Evolutionairy Computing

Evolutionary Computing (Vrije Universiteit Amsterdam)

Evolutionary Computing

Chapter 1

Problems can be classified in different ways:

- Black box model
- Search problems
- Optimization vs constraint satisfaction
- NP problems

1. Black box model

Three components: when 1 component is unknown → other problem types

- Optimization: Model and the desired output is known, find input
  Examples: • Timetables for university, call center or hospital • Design specifications • Traveling salesman problem (TSP) • Eight-queens problem, etc
- Modeling: Modelling: find model, e.g. evolutionary machine learning
- Simulation: we have a model and change input to see the effects on output (weather, climate, economics).

2. Search problems

Difference between

- Search problems, which define search spaces
- Problem-solvers, which move through search spaces (to find a solution).

3. Optimization vs constraint satisfaction

Objective function: a way of assigning a value to a solution that reflects its quality on scale

• Number of un-checked queens (maximize)
• Length of a tour visiting given set of cities (minimize)

Constraint: binary evaluation telling whether a given requirement holds or not

• Property of a chessboard configuration: are there queens that check each other? (we are good if “no”)
• Property of a tour: is city X is visited after city Y? (we are good if “yes”)

	Objective function	
Constraints	**Yes**	**No**
Yes	Constrained Optimisation Problem	Constraint Satisfaction Problem
No	Free Optimisation Problem	No problem ☺

How can we formulate the 8-queens problem as a FOP/CSP/COP?

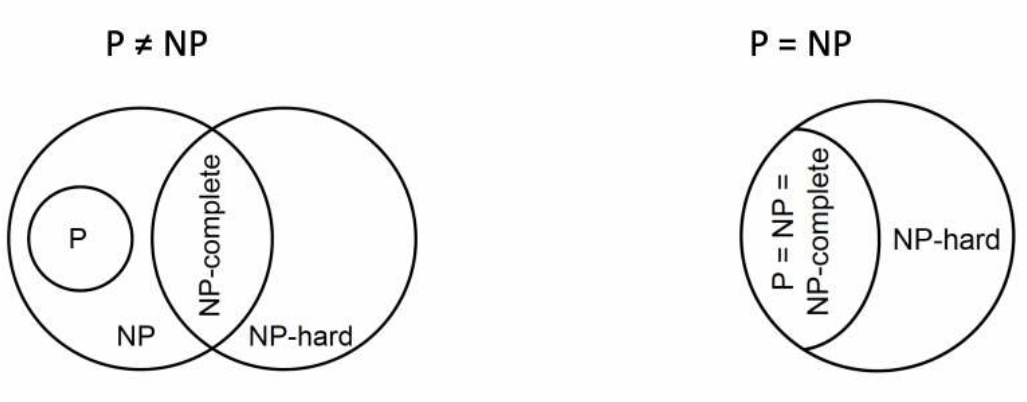
4. NP problems

Key notions

- Problem size: number of problem variables (dimensionality) and number of different values for the problem variables
- Running-time: number of operations the algorithm takes to terminate
    - Worst-case as a function of problem size
    - Polynomial: n2
    - Super-polynomial: nO(log log n)
    - Exponential: 2n
- Problem reduction: transforming current problem into another via mapping

The hardness / complexity of a problem can now be classified:

- Class P: some algorithm can **3** the problem in polynomial time (worst-case running-time for problem size n is less than F(n) for some polynomial formula F)
- Class NP: problem can be solved and any solution can be **verified** within polynomial time by some algorithm Note: P is a subset of NP
- Class NP-complete: problem belongs to class NP and any other problem in NP can be reduced to this problem by an algorithm running in polynomial time
- Class NP-hard: problem is at least as hard as any other problem in NP complete but solution cannot necessarily be verified within polynomial time



Chapter 2

Motivation for Evolutionairy Algorithms:

- Remember the one regarding the evolution of intelligence – this is “high level”
- There is another one regarding (heuristic) problem-solving – this is more practical
- ROBUST PROBLEM SOLVING technology needed, i.e., algorithms that work on a wide range of problems without much problem-specific adjustment

Darwinian Evolution: Survival of the fittest

- All environments have finite resources → *limited population*
- Life forms have basic instinct/ lifecycles geared towards reproduction
- Therefore some kind of selection is inevitable

- Those individuals that compete for the resources most effectively have increased chance of reproduction
- Note: fitness in natural evolution is a derived, secondary measure, i.e., we (humans)assign a high fitness to individuals with many offspring

Phenotypic traits: Physical & behavioural differences that affect response to environment
- Partly determined by inheritance, partly by factors during development (nature vs. nurture)
- Unique to each individual, partly as a result of random changes

If phenotypic traits:
• Lead to higher chances of reproduction
• Can be inherited
→ they will tend to increase in subsequent generations, leading to new combinations of traits

Population consists of diverse set of individuals
• Combinations of traits that are better adapted tend to increase representation in population
Individuals are "units of selection"
• Variations occur through random changes yielding constant source of diversity, coupled with selection means that: **Population is the "unit of evolution"**
• Note the absence of "guiding force"

Can envisage population with n traits as existing in a n+1-dimensional space (landscape) with height corresponding to fitness
• Each different individual (phenotype) represents a single point on the landscape
• Population is therefore a "cloud" of points, moving on the landscape over time as it evolves – adaptation

Selection "pushes" population up the landscape
Genetic drift:
• random variations in feature distribution
• (+ or -) arising from sampling error
• can cause the population "melt down" hills, thus crossing valleys and leaving local optima

The information required to build a living organism is coded in the DNA of that organism
- Genotype (DNA inside) determines phenotype (outside)
- The mapping genes à phenotypic traits is very complex
    - One gene may affect many traits (**pleiotropy**)
    - Many genes may affect one trait (**polygeny**)
- Small changes in the genotype lead to small changes in the organism (e.g., height, hair colour)

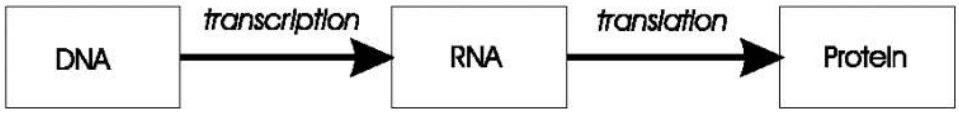
The complete genetic material in an individual's genotype is called the genome
- Bij mensen is 99% van de genen het zelfde
- Human DNA is organised into 23 chromosomes pairs

Gametes (sperm and egg cells) contain 23 individual chromosomes rather than 23 pairs

- Cells with only one copy of each chromosome are called **haploid**
- Gametes are formed by a special form of cell splitting called **meiosis**
- During meiosis the pairs of chromosome undergo an operation called **crossing-over**
  Note: biologists use “crossing-over”, EC folks use “crossover”
- New person cell (**zygote**)

Mutation:

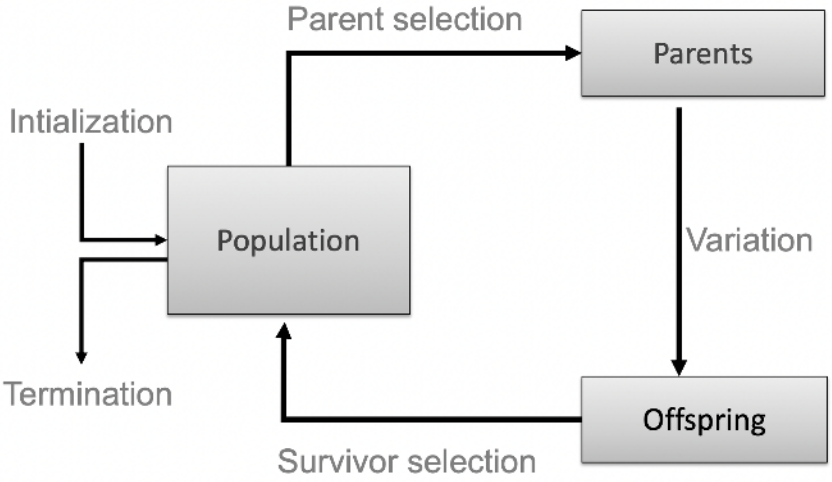
- Catastrophic: offspring is not viable (most likely)
- Neutral: new feature does not influence fitness
- Advantageous: strong new feature occurs



Chapter 3

Twee strijdende krachten

- Increase population diversity through mutation & recombination → Push towards novelty
  - Individual level
- Decrease population diversity through selection → push quality
  - Population level



Representation

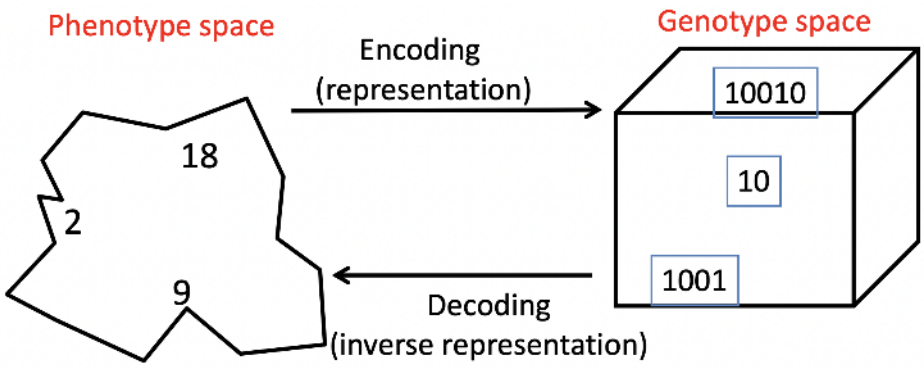
Role: provides code for candidate solutions that can be manipulated by variation operators
Leads to two levels of existence

1. phenotype: object in original problem context, the outside
2. genotype: code to denote that object, the inside (chromosome, “digital DNA”)

Implies two mappings:

1. Encoding : phenotype à genotype (not necessarily one to one)
2. Decoding : genotype à phenotype (must be one to one)

Chromosomes contain genes, which are in (usually fixed) positions called loci (**sing. locus**) and have a value (**allele**)



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

Evaluation / fitness function

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
Ook wel: 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

Population

Role: holds the candidate solutions of the problem as individuals (genotypes)
Formally, a population is a multiset of individuals, i.e. the same element might occur multiple times

- 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

- Diversity of a population refers to the number of different fitness values / phenotypes / genotypes present (note: not the same thing)
- 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

Selection

Role:

- Identifies individuals • to become parents • to survive
- Pushes population towards higher fitness
- Usually probabilistic / stochastic
    - High quality solutions more likely to be selected than low quality
    - But not guaranteed • even worst solution usually has non-zero probability
    - This stochastic nature can help escape from local optima

E.G. roulette wheel selection = fitness / total fitness

Survivor selection a.k.a. Replacement

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, e.g., elitism (the best n individuals always survive) is a deterministic rule.

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 multi-parent reproduction, is possible, seldom used in EC

There has been much debate about relative importance of recombination and mutation

Nowadays most EAs use both – pragmatic attitude is advisable

Variation operators must match the given representation

Mutation

Role: causes small, random variations

• Acts on one genotype and delivers another

• Element of randomness is essential and differentiates it from other heuristic operators

Recombination

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, maar hoop op verbetering
- Principle has been used for millennia by breeders of plants and livestock

Initialisation / Termination

Initialisation is usually done at random,

- Need to ensure even spread and mixture of possible alleles (values)
- Can include existing solutions, or use problem-specific heuristics, to “seed” the population

Termination condition checked every generation, e.g:

- Reaching some (known/hoped for) fitness level
- Reaching some maximum number of generations
- Reaching some minimum level of diversity
- Reaching some specified number of generations without fitness improvement (stagnation)

Different types of EAs

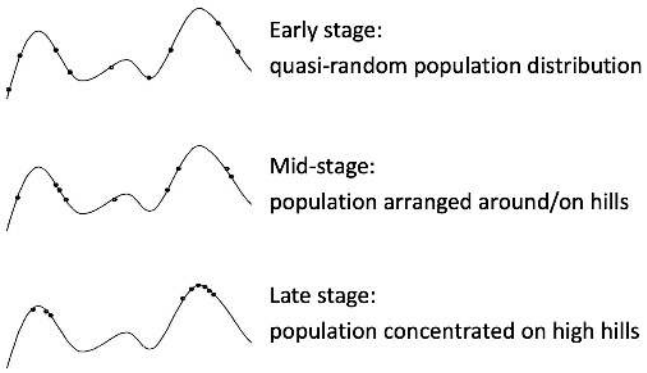
These historical differences are largely irrelevant, best strategy

- ☐ choose representation to suit problem
- ☐ choose variation operators to suit representation

The 8-queens problem

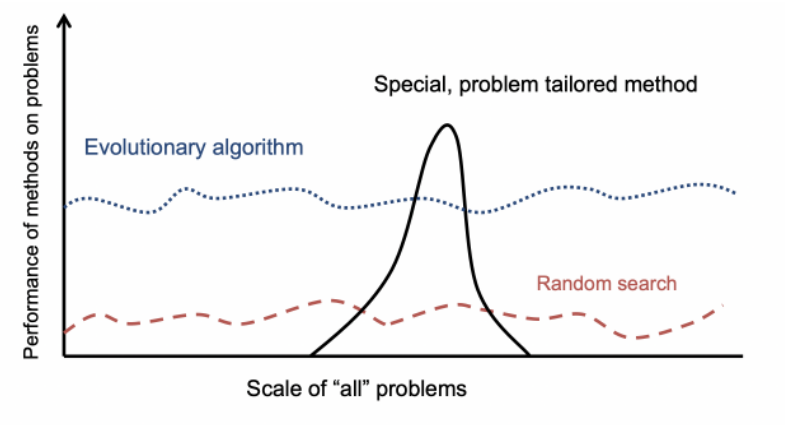
Representation

Je stopt op elke line 1 queen → zo heb je in ieder geval al 1 oplossing. Dan heb je 8 waardes met op welke hoogte de andere queens staan. Geeft een penalty voor het aantal queens die een queen checkt.



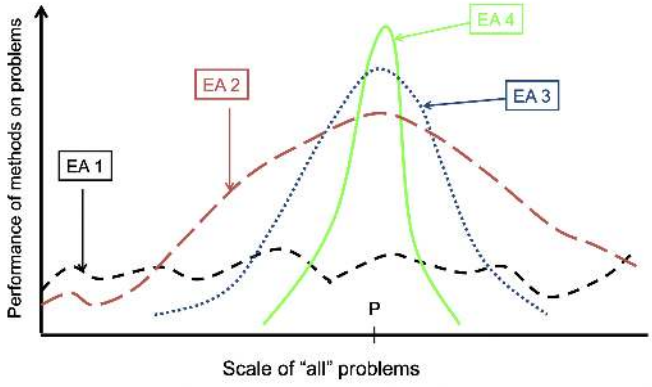
Evolutionary Algorithms in context

- EAs fall into the category of generate-and-test algorithms, a.k.a. trial-and-error algorithms
- They are stochastic, population-based search algorithms
- 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
- EAs are meant to provide robust tool that shows:
  - evenly good performance
  - over a range of problems and instances



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



Variation operators need to match the representation, selection operators are independent from the representation (hence, from the problem at hand)

Chapter 4

Index:

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

Role of representation and variation operators
First stage of building an EA and most difficult one: choose a good representation for the problem

Two sides of representation

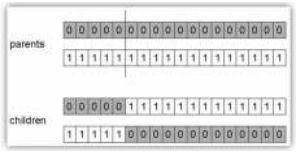
Good representation

- **<u>Small changes to a genotype induce small changes in the corresponding phenotypes (Strong causality principle)</u>**
    - Gray coding helpt het verschil tussen binary en integer verschil in verschillen
- Lack of this causes discontinuities
- Searchability

1. Binary Representation

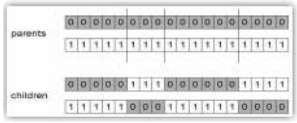
1-point crossover:

- Kies 1 plek en swap voor de kinderen
- Positional Bias



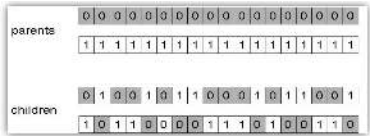
n-point crossover:

- Meer punten



Uniform crossover:

- Random kans bij elke bit
- Inheritance is independent of position



Zowel crossover als mutation is goed, niet 1 van beide

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

- Crossover is explorative, makes a big jump to areas somewhere “between” two parents

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

- Mutation is exploitative, creates random small variations, thereby staying near the parent

→ There is co-operation AND competition between them

2. Integer Representation

- Integer values vs categorical values
- N-point / uniform crossover operators work as previously
- Bit-flip mutation can be generalized to random resetting: with probability pm a new value is chosen at random
- Creep mutation: Add a small value to each gene with probability p

3. Real-Valued or Floating-Point Representation

A. Mutation

- ☐ Uniform mutation: the values of .. are drawn uniformly randomly from a range. This is analogous to bit-flipping for binary encodings and normally used with a position-wise mutation probability.

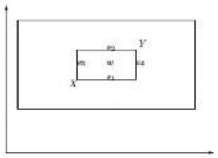
- ☐ Nonuniform mutation: similar to the creep mutation for integers. It is assigned so that usually, but not always, the amount of change introduced is small.
  → Gaussian distribution or Cauchy distribution
- ☐ Self-adaptive mutation: the essential feature is that the **step sizes evololve** are also included in the chromosomes and they themselves undergo variation and selection. Hence, ' coevolves with the solutions. The idea is that different mutation strategies would be appropriate in different stages. Self-adaptation can then be a mechanism adjusting the mutation strategy as the search is proceeding.
  - **Order is important: – first sigma then x**
  - Rationale: is is evaluated twice
    - Primary: x' is good if f(x') is good
    - Secondary: s' is good if the x' it created is good

  1. Uncorrelated mutation with one step size o The same distribution is used to mutate each . Therefore, we only have one strategy parameter '.
  2. Uncorrelated mutation with n step sizes. The idea is that one can treat dimensions differently. We want to be able to use different step sizes for different dimensions. After all, the fitness landscape can have a different slope in one direction.
  3. Correlated mutation. The idea is to allow ellipses to have any orientation by rotating them.

B. Recombination
Discrete: Uniform or n-point
Intermediate: children "between" parents hence arithmetic recombination

Arithmetic recombination
- zi = a xi + (1 - a) yi where a : 0 £ a £ 1.
- The parameter a can be:
  - constant: *uniform arithmetical crossover* (bvb 0.5)
  - variable (e.g. depend on the age of the population)
  - picked at random every time
- Single arithmetic crossover = 1 gen
- Simple arithmetic crossover = vanaf 1 gen alles
- Whole arithmetic crossover = alles

Blend Crossover
Maakt een nieuw allel dicht bij 1 van de ouders maar gebruikt het verschil van tussen de ouders om er toch iets vanaf te liggen



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

Multi-parent recombination, type 1

- Segment and recombine parents
- Diagonal crossover for n parents: • Choose n-1 crossover points

Multi-parent recombination, type 2

- Arithmetical combination of (real valued) alleles

4. Permutation Representation

Example: production scheduling: which elements are scheduled before others (order)
Example: Travelling Salesman Problem (TSP) :which elements occur close other (adjacency)
Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

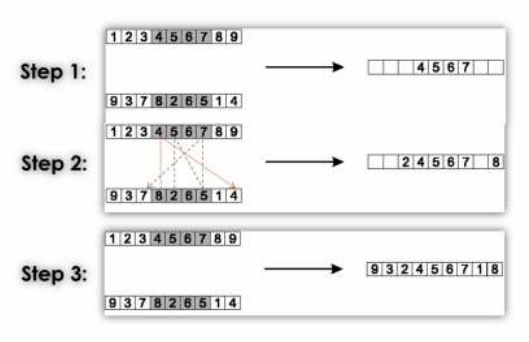
A. Mutations

- Swap mutation

- Insert Mutation
- Scramble mutation — Pak een groepje cellen at random en hussel ze rond
- Inversion mutation — Pak een groepje cellen en draai ze om

B. Crossover

- Order 1 crossover
- Partially Mapped Crossover (PMX)



- Copy randomly selected set from first parent

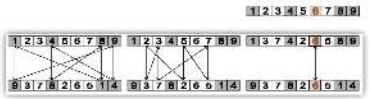
- Copy rest from second parent in order 1,9,3,8,2

- Cycle crossover
- Edge Recombination

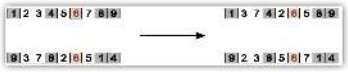
Informal procedure: once edge table is constructed

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

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



4. Tree Representation

In GA, ES, EP, chromosomes are linear structures (bit strings, integer string, realvalued vectors, permutations)

In 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

- Most common mutation: replace randomly chosen subtree by randomly generated tree
- Mutation has two parameters:
  • Probability pm to choose mutation
  • Probability to chose an internal point as the root of the subtree to be replaced
    - Remarkably pm is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf)
    - The size of the child can exceed the size of the parent

- Representation is essential when designing an EA
- A few data structures are enough to represent many (all?) problems, e.g., binary, integer, real-valued vectors, trees
- For any given problem there can be more suitable representations. Some are better than others – remember the strong causality principle !
- Reproduction operators must fit the representation
- Reproduction operators are stochastic
- Reproduction operators can be distinguished by arity
  - Unary: mutation
  - Binary: recombination, crossover
  - N-ary: multi-parent recombination
- Self-adaptive mutation is a powerful mechanism. But remember to change the sigma first!

Chapter 5

Population Management
Two different models:

1. Generational model
    - Each individual survives for exactly one generation
    - The entire set of parents is replaced by the offspring
2. Steady-state model
    - A few new individuals (children) are generated per generation
    - A few old members of the population are replaced

Generation Gap = The proportion of the population replaced

- = 1.0 means Generational EA,
- < 1.0 means Steady State EA

Selection operators work on whole individuals, hence they are representation **independent**

A. Fitness-Proportionate Selection (FPS)
:: Meer kans bij een hogere fitness
This method has problems including:
1. *Premature Convergence:* One highly fit member can rapidly take over, if the rest of the population is much less fit and population members become (nearly) identical. When this happens too early, we can end up in a local optimum.
2. Loss of selection pressure: At the end of runs when fitness values are similar, EA can degrade to random search
3. Highly sensitive to function transposition

Parent Selection: Scaling

- Windowing = subtract the worst fitness score from the fitness scores
- Substract mean fitness score (*s and constant) of the fitness scores, if negative then 0

→ these solutions are used for the first problem

B. Rank-based Selection
Attempt to remove problems of FPS by basing selection probabilities on relative rather than absolute fitness
Best rank = population size - 1,worst rank = 0

Linear Ranking - Parameterised by factor s: $1 < s \leq 2$, hoe lager hoe meer kans worst maakt
Exponential Ranking - More selection pressure
Sampling algorithms

- Roulette wheel alg.
- Stochastic universal sampling. Whenever more than one sample is to be drawn from the distribution the use of the stochastic universal sampling (SUS) algorithm is preferred

C. Tournament Selection

All methods above rely on global population statistics

- Could be a bottleneck especially on structured or very large populations
- Relies on external fitness function which might not exist: e.g., evolving game playing strategies

Idea for using only local fitness information:

- Pick k members at random then select the best of these
- Repeat to select more individuals

Probability of selecting individual 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

C. Uniform Selection

Gewoon een kans om geselecteerd te worden.

Survivor Selection

1. Fitness-based selection
   - Elitism (beste zoveel houd je)
   - Delete worst (large popu)
   - Round-robin tournament
     - Elke solutions tegen 1 aantal andere solutions, m aantal met meeste wins mag door naar volgende generatie
2. Age-based selection
   - “delete-random” (not recommended) or as first-in-first-out (a.k.a. delete-oldest)

(μ,l)-selection a.k.a. “**comma** strategy”

- Often used because can forget solutions
  - Leave local optima
  - using the + strategy bad s values can survive in áx,sñ too long if their host x is very fit

(μ+l)-selection a.k.a. “**plus** strategy”

The **takeover time** is the number of generations it takes until the application of selection completely fills the population with copies of the best individual.

Multimodality = Most interesting problems have more than one locally optimal solution.

Genetic drift = Finite population with global mixing (e.g., panmictic system, where everybody can mate with everybody) and selection eventually converges around one optimum

Approaches for Preserving Diversity

Explicit Approaches for Preserving Diversity: Fitness Sharing

- Where sh(d) is: 1 d/share. Thus, with 2 neighbours at distance 1 and 2 AND the point itself at distance 0, and lastly with of 3, we get sh(d) = (1 0) + (1 1/3) + (1 2/3) = 2. Then, if the original fitness was 2, the new fitness is now 2/2, thus 1.

Crowding

- take 2 parents and their 2 offspring
- set up parent vs. child tournaments such that the intertournament distances are minimal, that is, number the two p's (parents)and the two o's (offspring) such that
- d(p1,o1) + d(p2,o2) < d(p1,o2) + d(p2,o1)
- and let o1 compete with p1 and o2 compete with p2

Implicit Approaches for Preserving Diversity: Automatic Speciation

- Restrict mating to genotypically / phenotypically similar individuals or
- Restrict mating to individuals that have the same (or very similar) tag, where
    - A tag is an extra bit (or bits) in the genotype that • is initialized randomly and
    - is subject to recombination and mutation
- Island Model Parallel EAs
    - Run multiple populations in parallel
    - After a (usually fixed) number of generations –called an epoch– exchange individuals with neighbours
    - How often to exchange individuals? Niet te snel, anders heb je zelfde populatie maar ook niet te langzaam anders most authors use a range of 25-150 generations
    - How many individuals, which individuals to exchange ? usually ~2-5, but depends on population size. better to exchange randomly selected individuals than best individuals • "multi-culti" is even better (exchange most different ones)
- Cellular EAs
    - Individuals on a grid
    - Selection (hence recombination) and replacement happen using concept of a neighbourhood a.k.a. deme

Equivalent of 1 generation is:

- pick individual in pop at random
- pick one of its neighbours using roulette wheel
- crossover to produce 1 child, mutate
- replace individual if fitter
- circle through population until done

Different spaces

Genotype space (always) • Set of representable solutions

Phenotype space (always; it may be = genotype space)

- The object coded by the genotype
- Neighbourhood structure may bear little relation with genotype space

Fitness space (if quantifiable fitness is given)

- The space of fitness values: (multi-dimensional) real numbers

Algorithmic space (if a spatially structured EA is used)

• The structure of the population
• Akin to the geographical space on which life on earth has evolved

Q: Which of these distances exist in cellular EAs?

Chapter 6 - 1

A. Genetic Algorithms

- ☐ discrete function optimization
- ☐ benchmark
- ☐ straightforward problems binary representation
- not too fast
- missing new variants (elitsm, sus)
- often modelled by theorists

Holland's original GA is now known as the simple genetic algorithm (SGA)

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional – implemented by Roulette Wheel
Survivor selection	Generational

Has been subject of many (early) studies
• still often used as benchmark for novel GAs
• Shows many shortcomings, e.g.,
Representation is too restrictive
• Mutation & crossover operators only applicable for bit-string & integer representations
• Selection mechanism sensitive for converging populations with close fitness values
• Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

Good for combinatorial optimisation, constraint satisfcation • Many different variants – lots of design choices • Large academic fanbase • The name GA is often (ab)used for a different EA

B. Evolution Strategies

Typically applied to: • numerical optimisation

- ☐ fast
- ☐ good optimizer for real-valued optimisation
- ☐ relatively much theory
- ☐ Special: • self-adaptation of (mutation) parameters standard
- ☐ s is varied on the fly by the "1/5 success rule":

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	(μ,λ) or $(\mu+\lambda)$

The jet nozzle experiment with ES
Recombination ES

- Creates one child
- Averaging or selecting

Parent selection:

- Parents are selected by uniform random distribution whenever an operator needs one/some
- Random

	Two fixed parents	**Two parents selected for each i**
$z_i = (x_i + y_i)/2$	Local intermediary	Global intermediary
z_i is x_i or y_i chosen randomly	Local discrete	Global discrete

Given a dynamically changing fitness landscape (location of the optimum is shifted every 200 generations)

- Self-adaptive ES is able to – follow the optimum and – adjust the mutation step size after every shift

ES: Prerequisites for self-adaptation

μ > 1 to carry different strategies • l > μ to generate offspring surplus • (μ,l)-selection to get rid of misadapted s‘s • Mixing strategy parameters by (intermediary) recombination on them

Summary • Very good for numerical optimisation • Can work with • low computational budgets (low maximum of fitness evaluations) and • high dimensional problems (large number of variables) • Self-adaptation of (mutation) parameters standard • Fewer parameters and design choices than in GAs • Many applications in business and industry

C. Evolutionary Programming

- ☐ very open framework: any representation and mutation op’s OK
- ☐ crossbred with ES (contemporary EP)
- ☐ consequently: hard to say what “standard” EP is
- ☐ Special: • no recombination • self-adaptation of parameters standard (contemporary EP)

Representation	Real-valued vectors
Recombination	None
Mutation	Gaussian perturbation
Parent selection	Deterministic (each parent one offspring)
Survivor selection	Probabilistic (μ+μ)

EP: Prediction by finite state machines
• Less clear identity than GA and ES • Modern versions are hard to distinguish from ES • Branch is becoming less active / visible than other members of the EA family

C. Genetic Programming

- ☐ competes with neural nets and other ML algorithms
- ☐ needs huge populations (thousands) → slow

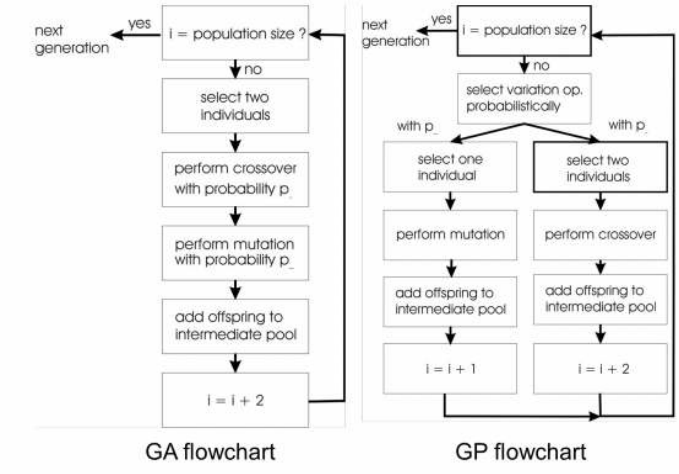
Special:

- ☐ non-linear chromosomes: trees, graphs
- ☐ mutation possible but not necessary

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Compare

- GA scheme using crossover AND mutation sequentially (be it probabilistically)
- GP scheme using crossover OR mutation (chosen probabilistically)



Bloat = “survival of the fattest”, i.e., the tree sizes in the population are increasing over time

Chapter 6 - 2

A. Differential Evolution: Quick overview

Typically applied to:

- Nonlinear and non differentiable real valued functions (hard problems)

Attributed features:

- populations are lists (sets where elements have a position)
- *four* parents are needed to create a new individual
- different variants exists due to changing the base vector
- **differential mutation**

Step 1: create a mutant vector population

$$v_i = a_i + F \times (b_i - c_i)$$

ai, bi, ci zijn random gekozen uit de populatie

Step 2: create trial vector population by crossover

Create ui by uniform crossover between vi and xi

Step 3: deterministic selection applied to each pair xi and ui

Beste pair

Different variants exists due to changing the base vector described as DE/a/b/c where a, b c are the placeholders for

- a denotes the way to choose the base vector (rand or best)
- b is the number of difference vectors to define perturbation vector.
  E.g., with b = 2 we use two difference vectors and we’d get
- c denotes the crossover scheme (“bin” is uniform crossover)
- DE/rand/1/bin is the notation for the version above

Representation	Real-valued vectors
Recombination	Uniform crossover
Mutation	Differential mutation
Parent selection	Given individual deterministically + Uniform random selection of the 3 necessary other vectors
Survivor selection	Deterministic elitist replacement (parent vs. child)

B. Particle Swarm Optimisation (PSO) Quick overview

Typically applied to:
- Optimising nonlinear functions

Attributed features:
- No crossover
- Every candidate solution carries it own perturbation vector

Every particle (= population member = individual) is a pair of
1. is a position vector (location) in IRn
2. is a velocity vector in IRn

New velocity vector is the weighted sum of 3 components:
- Current velocity vector
- Vector from current position to best past position of this particle
- Vector from current position to best past position of the population

$$\bar{v}' = weight1 \cdot \bar{v} + weight2 \cdot (\bar{y} - \bar{x}) + weight3 \cdot (\bar{z} - \bar{x})$$

Weight1 = w = exploration vs exploitation

For w > 1
- Velocities increase over time • Swarm diverges

For 0 < w < 1
- Particles decelerate

A particle = individual = population member *i* is a 3-tuple $\langle \bar{x}_i, \bar{v}_i, \bar{b}_i \rangle$ of
- a position vector (location) $\bar{x}_i$
- a velocity vector $\bar{v}_i$
- a best position vector of this particle in the past $\bar{b}_i$

Representation	Real-valued vectors
Recombination	None
Mutation	Adding velocity vector
Parent selection	Deterministic (each parent creates one offspring via mutation)
Survivor selection	Generational (offspring replaces parents)

C. Estimation of Distribution Algorithms (EDA)

1. Create initial population randomly
2. Calculate the fitness of each candidate
3. Select a subpopulation (consisting of the best candidates)
4. Select a model to fit a probability distribution over the candidates
5. Sample new population from this estimated distribution
6. Back to step 2

Model-assisted EAs
Idea: x% of fitness evaluations is real, 100-x% is model based

D. Learning Classifier Systems (LCS)

Typically applied to:
- Machine learning tasks working with rule sets
- Giving best response to current state environment

Attributed features:
- Combination classifier system and learning algorithm
- Cooperation (instead of the usual competition) among individuals
- Relies on genetic algorithms
- Two schools: Michigan-style (rule is an individual) vs Pittsburgh-style (rule set is an individual)

Representation
- Classic representation: each rule of the rule base is a tuple {condition:action:payoff} where payoff is the fitness
- Match set: subset of rules whose condition matches the current inputs from the environment
- Action set: subset of the match set advocating the chose action

In later versions, the rule-tuple included an accuracy value, reflecting the system's experience of how well the predicted payoff matches the reward received

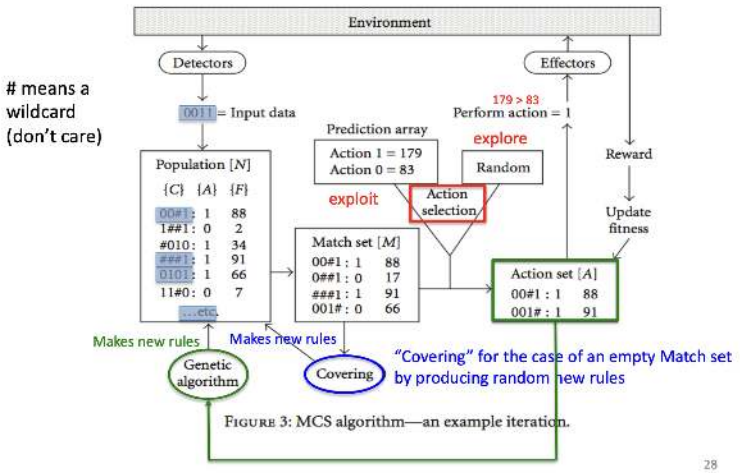


FIGURE 3: MCS algorithm—an example iteration.

28

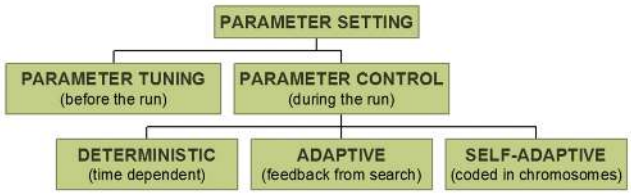
Representation	Tuple {condition:action:payoff,accuracy} conditions use {0,1,#} alphabet
Recombination	One-point crossover on conditions/actions
Mutation	Binary resetting as appropriate on action/conditions
Parent selection	Fitness proportional with sharing within environmental niches
Survivor selection	Stochastic, inversely related to number of rules covering same environmental niche
Fitness	Each reward received updates the predicted pay-off and the accuracy of rules in relevant action sets by reinforcement learning

Important points

- ☐ Some of these have a primary application area: • GA: discrete / combinatorial optimization • ES: continuous optimization • GP: machine learning, modeling

Chapter 7

Parameters and 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

Parameter control: setting values on-line, during the actual run, e.g predetermined time-varying schedule p = p(t)

- using (heuristic) feedback from the search process
- encoding parameters in chromosomes and rely on natural selection

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

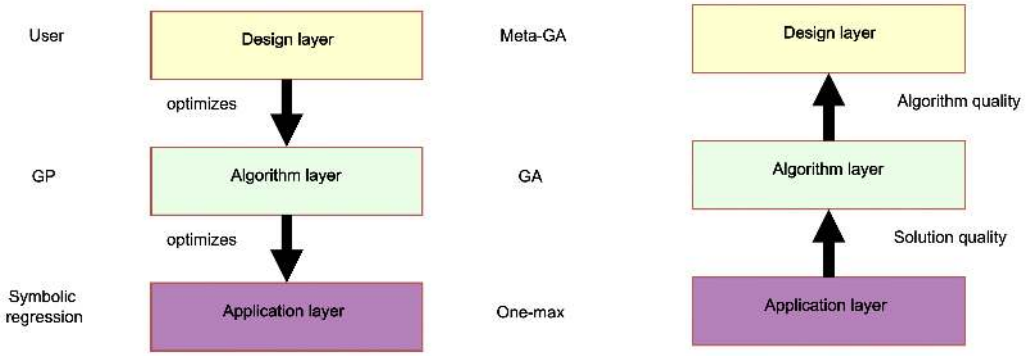
- Parameter control offers the possibility to use appropriate values in various stages of the search
- Adaptive and self-adaptive control can “liberate” users from tuning à reduces need for EA expertise for a new application
- Assumption: control heuristic is less parameter-sensitive thanthe EA

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

Wordt meer en meer aangewerkt:

- Traditional parameters: mutation and xover
- Non-traditional parameters: selection and population size
- All parameters è “parameterless” EAs (name!?)



Control flow Information flow

Parameter – performance landscape

- All parameters together span a (search) space
- One point – one EA instance
- Height of point = performance of EA instance on a given problem
- Parameter-performance landscape or utility landscape for each { EA + problem instance + performance measure }

- This landscape is • not the same as the fitness landscape ! • unlikely to be trivial, e.g., unimodal, separable • If there is some structure in the utility landscape, then we can do better than random or exhaustive search

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

Utility =

- **Mean Best Fitness**
- **Average number of Evaluations to Solution**
- **Success Rate**
- Robustness, … • Combination of some of these

Advantages of tuning
• Easier • Most immediate need of users • Control strategies have parameters too à 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

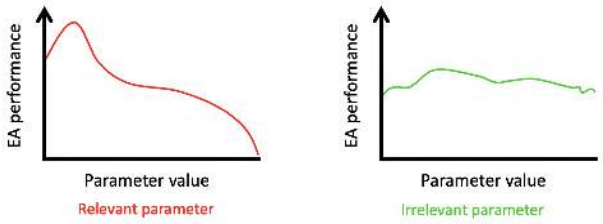
EA tuning is a search problem itself → Straightforward approach: generate-and-test

Testing parameter vectors

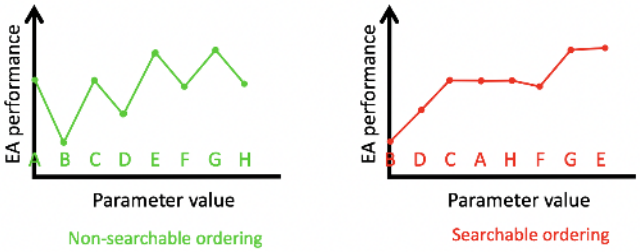
- Run & Record EA performance in each run e.g., by
  - Solution quality = best fitness at termination
  - Speed ≈ time used to find required solution quality
- EAs are stochastic à repetitions are needed for reliable evaluation à we get statistics

A. Numeric parameters

Sensible distance metric à searchable



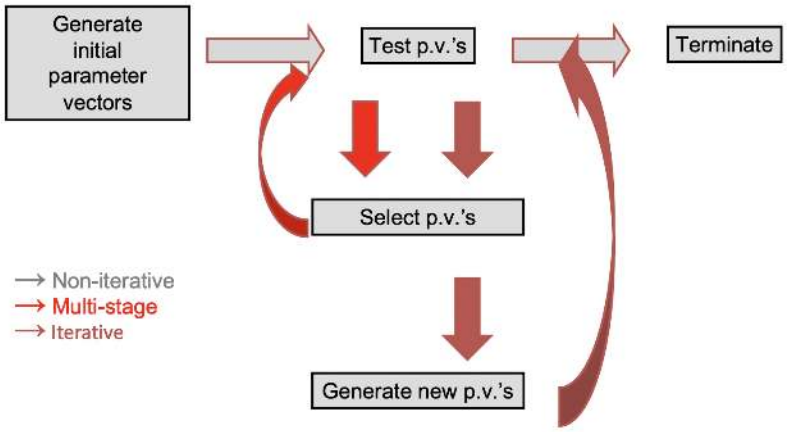
B. Symbolic parameters



No sensible distance metric à non-searchable, must be sampled

Notes on parameters

- A given value of a symbolic parameter can introduce a numeric parameter, e.g.,
  - Selection = tournament à tournament size
  - Populations_type = overlapping à generation gap
- Hence, parameters can have a hierarchical, nested structure
- Hence, number of EA parameters is not defined in general
- Cannot simply denote the design space / tuning search space by
- S = Q1 x … Qm x R1 x … x Rn with Qi / Rj as domains of the symbolic/numeric parameters



Total amount of computational work is determined by
A = number of vectors tested
B = number of tests per vector
C = number of fitness evaluations per test
Tuning methods can be positioned by their rationale:
To optimize A (iterative search)
To optimize B (multi-stage search)
To optimize A and B (combination)
To optimize C (non-existent)

Optimize A:
Applicable only to numeric parameters

Number of tested vectors not fixed, A is the maximum (stop cond.) Population-based search: Initialize with N << A vectors and Iterate: generating, testing, selecting p.v.'s

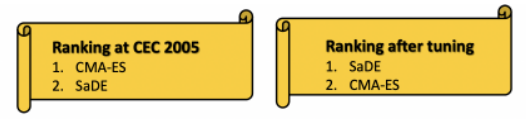
Optimize B:
Applicable to symbolic and numeric parameters
Number of tested vectors (A) fixed at initialization
Set of tested vectors can be created by

- regular method à grid search
- random method à random sampling
- exhaustive method à enumeration

Complete testing: nr. of tests per vector = B (thus, not optimizing)
Selective testing: nr. of tests per vector varies, ≤ B
Idea:

- Execute tests in a breadth-first fashion (stages), all vectors X < B times
- Stop testing vectors with poor utility (lost cause)

Which tuning method?
Differences between tuning algorithms • Maximum utility reached • Computational costs • Number of their own parameters – overhead costs • Insights offered about EA parameters (probability distribution, interactions, relevance, explicit model…)
Similarities between tuning algorithms • Can find good parameter vectors • Nobody is using them



The (near) future of automated tuning

- Hybrid methods for A & B
- Well-funded EA performance measures, multi-objective formulation à multi-objective tuner algorithms
- (Statistical) models of the utility landscape à more knowledge about parameters
- Open source toolboxes
- Distributed execution
- Good testbeds
- Adoption by the EC community
- Rollout to other heuristic methods with parameters

Chapter 8

Parameter control

A. Mutation rato
1. Use a function defined by some heuristic rule and a given measure of time t. The given parameter changes according to a fully deterministic scheme.
2. Incorporate feedback from the search process. For example Rechenberg's 1/5 success rule: the ratio of successful mutations to all mutationts should be 1/5. Is the ratio bigger, then the step size should be increased, etc. Changes in the parameter values are now based on feedback from the search.
3. Assign an individual mutation step to each solution and make these co-evolve with the values encoding the candidate solutions à ⟨!-, … , !., '⟩. In this way, not only the solution vector values (!!) but also the mutation step size of an individual undergo evolution. Note that we can also use a separate '! for each !!. Then, we are co-evolving n parameters of the EA instead of 1.

3 = elk individu een s
4 = elke parameter binnen een individu krijgt een s

Varying penalties
1.

Various forms of parameter control can be distinguished by:
- ☐ primary features:
  - ☐ what component of the EA is changed
  - ☐ how the change is made
- ☐ secondary features:
  - ☐ evidence/data backing up changes
  - ☐ level/scope of change

Three major types of parameter control:
- ☐ deterministic: some rule modifies strategy parameter without feedback from the search (based on some counter)
  - ☐ time or nr. of evaluations (deterministic control)
- ☐ adaptive: feedback rule based on some measure monitoring search progress
  - ☐ population statistics (adaptive control), progress made, population diversity, gene distribution, etc.
- ☐ relative fitness of individuals created with given values (adaptive or self-adaptive control)
- ☐ self-adaptative: parameter values evolve along with solutions; encoded onto chromosomes they undergo variation and selection

- Absolute evidence: predefined event triggers change, e.g. increase pm by 10% if population diversity falls under threshold x • Direction and magnitude of change is fixed
- Relative evidence: compare values through solutions created with them, e.g. increase pm by x% if top x% offspring came by high mutation rates, decrease otherwise • Direction and magnitude of change is not fixed

	Deterministic	Adaptive	Self-adaptive
Absolute	+	+	-
Relative	-	+	+

The parameter may take effect on different levels:
• environment (fitness function) • population • individual • sub-individual

Various forms of parameter control can be distinguished by:

	$\sigma(t) = 1-0.9^{*}t/T$	$\sigma' = \sigma/c$, if $r > 1/5$ ...	$(x_1, ..., x_n, \sigma)$	$(x_1, ..., x_n, \sigma_1, ..., \sigma_n)$	$W(t) = (C^{*}t)^{\alpha}$	$W'=\beta^{*}W$, if $b_i \in F$	$(x_1, ..., x_n, W)$
What	Step size	Step size	Step size	Step size	Penalty weight	Penalty weight	Penalty weight
How	Deterministic	Adaptive	Self-adaptive	Self-adaptive	Deterministic	Adaptive	Self-adaptive
Evidence	Time	Successful mutations rate	(Fitness)	(Fitness)	Time	Constraint satisfaction history	(Fitness)
Scope	Population	Population	Individual	Gene	Population	Population	Individual

Adaptive and self-adaptive parameter control
- offer users “liberation” from parameter tuning
- delegate parameter setting task to the evolutionary process
- the latter implies a double task for an EA: problem solving + self-calibrating (overhead)

Chapter 9

Experiment design
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

Repetitive problems: similar instances with some variations in details - Product Perspective
One-off problems: “unique” requirements and conditions - Design perspective

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

Test problems:
Classic: 5 DeJong functions • Modern: 25 “hard” objective functions

Algorithm design

Getting Problem Instances

- Testing on real data
    - A: Relevant D: Moeilijk, Commercially sensitive
- Standard data sets in problem repositories
    - A: Well-chosen, Comparable D: Not real, EAs can be tuned for this specific forms
- Problem instance generators produce simulated data for given parameter
    - A: • 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 •
    - D: • Not real – might miss crucial aspect • Given generator might have hidden bias

Basic rules of experimentation

- EAs are stochastic à → never draw any conclusion from a single run
- Always do a fair competition • use the same amount of resources for the competitors • try different comp. limits (to cope with turtle/hare effect: sommige zijn er snel bij sommige doen er wat langzamer over) • use the same performance measures

Number of evaluations? → Beste time measurements

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

Performance measures
No. of generated points in the search space = no. of fitness evaluations (do not use no. of generations!) • AES: average number of evaluations to solution • SR: success rate = % of runs finding a solution (individual with acceptable quality / fitness) • MBF: mean best fitness at

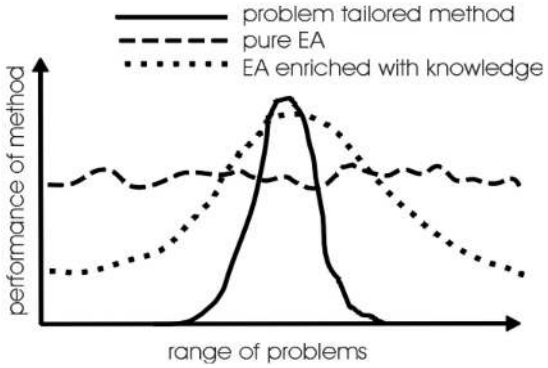
termination, i.e., take the best per run and take the mean of these over a set of runs • SR ¹ MBF • Low SR, high MBF: good approximizer (more time helps?) • High SR, low MBF: “Murphy” algorithm

Fair experiments
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

Chapter 10

Hybridisation: combination of an EA with another (type of) problem solving heuristic

Michalewicz’ view on EAs in context



Memetic Algorithm = The combination of an Evolutionary Algorithm with Local Search Operators that work within the EA loop

Meme = The newest “self-replicator” on Earth, That what is being imitated, Information copied from person to person; which are units of cultural transmission

Initialising Population → Multi-Start Local Search: pick popsize points at random, do some hill-climbing from these, seed initial population with the resulting points

selective initialisation

Local search: Neighbourhood N(x) of point x is the set of points that can be reached from x with one application of a move operator (this is an “operational” definition).
So we have pivot rule, depth (for stopping following the outcomes of the pivot rule) and the move operator.
Pivot rule: condition for stopping neighborhood search
Greedy ascent = stop as soon as a fitter neighbor is found

Steepest ascent = stop after the whole set of neighbours examined and choose the best

Local search can be done in genotype (8 queens, local search on the boar), or phenotype space (8 queens, local search in array of values).
Local search pushes strongly towards better points, thus is subject to quickly losing diversity.

- We an use **Boltzmann selection** where sometimes worse neighbours are selected depending on temperature factor c (high c, very likely; low c, not likely).
- Merz's DPX crossover explicitly generates individuals at same distance to each parent as they are apart

Lifetime adaptation

- ☐ Lamarckian: learned traits CAN be transmitted to o↵spring. Thus, replace individual with fitter neighbour (thus: result of local search replaces the original)
- ☐ Baldwinian: learned traits CANNOT be transmitted to o↵spring. Thus, individual receives the improved fitness from local search but the original member is kept.

Chapter 12

Multi-Objective Problems (MOPs)

- Wide range of problems can be categorised by the presence of n possibly conflicting objectives

Two problems: • finding a set of good solutions • choosing the best solution for a particular application

Paper

TO DO:

- Parameters
- Continuity
- Argumentatie voor enemy group in method

-
- Vergelijk met baseline paper
- Vergelijk met eerdere onderzoeken

How do different crossover approaches influence the optimal number of parents and the performance of an EA in a multi-enemy environment?

Comment:

- Tijd meten we niet toch?
- Waarom presenteren we beter dan de baselin paper

How to share code?
How to document?
Deadline code?
Deadline group work?

Goeie mensen om over te nemen:

- Konstantinos-Sakellariou
https://github.com/Konstantinos-Sakellariou/Evolutionary-Computing-Generalist-Agent-against-Evoman-framework/blob/main/Assign2_generalist.py
- Andere
https://github.com/TeamEightyEight/Assignment1/blob/main/approach1/genetic_optimization.py

python -m scoop multi_parent.py --population_size=25 --num_generations=50 --parallel