Genetic Algorithms for Programmable Matter

Giovanni Viglietta

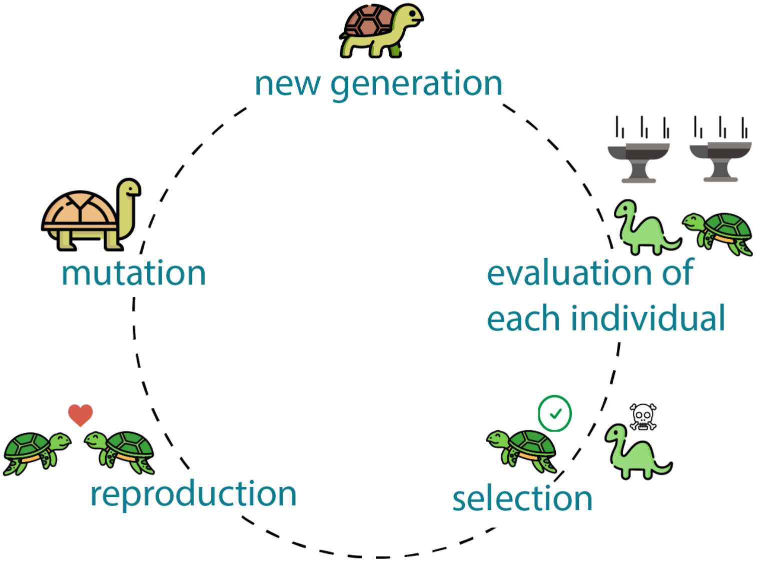
Joint work with Giuseppe A. Di Luna

(Work in progress...)

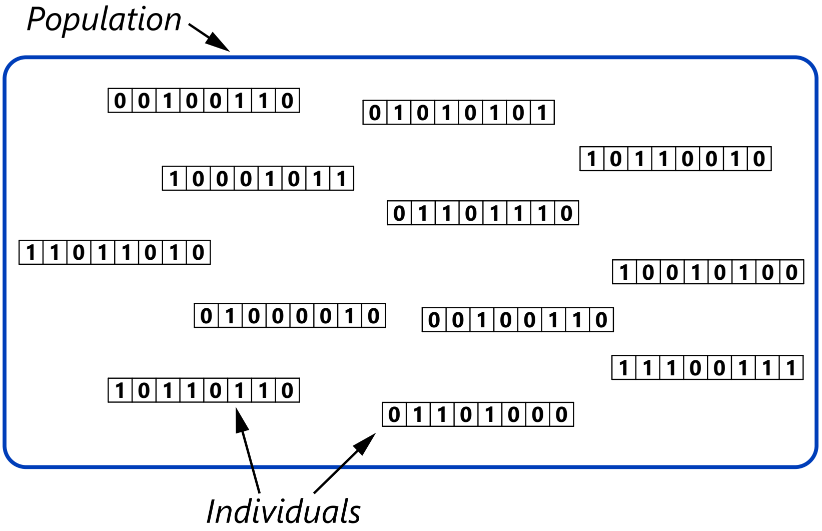
JAIST – April 27, 2022

Overview

Natural Selection and Evolution in the Real World

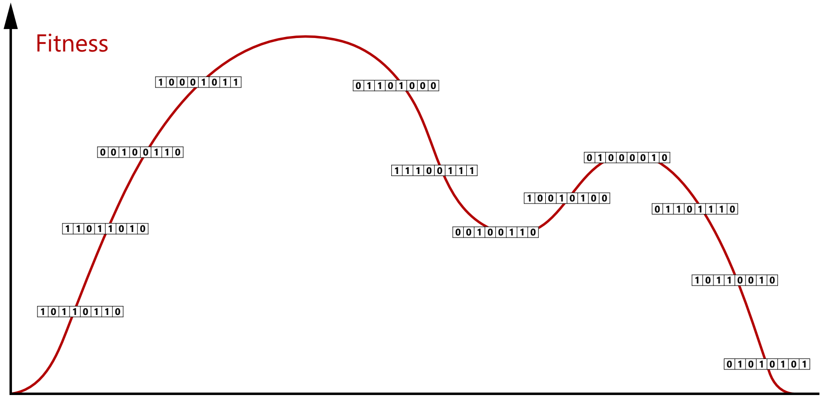


Genetic Algorithms



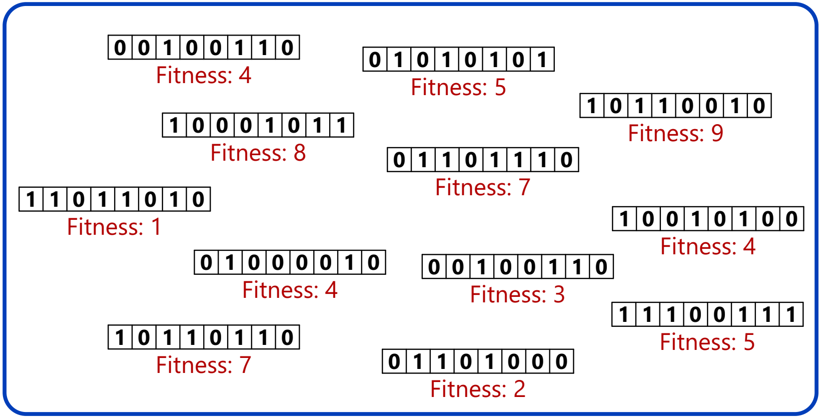
Genetic Algorithms attempt to solve optimization problems by evolving a "population" of feasible solutions.

Genetic Algorithms



Individuals may be represented by binary strings that encode solutions. Each individual has a “fitness value” given by the object function that we want to minimize or maximize.

Genetic Algorithms



The population evolves based on natural selection and genetics, where binary strings are treated as DNA sequences.
The individuals with higher fitness are more likely to reproduce and carry their genes over to the next generation.

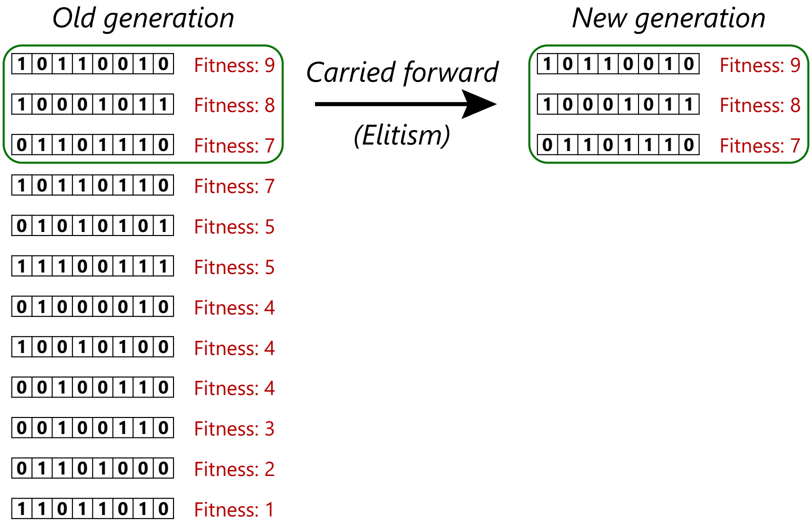
Genetic Algorithms

Old generation

1 0 1 1 0 0 1 0	Fitness: 9
1 0 0 0 1 0 1 1	Fitness: 8
0 1 1 0 1 1 1 0	Fitness: 7
1 0 1 1 0 1 1 0	Fitness: 7
0 1 0 1 0 1 0 1	Fitness: 5
1 1 1 0 0 1 1 1	Fitness: 5
0 1 0 0 0 0 1 0	Fitness: 4
1 0 0 1 0 1 0 0	Fitness: 4
0 0 1 0 0 1 1 0	Fitness: 4
0 0 1 0 0 1 1 0	Fitness: 3
0 1 1 0 1 0 0 0	Fitness: 2
1 1 0 1 1 0 1 0	Fitness: 1

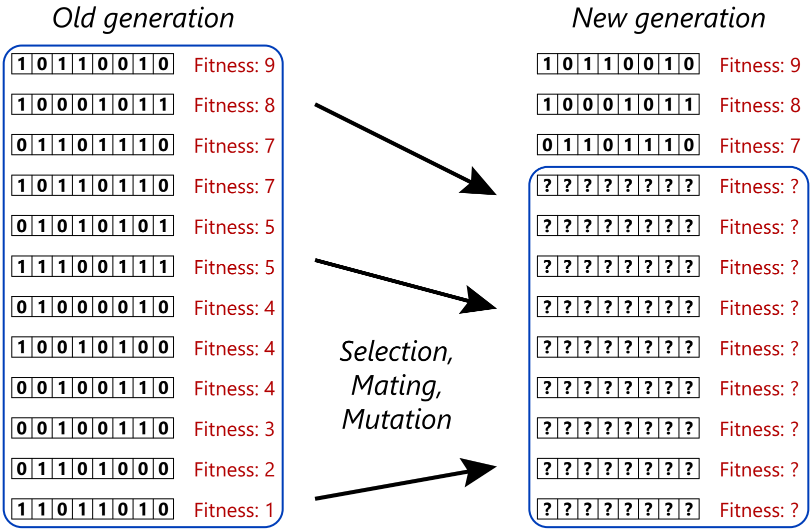
All individuals are sorted according to their fitness.

Genetic Algorithms


Old generation
1 0 1 1 0 0 1 0 Fitness: 9
1 0 0 0 1 0 1 1 Fitness: 8
0 1 1 0 1 1 1 0 Fitness: 7
1 0 1 1 0 1 1 0 Fitness: 7
0 1 0 1 0 1 0 1 Fitness: 5
1 1 1 0 0 1 1 1 Fitness: 5
0 1 0 0 0 0 1 0 Fitness: 4
1 0 0 1 0 1 0 0 Fitness: 4
0 0 1 0 0 1 1 0 Fitness: 4
0 0 1 0 0 1 1 0 Fitness: 3
0 1 1 0 1 0 0 0 Fitness: 2
1 1 0 1 1 0 1 0 Fitness: 1
Carried forward
(Elitism)
New generation
1 0 1 1 0 0 1 0 Fitness: 9
1 0 0 0 1 0 1 1 Fitness: 8
0 1 1 0 1 1 1 0 Fitness: 7

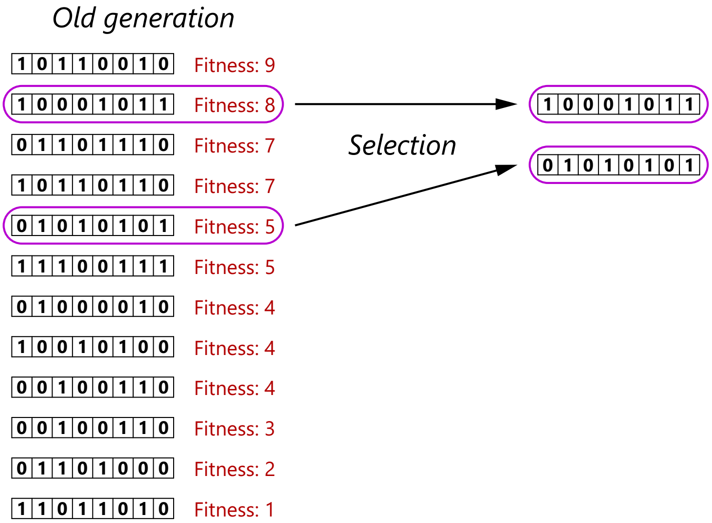
The individuals with highest fitness automatically survive.

Genetic Algorithms


Old generation
1 0 1 1 0 0 1 0 Fitness: 9
1 0 0 0 1 0 1 1 Fitness: 8
0 1 1 0 1 1 1 0 Fitness: 7
1 0 1 1 0 1 1 0 Fitness: 7
0 1 0 1 0 1 0 1 Fitness: 5
1 1 1 0 0 1 1 1 Fitness: 5
0 1 0 0 0 0 1 0 Fitness: 4
1 0 0 1 0 1 0 0 Fitness: 4
0 0 1 0 0 1 1 0 Fitness: 4
0 0 1 0 0 1 1 0 Fitness: 3
0 1 1 0 1 0 0 0 Fitness: 2
1 1 0 1 1 0 1 0 Fitness: 1
Selection,
Mating,
Mutation
New generation
1 0 1 1 0 0 1 0 Fitness: 9
1 0 0 0 1 0 1 1 Fitness: 8
0 1 1 0 1 1 1 0 Fitness: 7
? ? ? ? ? ? ? ? Fitness: ?
? ? ? ? ? ? ? ? Fitness: ?
? ? ? ? ? ? ? ? Fitness: ?
? ? ? ? ? ? ? ? Fitness: ?
? ? ? ? ? ? ? ? Fitness: ?
? ? ? ? ? ? ? ? Fitness: ?
? ? ? ? ? ? ? ? Fitness: ?
? ? ? ? ? ? ? ? Fitness: ?
? ? ? ? ? ? ? ? Fitness: ?

The other individuals die and are replaced by their children.

Genetic Algorithms

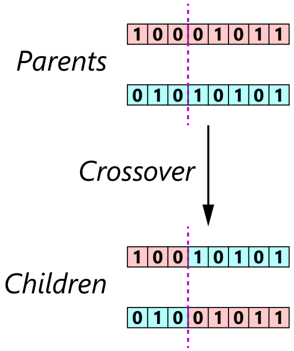

Old generation
1 0 1 1 0 0 1 0 Fitness: 9
1 0 0 0 1 0 1 1 Fitness: 8
0 1 1 0 1 1 1 0 Fitness: 7
1 0 1 1 0 1 1 0 Fitness: 7
0 1 0 1 0 1 0 1 Fitness: 5
1 1 1 0 0 1 1 1 Fitness: 5
0 1 0 0 0 0 1 0 Fitness: 4
1 0 0 1 0 1 0 0 Fitness: 4
0 0 1 0 0 1 1 0 Fitness: 4
0 0 1 0 0 1 1 0 Fitness: 3
0 1 1 0 1 0 0 0 Fitness: 2
1 1 0 1 1 0 1 0 Fitness: 1
Selection
1 0 0 0 1 0 1 1
0 1 0 1 0 1 0 1

Pairs of individuals are randomly chosen based on their fitness.

Genetic Algorithms

Old generation

1 0 1 1 0 0 1 0	Fitness: 9
1 0 0 0 1 0 1 1	Fitness: 8
0 1 1 0 1 1 1 0	Fitness: 7
1 0 1 1 0 1 1 0	Fitness: 7
0 1 0 1 0 1 0 1	Fitness: 5
1 1 1 0 0 1 1 1	Fitness: 5
0 1 0 0 0 0 1 0	Fitness: 4
1 0 0 1 0 1 0 0	Fitness: 4
0 0 1 0 0 1 1 0	Fitness: 4
0 0 1 0 0 1 1 0	Fitness: 3
0 1 1 0 1 0 0 0	Fitness: 2
1 1 0 1 1 0 1 0	Fitness: 1

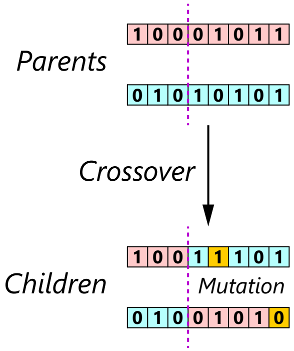


The DNAs of each pair are combined to produce two children.

Genetic Algorithms

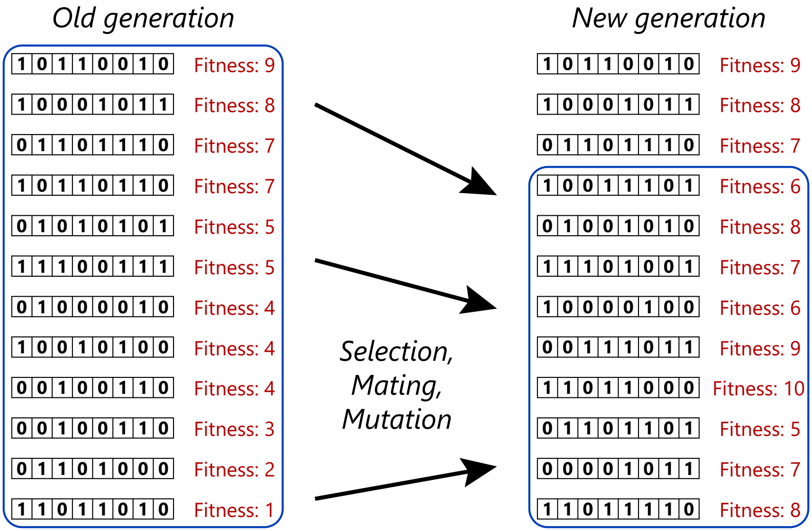
Old generation

1 0 1 1 0 0 1 0	Fitness: 9
1 0 0 0 1 0 1 1	Fitness: 8
0 1 1 0 1 1 1 0	Fitness: 7
1 0 1 1 0 1 1 0	Fitness: 7
0 1 0 1 0 1 0 1	Fitness: 5
1 1 1 0 0 1 1 1	Fitness: 5
0 1 0 0 0 0 1 0	Fitness: 4
1 0 0 1 0 1 0 0	Fitness: 4
0 0 1 0 0 1 1 0	Fitness: 4
0 0 1 0 0 1 1 0	Fitness: 3
0 1 1 0 1 0 0 0	Fitness: 2
1 1 0 1 1 0 1 0	Fitness: 1



Some random bits may be flipped to simulate genetic mutation.

Genetic Algorithms



Old generation	
1 0 1 1 0 0 1 0	Fitness: 9
1 0 0 0 1 0 1 1	Fitness: 8
0 1 1 0 1 1 1 0	Fitness: 7
1 0 1 1 0 1 1 0	Fitness: 7
0 1 0 1 0 1 0 1	Fitness: 5
1 1 1 0 0 1 1 1	Fitness: 5
0 1 0 0 0 0 1 0	Fitness: 4
1 0 0 1 0 1 0 0	Fitness: 4
0 0 1 0 0 1 1 0	Fitness: 4
0 0 1 0 0 1 1 0	Fitness: 3
0 1 1 0 1 0 0 0	Fitness: 2
1 1 0 1 1 0 1 0	Fitness: 1

New generation	
1 0 1 1 0 0 1 0	Fitness: 9
1 0 0 0 1 0 1 1	Fitness: 8
0 1 1 0 1 1 1 0	Fitness: 7
1 0 0 1 1 1 0 1	Fitness: 6
0 1 0 0 1 0 1 0	Fitness: 8
1 1 1 0 1 0 0 1	Fitness: 7
1 0 0 0 0 1 0 0	Fitness: 6
0 0 1 1 1 0 1 1	Fitness: 9
1 1 0 1 1 0 0 0	Fitness: 10
0 1 1 0 1 1 0 1	Fitness: 5
0 0 0 0 1 0 1 1	Fitness: 7
1 1 0 1 1 1 1 0	Fitness: 8

The new individuals are born, and their fitness is computed.

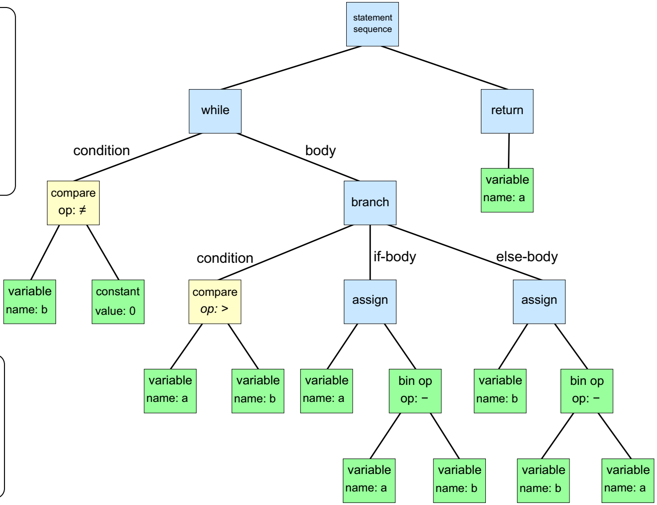
Genetic Algorithms

Old generation		*New generation*	
1 0 1 1 0 0 1 0	Fitness: 9	1 1 0 1 1 0 0 0	Fitness: 10
1 0 0 0 1 0 1 1	Fitness: 8	1 0 1 1 0 0 1 0	Fitness: 9
0 1 1 0 1 1 1 0	Fitness: 7	0 0 1 1 1 0 1 1	Fitness: 9
1 0 1 1 0 1 1 0	Fitness: 7	1 0 0 0 1 0 1 1	Fitness: 8
0 1 0 1 0 1 0 1	Fitness: 5	0 1 0 0 1 0 1 0	Fitness: 8
1 1 1 0 0 1 1 1	Fitness: 5	1 1 0 1 1 1 1 0	Fitness: 8
0 1 0 0 0 0 1 0	Fitness: 4	0 1 1 0 1 1 1 0	Fitness: 7
1 0 0 1 0 1 0 0	Fitness: 4	1 1 1 0 1 0 0 1	Fitness: 7
0 0 1 0 0 1 1 0	Fitness: 4	0 0 0 0 1 0 1 1	Fitness: 7
0 0 1 0 0 1 1 0	Fitness: 3	1 0 0 1 1 1 0 1	Fitness: 6
0 1 1 0 1 0 0 0	Fitness: 2	1 0 0 0 0 1 0 0	Fitness: 6
1 1 0 1 1 0 1 0	Fitness: 1	0 1 1 0 1 1 0 1	Fitness: 5

New generations are expected to have higher fitness than the old.

Abstract Syntax Trees

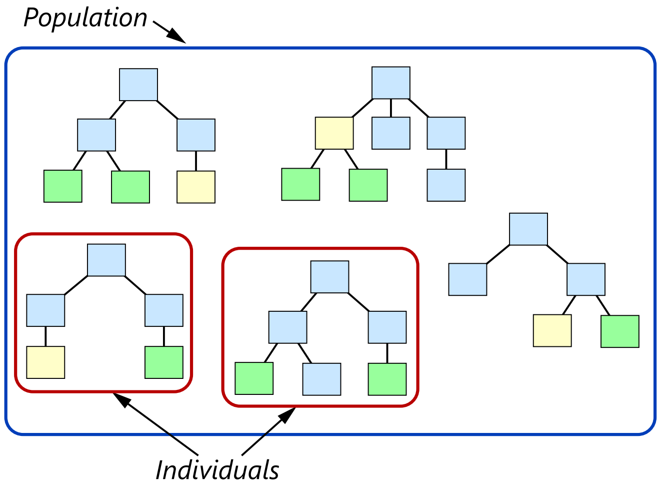
```
while b ≠ 0:
  if a > b:
    a := a - b
  else:
    b := b - a
return a
```



🟦	Instruction
🟩	Integer
🟨	Boolean

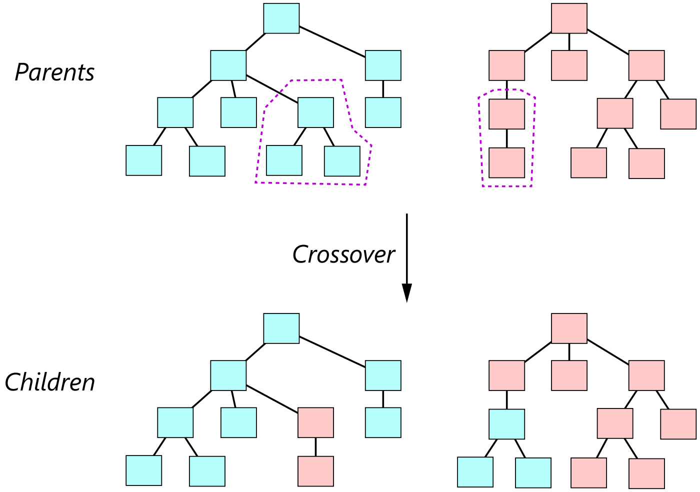
An Abstract Syntax Tree (AST) is a representation of the logical structure of a program. Each node has a type.

Genetic Programming



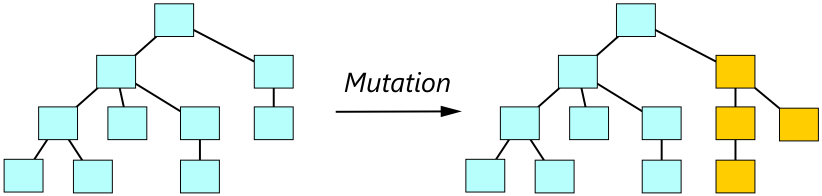
Genetic Programming is an extension of Genetic Algorithms where individuals are ASTs. The goal of Genetic Programming is to find a “good” program that solves a given problem.

Genetic Programming



When mating, the two parents' ASTs are combined by switching some randomly selected subtrees (having same-type roots).

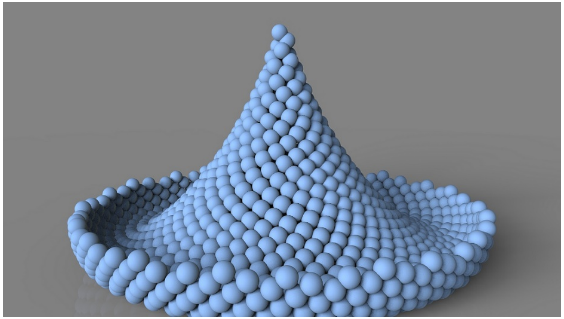
Genetic Programming

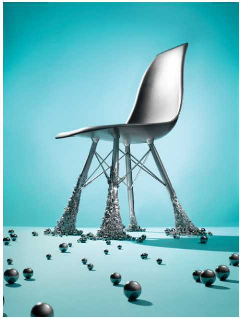


Mutation is done by replacing a randomly selected subtree with a randomly generated (well-formed) AST.

Programmable Matter

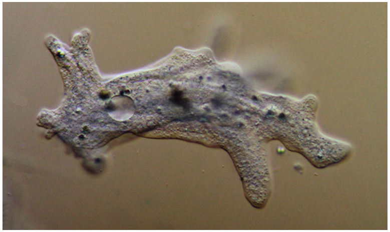
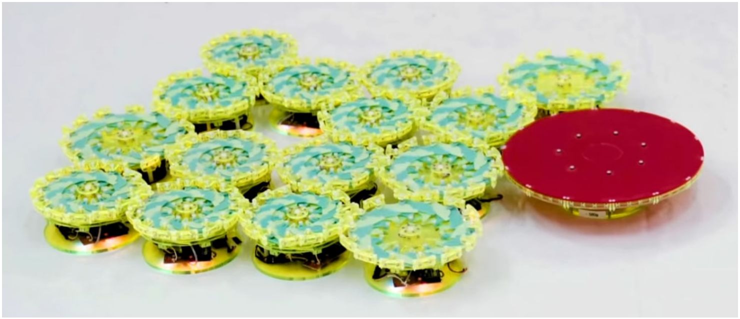
By “Programmable Matter” we mean a material (consisting of many nano-scale particles) that can change its physical properties based on autonomous sensing or user input.





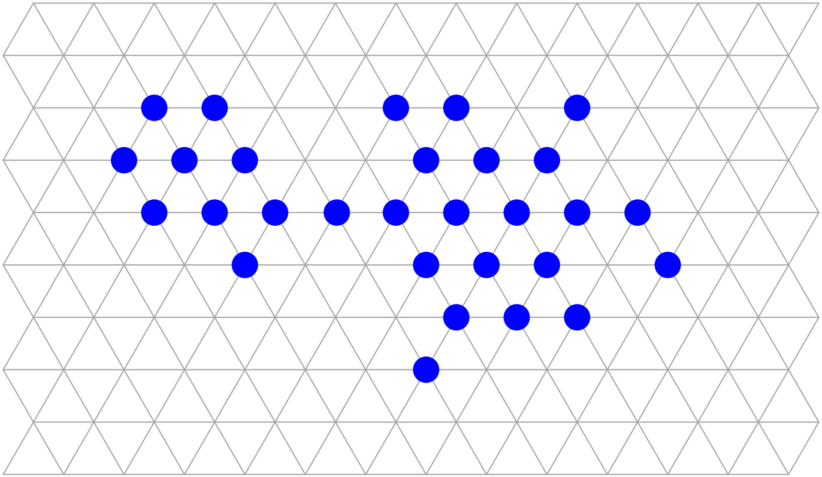
Futuristic applications include smart materials, autonomous monitoring and repair, minimal invasive surgery, etc.

Programmable Matter: Physical Models

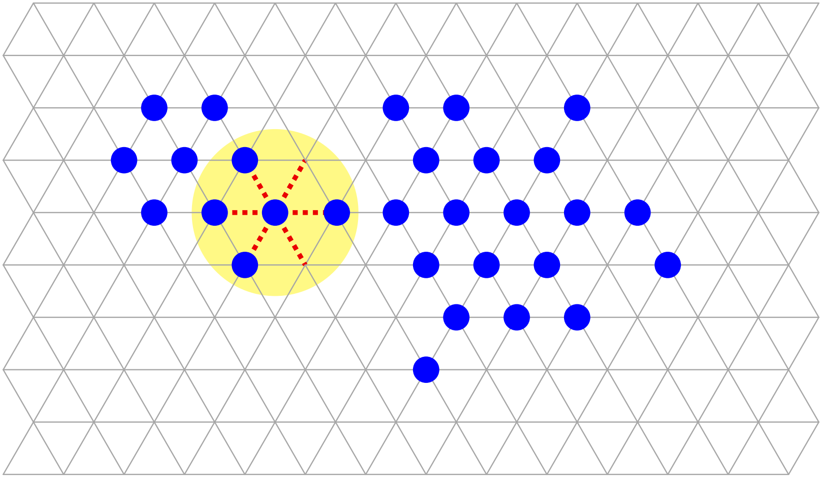
There are physical prototypes inspired by micro-organisms such as amoeba, which are able to move and surround objects.

Programmable Matter: Theoretical Models



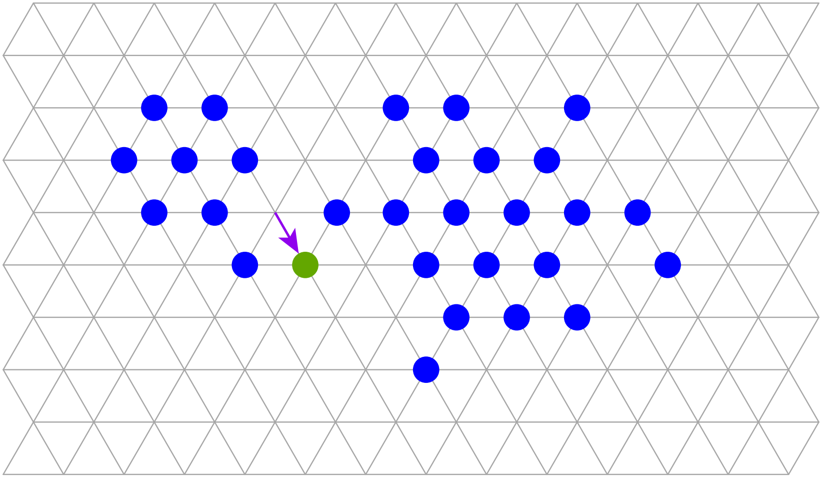
Theoretical models have been developed as well, where particles are finite-state agents on a regular grid.

Programmable Matter: Theoretical Models



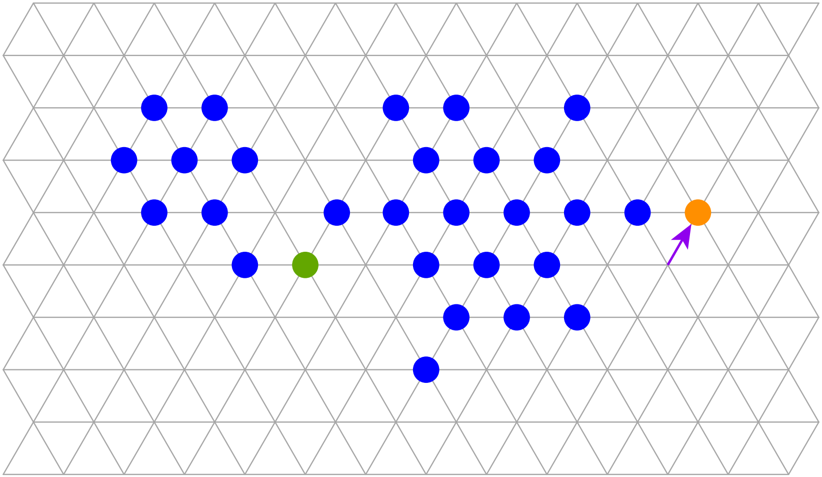
A “sequential scheduler” activates one particle at every time unit.
The activated particle looks in all neighboring locations.

Programmable Matter: Theoretical Models



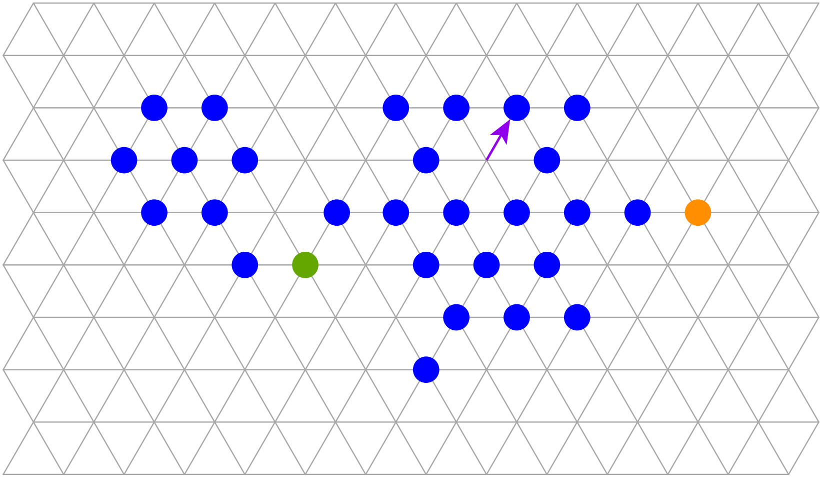
When the particle has looked around, it may decide to move to a neighboring empty location and/or change its internal state.

Programmable Matter: Theoretical Models



Note that individual particles do not see the overall configuration, and have to make decisions based on local observations only.

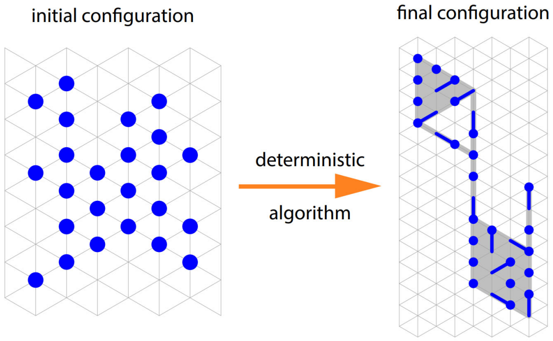
Programmable Matter: Theoretical Models



Note that individual particles do not see the overall configuration, and have to make decisions based on local observations only.

Programmable Matter: Pattern Formation

Our goal as theoretical researchers is to design distributed algorithms that allow particles to perform certain tasks by using the least amount of resources (e.g., internal memory, sensing range, synchronization mechanisms, etc.).

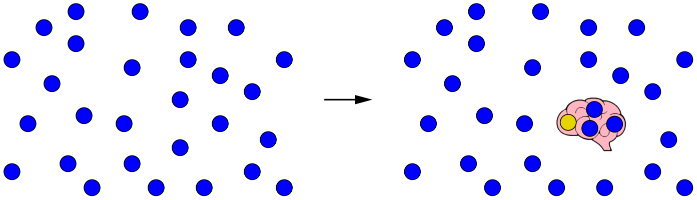


A fundamental task we have studied is "Shape Formation", where particles have to self-organize to form a given pattern.

Programmable Matter: State of the Art

Theorem (*Euro-Par 2020 / Dist. Comp., 2020*)

There is a distributed algorithm for finite-state particles that allows them to form any Turing-computable shape.

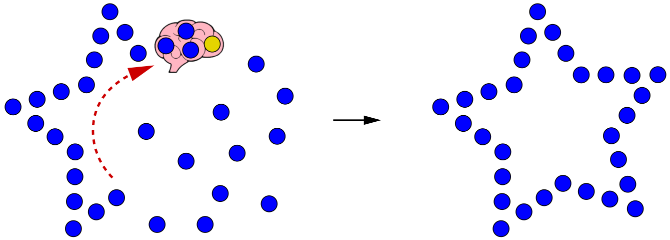


The algorithm starts with a deterministic Leader Election phase. The leader then recruits some particles to simulate a “moving Turing machine” that travels across the system and instructs every particle on where to go to form the final shape.

Programmable Matter: State of the Art

Theorem (*Euro-Par 2020 / Dist. Comp., 2020*)

There is a distributed algorithm for finite-state particles that allows them to form any Turing-computable shape.



The algorithm starts with a deterministic Leader Election phase. The leader then recruits some particles to simulate a "moving Turing machine" that travels across the system and instructs every particle on where to go to form the final shape.

Programmable Matter: New Approach

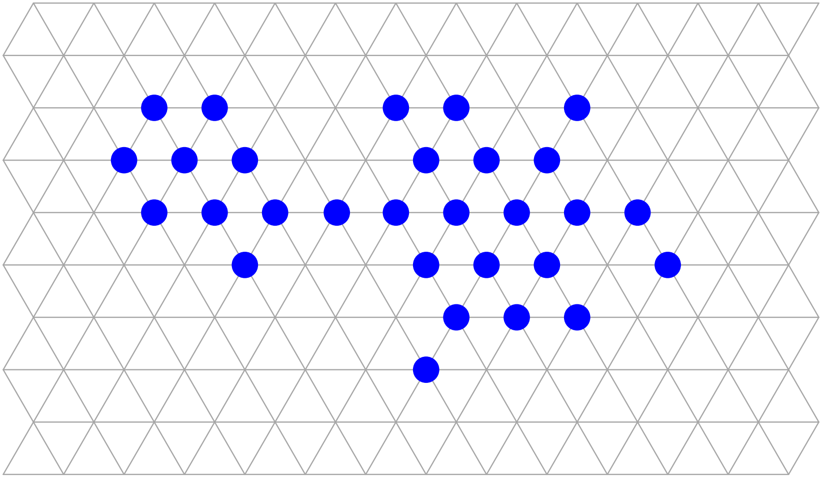
This approach has at least two major problems:

- The algorithm is very vulnerable to crash faults: if the leader malfunctions, the whole system fails to carry out the task.
- Simulating a Turing machine introduces a bottleneck that sequentializes the execution and fails to exploit the parallel nature of Programmable Matter.

To cope with these problems, we are exploring a new approach based on Genetic Programming:

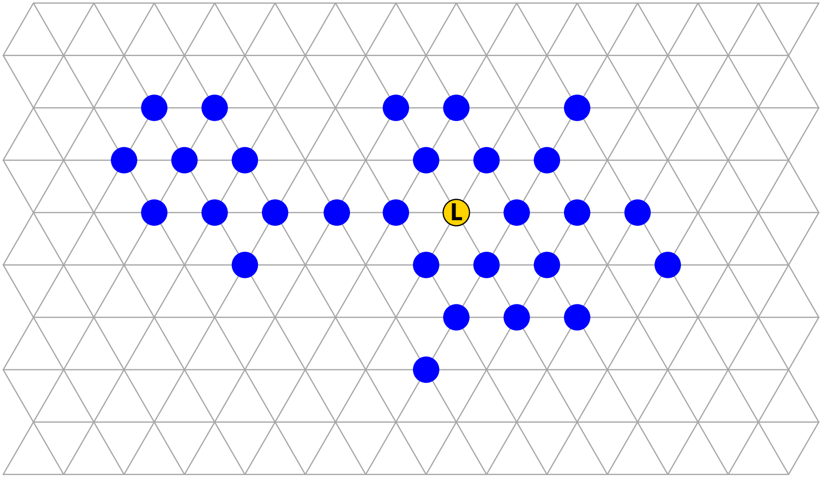
- We designed and developed a Programmable Matter simulator endowed with a general-purpose Genetic Programming framework that allows particles to autonomously discover algorithms for any given task.
- We tested this approach on several Programmable Matter tasks by devising suitable fitness functions and running our Genetic Programming framework on a supercomputer.

Programmable Matter: Additional Tasks



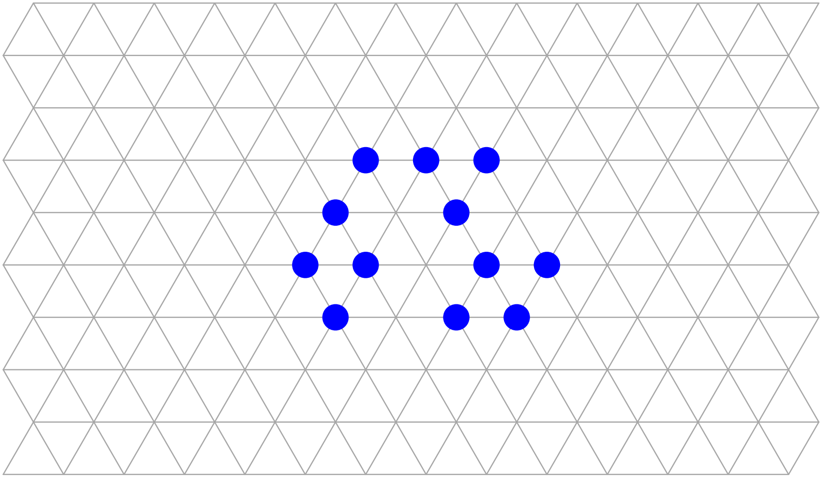
Leader Election: The particles must elect a unique leader without moving. All particles start in the same state.

Programmable Matter: Additional Tasks



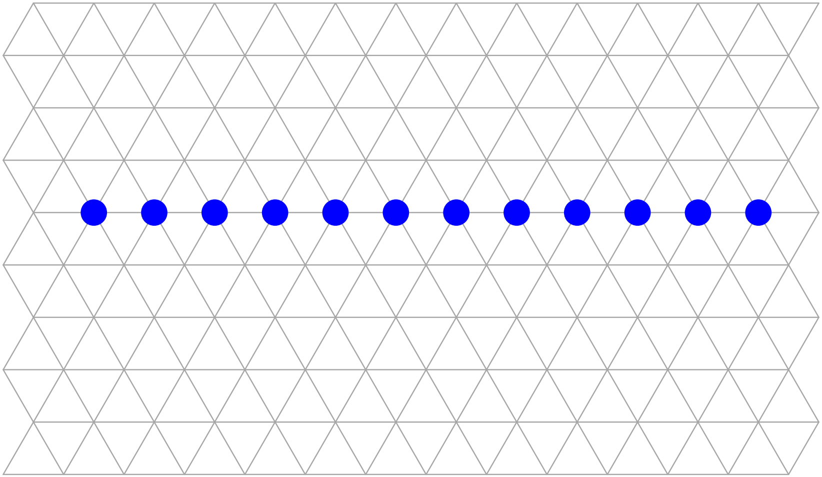
Leader Election: The particles must elect a unique leader without moving. All particles start in the same state.

Programmable Matter: Additional Tasks



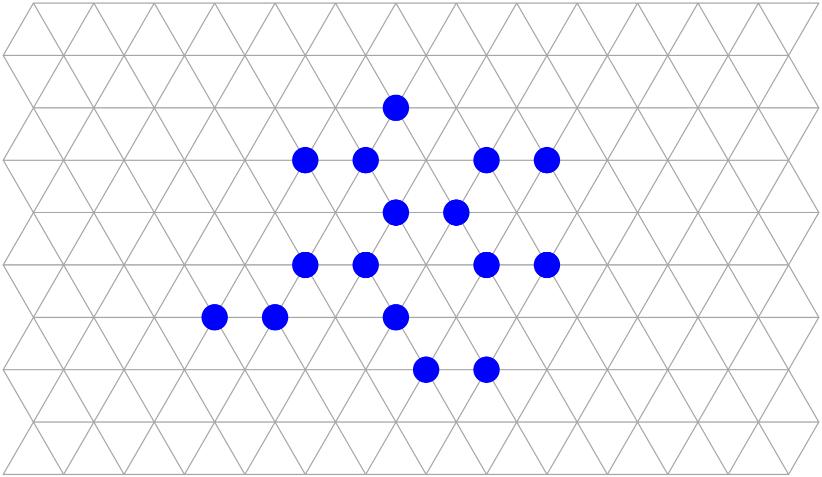
Line Formation: The particles must form a straight line. The initial configuration is assumed to be connected.

Programmable Matter: Additional Tasks



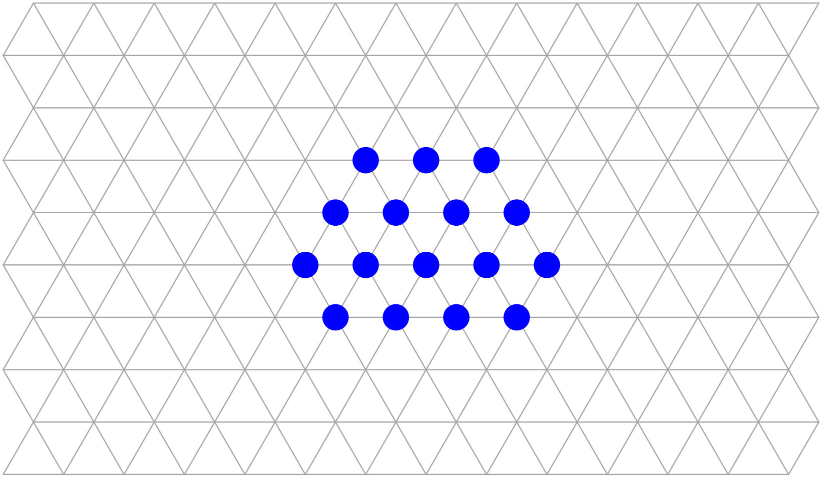
Line Formation: The particles must form a straight line. The initial configuration is assumed to be connected.

Programmable Matter: Additional Tasks



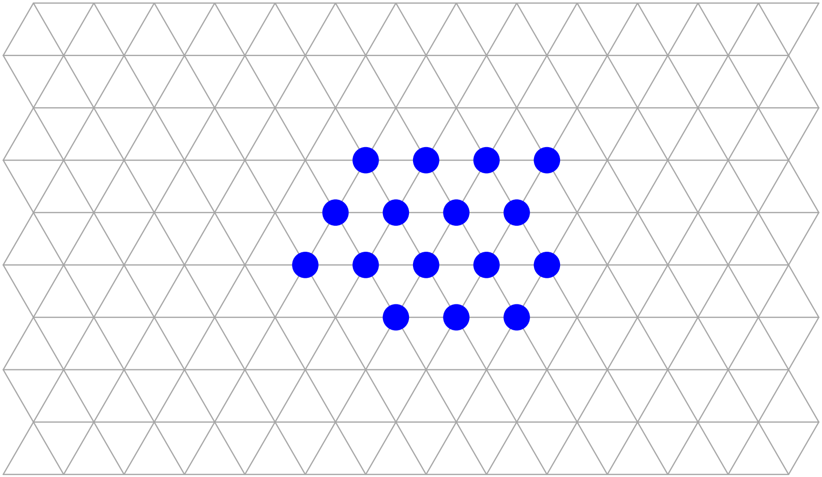
Compaction: The particles must form a configuration of minimum diameter. The initial configuration is assumed to be connected.

Programmable Matter: Additional Tasks



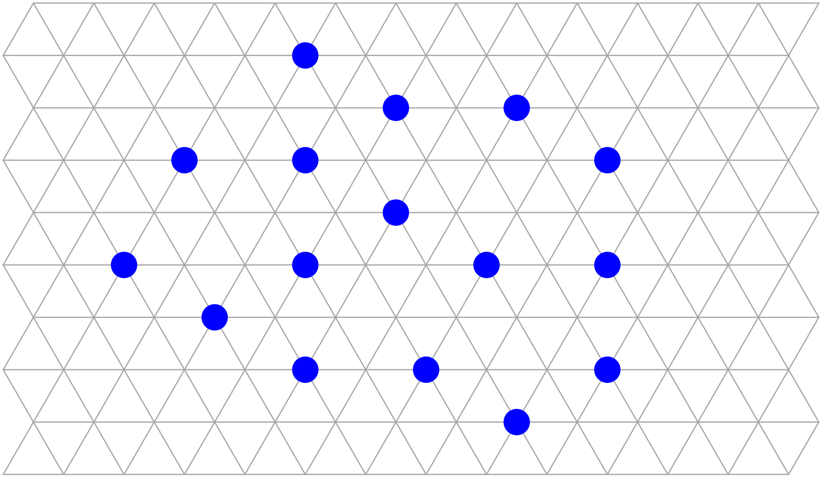
Compaction: The particles must form a configuration of minimum diameter. The initial configuration is assumed to be connected.

Programmable Matter: Additional Tasks



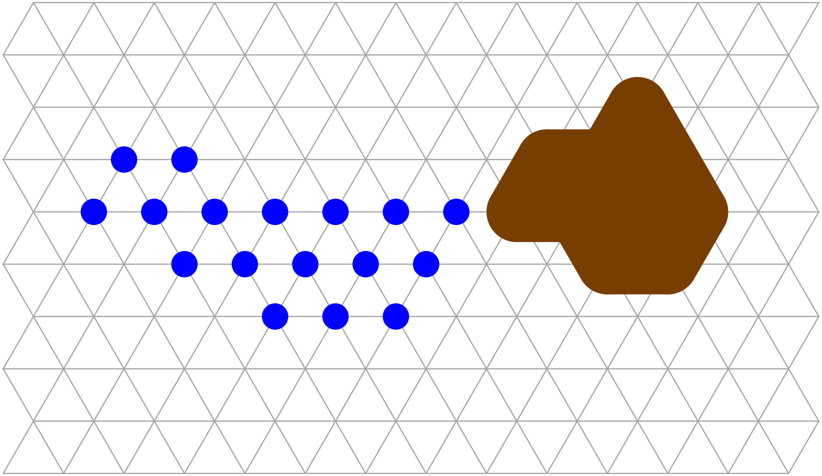
Scattering: The system must reach a configuration where no two particles are adjacent and no particle is moving.

Programmable Matter: Additional Tasks



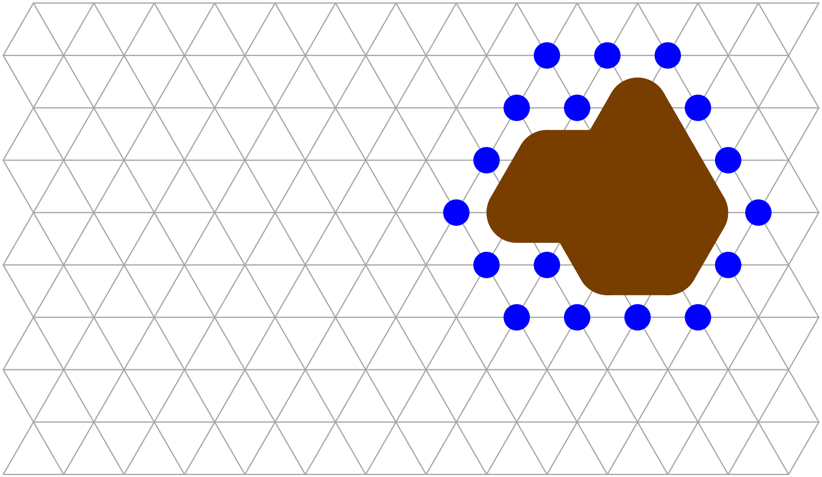
Scattering: The system must reach a configuration where no two particles are adjacent and no particle is moving.

Programmable Matter: Additional Tasks



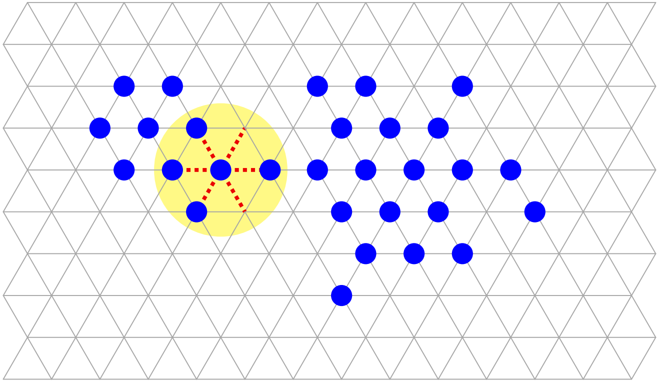
Coating: The particles must completely surround an object of unknown shape. Initially, only one particle is touching the object.

Programmable Matter: Additional Tasks



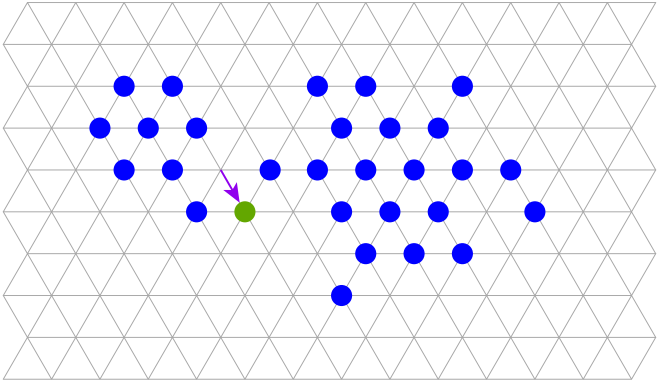
Coating: The particles must completely surround an object of unknown shape. Initially, only one particle is touching the object.

Programmable Matter: Algorithm Model



A local algorithm is a function that takes as input a particle's **internal state** and **list of neighbors**, each of which may be an empty location or a particle with a certain state. The output is the particle's **new state** and a **direction of movement**.

Programmable Matter: Algorithm Model



An algorithm is a function that takes as input a particle's **internal state** and **list of neighbors**, each of which may be an empty location or a particle with a certain state. The output is the particle's **new state** and a **direction of movement**.

Primitive Set

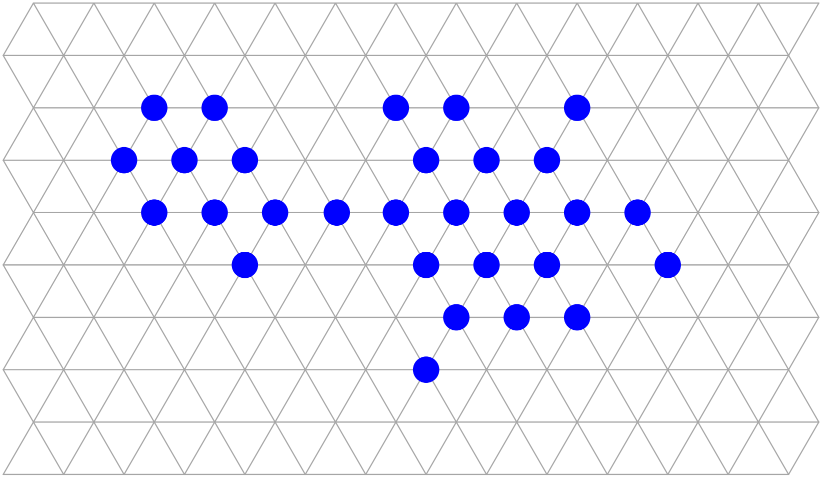
We take these “primitives” as building blocks of our algorithms:

- **Basic Instructions**
  - Concatenate [Instruction] and [Instruction]
  - If [Boolean] then [Instruction] else [Instruction]
  - Set state [Integer]
  - Set direction [Integer]
- **Integer Terminals**
  - Get state
  - Get neighbor [Integer]
  - Integer constants
- **Integer Operators**
  - Add [Integer] [Integer]
  - Subtract [Integer] [Integer]
  - Max [Integer] [Integer]
  - Min [Integer] [Integer]

Primitive Set

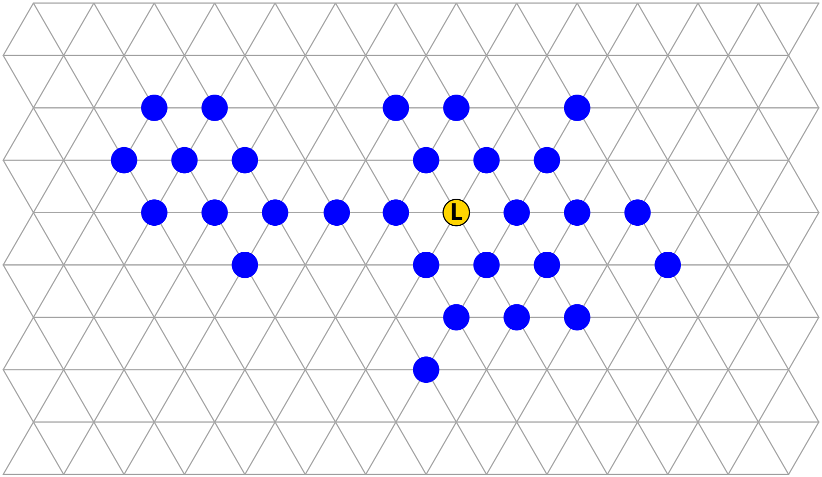
- **Boolean Terminals**
  - True
  - False
- **Boolean Operators**
  - Not [Boolean]
  - And [Boolean] [Boolean]
  - Or [Boolean] [Boolean]
  - Xor [Boolean] [Boolean]
  - Equals [Integer] [Integer]
  - Greater than [Integer] [Integer]
  - Less than [Integer] [Integer]
- **Counter Operations**
  - Set counter [Integer]
  - Get counter
  - Increment counter
  - Decrement counter

Fitness Functions



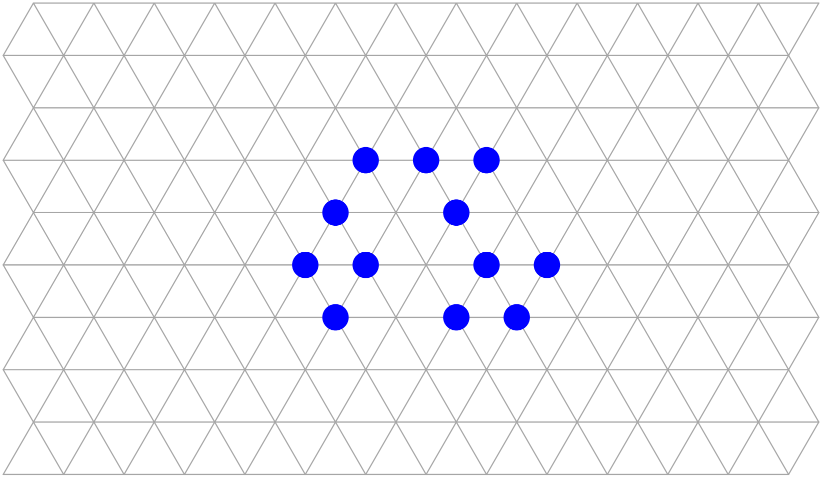
Leader Election: Give a large penalty if there is no leader in the system and a small penalty for having more than one leader.

Fitness Functions



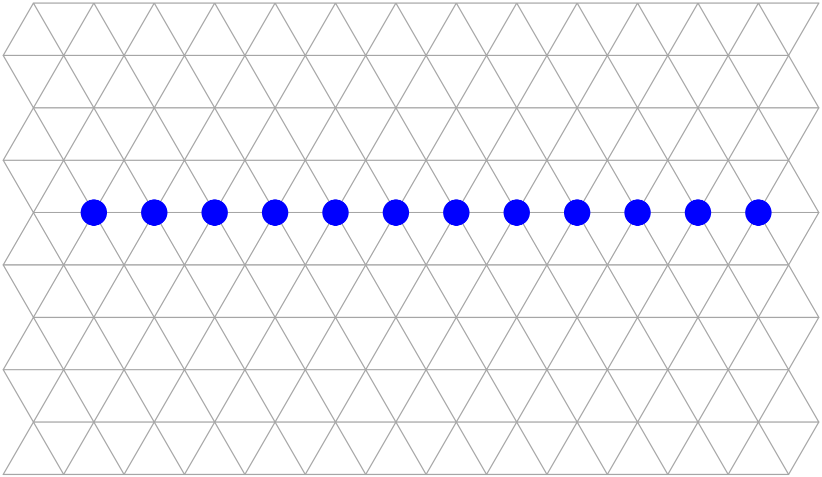
Leader Election: Give a large penalty if there is no leader in the system and a small penalty for having more than one leader.

Fitness Functions



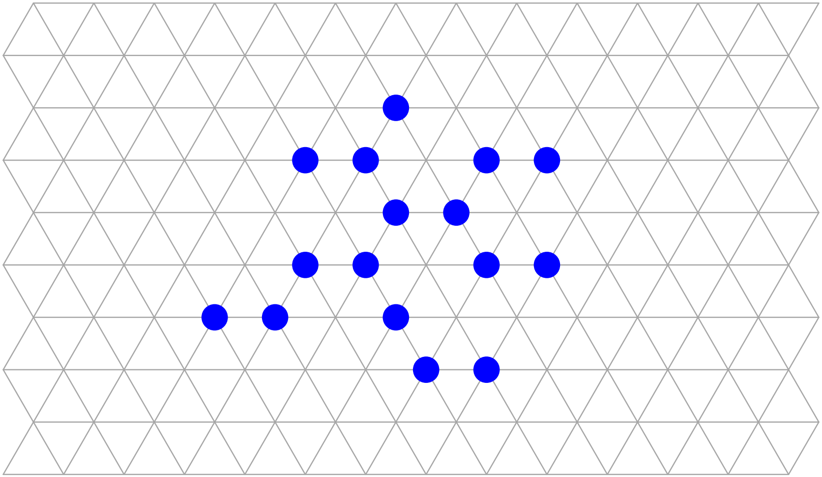
Line Formation: Give a penalty for every particle that does not have exactly two neighbors on opposite sides.

Fitness Functions



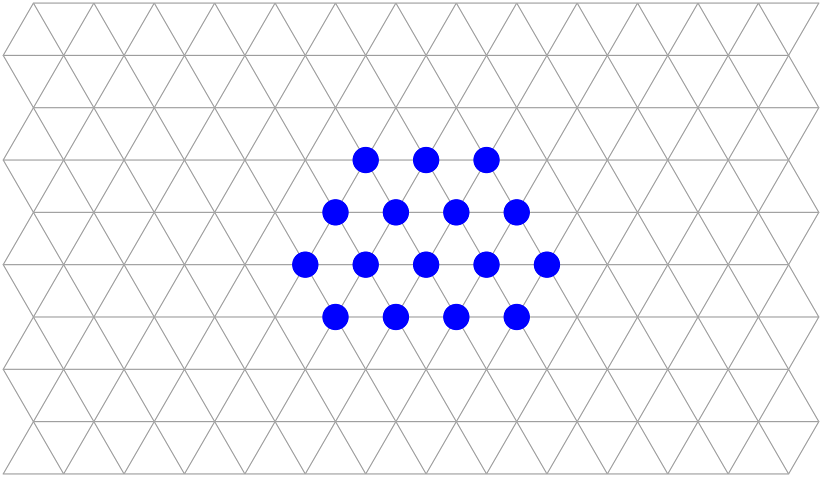
Line Formation: Give a penalty for every particle that does not have exactly two neighbors on opposite sides.

Fitness Functions



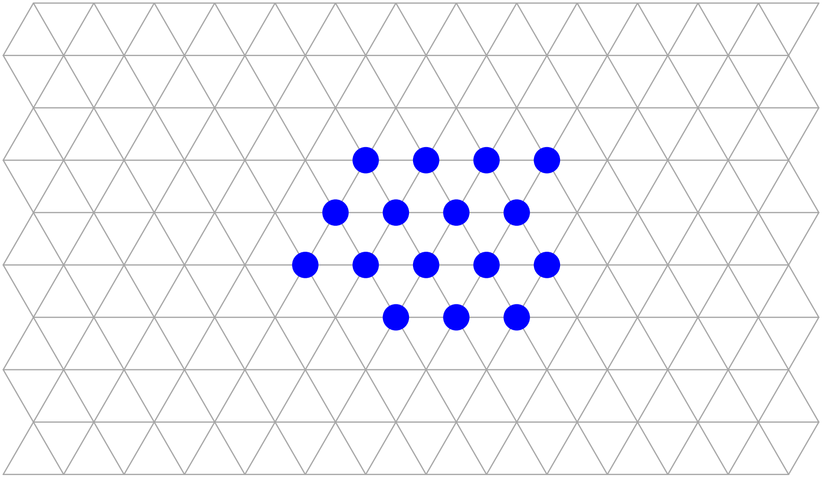
Compaction: Give a penalty for every particle that is not completely surrounded by other particles.

Fitness Functions



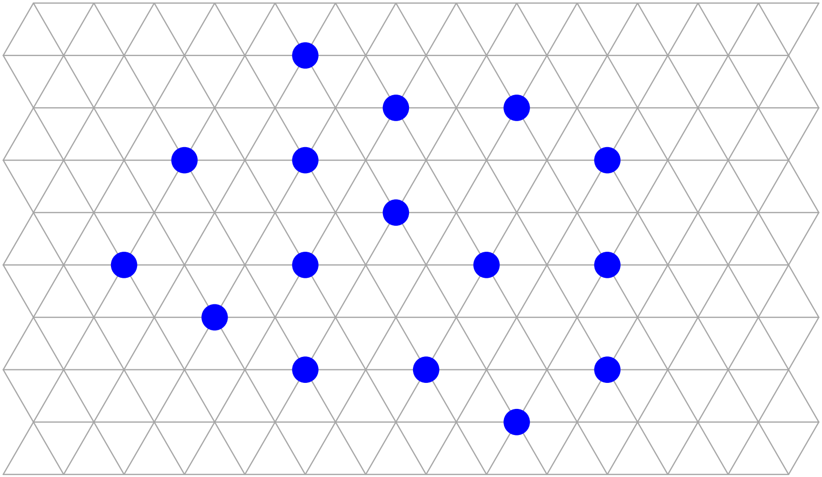
Compaction: Give a penalty for every particle that is not completely surrounded by other particles.

Fitness Functions



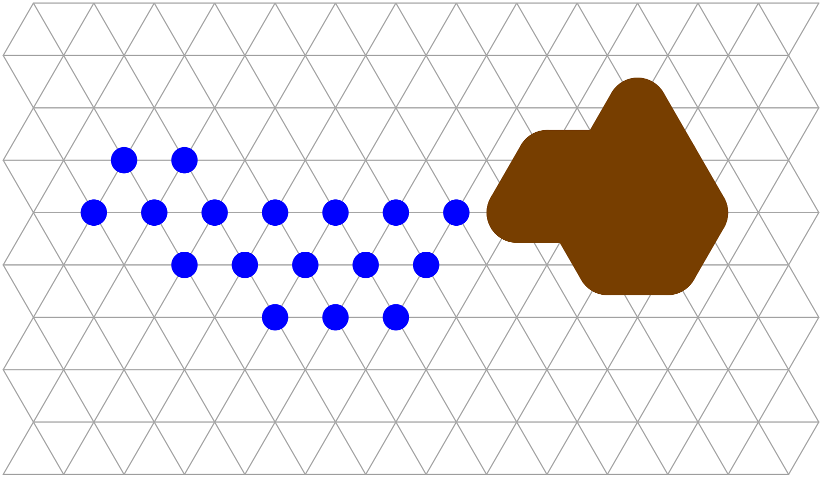
Scattering: Give a large penalty for every two neighboring particles, and a small penalty for particles that are too far apart.

Fitness Functions



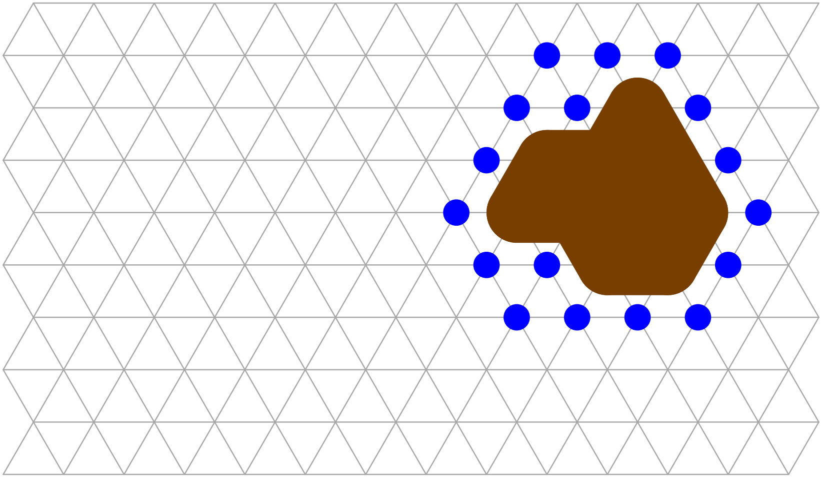
Scattering: Give a large penalty for every two neighboring particles, and a small penalty for particles that are too far apart.

Fitness Functions



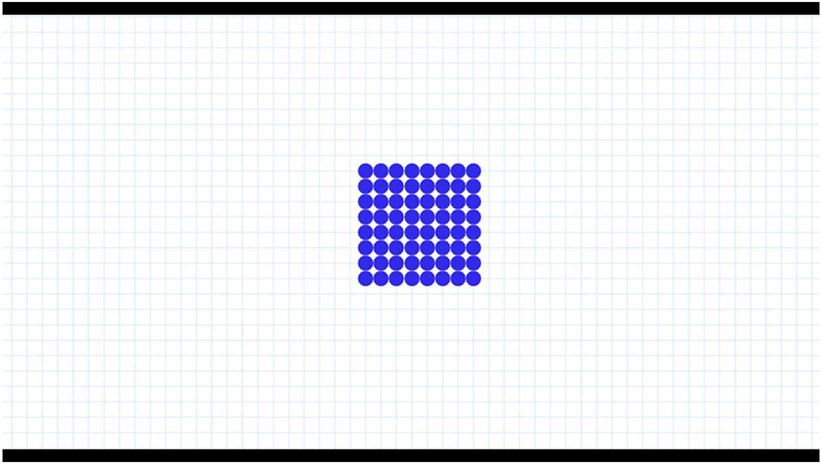
Coating: Give a penalty for every point on the object's surface that is not occupied by a particle.

Fitness Functions



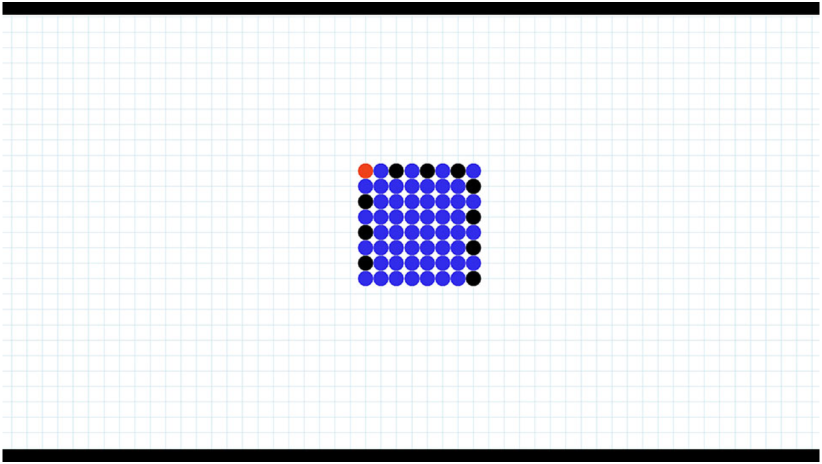
Coating: Give a penalty for every point on the object's surface that is not occupied by a particle.

Experimental Results



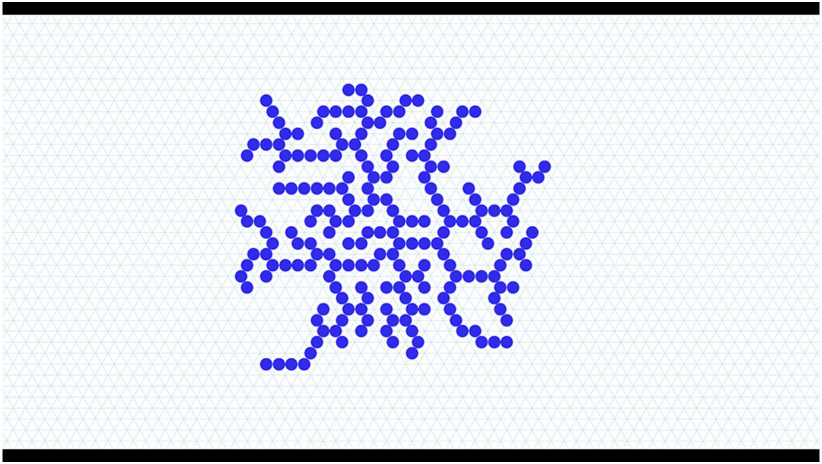
Leader Election in a rectangle

Experimental Results



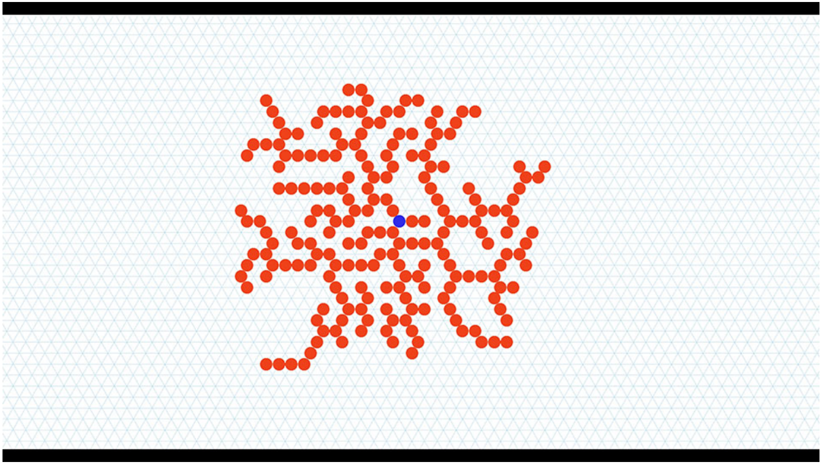
Leader Election in a rectangle

Experimental Results



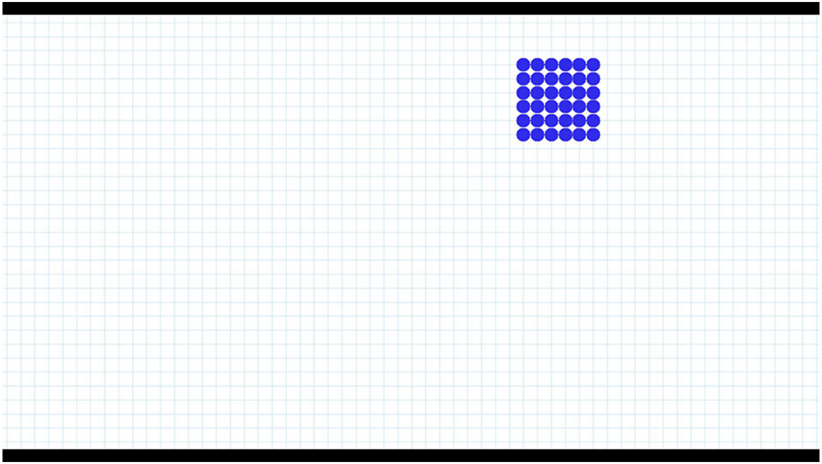
Leader Election in a tree

Experimental Results



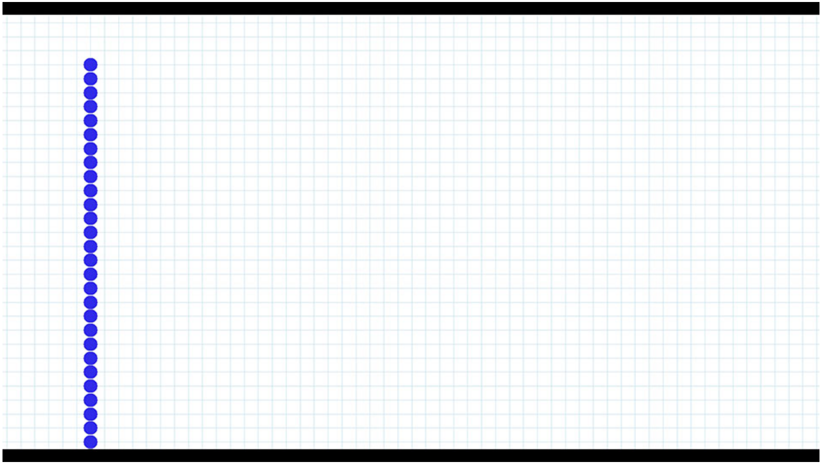
Leader Election in a tree

Experimental Results



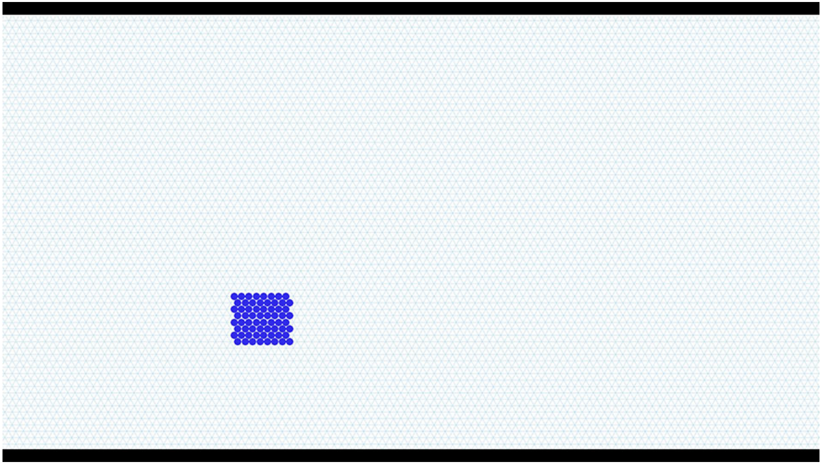
Line Formation from a rectangle in a square grid

Experimental Results



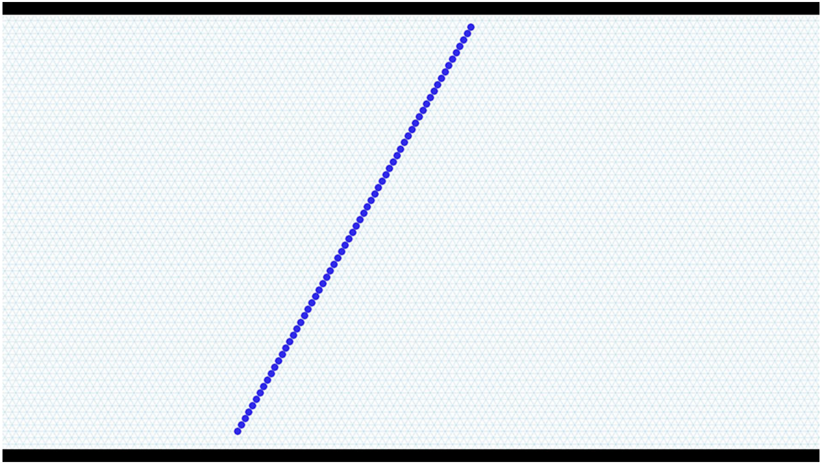
Line Formation from a rectangle in a square grid

Experimental Results



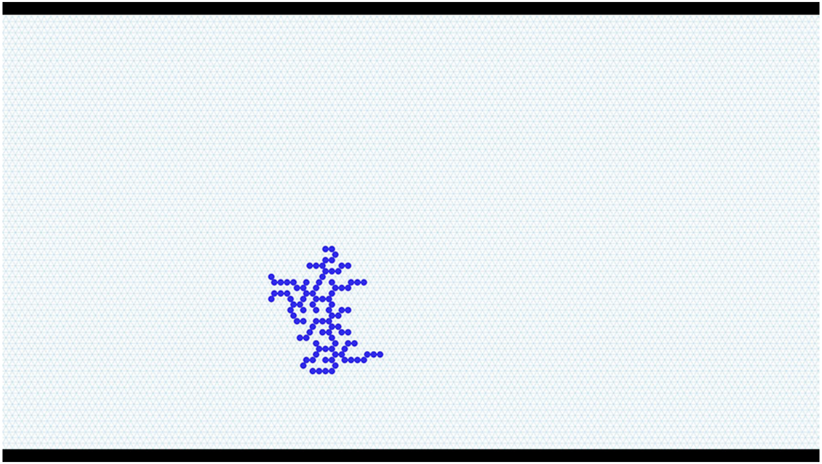
Line Formation from a box in a triangular grid

Experimental Results



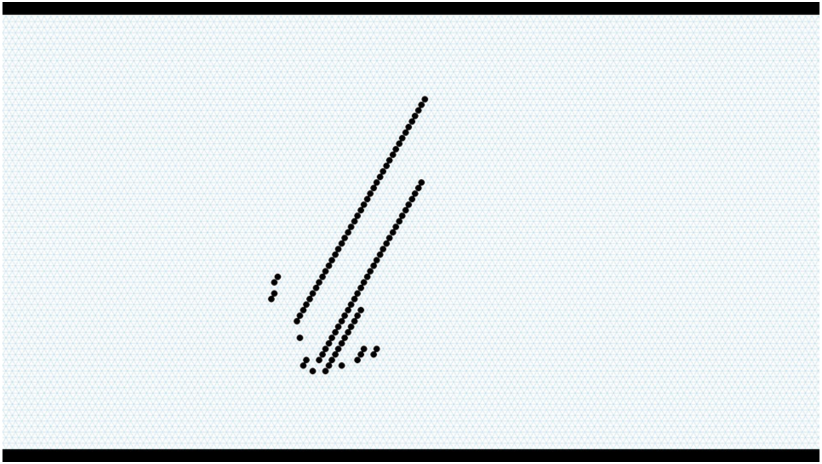
Line Formation from a box in a triangular grid

Experimental Results



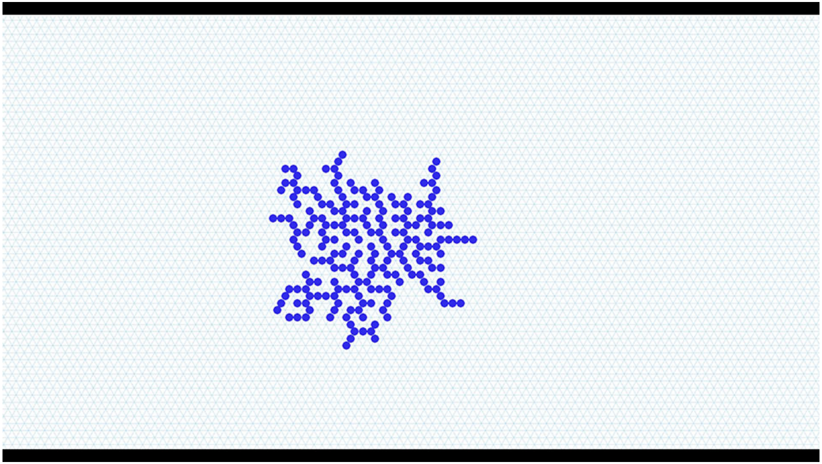
Line Formation from a tree

Experimental Results



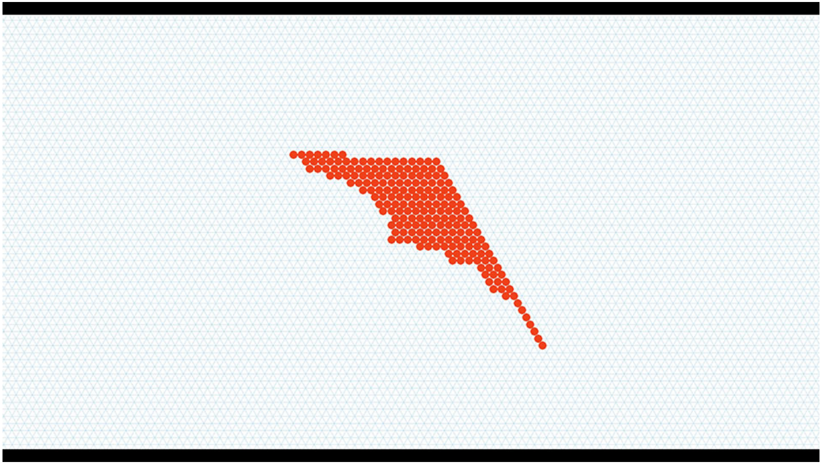
Line Formation from a tree

Experimental Results



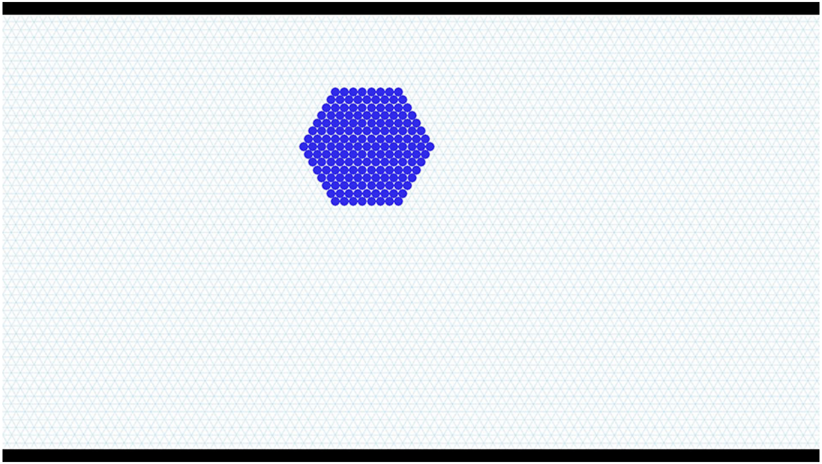
Compaction of a tree

Experimental Results



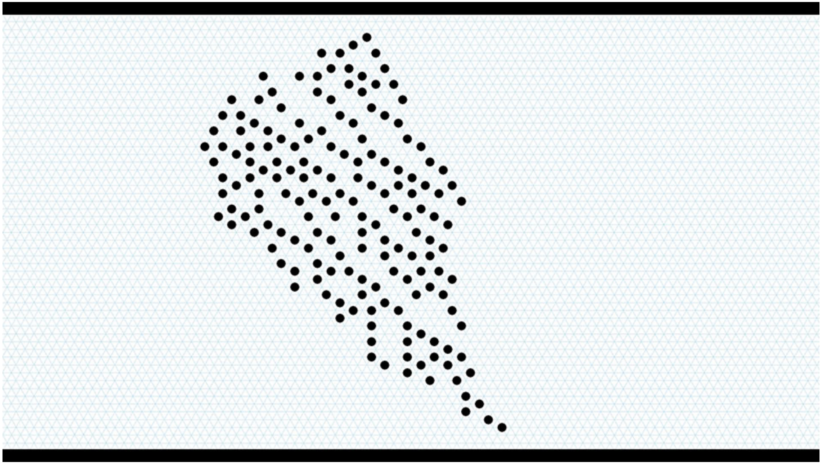
Compaction of a tree

Experimental Results



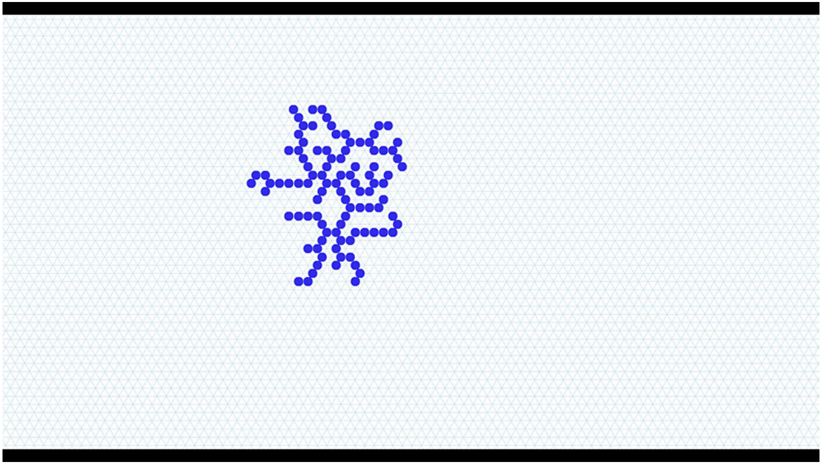
Scattering from a hexagon

Experimental Results



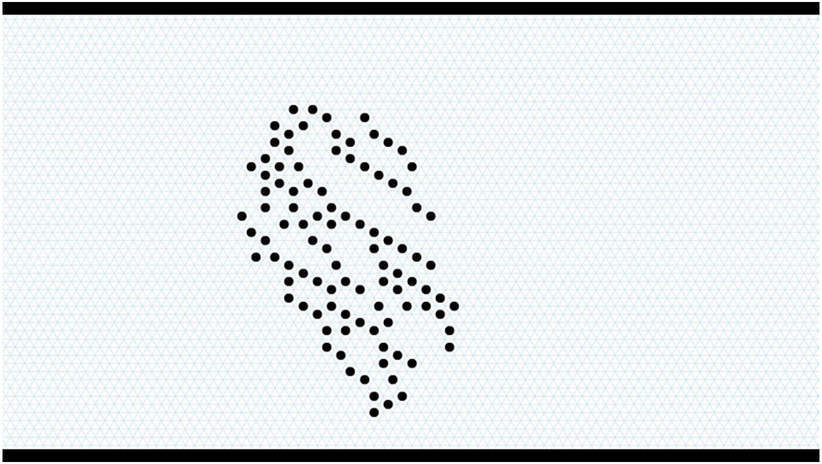
Scattering from a hexagon

Experimental Results



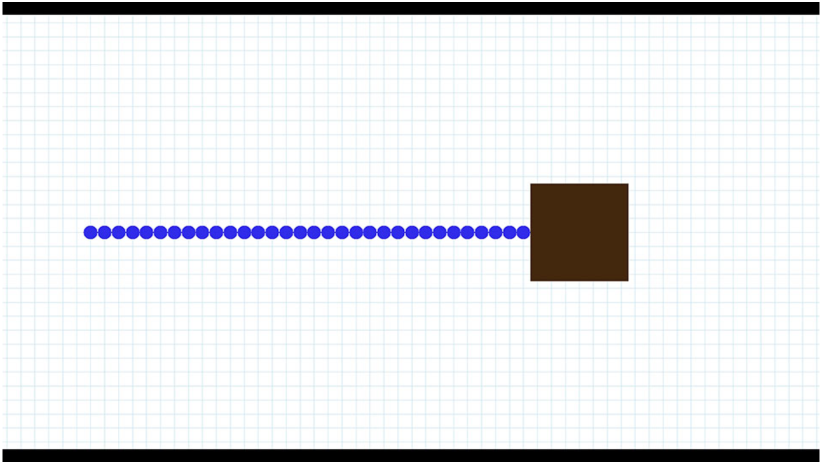
Scattering from a tree

Experimental Results



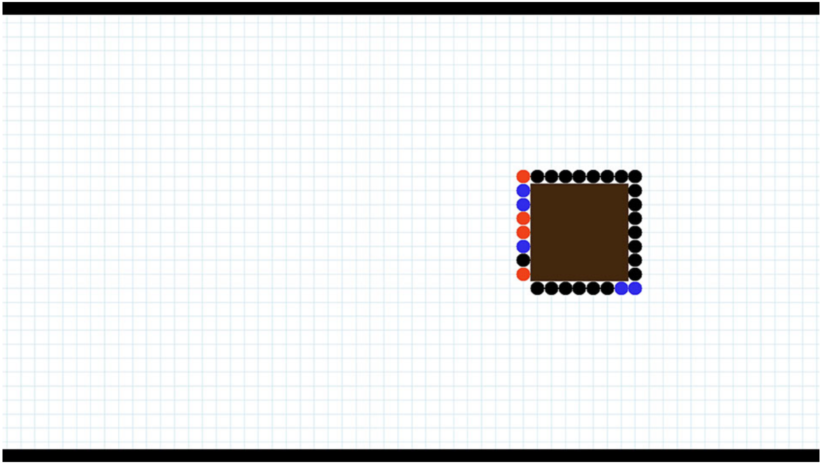
Scattering from a tree

Experimental Results



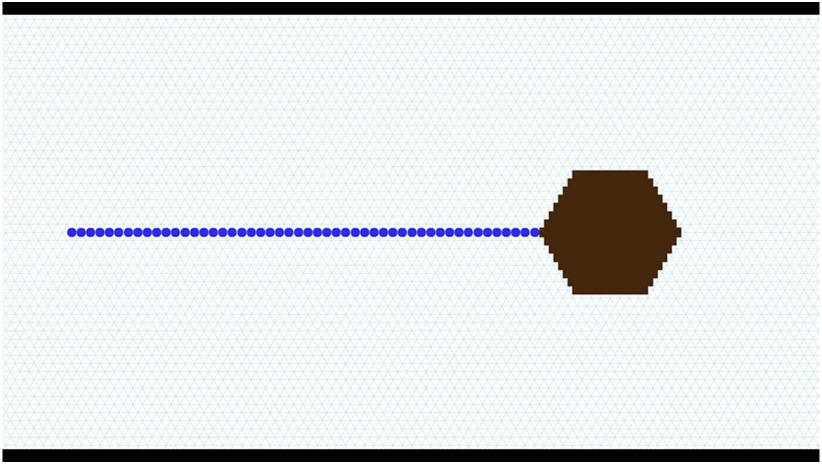
Coating of a rectangle

Experimental Results



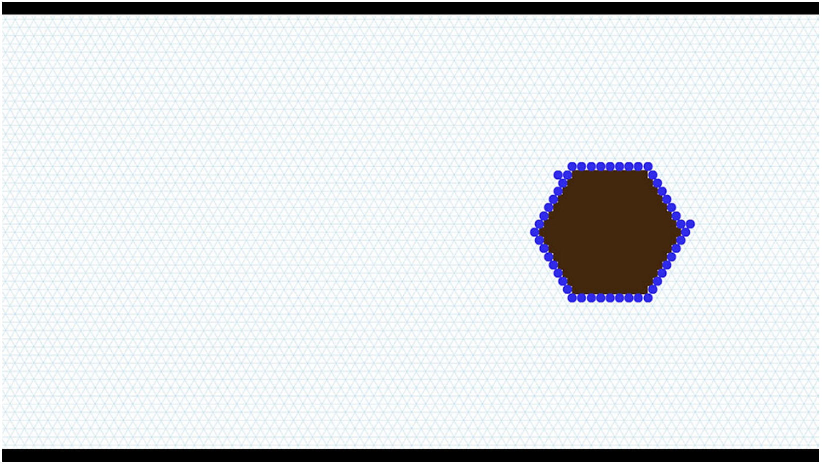
Coating of a rectangle

Experimental Results



Coating of a hexagon

Experimental Results



Coating of a hexagon

Conclusion

Summary:

- We have developed our Programmable Matter simulator and Genetic Programming framework in Python by extending the DEAP library.
- We have evolved our programs on a pair of AMD EPYC 7502 2.5 GHz 32C/64T processors with 16x32 GB DDR4 3200 MHz RAM and a 6 TB Hard Disk.
- The evolved programs can perform fundamental Programmable Matter tasks in some basic settings, and have also re-discovered known techniques such as Saturation.

Future work:

- Design more sophisticated and meaningful primitive functions.
- Perform harder tasks from more general initial configurations.
- Introduce faulty particles and implement fault tolerance.
- Produce humanly understandable algorithms for all tasks.