What is an Evolutionary Algorithm?

Chapter 2

Contents

- Traditional EA
- 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

Traditional EA

- Combined view <= Today
- 3 traditional schools of EA, which now has evolved into one (EC) and/or many!
  - Genetic Algorithms
    - Week 3
  - Evolutionary Strategies
    - **Self-study (Week 4)**
  - Evolutionary Programming
    - Week 3

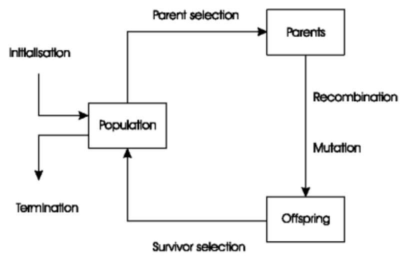
Recap of EC metaphor

- 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 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
```

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

Representations

- 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)
- 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

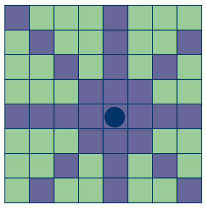
Survivor Selection

- 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



8x8
40320
Combinations
(~112 hours)

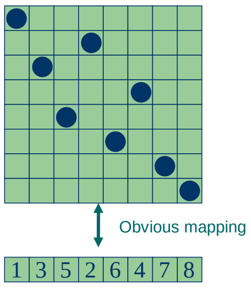
16x16
~21x10^12
Combinations
(~58x10^18 hours)

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

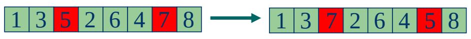


8 Queens Problem: Fitness evaluation

- 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

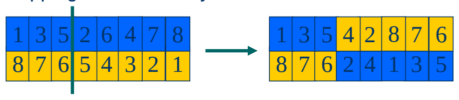
The 8 queens problem: Mutation

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



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

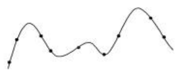
8 Queens Problem: summary

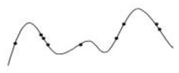
Representation	Permutations
Recombination	"Cut-and-crossfill" crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition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

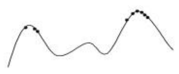
Note that is is ***only one possible***
set of choices of operators and parameters

Typical behaviour of an EA

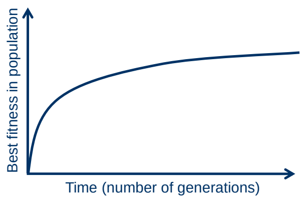
Phases in optimising on a 1-dimensional fitness landscape


Early phase:
quasi-random population distribution


Mid-phase:
population arranged around/on hills

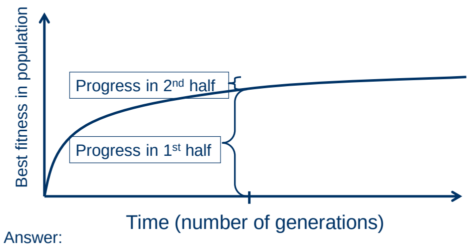

Late phase:
population concentrated on high hills

Typical run: progression of fitness



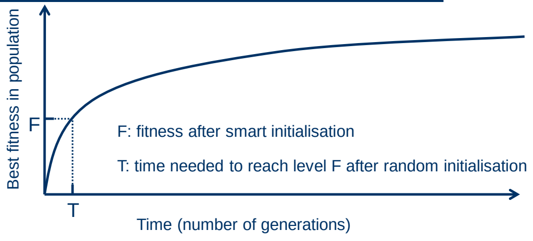
Typical run of an EA shows so-called “anytime behavior”

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?

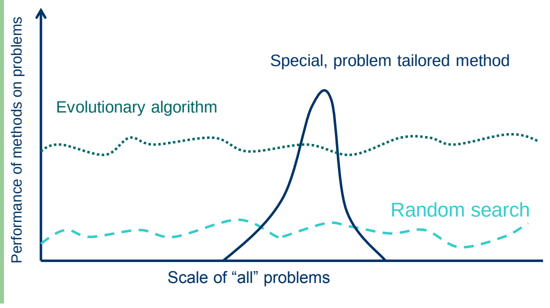


- Answer : it depends:
  - possibly, if good solutions/methods exist.
  - care is needed, see chapter on hybridisation

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

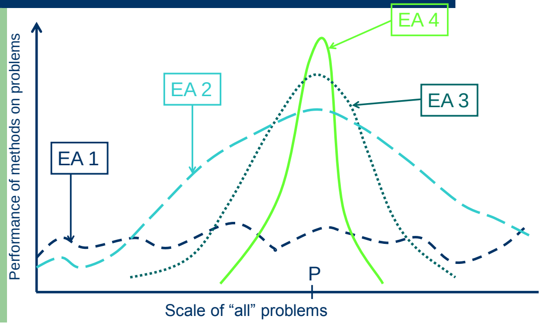
EAs as problem solvers: Goldberg's 1989 view



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

- 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^* , 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