331 – Intro to Intelligent Systems
Week10b
Genetic Algorithms

T.J. Borrelli

Genetic Algorithms

- Genetic techniques can be applied with a range of representations
- The best representation depends on the problem to be solved
- Usually, we start with a hypothesis consisting of a population of chromosomes (problem states)
- Each chromosome consists of a number of genes (the features that make up a state)

Basic Genetic Algorithm

Generate a random population of chromosomes (the first generation).

If termination criteria are satisfied, stop. Otherwise, continue with step 3.

Determine the fitness of each chromosome.

Apply crossover and mutation to selected genes from the current generation of chromosomes to generate a new population of chromosomes (the next generation).

Return to step 2.

Fitness

- Fitness is an important concept in genetic algorithms
- The fitness of a chromosome determines how likely it is that it will reproduce
- Fitness is usually measured in terms of how well the chromosome solves some goal problem
  - For example, if the genetic algorithm is to be used to sort numbers, then the fitness of a chromosome will be determined by how close to a correct sorting it produces
- Fitness can also be subjective (aesthetic)

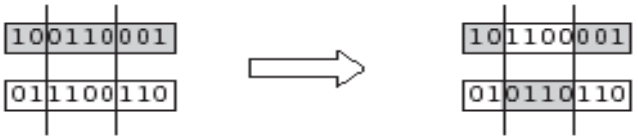
Crossover

Crossover is applied as follows:

- Select a random crossover point.
- Break each chromosome into two parts, splitting at the crossover point.
- Recombine the broken chromosomes by combining the front of one with the back of the other, and vice versa, to produce two new chromosomes.

Crossover

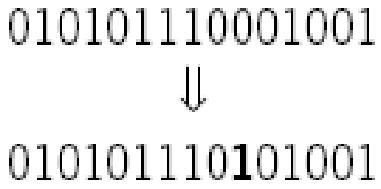
- Usually, crossover is applied with one crossover point, but can be applied with more, such as in the following case which has two crossover points:



- Uniform crossover involves using a probability to select which genes to crossover

Mutation

- A mutation is a unary operator (it applies to only one gene)
- A mutation randomly selects some genes (bits) to be “flipped”



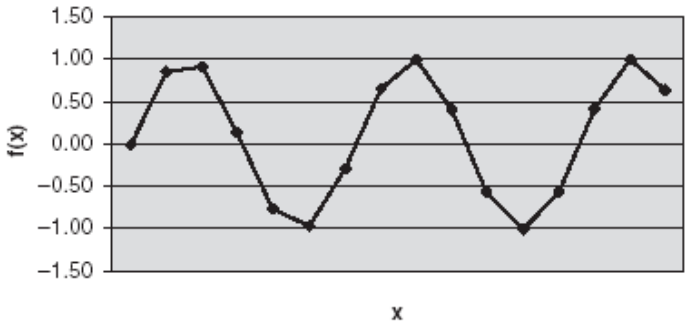
- Mutation is usually applied with a low probability, such as 1 in 1000

Termination Criteria

- A genetic algorithm is run over a number of generations until the termination criteria are reached
- Typical termination criteria are:
  - Stop after a fixed number of generations
  - Stop when a chromosome reaches a specified fitness level
  - Stop when a chromosome succeeds in solving the problem, within a specified tolerance
- Human judgment can also be used in subjective cases

Optimizing a Mathematical Function Using a Genetic Algorithm

A genetic algorithm can be used to find the maximum value for *x* in *f(x) = sin (x)*, over the range of *x* from 0 to 15, where *x* is in radians.



We will use a population size of 4 chromosomes. The first step is to generate a random population, which is our first generation:

c1 = 1001
c2 = 0011
c3 = 1010
c4 = 0101

Optimizing a Mathematical Function Using a Genetic Algorithm

To calculate the fitness of a chromosome, we need to first convert it to a decimal integer, and then calculate *f(x)* for this integer.

We will assign fitness as a numeric value from 0 to 100, where 0 is the least fit and 100 is the most fit.

f(x) generates real numbers between -1 and 1. We will assign a fitness of 100 to *f(x)* = 1 and a fitness of 0 to *f(x)* = -1. Fitness of 50 will be assigned to *f(x)* = 0. Hence, the fitness of *x*, *f'(x)*, is defined as follows:

$$f'(x) = 50\,(f(x) + 1) = 50\,(\sin(x) + 1)$$

Optimizing a Mathematical Function Using a Genetic Algorithm

The fitness ratio of a chromosome is that chromosome's fitness as a percentage of the total fitness of the population. This table shows the calculations that are used to find the fitness values for the first generation:

Chromosome	Genes	Integer Value	$f(x)$	Fitness	Fitness Ratio
c1	1001	9	0.41	70.61	46.30%
c2	0011	3	0.14	57.06	37.40%
c3	1010	10	-0.54	22.80	14.90%
c4	0101	5	-0.96	2.05	1.34%

Optimizing a Mathematical Function Using a Genetic Algorithm

Now we need to run a single step of the genetic algorithm to produce the next generation. The first step is to select which chromosomes will reproduce.

Roulette-wheel selection involves using the fitness ratio to randomly select chromosomes to reproduce. This is done as follows: The range of real numbers from 0 to 100 is divided up between the chromosomes proportionally to each chromosome's fitness. Hence, in the first generation, c1 has 46.3% of the range (i.e., from 0 to 46.3), c2 will have 37.4% of the range (i.e., from 46.4 to 83.7), and so on.

Optimizing a Mathematical Function Using a Genetic Algorithm

A random number is now generated between 0 and 100. This number will fall in the range of one of the chromosomes, and this chromosome will be selected for reproduction. The next random number is used to select this chromosome's mate.

This technique helps to ensure that populations do not stagnate, due to constantly breeding from the same parents.

We will generate 4 random numbers to find the four parents that will produce the next generation. Suppose the first random number is 56.7 (c2 is chosen), and next number is 38.2 (c1 is chosen).

Optimizing a Mathematical Function Using a Genetic Algorithm

We will now combine c1 and c2 to produce 2 new offspring. First, we randomly select the crossover point to be between the second and third genes (bits):

10 | 01

00 | 11

Crossover is now applied to produce 2 offspring, c5 and c6:

c5 = 1011

c6 = 0001

In a similar way, c1 and c3 are chosen to produce offspring c7 and c8, using a crossover point between the third and fourth bits:

c7 = 1000

c8 = 1011

Optimizing a Mathematical Function Using a Genetic Algorithm

The population c1 to c4 is now replaced by the second generation, c5 to c8 (c4 did not have a chance to reproduce, so its genes will be lost "survival of the fittest"). The fitness values for the second generation are below:

Chromosome	Genes	Integer Value	$f(x)$	Fitness	Fitness Ratio
c5	1011	11	-1	0.00	0.00%
c6	0001	1	0.84	92.07	48.10%
c7	1000	8	0.99	99.47	51.90%
c8	1011	11	-1	0.00	0.00%

At this point, c7 has been found to be the optimal solution, and the termination criteria would probably determine that the run could stop. (We could set a threshold value for fitness that must be exceeded.)

The Building Block Hypothesis

- Genetic algorithms manipulate short, low-order, high fitness schemata in order to find optimal solutions to problems
- These short, low-order, high fitness schemata are known as “building blocks”
- Hence genetic algorithms work well when small groups of genes represent useful features in the chromosomes
- This tells us that it is important to choose a correct representation!

Representations for GAs

- The representation selected must support the genetic operators
- Sometimes the bit-level representation is most natural
  - Crossover and mutation can be used to directly produce potential solutions
- What about using a GA to find potential solutions for the Traveling Salesperson problem?
  - Mutations and crossovers must preserve the property that the offspring have to be *legal paths* through all the cities, visiting each *only once*
  - How can a “good path” be passed on to a future generation?

Representations for GAs

- What do we mean by the “naturalness” of a representation?
- Example: Suppose we want our genetic operators to be able to order the numbers 6, 7, 8, and 9
- An integer representation gives a very natural and evenly spaced ordering
  - Within base 10 integers, the next item is simply one more than the previous
  - What if we used binary numbers as the representation?

Representations for GAs

Consider the bit pattern for 6, 7, 8, and 9:
0110 0111 1000 1001

Between 6 and 7, and between 8 and 9 there is a 1 bit change. Between 7 and 8 all four bits change! This representational anomaly can be a huge problem when trying to generate a solution that requires organizing these four bit patterns.

A number of techniques, usually under the general heading of *gray coding*, address this problem of non-uniform representation. In a gray code, each number is exactly one bit different from its neighbors. Using gray coding instead of standard binary numbers, the genetic operator's transitions between states of near neighbors is natural and smooth.

Gray Codes

The gray-coded bit patterns for the binary numbers 0 – 15

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Properties of Genetic Algorithms

- An important strength of GAs is the parallel nature of search
  - GAs implement a powerful form of hill-climbing that maintains multiple solutions, eliminates the unpromising, and improves good solutions
  - Initially, the solutions are scattered through the space of possible solutions; after several generations they tend to cluster around areas of higher solution quality

Properties of Genetic Algorithms

- GAs, unlike sequential forms of hill-climbing, do not immediately discard unpromising solutions
  - Even weak solutions may contribute to future candidates
- There is no strict ordering of states on an open list, as we saw with A*, DFS, or BFS
  - Rather, there is simply a population (family) of fit solutions to a problem, each potentially available to help produce new possible solutions within a paradigm of parallel search

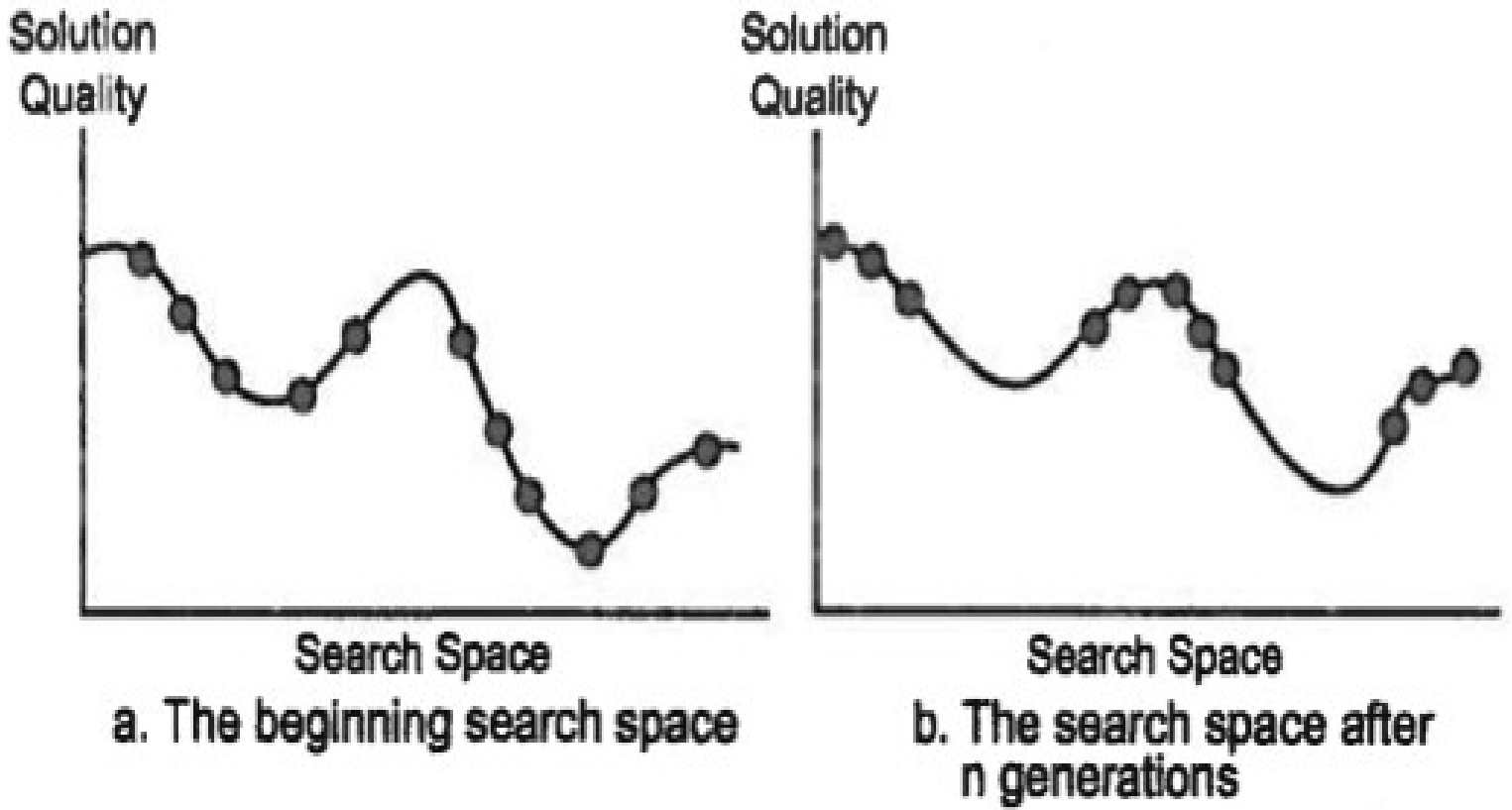


Figure 12.2 Genetic algorithms visualized as parallel hill climbing, adapted from Holland (1986).

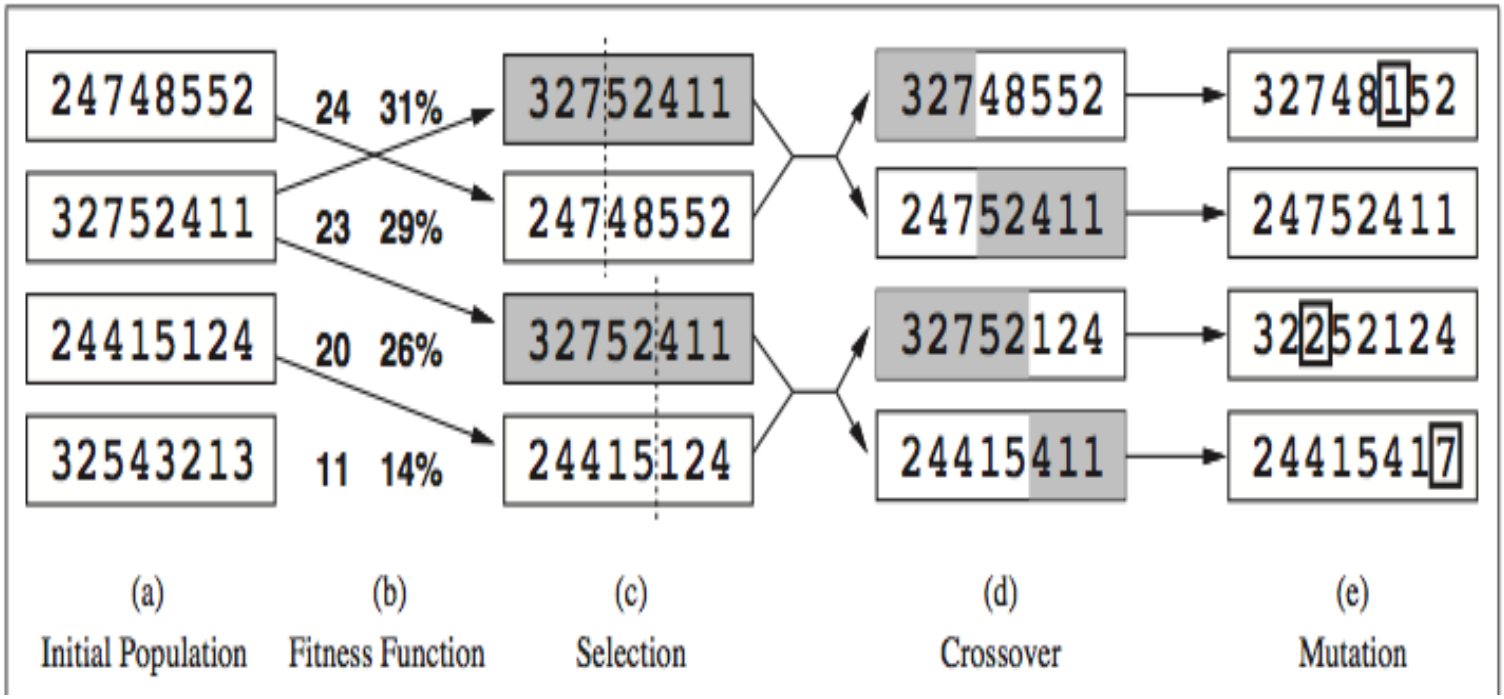
Properties of Genetic Algorithms

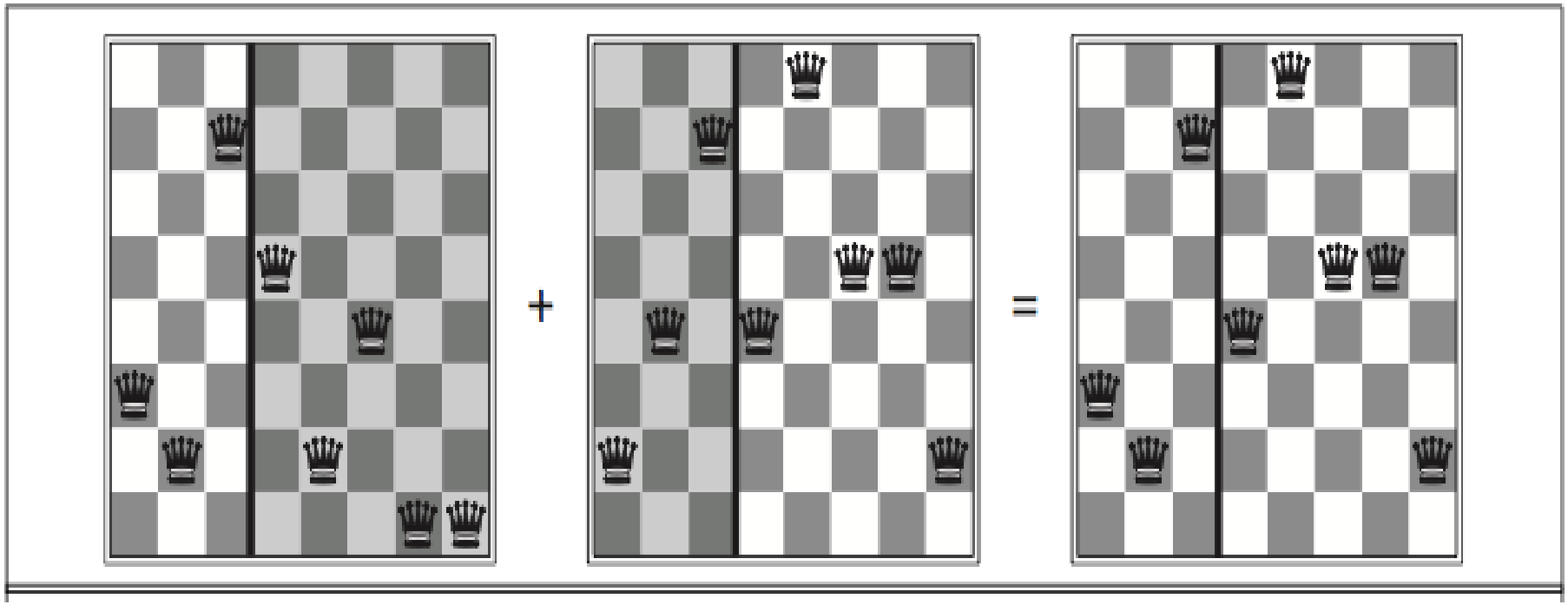
- An important strength of GAs is in the parallel nature of its search

- GAs implement a powerful form of hill climbing that maintains multiple solutions, eliminates the unpromising, and improves on good solutions

Schema

- Often used by GAs to solve problems
- A **schema** is a substring in which some of the positions can be left unspecified
- For example, considering the 8-queens problem, the string 246***** describes all 8-queens states in which the first three queens are in positions 2, 4 and 6, respectively
- Strings that match the scheme (e.g. 24613578) are called *instances* of the schema


24748552
32752411
24415124
32543213
24 31%
23 29%
20 26%
11 14%
32752411
24748552
32752411
24415124
32748552
24752411
32752124
24415411
32748152
24752411
32252124
24415417
(a)
Initial Population
(b)
Fitness Function
(c)
Selection
(d)
Crossover
(e)
Mutation



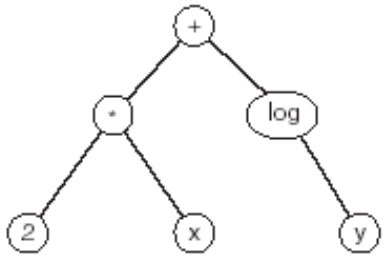
Schema

- It can be shown that if the average fitness of the instances of a schema is above the mean fitness, then the number of instances of the schema within a population will grow over time

- GAs work best when schemata correspond to meaningful components of a solution

Genetic Programming

- Genetic programming is a method used to evolve LISP S-expressions
- The S-expressions are represented as trees
- This diagram below shows an example of a tree representation of the S-expression (+(*2x) (log y))
- A random set of expressions is generated, and the “fittest” individuals reproduce to produce the next generation

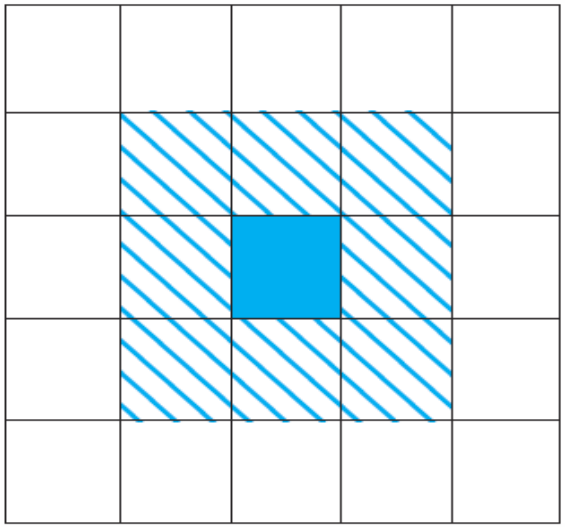


What is Life?

- What are the defining features of life?
  - Self-reproduction
  - Ability to evolve by Darwinian natural selection
  - Response to stimuli
  - Ability to die
  - Growth or expansion
- Not all living things obey these rules, and some things that are not alive do!
- Defining life is very difficult!

Conway's Life

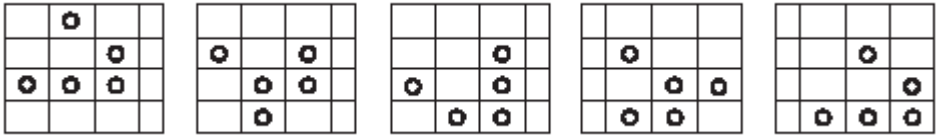
- Conway's Life is a game modeled on a two dimensional cellular automaton (grid of cells), where each cell can be alive or dead
- The cross-hatched cells indicate the set of neighbors for the center shaded cell (8-neighborhood)



Conway's Life

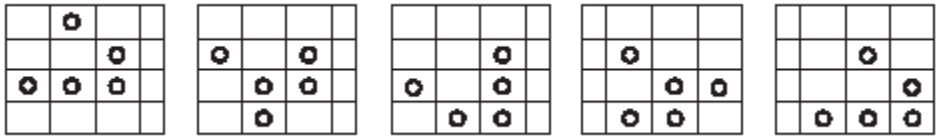
- A set of rules determines how the cells will change from one generation to the next:

  1. A dead cell will come to life if it has three living neighbors
  2. A living cell with two or three living neighbors, will stay alive
  3. A living cell with fewer than two living neighbors will die of loneliness
  4. A living cell with more than three living neighbors will die of overcrowding



Conway's Life

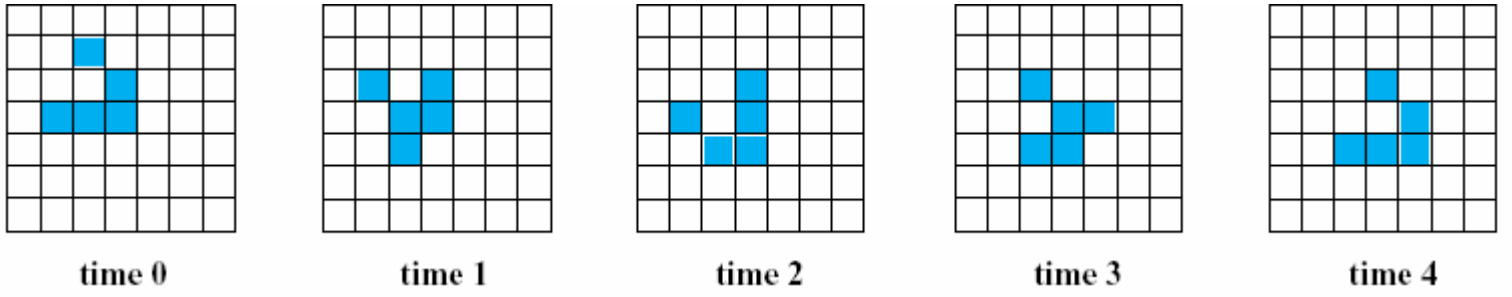
- Surprisingly complex "life-like" behavior can emerge from these simple rules
- This diagram shows a successive sequence of generations of Conway's Life



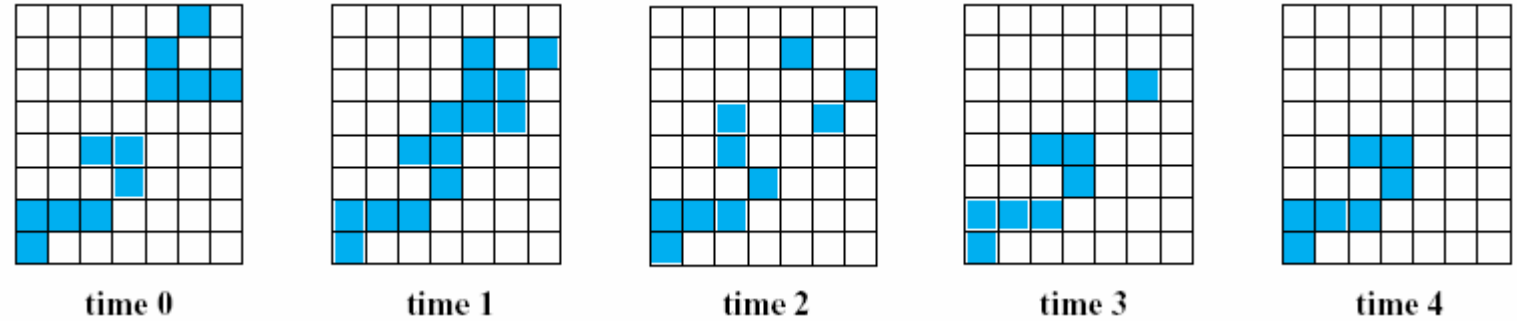
- This pattern is known as a "glider"
- There is also a pattern known as a "glider gun" which constantly fires out gliders

Conway's Life

A glider moves across the display:



A glider is consumed by another entity:



Emergent Behavior

- Emergent behavior is the idea that complex behavior can emerge from simple rules
- This is particularly prevalent in systems based on evolutionary methods, such as genetic algorithms
- Example:
  - Boids – simulation of flocking behavior of birds given very simple rules about how birds fly
  - The simulated flock learns to fly in such a way as to avoid large obstacles without being taught explicitly how to do so

Self-Reproducing Systems

- Von Neumann proposed a self-reproducing system based on cellular automata
- Ultimately we may have robots that can obtain the raw materials necessary to produce new versions of themselves
- This would be useful for exploring other planets, among other things
- Consider an “intelligence explosion” and “The Singularity”