Contents

1 Overview

Problem Type	GA/GP/MA	NN	ACO	PSO	CA	MC
Optimisation	✓		(✓)	✓		
Machine Learning	✓	✓	(✓)	✓		
Control	✓	✓	(✓)	✓		
Simulation	(✓)				✓	✓

Table 1: Suitability of algorithms to problems

GA Genetic Algorithm

GP Genetic Programming

MA Memetic Algorithm

NN Neural Network

ACO Ant Colony Optimisation

PSO Particle Swarm Optimisation

CA Cellular Automata

MC Membrane Computing

2 Genetic Algorithms

2.1 Biological Inspiration

Natural selection

Principle that every slight change in a trait that is beneficial is preserved.

Individuals that have traits that allow them to be better adapted to the environment are more likely to reproduce, traits are then passed to later generations.

Genetics

Candidate solutions to a problem represented in a chromosome composed of several genes.

Genes are passed from generation to generation with small changes (mutations).

2.2 Overview

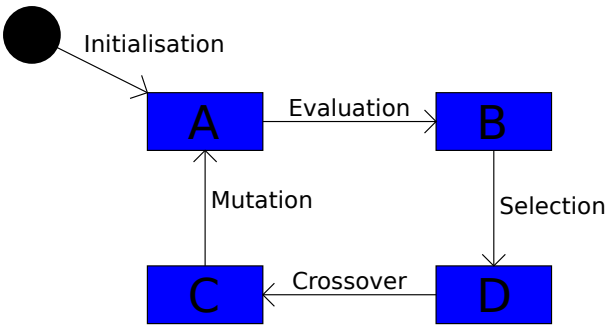


Figure 1: Genetic Algorithm Workflow

Pseudocode

```
population = init_random_population()
iteration = 0
while iteration < num_interations:
  evaluate(population)
  population = selection(population)
  population = crossover(population)
  population = mutation(population)
  iteration++
best = get_best_individual(population)
return best
```

Listing 1: Genetic algorithm pseudocode

2.3 Population

- Set of possible solutions to the problem
- Most often a set (chromosome) of variables (genes)
- Initial population created at random

2.4 Evaluation

- Giving a "goodness" value to each candidate solution
- Uses a fitness function which takes a candidate solution and determines how well it solves the problem

2.5 Selection

- Choosing individuals to be in the next population
- Rewards best individuals (i.e. those with the best fitness values)

2.5.1 Roulette Wheel Selection

- Probability of selection is proportional to fitness of individual
- Make as many selections (spins of wheel) as individuals to be selected
- Individuals may be selected multiple times

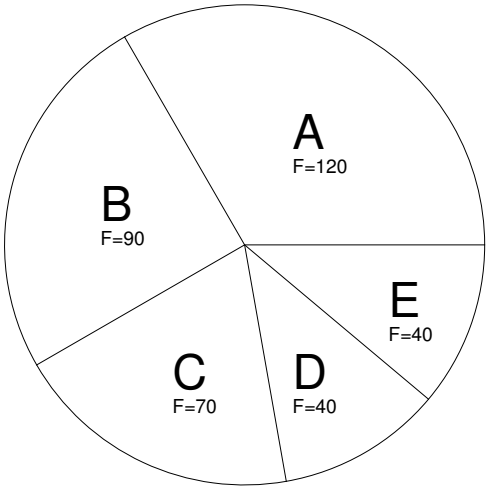


Figure 2: Roulette Wheel Selection

2.5.2 Stochastic Universal Sampling

- Similar to Roulette Wheel Selection
- Do one spin but divide "pointer" into as many individuals to be selected

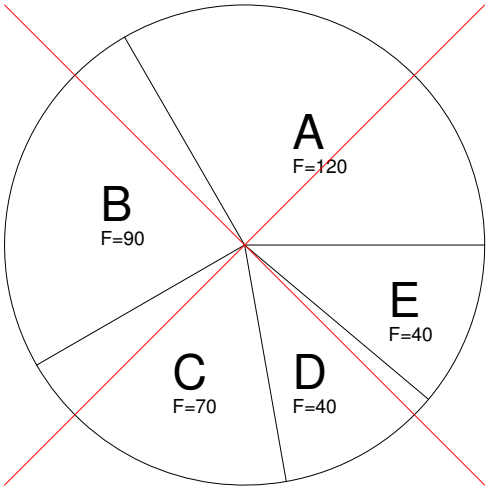


Figure 3: Stochastic Universal Sampling

2.5.3 Tournament Selection

- Select the best individual from a randomly selected subset of the population

```
new_population = []
while population.size() > 0:
  tournament = select_random_subset(population, tournament_size)
  tournament = sort_by_fitness(tournament)
  new_population.add(tournament[0])
return new_population
```

Listing 2: Tournament selection pseudocode

2.5.4 Truncation Selection

- Keep only the best n individuals in a population

2.5.5 Comparison of selection methods

Roulette Wheel & Stochastic Selection

- Fitness proportionate
- Chance of being selected is proportionate to fitness value
- Selection becomes random when fitness values are close (e.g. in later GA iterations)

Tournament & Truncation Selection

- Rank based
- Best individual will always win a tournament
- More stable selection pressure

2.6 Crossover

- Exchanging genes between two individuals
- Takes two individuals from the population and generates two offspring
- e.g. For a bit array, one offspring takes first section of array and second of another, and vice-versa
- e.g. For numerical genes the blend alpha operator picks a random value between the two parent genes

2.7 Mutation

- Making small/subtle modifications to an individual
- Probability P_m of mutation can be set either per chromosome or per individual
- e.g. Randomly flipping a bit where the chromosome is a bit array
- e.g. Adding a random value to a numerical gene

2.8 Replacement

- Alternative to generational genetic algorithm
- Steady state genetic algorithm
  - Elitism
  - Selection chooses two parents who produce two offspring
  - Offspring are inserted into the parent population, replacing the two individuals with lowest fitness

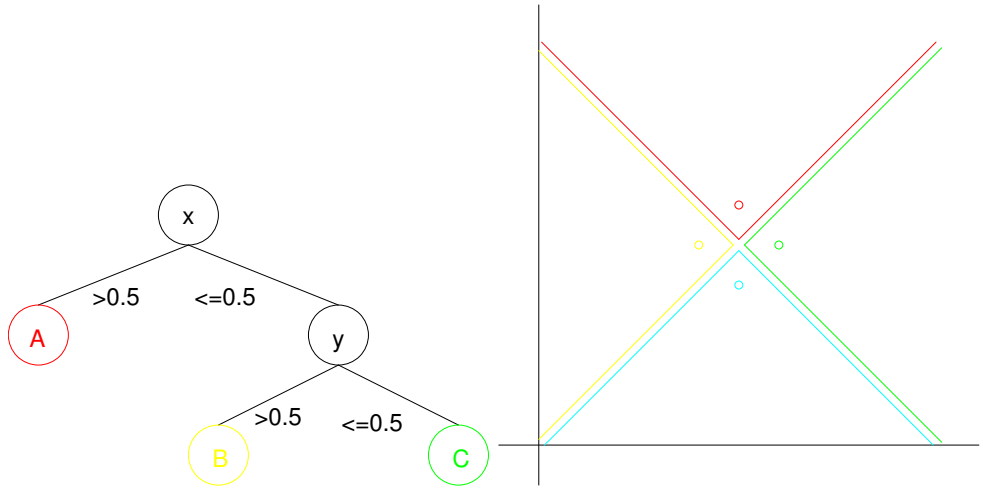
2.9 Knowledge Representation

Nominal attributes:

- Set of rules and logic predicates

Real valued attributes:

- Hyperrectangle (n dimension rectangle)
- Hyperellipsoid (n dimension circle)
- Decision trees
- Synthetic prototypes (e.g. nearest neighbour)
- Linear classifier (separate instances of classes for classification problems)



(a) Decision Tree (b) Nearest Neighbour

Figure 4

2.10 Tuning

- Required to ensure a proper evolutionary process
- GA learns well when exploration and exploitation are balanced

  Exploration Directing population to unknown areas of the problem space

  Exploitation Directing the population to most promising (based on fitness) parts of the problem space
- Too much exploitation can lead to convergence to a sub-optimal solution (premature convergence)
  - If population is too similar then crossover operator no longer provides effective exploration
- Too much exploration risks:
  - Slowing down the learning process
  - If probabilities are too high then crossover and mutation may harm the overall population fitness
- Innovation time t_i is the average time taken to create an individual with better fitness than the current best individual

2.11 Machine Learning

Supevised learning

- Learning to solve a problem
- If output is discrete →Classification (output value known as a class)
- If output is continuous →Regression
- Use a training set to train the genetic algorithm and a testing set to verify the generated model
- Optimise initialisation by creating initial population based on training data

Unsupervised learning

- Identifying patters/clusters

2.12 Parallel Genetic Algorithms

- Genetic algorithms tend to be slow
- Majority of genetic operations can be parallelised (all but crossover)
- Several systems exist for this:

  Master-Slave model GA cycle is is run on master, slaves perform operations
  Island Model Population is divided across nodes
  Cellular GA Population distributed across 2D lattice

3 Genetic Programming

- Very similar concept to Genetic Algorithms 2
- Instead of evolving a solution, evolve a program

3.1 Program Representation

- Classic method is to represent programs as a tree representing a formula
- Modern methods can also create other representations
  - e.g. Cartesian Genetic Programming creates graphs
- Internal nodes of tree are functions/operations
- Leaves are either variables/parameters or constants
  - Constants can be predefined or assigned a random value within a predefined range

Example

Figure 5 shows a tree representing the equation:

$$\frac{4 + P3}{P1 - sin(P2)}$$

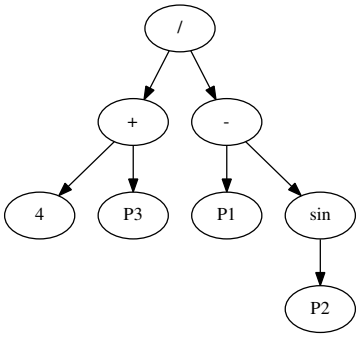


Figure 5: Example Tree

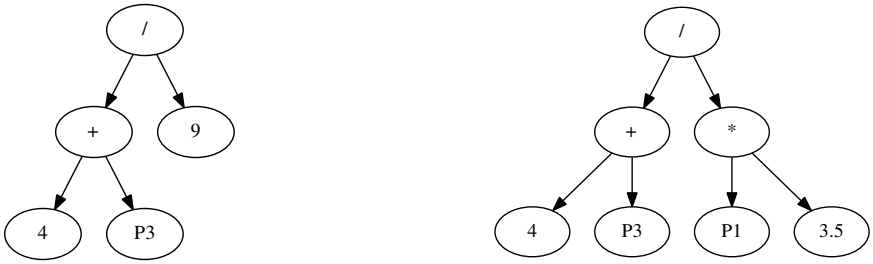
3.2 Initialisation

- Initial tree is generated randomly
- Maximum depth of tree is predefined
- Strategies for generation:

  Grow
  Generate a tree at random with leaves up to the maximum depth

  Fill
  Generate a tree at random with all leaves at the maximum depth

  Hybrid
  For each level of depth, initialise a uniform number of trees, half of which using Grow and half using Full



(a) Grow (b) Fill

Figure 6: Tree initialisation

3.3 Evaluation/Execution

- Evaluation is done by averaging error over all test cases, giving a fitness value
- Recursive execution is easy but inefficient
- Converting a tree to Reverse Polish Notation solves problem

```
for each element e:
  if e is value:
    push e to the stack
  else if e is operator:
    pop as many elements from the stack as the operator has parameters
    execute the operator
    push the result to the stack
return value at top of stack
```

Listing 3: Genetic programming execution pseudocode

3.4 Crossover

Exchange subtrees between parents.

Example

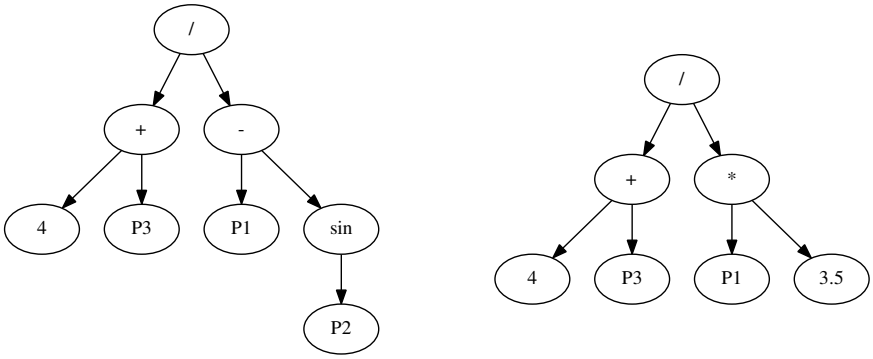


Figure 7: Tree crossover parents

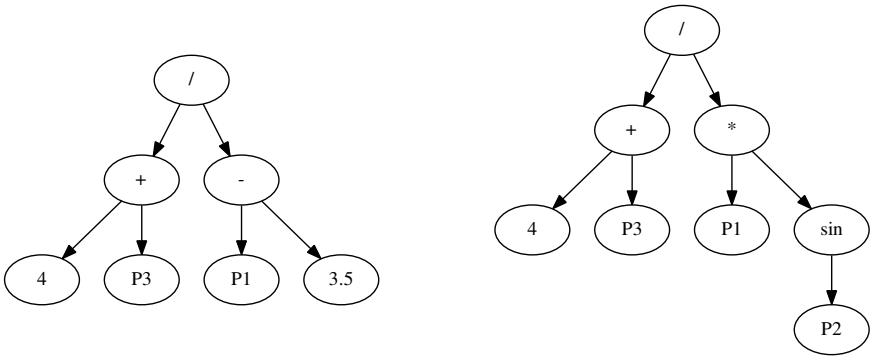
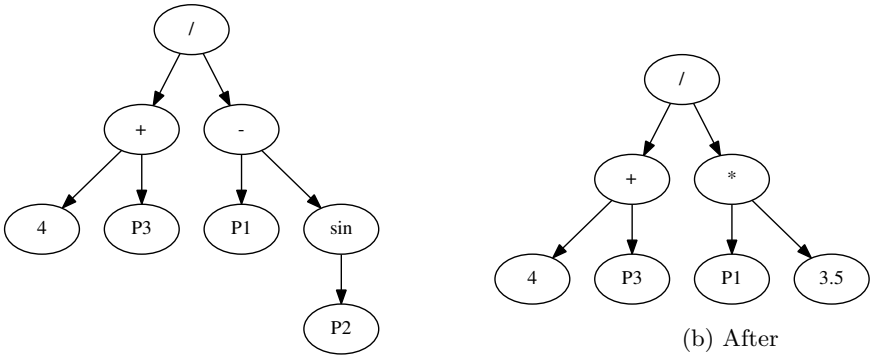


Figure 8: Tree crossover children

3.5 Mutation

Replace entire subtrees with a randomly generated subtree.

Example



(a) Before

(b) After

Figure 9: Tree mutation

3.6 Bloat

- Growth of program tree without improvement in fitness
- Bloat can affect any evolutionary paradigm where the representation has variable length
- Solutions:

  Restrict tree depth
  May affect evolutionary process making it unable to find a good solution

  Simplification
  Very difficult to do efficiently

  Add fitness penalty to large trees
  Smaller fitness for solutions with larger trees

  Consider tree size in selection
  When two individuals have equal fitness, prefer the smaller tree

3.7 Applications

Regression
Reverse engineering a mathematical formula given a dataset of output values for given input values.

This gives an approximation of the original formula that created the dataset.

Generating Electronic Circuits
Tree topology defines schematic.

Leaves define component values (inductance, capacitance, etc.).

Control
Optimising gains of a PID controller.

Puzzle Solvers
e.g. FreeCell solver using hybrid genetic algorithm and genetic programming.

Generation of Audio Synhesizers
Using Cartesian Generic Programming to generate a graph structure.

4 Neural Networks

- Connection of perceptrons
- Perceptron
  - Set of inputs from other perceptrons
  - Activation/transfer function
  - Output value
- Connection
  - Weighted
- Used for data problems (classification, regression, control)

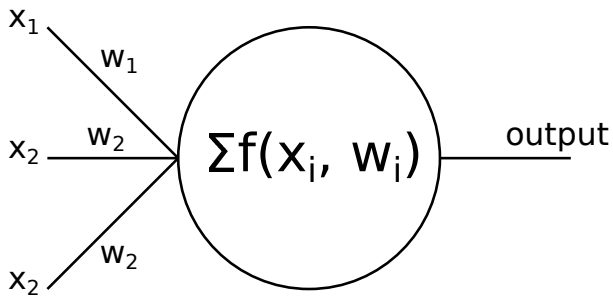


Figure 10: Perceptron

4.1 Multi Layer Perceptron

- Multi layer neural network where every perceptron in a layer is connected to every neuron in the adjacent layers
  - A single input layer with as many neurons as they are input parameters
  - Several inner layers
  - A single output layer with as many neurons as they are outputs
- Each layer represents a combination of features
  - e.g. phonemes → words → sentences
- A non linear multi layer perceptron with a single inner layer with enough neurons can learn any continuous function (universal approximation theory)

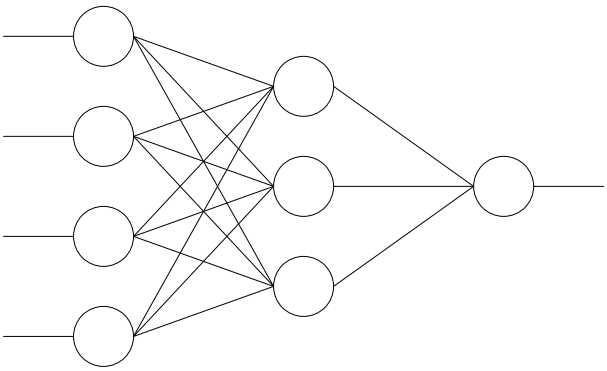


Figure 11: Multi Layer Perceptron

4.1.1 Learning Task

- Supervised learning given a set of inputs and the expected output
- Optimise the error function 2 to obtain the smallest value of E by selecting new values of the weights W
- Computationally difficult to do using traditional optimisation methods (e.g. steepest-descent, Newton, etc.)
- Classic method for MLP is using backpropagation algorithm
  - Two pass approach
  - Calculate error of output layer
  - Calculate weight updates of next layer back in turn

$$E = \frac{1}{n}\sum_{t=1}^{n} E(x') \tag{1}$$

$$= \frac{1}{n}\sum_{t=1}^{n} (F(x', W) - y_t)^2 \tag{2}$$

4.2 Self Organised Maps

- a.k.a Kohonen neural network
- Unsupervised neural network
- Maps data from a high dimensionality to a 2D lattice

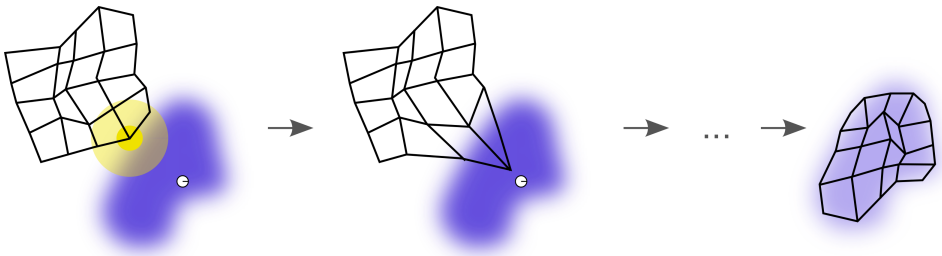


Figure 12: Self Organising Map Training

Training:

1 A new input vector is selected

2 The node with a weight vector closest to the input vector (best matching unit) is found

3 The weight vector of the BMU and it's neighbours is modified to bring them closer to the input vector

4.3 Deep Learning

- A neural network with lots (≥ 10) inner layers
- More inner layers allows construction/recognition of higher level features

Types:

- Purely supervised (deep NN)
  - Trained using classic methods (e.g. backpropagation)
  - Results in very slow training times due to network size
- Semi-supervised (deep belief networks)
  - Build inner layers incrementally in an unsupervised way
  - Add a final supervised layer
- Generative methods (convolutional NN)
  - Each layer applies a "filter" to the data to reduce its dimensionality
  - Commonly used with image/signal processing

4.3.1 Training semi-supervised networks

1 Train first layer in an unsupervised manner

2 Fix first layer parameters and train second layer using the output of the first layer as unsupervised input to the second

3 Repeat for all inner layers

4 Use the outputs of the final layer as inputs to a supervised layer and train using the raining data outputs

5 Free all parameters and train entire network using a supervised approach using existing weights as initial values

4.3.2 Restricted Boltzmann machine

- Unsupervised neural network that learns a probability distribution over a set of inputs
- The result is it will learn to recognise recurrent patterns

4.3.3 Convolutional Neural Network

- Each layer combines "patches" from the previous layer
- Compresses larger problems into smaller sets of features
- Filters are usually manually defined (i.e. not learned)
- Output of filtering is used to train a supervised model
- Requires neighbourhood regularity in input data (i.e. areas of a certain colour in an image)

5 Memetic Algorithms

- Derived form the concept of traits spreading from person to person
- Combines conventional genetic algorithm (global search) with local search
- Global search allows exploration of entire problem space
- Local search allows exploration of neighbourhood of each individual to attempt to improve its fitness

Pseudocode

```
population = init_random_population()
iteration = 0
while iteration < num_interations:
  evaluate(population)
  for i in range(len(population)):
    population[i] = select_local_best(population[i])
  population = selection(population)
  population = crossover(population)
  population = mutation(population)
  iteration++
best = get_best_individual(population)
return best
```

Listing 4: Memetic algorithm pseudocode

5.1 Motivation

- When decomposing complex problems, subproblems may be easier to solve using either GA or local search
- Fast to reach optimal solution using domain specific knowledge in local search operator
- Domain knowledge in local search operator can "repair" bad individuals found by a GA

5.2 Design considerations

Baldwinism vs Lamarckism

Two alternative theories for transfer of traits between individuals of a population.

Baldwinism

- Traits acquired by an individual can be passed to its offspring
- e.g. replace an individual with its fitter neighbour

Lamarckism

- Traits acquired by an individual cannot be passed to its offspring
- e.g. individual inherits fitness but not genotype

When to apply local search?

- After selection
- After crossover
- After mutation

In which stage of the genetic algorithm lifecycle?

- Uniformly throughout all iterations
- More in earlier or later iterations

Scope of local search?

- Entire population
- Probability of local search (fitness proportionate)
- Only individuals with specific range of fitness values

Choice of local search operator

- Use multiple, each of which having a fixed probability of being applied
- Have a fixed mapping of individuals to operator (domain specific)
- Evolve the operator choice using a GA

Other design issues

- Size of neighbourhood for local search
- Size of variations to make during local search operation

Diversity control

- Memetic algorithms naturally favour exploitation over exploration
- To ensure good evolution additional exploration must be done
- One option is to increase local search neighbourhood

6 Swarm Intelligence

- System of "items" collaborating to solve a task
- No external guidance/arbitration/synchronisation

6.1 Ant Colony Optimisation

- Key concept is stigmergy, the indirect communication between individuals in the system through the environment
- In ants this is done though a pheromone trail deposited by ants as they explore a path
- Ants will always prefer to follow the path with the strongest pheromone trail
- Can be applied to any problem assuming it can be represented as a graph
  - e.g. Ant-Miner: An unsupervised machine learning framework that constructs rules that identify patters in a set of input data.

6.1.1 Example: Travelling Salesperson Problem

Close problem to actual ant colony movement.

- Graph $G(N, E)$: N are cities and E are routed between cities
- $d_{i,j}$ is the fixed cost of travelling between cities i and j
- Edge cost is given by a combination of:
  1. Static cost: $\eta(r, s) = \frac{1}{d_{r,s}}$
  2. Dynamic trace: $\tau(r, s)$ deposited by ants (initially zero)
- All ants must visit all nodes in an iteration (a tour)
- At the end of an iteration all edges that are on the best tour receive additional pheromone
- At each iteration the pheromone count for all nodes may be reduced to reduce the chances of unpopular edges being selected in a tour
- Stopping criteria is either stagnation or a maximum number of iterations being performed.

Pseudocode

```
graph = init_graph(d)
place_ants()
while not stopping_criteria:
  while not all_cities_visited:
    for a in ants:
      move_to_next_city(a)
  place_ants()
  graph = update_graph(tau)
return graph
```

Listing 5: Ant colony optimisation for TSP pseudocode

6.2 Particle Swarm Optimisation

- General purpose optimisation for continuous variables
- Each particle searches for the optimum
- Each particle is moving with a velocity
- Each particle knows its last position and the position of its best result so far
- A particle has a neighbourhood associated with it
- A particle knows the position of the most optimal particle in its neighbourhood

6.2.1 Neighbourhood types

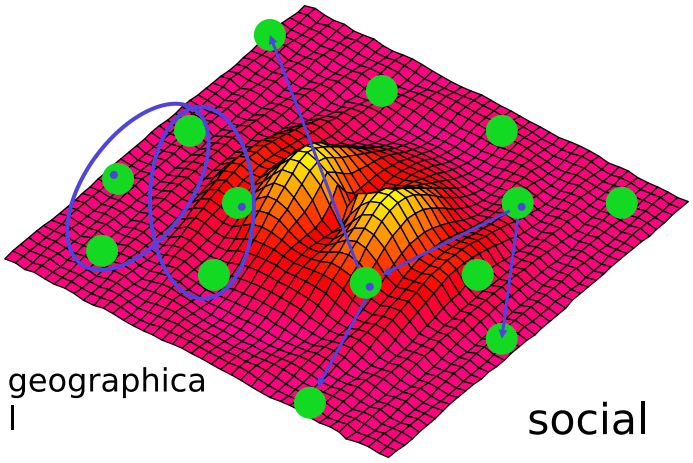


Figure 13: Geographical and social neighbourhoods

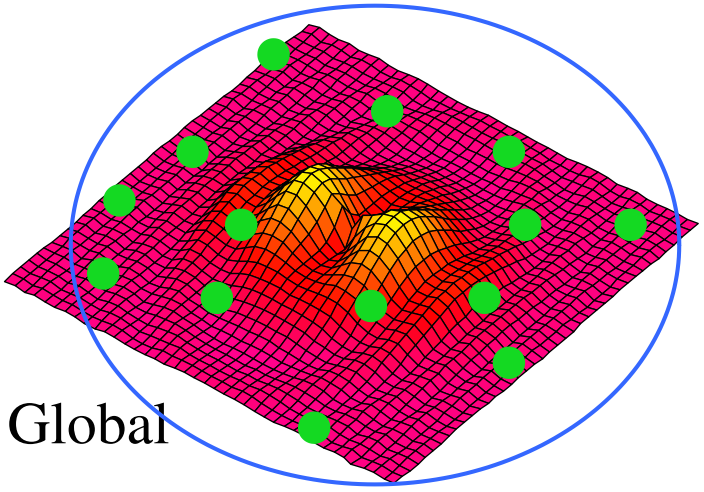


Figure 14: Global neighbourhood

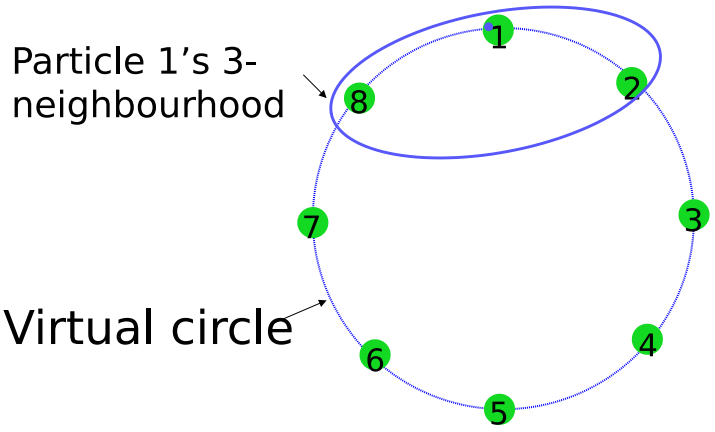


Figure 15: Circular neighbourhood

6.2.2 Algorithm

- In each time step a particles velocity and position are updated as per equations 3 and 4
- Particle velocity is limited to a fixed $max\_velocity$

$$\begin{aligned} velocity = a * velocity + \\ b * (neighbourhood\_best - position) + \\ c * (local\_best - position) \\ velocity = max(velcoity, max\_velocity) \end{aligned} \tag{3}$$

$$position = position + velocity \tag{4}$$

Pseudocode

```
particles = random_initial_particles()

while not stopping_criteria:
  for p in particles:
    fitness = evaluate(p)

    if fitness > p.best:
      p.best = fitness
    if fitness > global_best:
      global_best = fitness

  for p in particles:
    update_velocity(p)
    update_position(p)
```

Listing 6: Particle swarm optimisation pseudocode

7 Cellular Automata

- Biological inspiration is cells and cell reproduction
- Idea of a self replicating system
- Discrete systems that model complex behaviour using simple logical rules
- Collection of cells
  - Oriented in an n-dimensional grid
  - Have a state (taken from a finite set)
  - Have a neighbourhood
  - State of cell in next time step is computed using rules
  - Rules define states based on current state of cells in neighbourhood
- A cellular automation is evaluated generation by generation until some stopping criteria are reached or it is stopped manually

7.1 1D eight rule CA

- 1 dimensional gird
- Two states: 1 or 0
- 8 rules
- 256 possible rule sets

Classification of behaviour for different rule sets:

Class 1: Uniformity
Rapidly converge to a uniform state

Class 2: Repetition
Rapidly converge to a repetitive stable state

Class 3: Random
Appear to remain in a random state

Class 4: Complexity
Form areas of repetitive or stable sates while also forming structures which interact with each other in complex ways

7.2 2D cellular automata

- Very similar to 1 dimensional CA but with larger neighbourhood
- Larger neighbourhood means more complex rules (due to larger number of possible neighbourhood states for a given cell)
- Rule set may become too large to explicitly define

7.2.1 Neighbourhoods

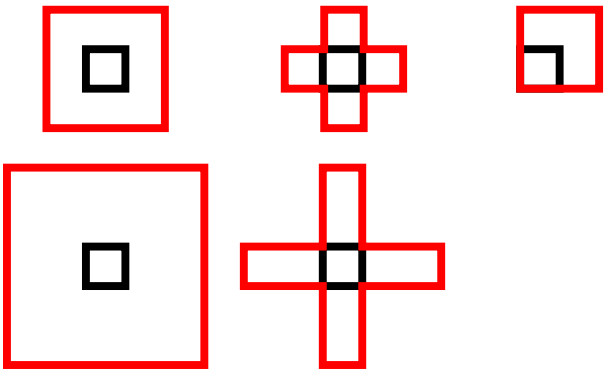


Figure 16: Examples of neighbourhoods

Left to right, top to bottom:

- Moore
- Von Neumann
- Margolus
- Extended Moore
- Extended Von Neumann

When a neighbourhood is at the edge of the 2D matrix it can either be cropped to form a smaller neighbourhood or wrapped around the matrix.

7.2.2 The Game Of Life

Goals

A "lifelike" result with a simple set of rules.

1 There should be no initial patters that can be proved to allow a population to grow indefinitely

2 There should be initial patters that appear to grow without limit

3 There should be initial patters that grow/change before coming to an end in one of three ways:

- Fading away completely (overcrowding or too sparse)
- Settling into a stable state that does not change between generations
- Entering an oscillating state

Rules

Whether a cell is on or off (lives or dies) in the next iteration is determined by the following rules:

Loneliness
A live cell with fewer then two live neighbours dies.

Companionship
A live cell with two or three live neighbours lives.

Overcrowding
A live cell with more then three live neighbours dies.

Reproduction
A dead cell with exactly three live neighbours becomes live.

Constructs

Groups of cells that exhibit specific behaviour.

Still life
Constructs that have reached a state that will not change between generations without interaction from other constructs.

Oscillating constructs
Constructs that cycle between two or more states over a given number of generations.

Moving constructs
Constructs that move across the grid over time, usually by means of an oscillating set of states.

Computation

Can perform simple computations using glider guns to create "sliding block memory" from which logic gates can be simulated.

Can create state machines (using sliding block memory as a counter) to form a universal Turing machine.

7.3 Variants

3D cellular automata
Same principal as 1D and 2D but with a larger neighborhood.

Different cell shapes
Assuming a shape can be oriented on a grid (tessellated) and assigned a neighbourhood then it can be used in a cellular automation.

e.g. Triangles and hexagons have been used.

Probabilistic rules
Have a set of rules that have probability weightings instead of being purely deterministic.

Continuous cellular automata
Using continuous values rather than discreet states.

Nested cellular automata
Used to simulate a hierarchical system.

e.g. County $\rightarrow$ City $\rightarrow$ Person $\rightarrow$ Cell

7.4 Applications

Computer graphics
Performing filter computations to an image.

e.g. Blurring is an operation that modifies each pixel depending on the states of the pixels in its neighbourhood.

Game development
Used for world generation, given a random seed (states in initial generation).

Cryptography
One way function used for public key encryption.

Given a rule set it is easy to calculate future states given an early generation, but very difficult to calculate past states given a later generation.

Simulation

Various biological and physical simulations.

e.g. Pattern formation, cell development, fluid flow, etc.

Fractals

Generation of patterns based on simple self replication rules.

8 Membrane Computing

8.1 Metaphor

- Cell is a hierarchical set of compartments
- In each compartment certain chemical reactions happen
- Under certain conditions, elements 9molecules) can move between compartments

8.2 Algorithm

- Paradigm designed for simulation
- A simulation consists of:
  - A set of hierarchical compartments
  - A set of objects that can exist in each compartment and units of each object existing in each compartment
  - Rules that can transform some objects into others and move objects between compartments
  - Stochastic constant (likelihood of activation) for each rule
- In a simulation the rules are applied to a given starting point (quantity of each object in each compartment)
- Multiple runs are usually performed and results combined

9 DNA Computing

Performing computations on biological devices, specifically DNA.

9.1 Structure of DNA

- Asymmetric double helix
- Complimentary strands A-T and C-G
- $5'$ denotes the front of a motif (upstream)
- $3'$ denotes the end of a motif (downstream)

9.2 Features useful for computation

- Massive parallelism
  - Small amount of DNA contains many strands
  - When strings are randomised then a small volume of DNA will contain a large number of unique strings
- Complementarity
  - Know that if a string binds to a larger string then the larger string must contain the complementarity string to the smaller string
  - e.g. if $5'$ ACGT $3'$ binds to a longer string S, you know that S contains the string $3'$ TGCA $5'$

9.3 Experimental Techniques

9.3.1 Polymerase Chain Reaction

Used to replicate DNA, creating many copies of the base pairs.

1 Separate two base strands at low heat

2 Add base pairs, primer sequences and DNA polymerise

- Creates double stranded DNA from single strand
- Primer sequence creates seed from which the double stranded DNA grows

3 Repeat, DNA grows exponentially

9.3.2 Electrophoresis

- Phosphate backbone of DNA is negatively charged
- Migration of DNA to agaraose gel shows visible pores on the gel surface
- An electric field is placed over the DNA to force migration
- Size of DNA fragments is determined by pore size

9.4 Solving the Hamiltonian Path Problem

Given a directed graph $G = (V, E)$, V_{start} and V_{end}, is there a directed path starting at V_{start} and finishing at V_{end} that visits all vertices exactly once.

9.4.1 Non-deterministic approach

1 Generate many random paths through G

2 Keep only paths that start at V_{start} and end at V_{end}

3 Keep only paths with length $n = |V|$

4 Keep only paths that visit each vertex once

5 If any paths remain then result is true, if no paths remain result is false

Exploits:

Massive parallelism to take care of non-deterministic nature of algorithm

Complementarity is used to select and filter solutions

9.4.2 Stage 1: Initialisation

Vertex and edge encoding

- A vertex is encoded by a 20-mer length string
- An edge is encoded by the back 10-mer and the front 10-mer from the two connected vertices
- Edges for V_0 (V_{start}) and V_n (V_{end}) contain the full string for the start and end vertices (i.e. so are 30-mer or 40-mer)

Building random paths

50 picomol of the complimentary sequence (v') (except V_0 and V_n) is mixed with 50 picomol of each edge encoding

9.4.3 Stage 2: PCR amplification

- Implements step 2
- Amplify using primers: V_0 and V_n'
- PCR will replicate everything with V_0 and V_n at the ends

9.4.4 Stage 3: Selecting paths of length $|V|$

- Double stranded DNA that represents paths going through exactly n vertices are selected
- Based on size measured through electrophoresis

9.4.5 Stage 4: Selecting paths that visit all vertices

- Double strands from stage 3 are denatured
- Put into a mix with V_1' which are magnetically attached to beads
- Strands containing V_1 anneal to V_1'
- Strands not containing V_1 are washed away
- Repeat for all vertices (except V_0 and V_n)

9.4.6 Stage 5: Result

- Remaining strands after stage 4 undergo PCR and electrophoresis
- If any strands remain in the gel then G has a Hamiltonian path

9.5 DNA origami

- Forcing DNA to form complex shapes and structures by giving it a certain sequence
- Structures formed of:

  Scaffold
  : A long single stranded DNA string

  Staples
  : A set of short unique strings that bind to the scaffold in specific places

- Designed using the De Brujin sequence