Simulation and Learning Techniques for Robot Control Architectures

by

Julian N. Bishop

(Under the direction of Walter D. Potter)

Abstract

This thesis investigates techniques for the development of mobile robot control architectures. The work is split into two main parts. In the first part (Chapters 2 and 3) a simulation technique is proposed by which potential control architectures could be investigated without recourse to constructing a physical robot. In the second part (Chapter 4) a machine learning technique is developed that could eventually be incorporated into a robot's control architecture. Neither part of this thesis represents a fully completed project, but rather the foundations from which to pursue two separate projects, both of which could contribute to our toolbox for the development of intelligent robots.

Index words: Mobile Robots, Simulation, Behavior Based Control, Simulated Behaviors, Dynamical Systems, Behavior Phase Space, ALCSR, A-Life, Learning Classifier System, Reinforcement Learning, Genetic Based Machine Learning.

Simulation and Learning Techniques for Robot Control Architectures

by

Julian N. Bishop

M.Eng, The University of Durham, 1998

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2006

Simulation and Learning Techniques for Robot Control Architectures

by

Julian N. Bishop

Acknowledgments

The three papers contained in this thesis represent a change in the direction of my life: from an uninspired software developer on a road to nowhere, to someone who is excited about his work and the future it might bring. I am very grateful to everyone at the University of Georgia Artificial Intelligence Center who made this change possible and also enjoyable. Firstly to Dr. Potter for his unswerving support and encouragement, beginning before I even enrolled; for providing a valuable source of motivation, and for repeatedly sparking my intellectual curiosity both in and out of class, leading to the first two papers in this thesis. To Dr. Nute for his kindness and support throughout my work on the NED project and in his AI class. To Dr. Rasheed for his many useful suggestions during my research projects on learning classifier systems, genetic algorithms and neural networks. Lastly, to Brian Smith, Daniel Tuohy, and again Dr. Potter, for indulging my propensities for idea bombardment, micro-questioning, and discussing the tiniest detail.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis investigates techniques for the development of mobile robot control architectures. The work is split into two main parts. In the first part (Chapters 2 and 3) a simulation technique is proposed by which potential control architectures could be investigated without recourse to constructing a physical robot. In the second part (Chapter 4) a machine learning technique is developed that could eventually be incorporated into a robot's control architecture. Neither part of this thesis represents a fully completed project, but rather the foundations from which to pursue two separate projects, both of which could contribute to our toolbox for the development of intelligent robots.

1.1 Simulation

The main problem for the use of simulation during the design phase of a real robot is ensuring that results from simulation will carry over to the real world. In particular, the accurate modeling and simulation of mechatronic systems is always problematic due to non-linear device characteristics, sensor noise, limited resolution and stochastic errors. We suggest a methodology, dubbed the "Simulated Behaviors Approach", for circumventing these problems in the domain of behavior based robots

Chapter 2 proposes the Simulated Behaviors Approach for using simulation to investigate the structure and function of control architectures for behavior based mobile robots. The philosophical basis of the approach is discussed and a set of guiding principles are suggested. Finally, progress is outlined and further lines of research are identified.

Chapter 3 describes the Simulated Behaviors Approach in detail. A methodology is suggested for specifying and simulating robot behaviors independently of any particular physical implementation. Finally, a robot design problem against which to evaluate the approach is described, and our ongoing research is reported.

1.2 Learning

At any point in time, given its current sensor readings and any internal state or representation it might have, a robot must select its next action with respect to some goal. These goals and the associated optimal behavior policy may be explicitly represented in the robot's software, or they may be implicit in the robot's control architecture, but in both cases they have been provided by the robot's human creators. An alternative approach is to design the robot's control architecture such that it may learn the optimal behavior policy by trial and error in response to rewards that it receives from its environment or from some reinforcement program. One machine learning paradigm that is well suited to reinforcement learning is the Learning Classifier System.

Learning Classifier Systems (LCS), introduced by John Holland [3], are complex adaptive systems designed for machine learning. A typical LCS employs reinforcement learning and evolutionary computation to learn a set of rules (classifiers) from online experience that represent some target concept or behavior policy. For a current overview of LCS, see [1]. Examples of LCS applications include control systems in engineering applications [5], modeling and optimization [4], data mining [9] and the control of artificial animals in simulated environments [6]. In [2] Dorigo and Colombetti apply their own version of the Holland style LCS, dubbed ALECSYS, to controlling several simulated and real robots. It is worth noting however, that ALECSYS still employed an input alphabet of zero, one and don't care {0, 1, #}.

Holland's original model, while still inspiring, is known to be difficult to implement effectively due to its complexity. Furthermore, its {0, 1, #} input alphabet limits the type of inputs it can learn from. In [2], Dorigo and Colombetti concluded that they had exhausted the potential of such systems for learning in robot control problems. In [7] Wilson introduced XCS which is a simplified, modified version of the original LCS model. XCS has been successfully implemented by a number of researchers and has been shown to learn minimal representations of optimal solutions in some problems [7], making it the first generally competent Michigan style LCS. Some variants of XCS have even learned successfully in classification problems with integer and real valued inputs [9], [8]. Since a mobile robot is embodied in the real world, its sensors typically detect real quantities that have real values. Hence we are encouraged to believe that a real-valued LCS which departs from Holland's original design might have greater potential for learning in robot control problems than the LCS used by Dorigo and Colombetti.

Admittedly, a classification problem is typically an easier problem than controlling a mobile robot. However, even simple classification problems with real-valued inputs still represent a challenge for the LCS paradigm, particularly when the training data that constitute the system's experience are distributed non-uniformly, and when different classes cover different proportions of the input space. Such non-uniform distribution of experience is likely to be encountered in the learning problems faced by any mobile robot operating in a real environment. Hence it is of paramount importance that any LCS for robot control should be able to deal with the imbalance effectively. This is the problem we address in Chapter 4 with the development of a new real-valued LCS, called ALCSR. In order to focus on real valued inputs and non-uniform experience distributions, other aspects of ALCSR are simplified: we restrict its first incarnation to single-step classification problems in which there are only two possible classes, i.e. binary classification. ALCSR stands for Artificial Life Classifier System for Real valued inputs. The term Artificial Life is used here to refer to aspects of ALCSR's internal design which are guided by an abstract model of how a population of artificial organisms might evolve usefully within the reinforcement learning setting. The results of our initial experiments are presented and discussed, and future research directions are outlined.

1.3 References

[1] Bull, L., Kovacs, T. (eds.) (2005) “Foundations of Learning Classifier Systems”, Series: Studies in Fuzziness and Soft Computing, Vol. 183, Springer, 2005.

[2] Dorigo, M., Colombetti, M. (1998) “Robot Shaping: An Experiment in Behavior Engineering”, MIT Press / Bradford Books, 1998.

[3] Holland, J.H. (1975) “Adaptation in Natural and Artificial Systems”, University of Michigan, Ann Arbor, 1975.

[4] Smith, R.E., Dike, B.A., Ravichandran, B., El-Fallah, A., Mehra, R.K. (2000) “The Fighter Aircraft LCS: A Case of Different LCS Goals and Techniques”, In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.), Learning Classifier Systems: From Foundations to Applications, volume 1813 of LNAI, Springer-Verlag, Berlin, pp 282-289, 2000.

[5] Vargas, P.A., Filho, C.L., Von Zuben, F.J. (2002) “On-line approach for loss reduction in electric power distribution networks using learning classifier systems”, In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems, volume 2321 of LNAI, Springer-Verlag, Berlin, pp 181-196, 2002.

[6] Wilson, S.W. (1987) “Classifier Systems and the Animat Problem”, In: Machine Learning 2:199-228

[7] Wilson, S.W. (1995) “Classifier Fitness Based on Accuracy”, Evolutionary Computation 3(2): 149-175

[8] Wilson, S.W. (2000) "Get Real! XCS with Continuous-Valued Inputs", In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Learning Classifier Systems: From Foundations to Applications, volume 1813 of LNAI, Springer-Verlag, Berlin, pp 209-219, 2000.

[9] Wilson, S.W. (2001) "Mining Oblique Data with XCS", in Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems, pp 158-174

Chapter 2

Towards Developing Behavior Based Control Architectures for Mobile Robots Using Simulated Behaviors

2.1 Introduction

This chapter[1] is concerned with the design, development and investigation of control architectures for behavior based mobile robots [2] through the use of simulations. The use of simulation has the potential to save time and money in the robot development process, bring more resources to bear against the project objectives, facilitate research and lead to improved designs. While this paper is motivated by the development of control architectures by hand, other applications of simulation include robot design by the techniques of evolutionary computing [8], and research into embodied cognitive science [9]. Space prohibits a general discussion of the merits of robot simulation here, but one can be found in [5].

This paper will take the term 'behavior' to mean, *an observable and repeated pattern in the relationships among spatio-temporal events associated with an agent and its environment.* An agent will be taken as anything that has the ability to ascertain and cause events in its environment. These terms are intended to apply equally well to robots, animals, people, and even computer programs such as virtual organisms and soft-bots [4].

A discussion of emergent behavior is provided by [15], and a more concise description is given in [13]. In this paper the term *emergent* is used to refer to a behavior, the existence of which depends upon specific interactions between other behaviors of the same agent and/or

[1]Bishop, J.N., Potter, W.D. (2004) "Towards Developing Behavior Based Control Architectures for Mobile Robots Using Simulated Behaviors", In: Arabnia, H.R. (ed.) Proceedings of The International Conference on Artificial Intelligence IC-AI'04 Volume II, Las Vegas, Nevada, 2004, pp. 575-581. Reprinted here with permission of publisher.

other agents. It is implied therefore, that there must be some behaviors that can exist independently of other behaviors. Mataric coined the term "*basis behavior*" to refer to such behaviors, and used robots with behavior based control architectures to study the use of basis behaviors as building blocks for synthesizing and analyzing complex group behavior in multi-agent systems [14][13]. In this paper, the same idea is employed only the means has become the ends: we suggest the use of simulated behaviors at a lower functional granularity to study control architectures on a single agent. In order to separate this unproven approach from the work by Mataric, we use the term "*base behavior*" rather than basis behavior.

2.2 Philosophical Issues

It is worth remaining circumspect and asking whether there are any fundamental obstacles to the success of efforts to simulate behavior based mobile robots that should be kept in mind from the outset.

In [6], and also partly in [3] and [7], Brooks describes the key ideas of situatedness, embodiment, physical grounding and emergent behavior as important characteristics of a behavior-based mobile robot. The field of behavior based robotics is further described in [12], and [16]. Before attempting to simulate such robots it should be asked whether it is even possible for these key characteristics to be honored by a simulation. Essentially, this is the same as asking if it is possible for these characteristics to be honored by a software or 'virtual' agent.

2.2.1 Situatedness

The situatedness of behavior based robots has been characterized as referring to all of the following ideas: "The robots are situated in the real world - they do not deal with abstract descriptions, but the here and now of the environment which directly influences the behavior of the system." [6][10]. "...predictability and stability of environment have a direct impact on the complexity of the agent that must exist in it" [1]. "A situated agent must respond in a timely fashion to its inputs" [6]. "The world is its own best model an agent uses its perceptions of the world instead of an objective world model" [6]. "(The world) is always exactly up to date. It always contains every detail there is to be known." [7].

Apart from existing "in the real world", there is nothing in the preceding paragraph that cannot be equally true of a virtual agent, and this position is well argued by Etzioni in [4]. Further evidence of this viewpoint can be found in [17]. The significance of the real world to situatedness would seem to be that an *agent* in the real world, such as a mobile robot, is inevitably subject to all of the above conditions[2], whereas a virtual agent may be subject to only some subset of them depending on the particular implementation of that agent and its environment. The important point here for efforts at simulation is that care must be taken to ensure that these conditions are indeed met by our particular implementation.

2.2.2 Embodiment

Embodiment refers to the significance of an agent having a body. How this significance should be understood is a matter of ongoing discussion. One position already considers it possible for a virtual agent to be embodied: "Embodiment is a type of situatedness having a physical body and thus interacting with the environment through the constraints of that body Physical robots are embodied, as are simulations whose behavior is constrained and affected by (models of) physical laws." [1]. Several different types of embodiment, and the extent to which they apply to robots and virtual agents are defined and discussed in [18], [19] and [20], which provide a good overview of the ideas involved. An ontologically independent definition of embodiment is suggested in [11], which emphasizes the importance of structural coupling between a system and its environment.

In the context of a behavior based mobile robot, structural coupling is well expressed as follows: "The robots have bodies and experience the world directly - their actions are part of a dynamic with the world, and their actions have immediate feedback on the robots' own

[2]This is unsurprising since the idea of situatedness was originally derived by considering agents in the real world.

sensations." [6]. This structural coupling will be taken as the aspect of embodiment that must be honored in simulation. It is interesting to note that while Brooks was actually advocating the importance of a *physical* body in the *real* world, neither of these words appear in the quotation just cited, which, it is felt here, applies equally well to virtual worlds and bodies as to real ones (with the substitution of 'agent' for 'robot'). Support for this position can be found in [4] and [11].

2.2.3 Physical Grounding

Having conveniently side-stepped the issue of having a real physical body in the contexts of situatedness and embodiment, it must now be met head on. One of the key reasons Brooks advocates the importance of *physical* embodiment is expressed in his Physical Grounding Hypothesis: " to build a system that is intelligent it is necessary to have its representations grounded in the physical world." [7]. With regards to physical embodiment Brooks explains: "Only through a physical grounding can any internal symbolic or other system find a place to bottom out, and give 'meaning' to the processing going on within the system. The world grounds (the) regress (of meaning giving)" [6]. As to how this grounding is performed for a mobile robot, Brooks says "it is necessary to connect it to the world via a set of sensors and actuators." [7].

There is an implication in the Physical Grounding Hypothesis that the information a real robot gleans from its environment through its sensors is somehow more meaningful than the information a virtual agent could glean from its virtual environment through virtual sensors. This implication is disputed here. Any sensor, be it real or virtual, produces an abstraction of some aspect of the environment. How such abstractions are ultimately represented and operated against within the agent is beside the point when it comes to evaluating the meaning of the abstraction itself. While we agree that the only meaningful abstractions for a real robot are those made from its real environment using its own real sensors, we further suggest that the only meaningful abstractions for a virtual agent are those made from its virtual environment using its own virtual sensors. As such, both the real robot and virtual agent may be equally well “grounded” in their own real and virtual realities respectively. Again, this position is very similar to that of Etzioni in [4].

2.2.4 Validity of Simulation

While it is claimed here that there is no reason in principle why a virtual or simulated agent cannot be every bit as situated, embodied and grounded in its world as a real robot, it is not implied that this is an easy thing to achieve. Nor is this claim intended to dismiss concerns about the validity of simulating real robots. However, it is felt that such concerns are really getting at something different. Specifically, when it is claimed that a virtual environment in which a virtual agent is situated, embodied and grounded, is such a good model of a real environment in which a real robot is situated, embodied and grounded, that experimental results in the virtual system should be expected to carry over to the real system.

It should be asked whether such a good model is even possible. Fortunately, as described in [11], the field of evolutionary robotics has already provided an answer in the affirmative. Quick goes on to suggest the following axiom: “Where behavior emerges from the interplay between system and environment, if exactly the same system-environment relationship is instantiated in two cases then the same characteristic behaviors are seen to emerge”. This axiom corresponds to preserving the spatio-temporal relationships between an agent and its environment.

Of course there is also plenty of anecdotal evidence about attempts to simulate real robots that have failed to be useful, which may have fueled the idea that such attempts are futile and led researchers to postulate reasons why: the simulated robot cannot be embodied, for example. Why then do some attempts succeed and others fail? This is surely an important question to ask before embarking on any such attempt. Moreover, the modeling process bears closer examination.

2.2.5 Making The Right Abstractions

In order to design a virtual environment and an inhabiting agent that models an existing real environment and robot, it is necessary to make a great many abstractions about the real system. Clearly, the real system contains a far greater complexity of information and physical laws than could ever be entirely reproduced in a computer simulation. So, it is necessary to identify all aspects of the real system that are *relevant* to the subset of that system's behaviors which will be modeled by the virtual system. Key questions include - what are the behaviors of interest? what must happen in the real system in order that those behaviors are seen to emerge? and in particular, on what abstractions of the real environment is the agent operating? This first step is exactly the same as the first step that must have been performed when the robot itself was designed. The robot may exist in a world that "always contains every detail there is to be known", but it is only ever privy to the tiny fraction of that information that its designers deemed necessary when they selected its sensors. Although the robot engineer does not have to build the environment, and the simulation engineer does, both must make the same set of abstractions about that environment first. Consequently, designing a robot is subject to the same problem of making the right abstractions as arises when building a simulation of that robot.

However, the consequences of mistakes in this process are different for simulation than for the real robot. For the real robot, erroneous or incomplete abstractions on the part of its designer lead to a failure to achieve the desired behaviors i.e. the robot doesn't work, or if it does, it may not be for quite the same reasons its designers anticipated. In contrast, since the simulation is a simplification of reality, it may still appear to achieve the desired behaviors, but the results obtained do not transfer to the real system i.e. the simulation is not useful. It would seem unfair to criticize simulation on the grounds that some simulations are not useful, when some robots do not work for much the same reasons. Moreover, any comparison of these different consequences should be on the basis of the cost to remedy them.

In either case, there is no formal systematic way to ensure correctness from the outset and it comes down to the individual insight and aptitude of the researcher to make the right abstractions. Taken together with the fact that not all simulation problems are of equal difficulty, this readily answers the question of why some attempts succeed and others fail. Of course, actually building a robot and building a simulation do require different skill sets, and this further complicates the picture.

2.2.6 Evaluating the Model

So when results do transfer successfully from simulation to reality, does this mean that the model is perfect? No, merely that it is good enough for the purposes at hand in that at least a bare minimum of abstractions have been correctly made and well implemented by the simulation designers. Moreover, for other purposes the model may fail completely, but whether or not anyone should care about this is a matter of perspective.

An obvious question is: if the only way to validate a simulation is to compare its behavior with the real system being modeled, which requires that the real system be built, why spend time developing the simulation? The answer is that some experiments may still be faster and cheaper to perform on a validated simulation, or, as is the case with some examples of evolutionary robotics, impossible to perform with real robot hardware. In section 3, an approach to robot simulation is proposed that does not depend on the simulation being validated against a real system.

2.3 The Simulated Behaviors Approach

It was noted in the previous section that in order to prove the accuracy of a simulation it is necessary to compare its behavior to that of the real system being modeled. On what basis is this comparison made? Is there some formal systematic way in which the behavior of any two systems can be independently described so that no doubt remains in the evaluation of whether or not they are the same? We are not aware of such a methodology at this time, and in general it will be down to the each researcher to determine a means of behavioral comparison that is appropriate for the particular problem at hand. For an example see [14]. The idea of developing a general approach to behavioral specification is intriguing, but unfortunately it cannot be addressed within the bounds of this paper.

The same questions regarding behavioral specification should be asked of endeavors to build behavior based robots even if no simulation will be attempted. Ultimately, it is just such a specification of what the robot is supposed to do that determines the design decisions.

Regardless of the way in which a behavioral specification is expressed, an idea of the desired behavior necessarily precedes the process of abstraction for both designing a robot and designing a simulation of that robot. We ask here if there is any aspect of the robot that can be investigated using a behavioral specification alone, i.e. without building the robot as well. If the behavioral specification refers only to the overall observed behavior of the finished robot as it operates within its environment then the answer is no. However, this is not the case for a behavior based robot whose initial design has been specified in terms of multiple parallel base behaviors that operate simultaneously and combine to produce emergent behaviors. For such a robot, an additional system is required to perform 'arbitration', that is to coordinate the base behaviors. Whether the form of the additional system is regarded as simply a set of interconnections between the sub-systems responsible for the base behaviors, or an additional behavior itself, or a higher level independent 'executive controller', it will be described here as the control architecture.

The function of the control architecture is inextricably linked to the ways in which the base behaviors can be influenced by inputs to their respective sub-systems. The 'ways' and 'inputs' can be viewed as behavior control functions or interfaces to each base behavior that could be specified in the same terms as the base behaviors themselves. This amounts to parametizing the behavioral specifications and providing a control input for each parameter. The simplest example is an on/off input parameter which allows inhibition and sequencing of otherwise concurrent behaviors. For a behavior at quite a high level of granularity like wall-following, an example parameter might be the distance from the wall.

The behavior interfaces can be considered as the beginnings of the control architecture. To complete the control architecture the behaviors must be connected through their interfaces either to each other or via additional systems. The details of the eventual physical implementation of the base behaviors and any connecting systems are not under consideration. What is of interest here is the structure of the interconnections, the meaning of the signals carried, and the resulting overall function performed in terms of behavior coordination. The inner workings of the base behaviors are hidden from the control architecture (as in [14]), which, to some extent, is then desensitized to the particular abstractions of the robot's environment on which the base behaviors operate.

Recall now the definition of a behavior given in section 2.1: the definition implies no distinction between a base behavior of a real robot and a base behavior of a simulated robot as long as the important spatio-temporal relationships among events are preserved. If, in this way, the corresponding real and simulated base behaviors adhere to a common behavioral specification, then the required overall function of the control architecture should be the same in both the real and simulated case. Given such a specification of the base behaviors and the function of their control interfaces, it would seem possible to investigate the required function of the control architecture in simulation without constructing the real robot. Of course, this position supposes that the same behavioral specification would be rigorously applied to the base behaviors of any real robot (constructed later) to which the findings of the simulation were intended to be relevant. This approach has not yet been proven, but it is the subject of ongoing work.

Even if no real robot were ever constructed, it would still seem that there is something to be learned about control architectures by following the simulated behaviors approach. Consider the suggestion that as a result of designing and building behavior based agent A, insight was gained that proved helpful when designing and building behavior based agents

B and C with the same control architecture. If agent A is a wheeled differential drive robot, agent B is a wheeled robot capable of holonomic motion, and agent C is a robot that walks like an insect, then the original suggestion sounds reasonable. What if agent A is actually a faithful and fully validated simulation of the differential drive robot? Lastly, consider that agent A is actually a simulation of the base behaviors of the same, but now hypothetical, differential drive robot and the insights claimed are limited to the structure and function of the control architecture only, and not the details of any physical implementation.

There are a few caveats about the simulated behavior approach that are worth stating.

1. **Beware of Imitations** - The definition of a behavior that has been used does require the involvement of an agent with the ability to ascertain and cause events. Consider a simulation of a wall-following robot in which a representation of the robot is seen to follow a representation of the wall in observably the same way, but only because it is following a pre-programmed path without sensing the wall. Such an imitation does not fit the definition of a behavior and so is not a simulated behavior in the intended sense.

2. **Identical overall behavior is not required** - Suggesting that the required function of the control architecture for a real robot and for a simulation of that real robot's base behaviors should be the same, is not the same thing as saying that the resulting emergent behavior should be absolutely identical in both cases. Moreover, it is generally unlikely that any mobile robot simulation should behave identically to the system it models, and further it is not felt that such identical overall behavior is necessary for the simulation to be useful. It is hoped, however, that any differences in overall behavior that do emerge would come down to a matter of scalar parameters, and not a difference in fundamental structure, control functions or achievable behaviors.

3. **Feasible and unfeasible behaviors** - One advantage of using simulated base behaviors is that the control architecture is isolated from the inner workings of each base

behavior. It then becomes possible to capitalize on some simplifications of reality that would otherwise ruin simulation validity. In particular, the availability of perfect sensor data makes it far easier to implement certain base behaviors in simulation than it is on a real robot where issues of noise and resolution cause significant complications. Taken far enough, this could lead to a simulated behavior that is actually unfeasible to implement in the real world. Care must be taken to evaluate the feasibility of any simulated base behavior by comparing it with behaviors known to have been implemented successfully on real robots. If insight into control architectures on currently realizable robots is expected, only feasible behaviors can be simulated. On the other hand, an intriguing possibility presents itself. The investigation of control architectures can be freed of the constraints imposed by limitations of current robot hardware. Such investigations could provide motivation for directions to push sensor and actuator technology.

In summary, the key idea of the approach is to simulate behaviors, not real robots, with each behavior having a control interface via which it can be integrated into a behavior control architecture. The approach rests on the condition that as long as each base behavior is properly specified and modeled, and as long as the simulated behaviors meet the same defining criteria for being “a behavior” as can be used for real robots, then although the mechanisms that give rise to each base behavior are different depending on the system - be it physical robot A, physical robot B or simulation C (or even life-form D), the function of the control architecture required to achieve target emergent behavior should be the same, or at least sufficiently similar such that work on it in simulation bears utility for the real world.

Obviously, such simulations tell us nothing about sensor/actuator issues because no attempt is being made to model real ones. Essentially, this is an attempt to take advantage of the oft criticized fact that simulation can side step numerous real world issues at the sensor/actuator level, by purposefully side-stepping said issues (saving a lot of time in the process) in such a way that it does not matter for the objectives of the simulation. These simulations will not try to provide a development platform for code that can be reused on real robots. Instead they will try to gain insight into the function and logical design of control architectures in a way that is independent of their eventual physical implementation in the real world, which could be anything: hard-wired solid-state logic circuits, single processor embedded micro-controller, parallel multi-processors, and analogue circuits to name a few.

2.4 Principles for Validity

These principles do not represent an exhaustive list: they are the ideas distilled from an ongoing piece of work. For a general discussion of principles for implementing robot simulators see [5].

1. Space should be represented and managed such that
    (a) Dimensions are to scale with the real world.
    (b) The continuous nature of space is preserved or approximated at an arbitrary level of resolution that should be chosen to be very much smaller than any distances of interest.
    (c) No point in space may be occupied by more than one thing simultaneously.
    (d) Agents can be embodied in the sense of structural coupling as discussed in section 2.2.2
    (e) Agents can be situated as discussed in section 2.2.1
    (f) Agents are all equally subject to a set of physical laws regardless of processing load on the simulation.
    (g) Behaviors can be observed in the same way (visually) as they could be in the real environment.
    (h) Agents can be grounded in that all information on which a behavior operates internally is an abstraction of some aspect of the representation of the environment

(including the agent's representation), or derives from such abstractions via other behaviors.

2. Time should be represented and managed such that
   (a) Simulated time proceeds independently of any behavior.
   (b) The continuous nature of time is preserved or approximated at an arbitrary level of resolution that should be chosen to be much smaller than any time intervals of interest.
   (c) Relationships in simulated time are preserved independently of the amount of real time taken by the simulation to perform its processing.
   (d) Any number of behaviors may be active simultaneously without causing a distortion of spatio-temporal relationships due to processing load on the simulation.
   (e) The opportunity to interact with the environment afforded to a behavior is proportional to its response time and not the time taken for the simulation to perform processing associated with the environment.
   (f) The response time of a behavior may be treated as being different from the time taken for the simulation to perform processing on behalf of that behavior.
   (g) The relative orders of magnitude of the time taken for an agent to move a percentage of its body length, and the response times of the behaviors involved, are the same in simulated time as they are in the real world.

2.5 Ongoing Work

A crude but working prototype of a 2D mobile robot simulator has been developed and used as a test-bed to investigate techniques for implementing the principles for validity outlined in section 2.4. The principles associated with space (1 a - h) were all implemented with varying degrees of success. However, the prototype exhibited no success with all but the first of the principles associated with time (1 a). Consequently, the prototype is not felt to be up to the task of investigating control architectures, and only single, serial behaviors at a coarse level of granularity have been implemented and tested.

Despite these shortcomings, a number of simulated agents equipped with only a simple 'bump' detection capability did exhibit some interesting emergent behavior. Specifically, despite having sufficient maneuverability to cover the whole environment, a rectangular agent with a differential drive steering geometry was seen to get stuck against a wall from time to time in areas where its 'back up and turn' behavior was too simple to reliably cope with its surroundings. Another agent, circular in shape and capable of holonomic movement, was seen to effectively explore the entire environment without ever getting stuck while executing only a 'move at random' behavior. While watching these agents, the similarity between their behavior and that which would reasonably be expected of their real counterparts was quite striking, and provides a source of encouragement.

Multiple lines of work would be beneficial to this project:

1. Research into more rigorous and systematic approaches for
    (a) Making the right abstractions when modeling a behavior based robot and its environment.
    (b) Specifying behaviors and behavior control interfaces.
2. Find or build a simulator that meets the principles for validity outlined in section 2.4.
3. The simulator should be used to model a number of base behaviors at a low level of granularity for a single agent so that control architectures can be investigated in simulation.
4. An attempt should be made to transfer a control architecture developed in simulation to a real robot that implements the base behaviors as they were specified for the simulation.

2.6 Conclusions

This project has only just begun and certainly has a long way to go before any of the hopes and claims made here about the simulated behavior approach can be substantiated. In this paper, the motivation, philosophical position and direction of the project have been set out. Initial prototyping and experiments have provided some encouragement along with insight into the technical challenges that lie ahead, and multiple lines of work have been suggested.

2.7 References

[1] Mataric, M.J. (2001) "Learning in behavior-based multi-robot systems: policies, models, and other agents", In: Journal of Cognitive Systems Research 2 81-93, 2001.

[2] Arkin, R.C. (1998) "Behavior-Based Robotics", The MIT Press, 1998.

[3] Brooks, R.A. (1991) "Intelligence Without Representation", In: Brooks, R.A., Cambrian Intelligence, The MIT Press, 1999.

[4] Etzioni, O. (1993) "Intelligence Without Robots (A Reply To Brooks)", In: AI Magazine 14(4): 7-13.

[5] Torrance, M.C. (1992) "The Case for a Realistic Mobile Robot Simulator", In: Working Notes of the AAAI Fall Symposium on Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots, Cambridge, MA, October 23-25 1992.

[6] Brooks, R.A. (1991) "Intelligence Without Reason", In: Brooks, R.A., Cambrian Intelligence, The MIT Press, 1999.

[7] Brooks, R.A. (1990) "Elephants Don't Play Chess", In: Brooks, R.A., Cambrian Intelligence, The MIT Press, 1999.

[8] Brooks, R.A. (1992) “Artificial Life and Real Robots”, Towards a Practice of Autonomous Systems: European Conference on Artifcial Life, Paris, France, MIT Press, December 1991, pp. 3-10

[9] Ziemke, T. (2003) “On the Role of Robot Simulations in Embodied Cognitive Science”, In: AISB Journal 1(4)

[10] Brooks, R.A. (1991) “New Approaches To Robotics”, In: Brooks R.A., Cambrian Intelligence, The MIT Press, 1999.

[11] Quick, T., Dautenhahn, K., Nehaniv, C.L., Roberts, G. (1999) “The Essence of Embodiment: A Framework for Understanding and Exploiting Structural Coupling Between System and Environment”, 3rd Int. Conf. Comp. Anticipatory Syst. Liege, Belgium, 1999.

[12] Mataric, M. (1992) “Behavior-Based Control: Main Properties and Implications”, In: IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, Nice, France, pp. 46-54

[13] Mataric, M. (1992) “Designing Emergent Behaviors: From Local Interactions to Collective Intelligence”, In: J.-A. Meyer, H. Roitblat & S. Wilson, eds, From Animals To Animats: International Conference on Simulation of Adaptive Behavior.

[14] Mataric, M. (1994) “Interaction and Intelligent Behavior”, Technical Report AI-TR-1495, MIT Artificial Intelligence Lab.

[15] Steels, L. (1990) “Towards a Theory of Emergent Functionality”, From Animals To Animats: International Conference on Simulation of Adaptive Behavior

[16] Mataric, M. (1999) “Behavior-Based Robotics”, in the MIT Encyclopedia of Cognitive Sciences, Robert A. Wilson and Frank C. Keil, eds., MIT Press, April 1999, pp. 74-77

[17] Maes, P. (1994) “Modeling Adaptive Autonomous Agents”, From Animals To Animats: International Conference on Simulation of Adaptive Behavior

[18] Ziemke, T. (1999) “Life, Mind and Robots: The Ins and Outs of Embodied Cognition”, In: Wermter, S. & Sun, R. (eds) Hybrid Neural Systems. Springer Verlag, 1999.

[19] Ziemke, T. (2001) “Disentangling Notions of Embodiment”, Workshop on Developmental Embodied Cognition, Edinburgh, July 31, 2001.

[20] Duffy, B.R., Joue, G. (2000) “Intelligent Robots: The Question of Embodiment”, BRAIN-MACHINE 2000 December 20-22, 2000, Ankara, Turkey, 2000.

Chapter 3

The Simulated Behaviors Approach to Developing Robot Control Architectures

3.1 Introduction

The main problem for the use of simulation during the design phase of a real robot is ensuring that results from simulation will carry over to the real world. In particular, the accurate modeling and simulation of mechatronic systems is always problematic due to non-linear device characteristics, sensor noise, limited resolution and stochastic errors. In [1] we suggested a methodology, dubbed the "*Simulated Behaviors Approach*", for circumventing these problems in the domain of behavior based robots. Previously, we focused on a discussion of philosophical issues and general principles. In this paper we present the practical details of the approach and consider its potential usefulness and limitations.

The basic tenet of the Simulated Behaviors Approach is that if a robot's elementary behaviors, dubbed "*base behaviors*", are specified in a manner that is independent of any particular implementation, then a simulation of those base behaviors can provide useful insight into how a control architecture must coordinate them in order to produce the desired overall emergent behavior. We use the term "*base behaviors*" to separate our work from Mataric's "*basis behaviors*" which are used as higher level building blocks for complex group behavior in multi-agent systems [6][7]. We consider any behavior that results from the interactions of base behaviors to be emergent. For discussions of the term see [7][8][9].

Key to our approach is finding a suitable formalism for writing the behavior specifications to which both the simulated and real base behaviors must adhere. Note that no consideration of real device characteristics is required or advisable for behavior specification and simulation that is to be implementation independent. Clearly, such simulations are not intended to inform us about how to construct the mechatronics of each base behavior system on the real robot. Instead, by defining a behavior control interface as part of the specification of each base behavior, we aim to use the simulator to investigate the function of those interfaces and the architecture of the connecting and controlling systems with respect to resulting emergent behavior. In this way, the simulation can give us experimental freedom that would be far more costly and time consuming to pursue on the real robot. The output of the simulation (apart from seeing it run) is implementation independent specifications that set modular targets for robot construction.

3.2 The Simulated Behaviors Approach

The Simulated Behaviors Approach can be broken down into 6 stages, each of which will be discussed in turn.

1. Specification, by hand, of the base behaviors and their control interfaces.
2. Simulation of the base behaviors and their control interfaces.
3. Development (in simulation) of a control architecture that results in the desired overall behavior emerging in the simulator.
4. Generation, by the simulator, of a complete set of behavior specifications for all the base behaviors and the control architecture.
5. Construction of the real robot with parallel behavior systems, each one implementing one of the base behaviors according to its specification.
6. Implementation on the real robot of a control architecture according to the specifications generated by the simulator.

3.2.1 Specifying Behaviors

The dynamical systems approach [2][3][4] to robotics can provide us with just the formalism we require to specify behaviors independently of any implementation. Behaviors are defined in terms of state variables each of which represents some measurable quantity or aspect of the robot and/or its environment. Examples include the velocity of a particular point on the robot, position measures relative to something in the environment, and the angles and positions of any moveable parts of the robot. A behavior is represented by an n-dimensional phase space diagram where n is the number of state variables. Of particular interest is the topology of the phase space in terms of attractors, repellors, saddle points and separatrices. A good overview of these concepts is presented in [3]. By itself, the phase diagram contains no information about time, so a vector field is defined across the space which specifies the direction and rate of the behavior's evolution from any point within the space. Figure 3.1 presents an example of a behavior phase diagram. For a discrete-time system, the vector field becomes a description of where the system will be on the next time step.

While these concepts are very helpful for visualizing the dynamics of a behavior, there is more than one way to apply them to robotic systems. In [3], Beer allows total freedom in selecting the state variables from all quantities that relate an agent[1] to its environment, and all quantities internal to either the agent or the environment. He also defines the boundary between agent and environment to be wherever it is most usefully drawn for each particular analysis. He then couples parameters affecting the vector field of the environment's phase space, with state variables of the agent's phase space, and also vice versa. While this affords considerable predictive and explanatory power, it does not seem appropriate for our approach here, as it delves too deeply into the inner workings of each behavior system to avoid being forced to consider real device characteristics, which is one of our main objectives.

In [4], Schner constructs mathematical equations to describe a behavior's desired phase space, and then relies on solutions of those equations to generate the desired behavioral

[1]An agent is anything that can sense and act. A robot is an agent, and so is a simulated robot.

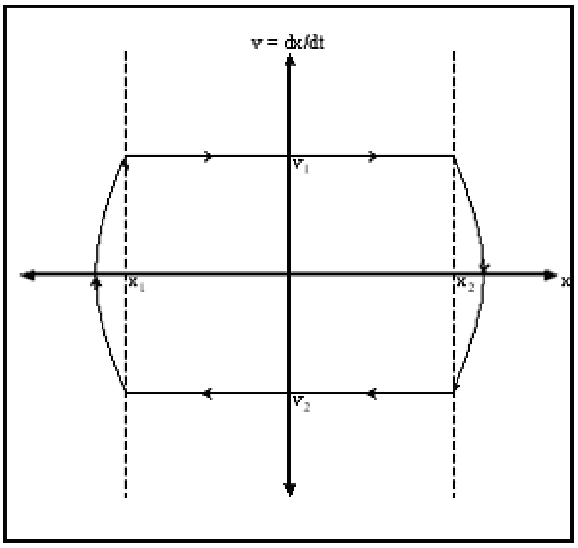

Figure 3.1: The phase space for a ‘patrolling’ behavior involving moving backwards and forwards following a line. The origin represents the midpoint of the line, and x represents the distance moved along the line from the midpoint. For clarity, transient behavior is not shown. The periodic attractor represents the long term behavior of moving at constant velocity until at a distance of x_1 or x_2, and then changing direction. The arrows indicate the direction of the vector field on the attractor. Again for clarity, the magnitude of the vector field is not indicated. Obvious candidates for behavior control parameters are x_1, x_2, v_1, and v_2.

dynamics on a real robot. We prefer not to say anything about how the dynamics are generated, and prefer to make our approach accessible without the need for equation solving.

For the Simulated Behaviors Approach we will specify our base behaviors using the following guidelines:

- We acknowledge that a real robot has a clear physical boundary delimiting it from its environment, and we permit no freedom to consider the boundary to be otherwise located.

- We restrict our state variables to those that relate the agent to its environment according to the boundary just defined. More specifically:

  - Our state variables must require consideration of both agent and environment in order to define them.
  - No internal access to the agent should be required in order to measure the value of any state variable.

- State variables may be a mixed bag of continuous and discrete quantities.
- We consider a behavior control interface to consist of:
  - One or more control parameters. The vector field on the behavior's phase space has a functional dependence on the values of the control parameters, the definition of which is part of the behavior's specification. The values of the control parameters are set externally to the behavior and may be set independently of each other and independently of other state variables.
  - One or more output parameters. Each output parameter has a functional dependence on the behavior's state variables, the definition of which is part of the behavior specification.
- We develop our behavior specifications graphically, by hand, either on paper and/or aided by facilities in our simulator.

The most trivial example of a behavior control interface consists of a single two-valued parameter that is used to inhibit (switch off) the behavior. This idea is used extensively (although not expressed in these terms) in the Subsumption architecture by Brooks [5]. In terms of our specifications, inhibition would be represented by setting the magnitude of all vectors in the field of the behavior's state space to zero, so that the behavior no longer makes any contribution to the values of any state variables.

Clearly these methods become unwieldy for behaviors of high dimensionality. So an important design consideration for the Simulated Behaviors Approach is to try to keep the dimensionality of each base behavior as low as possible. Where a behavior appears to be of high dimensionality, efforts should be made to decompose it into many behaviors of fewer dimensions, which could possibly be coordinated via appropriate control interfaces such that the more complex behavior is seen to emerge.

3.2.2 Simulating Behaviors

The state variables of a behavior can be categorized as either sensed, driven, or both. A behavior's specification tells us how it causes the driven state variables to evolve, but only in the absence of influences external to the systems that implement the behavior. In the presence of external influences on a driven state variable, it falls to the simulator to resolve any conflicts and determine how the driven variables may be perturbed from the values indicated by the behavior specification. When such influences arise due to aspects of the environment not accounted for in the state variables, the simulator must model the environmental physics sufficiently well to realistically resolve conflicts.

There is another potential source of conflict that depends on the base behaviors that have been specified. To be consistent with the behavior based paradigm [10][11][12], all behaviors should have the potential to run simultaneously. If a particular state variable is driven by more than one behavior simultaneously then again it falls to the simulator to resolve the conflict, but now the situation is potentially more difficult. The separate specifications of two base behaviors which have a driven state variable in common will not generally provide sufficient information by themselves to determine the specification of the two behaviors operating simultaneously. If the superposition of the two behaviors would occur naturally in the environment under physics, then the simulator's physics may be enough to resolve the conflict. However, if the superposition of behaviors would take place internally to the robot, as may be the case when two behaviors have a common actuator, then simulated environment physics cannot help. This is a limitation of our approach, and the price to be paid for abstracting away the inner details of each behavior. The significance of this limitation is discussed in section 3.3.

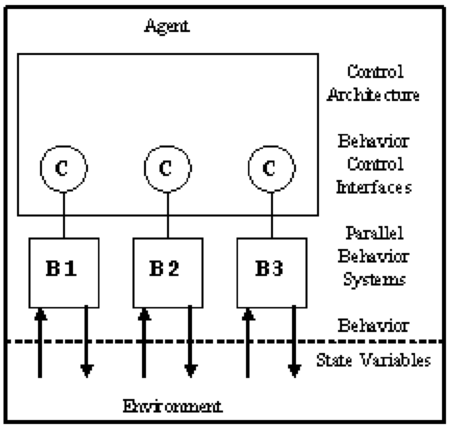

Figure 3.2: Logical view of three parallel base behaviors

There are three ways to address the problem of internal behavior superposition and still maintain the validity of the simulation:

- Treat the simultaneous operation of the conflicting behaviors as a separate behavior with its own specification. (There may be little point if that specification was the subject of the investigation.)
- If the function by which internal behavior superposition takes place is precisely known and considered part of the robot's logical design, then it may be worth bending our own rules and coding the function into the simulator.
- Redesign so that the conflicting behaviors are never active simultaneously.

Figure 3.2 depicts a logical view of three parallel base behaviors and their control interfaces. Each base behavior can be simulated using two possible approaches:

- Behavior specifications are sketched by hand. The simulator allows behavior code modules to be added, and provides facilities to run each behavior module individually and output plots of the behavior's phase space vector field. The code for each behavior can be adjusted until it produces plots that adhere to all the main features of the sketched specifications. Hence it is not necessary to fully specify the entire vector field initially, as the code for the simulated behavior effectively fills in the blanks.

- The simulator provides graphical tools for the user to develop the behavior phase space specifications on-screen. Since the behavior specifications are entered directly into the simulator, it can use them as a lookup array when running, and the user need not write any code module for the behavior. The simulator can use curve fitting to interpolate between points in the array if required. This approach places greater demands on the simulator but should result in more rapid prototyping of behavior specifications by the simulator users.

As pointed out in [1], care must be taken not to specify behaviors and control interfaces that are unfeasible to implement on a real robot, (although the freedom to do just that does allow for simulations of robots with purely hypothetical behaviors).

3.2.3 Developing the Control Architecture

With the base behaviors simulated, the simulator should allow total freedom to add one or more code modules that implement a control architecture of any paradigm. Examples include top-down hierarchical, subsumption, and dynamical systems. The control architecture modules operate against the control interfaces of the base behaviors, as shown in Figures 3.2 and 3.3. Each control module would be simulated as another process in parallel with the base behaviors.

It is possible to view each control module as a sort of pseudo-behavior, only at a higher level of abstraction, and apply the same approach to their design and specification as used for the base behaviors. Figure 3.3 illustrates an example of such a scheme. In general, each additional level of abstraction should cause a reduction in the number of control parameters.

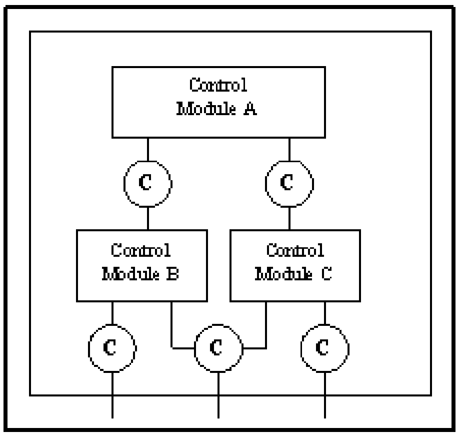

Figure 3.3: An example control architecture

3.2.4 Generation of Complete Specifications

When the desired overall behavior is achieved in the simulation, the simulator can plot a complete set of phase space specifications for every module (base behaviors and control architecture). These specifications are independent of any particular implementation and hence can be reused for more than one implementation. Furthermore, such specifications provide modular targets during robot development, and help the robot designers make the right abstractions about the robot and its environment when determining what sensors and actuators are required.

The use of specifications generated by a discrete time simulation to guide the development of real robot systems does rest upon a hitherto unstated assumption. Specifically, that if the duration of each time step in simulation is much smaller than any timescales of interest on the real robot, then the continuous time dynamics will be closely approximated by the discrete time dynamics.

3.2.5 Constructing the Real Robot

The specifications generated from the simulator say nothing about how to construct robotic systems that adhere to them, which remains a difficult engineering problem. In order to determine whether or not a base-behavior system on a real robot adheres to its specifications, it is necessary to measure the performance of that system on the real robot and compare the results with the specifications. This comparison may prompt adjustment of the robot's systems until the specifications are met. Alternatively, it may be easier to adjust the simulated version of the behavior to match what the robot is doing, in which case stages 2 through 5 may become iterative. As long as compensatory adjustments in the simulated control architecture can maintain the desired overall behavior then such iterations should not cause any problems.

Stage 6 is similar to stage 5, except that it begins after the base-behaviors have been implemented according to specification on the real robot, and focuses on building the control architecture.

3.3 Evaluating the Approach

In section 3.2.2 a limitation of the Simulated Behaviors Approach was identified. Specifically, when the superposition of parallel behaviors occurs internally to a robot's actuator systems, we have no obvious way to determine the results for those actuators. Some workarounds are suggested, and although none of them are ideal, designing our base behaviors so that such problematic situations do not arise seems to be the best solution. In fact, on many behavior based robots the control architecture performs arbitration specifically to avoid such internal superpositions as the results are unpredictable. Consequently, this limitation is not felt to be of great significance.

The Simulated Behaviors Approach does incur some overhead during construction of the real robot, as it is necessary to measure the performance of each behavior system and produce behavior phase space diagrams for the real robot. This requirement should be kept in mind from the outset of any project that will follow our approach.

In order to thoroughly evaluate the Simulated Behaviors Approach, we should identify a task that would demonstrate the approach to be of value and then undertake that task. An appropriate task would be the application of stages 1 through 6 to a robot design problem in which the target overall behavior is non-trivial and unlikely to be immediately apparent from the specification of its base behaviors. In order that our problem is amenable to 2D simulation, we will formulate it in terms of movement on a level surface.

The problem is as follows. An everyday flat surface, such as the floor of an office or lab, is cluttered with lightweight objects of various sizes and shapes except that they are all 6 to 8 inches tall. Objects with a variety of colors and reflectivity are used, and some of them may even be translucent. Some of the objects are fixed to the floor and others are not. A number of constant light sources are placed in this environment, all of which emit light at a height of about 6 to 7 inches. Some light sources emit in all directions (like a bulb), and others emit only in certain directions (like a flashlight). Some of the light sources are suspended from above such that they have a minimum ground clearance of about 6 inches. Other light sources are free standing, and some of these are battery powered and not fixed to the floor.

A small mobile robot, limited in height such that it may pass directly beneath a suspended light source without hitting it, is to move through the environment and shunt the unfixed objects around with the goal of maximizing darkness. Generally it may accomplish this by walling off the light sources with light blocking objects. Sometimes there may not be enough light blocking objects to go around and it may be necessary to move an unfixed light source over to another light source so that they may be walled in together. When the robot determines that it has done all it can, it should find the darkest spot available, and sit in it. The robot may not be provided with any prior maps, although an ontology of the environment will be implicit in its design, systems and programs.

The problem described above is felt to be rather challenging, not least because when the robot is shunting an object it will not generally be able to ‘see’ past it. Furthermore, any movement of objects within the environment changes the patterns of light and shadow. There are a number of obvious ways to simplify the problem if necessary. In particular, making all the moveable objects the same size and shape, like uniform blocks, and enabling the robot to carry one without significantly obstructing its sensors’ lines of sight, would greatly simplify the problem. Such simplifications are to be resisted if at all possible.

It is not immediately apparent how best to solve this design problem, and we feel that the ability to do all the experimentation in simulation and then generate working behavior specifications before beginning robot construction will show the value of our approach.

3.4 Ongoing Work

As described in [1], an initial prototype of a 2D simulator was found to be inadequate. However, the reasons for its inadequacy have been identified and a new simulator is under development. This paper has identified several features that will be required of our simulator. When developed, the simulator will be used to tackle the robot design problem described in section 3.3.

3.5 Conclusions

Practical considerations for the Simulated Behaviors Approach have been discussed at some length, and the potential usefulness and limitations of the approach have been identified. In particular a specific example has been set out in terms of a mobile robot problem against which to test the approach.

While research following the Simulated Behaviors Approach is ongoing in the context of 2D simulation of single mobile robots, the ideas presented should scale up naturally to multi-agent simulation, and 3D simulation.

3.6 References

[1] Bishop, J.N., Potter, W.D. (2004) “Towards Developing Behavior Based Control Architectures for Mobile Robots Using Simulated Behaviors”, In: Arabnia, H.R. (ed.) Proceedings of The International Conference on Artificial Intelligence IC-AI’04 Volume II, Las Vegas, Nevada, 2004, pp. 575-581.

[2] Steels, L. (1994) “Mathematical Analysis of Behavior Systems”, In: From Perception to Action Conference: Proceedings of the Perarc Conference, pp. 88-95, New York: IEEE Press.

[3] Beer, R.D. (1995) “A dynamical systems perspective on agent-environment interaction”, Artificial Intelligence 72:173-215.

[4] Schner, G., Dose, M., Engels, C. (1995) “Dynamics of behavior: theory and applications for autonomous robot architectures”, Robotics and Autonomous Systems 16, 1995, pp. 213-245.

[5] Brooks, R.A. (1986) “A Robust Layered Control System for a Mobile Robot” In: IEEE Journal of Robotics and Automation, RA-2 April, 1986, pp. 14-23

[6] Mataric, M. (1994) “Interaction and Intelligent Behavior”, Technical Report AI-TR-1495, MIT Artificial Intelligence Lab.

[7] Mataric, M. (1992) “Designing Emergent Behaviors: From Local Interactions to Collective Intelligence”, In: Meyer, J.-A., Roitblat, H., Wilson, S. (eds.), From Animals To Animats: International Conference on Simulation of Adaptive Behavior.

[8] Steels, L. (1990) “Towards a Theory of Emergent Functionality”, From Animals To Animats: International Conference on Simulation of Adaptive Behavior

[9] Steels, L. (1994) “The Artificial Life Roots of Artificial Intelligence”, Artificial Life Journal, vol. 1, nr. 1-2, pp. 89-125, Cambridge, MA: The MIT Press.

[10] Arkin, R.C. (1998) “Behavior-Based Robotics”, The MIT Press, 1998.

[11] Mataric, M. (1992) “Behavior-Based Control: Main Properties and Implications”, In: IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, Nice, France, pp. 46-54

[12] Mataric, M. (1999) “Behavior-Based Robotics”, In: Wilson, R.A, Keil, F.C. (eds.), the MIT Encyclopedia of Cognitive Sciences MIT Press, April 1999, pp. 74-77

Chapter 4

An Artificial Life Classifier System for Real-Valued Inputs

4.1 Introduction

Classification problems with real-valued inputs still represent a challenge for Learning Classifier Systems (LCS), particularly when the training data is distributed non-uniformly or different classes cover different proportions of the input space. This motivates the design of a new LCS following an Artificial Life model.

This chapter

4.2 System Development Goals

1. **Problem**: The first version of ALCSR (presented here) should handle noise-free single-step classification problems with discrete fixed reward levels P over the space of continuous input values X and discrete actions A.

2. **Extensions**: The system should be extensible to handle noisy problems and also multi-step problems with delayed rewards.

3. **Model**: ALCSR should learn a complete map ($X \times A \Rightarrow P$) of the reward levels that is independent of any exploitation policy (like an accuracy-based system such as XCS).

4. **Performance**: In the absence of noise, ALCSR should achieve stable perfect performance using a simple exploitation policy (at least in single-step classification problems).

5. **Niching**: All niches in the input space should be discovered and populated regardless of their relative sizes, or the relative frequencies with which representative inputs occur. In other words, if a niche exists, ALCSR should learn accurate rules to cover it even if it is small and surrounded by much larger niches of different types.

6. **Optimality**: The population should continually evolve toward the minimal set of maximally general rules that could solve the problem without becoming stalled on a sub-optimal population.

7. **Currency**: Each rule in the population should have its own reservoir of currency (like a strength-based system) so that ALCSR could be extended to investigate ways in which transactions between rules might improve the systems capabilities and performance.

8. **EA Scope**: ALCSR is intended to serve as a platform for broadening the scope of the evolutionary algorithm within an LCS. It would be interesting to extend the rules genetic representation to facilitate the evolution of new features. This agenda will

include the evolution of condition morphologies and cooperative interactions between rules.

4.3 System Description

4.3.1 Natural Problem Topography

A-Life Model: Each data-point/training example experienced by ALCSR is a vector X of n real values, and is taken to represent a single unit of food welling up from the ground. The n real values locate the unit of food on an n-dimensional landscape in the space of possible system inputs. If the system has c potential actions, then c completely separate landscapes are considered to exist, one for each action. The type of food is not determined until ALCSR has executed an action and a non-zero reward has been returned to it. The reward level identifies the type of food; the quantity is always 1 unit.

Each landscape is to be colonized by a population of organisms that cannot move around once established. Organisms on different landscapes are considered to be completely different major species and are never permitted to interact in any way. Within each landscape, organisms may be adapted to eating just one type of food. Organisms that are adapted to eating different types of food are considered to be different sub-species. In order to survive, each organism must maintain positive strength by feeding on the appropriate food-type for its sub-species.

Each organism has a body which covers a region of the landscape within which it can feed. The size of an organism is defined as the n-dimensional hyper-volume it covers on the landscape. Multiple organisms can cover the same region of landscape.

Rule Implementation: A rule is completely described by {Condition, Action, Prediction, Strength, Experience, Mature, Starved, CanBreed, Damaged}. Condition is the conjunction of n half-open intervals $[l_i, u_i)$, each represented as an ordered pair of bounds, which is satisfied if $l_i \leq x_i < u_i$. The entire condition describes a hyper-rectangle in the space of possible inputs. The generality of a rule is the hyper-volume of its condition. Hyper-volumes are maintained in normalized form such that a volume of 1 covers the entire known landscape. Action is actually implied by a rules containing population where it is represented as an integer code. Prediction is a real number representing the expected reward level should Action be executed. Strength is a real number representing the rules state of health. Experience is an integer representing how many times the rule has been in the action set. The remaining four attributes are Boolean flags.

Table 4.1: Analogy key 1

A-Life term	LCS Analogue
Landscape	Map of reward levels over the space of system inputs for a single action
Organism	Rule / Classifier
Body	Rule / Classifier condition
Size	Rule generality
Food, 1 unit	Data-point / training example
Food-type	Reward level
Major species	Rules predicting for the same action
Population	Set of all rules predicting for the same action
Sub-species	Rules predicting same reward for the same action

4.3.2 Performance System

Implementation: For each population, all rules whose condition is satisfied by the input vector are added to a potential action set; there is a separate potential action set for each possible action. If any of the potential action sets are empty, one of the corresponding actions is selected at random for exploration. If no potential action set is empty then with probability P(explore) an action is chosen at random for exploration. With probability 1−P(explore) this operating cycle is treated as a test problem and the rules in each potential action set take a weighted vote on what the reward level will be if their corresponding action is executed. The weight for each rule is calculated as its strength per unit hyper-volume of its condition. A rule that is not mature is allowed to vote, but the weight of its vote is reduced by a factor of 10. The action with the highest predicted reward level is selected. This exploitation scheme is based on the prediction array used in XCS but with strength per unit hyper-volume replacing fitness. ALCSR presents its selected action to its environment which returns one of a finite set of real-valued reward levels. All experiments so far have used {+1000, -1000}. In test problems, this is the end of the operating cycle.

4.3.3 Accuracy Pressure

A-Life Model: Each single unit of food is evenly split m ways between all m organisms covering the location of the food on the landscape corresponding to the action just executed by the system. Whether or not the consuming organisms benefit from this food depends upon how well adapted they are to metabolizing food of that type. The gain in strength to each organism is offset by a fixed cost of metabolism and the potentially fatal loss of strength that occurs if an organism is poisoned by food of a type it is not well adapted for. The less well adapted the more loss occurs. An organism whose strength becomes negative is considered to have died. This is the death pressure against mal-adaptation. The fixed cost of metabolism introduces an additional pressure against overcrowding.

Table 4.2: Analogy key 2

A-Life term	LCS Analogue
Feeding organisms	Rules in the action set
Number of times fed	Action set experience
Degree of adaptation	Accuracy of reward prediction
Poisoned by food	Inaccurate reward prediction
Starved	rl.Strength ≤ 0
Damaged	rl.Error > 0.01
Organism dies	Rule removed from population

Reinforcement Implementation: The strength of each rule rl in the action set is updated using: rl.Strength += $(1 -$ rl.Error $\cdot$ ErrorCost$)/m$ $-$MetabolismCost

rl.Error is the rules prediction error as a fraction of the reward actually returned; ErrorCost is the damage that would occur for a 100% error; m is the number of rules in the action set; MetabolismCost is a flat cost set to 1/50. If the action set contains more than 49 rules, then they cannot gain in strength even if they all make zero error, and it is possible for the weaker rules to starve to death due to overcrowding. ErrorCost is set to 2000 to provide a very strong death pressure against inaccurate rules; it is insensitive to relative accuracy (differing from XCS).

4.3.4 Genetic Generalization Pressure

Creating More General Rules

A-Life Model: A breeding pool is formed by an organism and its k_b-nearest neighbors on the landscape any time a sufficient proportion of them satisfy breeding criteria (currently half or more). Breeding criteria are that all organisms that will participate in a breeding pool must be of the same sub-species and must have their strength and experience above fixed thresholds per unit hyper-volume of their bodies. Setting these thresholds per unit hyper-volume is intended to result in all organisms being fairly evaluated for breeding. Furthermore, if food is uniformly distributed over the landscape then the breeding rate should be about the same in all niches regardless of their size. Requiring that a certain proportion of an organisms k_b-nearest neighbors satisfy breeding criteria should focus breeding within niches rather than between them. In our experiments $k_b = 12$.

Once a breeding pool is formed, breeding events occur in it until the breeding criteria are no longer satisfied by enough of its members. A breeding event consists of two distinct parents randomly selected (but not removed) from the breeding pool producing a single offspring of the same sub-species by crossover, mutation and strength transfer. Crossover provides the genetic pressure toward the production of increasingly large organisms, i.e. offspring tend to be larger than their parents. If the resulting offsprings body precisely duplicates that of another organism then small mutations are applied until the offspring is unique.

Implementation: MatureLevel = 500. Crossover: the condition of the offspring is formed such that with probability 2/3 in each dimension, it will be a generalization of its parents.

With probability 1/3 in each dimension, the offsprings condition will be a copy of one of its parents, picked randomly as in uniform crossover. Mutation: with probability 1/12 each condition bound is independently mutated by a random amount, at most 1% of the corresponding inputs value range. Strength Transfer: the strength each parent transfers to the offspring is calculated as:

P_1.Transfer = stf · BreedLevel · Min(P_1.Gen, O.Gen · P_1.RelSize)

P_2.Transfer = stf · BreedLevel · Min(P_2.Gen, O.Gen · P_2.RelSize)

P_1 and P_2 denote the parents; O denotes the offspring; Gen denotes generality which is a normalized hyper-volume; stf denotes StrengthTransferFraction = 0.43; BreedLevel = 376; and:

P_x.RelSize = P_x.Gen / (P_1.Gen + P_2.Gen)

The offsprings experience is set to zero and all its flags are set to False.

Table 4.3: Analogy key 3

A-Life term	LCS Analogue
Distance between organisms:	Euclidean distance between geometric centers of rule conditions
Mature	rl.Experience > MatureLevel · Generality
CanBreed	rl.Strength ≥ BreedLevel · Generality

Removing Less General Rules

A-Life Model: Organisms of the same sub-species that cover the same region of the landscape are in competition for the same units of food, and so the larger organisms constantly initiate combat with the smaller ones, attempting to kill and eat them. However, such attacks are only made by mature organisms against other mature organisms, and are only successful if the aggressor has greater strength than its intended victim. Conveniently, the cost to the aggressor of killing another organism is equal to that organisms strength and therefore precisely replenished by eating its dead body. A further constraint on successful attacks is that the aggressor must fully engulf (logically subsume) its victim. (This could be relaxed to investigate fuzzy subsumption schemes and a stochastic combat model.) In the present model, all constraints are enforced and all possible fights are always played out among the feeding organisms on each system cycle. This is the death pressure against needlessly small organisms (over specific rules).

4.3.5 Covering Pressure for Specificity

A-Life Model: When an organism dies due to eating too much of the wrong type of food its corpse remains inactive on the landscape, and those parts of its body that caused it to die are wizened and infertile. However, any part of the organisms body that had allowed it to gain strength from feeding is considered to have the potential to sprout new organisms of the same sub-species. Such sprouting only occurs in the absence of other rules of that sub-species on that part of the landscape. Sprouting is intended to produce new organisms with at least one surface of their body likely to be in the region of the boundary between different food-types on the landscape. This is the pressure toward the production of organisms that are small enough to eat only a single type of food (rules that are specific enough to make no prediction errors). Additionally, sprouting should increase the chance of introducing useful alleles into the sub-species' gene pool as new organisms are created progressively closer to the food-type boundaries the sub-species cannot cross.

When ALCSR is initiated there are no organisms and the landscapes are completely undiscovered so a seeding process is required to get the populations started. Seeding is stopped when there are enough corpses for sprouting to take over.

Table 4.4: Analogy key 4

A-Life term	LCS Analogue
Seeding	Traditional random covering
Sprouting	ALCSR Adaptive covering
Food-type boundary	Concept decision surface

Implementation: This step is only performed if the system has just explored the action corresponding to an empty potential action set. A new rule is created with zero strength, zero experience and all flags set to False. Its prediction is set to the reward level just returned. Defining the new rules condition is more complicated. Each population has associated with it a set, termed mausoleum, for storing rules that have died due to making inaccurate predictions. The new rules condition bounds are randomly generated as:

$l_i = x_i - \text{Rnd}[0, \text{LowerCoverFraction}_i \cdot r_i)$ and

$u_i = x_i + \text{Rnd}[0, \text{UpperCoverFraction}_i \cdot r_i)$ where r_i is input i's range

If there are fewer than 20 rules in the relevant mausoleum, then all CoverFractions are set to 0.5. If there are 20 or more rules in the relevant mausoleum then the CoverFractions are set by adaptive covering. This is the most challenging part of the artificial life model to implement because it requires the rules in the mausoleum to know in which regions of their conditions their predictions were accurate or inaccurate, which must be estimated.

Each dimension i of a rules condition has four attributes: StrengthCenter_i, WeaknessCenter_i, $\text{LowerCoverFraction}_i$ and $\text{UpperCoverFraction}_i$. StrengthCenter_i and WeaknessCenter_i are maintained as the average ordinate value in each dimension i of all correct and incorrect predictions respectively. When a rule dies with Damaged set to True, it is added to the appropriate mausoleum and its CoverFractions are set to represent its size in each dimension as fractions of each full attribute range. The dimension in which StrengthCenter and WeaknessCenter are furthest apart is guessed as being most likely to span a decision surface, and one of the rules CoverFractions in that dimension is reduced accordingly by a factor of 2. If the StrengthCenter has a lower value than the WeaknessCenter then the UpperCoverFraction is chosen as the one to be reduced, otherwise the LowerCoverFraction is reduced. The CoverFractions of the dead rules k_c-nearest neighbors in the mausoleum are updated to reflect this new information using a Widrow-Hoff update in which the learning rate is inversely proportional to the distance of the neighbor.

When adaptive covering creates a new rule, the k_c corpses in the mausoleum nearest to the input vector X take an inverse-distance weighted vote on what values should be assigned to the new rules CoverFractions, which are then used to generate the condition bounds. If a condition lower or upper bound is generated that lies outside the range of values that ALCSR has experienced in that dimension, the bound is set to the minimum or maximum value experienced respectively. Note that this increases the probability of generating a bound equal to the range limit as discussed in [5]. In our experiments $k_c = 5$.

4.4 Initial Experiments

In [7], Wilson presented the results of two experiments in which XCSR learned a real-valued version of the Boolean 6-multiplexer function. Instead of 6 binary values, the input vector consists of 6 real values $0.0 \leq x_i < 1.0$, generated randomly with uniform distribution. In each dimension i a decision threshold is used to interpret the component value as 0 or 1 so that the multiplexer can be applied. In order to present our initial results in an established context, Wilsons experiments have been repeated using the implementation of ALCSR described in Section 4.3, run for 300,000 system cycles with P(Explore) = 0.9. As in [7], each experiment was run five times. Results with ALCSR were found to vary very little between runs, so the figures below present results from single representative runs.

In the following results percentage performance is the classification rate over the previous 50 test problems. Where 50 test problems do not provide sufficient precision to report results, other measures are stated specifically. System error is calculated as a moving average, also over the previous 50 test problems, and is presented as a fraction of the reward range (2000) in these experiments.

Experiment 1: All decision thresholds are set to 0.5. In [7] Wilson reports that XCSR achieves a maximum performance of approximately 98% after about 15,000 explore problems. Figure 4.1 presents typical performance and system error for ALCSR. In 15,000 explore problems ALCSR achieves a lower performance of approximately 95%, but continues to improve

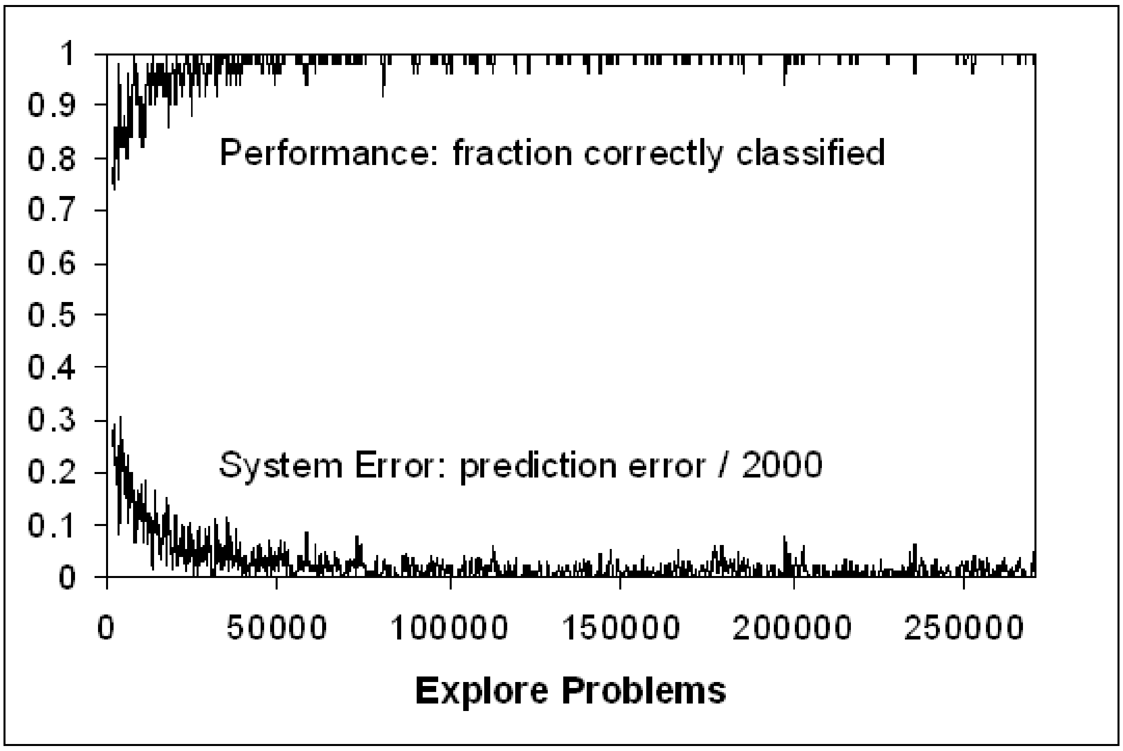

Figure 4.1: ALCSR Performance and System Error in Experiment 1

reaching 98% after about 35,000 explore problems. For the last 10,000 test problems in a run (after about 180,000 explore problems), ALCSR typically provides the correct classification for more than 9,970.

Figure 4.2 shows how the population size varies during a typical run, and also the number of births and deaths. The total rule count peaks around 2,000 after about 20,000 explore problems and then falls to around 535 by the end of the experiment, although this varies more than the performance results: the smallest final population was 452, and the largest was 607.

Experiment 2: The decision thresholds are set at 0.25 and 0.75 in alternate dimensions which skews the probability of generating example points in the different niches. In [7]

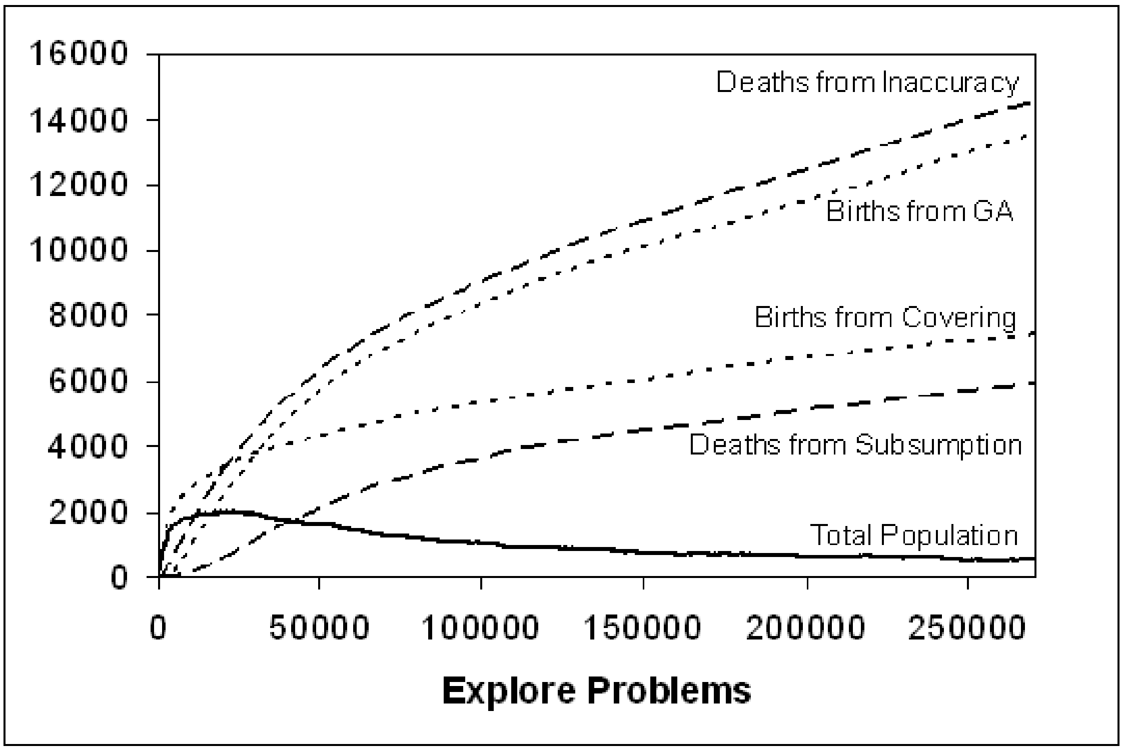

Figure 4.2: ALCSR Population Dynamics in Experiment 1

Wilson reports that XCSR achieves a maximum performance of approximately 93% after about 10,000 explore problems. Figure 4.3 presents typical performance and system error for ALCSR. In 10,000 explore problems ALCSR also achieves 93% performance, but again, continually improves thereafter, reaching 98% after about 50,000 explore problems. Over the last 10,000 test problems in a run, ALCSRs performance is the same as in Experiment 1.

Figure 4.4 shows how the population size varies during a typical run, and also the number of births and deaths. The total rule count peaks around 2,100 after about 27,000 explore problems and then falls to around 950 by the end of the experiment. The smallest final population was 859, and the largest was 1112.

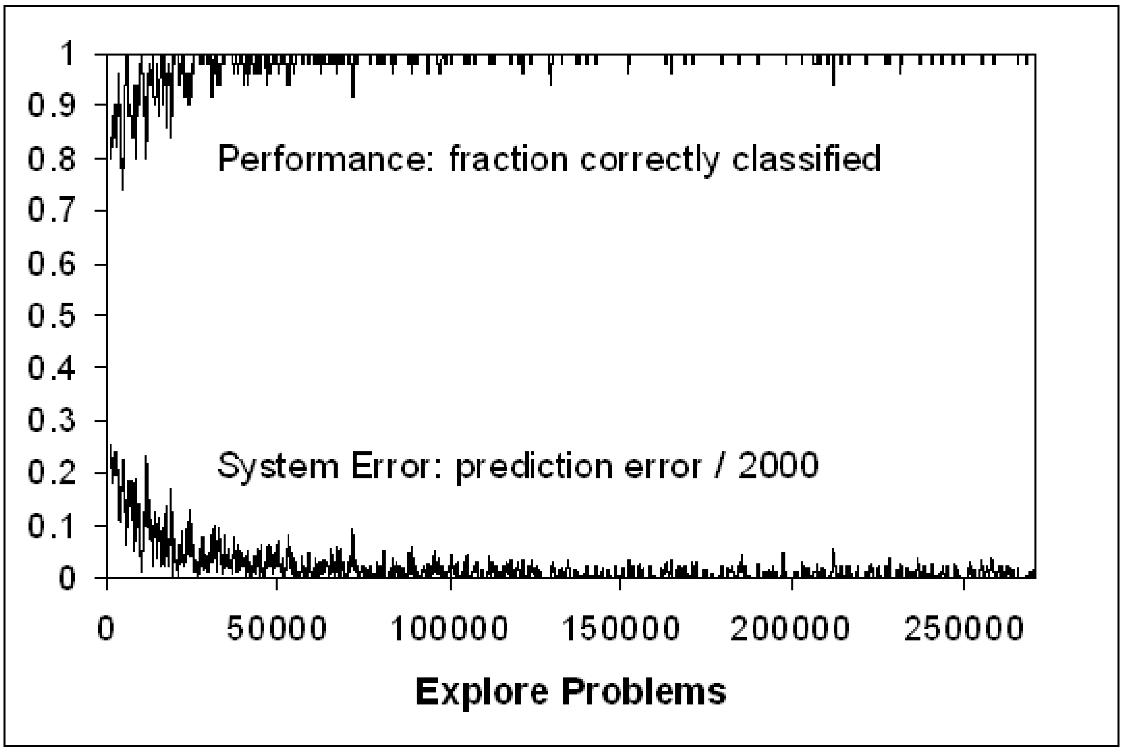

Figure 4.3: ALCSR Performance and System Error in Experiment 2

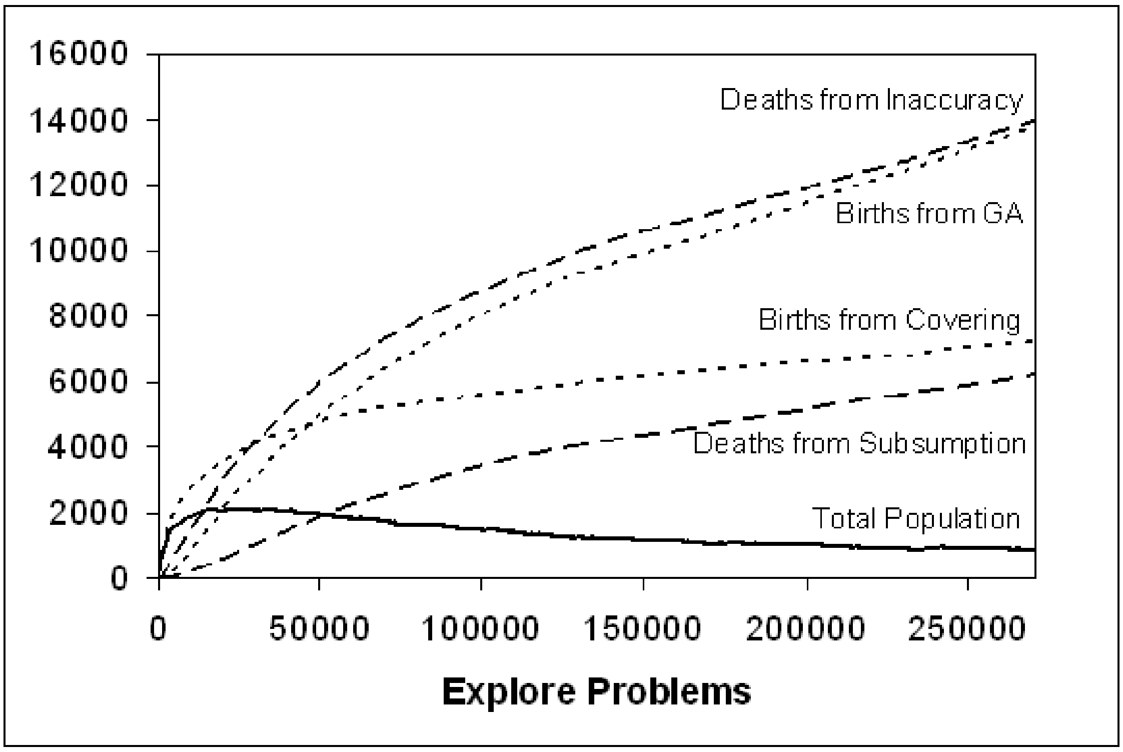

Figure 4.4: ALCSR Population Dynamics in Experiment 2

4.5 Discussion

The version of XCSR used in [7] employed a center-spread representation for its condition bounds, while ALCSR uses ordered bounds. In [5], Stone and Bull investigate these representations for XCSR and also an unordered bounds representation, providing many useful insights. They found system performance for the real-valued 6-multiplexor consistent with Wilsons results in [7] and affected very little by the choice of representation (although other measures were affected as was performance on other problems).

The fact that ALCSRs best reached classification performance in Experiment 2 is as good as Experiment 1 is encouraging with respect to the Niching goal (5). However, Experiment 2 does cause ALCSR to learn at a slower rate than in Experiment 1, and produce slightly larger populations.

Compared to XCSR, ALCSR learns much more slowly but eventually improves beyond XCSRs maximum performance achieving 99.7% measured over 10,000 test problems, which is encouraging with respect to the Performance goal (4). An analysis of why ALCSR still makes any mistakes at all would be of interest.

Examinations of the final populations from both experiments have identified very good representatives of all niches, even the smallest, which is encouraging with respect to the Model goal (3). However, the majority of the rules in the final populations are very specific rules clustered around the decision surfaces in the input space. The adaptive covering scheme was designed to have this effect so that increasingly useful condition bounds (alleles) would continue to be discovered throughout the run. However, in its current implementation adaptive covering appears to be too aggressive for the system to achieve the Optimality goal (6), unnecessarily bloating the population. About half the rules in each final population are so specific that they very rarely participate in system operation and are mostly a drain on computing resources. If the persistence of these rules could be avoided, the Optimality goal (6) would be within reach.

4.6 Conclusions and Ongoing Work

ALCSR has a number of features that make it unusual among learning classifier systems. Specifically: ALCSR identifies individuals suitable for breeding by the use of speciation and by exploiting the natural topography of the problem; ALCSR employs an endogenous fitness scheme; GA timing and population size are emergently controlled; ALCSR uses adaptive covering which continues throughout training to provide the GA with increasingly useful alleles so mutation is not relied upon for allele discovery and is just used to avoid duplicate conditions; Since there are no duplicate rules, numerosity is obviated.

Although some of the goals set out in Section 4.2 have not yet been addressed, inroads have been made towards goals 1, 3, 4, 5, 6 and 7, and Section 4.2 represents an agenda for ongoing research. Additionally, a minimal set of necessary system parameters for ALCSR needs to be identified, and self-adaptive parameter schemes investigated.

The immediate priority for ALCSR is an analysis of the current implementations population dynamics in order to understand and correct undesirable characteristics such as slow learning and over specialization at the decision surfaces. Such analysis will probably result in modifications to the implementation and possibly to the artificial life model as well. The modified ALCSR should also be tested against a wider range of problems such as more challenging multiplexers and the Checkerboard problem described in [5].

4.7 Acknowledgements

I am grateful to Alwyn Barry, Stewart Wilson, Khaled Rasheed, Don Potter, Brian Smith and Daniel Tuohy for helpful comments and discussions about this work and/or related issues.

4.8 References

[1] Booker, L.B. (2000) “Do We Really Need to Estimate Rule Utilities in Classifier Systems?” Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Learning Classifier Systems: From Foundations to Applications, volume 1813 of LNAI, Springer-Verlag, Berlin, pp 125-141, 2000.

[2] Booker, L.B. (2001) “Classifier systems, endogenous fitness, and delayed rewards: A preliminary investigation” In Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E., eds., Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pages 921-926, San Francisco, California, USA, 7-11 July 2001.

[3] Butz, V.B., Wilson, S.W. (2001) “An algorithmic Description of XCS”, In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems, pp 253-272

[4] Holland, J.H. (1995) “Hidden Order: How Adaptation Builds Complexity” Helix Books.

[5] Stone, C., Bull, L. (2003) “For Real! XCS with Continuous-Valued Inputs” Evolutionary Computation 11(3): 299-336

[6] Wilson, S.W. (1995) “Classifier Fitness Based on Accuracy” Evolutionary Computation 3(2): 149-175

[7] Wilson, S.W. (2000) “Get Real! XCS with Continuous-Valued Inputs”. Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Learning Classifier Systems: From Foundations to Applications, volume 1813 of LNAI, Springer-Verlag, Berlin, pp 209-219