Looking For Company: A Stochastic Approach

Kuan H. Chen
Chemical Engineering
Applied and Computational Mathematics -- Physical and Engineering Sciences

A Fun Narrative

In a fictional world where insects take over the world, human survival is put to the test. Past modeling tools that humans have used to predict the population growth of insects have been defied -- they have underestimated insectile capabilities to the environment. As humans have now learned, insects do not need as many resources from the environment as they had expected; insects also multiply at rates akin to viruses. Humanity depends on its ability to understand insectile population growth, but all deterministic methods from the literature have failed, as parameters used in them have been ill-conceived. Luckily, at the brink of her extermination, humanity discovered this report and has unearthed knowledge that will soon lead to a revolution against the insect kingdom...

Abstract

This report explores an alternative to the traditional deterministic-based method for modeling population growth. While deterministic population growth models focus on the macroscopic behavior traits of a population and integrate them into dynamical systems in the form of various parameters, the stochastic population growth model focuses on local interactions between entities in it that are used to make broader speculations of how a population will evolve. Two stochastic models were built for different types of modeling problems, each used to answer different questions about a population.

The implementation of the stochastic model is done by creating a Cellular Automaton (CA) using Object-Oriented Programing with Java, where each entity in the population is represented by the box on an N by N grid. Animations are created with the DrawingPanel utility, which comes courtesy of Professor Stuart Reges and Marty Stepp at the University of Washington and Stanford University, respectively. This tool is used in conjunction with the Java.awt package by Oracle.

Key Words: Deterministic Model, Stochastic Model, Cellular Automata

Problem Description

The long term behavior of populations of species has traditionally been understood through solving a system of differential equations. **Equation 1** is a one-dimensional dynamical system that is commonly used to model the population growth of various systems, such as fisheries. In this model, $N(t)$ represents the population of the said species as a function of time; r and K each represents the average rate of growth of the population per unit time and the environmental carrying capacity, respectively.

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right) \quad (1)$$

By solving this differential equation, one can gain insight into how the population varies with time. The solution to the equation was first used to model the population of a colony of fish and published by Pierre François Verhulst in 1838 [1] . **Figure 1** presents a graphical solution to the fishery population problem.

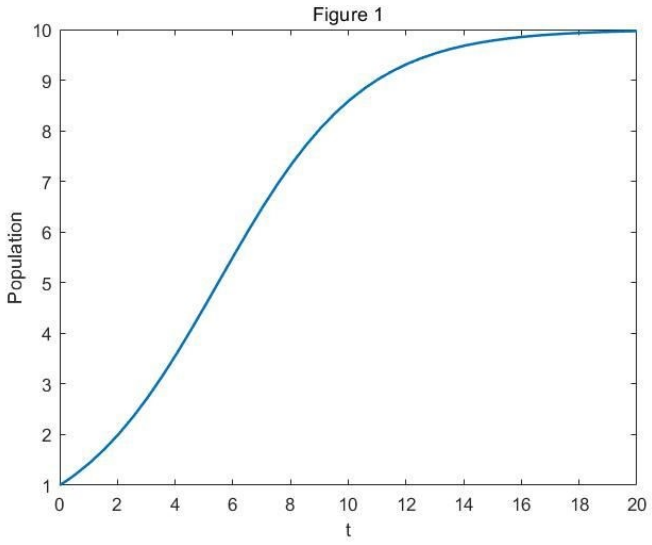


Figure 1: Solution to the one-dimensional continuous model

There are major flaws to the one-dimensional population model. For simplicity, it assumes that the birth and death rates of the said species (as well as other relevant parameters) do not vary with time; they do not account for external factors such as diseases or natural predators. Furthermore, it assumes that the species does not migrate and exhibit no interactions with other species surrounding it. None of these assumptions can be safely made in the real world, and more comprehensive models have been developed to compensate for its shortcomings.

The Lotka-Volterra system of equations, also known as the predator-prey model, was first proposed by Alfred J. Lotka in 1910 to model chemical reactions and is presented in **Equation 2** and **3** [2] .

$$\frac{dx}{dt} = \alpha x - \beta xy \quad (2)$$

$$\frac{dy}{dt} = \delta xy - \gamma y \quad (3)$$

The model has since been used as a more comprehensive model for population growth -- it extends the principles derived in the one-dimensional system to two interacting species, providing better predictions on how the population of species evolves with time in conjunction with one another. **Figure 2** presents various solutions to the system of equations.

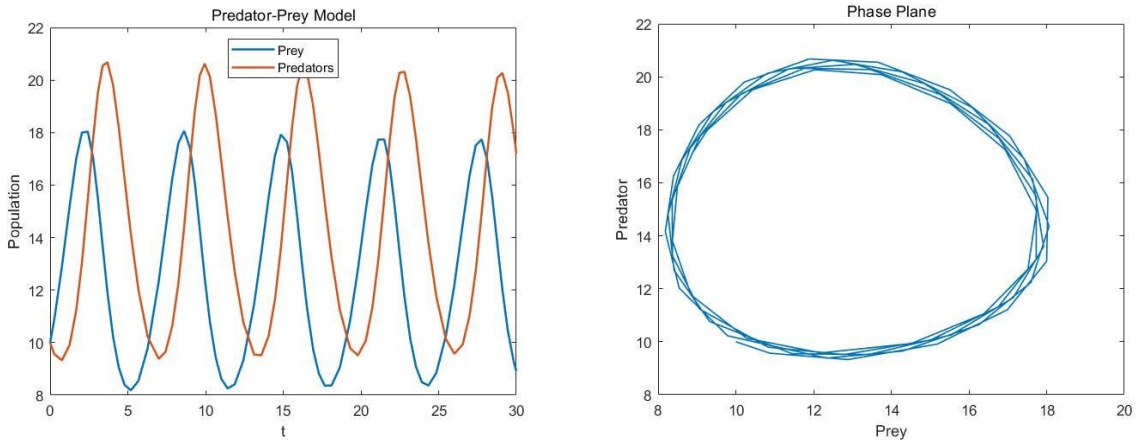


Figure 2: Solution to the Lotka-Volterra system

Despite the improvements to the population model, flaws to the one-dimensional model persist in the two-dimensional system -- the parameters adopted in the model are still assumed to be constant, and two interacting species may very well be both affected by a third external factor, such as competition from a common predator, or the presence of a third primary consumer. This model is also quite limited -- it focuses on the predator-prey relationship between species and fails to model more complex systems, such as mutualism or parasitism, both commonly observed natural phenomena. Different model equations with different parameters would need to be proposed to account for such situations.

The disadvantages to the deterministic approach are clear -- building an appropriate model for a population of a species often takes a long period of time of research and collecting data. Further, the accuracy of these models are highly limited by the computational power of the tools available -- the more variables introduced to the system, the more difficult a solution can be arrived at. These difficulties provide sufficient motivation for one to pursue a model that allows more flexibility and simplicity.

The stochastic approach to the population growth problem focuses on the local interactions of entities and uses them as a basis for broader predictions of how a population evolves. It uses a Cellular Automaton (CA) as its primary analysis tool to obtain data. In the CA modeling scheme, species are represented as boxes of various colors on a checkerboard-like grid. The behavior of these boxes in relation to other boxes in their Moore's Neighborhood (i.e., the 9 other boxes in its surrounding) is meant to reflect how they would behave in the real world. The solution to the problem is described by the overall long-term survival or demise of the species as a whole and is directly affected by the number of behavioral traits that are programmed into them. The more factors accounted for in the model, the more accurate it becomes. Unlike the deterministic

model, simplicity is more easily retained in arriving at a solution. The rest of this report explores how the CA approach compares to the deterministic approach in modeling populations and how it can be used to model the peculiar nature of cicadas and their relation to their predators.

Mathematical Model and Simplifications

With little limitations to what can be programmed into a CA, determining what behavior traits are significant or trivial to the growth of the population becomes key in building a model. As one can see, the traits that will be deemed trivial or not to the model is largely dependent on the type of problem to be solved. In this report, the one-dimensional population growth model by CA is used to demonstrate how the stochastic method can be applied to scenarios similar to those that can be solved by **Equation 1**; the multivariable predator-prey model by CA is used to explain the oddities in the insect cicadas' life-cycle.

To construct a model that can be generally applied to all species, such as the one in **Equation 1**, no particular genus of animal is specified. This allows one to avoid the need for accounting for behavioral traits that may be unique to certain species. The "generic" animals in the one-dimensional growth model are represented by binary colors, black and white, on an N by N grid -- black signifying the presence of an animal, and white signifying the absence of one. Sample populations, one representing an overpopulated initial state, one populating 50% of the grid, and the other representing an initial endangered state is presented in **Figure 3 a - c**.

(a) **(b)** **(c)**

Figure 3: (a) Endangered population (b) 50% of grid populated (c) Overpopulated population

To be able to make comparisons between this model and **Equation 1**, the only rules delineated for the entities in the model to follow pertain to birth rules and death rules, which are similar to the *r* and *K* parameters used in **Equation 1**. These rules are listed as follows:

- Under the condition that a box is empty (i.e., white), if exactly three animals are present in its Moore's Neighborhood, then an animal is given birth to and occupies the empty space (i.e., it turns to black). The model assumes that two animals only mate under the protection of a third of its kind.

- Under the condition that a box is occupied by an animal (i.e., black), if less than 2 of the boxes in its Moore's Neighborhood are occupied, then the animal is too vulnerable to defend itself from outside forces (e.g., a predator) to survive. Thus, it will die (i.e., it turns to white).
- Under the condition that a box is occupied by an animal, if more than 4 of the boxes in its Moore's Neighborhood are occupied, then the animal faces the competition of resources from its own kind and dies of starvation.

Conversely, in the multivariable predator-prey CA model, the identities of the species in play become significant, and their behaviors must be encoded such as to distinguish them from each other in the simulation. In this case, cicadas larvae that mature underground in the model are represented by black boxes; cicadas adults ready for breeding are represented by blue boxes; birds that feed on the cicadas adults are represented by red boxes, and empty spaces are represented by the usual white box. **Figure 4** depicts the initial state of the predator-prey CA simulation where 50% of the boxes are empty, 40 - 45% of them are occupied by cicadas larvae, and 5 - 10% are occupied by birds.

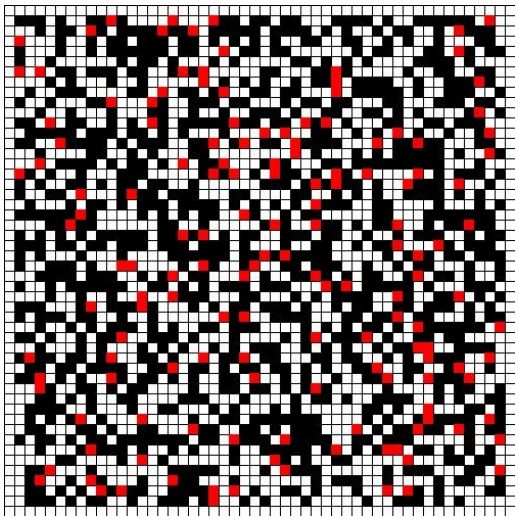


Figure 4: Initial population of predator-prey model

Many complex rules are encoded in the simulation, most of which are described in the article *Emergence of Prime Numbers as the Result of Evolutionary Strategy* by Campos et al. It is not clear from the article whether or not certain obvious facts were included in their simulation, thus additional assumptions were made for the simulations described in this report. Some of the central assumptions made in the article, as well as additional assumptions made in the simulations for this report, are listed as follows:

- Cicadas larvae (black) have an incubation period of x years and will not become an adult (blue) until after x iterations. During their incubation period, the larvae stay underground and are undetectable by birds (red).
- Cicadas adults wait above ground to breed and will not die until they meet a partner to mate with or get eaten by a bird. If either occurs, they die (white).

- Birds normally eat adult cicadas in their Moore's Neighborhood but have enough resources to give birth to another bird if there are at least 2 cicadas to feed on and another bird to mate within their Moore's Neighborhood.
- Cicadas adults will die from overpopulation if it is surrounded by 4 or more other cicadas adults. ● Neither the simulations in this report nor the ones described in *Emergence of Prime Numbers as*

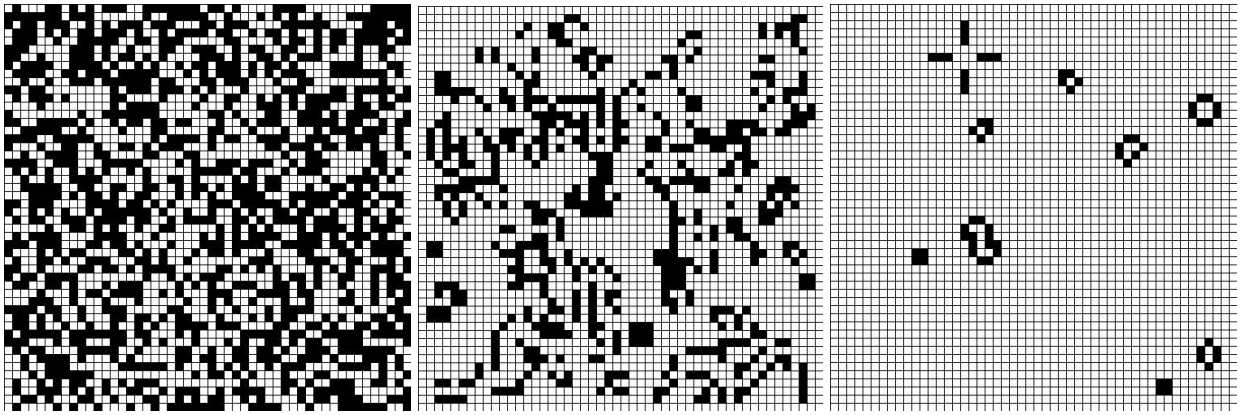
- *the Result of Evolutionary Strategy* More rules can be found by reading the aforementioned article by Campos et al assume an age limit to cicadas adults. [3] .

For the purposes of this report, an optimal survival strategy for the cicadas is one that results in the near-complete extermination of the birds. To ensure consistent modeling conditions among trials, a grid size of 50 by 50, resulting in 2500 boxes, was used. Trials in the one-dimensional population growth model start with 50% of all boxes being occupied by animals; all trials in the multivariable predator-prey model started with 50% of empty space, 45% of incubating cicadas, and 5% of birds on the grid.

In either situation, "randomness" is at the heart of the simulation. The populations in each simulation will evolve in ways that are impossible to foresee and are entirely dictated by chance. However, in many cases, the results obtained from the completely "random" simulations will align with solutions obtained from deterministic models and definitive conclusions can be drawn from each simulation.

Solutions and Results of the Mathematical Problem

Based on the assumptions made for the one-dimensional stochastic population growth model, a simulation of how the species will interact with one another can be observed as an animation, snapshots of which are presented in **Figure 5 a-c**.



(a) **(b)** **(c)**

Figure 5: (a) Initial population (b) evolving population (c) stable equilibrium

When the population reaches the state depicted in **Figure 5 c**, stable equilibria is observed, and the population will no longer increase or decrease as time progresses. This observation is analogous to the emergence of a stable fixed point when the phase diagram of **Equation 1** is graphed. Upon solving for the stable fixed point in **Equation 1**, one can conclude that the population of the animals stabilizes at $N = K$, the carrying capacity; similarly, the stochastic approach would conclude that the population of animals stabilize when an X% increase or decrease of its original population is met. Thus, it can be demonstrated that the solution to the stochastic growth model is consistent with that of the deterministic model.

A solution to the predator-prey CA model is complex and can have a wide variety of interpretations. A sample solution where the incubation period of the cicadas was 13 years based on the rules described for the model is shown in **Figure 6**.

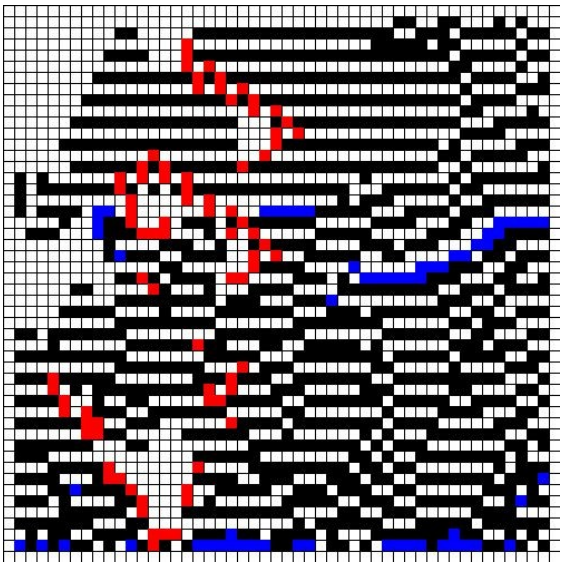


Figure 6: End state of a predator-prey simulation where the incubation period of cicadas is 13 years

As can be seen, clusters of birds form around the cicadas, presumably to meet their survival needs. As discussed in *Emergence of Prime Numbers as the Result of Evolutionary Strategy*, the solution appears to be a result of the odd nature of the cicadas' breeding cycle -- the cicadas stay under the ground for long periods of time to avoid being eaten, but when they finally emerge, they die soon after if they breed successfully, preventing themselves from falling prey to their predators. This result is, intriguingly, observed for a select range of cicadas maturation periods; other incubation periods result in the extermination of all birds, as seen in **Figure 7**. This suggests that there indeed is an optimal incubation period for the cicadas to adopt to preserve its population and eradicate its predator, but this value cannot be determined explicitly from the stochastic model.

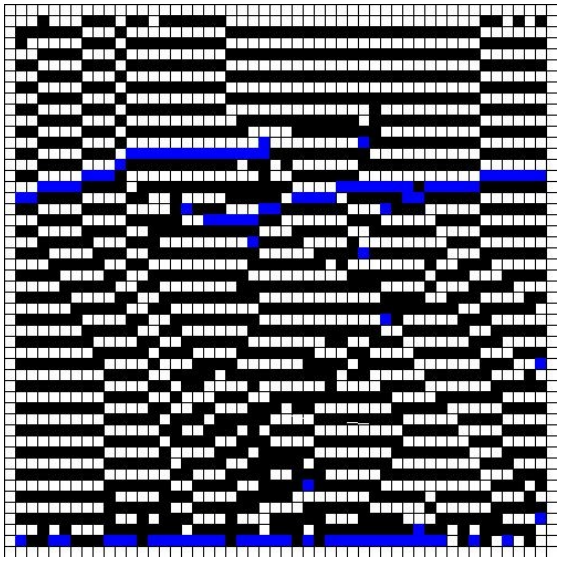


Figure 7: End state of a predator-prey simulation where the incubation period of cicadas is 14 years

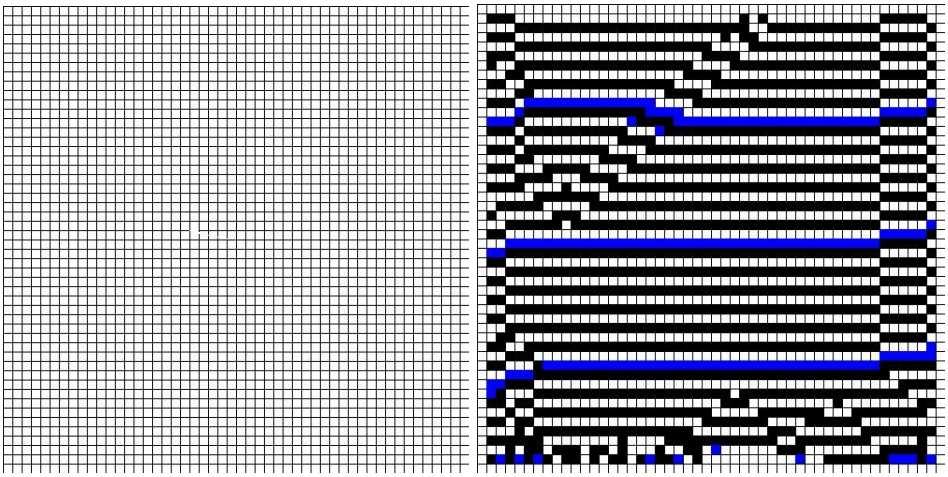
Conclusions

Several trials to the one-dimensional population growth model by CA were conducted. The model generates results that are similar to the one-dimensional continuous model -- most initial conditions tested resulted in a stable equilibrium among the population. Exceptions to this result occur when the initial condition of the population takes up less than 10% or more than 80% of the grid, in which cases the population tends towards extermination. These results can be understood as the population dying out from lack of protection or ability to procreate, or from lack of natural resources as a result of overpopulation. The former result, in fact, is more realistic to what one might expect than the conclusions drawn from the continuous model -- due to the presence of an unstable fixed point in **Equation 1**, even if the initial population of the species is $N = 1$, it will eventually tend towards the stable fixed point where $N = K$. On the other hand, the latter result obtained from the CA model is less realistic to what one would expect than that of the continuous model. In reality, when overpopulation occurs, a species is expected to seek equilibrium with the environment by means of reducing its birth rate, or having higher mortality rates. The CA model, however, suggests that the population will exterminate itself, which is a result so extreme that it may only be able to accurately model select, distinct cases.

Several trials to the multivariable predator-prey model by CA were performed with different cicadas incubation periods. Results differ from what is demonstrated in *Emergence of Prime Numbers as the Result of Evolutionary Strategy* primarily because of the aforementioned differences in assumptions made about the population. However, conclusions that align with what is observed in nature may be drawn.

Emergence of Prime Numbers as the Result of Evolutionary Strategy suggests that any incubation period that is a prime number will produce optimal survival results for the population while non-prime incubation periods will result in near-complete extermination of the cicadas population. While these results can be used to explain the 13 and 17-year incubation periods that cicadas populations adapt in nature, it fails to explain why other prime numbers are not observed. Further, some assumptions, or lack thereof, made in the article do not match up with what one might observe in nature. For instance, it is unclear whether or not death occurs to the cicadas adults in the article upon breeding, which is a distinct feature of the cicadas life-cycle.

In the simulations used for this report, cicadas are assumed to die on the following iteration after they breed. Cicadas adults and birds are also assumed to have laxer breeding conditions than those described in the article. As a result, the model built for this report demonstrate that incubation periods ranging from 1 - 6 years result in near-total extermination of both species 80% of the time or near-total extermination of only the birds 20% of the time. These results are shown in **Figure 8 a - b**.



(a) (b)

Figure 8: Simulations for an incubation time of 4 years
(a) all species exterminated (b) all birds exterminated

Due to the inconsistencies in whether or not the cicadas population will be exterminated or not, the simulations suggest that these incubation times are not optimal for the species' survival.

For incubation times ranging from 7 - 12 years, a stable equilibrium between the two species is achieved with the two populations continually declining and growing as a result of competition from each other 100% of the time. An example from an incubation time of 10 after 5000 iterations is shown in **Figure 9**.

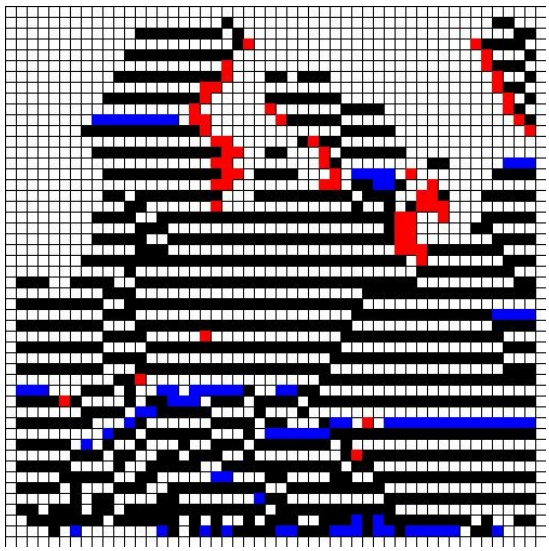


Figure 9: Equilibrium between the birds and cicadas; incubation period of 10 years

These results align with what is expected in nature for regular insect species that interact with a predator, suggesting that the incubation times play no significance in the species survival. These results are also similar to solutions that can be obtained from the Lotka-Volterra system where species are continuously growing and declining as a result of each other.

Starting at an incubation period of 13 years, an intriguing transition occurs. For 70% of the time, the simulation results in a stable equilibrium between the two species, similar to the results observed for incubation times between 7 - 12 years. However, 30% of the time the population of birds naturally decline and exterminate, resulting in the entire grid being populated by cicadas insects. With incubation times from 14 - 18 years, the population of the birds exterminate 100% of the time. Results from these simulations are shown in **Figure 10**.

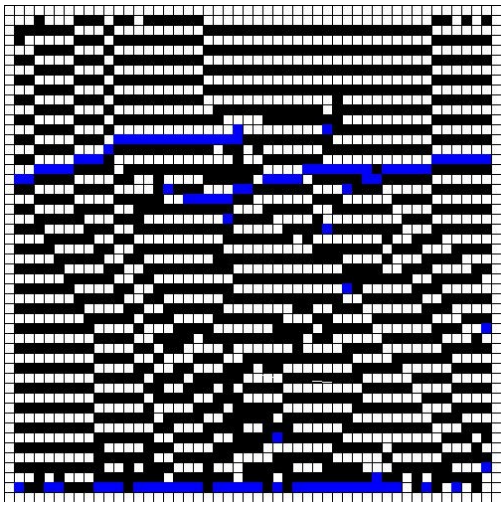


Figure 10: All birds exterminated at end state with incubation time of 14 years

The main thesis of *Emergence of Prime Numbers as the Result of Evolutionary Strategy* is to suggest that incubation times of 13 and 17 are natural products of evolution; the transition that occurs at an incubation time of 13 years in the simulations for this report suggests that an incubation time of at least 13 years is required for optimal survival results in the cicadas population. Although these results are not perfectly in-line with what is observed in nature, it provides evidence that 13 plays a mysterious role in the survival of cicadas populations.

Improvements

In comparing the continuous approach to the stochastic approach in modeling population growth, one can see that the major drawback of the stochastic system is its inability to predict intermediate results that lie between the beginning and end of the simulation. This can be improved by keeping a counter that tracks the population of species on the grid at every iteration. Further, inconsistencies in the final result among trials of the simulations are frequent, which may lead to misleading results.

There are unlimited ways in which the CA simulations can be improved, as the simulations can never adequately model what occurs in reality. In the one-dimensional population growth model by CA, in order to avoid the observation of the population exterminating itself due to overpopulation, tests should be implemented that allow a different survival strategy among the animals to avoid extinction. A few arguably arbitrarily defined parameters in the predator-prey model by CA should be examined and

modified to see if the same results can be obtained, such as the grid size and starvation limit that the birds can endure.

References

1. Vogels, Maurice, et al. “P. F. Verhulst's ‘Notice Sur La Loi Que La Populations Suit Dans Son Accroissement’ from Correspondence Mathematique Et Physique. Ghent, Vol. X, 1838.” *Journal of Biological Physics*, vol. 3, no. 4, 1975, pp. 183–192.
2. Lotka, Alfred J. J. “Contribution to the Theory of Periodic Reactions.” *Journal of Physical Chemistry*, vol. 14, no. 3, 1910, pp. 271–274.
3. Campos, Paulo R A, et al. “Emergence of Prime Numbers as the Result of Evolutionary Strategy.” *Physical Review Letters*, vol. 93, no. 9, 2004, p. 098107.

Appendix A Java code used to implement the simulations:

Main method:

```
import java.util.*;

public class CAMain { // Used for debugging

  public static void main(String[] args) {
    CellularAutomata CA = new CellularAutomata(50);
PredatorPreyCA PPCA = new PredatorPreyCA(50);
    int[][] grid = CA.getGrid();
    PPCA.graphGrid();



    animal[][] grid = PPCA.getGrid();
PPCA.graphGrid();       for (int i = 0;
i < 5000; i++)
{         PPCA.graphGrid();
PPCA.nextGeneration();
       try {
          Thread.sleep(1);
       } catch (InterruptedException e)
{            e.printStackTrace();
       }


    }
    for (int i = 0; i < 1000; i++)
{         CA.graphGrid();
CA.nextGeneration();
       try {
          Thread.sleep(10);
       } catch (InterruptedException e)
{            e.printStackTrace();
       }


    }
  }
}
```

One-dimensional population growth model by CA:

```
import java.util.*; import
java.awt.*;
public class CellularAutomata {

  private int[][] grid;
private int[][] nextGrid;
  private int size;    private Random rand
= new Random();    private
DrawingPanel panel;    private Graphics
g;
  private int cellLength;

  private static final int DEFAULT_GRID_DIMENSIONS = 10;
  private static final int PANEL_SIZE = 500;

  public CellularAutomata() {
    this(DEFAULT_GRID_DIMENSIONS);
  }     public CellularAutomata(int
size) {       grid = new int[size][size];
nextGrid = new int[size][size];
this.size = size;
    cellLength = PANEL_SIZE/size;
initializeGrid();
    panel = new DrawingPanel(PANEL_SIZE, PANEL_SIZE);
g = panel.getGraphics();
  }
  private void initializeGrid() {

    for (int j = 0; j < size; j++) {


      for (int k = 0; k < size; k++)
{           int temp =
rand.nextInt(101);            if (temp <
50) {               grid[j][k] =
1;           } else
{              grid[j][k] = 0;
          }

      }
    }
```

```
  }

  public void graphGrid()
{      int counter = 0;       for (int j
= 0; j < size; j++) {         for (int i
= 0; i < size; i++) {
          g.setColor(Color.WHITE);
          g.fillRect(i*cellLength,j*cellLength,cellLength,cellLength);
if (grid[j][i] == 0) { // white spots are dead
g.setColor(Color.BLACK);
            g.drawRect(i*cellLength,j*cellLength,cellLength,cellLength);
          } else
{              counter++;
            g.setColor(Color.BLACK); // black spots are alive
            g.fillRect(i*cellLength,j*cellLength,cellLength,cellLength);
          }

        }
    }
  }

  public void nextGeneration()
{      for (int j = 1; j < size - 1; j++)
{
        for (int i = 1; i < size - 1; i++) {
          rules(j, i);
        }
    }

    grid = nextGrid;
    nextGrid = new int[size][size];
  }

  private void rules(int j, int i)
{      int neighborhoodSum = 0;
for (int y = -1; y <= 1; y++)
{         for (int x = -1; x <= 1; x++)
{
          neighborhoodSum += grid[j+y][i+x];
        }
```

```
      }
      neighborhoodSum -= grid[j][i];

      if (grid[j][i] == 1 && neighborhoodSum < 2) {
        nextGrid[j][i] = 0;
      } else if (grid[j][i] == 1 && neighborhoodSum >
3){         nextGrid[j][i] = 0;
      } else if (grid[j][i] == 0 && neighborhoodSum == 3) {
        nextGrid[j][i] =
1;       } else {
        nextGrid[j][i] = grid[j][i];
      }
    }

    public int[][] getGrid()
{       return grid;
    }

    public int getGridDimensions()
{       return size;
    }
}
```

Multivariable predator-prey model by CA:

```
import java.util.*;
import java.awt.*;


public class PredatorPreyCA {

   private animal[][] grid;
private animal[][] nextGrid;
   private int size;
   private Random rand = new Random();
private DrawingPanel panel;     private
Graphics g;
   private int cellLength;



   private static final int DEFAULT_GRID_DIMENSIONS = 10;
private static final int PANEL_SIZE = 500;     private static final
int INITIAL_EMPTY_PERCENTAGE = 50;     private static final
int INITIAL_PREY_PERCENTAGE = 45;     private static final
int INITIAL_PREDATOR_PERCENTAGE =
          100 - INITIAL_EMPTY_PERCENTAGE - INITIAL_PREY_PERCENTAGE;
   private static final int K_PREY = 2; // parameter defined by model
private static final int K_PREDATOR = 2; // parameter defined by model

   public PredatorPreyCA() {
      this(DEFAULT_GRID_DIMENSIONS);
   }



   public PredatorPreyCA(int size)
{        grid = new animal[size][size];
nextGrid = new animal[size][size];
this.size = size;
       cellLength = PANEL_SIZE / size;
initializeGrid();
       panel = new DrawingPanel(PANEL_SIZE, PANEL_SIZE);
g = panel.getGraphics();
   }
```

```
   private void initializeGrid()
{        initializeEdges();
      for (int j = 1; j < size - 1; j++) {
         for (int k = 1; k < size - 1; k++) {
            int temp = rand.nextInt(101);
            if (temp < INITIAL_EMPTY_PERCENTAGE) {
               grid[j][k] = new Empty();
            } else if (temp < INITIAL_EMPTY_PERCENTAGE + INITIAL_PREY_PERCENTAGE)
{                  grid[j][k] = new Cicadas();
            } else {
               grid[j][k] = new Predator();
            }
         }
      }
   }

   private void initializeEdges()
{        for (int j = 0; j < size; j++)
{           grid[j][0] = new Empty();
         grid[j][size - 1] = new Empty();
      }
      for (int i = 0; i < size; i++)
{           grid[0][i] = new Empty();
         grid[size - 1][i] = new Empty();
      }
   }

   public void graphGrid()
{        int cicadasCounter = 0;
int predatorCounter = 0;        for
(int j = 0; j < size; j++) {            for
(int i = 0; i < size; i++) {
            g.setColor(Color.WHITE);
            g.fillRect(i * cellLength, j * cellLength, cellLength, cellLength);
if (grid[j][i].getID() == 0) { // white spots are dead
g.setColor(Color.BLACK);
               g.drawRect(i * cellLength, j * cellLength, cellLength, cellLength);
            } else if (grid[j][i].getID() == 1 && grid[j][i].getAge() <= grid[j][i].getLifeSpan())
{                  cicadasCounter++; // black spots are incubating prey
g.setColor(Color.BLACK);
               g.fillRect(i * cellLength, j * cellLength, cellLength, cellLength);
```

```
            } else if (grid[j][i].getID() == 1 && grid[j][i].getAge() > grid[j][i].getLifeSpan())
{                   cicadasCounter++;// blue spots are emerged prey
g.setColor(Color.BLUE);
                g.fillRect(i * cellLength, j * cellLength, cellLength,
cellLength);                } else {
                predatorCounter++;
                g.setColor(Color.RED); // red spots are predators
                g.fillRect(i * cellLength, j * cellLength, cellLength, cellLength);
            }

        }
    }
  }

  public void nextGeneration()
{       for (int j = 1; j < size - 1; j++)
{           for (int i = 1; i < size - 1; i++)
{               grid[j][i].mutate();
grid[j][i].age();               rules(j, i);
        }
    }

    grid = nextGrid;
initializeEdges();
    nextGrid = new animal[size][size];
  }

  private void rules(int j, int i) {
    if (grid[j][i].getID() == 0) { // check what would happen to the empty slot
int predatorCounter = 0;            int emergedCicadasCounter = 0;
        predatorCounter = calculatePredators(j, i, predatorCounter); // examine Moore Neighborhood
emergedCicadasCounter = calculateEmergedCicadas(j, i, emergedCicadasCounter);

        // empty rules are as follows:
        if (emergedCicadasCounter >= K_PREY) {
            nextGrid[j][i] = grid[j][i].birtheAndInherit(1); // empty cell occupied by new cicadas that
inherits properties of parent
            // after birth, Cicadas is marked for death
outerloop:
            for (int y = -1; y <= 1; y++)
{                   //boolean calledBreak =
```

```
false;                    int parentCounter = 0;
for (int x = -1; x <= 1; x++) {
                  if (x != 0 || y != 0)
{                       if (grid[j + y][i + x].getID() == 1)
{                          grid[j + y][i + x].giveBirth();
parentCounter++;                          if
(parentCounter == 2)
{                             //calledBreak = true;
break outerloop; // break since only one of the
predators will eat the prey.
                           }

                       }
                     }
                   }
                 }

          }
else {
             nextGrid[j][i] = grid[j][i];
          }
      } else if (grid[j][i].getID() == 1) { // check what would happen to the cicadas in its next iteration
if (grid[j][i].getAge() > grid[j][i].getLifeSpan()) { // cicadas is emerged and is now in a vulnerable state
if (grid[j][i].gaveBirth()) {                  nextGrid[j][i] = new Empty();
             } else {
               int predatorCounter = 0;
int emergedCicadasCounter = 0;
               predatorCounter = calculatePredators(j, i, predatorCounter); // examine Moore Neighborhood
emergedCicadasCounter = calculateEmergedCicadas(j, i, emergedCicadasCounter);              //
cicadas survival rules are as follows:
               if (predatorCounter >= K_PREDATOR) { // there are enough predators to replace the cicadas
nextGrid[j][i] = grid[j][i].birtheAndInherit(2); // cicadas turns into predator that inherits properties of
parent
               } else if (predatorCounter > 0 && predatorCounter < K_PREY) { // enough predators to eat
the cicadas
                  nextGrid[j][i] = new Empty(); // cicadas turns into empty
                  // the predator has now eaten
outerloop:
                  for (int y = -1; y <= 1; y++)
{                       //boolean calledBreak = false;
for (int x = -1; x <= 1; x++) {                       if (x != 0
```

```
|| y != 0) { // do not count self                    if
(grid[j + y][i + x].getID() == 2)
{                          grid[j + y][i + x].resetAge();
                          //calledBreak = true;                    break outerloop; // break
since only one of the predators will eat the prey.
                        }
                      }
                    }
                    //if (calledBreak) {
                    //    break;
                    //}
                  }
              } else if (emergedCicadasCounter >= 4)
{                 nextGrid[j][i] = new
Empty();              } else { // cicadas retains
original state                nextGrid[j][i] = grid[j][i];
              }
            }
          } else {
            // else case is when the cicadas is still incubating, in which state it is invincible.
            // Its state will not be changed.
            nextGrid[j][i] = grid[j][i];
          }
      } else { // check what would happen to the predator in its next iteration
          // predator survival rules
          if (grid[j][i].getAge() == grid[j][i].getLifeSpan()) { // predator dies of starvation
nextGrid[j][i] = new Empty();
          } else {
            nextGrid[j][i] = grid[j][i];
          }
      }
    }

   private int calculatePredators(int j, int i, int predatorCounter)
{       for (int y = -1; y <= 1; y++) {          for (int x = -1; x <=
1; x++) {              if (x != 0 || y != 0) { // Do not count self
if (grid[j + y][i + x].getID() == 2)
{                    predatorCounter++;
              }
            }
          }
```

```
        }
        return predatorCounter;
    }

    private int calculateEmergedCicadas(int j, int i, int emergedCicadasCounter)
{       for (int y = -1; y <= 1; y++) {             for (int x = -1; x <= 1; x++) {
            if (x != 0 || y != 0) { // Do not count self
if (grid[j + y][i + x].getID() == 1 &&
                    grid[j + y][i + x].getAge() > grid[j + y][i + x].getLifeSpan())
{                     emergedCicadasCounter++;
                }
            }
          }
        }
        return emergedCicadasCounter;
    }

    public animal[][] getGrid() {
        return grid;
    }

    public int getGridDimensions()
{       return size;
    }

}
```

```
import java.util.*;

public class animal {

   private int ID;
private int lifeSpan;
   private double mutationProbability; // whether mutation occurred should be tested at every iteration
private int age; // changes at every iteration
   //private int company;
private boolean gaveBirth;
   private Random rand = new Random();

   public animal(int ID, int lifeSpan, double mutationProbability)
{       this.ID = ID;       this.lifeSpan = lifeSpan;
this.mutationProbability = mutationProbability;       //
this.company = company; // k_prey or k_predator       this.age =
age;       this.gaveBirth = false;
   }

   public void age() {
      age++;
   }

   public void mutate() { // if mutation succeeds, lifespan will increase by 1.
      int factor = rand.nextInt((int)(1/mutationProbability + 1));       lifeSpan +=
factor*mutationProbability; // if succeeds in mutation, factor*mutationProbability = 1; otherwise = 0.
   }

   public int getID()
{       return ID;
   }

   public int getLifeSpan()
{       return lifeSpan;
   }

   public double getMutationProbability() {
      return mutationProbability;
   }
```

```
import java.util.*;
import java.awt.*;

   public int getAge()
{        return age;
   }

   public void resetAge() { age = 0; }

   public boolean gaveBirth() {
     return gaveBirth;
   }

   public void giveBirth() {
      gaveBirth = true;
   }

   public animal birtheAndInherit(int ID) { // does not inherit lifespan yet!!!!
      if (ID == 1) {

         return new
Cicadas();        } else if (ID
== 2) {           return new
Predator();
      } else {
         return new Empty();
      }
   }

}

public class Cicadas extends animal {

   private static final int CICADAS_ID = 1;     private
static final int INCUBATION_PERIOD = 13;
   private static final double MUTATION_PROBABILITY = 0.00001;

   public Cicadas() {
      super(CICADAS_ID, INCUBATION_PERIOD, MUTATION_PROBABILITY);
   }
```

```
    @Override     public
String toString() {
        return "Cicadas";
    }

    public boolean isEmerged() {
        return getAge() > getLifeSpan();
    }
}
```

```
import java.util.*;
import java.awt.*;

public class Empty extends animal {

   private static final int EMPTY_ID = 0;
   private static final int EMPTY_LIFESPAN = Integer.MAX_VALUE;
private static final double MUTATION_PROBABILITY = 0.0; // irrelevant
   //private static final int COMPANY = -1; // irrelevant

   public Empty() {
      super(EMPTY_ID, EMPTY_LIFESPAN, MUTATION_PROBABILITY);
   }

   @Override     public
String toString() {
      return "Empty";
   }
}
```

```
import java.util.*;
import java.awt.*;

public class Predator extends animal {

   private static final int PREDATOR_ID = 2;
private static final int STARVATION_LIMIT = 15;
   private static final double MUTATION_PROBABILITY = 0.00001;
   // private static final int COMPANY = 4;

   public Predator() {
      super(PREDATOR_ID, STARVATION_LIMIT, MUTATION_PROBABILITY);
   }

   @Override     public
String toString() {
      return "Predator";
   }

   public void eat()
{       resetAge();
   }
}
```

```
import java.util.*;
import java.awt.*;
```