

USING SIMULATION ACROSS THE CURRICULUM

Matthew Dickerson

*Department of Mathematics and Computer Science
Middlebury College, Middlebury VT
dickerso@middlebury.edu*

Timothy Huang

*Department of Mathematics and Computer Science
Middlebury College, Middlebury VT
huang@middlebury.edu*

Ingrid Russell

*Department of Computer Science
University of Hartford
irussell@mail.hartford.edu*

ABSTRACT

Computer simulation has been well recognized as a valuable modeling tool in many disciplines, and many computer science educators already make use of simulations in various courses. To fully exploit its benefits, however, we believe that computer simulation should be integrated more broadly and intentionally across the curriculum rather than taught in an isolated, ad hoc fashion. In this paper we advocate the teaching and use of simulations across the undergraduate computer science curriculum at a liberal arts college. We focus on the development of simulations by students, though we also address the use of existing simulations. We present both advantages and disadvantages, and argue that in many situations the advantages outweigh the drawbacks. We then present in outline specific examples of how, with limited resources, simulations can be taught and used in several courses that are widely taught at many undergraduate institutions. An on-line version of this paper along with several resources and links can be found at: www.middlebury.edu/~dickerso/simulation/.

1. INTRODUCTION

The Encyclopedia Britannica Online (www.eb.com) defines computer simulation as "the use of a computer to represent the dynamic responses of one system by the behavior of another system modeled after it. A simulation uses a mathematical description, or model, of a real system in the form of a computer program." The Webopedia (webopedia.internet.com) defines

computer simulation more simply as "the process of imitating a real phenomenon with a set of mathematical formulas." The definition goes on to state, "Advanced computer programs can simulate weather conditions, chemical reactions, atomic reactions, even biological processes. In theory, any phenomena that can be reduced to mathematical data and equations can be simulated on a computer." In particular, computers can also simulate themselves, i.e., they can simulate the complex computational processes by which they operate. In this discussion, we focus on the computer simulation of computational processes themselves.

We examine the use of computer simulations as a pedagogical tool for teaching computer science, with the emphasis that simulations should be used across the curriculum rather than only in isolated instances. We begin by discussing some advantages and disadvantages of using simulation. We then describe several specific simulation projects and assignments that can be implemented with minimal resources and used in widely taught courses. In the context of smaller undergraduate institutions with a computer science major requiring only about twelve courses, we have found that the pros far outweigh the cons. Moreover, we have found that using simulation has been effective across the curriculum, from introductory courses for non-majors and lower-level courses in data structures and machine organization, to upper-level courses in operating systems, algorithms, theory of computation, and artificial intelligence.

2. ADVANTAGES AND DISADVANTAGES

We begin with an overview of some reasons why we advocate the use of computer simulations across the curriculum, and then discuss some potential obstacles and drawbacks.

2.1 Advantages

The benefits of using computer simulation are many, and we outline here the most important ones. First, students can gain a better understanding of the underlying concepts of a system, task, or algorithm by studying it via simulation. For example, one of the best ways for one to really understand an algorithm is to implement it. In building a simulation for some computational process, students are pushed to think at the level of individual steps and specific parameters. This helps ensure that they recognize and understand not only the overall concepts but also the subtle details of the algorithm or computational process.

Second, computer simulations often provide a useful abstraction of the components of a system. The parts of a large system interact with each other in many ways. A student trying to understand the workings of the whole system can easily get confused following the myriad, often implementation-specific relationships among components. Simulations can be beneficial by facilitating focused examination of individual parts. As students learn the specific roles of individual parts, they become better prepared to grasp how those parts come together.

Third, existing simulations help students gain a better sense of how systems operate. Any large system will probably have a number of parameter variables that influence its operation. Students running simulations can test how different input parameters affect the system's

performance. Furthermore, they can usually run these simulations at their own pace. This allows them to see the effect of each step and to spend more time on steps that are less clear.

Fourth and finally, computer simulations are themselves used in many important real world applications. (Indeed, computer simulation is still a traditional topic in many data structures courses [W99,pp.196-197].) Some examples include the modeling and predicting of: weather conditions, chemical reactions, vehicle dynamics, biological processes, and economic systems. The Webopedia adds, "In addition to imitating processes to see how they behave under different conditions, simulations are also used to test new theories. After creating a theory of causal relationships, the theorist can codify the relationships in the form of a computer program. If the program then behaves in the same way as the real process, there is a good chance that the proposed relationships are correct." Thus, by learning to design and use computer simulations, students acquire a fundamental research tool. They gain valuable practice not only from implementing simulations but also from developing experiments to confirm or challenge their understanding of the simulated system's behavior.

2.2 Potential Drawbacks and Other Approaches

We now consider some potential drawbacks to using simulations, along with some alternative pedagogical approaches.

One of the biggest challenges to some simulations is the problem of finding realistic test data. Many simulations use a stream of input data to drive the simulation. For example, an operating system's CPU scheduler takes a stream of process requests for CPU time and decides the order in which the requests are serviced. To accurately assess the effectiveness of different strategies and configurations, a student should be able to provide the simulation with a stream of input data that resembles what an actual system might encounter.

A related issue is that some systems are so complex that any simulation is likely to be an over-simplification [SGG00,p.165]. As the Webopedia states, "simulation is extremely difficult because most natural phenomena are subject to an almost infinite number of influences. One of the tricks to developing useful simulations, therefore, is to determine which are the most important factors." A critical factor to the performance of many systems is how human beings actually use those systems. Unfortunately, straightforward mathematical models of usage patterns may be difficult to construct, and faulty models may lead students to arrive at incorrect conclusions about how to achieve optimal performance.

An alternative to using computer simulations is to have students implement "the real thing." For example, in the area of operating systems, students could build or enhance a working operating system with limited functionality. Given enough time and resources, on the part of both students and faculty, this approach probably would provide the most comprehensive educational experience. There seems to be a substantial pedagogical benefit to requiring students to examine the details of an actual operating system, including how various subsystems interact. This approach has been used successfully with the Nachos instructional operating system [CPA93] at many large universities, including Duke University, the University of California, and the University of Washington. In those environments, students enter the

operating systems course expecting to work, say, 20 hours per week for 14 or 15 weeks. Even so, the assignments are so complex that they usually work in teams of 4 or 5 students. In a liberal arts college environment, with fewer courses in the major, fewer students in each class, fewer resources for teaching and lab assistants, and an expectation that the course workload will leave time for other classes, a simulation-based approach may be more realistic.

Furthermore, there are disadvantages to working on a real system. Students may miss important theoretical concepts while trying to debug complex code. For some students, trying to write or even modify a real operating system is such a monumental task that they focus simply on getting their systems to work in the most basic manner, without really understanding the big picture. Also, from the perspective of instruction time, a course based on implementing a real system would require much more in-class time for explaining and clarifying the assignments. This might necessitate the removal of other important concepts from the course curriculum.

Yet another pedagogical approach involves designing real experiments to perform on existing, implemented systems [D99]. This approach is akin to reverse engineering, in that students try to deduce system parameters by examining output rather than changing parameters to study the effect on output. We see this as a complementary rather than competing approach.

3. EXAMPLES FROM ACROSS THE CURRICULUM

We now provide a small number of examples of simulation projects and assignments from across the curriculum, illustrating a variety of the uses and benefits discussed in the previous section.

3.1 Operating Systems: Trace-driven simulation

An upper-level "Operating Systems" (OS) course provides perhaps the most natural context for students to learn by building simulations. Since the various components of an OS are themselves blocks of program code, an assignment or project based on building a simulation could involve constructing an abstracted version of one of those components. This approach provides the opportunity for students to wrestle with OS concepts at a detailed level, and it can be used even in class situations where it would be infeasible for students to modify and enhance an actual, working OS. Over the years, students in our OS courses have consistently and emphatically pointed to the simulation assignments as among the experiences most helpful to their mastery of key concepts.

Most components of an OS, including the CPU scheduler, memory manager, and disk scheduler, can be readily incorporated in a simulation assignment or project. For example, one project we have used involves simulating a virtual memory manager. Students write simulations that handle a stream of memory accesses from multiple processes. Besides implementing the scheduler and several memory management schemes, students also design experiments to test the effect that allocation strategies and memory access patterns have on cache hit ratios and page fault rates. An example of a student-written simulation of four virtual memory management

strategies can be found at the web page for this paper: www.middlebury.edu/~dickerso/simulation/. See also [CNE].

The biggest problem that we have encountered in using simulations to teach operating systems concepts is that the data used, e.g., the streams of memory references, may be unrealistic. It is difficult to predict or model a realistic pattern of requests to a CPU, to memory, or to a disk. This can be addressed by saving to a log the sequence of references generated by a real system. This trace could then be used to drive the operation of the simulation.

3.2 Simulations of Distributed Architecture and Machine Learning

The use of simulations can help students understand the theoretical concepts of neural networks, the branch of artificial intelligence (AI) dealing with cognitively inspired models for distributed knowledge representation and machine learning.

A neural network is a function approximator that is specified in part by a set of real-valued parameters called weights. The network can be trained to perform classification tasks by presenting it with a series of known input / output associations and using a learning algorithm to modify the network's weights. If trained effectively, the network can produce the correct output classification not only for previously encountered input cases but also for new, previously unseen cases.

We have used simulations of machine learning models to teach neural network learning to students in both lower-level and upper-level courses. In one project, students work on the simulation of a single layer network with modifiable weights, observe how it can be trained to perform some classification tasks, and explore the different types of problems it can solve. Students study the model's convergence rate and the types of problems for which a solution is guaranteed. They reach the conclusion that convergence is guaranteed for linearly separable classification problems but not for other problems, such as for computing XOR.

Through these simulations, students study the effect of factors such as learning rate, order of presentation of training data set, the number of training iterations, and the addition of a momentum constant on the rate of convergence and the number of training epochs required for the network to learn the associations. Students then test the performance of these models in making correct associations, particularly for test data that is not close to the data used for training.

Several highly graphical neural network simulators are now available on the web which students can use. One such package is BrainWav which is written in Java and runs directly from a web browser [SWG]. BrainWav supports the modeling of several neural network architectures including competitive learning, backpropagation, Hebbian learning, and Hopfield learning. As an advanced project in our upper level AI course, students use these simulators to create a simple character or word recognition network. Other sites that include web-based demos are also available that help students understand neural network concepts and models [AU][MH].

3.3 Simulations in Theoretical Computer Science

Other courses where we have used projects involving some type of simulation include the "Theory of Computation" and "Design and Analysis of Algorithms" courses. These are required courses for the computer science major and are usually taken by students during their third or fourth year, with CS-2 and Discrete Mathematics as prerequisite courses. Although the projects we have used in these classes do not fit the strict definition of computer simulation, we include these examples to illustrate the benefits of using simulation in theoretical courses.

In our "Theory of Computation" course, we have used simulators of many of the theoretical machine models studied during the course, especially Turing machines and finite automata. See both [E] and [LW95,pp.95-104]. Using one of the various Turing machine simulators on the web, students enter descriptions of Turing machines into the simulator and test the machines on a variety of input values. Students must determine the output by hand before running the simulators. As an advanced assignment, students design and implement their own simulator, perhaps of a simpler model such as a deterministic finite automata. Of course, the use of simulations is not intended to displace theory, but rather to supplement it.

Similarly, our class on the "Design and Analysis of Algorithms" is taught largely as a theoretical course with written homework assignments. It is not a programming class, but focuses instead on the formal development, analysis, and proofs of algorithms. However, we have found it useful during the semester to assign a single programming project where the students implement, test, and report on one of the algorithms discussed during the semester. They have the choice of implementing competing algorithms for the same problem and performing an empirical analysis, or of writing an algorithm visualizer that allows the user to step through an algorithm with some graphical display of the algorithm execution. For example, one very successful student project for this course involved the comparison of various collision-resolution schemes for direct-address hashing.

Admittedly, neither of these assignments fall under the strict definition of a simulation. They are not simulating real-world systems, but simply trying to simulate how a computational process might work on real-world data through the random generation of test data. However, both assignments help teach some of the same principles as are used in more traditional simulations. In the first case, where students implement competing algorithms for the same problem, they have the challenge of generating realistic data set that simulates how the algorithm might run in a natural environment for some application. In the case of algorithm animations, students create a visualization of what might be a complex system. In both cases, we find that the assignment increases student understanding in a manner similar to other simulation examples described in this paper. In particular, the assignments give the students a deeper appreciation both for efficiency issues and for implementation details.

3.4 Computer Science for Non-Computer Scientists

At the opposite extreme from the courses described in the previous section is our computing course for non-majors. Since students enter this course with no programming experience, we cannot expect them to design and code their own simulations. Students are limited to experimenting with existing simulations. However, we have found that student understanding of important concepts in the course is aided by having them explore complex systems using these simulations, especially simulations that allow them to proceed through the system a step at a time.

Several simulations have been used throughout in this course. For example, early in the semester we use software that allows students to design and simulate simple circuits using Boolean gates (AND, OR, and NOT) [LW95,pp.47-54]. When we discuss computer systems organization, students run simulations of a generic von Neumann machine to get a better sense of the many lower level operations involved in performing even simple computational tasks [LW95,pp.55-68]. When we cover computers and society, students are exposed to the use of computer simulations to model real world phenomenon. Students then get the opportunity to try some of these simulations, such as those based on population growth models.

3.5 Simulation as the Solution Itself

The preceding examples illustrate how building and using simulations can help elucidate the algorithmic solution to a problem or the fundamental operations of a system being simulated. In many situations, however, a simulation itself can be the actual solution. Some problems are so complex that computing an analytic solution is intractable. Such problems can often be addressed using a class of approaches based on stochastic simulation. Stochastic simulation involves random sampling of the possible outcome states of a non-deterministic system in such a way that the statistics describing the resulting distribution of states provide a reasonable approximation of the solution to the problem.

In our upper-level "Artificial Intelligence" course, where we discuss problems of reasoning under uncertainty, we have taught how simulation can be the solution itself. A standard task in reasoning under uncertainty involves computing the posterior probability distribution over a set of unknown variables when given specific values for a set of known variables. This task, often referred to as inference, is NP-hard. An effective approximation approach based on stochastic simulation involves generating multiple outcome states for the unknown variables according to the probabilistic relationships induced by the values of the known variables. As the number of samples increases, the percentage of time that each variable takes on a given value converges to the actual posterior probability that the variable will have that value.

When we present this material to students, our approach is to first consider analytic approaches based on finding exact solutions. When it becomes clear that such methods can be infeasible even for relatively small problems, we move on to possible alternatives, including stochastic simulation. This culminates with a substantial programming project in which students implement a basic system for reasoning under uncertainty using stochastic simulation. Besides implementing the system, students also perform experiments to observe and measure the rate

of convergence as the number of trials in a simulation increases. (See the paper website for a fuller description of this assignment.)

4. CONCLUSION

We presented an argument for the use of simulations as an effective learning tool across the liberal arts undergraduate computer science curriculum. Part of our argument is captured in a comment by Miller and Boxer [MB00,p.123]: "One of the major focuses of computational science is on the knowledge and techniques required to perform computer simulation. The importance of simulation can be found in 'grand challenge' problems in areas such as structural biology, materials science, high-energy physics, economics, fluid dynamics, and global climate change, to name a few."

We described ways to effectively integrate simulation into the curriculum, providing several examples from five different courses, all widely taught at undergraduate institutions with computer science majors. The examples can be implemented with minimal effort and few resources. We hope that our experiences will benefit others who wish to incorporate computer simulations throughout the curriculum.

A copy of this paper with links to some sample assignment and other resources may be found at the first author's website at <http://www.middlebury.edu/~dickerso/simulation/>

REFERENCES

- [AU] Applets for Neural Networks and Artificial Life, "<http://www.aist.go.jp/NIBH/~b0616/Lab/Links.html>"
- [CNE] A Virtual Memory simulation (part of the CNE workbench), "<http://www.cne.gmu.edu/workbenches/vmsim/vmsim.html>"
- [CPA93] W.A. Christopher, S.J. Procter, and T.E. Anderson, "The Nachos Instructional Operating System." Proceedings of the 1993 Winter USENIX Conference, January 1993, pp. 479-488.
- [D90] H. Deitel, An Introduction to Operating Systems, 2nd edition, Addison-Wesley, 1990.
- [D99] Allen B. Downey, "Teaching Experimental Design in an Operating Systems Class." Proceedings of the Thirtieth SIGSCE Technical Symposium on Computer Science Education, March 1999, pp. 316-320.
- [E] Andreas Ehrencrona's simulations of Turing machines and other models, "<http://cgi.student.nada.kth.se/cgi-bin/d95-ae/umeng>"
- [H] Java Demonstrations of Neural Net Concepts, neuron.eng.wayne.edu/software.html
- [IR92] A Neural Network Simulation Project, The Journal of Artificial Intelligence in Education, Vol 3, No 1,1992.

- [IR93] Neural Networks, The Journal of Undergraduate Mathematics and its Applications, Spring 1993.
- [LW95] K.Lambert and T.Whaley, An Invitation to Computer Science: Laboratory Manual, West Publishing, 1995, Chapters 4,5,6,10.
- [LW99] K.Lambert and T.Whaley, An Invitation to Computer Science, 2nd edition, PWS Publishing, 1999.
- [MB00] R.Miller and L.Boxer, Algorithms Sequential and Parallel, Prentice Hall, 2000.
- [MH] Java Demonstrations of Neural Net Concepts, "<http://neuron.eng.wayne.edu/software.html>"
- [SGG00] A.Silberschatz, P.Galvin, and G.Gagne, Applied Operating System Concepts, John Wiley and Sons, 2000.
- [SWG] BrainWav Neural Network Simulator, "<http://www2.psy.uq.edu.au/~brainwav/>"
- [W99] M. Weiss, Data Structures and Algorithm Analysis in Java, Addison-Wesley, 1999.