over those using a three-level hierarchy.

A surprising result is that for many of the composite tasks evaluated, there was no statistically significant difference between controllers using the various state abstraction levels with the exception of the **Small** abstraction level. While the use of abstraction can significantly reduce the size of the state space, the possibility of over-abstracting the state space and negatively impacting the controller's performance exists. However, in general, this was not observed.

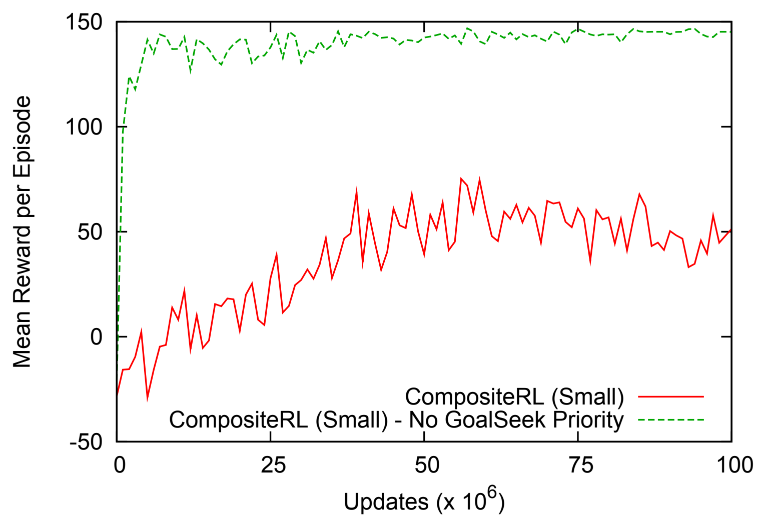
Not reflected in these results is the computational effort required to develop the primitive behaviors used by the controllers using adaptive fuzzy behavior hierarchies. The primitive behaviors used in these experiments were manually developed and were designed to be effective, but not optimal. Since the primitive tasks associated with these behaviors are relatively simple, the process of manually creating these rulesets was straightforward. However, if manually creating the behaviors is impractical, results show that effective policies for the single-agent behaviors can be easily learned. Even when the the computational effort of creating the primitive behaviors is included, developing controllers that use adaptive fuzzy behavior hierarchies is still far more beneficial and practical than the other approaches evaluated.

The performance of controllers developed with composite reinforcement learning and using the **Small** abstraction level are of particular interest as it may indicate that we have over-abstracted the state space. In many of the tasks used, an effective policy could be learned using the **Small** abstraction level, but reinforcement learning was unable to *converge* to an effective policy. While there are a number of potential reasons for this lack of convergence, we do not believe this is an inherent problem with the dynamic priorities used by the **Small** abstraction level since reinforcement learning converged to effective policies for the FLOCKING task using the **Small** abstraction level. There were a number of composite tasks for which controllers using the **Small** abstraction level provided effective control. We believe the root cause of the problem lies with the dynamic priority generated for the GOALSEEK primitive task. In each composite task using GOALSEEK, the development of controllers using the **Small** abstraction level was unable to converge.

To evaluate our hypothesis, we altered the **Small** abstraction level by replacing the GOALSEEK adaptive priority with the GOALSEEK state information from the **Full** abstraction level (i.e., the arrival time at and direction to the goal location) and performed a number of the experiments again. The results of these experiments using the altered **Small** abstraction level were compared to the results using the original **Small** abstraction level. The results for the single-agent composite tasks are shown in Figure 14. Controllers that did not use the GOALSEEK adaptive priority were learned faster, had higher performance, or both when compared to controllers which did use the priority. Although not shown, the performance of controllers



(a) CA-GS



(b) CA-GS-RA

Fig. 14. Results on the validation set environments for the CA-GS and CA-GS-RA composite tasks comparing different approaches for the **Small** abstraction level.

using the modified **Small** abstraction level were comparable to the performance of the controllers learned using the other abstraction levels. This indicates that the GoalSeek adaptive priority was somehow the contributing factor to the low performance in the previous experiments. Exactly why the GoalSeek adaptive priority causes such problems is unknown and worthy of future investigation.

6.4 Discussion

As these results demonstrate, neither monolithic or modular reinforcement learning produce effective controllers as the complexity of the composite task increases. For monolithic reinforcement learning, the reason for this inability to gain traction on the problem is that it simply cannot cope with the complexity of the state space. However, this simple explanation does not work for modular reinforcement learning since it was explicitly designed to handle such complexity. While the exact reason for modular reinforcement learning's inability to produce effective control policies is unknown and the subject of future research, our work has illuminated a number of problem areas that could affect its use in developing controllers for the type of tasks under study in this chapter.

First, modular reinforcement learning makes the implicit assumption that rewards are consistent across all the primitive tasks which complicates the process of learning the policies for the primitive tasks (Bhat et al., 2006). While the construction of the composite task's reward function is designed to promote specific traits in the composite task's policy (e.g., a risk-averse policy versus a risk-taking policy), the unintended consequence is that the policies of the primitive tasks show the effects of these traits. This is due to the fact that, in modular reinforcement learning, all learning takes place in the policies of the primitive tasks. As a result, the policy for a given primitive task could potentially only be useful in the composite task for which it was originally learned.

A further potential problem, as modular reinforcement learning is currently implemented, is that it requires that the policies for primitive tasks provide the learned Q-value for a given action. Therefore, it is unable to reuse polices developed using methods other than reinforcement learning. While it is possible, in general, to learn the Q-values for an existing policy offline, this method can produce the same bias in the Q-values that the use of Q-learning produces since the Q-values must reflect the shared control of the agent. Even though alternative methods could be used to provide utility values without the use of Q-values (Pirjanian & Matarić, 2001), modular reinforcement learning still depends on control decisions flowing up from the low-level policies. As a result, there is no scaling advantage to extending beyond a two-level hierarchy where learning occurs at the lowest level since the top level merely uses a simple heuristic to combine the lower-level results.

In light of these difficulties, the success of composite reinforcement learning in these experiments is even more significant. It was the only approach which was consistently able to produce effective controllers. Furthermore, it was the only approach which was able to produce effective controllers for the complex FLOCKING-CA-GS-RA task.

7. Conclusion

The most significant result of these experiments is that, in the problem domains used in this chapter, the abstraction of an agent's action space provided more tangible benefits in the development of agent controllers than abstraction of an agent's state space. In a direct comparison, controllers that used significant action abstraction and no state abstraction had higher performance and were developed faster than controllers that made extensive use of state abstraction and moderate action abstraction. This is due to the fact that action abstraction changed the focus of the controller from one of low-level control to one of high-level coordination. This change in focus not only made the development of controllers for complex composite tasks more practical, but in many of the tasks, it also allowed the controller to have higher performance.

One aspect that is fundamental to the improved performance and rate of development of controllers using adaptive fuzzy behavior hierarchies was the ability to reuse existing primitive and composite behaviors. The ability to reuse, without modification, behaviors developed for one task in another task allowed for the development of controllers in individual pieces. The benefits of this approach are apparent when compared to the other approaches evaluated in these experiments which attempted to develop a controller all at once. As a result of this reuse, controllers for complex composite tasks that were once impractical to develop can now be developed with reasonable effort.

The results of the experiments shown in this work demonstrate that while the use of modular reinforcement learning has been successful in more constrained problem domains, it was unable to consistently produce effective control policies in the problem domains used here. For the problem domains used, the prospect of simultaneously learning effective policies for each primitive task proved to be too complicated. In contrast, composite reinforcement learning was the only approach that was consistently able to produce effective control policies. As discussed above, we believe that this was due to the use of action abstraction and the ability to reuse existing primitive behaviors, regardless of their implementation.

The most immediate opportunity for future work is a more in-depth investigation of the erratic behavior of using the **Small** abstraction level with the GOALSEEK primitive task. The next opportunity for future work is to use fuzzy reinforcement learning to learn composite behavior policies instead of the discrete Sarsa approach (Jouffe, 1998). The use of fuzzy reinforcement learning offers the potential for faster learning since the agent's state is no longer confined to a single discrete value. Lastly, the results of the current work can be used to develop more complex controllers in a variety of ways. One way is to directly use adaptive fuzzy behavior hierarchies to create far more complicated behavior hierarchies. Since our ultimate focus is in the development of agent controllers for use in complex, multi-agent tasks, the results of this work provide significant contributions in making the development of such controllers practical.

Acknowledgements

We would like to thank all the members of the AI research group at the University of Oklahoma for their contributions. We would like to especially thank to John Antonio, Sesh Commuri, Andrew Fagg, and Amy McGovern for their input and insights. Some of the computing for this project was performed at the OU Supercomputing Center for Education & Research (OSCER) at the University of Oklahoma.

8. References

Berenji, H. R. (1992). A reinforcement learning–based architecture for fuzzy logic control, *International Journal of Approximate Reasoning* **6**(2): 267–292.

Bhat, S., Isbell Jr., C. L. & Mateas, M. (2006). On the difficulty of modular reinforcement learning for real-world partial programming, *National Conference on Artificial Intelligence (AAAI)*, Vol. 21, AAAI Press, pp. 318–325.

Claus, C. & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multi-agent systems, *National Conference on Artificial Intelligence (AAAI)*, pp. 746–752.

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*, MIT Press, Cambridge, MA, USA.

Crites, R. H. & Barto, A. G. (1998). Elevator group control using multiple reinforcement learning agents, *Machine Learning* **33**(2): 235–262.

de Oliveira, L. S., Sabourin, R., Bortolozzi, F. & Suen, C. Y. (2003). A methodology for feature selection using multiobjective genetic algorithms for handwritten digit string recognition, *International Journal of Pattern Recognition and Artificial Intelligence* **17**(6): 903–929.

Dietterich, T. G. (2000). An overview of MAXQ hierarchical reinforcement learning, *International Symposium on Abstraction, Reformulation, and Approximation*, Springer-Verlag London, UK, pp. 26–44.

Driankov, D., Hellendoorn, H. & Reinfrank, M. (1996). *An Introduction to Fuzzy Control*, second edn, Springer-Verlag.

Er, M. J. & Zhou, Y. (2006). A novel reinforcement learning approach for automatic generation of fuzzy inference systems, *IEEE International Conference on Fuzzy Systems*, Vancouver, BC, pp. 100–105.

Eskridge, B. E. & Hougen, D. F. (2006). Prioritizing fuzzy behaviors in multi-robot pursuit teams, *IEEE International Conference on Fuzzy Systems*, pp. 1119–1125.

Eskridge, B. E. & Hougen, D. F. (2009). Extending adaptive fuzzy behavior hierarchies to multiple levels, *Technical Report TR-OU-REAL-09-001*, Department of Computer Science, University of Oklahoma, Norman, OK.

Glorennec, P. Y. & Jouffe, L. (1997). Fuzzy Q-learning, *IEEE International Conference on Fuzzy Systems*, Vol. 2, pp. 659–662.

Guyon, I. & Elisseeff, A. (2003). An introduction to variable and feature selection, *Journal of Machine Learning Research* **3**: 1157–1182.

Guyon, I., Gunn, S., Nikravesh, M. & Zadeh, L. A. (2006). *Feature Extraction: Foundations and Applications*, Studies in Fuzziness and Soft Computing, Springer-Verlag, Secaucus, NJ, USA.

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

Jouffe, L. (1998). Fuzzy inference system learning by reinforcement methods, *IEEE Transactions on Systems, Man and Cybernetics, Part C* **28**(3): 338–355.

Karlsson, J. (1997). *Learning to Solve Multiple Goals*, PhD thesis, University of Rochester, Rochester, NY, USA.

Konidaris, G. & Barto, A. G. (2007). Building portable options: Skill transfer in reinforcement learning, *International Joint Conference on Artificial Intelligence*, pp. 895–900.

Lin, L.-J. (1993). Scaling up reinforcement learning for robot control, *International Conference on Machine Learning*, Morgan Kaufmann, pp. 182–189.

Matarić, M. J. (1997). Reinforcement learning in the multi-robot domain, *Autonomous Robots* **4**(1): 73–83.

Pirjanian, P. & Matarić, M. J. (2001). Multiple objective vs. fuzzy behavior coordination, *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, Vol. 61 of *Studies in Fuzziness and Soft Computing*, Springer-Phisica Verlag, chapter 10, pp. 235–253.

Raymer, M. L., Punch, W. F., Goodman, E. D., Kuhn, L. A. & Jain, A. K. (2000). Dimensionality reduction using genetic algorithms, *IEEE Transactions on Evolutionary Computation* **4**(2): 164–171.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model, *Computer Graphics* **21**(4): 25–34.

Reynolds, C. W. (1999). Steering behaviors for autonomous characters, *Proceedings of the Game Developers Conference*, pp. 763–782.

Rohanimanesh, K., Jr., R. P., Mahadevan, S. & Grupen, R. A. (2004). Coarticulation in Markov decision processes, *Conference on Neural Information Processing Systems*, pp. 1137–1144.

Rohanimanesh, K. & Mahadevan, S. (2002). Learning to take concurrent actions, *Conference on Neural Information Processing Systems*, MIT Press, pp. 1619–1626.

Rummery, G. A. & Niranjan, M. (1994). On-line Q-learning using connectionist systems, *Technical Report CUED/F-INFENG/TR166*, Cambridge University.

Russell, S. J. & Zimdars, A. (2003). Q-decomposition for reinforcement learning agents, *International Conference on Machine Learning*, AAAI Press, pp. 656–663.

Smart, W. D. & Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots, *International Conference on Robotics and Automation*, Vol. 4, IEEE, pp. 3404–3410.

Sprague, N. & Ballard, D. H. (2003). Multiple-goal reinforcement learning with modular Sarsa(0), *International Joint Conference on Artificial Intelligence*, pp. 1445–1447.

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

Tan, M. (1993). Multi-agent reinforcement learning: Independent versus cooperative agents, *International Conference on Machine Learning*, pp. 330–337.

Tunstel, E. (1999). Fuzzy-behavior modulation with threshold activation for autonomous vehicle navigation, *18th International Conference of the North American Fuzzy Information Procesing Society (NAFIPS)*, New York, NY, pp. 776–780.

Tunstel, E. (2001). Fuzzy-behavior synthesis, coordination, and evolution in an adaptive behavior hierarchy, *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, Vol. 61 of *Studies in Fuzziness and Soft Computing*, Springer-Phisica Verlag, chapter 9, pp. 205–234.

Watkins, C. J. & Dayan, P. (1992). Q-learning, *Machine Learning* **8**(3-4): 279–292.

Whitehead, S. D. (1992). *Reinforcement Learning for the Adaptive Control of Perception and Action*, PhD thesis, University of Rochester.

Yang, J. & Honavar, V. (1998). Feature subset selection using a genetic algorithm, *IEEE Intelligent Systems and Their Applications* **13**(2): 44–49.