Lecture notes EC

Evolutionary Computing (Vrije Universiteit Amsterdam)

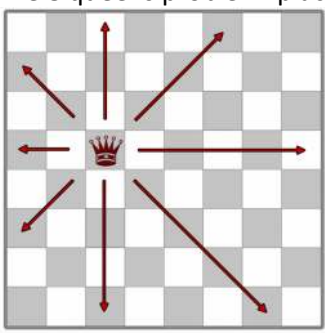
Lecture 1

What is evolutionary computing?

- The field for designing, applying, and studying evolutionary algorithms

Evolutionary heuristic example

The 8 queens problem: place 8 queens on the board



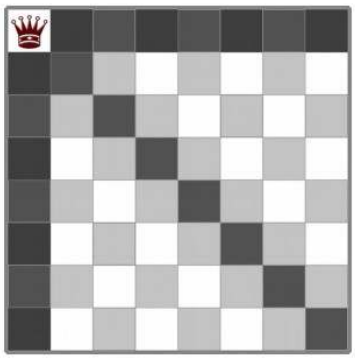
There are two types of problems: generic vs specific
General problem: N > 1 queens. So you don't specify a number (N queens problem)
Specific: N = 8 (8 queens problem instance)

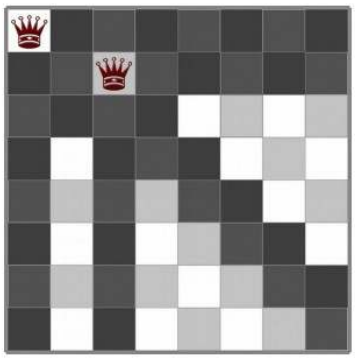
Problem instance is defined by the property 'size' is sufficiently specific
Problem ≠ problem instance

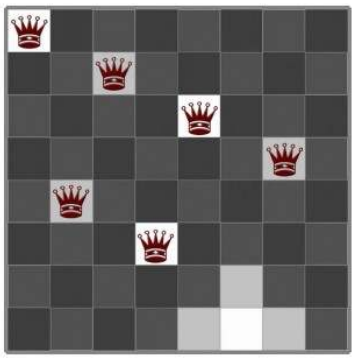
How to solve this problem?

- Method 1:
  - Place queens one-by-one
  - Foll rows from top to bottom, within a row: first available from left to right
  - Backtrack if stuck → we remove the previous queen to an alternative position

Works by extending an empty solution → constructive method
Recursive
Blind (is not doing something smart)
Search trajectory via correct but incomplete configuration



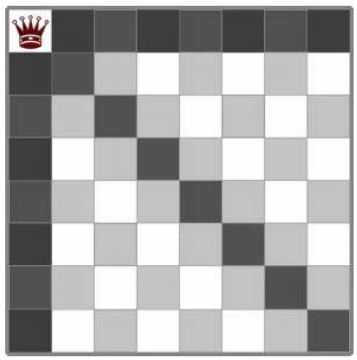


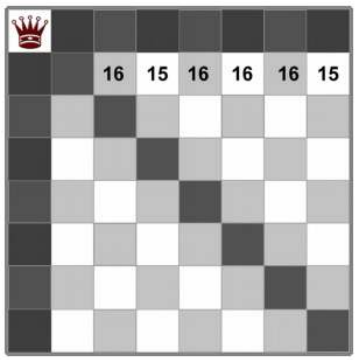


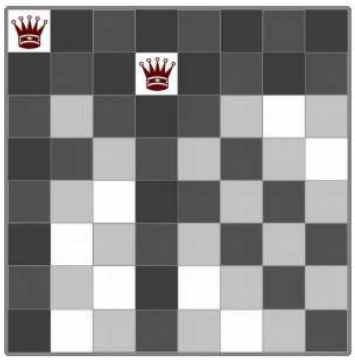
- Method 2:
  - Almost same as method 1
  - Within a row: choose position that checks the least number of other positions
  - Scan from left to right

So we put the second queen at the first number 15. That is the least number of checks for that row

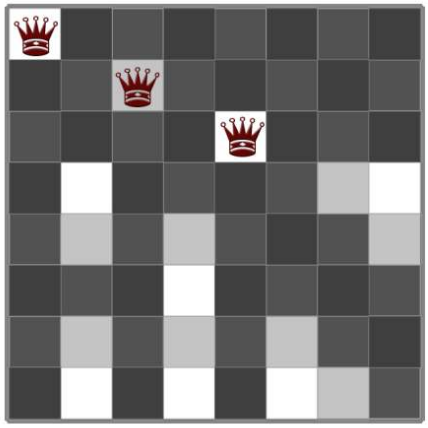
Almost same as method 1
Works by extending an empty solution → constructive method
Recursive
Heuristic: try to minimize need for backtracks
Search trajectory via correct but incomplete configuration

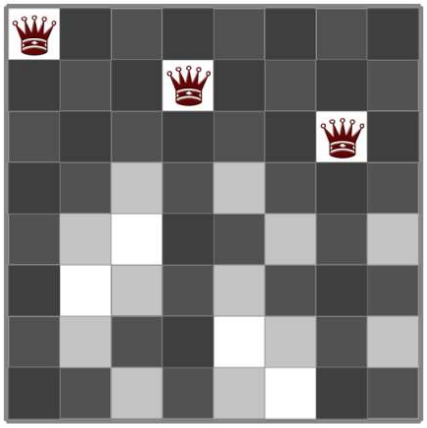






- Solver 1 vs. solver 2





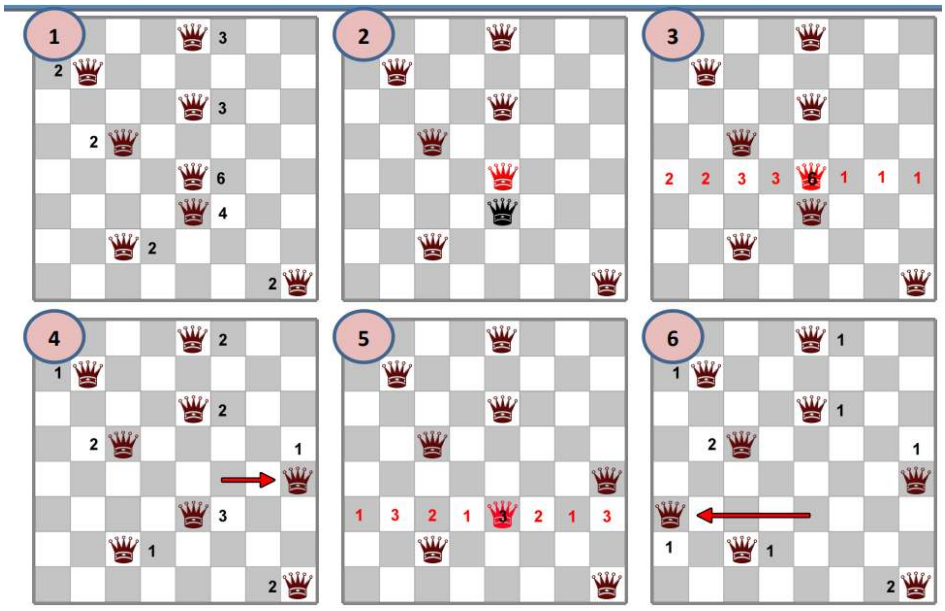
First possible

Minimize “checks”

Solver 1 is more greedy, it takes the first possible position
Solver 2 is more smarter because it is minimizing the checks

- Method 3
    - Place al queens random
    - Repair errors by:
        - Choose queen with most conflicts
        - Move it to the best position in the same row
    - ‘best position’ = with min. number of conflicts
    - Random move if stuck (instead of backtracking)

Works by improving a solution → iterative improvement method
Heuristic: try to maximize improvement via educated guess
Search trajectory via complete but incorrect configurations (so each step all queens are on the board)



- Method 4:
  - Place all queens
  - Improve configuration by
    - Make K > 1 new configurations by a few random changes (‘mutations’)
    - Discard the worst K-1 ‘mutants’ (this keep the best only)
  - Iterate

Works by improving a solution → iterative improvement method
Heuristic: try to maximize improvement via blind mutation and quality based selection among mutants
Search trajectory via complete but incorrect configurations (so each step all queens are on the board)

The EC metaphor
Links the context of natural evolution and artificial evolution

Evolution		Problem solving
Environment	←→	Problem
Individual	←→	Candidate solution
Fitness	←→	Quality

Fitness: chances of survival and reproduction
Quality: change of seeding new solution

EC toolkit:

- Evolvable objects (‘what do you want’) → phenotypes
- Genetic code (to create new individuals) → genotypes
- Reproduction
- Fitness
- Selection

	Natural evolution	Evolutionary algorithms
Fitness	Observed quantity: *a posteriori* ("in the eye of the observer").	Predefined *a priori* quantity that drives selection and reproduction.
Selection	• Complex multi-factor force (environment, other individuals, predators). • Viability is tested continually; reproducibility at discrete times.	• Simple randomized operator based on given fitness values. • Parent selection and survivor selection both happen at discrete times.
Genotype-phenotype mapping	Highly complex biochemical process.	Simple mathematical transformation.
Variation	One or two parents (asexual or sexual reproduction).	Any number of parents, one, two, or many.
Execution	• Parallel, decentralized execution. • Birth and death events are not synchronised.	• Typically centralized execution • Synchronised birth and death.
Population	• Spatial embedding implies structured populations. • Population size varies according to the relative number of death and birth events; populations can and do go extinct.	• Typically unstructured and panmictic (all individuals are potential partners). • Population size is usually kept constant by synchronising time and number of birth and death events.

Lecture 2:

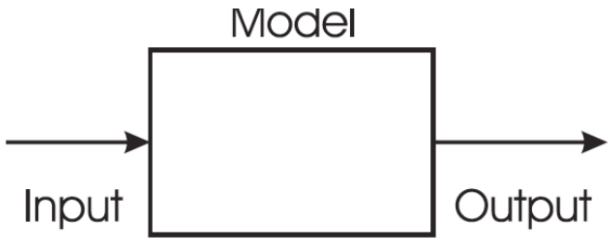
Chapter 1: Problems to be solved

Problems can be classified in different ways:

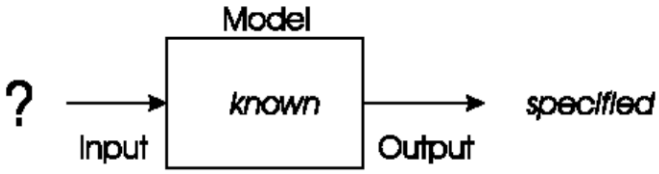
- Black box model
- Search problems
- Optimization vs. constraint satisfaction
- NP problems

- Black box model

Black box models consists of 3 components: input, model and output



Examples of when input is unknown, but model and output are known (optimization problem):

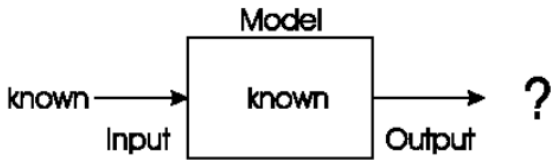


- Time tables
  Big search space
  'Good' is defined by a number of competing criteria
- Traveling salesman problem
- Eight queen problem

Examples of when model is unknown (modelling problem):

- Evolutionary machine learning
- Predicting stock exchange
- Voice control system for smart homes

Examples when output is unknown (simulation):



- Weather forecast system
- Impact analysis new tax system

- Search problems

Simulation is different from optimalisation/modelling
Optimalisation/modelling problems search trough huge space of possibilities
Search space: collection of all objects of interest including desired solution

Problem vs. problem-solvers

- Search problems: which define search spaces
- problem-solvers: which move through search spaces (to find a solution)

- Optimization vs. constraint satisfaction

Objective function: a way of assigning a value to a possible solution that reflects its quality on scale (dus wat je uiteindelijk wilt minimaliseren of maximaliseren)

- Number of un-checked queens (maximized)
- Length of tour visiting a given set of cities (minimize)

Constraint: binary evaluation telling whether a given requirement holds or not

- Property of a chessboard configuration: are there queens that check each other (we are good if 'no')
- property of a tour: is city X visited after city Y (we are good if yes)

we can combine these two:

	Objective function	
Constraints	**Yes**	**No**
Yes	Constrained Optimisation Problem	Constraint Satisfaction Problem
No	Free Optimisation Problem	No problem ☺

Note: constraint problems can be transformed into optimalisation problems

- NP problems

Classification scheme by looking at properties of the problem soler

Benefit: we can define 'problem difficulty' by defining a problem is hard to solve

e.g. this stone is 100 kg → this is a fact vs. this stone is heavy to lift → depends on who is going to lift it

key notations:

- Problem size: number of problem variables (dimensionality)
- Running time: number of operations the algorithm takes to terminate
- Problem reduction: transforming current problem not another via mapping

Class:

The hardness / complexity of a problem can now be classified:

- Class P: some algorithm can solve the problem in polynomial time (worst-case running-time for problem size n is less than F(n) for some polynomial formula F)
- Class NP: problem can be solved and any solution can be verified within polynomial time by some algorithm Note: P is a subset of NP
- Class NP-complete: problem belongs to class NP and any other problem in NP can be reduced to this problem by an algorithm running in polynomial time
- Class NP-hard: problem is at least as hard as any other problem in NP-complete but solution cannot necessarily be verified within polynomial time

Important points:

- Optimization – modeling – simulation scheme (where is the "?" in the diagram)
- FOP – COP – CSP scheme (2 x 2 matrix)
- Problem hardness scheme (based on solver properties)
- Other schemes exist, e.g., by the type of variables (continuous vs. discrete)

Chapter 2: EC the origins

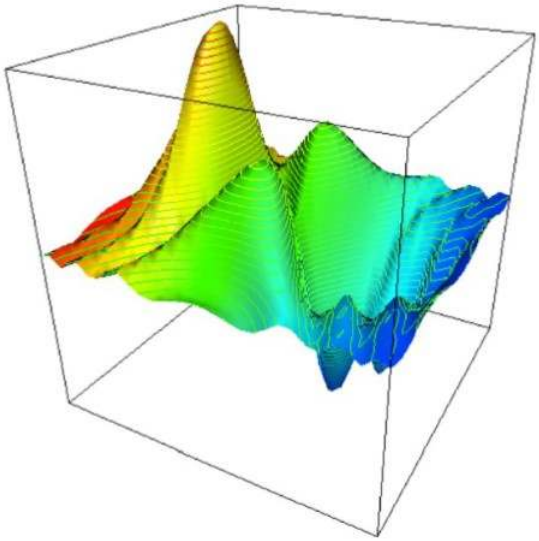
Darwinian evolution

- survival of the fittest (selection):
  - All environments have finite resources (i.e. can only support a limited number of individuals)
  - Life forms have basic instinct/ lifecycles towards reproduction
  - Therefore some kind of evolution is inevitable
  - Individuals that compete for the resources most effectively have increased change of reproduction
- Diversity drives change
  - Phenotype traits
    - Physical & behavioral differences that affect response to environment
    - Partly determined by inheritance, partly by other factors (nature vs. nurture)
    - Unique to each individual, partly as a result of random changes
  - If phenotype trains:
    - Lead to higher chances of reproduction
    - Can be inherited

➔ Then they will tend to increase in subsequent generations, leading to new combination of traits

Adaptive landscape metaphor

- Population can be represented with n traits (like speed, muscle mass ect.) existing in a n+1-dimensional space (+1 stands for the fitness of an individual, corresponds to the height)
- Each individual (different phenotype) is a single point in the landscape
- Population is therefore a 'cloud' of points moving over the landscape as it evolves in time



- Selection 'pushes' population on the landscape
- Genetic drift:
  - Random variation in the feature distribution
  - Can cause the population to 'melt down' (creating valleys) → can cause the population to leave a local optimum!

Genetics

- Genotype: DNA (inside)
- Phenotype: outside

Kijk slides voor verdere info is alleen biologie

Lecture 3

Chapter 3: what is an Evolutionary Algorithm

Common model of evolutionary process

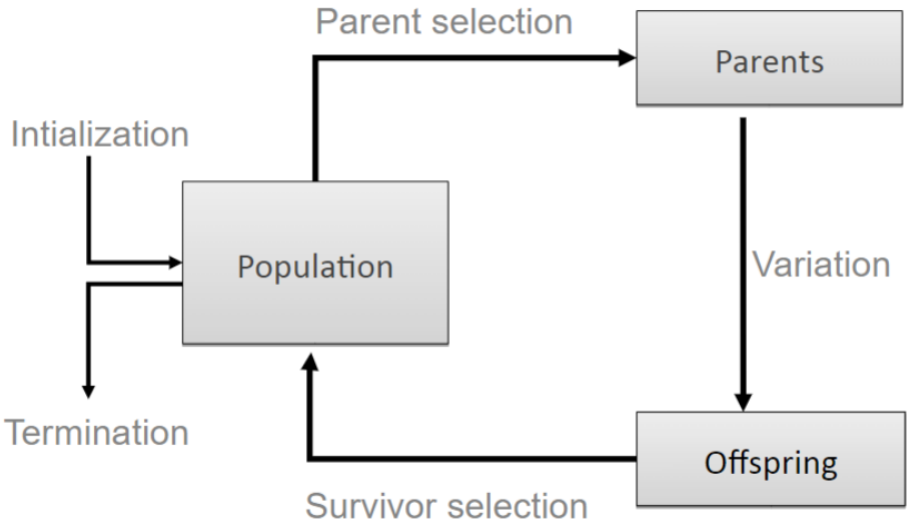
- Population of individuals
- Individuals have a fitness
- Reproduction/variation operators
  - Mutation
  - Recombination (a.k.a. crossover)
- Selection towards higher fitness
  - 'survival of the fittest' and
  - 'mating of the fittest'
- Fitness of the population increases over time

- There are two competing forces in evolution:

Increasing population **diversity** by variation • mutation • recombination Push towards **novelty**	**Decreasing** population **diversity** by selection • of parents • of survivors Push towards **quality**
Brings in fresh blood, new competencies	removing old stuff, not letting stuff reproduce

  - There should be a good balance between novelty and quality!
  - Selection operators act on population level → because they need information about how good is the population
  - Variation operators act on individual level → e.g. one mutated individual

- General scheme of EA:



1. You need an initial population (e.g. 100 random generated tours for TSM problem)
2. select best/better parents for reproduction
3. apply variation/reproduction operators that lead to a new set of indivuals (which are the children/offspring)
   The children compete with the individuals in the population, because the population size is limited and fixed (e.g. 100 in initial population, 100 in the next ect.)
4. Stops when terminal criteria are satisfied

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Main EA components

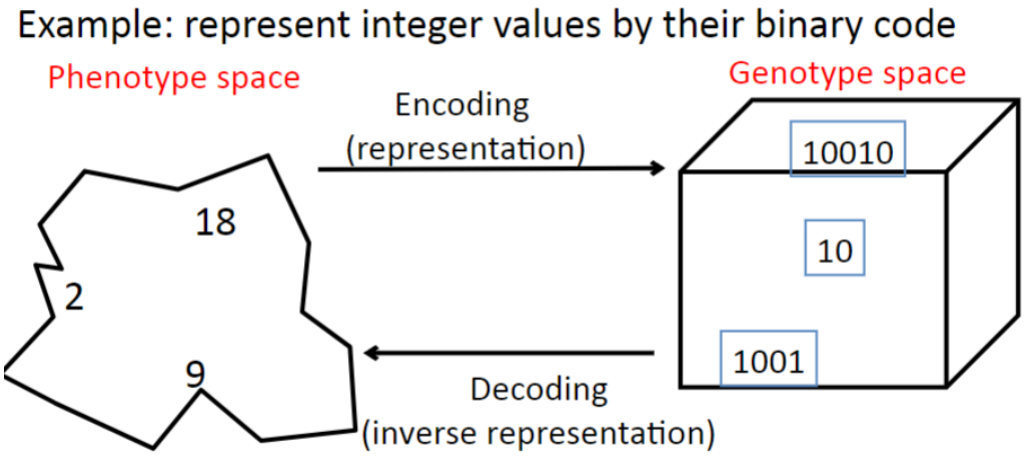
Representation

- Role: provides code for candidate solutions that can be manipulated by variation operators
- the phenotype → original problem context, things you want to encode by 'artificial DNA'
- genotype → to denote the object, the inside

This means we also need two mappings:

- Encoding: phenotype → genotype (doesn't need to be one-to-one)
- Decoding: Genotype → phenotype (should always be one-to-one because otherwise you can't measure the goodness of fit)

Visual illustration of the space:



In the phenotype space we have integer number, we represent the numbers (genotype space) by binary numbers

We see a one-to-one mapping → each integer has only one binary code and for each binary code there is only one integer

Evaluation/fitness function

- Role:
  - Represents the task to solve, the requirements to adapt

- Enables selection

Also known as quality function or objective function

- Assigns a real-valued fitness to each phenotype which forms the basis for selection
  - So the more discrimination (different fitness values) the better → easier for the gradient if landscape is more difficult
- Typically we want to maximize the fitness, but mathematically we can also minimize

Population

- Role: holds the candidate solutions of the problem as individuals (genotypes)
- Formally, a population is a multiset of individuals, i.e. the same element might occur multiple times
- Population is the basic unit of evolution, i.e. the population is evolving, not the individuals!
- Selection operators act on population level
- Variation operators act on individual level
- Some EA assert a spatial structure on the population e.g. a grid
- The selection operators usually take the whole population into account, i.e. reproductive possibilities are relevant to current generation
- Diversity of a population refers to the number of different fitness values/phenotypes/ genotypes present → most important number after the fitness function
  - Diversity is good, because otherwise crossover will not bring you anything new anymore

Selection

Role: pressing the population towards high quality

- Two operators:
  1. Who will reproduce (will become the parents)
  2. Who will survive
- Usually probabilistic /stochastic
  - High quality solutions more likely to be selected than low quality solutions
  - But not guaranteed! (so you could also lose the fittest individual)
- Stochastic nature can help escape the model from local optima!
- In principle, any selection mechanism can be used for parent selection as well as for survivor selection

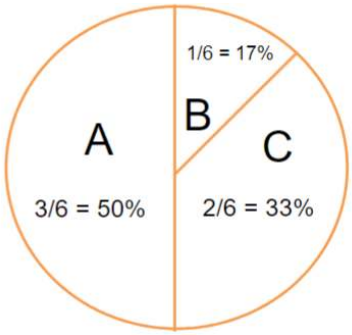
Examples of selection methods:

- Roulette

fitness(A) = 3

fitness(B) = 1

fitness(C) = 2



The proportion of the roulette wheel is related to the fitness value
So A represents half of the population fitness
Then you 'throw in a ball' (like roulette) and it stops somewhere, then there is no bias anymore.
That means that A has 50% chance of being selected, and B 17% ect.

Each individual gets a chance but better individuals have a higher chance

- Replacement (or survival selection)

Most EAs use fixed population size, so there need to be a way of going from parents + offspring to next generation

- Usually deterministic
    - Fitness based: e.g. rank parents + offspring and take best
    - Age based: make as many offspring as parents and delete all parents
- Sometimes a combination of stochastic and deterministic
- e.g. elitism (the best n individuals survive) is a deterministic rule
  if this is the case, the fitness (plot) never goes down!
  Sometimes you don't want elitism because it slows down convergence

<u>Variation operators</u>

- Role: generate new candidate solutions
- Can be distinguished by their arity (number of inputs)

- Arity 1: mutation operators (one input and produce one output)
- Arity > 1: recombination operators
- Arity = 2: crossover (two inputs, produce one or 2 outputs)
- Arity >2: multi parent reproduction (not very often used)

You can use both operators at the same time! (so you don't need to only use mutation or only crossover)
Variation operators must match the given representation

<u>Mutation</u>

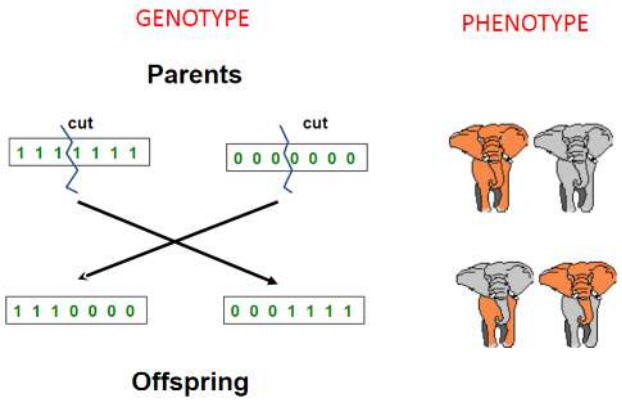
Role: causes small, random variations

- Randomness is very important here! So you don't have a bias
- Acts on one genotype and delivers another
- May guarantee connectedness of search space and hence convergence proofs (will come back later)

<u>Recombination</u>

Role: merges information from parents into offspring

- Combining different parts of the parents genotypes
- Choice of what information to merge is stochastic → of course you want only the good traits to combine but it is stochastic
- Most offspring may be worse, or the same as the parents → not bad because we have selection, and over time the good qualities will survive anyway



Initialization/termination

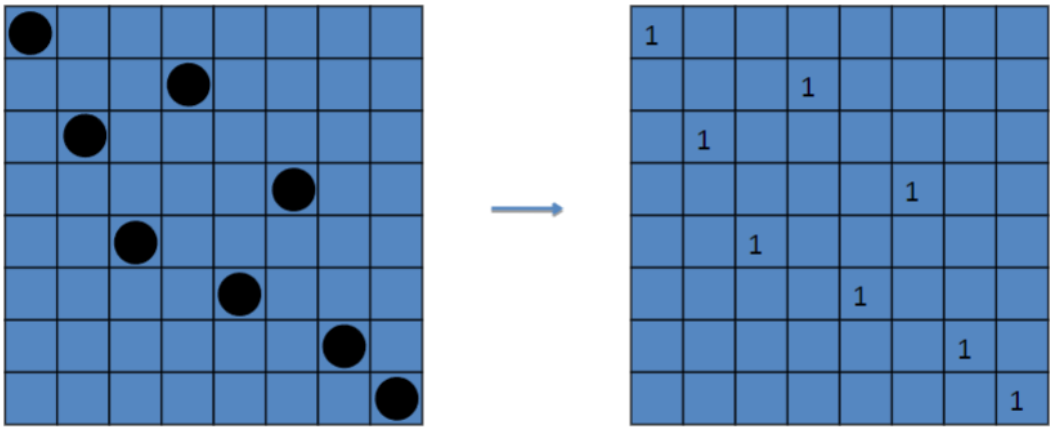
- Initialization
  - Usually random
  - Need to ensure even spread and mixture of possible values
  - Can include existing solutions or use problem-specific heuristics to 'seed' the population
- Termination
  - Checked every generation
  - Stop when reaching some fitness level
  - When reaching some max number of generations
  - When reaching some min level of diversity
  - When reaching some specified number of generations without fitness improvement

Different types of EAs

- Different types of EAs have been associated with different data types to represent solutions:
  - Binary strings: genetic algorithms
  - Real-values vectors: evolution strategies
  - Finite state machines: evolutionary programming
  - LISP trees: genetic programming
- These historical differences are largely irrelevant, the best strategy is:
  - Choose representation that suit your problem
  - Choose variation operators that suit your representation
- Selection operators only use fitness and therefore independent of representation

Exercise: 8-queens problem

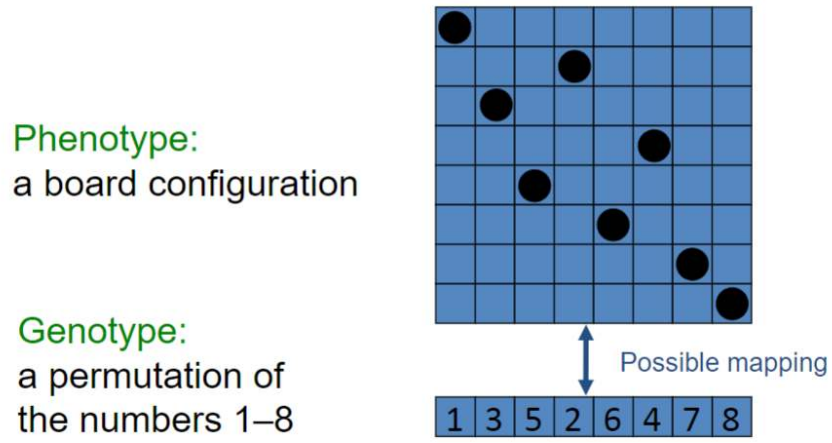
Representation:



Phenotype genotype

Represent by putting a 1 where there is queen

This leads to: table of 8x8 or bitstring of length 64
But there is more clever representation:



The genotype shows the height in the column of the queen
Here you can exclude the possibility of two queens being in the same column
Also use permutation, so there are no identical elements i.e. no identical row numbers

- Reduction of dimensional space
- Here there is no mutation possible!!

Fitness evaluation:

- Penalty of one queen: the number of queens she can check
- Penalty of a configuration: the sum of all penalties of all queens
- Note: the penalty is to be minimized

Mutation:

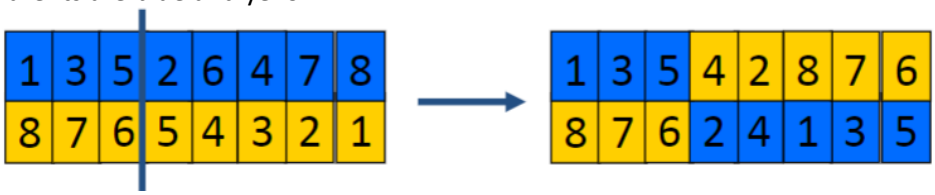
- Small variation in one permutation, e.g., swapping values of two randomly chosen position

Recombination:

- Combining two permutations into two new permutations:
  - Choose random crossover point

- Copy first parts into children
- Create second part by inserting values from other parent:
  - In the order they appear
  - Beginning after crossover point
  - Skipping values already in child

Parents are blue and yellow



Selection:

- Independent of representation
- Parent selection
  - Pick 5 parents randomly and take best two to undergo crossover
- Survivor selection (replacement)
  - When inserting a new child into the population, choose an existing member to replace e.g.:
    - Worst individual
    - The best individual that is worse than this child
      - Sorting whole population by decreasing fitness
      - Enumerating this list from high to low
      - Replacing the first with a fitness lower than the given child

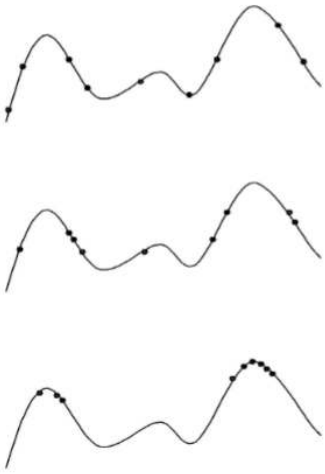
Summary:

Representation	Permutations
Recombination	"Cut-and-crossfill" crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluation

Typical EA behaviour: stages

- Stages in 1-dimensional fitness landscape

In early stage they are equally distributed over the whole fitness landscape (all over the place)



Early stage:
quasi-random population distribution

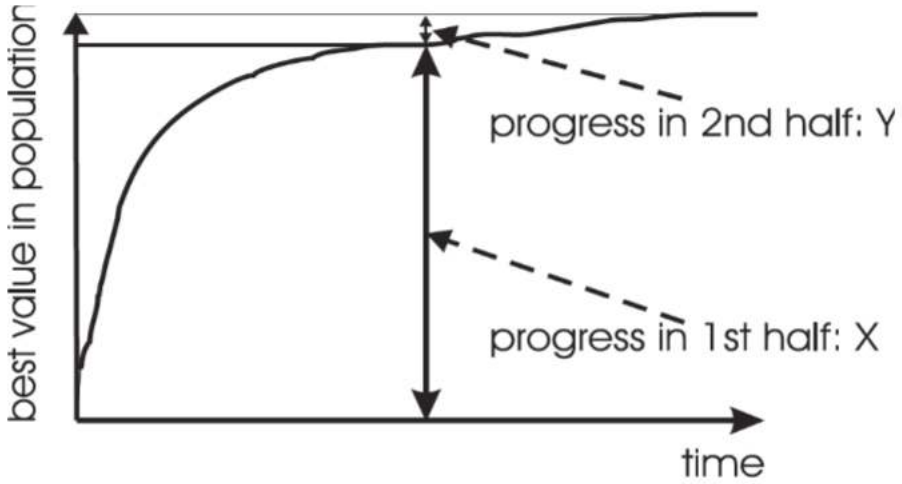
Mid-stage:
population arranged around/on hills

Late stage:
population concentrated on high hills

- Progress or behavior curve:

This behaviour is called 'anytime behavior' because you can stop it anytime

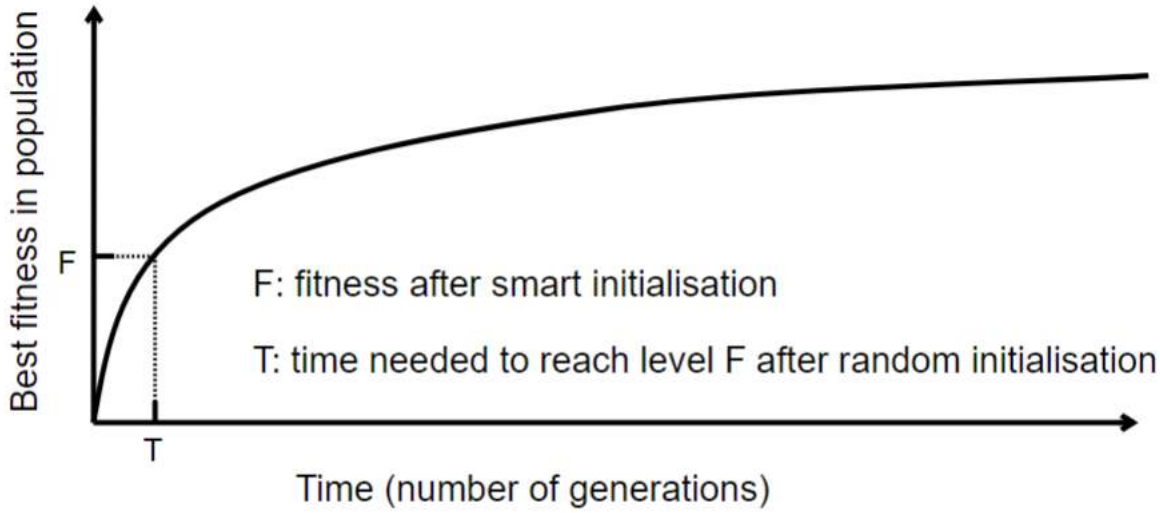
- Are long runs beneficial?



- It depends on how much you want the last bit of progress (in the beginning a lot of process, after less)
- May be better to do more short runs (e.g. two short ones instead of 1 long one)

- Try to fit these curves and choose a stopping time

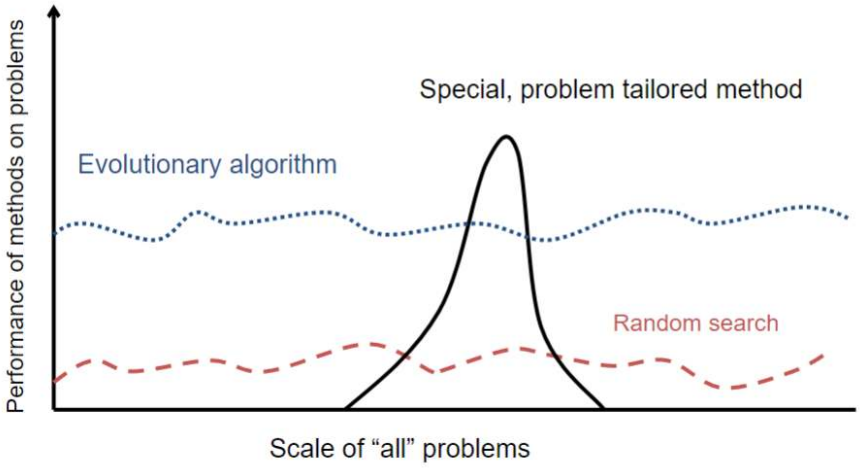
- Is it worth to have a smart initialization?



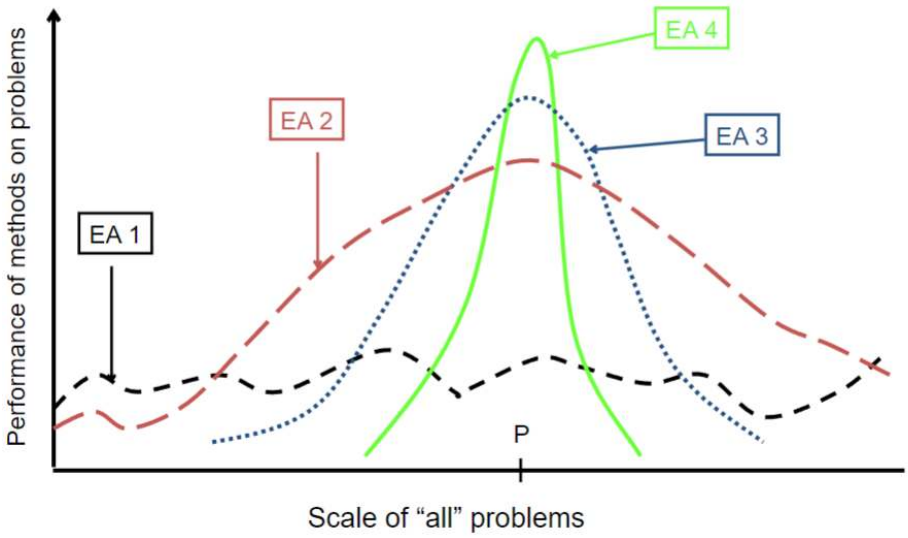
- Possibly good, if good solutions/methods exist
- May to be necessary because EA will catch up quickly
- But could be good, if you have domain knowledge
- But usually don't bother to do this

Evolutionary algorithms in context

- EAs fall into the category of generate-and-test algorithms, a.k.a. trial-and-error algorithms
- They are stochastic, population-based search algorithms
- For most problems, a problem-specific tool may:
    - perform better than a generic search algorithm on most instances
    - have limited utility
    - not do well on all instance
- EAs are meant to provide a robust tool that shows:
    - evenly good performance
    - over a range of problems and instances



- Adding domain knowledge will increase the performance curve:



EC and global optimization

- Global optimisation: search for finding best solution x* out of some fixed set S
- Deterministic approaches
  - e.g. box decomposition (branch and bound etc)
  - Guarantee to find x*
  - May have bounds on runtime, usually super-polynomial
- Heuristic approaches (generate and test)
  - rules for deciding which $x \in S$ to generate next
  - no guarantees that best solutions found are globally optimal
  - no bounds on runtime

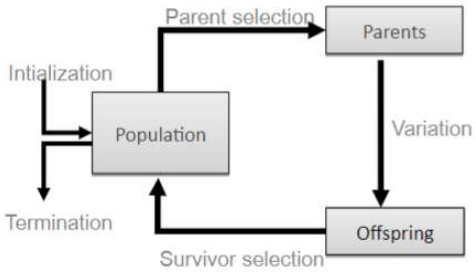
EC and neighbourhood search

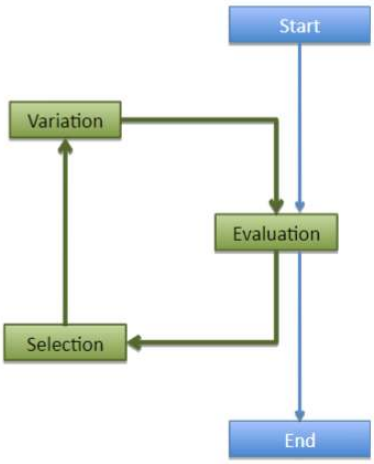
- Many heuristic approaches impose a neighbourhood structure on S
- Such heuristics may guarantee that best point found is locally optimal
  - But problems often exhibit many local optima
  - Often very quick to identify good solutions
- EAs are distinguished by (the combination of):
  - Use of population
  - Use of multiple, stochastic search operators
  - Variation operators with arity > 1, thus combining information of more candidate solutions
  - Stochastic selection

Two ways to visualize the general scheme

They are both the same

- Right one the blocks stand for operators and describes the process
- left describes sets of individuals and the arrows are the operations





chapter 4: representation, mutation, recombination

First stage of building an EA and most difficult one: choose a good representation for the problem
Variation operators: mutation and crossover
Type of variation operators needed depends on chosen representation

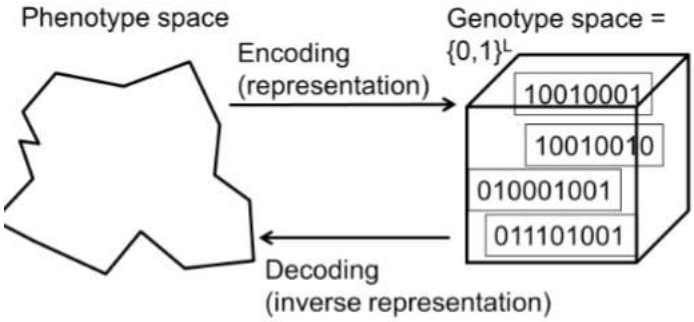
- Most common representation of genomes:
  - Binary
  - Integer
  - Real-valued or floating point
  - Permutation
  - Tree

- A representation is:
  - The genotype space (where to map phenotypes)
  - The mapping (how is this done)

Example of representation (dis)continuity
Phenotypes: positive integers
Genotypes: binary strings
Mapping: the usual binary representation of integers
Problem: the phenotypic distance between 7 and 8 is small, but the genotypic distance is big (111 and 1000) → so there is a discontinuity since the distance has a break
Solution: other mapping, for instance gray code (a.k.a. reflected binary code)

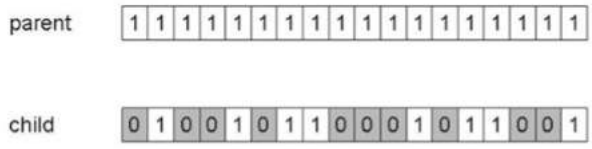
Strong causality principle (to solve the discontinuity problem)
- Small alterations in the underlaying structure of an object cause small changes in the object's behavior
- Small change in cause (genotype) → small change of the effect (phenotype)
- Without this principle, no continuity would be ensured in the evolutionary process and you just have random sampling

Binary representation
- One of the earliest representations
- Genotype consist of a string of binary bits



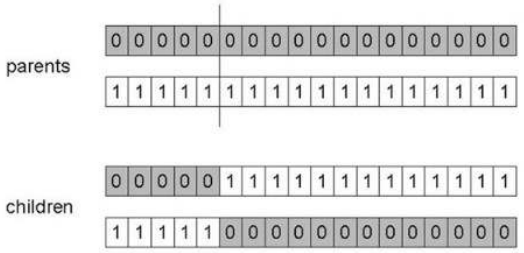
Mutation

- Alter each gene (so flip the number) independently with a probability p_m
- P_m is called the mutation rate
    - Typically between 1/pop_size and 1/chromosome length



1-point crossover

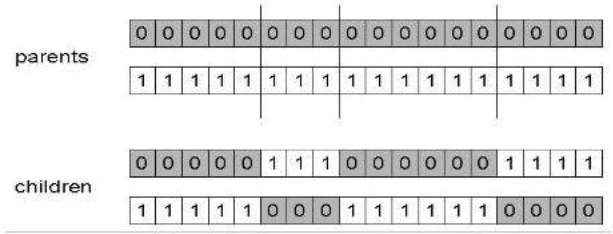
- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



n-point crossover (Alternative crossover)

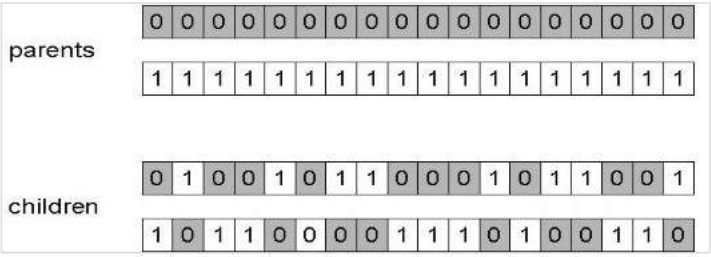
- There are some problems with the 1-point crossover:
    - It depends on the order that variables occur in the representation
    - More likely to keep together genes that are near each other
    - Can never keep together genes from opposite ends of string
    - ➔ This is known as positional bias

- n-point crossover
    - choose n random crossover points
    - split along those points
    - Glue parts, alternating between parents
    - Generalization of 1-point (still has some positional bias)



Uniform crossover

- Assign ‘heads’ to one parent, ‘tails’ to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position



Crossover OR mutation?

- There was a long debate which one is better
- In general it is good to have both (both have another role)
- there are mutation-only-EAs (some evolution strategies), as well as xover-only-EAs (some genetic programming)

- Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem
- Exploitation: Optimising within a promising area, i.e. using information
  - ➔ Make trade-off

- Crossover is explorative, it makes big jumps to an area somewhere ‘in between’ two (parent) areas
- Mutation is exploitive, it creates random small variations, thereby staying near the parent

- Only crossover can combine information from two parents
- Only mutation can introduce new information (new alleles, ‘fresh blood’)
- Crossover does not change the allele frequencies of the population

Integer representation

- Nowadays, it is generally accepted that it is better to encode numerical variables directly (integers, floats) instead of encoding them as binaries
- Some problems already have naturally integer values e.g. image processing parameters, where you have natural ordering and notion of distance for numbers
- Others take categorical values from a fixed set (e.g. blue, green, yellow), where you do not have a logical order or distance measure → these are symbols, not numbers

- N-point / uniform crossover operators work as previously
- Bit-flip mutation can be generalized to random resetting: with probability p_m a new value is chosen at random

- New options are possible e.g. we can extend bit-flipping mutation to make a 'creep' i.e. more likely to move to similar value (if similar is meaningful by adding a small (positive or negative) value to each allele with probability p)

Real-values or floating point representation

- When problems occur as real valued problems (real numbers), we need continuous parameter optimization

Mapping real values on bit strings

$z \in [x,y] \subseteq \mathcal{R}$ represented by $\{a_1,\ldots,a_L\} \in \{0,1\}^L$ where L is the user defined chromosome length

- $[x,y] \rightarrow \{0,1\}^L$ must be invertible (one phenotype per genotype)
- $\Gamma: \{0,1\}^L \rightarrow [x,y]$ defines the representation
  - If you choose to represent x and y onto a binary string
    - E.g. 10 long → 2^10=1024 different discrete points
  - Must also be invertible → for every binary string you must be able to determine a real value number
  - Be aware that you are losing information and accuracy
    - To increase precision the chromosomes need to be longer → slow evolution

Uniform mutation

- If you don't want to do the mapping real values on bit strings and want use the real values → use real value vectors
- General scheme of floating point mutation

$$\bar{x} = \langle x_1, \ldots, x_l \rangle \rightarrow \bar{x}' = \langle x_1', \ldots, x_l' \rangle$$

$$x_i, x_i' \in [LB_i, UB_i]$$

Convert vector x with mutation into x' in a way the values in x' are still between the lower bound and upper bound of the particular component

- Uniform mutation

$$x_i' \text{ drawn randomly (uniform) from } [LB_i, UB_i]$$

Analogous to bit-flipping (binary) or random resetting (integers)

Non uniform mutation

- Use Non uniform mutations when you want to:
  - Change the range of the chain
  - Time varying range
- Most schemes are probabilistic but usually only make a small change to value
- Most common method is to add random noise to each variable separately, taken from N(0,σ) gaussian distribution and then curtail to range
  - X'_i=x_i+ N(0,σ)

    - Keeping the mean 0 is good because then on average stepping left or right will on average end up in the middle
- Standard deviation σ is called the mutation step size → controls amount of change (2/3 of random drawings will lie in range (-σ to + σ)

Self-adaptive mutation

- Encode mutation step size sigma into the chromosomes → the values of sigma will evolve with the values of the original vector we want to optimize
- Step-sizes are included in the genome and undergo variation and selection themselves: $\langle x_1,\ldots,x_n, \sigma \rangle$
- Mutation step size is not set by user but coevolves with solutions
- Different mutation strategies may be appropriate in different stages of the evolutionary search process

- Mutate σ first
- Net mutation effect: <x, σ> → <x', σ'>
- Order is important!:
    - First change σ so: σ → σ'
    - Then calculate x with new σ': x→ x'= x + N(0, σ')
- Why do it like this?
    - σ cannot be evaluated directly (there is not fitness value defined for σ, only for x)
    - so we can calculate x' with f(x') but not f(σ')
    - x' is good if f(x') is good, σ' is good if x' created is good
- reversing mutation order DOES NOT WORK!

Uncorrelated mutation with one σ

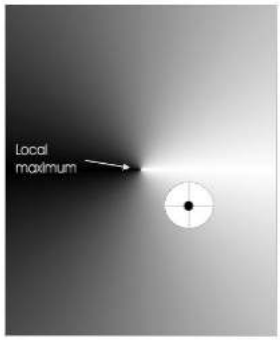
Lognormal mutation algorithm

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma \rangle$
    - $\sigma' = \sigma \cdot \exp(\tau \cdot N(0,1))$
    - $x'_i = x_i + \sigma' \cdot N_i(0,1)$
- Typically the learning rate $\tau \propto 1/\sqrt{n}$

  symbol ∝ means "close to"
- And we have a boundary rule $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$

- For every i in x'_i you draw a new random number → so you don't add the same random noise for each value
- τ has some recommended value from the literature
- boundary rule: if the new σ (i.e. σ') is too close to 0 we change it to our boundary value

- What we do is shown on this picture:



- this is the fitness landscape
- The black dot is one particular point (individual on this landscape)
- the circle around the point denotes all the that have equal probability to become the mutant

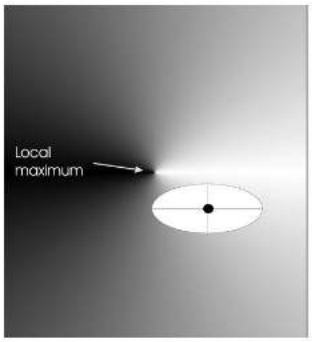
Uncorrelated mutation with n σ

- Generalization of what is shown above
- Separate mutation step size of every coordinate
- For each x_i we have a value of σ_i
- So we have a length of 2n
- We have two learning rates and two random numbers
  - τ' is same for all coordinates
  - τ does depend on the coordinate

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n \rangle$
  - $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
  - $x'_i = x_i + \sigma'_i \cdot N_i(0,1)$
- Two learning rate parameters:
  - τ' overall learning rate
  - τ coordinate wise learning rate
- $\tau' \propto 1/(2n)^{1/2}$ and $\tau \propto 1/(2n^{1/2})^{1/2}$
- Boundary rule: $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$

Now we have to adjust our picture

- Mutants with equal likelihood:



- Now it does not hold that you step away in every direction with the same probability → directions with bigger σ values will be preferred
- More ellipse shape

Correlated mutations

- Here the mutations are not independent for each coordinate
- Chromosomes are even longer
- k are the different values in the covariance matrix

- Chromosomes: $\langle x_1,...,x_n, \sigma_1,..., \sigma_n, \alpha_1,..., \alpha_k \rangle$

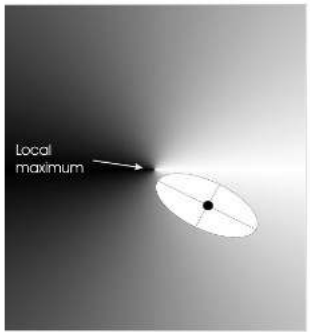
  where $k = n \bullet (n-1)/2$
- Covariance matrix C is defined as:
  - $c_{ii} = \sigma_i^2$
  - $c_{ij} = 0$ if i and j are not correlated
  - $c_{ij} = \frac{1}{2} \bullet (\sigma_i^2 - \sigma_j^2) \bullet \tan(2\alpha_{ij})$ if i and j are correlated
- Note the numbering / indices of the α's

  - Also change the α values
  - X' will be calculated using the covariance matrix
  - Complicated mechanism but it is very popular
    - It can change the step sizes in such a way that you have a very powerful mutation algorithm
    - Especially for numerical optimization!

The mutation mechanism is then:

- $\sigma'_i = \sigma_i \bullet \exp(\tau' \bullet N(0,1) + \tau \bullet N_i(0,1))$
- $\alpha'_j = \alpha_j + \beta \bullet N(0,1)$
- $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{N(0,C')}$
  - $\boldsymbol{x}$ stands for the vector $\langle x_1,...,x_n \rangle$
  - $\boldsymbol{C'}$ is the covariance matrix $\boldsymbol{C}$ after mutation of the α values
- $\tau \propto 1/(2n)^{\frac{1}{2}}$ and $\tau \propto 1/(2n^{\frac{1}{2}})^{\frac{1}{2}}$ and $\beta \approx 5°$
- $\sigma'_i < \varepsilon_0 \Rightarrow \sigma'_i = \varepsilon_0$ and
- $|\alpha'_j| > \pi \Rightarrow \alpha'_j = \alpha'_j - 2\pi \, \text{sign}(\alpha'_j)$
- Note: Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is one of the best EAs for numerical optimisation (open source!)

  - Now the mutant space looks like this:



Real-valued representation: Recombination

- Discrete:

- Each allele value in offspring z comes from one of its parents (x,y) witg equal probability: z_i = x_i or y_i
- Could use n-point crossover or uniform uniform crossover

- Using real valued numbers:
  - Exploits idea of creating children 'between' parents (a.k.a. arithmetic recombination (mean))
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $\alpha : 0 \leq \alpha \leq 1$.
  - The α can be:
    - Constant: uniform arithmetical crossover
    - Variable (e.g. depend on the age of the population)
    - Picked at random every time

Single arithmetic crossover

- Parents: $\langle x_1,...,x_n \rangle$ and $\langle y_1,...,y_n \rangle$
- Pick a single gene (k) at random,
- $child_1$ is:

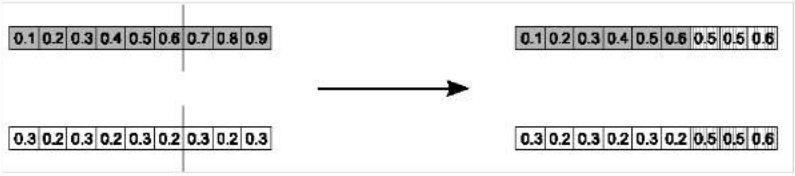
$$\langle x_1,...,x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k,...,x_n \rangle$$

- Reverse for other child. e.g. with $\alpha = 0.5$

Simple arithmetic crossover

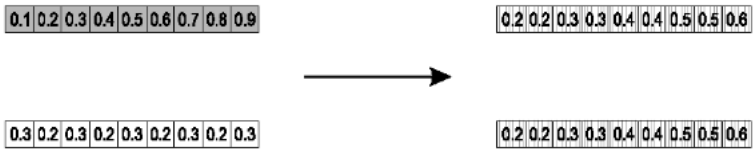
- Parents: $\langle x_1,...,x_n \rangle$ and $\langle y_1,...,y_n \rangle$
- Pick a random gene (k) after this point mix values
- $child_1$ is:

- reverse for other child. e.g. with $\alpha = 0.5$



Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1,...,x_n \rangle$ and $\langle y_1,...,y_n \rangle$
- $Child_1$ is:
- reverse for other child. e.g. with $\alpha = 0.5$

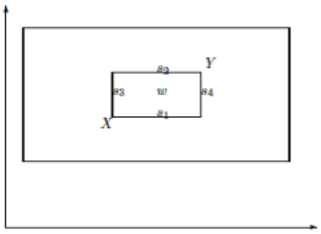


Blend crossover

- Parents: $\langle x_1,...,x_n \rangle$ and $\langle y_1,...,y_n \rangle$
- Assume $x_i < y_i$
- $d_i = y_i - x_i$
- Random sample $z_i = [x_i, y_i]$
- Random sample $z_i = [x_i - \alpha d_i, y_i + \alpha d_i]$
- Original authors had best results with $\alpha = 0.5$

  - New value of the child is drawn from an interval larger than the interval of the parents

  - Example where the values will be for the crossovers:
    - Single arithmetic places the offspring somewhere on s_1, s_2, s_3 or s_4
    - Whole is anywhere in the inner box, rectangle between s_1, s_2, s_3 and s_4
    - Blend is in the outer box → gives more exploration power than traditional operators



- Single arithmetic: $\{s_1, s_2, s_3, s_4\}$
- Whole arithmetic: inner box (w if alpha = 0.5)
- Blend crossover: outer box

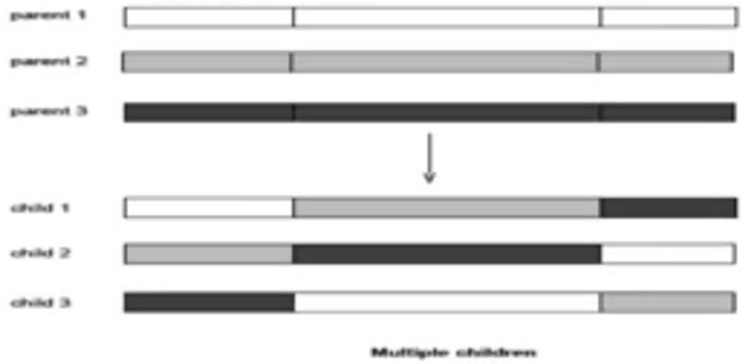
Multi-parent recombination

- We are not restricted to 2 parents (like in nature)
- Noting that mutation uses n = 1 parent, and traditional crossover n = 2, the extension to n > 2 is natural to examine
  - Been around since 1960s, still rare but studies indicate useful

Multi-parent recombination, type 1

- Idea: segment and recombine parents

- Example: diagonal crossover for *n* parents:
  - Choose *n-1* crossover points (same in each parent)
  - Compose *n* children from the segments of the parents in along a "diagonal", wrapping around



- This operator generalizes 1-point crossover

Multi-parent recombination, type 2

- Idea: arithmetical combination of (real-valued alleles)
- Example: arithmetic crossover for n parents (take mean of all parents for position in child)
  - i-th allele in child is the average of the parents' i-th alleles
- creates center of mass as child
- odd in genetic algorithms, long known and used in evolution strategies → called intermediary recombination

Permutation representation

- Ordering/sequencing problems form a special type
- Task is (or can be solved) by arranging some objects in a certain order
- These problems are generally expressed as a permutation:
  - If there are n variables then the representation is a list of n integers, each of which occurs exactly once

TSP example

So listing the cities in the order they will be visited

- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:
- for 30 cities there are 30! ≈ 10^{32} possible tours

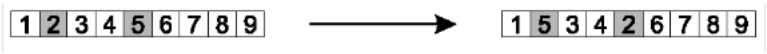
Permutation representation: mutation

- Normal mutation operators could lead to inadmissible solutions
  - E.g. bit-wise mutation: let gene i have value j,
  - changing some other value to k would mean tot k occurred twice and j no longer occurred

  ➔ therefore must change at least two values
- mutation parameters now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

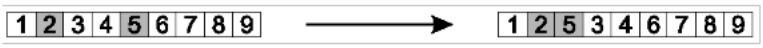
Swap mutation

- pick two alleles at random and swap their positions



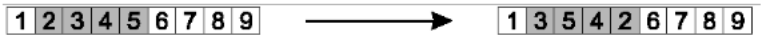
Insert mutation

- pick two alleles at random and swap their positions
- Move the second to follow the first, shifting the rest along
- Note that this preserves most of the order and the adjacency information



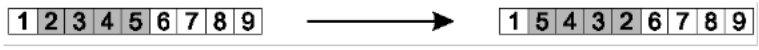
Scramble mutation

- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions



Inversion mutation

- Pick two alleles at random and invert the substring between them
- Preserves most adjacency information (only breaks two links) but disruptive of order information



Permutation representations: crossover

- Normal crossover operators will often lead to inadmissible solutions:

Order 1 crossover

- Idea: preserve relative order that elements occur
- Procedure:
  1. Choose an arbitrary part from first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child
     a. Starting right from cut point of the copied part
     b. Using the order of the second parent
     c. Wrapping around the end

4. Analogous for second child, with parent roles reversed

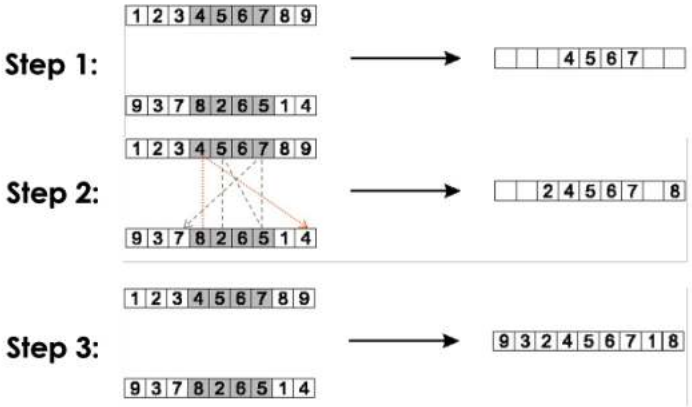
- Copy randomly selected set from first parent

- Copy rest from second parent in order 1,9,3,8,2

Partially mapped crossover (PMX)

- Procedure:
  1. Choose random segment and copy from P1
  2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
  3. For each of these i look in offspring to see what element j has been copied in its place from P1
  4. Place i into the position occupied j in P2, since we know that we will not be putting j there (as is already in offspring)
  5. If the place occupied by j in P2 has already been filled in the offspring k, put i in the position occupied by k in P2
  6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

  Second child: same procedure with P1 & P2 inverted

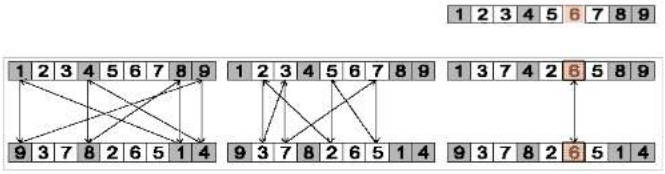


Cycle crossover

- Each allele comes from one parent together with its position.
- Informal procedure:
  1. Make a cycle of alleles from P1 in the following way.
     a. Start with the first allele of P1.
     b. Look at the allele at the same position in P2.
     c. Go to the position with the same allele in P1.
     d. Add this allele to the cycle.
     e. Repeat step b through d until you arrive at the first allele of P1.

2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring

Edge recombination

- Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark with a +
  - e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

- Informal procedure: once edge table is constructed
  1. Pick an initial element, entry, at random and put it in the offspring
  2. Set the variable current element = entry
  3. Remove all references to current element from the table
  4. Examine list for current element:
     a. If there is a common edge, pick that to be next element
     b. Otherwise pick the entry in the list which itself has the shortest list
     c. Ties are split at random
  5. In the case of reaching an empty list:
     a. a new element is chosen at random

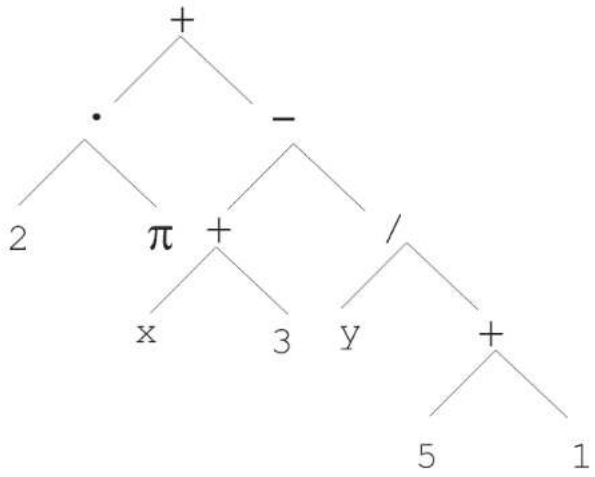
Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Choices	Element selected	Reason	Partial result
All	1	Random	[1]
2,5,4,9	5	Shortest list	[1 5]
4,6	6	Common edge	[1 5 6]
2,7	2	Random choice (both have two items in list)	[1 5 6 2]
3,8	8	Shortest list	[1 5 6 2 8]
7,9	7	Common edge	[1 5 6 2 8 7]
3	3	Only item in list	[1 5 6 2 8 7 3]
4,9	9	Random choice	[1 5 6 2 8 7 3 9]
4	4	Last element	[1 5 6 2 8 7 3 9 4]

Tree representation

trees are universal form:

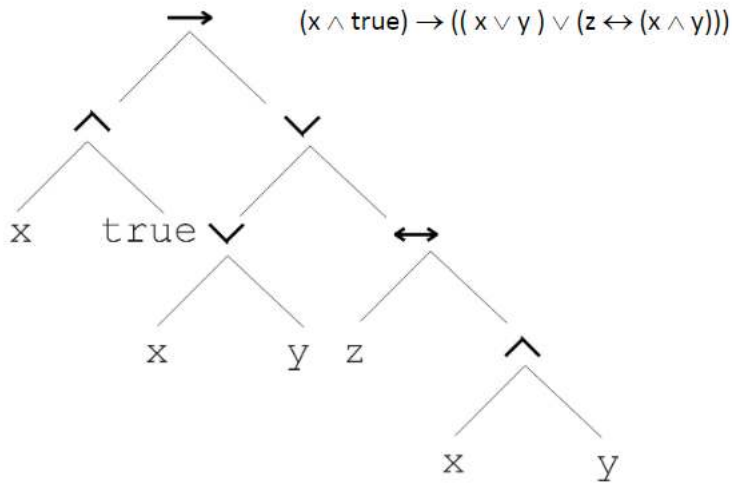
- Arithmetic formula: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$

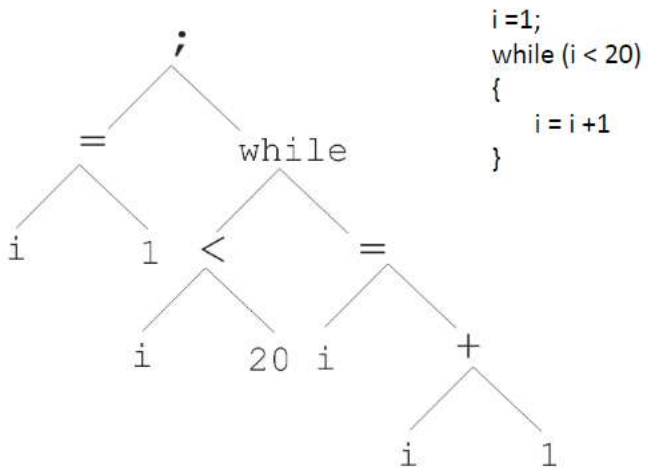


- Logical formula: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

- Program:

```
i =1;
while (i < 20)
{
    i = i +1
}
```






```
i =1;
while (i < 20)
{
    i = i +1
}
```

- In GA, ES, EP chromosomes are linear structures (bit strings, integer strings, real-valued vectors, permutations)
- Tree shaped chromosomes are non-linear structures
- In GA, ES, EP the size of the chromosomes is fixed
- Trees in GP may vary in depth and width

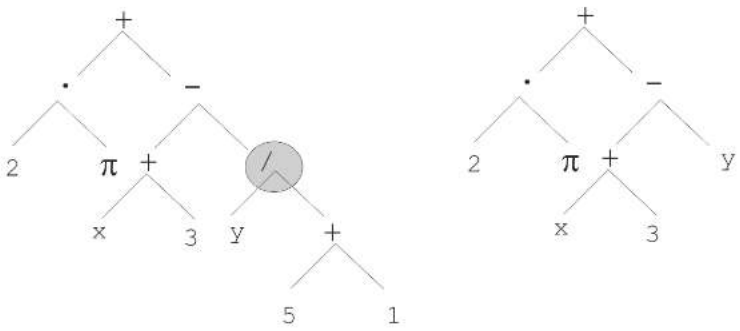
- Symbolic expressions can be defined by:
  - Terminal set T → the ends of the tree
  - Function set F (with the arities of function symbols) e.g. arithmetic operators
- Adopting the following general recursive definition:
  1. Every $t \in T$ is a correct expression
  2. $f(e_1, \ldots, e_n)$ is a correct expression if $f \in F$, arity(f)=n and $e_1, \ldots, e_n$ are correct expressions
  3. There are no other forms of correct expressions

- In general, expressions in GP are not typed

Tree representations: mutation

- Logic is the same

- Most common mutation: replace randomly chosen subtree by randomly generated tree:



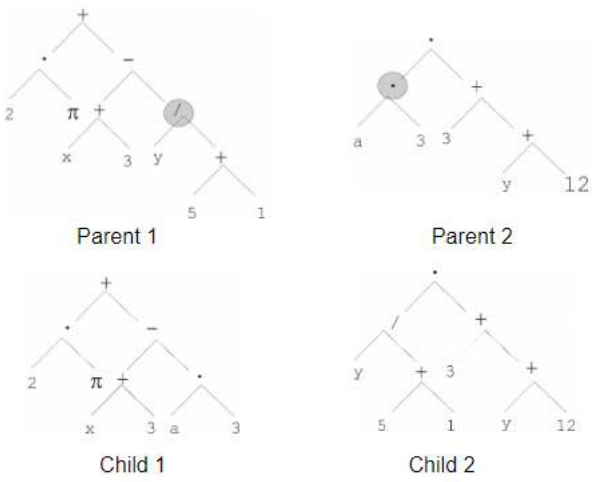
- Mutation has two parameters:
  - Probability p_m to choose mutation
  - Probability to chose an internal point (rather than terminal point) as the root of the subtree to be replaced

- Remarkably p_m is advised to be 0, or very small (like 0.05)
- The size of the child can exceed the size of the parent

Tree representations: recombination

- Same logic holds

- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
    - Probability p_c to choose recombination
    - Probability to choose an internal point within each parent as crossover point
- The size of the offspring can exceed that of the parents



Important points

- Representation is essential when designing an EA
- A few data structures are enough to represent many (all?) problems, e.g., binary, integer, real-valued vectors, trees
- For any given problem there can be more suitable representations. Some are better than others – remember the strong causality principle → Small changes shouldn't cause too much change in the phenotype
- Reproduction operators must fit the representation
- Reproduction operators are stochastic
- Reproduction operators can be distinguished by arity
    - Unary: mutation
    - Binary: recombination, crossover
    - N-ary: multi-parent recombination
- Self-adaptive mutation is a powerful mechanism. But remember to change the sigma first!

Chapter 5: Fitness, selection and population management

- Selection is one of the two fundamental forces in evolutionary systems
- Selection increases quality, decreases diversity

- Related machinery in EC contains:
  - Population management models
  - Selection operators
  - Diversity preserving tricks

Population management

- Two different population management models exist
  - Generational model:
    - Each individual survives for exactly one generation
    - The entire set of parents is replaced by the offspring
  - Steady-state model:
    - A few new individuals (children) are generated per generation
    - A few old members of the population are replaced
    - This method is more 'smooth'
- Generation gap:
  - The proportion of the population replaced
  - Parameter value:
    - =1.0 means generational EA
    - <1.0 means steady state EA

Fitness based competition

- Selection can occur at two places
  - Individuals from current generation to take part in mating (parent selection)
  - Individuals from parents + offspring to go into next generation (survivor selection)
- Selection operators work on whole individuals, hence they are representation independent
- Distinction between selection
  - Operators: define selection probabilities
  - Algorithms: define how probabilities are implemented
  - This distinction is often forgotten of deemed irrelevant

<u>Fitness-proportionate selection (FPS) (selection mechanism)</u>

- Probability for individual i to be selected from mating in a population size of μ with FPS is

$$P_{FPS}(i) = f_i \Big/ \sum_{j=1}^{\mu} f_j$$

- This method has problems including:
  1. Premature convergence: highly fit members can rapidly take over, if the rest of the population is much less fit → population members become (nearly) identical
     We can get stuck in a local optima because of this
  2. Loss of selection pressure: at the end of runs when fitness values are similar, EA can degrade to random search
  3. Highly sensitive to function transposition

FPS and function transposition:
The selection probability for fitness function is easily calculated (0.1, 0.4, 0.5)
If we transpose this and push this landscape up, and for example add 10, and calculate the selection probability for those values, we have different values
Pushing these values up more, gives almost the same selection probabilities for all individuals

individual	Fitness for f	Sel. Prob. for f	Fitness for f+10	Sel. Prob. for f + 10	Fitness for f+100	Sel. Prob. for f + 100
A	1	**0.1**	11	**0.275**	101	**0.326**
B	4	**0.4**	14	**0.35**	104	**0.335**
C	5	**0.5**	15	**0.375**	105	**0.339**
SUM	10	**1**	40	**1**	310	**1**

Parent selection: scaling

There are some methods to accompany fitness proportion selection (FPS) so you don't have the problems stated above:

- Windowing: $f'(i) = f(i) - \beta$
  Where β is the worst fitness in the last n generations (n=0 → in this generation)

- Sigma scaling:

$$f'(i) = \max(f(i) - (\bar{f} - c \bullet \sigma_f), 0)$$

  f- = the populations avg fitness, σ_f = st. dev., c = constant, usually 2
  So if the individual has a fitness lower than the average fitness, then it can become 0
  Tries to 'flatten' selection pressure over the course of evolution

Rank-based scaling

- Attempt to remove problems of FPS by basing selection probabilities on relative rather than absolute fitness
  - So not based on absolute values e.g. fitness of 10, 13, 15, 9 ect.
- Idea: rank population according to fitness and base selection probabilities on rank
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Population size =	μ
Rank of best	μ - 1
...	
Rank of worst	0

Rank based selection: linear ranking

- Selection probability of an individual i is given by this formula

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

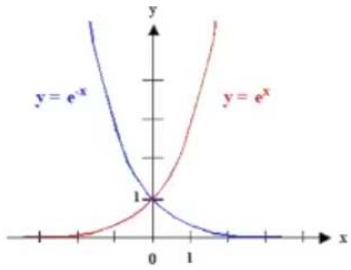
- Parameterized by factor s: 1 < s <= 2
  - S measures advantage of best individual

Individual	Fitness	Rank	P_{selFP}	P_{selLR} $(s=2)$	P_{selLR} $(s=1.5)$
A	1	0	0.1	0	0.167
B	4	1	0.4	0.33	0.33
C	5	2	0.5	0.67	0.5
Sum	10		1.0	1.0	1.0

Rank based selection: exponential ranking

- Linear ranking is limited in selection pressure
- Exponential ranking is much more biased into pushing quality
- Normalize constant factor c according to population size

$$P_{\exp-rank}(i) = \frac{1-e^{-i}}{c}$$

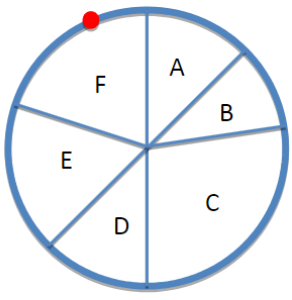


Sampling algorithms

There is a difference between selection algorithms and selection operators.

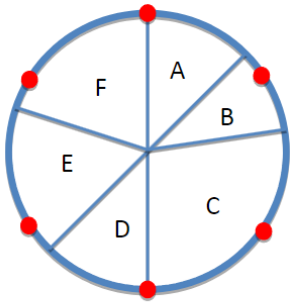
See the two circle divided by the proportion of fitness:

Roulette wheel alg.



6 x

Stochastic universal sampling alg.



1 x

On the left hand side, we see one ball in the roulette wheel and we have to spin the wheel 6 times
On the right, we see 6 balls evenly spaced, and only turn the wheel once.

Tournament selection

- The methods above rely on global population statics
  - So to calculate the ranks and probabilities you have to know the fitness of every individual in the population

    - Could be a bottleneck especially on structured or very large populations
    - Relies on external fitness function which might not exist: e.g. evolving game playing strategies
- Idea for using only local fitness information:
    - Pick k members at random, then select the best of these
    - Repeat to select more individuals

- Tournament selection is very good because it allows you to scale the selection pressure
- Probability of selecting individual i will depend on
    - Rank of i
    - Size of sample k
        - Higher k increases selection pressure
- There are variations:
    - Whether contestants are picked with replacement
        - Picking without replacement increase selection pressure
    - Whether fittest contestant always wins (deterministic) or this happens with probability p

Parent selection: uniform

- This method doesn't use the fitness value
- Gives equal probability to each population member:

$$P_{uniform}(i) = \frac{1}{\mu}$$

- Parents are selected by uniform random distribution whenever an operator needs one or some individual(s)
- Uniform parent selection is unbiased → every individual has the same probability to be selected
- When working with extremely large populations, over-selection can be used

Survivor selection

- From parents and offspring you have to choose the next population
- Select μ individuals to form the next generation from μ parents and λ offspring
- Parent selection mechanisms can also be used for selecting survivors
- Survivor selection can be divided into two approaches:
    - Fitness-based selection
    - Age-based selection
        - Does not take fitness into account
        - In SSEA this can be implemented as 'delate random' (not recommended!) or as fifo (a.k.a. delete the oldest)

Fitness based survivor selection

- Elitism
    - Always keep at least one copy of the fittest solution so far → so you never forget your best individual
    - Widely used in both population models (generational, steady state)
    - Can keep more than one 'champion', e.g. the best N individuals or the best x%
- Delete worst (a.k.a. GENITOR)

    - From whitleys original steady-state algorithm (he also used linear ranking for parent selection)
    - Rapid takeover: use with large populations or 'no duplicates' policy
- Round-robin tournament
    - P is the set of μ parents, O the set of λ offspring (so 2 sets)
    - Pairwise competitions in round-robin format:
        - Each individual (x) from the unified set P and O is compared with q other random chosen individuals (so one pair x compared to all q other pairs)
        - For each comparison, a 'win' is assigned to x if it is better than its opponent
        - The μ solutions with the greatest number of wins are selected to be parents of the next generation
    - Parameter q allows tuning selection pressure (typically q=10)

- (μ, λ)-selection a.k.a. comma strategy
    - Based on the set of children only (λ > μ)
    - Choose best μ survivors out of λ children
- (μ+ λ)-selection a.k.a. plus strategy
    - Based on the set of parents and children
    - Choose best μ out of μ+λ

- Often (μ, λ)-selection is preferred because it can 'forget' and hence
    - It can help you leave local optima
    - It is better in following a moving target/optima
    - Using the + strategy is not a good combination with sample permutation? approach because bad σ values can survive in <x, σ> too long if their host x is very fit
- Using (μ, λ)-selection you need to make a decision:
    - How much more children do you make than you really need
    - Classic ratio was 1:7, so if population size is 10 you generate 70 children
    - Lately the ratio is going down: 1:3

Selection pressure

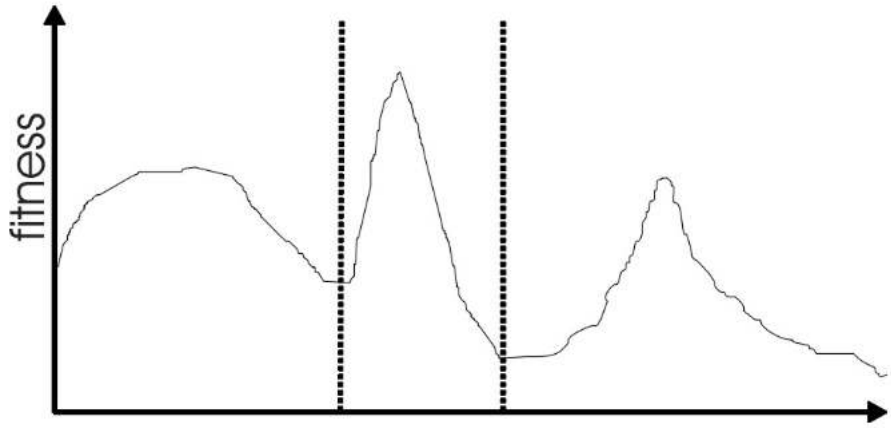
- Take over time τ^* is a measure to quantify the selection pressure
- The number of generations it takes until the selection makes the population completely uniform → meaning having a population without using mutation and recombination but only selection
- You check how long it takes until you lose all diversity

$$\tau^* = \frac{\ln \lambda}{\ln(\lambda / \mu)}$$

- Theoretic studies showed that these values can very differ
    - An evolution strategy with μ = 15 and λ=100: τ^*≈2
    - A GA with and proportional selection τ^*=460

Multimodality

- The number of hills on a fitness landscape
- A uniform landscape has only one hill to climb
- Of course you want to climb the highest or one of the highest



- Most problems have more than one local optimal solution

Multimodality: genetic drift

- A typical behaviour of EA on multimodel landscapes is that sooner or later it converges around one optimum
- This is not always desirable because we might want to identify several possible peaks/optima
- So you need to keep your diversity higher → losing diversity in an algorithm is always bad

Approaches to preserve diversity

Explicit vs. implicit

- Explicit approaches
  - Always based on competition
  - Make similar individuals compete for resources (fitness)
  - Make similar individuals compete with each other for survival
- Implicit approaches
  - Try to segment a population
  - Impose an equivalent of geographical separation
  - Impose an equivalent of speciation

Explicit approaches to preserve diversity: Fitness sharing

- Restricts individuals within a given niche by 'sharing' their fitness, so as to allocate individuals to niches in proportion to the niche fitness
- Need to set the size of the niche σ_share by distance in either genotype or phenotype space (thus σ_share is NOT the number of individuals in that niche
- Run EA as normal but after each generation set f(i) to f'(i)

$$f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i,j))} \qquad sh(d) = \begin{cases} 1 - d/\sigma & d \le \sigma \\ 0 & otherwise \end{cases}$$

- Fitness of f is replaced by f'(i) by dividing it by a sum
- Sum: share deviated to the distance
- Note, if we use sh(d)=1 instead of sh(d)=1-d/ σ then the sum that reduces the fitness would simply count the number of neighbors → i.e. individuals closer than σ (a.k.a. σ_share)

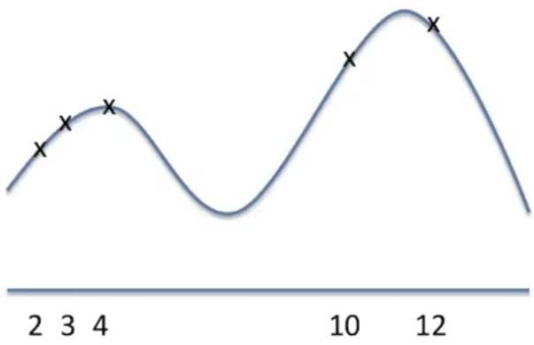
- So being alone in your neighborhood brings you NO reduction! This is an advantage
- Using 1-d/ σ_share instead of 1 implies that we count distant neighbors less than close neighbors

Example

$$f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i,j))} \qquad sh(d) = \begin{cases} 1 - d/\sigma & d \le \sigma \\ 0 & otherwise \end{cases}$$

Let σ=3 and

- f(2)=2
- f(3)=2,5
- f(4)=3
- f(10)=4
- f(12)=5



Indiv. 2 has 2 neighbours

Indiv. 10 has 1 neighbour

f'(2) = 2/(1+1/3+2/3)=1

f'(10) = 4/(1+1/3)=3

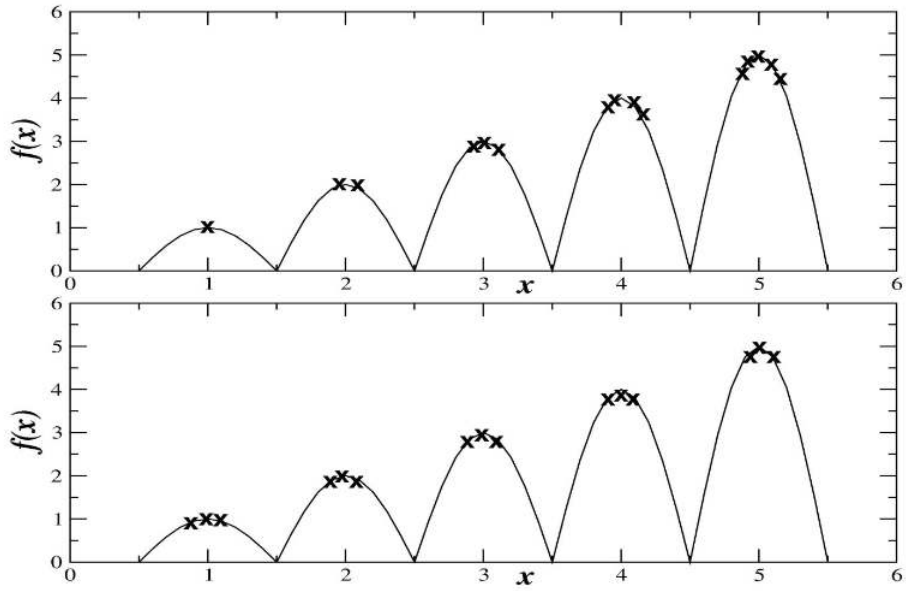
There are 5 points (2,3,4,10 and 12) and 5 fitness values
If we have σ = 3 → individual 2 has 2 neighbors (3 and 4 are within a distance of 3)

We can also see that the reduction of individual 2 is more serious than of individual 10

Explicit approaches to preserve diversity: Crowding

- Attempts to distribute individuals evenly amongst niches
- Assumption: offspring will be close to parents
- Uses a distance metric in either phenotype or genotype space
- Take 2 parents and their 2 offspring
    - Makes tournament that one of the parents is competing with one of the children and the other parent with the other child
    - Set up parent vs. child tournaments (the pairing) such that the inter-tournament distances are minimal, such that:
      d(p1,o1) + d(p2,o2) < d(p1,o2) + d(p2,o1)
    - And let o_1 compete with p_1 and o_2 with p_2

Comparison crowding vs. fitness sharing



For fitness sharing: the higher the hills the more individuals, because there is more to share
For crowding: an opposite effect occurs

Implicit approaches to preserve diversity: automatic speciation

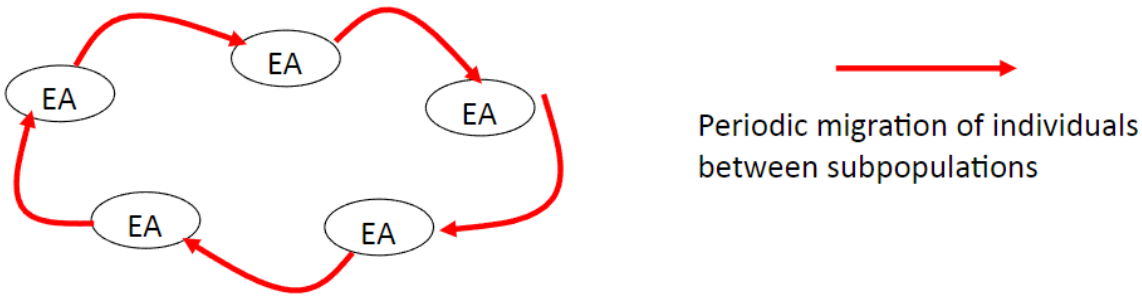
- Restrict mating to genotypically/phenotypically similar individuals

Or

- Restrict mating to individuals that have the same (or very similar) tag, where
  - A tag is an extra bit (or bits) in the genotype that
  - Is initialized randomly and
  - Is subject to recombination and mutation

Implicit approaches to preserve diversity: island model parallel EAs

- Run multiple population in parallel



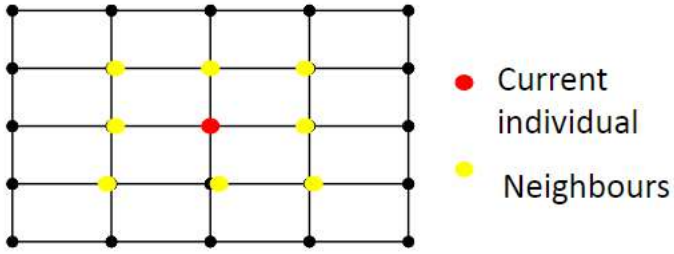
- After a (usually fixed) number of generations (called epoch) exchange individuals with neighbors
- Repeat until stopping criteria met
- Inspired by parallel/clustered systems

Parameters:

- Very hard to optimize
- How often to exchange individuals?
  - Too quick and all sub-populations converge to same solution

    - Too slow and waste time
    - Most authors use a range of 25-150 generations
    - Can do it adaptively (stop each population when no improvement for X generations)
- How many individuals, which individuals to exchange
    - Usually only around 2-5 individuals, but depends on the population size
    - Two options: Copied (so it is kept on its own island) vs moved (removed from own island)
    - Some results show that:
        - Better to exchange randomly selected individuals than best individuals
        - 'multi-culti' is even better → exchange most different ones
- Variation operators can differ per sub-population/island

Implicit approaches to preserve diversity: Cellular EAs



- Also uses spatial component
- Consider each individual to exist on a grid (usually rectangular toroid)
- Selection (hence recombination) and replacement happen ONLY in the neighborhood a.k.a. deme → so those who are just one step away. The yellow dots
- Leads to different parts of grid searching different parts of space, good solutions diffuse across grid over a number of generations

- Equivalent of 1 generation is:
    - Pick individual in pop at random
    - Pick one of its neighbors using e.g. roulette wheel or some other method
    - Apply crossover to produce 1 child, mutate the child to get final offspring
    - Replace individual if offspring is fitter
    - Circle through whole population until done

Different spaces in EAs

- Genotype space (always)
    - Set of representable/ possible solutions
- Phenotype space (always; it may be = genotype space)
    - The object coded by the genotype
    - Neighbourhood structure may bear little relation with genotype space
- Fitness space/quality of individual (if quantifiable fitness is given)
    - The space of fitness values: (multi-dimensional) real numbers
- Algorithmic space (if a spatially structured EA is used)
    - The structure of the population
    - Independent of genotype and phenotype space
    - Akin to the geographical space on which life on earth has evolved

- Distance in these spaces may (and will) differ

Important points

- Selection increases quality, decreases diversity
- Selection can occur at two places in the evolutionary loop
    - parent selection
    - survivor selection
- Most selection operators can be used for both purposes
- Selection pressure is an important feature of EAs
    - Strong selection → (too) fast convergence, possibly local optimum
    - Weak selection → slow progress, almost random walk
- Keeping up diversity is important
    - There are specific mechanisms for this in EAs