What is an Evolutionary Algorithm?

Presented by:
Faramarz Safi (Ph.D.)
Faculty of Computer Engineering
Islamic Azad University, Najafabad Branch

Chapter 2

Contents

- Recap of Evolutionary Metaphor
- Basic scheme of an EA
- Basic Components:
  - Representation / Evaluation / Population / Parent Selection / Recombination / Mutation / Survivor Selection / Termination
- Examples : eight queens / knapsack
- Typical behaviours of EAs
- EC in context of global optimisation

Aims of this chapter

The most important aim of this chapter is to describe what an evolutionary algorithm is. This description is deliberately based on a unifying view presenting a general scheme that forms the common basis of all evolutionary algorithm (EA) variants.

The main components of EAs arc discussed, explaining their role and related issues of terminology. This is immediately followed by two

example applications (unlike other chapters, where example applications are typically given at the end) to make things more concrete.

Further on we discuss general issues of the working of EAs. Finally, we put EAs into a broader context and explain their relation with other global optimisation techniques.

What is an evolutionary algorithm

The common underlying idea behind all these techniques is the same:

- given a population of individuals,
- the environmental pressure causes natural selection (survival of the fittest), which causes a rise in the fitness of the population.
- Given a quality function to be maximized, we can randomly create a set of candidate solutions, i.e., elements of the function's domain, and apply the quality function as an abstract fitness measure – the higher the better.
- Based on this fitness, some of the better candidates are chosen to seed the next generation by applying recombination and/or mutation to them. Recombination is an operator applied to two or more selected candidates (the so-called parents) and results one or more new candidates (the children).
- Mutation is applied to one candidate and results in one new candidate.
- Executing recombination and mutation leads to a set of new candidates (the offspring) that compete Based on their fitness (and possibly age)- with the old ones for a place in the next generation.
- This process can be iterated until a candidate with sufficient quality (a solution) is found or a previously set computational limit is reached.

What is an evolutionary algorithm
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

What is an evolutionary algorithm

In this process there are two fundamental forces that form the basis of evolutionary systems:

- Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty.
- Selection acts as a force pushing quality.

What is an evolutionary Algorithm

The combined application of variation and selection generally leads to improving fitness values in consecutive populations.

It is easy (although somewhat misleading) to see such a process as if the evolution is optimising, or at least “approximising” by approaching optimal values closer and closer over its course.

Alternatively, evolution it is often seen as a process of adaptation. From this perspective, the fitness is not seen as an objective function to be optimised, but as an expression of environmental requirements.

Matching these requirements more closely implies an increased viability, reflected in a higher number of offspring. The evolutionary process makes the population increasingly better at being adapted to the environment.

What is an evolutionary Algorithm

Let us note that many components of such an evolutionary process are stochastic.

During selection, fitter individuals have a higher chance to be selected than less fit ones, but typically even the weak individuals have a chance to become a parent or to survive.

For recombination of individuals the choice of which pieces will be recombined is random.

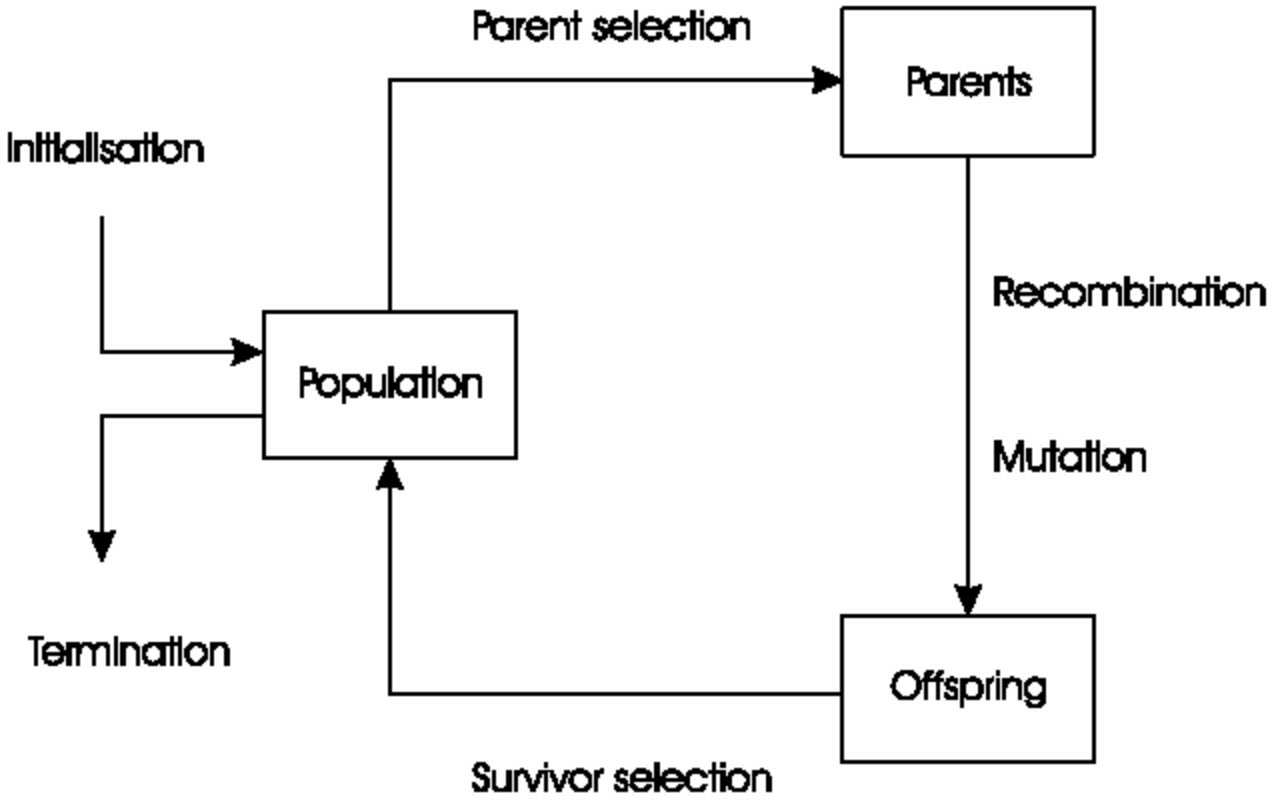
Similarly for mutation, the pieces that will be mutated within a candidate solution, and the new pieces replacing them, are chosen randomly.

The general scheme of an evolutionary algorithm can is given in Fig. 2.1 in a pseudo code fashion; Fig. 2.2 shows a diagram.

Recap 2:

- EAs fall into the category of “generate and test” algorithms
- They are stochastic, population-based algorithms
- Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty
- Selection reduces diversity and acts as a force pushing quality

General Scheme of EAs



Pseudo-code for typical EA

```
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
```

Fig 2.1: The general scheme of an evolutionary algorithm in pseudo-code

What are the different types of EAs

- Historically different flavours of EAs have been associated with different representations
  - Binary strings : Genetic Algorithms
  - Real-valued vectors : Evolution Strategies
  - Finite state Machines: Evolutionary Programming
  - LISP trees: Genetic Programming
- These differences are largely irrelevant, best strategy
  - choose representation to suit problem
  - choose variation operators to suit representation
- Selection operators only use fitness and so are independent of representation

Components of Evolutionary Algorithms

EAs have a number of components, procedures, or operators that must be specified in order to define a particular EA. The most important components, indicated by italics in Fig. 2.1, are:

- Representation (definition of individuals)
- Evaluation function (or fitness function)
- Population
- Parent selection mechanism
- Variation operators, recombination and mutation
- Survivor selection mechanism (replacement)

Each of these components must be specified in order to define a particular EA. Furthermore, to obtain a running algorithm the initialisation procedure and a termination condition must be also defined.

Representations (Definition of Individuals)

- The first step in defining an EA is to link the "real world" to the "EA world", that is, to set up a bridge between the original problem context and the problem-solving space where evolution takes place. Objects forming possible solutions within the original problem context are referred to as pheno-types, while their encoding, that is, the individuals within the EA, are called genotypes.
- The first design step is commonly called representation, as it amounts to specifying a mapping from the phenotypes onto a set of genotypes that are said to represent these phenotypes.
- Candidate solutions (individuals) exist in *phenotype* space
- They are encoded in chromosomes, which exist in *genotype* space
  - Encoding : phenotype=> genotype (not necessarily one to one)
  - Decoding : genotype=> phenotype (must be one to one)
- A place-holder is commonly called a variable, a locus (plural: loci), a position, or in a biology-oriented terminology a gene. An object on such a place can be called a value or an allele. Alternatively, chromosomes contain genes, which are in (usually fixed) positions called loci (sing. locus) and have a value (allele)

In order to find the global optimum, every feasible solution must be represented in genotype space

Evaluation (Fitness) Function

- Represents the requirements that the population should adapt to
- a.k.a. *quality* function or *objective* function
- Assigns a single real-valued fitness to each phenotype which forms the basis for selection
  - So the more discrimination (different values) the better
- Typically we talk about fitness being maximised
  - Some problems may be best posed as minimisation problems, but conversion is trivial

Population

- Holds (representations of) possible solutions
- Usually has a fixed size and is a *multiset* of genotypes
- Some sophisticated EAs also assert a spatial structure on the population e.g., a grid.
- Selection operators usually take whole population into account i.e., reproductive probabilities are *relative* to *current* generation
- Diversity of a population refers to the number of different fitnesses / phenotypes / genotypes present (note not the same thing)

Parent Selection Mechanism

- Assigns variable probabilities of individuals acting as parents depending on their fitnesses
- Usually probabilistic
  - high quality solutions more likely to become parents than low quality
  - but not guaranteed
  - even worst in current population usually has non-zero probability of becoming a parent
- This *stochastic* nature can aid escape from local optima

Variation Operators

- Role is to generate new candidate solutions
- Usually divided into two types according to their arity (number of inputs):
  - Arity 1 : mutation operators
  - Arity >1 : Recombination operators
  - Arity = 2 typically called crossover
- There has been much debate about relative importance of recombination and mutation
  - Nowadays most EAs use both
  - Choice of particular variation operators is representation dependant

Mutation

- Acts on one genotype and delivers another
- Element of randomness is essential and differentiates it from other unary heuristic operators
- Importance ascribed depends on representation and 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

Recombination

- Merges information from parents into offspring
- Choice of what information to merge is stochastic
- Most offspring may be worse, or the same as the parents
- Hope is that some are better by combining elements of genotypes that lead to good traits
- Principle has been used for millennia by breeders of plants and livestock

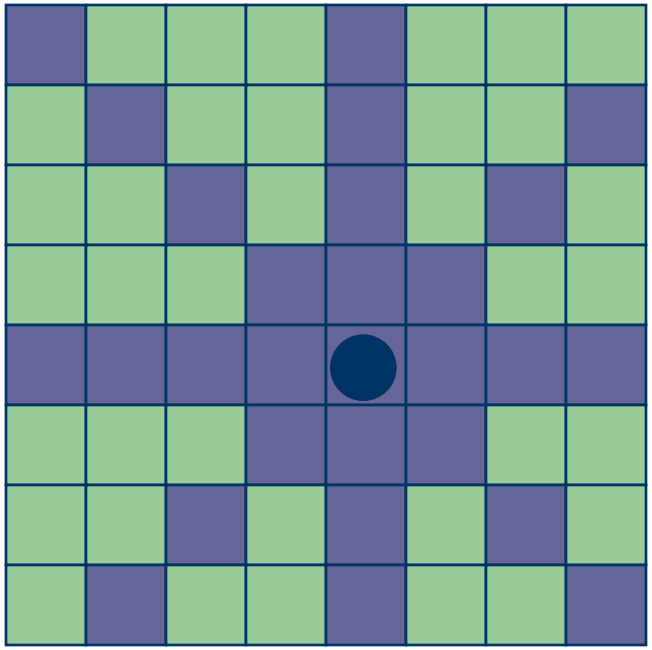
Survivor Selection

- a.k.a. ***replacement***
- Most EAs use fixed population size so need a way of going from (parents + offspring) to next generation
- Often deterministic
  - Fitness based : e.g., rank parents+offspring and take best
  - Age based: make as many offspring as parents and delete all parents
- Sometimes do combination (elitism)

Initialisation / Termination

- Initialisation usually done at random,
  - Need to ensure even spread and mixture of possible allele values
  - Can include existing solutions, or use problem-specific heuristics, to “seed” the population
- Termination condition checked every generation
  - Reaching some (known/hoped for) fitness
  - Reaching some maximum allowed number of generations
  - Reaching some minimum level of diversity
  - Reaching some specified number of generations without fitness improvement

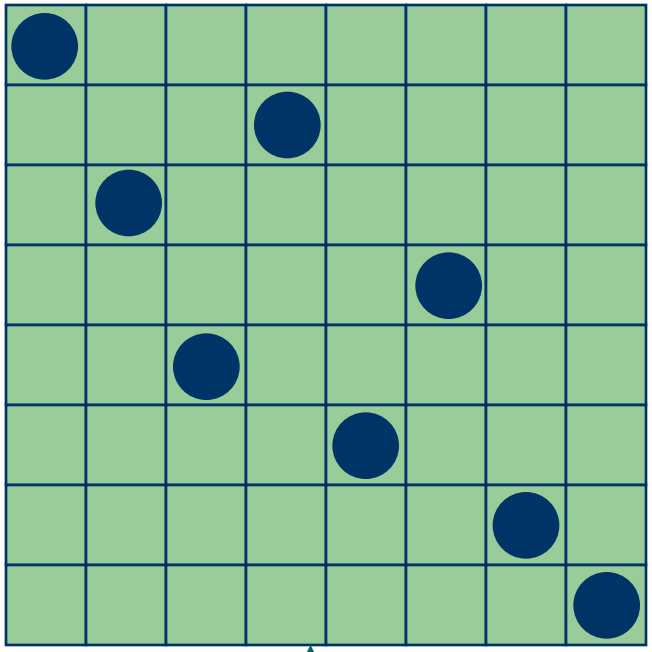
Example: the 8 queens problem



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



Obvious mapping

Genotype:
a permutation of
the numbers 1 - 8

1	3	5	2	6	4	7	8

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

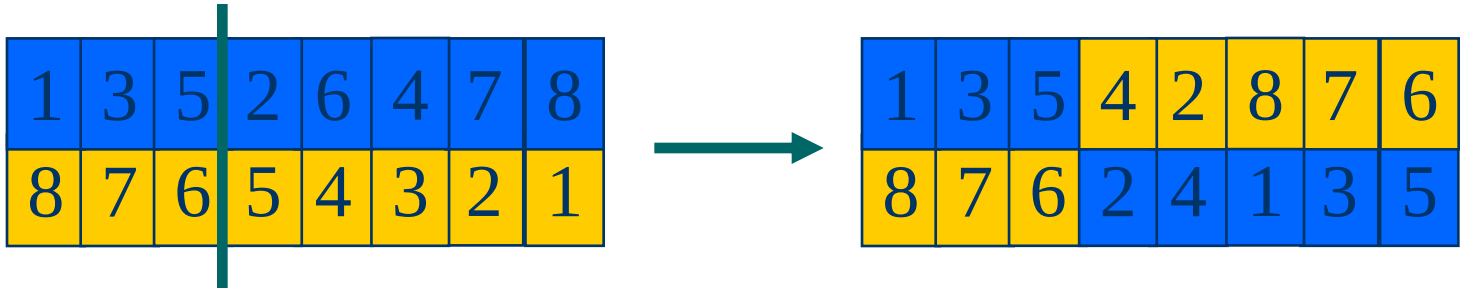
Small variation in one permutation, e.g.:

- swapping values of two randomly chosen positions,

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



Cut and Crossfill Crossover

1. Select a random position, the crossover point, $i \in \{1, \ldots, 7\}$
2. Cut both parents in two segments after this position
3. Copy the first segment of parent 1 into child 1 and the first segment of parent 2 into child 2
4. Scan parent 2 from left to right and fill the second segment of child 1 with values from parent 2, skipping those that are already contained in it
5. Do the same for parent 1 and child 2

Fig. 2.3. "Cut-and-crossfill" crossover

The 8 queens problem: Selection

- Parent selection:
  - Pick 5 parents and take best two to undergo crossover
- 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

8 Queens Problem: 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

Note that is is ***only one possible***
set of choices of operators and parameters

Example: 0-1-KNAPSACK problem

The **knapsack problem** is a problem in *combinatorial optimization*: **Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible**.

It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most useful items.

http://www.nils-haldenwang.de/computer-science/computational-intelligence/genetic-algorithm-vs-0-1-knapsack

Introduction – Knapsack Problem

We have a list of positive integers $a_1, \ldots, a_n$, and another integer b.

Find a subset $a_{i1}, \ldots, a_{ik}$, so that $a_{i1} + \ldots + a_{ik} = b$.

Pack Volume=b

Can we find k objects which will fit the pack volume b perfectly?

Size A1

Size A2

Size A3

Size A4

Size A7

Size A5

Size A6

Knapsack Problem: Implementation question

- From the viewpoint of **programming language**, how can you create/**represent individuals** and the **population** to manipulate them?
- i.e. an individual could be a bit, byte, int, char, ..., and the population is a vector, matrix, structure, etc.? Justify your answer.

Knapsack Problem

Candidate Solutions can be represented as knapsack vectors:

$S=(s_1, \ldots, s_n)$ where s_i is 1 if a_i is included in our solution set, and 0 if a_i is not.

Example:

We are given a_1, a_2, a_3, a_4, a_5, a_6 and b.

A potential solution is the subset a_1, a_2, a_4 .

We represent it as a knapsack vector:

(1, 1, 0, 1, 0, 0)

Knapsack Problem

Start with a population of random knapsack vectors:

(0,0,1,0,1,0,0,1,1,1)

(1,1,0,0,0,1,0,0,1,0)

(1,0,1,0,0,0,1,1,0,1)

…..

Compute fitness scores

7

8

20

…..

Reproduce

(1,1,0,0,0,1,0,0,1,0)

(1,0,1,0,0,0,1,1,0,1)

(1,1,0,0,0,0,1,1,0,1)

(0,0,1,0,1,0,0,1,1,1)

(1,0,1,0,0,0,1,1,0,1)

(0,0,1,0,1,0,1,1,0,1)

….

….

Knapsack Problem

Random mutation

(1,1,0,0,0,0,1,1,0,1) ⟶ (1,1,0,0,0,0,1,1,0,1)

(0,0,1,0,1,0,1,1,0,1) (0,0,1,0,1,***1***,1,1,0,1)

…. ….

Repeat reproduction and mutation process until

1. A valid solution is found
2. 200,000 iterations executed

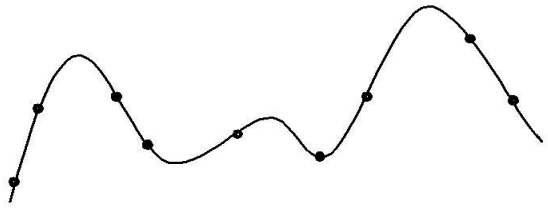
Knapsack Problem: summary

Representation	binary strings of length n
Recombination	One point crossover
Recombination probability	70%
Mutation	each value inverted with independent probability p_m per position
Mutation probability p_m	$1/n$
Parent selection	best out of random two
Survival selection	generational
Population size	500
Number of offspring	500
Initialisation	random
Termination condition	no improvement in last 25 generations

Tableau describing the EA for the Knapsack Problem

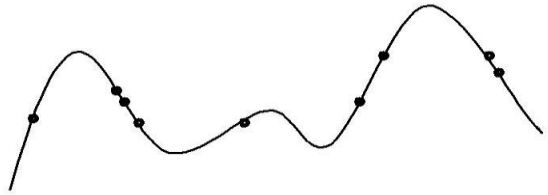
Typical behaviour of an EA

Phases in optimising on a 1-dimensional fitness landscape



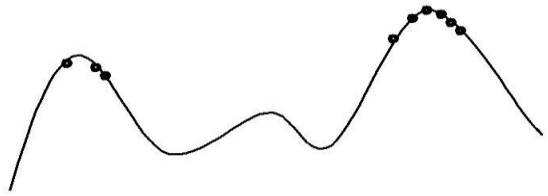
Early phase:

quasi-random population distribution



Mid-phase:

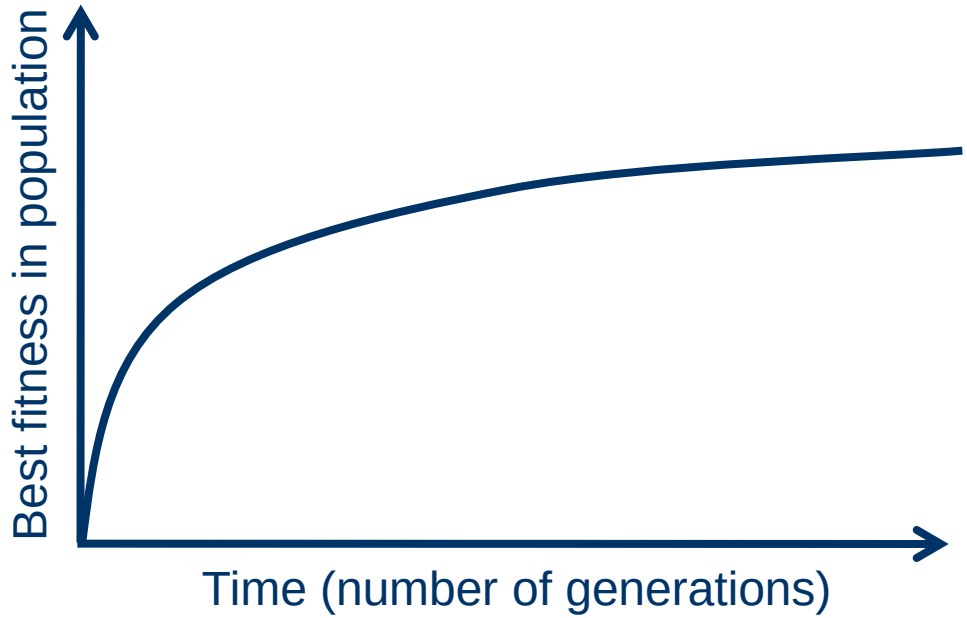
population arranged around/on hills



Late phase:

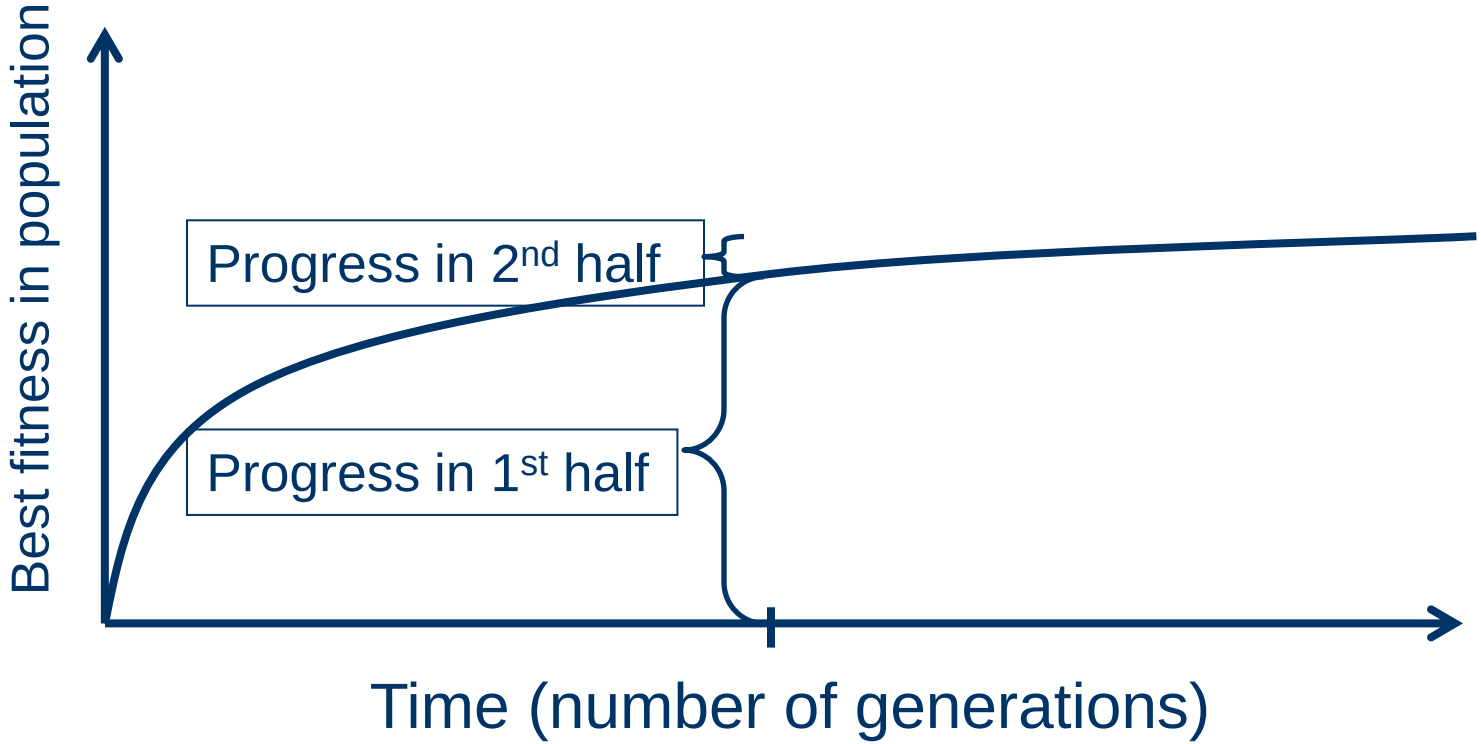
population concentrated on high hills

Typical run: progression of fitness



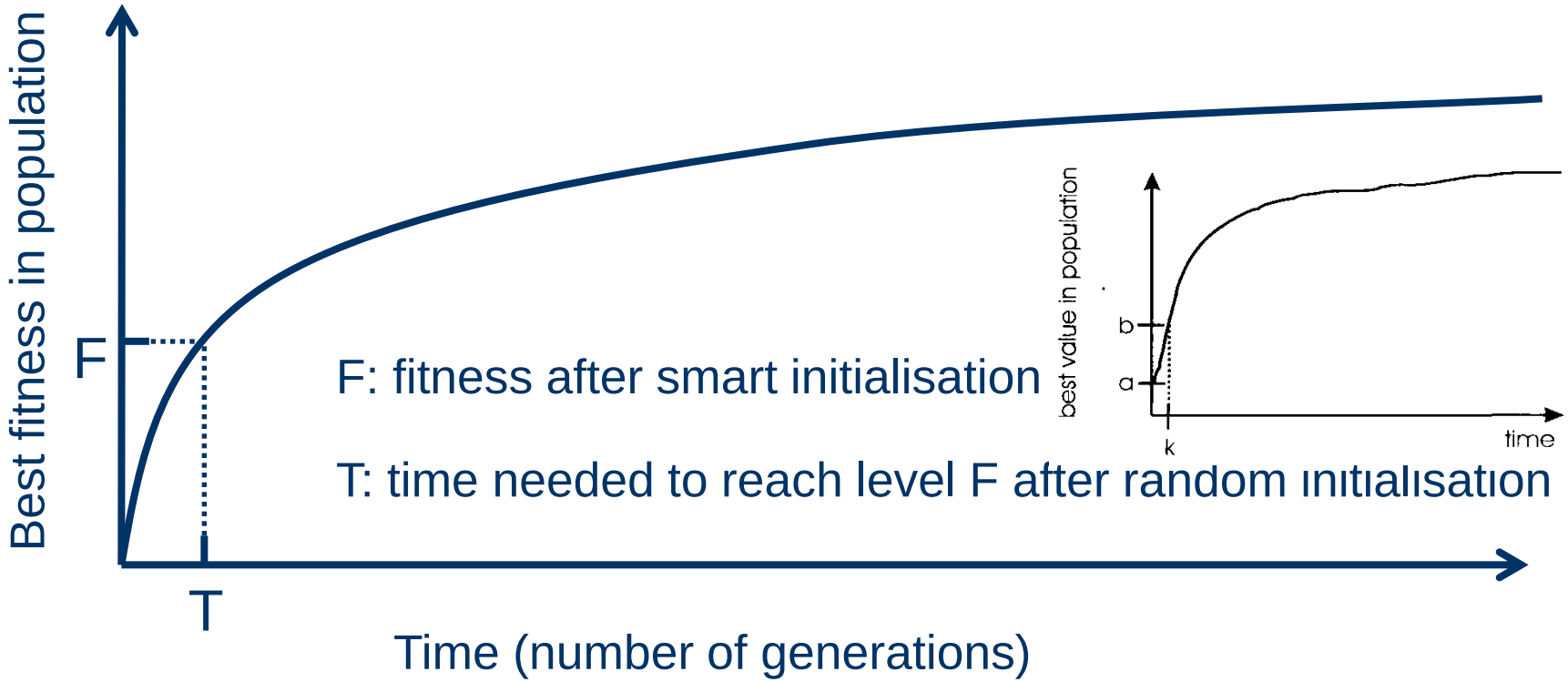
Typical run of an EA shows so-called “anytime behavior”

Are long runs beneficial?



- Answer:
  - it depends how much you want the last bit of progress
  - it may be better to do more shorter runs

Is it worth expending effort on smart initialisation?

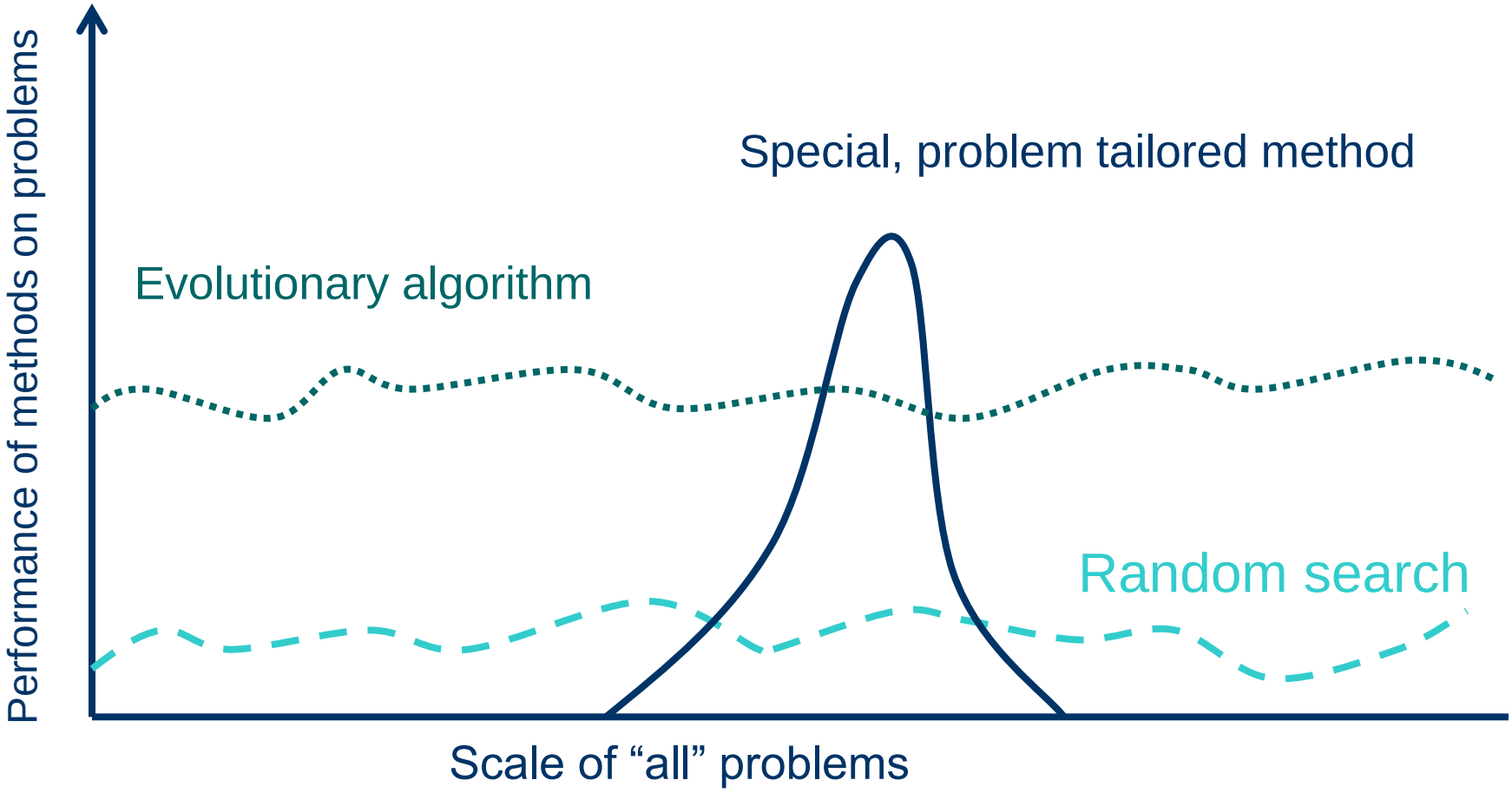


- Answer : it depends:
  - possibly, if good solutions/methods exist.
  - care is needed, see chapter on hybridisation

Evolutionary Algorithms in Context

- There are many views on the use of EAs as robust problem solving tools
- 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 instances
- Goal is to provide robust tools that provide:
  - evenly good performance
  - over a range of problems and instances

EAs as problem solvers: Goldberg's 1989 view

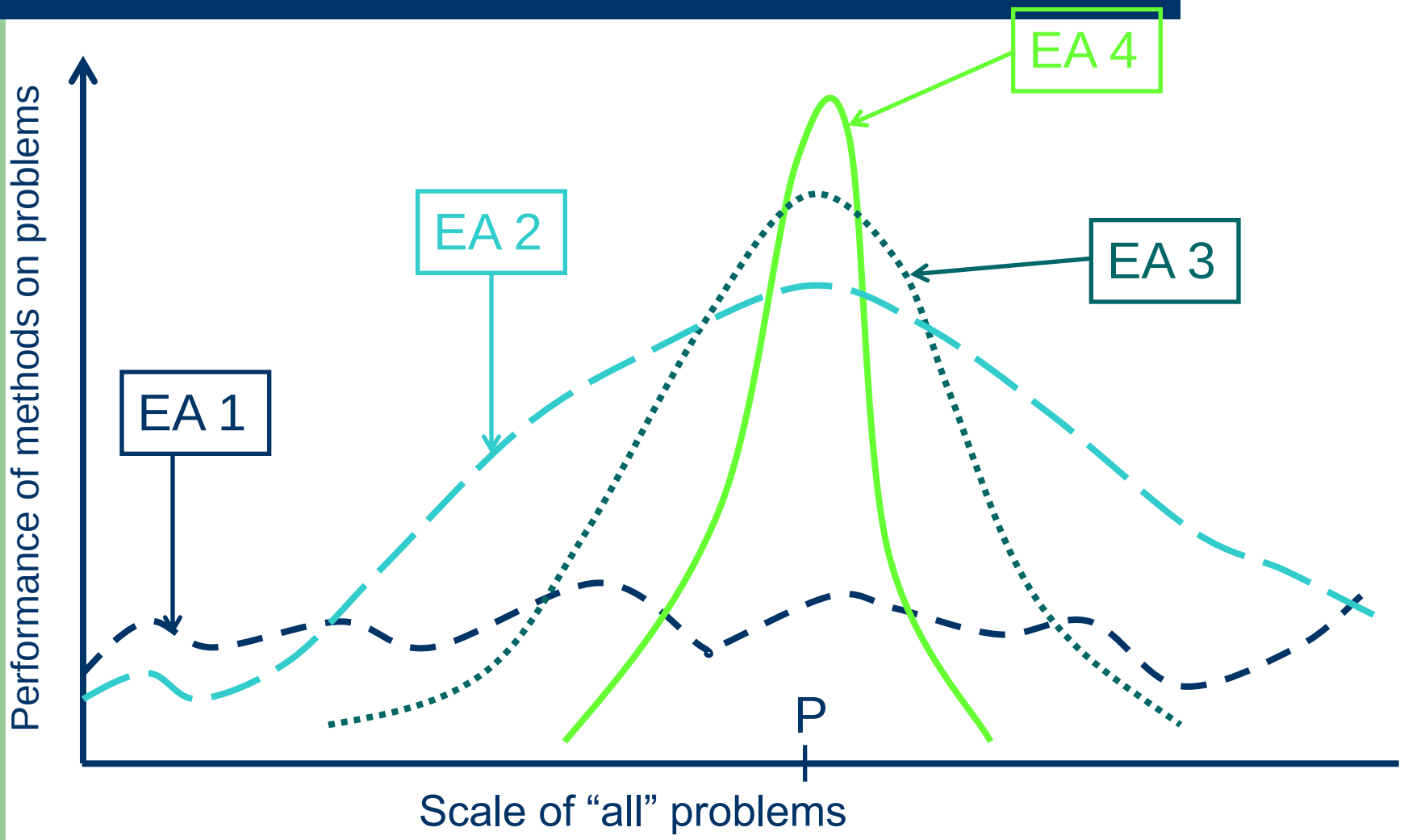


EAs and domain knowledge

- Trend in the 90's:
  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

- Recent theory suggests the search for an "all-purpose" algorithm may be fruitless

Michalewicz' 1996 view



EC and Global Optimisation

- We will use the term global optimisation to refer to the process of attempting to find the solution x* out of a set of possible solutions S that has the optimal value for some fitness function f.
- Deterministic approaches
  - e.g. box decomposition (branch and bound etc)
  - Guarantee to find x^*, but may run in super-polynomial time

- Heuristic Approaches (generate and test)
  - rules for deciding which $x \in S$ to generate next
  - no guarantees that best solutions found are globally optimal

EC and Neighbourhood Search

- Many heuristics impose a neighbourhood structure on *S*
- Such heuristics may guarantee that best point found is *locally optimal* e.g. Hill-Climbers:
  - **But** problems often exhibit many local optima
  - Often very quick to identify good solutions
- EAs are distinguished by:
  - Use of population,
  - Use of multiple, stochastic search operators
  - Especially variation operators with arity >1
  - Stochastic selection