Co-evolutionary Learning: Machines and Humans Schooling Together

Elizabeth Sklar
Dept. of Computer Science
Brandeis University
Waltham, MA 02254, USA
+1-781-736-2741
sklar@cs.brandeis.edu

Alan D. Blair
Dept. of Computer Science
University of Queensland
4072, Australia
+61-7-3365-1999
blair@cs.uq.edu.au

Jordan B. Pollack
Dept. of Computer Science
Brandeis University
Waltham, MA 02254, USA
+1-781-736-2741
pollack@cs.brandeis.edu

Abstract

We consider a new form of co-evolutionary learning in which human students and software tutors become partners in a cooperative learning process. While the students are learning from the tutors, the tutors will also be learning how to do their job more effectively through their interactions with the students. Earlier work on game-playing machine learners has brought to light important issues for co-evolutionary learning; in particular, that these interactions can be modeled as a *meta-game* between teachers and students, and that learning may fail due to *suboptimal equilibria* – for example, because of *collusion* between the individual players – in this meta-game of learning. We discuss some of the challenges that these issues may raise for the designers of intelligent software tutoring systems and describe a prototype Java applet that we have built in an effort to explore how these challenges may best be met.

1. Introduction

Advancing technology is opening up exciting new possibilities for education. We can envision a future paradigm in which students play a much greater role in structuring their own education, with the assistance of sophisticated software agents that can guide them through the learning process, adapting to their own individual needs. These software tutors could fulfill a number of functions – searching for information requested by the students, directing the students to appropriate sources and multimedia teaching materials, asking questions, suggesting certain topics or lines of inquiry – making sure the students gain a good grasp of the core concepts while encouraging development of a student's own particular skills and interests.

This interaction can be viewed as a form of *co-evolutionary learning* between the human students and the software tutors. While the students are learning from the tutors, the tutors will also be learning how to do their job more effectively through their interactions with the students.

We have studied co-evolutionary learning in another context – namely, that of machine learners improving their strategies in competitive games by playing each other and observing the results. Such studies have brought to light important issues for co-evolutionary learning – in particular, that these interactions may be modeled as a *meta-game* between teachers and students, and that learning may fail due to *suboptimal equilibria* – for example, because of *collusion* between the individual players – in this meta-game of learning. We believe that similar issues are likely to arise in the context of human-student/software-tutor co-evolution, and that a better understanding of these issues will help improve the design and implementation of software tutoring systems.

This paper is arranged as follows: Section 2 provides background on co-evolutionary and human learning. Section 3 discusses the problems of collusion and suboptimal equilibria in the meta-game of learning. Section 4 examines certain game learning domains which seem to have avoided these problems. Section 5 describes an experimental educational software Java applet that attempts to address some of these issues.

2. Co-Evolutionary Learning, Human Learning

The goal of *machine learning* (ML) is to design software systems that can learn, through interacting with their environment, information which will help them to perform particular tasks better. Such systems, if successful, will automatically adapt to new environments without re-programming, thus saving humans the trouble of designing all the relevant features by hand.

The success of a machine learning system depends very much on the learning environment in which it is placed. Typically, such a system extracts information from its environment, as it is ready; then it may need to be placed in a new environment in order to progress. *Incremental* learning (Langley, 1995) occurs when a learner is placed into a pre-defined series of environments one after the other, as it progresses. However, designing an appropriate series of environments is difficult and expensive. These costs could be avoided if there were some way for the learner and its environment to *co-evolve* with each other, so that the one would always be appropriate for the other.

The subfield of co-evolutionary machine learning attempts to achieve just this, building on earlier work in genetic algorithms and genetic programming. Some pertinent applications include Prisoner's Dilemma (Axelrod, 1984; Haynes et al., 1995), the game of tag (Reynolds, 1994), tic-tac-toe (Angeline & Pollack, 1993), and backgammon (Pollack et al., 1996).

How does the notion of co-evolutionary learning apply to human learning, and, in particular, fit into today's framework of intelligent tutoring? The current trend in educational culture transfers control away from the traditional teacher in favor of the student, the learner (Corte, 1995). In this type of *constructionist* environment, students are able to explore ideas for themselves without having to stick to a fixed curriculum (Papert, 1993); and students at all levels of ability are provided with an opportunity to learn. Forrester promotes *learner-centered learning*, where students are not considered "passive receptors". Students and teachers participate in the learning process together as colleagues (Forrester, 1992).

The issue of *learner control* is addressed by John Seely Brown. He describes systems where the "locus of control" belongs either solely to the computer (e.g. frame-based CAI) or solely to the student (e.g. LOGO (Papert, 1980)). He cites disadvantages with both extremes – with the former, the student is restricted to a fixed path and cannot explore different avenues of interest; with the latter, the student may get stuck and may not be exposed to the full range of problems in the domain being learned. Brown maintains that some combination of the two schools is best, but is also the hardest to build (Brown & Burton, 1978).

Intelligent Tutoring Systems (ITS) branch out from frame-based approaches, exhibiting such desirable characteristics as providing a trace of a problem-solving session, individualizing for each user, dynamically selecting what to do next and coaching the user at opportune times (Clancey, 1986). Early work combined some or all of these, each emphasizing different issues, such as memory modelling (Schank, 1981; Kolodner, 1983), construction of rules (ACT) (Anderson, 1982), modelling student's misconceptions (VanLehn, 1983; Soloway et al., 1981). But all of these require considerable engineering effort on the part of the system designer/programmer.

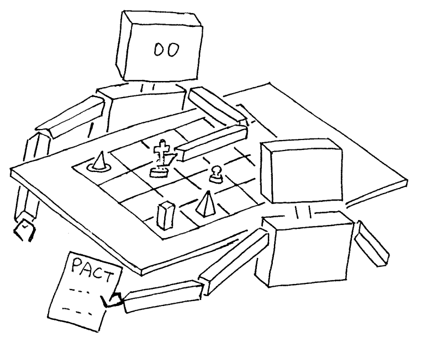
More recent work is often still highly dependent on a large engineering effort. POLA (Conati & VanLehn, 1996) uses probabilistic methods to model students solving problems in introductory physics. This work involves complex analysis of the problems being solved and attempts to combine knowledge tracing and model tracing. INSTRUCT (Mitrovic et al., 1996) tries to induce models of procedural skill by observing students and collecting information both implicitly and explicitly.

Intelligent Tutoring Systems take on harsh criticism from (Kinshuk & Patel, 1996), who state that "the absence of substantial and usable ITS after years of research and development activities should encourage a re-examination of the objectives and the design principles of a tutoring system." In this work, we are attempting just such a re-examination by considering a role for co-evolutionary learning in an ITS. We believe that the same environmental elements necessary for a successful co-evolutionary learning scenario are also relevant for achieving success in human learning.

3. Games, Meta-Games and Collusion

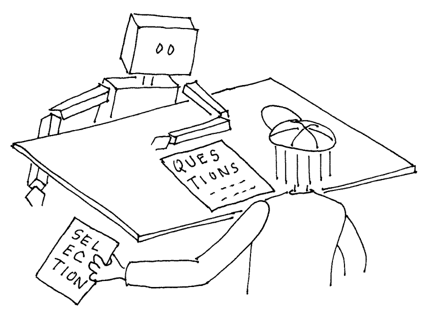
A good opportunity for studying co-evolutionary machine learning is provided by strategic games like chess, tic-tac-toe or backgammon. The theory is that several machine learners trying to master a competitive game could all learn to improve their strategies simultaneously by playing each other and observing the outcomes. Each player would be acting as a learner in a learning environment defined by its pool of potential opponents. While this idea has been around since the early days of artificial intelligence (Samuel, 1959; Michie, 1961), interestingly some applications of it have been very successful while others have run into serious difficulties.

While attempting to understand the nature of these difficulties, we have observed that a machine learning system can, in general, be modeled as an interaction between a teacher and a student (and possibly other participants) and that the manner in which these participants are rewarded defines a *meta-game of learning*, or MGL (Pollack & Blair, 1998), to which we apply a game theoretical analysis. The game-playing machine learner could be thought of as the student, while its opponent assumes the role of the teacher. The teacher's goal is to discover the student's weaknesses and help to correct them, while the student's goal is to placate the teacher and avoid further correction. In the case of game-playing learners, opposing players essentially take turns filling the roles of teacher and student and are thus evaluating *each other*. This situation provides opportunities for *collusion* between teacher and student, which show up as *suboptimal equilibria* in the MGL. Many of the problems encountered by co-evolutionary machine learners can be attributed to suboptimal equilibria of this kind.



co-evolutionary machine learners evaluate each other by playing games and observing the outcomes; they can *collude* by:

- *narrowing the scope* – playing the same kinds of games over and over
- *silent pact* – trading easy wins or playing for a 'draw', rather than a win or loss



software tutor and human student evaluate each other through a system of selection, attention and evaluation; they can *collude* by:

- *narrowing the scope* – sticking to material that is particularly favored by the student
- *silent pact* – avoiding difficult topics and providing entertainment in exchange for positive evaluation

Figure 1 The MGL: a picture of collusion

We believe the interaction between human students and software tutors can also be modeled as an MGL, as illustrated in Figure 1. This MGL differs from that of game-playing learners, taking into account the complex mix of motivations that the human student brings to the educational task. Some key issues are:

- Students should be exposed to the full *scope* of material being learned. But...
  - if the tutor is rewarded for providing more comprehensive coverage, the opportunity to be flexible and adapt to each student's individual needs may be lost; and
  - if the students are presented with too much unfamiliar material, they may become confused; i.e.

...a suboptimal equilibrium may result where the students are exposed to a broad range of material but are unable to absorb it.

- Students should receive positive *feedback*. But...
  - if the tutor is rewarded for just asking questions the student is able to answer, the tutor may stick to a narrow range of material already familiar to the student and avoid new or difficult concepts;
  - if the students are always shown familiar material, they may get bored; i.e.

...a suboptimal equilibrium may result where the scope of the material is narrowed and the students are not learning new things.

- Students should be *challenged* to explore new ideas. But...
  - if the tutor is rewarded for just asking questions the student is *unable* to answer, the tutor may end up asking questions that are well beyond the student's ability; and
  - if students are constantly attempting questions they can't answer, they may get discouraged; i.e.

...a suboptimal equilibrium may result where the students are confused and not moving ahead.

- Students should be *interested* and *attentive*. But...
  - if the tutor's reward is based just on the attention, selection and/or explicit evaluation provided by the student, the tutor may become either obsolete (where the student is self-taught) or a mere entertainer (pandering to the student's whims in ways that do not promote learning); and
  - if the students are rewarded based on their ability to 'get along' with the tutor, their own interests and needs may be subverted; i.e.

...a suboptimal equilibrium may result where the evaluation mechanism distorts the learning process.

These issues of *scope*, *feedback*, *challenge*, *interest* and *attention* need to be considered both individually and collectively, allowing for dynamical adjustment of each during the learning process. We believe that a careful balancing of these criteria will be required in order to avoid suboptimal equilibria in the MGL.

4. Avoiding Suboptimal Equilibria and Collusion

There have been a few notable cases of ML applications in which the problems of suboptimal equilibria and collusion have apparently been avoided. One such instance came to light when (Tesauro, 1992) compared two different methods for training neural networks to play the game of backgammon. The first network was trained on a large database of hand-crafted positions, with corresponding moves chosen by a human expert; the second network was trained by having it play against itself thousands of times and using the outcome of each game to make a small adjustment in its strategy according to the *temporal difference* (Sutton, 1988). Surprisingly, the network that was trained by self-play, although initially playing a poor (essentially random) game, eventually surpassed the network trained on the expert database.

A second example is provided by the *evolving virtual creatures* (EVC) domain. In a game devised by Karl Sims, two virtual creatures compete in a world with simulated physics for control of a cube initially placed between them (Sims, 1995). In each round of competition, all creatures from one species played against the champion of the other species from the previous round. Over several generations, competing species were observed to leap-frog each other in evolutionary arms races, as they each discovered methods for reaching the cube and then further evolved strategies to counter the opponent's behaviour.

While the exact reasons for these successes were unclear at first, our hypothesis is that these two domains have special attributes which help to prevent sub-optimal equilibria in the MGL. In backgammon, the randomness of the dice rolls sometimes allows a weak player to score a victory over a strong one – an experience from which both can learn – and also tends to lead the learner into a much larger portion of the strategy space than would likely be explored in a deterministic game (Tesauro, 1992). Moreover, 'draws' are impossible in backgammon and its particular dynamics work to prevent trading of easy wins (Pollack & Blair, 1998).

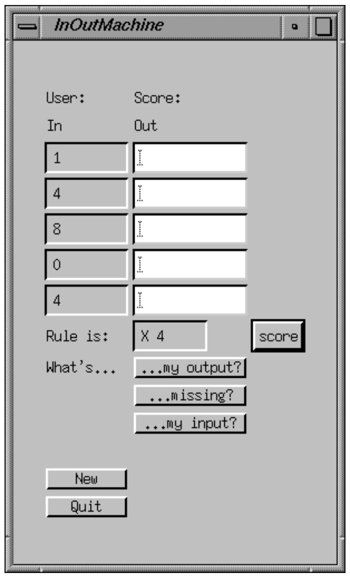
Both the backgammon and EVC domains provide a broad spectrum of opportunity such that the machine learner has available to it, at any given time, a number of avenues for improvement and a number of ways to proceed along each avenue. In backgammon, there are many aspects of the game which can be developed independently. Virtual creatures, like their biological counterparts, can improve by developing a slightly longer arm, slightly better sensors, or becoming slightly faster, etc. These are in contrast to some other applications, where learning can only proceed along a set path.

Perhaps these domains providing a broad spectrum of opportunity, where learners can explore in any order, may be compared with *constructionist* learning environments that allow the user to explore a problem from a number of different angles, and thus gain a more solid grasp of the fundamental concepts (Wilensky, 1996). It would be interesting to see whether judicious attention to these attributes in the design of software tutoring systems might work to prevent collusion in the MGL.

5. The InOutMachine: A Prototype

We have built an educational software tool as a prototype for exploring these concepts. Embedded in the application are software tutors, or *tubots*. The prototype is called InOutMachine (see Figure 2), which students can use to practice simple arithmetic.

InoutMachine is based on the fundamental principle that arithmetic equations start with an input, apply a rule to that input and end with a related output. Users (students) are asked to provide the "missing elements" – the input and/or output – for a given set of addition, subtraction, or multiplication problems.



We use ideas from genetic algorithms (GA) (Holland, 1975) to select tubots for students to work with. Each type of problem is encoded to form a *chromosome* that translates into a single *tubot* and an entire problem set is a *population* of tubots.

Tubots are represented by 6-bit binary strings, where the first two digits indicate the operator (+, − or ×) and the last four digits indicate a number from 0 to 15. Thus, there are 64 types of problems. For example, in the problem set shown on the left, the type of problem being learned is "× 4". The encoding for the tubot that helps a student learn "× 4" is $100100_{base2} = 36_{base10}$.

A student completes five problems given for "× 4" and is then given another problem set to complete, corresponding to another tubot. There are (currently) 10 tubots in a population, and a student is given one problem set for each tubot in the process of "evaluating" that population. Once the student has been visited by all 10 tubots in the population, the *next* population will be generated. This is done by selecting any tubots from the current population whose score exceeds a given threshold and reproducing them using mutation of a single randomly chosen bit.

Figure 2 The InOutMachine

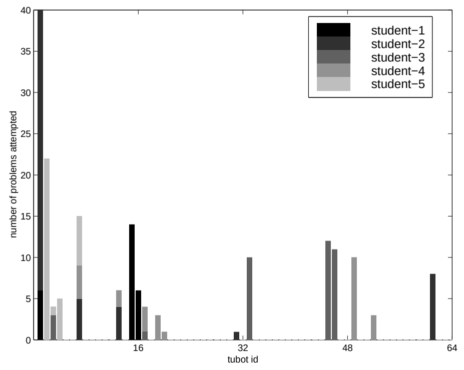
In this way, the student is exposed to a small portion of the domain (10 out of 64 tubots) at one time and is only introduced to new portions of that domain when ready. Additionally, by using mutation of only one bit to select a new tubot, the system is forced to choose a new tubot that represents a portion of the domain that is relatively close to the old tubot. For example, if a student has mastered "$\times$ 4", the replacement tubot will differ in either the operator or the number but not both, so a subsequent tubot could be "+ 4" or "$\times$ 5", but not "+ 5". This harkens back to the idea of incremental learning, where the new material that is presented should be only slightly more difficult (or different) from previously learned material.

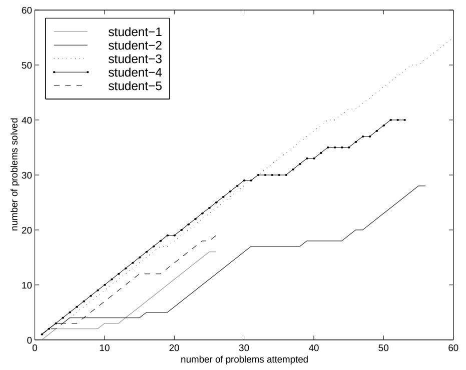
It should be noted that this method of reproduction does not embody any heuristic for deciding which tubot should be selected next, only that the new tubot will vary *slightly* from the previous. We have chosen to follow this course purposely, rather than pre-engineer "levels" within the domain. For example, rather than deciding *a priori* that no student should be presented with a multiplication problem until s/he has mastered all addition and subtraction problems, we let the system and the student together select which new problems are appropriate, based on the student's prior performance with the system.

A useful and interesting by-product of this co-evolutionary learning approach is an emerging student model, expressed in terms of the tubot population. A database is maintained for each student, containing a score for each tubot. When a new student comes to the system, her scores are all initialized to zero; then as she proceeds, scores are collected that describe her initial knowledge. Later, as that knowledge broadens, the scores change along with the student, giving an emerging picture of the student's comprehension of the domain being learned.

For experimental purposes, the InOutMachine was designed to operate in three modes: *user*, *random* and *learning*. In user mode, the student determines which tubots to work with, exemplifying a system driven primarily by user *feedback*. Random mode, where the tubot is selected purely at random, simulates a system making selections primarily to maximize *coverage*. In learning mode, the GA determines the tubots with which the student needs practice and continues to present those tubots to the student, until the student's score with those tubots improves; here is where a balance of selection strategies is used.

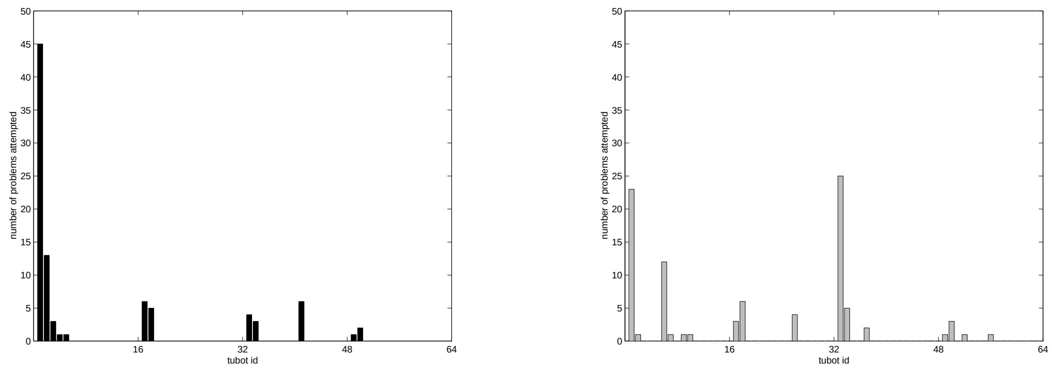
Initially, InOutMachine was implemented on a Macintosh and installed in a primary grade classroom (see Figure 3). Recently, we have re-implemented the system as a Java applet (see Figure 4).





These two graphs show data taken from the classroom experiment. 22 students participated in the study; data collection occurred over a one month period. For clarity, data from 5 students were chosen and plotted; all three modes are combined. The left graph illustrates the coverage provided by the system, not only are the students being exposed to a range of tubots, but also different students and different tubots are being matched together. The right graph illustrates the success rate for these five students; some students solve problems correctly at a steady pace, while others' success rates level off and then climb again.

Figure 3 Classroom Data Analysis



These two graphs contain data collected over the Web, from the Java applet. 24 different users are represented. The left graph shows data taken in user mode, the right graph shows learning mode. 90 problem sets are illustrated in each graph. A better mix of tubots are selected by the learning mode than user mode.

Figure 4 Web Data Analysis

Our hypothesis is that, while user mode emphasizes the interest and positive feedback needs of a learning system, if left on their own, students will stick to a narrow scope in the domain, steering away from new and challenging areas. Similarly, while random mode forces exploration of broader scope and presentation of challenges, this mode fails to accommodate the need for positive feedback or maintaining user interest. Learning mode attempts to balance these attributes, allowing the student to drive the learning process through his performance in the domain rather than explicitly specifying his own preferences or relinquishing all control to a pre-defined syllabus.

6. Conclusion and Further Work

In ongoing work, we are refining the Java version of the InOutMachine. This version is accessible at: `http://www.demo.cs.brandeis.edu/inout`. Due to the wealth of users available on the Internet, we are now collecting a much larger data set. We are also continuing with classroom experiments, since there we have more control over the population of users and can track progress and demographics of individuals more reliably than on the Web.

In this paper, we have built on earlier work in machine learning, using a *meta-game of learning* analysis to identify a range of issues and challenges which we expect to play an important role in the successful design of intelligent tutoring systems. We have presented a new form of co-evolutionary learning in which human students and software tutors become partners in a cooperative learning process. Branching out from typical frame-based educational applications, such evolving software tutors could potentially fulfill a variety of educational needs, taking advantage of the particular strengths that a computer partner brings to the mutual learning task.

7. Acknowledgments

Thanks to Maria Hastings Elementary School for participating in the classroom study. This work was partially funded by a University of Queensland Postdoctoral Fellowship.

References

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychology Review*, 89.

Angeline, P. J. & Pollack, J. B. (1993). Competitive environments evolve better solutions for complex tasks. In Forrest, S. (Ed.), *Genetic Algorithms: Proc. 5th Int'l Conference (GA93)*.

Axelrod, R. (1984). *The Evolution of Cooperation*. Basic Books.

Brown, J. S. & Burton, R. B. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2(2).

Clancey, W. J. (1986). Intelligent tutoring systems: A tutorial survey. Technical Report STAN-CS-87-1174, Stanford University.

Conati, C. & VanLehn, K. (1996). Pola: a student modeling framework for probabilistic on-line assessment of problem solving performance. In *Proc. 5th Int'l Conf. User Modeling (UM-96).*

Corte, E. D. (1995). Learning theory and instructional science. In Reimann, P. & Spada, H. (Eds.), *Learning in Humans and Machines: Towards an Interdisciplinary Learning Science.* Pergamon.

Forrester, J. (1992). System dynamics and learner-centered-learning in kindergarten through 12th grade education. Technical Report D-4337, MIT.

Haynes, T., Wainwright, R., Sen, S., & Schoenefeld, D. (1995). Strongly typed genetic programming in evolving cooperative strategies. In Eshelman, L. (Ed.), *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, San Francisco, CA.

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems.* University of Michigan Press, MI.

Kinshuk & Patel, A. (1996). Intelligent tutoring tools: Redesigning itss for adequate knowledge transfer emphasis. In *Proc. 1996 Int'l Conf. on Intelligent and Cognitive Systems.*

Kolodner, J. (1983). maintaining organization in a dynamic long-term memory. *Cognitive Science*, 7.

Langley, P. (1995). Order effects in incremental learning. In Reimann, P. & Spada, H. (Eds.), *Learning in Humans and Machines: Towards an Interdisciplinary Learning Science.* Pergamon.

Michie, D. (1961). Trial and error. *Science Survey*, part 2:129–145.

Mitrovic, A., Djordjevic, S., & Stoimenov, L. (1996). Instruct: Modeling students by asking questions. *User Modeling and User-Adapted Interaction*, 6(4).

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas.* BasicBooks.

Papert, S. (1993). *The Children's Machine.* BasicBooks.

Pollack, J. B. & Blair, A. D. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning (to appear).*

Pollack, J. B., Blair, A. D., & Land, M. (1996). Coevolution of a backgammon player. In Langton, C. (Ed.), *Proceedings Artificial Life V.* MIT Press.

Reynolds, C. W. (1994). Competition, coevolution and the game of tag. In Brooks, R. A. & Maes, P. (Eds.), *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Cambridge, MA. MIT Press.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229.

Schank, R. C. (1981). Failure-driven memory. *Cognition and Brain Theory*, 4(1).

Sims, K. (1995). Evolving 3d morphology and behavior by competition. In *Proceedings of the Fourth International Conference on Artificial Life.* MIT Press.

Soloway, E. M., Woolf, B., Rubin, E., & Barth, P. (1981). Meno-ii: An intelligent tutoring system for novice programmers. In *Proc. 7th Int'l Joint Conf. on Artificial Intelligence (IJCAI).*

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning 3.*

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning 8.*

VanLehn, K. (1983). Human procedural skill acquisition: Theory, model, and psychological validation. In *Proceedings of the National Conference on AI.*

Wilensky, U. (1996). Making sense of probability through paradox and programming. In Kafai, Y. & Resnick, M. (Eds.), *Constructionism in Practice: Designing, Thinking and Learning in a Digital World*, Mahwah, NJ. Erlbaum.