Replicating the Classic Sugarscape in MASON*

Tony Bigbee, Claudio Cioffi-Revilla and Sean Luke†
Center for Social Complexity and Evolutionary Computation Lab
Krasnow Institute for Advanced Study
George Mason University, Fairfax, VA 22030 U.S.A.
E-mail: ccioffi@gmu.edu

April 26, 2006

Abstract

Replication is essential in science and agent-based computational social science is no exception. We present results from a replication of the Sugarscape model (Eptstein and Axtell, 1996) as initially presented in *Growing Artificial Societies* (*GAS*). Sugarscape is a classic agent-based model and contemporary simulation toolkits usually only have a partial replication consisting of a few core model rules without documenting simulation outcomes; code supplied with Repast, Swarm, and NetLogo, for example implement very few of the rules in Sugarscape. By contrast, we demonstrate a detailed replication of Sugarscape in the MASON (Multi-Agent Simulator of Neighborhoods and Networks) environment, and describe various outcomes such migration waves. For major outcomes documenteed in GAS, we describe the degree to which we replicated those results and conclude by outlining major challenges in replication activities and the young field of agent-based modeling in general.

Keywords

Sugarscape, replication, docking, MASON, agent-based modeling, computational social science

1 Introduction

Replication is a hallmark of all science and agent-based computational social science is no exception. Multiple independent replications are especially needed in this emerging field, particularly

*Prepared for the Third International Conference on Complexity, Groupement de Recherche en Economie Quantitative d'Aix-Marseille (GREQUAM), Aix-en-Provence, France, 17–20 May, 2006. Earlier versions of this study (Bigbee et al. 2005a, 2005b) were presented at the 2005 annual conferences of the European Social Simulation Association (ESSA) and the Pacific Asian Association for Agent-based Approach in Social Systems Sciences (PAAA).

†The authors thank Hiroshi Deguchi, Nigel Gilbert, David Hales, Scott Moss, Juliette Rouchier, Keiki Takadama, Takao Terano and Klaus Troitzsch for the comments on earlier versions of this paper, and Robert Axtell and Joshua Epstein for extensive comments on Bigbee (2005). The MASON Project is funded by the Center for Social Complexity and the Evolutionary Computation Lab of George Mason University, and by grants from the National Science Foundation, DARPA, and the Air Force Office of Scientific Research.

since reported claims and findings may depend on algorithmic implementation and software implementation choices in general. We present results from a replication of the classic Sugarscape model (Eptstein and Axtell, 1996) as originally described in *Growing Artificial Societies* (*GAS*).

Sugarscape is one of the first-generation agent-based models and contemporary simulation toolkits usually only have a very simple replication of a few core rules. Thus, there is scant evidence of significant replication of the Sugarscape model, specifically in terms of simulation *rules* and *outcomes* when these are viewed as target data. Demonstrations supplied in the simulation environments of Repast, Swarm, and NetLogo implement only a minority of the rules in Sugarscape. For example, the standard Repast distribution only implements the rules of **Growback**, **Movement**, and **Replacement**, omitting all the others included in our study. This is because Sugarscape implementations in these toolkits are intended only as basic demonstrations of how well-known social models might be implemented, rather than achieving scientific replication.

A major goal in our investigation included assessing the maturity of the new MASON toolkit, testing it through a detailed replication of the classic Sugarscape model. As detailed in the next sections, the MASON (Multi-Agent Simulator of Neighborhoods and Networks) system

> *is a fast discrete-event multiagent simulation library core in Java, designed to be the foundation for large custom-purpose Java simulations, and also to provide more than enough functionality for many lightweight simulation needs* (Luke et al., 2005: 517).

Since MASON was designed to be a tool for both social science simulations and artificial intelligence research, among other uses, replication of one of the most recognized agent-based social science models would demonstrate its maturity and usability for its intended purpose.[1]

Replication of well-known models is also important given the relative novelty of agent-based modeling in social science (Cioffi-Revilla 2002; Edmonds & Hales 2003). Better tools and techniques for facilitating the advancement of computational social science goals are also desirable outcomes of a replication exercise.

2 Computational Methodology: The Object-Oriented (OO) Approach

The Sugarscape framework, as originally developed by Epstein and Axtell (1996) for agent-based modeling and simulation in social science, adopts the following scheme:

Agents The "people" of artificial societies, with internal *states*, behavioral *rules* (see below), and *interactions* with the environment. Some states, such as economic preferences, wealth, cultural identity, and health can change as agents move and interact with the environment and with other agents.

Environment The space, landscape or medium distinct from agents, "*on* which the agents operate and *with* which they interact" (p. 5). There are landscapes of renewable resources and more abstract structures, such as communication networks. The space is a two-dimensional lattice called a "sugarscape", implemented as a *torus* or *toroidal surface* (*GAS*, p. 22) .

Rules The behavioral drivers for agents and sites of the environment. Sites/cells of the environment can be coupled via rules. In general, there are agent-environment-rules, environment-environment rules, and agent-agent rules that govern interactions such as mating, combat, or trade.

This is now common in most agent-based social simulation models . Following Simon's principles of social complexity based on bounded rationality, Epstein and Axtell (1996: 6) also stated the following defining feature of the Sugarscape model:

[1] Other MASON replications of classical models (e.g., Schelling's Segregation model, Heatbugs, Conway's Life, and others) are found at the MASON website: http://cs.gmu.edu/∼eclab/projects/mason/

the fundamental social structures and group behaviors emerge from the interaction of individual agents operating on artificial environments under rules that place only bounded demands on each agents information and computational capacity.

Epstein and Axtell (1996,5) developed Sugarscape is written in *object-oriented* (OO) code (Epstein and Axtell 1996, 5), and the model consists of:

Variables Instance member *variables* representing the internal states or attributes of each **Agent**, such as the agents' attributes of sex, age, or wealth;

Behaviors *Operations* or *Methods* for agents' rules of behavior (such as eating, trading, combating); and

Encapsulation The hallmark of the OO modeling *paradigm*, in the sense of Kuhn (1970), but also as a *progressive research programme*, in the sense of Lakatos (1973). *Encapsulation* means the computational combination or binding of *both* agent internal states, variables or attributes *and* the corresponding behaviors, methods or rules. Encapsulation implements the close relationship observed between social attributes and the way they change, thereby facilitating the agent-based model construction of social systems.

Details on object techniques and implementations in Sugarscape are generally omitted from *GAS*, except for Appendix A. Polymorphism is not discussed and inheritance was considered but was not used due to "efficiency considerations [...]. In total, each agent has over 100 methods" (*GAS*, p. 180). By comparison, the `Agent` class in MASON Sugarscape has approximately 32 methods, although only 75–80 percent of all Sugarscape rules were implemented. MASON Sugarscape generally does not employ inheritance nor polymorphism, except as first-level inheritance from selected MASON. Concretely, there is one class representing all types of agents rather than many different classes.

3 MASON Implementation

Although the MASON distribution includes a variety of graphics, utility, and other supporting infrastructure and examples, discussion of MASON in this paper focuses mostly on timing and scheduling. Controlling and determining which behaviors are executed, and when, is a critical aspect of simulation. In addition to overt requirements—such as an agent having to scan its neighborhood and move to a site before it can harvest resources—synchronous and asynchronous interactions occur between entities. A necessary, but not sufficient, condition is that actions be executed in time exactly as desired—this requires a scheduling mechanism providing precise contro.

3.1 Timing and Scheduling

The MASON Version 8 `Schedule` class sequences and executes objects that implement the `Steppable` interface. A primary construct in `Schedule` is order, a collection of executable objects that enable a deterministic sequence of execution. For each time step, all order zero objects are executed first, followed by order one, order two, and so forth. Implementation of rules in a specific order is a key aspect of this replication, and the final order for rules in the general model was:

(0) **Environment** rules

(1) **Agent** rules

(2) open/unused

(3) **Season** rules

(4) statistics and logging

(5) open/unused

(6) text histogram chart,

where 'open' indicates the order is not used and is simply a result of evolving the model and experimenting with MASON. Orders for **Agent** rules and **Environment** rules are specified in the primary model class as constants, with all simulations results described in the Simulation Infrastructure section. These constants could easily be redesigned as parameters in the primary configuration file for more flexible experimentation. Finally, the text histogram chart class uses order 6; this diagnostic tool is further described in the Rules section below (3.3).
Although **Seasons** used the `MultiStep` class, an alternative to `MultiStep` is to use a `Schedule.setRepeating` method that has an interval. This has a performance advantage over `Multistep` when orders have more than one object.

Within each order, the `Steppable` entities are executed once in a random sequence, and the sequence is *randomized every time step*. The other critical timing mechanism used in this implementation is that agents do not determine which of their rules are executed, nor in what sequence. Instead, a new class named `RulesSequence` was created to wrap `Sequence`, a MASON class for holding a static sequence of `Steppables`. This static sequence contains a collection of rules, each of which is invoked in turn. A benefit of using `Sequence` is that the `step()` method in the agents or environment objects are small and simple, primarily calling `step()` for its `RuleSequence`. The `RuleSequence` instance, in turn, calls each rule in the original order specified when the rules were added during initialization. Sequences are used instead of adding the rule objects directly to the schedule as direct schedule usage has order n entities $\times$ m rules time complexity, and `Sequence` only has order m rules time complexity.

To provide for a flexible means of experimentation, a primary configuration file specifies rule execution order during runtime. Using the reflection facilities of the Java language, initialization code translates text names for rules classes into appropriate object instantiations. Abbreviations are also specified in the configuration file, providing a quick way to specify desired rule sequences without having to recompile source code. The example below shows an extract from a configuration file specifying agent rule classes, abbreviations for each, and execution order for the agent rules:

1. `agent_rule1` = **Movement, M**
2. `agent_rule2` = **Biological, B**
3. `agent_rule3` = **Reproduction, S**
4. `agent_rule4` = **Culture, K**
5. `agent_rule5` = **Trade, T**
6. `agent_rule6` = **Pollution, P**
7. #rules will be executed in the order listed
8. `agent_sequence_rules` = **M, T, B**

While most rules in Sugarscape involve sequential execution of behavior for each agent or site, there are rules that involve cellular automata (CA) type synchronous treatment of all instances of an entity – agents and resources sites (sugar, spice) are both represented as entities. **Pollution Diffusion** (**D**) is an example of an environment rule in which the states of all sites are updated synchronously and the rule does not invoke any MASON scheduling machinery.

UML sequence diagrams are useful for helping to understand interactions in time, but many agent-based models have discrete space aspects. As an example, interactions in space and time are illustrated in Figure 1 below. This diagram provides insight into the local effects of harvesting under two different situations: a) when sites grow back during the harvesting time step (Fig. 2, top); and b) sites do not grow back until one full time step *after* the time step during which harvest occurred (bottom). An agent is represented as a blue circle. At the end of time T, the agent has moved to a site, and harvested the sugar there. For this scenario, with **Growback** at every time step, the order in the schedule is agents are executed first, then sites. In the top row, the agent sees that the site with 3 units is the nearest and highest level sugar site

and moves there during T+1. Since the sites execute in a later order, the new site regenerates to 1 at the end of T+1. At the end of T+2, the agent has moved up one cell and harvests that site, which regenerates to 1. The agent could have gone back to its original time T site (but the upper site was randomly selected). Finally, at the end of T+3, the agent moves down again to the site with the highest level of sugar (2 units).

In the second row, the modified **Growback** rule is such that a site can not regenerate during a time step–accomplished by only one line of code. This is why at the end of each time step, occupied sites have level zero. One can see that the local effect with this rule change is that the agent, in the shown spatial configuration, will not backtrack and that sites where agents have been are one unit less than at the same time step in the top row, *even when not occupied by the agent*. At the end of T+3, the agent has moved out of view to a hypothetical site with 2 units (either up or right of its position in T+2)–it could not backtrack. The persistence and spatial impact of this effect is mitigated when population density is low—i.e., less competition for resources—and due to the random selection of equidistant sites having equal resources. The high density starting the agent block of Animation II-6 (see the Results section below) and the emergent wave phenomenon graphically illustrate the global, emergent effects of growback prohibition during a harvest period.

3.2 Randomization

The order in which behaviors occur, from micro-level agent behaviors such as which direction is scanned first, to the order in which entities are executed, can affect the qualitative outcomes of simulations. Epstein and Axtell explicitly discussed using randomization throughout the implementation of GAS, and during the development of MASON Sugarscape, the first author had to revise some code blocks several times due to insufficient attention to randomization requirements.

Because randomization can manifest itself in direct and indirect ways, we propose to characterize elements of randomness in four types:

`Type 1` - Initial distributions of entities' states, such as agents' genetic characteristics and initial locations.

`Type 2` - Dynamic state assignments from distributions such as the next direction to look in agent visual scan patterns or an agent life expectancy at birth.

`Type 3` - Order of execution among entities (such as agents), and between agents for asynchronous interactions. In both the classic and MASON models, agents are randomly selected for execution/behavior each time step, but the sequence of agent and environment execution is deterministic.

`Type 4` - Simulation outcomes that can be characterized by theoretical tools such as Markov fields. For example, Culture (K) eventually reaches equilibrium in one of three absorbing states (all blue, static levels of the two colors on the sugar mountains, or all red)

MASON Sugarscape depends entirely on uniform distributions generated by the MersenneTwisterFast class in a sister project – ECJ (Luke et. al., 200x). A search of the MASON Sugarscape source code indicates that there are 24 statements (calls to `random.next`) for Type 1 and Type 2 randomization. We are unaware of any systematic way to prove that requirements for randomness in pseudocode/rules specification is accomplished in source code implementation other than software unit testing approaches. One may argue that the the creation of the tests themselves is an act of creativity and judgement on the part of the developer and nothing is 'proven.'

Young (2006) analyzes several kinds of agent-based models by 'analyzing the long-run behavior of such systems using the theory of large deviations in Markov chains.' He asserts that simulations can help understand short and medium run dynamics of a system, but Markov chains analysis 'avoid[s] the hazards of drawing conclusions solely from simulations.' At the end of this paper, we propose that tools like this should be used by researchers to better understand simulation outcomes.

3.3 Primary Data Structures

The two primary data structures hold the two types of entities in the model: agents and site-resources. The original Sugarscape model contained agents and sites on a lattice—a discrete grid-like space. MASON has provisions for a variety of spatial data structures, including several types of grids, and each grid type has different computational performance characteristics. To hold agents, this model uses the `SparseGrid2D` class. To store resource sites—sugar and/or spice—the model uses the `ObjectGrid2D` class.

3.4 Sugarscape Rules

The table below describes the Sugarscape rules implemented from the classic model and whether the rule can be added, removed, or reordered simply by editing the `agent_rules_sequence` or `environment_rules_sequence` lines in the runtime configuration file. The Results section describes results from simulation runs using each rule and combinations of rules as described throughout *GAS*.

Table 1. Rules implemented from the Sugarscape model in the MASON environment.

Symbol	*Rule Name*	*SLOC*
$\mathbf{G}_{\alpha}$	Sugarscape growback	42
M	Agent movement	323
$\mathbf{R}_{[a,b]}$	Agent replacement	17
$\mathbf{S}_{\alpha\beta\gamma}$	Seasonal growback	16
$\mathbf{P}_{\pi,x}$	Pollution formation	18
$\mathbf{D}_{\alpha}$	Pollution diffusion	56
S	Agent mating	163
None	Agent cultural transmission	128
None	Group membership	128
K	Agent culture	128
T	Agent trade	177

Source: Based on Epstein & Axtell (1996).

Total source lines of code (SLOC) were approximately 3,500, with approximately 100 lines being optional diagnostic print out lines. Total source lines of code for the rules is 934, with the ratio of rules code to other code being approximately 1:3. The non-rules source lines of code involve model initializing, graphs and statistics, logging, instantiating agents, their rules, and sites, and simulation management infrastructure such as parameter sweeping. The total SLOC count for MASON version 8 itself, not including the supplied demonstration applications, is approximately 18,000. This includes many classes not used by MASON Sugarscape, including other portrayals and 3-D visualization infrastructure. SLOC statistics were generated by the SLOCCOUNT tool (Wheeler, 2005).

4 Results

In this section we present our overall results and highlight some of the more specific findings regarding the generative replication of Pareto's Law—one of the most important computational result in the original Sugarscape. Additional details may be found in Bigbee (2005) and the MASON website, including source files and other documentation. The overall results from this MASON-based replication are generally supportive of the main claim in *GAS*), as detailed below. Nonetheless, details of some departures or partial success are as informative.

4.1 Overall Findings

Table 2 presents an overall synthesis of the simulation outcomes and rule sets investigated as part of this replication. Qualitative and quantitative criteria are described in the table, such as whether aggregate statistics and shapes of graphs were matched. The overall level of replication achieved was rated as exact, general, or partial, pending on the judged concordance between original outcomes reported in *GAS* and our MASON simulation model of Sugarscape.

Table 2. Simulation outcomes and rule sets implemented in the MASON Sugarscape replication.

Outcome	*Rule Set*	*Replication Criteria*
Animation II-2, p. 29	($\{G_1\}$, $\{M\}$)	Hiving, peak clustering, terrace sticking
Figure II-5, p. 31	($\{G_1\}$, $\{M\}$)	Small positive slopes, equally spaced lines, visually estimated line
Animation II-3, p. 34	($\{G_1\}$, $\{M, R_{[60,100]}\}$)	Pareto distribution, maximum wealth bin
Animation II-4, p. 38	($\{G_1\}$, $\{M, R_{[60,100]}\}$)	Gini coefficient evolution
Animation II-6, p. 43	($\{G_1\}$, $\{M\}$)	Visual wave phenomenon
Animation II-7, p. 46	($\{S_{1,8,50}\}$, $\{M\}$)	Seasonal clustering
Animation II-8, p. 49	($\{G_1, D_1\}$, $\{M, P_{11}\}$)	Migration patterns
Figure III-1, p. 58	($\{G_1\}$, $\{M, S\}$)	Stable time series
Animation III-1, p. 58	($\{G_1\}$, $\{M, S\}$)	Approximately stationary age distribution
Figure III-2, p. 62	($\{G_1\}$, $\{M, S\}$)	Diverging Vision, Metabolism
Figure III-3, p. 64	($\{G_1\}$, $\{M, S\}$)	Small amplitude oscillations
Figure III-4, p. 65	($\{G_1\}$, $\{M, S\}$)	Large amplitude oscillations
Figure III-5, p. 66	($\{G_1\}$, $\{M, S\}$)	Severe population swings, extinction
Animation III-6, p. 75	($\{G_1\}$, $\{M, K\}$)	Homogeneous population
Figure III-8, p. 77	($\{G_1\}$, $\{M, K\}$)	Time series extremes, random group convergence
Animation IV-1, p. 100	($\{G_1\}$, $\{M\}$)	Peak hopping, small population
Figure IV-4, p. 110	($\{G_1\}$, $\{M, T\}$)	Significant trade volumes over time

The overall pattern of replication results is that the outcomes documented in *GAS* were generally replicated in our MASON 8 Sugarscape simulations.[2] The most successful replications occurred for outcomes that did not involve agent survival–i.e. those dependent on movement or welfare; successful outcomes included Culture, Pollution Diffusion, Seasons, and other spatial phenomena.

4.2 Generative Proof of Pareto's Law

An example of general replication achieved is Animation II-3 (*GAS*, p. 34), which is replicated by Figure 2 below. The significance of this replication is that Animation II-3 in *GAS* provided the *first* generative demonstration or computational proof of Pareto's Law.

Definition 1 (Pareto's Law of Individual Wealth) *The number N_w of individuals with wealth $w \in W$ or greater is inversely proportional to w, or*

$$N_w \propto w^{-\alpha}, \tag{1}$$

for some $\alpha > 0$, so

$$\log N_w = \log k - \alpha \log w, \tag{2}$$

where k is a proportionality constant and α is called the Pareto index.

Note that (i) k and α are scale and shape parameters, respectively; (ii) α is an exponent (or power, hence the complexity-theoretic term power-law) in equation 1, but a slope in equation 2; and (iii) N_w is isomorphic to the survival function $S(w)$ for the random variable W, which is

[2]Since this study was first conducted, MASON versions 9, 10, and 11 have been released at the MASON website.

defined as the complementary c.d.f. of W, or $1 - \Phi(w)$. Besides their application to economic inequality, equations 1 and 2 also apply to numerous other social phenomena, including organizations, warfare, language, city sizes, social networks, and others (Cioffi-Revilla 2003)—hence the deep significance of Pareto's law and similar power laws in the social sciences.

In addition to histogram plots, the original Sugarscape model also measured inequality by the so-called Gini index, defined as the difference between a perfectly egalitarian distribution and the actual wealth distribution of a population. In practice, this can be defined in terms of the Pareto index.

Definition 2 (Gini index) *The measure of inequality g, called the Gini index, is defined as follows:*

$$g = \alpha/(\alpha - 1), \tag{3}$$

where α is the Pareto coefficient.

By definition, $g = 1$ in a perfectly egalitarian society (uniform distribution) and increasing values denote increased inequality (decreased equality).

In this outcome, agents are replaced between the ages of 60 to 100 years (time steps) via rule **R**. As shown in Figure 2, the qualitative outcome in Animation II-3 is an initially symmetric (normal) distribution of wealth, which subsequently evolves into the highly skewed Pareto-like distribution where many (most) agents are poor, some are halfway, and a few are very wealthy (the typical 80–20 rule or histogram with many-some-few frequencies; Kleiber & Kotz 2004).

The significance of this replication cannot be overstated: *The Sugarscape simulation model provided the first generative proof—formally, a computational derivation—of Pareto's Law.* As shown in Figure 2, for time steps 0 and 500, simulation results generally replicate the Pareto distribution. The only difference, if any, may be that the final frame in Animation II-3 features a maximum wealth bin that is somewhat greater than the maximum wealth bin in the MASON Sugarscape. In power-law terms, the MASON upper-tail is slightly thinner than what appears in the target data, Animation II-3 (*GAS*, p. 34). However, the time step for the final frame in Animation II-3 is not specified in *GAS*, so it is possible that the original model did not run as long as ours. In any case, agreement between the two long-term asymptotic regimes is the signficant important replication (Cioffi-Revilla 2002: 7315).

4.3 Emergence of Collective Migratory Waves

Another example of a general replication—although without a special modification the replication is questionable at best—is Animation II-6 (*GAS*, p. 43). As shown in Figure 3, upper left frame, by starting the agents in a block configuration, and increasing maximum vision to 10 (minimum vision is unspecified), waves emerge in a diagonal direction. The significance of this outcome is that it is a demonstration of emergent behavior without any apparent link to individual behavior. In particular, none of the individual agents can move in a diagonal direction, but ensembles of these agents do appear to move diagonally—as a wave that propagates along a SW-NE axis.

This phenomenon could not be replicated in MASON Sugarscape without delaying the growback for a site that had its sugar harvested for a full time period. Ordinarily, sites growback during a time period even if an agent is on the site and has harvested all the sugar. The line of code that controls this behavior is in the **Movement** rule,

```
s.time_since_last_regen[a] = 0;
```

which changes to 1 for normal growback behavior.

This slight modification was the simplest change we could implement that successfully generated the wave phenomenon. The space-time diagram depicted earlier (Figure 1) and the Timing and Scheduling Subsection explain why this change enables the wave phenomenon; preventing growback during a harvest step prevents an agent from backtracking until two time steps later.

Finally, the waves were more coherent and similar to Animation II-6 if minimum vision was increased to 5 and maximum vision increased to 15. Such simple tuning yielded a close replication of Animation II-6.

The replication results of this outcome are significant because they highlight how the smallest changes in source code can result in dramatic, emergent effects that might be mistaken for meaningful results. While computational artifacts

5 Discussion

Epstein and Axtell described a set of simple rules that generate complex, emergent behavior. Their use of simple, concrete language, and vivid understandable graphics and diagrams, it is easy to assume that implementing and replicating their model . There are two forms of complexity

Software engineering and the methodology of computational social science are young fields, although tools and techniques have emerged to support error/bug reduction, automated testing, and faster development. While OO-based software development has been touted as an effective way of constructing software applications, OO methodologies do not eliminate cognitive error nor eliminate complexity in development. Close examination of the psychology of software development is beyond the scope of this paper, but attention to the biases and heuristics literature and cognitive errors literature might yield some techniques and suggest tools appropriate for developing agent-based models.

Clearly more experienced software developers will be more likely to produce correct code more quickly and develop more evolvable code. The ideal agent based modeler, assuming a single-person approach, will have significant development experience and deep domain knowledge regarding the phenomena of interest. It is probably not reasonable to expect significant numbers of researchers with these qualifications, which explains why collaborative teams are increasingly common in computational social science research.

Although some exploration and excursions are described, the authors did not spend much time (for understandable reasons), describing dead ends or techniques they used to search the rules design space. In the case of *GAS*, the authors' command of the social sciences literature clearly influenced model evolution. Design is a creative process, and searching high dimensional spaces will remain an art, though perhaps moving from different regions of these spaces can be made less laborious, quicker, and less prone to error by carefully derived techniques and tools. Clancy et al. (1997) have created a set of techniques for revising qualitative simulation models of systems with discrete structural components that might be adapted. The cognitive science literature on diagnosis of complex systems may be another source of techniques or guidance.

Another barrier to successful implementation was the use of the animations—both statically presented in *GAS* and dynamically during MASON simulation runs—to understand local and global behavior. The power of the human visual system is that it can quickly identify patterns and detect diverse and subtle changes over time. The visual system is biased in certain ways, and these biases can be reinforced by the way data are presented. In this case, both the static animation panels and MASON visualizations portray the end of each time step, but the relationship between the last time step and the completed one as visualized is not direct. For example, one cannot ascertain the potential trading partners of a particular agent by looking at the neighbors displayed for the most recently completed time step. Any of those neighbors might have executed after the target agent, or there might have been agents who were neighbors, but moved away before the target agent executed. While this is not difficult to understand, the influence of the current state as graphically depicted does lead to incorrect inferences at times.

Before publishing *GAS*, Axtell and Epstein produced a short paper entitled "Agent-Based Modeling: Understanding Our Creations," the theme of which is summarized at the outset:

> if we cannot understand these artificial complex systems any better than we understand the real ones then we havent made progress (Axtell and Epstein, 1994, p. 28).

In that same paper, they offered a four-level scale linking performance measures and analysis requirements, where 0 = *caricature* models and 3 = *quantitatively precise* models, with 1 and 2 in between these extremes. Although the Axtell-Epstein scale is not explicitly discussed in *GAS*, it appears to be the methodological standard used for verification and validation. By such as standard the replication of the Pareto Law attains level 3, whereas replication of the migratory wave seems closer to levels 0 or 1, because empirical details are not targeted.

Exploiting knowledge of computational processes is would help in verification and validation. Progress has been made in recent decades in understanding computational behavior and artifacts in 1-D and 2-D cellular automata. Huberman and Hogg (1988) promote the study of 'ecology of computation' and have identified many different behavioral regimes for populations involving local behavior and global effects. How effectively can we use ecology of computation results for difference and differential equations approaches to better predict, understand, and verify agent-based models and simulations? A recent paper by Young (2006) suggests that Markov field analysis can help understand long running simulation outcomes for complex models.

Edmonds and Hales (2003) recently attempted to replicate Riolo's model of cooperation evolution via tags involving 100 agents over 30,000 generations. In addition to strong claims about invalidating some of Riolo's results, they provide heuristics and guidelines regarding replication, including that there should be a publication norm regarding results—"namely that the description of the simulation should be sufficient for others to be able to replicate (i.e. re-implement) the simulation." Edmonds and Hales have also recommended replicating a simulation model using two different languages and two different implementers, though it is unclear why different languages should be a primary technique in replication. They suggest using unseen or new parameter settings and running simulations for each implementation using these new parameter sets to provide an additional verification. Other suggested heuristics involve the use of statistical tests, such as the Kolmogorov-Smirnov test, to verify or at least attempt to refute a hypothesis regarding replication (e.g., outcomes come from the same distribution).

The following list provides a summary of lessons learned during development and replication of the MASON Sugarscape model:

1. Simple models (in the sense of Simon) do not necessarily result in simple code.
2. It is often difficult to predict what computational behavior and artifacts will emerge in simulations and/or to determine cause and effect.
3. Very small changes to core rules can have dramatic, ripple effects across simulation outcomes under interaction rule configurations.
4. It can be difficult to diagnose one's code and/or infer correct behavior from others' pseudocode, screenshots, and a few aggregate statistics.
5. How rules and models were evolved and generated may not be described.
6. The architecture used for rules, action, and sequence will evolve significantly.
7. Any one of (i) lack of domain knowledge, (ii) lack of software language experience, or (iii) lack of knowledge regarding the simulation toolkit in use can dramatically affect both development efficiency and accuracy.
8. It is easy to assume that even relatively simple code is correct simply by viewing graphical behavior output.
9. Discrete time step based visualizations of agent state and space (model state) can be deceiving.
10. Effective visual communication of models can lead to a self-deceptive belief that implementing and/or replicating will be easy.
11. Distributing simulations with different parameter sets across computing resources was cumbersome, more so than initially expected.

Based on this experience, ideas from relevant literature, we recommend the following as potential heuristics:

1. Source lines of code counts and other software complexity metrics for original work should be published so that correlations can be made between software metrics and rules to better understand their scope and complexity.
2. Literature and results from 1D and 2D cellular automata and ecology of computation fields should be mined and distilled to better differentiate results from artifacts and to help evolve models. Methods from qualitative simulation should be explored and integrated to help improve the search of model space.
3. Emphasis on the correctness of the core rules is paramount as well as explicit attention to the use of randomization.
4. All uses of randomization should be carefully documented and continually
5. Researchers should consider archiving and publishing detailed simulation output—object states, a variety of aggregate measures, and perhaps fractal dimension measures for every significant outcome. This would help with both code verification diagnosis and model replication.
6. Researchers should attempt to document how the model design space was searched.
7. #3 notwithstanding, implementation of complex rules—rules interacting with other rules should not be deferred for too long due to design impacts. A balance is necessary.
8. Researchers will have to spend significant time in multiple disciplines to become effective.
9. Unit testing methods ought to be used as frequently as possible.
10. Sequence diagrams, space-time diagrams, and other de-biasing and cognitive aids should be used.
11. One approach to grid computing, distributing parameter sets across nodes (rather than distributing elements of a single simulation) ought to be considered for inclusion in future agent-based modeling and simulation toolkits.

6 Summary

In support of a broader objective of advancing and examining the practice of agent-based modeling in computational social science, this research had two primary goals: to demonstrate the suitability of MASON to replicate a classic agent-based model, Sugarscape; and to demonstrate that general replication of Sugarscape is possible. Sugarscape consists of a set of very simple rules—some are not simple to implement—yet the resulting dynamics in simulation produce complex, emergent phenomena. General replication was achieved, although significant discrepancies remain unresolved, particularly for population-size based outcomes. MASON was mature enough and provided core capabilities to replicate Sugarscape; it was extended in a few ways to enhance the process of simulation testing, such as automated parameter sweeping infrastructure and automated logging. During the process of replication, obstacles emerged such as numerous researcher induced errors and the difficulty of understanding the relationship between source code and emergent behavior. Reducing these barriers for social scientists through better tools and techniques is a significant challenge.

7 References

Axtell, Robert L., and Joshua M. Epstein. 1994. Agent-Based Modeling: Understanding Our Creations. *The Bulletin of the Santa Fe Institute Winter* (Winter):2832.

Bigbee, Tony, C. Cioffi-Revilla & S. Luke (2005a) Replication of Sugascape Using MASON. *Proceedings of the 4th International Workshop on Agent-Based Approaches in Economic and Social Complex Systems AESCS 2005*, Tokyo Institute of Technology, Tokyo, Japan, July 913, 2005.

Bigbee, Tony, C. Cioffi-Revilla & S. Luke (2005b) Replication of Sugarscape Using MASON. *Representing Social Reality: Pre-Proceedings of the Third Conference of the European Social Simulation Association*, edited by Klaus G. Troitzsch, Koblenz, Germany, September 5–9, 2005.

Brajnik, G. and Lines, M. (1998). 'Qualitative Modeling and Simulation of Socio-Economic Phenomena. *Journal of Artificial Societies and Social Simulation*, 1 (January). Retrieved August 20, 2004, from http://www.soc.surrey.ac.uk/JASSS/1/1/2.html.

Cioffi-Revilla, Claudio. (2002). Invariance and Universality in Social Agent-Based Simulations. *Proceedings of the National Academy of Science of the U.S.A.* 99 (Supp. 3) (14):7314–7316.

Clancy, D. , Brajnik, G., and Kay, H. (1997). Model Revision: Techniques and Tools for Analyzing Simulation Results and Revising Qualitative Models. In *Proceedings of the 11th International Workshop on Qualitative Reasoning*, Cortona, Siena, Italy; June 1997, pp. 53–66. Retrieved August 20, 2004, from http://www.dimi.uniud.it/ giorgio/papers/qr97-mr.ps.gz

Densmore, O. (2005). Sugarscape. Retrieved April 1, 2005, from http://backspaces.net/Models/sugarscape.html

Doran, J. (2000a) Trajectories to Complexity in Artificial Societies. In T. Kohler & G. Gumerman (Eds.), *Dynamics in Human and Primate Societies: Agent-based Modeling of Social and Spatial Processes.* New York: Oxford University Press.

Doran, J. (2000b). Questions in the Methodology of Artificial Societies. In R. Suleiman, K. Troitzsch, & N. Gilbert (Eds.), *Tools and Techniques for Social Science Simulation.* Heidelberg: Physica-Verlag.

Edmonds, Bruce, & David Hales (2003). Replication, Replication and Replication: Some Hard Lessons from Model Alignment. *Journal of Artificial Societies and Social Systems* 6 (4).

Epstein, J. & Axtell, R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up.* Washington, D.C.: Brookings Institution and MIT Press.

Galan, Jose Manuel & Luis R. Izquierdo (2005). Appearances Can Be Deceiving: Lessons Learned Re-Implementing Axelrod's 'Evolutionary Approach to Norms', *Journal of Artificial Societies and Social Simulation* vol. 8, no. 3 http://jasss.soc.surrey.ac.uk/8/3/2.html.

Gilbert, N. & Doran, J. (Eds.). (1994), *Simulating Societies: The Computer Simulation of Social Phenomena.* London: UCL Press.

Gizzi, M., Lairson, T., & Vail, R. (2003). Modifications to an original model created by Uri Wilensky (1998). Mesa State College, Center for Agent-Based Modeling. Retrieved August 10, 2004, from http://www.modelingcomplexity.org.

Hegselmann, R., & Flache, A. (1998, June). Understanding Complex Social Dynamics: A Plea for Cellular Automata Based Modeling. *Journal of Artificial Societies and Social Simulation*, 3, Retrieved August 1, 2004, from http://www.soc.surrey.ac.uk/JASSS/1/3/1.html

Huang, C., Sun, C., Hsieh, J., & Lin, H. (2005). Simulating SARS: Small-World Epidemiological Modeling and and Public Health Policy Assessments. *Journal of Artificial Societies and Social Simulation*, 7, Retrieved April 1, 2005, from http://jasss.soc.surrey.ac.uk/7/4/2.html

Huberman, B., and Hogg, T. (1988). The Behavior of Computational Ecologies. In B. Huberman (Ed.), *The Ecology of Computation*. Amsterdam: North-Holland.

Kleiber, C. & Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*. Hoboken, NJ: John Wiley and Sons, Inc.

Kliemt, H. (1996). Simulation and Rational Practice. In R. Hegselmann, U. Mueller (Eds.), *Modelling and Simulation in the Social Sciences from a Philosophy of Science Point of View*. Dordrecht: Kluwer.

Lakatos, Imre. 1973. Falsification and the methodology of scientific research programs. In *Criticism and the Growth of Knowledge*, edited by I. Lakatos and A. Musgrave. Cambridge: Cambridge University Press.

Luke, S., Cioffi-Revilla, C., Panait, L., & Sullivan, K. (2004). MASON: A New Multi-Agent Simulation Toolkit. Retrieved September 9, 2004, from http://cs.gmu.edu/ eclab/projects/mason/publications/SwarmI

Luke, S., Cioffi-Revilla, C., Panait, L., &Sullivan, K. (2005) MASON: A Java Multi-Agent Simulation Environment, *Simulation: Transactions of the Society for Modeling and Simulation International* 81(7)2005:517–527.

Nowak, A. & Lewenstein, M. (1996). Modeling Social Change with Cellular Automata. In R. Hegselmann & U. Mueller (Eds.), *Modelling and Simulation in the Social Sciences from a Philosophy of Science Point of View.* Dordrecht: Kluwer.

Wheeler, D. (2005) SLOCCount. Retrieved April 1, 2005, from http://www.dwheeler.com/sloccount/

Wilensky, U. (1998). NetLogo Wealth Distribution Model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Retrieved August 10, 2004, from http://ccl.northwestern.edu/netlogo/models/WealthDistribution.

Wright, Henry T. (2000). Agent-Based Modeling of Small-Scale Societies. In T. Kohler & G. Gumerman (Eds.), *Dynamics in Human and Primate Societies: Agent-based Modeling of Social and Spatial Processes.* New York: Oxford University Press.

Young, H. Peyton. (2006). Social Dynamics: Theory and Applications. In Handbook of Computational Economics, Vol. II., edited by K. Judd and L. Tesfatsion. Amsterdam: North Holland, 2006.

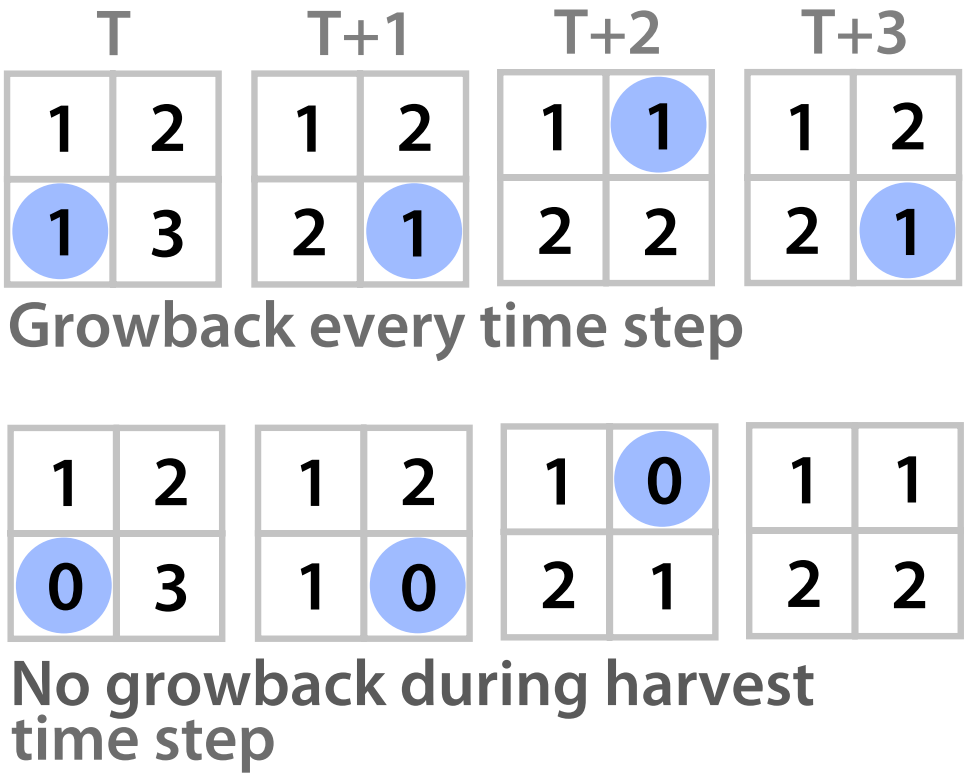

Figure 1: Step-by-step operation of the **Growback** rule .

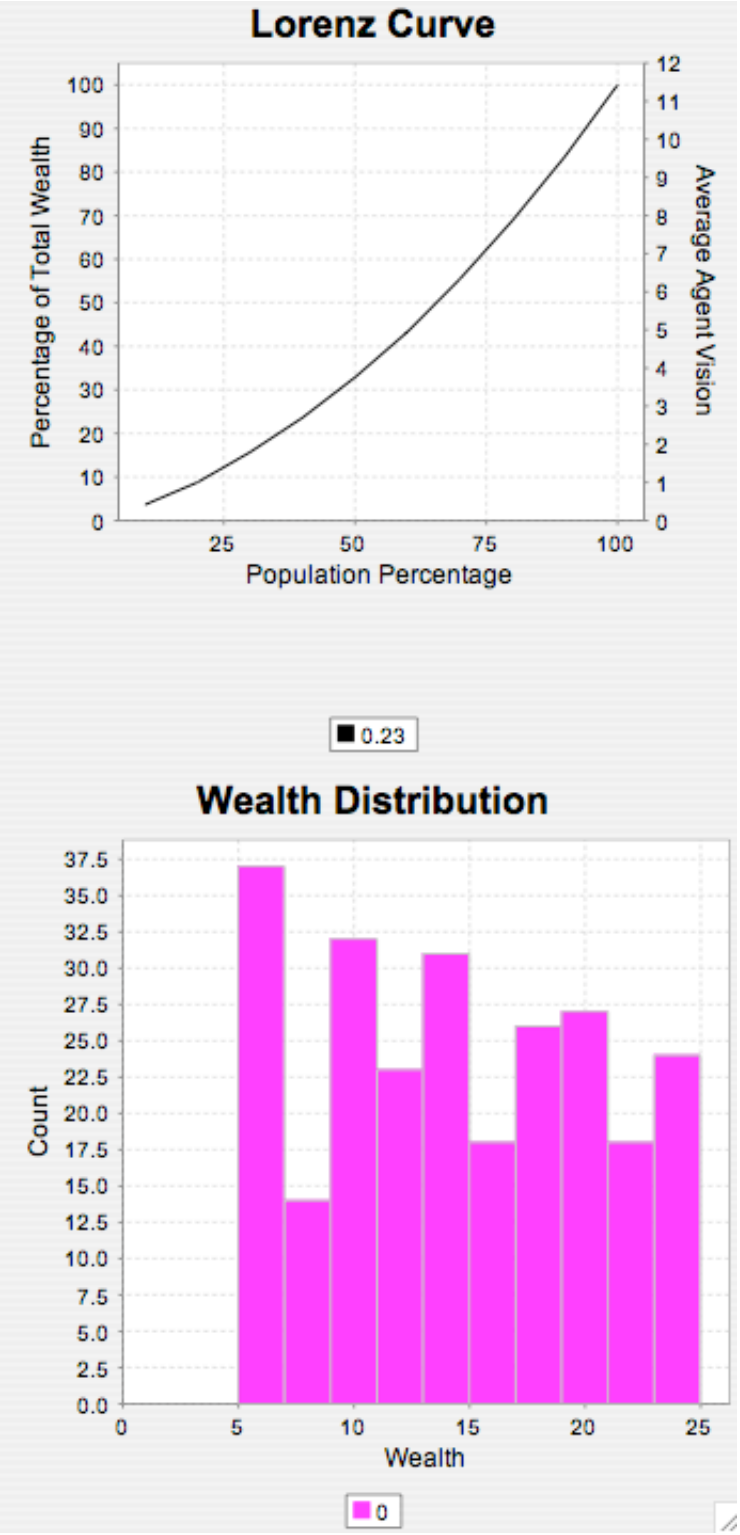

Figure 2: Lorenz curves and distribution histograms as emergent behaviors in the Sugarscape model, for initial (Step 0) and evolved (Step 500) states of the simulation.

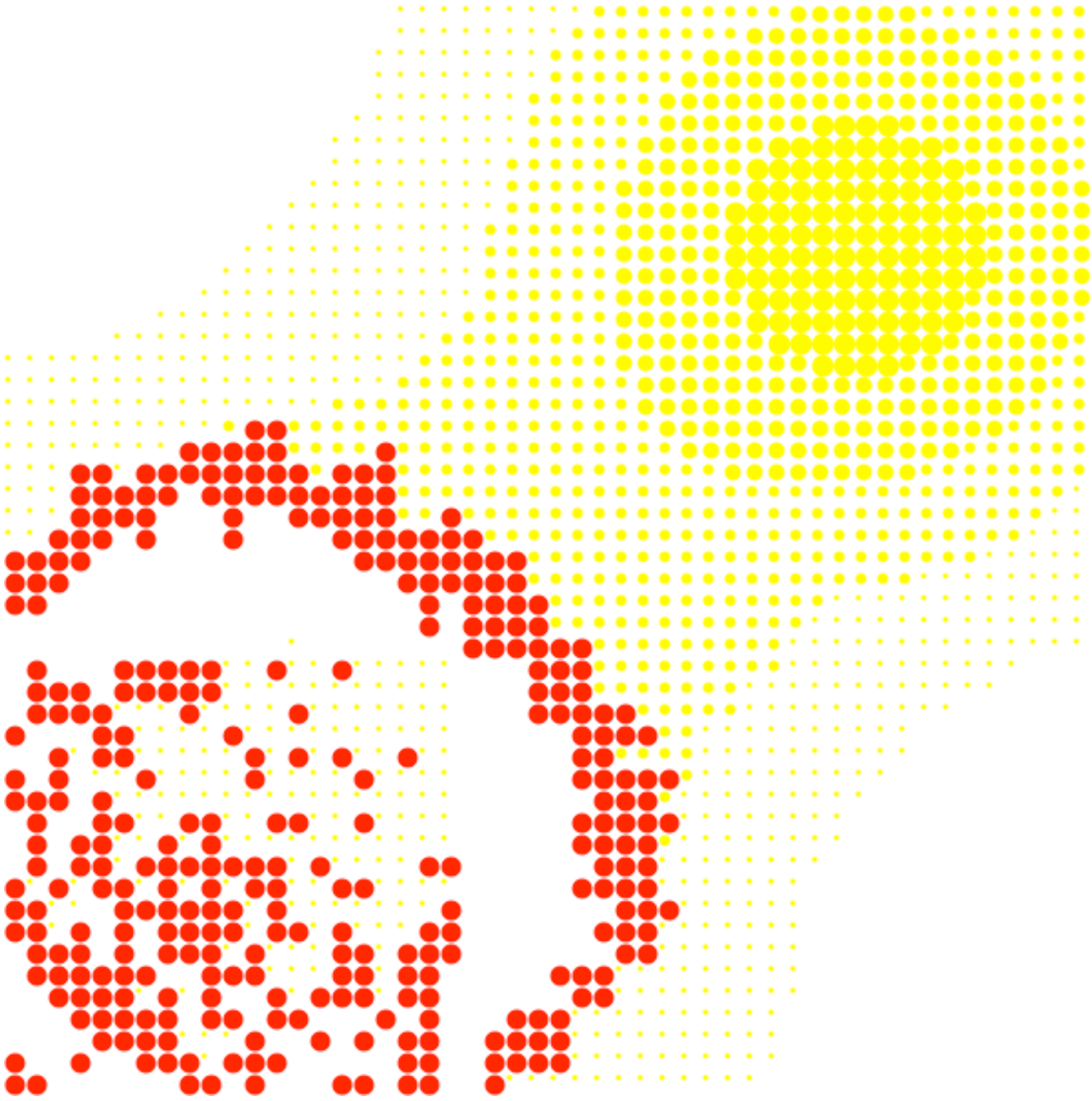

Figure 3: The so-called "wave" as emergent behavior in the Sugarscape model.