AN ABSTRACT OF THE THESIS OF

Caryl J. Westerberg for the degree of Masters of Science in Computer Science presented on May 21, 1993.

Title: Investigation of Automatic Construction of Reactive Controllers

Abstract approved: Redacted for Privacy

Professor Bruce D'Ambrosio.

In real-time control systems, the value of a control decision depends not only on the correctness of the decision but also on the time when that decision is available. Recent work in real-time decision making has used machine learning techniques to automatically construct reactive controllers, that is, controllers with little or no internal state and low time complexity pathways between sensors and effectors. This paper presents research on 1) how a problem representation affects the trade-offs between space and performance, and 2) off-line versus on-line approaches for collecting training examples when using machine learning techniques to construct reactive controllers. Empirical results show that for a partially observable problem both the inclusion of history information in the problem representation and the use of on-line rather than off-line learning can improve the performance of the reactive controller.

Investigation of Automatic Construction of Reactive Controllers

By Caryl J. Westerberg

A Thesis submitted to
Oregon State University

in partial fulfillment of the
requirements for the degree of

Masters of Science

Completed May 21, 1993
Commencement June 1994.

Approved:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Head of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis presented May 21, 1993

Typed by Caryl J. Westerberg

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Bruce D'Ambrosio, my advisor, for his wonderful ideas, advice, and patience. I would also like to thank the members of my committee, Tom Dietterich, and Prasad Tadepalli for their advice and insight.

I would especially like to thank Jim Holloway and Dietrich Wettschereck for their detailed comments and suggestions on drafts of this document.

Finally, I would like to dedicate this work to Patrick McDaid whose support and encouragement made this whole thing possible.

This research was funded in part by NSF and Air Force Office of Scientific Research under grant number 90-0348.

Table of Contents

Chapter		Page
1	Introduction	1
1.1	Related Work	4
1.1.1	Kaul's Work	4
1.1.2	Other Related Work	6
1.2	Objectives	7
1.3	Research Approach	9
1.4	Summary of Results	10
1.5	Outline of Paper	11
2	Testbed	12
2.1	Simulator	13
2.1.1	The Simulated Circuit	13
2.2	The On-Line Maintenance Agent (OLMA)	15
2.2.1	Reactive and Deliberative Controllers	16
2.2.2	Learning Controller	17
2.2.3	Testing	18
3	Approach	21
3.1	Problem Representation	21
3.2	Data Collection	25

Table of Contents (Cont.)

Chapter — Page

3.2.1 Q-Learning 26

3.2.2 C4.5 27

3.3 Experimental Approach 28

4 Results 30

4.1 Problem Representation Performance 32

4.2 Influence of On-line Learning 35

4.3 Time and Space Complexity 37

5 Discussion 39

5.1 Problem Representation 39

5.2 Off-line versus On-line 41

5.3 Space and Performance Tradeoffs 43

5.4 How Do the Controllers Compare? 43

5.5 Conclusions 46

5.6 Future Directions 47

Bibliography 48

Appendix A Validating Predicted Performance 50

List of Figures

Figure		Page
1.	The half adder circuit	14
2.	Off-line collection of training examples	17
3.	On-line collection of training examples	19
4.	Testing the learning agent	20
5.	Problem representation methods	23
6.	Simulating On-line learning algorithm using C4.5	27
7.	Average predicted performance for each problem representation when trained off-line with various sizes of training sets.	31
8.	Average predicted performance of off-line versus on-line learning.	36
9.	Example of off-line training data for the Basic representation with a failure of gate NOR1.	41
10.	Example on-line training data and the corresponding control actions for the Basic representation with a failure of gate NOR1.	42
11.	Space versus performance	44
12.	Comparison of the actual runs of the better controllers	45
13.	Regression line and scatter plot of cumulative cost and predicted performance for a simulated run of 1200 cycles.	53
14.	Regression line and scatter plot of predicted performance versus number of failures for a simulated run of 1200 cycles.	54

List of Tables

Table | Page

1. Average and standard error for each problem representation when trained off-line on 6000 training examples 32
2. Features used by the decision trees. 34
3. Probabilities that off-line and on-line means are equal from a statistical t-test. 35
4. Time and space complexities of each method when trained on 6000 training examples. 37
5. Amount of history information and resulting predicted performance for each representation in the half adder circuit domain. 40

Investigation of Automatic Construction of Reactive Controllers

Chapter 1

Introduction

In real-time control problems such as robot navigation, operating a power plant, or operating a chemical refinery, the value of a control decision depends not only on the correctness of the decision but also on the time when that decision is available [Sch91, PB91]. Work in intelligent real-time systems has begun addressing the time critical nature of such real-time systems through the use of reactive controllers, that is, controllers with little or no internal state and low time complexity pathways between sensors and effectors. These reactive controllers are created by compiling into rules the information about the state of a controlled system and a computed decision based on the state of that controlled system. The reactive controllers use these rules in choosing control actions for the controlled system [Sch87, LNR87, Kau91].

The work presented here uses machine learning techniques to explore approaches to constructing reactive knowledge-based controllers from deliberative solutions for the On-Line Maintenance Agent (OLMA) [DF91]. In his paper [Gin89], Ginsberg claims that to compile such a reactive solution is infeasible and that

> "an effective approach to acting in uncertain domains...must always be to look, to think, and only then to leap."

Ginsberg claims that when attempting to analyze all possible situations in advance and storing appropriate responses in a table, this table must grow exponentially with the complexity of the domain.

In this work we applied a machine learning program, C4.5, to generate decison trees that store the relevant situations and appropriate responses for reactive controllers. Since these decision trees can represent all possible situations, they can implement the table that Ginsberg describes. However, if C4.5 determines that a feature is not useful in making a decision, that feature is not tested, resulting in a smaller decision tree. We measured the size and performance of all the reactive controllers we created and determined that for our problem domain, the size of a reactive controller need not be exponential in the size of the problem representation.

Because the OLMA maintains a partially observable real-time control system, this experimental study explores a problem that is more complex than the problems discussed in [Gin89] for the following two reasons:

1. The controlled system is comprised of several components which can fail in multiple ways. Each of these failures represents a class of problems. If any

one of these problem classes is not addressed by the controller, then the system is likely to get into an unresolved, broken state where one or more failed components are never repaired.

2. The controlled system is a hidden Markov decision process. That is, the entire state of the controlled system is never available to the controller for problem solving.

The task of the OLMA is to provide control actions for a continuously running system. In continuously running controlled systems, failure of the controller to rectify at least one instance of each class of potential problems may result in the controlled system entering an irrecoverably broken state. For example, a problem class might consist of a type of failure for a component. If the controller has no rule or procedure for addressing that problem class, when the component fails, the component may remain broken throughout the life of the system. When constructing reactive controllers for continuously running systems the question becomes: *Can we build reactive controllers that will find a solution for every problem class?*

This problem of the controlled system entering an unresolvable broken state is compounded by the limited visibility the controller has of the state of the controlled system. Without complete state information, the controller must use information from earlier states to estimate the current state [DW91]. When this history information is incorporated, the space needed to represent all possible situations becomes infeasibly large. Our research addresses this space problem by drawing from work on indexical representations [AC87, WB90] in that "a system need not name and describe every object in the domain, but should register information only about objects that are relevant to the task at hand." [WB90] In the research discussed here, the relevant question becomes: *What history information is necessary to achieve an adequate representation of the current state of the controlled system?*

1.1 Related Work

1.1.1 Kaul's Work

The work presented here is a continuation of the research of Lothar Kaul's Master's thesis [Kau91]. In his thesis, Kaul explored the feasibility of automatically constructing reactive systems from a deliberative solution for the OLMA.

He used inductive learning techniques, namely Cascade Correlation [FL90], and C4.5 [Qui86] to generate reactive solutions. The training examples were generated by a deliberative controller and consisted of the previous act, the observations from the controlled system, and the prior probability distribution of the state of the controlled system. The output classes consisted of a control action and a posterior probability distribution describing the state of the controlled system [Kau91].

Kaul used a batch or off-line approach when compiling reactive solutions. He collected the training examples from the decision theoretic controller as it maintained a continuously running system, then used these training examples as input to various inductive learning programs. Using these learning algorithms the first stage of Kaul's approach predicted the control action the deliberative agent would have taken given the observations and the prior probability distribution. The second stage generated a single value of the joint probability distribution given the sense input and the prior probability distribution. In this approach the controller uses an internal representation of the probability distribution to maintain the controller's beliefs of the current state of the controlled system. After generating reactive solutions with this approach, the resulting solutions were then evaluated with respect to their size and error rate.

Although Kaul found that compiling reactive solutions was feasible, the primary drawback with this off-line learning approach stemmed from the distribution of training examples. These training examples consist of inputs and control actions. The possible types of control actions are *Probe*, *Replace*, and *Nothing.* Because failures occur infrequently, the *Probe* and *Replace* actions performed by the deliberative controller, as reflected in the training examples, are also infrequent as compared to the number of *Nothing* control actions. Due to the generalizing nature of inductive learning methods, the overriding learned response was to issue *Nothing* control actions. To address the problem of infrequent actions, the training data was filtered to delete a fraction of the nothing actions thereby increasing the frequency of the *Probe* and *Replace* actions. This approach proved more successful. Occasionally, however, a learned controller would make a mistake. This mistake would take the system into states that had not arisen during training. Consequently, the learned rules could not handle these states properly, resulting in poor performance [Die91].

1.1.2 Other Related Work

In defining the functionality of the reactive controller, our work draws from research in real-time systems in which the agent is embedded in the environment. Some of the earlier work in this area is work done by Brooks, and Agre and Chapman. Brooks proposed a layered, subsumption architecture for controlling robots [Bro68]. Agre and Chapman investigated the use of indexical representations and combinatorial circuitry to implement a game playing penguin in Pengi [AC87].

Then, Watkins introduced Q-learning in which an agent explores its environment and learns an optimal policy for behavior from rewards in the environment [Wat89]. With the advent of Q-learning, several researchers have used reinforcement learning methods such as Q-learning to develop situated, learning agents. Kaelbling addressed the problem of designing algorithms for reinforcement learning in embedded systems [Kae90]. Mahadevan and Connell combined a subsumption architecture with reinforcement learning for the automatic programming of robots [MC91]. Whitehead applied concepts from indexical representations combined with reinforcement learning to the adaptive control of sensory-motor systems [Whi91].

1.2 Objectives

The overall objective of OLMA research is to develop a embedded, real-time agent which uses cooperative deliberative and reactive controllers for situated action. The goals of the OLMA research are commensurate with the goals of reactive planning and situated action where "the general idea is to build embodied agents that behave intelligently in physical surroundings" [Sch89]. As part of the OLMA, the current deliberative controller is able to provide appropriate control actions [DF91], but this deliberative controller uses a decision theoretic approach which is NP-hard [Coo87]. As a result, this deliberative controller is unable to make decisions as quickly as real-time systems require.

As part of the OLMA research, the objectives of our work, similar to Kaul's, are to explore automated construction of reactive controllers using the input/output behavior of the deliberative controller, and to study the resulting space/performance tradeoffs. The desired behavior of the reactive controllers is to approximate the function of the deliberative controller and yet produce an answer in much less time. Unlike Kaul's work, our work 1) explores both an off-line and on-line approach to learning, 2) maps the current state directly to the next action, and 3) investigates the use of previous actions and observations as history information in the problem representation.

By providing training examples that are tailored specifically to the learning task, we believe that learning is likely to be more successful. Such tailoring is enabled through embedding the learning component into the environment and performing on-line learning. This study explores the effect on performance that this embedded, on-line learning has compared to off-line learning when generating a reactive controller.

In addition to tailoring the distribution of training examples through on-line learning, this approach differs from Kaul's in the functional definition of the reactive controller. Kaul used an internal model to represent the state of the controlled system [Kau91]. The goal of our approach is to find the minimal amount of space needed when compiling a reactive controller. We approach this by mapping the observations directly to the actions circumventing the internal representation of the problem domain.

In lieu of an explicit internal representation of the problem domain, our approach explores the use of various amounts and aspects of history information. We hope to determine if, by capturing and using the right history information, one can successfully build a reactive controller in which the size does not necessarily grow exponentially in the size of the decision theoretic method and can keep a controlled system from entering an unrecoverable broken state.

The objectives of this work can be summarized as follows:

- To study the use of history information in problem representation, and to study on-line versus off-line approaches when automatically constructing reactive systems.

- To examine the space and performance tradeoffs in an intelligent reactive controller.

1.3 Research Approach

This research uses learning controllers to facilitate development and testing of potential reactive controllers. The research began with the development of a series of problem representations which use an increasing amount of history information. Then, for each problem representation, learning controllers were generated using both off-line and on-line learning. The research has been conducted by varying the following parameters:

1. Problem Representation: The amount of history used in each problem representation varies from none to several previous observations and control actions. Each problem representation also uses the current observation.
2. Methods of Collecting Training Examples: Training examples are collected using either on-line or off-line learning methods. In off-line learning the training examples are collected in the first step, and the learning controller is trained in the second step. In on-line learning, a training example is generated by capturing observations from the controlled system and the control action recommended by the deliberative controller. This training example is included with previous training examples and a learning controller is trained with this set of training examples. Then, this learning controller chooses a

control action based on the same observations as in the training example. The controlled system executes the learning controller's control action and generates a new set of observations, repeating the cycle.

Once each learning controller was generated, a quantitative measure was used to predict the number of significant problem classes each controller could solve. In this study the significant problem classes consisted of single component failures. This quantitative measure presented examples from the significant problem classes and recorded the number of problem classes in which the learning controller gave at least one correct control action. With this predicted performance measure we compared the performance of each problem representation and of off-line versus on-line learning. Then, for each problem representation we found both space and time requirements.

1.4 Summary of Results

The results show the following:

1. In general, a greater amount of history information does improve the performance.

2. On-line learning approaches can provide some performance improvement over off-line learning.

3. The inductive learning program, C4.5, is able to perform sufficient generalizations such that the actual space requirements for each problem, as mea-

sured experimentally by the size of the tree, is significantly smaller than the theoretical worst case space requirement.

1.5 Outline of Paper

The remainder of this paper is structured as follows: Chapter 2 describes the testbed for the OLMA. Chapter 3 presents the problem representations, the off-line and on-line approaches to training data collection, and the experimental approach. Chapter 4 presents the results from experiments described in Chapter 3. Finally, Chapter 5 discusses the results and suggests some directions for future work.

Chapter 2

Testbed

The testbed was developed as a modular and flexible means for exploring a variety of agents and problem domains. The testbed consists of a simulator, an agent, and the communication links between the simulator and the agent. The simulator provides many of the operational characteristics and behaviors of a real system which, in this case, is a digital circuit. The agent is an instantiation of the OLMA which consists of a reactive controller, a deliberative controller, and for purposes of this study, a learning controller. The goal of this OLMA is to minimize over time the cost of maintaining the simulated circuit. The development and implementation of the testbed was a coordinated effort among Bruce D'Ambrosio, Tony Fountain, and myself.

For purposes of this study, processing within the testbed alternates between the simulator and the agent. The simulator first processes control actions from the agent, then executes a complete cycle of the simulated system. At the end of this cycle the observable values, including any requested probe values, are made available to the agent as observations. The agent retrieves these observations from the simulator, and generates a control action for use by the simulator.

2.1 Simulator

We designed and developed the simulator as a discrete event simulation tool to model a variety of continuously running controlled systems. As a part of the testbed, the simulator provides many of the operational characteristics and behaviors of an actual system without incurring the cost of using the real system.

Processing in the simulator is conducted through the creation and execution of events. For each cycle of the simulation, the simulator begins by retrieving and processing any control actions it may have received from the agent. After processing these control actions, the simulator begins a process of first removing and executing the event at the top the event queue, then checking whether executing that event triggers other components within the model to generate new events, and finally incorporating these new events, if any, into the event queue. The simulator repeats this process until a condition, specific to the simulated system, signals the end of the cycle.

2.1.1 The Simulated Circuit

In this study the simulator modeled a simple digital half adder (see Figure 1). This four gate circuit consists of two NOR gates, a NAND gate, and an INVERTER. Each of these gates has a probability of failure of .001 per cycle. The

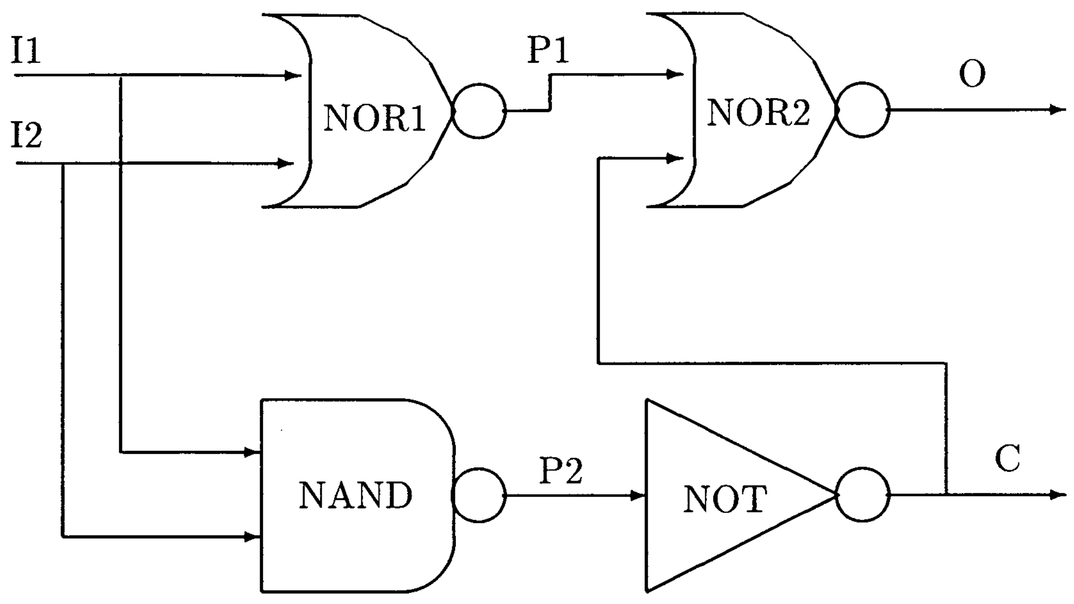


Figure 1. The half adder circuit

circuit has two inputs, I1 and I2; two outputs, O and C; and two probe points, P1 and P2. For this circuit, the simulator accepts seven control actions which include:

- four *Replace* acts, one for each gate in the circuit. A gate is replaced with a properly functioning gate. That new gate can fail immediately.
- two *Probe* acts, each causes the simulator to return the value on the corresponding internal connection line at the end of each cycle. (In Figure 1 these lines are labeled P1 and P2.); and
- one *Nothing* act, where the simulator effects no changes.

At the start of each cycle, a pair of randomly generated 0s and 1s are provided as input to the simulated circuit. This triggers gates NOR1 and NAND to generate and execute events[1], propagating the results to gates NOR2 and NOT. Upon receipt of this input gates NOR2 and NOT trigger and execute, generating new events. Since the output of NOT is connected to the input of NOR2, the events from the NOT gate trigger gate NOR2 a second time and it generates a second set of events. At the end of each cycle the randomly generated input and resulting output values along with any requested probe value are retrieved and made available to the OLMA as observations.

2.2 The On-Line Maintenance Agent (OLMA)

With our OLMA we are investigating a real-time architecture where reactive and deliberative controllers cooperate to monitor and keep operational a continuously running electrical or mechanical system at minimal cost. Costs are incurred by the agent when the circuit runs in a broken state, the agent asks for more information about the state of the circuit (a *Probe* act), or the agent replaces a gate in the circuit (a *Replace* act). For purposes of this study, a cost of 1 is incurred for each cycle the circuit runs in a broken state and is independent of the number of failed components. That is the same cost is incurred if one component is broken or all four components are broken. Also, a cost of 1 is incurred for each probe act, and a cost of 10 is incurred for each replace act. As the controlled system operates, the agent monitors the state of the circuit through the observations from the simulator.

[1]These are the same events as described in section 2.1

When faulty input is observed, the agent uses the probe and replace control actions to isolate the fault and repair the broken circuit.

In addition to the reactive and deliberative controllers, this study incorporates an added component into the OLMA called the "learning controller" to facilitate the development and analysis of reactive controllers.

2.2.1 Reactive and Deliberative Controllers

The reactive and deliberative controllers each consist of a set of decision processes [DF91]. Because we are in the process of understanding reactive decision processes, only the simplest has been implemented. When invoked, if a deliberative decision process is not running, the current reactive decision process initiates a deliberative decision process, otherwise it does nothing [DF91].

For the deliberative controller we have implemented a decision process which explicitly computes the control action having maximum subjective expected utility from a decision model and the given observations, the prior probabilities, and the expected time it has to act [DF91]. This decision process is quite slow; for the half adder domain it takes around one real-time minute to compute a decision. To collect training examples, we adjusted the timing of the testbed, allowing this decision process to compute a decision for every observation.

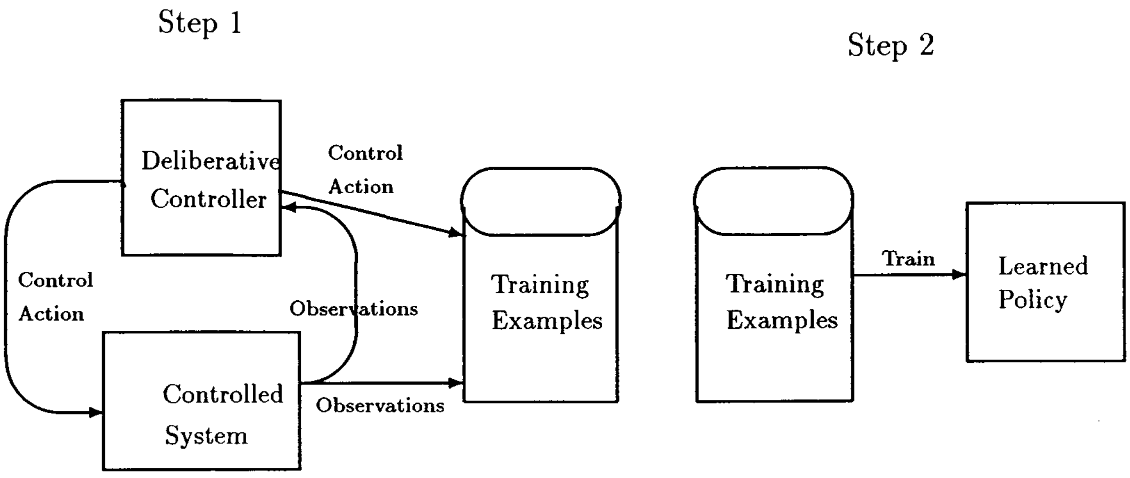


Figure 2. Off-line collection of training examples

2.2.2 Learning Controller

For purposes of this study we incorporated a learning controller into the testbed to facilitate the implementation and study of reactive controllers. The learning controller supports off-line learning, on-line learning, alternate problem representations, and testing of reactive controllers. As a part of the OLMA, the learning controller is invoked after the deliberative controller has decided on a control action and before it has passed this action on to the simulator.

Off-Line Learning

In off-line learning (Figure 2) the learning controller writes both the control action from the deliberative controller along with the corresponding observations to a file as a training example (Figure 2, step 1). After a sufficient number of examples have been collected, the learning controller generates a policy from these training examples using an inductive learning algorithm (Figure 2, step 2). The learning controller uses this policy for controlling the controlled system.

On-Line Learning

In on-line learning (Figure 3), the control action from the deliberative controller along with the corresponding observations are written to a file as a training example, and the learning controller generates a policy from the set of training examples collected so far. After training, the learning controller uses the observations for selecting a control action. It is *this* control action from the learning controller rather than the one chosen by the deliberative controller that is given to the simulator.

2.2.3 Testing

Figure 4 depicts the testing of the learning controller. This is done after either off-line or on-line training has been completed. In this stage, the learning controller uses the observation from the simulator to choose the next action to take; there is no interaction with the deliberative controller.

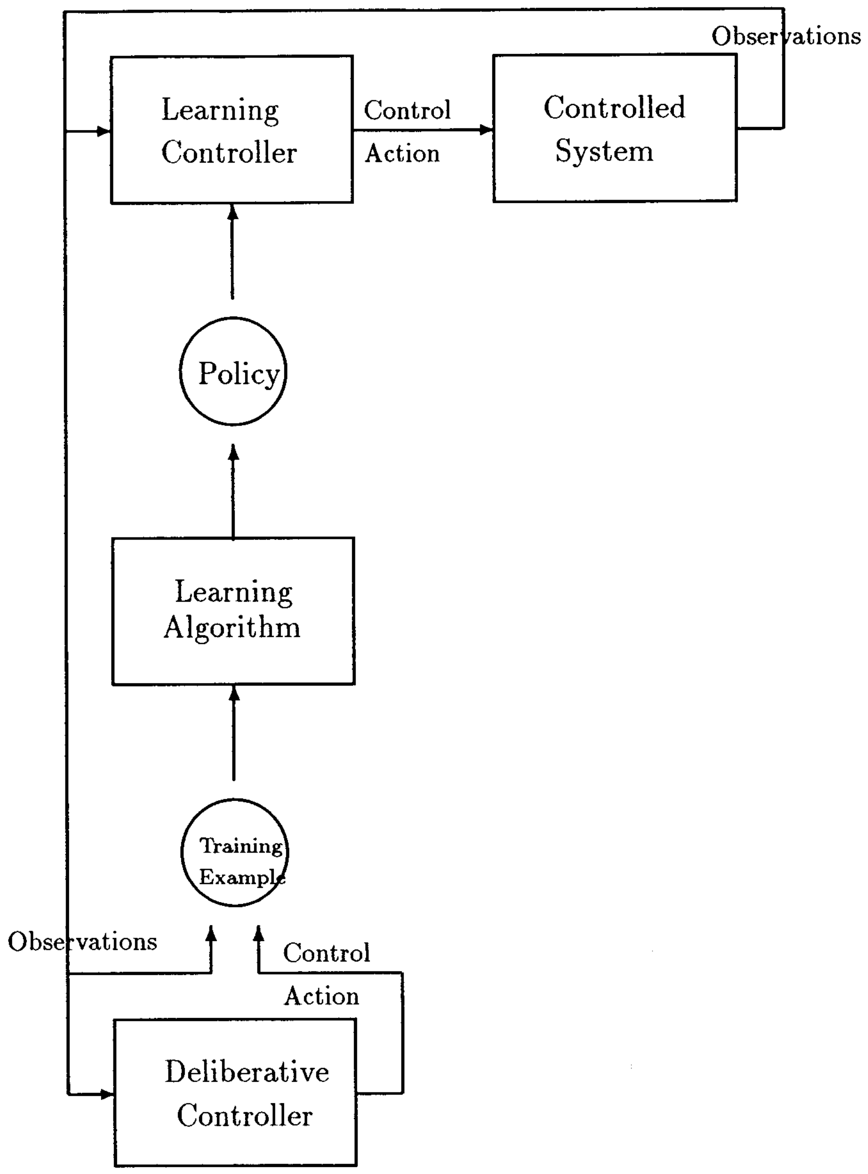


Figure 3. On-line collecting of training examples

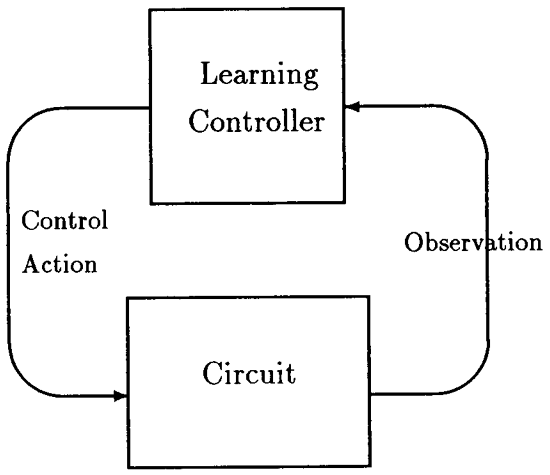


Figure 4. Testing the learning agent

Chapter 3

Approach

This chapter presents, first, the problem representations and methods of data collection employed to construct reactive controllers, and second, the experiments conducted to evaluate these controllers.

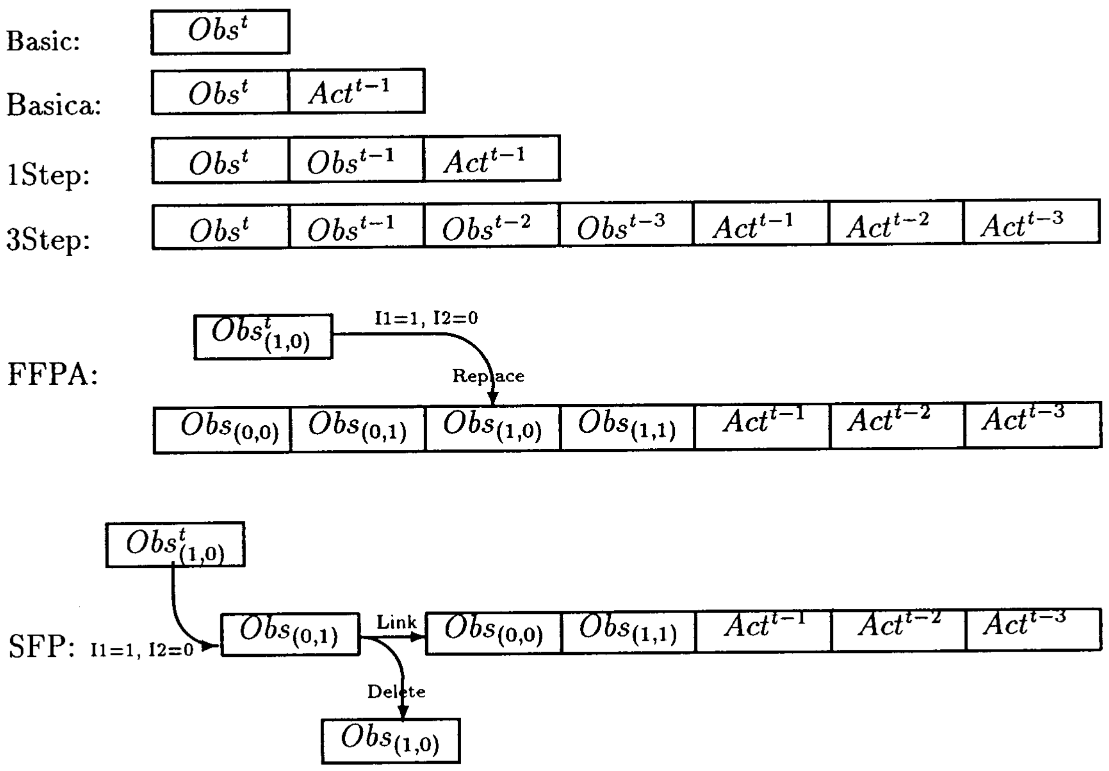
3.1 Problem Representation

In contrast to the earlier work on this project, which used an internal represention of the problem state [Kau91], the emphasis in this work is to minimize the use of space. In order to minimize space, we circumvent an explicit internal representation of the state of the problem domain by exploiting the structure of the problem domain. This relies on the fact that each observation from the controlled system represents a state of a partially observable Markov process. A Markov process has the property that the knowledge of the current state of a system fully captures the current state of that system and no additional knowledge is necessary. A paritally observable Markov process is a process in which all the knowledge of the actual state is not available [Abu88]. Through capturing the *right* history of previous observations and actions, we should be able to recover the actual state. Because it is infeasible to maintain and use all history for most real world problems, our fundamental problem is that we need an effective way to recover and represent the relevant history. When constructing problem representations we explored methods that use information about the patterns and structures evident in the history and problem domain.

To capture state information through the relevant history of observations and actions we explored several methods to encode history. We began by establishing a lower and upper bound on the sizes of problem representations to explore.

Our lower bound simply consists of the current observations and no history. Since the half adder circuit consists of two binary inputs (I1 and I2) which result in a total of four possible combinations of observations, we set the upper bound at a maximum of four observations. We term a collection of all four of these possible combinations of observations a *fingerprint.*

The problem representations are discussed below and presented in Figure 5. The range of problem representations we explored begins with a problem representation (Basic) which uses no history information and ends with two problem representations (FFPA and SFP) which use the fingerprint encodings. The problem representations between Basic and SFP use increasing amounts of history. In each of these encodings except one (FFPA) the observations consist of the observed inputs, outputs, and probe values; in FFPA the observations consist of the output



Basic:
Obs^t
Basica:
Obs^t Act^{t-1}
1Step:
Obs^t Obs^{t-1} Act^{t-1}
3Step:
Obs^t Obs^{t-1} Obs^{t-2} Obs^{t-3} Act^{t-1} Act^{t-2} Act^{t-3}
FFPA:
$Obs^t_{(1,0)}$
I1=1, I2=0
Replace
$Obs_{(0,0)}$ $Obs_{(0,1)}$ $Obs_{(1,0)}$ $Obs_{(1,1)}$ Act^{t-1} Act^{t-2} Act^{t-3}
SFP: I1=1, I2=0
$Obs^t_{(1,0)}$
$Obs_{(0,1)}$
Link
$Obs_{(0,0)}$ $Obs_{(1,1)}$ Act^{t-1} Act^{t-2} Act^{t-3}
Delete
$Obs_{(1,0)}$

Figure 5. Problem representation methods. Except in FFPA, the observations (Obs) consist of the inputs, outputs and probe values (I1, I2, O, C, P1, and P2) from the controlled system; FFPA uses only the outputs and probe values. The actions (Act) are previously executed control actions. The superscripts indicate the time an observation or an action occurred, where t is the current time. The subscripts indicate the values of inputs I1 and I2 where, for example $Obs^t_{(1,0)}$ indicates the current observation with I1 $=$ 1 and I2 $=$ 0.

and probe values.

- **Basic**: This problem representation establishes a base case for the smallest size of encoding; it uses only the current observation.
- **Basic plus Action (Basica)**: Here we enhanced the Basic encoding with one additional piece of history: the previous action.
- **One Step (1Step)**: This representation includes both the previous observation and the previous action with the current observation.
- **Three Step (3Step)**: This representation uses four steps of history which consist of the previous three observations, the current observation, and the previous three control actions.
- **Fixed Fingerprints (FFPA)**: The fixed fingerprints encoding uses fingerprint information by keeping the most recent output, carry, and probe values for each relevant combination of the four possible inputs. It is *fixed* because the representation saves fingerprint information in four fixed positions, each of which corresponds to a unique combination of the I1 and I2 values.
- **Sliding Fingerprints (SFP)**: This encoding also uses the fingerprint information, but the observations are not in a fixed position. In this approach we maintained a list of four previous observations indexed by the values of I1 and I2. When a new observation arrives, the old observations with the same values for I1 and I2 are removed from the list, and the new observation

is pushed onto the front of the list.

For each problem representation, we investigated the resulting performance from both off-line and on-line approaches to collecting training data.

3.2 Data Collection

Typical approaches in supervised learning use inductive learning algorithms such as C4.5 for batch or, "off-line," learning where training examples are collected in an initial step and a classifier is generated from the training examples in the second step. As discussed in Section 1.1, an off-line approach using stratified sampling produced an inadequate distribution of training examples, even after the training data had been filtered to emphasize more failures in the controlled system. On-line learning attempts to remedy this inadequate distribution of training examples by allowing the learning controller to choose the training examples.

In on-line learning, the learning controller is embedded in and acts as the controller of the controlled system. When functioning as part of a continuously running system, the embedded learning controller often issues incorrect control actions. Because these control actions are different than the control actions the deliberative controller issues, the controlled system enters states not normally encountered by the deliberative controller. For each state visited, the deliberative controller supplies the correct action as a training example. This on-line collection of training examples allows the learning controller to detect when it has made an error and to learn how to avoid the error in the future.

Initially we intended to use two inductive learning algorithms, Q-learning and C4.5, to explore the differences in performance between traditional off-line learning and on-line learning methods. Both algorithms map inputs to classifications. The first algorithm, Q-learning, is naturally incremental and easily runs in an on-line manner, but the second algorithm, C4.5, is not inherently incremental and it was necessary to simulate incrementality to achieve on-line learning. Unfortunately, as explained below, using Q-learning was infeasible. As a result, the efforts of this study focused on the use of C4.5.

3.2.1 Q-Learning

Discovered by Watkins [Wat89], 1-step Q-learning is an reinforcement learning algorithm which does not make use of a teacher such as the deliberative controller, instead it learns through rewards. It is guaranteed to converge to an optimal policy under certain conditions. One of these conditions is that the underlying decision process must be Markovian. However, since the half adder circuit is only partially observable, initial experiments using both 1-step Q-learning and Whitehead's Lion Algorithm [WB91], showed it was difficult to find an adequate Markov representation of the problem, one in which the problem representation did not grow exponentially in size of the representation. As a result, we chose to focus on C4.5.

```
On-Line-Learning:
    T := {};                                    /* set of training examples */
    Obs := {};                               /* set of current observations */
    Act := nothing;                                       /* control action */
    repeat
        Obs := Controlled-System(Act);
        t := Generate-Training-Example(Obs);
        T := T ∪ {t};
        Learning-Controller := C4.5(T);
                        /* Build a decision tree from the training examples */
        Act := Learning-Controller(Obs);
    until Done;
Generate-Training-Example (Obs):
        Act := Deliberative-Controller (Obs);
        t := {Obs,Act};
        return t;
```

Figure 6. Simulating On-line learning algorithm using C4.5

3.2.2 C4.5

This is a descendant of Quinlan's ID3 system [Qui86] for constructing decision trees. It learns classification rules from training examples. No special adjustments were necessary for off-line learning, but because C4.5 is not an incremental learning method, incremental on-line learning was simulated.

The algorithm in Figure 6 describes the simulated incremental learning using C4.5. The controlled system first generates a set of observations and the deliberative controller chooses a control action based on these observations. The observations plus the control action from the deliberative controller are combined to form a training example. This training example is added to the existing set of training examples, and C4.5 is run on this new set of training examples, building a decision tree. The learning controller uses this newly created decision tree with the observations to choose a control action. This control action is then passed to the controlled system, and the cycle starts again as the controlled system executes the control action and generates a new set of observations.

3.3 Experimental Approach

Recall that the objectives of this research are to survey the differences in performance with the use of history information with on-line and off-line collection of training examples when developing reactive controllers, and to provide insight on the space and time complexity requirements of such controllers. Described below are the experiments we conducted to investigate these objectives.

Initially we had intended to evaluate each approach only on the average cost per cycle, but because each learning controller does not fix every failure, it became difficult to derive objective evaluations for the controllers when a gate remained broken throughout the entire run. This prompted us to design a test which predicted the performance of each controller by estimating the number of failures it could fix. Appendix A describes the predicted performance measure in further detail and presents an empirical justification of this measure.

After establishing an effective measure, the next step was to determine an adequate training set size. Since we could generate any number of training examples given enough time, we wanted to ensure that the size of our training set did not limit our success. Yet, since time is an issue, we needed to bound the size of the training sets. Although the trend seems to imply further improvement could be obtained with more training examples, we stopped at 6000 training examples when some of the off-line controllers achieved a predicted performance of 8 where 8 is the best possible score.

To characterize the off-line performance of each problem representation, we generated six samples of size 6000 training examples collected off-line. For every problem representation we generated a controller from each set of training examples, then evaluated the resulting controllers.

We also wanted to characterize on-line performance for all problem representations with six samples trained on 6000 training examples, but due to time and resource considerations it was necessary to revise this approach. Instead we generated a full six samples for one problem representation, SFP, to see if on-line learning could improve performance over off-line learning. For all other problem representations, we chose to generate two samples to determine if the other representations also showed improved performance with on-line learning. Since 1Step showed good on-line learning performance (see Section 5.1) on the first two samples and has a representation size less than the upper bound of 4 steps of history, we generated two additional sets of training examples and evaluated the results.

We examined each problem representation to determine the worst case time and space requirements. Then, using both the off-line and on-line controllers, we also found the experimental average space requirements for each problem represention.

Chapter 4

Results

In this chapter we present the results of the experiments discussed in the previous chapter.

The first step in this investigation was to understand how performance was affected by the number of training examples. Figure 7 illustrates the effect training set size has on performance for each problem representation. Each line in the graph reflects the average predicted performance for six sets of training examples when trained off-line with various sizes of training sets. Note that this is a longitudinal comparison, that is, each larger training set subsumes previous, smaller training sets. Recall that the predicted performance reflects the number of problem classes a particular method can solve and that the best possible predicted performance is 8 classes.

Notice that at around 2000 training examples, improvement in performance begins to level off; that is, after 2000 training examples, performance improves much more slowly. Although at 6000 training examples the trend seems to indi-

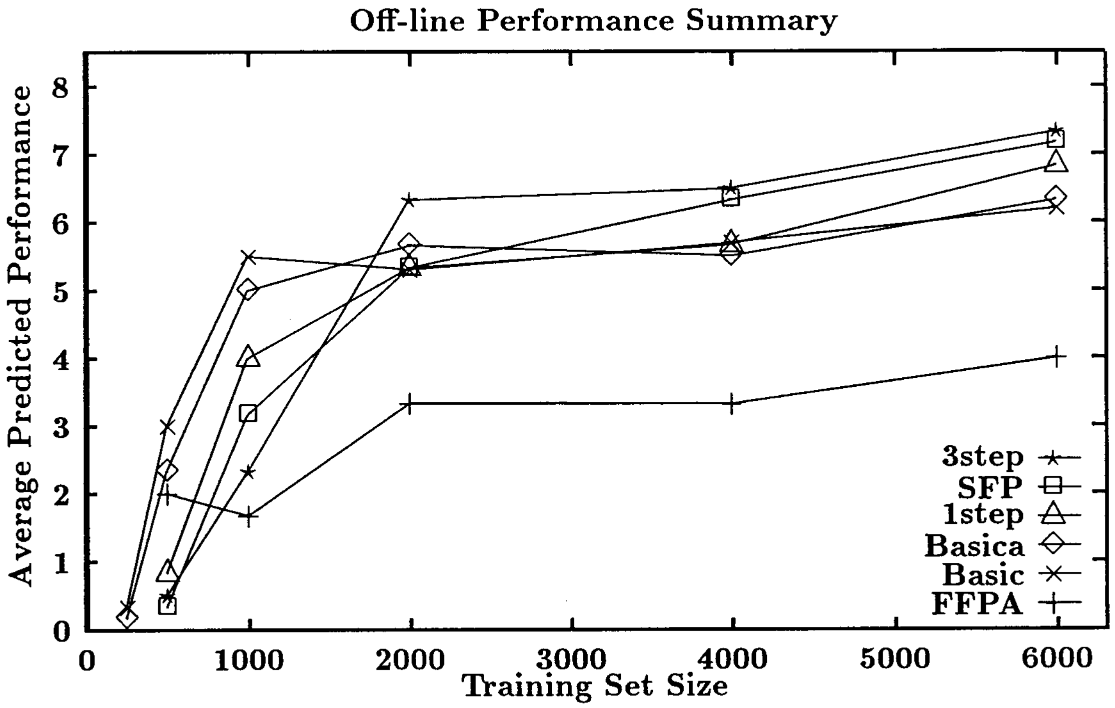


Figure 7. Average predicted performance for each problem representation when trained off-line with various sizes of training sets. Each point plotted represents an average of six data points. The maximum standard error for any of these points is 1.13 for SFP with 1000 training examples.

cate that further performance improvement could be obtained with more training examples, we chose to stop at 6000 training examples. The reason for this decision was that the learning controllers were beginning to reach an optimal predicted performance of 8 classes solved but more importantly, it was expensive in terms of time to generate these examples; it took over 1 1/2 minutes on a Sun(R) SPARC-station for the deliberative controller to generate and record a single decision and the simulator to update the circuit; collecting 6000 examples took close to one week.

4.1 Problem Representation Performance

Table 1. Average and standard error for each problem representation when trained off-line on 6000 training examples. Note that a predicted performance of 8 is the best possible score.

Method	Predicted Performance
Basic	6.2±0.31
Basica	6.3±0.42
1Step	6.8±0.48
3Step	7.3±0.33
FFPA	4.0±0.00
SFP	7.2±0.31

Table 1 shows the average off-line predicted performance for each problem representation at 6000 training examples. Some controllers created using 1Step, 3Step, and SFP methods achieved a perfect score of eight. Performing a one-way repeated measures statistical test [Inc89] on the data summarized in Table 1 using each method on each set of training examples gave an F value of 16.37 (df.=10,25, p=0.0001). This shows that each problem representation does not achieve equally good performance. A Duncan multiple range test [Inc89] with a significance level of .05 showed that methods 3Step, SFP, and 1Step are not significantly different from each other; neither are methods 1Step, Basic, and Basica. Results do show that the predicted performance for methods 3Step and SFP is significantly better than for Basic and Basica. Performance of the FFPA method is significantly lower than that of all other methods.

Table 2 shows, for each problem representation, the features used by actual decision trees. In this table the "-" indicate that the feature was not used in the representation. The "∞" indicate that the feature was not tested by the decision tree. The numbers indicate the first level at which the feature appears in the decision tree where "0" indicates a root node. The decision trees chosen for this analysis were ones which showed the best performance for the associated problem representation.

Table 2. Features used by the decision trees.

Feature	Basic		Basica		1Step		3Step		FFPA		SFP	
	Off	On	Off	On	Off	On	Off	On	Off	On	Off	On
I1	2	2	2	1	2	2	2	2	-	-	2	3
I2	1	1	2	2	3	2	1	1	-	-	1	1
O	2	2	1	1	2	1	3	2	∞	4	2	3
C	0	0	1	0	1	1	1	1	∞	2	1	2
P1	∞	∞	∞	∞	∞	∞	1	∞	∞	∞	1	∞
P2	4	∞	1	∞	1	∞	2	∞	∞	∞	2	∞
I1a	-	-	-	-	∞	1	∞	1	-	-	∞	7
I2a	-	-	-	-	∞	2	∞	3	-	-	∞	8
Oa	-	-	-	-	2	2	2	2	0	3	1	1
Ca	-	-	-	-	1	1	1	∞	1	1	2	4
P1a	-	-	-	-	∞	∞	∞	∞	∞	∞	∞	∞
P2a	-	-	-	-	∞	∞	∞	∞	∞	∞	∞	∞
I1b	-	-	-	-	-	-	4	∞	-	-	2	7
I2b	-	-	-	-	-	-	2	∞	-	-	∞	6
Ob	-	-	-	-	-	-	∞	5	∞	1	∞	3
Cb	-	-	-	-	-	-	3	4	∞	0	∞	3
P1b	-	-	-	-	-	-	∞	∞	∞	∞	∞	∞
P2b	-	-	-	-	-	-	∞	∞	∞	∞	∞	∞
I1c	-	-	-	-	-	-	∞	6	-	-	∞	6
I2c	-	-	-	-	-	-	3	∞	-	-	3	7
Oc	-	-	-	-	-	-	∞	∞	-	-	∞	2
Cc	-	-	-	-	-	-	4	6	-	-	∞	5
P1c	-	-	-	-	-	-	∞	∞	-	-	∞	∞
P2c	-	-	-	-	-	-	∞	∞	-	-	∞	∞
Aa	-	-	0	∞	0	0	0	0	2	2	0	0
Ab	-	-	-	-	-	-	3	1	2	3	3	1
Ac	-	-	-	-	-	-	4	∞	4	4	∞	∞

4.2 Influence of On-line Learning

Table 3. Probabilities that off-line and on-line means are equal from a statistical t-test. Where the probability is small (less than or equal to 0.05) we reject the hypothesis that the means are equal. These probabilities are shown in bold.

Number of Examples	Basic	Basica	1Step	3Step	SFP	FFPA
1000	**0.010**	1.000	0.657	0.247	0.085	0.152
2000	**0.005**	0.680	0.246	0.331	0.065	0.273
4000	0.114	**0.053**	**0.052**	0.267	0.081	**0.017**
6000	0.387	0.680	0.571	0.680	**0.022**	**0.000**

Figure 8 compares the performance of off-line and on-line training for each problem representation as a function of the number of training examples. Note that as in Figure 7, this is a longitudinal comparison, that is, each larger training set includes the previous, smaller training sets.

Multiple regression analysis of the on-line versus off-line learning plus the log of the sample size gives the prediction equation of:

$$\hat{Y} = -9.294 + 1.397X_1 + 1.814\log X_2 \qquad (4.1)$$

Where the predicted performance, $\hat{Y}$, is estimated by a binary value indicating off-line or on-line, X_1, and the sample size, X_2. The β coefficents 1.397 and 1.814

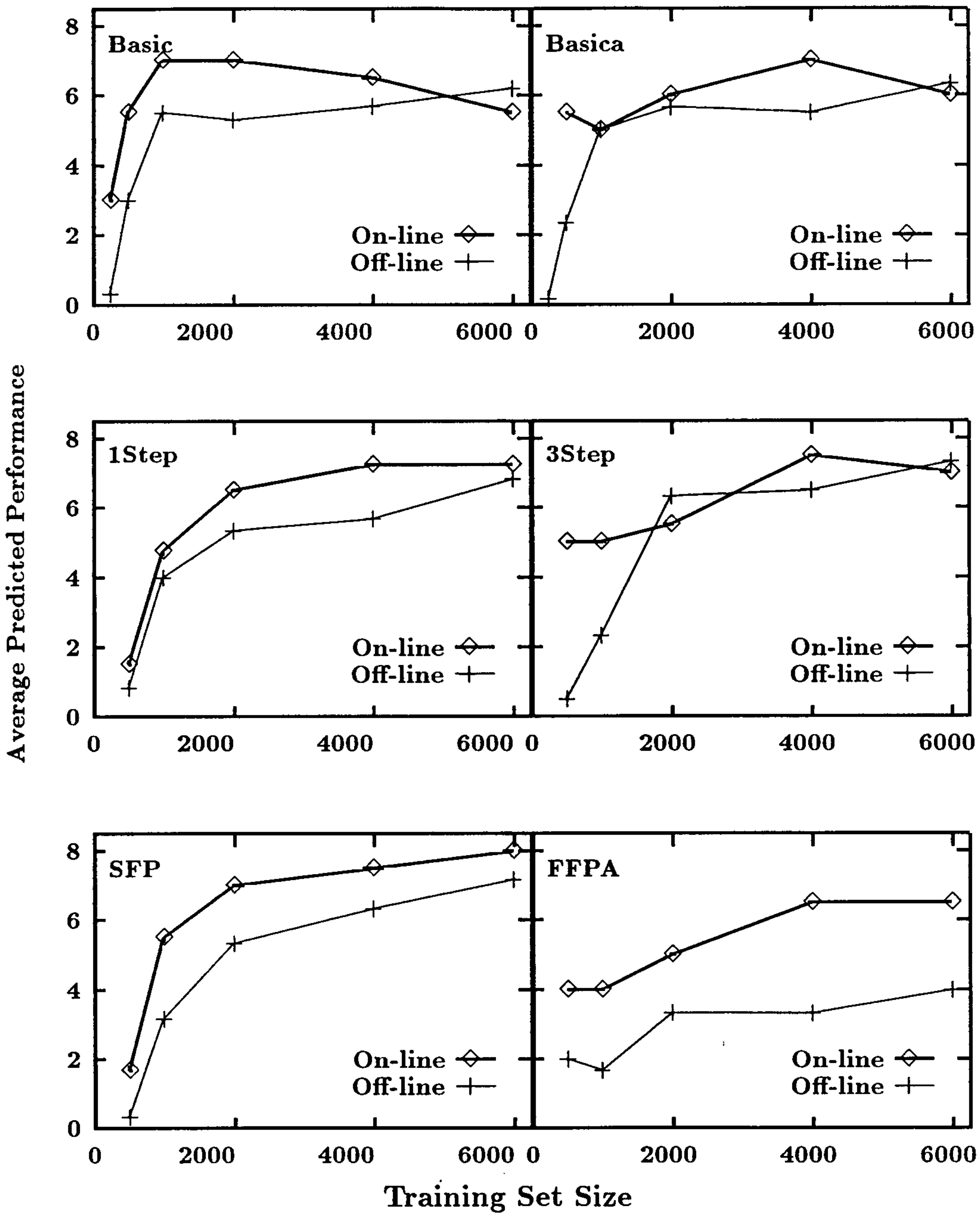


Figure 8. Average predicted performance of off-line versus on-line learning.

are highly significant, both have p values of .000000 and standard errors of .2159 and .1023 respectively.

4.3 Time and Space Complexity

Table 4. Time and space complexities of each method when trained on 6000 training examples.

Method	Time*	Tree Size (total nodes)			Performance	
		Off-Line Avg.	On-Line Avg.	Upper Bound **	Off-Line	On-Line
Basic	6	29.2	22.0	64	6.2	5.5
Basica	12	36.7	29.5	448	6.3	6.0
1Step	18	45.8	43.0	28,672	6.8	7.2
3Step	42	58.8	52.0	5.75e+09	7.3	7.0
FFPA	34	54.0	126.5	2.25e+07	4.0	6.5
SFP	42	57.2	126.7	5.75e+09	7.2	8.0

* Determined by the maximum number of comparisons needed to traverse the theoretically worst case decision tree.

** Size (number of nodes) of theoretically worst case decision tree.

Table 4 summarizes the time and space complexities, and predicted performance for each method. The time complexity in Table 4 is worst case, but even under the worst case assumption the time complexity is small. The space complexities given in Table 4 include the worst case tree and the average trees. Although the worst case tree is infeasibly large, the average trees are very small in comparison.

Notice the substantial differences between theoretical upper bound and experimental averages for tree sizes in each case. The difference clearly indicates that C4.5 produces a compact decision tree.

Chapter 5

Discussion

In this chapter we discuss the results of our experiments on the automatic construction of reactive, knowledge-based controllers from deliberative solutions for the On-line Maintenance Agent (OLMA). We also compare the performance of some reactive controllers against the performance of the deliberative controller, then discuss future directions for this work.

5.1 Problem Representation

When a controller has limited visibility of the system it is controlling, the amount of history information it has about the controlled system affects its performance. This work set out to find the amount of history information that was needed to keep the half adder circuit operational. Table 5 shows the amount of history information and the resulting predicted performance for each representation in the half adder circuit domain. The amount of history is quantified by the number of previous observations and actions used by each representation. The

Table 5. Amount of history information and resulting predicted performance for each representation in the half adder circuit domain

Method	History		Performance	
	Obs	Acts	Off-Line	On-Line
Basic	1	0	6.2	5.5
Basica	1	1	6.3	6.0
1Step	2	1	6.8	7.2
3Step	4	3	7.3	7.0
FFPA	4*	3	4.0	6.5
SFP	4*	3	7.2	8.0

* The encoded representations span a minimum of the last four observations.

predicted performance is given for both off-line and on-line methods of data collection for 6000 training examples. This table shows that history information is beneficial. That is, in general, as the amount of history information increases, the performance improves. The FFPA problem representation is an exception to this trend.

Although the problem representations 1Step and Basica differ in the amount of history by one previous observation, no controllers built with the Basica representation achieved a predicted performance score of eight while 1Step achieved an eight in 2 out of 6 off-line controllers, and in 2 out of 4 on-line controllers. 1Step therefore was the smallest problem representation to achieve good results.

Training Example	Control Action
0,0,0,0,?,?⇒NOTHING	NOTHING
0,0,0,0,?,?⇒NOTHING	NOTHING
1,1,0,1,?,?⇒NOTHING	NOTHING
0,0,1,0,?,?⇒NOR1	NOR1
0,0,0,0,?,?⇒NOTHING	NOTHING
1,1,0,1,?,?⇒NOTHING	NOTHING
0,0,0,0,?,?⇒NOTHING	NOTHING

Figure 9. Example of off-line training data for the Basic representation with a failure of gate NOR1. Each training example combines an observation with the control action recommended by the deliberative controller. The control action is the action actually issued to the controlled system.

The largest and most encompassing problem representation, SFP, consistently achieved a high rate of success. Controllers constructed via this method achieved a perfect score in 6 out of 6 on-line experiments. It is important to note that both FFPA and SFP use the fingerprint information, and that this fingerprint information is exponential in the inputs to the circuit.

5.2 Off-line versus On-line

The multiple regression analysis discussed in Section 4.2 showed that the method of collecting training (off-line or on-line) is a significant predictor of the predicted performance measure, and that on-line learning can improve performance over off-line learning.

Figures 9 and 10 show examples of actual training data using the Basic problem representation and the associated control actions. They illustrate why on-line can achieve better performance than off-line. In each figure, the left column

Training Examples	**Control Action**
1,0,1,0,?,?⇒NOTHING	NOTHING
0,1,1,0,?,?⇒NOTHING	NOTHING
1,0,1,0,?,?⇒NOTHING	NOTHING
0,0,1,0,?,?⇒NOR1	NOTHING
0,1,1,0,?,?⇒NOR1	NOTHING
0,0,1,0,?,?⇒NOR1	NOTHING
1,0,1,0,?,?⇒NOR1	NOTHING
1,0,1,0,?,?⇒NOR1	NOTHING
1,1,0,1,?,?⇒NOR1	NOTHING
1,0,1,0,?,?⇒NOR1	NOTHING
1,1,0,1,?,?⇒NOR1	NOTHING
0,1,1,0,?,?⇒NOR1	NOTHING
1,0,1,0,?,?⇒NOR1	NOTHING
0,0,1,0,?,?⇒NOR1	NOTHING
0,1,1,0,?,?⇒NOR1	NOTHING
1,0,1,0,?,?⇒NOR1	NOTHING
0,1,1,0,?,?⇒NOR1	NOR1
1,1,0,1,?,?⇒NOTHING	NOTHING
0,1,1,0,?,?⇒NOTHING	NOTHING
1,0,1,0,?,?⇒NOTHING	NOTHING

Figure 10. Example of on-line training data and the corresponding control actions for the Basic representation with a failure of gate NOR1. Each training example combines an observation with the action recommended by the deliberative controller. The control action is the action actually issued to the controlled system.

shows an example of the training data with a NOR1 failure; the right column shows the actual control actions that were issued with each training example.

Notice that in Figure 9 the off-line data provides only one training example with a *NOR1* control action when the NOR1 gate failed, whereas the on-line data provides 14 training examples with *NOR1* control actions for this single failure. In this on-line example, the deliberative controller effectively tailors the training examples to the learning controller by providing training examples with *NOR1* control actions until the inductive learning algorithm has enough examples to learn to issue a *NOR1* control action. Recall that in on-line learning, it is the learning controller rather than the deliberative controller which issues the control actions. Because this on-line approach provides many more distinct training examples with each failure, on-line learning can achieve better performance over off-line learning.

5.3 Space and Performance Tradeoffs

Figure 11 presents the average and standard error of the tree sizes with their predicted performance over all the learning controllers. As one might expect, as tree size increases, performance tends to improve. Recall that in Table 4 the empirically measured tree size is far smaller than the theoretical upper bound.

5.4 How Do the Controllers Compare?

Figure 12 shows the performance of the deliberative controller and some

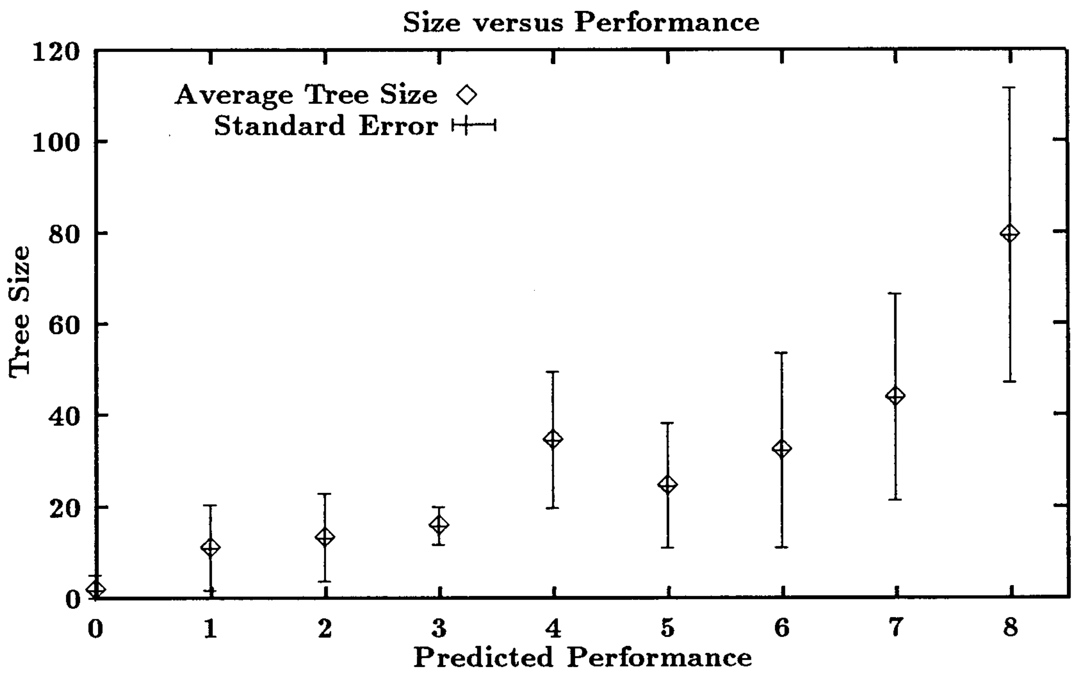


Figure 11. Space versus performance. Each data point represents the average tree size over all controllers with the same predicted performance.

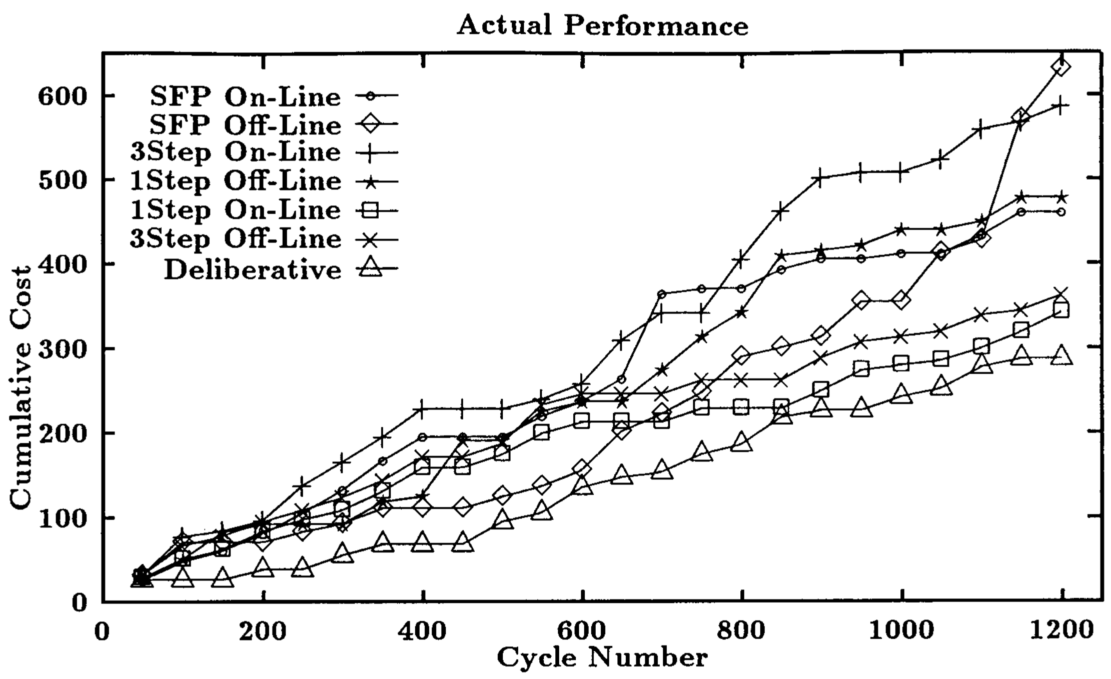


Figure 12. Comparison of the actual runs of the better controllers. Each reactive controllers solved all 8 problem classes as determined by the predicted performance problem measure.

of the learning controllers after training. The learning controllers chosen for this graph have shown they can solve all 8 problem classes defined by the predicted performance measure. Figure 12 gives the accumulated cost for each controller over a run of 1200 cycles. In each of these runs except for the deliberative run, the simulator was started with the same random seed. The significance of the graph in Figure 12 is that since the performance of the learning controllers do not differ significantly from that of the deliberative controller, we have succeeded in constructing learning controllers that can solve instances of each class of problem and keep the controlled system from entering an irrecoverably broken state.

Another point is that although the controllers in Figure 12 show similar performance, the deliberative controller used about one wall clock minute per decision while the reactive controllers took less than a second to evaluate the decision tree for each decision.

It is important to note that the graph in Figure 12 shows a single run for each controller. Because the variance between runs is large, the relative rank order as shown in the graph is not a good indicator of the overall relative performance of each controller.

5.5 Conclusions

We compared the performance of automatically constructed, low time complexity reactive controllers with little or no internal state. The results of our experiments show that for the problem domain of the half adder circuit, using history as part of the problem representation can improve performance. Our results also show that having a controller learn from another controller while interacting with the controlled system can improve the resulting performance of this learning controller. Analysis of the reactive controllers showed that for our problem domain, the actual decision trees are much smaller than a complete look up table. The look up table that Ginsberg describes in [Gin89], grows exponentially while the decision trees do not.

5.6 Future Directions

Suggestions for future research in this area include:

1. Explore approaches in speeding up the on-line learning process. One approach suggested by Dietterich [Die91] is instead of training with C4.5 on every cycle, have it keep a history of the training examples, then at regular intervals, retrain using C4.5.

2. Use recent advances in temporal difference methods such as Q-learning in place of C4.5. Because this is a control problem, it is suited for such policy learning algorithms. The difficultly we had in applying Q-learning stemmed from trouble in finding a feasible representation of the current state, but much of the recent work in Q-learning have been addressing this state representation problem [Tes92, CK91].

3. Extend the ideas suggested in this research work to more interesting and complex domains. This will provide further evidence as to whether these techniques scale.

BIBLIOGRAPHY

[Abu88] Maurice F. Aburdene. *Computer Simulation of Dynamic Systems*, chapter 7, pages 190–209. Wm. C. Brown, 1988.

[AC87] P. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth International Conference on Artificial Intelligence*, pages 268–272, Menlo Park, CA, 1987. AAAI, American Assiciation for Artificial Intelligence.

[Bro68] R. A. Brooks. A robust layered control system for a mobil robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1968.

[CK91] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *Proceedings of IJCAI*, pages 726–731, 1991.

[Coo87] G. F. Cooper. Probabilistic inference using belief networks is NP-hard. Technical Report KSL-87-27, Medical Computer Scienc Group, Stanford University, 1987.

[D'A] Bruce D'Ambrosio. Deciding in time.

[DF91] Bruce D'Ambrosio and Tony Fountain. Reaction in real-time decision making. In *Proceedings of the International Joint Conference of Artificial Intelligence*, 1991.

[Die91] Tom Dietterich. Email communication. October, 11, 1991.

[DW91] Thomas L. Dean and Michael P. Wellman. *Planning and Control.* Morgan Kaufmann, San Mateo California, 1991.

[FL90] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University, 1990.

[Gin89] Matthew L. Ginsberg. Universal planning an (almost) universally bad idea. *AI Magazine*, 10(4):40–44, winter 1989.

[Inc89] SAS Institute Inc. *SAS/STAT User's Guide*, volume 1, chapter 13. SAS Institute Inc, Cary,NC, 4 edition, 1989.

[Kae90] Leslie Pack Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University, Department of Computer Science, June 1990.

[Kau91] Lothar Kaul. A feasibility study on compiling reactive problem solution methods for an ai domain. Master's thesis, Oregon State University, Corvallis Oregon, May 1991.

[LNR87] J. Laird, A. Newell, and P. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[MC91] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings AAAI-91*, pages 768–773. AAAI, 1991.

[PB91] David W. Payton and Thomas E. Bihari. Intelligent real-time control of robotic vehicles. *Communications of the ACM*, 34(8):49–63, August 1991.

[Qui86] J. R Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Sch87] Marcel J. Schoppers. Universal plans for reactive robots in unpredictable domains. In *Proceedings of the Tenth International Joint Converence on Artificial Intelligence*, pages 1039–1046, Menlo Park, CA, 1987. International Joint Conference on Artificial Intelligence, Inc.

[Sch89] Marcel J. Schoppers. In defence of reaction plans as caches. *AI Magazine*, 10(4):51–60, winter 1989.

[Sch91] Marcel Schoppers. Real-time knowledge-based control systems. *Communications of the ACM*, 34(8):27–30, August 1991.

[Tes92] Gerald Tesauro. Temporal difference learning of backgammon strategy. *Proceedings of the Ninth Machine Learning Conference*, pages 451–457, 1992.

[Wat89] Chris Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.

[WB90] Steven D. Whitehead and Dana H. Ballard. Active perception and reinforecment learning. *Neural Computation*, 2:409–419, 1990.

[WB91] Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, pages 45–83, 7 1991.

[Whi91] Steven D. Whitehead. *Reinforcement Learning for the Adaptive Control of Perceptin and Action.* PhD thesis, University of Rochester, Rochester, New York, October 1991.

APPENDIX

Appendix A

Validating Predicted Performance

Initially, in evaluating the performance of reactive controllers we had intended to evaluate each approach only on the average cost per cycle for a simulated run, but because of the continuous and random operation of the circuit in the problem domain, if a gate failed in the circuit, it remained broken throughout the rest of an experiment unless the controller found the broken gate and replaced it. This scenario skewed the results of the experiment and made it difficult to derive an objective evaluation of each reactive controller. This prompted us to derive a deterministic test which could help predict the performance of an agent.

This predicted performance measure consisted of a set of test examples which simulated the failure of each gate in the system. A gate can fail three ways, stuck at 1, stuck at 0, and unknown. When a gate is stuck at 1 it outputs a 1 regardless of the input. Similarly when a gate is stuck at 0 it outputs a 0 regardless of the input. When the failure is unknown, the output of the gate is random 0s and 1s. The predicted performance measure uses all permutations of the possible observations for stuck at 1 and stuck at 0 failures for each gate in the system. Unknown failures were not explicitly included because they are covered by the stuck at 1 and stuck at 0 failures. There are four gates in the circuit making a total of eight problem classes to test. Using this test it was possible to identify which gates the controller could potentially repair and use this to quantify the predicted performance of each controller.

In this appendix the predicted performance measure is related to the performance of the actual system. Figure 13 plots the normalized cumulative cost and predicted performance for 59 runs of 1200 cycles. A close look at the plotted points and the regression line reveals that the relationship between predicted performance and cost is not linear. The plotted points indicate that although good performance gives a low cost the cost for poor performance is not a bad as the cost for medium performance. The high cost for medium performance is due to the system repeatedly repairing the wrong fault. Thus cost does not reflect a good measure of performance.

Instead, as a measure of the performance of the actual system, we used the number of failures which occur during a run of the simulated system. Intuitively, this is a good measure because the number of component failures in a run reflects the number of failures a controller can fix, indicating the performance abilities of the controller. To illustrate this point, consider a controller which knows one control action, *Nothing*; it can never repair a failed gate in the circuit. In a simulated run eventually all the gates in the circuit fail and are never fixed. Thus the total number of failures for the run is limited to the number of gates in the circuit. On the other hand, if a controller can fix all the failures that can occur, when a failure does occur it will be fixed and have the chance to fail again. Thus, oddly enough, the more failures that occur the better the controller.

To show the relationship between predicted performance and number of failures, we used correlation and linear regression analysis. Correlation analysis on this sample gave a Pearson correlation coefficient [Inc89] of 0.872 and a mean square error of 1.71. With a sample size of 59, this correlation coefficient is significantly larger than zero. Figure 14 shows the regression line and scatter plot of the number of failures versus the predicted performance when a reactive controller is run as the controller of the controlled system. This regression line has an F value of 180.85 (p=0.0001) and shows a direct linear relationship between the predicted performance and number of failures.

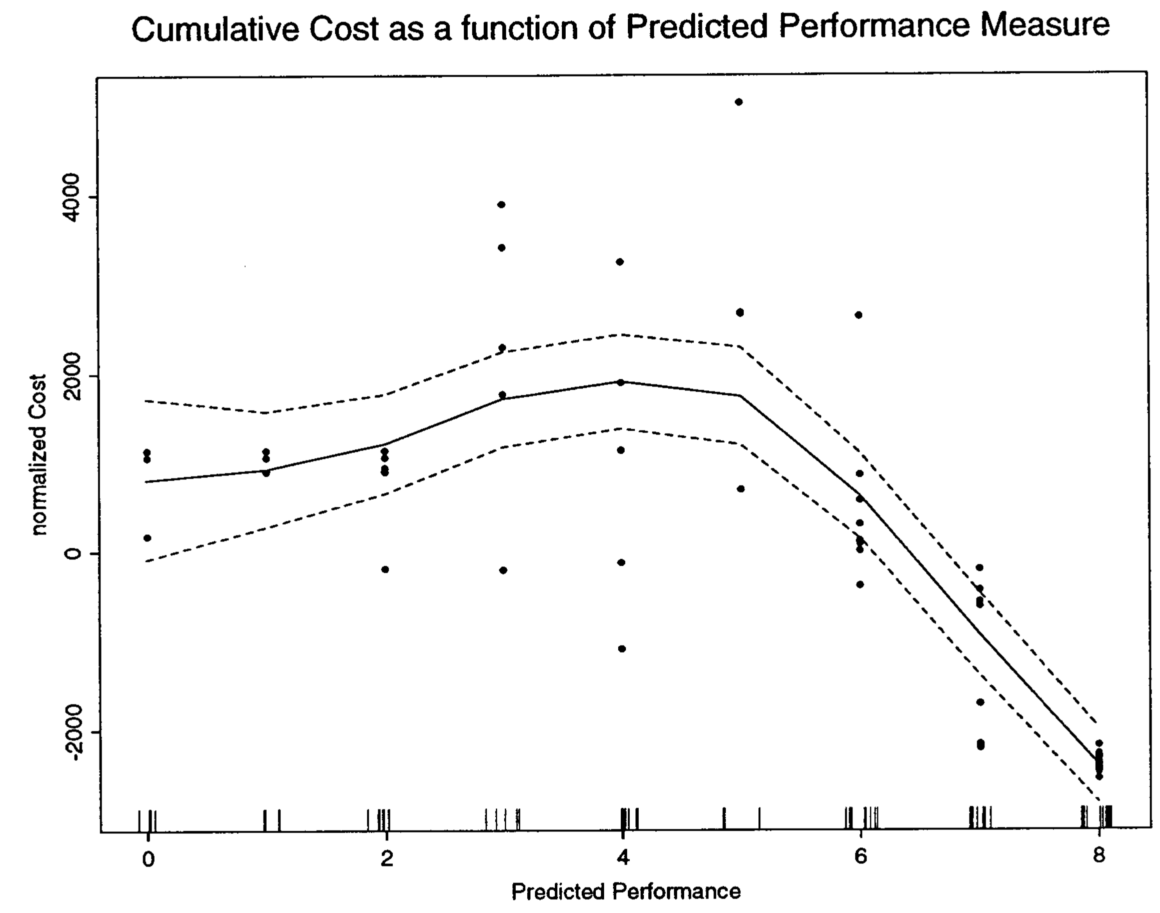


Figure 13. Regression line and scatter plot of cumulative cost versus predicted performance for a simulated run of 1200 cycles. The cumulative cost values have been normalized to have zero mean. The dotted error lines are 2 standard errors away from the predicted regression line.

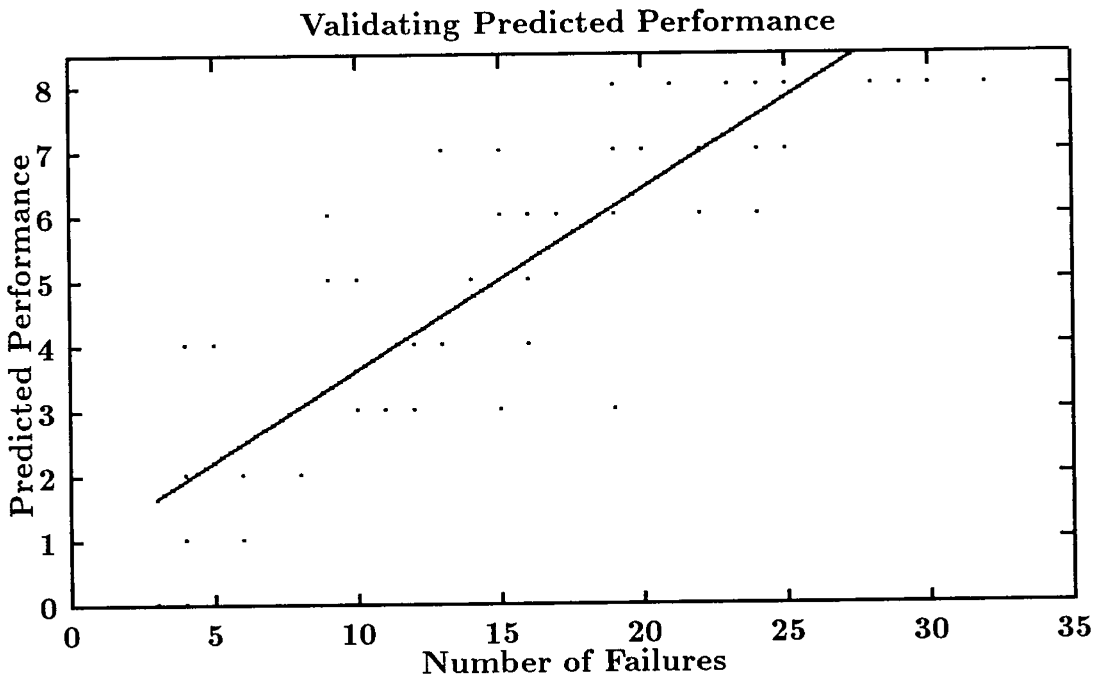


Figure 14. Regression line and scatter plot of predicted performance versus number of failures for a simulated run of 1200 cycles.