Introduction To Evolutionary Computing

Book *in* Natural Computing Series · January 2003

DOI: 10.1007/978-3-662-05094-1

2 authors:

A. E. Eiben
Vrije Universiteit Amsterdam

Jim Smith
University of the West of England, Bristol

Introduction to Evolutionary Computing II

A.E. Eiben

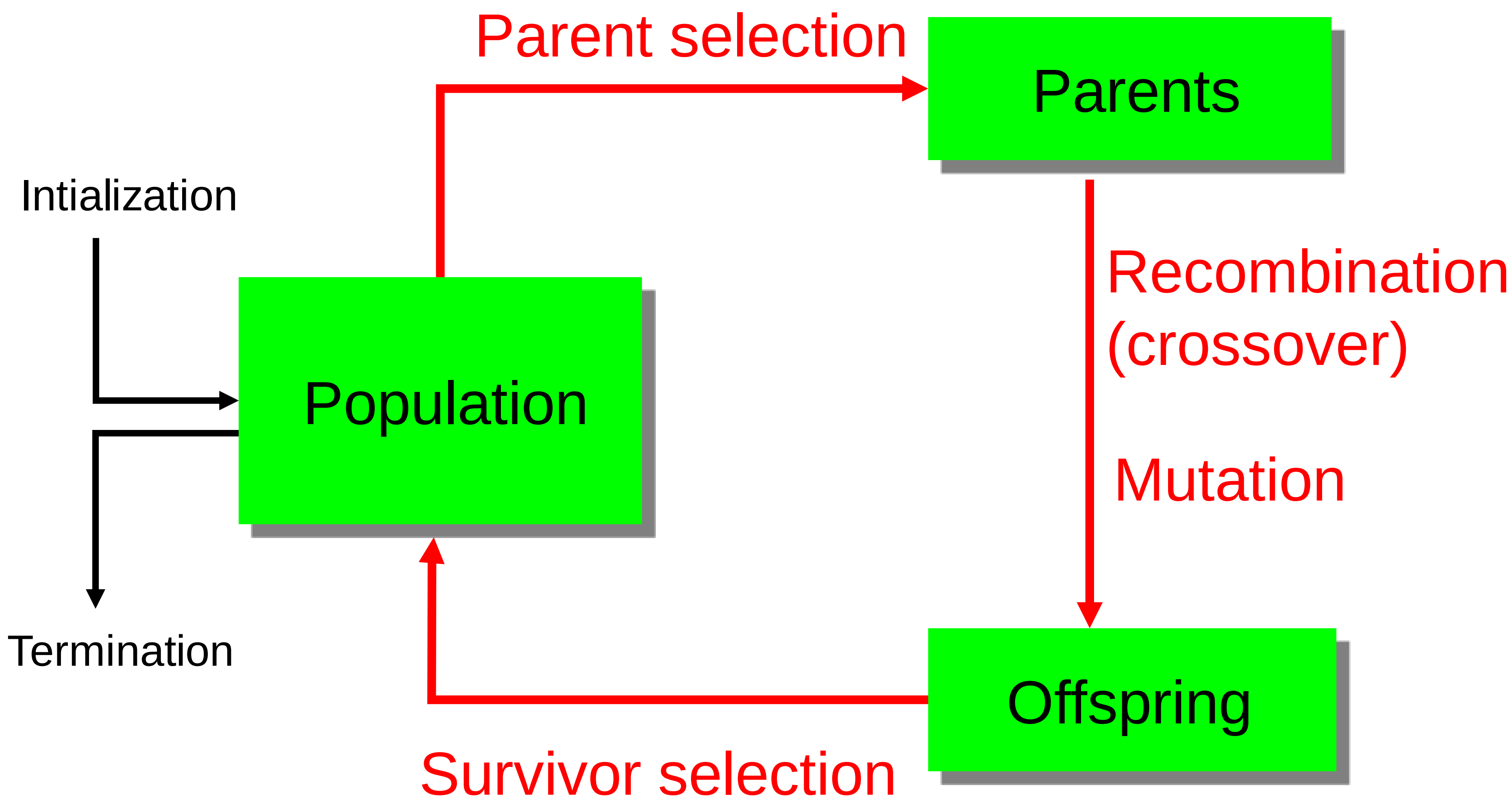
Free University Amsterdam

http://www.cs.vu.nl/~gusz/

with thanks to the EvoNet Training Committee and its “Flying Circus”

Contents

The main evolutionary cycle



The two pillars of evolution

There are two competing forces active

- Increasing population diversity by genetic operators
  - mutation
  - recombination

Push towards novelty

- Decreasing population diversity by selection
  - of parents
  - of survivors

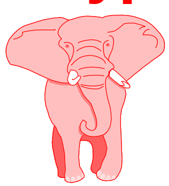
Push towards quality

Components: representation / individuals (1)

Individuals have two levels of existence

- phenotype: object in original problem context, the outside
- genotype: code to denote that object, the inside (a.k.a. chromosome, "digital DNA"):

phenotype:

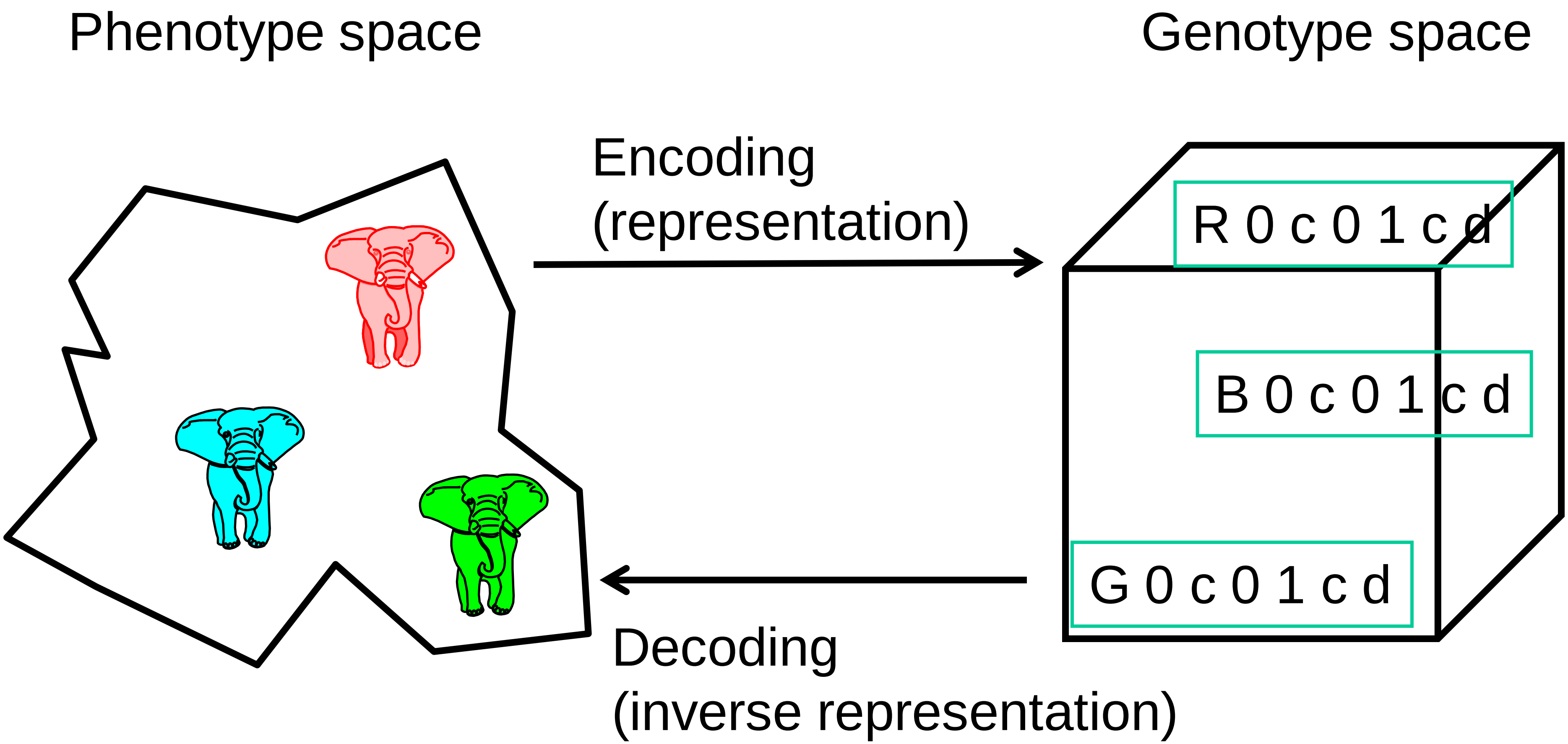


genotype:

a d c a a c b

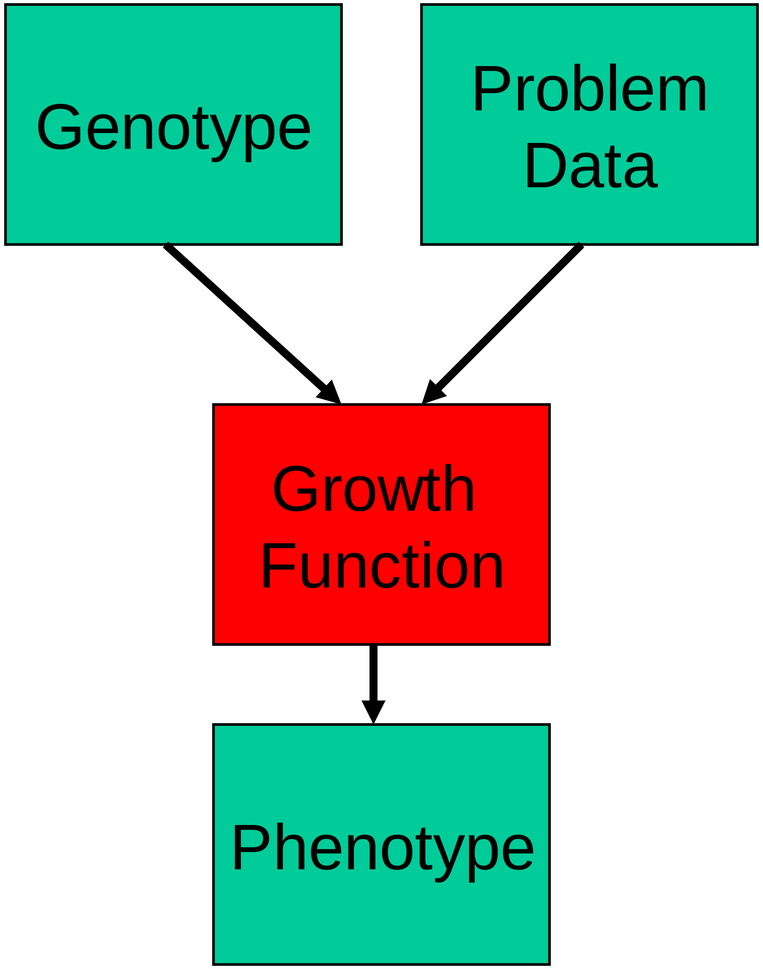
The link between these levels is called representation

Components: representation / individiuals (2)



Components: representation / individuals (3)

- Sometimes producing the phenotype from the genotype is a simple and obvious process.
- Other times the genotype might be a set of parameters to some algorithm, which works on the problem data to produce the phenotype



Components: representation / individuals (4)

- Search takes place in the genotype space
- Evaluation takes place in the phenotype space
  - *Repr: Phenotypes $\rightarrow$ Genotypes*
  - *Fitness(g) = Value(repr^{-1}(g))*
- *Repr* must be invertible, in other words decoding must be injective (Q: surjective?)
- Role of representation: defines objects that can be manipulated by (genetic) operators
- Note back on Darwinism: no mutations on phenotypic level! (right term: small random variations)

Components: evaluation, fitness measure

Role:

- represents the task to solve, the requirements to adapt to
- enables selection (provides basis for comparison)

Some phenotypic traits are advantageous, desirable, e.g. big ears cool better,

These traits are rewarded by more offspring that will expectedly carry the same trait

Components: population

Role: holds the candidate solutions of the problem as individuals (genotypes)

Formally, a population is a multiset of individuals,
i.e. repetitions are possible

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

Components: selection

Role:

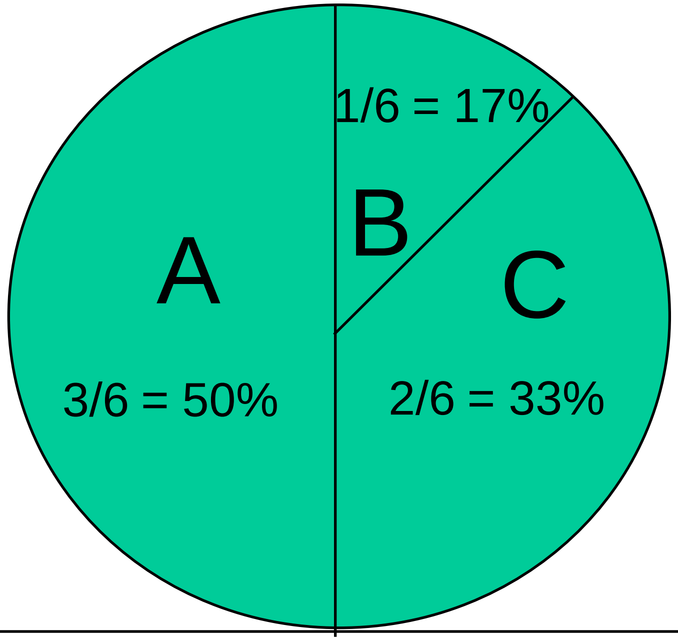
- Gives better individuals a higher chance of
  - becoming parents
  - surviving
- Pushes population towards higher fitness

E.g. roulette wheel selection

fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

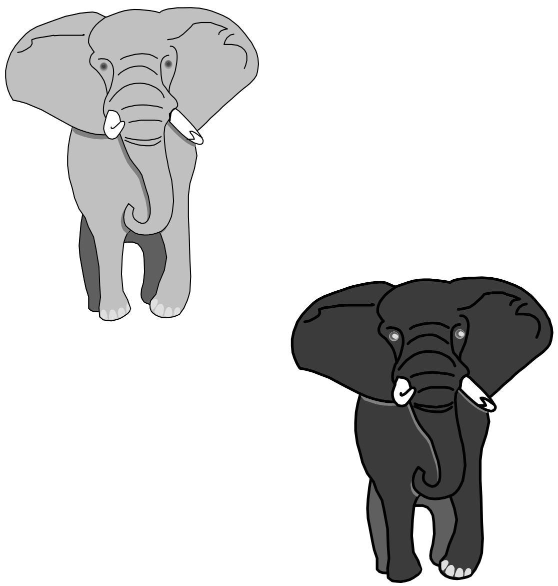


Components: Mutation

Role: causes small (random) variance

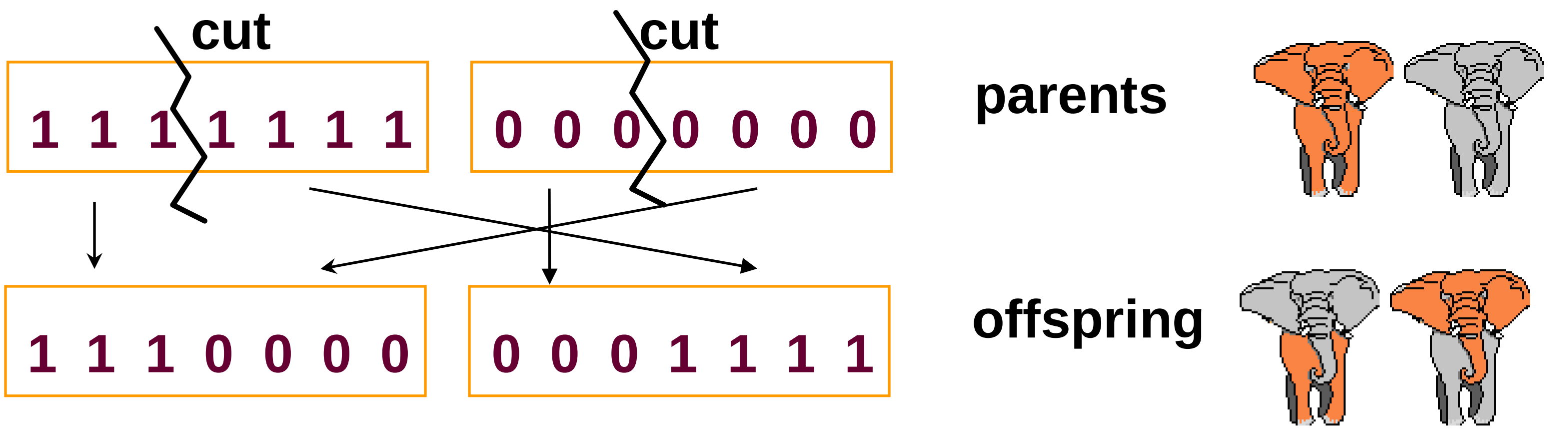
before 1 1 1 1 1 1 1

after 1 1 1 0 1 1 1

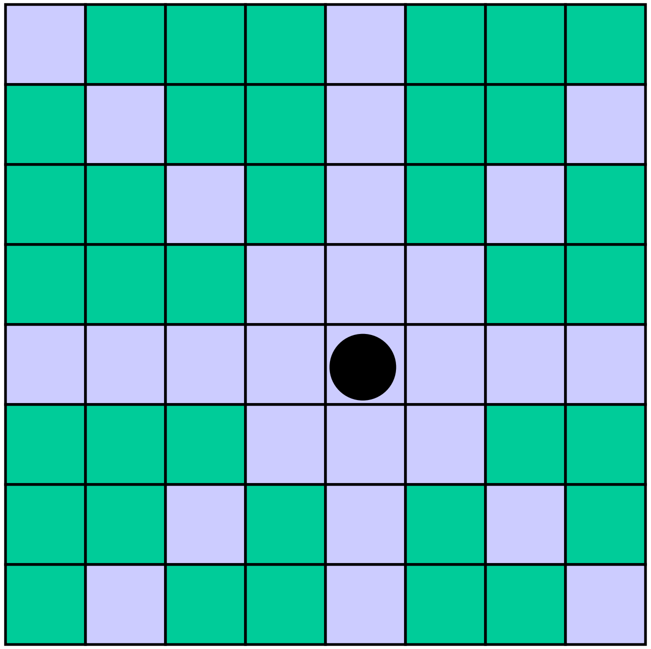


Components: Recombination

Role: combines features from different sources



Example: the 8 queens problem



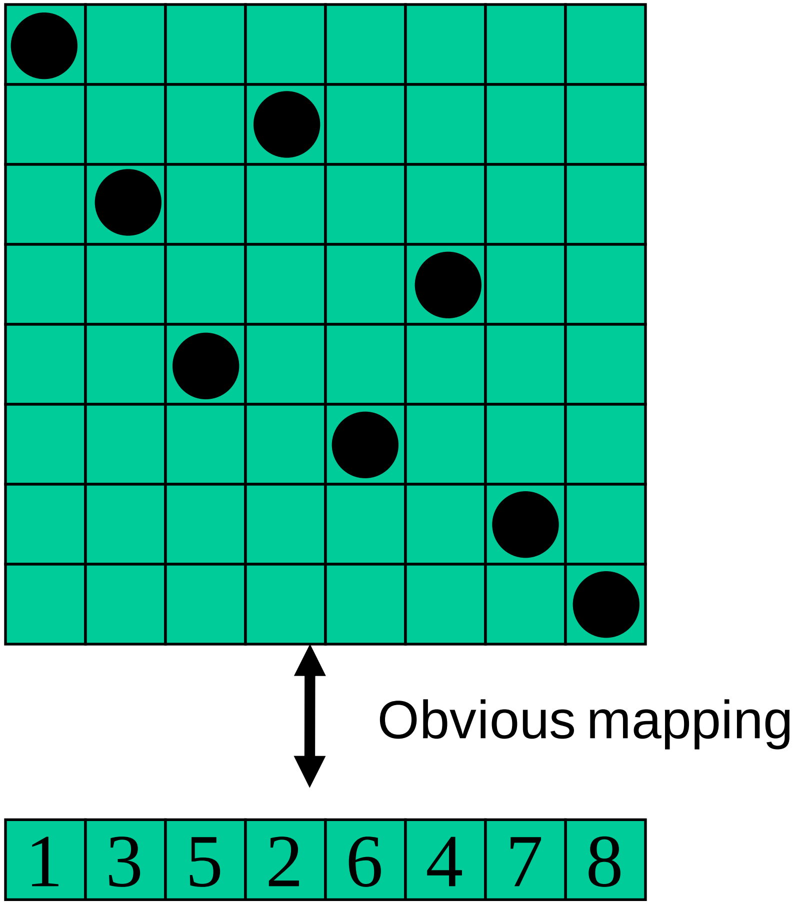
Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other

The 8 queens problem

Representation

Phenotype:
a board configuration

Genotype:
a permutation of
the numbers 1 - 8



The 8 queens problem

Fitness evaluation

Penalty of one queen:
the number of queens she can check.

Penalty of a configuration:
the sum of the 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, e.g.:

- swapping values of two randomly chosen positions, or
- inverting a randomly chosen segment

1	3	5	2	6	4	7	8

$\longrightarrow$

1	3	7	2	6	4	5	8

The 8 queens problem

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 there
    - beginning after crossover point
    - skipping values already in child

1	3	5	2	6	4	7	8
8	7	6	5	4	3	2	1

→

1	3	5	4	2	8	7	6
8	7	6	2	4	1	3	5

The 8 queens problem

Selection

Parent selection:

Roulette wheel selection, for instance

Survivor selection (replacement)

When inserting a new child into the population, choose an existing member to replace by:

- sorting the whole population by decreasing fitness
- enumerating this list from high to low
- replacing the first with a fitness lower than the given child

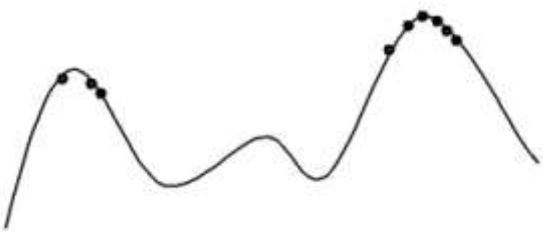
Note: selection works on fitness values, no need to adjust it to representation

Working of an EA

Phases in optimizing on a 1-dimensional fitness landscape

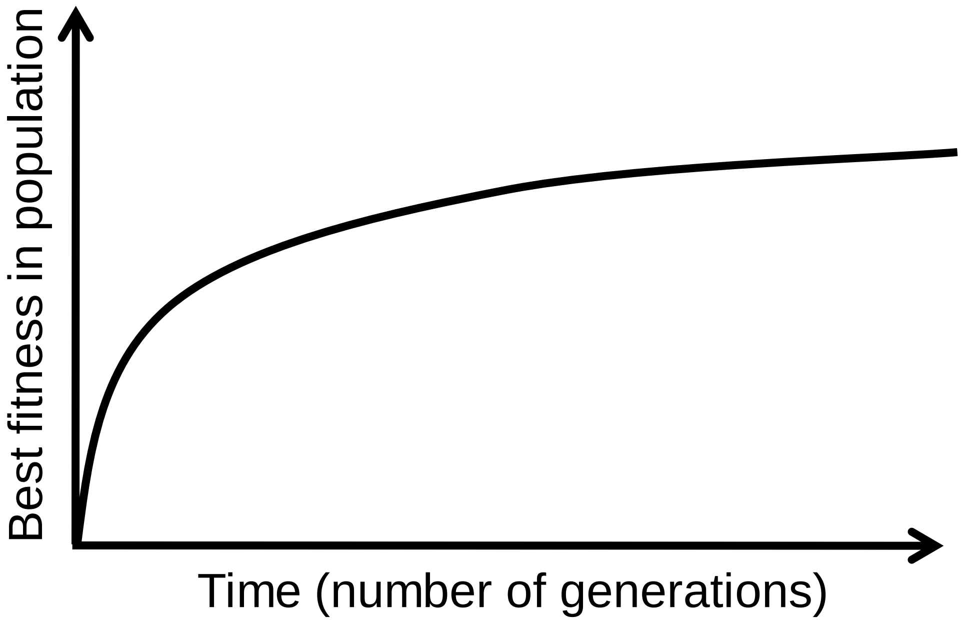
Early phase:
quasi-random population distribution

Mid-phase:
population arranged around/on hills



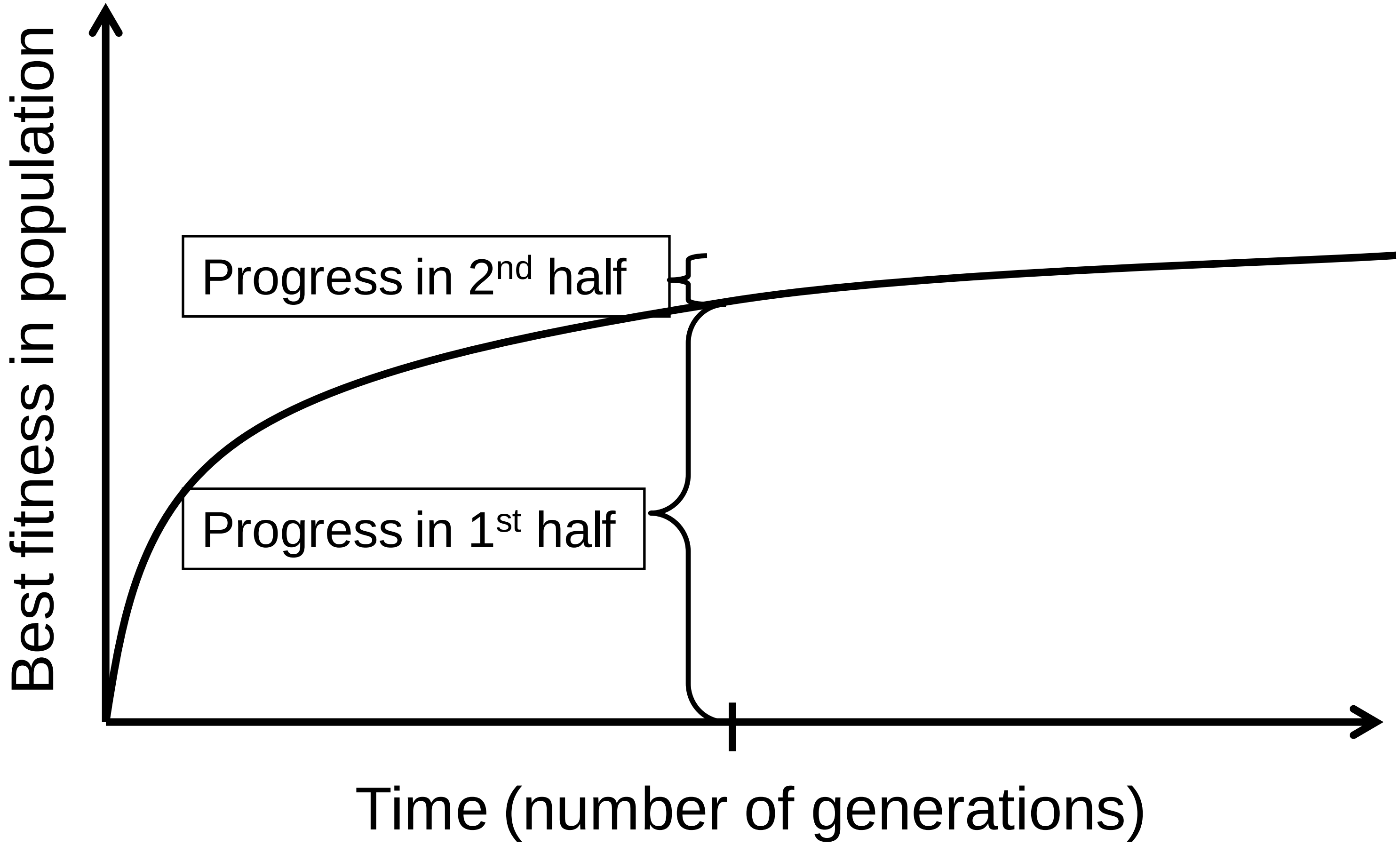
Late phase:
population concentrated on high hills

Typical run



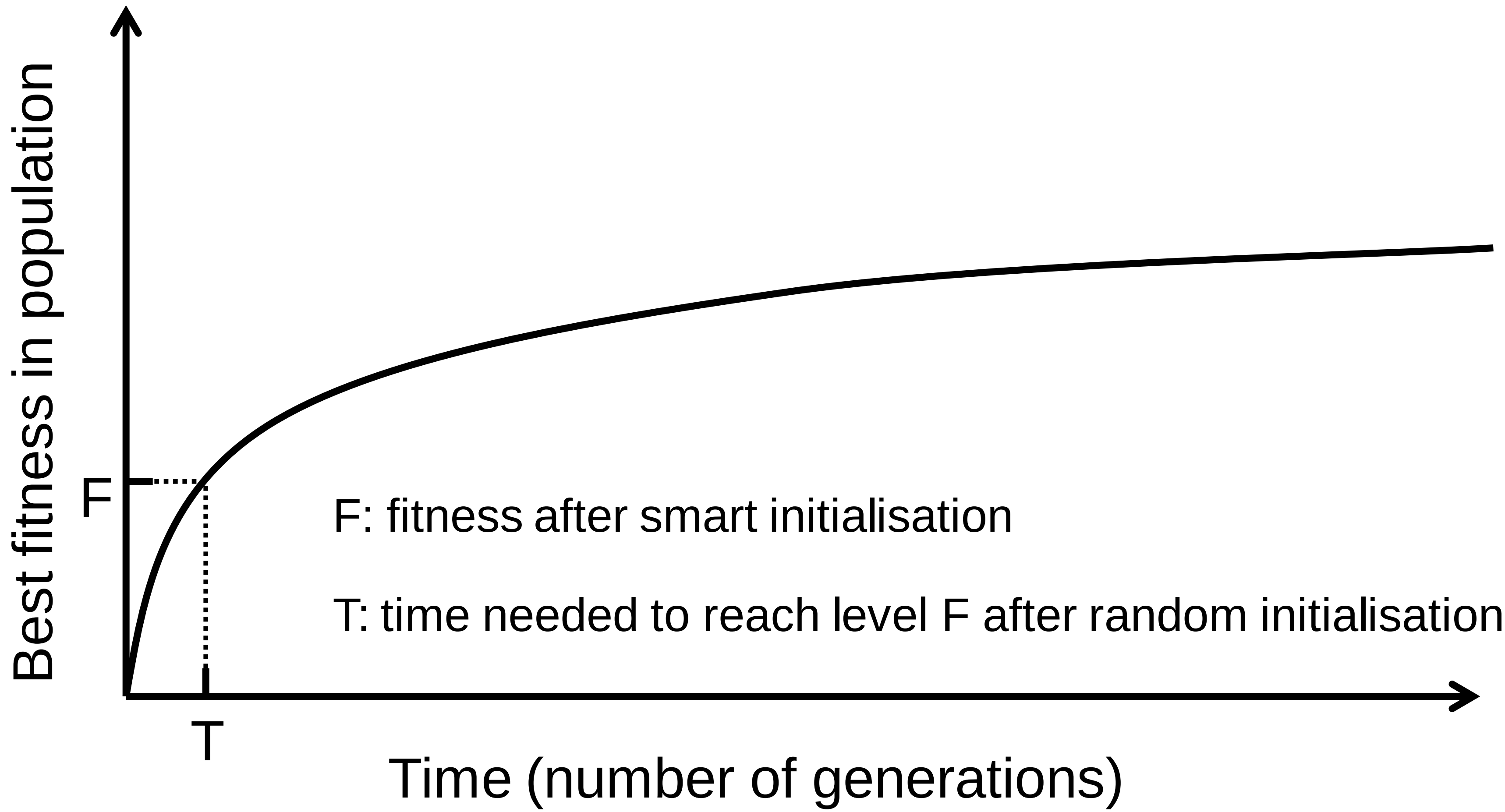
Typical run of an EA shows so-called “anytime behavior”

Long runs?

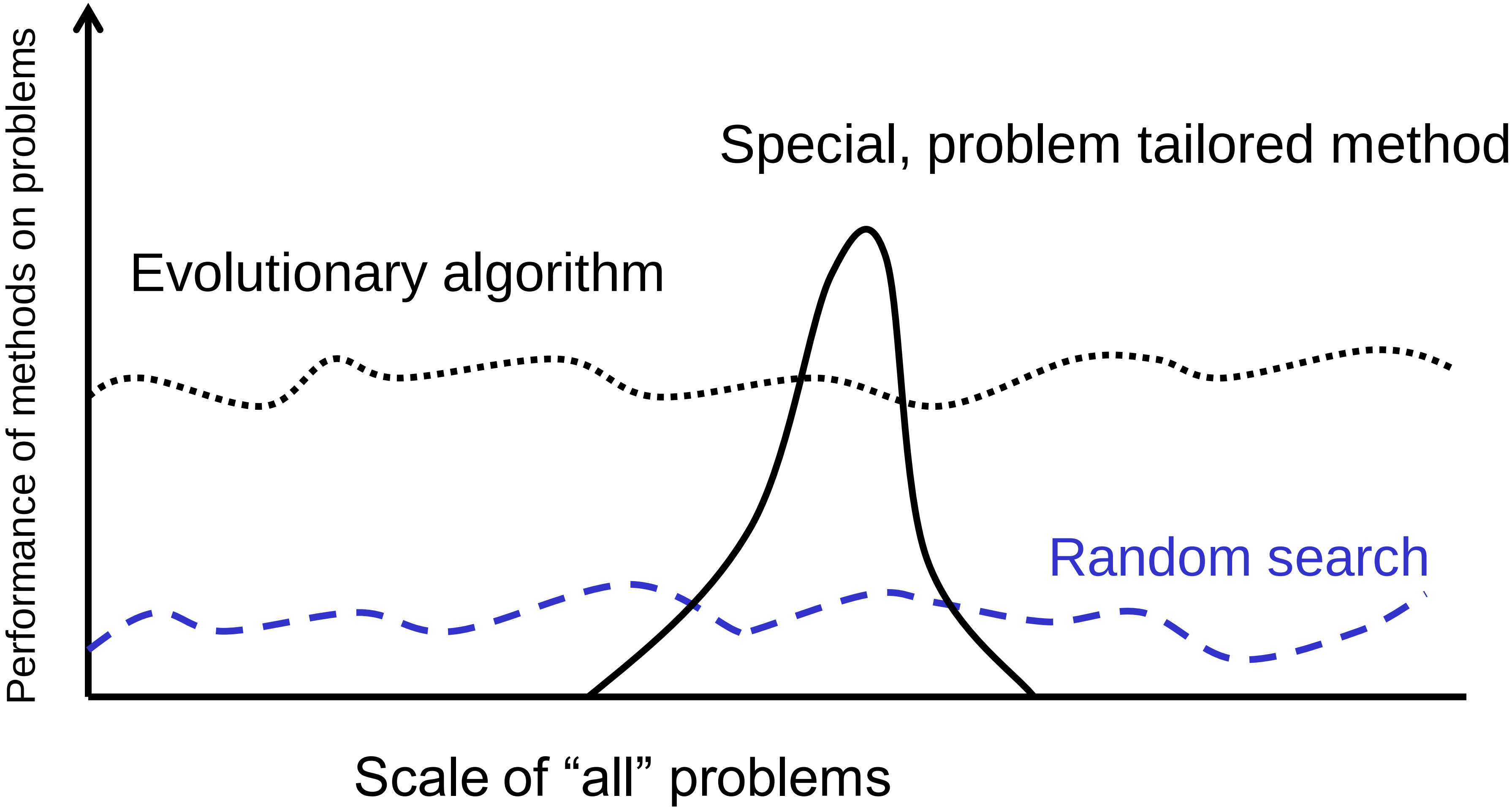


Best fitness in population
Progress in 2nd half
Progress in 1st half
Time (number of generations)

Smart initialisation?



Goldberg'89 view



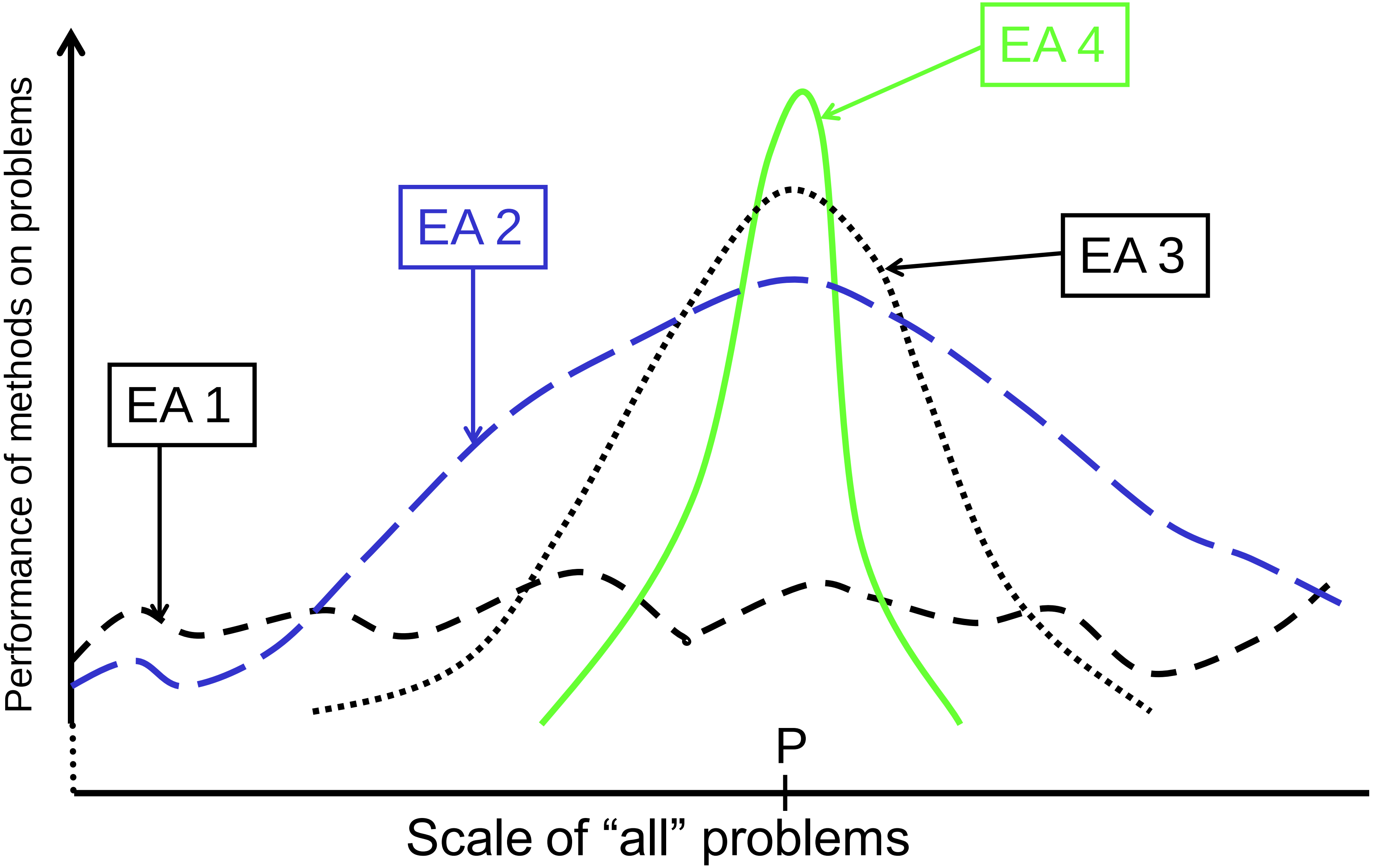
EAs and domain knowledge

- Trend in the 90ies: adding problem specific knowledge to EAs (special variation operators, repair, etc)

- Result: EA performance curve “deformation”:
  - better on problems of the given type
  - worse on problems different from given type

- Amount of added knowledge is variable

Michalewicz'96 view



General EA framework and dialects

There is a general, formal EA framework (omitted here)

- In theory:
  - every EA is an instantiation of this framework, thus:
  - specifying a particular EA or a type of EAs (a “dialect”) needs only filling in the characteristic features
- In practice
  - this would be too formalistic
  - there are many exceptions (EAs not fitting into this framework)
  - why care about the taxonomy, or label?

Genetic algorithms & genetic programming

Genetic algorithms (USA, 70's, Holland, DeJong):

- Typically applied to: discrete optimization
- Attributed features:
  - not too fast
  - good solver for combinatorial problems
- Special: many variants, e.g., reproduction models, operators

Genetic programming (USA, 90's, Koza)

- Typically applied to: machine learning tasks
- Attributed features:
  - competes with neural nets and alike
  - slow
  - needs huge populations (thousands)
- Special: non-linear chromosomes: trees, graphs

Evolution strategies & evolutionary programming

Evolution strategies (Germany, 70's, Rechenberg, Schwefel)

- Typically applied to:
  - numerical optimization
- Attributed features:
  - fast & good optimizer for real-valued optimization
  - relatively much theory
- Special:
  - self-adaptation of (mutation) parameters standard

- Evolutionary programming (USA, 60's, Fogel et al.)
  - Typically applied to: machine learning (old EP), optimization
  - Attributed features:
    - very open framework: any representation and mutation op's OK
  - Special:
    - no recombination
    - self-adaptation of parameters standard (contemporary EP)

Beyond dialects

- Field merging from the early 1990's
- No hard barriers between dialects, many hybrids, outliers
- Choice for dialect should be motivated by given problem
- Best practical approach: choose representation, operators, population model, etc. pragmatically (and end up with an "unclassifiable" EA)
- There are general issues for EC as a whole

Advantages of EC

- No presumptions w.r.t. problem space
- Widely applicable
- Low development & application costs
- Easy to incorporate other methods
- Solutions are interpretable (unlike NN)
- Can be run interactively, accommodate user proposed solutions
- Provides many alternative solutions
- Intrinsic parallelism,straightforward parallel implementations

Disadvantages of EC

- No guarantee for optimal solution within finite time
- Weak theoretical basis
- May need parameter tuning
- Often computationally expensive, i.e. slow

The performance of EC

- Acceptable performance at acceptable costs on a wide range of problems
- EC niche (where supposedly superior to other techniques): complex problems with one or more of the following features
  - many free parameters
  - complex relationships between parameters
  - mixed types of parameters (integer, real)
  - many local optima
  - multiple objectives
  - noisy data
  - changing conditions (dynamic fitness landscape)

Summary

Evolutionary Computation:

- is a method, based on biological metaphors, of breeding solutions to problems
- has been shown to be useful in a number of areas
- could be useful for your problem
- its easy to give it a try
- is FUN