Simulation-Assisted Inductive Learning*

Bruce G. Buchanan, John Sullivan, and Tze-Pin Cheng
Knowledge Systems Laboratory
Stanford University
Stanford, California 94305

Scott H. Clearwater
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
and Knowledge Systems Laboratory
Stanford University

Abstract

Learning by induction can require a large number of training examples. We show the power of using a simulator to generate training data and test data in learning rules for an expert system. The induction program is RL, a simplified version of Meta-DENDRAL. The expert system is ABLE, a rule-based system that identifies and locates errors in particle beam lines used in high energy physics. A simulator of beam lines allowed forming and testing rules on sufficient numbers of cases that ABLE's performance is demonstrably accurate and precise.

1 Introduction

Learning by induction is one important means of learning classification rules for expert systems [Buchanan and Mitchell, 1978; Michalski, 1983]. The major assumption in learning by induction is that a source of training examples exists. In many domains for which one wants to build expert systems, however, assembling libraries of training cases can present significant practical problems. Meta-DENDRAL, for example, worked with available mass spectra of just a few organic chemical compounds at a time because only a few compounds of the classes under consideration had been analyzed, and additional spectra were nearly impossible to obtain.

In the present paper we illustrate one way of overcoming these kinds of problems by using a simulator to generate large numbers of training examples, and we discuss some implications of doing so. This idea was developed by Brown [1982] in the context of tutoring electronics troubleshooting, by Samuel[1963] in the context of checkers, and recently by Kunstaetter[1987] in the context of tutoring medical diagnosis. An obvious assumption we make is that an accurate simulation model exists. This is frequently true in technical domains in which there are theoretical equations or other strong models describing the behavior of physical or biological systems. Another assumption is that the domain is complex enough that explaining data (e.g., for troubleshooting) cannot be performed by using the simulator in a random generate-and-test method.

One obvious question arises in the case where strong models exist: why build a heuristic program at all? The answer lies mostly in the asymmetry between prediction and explanation. The behavior of a system under ideal conditions may be predictable with high accuracy using relations that map descriptions of the initial state of the system onto descriptions of its final state (and from there to observable data). Heuristics are needed, however, to adjust such relations with respect to deviations from the ideal. More importantly, it is not generally possible to interpret equations or run simulations "backwards" in order to infer causes from their effects, because causal knowledge often maps many different causes onto the same manifestation.

1.1 The RL Program

For the induction program we use Rule-Learner (RL) [Fu, 1985; Fu and Buchanan, 1985], a generalization of Meta-DENDRAL. RL is a successive refinement program that searches a space of IF-THEN rules for acceptable classification rules.

The input to RL is: (a) a collection of training examples classified into one or more concept classes (with no assumption of complete correctness), and (b) some initial knowledge of the domain -- called the "half-order theory" -- that includes (i) the vocabulary of legal descriptions of examples, including names of classifications, (ii) some semantics of relationships among the terms in the vocabulary, (iii) heuristics to prune or order RL's search through the space of rules, e.g., plausible ranges of values of descriptors, plausible threshold values on the size or complexity of rules, and plausible (or implausible) combinations of terms in rules.

The output from RL is a set of rules that correctly classify (true positives) most of the examples (where "most" is defined in the half-order theory as a specified percent of all training examples) and that misclassify (false positives) an acceptably small number of examples (where "acceptably small" is similarly defined). Each rule is of the form:

$$\langle \text{feature}_1 \rangle \& .. \& \langle \text{feature}_i \rangle \Rightarrow \langle \text{classification}_k \rangle$$

where each feature describes a value or range of values for an attribute of an object, e.g., $\text{Attribute}_j > 2$.

1.2 The ABLE Program

A practical problem that fits this model came to our attention from high energy physics. Experimental high-energy physicists study the composition of matter using particle accelerators. These facilities all use beam lines to transport the high energy particle "beams" (bunches of particles) for studying properties of elementary particles, material science research, or other applications. The beam line itself is a

*The research reported in this report was funded in part by the following contracts and grants: DARPA N00039-86C-0033, NIH/SUMEX 5P41 RR-00785 and IBM SL-87046. Work was also performed under the auspices of the Department of Energy and supported by the U.S. Army Strategic Defense Command.

complicated arrangement of magnets, beam position monitors, accelerating sections, and drift sections forming a magnet optical lattice. The magnet optics are analogous to a conventional lens system that is used to focus light onto a particular area. Hence, the magnet optical elements (or simply "elements") bend or focus the electrically charged particles in the beam onto a target. The beam centroid is monitored along the trajectory from source to target with instruments referred to as beam position monitors or simply "monitors".

Particle accelerators are well described theoretically but their behavior is all too often not what is predicted or desired. For example, there may be misalignments or miscalibrations in the elements or monitors. Thus the actual beam may not behave as expected, as shown in Figure 1.

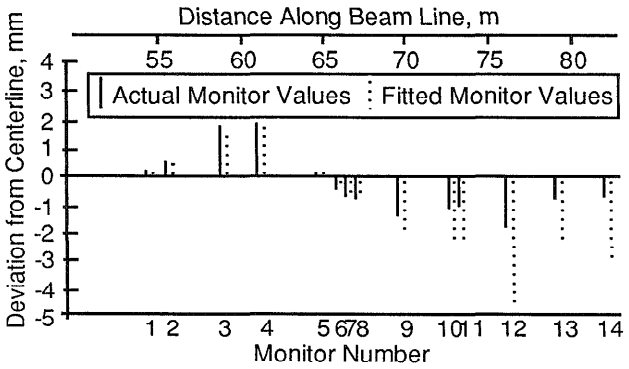

Figure 1
Diagram of a beam line, showing the measured and simulated values at each monitor. The two values have been slightly offset for clarity. An element error occurs somewhere before monitor number 9; downstream of monitor 9 the two values differ significantly.

One of the tools developed to assist in correcting problems is a simulator based on the theoretical model from which accelerators are constructed. It has been used in two modes:

(1) as a "what-if" tool showing what happens to a beam if operating conditions are perturbed, and

(2) as a predictive mechanism with a numerical optimization program that attempts to minimize the error between observed and predicted behavior of the beam.

We asked whether we could use the simulator in a third way, as a generator of training data, as described in the next section.

The Automated Beam Line Experiment project (ABLE) [Clearwater and Lee, 1987; Lee *et al.*, 1987a] is a research project investigating diagnosis of problems in particle beam lines. A prototype system was constructed as a rule-based expert system written in LISP.[1]

The present project blends ABLE and RL in an attempt to demonstrate the utility of learning programs for the particle accelerator domain.[2] In particular, we are studying machine learning in the presence of a very good quantitative domain simulator. We have assumed that a reasonable half-order theory can be specified more easily than an accurate rule set can be specified. This may not always be the case, but the items we requested for the half-order theory were not difficult for physicists to specify, especially with some feedback from the performance system. Figure 2 shows the relationship of the major elements discussed in this paper.

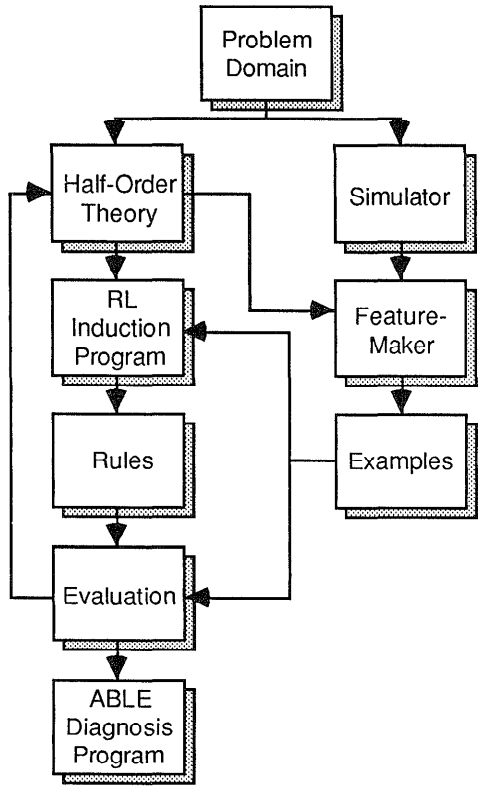

Figure 2
The relationship of the simulator, the RL induction program, and the ABLE diagnostic system.

2 Example Problem

2.1 Importance of Beam Line Verification

Because of the extreme complexity and expense involved in operating particle accelerator facilities and the high demand for access, there is a strong motivation to correct problems properly as quickly as possible.

The first task in operating a beam line is to verify the

[1]A FORTRAN version is also maintained for portability.

[2]RL and ABLE are implemented in Common Lisp and run on TI Explorer-II, Xerox 1186, and Apple MAC-II machines. Run times of RL for the present studies, involving 600 beam line segments, took 2-3 hours on an Explorer-II. ABLE uses the dozen or so rules to localize errors in 100 beam line test cases in run times in a few tens of minutes on a Symbolics 3600. Speeding up the programs is an obvious prerequisite for export which we have not yet undertaken.

design of the magnet lattice. This requires the expertise of accelerator physicists to analyze the data from beam line experiments. The beam line frequently needs verification because magnets are moved or their strengths are changed, resulting in a "new machine". Although data acquisition from the beam line diagnostics is often highly automated, analysis of these data by specialists has been a cumbersome and laborious task, requiring anywhere from hours to months.

2.2 The Simulator

Unlike many problem domains in AI, our application is able to utilize an excellent model. The model we used, COMFORT [Woodley *et al.*, 1983], is based on the well-known physics of particle beams propagating through a lattice of magnets. A beam line simulation program, PLUS [Lee *et al.*, 1987b], uses input beam characteristics to compute a value for the centroid of the beam at each monitor along the beam's "trajectory", using the design values for the magnet lattice.

The calculated design trajectory can then be compared with measurements from the actual (or simulated) machine. The differences between calculated and observed trajectories are used to localize and classify the cause of the problem. Making these interpretations is complicated in practice by the fact that there are usually more magnets than monitors and that there is noise in the data. The simulator has proved very useful in this context in solving actual problems [Lee *et al.*, 1987c].

3 Methods

3.1 Generation of Training Data

Training and test data were generated using the design lattice of the North Ring-to-Linac (NRTL) section of the SLAC Linear Collider. We further reduced the beam line into 20 monitors and 31 magnets corresponding to about 50m in length. Several hundred runs of the simulator generated cases for a training set, such that the error does not occur within three monitors of either end of the beam.[3] Each run was made by randomly choosing the "launch conditions" (the transverse off-set from the ideal beam centerline and the angle the beam makes with the centerline) of the input beam. Also each magnet had a small, random, transverse misalignment or strength miscalibration error in it to simulate realistic construction tolerances. In every case a random, large (but still realistic) magnet misalignment or miscalibration or monitor misalignment was added on top of the residual errors intrinsic to the system. It is these major misalignments or miscalibrations that the ABLE system is designed to find.

For each training example two segments (portions of the beam line delimited by monitors) were generated. We systematically chose segments that covered the problem space in the same way they would be applied, i.e., we focussed on the critical region where an error begins to manifest itself. Care is needed to assure that the systematically selected segments do not leave an important part of the problem space uncovered. Here again the simulator is crucial by providing a check on the performance of the rules on non-training data.

A segment had to contain at least 3 monitors and satisfy one of the following constraints:

(1) it ends exactly on the first monitor to show an error,
(2) it ends before the error, or
(3) it lies wholly after the error.

3.2 Data Abstraction

The examples generated by the simulator are of the form:

(m1 m2 m3 .. mi)
(s1 s2 s3 .. si)
(ELEMENT-ERROR-AROUND-MONITOR (X1..Xj))
(MONITOR-ERROR-AT (Y1..Yk)),

where i is the number of monitors, j is the number of element errors, and k is the number of monitor errors. Each m is the measured value of the beam trajectory at the corresponding monitor; each s is the simulated ideal value of the beam trajectory at a monitor.

RL input is in the form of a feature vector classified with respect to the concept being learned [Fu, 1985]. Since examples are not already in the form of RL's input, an additional component of the RL system, named the FeatureMaker, specifies RL's input feature vectors from the simulator's output. In Meta-DENDRAL, this rewriting function was performed by INTSUM [Buchanan *et al.*, 1976]. A complete list of terms used is shown in Appendix A.

3.3 Induction on the Training Data

3.3.1 The Vocabulary

In general, RL can operate with features whose values are numerical, symbolic, or boolean. For the ABLE domain, only numerically-valued features of RL have been used. In numerically-valued features, the range of possible values is subdivided by the use of pre-specified endpoints or markers. These markers are the only values present in the rules RL generates. The markers are chosen so that there are enough to distinguish positive examples from negative examples, but not so many that the computation becomes intractable. We plan to investigate ways to choose or adjust markers automatically, but have set them manually for the results presented here. Several runs of RL on different training sets were used to determine reasonable markers.

3.3.2 Separation of Concepts

Each example in the training set is classified according to the type of error ("Element-Error" or "Monitor-Error"). Recall that there are random fluctuations added to the descriptions of examples that may result in false classifications of these training data RL learns the classification rules for each target concept independently by regrouping the examples into exemplars and non-exemplars of that concept.

3.3.3 Threshold Adjustments

RL uses two simple thresholds to determine whether a rule matches the training set "well enough" to warrant further refinement, or inclusion in the concept definition if none of its specializations match well enough. These are:

$$\text{positive-coverage} = \frac{\text{number of true positive predictions}}{\text{total number of positive examples}}$$

[3]The reason for limiting the placement of errors is that the underlying physics rarely allows isolation of errors very close to the beginning or end of a beam line section.

$$\text{negative-coverage} = \frac{\text{number of false positive predictions}}{\text{total number of negative examples}}$$

Since we do not know the a priori optimal thresholds in a domain, we iterate on successively looser definitions. Initially the positive threshold was set to 0.90, and the negative threshold to 0.04, for the results presented here. This means that a potential rule will be rejected unless it covers at least 90% of the positive examples and no more than 4% of the negative examples in the training set. RL generates all rules that meet these criteria using an intelligent breadth-first search. If some of the positive examples in the training set remain uncovered by rules at the 90% level, the thresholds will be loosened[4], and RL will begin the top-down search again. In this way, the best rules are found first, and rules with less coverage are found only if the best rules are inadequate to explain the training set.

3.4 Problem Solving: The Rule Interpreter

By assuming that the model of problem solving is evidence gathering, or heuristic classification, we remove the burden from RL to find error-free rules. The rules predict "MONITOR-ERROR" or "ELEMENT-ERROR" for any beam line or segment of a beam line. However, from a physicist's standpoint it is crucial to be able to localize the error as well as classify it. Having this motivation, we added an outer loop to interpret the rules in the following way.

To localize errors we employ the GOLD Method [Lee and Clearwater, 1987]. This technique delineates the beam line sequentially into so-called "good regions", i.e., lengths of the line in which no error is believed to exist. Each rule makes some relevant comparisons between measured values of the beam trajectory and predicted values within a region. After any element-error rule fires, signifying the end of a good region, the simulated beam is launched again -- with new input parameters based on the monitors immediately after the end of the previous good region -- and the search for the next terminus of a good region begins.

At the same time the interpreter extends regions looking for element errors, monitor-error rules may also identify a particular monitor as erroneous and its corresponding data will subsequently be disregarded. If both element-error rules and monitor-error rules trigger on the same monitor then the error is called a monitor error. This is done because the false positive rate for element errors is much higher than for monitor errors, as discussed below, and because the real-world cost of making a false positive for an element error is much higher than for a monitor error. A more sophisticated conflict resolution and evidence-gathering strategy involving past performance of various rule combinations is under development.

[4]We use a step size of 10% in reducing the positive-coverage threshold; the negative-coverage threshold is kept constant for any run. We are experimenting with other strategies for changing these, and other, thresholds incrementally.

4 Results

4.1 Rules Generated by RL

Appendix B shows examples of two rules generated by RL. A typical rule for each type of error is shown. Even though the rule sets of which these are examples were generated independently in two separate runs, they share a considerable number of common rules. This results from RL's ability to find all plausible rules about the concepts, and from each random sample being large enough to be representative.

4.2 Test Case - SLAC Simulator

We used two training sets to learn rules. The rules were tested on a non-learning set of one hundred simulated cases. The element error rules achieved an average accuracy[5] of 98% and a false positive rate of 13%. Similarly, monitor error rules had an accuracy of 86% and a false positive rate of 5%. The precision[6] of the element error rules was 94% within 3 monitors and for monitor error rules was 95% exactly on the error. Several explanatory comments are necessary to understand these numbers. Three monitors is the minimum length of a good region and the interval within which element error rules can be said to be precise. Also, the effect of an error will usually not become immediately significant, so it is somewhat arbitrary what we mean by "within three monitors of the error." Monitor errors, on the other hand, are local to a particular monitor and must therefore be pinpointed. However, due to the noise in the data, it is possible for a large monitor fluctuation to mimic a monitor error and trigger a monitor error rule. All the rules used had a minimum positive coverage of .7 and a maximum negative coverage of .04. The tabulated results are consistent with our expectations from the rules we requested from RL. The measures of accuracy and precision calculated here, for the first time, provide the expert with a quantitative prediction of the efficacy of his techniques.

4.3 Test Case – SLAC Data

We obtained two separate sets of data from the NRTL that partially overlapped our training beam line. Both rule sets were tried and both found an element error in the same region as that found by an expert using the GOLD Method. A new element was later inserted into the actual beam line to compensate for this error.

4.4 Test Case – CERN Data

In general, every beam line has its own positioning, calibration, and resolution tolerances so that it would be necessary to run RL for every beam line where these tolerances differ in order to best calibrate the rules. However, we have run the rules formed by RL on training data from SLAC on a beam line from the CERN (European Center for Nuclear Research) SPS (Super Proton Synchrotron). This section of beam line from the SPS has 75 magnets, and 36 monitors and is 2.3km long. The data were the actual differences of the

[5]Accuracy for a class of rules means the fraction of times the rules fired on a beam line segment when there was an error of that class present.

[6]Precision is the fraction of times a rule correctly classifies and localizes an error to within some number of monitors.

monitor readings taken over a year apart after several magnets were repositioned. This example was solved by ABLE using the SLAC rules, and the program's element error conclusions agreed with the actual changes made.

5 Discussion

The main point of the present study is to demonstrate the utility and power of a strong device model, the simulator, for providing training examples to an induction system. One of the difficulties of using induction to learn rules for expert systems in medicine, mass spectrometry, or many other important disciplines is the inaccessibility of a large library of training examples. The strong predictive model of beam line physics that was already implemented in a simulation program allowed us to generate realistic cases in large numbers. Thus we could generate training cases to any extent needed to develop the parameters under which we felt learning would be successful, and then generate new test cases without bias. The simulator allowed us to generate and examine hundreds of examples. Thus we were able to see patterns and boundary cases and upgrade our techniques accordingly.

The parameters of the learning system, embodied in RL's half-order theory, require some adjustments from their initial, intuitive settings. For example, the cost of an expert system making false-positive predictions directly affects the threshold value on how many non-exemplars the induction system allows new rules to cover. Intuitively, we would like rules to be as general as possible, but when we consider the cost of false-positives we are forced to make them more specific. Another place where a tradeoff forced us to adjust values empirically was in setting the endpoint markers on the numerical ranges: fine resolution, while desirable in promoting precision in the rules, decreases their generality and increases the run time of the learning program.

One of the difficulties we encountered is suggestive of a fundamental conceptual problem deserving more analysis. In particular, the rule interpreter was designed to terminate a good region when an element-error rule fires. But the rules were initially formed without specifying where the error occurs. Thus using rules to define the ends of segments was found to cause false positive (mis)identification of errors. We then changed the definition of segments used as training examples with substantially better results. This suggests difficulties in formulating rules independently of their use. Interestingly, though, even when the segments were chosen systematically badly the overall performance of the rules after incorporating evidence-gathering knowledge improved significantly and became comparable to our best rules.

This domain involves a physical system and uses exclusively numeric data, unlike many AI systems. Thus, we were unable to exploit RL's ability to use hierarchies of symbolic values in successive specialization. Nevertheless, RL's method appears not to depend on having rich semantics for good performance and is robust enough to deal with this situation. It is obvious, however, that RL's performance depends very much on the accuracy of the simulator.

ABLE has shown the power expert system technology can have on beam line start-up; RL has shown that the level of automation can be increased even more and may lead to further productivity gains for accelerator facilities or in applications with good device models. The generality of the rules across accelerator facilities is untested, however, except for the SLAC and CERN examples. Different beam lines may have different tolerances but the descriptions of the underlying physics are generally the same. Thus rule sets for different beam lines will likely differ only in the endpoint markers for numerical intervals, and not in the features used in rules or in their conjuncts.

We have assumed that a single monitor error cannot be mistaken for an element error and vice-versa. However, in a single set of data it is possible for multiple element errors to be mistaken for a monitor error. In reality this case is not very frequent, but we are, in effect, assuming that a single error is a preferred explanation over a complex of errors. Note that we are not making a global assumption about single faults in a beam line, but we are assuming that each error-free region is broken by just a single fault.

There remain many interesting problems to examine using RL in this domain as an experimental laboratory. We intend to focus next on incremental rule formation, experiments with the efficacy of using rules that predict presence and absence of errors, conflict resolution, and automatic adjustment of interval markers.

Acknowledgments

We extend special thanks to Haym Hirsh for generous, constructive criticisms of early drafts of this paper and to James Rice for help optimizing RL on the Explorer. Li-Min Fu wrote the first version of RL. We also express our gratitude to Dr. Martin Lee for his expertise in accelerator physics and to Stephen Kleban, SLAC, and CERN for supplying us with data.

A Features used by FeatureMaker

Features used in the redescription (abstraction) of the training data, and in the left-hand sides of rules:

OBJECTIVE-VALUE:
A measure of the mean square difference between measured and simulated beam positions for the monitors in the segment.

The following definition will be useful in defining the other features:

DIFFERENCE TRAJECTORY (DT):
The absolute value of the difference between the measured and simulated beam positions at a monitor.

LARGEST-DIFFERENCE-TRAJECTORY:
The largest difference trajectory in the segment.

DOWNSTREAM-NEIGHBOR-OF-LARGEST-DT:
The difference trajectory at the monitor immediately after the monitor with the largest difference trajectory.

DOWNSTREAM-NEXT-NEIGHBOR-OF-LARGEST-DT:
The difference trajectory at the monitor two after the monitor with the largest difference trajectory.

B Sample Rules

Two typical rules generated by RL in a run of 300 training examples:

((LARGEST-DIFF-TRAJ (GT 0.3))
(DOWNSTREAM-NEIGHBOR-OF-LARGEST-DT (GT 0.5)
(DOWNSTREAM-NEXT-NEIGHBOR-OF-LARGEST-DT
(GT 0.5))) ⇒ (ELEMENT-ERROR YES)
80.1% of positives in the training set matched (181/226)
3.6% of negatives in the training set matched (24/674)

((LARGEST-DIFF-TRAJ (GT 0.7))
(DOWNSTREAM-NEXT-NEIGHBOR-OF-LARGEST-DT
(LE 0.7))) ⇒ (MONITOR-ERROR YES)
85.5% of positives in the training set matched (171/200)
3.7% of negatives in the training set matched (26/700)

References

[Brown *et al.*, 1982] J.S. Brown, R. Burton, and J. DeKleer. Pedagogical and Knowledge Engineering Techniques in SOPHIE I, II, and III. In Sleeman, D.H. and Brown, J.S. (editors), *Intelligent Tutoring Systems*. Academic Press, London, 1982.

[Buchanan and Mitchell, 1978] Bruce G. Buchanan and Tom M. Mitchell. Model-Directed Learning of Production Rules. In Waterman, D.A. and Hayes-Roth, F. (editors), *Pattern-Directed Inference Systems*, pages 297-312. Academic Press, New York, 1978.

[Clearwater and Lee, 1987] Scott H. Clearwater and Martin J. Lee. Prototype Development of a Beam Line Expert System. In Lindstrom, E.R. and L.S. Taylor (editors), *Proc. 1987 Particle Accelerator Conf.*, pages 532-534. Washington, D.C., March, 1987.

[Fu, 1985] Li-Min Fu. *Learning Object-Level and Meta-Level Knowledge in Expert Systems*. PhD thesis, Stanford University, March, 1985.

[Fu and Buchanan, 1985] Li-Min Fu and Bruce Buchanan. Learning Intermediate Concepts in Constructing a Hierarchical Knowledge Base. In *Proc. IJCAI 85*, pages 659-666. IJCAI, Los Angeles, CA, August, 1985.

[Kunstaetter, 1987] R. Kunstaetter. Intelligent Physiologic Modeling: An application of knowledge based systems. *Computer Methods and Programs in Biomedicine* 24(3):213-225, 1987.

[Lee *et al.*, 1987a] Martin J. Lee, Scott H. Clearwater, Stephen D. Kleban, and Lawrence J. Selig. Error-finding and Error-correcting Methods for the Start-up of the SLC. In Lindstrom, E.R. and L.S. Taylor (editors), *Proc. 1987 Particle Accelerator Conf.*, pages 1334-1336. Washington, D.C., March, 1987.

[Lee *et al.*, 1987b] Martin J. Lee, S. Clearwater, E. Theil, and V. Paxson. Modern Approaches to Accelerator Simulation and On-Line Control. In Lindstrom, E.R. and L.S. Taylor (editors), *Proc. 1987 Particle Accelerator Conf.*, pages 611-613. Washington, D.C., March, 1987.

[Lee *et al.*, 1987c] Martin Lee, S. Kleban, S. Clearwater, *et al.* Analysis of the Orbit Errors in the CERN Accelerators Using Model Simulation. In *Proc. Europhysics Conference on Control Systems for Experimental Physics.* 1987 (in press).

[Lee and Clearwater, 1987] Martin J. Lee and Scott H. Clearwater. GOLD: Integration of Model-based Control Systems with Artificial Intelligence and Workstations. In *Proc. of the Workshop on Model-based Accelerator Controls*, pages 31-38. Upton, New York, August, 1987.

[Michalski, 1983] R.S. Michalski. A Theory and Methodology of Inductive Learning. *AI*, 20(2):111-161, February, 1983.

[Samuel, 1963] A.L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. In Feigenbaum, E.A. and Feldman, J. (editors), *Computers and Thought*, chapter 3, pages 71-105. McGraw-Hill, 1963.

[Woodley *et al.*, 1983] M.D. Woodley, M.J. Lee, J. Jaeger, and A.S. King. COMFORT, Control of Machine Functions OR Transport Systems. In *Proc. 1983 Particle Accelerator Conference*. Santa Fe, New Mexico, March, 1983.