AARHUS
UNIVERSITY

Data Analytics and Machine Learning

Global Search Part 2

Henrik Karstoft

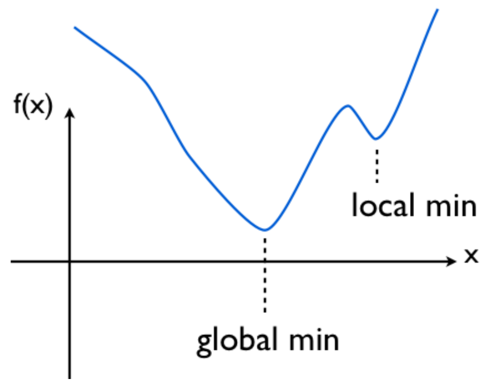
Carl Schultz

Alexandros Iosifidis

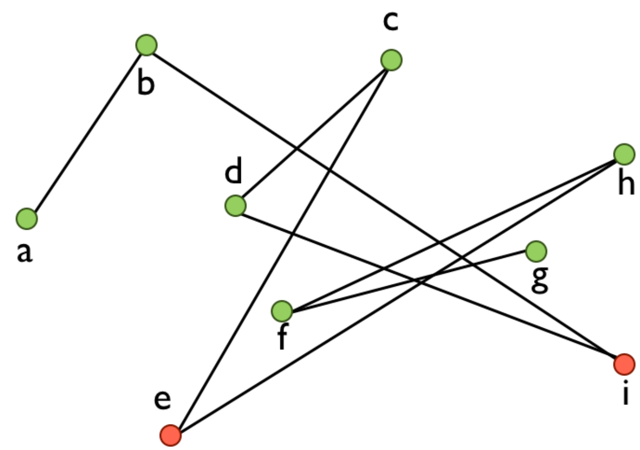
TODAY'S OUTLINE

quick recap

1. genetic algorithms
2. selection, mutation, crossover
3. why do they work?



- global vs local optimisation
- neighbourhood
- deterministic vs stochastic
- discrete vs continuous optimisation



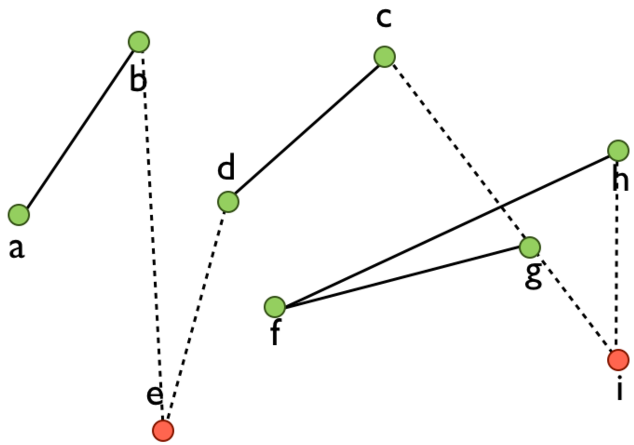
simulated annealing

- Let $s = s_0$
- For $k = 0$ through $k_{\max}$ (exclusive):
  - $T \leftarrow \text{temperature}(k / k_{\max})$?
  - Pick a random neighbour, $s_{\text{new}} \leftarrow \text{neighbour}(s)$
  - If $P(E(s), E(s_{\text{new}}), T) \geq \text{random}(0, 1)$: ?
    - $s \leftarrow s_{\text{new}}$
- Output: the final state s

https://en.wikipedia.org/wiki/Simulated_annealing

should we take a **worse** candidate solution?

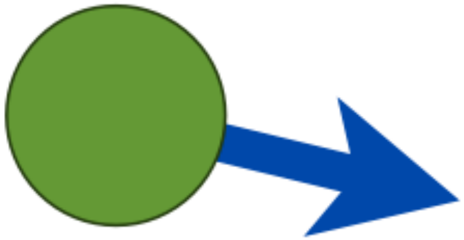
$$\exp\left(- \frac{(f(Z^{(i)}) - f(X^{(i)}))}{T^{(i)}} \right) > \text{random number between 0 and 1}$$

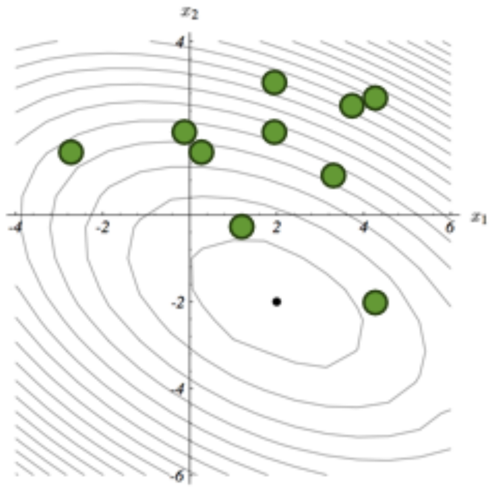


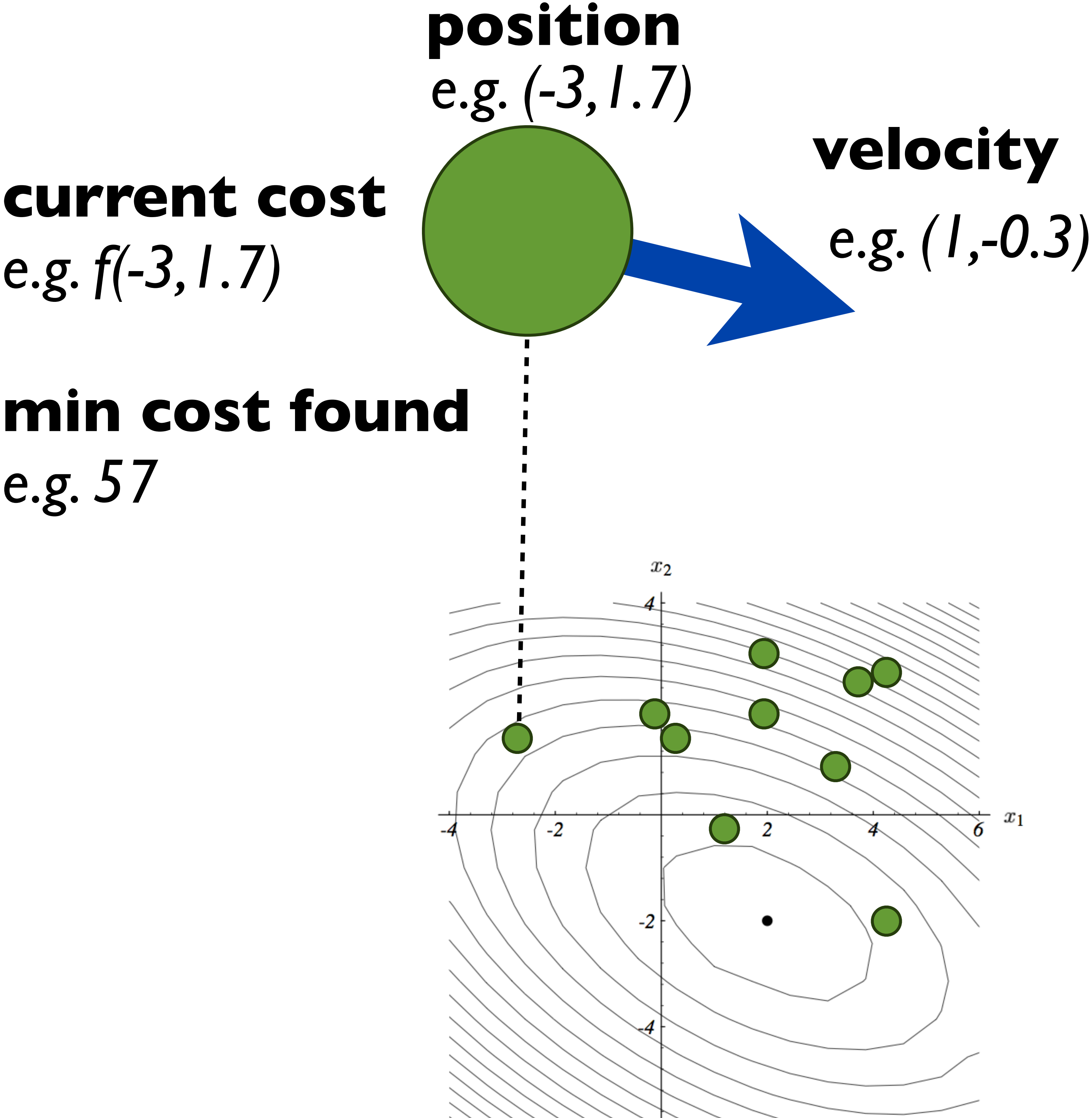
so, when are we more likely to take a worse solution?

particle swarm optimisation

- population of candidate solutions
- particle has position and velocity
- velocity update after each iteration





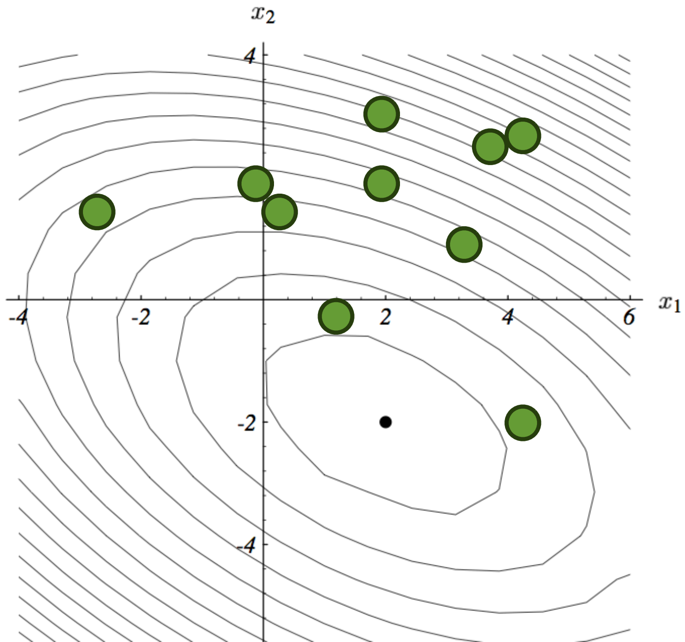

position
e.g. (-3,1.7)
velocity
e.g. (1,-0.3)
current cost
e.g. f(-3,1.7)
min cost found
e.g. 57
x_2
4
-2
-4
x_1
-4
-2
2
4
6

1. evaluate fitness of each particle

2. update individual and global fitnesses

3. update individual velocity and position

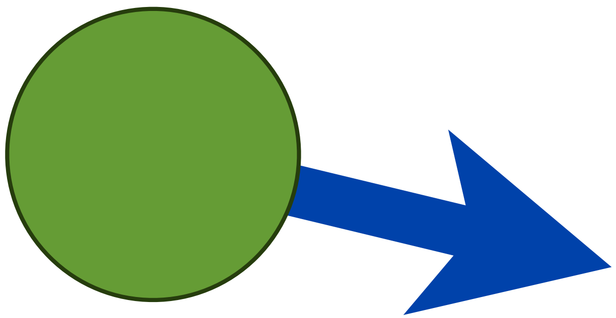
if stop condition not met, then repeat



update velocity of particle "i"

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

here the subscript means
"particle i", not the iteration



update velocity of particle "i"

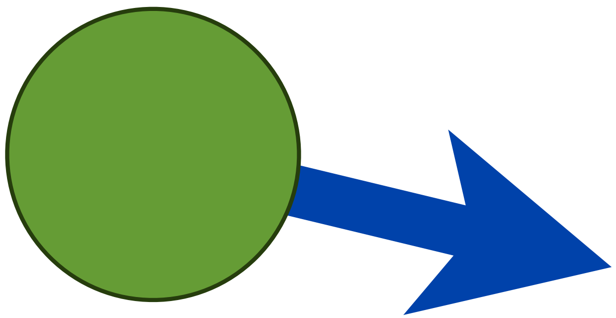
$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

velocity of particle "i"
at time *t*

best position from
any particle in the
swarm at time *t*

best position of
particle "i" up to time *t*

particle "i" position
at time *t*

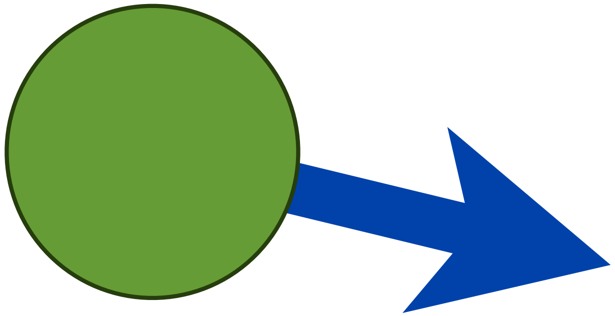


update velocity of particle “i”

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

once we stop the whole search algorithm, this is the value that is returned

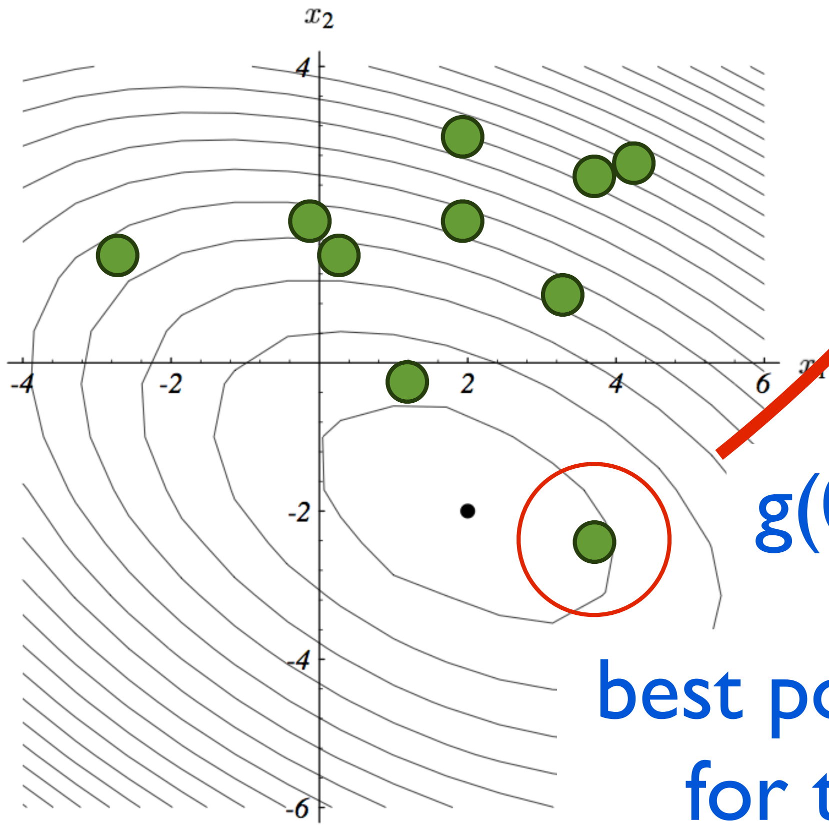
i.e. this is the position (input values for variables in X) that gave the minimum cost during the search (f(X))



update velocity of particle "i"

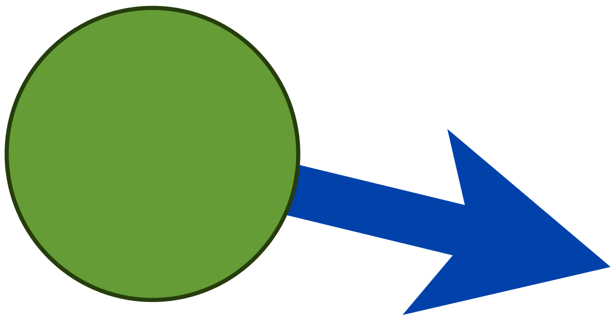
$$v_i(t+1) = wv_i(t) + c_1r_1[\hat{x}_i(t) - x_i(t)] + c_2r_2[g(t) - x_i(t)]$$

e.g. assume this is the first iteration, t=0



g(0)=(3.7,-2.2)

best position found for the swarm



update velocity of particle "i"

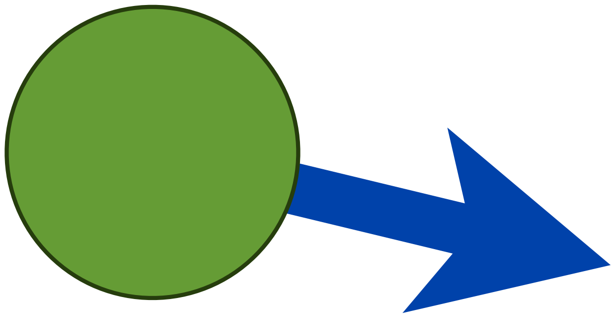
$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

random number

$0 \leq r_2 \leq 1$

random number

$0 \leq r_1 \leq 1$



update velocity of particle "i"

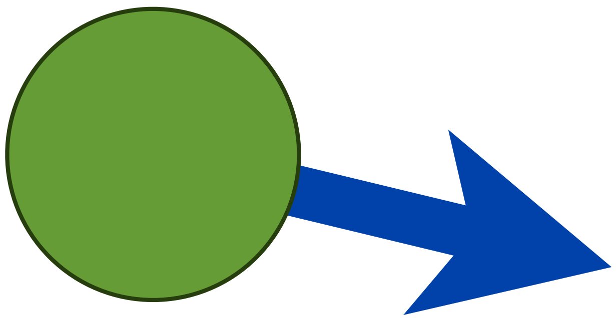
$$v_i(t+1) = wv_i(t) + c_1r_1[\hat{x}_i(t) - x_i(t)] + c_2r_2[g(t) - x_i(t)]$$

user-supplied coefficients

$0.8 \leq w \leq 1.2$

$0 \leq c_1 \leq 2$

$0 \leq c_2 \leq 2$

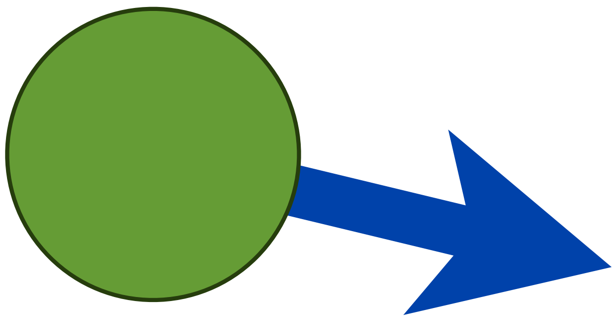


update velocity of particle “i”

$$v_i(t+1) = \underbrace{wv_i(t)}_{\text{inertia component}} + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

inertia component:
keeps particle moving in similar direction

lower w: speeds up convergence to local optima
higher w: encourages exploration

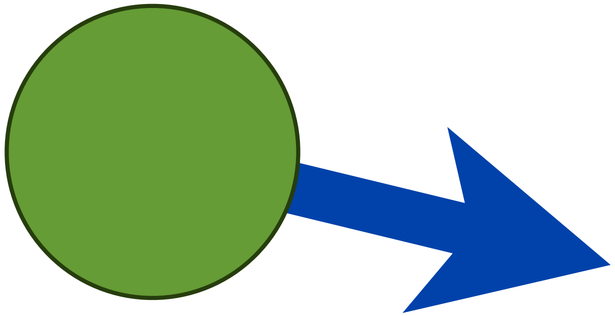


update velocity of particle “i”

$$v_i(t+1) = wv_i(t) + \underbrace{c_1 r_1[\hat{x}_i(t) - x_i(t)]} + c_2 r_2[g(t) - x_i(t)]$$

cognitive component:
particle’s memory, encourages particle to go back to best position

high c_1: take larger step towards best found position

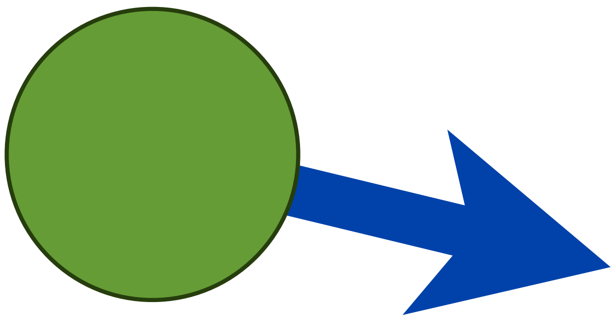


update velocity of particle “i”

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + \underbrace{c_2 r_2[g(t) - x_i(t)]}$$

social component:
encourages particle to move to swarm’s
best found position so far

high c_2: take larger step towards swarm best so far

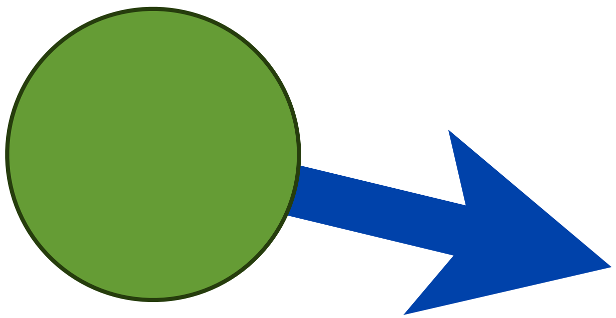


update velocity of particle "i"

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

update position of particle "i"

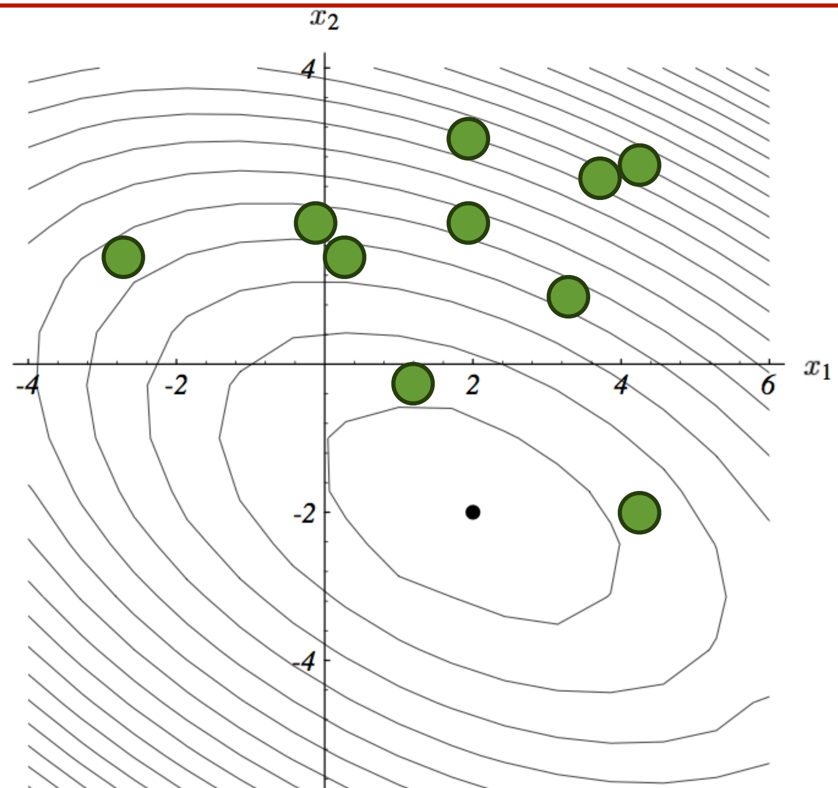
$$x_i(t+1) = x_i(t) + v_i(t+1)$$



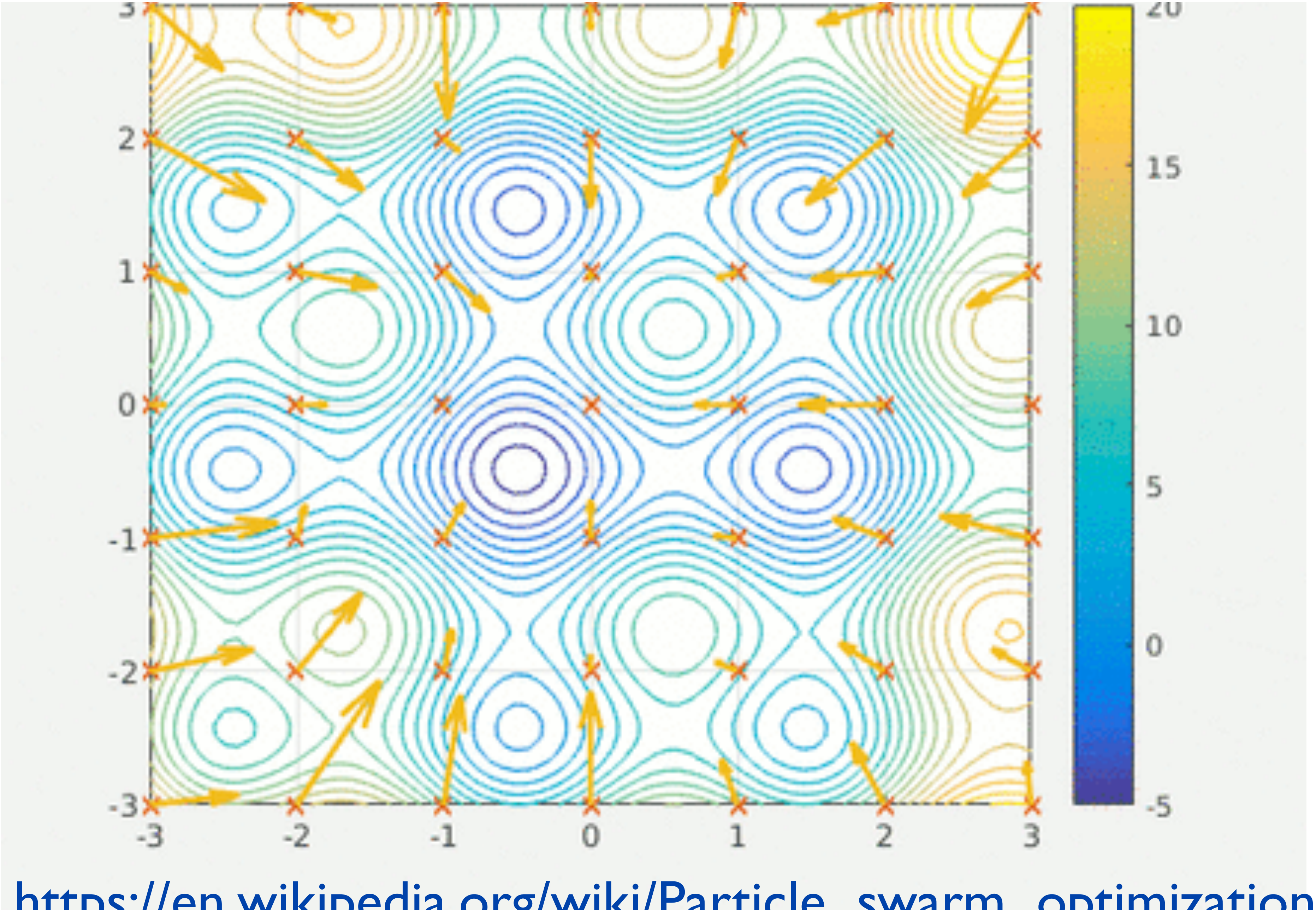
1. evaluate fitness of each particle

2. update individual and global fitnesses

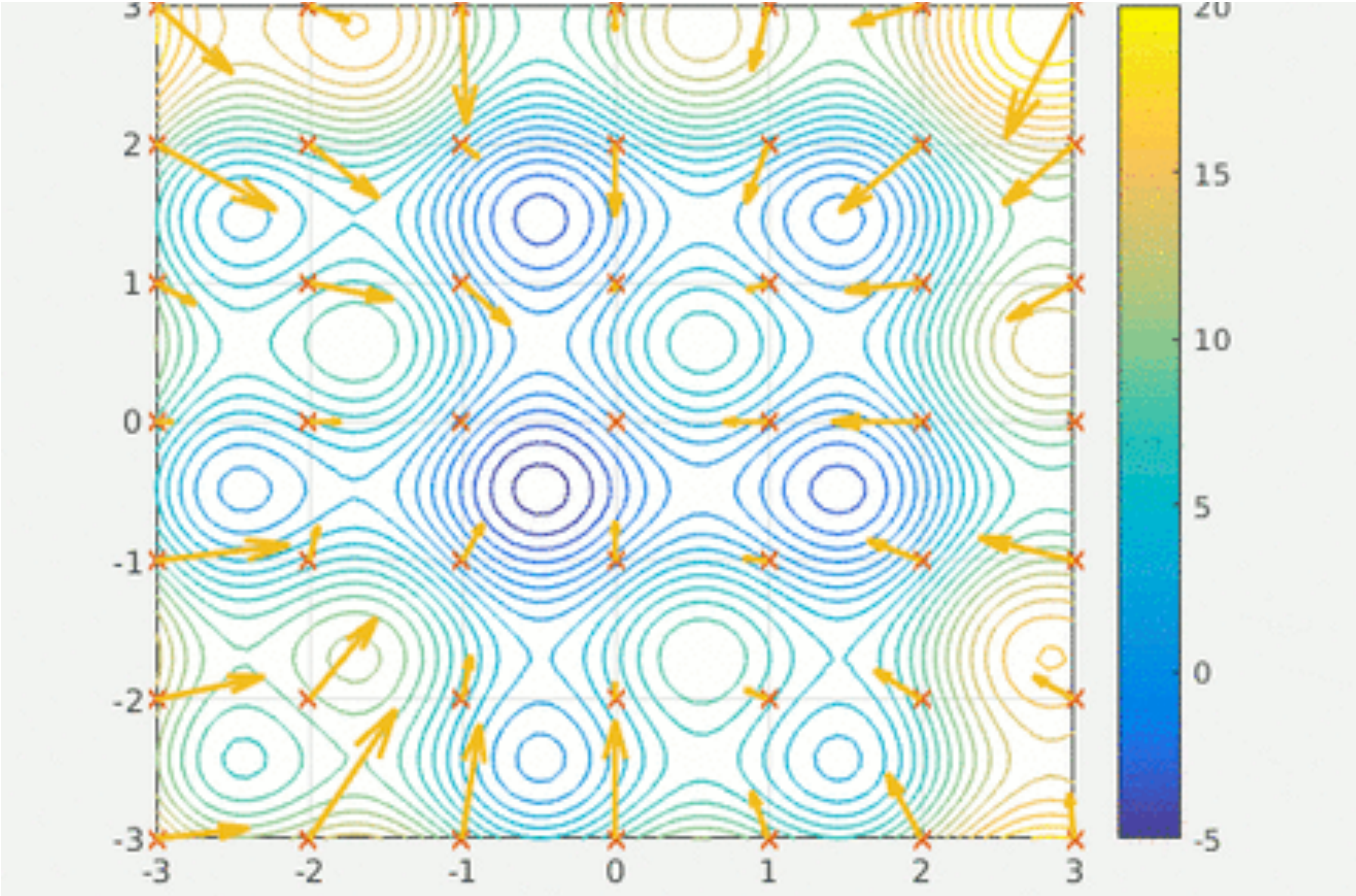
3. update individual velocity and position



consider this again, thinking about how at each step, each particle velocity is update (inertia, cognitive, social)

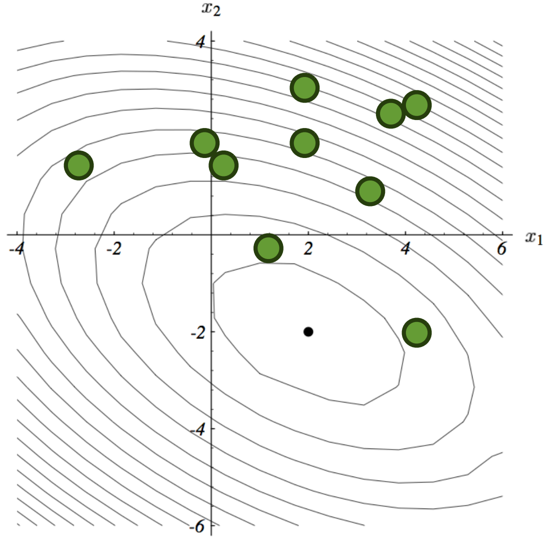


https://en.wikipedia.org/wiki/Particle_swarm_optimization swarm manages to find the global min (i.e. at least one particle went to this position at least once)



https://en.wikipedia.org/wiki/Particle_swarm_optimization

task. write pseudo-code for particle swarm optimisation



You are given N particles and T time steps.

Let: X, ^X, V be vector of length N (of 2D points)

X is the current particle positions

^X is the best position so far for each particle

V is the current velocity of each particle

Let g be the best location found so far (of any particle).

Assume X,V,^X,g have been initialised for you.

Also assume you are given constants: w, c1, c2

You have functions:

F(p): fitness of point p

RANDOM: returns a random number between 0 and 1

Examples:

X(i) : gives the current 2D point location of particle i

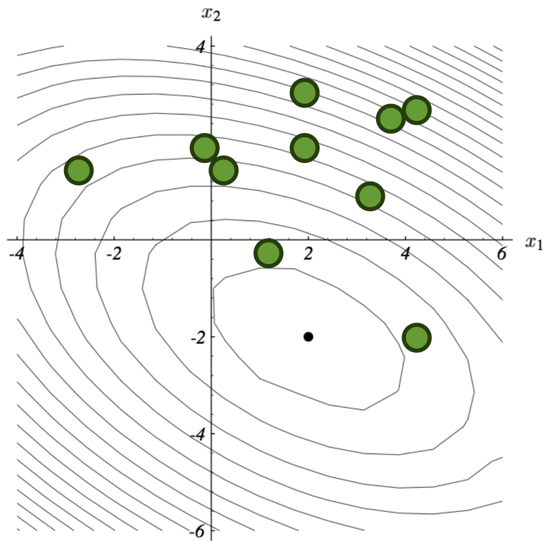
F(^X(i)) : gives best fitness of particle i so far

F(g) : gives the fitness of the best location so far

^X(i) = X(i) : updates best position of particle i to current position

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

task. write pseudo-code for particle swarm optimisation



You are given N particles and T time steps.

Let: X, ^X,V be vector of length N (of 2D points)

X is the current particle positions

^X is the best position so far for each particle

V is the current velocity of each particle

Let g be the best location found so far (of any particle).

Assume X,V,^X,g have been initialised for you.

Also assume you are given constants: w, c1, c2

You have functions:

F(p): fitness of point p

RANDOM: returns a random number between 0 and 1

Examples:

X(i) : gives the current 2D point location of particle i

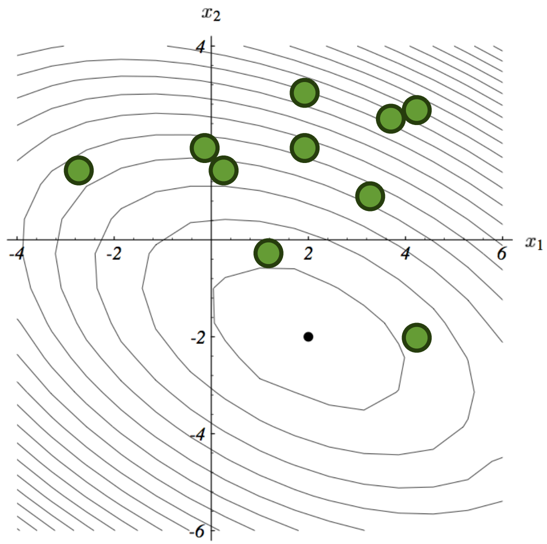
F(^X(i)) : gives best fitness of particle i so far

F(g) : gives the fitness of the best location so far

^X(i) = X(i) : updates best position of particle i to current position

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

task. write pseudo-code for particle swarm optimisation



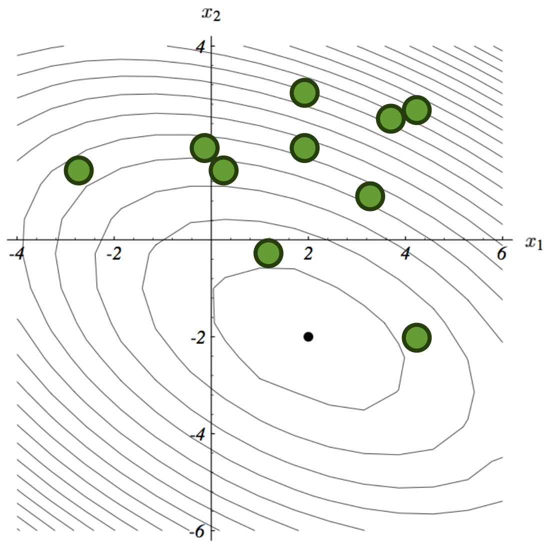
You are given N particles and T time steps.
Let: X, ^X, V be vector of length N (of 2D points)
X is the current particle positions
^X is the best position so far for each particle
V is the current velocity of each particle
Let g be the best location found so far (of any particle).
Assume X,V,^X,g have been initialised for you.
Also assume you are given constants: w, c1, c2

You have functions:
F(p): fitness of point p
RANDOM: returns a random number between 0 and 1

Examples:
X(i) : gives the current 2D point location of particle i
F(^X(i)) : gives best fitness of particle i so far
F(g) : gives the fitness of the best location so far
^X(i) = X(i) : updates best position of particle i to current position

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

task. write pseudo-code for particle swarm optimisation



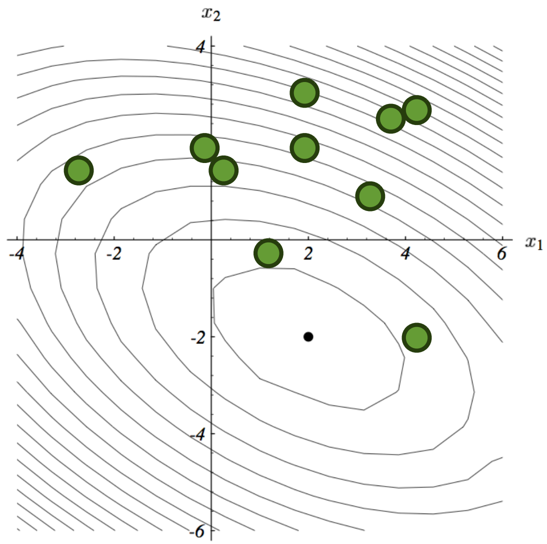
You are given N particles and T time steps.
Let: X, ^X, V be vector of length N (of 2D points)
X is the current particle positions
^X is the best position so far for each particle
V is the current velocity of each particle
Let g be the best location found so far (of any particle).
Assume X,V,^X,g have been initialised for you.
Also assume you are given constants: w, c1, c2

You have functions:
F(p): fitness of point p
RANDOM: returns a random number between 0 and 1

Examples:
X(i) : gives the current 2D point location of particle i
F(^X(i)) : gives best fitness of particle i so far
F(g) : gives the fitness of the best location so far
^X(i) = X(i) : updates best position of particle i to current position

$$v_i(t+1) = wv_i(t) + c_1r_1[\hat{x}_i(t) - x_i(t)] + c_2r_2[g(t) - x_i(t)]$$

task. write pseudo-code for particle swarm optimisation



You are given N particles and T time steps.

Let: X, ^X, V be vector of length N (of 2D points)

X is the current particle positions

^X is the best position so far for each particle

V is the current velocity of each particle

Let g be the best location found so far (of any particle).

Assume X,V,^X,g have been initialised for you.

Also assume you are given constants: w, c1, c2

You have functions:

F(p): fitness of point p

RANDOM: returns a random number between 0 and 1

Examples:

X(i) : gives the current 2D point location of particle i

F(^X(i)) : gives best fitness of particle i so far

F(g) : gives the fitness of the best location so far

^X(i) = X(i) : updates best position of particle i to current position

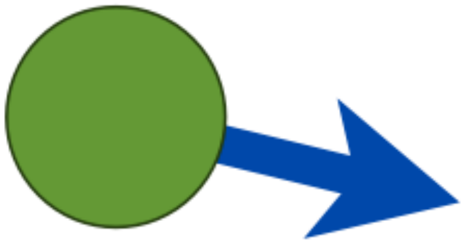
$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

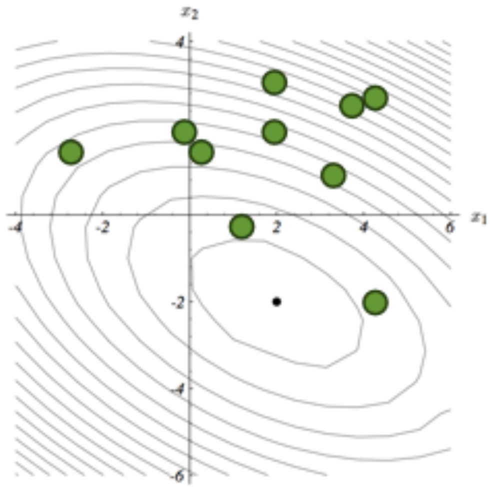
my idea:

```
FOR t = 1 TO T {

 // update fitness values of each particle
 FOR i = 1 TO N {
   IF F(X(i)) < F(^X(i)) THEN ^X(i) = X(i)
   IF F(X(i)) < F(g)     THEN g = X(i)
 }

 // update velocity and position of each particle
 FOR i = 1 TO N {
   r1 = RANDOM
   r2 = RANDOM
   V(i) = w*V(i) + c1*r1*(^X(i) - X(i)) + c2*r2*(g - X(i))
   X(i) = X(i) + V(i)
 }

}
```

SUMMARY particle swarm

- population of candidate solutions
- particle has position and velocity
- velocity update components: inertia, cognitive, social





Part 1
genetic algorithms

- genetic algorithms are a family of models
- evolution metaphor
- we can use genetic algorithms to do optimisation

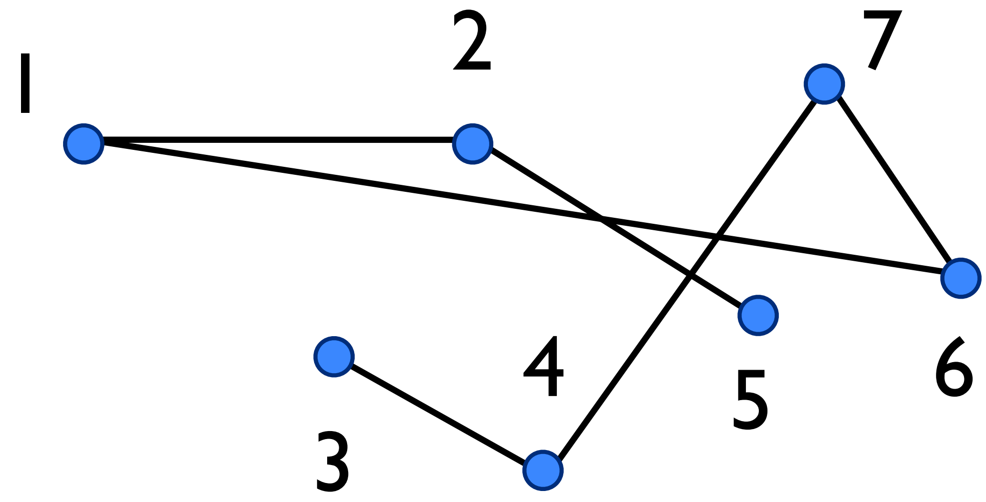
2006 NASA ST5 spacecraft antenna shape
"evolved" using genetic algorithm

https://en.wikipedia.org/wiki/Genetic_algorithm

- like particle swarm, we have a **population** of candidate solutions at each iteration
- instead of "particles", each candidate is an "individual" with a "chromosome" e.g.:

010 100 111 110 001 010 101

- each "chromosome" represents one candidate solution to our optimisation problem

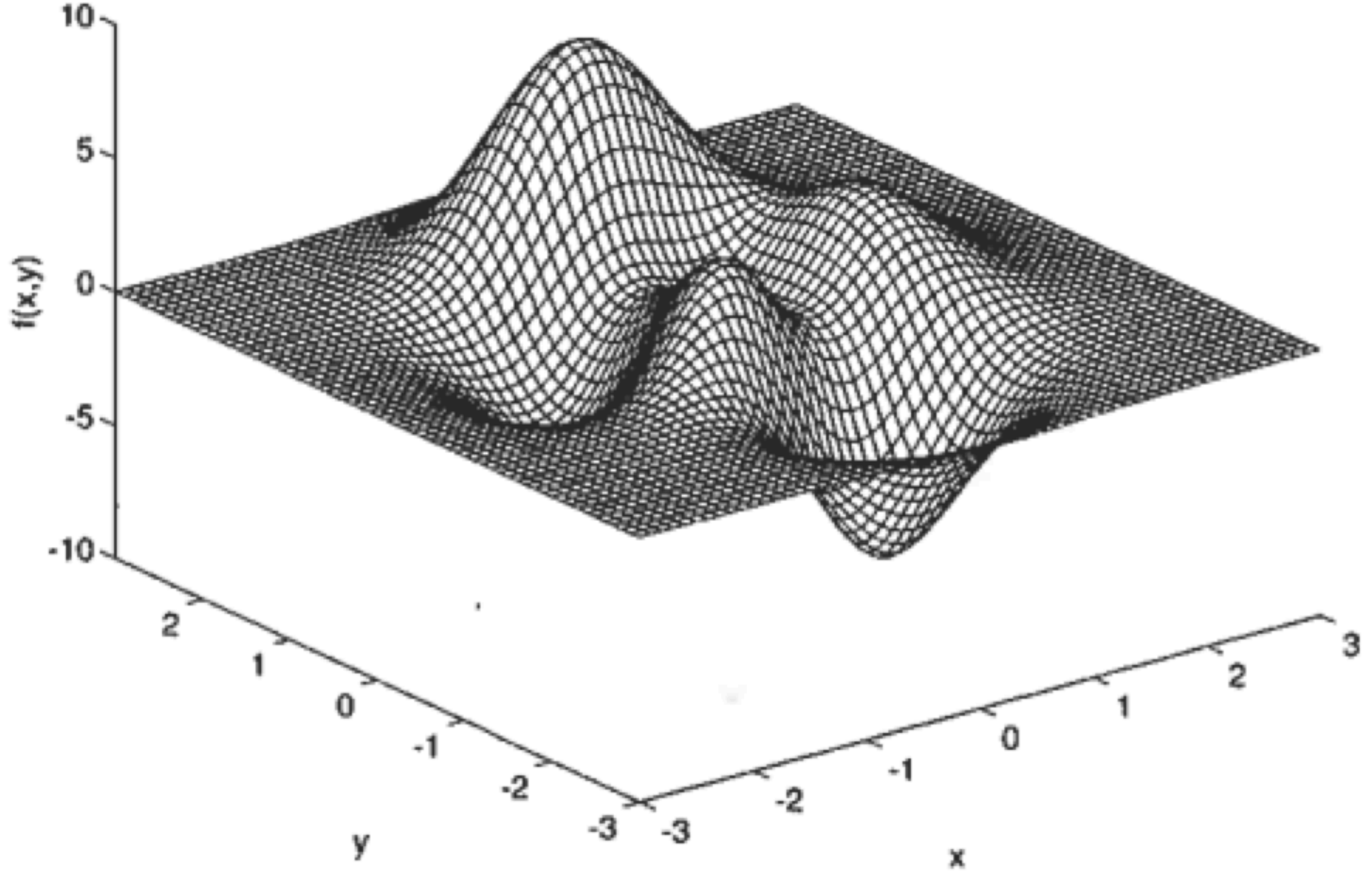


$$\underbrace{011}_{3}\ \underbrace{100}_{4}\ \underbrace{111}_{7}\ \underbrace{110}_{6}\ \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{101}_{5}$$

e.g. travelling salesman problem

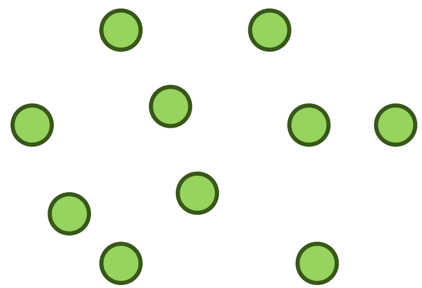
chromosome describes order to visit cities

$$f(x,y) = 3(1-x)^2\, e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)\, e^{-x^2-y^2} - \frac{e^{-(x+1)^2-y^2}}{3}$$

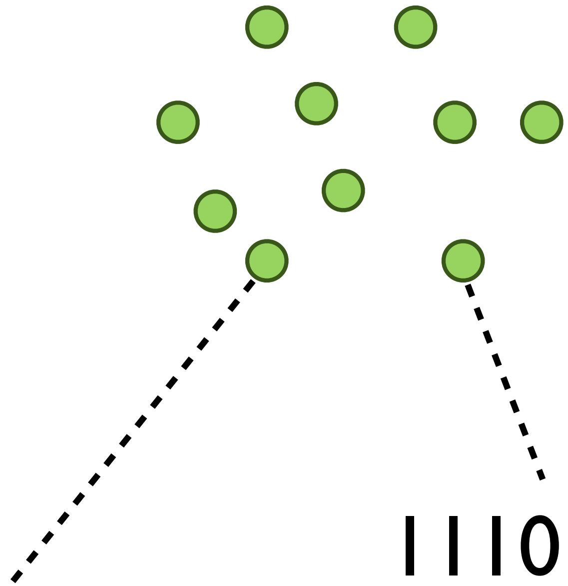


$$\underbrace{0101010101010101}_{\text{encoded } x = -1}\underbrace{1111111111111111}_{\text{encoded } y = 3}.$$

population at
time t
(called “generation t”)



population at
time t
(called “generation t”)

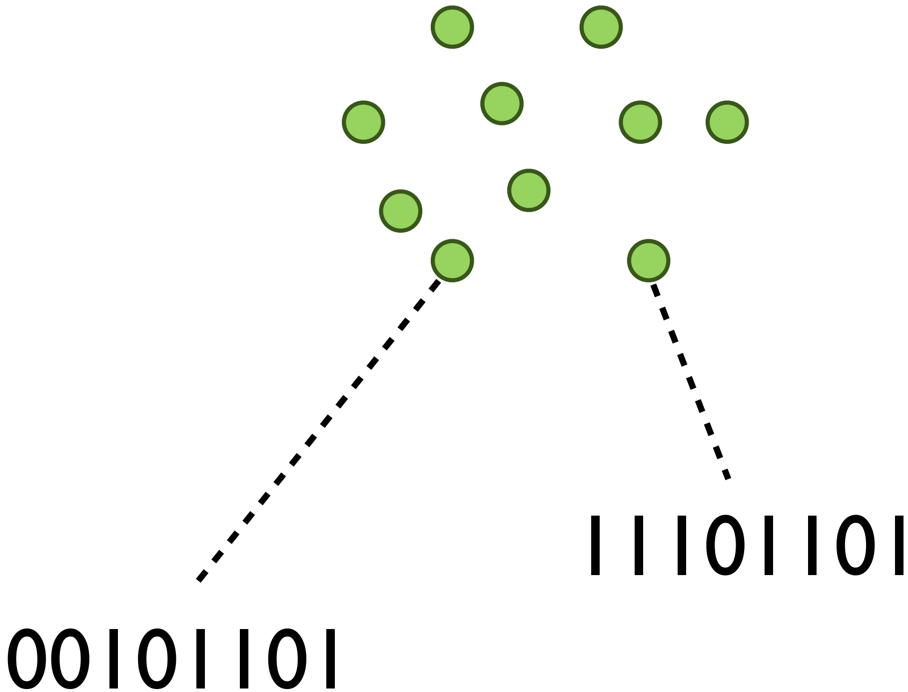


11101101

00101101

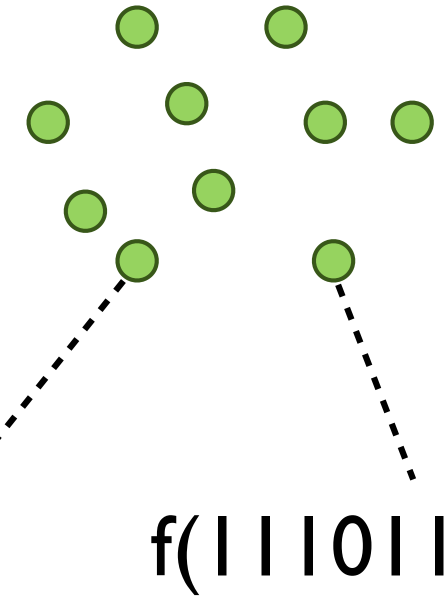
each “individual” in the population has a chromosome

population at
time t
(called "generation t")



...and each chromosome is a candidate solution to our optimisation problem

population at
time t
(called “generation t”)

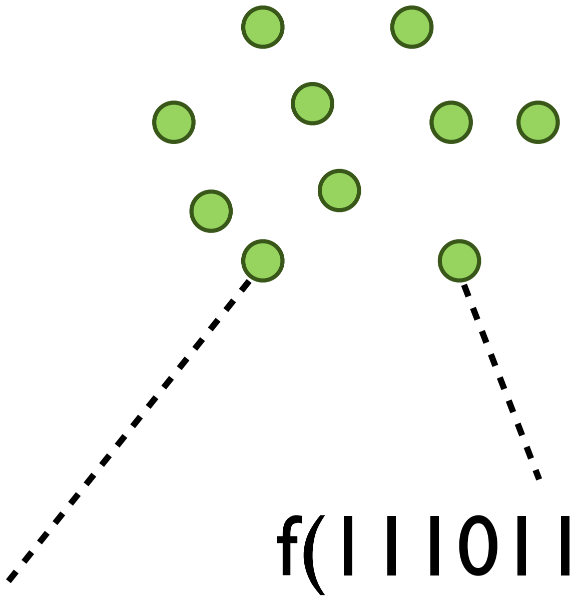


f(11101101)=10

f(00101101) = 3

so each individual has “fitness”, which is based on the
“objective function value” of that candidate solution

population at
time t
(called "generation t")

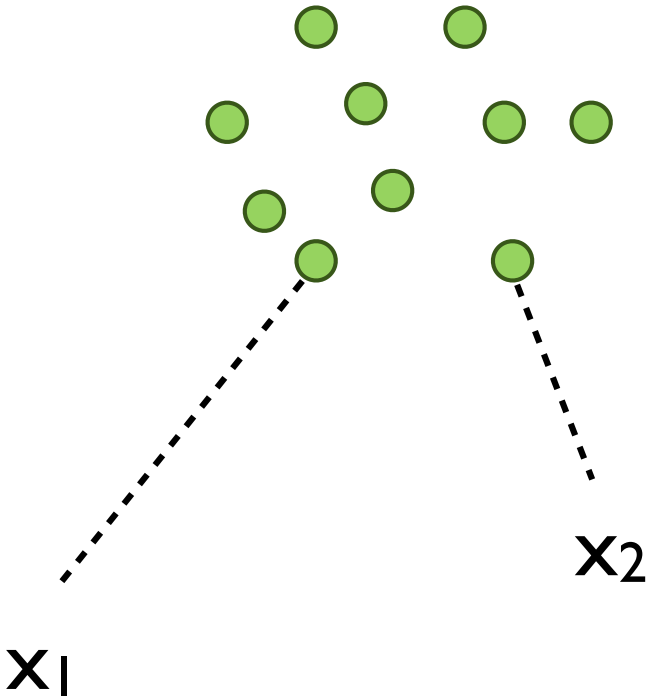


$f(11101101)=10$

$f(00101101) = 3$

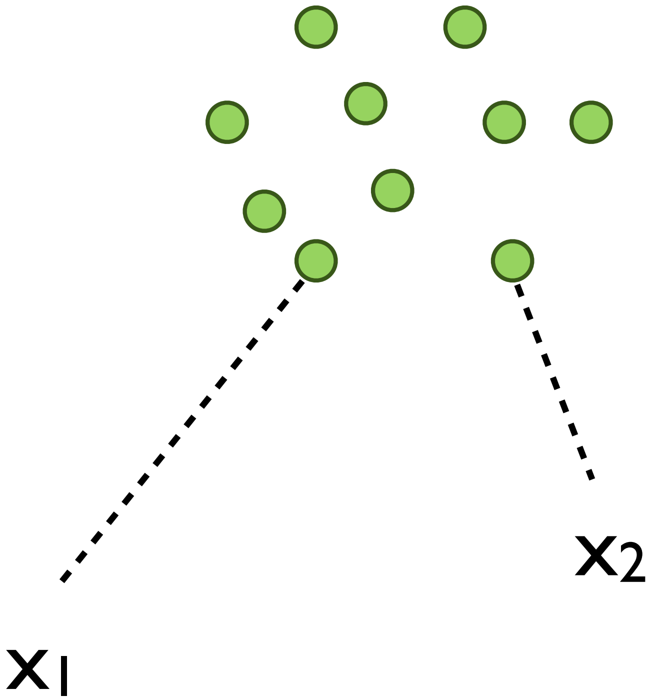
our overall goal is to maximise the objective function value

objective function value for x_1 $f(00101101) = 3$



objective function value for x_1 $\quad$ $f(00101101) = 3$

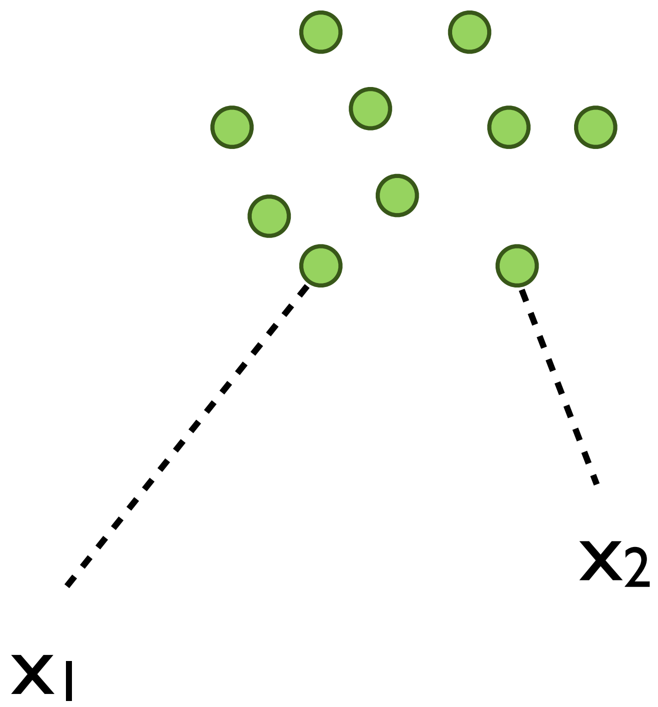
mean objective function value over population $\quad$?



objective function value for x_1 $\quad f(00101101) = 3$

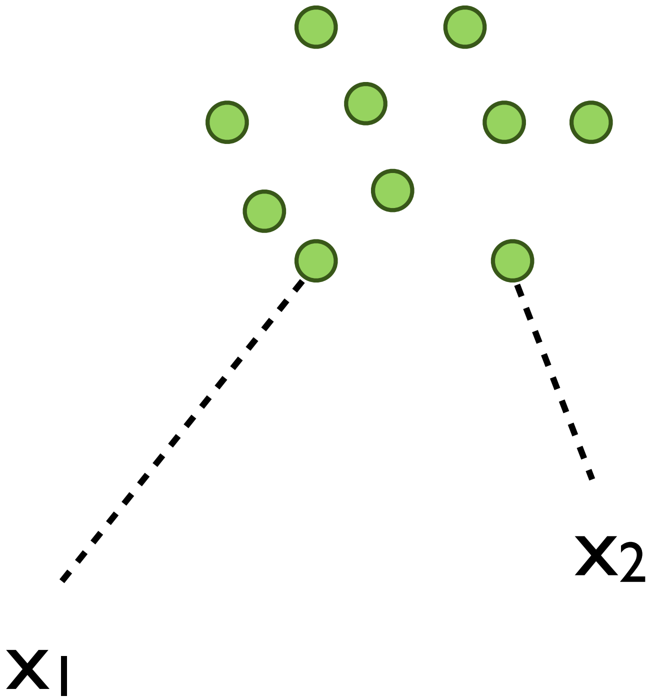
mean objective function value over population

$$\bar{f} = \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

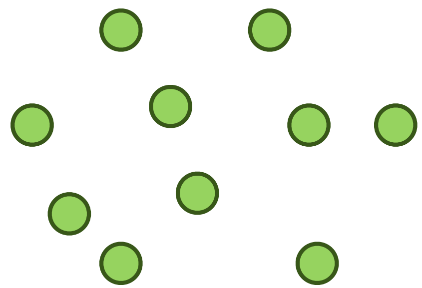


fitness function for x_i

$$\frac{f(x_i)}{\bar{f}}$$



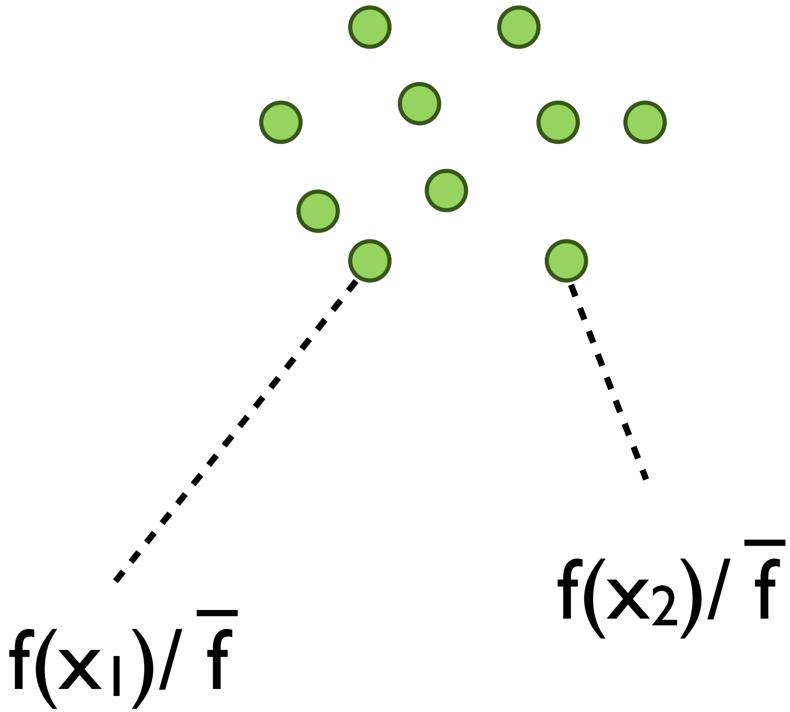
fitness function for x_i $$\frac{f(x_i)}{\bar{f}}$$



this turns objective score into some relative measure of "reproductive opportunities"

i.e. survival of the fittest

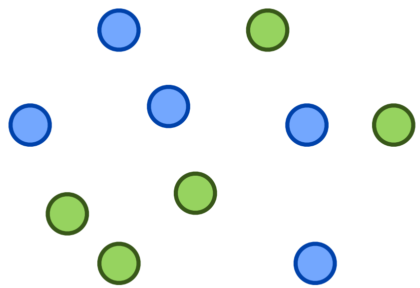
population at
time t
(called "generation t")



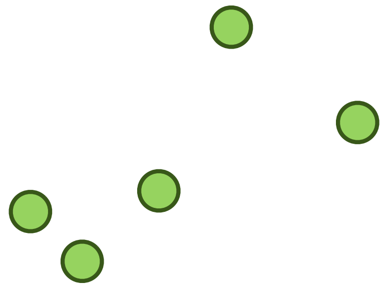
at each iteration (generation) each individual has a fitness value

select individuals based on their fitness in the population
(e.g. they tend to have higher fitness in this set)

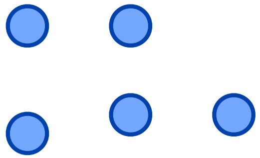
called the “selection” step

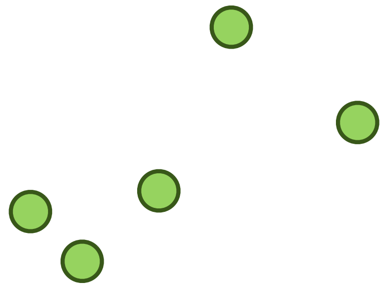


(lots of differents ways to implement this step)

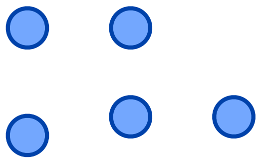


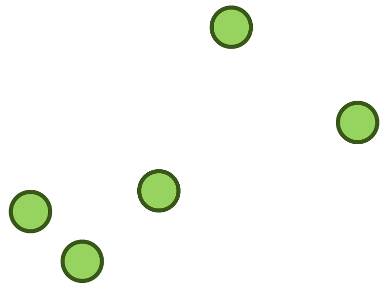
“intermediate” population, likely to be best individuals



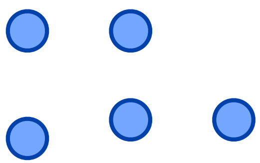


we’ll use these individuals to create our **next generation** of individuals

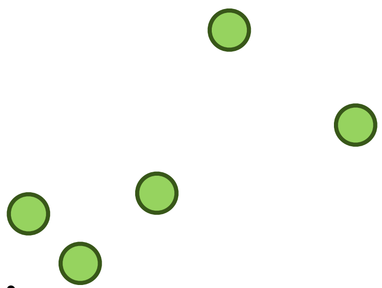




...via "crossover" and "mutation"

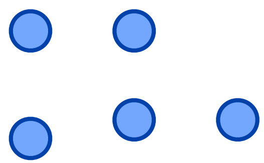


(also lots of ways to implement this step)

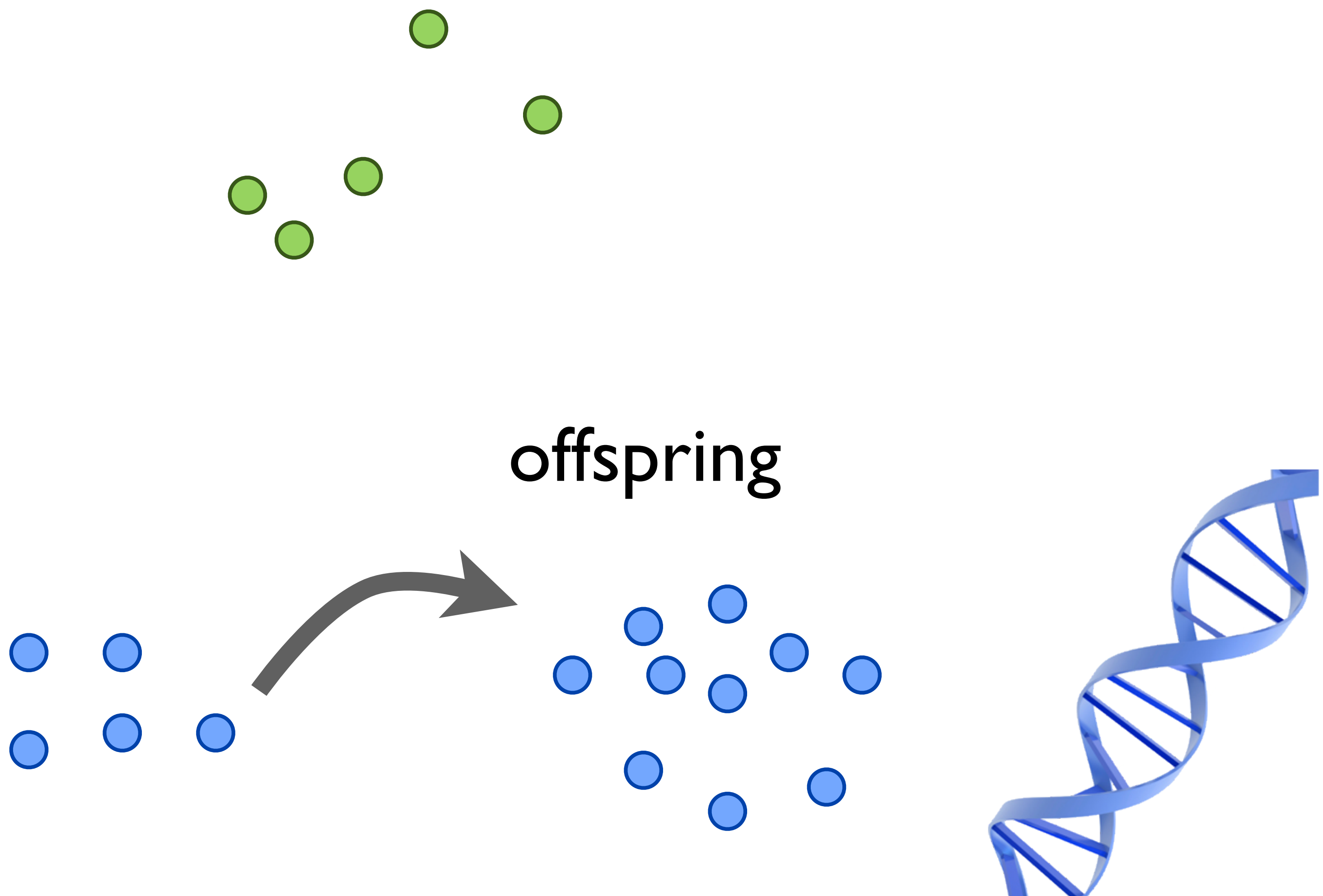


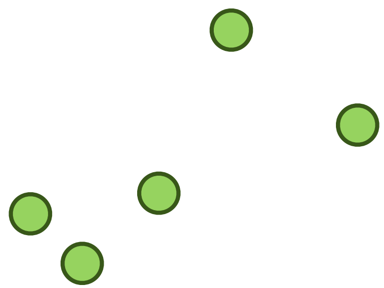
also called “recombination”

...via “crossover” and “mutation”

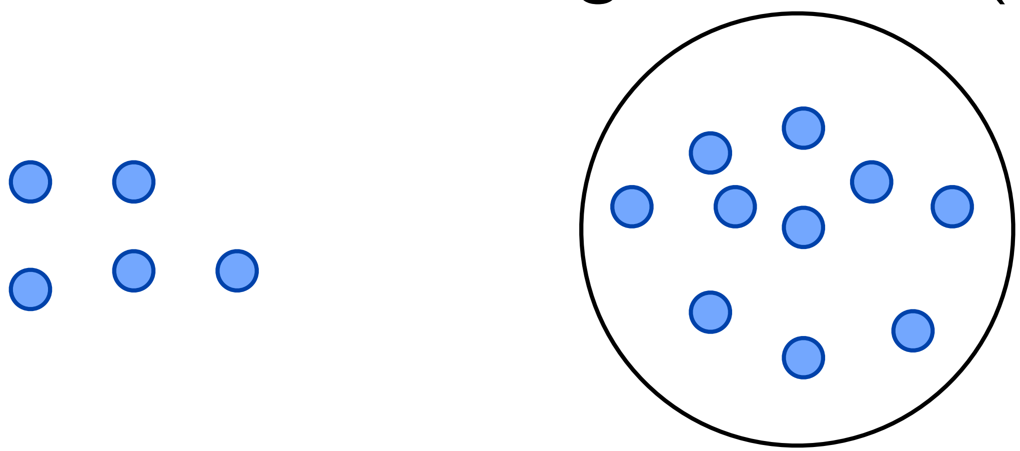


(also lots of ways to implement this step)

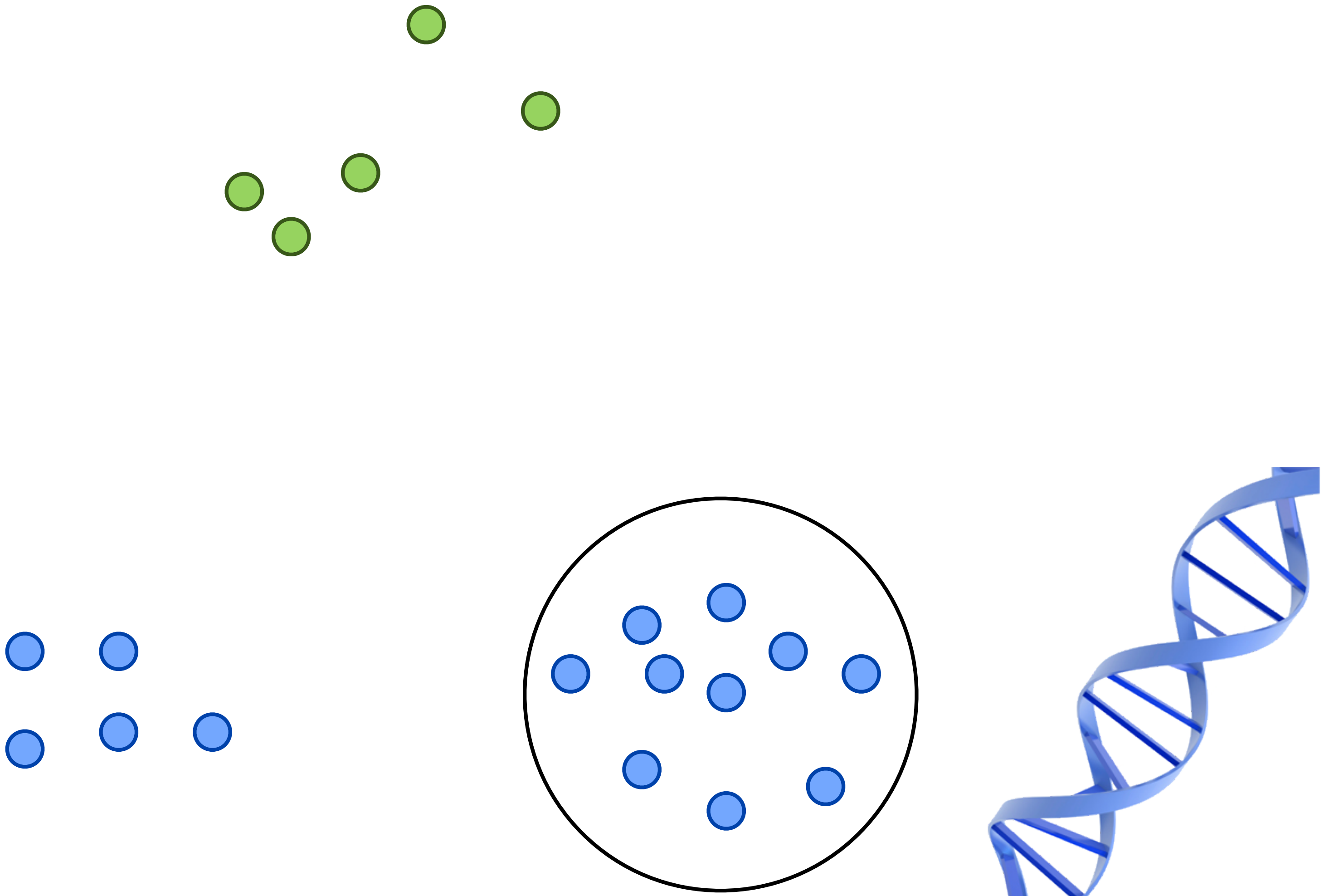

offspring

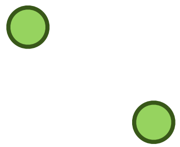


we have our population
for generation (t+1)



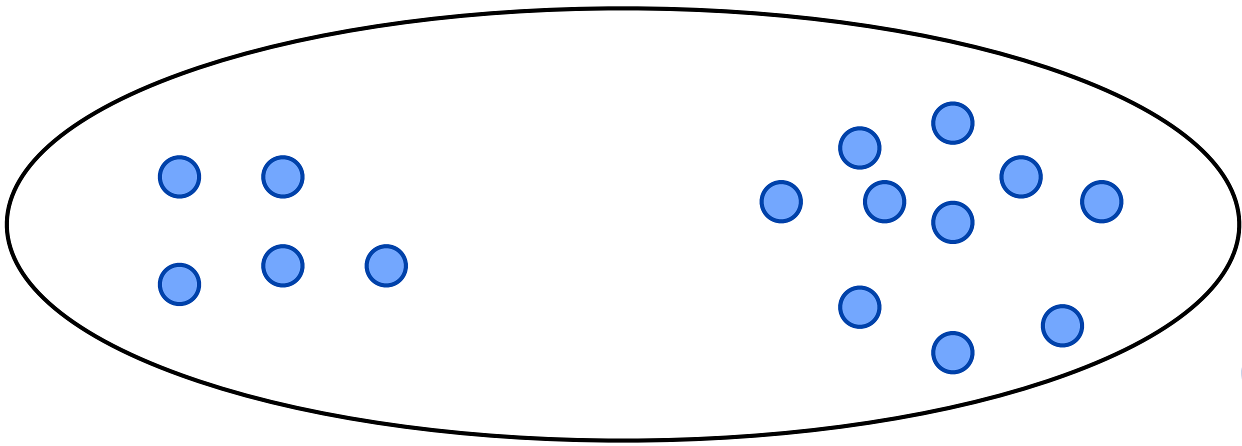
we discard the previous generation



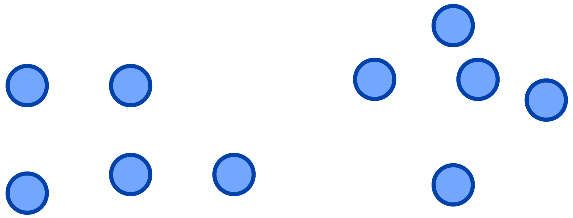


some approaches also add some of the best individuals to the next generation

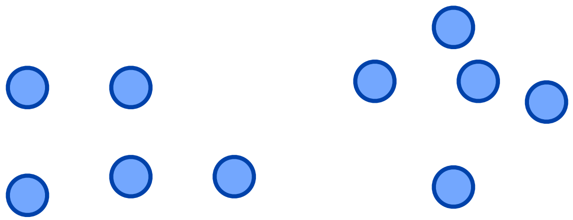
called “elitism”



we now have our population for generation (t+1)

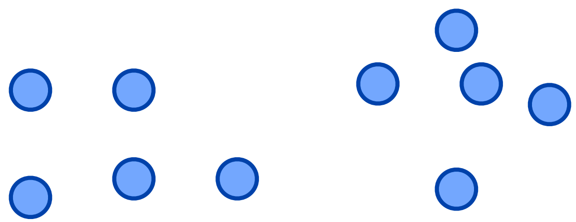


we now have our population for generation (t+1)



if "crossover" and "mutation" were effective then over time (hopefully) the average fitness of the population will improve

we now have our population for generation (t+1)



...and eventually we'll get an individual that has a really high objective value (maybe even optimal, i.e. max possible)

given initial population generation t_0

1. select intermediate population based on fitness

   selection

2. use intermediate population to create offspring

   crossover ***mutation***

3. population (t+1) is offspring

repeat until stop conditions met

initial population can be random or seeded

given initial population generation t_0

1. select intermediate population based on fitness
   selection
2. use intermediate population to create offspring
   crossover   ***mutation***
3. population (t+1) is offspring

repeat until stop conditions met

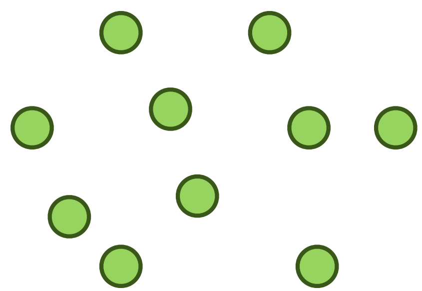
https://www.youtube.com/watch?v=uwz8JzrEwWY

Part 2

selection, mutation, crossover

selection

we need to pick individuals from the population to make the “intermediate population”

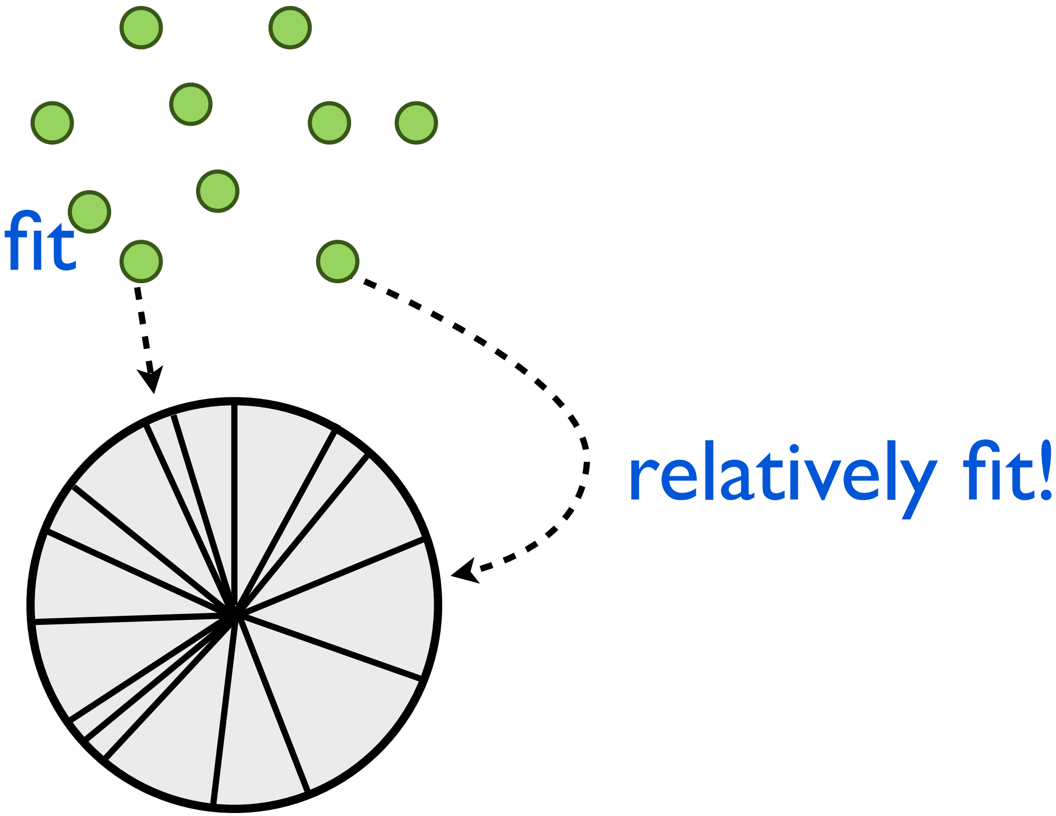


many approaches: an individual is more likely to be selected in proporition to their relative fitness within the population

fit individuals are more likely to be selected

selection

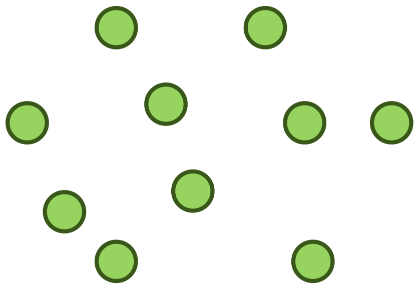
we need to pick individuals from the population to make the "intermediate population"



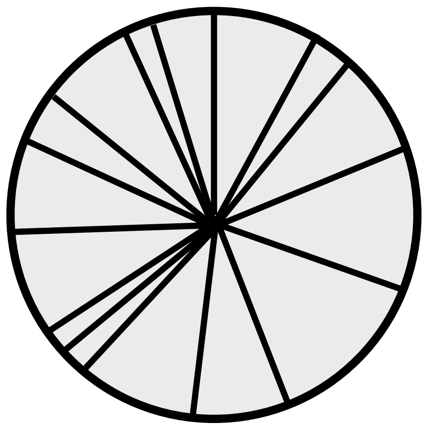
e.g. imagine a roulette wheel where each sector (wedge) corresponds to an individual - the area is in proportion to the individual's relative fitness

selection

we need to pick individuals from the population to make the "intermediate population"



spin the roulette wheel to select copies of individuals to add to the intermediate population



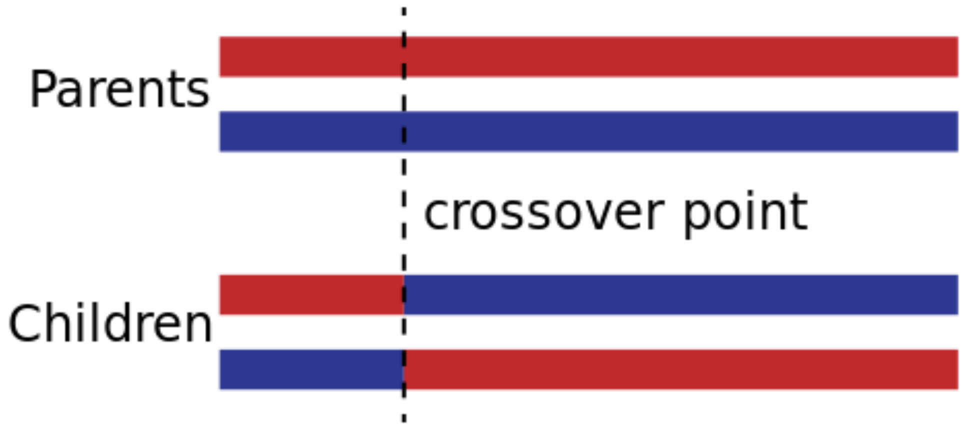
higher fitness
= more area
= more chance to get selected

crossover

- combining "parents" (candidate solutions) to get new "children" (new candidate solutions)
- randomly choose pairs of individuals
- swap parts of their chromosomes with each other
- result is "children"

https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

single-point crossover



select a crossover point
all bits of parents are swapped after that point

single-point crossover

parents

010 100 | 111 110 001 010 101

111 011 | 101 010 101 110 001

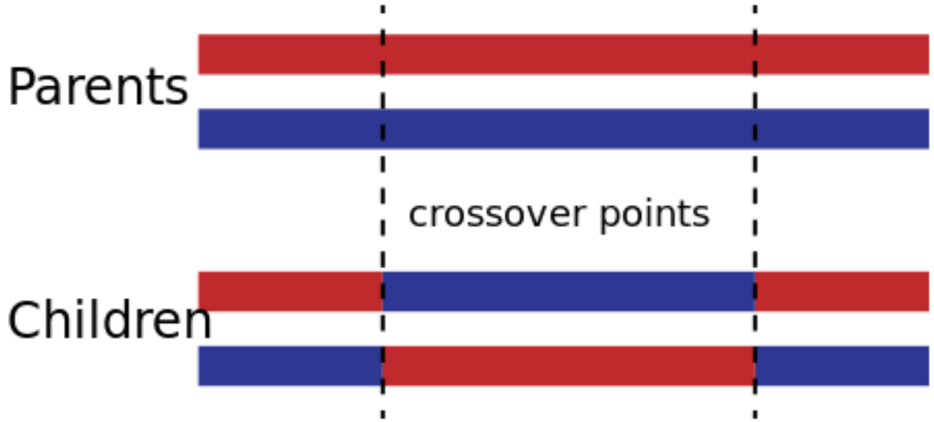
children

?

single-point crossover

parents	010 100	111 110 001 010 101
	111 011	101 010 101 110 001
children	010 100	101 010 101 110 001
	111 011	111 110 001 010 101

two-point crossover



select two crossover points
all bits of parents are swapped between these points

uniform crossover

children have some ratio of genes from either parent, e.g. “half uniform” children have approximately half genes from each parent (mixing ratio: 0.5)

uniform crossover

each bit in “Parent A” is evaluated,
randomly decide (with probability of mixing ratio) if swap
with corresponding bit from “Parent B”

mutation

- used to keep genetic diversity between populations
- analogous to biological mutation
- mutation alters bits in the chromosome

mutation

0010100100101011110101

step through each bit
randomly decide to “flip” bit e.g. with probabiliy 0.1%

mutation

001010010010101110101

step through each bit

randomly decide to “flip” bit e.g. with probabiliy 0.1%

mutation

0010100100000101110101

step through each bit
randomly decide to "flip" bit e.g. with probabiliy 0.1%

given initial population generation t_0

1. select intermediate population based on fitness

   selection

2. use intermediate population to create offspring

   crossover ***mutation***

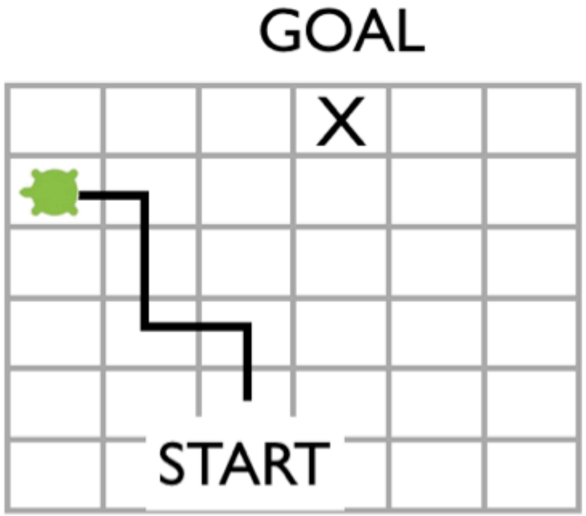
3. population (t+1) is offspring

repeat until stop conditions met

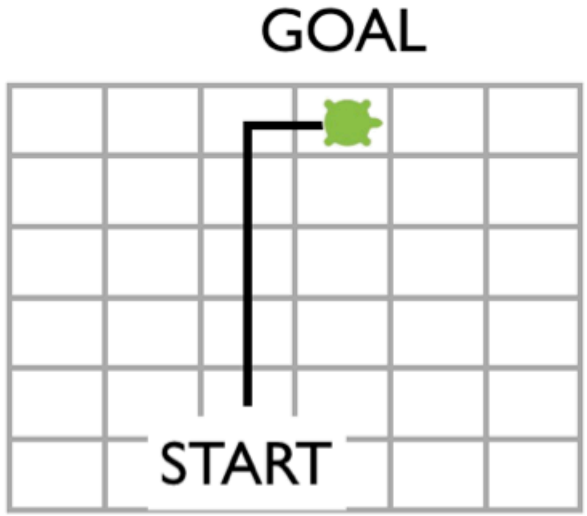
task.

We are helping a robot turtle find some treasure. A "turtle path" always begins at the START square and consists of five steps. At each step the turtle moves one square, either: up, down, left, or right. Below are some examples of turtle paths.

up, right, right, up, left



up, left, up, up, left



up, up, up, up, right

We want to use a genetic algorithm to find instructions so the turtle ends up at the GOAL (where the treasure is) once it stops moving. Each "turtle path" is a candidate solution.

---

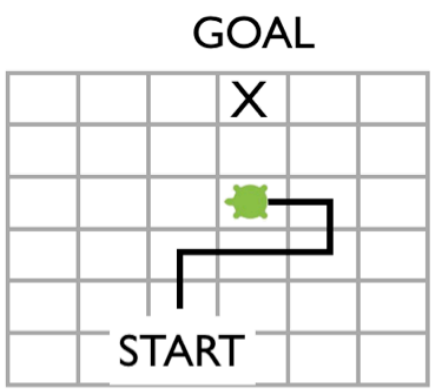
i) Develop a representation of a five step "turtle path" as a chromosome. Explain your representation, and give one example of a chromosome with the corresponding turtle path.

ii) Suggest an objective function that gives an appropriate cost to each candidate solution such that any turtle path that ends at the goal minimises the cost. Give the costs for the three example turtle paths in the pictures below.

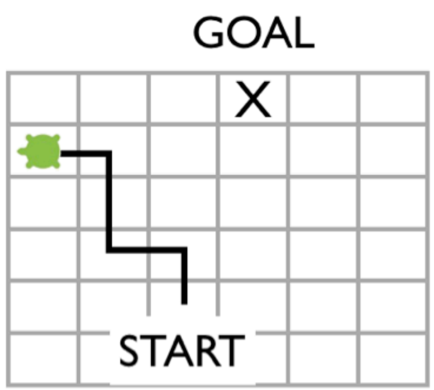
iii) Give an example of single-point cross-over using your chromosomes (choose an example that clearly demonstrates this).

iv) Give an example of mutation using your chromosomes.

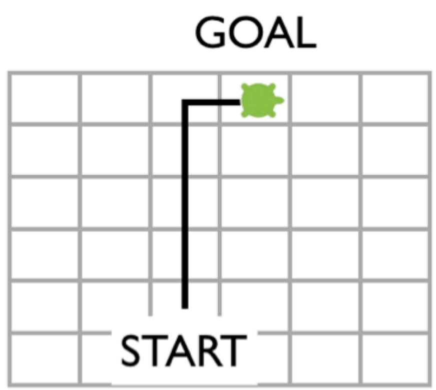
v) Given an initial population P of size N, explain the steps of your genetic algorithm search for ONE generation (about one or two sentences per algorithm step).



up, right, right, up, left

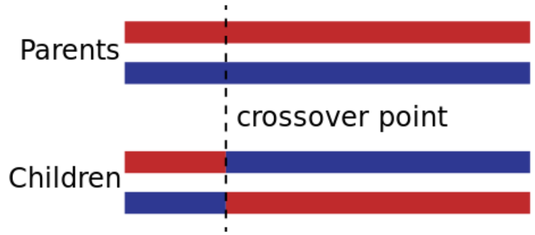


up, left, up, up, left



up, up, up, up, right

TODAY'S SUMMARY



1. genetic algorithms
2. selection, mutation, crossover
3. why do they work?