ADVANCED DATA MINING

Dr. Jason Roche { slides courtesy of Dr. Simon Caton}

Genetic Algorithms

National College of Ireland, School of Computing

www.ncirl.ie

Sources for this lecture

- Mitchell, T. M. (1997). Machine learning. McGraw-Hill Boston, MA
  - Chap 9

- Online Resources:
  - http://cran.r-project.org/web/packages/genalg/index.html
  - http://www.r-bloggers.com/genetic-algorithms-a-simple-r-example/
  - http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3
  - http://lancet.mit.edu/mbwall/presentations/IntroToGAs/
  - http://www.ai-junkie.com/ga/intro/gat1.html

- Other References:
  - Dueck, G., & Scheuer, T. (1990). Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, *90*(1), 161-175.
  - Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, *3*(2), 95-99.

GAs 101

- Hypothesis driven evolutionary learning mechanism
- “Good” hypotheses / solutions evolve over time

- A set of hypotheses is called a population

- Time is modelled as a discrete simulation (iterations)
- In each iteration / time period the population is evolved in accordance to Darwinistic notions of natural selection

- There are 2 key operators in GAs: Crossover, and mutation

- Huge number of use cases:
  - Optimisation problems e.g. parameterising another data mining method (e.g. ANNs)
  - Learning controls for robots
  - Learning computerised solutions to problems, i.e. genetic programming

(Mitchell, 1997 [Chap 9])

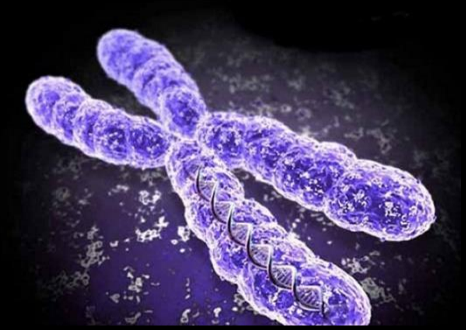
GA in a nutshell

GA’s are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics.

GA’s simulate the survival of the fittest among individuals over consecutive generation for solving a problem.

It works with a population of solutions in parallel.

Each generation consists of a population of solutions, and each solution has a set of properties that are analogous to the chromosome that we see in our DNA.



GA Overview

- Formally introduced in the United States in the 1970s by John Holland
  - based on the process of evolution by natural selection

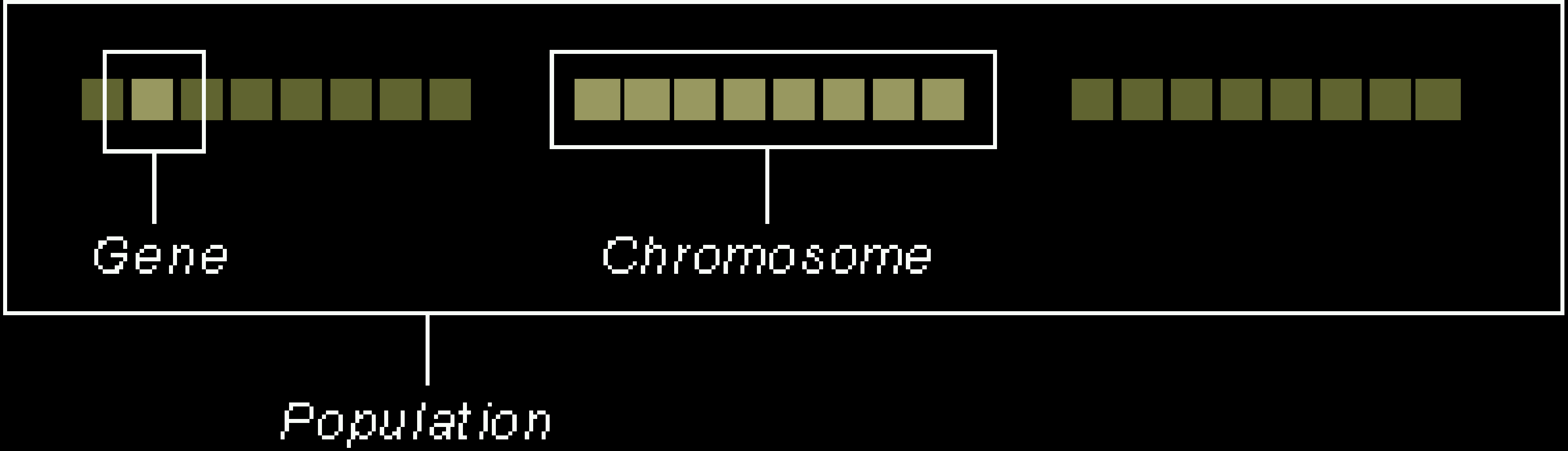
- Work very well on mixed (continuous *and* discrete), combinatorial problems
  - Can search a through a huge combination of parameters to find the best solutions

- Less susceptible to getting 'stuck' at local optima than other methods
  - it still will for complex problems though

- To use a genetic algorithm, you must represent a solution to your problem as a genome (or chromosome).

- Modus Operandi:
  - Create a population of solutions
  - Apply genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s)

GA Solution Structure



GA Critical Stages

In no particular order!

1. Define an objective function

2. Define and implement the genetic representation

3. Define and implement the genetic operators

- Once these have been defined, the generic genetic algorithm should work fairly well

Basic Algorithmic Process

1. Initialization
   - Create an initial population. This population is usually randomly generated and can be any desired size, from only a few individuals to thousands.

2. Evaluation
   - Each member of the population is evaluated by calculating a 'fitness' for that individual.
   - The fitness value is calculated by how well it fits with our desired requirements (objective function).

3. Selection
   - We want to be constantly improving our populations overall fitness.
   - Selection helps us to do this by discarding the bad designs and only keeping the best individuals in the population.
   - There are a few different selection methods but the basic idea is the same, make it more likely that fitter individuals will be selected for our next generation.

Basic Algorithmic Process

4. Crossover
   - Create new individuals by combining aspects of our selected individuals.
   - We can think of this as mimicking how sex works in nature: by combining certain traits from two or more individuals we will create an even 'fitter' offspring which will inherit the best traits from each of it's parents.

5. Mutation
   - Add a little bit randomness into our populations' genetics otherwise every combination of solutions we can create would be in our initial population.
   - Typically by making very small changes at random to an individuals genome.

6. And repeat! until we reach a termination condition.
   - Termination: found a solution which is good enough and meets a predefined minimum criteria or other computing constraints reached.

“Natural” Selection

- So how do we choose which solutions reproduce?

- Let’s assume we are:
  - elitist: always choosing the top k% “fittest” solutions
  - naïve: random selection
  - incestuous: cousins marry their cousins
  - probabilistic: define some function / distribution for selection
  - all of the above
  - some of the above
  - …

- How do we make these choices? and
- What’s going to happen if we do?

“Natural” Selection

- Irrespective of the method, we need to be able to rank / score / evaluate solutions

- For example the standard IBM interview question:
  - 10 of you are stranded on a desert island, after a nuclear holocaust you have a balloon, but only 2-4 of you will fit: how do you repopulate the world?
  - There are some properties of the group: doctor, male, female, jock, scientist, 21 year old blond, … and so on.

- Basic premises (components of an objective function):
  - Fitness function: scores how well the solution addresses the problem (repopulation of world)
  - Cost function: penalises a solution with respect to some given constraints (inhibiters of repopulating the world: e.g. only having males)

“Natural” Selection

- Typically, we do an “all of the above” approach to selection

- At the extremes:
  - Only choosing the fittest: minimises variation, and might lead to a local optimum etc.
  - Only choosing the costliest: well this is just stupid

- So we try to balance the two: probabilistically select the best, middle and worst
  - A solution’s probability to be selected is proportional to its own fitness and inversely proportional to the fitness of competing solutions (elitism)
  - Add some at random (irrespective of fitness)
  - …

- From this population we can:
  - Randomly choose parent solutions for new offspring
  - Mix fit and weak probabilistically
  - Rank order the solutions and use rank as a weight in probabilistic selection

(Mitchell, 1997 [Chap 9])

Simple R example: knapsack problem

National College of Ireland

You are going to spend a month in the wilderness. You're taking a backpack with you, however, the maximum weight it can carry is 20 kilograms. You have a number of survival items available, each with its own number of "survival points". You're objective is to maximize the number of survival points.

ITEM	SURVIVALPOINTS	WEIGHT
pocketknife	10	1
beans	20	5
potatoes	15	10
unions	2	1
sleeping bag	30	7
rope	10	5
compass	30	1

Let’s set up our problem

- install.packages("genalg”)

- library(genalg)

- dataset <- data.frame(item = c("pocketknife", "beans", "potatoes", "unions", "sleeping bag", "rope", "compass"), survivalpoints = c(10, 20, 15, 2, 30, 10, 30), weight = c(1, 5, 10, 1, 7, 5, 1))

- weightlimit <- 20

Basic Steps

1. Define and implement the genetic representation
   - 1001100 (pocketknife, unions, sleeping bag)
   - In R: chromosome = c(1, 0, 0, 1, 1, 0, 0)
   - We can visualise this as follows: dataset[chromosome == 1,]

2. Define an objective function
   - In simple terms we just need to count the survival points:
   - cat(chromosome %*% dataset$survivalpoints)

   - But we want this as a function so that we can call it frequently and also hand it over to the GA library (so that it does the work for us)

Basic Steps

```
evalFunc <- function(x) {
  current_solution_survivalpoints <- x %*% dataset$survivalpoints
  current_solution_weight <- x %*% dataset$weight

  if (current_solution_weight > weightlimit)
    return(0) else return(-current_solution_survivalpoints)
}
```

3. Define and implement the genetic operators
   - These are built in for us, we just need to call them

Running the GA

- Choose the number of iterations, design and run the model:

```
iter = 100
```

```
GAmodel <- rbga.bin(size = 7, popSize = 200, iters = iter, mutationChance = 0.01, elitism = T, evalFunc = evalFunc)
```

- rbga.bin (bin for binary, why binary?)

```
cat(summary.rbga(GAmodel))
```

Running the model

- The best solution is found to be **1111101**. This leads us to take the following items with us on our trip into the wild.

```
solution = c(1, 1, 1, 1, 1, 0, 1)
dataset[solution == 1, ]
```

```
##          item survivalpoints weight
## 1  pocketknife           10      1
## 2  beans                 20      5
## 3  potatoes              15     10
## 4  unions                 2      1
## 5  sleeping bag          30      7
## 7  compass               30      1
```

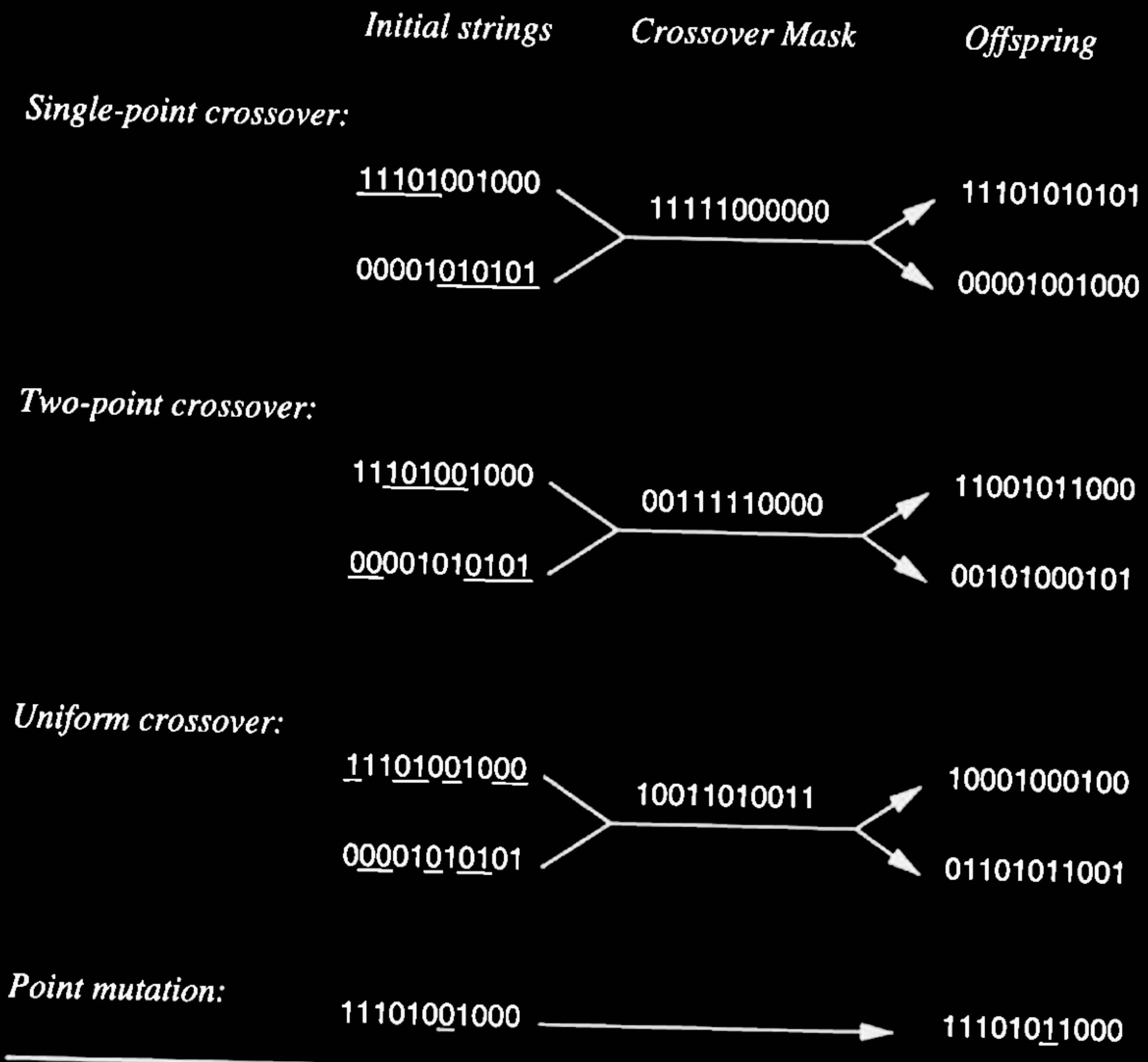
Mutation and Crossover

Mutation – defined by mutation rate

- This is essentially bit flipping: 0 becomes 1, 1 becomes 0
- usually a very low value for binary encoded genes, say 0.001

Crossover – defined by crossover rate: A good value for this is around .7

- Given two chromosomes, e.g.
  - 10001001110010010
  - 01010001001000011
  - Choose a random bit along the length, say at position 9, and swap all the bits after that points
- so the above become:
  - 10001001**101000011**
  - 01010001**010010010**



Initial strings *Crossover Mask* *Offspring*

Single-point crossover:

11101001000
00001010101
11111000000
11101010101
00001001000

Two-point crossover:

11101001000
00001010101
00111110000
11001011000
00101000101

Uniform crossover:

11101001000
00001010101
10011010011
10001000100
01101011001

Point mutation:

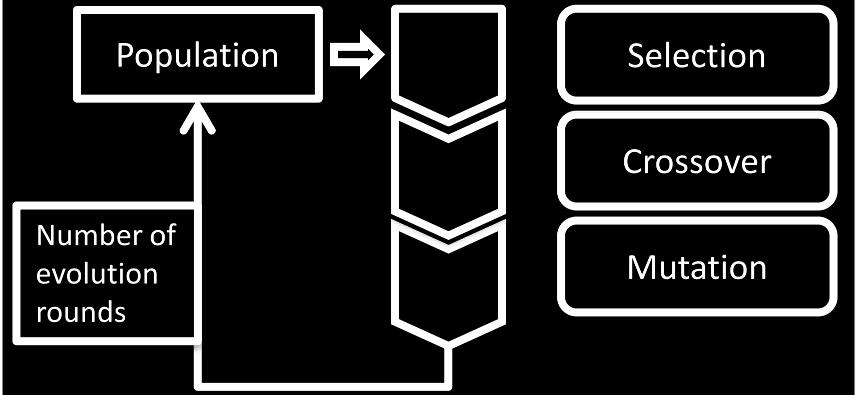
11101001000
11101011000

onal
ege of
and

Genetic Algorithms (GA)

Process

- Suitable for large search spaces
- Works with a **<u>population</u>** of solutions



Standard Parameters

GA parameters:

- Population size
- Crossover probability
- Mutation Probability
- Elitism (boolean)
- Number of evolution rounds

Runtime: O(mn^2)

(de Jong and Spears, 1991; Goldberg 1989)

Slide Credit: Christian Haas

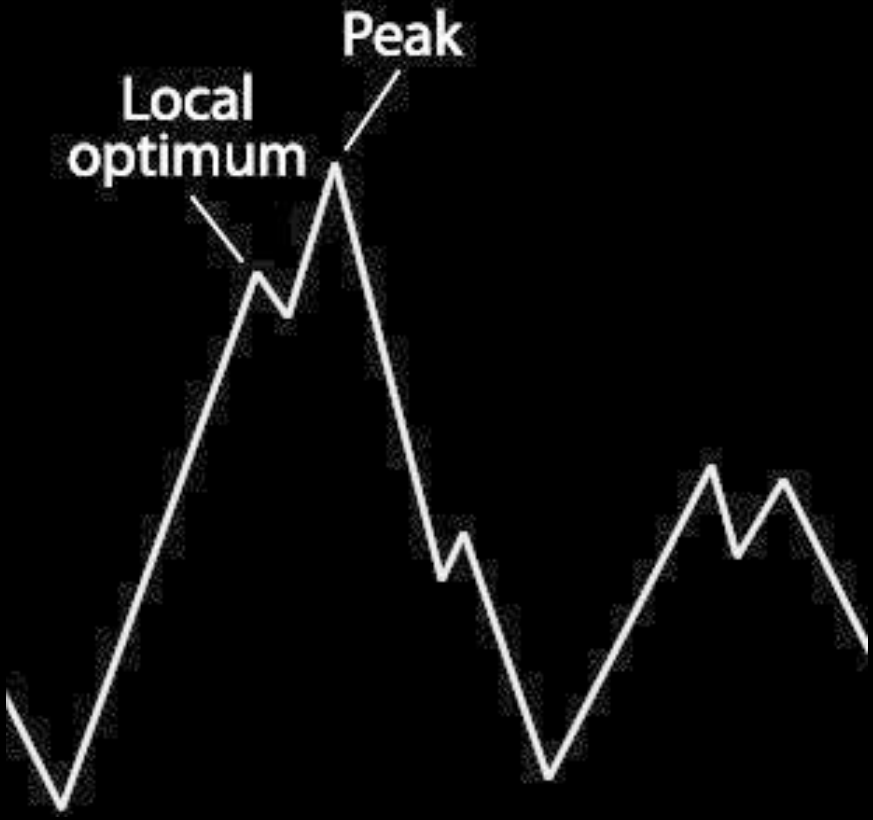
Be careful: Local Optima

- Imagine you are blind folded in a valley and have to find the peak

- How will you know when/if you are at the top?

- For complex problems, the best we can do is hope for is a close approximation of the optimal solution.

- For simple solutions, a GA can often break out of local optima



Be careful: Crowding

- Let's assume we have one solution that is relatively much fitter than all others

- Using elitism the propagation of this solution will be high, and thus the population will become incestuous w.r.t. this solution and the solution be become highly represented in the population
  - Thus diversity is reduced

- We can combat issues like this using many approaches:
  - Fitness sharing: fitness is reduced in the presence of similar solutions
  - Roulette wheel style selection
  - Selective breeding: only (dis)similar solutions are allowed to breed, thus creating sub-species
  - Spatial breeding: only solutions near to/far from each other in memory (or similar) may breed
  - …

(Mitchell, 1997 [Chap 9])

Things to consider

- Using selection alone will tend to fill the population with copies of the best individual from the population

- Using selection and crossover operators will tend to cause the algorithms to converge on a good but sub-optimal solution

- Using mutation alone induces a random walk through the search space.

- Using selection and mutation creates a parallel, noise-tolerant, hill climbing algorithm

Summary of GAs

- They conduct a randomised, parallel, hill-climbing search for solutions that optimise a predefined fitness function

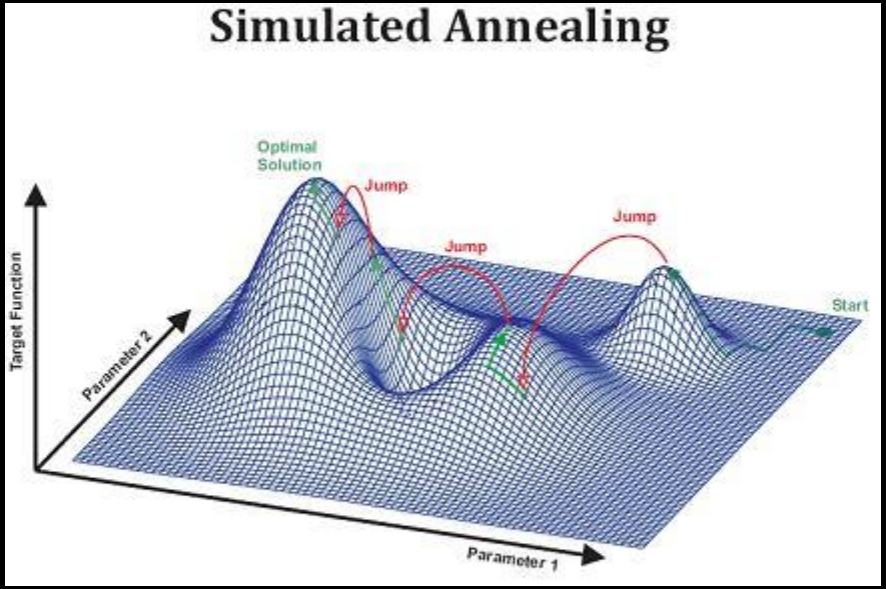
- Learning in this case is an optimisation task according to the maximisation of a fitness function
  - Note that this also implies that other optimisation techniques can also be applied to data mining

- Three operators: selection, mutation and crossover

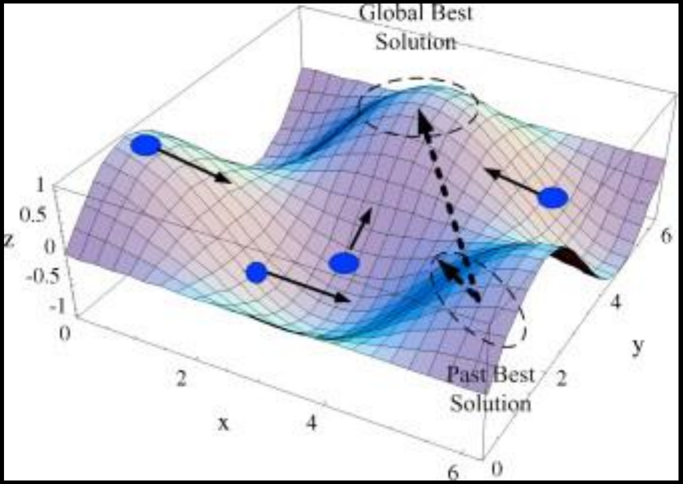
- Three stages: define objective function, genetic representation, operators

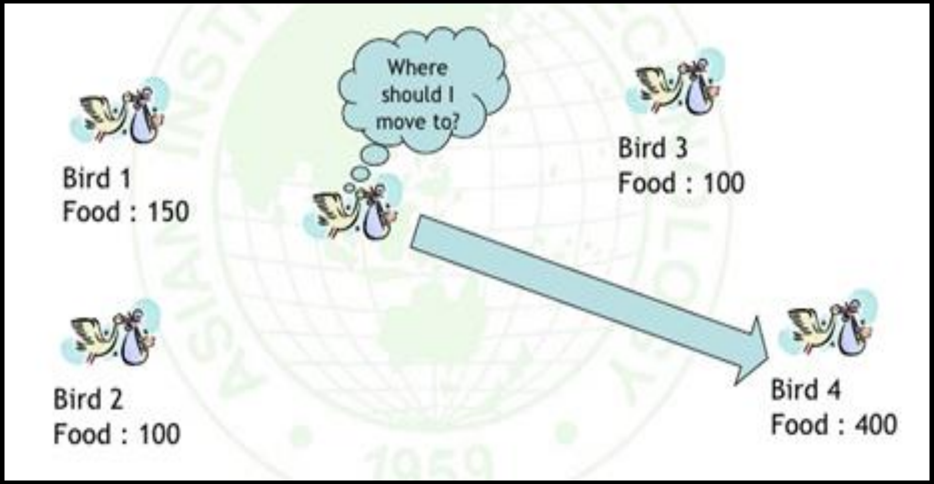
Particle Swarm Optimization

How to search across a complex solution space

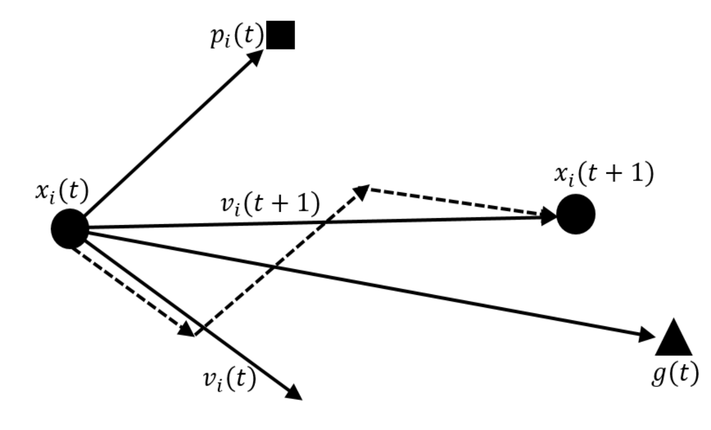


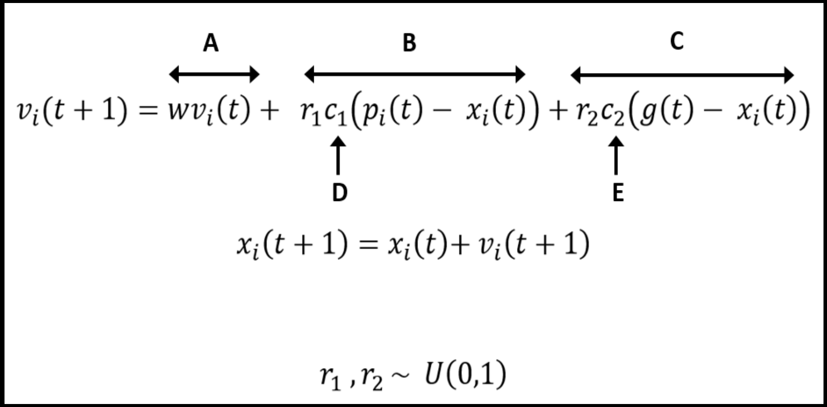
Particle Swarm Optimization





What is Particle Swarm Optimization?





$$v_i(t+1) = wv_i(t) + r_1c_1\big(p_i(t) - x_i(t)\big) + r_2c_2\big(g(t) - x_i(t)\big)$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$r_1, r_2 \sim U(0,1)$$

What is Particle Swarm Optimization?

Particle swarm is a population-based algorithm. In this respect, it is similar to the genetic algorithm.

A collection of individuals called particles move in steps throughout a region of the solution space. At each step, the algorithm evaluates the objective function at each particle. After this evaluation, the algorithm decides on the new velocity of each particle. The particles move, then the algorithm reevaluates.

The inspiration for the algorithm is flocks of birds or insects swarming. Each particle is attracted to some degree to the best location it has found so far, and also to the best location any member of the swarm has found. After some steps, the population can coalesce around one location, or can coalesce around a few locations, or can continue to move.

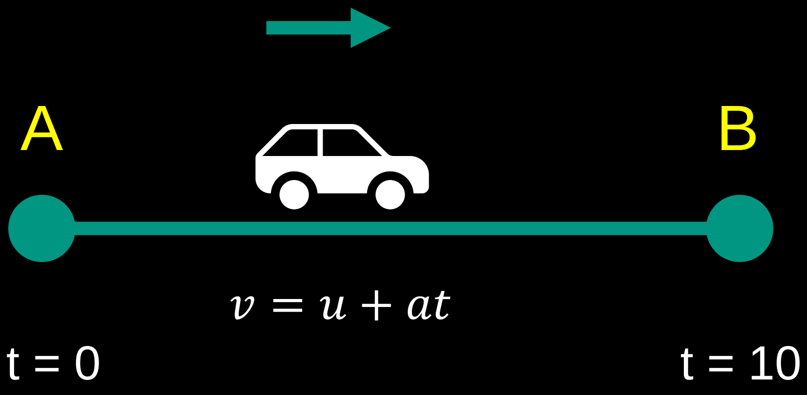
What is Particle Swarm Optimization?

The success of the algorithm is based on cooperation amongst the particles that fly through the solution space evaluating the fitness function. Two key principles that underlie this cooperation are **Communication** and **Learning**

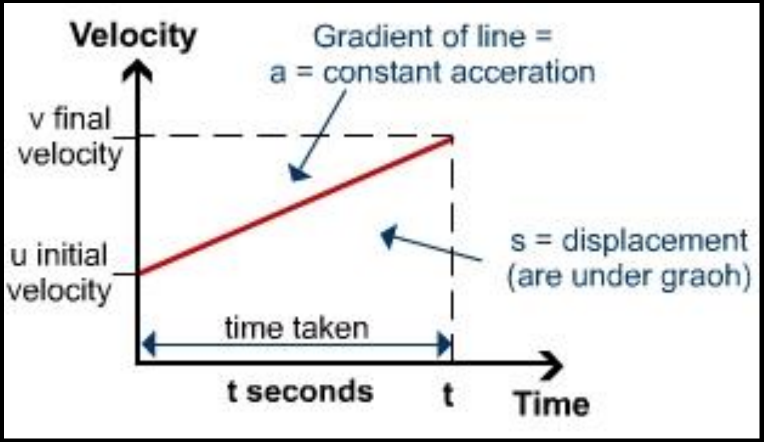
Communication: happens as particles share the success of their evaluation of the fitness function and thus make available their best position so far to other particles.

Learning: when one particle moves toward the position of another that particle learns that the new location is better than it previous one. Learning the concept of better is the main problem an optimizer should solve.

Equations of motion revision



$$v = u + at$$

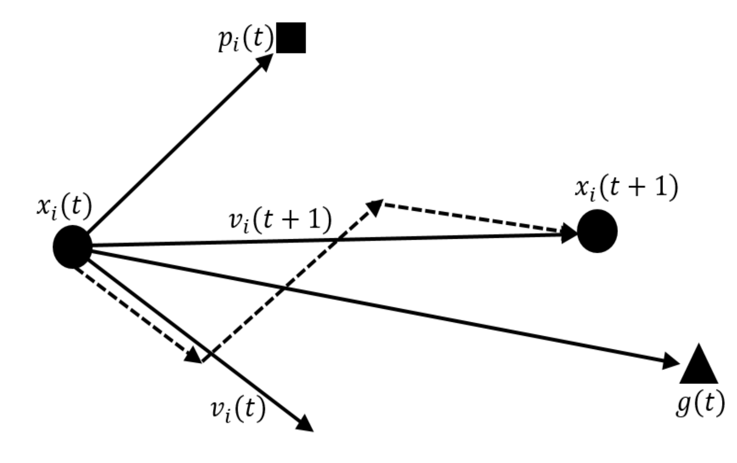


A car starts from rest (at time t = 0) and travels with a uniform acceleration of 5 meters per second (due east) squared for 10 seconds.

What velocity has it reached?

How far has it travelled? (ie what is it dsiplacment)

What is Particle Swarm Optimization?



$$v_i(t+1) = \overbrace{w v_i(t)}^{\textbf{A}} + \overbrace{\underset{\substack{\uparrow \\ \textbf{D}}}{r_1} c_1\big(p_i(t) - x_i(t)\big)}^{\textbf{B}} + \overbrace{\underset{\substack{\uparrow \\ \textbf{E}}}{r_2} c_2\big(g(t) - x_i(t)\big)}^{\textbf{C}}$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$r_1, r_2 \sim U(0,1)$$

At each particle has a position $x_i(t)$ and velocity $v_i(t)$. Additionally, each particle has a “memory” of its previous personal best position $p_i(t)$. As a member of a SWARM, particles are learning from each other from communication of their evaluation of the optimization function at each position of the solution space. The common or global best position of the SWARM is $g(t)$. The direction a particle moves in from its position $x_i(t)$ to $x_i(t+1)$ is influenced by its previous best position, the global position and its current velocity. The degree of this influence is captured in the weights; $w\,, c_1$ and c_2 respectively. r_1 and r_2 are uniform random numbers between 0 and 1.

The time step the evolution of the velocity of each particle is influenced by;

the current velocity $v_i(t)$ which is encapsulated in **A**, the **Inertia** term.
the position of its best position so far $p_{ij}(t)$ which is encapsulated in **B**, the **cogitative** term
the overall best (or global) position of all particles in the swarm so far $g_j(t)$ which is encapsulated in **C**, the **social** term

D and **E** are **acceleration** terms

Particle Swarm Optimization

$$v_i(t+1) = \overset{A}{\overleftrightarrow{w v_i(t)}} + \overset{B}{\overleftrightarrow{r_1 \underset{\overset{\uparrow}{D}}{c_1}\big(p_i(t) - x_i(t)\big)}} + \overset{C}{\overleftrightarrow{r_2 \underset{\overset{\uparrow}{E}}{c_2}\big(g(t) - x_i(t)\big)}}$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$$r_1\,, r_2 \sim \; U(0,1)$$

A: Inertia term
B: Cognitive term
C: Social term
D: Acceleration term
E: Acceleration term