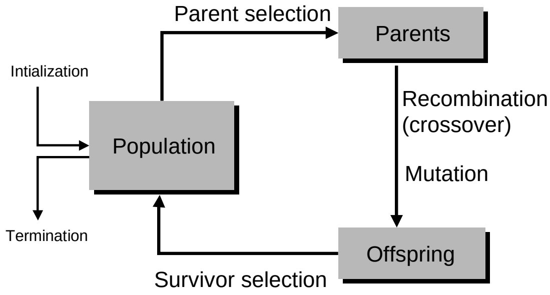
What is an Evolutionary Algorithm?

- The evolutionary mechanism and its components
- Examples: 8-queens problem, knapsack problem
- Working of an evolutionary algorithm
- The power of EC
- Positioning of EC w.r.t. other fields

The evolutionary mechanism: the main cycle



The evolutionary mechanism: the two pillars

There are two competing forces active

- Increasing population diversity by genetic operators
  - mutation
  - recombination

Push towards novelty

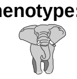
- Decreasing population diversity by selection
  - of parents
  - of survivors

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:

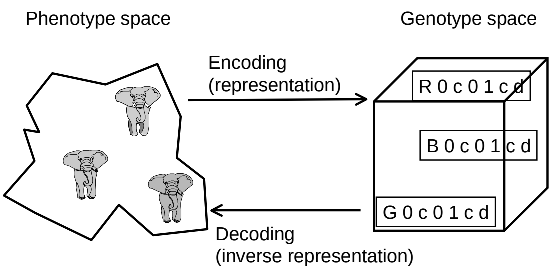


genotype:

a d c a a c b

The link between these levels is called representation

Components: representation / individiuals (2)



Components: representation / individuals (3)

- Search takes place in the genotype space
- Evaluation takes place in the phenotype space
  - *Repr: Phenotypes → 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

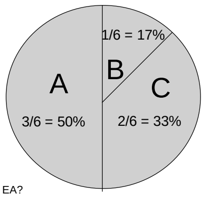
Components: selection

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

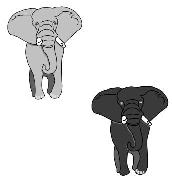


Components: Mutation

Role: causes small (random) variance

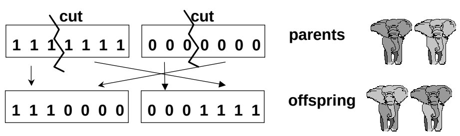
before 1 1 1 1 1 1 1

after 1 1 1 0 1 1 1



Components: Recombination

Role: combines features from different sources



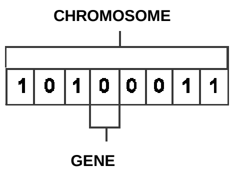
Example 1: The Knapsack Problem

- The problem is to choose which items to take in a knapsack
- Each item has a weight and a value
- We want to maximise the value of the items in the knapsack, without exceeding some maximum weight

  Note: This is not the best way to solve this problem with an evolutionary algorithm and an evolutionary algorithm is not the best way to solve this problem!

Example 1: the knapsack problem
Representation

- An array of bits - one for each item in the knapsack
- A “1” means - take the item
- A “0” mean don’t take the item



Create 100 random bit strings for the initial population

Example 1: the knapsack problem
Fitness evaluation

- Add up the value of the items in the knapsack to give the fitness.
- If the knapsack is overweight, then subtract from the fitness the amount overweight.

Example 1: the knapsack problem
Parent selection

To choose one parent:

- Choose two chromosomes randomly from the population.
- Whichever has the highest fitness is the parent.

Example 1: the knapsack problem
Mutation

Give each gene a small chance of flipping – say 1/(length of string)

0	1	0	0	1	1	1	1

→

0	1	0	0	0	1	1	1

Example 1: the knapsack problem
Recombination

For each gene choose randomly whether to take it from one chromosome or the other

Parents

0	1	1	0	0	0	1	1
1	1	0	0	1	1	1	0

→

Child

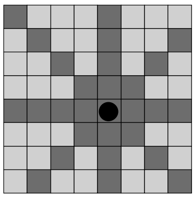
0	1	0	0	1	1	1	1

Example 1: the knapsack problem
Replacement or survivor selection

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

- Choosing two chromosomes randomly from the population.
- Killing whichever has the lowest fitness.

Example 2: the 8 queens problem

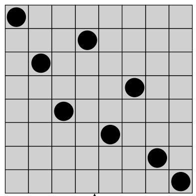


Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other

Example 2: 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

Example 2: 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

Example 2: the 8 queens problem

Parent selection

Roulette wheel selection, for instance

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

Example 2: 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 → 1 3 7 2 6 4 5 8

Example 2: 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 → 1 3 5 4 2 8 7 6
8 7 6 | 5 4 3 2 1 → 8 7 6 2 4 1 3 5

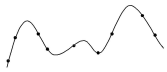
Example 2: the 8 queens problem
Replacement or survivor selection

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

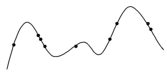
– sorting the whole population by decreasing fitness
– enumerating this list from high to low
– replacing the first with a fitness lower than the given child

Working of an evolutionary algorithm: phases

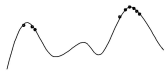
Phases in optimizing on a 1-dimensional fitness landscape



Early phase:
quasi-random population distribution

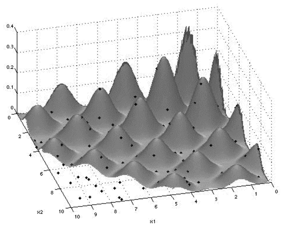


Mid-phase:
population arranged around/on hills



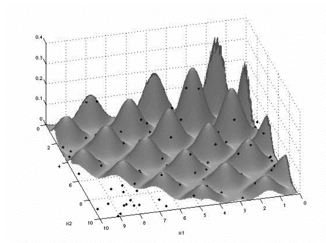
Late phase:
population concentrated on high hills

Working of an evolutionary algorithm demo: searching a fitness landscape 1



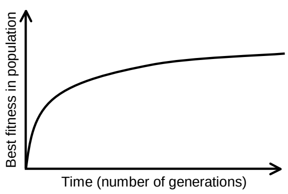
Population dynamics without niching (standard)

Working of an evolutionary algorithm demo: searching a fitness landscape 2



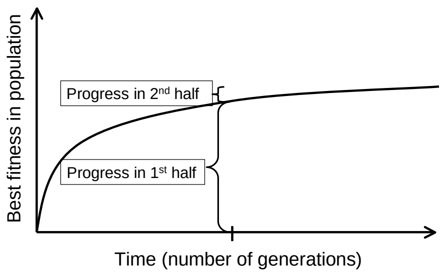
Population dynamics with niching

Working of an evolutionary algorithm: typical run

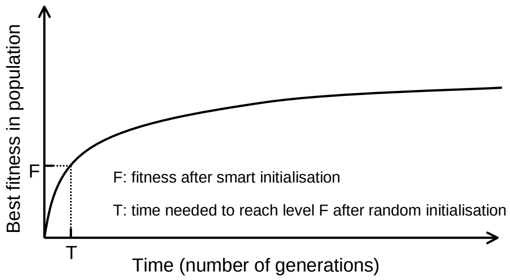


Typical run of an EA shows so-called “anytime behavior”

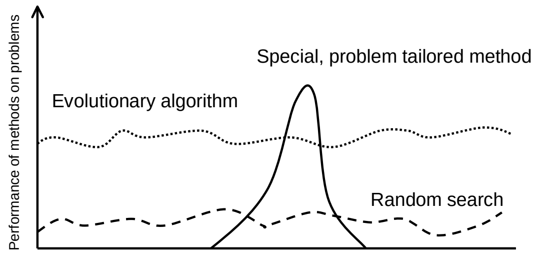
Working of an evolutionary algorithm: long runs might not pay off



Working of an evolutionary algorithm: smart initialisation might not pay off



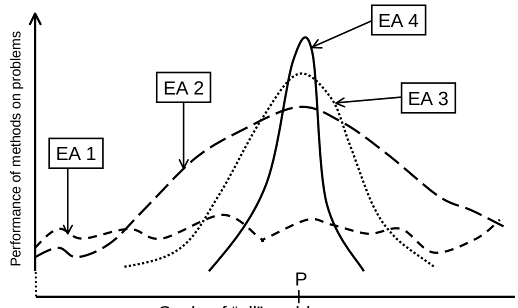
Performance of EA's: Goldberg'89 view



Performance of EAs

- 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

Performance of EA's: Michalewicz'96 view



The power of EC

- Acceptable performance at acceptable costs on a wide range of problems
- Intrinsic parallelism (robustness, fault tolerance, straightforward parallel implementations)
- Superior to other techniques on complex problems with one or more of the following features:
  - lots of data, many free parameters
  - complex relationships between parameters
  - many local optima
  - noisy data
  - changing conditions (dynamic fitness landscape)

The power of EC: advantages

- 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

The power of EC: disadvantages

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

The power of EC: existing evidences

- Academic comparisons between algorithms
- List of successful applications
  (NASA, Unilever, Ford, Daimler-Chrysler, British Telecom, Nortel, AEGON, Rabobank)
- Impressive examples
  (coffee-blend, Mondriaan, chess-playing)
- Ultimate challenge: EC vs. man

The power of EC: EC vs. man

Some criteria when an evolved result can be seen as competitive

- patented, improvement over a patent, or would qualify as patentable
- equal or better than a result published in a peer-reviewed journal
- wins a competition of human or human-written program contestants

The power of EC: EC vs. man

Koza et al. in GPEH journal, vol. 1, nr. 1, 2000

Evolutionary algorithm “bred” solutions in circuit design equal or better than:

- US patent 1 227 113
- US patent 1 958 742
- US patent 2 282 726
- US patent 2 663 806

Revolution in the computer?

No brains needed for problem solving

- Knowledge → selection
- Reasoning → random variation

Selection – variation cycle is fundamental

Everything evolves, or can be made evolvable

EAs vs. other search methods

EAs are global search methods (not necessarily optimizers)

EAs have a unique combination of distinguishing features:

- Stochastic – save computational efforts by rolling dice
- Population based – to diversify search
- Heuristic – an estimated quality measure (fitness) drives the search
- Applying special search operators – crossover combines two or more solutions

EAs show adaptive behavior recognizing and propagating strong (gene) patterns

Exploration ↔ Exploitation

EAs vs. random search

Evolution is often misinterpreted as blind random search (Monte-Carlo method)

Take the problem of finding a pre-selected binary string: $\bar{a}^* \in IB^L = \{0, 1\}^L$

Monte Carlo (MC) Algorithm:
1 k := 1;
2 Randomly generate a binary string $\bar{a}_k$;
3 IF ($\bar{a}_k \neq \bar{a}^*$) THEN
{ k := k + 1;
goto 2;}

EAs vs. random search

- Monte-Carlo method performs worse then enumeration: exponential # trials to hit a solution (for setup …)
- Reason: identical strings may be tested repeatedly.
- For the genetic information of human beings:
  - $L = 10^9$
  - 4 different symbols
  - 4^L variants
- "Conclusion":

Natural evolution was not blind random search

EAs vs. random search

(1+1) Evolutionary algorithm
1 k := 1; randomly generate a binary string $\bar{a}_k$;
2 Create a copy $\bar{a}'_k$ of $\bar{a}_k$;
3 Invert every bit of $\bar{a}'_k$ with probability p;
4 IF ($\bar{a}'_k$ matches $\bar{a}^*$ in more bits than $\bar{a}_k$) THEN
{
$\bar{a}_{k+1} := \bar{a}'_k$;
} ELSE {
$\bar{a}_{k+1} := \bar{a}_k$;
}
5 IF ($\bar{a}_{k+1} \neq \bar{a}^*$) THEN
{
k := k + 1;
goto 2;
}

EAs vs. random search

Analysis

Assume exactly *m* positions are still wrong.

- Probability to preserve all correct bits under step 3: $(1-p)^{L-m}$
- Probability to improve exactly one of the wrong bits: $mp \cdot (1-p)^{m-1}$
- Thus, the chance for an improvement:

$$P\{a'_k \text{ is better then } a_k\} \geq mp \cdot (1-p)^{m-1} \cdot (1-p)^{L-m} = mp \cdot (1-p)^{L-1}$$

- Thus, the expected number of iterations until an improvement occurs:

$$E_{1-\text{bit impr.}} \leq \frac{1}{mp \cdot (1-p)^{L-1}}$$

- Equivalently, for the expected total number of iterations, this implies:

$$E_{iter.}(L) = \sum_{m=1}^{l} E_{1-\text{bit impr.}} \leq \frac{1}{p \cdot (1-p)^{L-1}} \cdot \sum_{m=1}^{l} \frac{1}{m} \approx \frac{1}{p \cdot (1-p)^{L-1}} \cdot \ln L$$

(assuming we need only *L* one-bit improvements).

EAs vs. random search

Thus: $$E_{iter}(L) \approx \frac{1}{p \cdot (1-p)^{L-1}} \cdot \ln L$$

For *p = 1 / L*: $$E_{iter}(L) \approx \frac{1}{\frac{1}{L} \cdot (1-\frac{1}{L})^{L-1}} \cdot \ln L$$

With $$\lim_{L\to\infty}(1-\frac{1}{L})^{L-1} = \frac{1}{e}$$

$$\boxed{E_{iter}(L) \approx L \cdot e \cdot \ln L}$$

EAs vs. random search

- The analysis of algorithm 2 is oversimplified:
  - Only one-bit mutations
  - Only improving mutation
  - Only an upper bound on *E*(*L*)
  - We can assume to start with *L*/2 correct bits
- Evolution-like algorithms are logarithmic, not exponential, concerning their running time (for this simple example)
- Conclusion:

EA ≠ MC

Note: complex problems and/or EAs: no analysis for E_{iter}

EC and other fields

- EC vs. NN
  NN: adaptation on individual level (learning)
  EC: adaptation on population level
- EC vs. OR
  OR: approximate model, exact solution
  EC: exact model, approximate solution

Is EC part of Artificial Intelligence?

- Answer 1: the question is irrelevant
- Answer 2: yes, it is part of “nouvelle AI”
- Answer 3: no, see arguments on next slides

EC is part of Computational Intelligence

COMPUTATIONAL INTELLIGENCE or SOFT COMPUTING

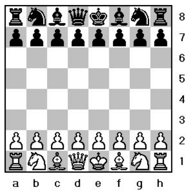
- **Neuro Computing**
- **Evolutionary Computing**
- **Fuzzy Computing**

Newest umbrella term: natural computing
- evolutionary computing
- neuro-computing
- DNA computing
- quantum computing, …

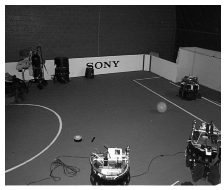
CI and AI are different

Each field has a different “canonical problem”

AI: computer chess



CI: robot soccer



CI and AI are different

AI: computer chess	CI: robot soccer
• static	• dynamic
• turn system	• real-time
• complete information	• incomplete information
• symbolic	• non-symbolic
• central control	• distributed system
• virtual space	• grounded in physical reality

Is EC part of Artificial Intelligence?

Classical AI:
- symbolic knowledge representation
- top down with imposing structures based on analysis
- main problem solving paradigm: reasoning

EC:
- numerical (sub-symbolic) knowledge representation
- ordered structure(s), solution(s) emerge bottom-up
- main problem solving paradigm: trial-and-error