

Stochastic Search Part II

Collective Computation and Genetic Algorithms

Manuel Pita | April 2020

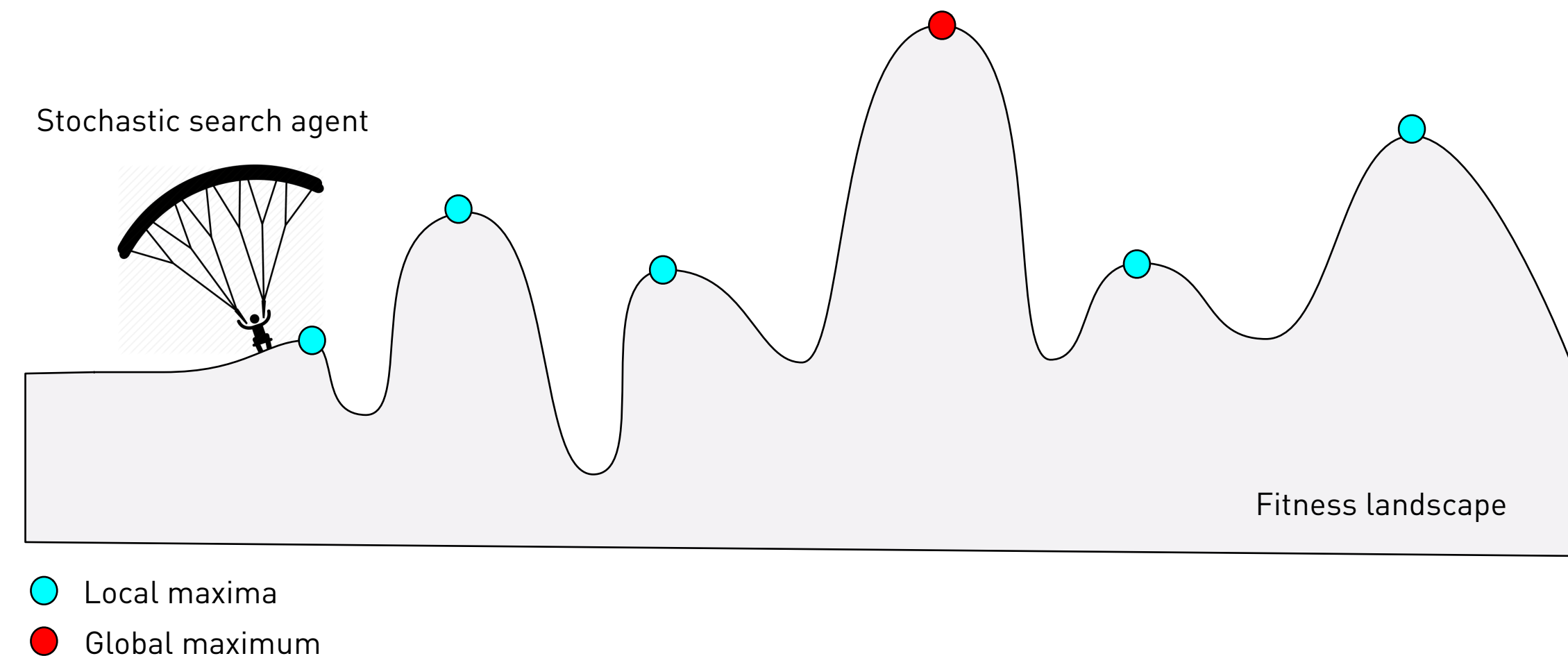
Today

- Simulated annealing recap and discussion
- The travelling salesman
- Side story: WaggleBees ✨🐝✨
- Cellular Automata (CA)
- The CA Majority Classification Task
- One search agent to many search agents
- A new kind of information: fitness
- A new kind of expansion: cross-over and mutation
- Initial intuitions about Genetic Algorithms

Stochastic Search

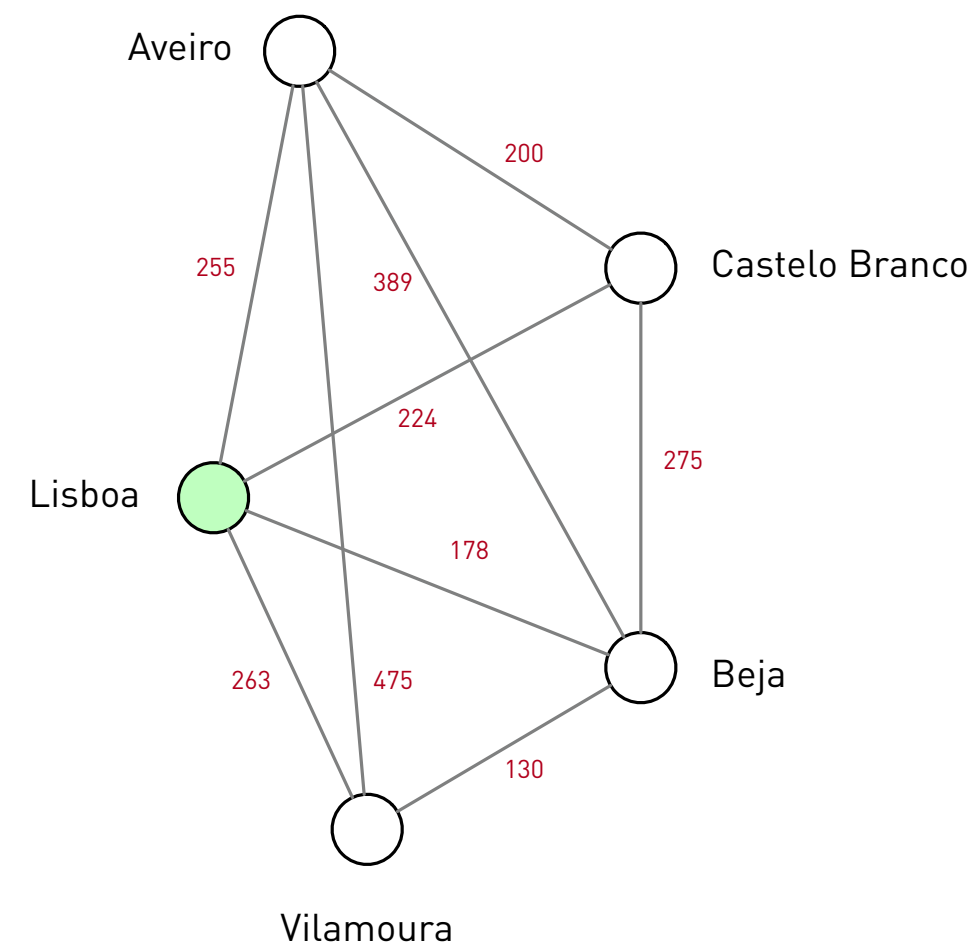
Simulated annealing

- Searching on unexplored space, with no map (recall deterministic vs. probabilistic and observability)
- Agent learns as they explore the search space
- Recall the basic local search algorithm
- Recall how basic local search gets agent stuck in local maxima (or minima)
- We add the T and α parameters to the local search. How does that help?
- Let's consider search graph again: nodes as step by step vs. whole-solution



Travelling Salesman

A basic algorithm to get started and to get you thinking!



```
edges = {('Lisboa', 'Aveiro'): 255,
        ('Lisboa', 'Beja'): 178,
        ...
        ('Beja', 'Aveiro'): 389,
        ...
}
```




0. set your parameters e.g. to $T = 0.5$, $\alpha = 0.01$, $\text{max_it} = T/\alpha$
1. define the **graph** of city connections
2. define a **random initial tour** as a permutation of the cities that are not the start city
3. put the start city as the start and the end of the tour
4. compute the tour's total length
5. add the **initial_tour** and its **total_length** to the **saved_paths**
6. start search loop head is current tour
 - 6.1 create a **new_tour** by swapping a random pair of **inner** cities
 - 6.2 compute **total traversed distance** for new_tour
 - 6.3 get a random number **dice** between zero and one
 - 6.4 if $\text{dice} \leq T$, **new_head** is the worst between new_tour and head else pick the best
 - 6.5 if new head is not in **saved_paths**, add new head to **saved_paths**
 - 6.6 $T = T - \alpha$
 - 6.7 Repeat for **max_it** iterations
7. return best tour in **saved_paths**

We are now exploring the space of candidate solutions

Solutions are still sequences of actions to get to goal

Side Story

An example of collective computation in nature

- How do bees  find food  to feed the beehive and to make ?
- Do they do perform search?
- if they search, do we know the *algorithm* they use?
- Let's watch and analyse a short video
- Discussion time

https://www.youtube.com/watch?v=bFDGPgXtK-U&list=PLZtR7YaF_Q5TCHqkHf32RgKF4eW7VjAqS&index=5&t=318s

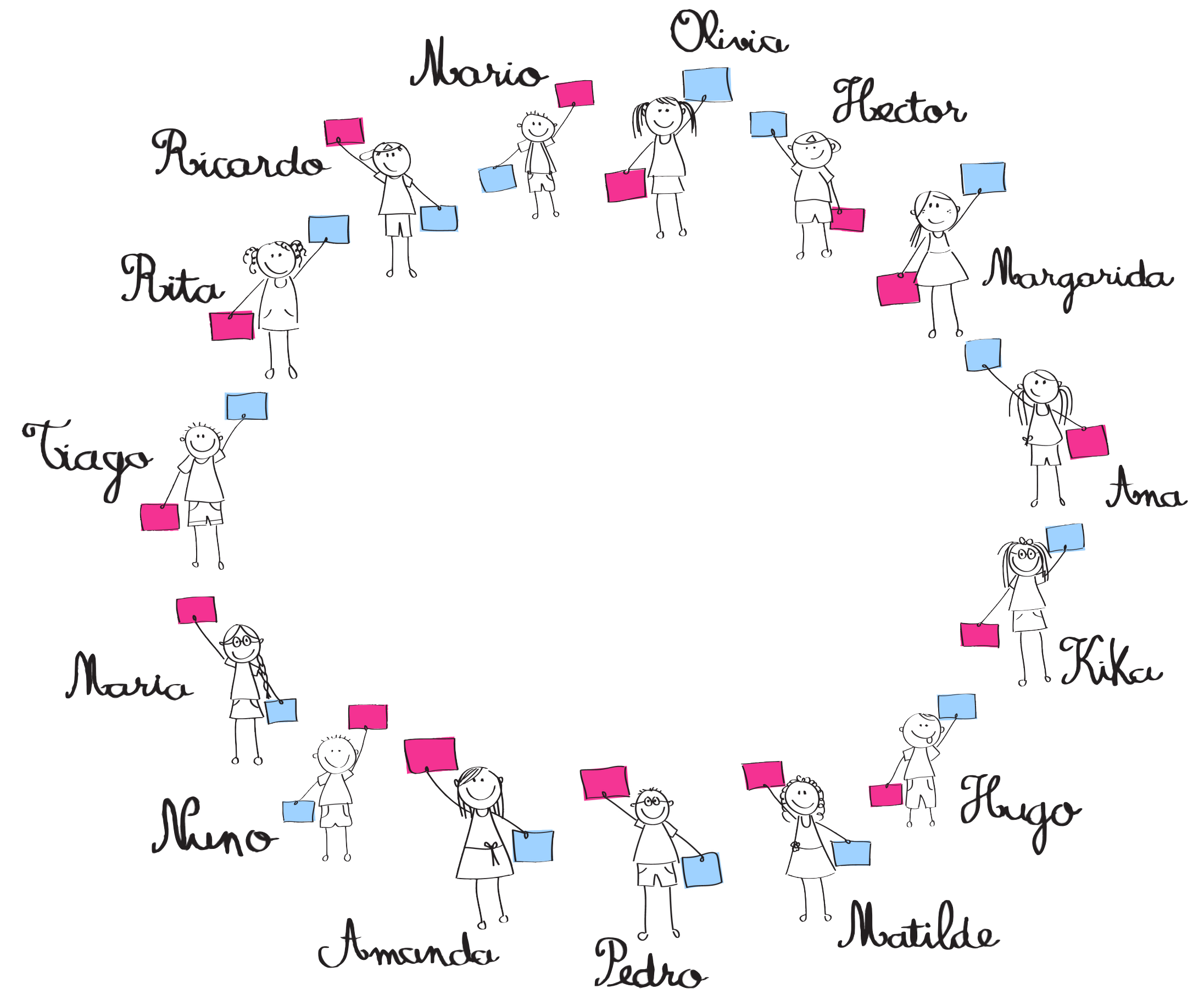
Cellular Automata

They are like a network of Turing machines!

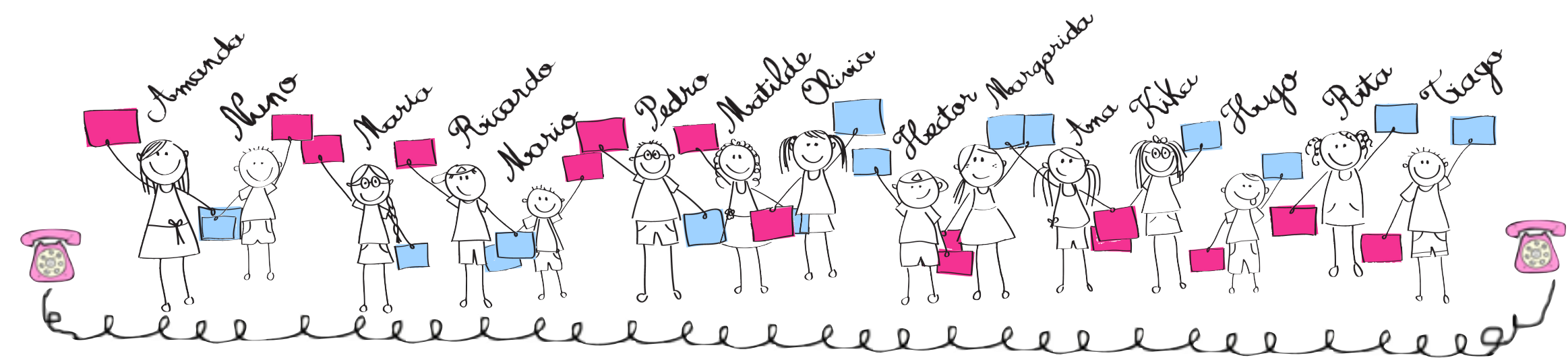
- Invented in the 1940s by two geniuses: **John Von Neumann** and **Stanislaw Ulam**
- CAs are the equivalent of networked Turing machines
- CAs are good models of natural computation
- Inter-cellular skin regulation, leaf respiration by plants, and many other examples in Biology
- CAs are likely to revolutionise computer processing power in the future by allowing us to make networked processors with many very cheap chips. More power, way less heat
- Also computable nano-fabrics
- Homework: study the **Game of Life** created by **John Conway**
- **What is the problem? We don't know how to formalise Turing machines in the CA style yet.**

Cellular Automata

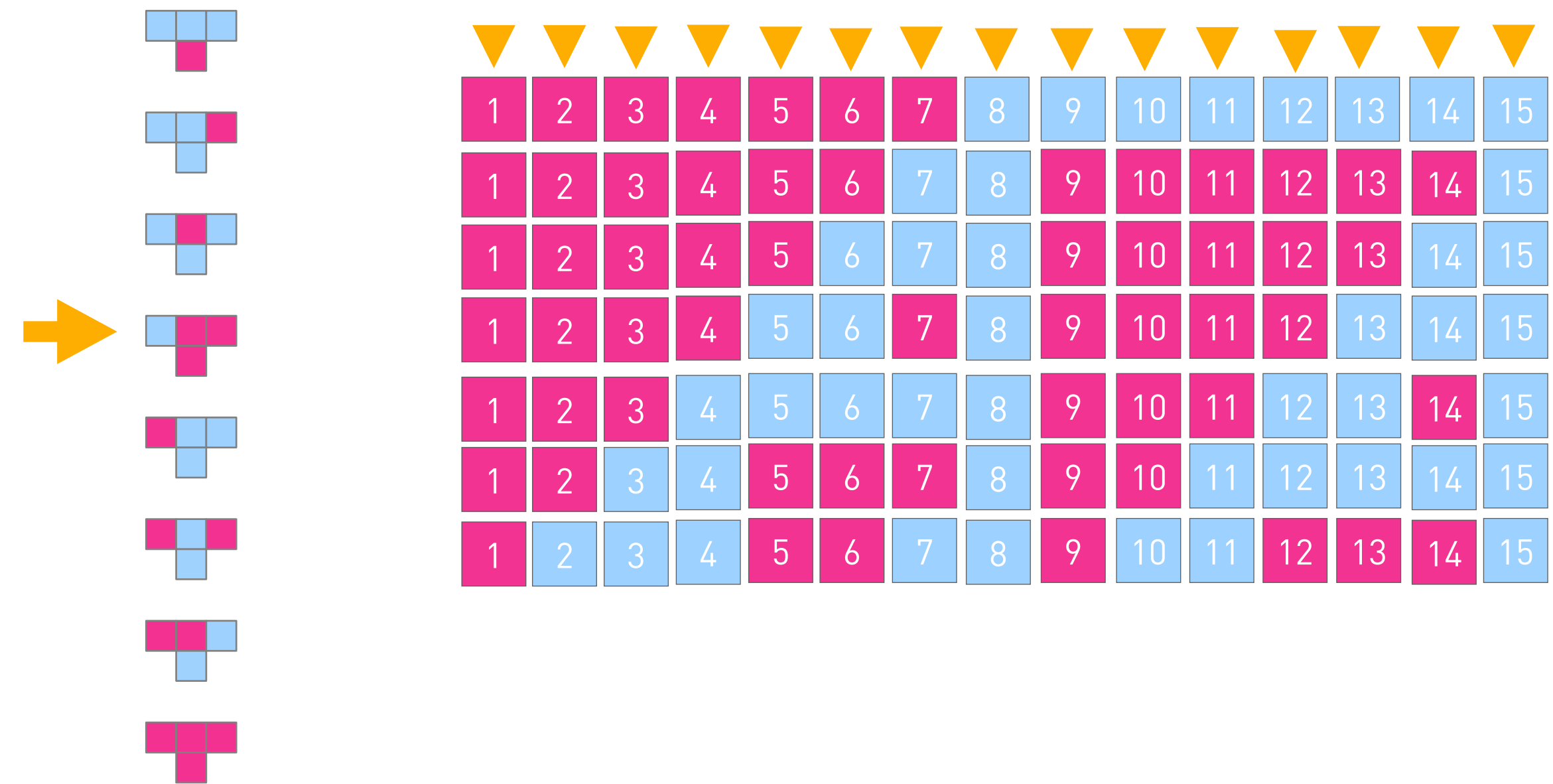
- This is a $n = 15$ children's game, each child is an automaton 🧒 → 🤖
- Each kid can see some of their neighbours to both sides $r = 1, 2, 3 \dots$
- This creates local neighbourhoods for each with sizes $n = 2r + 1 = 3, 5, 7 \dots$
- Each kid has two cards, a pink one and a blue one, $s = 2$ states
- At every turn of the game, $t+1$ every kid shows one of the cards
- They decide what card to show at time t , following a **fixed rule** based on neighbourhood



Cellular Automata

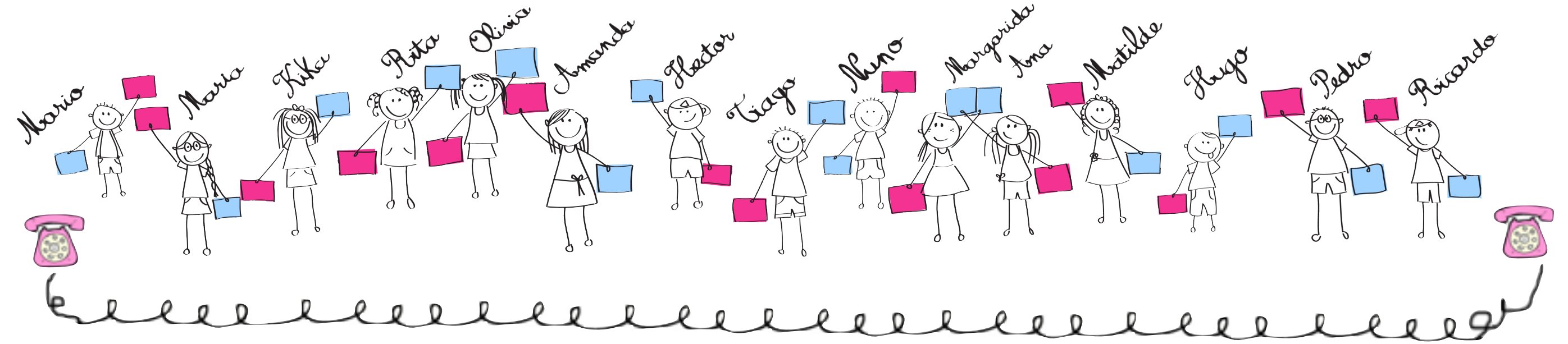


- The card shown at $t+1$ always depends on the neighbourhood at time t
- Time is discrete, tic, toc, tic, toc...
- Notice that over time cells can communicate (space-time computation)
- CAs converge to repeating patterns
- Python implementation is easy, but we have to be careful with the boundaries: start working on it, because this will be part of your first assessed project.



Try to compute the next row!

Cellular Automata



- Remember Turing machines, the way we had tables with *conditions* and *actions*?
- The *conditions* in a CA rule are defined by the number of states and size of local neighbourhood
- If two states (pink and blue) and $r=1$, what is the size of the rule?
 - Local neighbourhood has three and rule size is two to the power of three, which is eight!
- How many action combinations for the eight conditions?
 - Two to the power of eight which is two hundred and fifty six different games we can play
- CAs in $r=1$, $n=3$, with 2 states are called **Elementary Cellular Automata**
- What do you think is the equivalent to the *internal state* of the Turing machine in cellular automata?

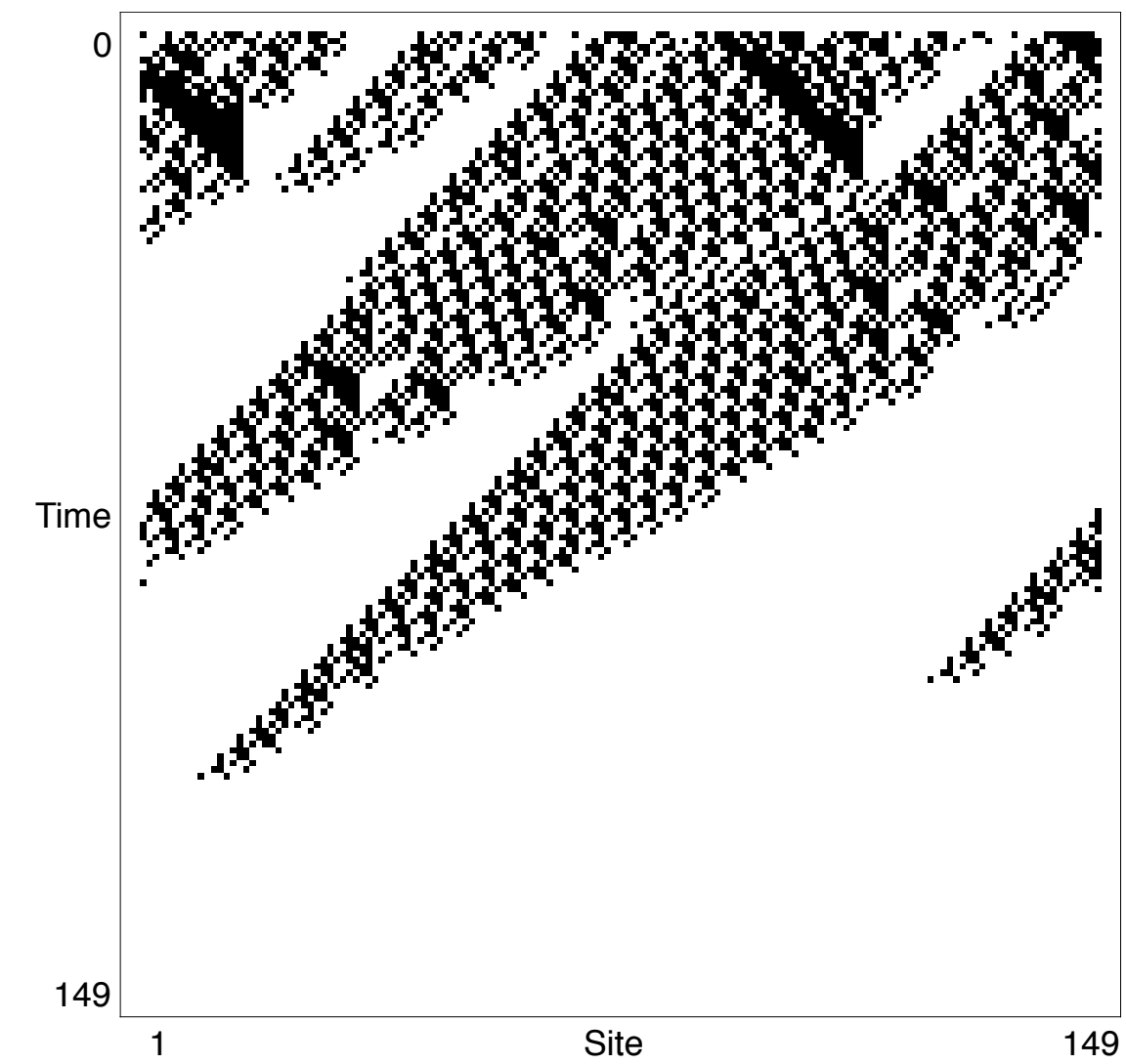
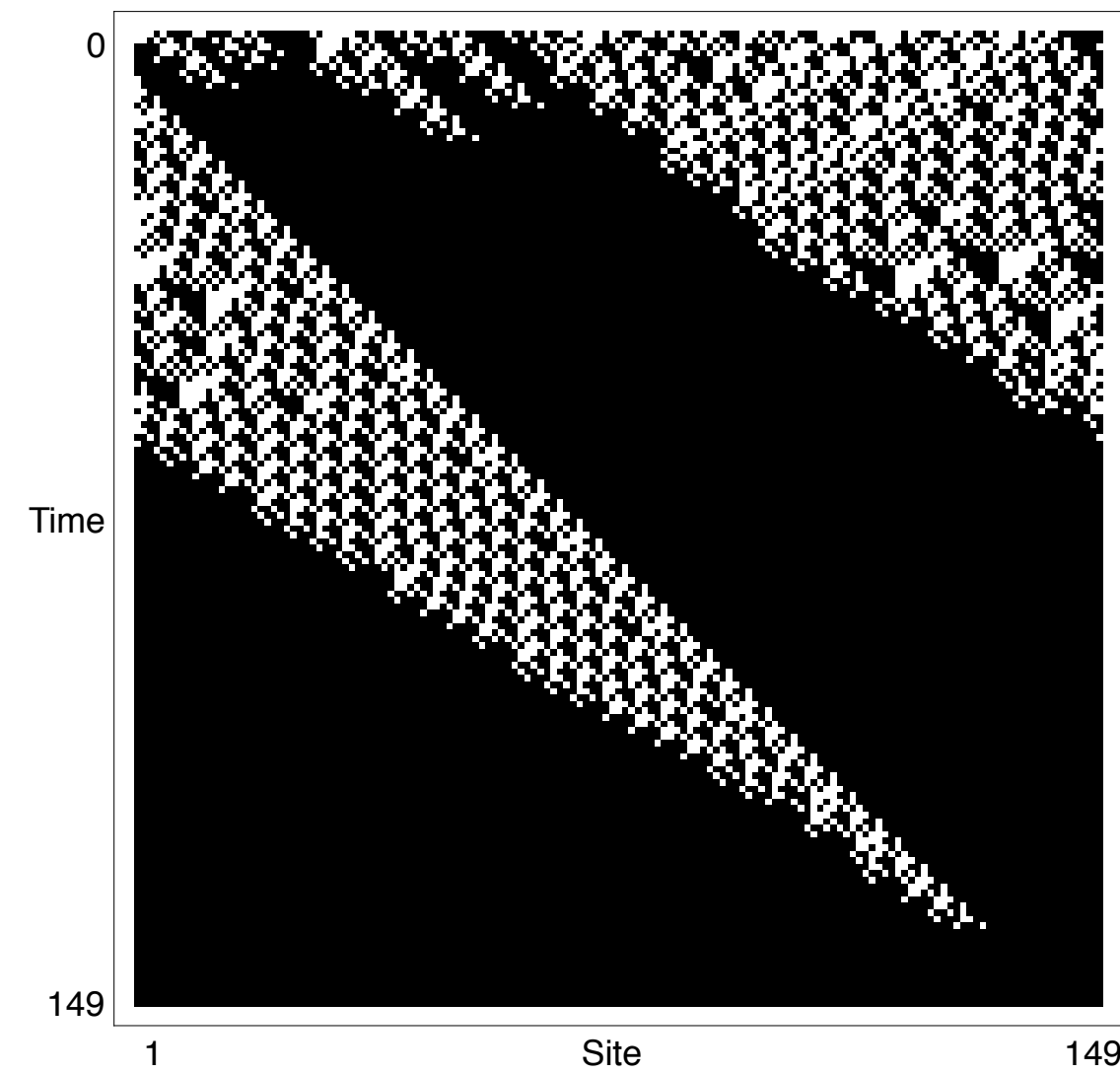
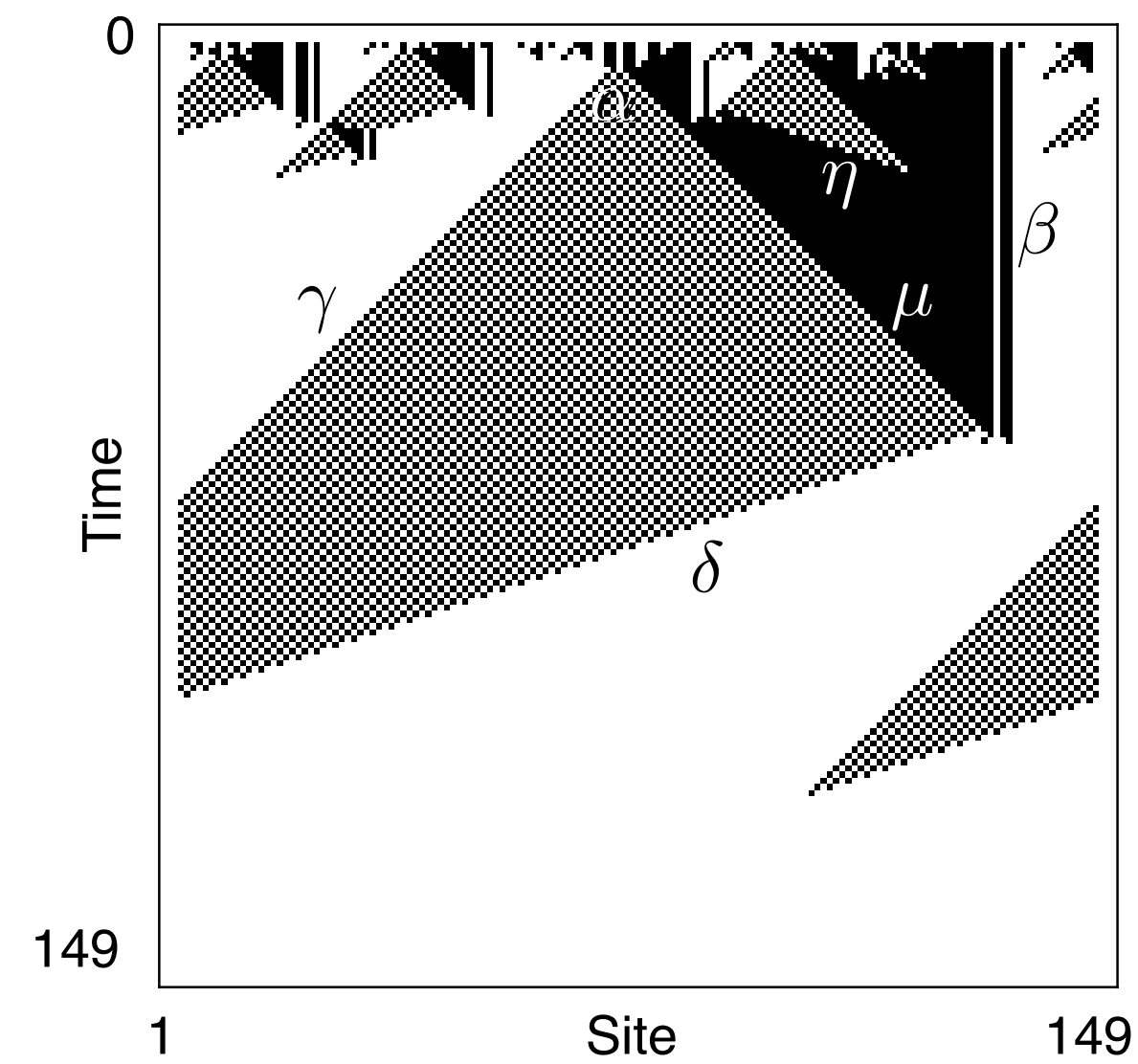
Cellular Automata

The majority classification task

- We have an odd number of players
- Whatever random selection of cards there is always one majority colour
- Can we figure out a rule that makes all players show the majority colour?
- Why is this *complex*?
- What do we know about this special game?
 - The rule needs to consider three neighbours on either side
 - There is no perfect solution

Cellular Automata

The majority classification task



Cellular Automata

The majority classification task

- How can we think about the majority classification problem as a search problem?
- What are the nodes in my search graph?