Complex Adaptive Systems

AN INTRODUCTION TO COMPUTATIONAL MODELS OF SOCIAL LIFE

John H. Miller and Scott E. Page

PRINCETON UNIVERSITY PRESS
PRINCETON AND OXFORD

organisms form firms, and firms form nations. What underlies this apparent order? When will organizations emerge or dissipate?

The question of organization touches on many of the previous themes. Organizations can emerge from a variety of substrates. They tend to persist, even though their constituent parts do not. They exist in a state that allows them to be productive without being dissipative.

A.19 What Are the Origins of Social Life?

The origin of life question has played a central role in the biological sciences. Alas, the origin of social life has had much less attention. Such questions lie at the heart of understanding our world. How do we recognize social life? What are the minimal requirements for it to arise? What are the deep, common elements in social systems that transcend time and agents? Is social life inevitable?

Various research efforts in complex systems have shown how key social features, like cooperation or communication, can emerge. Yet, even these models tend to rely on some previously defined atomic structures. For example, agents are assumed to have strategic frameworks or are endowed with the ability to send and receive communication tokens. Is it possible to unwind these models further, allowing even more to emerge? Ultimately, can we realize *in silico* a dream similar to Darwin's, where starting from so simple a beginning we see endless social forms, most beautiful and most wonderful, arise?

APPENDIX B

Practices for Computational Modeling

Yet the question of its *modus operandi* is still undetermined. Nothing has been written on this topic which can be considered as decisive—and accordingly we find every where men of mechanical genius, of great general acuteness, and discriminative understanding, who make no scruple in pronouncing the Automaton a pure machine, unconnected with human agency in its movements, and consequently, beyond all comparison, the most astonishing of the inventions of mankind. And such it would undoubtedly be, were they right in their supposition.

—*Edgar Allan Poe, Southern Literary Messenger*

In 1769 Baron Wolfgang von Kempelen created an "automaton" chess player. The device consisted of an artificial "Turk" seated behind a cabinet full of mechanical marvels. The Turk would move the pieces with its mechanical arm and even nod its head disapprovingly when the opponent made an illegal move (apparently a feature that was well utilized when it played Napoleon Bonaparte). The automaton was a sensation around the world.

The most important feature of the automaton was indeed its clever mechanical design—especially the feature that allowed a skilled human chess player to remain concealed to the audience. While consciously hidden assistants are not a scientific concern, the area of computational modeling is new enough that adhering to a set of best practices will do much to advance the acceptance and productivity of this approach. Like all scientific fields, the biggest danger with computational models is not outright fraud (a relatively rare event across all fields of science), but the unconscious acceptance of faulty results. It is easy for scientists to fool themselves, and all scientists must be their own harshest critics and do everything they can to maintain and expose the full integrity of their work.

In this appendix, we begin to outline some practices that should help promote quality science in computational modeling. Note that most of these practices have very little to do with computational methods per se, and that is by design—good computational modeling is more about modeling than computation.

B.1 Keep the Model Simple

Making sure that your model has just enough of the right elements and no more is the most fundamental practice for any kind of modeling, and computational work is no exception. With a few well-placed lines of a pen, an artist can represent a complex world in a very simple and understandable way. Scientific modelers must aim for a similar level of simplicity and clarity. Modeling is like stone carving: the art is in removing what you do not need.

It is often easy to recognize a simple, well-formulated model after the fact, as such models have a strong intuitive appeal. Getting to this point usually requires a combination of skill, practice, effort, revision, and art—a mix of abilities that is difficult to teach directly. A first step in developing such skills is appreciating the elegance of seminal models from a variety of fields.[1] Great artists study the masters; so too must great modelers.

Once simplicity is achieved in the model, many of the practices suggested here become much easier to attain.

There is a necessary tension between pursuing simplicity and exploiting the ability of computational models to interconnect key parts of the world. For example, to understand the dynamics of an epidemic, we need to understand how diseases are transmitted between people; how people come in contact with one another via transportation, work, and home environments; and how people react once they become ill or hear of illness in others. As we incorporate each of these elements into the model, it becomes more complicated—though avoiding this type of complication is difficult if we truly want to understand this type of problem. We can admit such complications into our models if we are careful to keep each of the constituent parts simple. Good models strip phenomena down to their essentials, yet retain sufficient complication to produce the needed insights.

B.2 Focus on the Science, Not the Computer

The most important feature of a computational model is the model, not the computer. Thus, computational models must be justified solely by the model and not on their clever (or fast, current, etc.) algorithm (or, for

[1]In economics, works like Akerlof (1970), Hotelling (1929), Schelling (1978), and Stigler (1961) offer a few examples.

that matter, hardware, software, etc.).[2] New technological developments may enhance our ability to explore better existing models or create new ones, but without a solid model underlying the work, such improvements are meaningless. Thus, being able to take a simple model that produces high-quality science and scaling it up by, say, adding more agents or performing some additional experimental conditions may be a good application of new technological developments. Using lovely real-time graphical displays to illustrate your results can also be productive, as long as there is good science underlying the results.

B.3 The Old Computer Test

One way to promote both of these practices is to write a computational model as if it must run on a very slow and limited computer. Writing a program under such a constraint will often force the modeler to simplify the work in productive ways. (Of course, once the model is finalized, you can always run it on the best-quality machine available.) Some of the best examples of "computational" models, such as Schelling's Segregation model, do not even require a computer.

B.4 Avoid Black Boxes

Each part of a model must be as clear and accessible as possible. To achieve this end, modelers should always default to simple, straightforward mechanisms for each element and avoid having parts of the computation rely on "black boxes" that are composed of massive amounts of code rife with assumptions and choices that are essentially hidden from any potential consumers of the model. When models implement such code, it is difficult to determine whether the outcome that is observed is being driven by some quirk hidden within the black box or by a more fundamental law of nature.[3]

Of course, what is a black box to one person might be a fundamental component to someone else. For example, one approach pursued in the cognitive science and artificial-intelligence communities is the creation of relatively large and complicated computational models of cognitive processes. These models are composed of various components like

[2]Indeed, it is not clear to us why speakers feel a need to discuss issues surrounding hardware or software choices (or, even worse, display computer code) during scientific seminars. Such discussions are best left for other contexts.

[3]The fundamental issue here is captured in a cartoon by Sidney Harris that has two scientists discussing sets of equations on a blackboard divided by the words "then a miracle occurs." One says to the other, "I think you should be more explicit here in step two."

memory, categorization, inference, and problem solving, each of which is implemented through an often complicated set of assumptions and specializations for a given problem domain. If we accept these efforts as well-formulated and understood models of adaptive behavior, then one could use them as the basis for creating social systems of adaptive agents, notwithstanding the black box nature of the computation involved. Nonetheless, wherever possible, computational modelers should follow the dictates of Ockham's razor and favor the adoption of simple structures. If a simple, low-level adaptive algorithm is able to capture the behavior of a system adequately, then it should be preferred over more complicated mechanisms.

B.5 Nest Your Models

There is often an opportunity to create models where special cases of the assumptions result in well-known (and hopefully understood) examples. Nesting standard models within computational ones is usually a very natural process, as computational models are often employed to extend standard models in interesting, but previously inaccessible, directions. Once nested, it is easy to compare the model's predictions in the special cases with known results, and then to show how the model verifies known results and observations or, if not, to explain why there is a divergence.

In the course of developing and fully understanding a computational model, there are usually plenty of opportunities to include "sanity checks," that is, special conditions under which the behavior of the model is known a priori. Almost any component of a computational model can be turned off and replaced by a simple alternative. For example, if the model relies on adaptive agents, complicated objective functions can be temporarily replaced with simple ones to demonstrate that the agents can find the optima in such a case. These types of experiments are a good way to check the basic foundations of the model and should be a routine part of creating any computational model.

B.6 Have Tunable Dials

One way to nest models is to rely on "tunable" dials for controlling key assumptions. One interesting dial for economic problems would be a way to tune agent rationality. Ideally, this dial would allow us to take a fully optimizing agent and slowly degrade its behavior toward less and less rationality. Such a dial would provide fodder for a variety of interesting research questions: Under what conditions does the dial matter? Does the

behavior of the system undergo slow changes as we turn the dial, or does the system experience rapid phase transitions where small movements of the dial result in abrupt changes in the system? What effect does the very first movement of the dial away from optimality have on the system?

Finding a simple way to represent a rationality dial would be a great advance, and at the moment even the existence of such a dial is an open question. That being said, there are areas where dials can be easily employed; for example, we can vary the degree of look-ahead used by an agent, the amount of processing, the number of interactions, and so on.

B.7 Construct Flexible Frameworks

Often it is possible to create computational models with simple, flexible frameworks that "get filled in" by the computation. For example, genetic algorithms rely on representations of potential solutions that are fairly general, and it is the evolution of the system that fills in the details. A well-designed framework puts very few a priori constraints on the model, and thus the outcome is rich in possibilities. Such frameworks provide enough flexibility so that the model can explore areas that were not fully anticipated by the researcher. It is not uncommon for, say, a genetic algorithm to yield an answer that initially appears wrong, only to find on closer examination that the algorithm discovered a perfectly sensible, but wholly unanticipated, result.

Note that even the simplest of frameworks can result in outcomes that are difficult to understand. For example, consider a simulated neural network composed of a group of artificial neurons controlled by a straightforward adaptive algorithm. Such a network provides a very flexible framework that is constructed of extremely simple components. However, understanding the outcome of this system can be very difficult, as it produces a nonlinear function that is often difficult to unravel. Thus, in addition to simplicity and flexibility, we need models that are transparent in their operation and outcomes.

B.8 Create Multiple Implementations

Even with an apparently well-defined model, there may be a variety of choices that must be made before it can be fully implemented. Most of the time, such choices matter little to the outcome of the model, but on occasion they may lead to important differences (and hence insights). Therefore, it is always useful to implement key modeling choices in a variety of ways.

One useful way to facilitate the creation of multiple implementations of a model is to pose it in relatively general terms to different modelers and have each of them separately define an implementation. By comparing these various renditions, one can begin to identify the key issues that need to be addressed. An alternative technique is to have at least two groups separately code the model (preferably using two different computer languages). Not only does this process help clarify the important issues, but it also results in two versions of the model that can be run in parallel to confirm results and gain insights (Axtell et al., 1996).

Even within a single, well-defined computer program, there are often opportunities to implement key features of a model in different ways. For example, assumptions about, say, probability distributions, agent matching, timing, and so on, can be manipulated easily. Computational modelers should always try to have multiple implementations of any features of the model that will be closely associated with scientific claims. Thus, it is often useful to implement a variety of adaptive algorithms to make sure that any particular claims of the impact of adaptive behavior are not tied to some quirk in a specific algorithm.

B.9 Check the Parameters

Along with implementing various parts of the model in different ways, computational models should always be subject to a sensitivity analysis of key parameters. This can either be done manually or through the use of automated techniques, like Active Nonlinear Tests (ANTs) (Miller, 1998). Under ANTs, researchers specify their main assumptions (including parameter values, distribution choices, and so on) and then use an automated procedure that is designed to "break" the model by searching over acceptable variations of the assumptions. ANTs can be used to identify a model's inherent limits and key driving assumptions.

B.10 Document Code

Most of the coding effort devoted to a computational model, like most programming, is devoted to documenting the model and getting data into and out of the system, as opposed to specifying core behavior. Care and time spent in this domain are necessary for ensuring that the results can be fully analyzed and easily tied to the exact conditions that produced the outcome. Software tools, such as Concurrent Versions Systems (CVS), can assist in the tracking of the various revisions made to the program during a research project.

B.11 Know the Source of Random Numbers

Models that rely on random numbers must ensure that the pseudo-random-number generators used are adequate to the task. It is easy for the uninitiated to make serious errors in this realm; for example, using only the lower-ordered bits coming from a pseudo-random-number generator can be problematic as in some algorithms these bits alternate in value. Users should always be aware of the algorithmic details of their generators and take whatever appropriate actions (like shuffling the resulting values) are needed to avoid problems.

B.12 Beware of Debugging Bias

In all scientific work, there are natural human biases that often confound good practice. All scientists have some expectations (and dreams) about the results they will find, and when these are met, it is natural to accept the findings. If you look at the historical estimates of, say, the speed of light, you will find that they do not randomly vary centered on the currently accepted value, but rather they converge on this value biased by previously published estimates (Henrion and Fischhoff, 1986).

A similar bias can happen in computational models with respect to debugging. When modelers observe results that are not as expected, they are likely to spend a lot of effort debugging their code. When their expectations are met, little such effort is expended. Thus there is an inherent tendency in researchers that could, if sufficient caution is not exercised, bias results toward prior expectations.

B.13 Write Good Code

Various software development practices can improve the code underlying a computational model. Some key lessons from software development that are applicable here include the following. First, there is a trade-off among scope (the number and types of features included in the software), quality (the ability of the code to be understood, modified, and executed without error), cost (both in terms of development and maintenance), and delivery date. The trade-off is such that improving one of these areas causes one of the other areas to degrade (or, as it is sometimes stated: scope, quality, cost, time, choose three). Second, the majority of time on a well-run software development project is not spent in coding, but rather it is devoted to developing a solid, well-articulated design for the software. The third important lesson from software development is that the cost of correcting errors dramatically increases with the amount of time that elapses before the error is identified and fixed. Thus, an error in the design stage costs ten times more to correct in the coding stage and a hundred times more to fix after the program is in use. Finally, there are dramatic differences in the practices of professional versus amateur software coders (even among the professional coders, the productivity can differ by a factor of ten or more), and all computational modelers should make sure that their skills are sufficiently advanced for the task at hand. McConnell (1993) provides a nice overview of the basics for writing high-quality, extendable, easily communicated code.

Good software development is extremely difficult, and given the complexity of the underlying task, newness of the area, and rapid changes in technology, it is not too surprising that ideas about how best to accomplish this task are in constant flux. Much like business management fads, every few years new methods of software development are proposed that promise to solve all of the existing problems.

While it is doubtful that such a silver bullet will emerge anytime soon, there may be parts of certain development methodologies that are useful for computational modelers. For example, the Extreme Programming movement incorporates a number of practices that closely match the needs of scientific modelers, six of which we list here: key features of the program are added in the order of most value; overall planning is limited to current needs; each component of the program is associated with a comprehensive unit test that ensures that it is implemented appropriately; an effort is made to improve the code (refactor) whenever possible; development time estimates are based on recent experience; and new versions of the program are released on very short time scales.

Scientific researchers undertaking the development of ambitious software projects should consider adopting a real software development "process" (versus the usual "code and fix" methodology). Although real processes do introduce some additional overhead costs into the project (especially initially), the higher-quality software that is produced by such methods allows these overhead costs to be quickly recovered.

B.14 Avoid False Precision

When reporting computational work, be aware that there is an appropriate level of precision that can be associated with the results. This

concept holds at all levels of the model. At the lowest level, make sure that numeric results are appropriately reported. At a higher level, given the inherent nature of all modeling, beware of making too much of subtle differences. The important results from a model are typically not very subtle and tend to be obvious, both qualitatively and quantitatively, across a variety of conditions.

B.15 Distribute Your Code

Computational models should be easily accessible to other researchers. Models that follow the practices outlined here are often more accessible to others, because they entail simple, high-quality code, that is well written and documented, and thus easily understood and implemented by others. Code for published models should be easily available to others so that they can replicate the results. Some researchers are also willing to distribute code prior to publication (perhaps with some stipulations on its use).

B.16 Keep a Lab Notebook

Once developed, computational models are often deployed in a way that is very similar to laboratory experiments in biology or chemistry. In these areas, there is a long tradition of keeping a laboratory notebook that details particular hypotheses, the experiments designed to test the hypotheses, and the outcomes and conclusions of these tests. Such notebooks not only provide an important historical record of the work, but they may also give the researcher additional insights and understanding about the system. Such research notebooks may have a similar value for computational experiments.

B.17 Prove Your Results

Whenever possible, computational results should be clarified and verified as thoroughly as possible. Computational results should always be placed in their appropriate statistical context. More important, researchers should strive to eliminate as many alternative hypotheses as possible through well-designed experiments. Indeed, a better name for this section might be "disprove your results"—researchers should always try to disprove their key results to the best of their ability.

Researchers should try to use alternative means to solidify key findings. Thus, it is often possible to apply more traditional tools to parts of the analysis by, say, proving a theorem about a particular phenomena observed in the model. Although such techniques may not work on the whole model, they often work on particular parts or special cases. Whenever possible, the analysis of computational models should be enhanced with complementary modeling efforts.

B.18 Reward the Right Things

Like any branch of science, one needs to reward the right accomplishments. While it may be true that lovely graphics, advanced coding techniques, frontier hardware, and so on may enhance computational models, ultimately it is the resulting science that must be judged. Scientific judgments in this area should focus not on the computer per se, but on the quality and simplicity of the model, the cleverness of the experimental designs, and the new insights gained by the effort.

Bibliography

Abbott, Edwin A. 1884. *Flatland: A Romance of Many Dimensions*. New York: Dover, 1952.

Akerlof, George A. 1970. "The Market for 'Lemons': Quality Uncertainty and the Market Mechanism." *Quarterly Journal of Economics* 84:488–500.

Albin, Peter S. 1975. *The Analysis of Complex Socioeconomic Systems*. Lexington, Mass.: Lexington Books.

Anderson, Phil W. 1972. "More Is Different." *Science* 177:393–96.

Andreoni, James, and John H. Miller. 1991. "Can Evolutionary Dynamics Explain Free Riding in Experiments?" *Economics Letters* 36:9–15.

———. 1995. "Auctions with Adaptive Artificial Agents." *Journal of Games and Economic Behavior* 10:39–64.

Arrow, Kenneth J. 1951. *Social Choice and Individual Values*. New Haven: Yale University Press.

Arthur, W. Brian. 1994. "Inductive Reasoning and Bounded Rationality." *American Economic Review Papers and Proceedings* 84:406–11.

Arthur, W. Brian, John H. Holland, Blake LeBaron, Richard G. Palmer, and Paul J. Tayler. 1997. "Asset Pricing under Endogenous Expectations in an Artificial Stock Market." In *The Economy as an Evolving Complex System II*, edited by W. Brian Arthur, Steven Durlauf, and David Lane, 15–44 Redwood City, Calif.: Addison-Wesley.

Axelrod, Robert. 1984. *The Evolution of Cooperation*. New York: Basic Books.

Axtell, Robert. 2001. "Zipf Distribution of U.S. Firm Sizes." *Science* 293:1818–20.

Axtell, Robert, Robert Axelrod, Josh Epstein, and Michael Cohen. 1996. "Aligning Simulation Models: A Case Study and Results." *Computational and Mathematical Organization Theory* 1:123–41.

Bak, Per. 1996. *How Nature Works*. New York: Springer-Verlag.

Bednar, Jenna, and Scott E. Page. 2006. "Can Game(s) Theory Explain Culture? The Emergence of Cultural Behavior within Multiple Games." *Rationality and Society* 18:345–73.

Camerer, Colin. 2003. *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton, N.J.: Princeton University Press.

Camerer, Colin, and Teck-Hua Ho. 1999. "Experience-Weighted Attraction Learning in Normal Form Games." *Econometrica* 67:837–74.

Challet, D., and Y. C. Zhang. 1997. "Emergence of Cooperation and Organization in an Evolutionary Game." *PhysicaA* 246:407–18.

———. 1998. "On the Minority Game: Analytical and Numerical Studies." *PhysicaA* 256:514–32.

Coates, Robert M. 1956. "The Law." In *The World of Mathematics*, Vol. 4, edited by James R. Newman, 2268–71. New York: Simon and Schuster.