Modeling Law: Using Cellular Automata to Study Legal Regulations of Conflicting Land Use

Seth J. Chandler (SChandler@uh.edu)
University of Houston Law Center, Houston, Texas, USA

Abstract

This paper introduces cellular automaton technology to the field of law and economics. It provides this supplement to the traditional “constrained optimization” approaches practiced in the discipline by using the Mathematica computer language to set up economies modeled as lattices whose sites contain actors with diverse, imperfect, and dynamic learning styles. These lattices evolve according to potentially complex local rules and global properties of the automaton that attempt to capture the interaction of the legal system, the economy, and the actors within it. The technology is illustrated through examination of a traditional legal problem: regulation of conflicting uses of adjoining land. The theoretical construct employed, and its implementation in the Mathematica language, should extend to a variety of problems in which the economic effects of a given legal rule depend on the local and global interaction of a variety of actors with the capability to evolve.

1. Introduction

The legal problems created by conflicting use of adjoining land is ancient\(^1\), continuing\(^2\), and transnational.\(^3\) Traditional economic analysis of this problem has assumed that landowners are perfect calculators, assimilating all local and global properties of the economic and legal system, and that all adjustments to equilibrium take place instantaneously. Such assumptions have, for example, generally nourished analysis of the Coase Theorem, a central concept in the economic analysis of legal rules, which states that, in the absence of transaction costs, parties (such as adjacent landowners) with the potential to interact in a disadvantageous fashion will negotiate to an “economically efficient” outcome that maximizes total wealth regardless of the choice of legal rule governing relations among the parties.\(^4\) These models fail to address more realistic settings in which spatially arrayed actors interact locally and learn on the basis of limited knowledge regarding the local and global environment.
This paper creates a template for study of the effect of legal rules on more realistic economic systems. Specifically, it demonstrates the use of cellular automata to examine the effect of legal rules when the fortunes of individual actors are determined by a combination of certain observable gross features of the economy (prices) and local interactions about which the actors have limited knowledge. It demonstrates the utility of this approach by study of the land use problem, the evolutionary development of a lattice (ring), each site of which has the potential to engage in an activity that conflicts to a greater or lesser extent with the activities of its neighboring sites. To do so, it builds on the work of Gaylord and Nishidate in modeling learning behavior with an automaton, but both generalizes the processes described in their work and extends it to the domain of legal rules. Unlike previous efforts in the field, in which learning styles are exogenously imposed, here learning styles evolve.

2. Constructing A General Learning Model for a Cellular Automaton Using Global and Local Properties

The central tool employed in the model is the *spatialevolution* function implemented in the *Mathematica* computer programming language within a Cellular Automata “package.” The function shows how a lattice evolves over time according to specified rules. This section explains how this tool, useful in economic modeling such as that done here and in other forms of cellular automata research, is forged from the more primitive *latticeapplicator* and *latticeapplicatorlist* functions. These functions are in turn constructed from built-in *Mathematica* functional constructs.

The *latticeapplicator* function has two arguments, a lattice and a list called *fnlist* whose first part is a symbol called *func* and whose second part is another list called *neighborhood*. As verified with the pattern testing function *integerlistQ*, each element of *neighborhood* is an integer encapsulated within a *List* head. The *latticeapplicator* function uses the built-in *MapThread* construct to create a lattice of *Mathematica* expressions with the same dimensionality as an original lattice provided. Each of these expressions in the new lattice, prior to evaluation, contains *func* as its head followed by a sequence of parts, each of which is a selection from the original lattice. The selection of these parts depends on data contained in the *neighborhood* part of the *fnlist*. This neighborhood data is then “Mapped” over a *RotateRight* function that has the original lattice as its first argument. The selected lattice sites within each of these *Mathematica* expressions in the new lattice thus bear a consistent spatial relationship to each other.

```mathematica
latticeQ[x_] := MatchQ[x, _List]
integerlistQ[x_List] := And @@ Map[MatchQ[#, _Integer] &, x]
latticeapplicator[
latticeapplicator[
  _?latticeQ, {func_Symbol, neighborhood_List?integerlistQ}]:=
  MapThread[func, Map[RotateRight[1, #1] & , neighborhood], Length[First[neighborhood]]])
```
The following code illustrates this algorithm in “slow motion.” The Map construct creates various rotations of the original lattice with the arguments to the rotation procedure determined by the neighborhood variable. The MapThread construct then makes \( f \) the head of a sequence of expressions each of which (in this one-dimensional example) may be thought of as a “column” of the rotations created by the Map construct.

\[
f/: \text{neighborhood}[f] = \{\{0\}, \{-1\}, \{1\}\}
\]

\[
\text{rotations} = \text{Map[RotateRight[Array[s,\{5\}],\#1\]&,neighborhood[f]]} // \text{ColumnForm}
\]

\[
\{s[1], s[2], s[3], s[4], s[5]\}
\]

\[
\{s[2], s[3], s[4], s[5], s[1]\}
\]

\[
\{s[5], s[1], s[2], s[3], s[4]\}
\]

\[
\text{MapThread}[f,\text{rotations,Length[First[neighborhood[f]]]}]
\]

\[
\{f[s[1],s[2],s[5]], f[s[2],s[3],s[1]], f[s[3],s[4],s[2]], f[s[4],s[5],s[3]], f[s[5],s[1],s[4]]\}
\]

The latticeapplicatorlist function now combines the constructed latticeapplicator function with the built-in Mathematica Fold function to model a series of functional operations on the lattice. The beginning functions in the fnlist may be used to compute some sort of “score” for each lattice site while the last functions in fnlist can implement learning by assigning values to each site based on the “scores” of its neighbors. The neighborhoods of each function can differ.

\[
latticeapplicatorlist[\_?latticeQ,fnlist:{\{\_Symbol,\_List?integerlistQ\}...}]:=
  \text{Fold[latticeapplicator,1,fnlist]}
\]

To illustrate:

\[
g/:\text{neighborhood}[g] = \{\{2\}, \{-1\}\};
latticeapplicatorlist[Array[s,\{5\}], \{\{f, \text{neighborhood}[f]\}, \{g, \text{neighborhood}[g]\}\}] // \text{ColumnForm}
\]

The lattice evolution function is then constructed out of these primitives by using Mathematica’s built-in NestList construct. Essentially, one uses NestList to repetitively apply a function called globalandlocal to the lattice. The globalandlocal function is itself created from the compound of a globalpre
function (set to a Null default using the Options technique) that evaluates global properties of the lattice and the latticeapplicatorlist function described above.

\[
\text{Options[spatialevolution] = \{globalpre -> Function[Null]\};
\]
\[
\text{spatialevolution[l_?latticeQ, fnlist : {{_Symbol, List?integerlistQ}...}, n_Integer, opts ___] :=
\]
\[
\text{Module[\{globalandlocal, g\},
\]
\[
g = \text{globalpre} /. \{\text{opts}\} /. \text{Options[spatialevolution]};
\]
\[
\text{globalandlocal} = \text{Function[}
\]
\[
\text{CompoundExpression[}\#\text{, latticeapplicatorlist[}\#, fnlist\text{]]];
\]
\[
\text{NestList[globalandlocal, 1, n]}\]
\]

3. Applying The Cellular Automaton Functions to Legal Rules

Methodology

With the necessary general research tool now manufactured, cellular automata can be deployed to examine ways in which different legal rules shape the evolutionary spatial development of potentially conflicting economic activities. To illustrate the use of this technology, I reexamine an archetype of law and economics: the conflicting land use problem.

I create a lattice each site of which models a plot of land that devotes itself to farming or ranching. In the unfenced world modeled by the lattice, cows from each ranch may roam onto neighboring farms, trampling some of their crops. There are no destructive interactions between neighboring farms and neighboring ranches. Within this lattice, farmers decide how many crops to raise and ranchers decide how many cows to raise. The initial score for each site depends on (1) the activities at the site itself (a local property); (2) for farming sites, the destructive effects of any neighboring local cows; (3) legal rules regarding compensation for damage caused by roaming cows that trample crops; and (4) prices for crops and cows prevailing in the economy, which is assumed to result from the global number of crops and cows produced by all sites in the lattice economy.

The lattice is implemented within the Mathematica language as a fifty part list of site expressions each of which has seven parts: (1) the profitability of the site (initially left blank); (2) the type of activity conducted at the site (either farming or ranching, chosen randomly); (3) the amount of activity conducted at the site (a random value chosen from 0 to 100); (4) the number of cows neighboring the site (initially left blank); (5) the total damage done to the site by roaming cows (initially left blank), (6) an “expansion slot” not implemented in the current model; and (7) an initial randomly selected choice from four learning styles implemented as a global variable called learningstyles bearing a List head. Thus, at the outside, a “Short” version of the global variable randomlandscape might look as follows:

\[
\{\text{site[_, farm, 82.5938, _, _, broadsearchcomplex]},
\text{site[_, ranch, 70.5943, _, _, broadsearchcomplex]}, <<47>>,
\text{site[_, farm, 59.8775, _, _, narrowsearchsimple]}\}
\]
The **economy** function examines the entire automaton to determine various global aspects of the lattice economy. Here, the examination determines (1) the price of cows and crops that will equilibrate exogenously determined demand to the aggregate supply of cows and crops produced by all sites in the lattice, and (2) the “gross national product” that results from producer profits and consumer surplus associated with production and consumption of crops and cows. Because, here as elsewhere in this article, a full exposition of the *Mathematica* code implementing this operation is inessential to the presentation, I show simply the following “shallow code sketch.” The sketch shows that a side effect of the **economy** function is to store relevant results are stored in the global variables *cropprice*, *cowprice*, and *gnp*.

```
Shallow[DownValues[economy], 8]
{HoldPattern[economy[l_?latticeQ]] :>
 With[{inversedemand=Times[«2»]&, sites=50}, Module[
 {totalcrops, totalcows, thecrops, thecows},
 thecrops=Cases[«2»];
 totalcrops=Apply[«2»];
 thecows=cases[«2»];
 totalcows=Apply[«2»];
 {«2»}=Map[«2»]; gnp=+«3»;]
 ]}
```

The functions **surroundingcows**, **trample** and **profit** are used sequentially to determine the profitability of each site in the lattice. As suggested by the following “shallow code sketches,” the function **surroundingcows** modifies a site to include information on the number of cows in neighboring sites. The function **trample** modifies a site to determine the number of crops (if any) in neighboring sites trampled by any cows within the site. The **profit** function uses information regarding a site, its neighbors, the legal rule regarding compensation in effect, and the globally determined variables *cowprice* and *cropprice*, to modify a site to specify its profitability.

```
Map[Shallow[DownValues[#], 8]&, {surroundingcows, trample, profit}]
{HoldPattern[surroundingcows[Pattern[«2»], Pattern[«2»]]]:>
 If[Part[«2»]==farm, With[{[«1»]}, ReplacePart[«3»], self]],
 HoldPattern[trample[Pattern[«2»], Pattern[«2»]]]:>
 If[Part[«2»]==ranch, With[{[«1»]}, ReplacePart[«3»], self]],
 HoldPattern[profit[Pattern[«2»]]]:> s[[1]],
 HoldPattern[profit[Pattern[«2»], Pattern[«2»]]]:>
 Which[sitetype[«1»]==farm,With[{[«1»]}, ReplacePart[«3»]],
 sitetype[«1»]==ranch,ReplacePart[self, +«3», 1]]}
```

Having completed the “scoring” for each lattice site, I define a learning method under which the site scans the profitability of itself and its nearest neighbors. Almost all the time, it copies the learning style of its most profitable neighbor. To prevent lock-in of learning styles, however, the site “experiments” a small fraction of the time with randomly selected learning styles. The learning method of each site is thus endogenous, at least within the restricted class of permitted learning styles.\(^9\)

```
sitesequence=Sequence[self_site, neighbors_site];
learntolearn/: neighborhood[learntolearn]={{0}, {-1}, {1}};
```
learntolearn[sitesequence] := With[{oldlearningmethod = Function[#[[7]]],
newlearningmethod = Function[ReplacePart[#1, #2, 7]],
nonrandom = 0.95,
If[Random[Real, {0, 1}] <= nonrandom, newlearningmethod[
   self, oldlearningmethod[Last[Sort[{self, neighbors}]]]],
   newlearningmethod[self, learningstyles[[Random[Integer, {1, 4}]]]]]
}

Using its own learning style, the site then engages in some mixture of its own
prior behavior and the behavior of a more profitable neighbor, with the precise method
for conducting this “mating” depending on the learning style adopted. It does so
through the learn function, which, harnessing the functional powers of the
Mathematica language, takes the last part of a site (which houses the learning style) to
construct a Mathematica expression with the learning style of that site as its head and
the sites in the learn neighborhood\textsuperscript{10} as its arguments. These neighborhoods may be
quite broad or quite narrow. More complex learning styles that use more information
or more computational efforts are more costly. This function is in turn evaluated by the
Mathematica kernel to determine the future behavior of the site.

learn/: neighborhood[learn] = {{0}, {-1}, {1}, {-2}, {2}};
learn[sitesequence] := With[{oldlearningmethod = Function[#[[7]]],
   oldlearningmethod[self][self, neighbors]}

To illustrate this critical process in slow motion, suppose that after application
of the learntolearn function to the entire lattice, part of the lattice looked as follows.

<table>
<thead>
<tr>
<th>Site Number</th>
<th>Site Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>site[5.30776,farm,43.7214,55.8666,0,_,narrowsearchcomplex]</td>
</tr>
<tr>
<td>26</td>
<td>site[8.89917,farm,41.5888,105.605,0,_, broadsearchcomplex]</td>
</tr>
<tr>
<td>27</td>
<td>site[-1.10083,farm,37.808,105.605,0,_, broadsearchcomplex]</td>
</tr>
<tr>
<td>28</td>
<td>site[168.614,ranch,90.4232,85.5871,21.1126,_,broadsearchcomplex]</td>
</tr>
<tr>
<td>29</td>
<td>site[-24.5141,ranch,10.7712 ,59.3407,0,_,broadsearchcomplex]</td>
</tr>
</tbody>
</table>

The following expression, the Short form of a Trace of a learn expression,
shows how site 27 would be updated as part of the spatialevolution process. A similar
process is conducted for all sites in the lattice.

{broadsearchcomplex[
   site[-1.10083,farm,37.808,105.605,0,_,broadsearchcomplex],
   site[8.89917,farm,41.5888,105.605,0,_,broadsearchcomplex],
   site[168.614,ranch,90.4232,85.5871,21.1126,_,broadsearchcomplex],
   site[5.30776,farm,43.7214, 55.8666,0,_,narrowsearchcomplex],
   site[-24.5141,ranch,10.7712,59.3407,0,_,broadsearchcomplex]],
With[profitability$=#1[[1]]& ,sitetype$=#1[[2]]& ,output$=#1[[3]]& ],
Module[{best$},<<1>>],Module[best$] ,<<1>>],
   site[-7.10083,farm,41.5888,105.605,0,_,broadsearchcomplex]}
An Experiment Using the Methodology

With this background, I now conduct an experiment. I take an initially random landscape and compare legal regimes in which farmers are fully compensated for the damages done by roaming cows (compensationfactor=1) with legal regimes in which farmers are not compensated (compensationfactor=0). I tell the `spatialEvolution` function to update the lattice fifty times. I conduct the experiment one hundred times to determine statistical properties of the evolutionary process.

```math
x = Function[
    CompoundExpression[economy[#], gnphistory={gnphistory, gnp}]];
experiment[cf_] := CompoundExpression[compensationfactor=cf,
    gnphistory={},
    landthroughtime[cf]= spatialEvolution[randomlandscape, {
        {surroundingcows, neighborhood[surroundingcows]},
        {trample, neighborhood[trample]}, {profit, neighborhood[profit]},
        {learntolearn, neighborhood[learntolearn]},
        {learn, neighborhood[learn]}}, 50, globalpre->x],
    thegnphistory[cf]=Flatten[gnphistory]];
```

The following graphics illustrate the evolution of the economy in a typical sample run of the experiment under these two legal rules. Figure 1 shows the evolutionary learning process moving under the guidance of different legal rules from a disordered random landscape (at the top) into a much more ordered and largely homogeneous landscape. The dark areas represent farming, while the lighter areas represent ranching. The output of crops or cows at a site determines the precise shade of gray.

The solution reached by co-evolution among sites unable to bargain with themselves is sites is similar to what would be achieved if bargaining were costless. The landscape is transformed from a random one in which significant costs are generated from adjacent farming and ranching operations to one in which such conflicts are greatly reduced. As expected, the ultimate allocation of the land is bound to the choice of legal rule. The area of land that is ranched is less when ranchers are obligated to compensate farmers for damage done by roaming cows is somewhat less
than that ranched when the law fails to impose compensation. While the precise evolutionary paths followed vary depending on the initial random landscape selected, the emergence of order, coupled with the tie of land allocation to legal rule, appears invariant.

The experiment also generates data on the evolution of learning styles within the lattice. Figure 2, a “stacked bar graph,” shows these results for a particular initial random landscape. Under both legal rules, the learning styles evolve away from a random distribution in which broad and narrow learning modes are equally present. They evolve towards less costly “narrow” searches in which the site examines only its two nearest neighbors. And among the “narrow” learning methods, the lattice evolves in both cases towards the simpler and less costly algorithm for evaluating future behavior. Again, while the details of the process vary, these identified trends repeat over different initial random landscapes. The choice of whether law (whether or not to compensate farmers for roaming cows) made no statistically significant difference in the evolution towards narrow simple searches.

Figure 3 shows a time series for the “gross national product” under each legal rule for a particular initial random landscape. While the gross national products under both regimes ultimately stabilize at largely equivalent levels, for this randomly generated landscape, the full compensation regime takes longer to attain its peak value. In an individual case, then, the choice of legal rules matters.
Mathematica's Statistics' LinearRegression package permits analysis of these one hundred trials to determine whether any obvious relationship exists between gross features of the initial random landscape and the rule of law that maximizes either total gross national product or gross national product at the end of each trial. Figure 4 shows the results. A weak and statistically borderline correlation was found between the number of farms (numberoffarms) in the initial configuration and the desirability of a full compensation rule. This relationship probably exists because denial of compensation tends to induce more costly switches from farming to ranching. Out of 100 trials, however, there were twenty seven "discrepant trials" in which, either (a) despite an initial configuration that had more farms than ranches, denial of compensation maximized long run gross national product or (b) despite an initial configuration that had more ranches than farms, full compensation maximized long run gross national product. The number of "discrepant" trials increases to thirty seven if the terminal gross national product rather than the long run gross run product is examined.

The data shows the difficulty of predicting the effects of legal rules with any confidence when numerous sites interact in a complex fashion. Like many cellular automaton models, this one appears subject to computational irreducibility. By switching just the location of a few sites in the initial random landscape, for example, a full compensation rule can move from being superior to being inferior.
4. Conclusion

This brief article does not pretend to a comprehensive study of the legal issues involved in land use regulation. Rather, it uses the archetypal land use problem to illuminate an alternate path for the economic analysis of law. It shows how the natural coalescence of the Mathematica programming language with the cellular automaton can be used to model the economic evolutionary effects of varying legal rules. Further experiments within the particular land use application might (1) examine the transitional effects of changing legal rules in an evolving landscape and (2) search more broadly for a thus-far elusive simple heuristic device that converts data regarding gross and observable features of the current economic landscape into a reliable choice of the superior legal rule.

5 Wolfram, S., 1994, Cellular Automata and Complexity (Addison-Wesley)
7 In the example provided, I use an upvalue of a symbol to store and pass the neighborhood, but this need not be the case.
10 The neighborhood of the learn function is the broadest neighborhood used by any of the learning styles.