0. ABSTRACT

1. INTRODUCTION

- background of the problem
- general information, related work and state of the art.
- motivation, why biological , similarity with nature

1.1 Cellular Automata

Cellular computing is an unconventional model that attempts to go beyond the conventional paradigms such as today's popular von Neumann architecture. It is a biologically inspired method that draws mechanisms and behaviours from natural phenomena, therefore, it is a common point of interdisciplinary knowledge where biology, chemistry and computer science meet.

These new cellular models can build complex systems just by creating copies of the same fundamental unit: the cell. A cell by itself cannot do much work, but when teamed up together with other cells, the whole system is capable of advanced computation. Every cell is limited to local interaction i.e. every cell can only influence the fate and behaviour of its neighbours.

- explain other types of CA

The result is a system with three main characteristics: [The Emergence of Cellular Computing]

- Simplicity:

The basic computational unit of a CA is a cell. It is a simple unit capable of a very limited number operations.

- Vast Parallellism:

Most of today's parallel computer contain around a few dozen processors. Cellular computing means parallelism on a much bigger scale, often involving exponential number of cells.

- **Locality:**

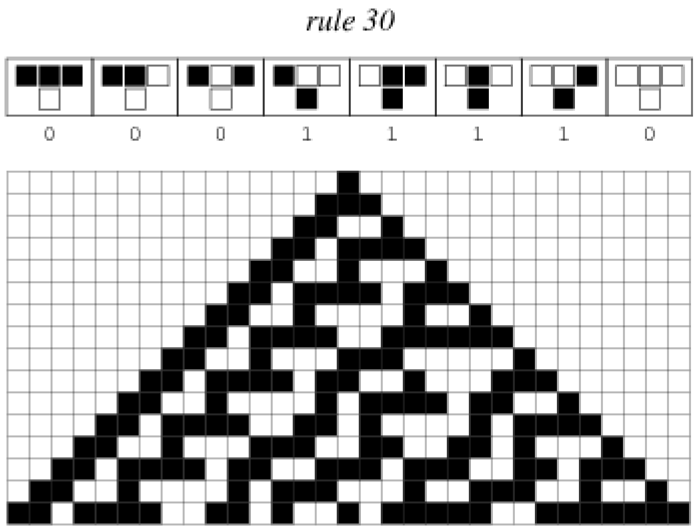
Each cell is only allowed to interact with its immediate eighbours and his connections carry a very minimal amount of information. In CAs there is no main controller or “brain” that has an external view of the entire system

In order to define and study CA, we will consider as genotype the state transition table (common to each cell) and as phenotype measurements such as attractor length (section 2.1)

- Hypothesis: Maybe some parameters that can be computed directly from the genotype of a CA can speed up the process of creating more complex organisms.

First Implementation: Uniform 1D CA:

- array of N boolean states
- update function FIXED
- strong locality (left, middle and right)



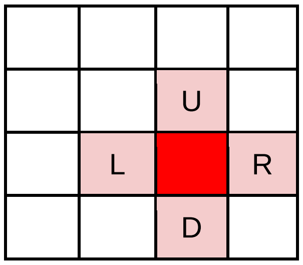
pic : http://mathworld.wolfram.com

Undergo testing:

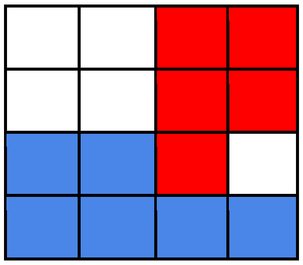
- **Compare to 250 Wolfram’s rules**

Second Implementation: Uniform 2D CA:

- NxN matrix of 3 states (0 = dead, 1 = type 1, 2 = type 2) to preserve multicelularity
- update function FIXED
- locality (up, down, right, left, middle)
- details in experimental setup!!!
- von neumann neighbours (5)
- *torus structure (cant representinfinity)*



Step 0



Step N

red = type 1 blue= type 2 white = dead

Undergo testing:

- **basic cases**
- **game of life**

1.2 Development Process: Genetic Algorithms

Genetic Algorithms (GA) is a search heuristic inspired by the natural process of biological evolution. Solutions to a problem are identified as organisms of a population, and a fitness function can resolve how well an organism solves the given problem.

Every generation, the fitness of every organism is evaluated with the fitness function, and then the more suitable solutions are selected to produce offspring by recombining (and maybe also modifying) their genotype to form the next population.

This process is repeated multiple times until a maximum number of generations is achieved or a valid solution is found

Limitations of GA

GA are far from perfect, here are some limitations

1. If the problem requires a computationally expensive fitness function, the cost of the algorithm is prohibitive

2. GA do not scale well with complexity. If the search space is too big, the algorithm will not find exact solutions. One typical technique to avoid this issues is to break down the problem into simpler, smaller problems.

3. GA may converge to local optima, that is, following blindly the fitness function and ignoring other paths that may lead to better solutions. This can be averted by using a finer fitness function or by using the mutation operation (at the cost of complexity and efficiency).

-> find references

1.3. Developmental systems

- coding everything in the genotype can be too computationally expensive
- nature does not store in the DNA all the details of the organism, only guidelines on how to build one.
- Map the genotype with a more complex phenotype, but only the genotype passes on to the next generation

2. MOTIVATIONS

- study complexity - what is complexity? [kowaliw]
- study methods of measuring this complexity
- bend this complexity to desired results (parameters)

- study complexity in phenotype. get information from genome to predict complexity. use parameters to predict emergent complexity

3. REVIEW

3.1 Trajectories and Attractors

- what they are
- why we are studying them - the edge of chaos
- trajectory = full path
- attractor = loop

3.2 Parameters

paper with other parameters

- why introduce parameters?
- what do parameters measure?
- Lambda vs Majority

3.4 The Lambda Parameter

- what does Lambda represent? rough interpretation + mathematical definition

3.3 The Majority Parameter

- what does M represent? rough interpretation + mathematical definition

4.EXPERIMENTAL SETUP

- 2D Cell automata
- Genetic algorithm
    - use M/lambda parameter inside the fitness function (50% attractor length, 50% M?)
        - push M/lambda towards 0
        - push M/lambda towards theoretical optimum (0.3/0.6)
    - use M/lambda to discard unwanted individuals for the next generation

low population (10)
generations = 50k
20 runs

5.RESULTS

6.CONCLUSIONS

7. REFERENCES