Reminder of previous slides

September 12, 2025 2:49 PM

EA 1. Start with a representation

we can assume a set of real number parameters that optimise a real problem

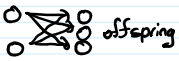
lets say a_1 a_2 a_3 a_4

go together

[, , ,]

[ ][ ][ ][ ]

2. variation operations

- mutation O → O
- crossover  offspring

3. Fitness Evaluation

- fitness comes from the problem

Now that we have our "Structure" we initialize with weights

parameters include:

1 - population (keep it big)
↑ doesn't have to be constant

2 - fitness doesn't have to be positive or minimum, hell it doesn't even need to be normalized.

3 - Termination criteria
- goal reached
- stagnation
- Exhaustion of resources

4 - Select Parents
- generate offspring
- evaluate offsprings

5 - Survivor Selection =
≤ {Population, offspring}

Lec1-What_is_an_EA-2025

Evolutionary Computing

Chapter 3

Recap of EC metaphor (1/2)

- 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

1 / 41

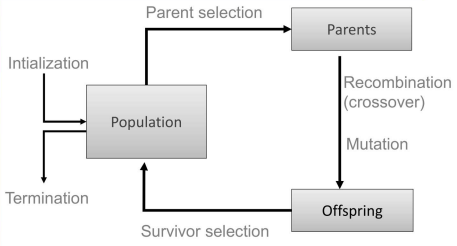
Recap of EC metaphor (2/2)

- 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

Chapter 3: What is an Evolutionary Algorithm?

- Scheme of an EA
- Main EA components:
  - Representation / evaluation / population
  - Parent selection / survivor selection
  - Recombination / mutation
- Examples: eight-queens problem
- Typical EA behaviour
- EAs and global optimisation
- EC and neighbourhood search

Scheme of an EA: General scheme of EAs



Scheme of an EA: EA scheme in pseudo-code

```
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
```

Scheme of an EA: Common model of evolutionary processes

- Population of individuals
- Individuals have a fitness
- Variation operators: crossover, mutation
- Selection towards higher fitness
  - "survival of the fittest" and
  - "mating of the fittest"

Neo Darwinism:
Evolutionary progress towards higher life forms
=
Optimization according to some fitness-criterion
(optimization on a fitness landscape)

Scheme of an EA: Two pillars of evolution

There are two competing forces

Increasing population **diversity** by genetic operators
- mutation
- recombination

Push towards **novelty**

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

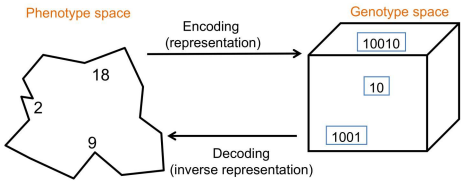
Push towards **quality**

Main EA components: Representation (1/2)

- Role: provides code 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")
- Implies two mappings:
  - 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)

Main EA components: Representation (2/2)

Example: represent integer values by their binary code



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

Main EA components: Evaluation (fitness) function

- Role:
  - Represents the task to solve, the requirements to adapt to (can be seen as "the environment")
  - Enables selection (provides basis for comparison)
  - e.g., 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
- 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

Main EA components: Population (1/2)

- 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

Main EA components: Population (2/2)

- 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)

Main EA components: Selection mechanism (1/3)

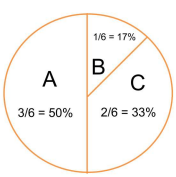
Role:

- Identifies individuals
  - to become parents
  - to survive
- Pushes population towards higher fitness
- Usually probabilistic
  - high quality solutions more likely to be selected than low quality
  - but not guaranteed
  - even worst in current population usually has non-zero probability of being selected
- This *stochastic* nature can aid escape from local optima

Main EA components: Selection mechanism (2/3)

Example: roulette wheel selection

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

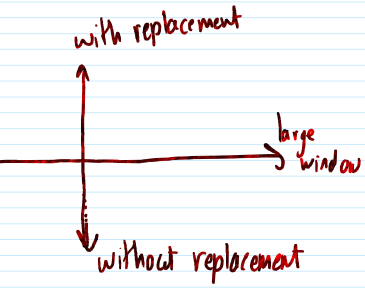
- Every EA has a selection mechanism
- Acts at the parent level or for survivors (depending if we are doing parent or survivor selection)
- Fitness proportional selection: select based off fitness functions
- tournament selection, pick randomly say 3 then choose them or rank them depending on what point this is acting on

Replacement is when we put back in the pot

☆ How does the size of the window and replacement effect the selection process] 100% a question

- small window size and replacement is tournament selection
- if the window size is the whole population then it is fitness proportional
- understand the effect of replacement on tournament selection

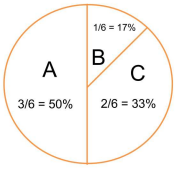
small window


with replacement
large window
without replacement

Main EA components: Selection mechanism (2/3)

Example: roulette wheel selection

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 (3/3)

- 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 (while parent selection is usually stochastic)
  - Fitness based : e.g., 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)

Main EA components: Variation operators

- Role: 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
  - Arity > 2 is formally possible, 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

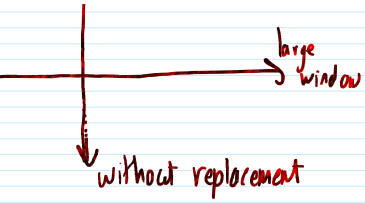
Main EA components: Mutation (1/2)

- Role: causes small, random variance
- 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 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

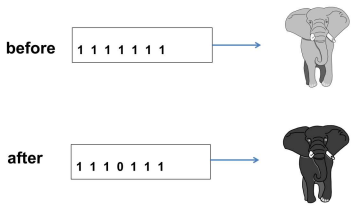
D How does the size of the window and replacement effect the selection process] 100% a question

small window

- small window size and replacement is tournament selection
- if the window size is the whole population then it is fitness proportional
- understand the effect of replacement on tournament selection


large
window
without replacement

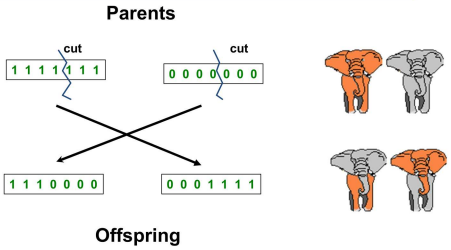
Main EA components: Mutation (2/2)



Main EA components: Recombination (1/2)

- Role: 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

Main EA components: Recombination (2/2)



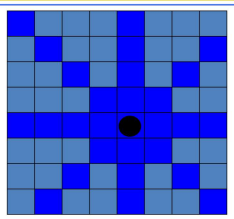
Main EA components: Initialisation / Termination

- 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

Main EA components: What are the different types of EAs

- Historically different flavours of EAs have been associated with different data types to represent solutions
  - 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

Example: The 8-queens problem

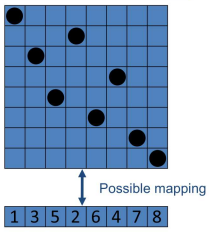


Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other

The 8-queens problem: Representation

Phenotype:
a board configuration

Genotype:
a permutation of the numbers 1–8



The 8-queens problem: Fitness evaluation

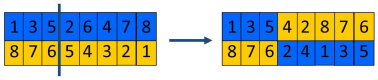
- Penalty of one queen: the number of queens she can check
- Penalty of a configuration: the sum of penalties of all queens
- Note: penalty is to be minimized
- Fitness of a configuration: inverse penalty to be maximized

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

The 8-queens problem: Summary

Dont need but good example

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 ***only one possible***
set of choices of operators and parameters

Typical EA behaviour: Stages

Stages in optimising on a 1-dimensional fitness landscape

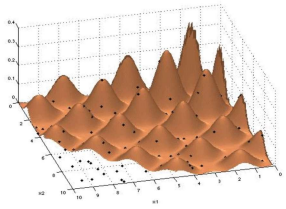
Early stage:
quasi-random population distribution

Mid-stage:
population arranged around/on hills

Late stage:
population concentrated on high hills

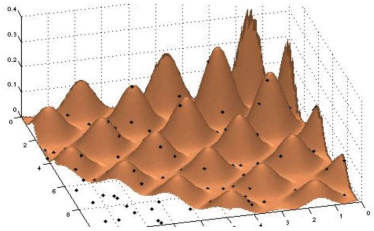
Typical EA behaviour: Working of an EA demo (1/2)

Searching a fitness landscape without “niching”



Typical EA behaviour: Working of an EA demo (2/2)

Searching a fitness landscape with “niching”

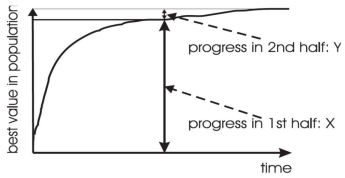


Typical EA behaviour: Typical run: progression of fitness

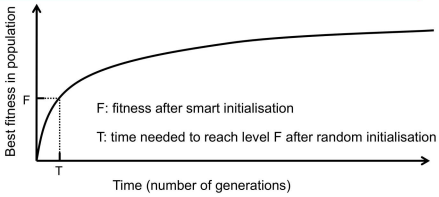
Typical run of an EA shows so-called “anytime behavior”

Typical EA behaviour: Are long runs beneficial?

- Answer:
  - It depends on how much you want the last bit of progress
  - May be better to do more short runs



Typical EA behaviour: Is it worth expending effort on smart initialisation?

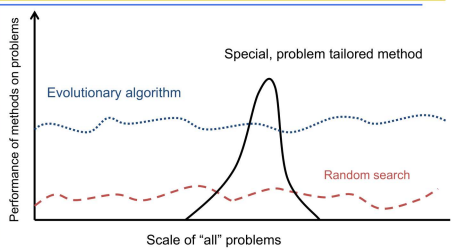


- Answer: it depends.
  - Possibly good, if good solutions/methods exist.
  - Care is needed, see chapter/lecture on hybridisation.

Typical EA behaviour: 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

Typical EA behaviour: EAs as problem solvers: Goldberg view (1989)



There will always be a question about how to maintain diversity

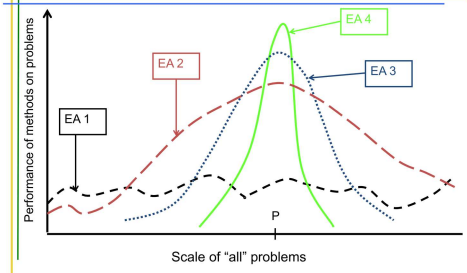
↓rest was skipped

Typical EA behaviour: 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

Typical EA behaviour: EAs as problem solvers: Michalewicz view (1996)



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^*,
  - May have bounds on runtime, usually super-polynomial
- Heuristic Approaches (generate and test)
  - rules for deciding which $x \in S$ to generate next
  - no guarantees that best solutions found are globally optimal
  - no bounds on runtime

- "I don't care if it works as long as it converges"
  vs.
- "I don't care if it converges as long as it works"

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

- Question: what is the neighbourhood in an EA?

Lecture 2 Slides
September 12, 2025 2:45 PM

Lec2-Representation_Mutation_Recombination-2025

Evolutionary Computing

Chapter 4

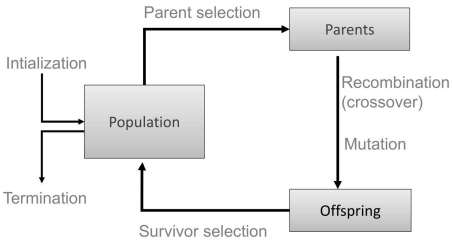
Chapter 4: Representation, Mutation, and Recombination

- Role of representation and variation operators
- Most common representation of genomes:
  - Binary
  - Integer
  - Real-Valued or Floating-Point
  - Permutation
  - Tree

Data doesn't need to be homogeneous or heterogeneous they can be literally anything

1 / 62

Scheme of an EA: General scheme of EAs



2 / 62

Genotype versus Phenotype

- we apply mutations on the genotype + crossover too
- phenotype is what is evaluated for fitness

- initialization is generally done randomly

Mutation involves 1
Crossover involves ≥ 2
Survivor selection is how we choose the offsprings

Basically slide 21&2 is hella important

Role of representation and variation operators

- First stage of building an EA and most difficult one: choose *right* representation for the problem
- Variation operators: mutation and crossover
- Type of variation operators needed depends on chosen representation

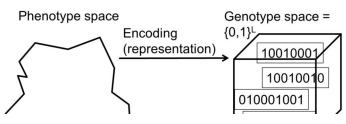
- TSP problem
  - What are possible representations?

3 / 62

★ Given a problem what kind of representation is proper? look into permutation

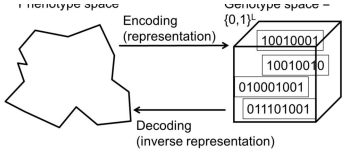
Binary Representation

- One of the earliest representations
- Genotype consists of a string of binary digits



Genotype
↳ where the variation operators act (data structure)

Phenotype



4 / 62

operators act (data structure)

Phenotype ← What we evaluate for fitness

Binary Representation: Mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/ chromosome_length

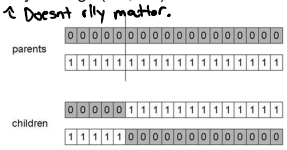
- Mutation can cause variable effect (use gray coding)

← one of the oldest mutations, basically the probability of a bit flip.

5 / 62

Binary Representation: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- ~~P_c typically in range (0.6, 0.9)~~ ↑ Doesn't rly matter.



two weaknesses
1. parts at the two ends will never be kept together in a child.
2. keeps parts that are close to each other together.

Probability of mutation is on the bit itself
Probability of crossover is on chunks of bits

7 / 62

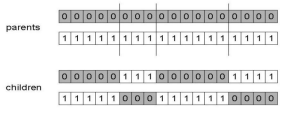
Binary Representation: Alternative Crossover Operators

- Why do we need other crossover(s)? → fix the weaknesses we already mentioned
- Performance with 1-point crossover depends on the order that variables occur in the representation
  - More likely to keep together genes that are near each other
  - Can never keep together genes from opposite ends of string
  - This is known as Positional Bias
  - Can be exploited if we know about the structure of our problem, but this is not usually the case

8 / 62

Binary Representation: n-point crossover

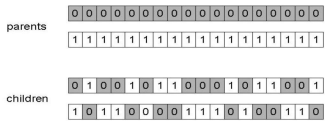
- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1-point (still some positional bias)



Study 1 pt, 2 pt ... n point crossover
dont confuse with uniform.

Binary Representation: Uniform crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position



9 / 62

Weakness:
1. the likelyhood that you keep things that are very close to each other is very unlikely

Binary Representation: Crossover OR mutation? (1/3)

- Decade long debate: which one is better / necessary / main-background
- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role
  - mutation-only-EA is possible, xover-only-EA would not work

10 / 62

☆ Do you think that for a binary representation a single point or n point crossover is sufficient to search the space of possible solutions?
↑ I think the answer is the edges

Binary Representation: Crossover OR mutation? (2/3)

↓ Searching the solution space

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

[Optimising, we want to maximise fitness

There is co-operation AND competition between them

↓ Crossover later on can be more exploitative

• Crossover is explorative, it makes a *big* jump to an area somewhere "in between" two (parent) areas

• Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

[depends on step size, with a high mutation rate it could be used for searching spaces

11 / 62

Binary Representation: Crossover OR mutation? (3/3)

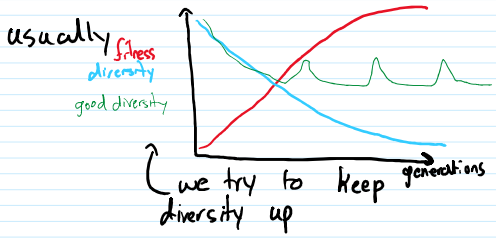
- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, ?% after performing n crossovers)
- To hit the optimum you often need a 'lucky' mutation

12 / 62

Exploration Versus Exploitation

mutation { small to optimise
large to introduce diversity → Takes you out of a subopti

crossover { effective as long as you have diversity



[we try to keep diversity up

* maintaining diversity is critical.

mal position 3

Integer Representation

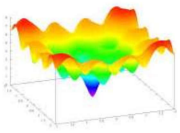
- Nowadays it is generally accepted that it is better to encode numerical variables directly (integers, floating point variables)
- Some problems naturally have integer variables, e.g. image processing parameters
- Others take categorical values from a fixed set e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make
  - "creep" i.e. more likely to move to similar value
    - Adding a small (positive or negative) value to each gene with probability p
  - Random resetting (esp. categorical variables)
    - With probability p_m a new value is chosen at random
- Same recombination as for binary representation

Real-Valued or Floating-Point Representation

maybe? *

- Many problems occur as real valued problems, e.g. continuous parameter optimisation $f: \mathbb{R}^n \rightarrow \mathbb{R}$
- Illustration: Ackley's function (often used in EC)

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n}\cdot\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$



ok we need to know how to do integer and floating point mutation and crossover. the names and all

Real-Valued or Floating-Point Representation: Mapping real values on bit strings

$z \in [x,y] \subseteq \mathbb{R}$ represented by $\{a_1,\ldots,a_L\} \in \{0,1\}^L$

- $[x,y] \rightarrow \{0,1\}^L$ must be invertible (one phenotype per genotype)
- $\Gamma: \{0,1\}^L \rightarrow [x,y]$ defines the representation

$$\Gamma(a_1,\ldots,a_L) = x + \frac{y-x}{2^L-1}\cdot\left(\sum_{j=0}^{L-1} a_{L-j}\cdot 2^j\right) \in [x,y]$$

- Only 2^L values out of infinite are represented
- L determines possible maximum precision of solution
- High precision → long chromosomes (slow evolution)

Real-Valued or Floating-Point Representation: Uniform Mutation

- General scheme of floating point mutations

$$\bar{x} = \langle x_1, \ldots, x_l \rangle \rightarrow \bar{x}' = \langle x_1', \ldots, x_l' \rangle$$

$$x_i, x_i' \in [LB_i, UB_i]$$

- Uniform Mutation

  x_i' drawn randomly (uniform) from $[LB_i, UB_i]$

- Analogous to bit-flipping (binary) or random resetting (integers)

Real-Valued or Floating-Point Representation: Nonuniform Mutation

- Non-uniform mutations:
  - Many methods proposed, such as time-varying range of change etc.
  - Most schemes are probabilistic but usually only make a small change to value
  - Most common method is to add random deviate to each variable separately, taken from N(0, σ) Gaussian distribution and then curtail to range
    $x'_i = x_i + N(0,\sigma)$
  - Standard deviation σ, *mutation step size*, controls amount of change (2/3 of drawings will lie in range (- σ to + σ))

Real-Valued or Floating-Point Representation: Self-Adaptive Mutation (1/2)

- Step-sizes are included in the genome and undergo variation and selection themselves: $\langle x_1,\ldots,x_n, \sigma \rangle$
- Mutation step size is not set by user but coevolves with solution
- Different mutation strategies may be appropriate in different stages of the evolutionary search process.

Real-Valued or Floating-Point Representation: Self-Adaptive Mutation (2/2)

- Mutate σ first
- Net mutation effect: $\langle x, \sigma \rangle \rightarrow \langle x', \sigma' \rangle$
- Order is important:
  - first $\sigma \rightarrow \sigma'$ (see later how)
  - then $x \rightarrow x' = x + N(0,\sigma')$
- Rationale: new $\langle x', \sigma' \rangle$ is evaluated twice
  - Primary: x' is good if f(x') is good
  - Secondary: σ' is good if the x' it created is good
- Reversing mutation order this would not work

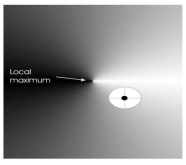
Real-Valued or Floating-Point Representation: Uncorrelated mutation with one σ (1/2)

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma' \rangle$ ← mutation step size
  - $\sigma' = \sigma \cdot \exp(\tau \cdot N(0,1))$
  - $x'_i = x_i + \sigma' \cdot N_i(0,1)$
- Typically the "learning rate" $\tau \propto 1/\sqrt{n}$
- And we have a boundary rule $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$

These are always asked about - self adaptive step sizes, look at these stuff and all it's kinds.

Real-Valued or Floating-Point Representation: Uncorrelated mutation with one σ (2/2)

Mutants with equal likelihood



Circle: mutants having the same chance to be created

just agree with the

Circle: mutants having the same chance to be created

just agree with the evolutionary vibes

62

Real-Valued or Floating-Point Representation: Uncorrelated mutation with n σ's (1/2)

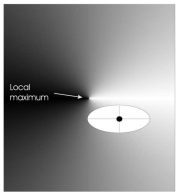
- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n \rangle$
  - $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
  - $x'_i = x_i + \sigma'_i \cdot N_i(0,1)$
- Two learning rate parameters:
  - τ' overall learning rate
  - τ coordinate wise learning rate
- $\tau' \propto 1/(2\,n)^{½}$ and $\tau \propto 1/(2\,n^{½})^{½}$
- Boundary rule: $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$

← learning rate feels like what we do in ML

22 / 62

Real-Valued or Floating-Point Representation: Uncorrelated mutation with n σ's (2/2)

Mutants with equal likelihood



Ellipse: mutants having the same chance to be created

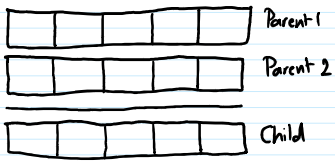
23 / 62

Real-Valued or Floating-Point Representation: Crossover operators

- Discrete: Weighing both the same
  - each allele value in offspring *z* comes from one of its parents *(x,y)* with equal probability: $z_i = x_i$ or y_i
  - Could use n-point or uniform
- Intermediate
  - exploits idea of creating children "between" parents (hence a.k.a. *arithmetic* recombination)
  - $z_i = \alpha\, x_i + (1 - \alpha)\, y_i$ where $\alpha : 0 \le \alpha \le 1$.
  - The parameter α can be:
    - constant: uniform arithmetical crossover
    - variable (e.g. depend on the age of the population)
    - picked at random every time

24 / 62

Discrete



you need to know the crossover operators & stuff

Real-Valued or Floating-Point Representation: Single arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick a single gene (k) at random,
- child$_1$ is:

$$\langle x_1, \ldots, x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, \ldots, x_n \rangle$$

- Reverse for other child. e.g. with α = 0.5

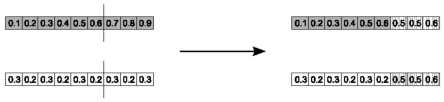
25 / 62

Real-Valued or Floating-Point Representation: Simple arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick a random gene (k) after this point mix values
- $child_1$ is:

$$\left\langle x_1,...,x_k,\alpha \cdot y_{k+1}+(1-\alpha)\cdot x_{k+1},...,\alpha \cdot y_n+(1-\alpha)\cdot x_n \right\rangle$$

- reverse for other child. e.g. with α = 0.5

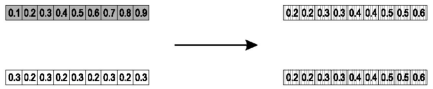


Real-Valued or Floating-Point Representation: Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- $Child_1$ is:

$$a \cdot \bar{x} + (1-a) \cdot \bar{y}$$

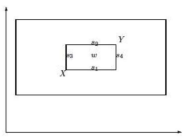
- reverse for other child. e.g. with α = 0.5



Real-Valued or Floating-Point Representation: Blend Crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Assume $x_i < y_i$
- $d_i = y_i - x_i$
- Random sample $z_i = [x_i - \alpha d_i, x_i + \alpha d_i]$
- Original authors had best results with α = 0.5

Real-Valued or Floating-Point Representation: Overview different possible offspring



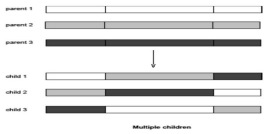
- Single arithmetic:
- $\{s_1, s_2, s_3, s_4\}$
- Simple arithmetic / whole arithmetic:
- inner box (w = alpha 0.5)
- Blend crossover:
- outer box

Real-Valued or Floating-Point Representation: Multi-parent recombination

- Recall that we are not constricted by the practicalities of nature
- Noting that mutation uses n = 1 parent, and "traditional" crossover n = 2, the extension to $n > 2$ is natural to examine
- Been around since 1960s, still rare but studies indicate useful

Real-Valued or Floating-Point Representation: Multi-parent recombination, type 1

- Idea: segment and recombine parents
- Example: diagonal crossover for n parents:
  - Choose n-1 crossover points (same in each parent)
  - Compose n children from the segments of the parents in along a "diagonal", wrapping around



- This operator generalises 1-point crossover

Real-Valued or Floating-Point Representation: Multi-parent recombination, type 2

- Idea: arithmetical combination of (real valued) alleles
- Example: arithmetic crossover for n parents:
  - i-th allele in child is the average of the parents' i-th alleles
- Creates center of mass as child
- Odd in genetic algorithms, long known and used in evolution strategies

Permutation Representations

- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
  - Example: production scheduling: important thing is which elements are scheduled before others (order)
  - Example: Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (adjacency)
- These problems are generally expressed as a permutation:
  - if there are n variables then the representation is as a list of n integers, each of which occurs exactly once

Permutation Representation: TSP example

- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , n
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:
  for 30 cities there are 30! ≈ 10^{32} possible tours

He told us to make sure we study this a lot

Permutations: how you organise a set of finite elements in relation to each other without adding or removing

↑ basically an organisation of a finite set of problems.

Permutation Representation: TSP example

- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:
  for 30 cities there are 30! ≈ 10^{32} possible tours

Permutation Representations: Mutation

- Normal mutation operators lead to inadmissible solutions
  - e.g. bit-wise mutation: let gene *i* have value *j*
  - changing to some other value *k* would mean that *k* occurred twice and *j* no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

Permutation Representations: Swap mutation

- Pick two alleles at random and swap their positions

Permutation Representations: Insert Mutation

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information

He told us to make sure we study this a lot

Permutations: how you organise a set of finite elements in relation to each other without adding or removing

↳ basically an organisation of a finite set of problems.

Permutation Representations: Scramble mutation

- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions

Permutation Representations: Inversion mutation

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information

Permutation Representations: Crossover operators

- "Normal" crossover operators will often lead to inadmissible solutions

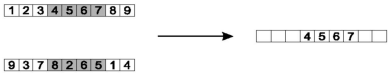
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Permutation Representations: Order 1 crossover (1/2)

- Idea is to preserve relative order that elements occur
- Informal procedure:
  - 1. Choose an arbitrary part from the first parent
  - 2. Copy this part to the first child
  - 3. Copy the numbers that are not in the first part, to the first child:
    - starting right from cut point of the copied part,
    - using the **order** of the second parent
    - and wrapping around at the end
  - 4. Analogous for the second child, with parent roles reversed

Permutation Representations: Order 1 crossover (2/2)

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2

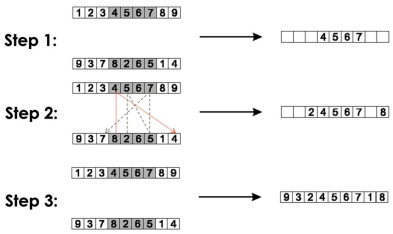
Permutation Representations: Partially Mapped Crossover (PMX) (1/2)

Informal procedure for parents P1 and P2:

1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied *j* in P2, since we know that we will not be putting *j* there (as is already in offspring)
5. If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *i* in the position occupied by *k* in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

Permutation Representations: Partially Mapped Crossover (PMX) (2/2)



Permutation Representations: Cycle crossover (1/2)

Basic idea:

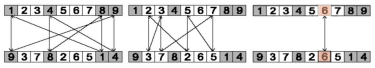
Each allele comes from one parent *together with its position*.

Informal procedure:

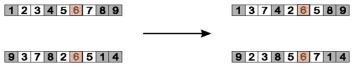
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

Permutation Representations: Cycle crossover (2/2)

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Permutation Representations: Edge Recombination (1/3)

- Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark with a +
- e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Permutation Representations: Edge Recombination (2/3)

Informal procedure: once edge table is constructed

1. Pick an initial element, *entry*, at random and put it in the offspring
2. Set the variable *current element* = *entry*
3. Remove all references to *current element* from the table
4. Examine list for current element:
   - If there is a common edge, pick that to be next element
   - Otherwise pick the entry in the list which itself has the shortest list
   - Ties are split at random
5. In the case of reaching an empty list:
   - a new element is chosen at random

Permutation Representations: Edge Recombination (3/3)

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Choices	Element selected	Reason	Partial result
All	1	Random	[1]
2,5,4,9	5	Shortest list	[1 5]
4,6	6	Common edge	[1 5 6]
2,7	2	Random choice (both have two items in list)	[1 5 6 2]
3,8	8	Shortest list	[1 5 6 2 8]
7,9	7	Common edge	[1 5 6 2 8 7]
3	3	Only item in list	[1 5 6 2 8 7 3]
4,9	9	Random choice	[1 5 6 2 8 7 3 9]
4	4	Last element	[1 5 6 2 8 7 3 9 4]

Trees will not be in the exam = What is a tree representation good for., that's all to study for trees

Tree Representation (1/6)

- Trees are a universal form, e.g. consider
- Arithmetic formula: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program:

```
i =1;
while (i < 20)
{
      i = i +1
}
```

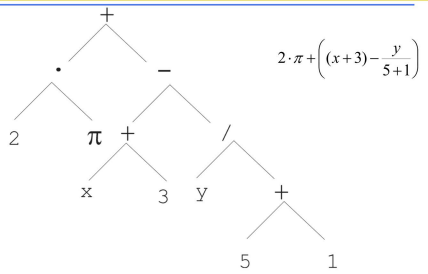
Trees will not be in the exam = What is a tree representation good for?, that's all to study for trees

Tree Representation (1/6)

- Trees are a universal form, e.g. consider
- Arithmetic formula: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program:

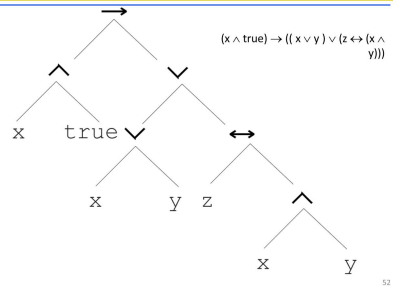
```
i =1;
while (i < 20)
{
    i = i +1
}
```


Tree Representation (2/6)



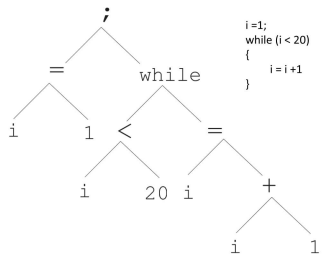
$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$

Tree Representation (3/6)



$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

Tree Representation (4/6)




```
i =1;
while (i < 20)
{
    i = i +1
}
```

Tree Representation (5/6)

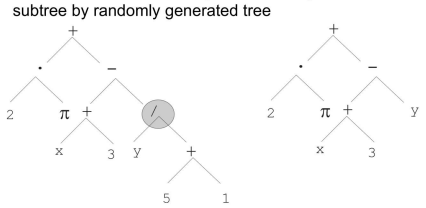
- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations)
- Tree shaped chromosomes are non-linear structures
- In GA, ES, EP the size of the chromosomes is fixed
- Trees in GP may vary in depth and width

Tree Representation (6/6)

- Symbolic expressions can be defined by
  - Terminal set T
  - Function set F (with the arities of function symbols)
- Adopting the following general recursive definition:
  - Every $t \in T$ is a correct expression
  - $f(e_1, \ldots, e_n)$ is a correct expression if $f \in F$, arity(f)=n and $e_1, \ldots, e_n$ are correct expressions
  - There are no other forms of correct expressions
- In general, expressions in GP are not typed (closure property: any $f \in F$ can take any $g \in F$ as argument)

Tree Representation: Mutation (1/2)

- Most common mutation: replace randomly chosen subtree by randomly generated tree



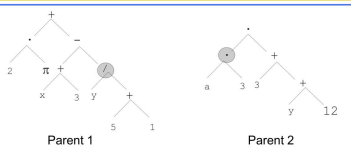
Tree Representation: Mutation (2/2)

- Mutation has two parameters:
  - Probability p_m to choose mutation
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

Tree Representation: Recombination (1/2)

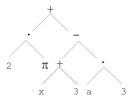
- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents

Tree Representation: Recombination (2/2)

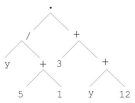


Parent 1

Parent 2



Child 1



Child 2

lecture 3

October 3, 2025 4:23 PM

Lec3-Fitness_Selection_PopulationManagement-2025

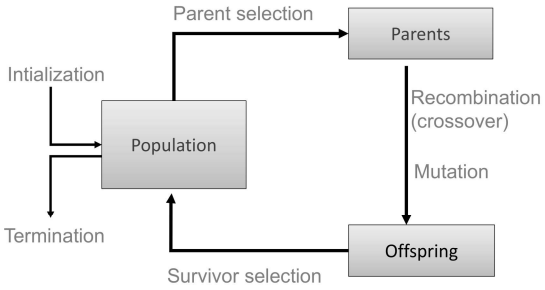
Evolutionary Computing

Chapter 5

Chapter 5:
Fitness, Selection and Population Management

- Selection is second fundamental force for evolutionary systems
- Components exist of:
  - Population management models
  - Selection operators
  - Preserving diversity

Scheme of an EA:
General scheme of EAs



Population Management Models: Introduction

- Two different population management models exist:
  - Generational model
    - each individual survives for exactly one generation
    - the entire set of parents is replaced by the offspring
  - Steady-state model
    - one offspring is generated per generation
    - one member of population replaced
- Generation Gap
  - The proportion of the population replaced
  - Parameter = 1.0 for GGA, = 1/pop_size for SSGA

Population Management Models: Fitness based competition

- Selection can occur in two places:
  - Selection from current generation to take part in mating (parent selection)
  - Selection from parents + offspring to go into next generation (survivor selection)
- Selection operators work on whole individual (fitness)
  - i.e. they are representation-independent!
- Distinction between selection
  - Operators: define selection probabilities
  - Algorithms: define how probabilities are implemented

Parent Selection: Fitness-Proportionate Selection

- Probability for individual *i* to be selected for mating in a population size μ with FPS is

$$P_{FPS}(i) = f_i \Big/ \sum_{j=1}^{\mu} f_j$$

- Problems include
  - One highly fit member can rapidly take over if rest of population is much less fit: Premature Convergence
  - At end of runs when fitnesses are similar, loss of selection pressure
  - Highly susceptible to function transposition (example next slide)
- Scaling can fix last two problems
  - Windowing:

$$f'(i) = f(i) - \beta'$$

where β is worst fitness in this (or last n) generations

  - Sigma Scaling:

$$f'(i) = \max(f(i) - (\bar{f} - c \bullet \sigma_f), 0)$$

where *c* is a constant, usually 2.0

How to scale fitness?
How to rank fitness?

Parent Selection: Rank-based Selection

- Attempt to remove problems of FPS by basing selection probabilities on *relative* rather than *absolute* fitness
- Rank population according to fitness and then base selection probabilities on rank (fittest has rank μ-*1* and worst rank 0)
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Convert the fitness to a

Rank-based Selection:

Rank-based Selection: Linear Ranking

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

- Parameterised by factor *s:* 1 < *s* ≤ 2
  - measures advantage of best individual
- Simple 3 member example (i=rank, u=pop size)

Individual	Fitness	Rank	P_{selFP}	P_{selLR} $(s=2)$	P_{selLR} $(s=1.5)$
A	1	0	0.1	0	0.167
B	4	1	0.4	0.33	0.33
C	5	2	0.5	0.67	0.5
Sum	10		1.0	1.0	1.0

8 / 32

Rank-based selection: Exponential Ranking

$$P_{\exp-rank}(i) = \frac{1-e^{-i}}{c}$$

- Linear Ranking is limited in selection pressure
- Exponential Ranking can allocate more than 2 copies to fittest individual
- Normalise constant factor *c* according to population size, to ensure the sum of the probabilities = 1.

Sample mating pool from the selection probability distribution (roulette wheel, stochastic universal sampling)

9 / 32

Parent Selection: Tournament Selection (1/2)

- All methods above rely on global population statistics
  - Could be a bottleneck esp. on parallel machines, very large population
  - Relies on presence of external fitness function which might not exist: e.g. evolving game players
- Idea for a procedure using only local fitness information:
  - Pick *k* members at random then select the best of these
  - Repeat to select more individuals

10 / 32

Parent Selection: Tournament Selection (2/2)

- Probability of selecting *i* will depend on:
  - Rank of *i*
  - Size of sample *k*
    - higher *k* increases selection pressure
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
  - Whether fittest contestant always wins (deterministic) or this happens with probability *p*

11 / 32

Convert the fitness to a rank

- you a need to know rank based selection and how you could put it in a formula like

Parent Selection: Uniform

Prof called this silly but spent time on it

$$P_{uniform}(i) = \frac{1}{\mu}$$

- Parents are selected by uniform random distribution whenever an operator needs one/some
- Uniform parent selection is unbiased - every individual has the same probability to be selected
- When working with extremely large populations, over-selection can be used.

Survivor Selection

- Managing the process of reducing the working memory of the EA from a set of μ parents and λ offspring to a set of μ individuals forming the next generation
- The parent selection mechanisms can also be used for selecting survivors
- Survivor selection can be divided into two approaches:
  - Age-Based Selection - ehh and skipped
    - Fitness is not taken into account
    - In SSGA can implement as "delete-random" (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
  - Fitness-Based Replacement

Fitness-based replacement (1/2)

- Elitism → ⭐⭐ Must study elitism
  - Always keep at least one copy of the fittest solution so far
  - Widely used in both population models (GGA, SSGA)
- GENITOR: a.k.a. "delete-worst"
  - From Whitley's original Steady-State algorithm (he also used linear ranking for parent selection)
  - Rapid takeover: use with large populations or "no duplicates" policy
- Round-robin tournament
  - P(t): μ parents, P'(t): μ offspring
  - Pairwise competitions in round-robin format:
    - Each solution x from U = P(t) ∪ P'(t) is evaluated against q < U other randomly chosen solutions from U
    - For each comparison, a "win" is assigned if x is better than its opponent
    - The μ solutions with the greatest number of wins are retained to be parents of the next generation
  - Parameter q allows tuning selection pressure
  - Typically q = 10

Fitness-based replacement (2/2)

- (μ,λ)-selection
  - based on the set of children only (λ > μ)
  - choose best μ of children pool
- (μ+λ)-selection
  - based on the set of parents and children
  - choose best μ of parents + children combined pools

- Often (μ,λ)-selection is preferred for:
  - Better in leaving local optima
  - Better in following moving optima
  - Using the + strategy bad σ values can survive in ⟨x,σ⟩ too long if their host x is very fit
- λ ≈ 7 · μ is a traditionally good setting (decreasing over the last couple of years, λ ≈ 3 · μ seems more popular lately)

Need to know these!! understand μ and λ

Selection Pressure

- Takeover time *tau** is a measure to quantify the selection pressure
- The number of generations it takes until the application of selection completely fills the population with copies of the best individual of the 1st generation
- Goldberg and Deb showed that for ES (μ,λ):

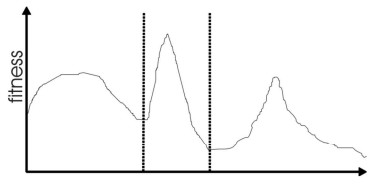
$$\tau^* = \frac{\ln \lambda}{\ln(\lambda / \mu)}$$

- For proportional selection (FPS) in a GA, the takeover time is λ *ln* λ

Forget it only applies to very standard stuff

Multimodality

Most interesting problems have more than one locally optimal solution.



Who cares?

Multimodality: Genetic Drift

- Finite population with global mixing and selection eventually convergence around one optimum
- Why?
- Often might want to identify several possible peaks
- Sub-optimum can be more attractive

Approaches for Preserving Diversity: Introduction (1/2)

- Explicit vs implicit
- Implicit approaches:
  - Impose an equivalent of *geographical* separation
  - Impose an equivalent of *speciation*
- Explicit approaches
  - Make *similar* individuals compete for resources (divide fitness between them)
  - Make *similar* individuals compete with each other for survival (selection for parenting or survival)

Very important

Approaches for Preserving Diversity: Introduction (1/2)

Different spaces:

- Genotype diversity
  - Set of representable solutions
- Phenotype diversity
  - The end result
  - Neighbourhood structure may bear little relation with genotype space
- Algorithmic diversity (different indi's handled differently, and not due to genotype or fitness)
  - Equivalent of the geographical space on which life on earth has evolved
  - Structuring the population of candidate solutions

Explicit Approaches for Preserving Diversity: **Fitness Sharing** (1/2)

- Restricts the number of individuals within a given niche by "sharing" their fitness, so as to allocate individuals to niches in proportion to the niche fitness
- need to set the size of the niche σ_{share} in either genotype or phenotype space
- run EA as normal but after each generation set

$$f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i,j))} \qquad sh(d) = \begin{cases} 1 - d/\sigma & d \le \sigma \\ 0 & otherwise \end{cases}$$

understand the formula

Explicit Approaches for Preserving Diversity: Fitness Sharing (2/2)

- Note: if we used sh(d) = 1 for d < σ_{share} then the sum that reduces the fitness would simply count the number of neighbours, i.e., individuals closer than σ_{share}
- This creates an advantage of being alone in the neighbourhood
- Using 1 – d/ σ_{share} instead of 1 implies that we count distant neighbours less

Explicit Approaches for Preserving Diversity: **Crowding** (1/2)

- Attempts to distribute individuals evenly amongst niches
- relies on the assumption that offspring will tend to be close to parents
- uses a distance metric in pheno/geno-type space
- randomly shuffle and pair parents, produce 2 offspring
- set up the parent vs. child tournaments such that the inter-tournament distances are minimal

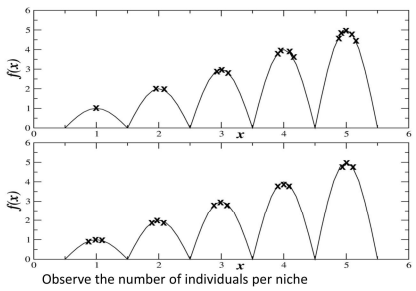
understand how it is a competition between parents & children

you need sharing and crowding to have multiple peaks

Explicit Approaches for Preserving Diversity: Crowding (2/2)

- That is, number the two p's (parents) and the two o's (offspring) such that
- $d(p_1,o_1) + d(p_2,o_2) < d(p_1,o_2) + d(p_2,o_1)$
- and let o_1 compete with p_1 and o_2 compete with p_2

Explicit Approaches for Preserving Diversity: Crowding or Fitness sharing?



Observe the number of individuals per niche

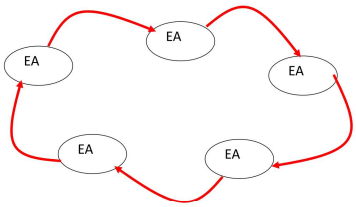
Diversity

Implicit Approaches for Preserving Diversity: Automatic **Speciation**

- Either only mate with genotypically / phenotypically similar members or
- Add bits (tags) to problem representation
  - that are initially randomly set
  - subject to recombination and mutation
  - when selecting partner for recombination, only pick members with a good match

know what it is, assign a color

Implicit Approaches for Preserving Diversity: "**Island**" Model Parallel EAs (1/4)



Periodic migration of individual solutions between populations

27 / 32

Multiple populations evolving independally with occasional migration

Implicit Approaches for Preserving Diversity: "Island" Model Parallel EAs (2/4)

- Run multiple populations in parallel
- After a (usually fixed) number of generations (an ***Epoch***), exchange individuals with neighbours
- Repeat until ending criteria met
- Partially inspired by parallel/clustered systems

28 / 32

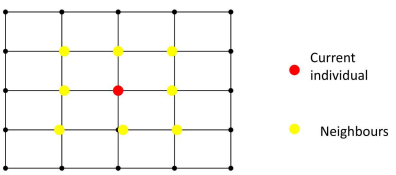
Island Model: Parameters

- How often to exchange individuals ?
  - too quick and all sub-populations converge to same solution
  - too slow and waste time
  - most authors use range~ 25-150 generations
  - can do it adaptively (stop each pop when no improvement for (say) 25 generations)
- How many, which individuals to exchange ?
  - usually ~2-5, but depends on population size.
  - Copied vs moved
  - Martin et al found that better to exchange randomly selected individuals than best
- Operators can differ between the sub-populations

29 / 32

Implicit Approaches for Preserving Diversity: **Cellular EAs** (1/3)

- Impose spatial structure (usually grid) in 1 pop



30 / 32

Distance based allowance of crossover. it's implicit.

Implicit Approaches for Preserving Diversity: Cellular EAs (2/3)

- Consider each individual to exist on a point on a (usually rectangular toroid) grid
- Selection (hence recombination) and replacement happen using concept of a neighbourhood a.k.a. ***deme***
- Leads to different parts of grid searching different parts of space, good solutions diffuse across grid over a number of gens

Implicit Approaches for Preserving Diversity: Cellular EAs (3/3)

- Assume rectangular grid so each individual has 8 immediate neighbours
- Equivalent of 1 generation is:
  - **pick** individual in pop at random
  - pick one of its **neighbours** using roulette wheel
  - **crossover** to produce 1 child and mutate it
  - replace individual (parent) if **fitter**
  - circle through **population** until done

Lec4-Parameters_and_Parameter_Tuning-2025

Evolutionary Computing

Chapter 7

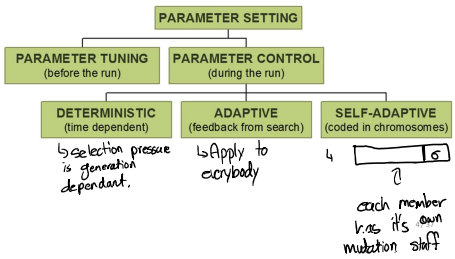
Chapter 7: Parameters and Parameter Tuning

- History
- Taxonomy
- Parameter Tuning vs Parameter Control
- EA calibration
- Parameter Tuning
  - Testing
  - Effort
  - Recommendation

Brief historical account

- 1970/80ies "GA is a robust method"
- 1970ies + ESs self-adapt mutation stepsize σ
- 1986 meta-GA for optimizing GA parameters
- 1990ies EP adopts self-adaptation of σ as 'standard'
- 1990ies some papers on changing parameters on-the-fly
- 1999 Eiben-Michalewicz-Hinterding paper proposes clear taxonomy & terminology

Taxonomy



- Parameter tuning
tune before the ran, they are set before.

- Adaptive tuning
done during the ran.

- Self adaptive
↳ it is subject to the evolutionary process

step size or frequency

deterministic: value of mutation as a function of generations/e only
↳ only a function of time

Parameter control

Parameter Tuning

Parameter tuning: testing and comparing different values ***before*** the "real" run

Problems:

- users mistakes in settings can be sources of errors or sub-optimal performance
- costs **much** time
- parameters interact: exhaustive search is **not** practicable
- good values **may** become bad during the run

evaluations / time

→ It is subject to the evolutionary process

Parameter Tuning

Parameter tuning: testing and comparing different values ***before*** the "real" run

Problems:

- users mistakes in settings can be sources of errors or sub-optimal performance
- costs **much** time
- parameters interact: exhaustive search is **not** practicable
- good values **may** become bad during the run

Can we tie each GA run to a different thread / core on our CPU?

Parameter Control

Diversity Maintanence

Make sure you look at diversity.

Parameter control: setting values on-line, *during* the actual run, e.g.

- ***Preset***: predetermined time-varying schedule p = p(t)
- ***Adaptive***: using feedback from the search process to adapt the values, but the manner of adaptation is fixed.
- ***Self-adaptive***: encoding parameters in chromosomes and rely on natural selection to adapt the values of those parameters.

Problems:

- finding optimal p is hard, finding optimal p(t) is ***harder***
- still user-defined feedback mechanism: ***how*** to "optimize"?
- ***when*** would natural selection work for algorithm parameters?

Notes on parameter control

- Parameter control offers the possibility to use appropriate values in various stages of the search
- Adaptive and self-adaptive control can "liberate" users from tuning → reduces need for EA expertise for a new application
- Assumption (ie, which in cases may be wrong): control heuristic is less parameter-sensitive than the EA

BUT

- State-of-the-art is a mess: literature is a potpourri, no generic knowledge, no principled approaches to developing control heuristics (deterministic or adaptive), no solid testing methodology

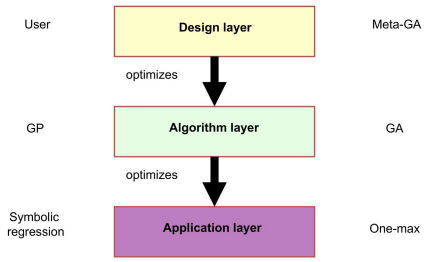
WHAT ABOUT AUTOMATED TUNING?

Historical account (cont'd)

Last decade:

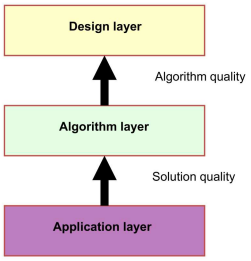
- More & more work on **parameter control**
  - Traditional parameters: mutation and x-over
  - Non-traditional parameters: selection method and population size
  - All parameters → "parameterless" EAs (name!?)
- Not much work on **parameter tuning**, i.e.,
  - Almost nobody reports on tuning efforts behind their EA published
  - A handful papers on tuning methods / algorithms

Control flow of EA calibration / design



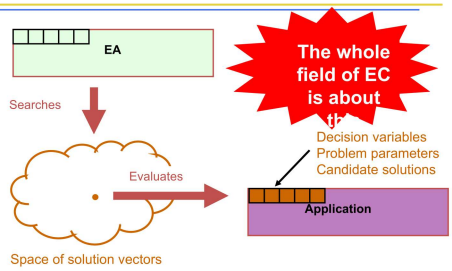
9 / 37

Information flow of EA calibration / design



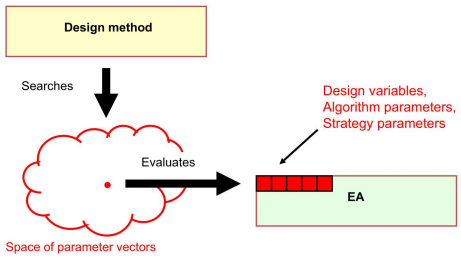
10 / 37

Lower level of EA calibration / design



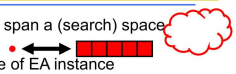
11 / 37

Upper level of EA calibration / design



12 / 37

Parameter – performance landscape

- The parameters **themselves** span a (search) space
- One point – one EA instance 
- Height of point = performance of EA instance on a given problem (or problem type)
- Parameter-performance landscape or utility landscape for each { EA + problem instance/class + performance measure }
- This landscape is **unlikely** to be trivial, e.g., unimodal or separable
- *If* there is some **structure** in the utility landscape, then we can do better than random or exhaustive search

Ontology - Terminology

	LOWER PART	UPPER PART
METHOD	EA	Tuner
SEARCH SPACE	Solution vectors	Parameter vectors
QUALITY	Fitness	Utility
ASSESSMENT	Evaluation	Test

- Fitness ≈ objective function value
- Utility = ?
  - **M**ean **B**est **F**itness
  - **A**verage number of **E**valuations to **S**olution
  - **S**uccess **R**ate (%)
  - Robustness (e.g., to ranges of parameter value)
  - Combination of some of these

Off-line vs. On-line calibration / design

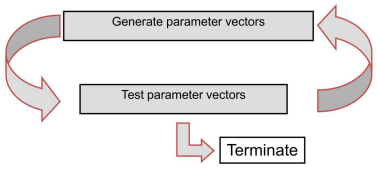
Design / calibration method

- ***Off***-line (before the real run) → parameter ***tuning***
- ***On***-line (during the real run) → parameter ***control***

- Advantages of tuning
  - Less complicated and more direct
  - Addresses the most immediate need of users
  - Control strategies have parameters *too* → need tuning themselves!
  - Knowledge about tuning (utility landscapes) can help the design of good control strategies
  - There are indications that good tuning works better than control

Tuning by generate-and-test

- EA tuning is a search problem ***itself***
- Straightforward approach: generate-and-test

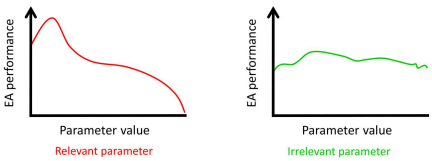


Testing parameter vectors

- Run EA with these parameters on the given problem or problems
- Record EA performance in that run e.g., by
  - Solution quality = best fitness at termination
  - Speed to solution ≈ time used to find required solution quality
- EAs are stochastic → repetitions are needed for reliable evaluation → we get statistics, e.g.,
  - Average performance by solution **quality**, **speed** (MBF, AES, AEB)
  - Success rate = % runs ending with **success**
  - Robustness = **variance** in those averages over different problems
- Big issue: how many repetitions of the test

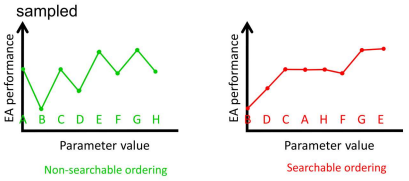
Numeric parameters

- E.g., population size, xover rate, tournament size, …
- Domain is subset of R, Z, N (finite or infinite)
- Sensible distance metric → searchable



Symbolic parameters

- E.g., x-over_operator, elitism, selection_method
- Finite domain, e.g., {1-point, uniform, averaging}, {Y, N}
- No sensible distance metric → non-searchable, must be sampled



Notes on parameters

- A value of a symbolic parameter can introduce a numeric parameter, e.g.,
  - Selection = tournament → tournament size
  - Populations_type = overlapping → generation gap
- Parameters can have a hierarchical, nested structure
- Number of EA parameters is ***not*** defined in general
- Can ***not*** simply denote the design space / tuning search space by

$$S = Q_1 \times \ldots Q_m \times R_1 \times \ldots \times R_n$$

with Q_i / R_j as domains of the symbolic/numeric parameters

What is an EA? (1/2)

	ALG-1	ALG-2	ALG-3	ALG-4
SYMBOLIC PARAMETERS				
Representation	Bit-string	Bit-string	Real-valued	Real-valued
Overlapping pops	N	Y	Y	Y
Survivor selection	–	Tournament	Replace worst	Replace worst
Parent selection	Roulette wheel	Uniform determ.	Tournament	Tournament
Mutation	Bit-flip	Bit-flip	N(0,σ)	N(0,σ)
Recombination	Uniform x-over	Uniform x-over	Discrete x-over	Discrete x-over
NUMERIC PARAMETERS				
Generation gap	–	0.5	0.9	0.9
Population size	100	500	100	300
Tournament size	–	2	3	30
Mutation rate	0.01	0.1	–	–
Mutation step size	–	–	0.01	0.05
Crossover rate	0.8	0.7	1	0.8

What is an EA? (2/2)

Make a principal distinction between EAs and EA instances and place the border between them by:

- Option 1
  - There is only one EA, the ***generic*** EA scheme
  - Previous table contains 1 EA and 4 EA-instances
- Option 2
  - An EA = particular configuration of the ***symbolic*** parameters
  - Previous table contains 3 EAs, with 2 instances for one of them
- Option 3
  - An EA = particular ***configuration*** of parameters
  - Notions of EA and EA-instance coincide
  - Previous table contains 4 EAs / 4 EA-instances

Lec5-Working_with_EAs-2025 (1)

October 3, 2025 3:35 PM

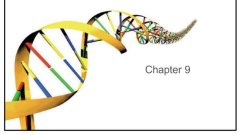
Lec5-Working_with_EAs-2025 (1)

Just get an idea of "What I care about"?

Q: if you are trying to do this what kind of quality measurement do you need?

10/3/2025

Evolutionary Computing



Chapter 9

1

Chapter 9: Working with Evolutionary Algorithms

- Experiment design
- Algorithm design
- Test problems
- Measurements and statistics
- Some tips and summary

2

Experimentation

- Has a goal or goals
- Involves algorithm design and implementation
- Needs problem(s) to run the algorithm(s) on
- Amounts to running the algorithm(s) on the problem(s)
- Delivers measurement data, the results
- Is concluded with evaluating the results in the light of the given goal(s)
- Is often documented

3

Experimentation: Goals

- Get a good solution for a given problem
- Show that EC is applicable in a (new) problem domain
- Show that *my_EA* is better than *benchmark_EA*
- Show that EAs outperform traditional algorithms (sic!)
- Find best setup for parameters of a given algorithm
- Understand algorithm behavior (e.g. pop dynamics)
- See how an EA scales-up with problem size
- See how performance is influenced by parameters
- …

4

Prof likes to ask this stuff

Example: Production Perspective

- Optimising Internet shopping delivery route
  - Different destinations each day
  - Limited time to run algorithm each day
  - Must always be reasonably good route in limited time

5

What do we care about

Example: Design Perspective

- Optimising spending on improvements to national road network
  - Total cost: billions of Euro
  - Computing costs negligible
  - Six months to run algorithm on hundreds computers
  - Many runs possible
  - Must produce very good result just once

6

How would you optimise it.

One time problem

- We only need the EA once.

- Don't care about it being absolutly optimal.
- Time required to get a good enough solution + the std. (standard deviation)

1

10/3/2025

Perspectives of goals

- Design perspective: find a very good solution at least once
- Production perspective: find a good solution at almost every run
- Publication perspective: must meet scientific standards (huh?)
- Application perspective: good enough is good enough (verification!)

These perspectives have very different implications on evaluating the results (yet often left implicit)

7

Algorithm design

- Design a representation
- Design a way of mapping a genotype to a phenotype
- Design a way of evaluating an individual
- Design suitable mutation operator(s)
- Design suitable recombination operator(s)
- Decide how to select individuals to be parents
- Decide how to select individuals for the next generation (how to manage the population)
- Decide how to start: initialization method
- Decide how to stop: termination criterion

8

Test problems

- 5 DeJong functions
- 25 "hard" objective functions
- Frequently encountered or otherwise important variants of given practical problem
- Selection from recognized benchmark problem repository ("challenging" by being NP-- ?!)
- Problem instances made by random generator

Choice has severe implications on
- generalizability and
- scope of the results

9

Bad example (1/2)

- I invented "tricky mutation"
- Showed that it is a good idea by:
  - Running standard (?) GA and tricky GA
  - On 10 objective functions from the literature
  - Finding tricky GA better on 7, equal on 1, worse on 2 cases
- I wrote it down in a paper
- And it got published!
- Q: what did I learned from this experience?
- Q: Is this good work?

10

Bad example (2/2)

- What did I (my readers) did not learn:
  - How relevant are these results (test functions)?
  - What is the scope of claims about the superiority of the tricky GA?
  - Is there a property distinguishing the 7 good and the 2 bad functions?
  - Are my results generalizable? (Is the tricky GA applicable for other problems? Which ones?)

11

Getting Problem Instances (1/3)

- Testing on real data
- Advantages:
  - Results could be considered as very relevant viewed from the application domain (data supplier)
- Disadvantages
  - Can be over-complicated
  - Can be few available sets of real data
  - May be commercial sensitive – difficult to publish and to allow others to compare
  - Results are hard to generalize

12

2

Getting Problem Instances (2/3)

- Standard data sets in problem repositories, e.g.:
  - OR-Library
    http://www.ms.ic.ac.uk/info.html
  - UCI Machine Learning Repository
    www.ics.uci.edu/~mlearn/MLRepository.html
- Advantage:
  - Well-chosen problems and instances (hopefully)
  - Much other work on these → results comparable
- Disadvantage:
  - Not real – might miss crucial aspect
  - Algorithms get tuned for popular test suites

13

Getting Problem Instances (3/3)

- Problem instance generators produce simulated data for given parameters, e.g.:
  - GA/EA Repository of Test Problem Generators
    http://www.cs.uwyo.edu/~wspears/generators.html
- Advantage:
  - Allow very systematic comparisons for they
    - can produce many instances with the same characteristics
    - enable gradual traversal of a range of characteristics (hardness)
  - Can be shared allowing comparisons with other researchers
- Disadvantage
  - Not real – might miss crucial aspect
  - Given generator might have hidden bias

14

Basic rules of experimentation

- EAs are stochastic →
  never draw any conclusion from a single run
  - perform sufficient number of independent runs
  - use statistical measures (averages, standard deviations)
  - use statistical tests to assess reliability of conclusions
- EA experimentation is about comparison →
  always do a fair competition
  - use the same amount of resources for the competitors
  - try different comp. limits (to coop with turtle/hare effect)
  - use the same performance measures

15

Things to Measure

Many different ways. Examples:

- Average result in given time
- Average time for given result
- Proportion of runs within % of target
- Best result over *n* runs
- Amount of computing required to reach target in given time with % confidence
- …

16

What time units do we use?

- Elapsed time?
  - Depends on computer, network, etc…
- CPU Time?
  - Depends on skill of programmer, implementation, etc…
- Generations?
  - Difficult to compare when parameters like population size change
- Evaluations?
  - Evaluation time could depend on algorithm, e.g. direct vs. indirect representation

17

Measures

- Performance measures (off-line)
  - Efficiency (alg. speed)
    - CPU time
    - No. of steps, i.e., generated points in the search space
  - Effectivity (alg. quality)
    - Success rate
    - Solution quality at termination
- "Working" measures (on-line)
  - Population distribution (genotypic)
  - Fitness distribution (phenotypic)
  - Improvements per time unit or per genetic operator
  - …

18

Performance measures

- No. of generated points in the search space
  = no. of fitness evaluations
  (don't use no. of generations!)
- AES: average no. of evaluations to solution
- SR: success rate = % of runs finding a solution
  (individual with acceptabe quality / fitness)
- MBF: mean best fitness at termination, i.e., best per run, mean over a set of runs
- SR ↔ MBF
  - Low SR, high MBF: good approximizer (more time helps?)
  - High SR, low MBF: "Murphy" algorithm

19

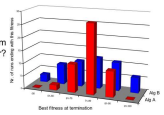
Fair experiments

- Basic rule: use the same computational limit for each competitor
- Allow each EA the same no. of evaluations, but
  - Beware of hidden labour, e.g. in heuristic mutation operators
  - Beware of possibly fewer evaluations by smart operators
- EA vs. heuristic: allow the same no. of steps:
  - Defining "step" is crucial, might imply bias!
  - Scale-up comparisons eliminate this bias

20

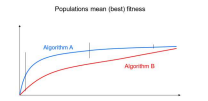
Example: off-line performance measure evaluation

Which algorithm is better? Why? When?



21

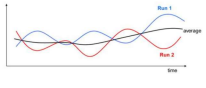
Example: on-line performance measure evaluation



Which algorithm is better? Why? When?

22

Example: averaging on-line measures



Averaging can "choke" interesting information

23

Example: overlaying on-line measures

Overlay of curves can lead to very "cloudy" figures

24

Statistical Comparisons and Significance

- Algorithms are stochastic, results have element of "luck"
- If a claim is made "Mutilation A is better than mutation B", need to show statistical significance of comparisons
- Fundamental problem: two series of samples (random drawings) from the SAME distribution may have DIFFERENT averages and standard deviations
- Tests can show if the differences are significant or not

25

Example

Trial	Old Method	New Method
1	500	697
2	600	543
3	566	654
4	573	555
5	420	654
6	590	712
7	700	456
8	472	564
9	534	675
10	512	643
Average	545.7	612.3

Is the new method better?

26

Example (cont'd)

Trial	Old Method	New Method
1	500	697
2	600	543
3	566	654
4	573	555
5	420	654
6	590	712
7	700	456
8	472	564
9	534	675
10	512	643
Average	545.7	612.3
SD	73.39625085	73.54773317
T-test	**0.070867706**	

- Standard deviations supply additional info
- T-test (and alike) indicate the chance that the values came from the same underlying distribution (difference is due to random effects) E.g. with 7% chance in this example.

27

Statistical tests

- T-test assummes:
  - Data taken from continuous interval or close approximation
  - Normal distribution
  - Similar variances for too few data points
  - Similar sized groups of data points
- Other tests:
  - Wilcoxon – preferred to t-test where numbers are small or distribution is not known.
  - F-test – tests if two samples have different variances.

28

Statistical Resources

- http://fonsg3.let.uva.nl/Service/Statistics.html
- http://department.obg.cuhk.edu.hk/ResearchSupport/
- http://faculty.vassar.edu/lowry/webtext.html
- Microsoft Excel
- http://www.octave.org/

29

Better example: problem setting

- I invented myEA for problem X
- Looked and found 3 other EAs and a traditional benchmark heuristic for problem X in the literature
- Asked myself when and why is myEA better

30

Better example: experiments

- Found/made problem instance generator for problem X with 2 parameters:
  - n (problem size)
  - k (some problem specific indicator)
- Selected 5 values for k and 5 values for n
- Generated 100 problem instances for all combinations
- Executed all alg's on each instance 100 times (benchmark was also stochastic)
- Recorded AES, SR, MBF values w/ same comp. limit (AES for benchmark?)
- Put my program code and the instances on the Web

31

Better example: evaluation

- Arranged results "in 3D" (n,k) + performance (with special attention to the effect of n, as for scale-up)
- Assessed statistical significance of results
- Found the niche for my_EA:
  - Weak in … cases, strong in - - cases, comparable otherwise
  - Thereby I answered the "when question"
- Analyzed the specific features and the niches of each algorithm thus answering the "why question"
- Learned a lot about problem X and its solvers
- Achieved generalizable results, or at least claims with well-identified scope based on solid data
- Facilitated reproducing my results → further research

32

Some tips

- Be organized
- Decide what you want & define appropriate measures
- Choose test problems carefully
- Make an experiment plan (estimate time when possible)
- Perform sufficient number of runs
- Keep all experimental data (never throw away anything)
- Use good statistics ("standard" tools from Web, MS, R)
- Present results well (figures, graphs, tables, …)
- Watch the scope of your claims
- Aim at generalizable results
- Publish code for reproducibility of results (if applicable)
- Publish data for external validation (open science)

33

Lec6-IntroductionToMultiobjective_Evolutionary_Algorit...

- Understand dominance non dominated

Evolutionary Computing

Chapter 12

Chapter 12: Multiobjective Evolutionary Algorithms

- Multiobjective optimisation problems (MOP)
  - Pareto optimality
- EC approaches
  - Evolutionary spaces
  - Preserving diversity

1 / 26

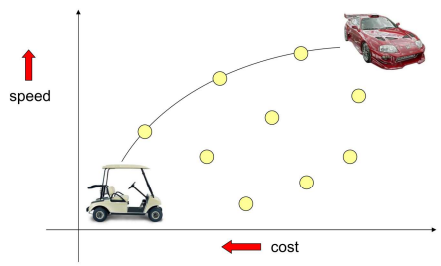
Multi-Objective Problems (MOPs)

- Wide range of problems can be categorised by the presence of a number of *n* possibly conflicting objectives:
  - buying a car: speed vs. price vs. reliability
  - engineering design: lightness vs. strength
- Two problems:
  - finding set of good solutions
  - choice of best for particular application

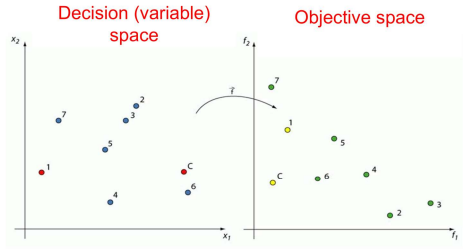
2 / 26

An example: Buying a car

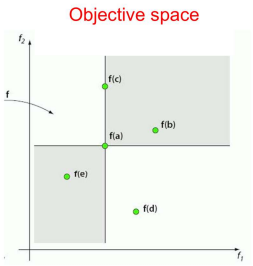
objectives can have different values, but what if we have more than one objective...



Two spaces



Comparing solutions

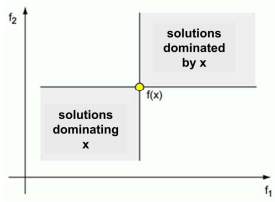


- Optimisation task: Minimize both f_1 and f_2
- Then:
  a is better than b
  a is better than c
  a is worse than e
  a and d are incomparable

← e & d are the non dominated solutions here

Dominance relation

- Solution x dominates solution y, ($x \preceq y$), if:
  - x is better than y in at least one objective,
  - x is not worse than y in all other objectives



Pareto optimality

- Solution x is non-dominated among a set of solutions Q if no solution from Q dominates x
- A set of non-dominated solutions from the entire feasible solution space is the Pareto-optimal set, its members Pareto-optimal solutions
- Pareto-optimal front: an image of the Pareto-optimal set in the objective space

Illustration of the concepts

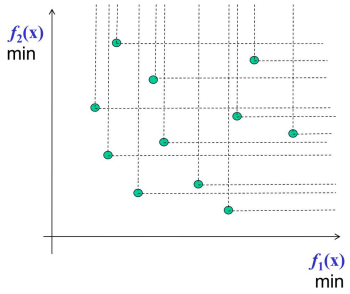
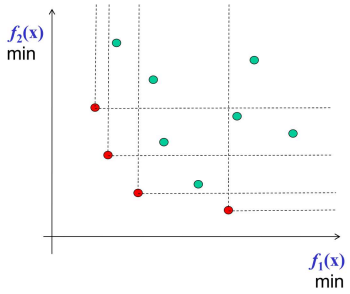


Illustration of the concepts



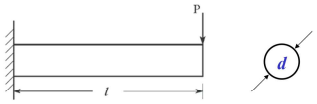
△ ∈ True multi objectives.

Q: given points, select which of the points in non dominated

↳ given rank 0 is removed, what is rank 1

A practical example: The beam design problem

Minimize weight and deflection of a beam (Deb, 2001):



exam: 30 questions
equally weighted
exam starts at 3
till 4:15

TA said exam is difficult
so make sure you read.

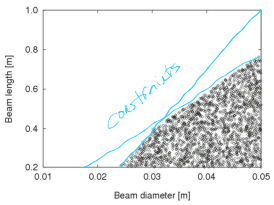
Formal definition

- Minimize $f_1(d,l) = \rho \frac{\pi d^2}{4} l$ (beam weight) O_1
- minimize $f_2(d,l) = \delta = \frac{64Pl^3}{3E\pi d^4}$ (beam deflection) O_2
- subject to $0.01\text{ m} \le d \le 0.05\text{ m}$ var1
  $0.2\text{ m} \le l \le 1.0\text{ m}$ var2
  Constraints
  $\sigma_{max} = \frac{32Pl}{\pi d^3} \le S_y$ (maximum stress)
  $\delta \le \delta_{max}$

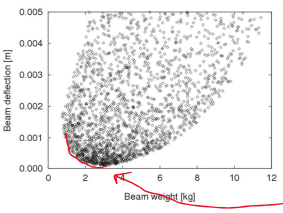
where $\rho = 7800\text{ kg/m}^3,\ P = 2\text{ kN}$
$E = 207\text{ GPa}$
$S_y = 300\text{ MPa},\ \delta_{max} = 0.005\text{ m}$

Feasible solutions

Decision (variable) space

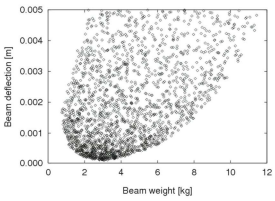


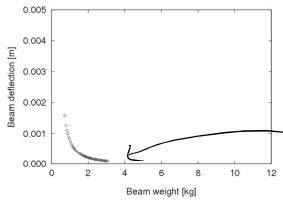
Objective space



Non dominating solution

Goal: Finding non-dominated solutions

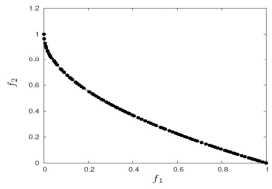




These arnt all the best yet, obviously they are for the same generation, but the next generation may be better.

Goal of multiobjective optimisers

- Find a set of non-dominated solutions (approximation set) following the criteria of:
  - convergence (as close as possible to the Pareto-optimal front),
  - diversity (spread, distribution)



Single- vs. multiobjective optimisation

Characteristic	Singleobjective optimisation	Multiobjective optimisation
Number of objectives	one	more than one
Spaces	single	two: decision (variable) space, objective space
Comparison of candidate solutions	*x* is better than *y*	*x* dominates *y*
Result	one (or several equally good) solution(s)	Pareto-optimal set
Algorithm goals	convergence	convergence, diversity

Two approaches to multiobjective optimisation

- Preference-based:
  traditional, using single objective optimisation methods
- Ideal:
  possible with novel multiobjective optimisation techniques, enabling better insight into the problem

We want a set of solutions that are diverse and equally dominant so we select based off rank

Preference-based approach

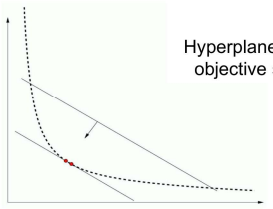
- Given a multiobjective optimisation problem,
- use higher-level information on importance of objectives
- to transform the problem into a singleobjective one,
- and then solve it with a single objective optimisation method
- to obtain a particular trade-off solution.

 → How to join multiple objectives into 1

An example approach: Weighted-sum

- Modified problem:

$$F(\mathbf{x}) = \sum_{m=1}^{M} w_m f_m(\mathbf{x}), \quad w_m \in [0,1], \quad \sum_{m=1}^{M} w_m = 1$$



Hyperplanes in the objective space!

Ideal approach

- Given a multiobjective optimisation problem,
- solve it with a multiobjective optimisation method
- to find multiple trade-off solutions,
- and then use higher-level information
- to obtain a particular trade-off solution.

Multiobjective optimisation with evolutionary algorithms

- Population-based method
- Can return a set of trade-off solutions (approximation set) in a single run
- Allows for the ideal approach to multiobjective optimisation

EC approach: Advantages

- Population-based nature of search means you can *simultaneously* search for set of points approximating Pareto front
- Don't have to make guesses about which combinations of weights might be useful
- Makes no assumptions about shape of Pareto front - can be convex / discontinuous etc.

21 / 26

EC approach: Requirements

- Way of assigning fitness,
  - usually based on dominance
- Preservation of diverse set of points
  - similarities to multi-modal problems
- Remembering all the non-dominated points you have seen
  - usually using elitism or an archive

22 / 26

EC approach: Fitness Assignment

- Could use aggregating approach and change weights during evolution
  - no guarantees
- Different parts of population use different criteria
  - e.g. VEGA, but no guarantee of diversity
- Dominance
  - ranking or depth based
  - fitness related to whole population

23 / 26

EC approach: Diversity maintenance

- Usually done by niching techniques such as:
  - fitness sharing
  - adding amount to fitness based on inverse distance to nearest neighbour (minimisation)
  - (adaptively) dividing search space into boxes and counting occupancy
- All rely on some distance metric in genotype / phenotype space

EC approach: Remembering Good Points

- Could just use elitist algorithm
  - e.g. (μ + λ) replacement
- Common to maintain an archive of non-dominated points
  - some algorithms use this as second population that can be in recombination etc.
  - others divide archive into regions too, e.g. PAES

MOP - summary

- MO problems occur very frequently
- EAs are very good in solving MO problems
- MOEAs are one of the most successful EC subareas