5 Evolutionary Algorithms

Recap of EC Metaphor

- a population of individuals exists in an environment with limited resources
- competition for those resources causes selection of those fitter individuals that are better adapted to the environment
- these individuals act as seeds for the generation of new individuals through recombination and mutation
- the new individuals have their fitness evaluated and compete - possibly also with parents - for survival
- over time, natural selection causes a rise in the fitness of the population

Recap of EC Metaphor

- evolutionary algorithms are stochastic and population-based
- crossover and mutation are the variance operators
- they create diversity, facilitating the occurrence of novel candidate solutions
- selection reduces diversity by weeding out the lowest quality solutions
- so acts as a force pushing quality

Evolutionary Algorithms

subtopics:

- general scheme of an EA
- main EA components:
  - representation / evaluation / population
  - parent selection and survivor selection
  - recombination (crossover) and mutation
- example: the eight-queens problem
- typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

General Scheme of an EA

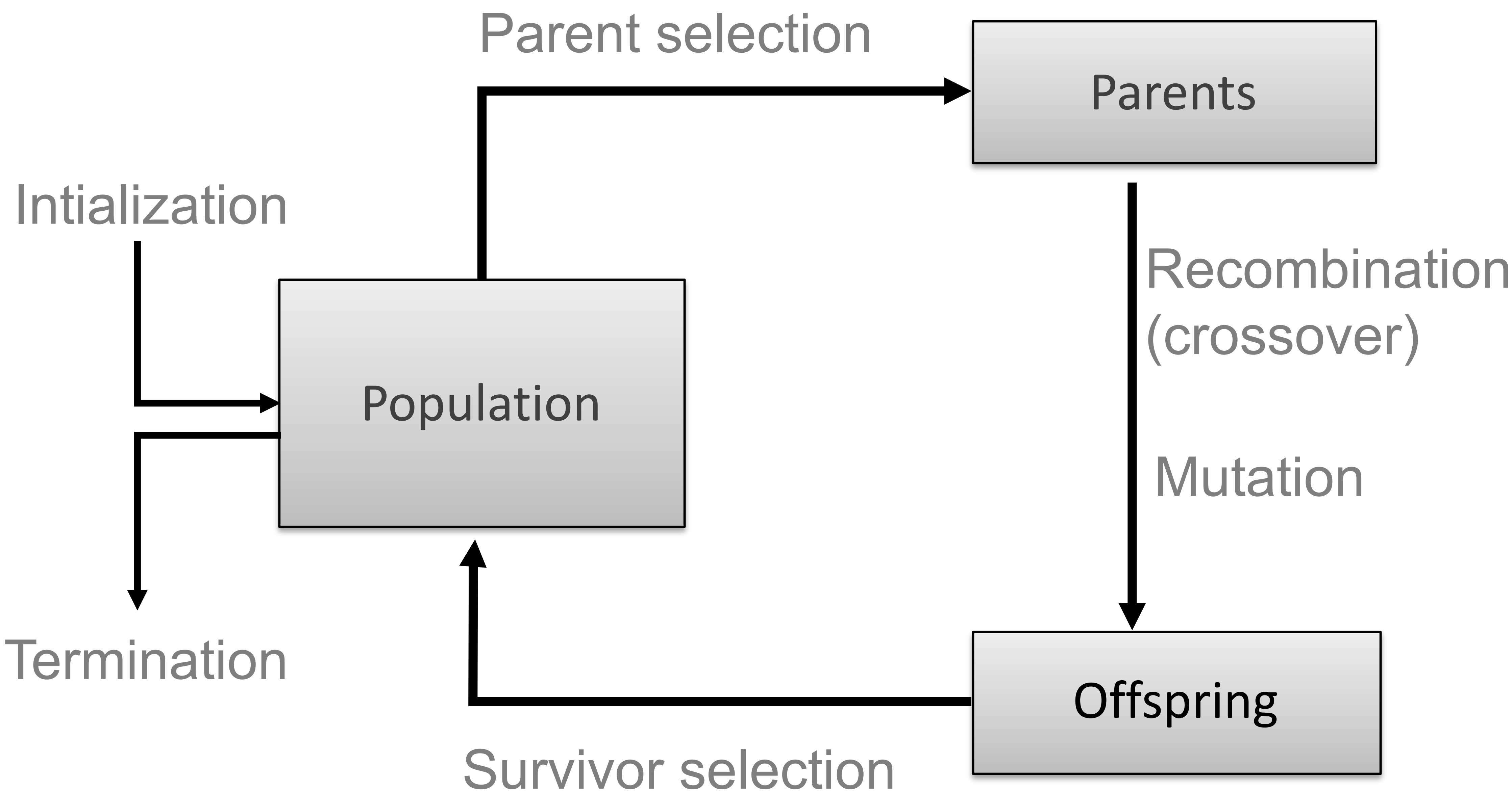


figure 3.2, Introduction to Evolutionary Computation

General Scheme of an EA: Pseudocode

```
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
```

figure 3.1, Introduction to Evolutionary Computation

General Scheme of an EA: Common Model of Evolutionary Processes

- population of individuals
- individuals have a fitness level
- variation operators: crossover, mutation
- selection towards higher fitness
  - “survival of the fittest” &
  - “mating of the fittest”
- *Neo Darwinism:*

evolutionary progress towards higher life forms

=

optimization according to some fitness-criterion

(optimization on a fitness landscape)

General Scheme of an EA: Two Pillars of Evolution

- two competing forces:

increasing population diversity through genetic operators:
- mutation
- recombination

push towards novelty

decreasing population diversity through selection:
- of parents
- of survivors

push towards quality

rise in fitness of population

Main EA Components: Representation

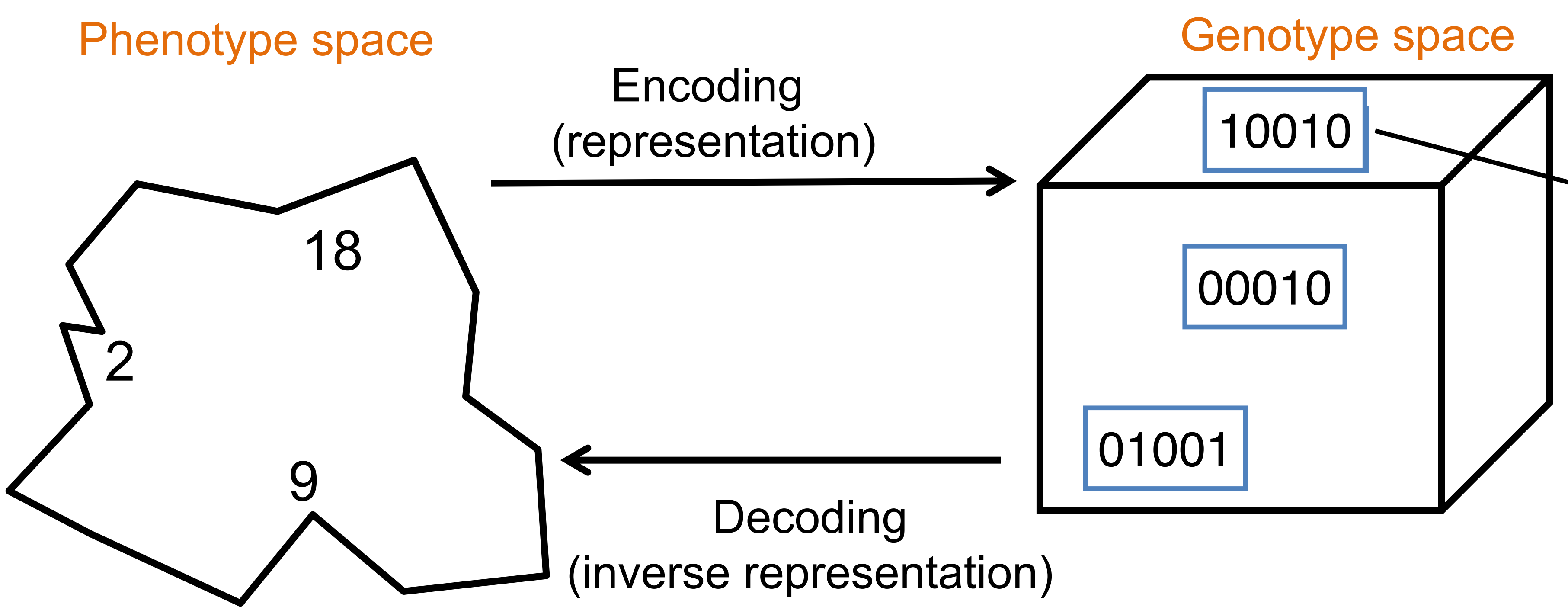
- provides an encoding for candidate solutions that can be manipulated by variation operators
- leads to two levels of existence
  - phenotype: object in original problem context
    - the ‘outside’
  - genotype: code to denote that object
    - the ‘inside’
    - (chromosome, ‘digital DNA’)

Main EA Components: Representation

- implies two mappings:
  - encoding : phenotype → genotype
    - not necessarily one to one
  - decoding : genotype → phenotype
    - must be one to one
- genotypes contain genes: 位置 + 值
  - in (usually fixed) positions called loci
  - have a value (allele)

Main EA Components: Representation

- example: represent integer values by their binary code



- each binary string is a genotype
- each bit is a gene
- the value of each bit is an allele

- to be able to find the global optimum, every feasible solution must be represented in the genotype space

Main EA Components: Evaluation Fitness Function

- represents the task to solve, the requirements to adapt to
  - can be seen as ‘the environment’
- enables selection by providing basis for comparison
- for example: some phenotypic traits are advantageous:
  - big ears cool better
  - so these traits are rewarded by more offspring
  - who will most likely carry the same trait

Main EA Components: Evaluation Fitness Function

- also known as 'quality function' or 'objective function'
- assigns a single real-valued fitness to each phenotype which forms the basis for selection
- so the more different values observed the better
- typically we refer to maximising fitness
- but some problems may be best posed as minimisation problems
    - and conversion is trivial

Main EA Components: Population

- holds the candidate solutions of the problem as individuals
  - as genotypes
- because a population is a multiset of individuals, repetitions are possible
- population is the basic unit of evolution
  - we think of the population is evolving, not the individuals
- selection operators act on population level
- variation operators act on individual level

Main EA Components: Population

- some sophisticated EAs also assert a spatial structure on the population, so that individuals have a neighbourhood
  - this affects which individuals are able to reproduce with each other
  - and can be used in selection when making fitness comparisons
- but generally, selection operators usually take whole population into account
- so reproductive probabilities are relative to the whole current generation
- the diversity of a population can be measured by the number of different fitnesses, phenotypes or genotypes present
  - and note that these may all be different values

Main EA Components: Selection Mechanism

- identifies which individuals will:
    - become parents
    - survive into the next generation
- pushes the population towards higher fitness
- parent selection is usually probabilistic
    - high quality solutions more likely to be selected than low quality
    - but not guaranteed
    - even the worst member of the current population usually has a non-zero probability of being selected
- this stochastic nature can aid escape from local optima

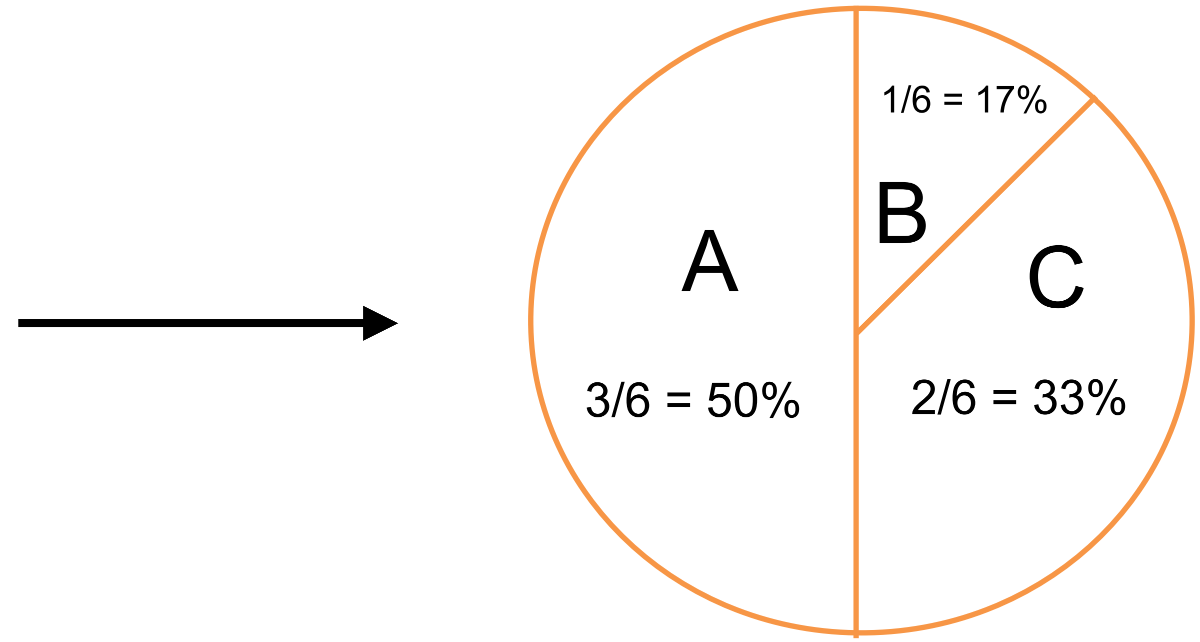
Main EA Components: Selection Mechanism

- example: roulette wheel selection over 3 individuals, A, B and C:

fitness(A) = 3

fitness(B) = 1

fitness(C) = 2



- in principle, any selection mechanism can be used for parent selection as well as for survivor selection

Main EA Components: Selection Mechanism

- (also known as replacement)
- determining which individuals survive into the next generation
- most EAs use a fixed population size, so need a way of going from (parents + offspring) to next generation
- survivor selection is often deterministic, such as:
    - fitness based: 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 (elitism)
- elitism example: ensuring that top 10% of current generation survive to next generation

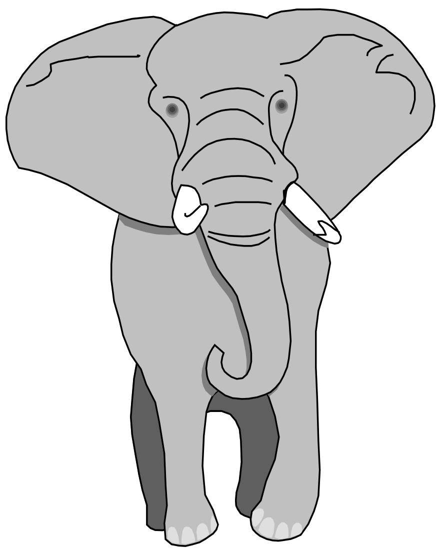
Main EA Components: Variation Operators

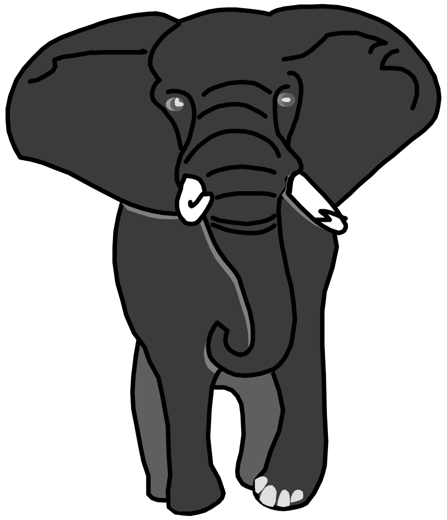
- these generate new candidate solutions
- usually divided into two types according to the number of inputs, or ‘arity’
  - arity 1 : mutation operators
  - arity >1 : recombination operators
  - arity = 2 typically called crossover
  - arity > 2 is possible, but is seldom used in EC
- there has been much debate about relative importance of recombination and mutation
- nowadays most EAs use both
- variation operators must match the given representation

Main EA Components: Mutation

- causes small, random variance
- acts on one genotype and delivers another
- element of randomness is essential and differentiates it from other unary heuristic operators
- the importance ascribed to mutation depends on the representation being used and and historical dialect:
    - binary GAs – background operator responsible for preserving and introducing diversity
    - EP for FSM's / continuous variables – only search operator
    - GP – hardly used
- may guarantee connectedness of search space and hence convergence proofs

Main EA Components: Mutation

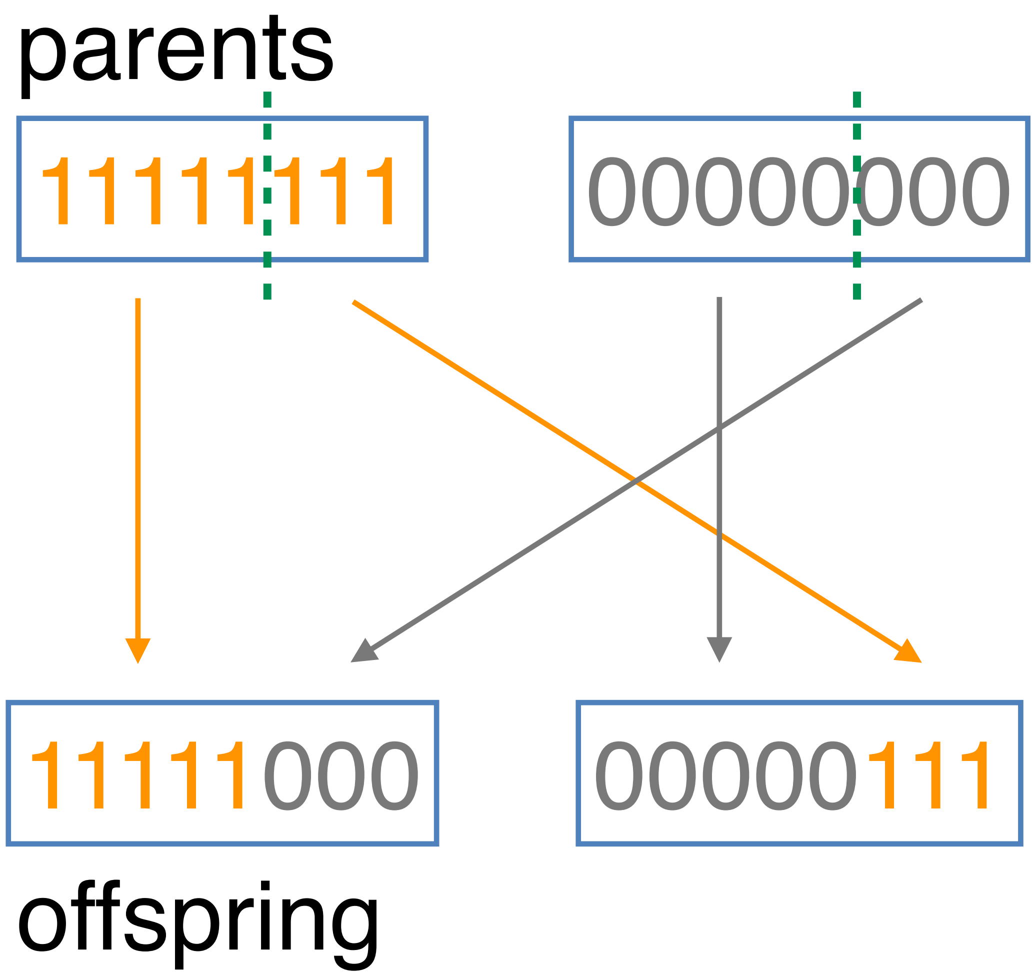
before: 10011101 → 

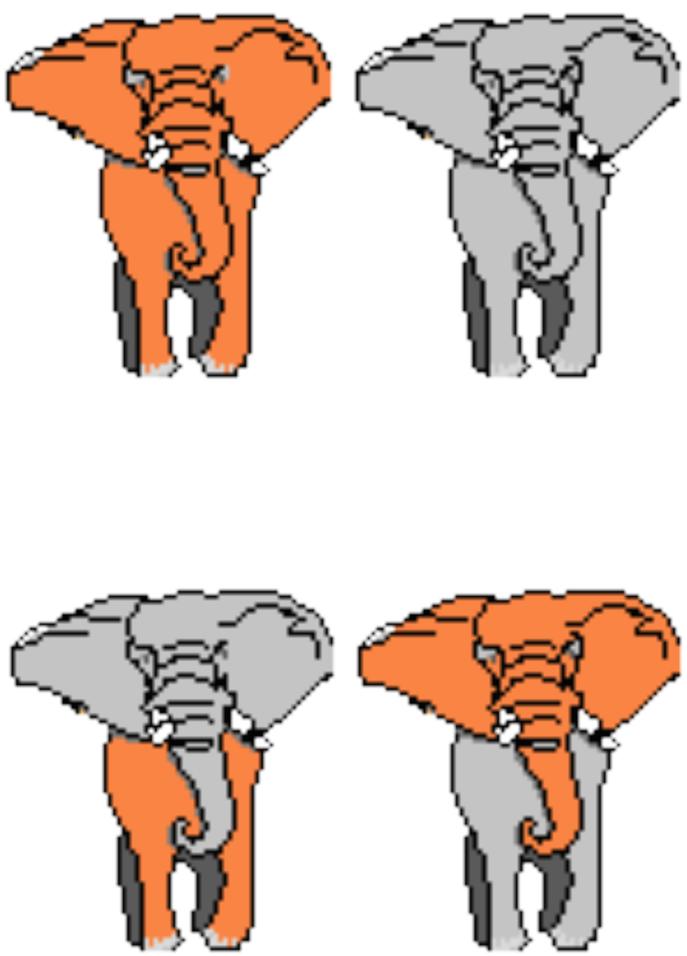
after: 10010101 → 

Main EA Components: Recombination

- merges information from parents into offspring
- choice of what information to merge is stochastic
- most offspring may be worse than the parents, or the same
- but the hope is that some offspring are better by combining elements of genotypes that lead to good traits
- principle has been used for millennia by breeders of plants and livestock

Main EA Components: Recombination





hence: *'crossover'*

Main EA Components: Initialisation & Termination

- initialisation usually done randomly
  - need to ensure even spread and mixture of possible allele values
  - can include existing solutions
  - can use problem-specific heuristics to "seed" the population
- termination condition checked every generation to determine if have reached:
  - a specified fitness level
  - a maximum allowed number of generations
  - a minimum diversity level
  - a specified number of generations without any fitness improvement

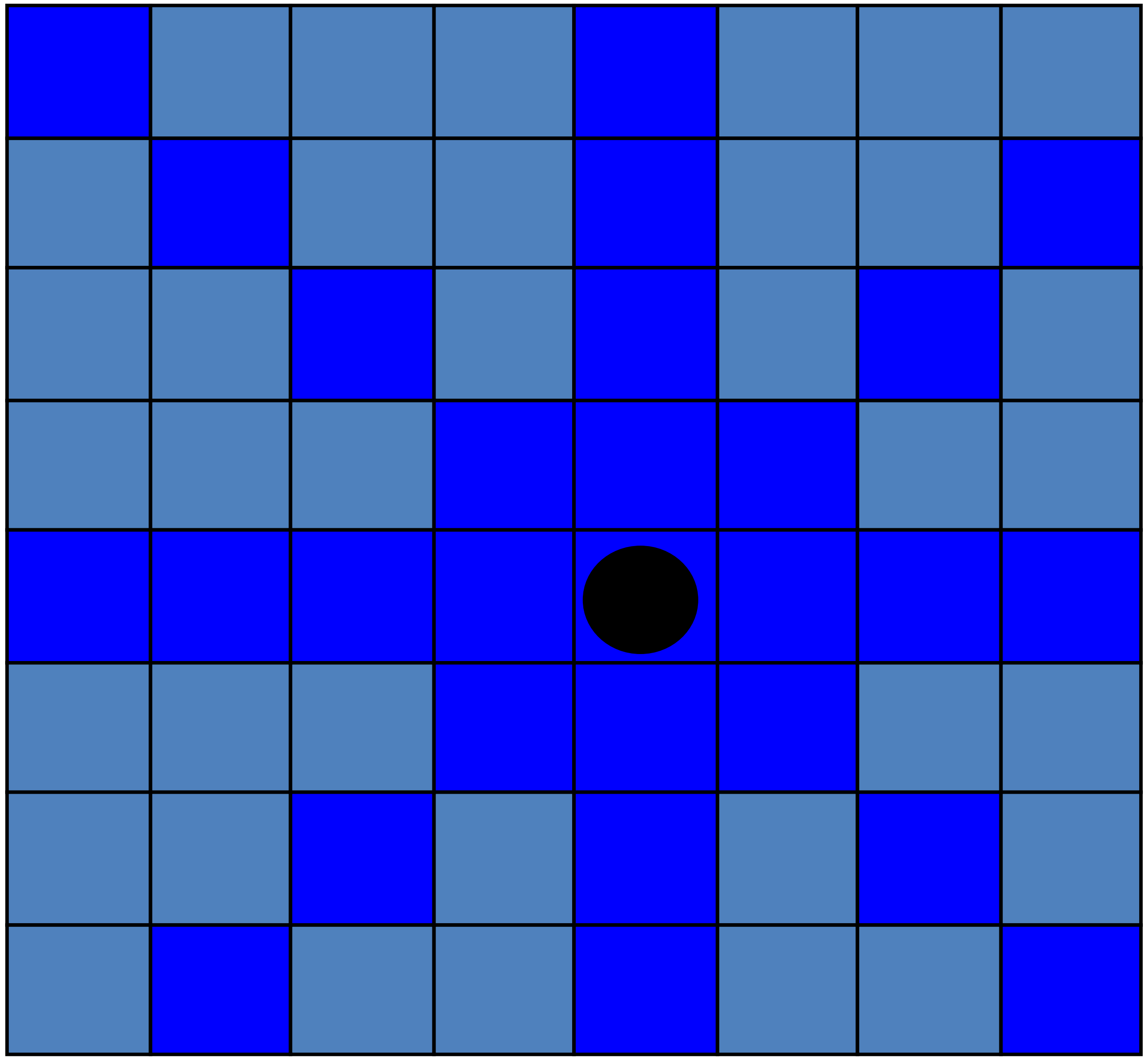
Main EA Components: Different Types of EAs

- historically different flavours of EAs have been associated with different data types to represent solutions
    - binary strings for Genetic Algorithms
    - real-valued vectors for Evolution Strategies
    - finite state machines for Evolutionary Programming
    - LISP trees for Genetic Programming
- these differences are largely irrelevant, so the best strategy is to:
    - choose a representation to suit the problem
    - choose variation operators to suit the representation
- note that selection operators only use fitness and so are independent of the representation

Summary of Roles

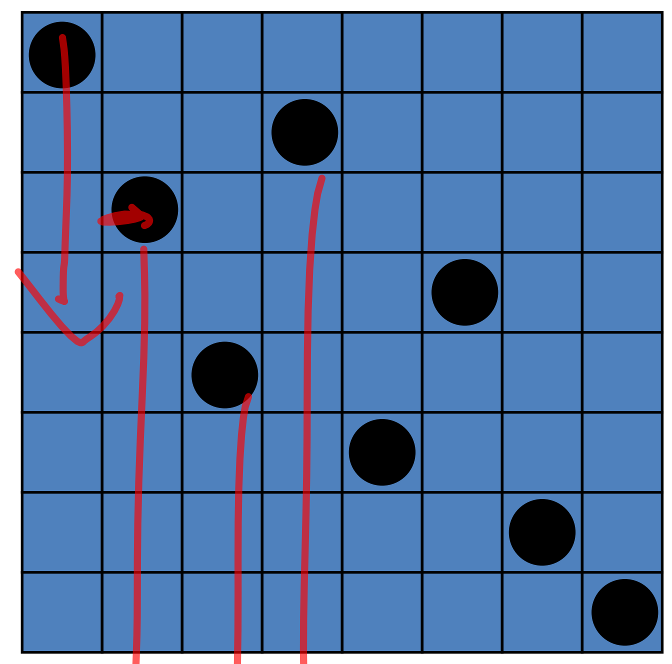
representation	provides an encoding for candidate solutions that can be manipulated by variation operators
fitness function	represents the task to solve, the requirements to adapt to; the environment
population	holds the candidate solutions of the problem as individuals; as genotypes
selection mechanism	identifies which individuals will become parents and survive into the next generation
variation operators	generate new candidate solutions
mutation	causes small, random variance; acts on one genotype and delivers another
recombination (crossover)	merges information from parents into offspring
initialisation	done randomly to ensure even spread and mixture of possible allele values
termination criteria	checked every generation to determine if have reached a desired fitness level, or if there's no reason to keep going

Example:
Back to the 8-Queens Problem



place 8 queens on an 8x8 chessboard in such a way that they cannot check each other

The 8-Queens Problem: Representation



Phenotype:
a board configuration

Genotype:
a permutation of
the numbers 1–8

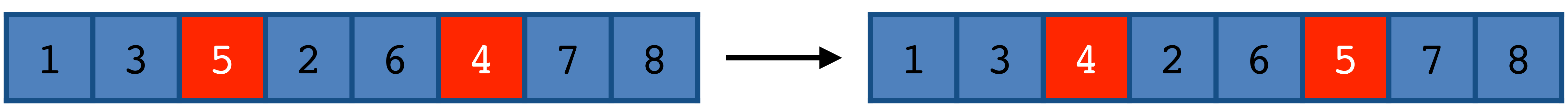
1	3	5	2	6	4	7	8

The 8-Queens Problem: Fitness Evaluation

- penalty of one queen: the number of queens she can check
- penalty of a configuration: the sum of penalties of all queens
- note: penalty is to be minimized
- fitness of a configuration: inverse penalty to be maximized

The 8-Queens Problem: Mutation

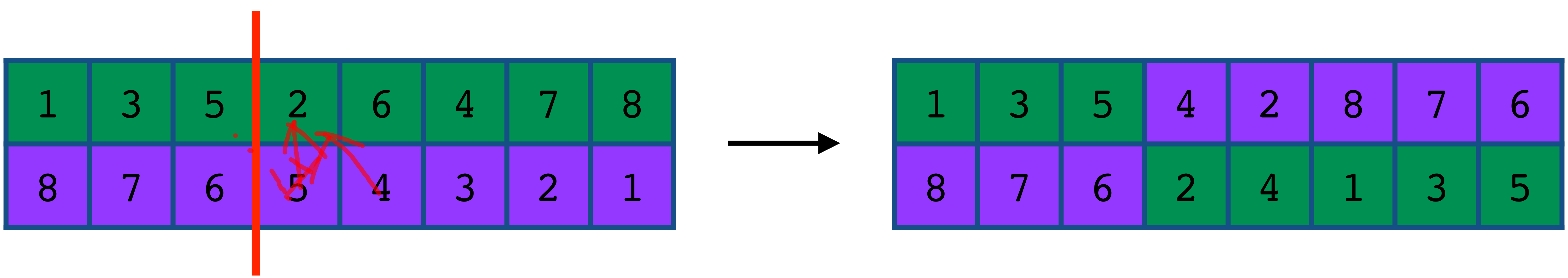
- small variation in one permutation
- such as swapping the values of two randomly chosen positions



The 8-Queens Problem: Recombination

combine two permutations into two new permutations using *"cut-and-crossfill"*:

- choose random crossover point
- copy first parts into children
- create second part by inserting values from other parent:
  - in the order they appear there
  - beginning after crossover point
  - skipping values already in child



The 8-Queens Problem: Selection

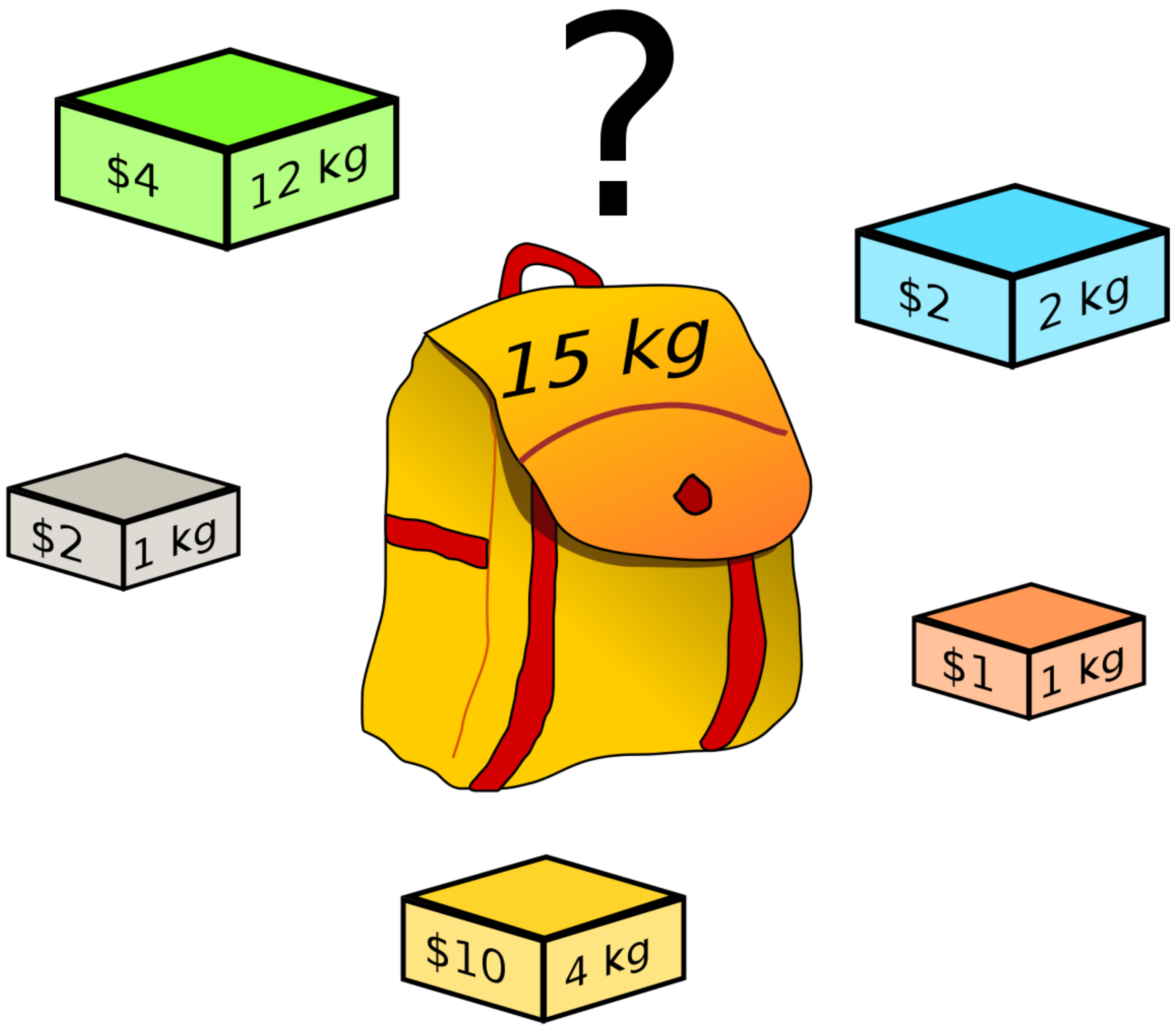
- parent selection:
  - pick 5 parents
  - take best two to undergo crossover
- survivor selection (replacement):
  - merge the existing population with the two new children
  - remove the worst two individuals

The 8-Queens Problem: Summary

Representation	Permutations
recombination	‘cut-and-crossfill’ crossover
recombination probability	100%
mutation	swap two genes
mutation probability	80%
parent selection	best 2 from random group of 5
survival selection	replace worst
population size	100
number of offspring	2
initialisation	random
termnation	find solution, or max 10000 generations

- remember: this is only one possible set of operators and parameters
- there are many other possible sets!

Example:
The 0-1 Knapsack Problem



The 0-1 Knapsack Problem

Given a set of `n` items, each with a weight and a value, determine which items to include in a collection so that the total weight is less than or equal to a given limit `W`, and the total value is as large as possible.

$$\text{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0,1\}.$$

The 0-1 Knapsack Problem

- measure of fitness:
  - sum of values of chosen items
  - to be maximised
  - constrained by the weight staying within the limit W
- representation:
  - genotype: binary string of length n:
    - 0 if item not included
    - 1 if it is
  - genotype space G has size 2^n

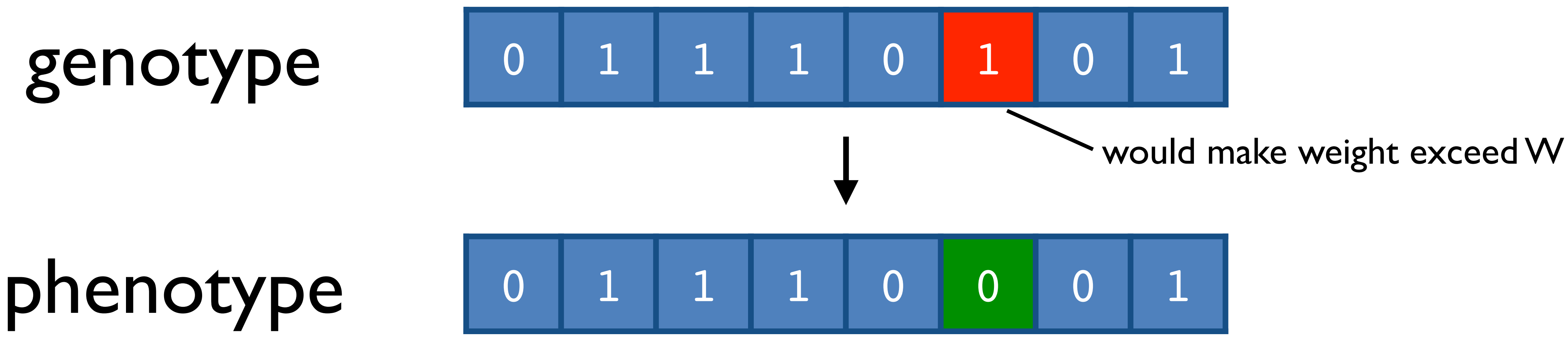
0	0	1	1	0	0	0	1

The 0-1 Knapsack Problem

- phenotype:
    - how to represent this?
- first-cut effort:
    - suppose that the phenotype space P and the genotype space G are identical
    - problem: some genotypes will map to invalid (too heavy) solutions
- second-cut effort:
    - use a decoder function:
    - read from left to right along the binary string, keeping a running tally of the weight of included items
    - when we encounter a value `1`, we first check to see whether including the item would break our weight constraint
    - if not, copy `1` into phenotype, else copy `0` into phenotype
    - in other words, rather than interpreting a value `1` as meaning include this item, we interpret it as meaning include this item IF it does not take us over the weight constraint

The 0-1 Knapsack Problem

- we interpret 1 as meaning include this item IF it does not take us over the weight constraint



- implication:
  - the mapping from genotype to phenotype is many-to-one
  - because for many genotypes the bits at the right end of the string will be irrelevant

The Knapsack Problem: Summary

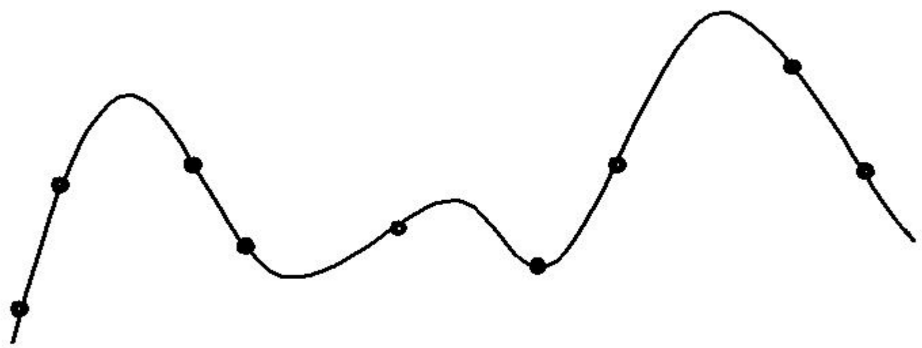
Representation	Permutations
recombination	1 point crossover
recombination probability	70%
mutation	bit flipping (from 0 to 1, or from 1 to 0)
mutation probability	1/n, where n is number of bits in genotype
parent selection	size 2 tournament (pick best from 2)
survival selection	generational - all replaced
population size	100
number of offspring	100
initialisation	random bit values
termnation	when no improvement seen in 25 generations

- again, remember that this is only one possible set of operators and parameters
- many others are possible

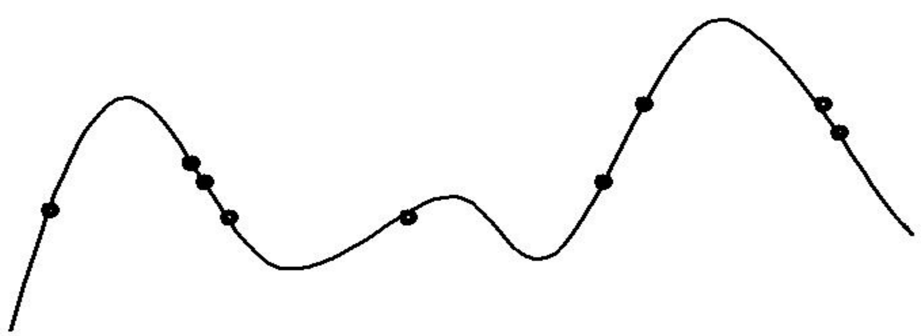
Typical EA Behaviour: Stages of Optimisation

represented on a one-dimensional fitness landscape:

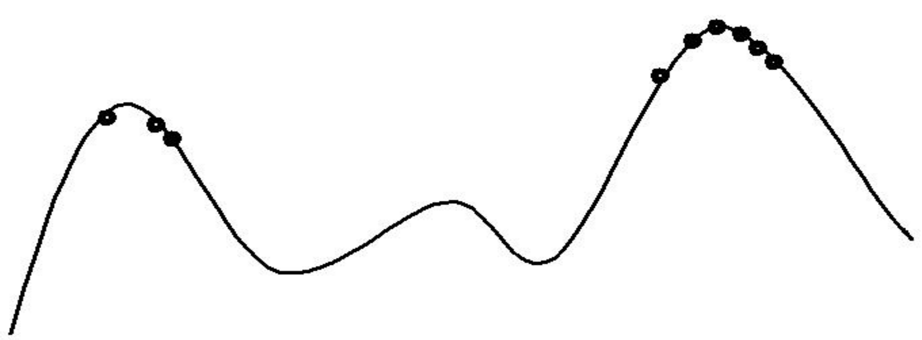
early:
quasi-random population distribution



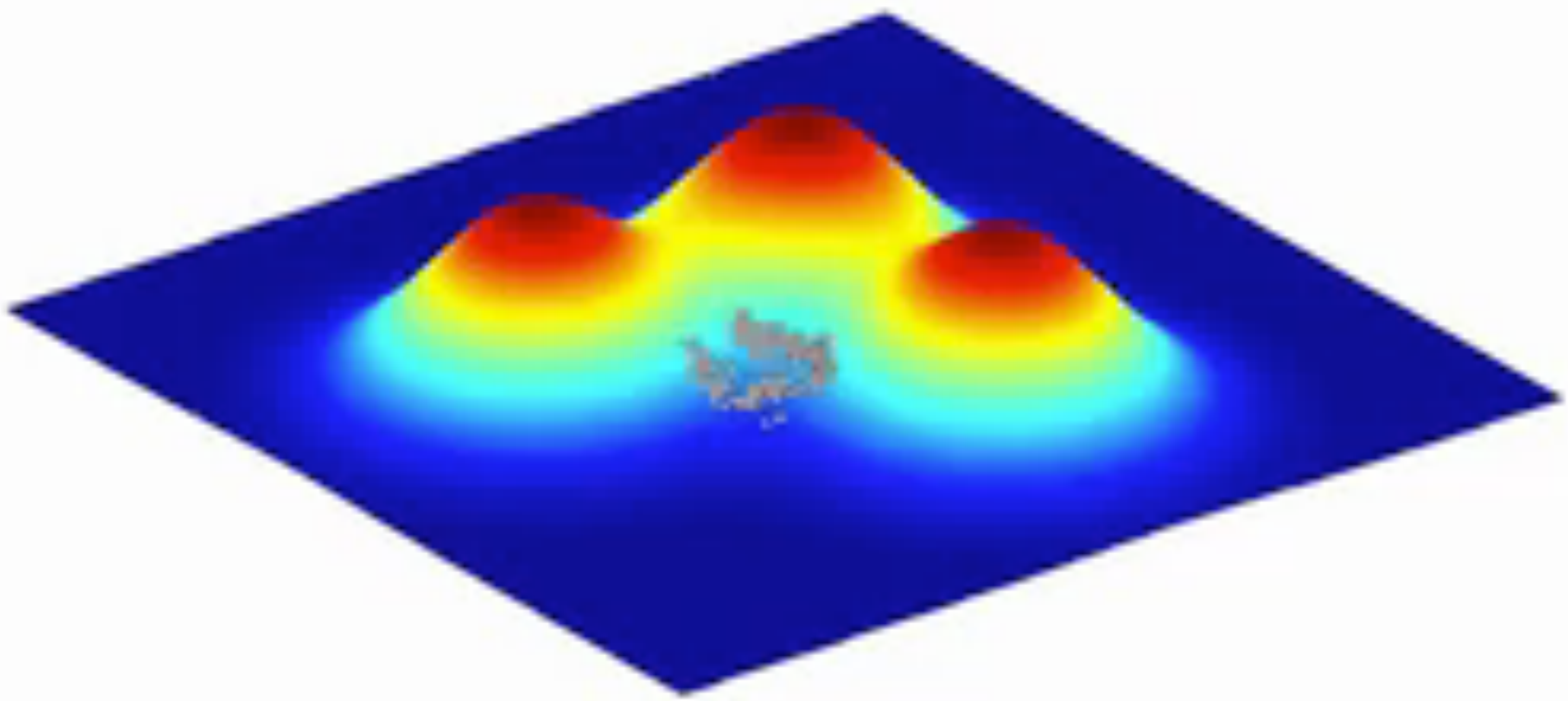
mid:
population arranged on or around hills



late:
population concentrated on high hills



Typical EA Behaviour: Population Evolves to One Peak

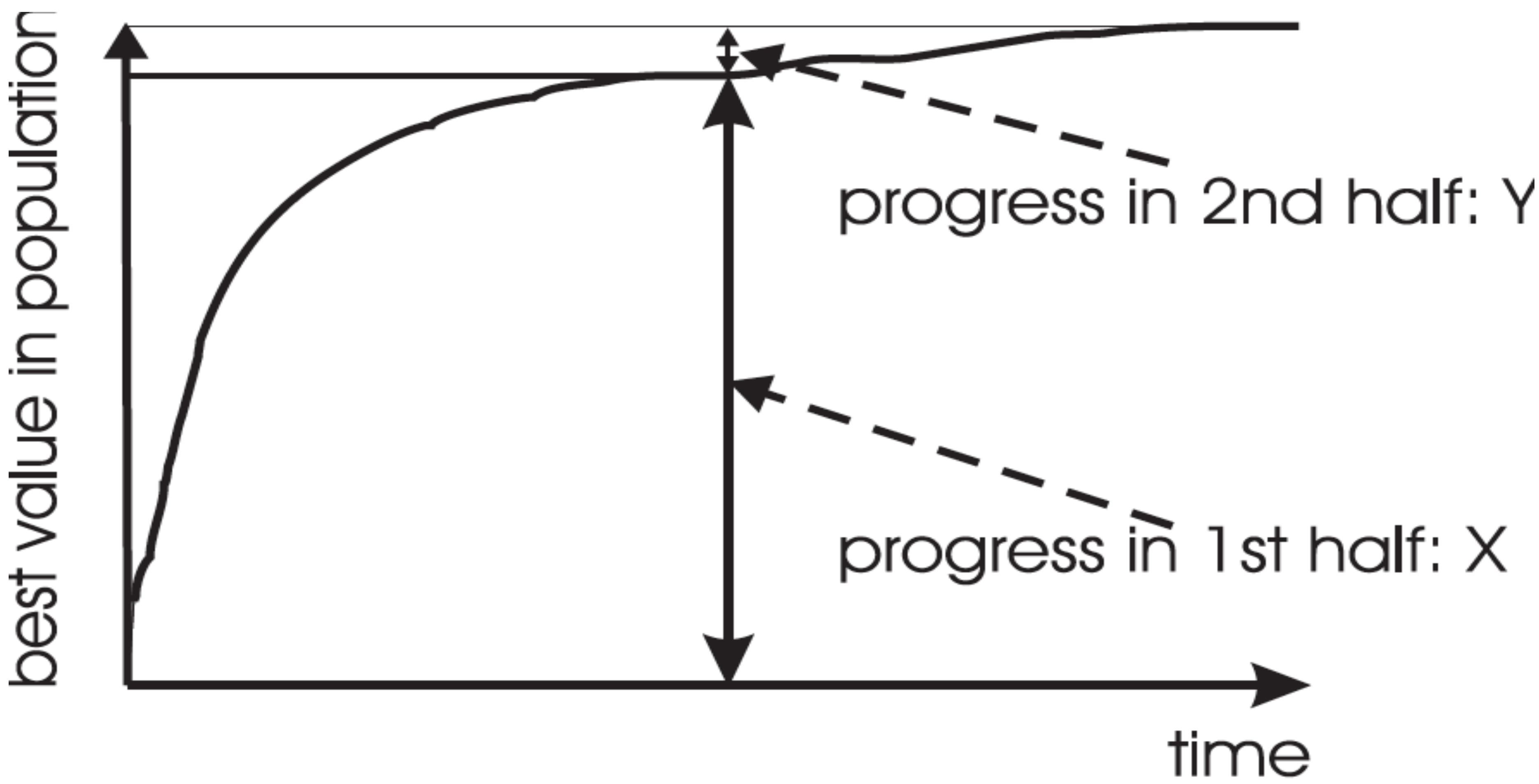


© Bjørn Østman and Randy Olson

Typical EA Behaviour: Progression of Fitness

'anytime behaviour'

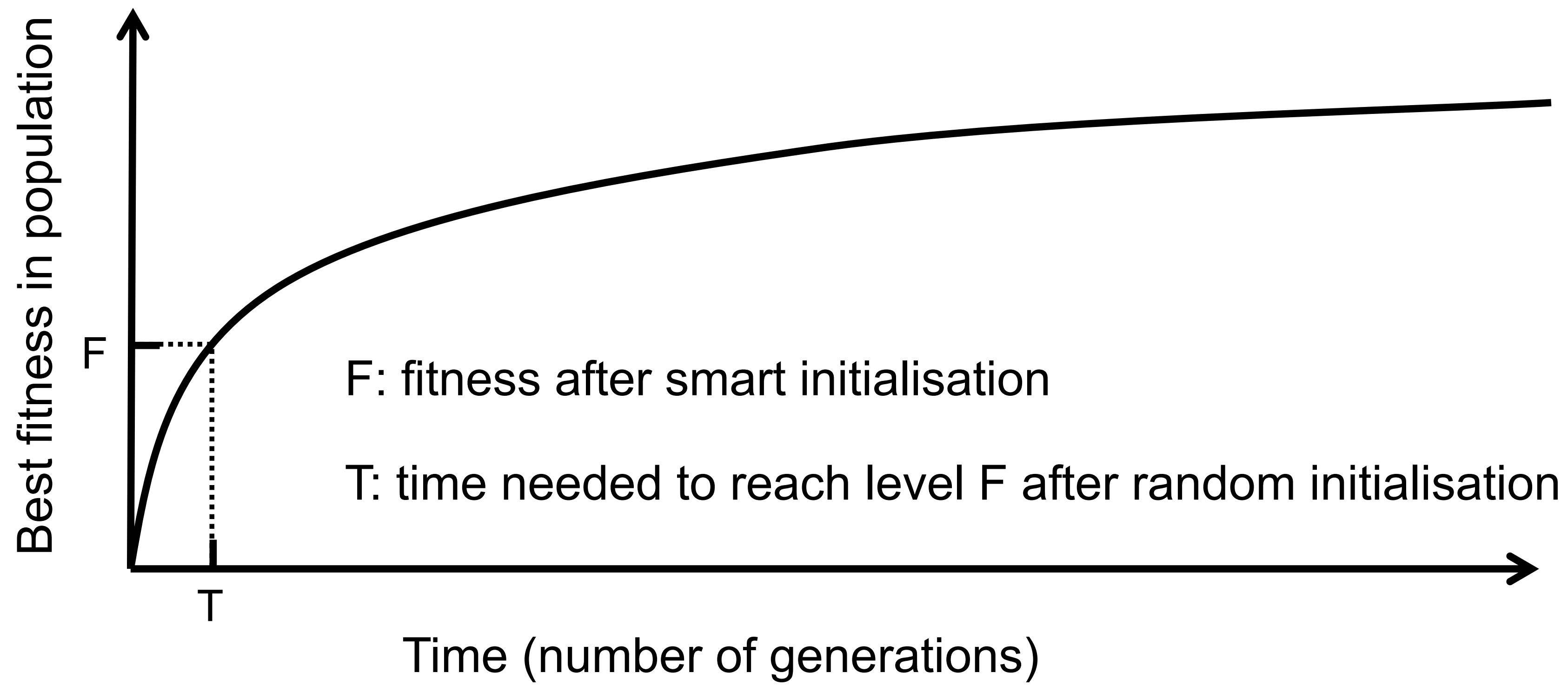
Typical EA Behaviour: Are long runs beneficial?



- it depends on how much you need the last bit of progress
- it may be better to do several short runs instead

Typical EA Behaviour:

Is it worth expending effort on smart initialisation?

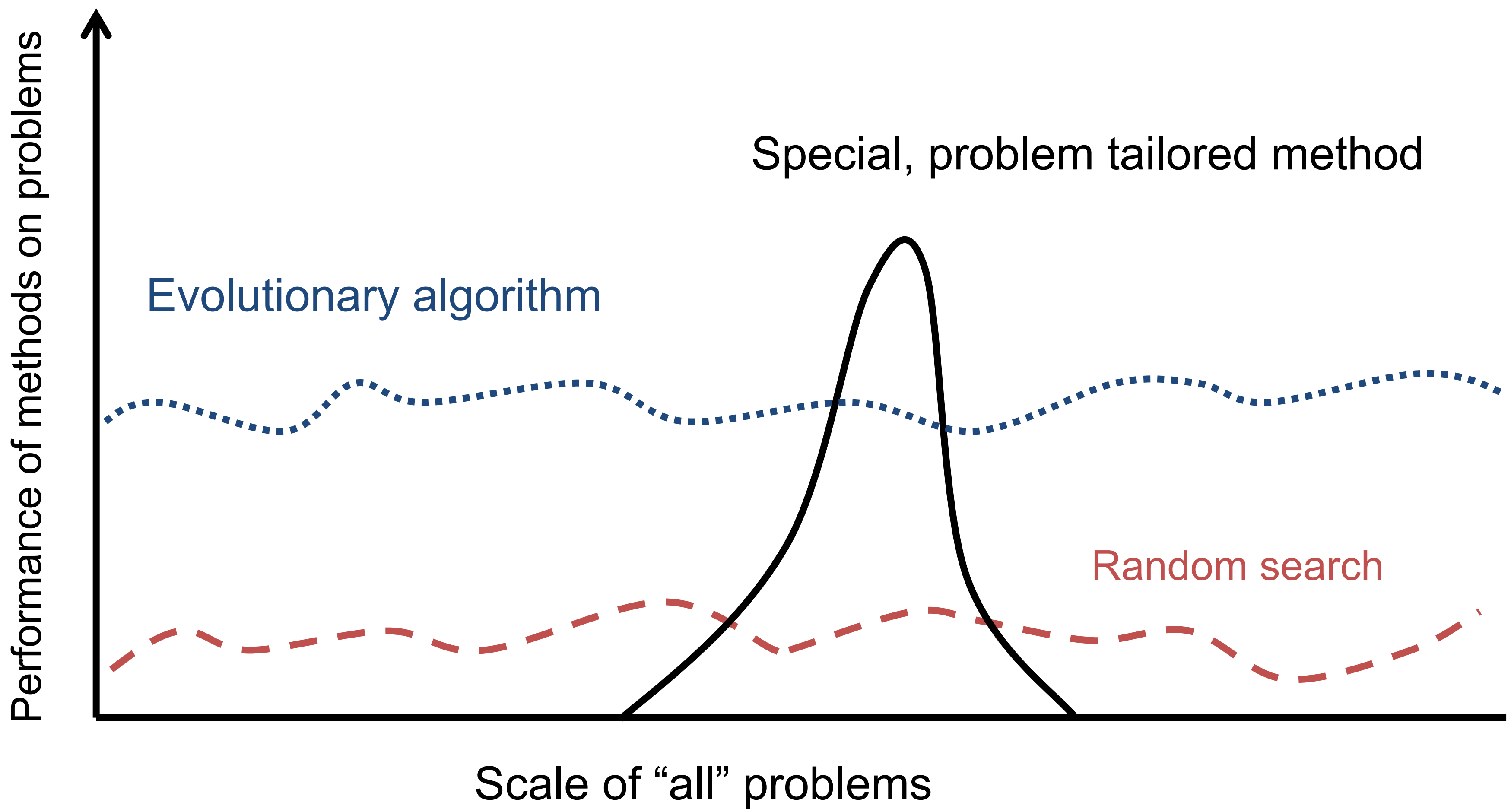


- again, it depends…
- it's possibly worthwhile, if good solutions/methods are known to exist
  - with the assumption that better solutions live nearby on the fitness landscape
- but, as seen here, an EA with random initialisation typically reaches the same level of fitness in a very short amount of time (T)

EA in Context: A Flexible Approach

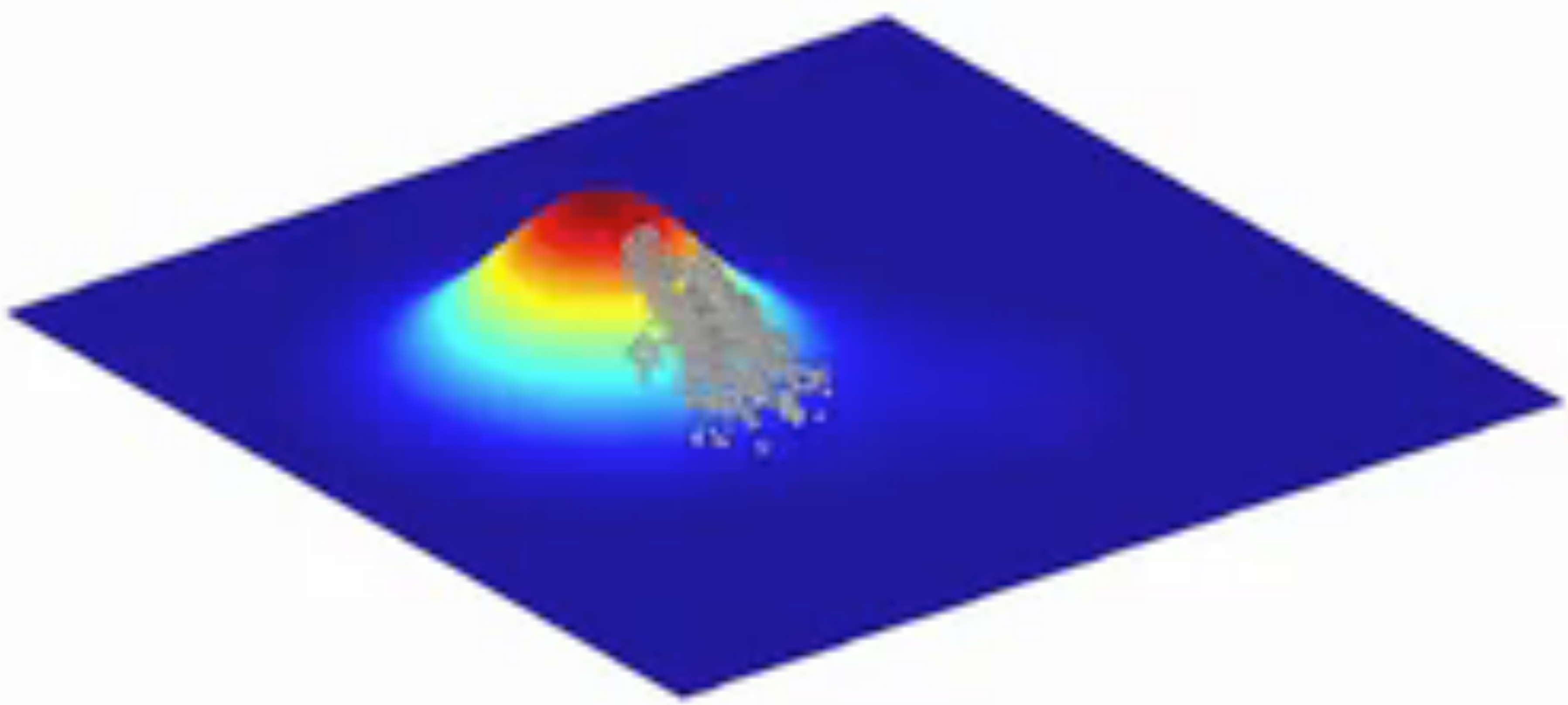
- there are many views on the use of EAs as robust problem solving tools
- for most problems a problem-specific tool might perform better than a generic search algorithm
- but will not transfer well to other, similar problems
- in contrast EA can provide:
  - evenly good performance over a range of problems and instances

EA in Context: A Flexible Approach



Goldberg (1989)

EA in Context: A Flexible Approach Dynamic Fitness Landscape



© Bjørn Østman and Randy Olson

EAs and Domain Knowledge

- an earlier trend in EA research was to add problem specific knowledge to EAs
  - such as special variation operators, repair
- the result?
  - EA performance curve 'deformation':
    - performs better on problems of the given type
    - performs worse on problems different from given type
      - (sound familiar?)
- recent theory suggests the search for an 'all-purpose' algorithm may be fruitless

EC and Global Optimisation

- global optimisation is the search for the best solution x^* out of some fixed set S
- deterministic approaches guarantee to find x^*
  - and can sometimes be fast
  - but in worst case they can be no better than a brute force search
- heuristic approaches, such as EA, use rules for deciding which $x \in S$ to generate next
  - they have no bounds on runtime
  - and provide no guarantee that the best solutions found so far are globally optimal, but…

EC and Neighbourhood Search

- many heuristics impose a neighbourhood structure on S
- such heuristics may guarantee that best point found is locally optimal
    - for example hill-climbers
- while problems often exhibit many local optima, these heurisitics are often very quick to identify good solutions
- EAs are distinguished within the class of heuristic search methods by their use of:
    - a population
    - multiple, stochastic search operators
        - especially variation operators with arity >1
    - stochastic selection

Reading & References

- slides largely based on and adapted from, Chapter 3 slides for Eiben & Smith's *Introduction to Evolutionary Computing*
- A.E. Eiben. Evolutionary computing: the most powerful problem solver in the universe?
- An overview of Evolutionary Algorithms for parameter optimisation