Using Priorities to Simplify Behavior Coordination

Brent Eskridge ESKRIDGE@OU.EDU
Robotics, Evolution, Adaptation and Learning Laboratory (REAL Lab)
School of Computer Science
University of Oklahoma, Norman, Oklahoma 73019-6151, USA

Abstract

Real-world behavior-based robot control problems require the coordination of a large number of competing behaviors. However, coordination becomes increasingly difficult as the number of competing behaviors increases. Behavior hierarchies address this difficulty, but do not solve it since high-level behaviors still require state information of lower-level behaviors. Abstracting the state of lower-level behaviors through priorities removes this restriction, which should allow the behavior hierarchy to better scale. However, whether or not this abstraction negatively impacts the effectiveness of the hierarchy is unknown. In this work, both the quality of priority-based behavior hierarchies and their ease of development are evaluated. This is done by using grammatical evolution to learn how to coordinate low-level behaviors to accomplish a task. I show that not only do priority-based behavior hierarchies perform just as well as standard hierarchies, but that they promote faster learning of solutions that are better suited as components in larger hierarchies.

1. Introduction

Real-time control is necessary for robots to operate in real-world environments. A popular approach for providing this real-time capability in dynamic environments is to use a behavior-based architecture (Brooks, 1986). One of the most significant problems for these architectures is the coordination, or arbitration, of multiple, competing behaviors. As the number of behaviors is scaled up, the coordination of behaviors becomes an increasingly complex and difficult task. While this problem is not restricted to robot control problems (Pirjanian, 1999), a solution is required if robots are expected to successfully complete complex tasks in dynamic environments.

This work seeks to address this scalability problem by providing an abstraction layer that separates the implementation of a given behavior from its use. This abstraction is accomplished by encapsulating all of the sensory input of a behavior into a single *priority*. Through this abstraction, priorities can significantly reduce the amount of state information required to effectively coordinate the behaviors. Furthermore, when behaviors are organized into a hierarchy that utilizes priorities, scaling to a large number of behaviors becomes possible.

In this work, I evaluate the effectiveness of priority-based architectures in efficiently controlling an agent and compare the results with standard architectures. In addition, I compare the ease with which each can be developed by using grammatical evolution, a form of genetic programming.

2. Related work

One approach to arbitrating behaviors decomposes the control problem into specific tasks (Nicolescu & Matarić, 2002). In this method, a behavior network is constructed, which is capable of determining the current task and, in turn, activates the appropriate behavior. Machine learning has even been successfully applied using this technique (Nicolescu & Matarić, 2001), thus reducing the development overhead for large systems. While the hierarchical nature of this particular approach does address some scaling issues, it does not address the situation where more than one behavior is active at any given time, which is where the more difficult problem lies.

Another method is to approach the problem from a multiple-objective function perspective (Pirjanian &

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

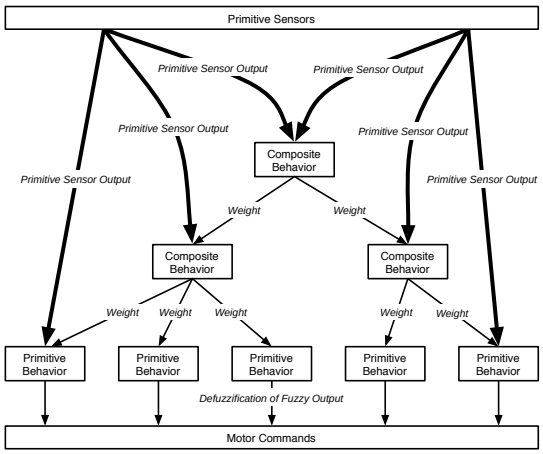

Figure 1. A sample fuzzy behavior hierarchy, as used by Tunstel (2001). Behaviors are implemented as fuzzy rule sets and output either weighting values for other behaviors or control values to be defuzzified. For simplicity, not all sensor output links are shown.

Matarić, 2000). In this method, a weighting function is developed that allows many behaviors to execute concurrently and then weights the resulting actions into a single action. However, in this technique, a new weighting function must be developed whenever the set of behaviors is modified, thus preventing reuse of the function when set of behaviors is modified.

Saffiotti (1997) uses fuzzy logic rule sets to allow for smooth transitions between behaviors and to improve the overall robustness of the system. The effectiveness of this approach is demonstrated by the development of a team of robots for the Aibo RoboCup League (Saffiotti & Wasik, 2003). The approach used in this work combines a number of these approaches into a fuzzy behavior hierarchy (Saffiotti & Wasik, 2003; Tunstel, 2001). Figure 1 shows one implementation of such a hierarchy, as described by Tunstel (2001). Rather than performing behavior arbitration in which a single behavior is selected for control, fuzzy behavior hierarchies typically use behavior fusion. This results in more than one behavior participating in the control of the agent and each behavior being individually weighted according to its applicability. While using this type of hierarchy does not solve the behavior coordination problem, it can simplify it by decomposing it into smaller problems.

To aid in the development of the behavior hierarchies, machine learning techniques can be utilized. For example, it has been shown that genetic programming can be used to evolve what are called *composite behaviors* (Tunstel, 2001). These behaviors are responsible for coordinating lower-level behaviors through weighting values called *degrees of applicability*. This coordination scheme abstracts out many of the details of the lower-level behaviors. Furthermore, lower-level behaviors can be developed in isolation as they contain no internal reference to their parent composite behaviors. The hierarchical nature of the set of behaviors, along with this encapsulation, minimizes many of the negative effects encountered when scaling up the number of behaviors. Perhaps the most important of these is that once a behavior has been developed, it does not require any modifications to work in conjunction with other behaviors. Handling the interaction with other behaviors is the domain of the composite behavior.

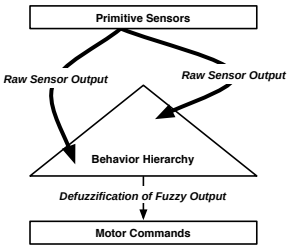

Figure 2. A high-level view of a fuzzy behavior hierarchy, as used by Tunstel (2001). Note that the primitive sensor outputs are used throughout the behavior hierarchy.

Unfortunately, scaling issues still remain. The most notable is that while the hierarchy makes composite behaviors simpler as the hierarchy is scaled up, the inputs to the composite behaviors increase exponentially. This occurs because there is no abstraction layer for a primitive behavior's sensory inputs (see Fig. 2). As a result, composite behaviors are still required to process all of the inputs of any child primitive behaviors. This couples a given composite behavior to a particular implementation, or context, of a sub-behavior. For example, a composite behavior at the top of the hierarchy would still be tied to the definition what it means to be "too close" to an obstacle. Previous research has not addressed this concern. This is most likely due to the fact that most behavior hierarchies described in the literature are relatively shallow.

3. Prioritizing behaviors

To remove this coupling, I propose adding abstraction layers between the sensor outputs and the behavior hierarchy. As a result, high-level composite behaviors use high-level information to coordinate behaviors, thus reducing the amount and granularity of the information required. This abstraction layer is implemented using *virtual sensors*. A virtual sensor is used to not only provide "normal" output for use by be-

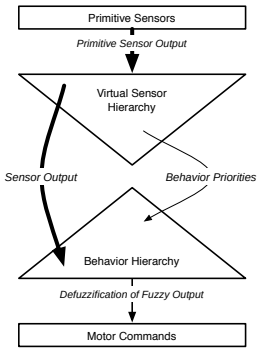

Figure 3. The sensor hierarchy mirrors the behavior hierarchy. Note that while primitive sensor information is available to primitive behaviors, priorities are provided to higher-level composite behaviors.

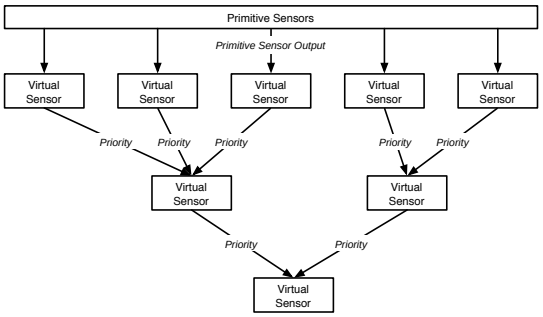

Figure 4. A sample sensor hierarchy that could be used in conjunction with a behavior hierarchy. Virtual sensors produce a priority for a corresponding behavior that is then used by virtual sensors at a higher level to produce further abstracted priorities.

haviors but to also provide an abstracted view of this output, hereafter referred to as a *priority*. Each virtual sensor is related to a particular behavior present in the behavior hierarchy and indicates the current *priority* of that behavior, given the information available. This priority value is then used by higher-level behaviors instead of the normal output provided by the sensor. We see examples of this approach frequently in our everyday lives. For example, when an engine warning light in a car turns on, it does not indicate the exact problem, but rather the general fact that a higher weight should now be given to behaviors such as pulling to the shoulder and turning off the engine.

By adding these virtual sensors to abstract the sensor data to higher and higher levels, I have effectively created a sensor hierarchy that mirrors the existing behavior hierarchy (Fig. 3). In fact, the sensor hierarchy is built at the same time as behavior hierarchy since composite behaviors require the priorities that the sensor hierarchy provides. Not only has the sensor data been abstracted for high-level behaviors, it has also been abstracted for high-level virtual sensors (Fig. 4). As a result, the high-level sensors benefit from the abstraction and can be conceptually simple.

Using priorities and virtual sensors, the system has four distinct stages (Fig. 3). First, the primitive sensors sense the environment and generate raw output. Second, virtual sensors process these raw outputs so they are usable by behaviors and then generate priorities which give overall indications of the importance of the current state. Next, these sensor outputs and priorities are used by the behavior hierarchy to produce control commands. Last, the control commands are sent to motor controllers for execution. It is important to note that there is no requirement for the individual stages to be performed synchronously. In fact, each stage could be evaluated at different rates.

The effectiveness of this system is dependent on the process used to generate the priorities and their quality. The amount of effort is similar to that of the standard approach, but the responsibility has been shifted from the behaviors to the virtual sensors. In doing so, I have reduced the composite behavior to its essence.

Note that there is no restriction on the method for determining or calculating a priority value. For example, a sensor used for determining the range and direction to obstacles could simply base the priority on the distance to the closest obstacle or could factor in the direction to the obstacle as well.

Further, note that although each virtual sensor is capable of sending its output (such as the distance to an obstacle) to its corresponding behavior, this is only necessary for behaviors that directly affect the control commands (i.e., are not composite behaviors). This is due to the fact that the priorities provide enough information for a composite behavior to coordinate the execution of the sub-behaviors. Furthermore, the abstraction of this output means that composite behaviors need not know the nature of that output. For example, fuzzy behaviors can make use of fuzzified values, but since behaviors are not restricted to fuzzy implementations, there is no limit imposed on the nature or volume of information provided to behaviors.

4. Learning coordination

In order for priorities to be effective, they must provide the agent with performance comparable to that provided by architectures that do not use priorities. In addition, priorities should facilitate faster development than non-priority-based architectures. To eval-

uate both these criteria, I apply machine learning to the problem of coordinating two different composite behaviors. For each composite behavior, both priority-based solutions and non-priority-based solutions are learned. If the performance of priority-based solutions is better than, or equal to, that of non-priority-based solutions, we can conclude that there is significant evidence that using priorities does not negatively impact the performance of agents that use them. Second, if priorities do in fact promote faster development, performance during trials that use priorities should improve at a faster rate than on trials that do not use priorities.

The first composite behavior learned is one that combines the primitive behaviors *collision avoidance* and *goal seeking*. This composite behavior is hereafter referred to as the CAGS behavior. While a simple composite behavior, the two primitive behaviors do compete against one another and intelligent arbitration is required for an agent to successfully arrive at the goal. The second composite behavior learned adds the primitive behavior *runaway* to CAGS and is hereafter referred to as the CAGSRA behavior. The addition of the third primitive behavior increases the complexity of the composite behavior and, therefore, has the potential to impact both the performance and the learning rate of the solutions. The sensor inputs for each of the three primitive behaviors are egocentric and two-dimensional in nature and consist of direction and magnitude components. Therefore, the use of priorities reduce the number of inputs in half from six to three. While this reduction may not seem meaningful, when the number of behaviors is scaled up, this reduction can become significant.

Grammatical evolution (Ryan et al., 1998) is used to learn each of the composite behaviors for a number of reasons. First, genetic programming, of which grammatical evolution is a special modification, has already been shown to be useful in learning composite behaviors (Tunstel, 2001). Second, the grammatical nature of fuzzy rulesets lends itself to this learning method. A standard, variable-length bit-string representation was used with eight bits representing a codon. For each experimental case, the evolutionary process was run for 50 generations using a population of 100 individuals. The initial population was generated randomly with subsequent generations being created through standard one-point crossover with a 1% probability of mutation. For evaluation, the bit-string was mapped to a set of fuzzy rules using either a grammar utilizing priorities or not utilizing priorities depending on the experiment. Figure 5 shows a portion of one of the grammars used.

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

⟨*antecedent*⟩ ::= ⟨*antecedent*⟩ ⟨*antecedent*⟩
  | ⟨*collision-theta-dir*⟩
  | ⟨*time-till-collision*⟩
  | ⟨*goal-dir-theta*⟩
  | ⟨*goal-arrival-time*⟩

⟨*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)

Figure 5. A portion of a non-priority grammar used in grammatical evolution.

Since the fitness function dictates what the system will learn, the proper selection of a fitness function determines what the resulting solution will look like. Three different fitness functions are used to evaluate solutions. The first fitness function was task-specific. For example, in the collision avoidance, goal-seeking behavior, the fitness of a solution was based on the ability of an agent to get closer to the goal at each timestep. Survivability (i.e., avoiding collisions) was implicitly rewarded by the opportunity for future reward. The second fitness function rewarded solutions that weighted the primitive behaviors at the level suggested by the priority and was computed by

$$fitness = priority \times (priority - weight)$$

Furthermore, the solution was penalized if it did not weight a primitive behavior with a non-zero priority. If a behavior with a nonzero priority is not weighted by any fuzzy rule, it received zero fitness for that timestep. This penalty was motivated by the observation that in initial runs, solutions gave some behaviors a weight of zero not by applying a weight of zero, but by not applying any weight at all. The last fitness function combined both the task-oriented and priority-oriented fitness functions into a single function.

5. Results

Both priority and non-priority versions of a given composite behavior were evolved on the same set of environments specially developed for training. The best individuals for each generation were then evaluated on a separate set of testing environments using the task-oriented fitness function. This fitness function was used since the effectiveness of the agent in accomplishing the task is the determining factor in the overall success of the system.

Table 1. The mean fitness, including standard deviation, in the test environments as measured by the task-oriented fitness function for the best of run solutions for the composite behavior that coordinates collision avoidance and goal-seeking behaviors (CAGS).

Fitness	Non-priority	Priority
Task	969.36 ± 105.49	979.456 ± 62.99
Priority	365.84 ± 201.41	386.73 ± 297.58
Composite	949.90 ± 79.81	982.73 ± 47.49
Heuristic	*N/A*	714.76
Random	214.56 ± 140.49	

Table 2. The mean fitness, including standard deviation, in the test environments as measured by the task-oriented fitness function for the best of run solutions for the composite behavior that coordinates collision avoidance, goal-seeking, and run-away behaviors (CAGSRA).

Fitness	Non-priority	Priority
Task	1067.02 ± 40.13	1062.90 ± 83.58
Priority	361.64 ± 186.83	421.80 ± 229.52
Composite	1072.69 ± 21.33	1077.79 ± 8.78
Heuristic	*N/A*	839.55
Random	611.16 ± 118.85	

For the CAGS composite behavior, there is no statistically significant difference in fitness between the priority and non-priority-based solutions for a given fitness function (see Table 1). The same is true for the CAGSRA composite behavior (see Table 2). This result is exactly what I was looking for as it indicates that although priorities abstract out many of the details of an agent's state, their use does *not* negatively impact the effectiveness of the agent. It is important to note, however, that these composite behaviors are relatively simple compared to those for which this technique has been developed. Further investigation using more complex composite behaviors is warranted before I can be certain that this abstraction will work as the system is scaled up.

For both the CAGS and CAGSRA composite behaviors, the evolved solutions that utilized a task-aware fitness function outperformed both the naïve heuristic composite behavior and composite behavior that randomly weights all the sub-behaviors.

Figures 6–8 show the results for the CAGS composite behavior using the three different fitness functions. It is important to remember that these results were obtained by evaluating the best-of-generation individuals in the testing environments using the task-based fitness function, regardless of which fitness function was used during evolution. As such, the learning rates

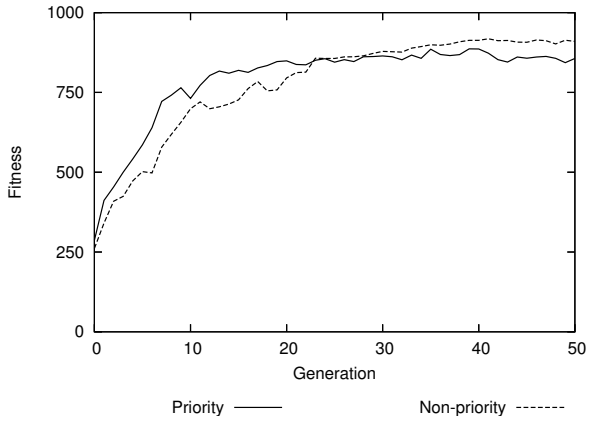

Figure 6. Comparison of priority-based and non-priority based learning using the task-oriented fitness function for the CAGS behavior.

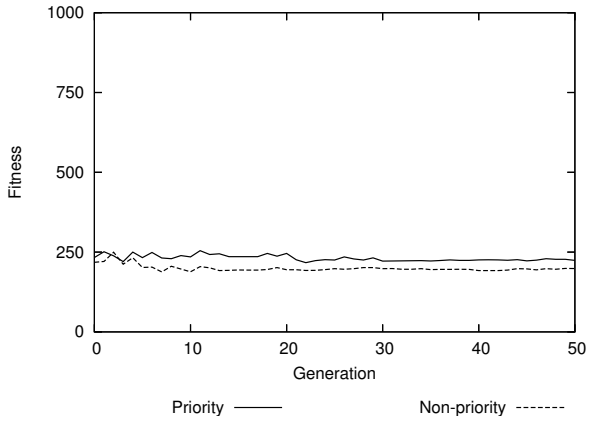

Figure 7. Comparison of priority-based and non-priority based learning using the priority-oriented fitness function for the CAGS behavior.

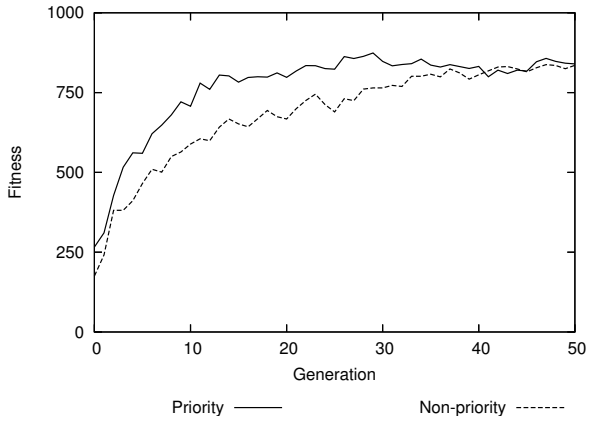

Figure 8. Comparison of priority-based and non-priority based learning using the composite fitness function for the CAGS behavior.

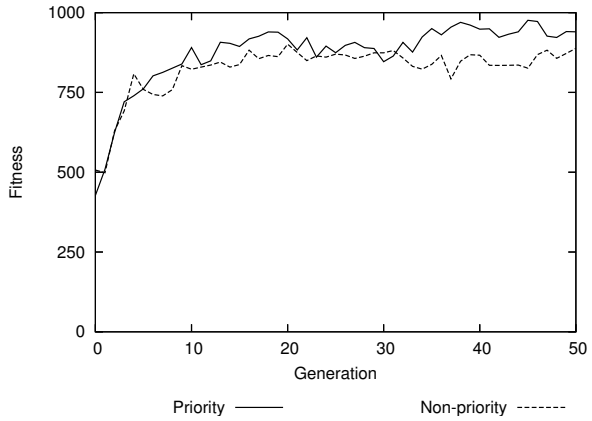

Figure 9. Comparison of priority-based and non-priority based learning using the task-oriented fitness function for the CAGSRA behavior.

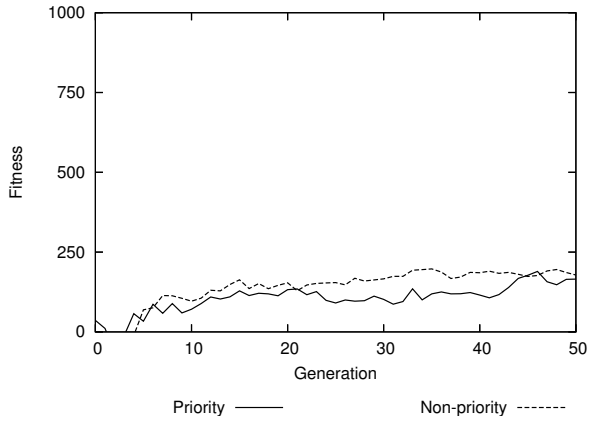

Figure 10. Comparison of priority-based and non-priority based learning using the priority-oriented fitness function for the CAGSRA behavior.

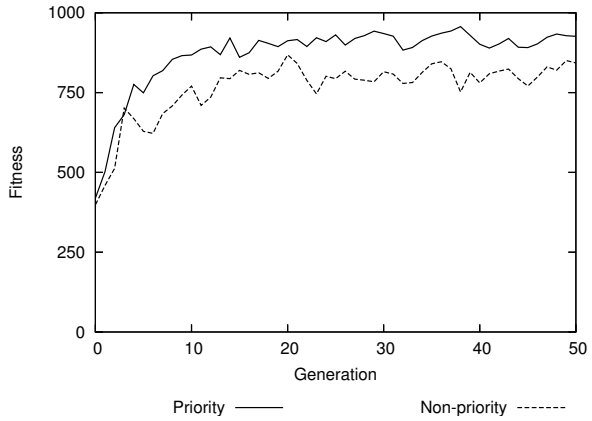

Figure 11. Comparison of priority-based and non-priority based learning using the composite fitness function for the CAGSRA behavior.

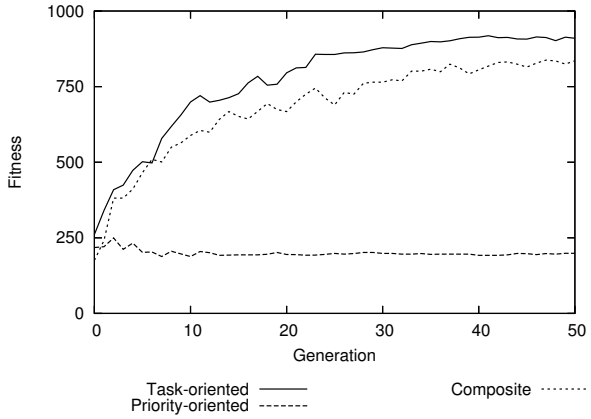

Figure 12. Comparison of the learning rates for non-priority-based solutions learning the CAGS behavior.

of each experiment differ from those shown when the appropriate fitness function is used. For example, Figure 7 seems to indicate that no learning occurred throughout the entire 50 generations, but the fitness values using the priority-based fitness function do in fact show learning. However, what can be seen in this particular figure is that the priority-based fitness function does not currently adequately represent the given task as success when using the priority-based fitness function does *not* lead to success when using the task-based fitness function. Figures 6 and 8 indicate that CAGS composite behaviors that utilize priorities improve at a faster rate than those that do not use priorities. Similar results can be seen in Figures 9 and 11 for the CAGSRA composite behavior. Again, this is the result that I hypothesized. It demonstrates that the abstraction provided by priorities aids learning by simplifying the search space. While the difference in learning rates is small in these experiments, I predict that as the priorities are used in composite behaviors that are far more complex than the ones evaluated here, the difference will be even more significant. Furthermore, the learning technique used here does not require an enumeration of the search-space for learning to occur. When other learning techniques are used that require such enumeration, such as Q-learning, the improvements in learning rate will be even more substantial, regardless of the complexity of the composite behavior being learned.

Figures 12-15 show that, regardless of the composite behavior being learned, a task-based fitness function is a necessity for success. While the solutions that utilized the priority-based fitness function improved their performance in the learning environments, that success does not translate to the test-environments using the task-based-fitness function. These results were

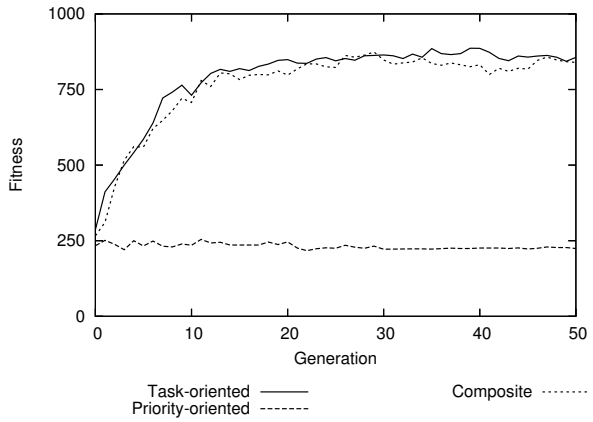

Figure 13. Comparison of the learning rates for priority-based solutions learning the CAGS behavior.

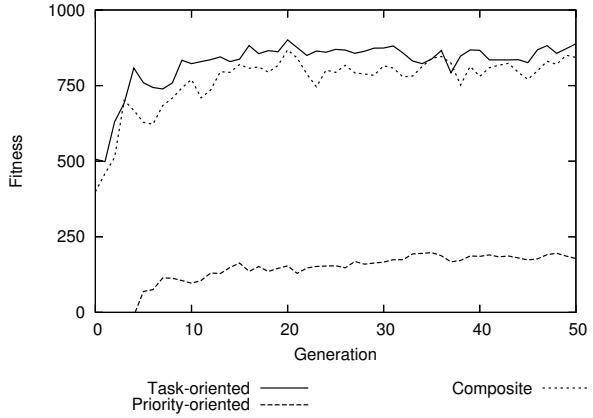

Figure 14. Comparison of the learning rates for non-priority-based solutions learning the CAGSRA behavior.

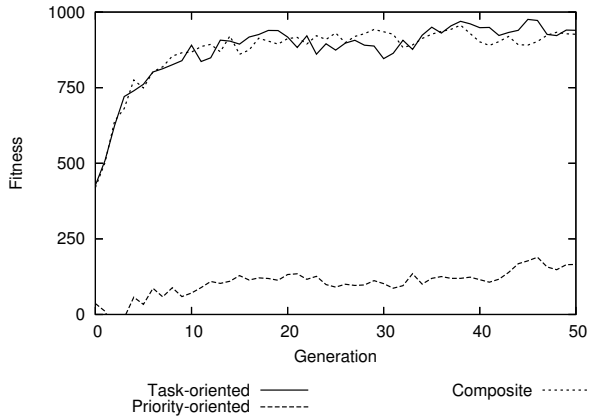

Figure 15. Comparison of the learning rates for priority-based solutions learning the CAGSRA behavior.

not the ones hoped for as they indicate that a simple fitness function that does not explicitly reward task-specific success may not be feasible. The appeal of such a fitness function is that although success for a complex composite behavior may be hard to quantify, a priority-based fitness function would be easy to construct thus simplifying the learning process. However, more investigation into why the priority-oriented fitness function failed is warranted before it is rejected.

6. Conclusions

These results show that for the composite behaviors CAGS and CAGSRA (defined in Section 4), using priorities does not negatively impact the efficiency of an agent and aids in learning to effectively coordinate their sub-behaviors. This indicates that priorities have the potential to allow behavior hierarchies to be scaled beyond their current constraints since many of the complexities of lower-level behaviors can be abstracted out without loss of performance. Furthermore, for these two composite behaviors, a task-oriented fitness function is required for an effective behavior to be learned.

The development of a priority that communicates the state a behavior operates in without exposing the low-level information used by the behavior is complex and requires great care. While the priorities used here were calculated using heuristics developed based on experience with the behaviors themselves, it was still a process that required tweaking of the calculation. In the future, I want to investigate the use of machine learning techniques to develop better algorithms for determining the priority of a given behavior. Improvements in the methods for determining a priority may even allow for the use of the priority-oriented fitness function.

Beyond the priority calculation, there are other areas of future work. Our final goal is the scaling of the system to a large number of competing behaviors. These behaviors include not only single agent behaviors, like the ones shown in this work, but also multi-agent behaviors that focus on the coordination of agents within a team. Furthermore, I wish to investigate the use of other machine learning techniques, such as neural networks and fuzzy Q-learning, in learning composite behaviors that utilize priorities.

Acknowledgments

I would like to thank Dean F. Hougen, PhD, for his guidance and assistance provided throughout this research. Thanks also go to all the members of the AI Research group at the University of Oklahoma for their

contributions. This material is based upon work supported by the U. S. Army Research Laboratory and the U. S. Army Research Office under grant number DAAD19-03-1-0142.

References

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

Nicolescu, M., & Matarić, M. (2001). Experience-based representation construction: Learning from human and robot teachers. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, Hawaii, USA* (pp. 740–645).

Nicolescu, M., & Matarić, M. J. (2002). A hierarchical architecture for behavior-based robots. *International Joint Conference on Autonomous Agents and Multiagent Systems*. Bologna, Italy.

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

Pirjanian, P., & Matarić, M. J. (2000). Multiple objective vs. fuzzy behavior coordination. In D. Drainkov and A. Saffiotti (Eds.), *Fuzzy logic techniques for autonomous vehicle navigation*, 235–253. Springer-Verlag.

Ryan, C., Collins, J. J., & O Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. *Proceedings of the First European Workshop on Genetic Programming* (pp. 83–95). Paris: Springer-Verlag.

Saffiotti, A. (1997). Fuzzy logic in autonomous robotics: behavior coordination. *Proc of the 6th IEEE Intl Conf on Fuzzy Systems* (pp. 573–578). Barcelona, Spain.

Saffiotti, A., & Wasik, Z. (2003). Using hierarchical fuzzy behaviors in the robocup domain. In D. M. C. Zhou and D. Ruan (Eds.), *Autonomous robotic systems*, 235–262. Berlin, DE: Springer-Verlag.

Tunstel, E. (2001). Fuzzy-behavior synthesis, coordination, and evolution in an adaptive behavior hierarchy. In A. Saffiotti and D. Driankov (Eds.), *Fuzzy logic techniques for autonomous vehicle navigation*, vol. 61 of *Studies in Fuzziness and Soft Computing Series*, chapter 9. Physica-Verlag.