Advanced Algorithms

Floriano Zini

Free University of Bozen-Bolzano
Faculty of Computer Science

Academic Year 2013-2014

Lecture 10 – Evolutionary algorithms (cont.)

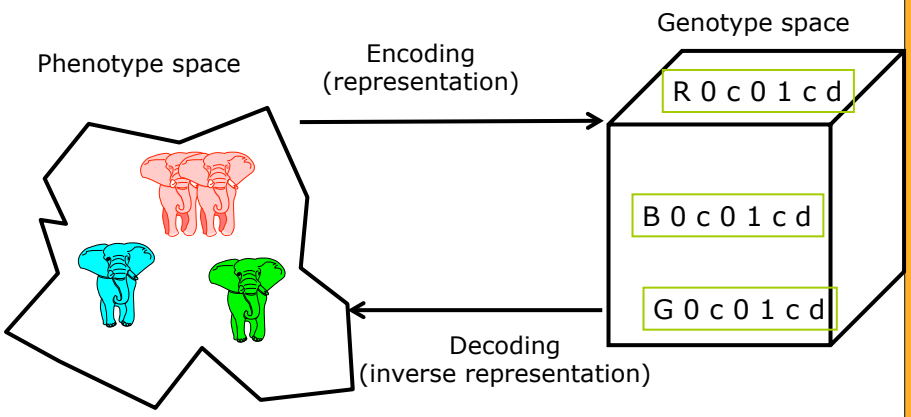
These slides are mainly taken from **A.E. Eiben** and **J.E. Smith**, **Introduction to Evolutionary Computing**

Evolutionary algorithms

EAs are **randomized Monte Carlo** algorithms that can be used to find (approximate) solutions to a **wide range of computational problems** such as:

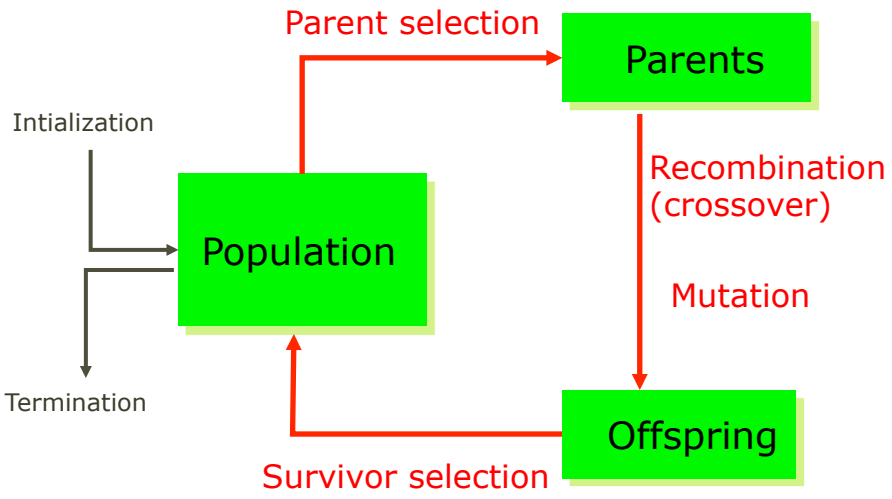
- **Optimization**
  - E.g., university timetabling
- **Modeling**
  - E.g., finding a classification model
- **Simulation**
  - E.g., evolving economical models

Representation



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

General schema of EAs

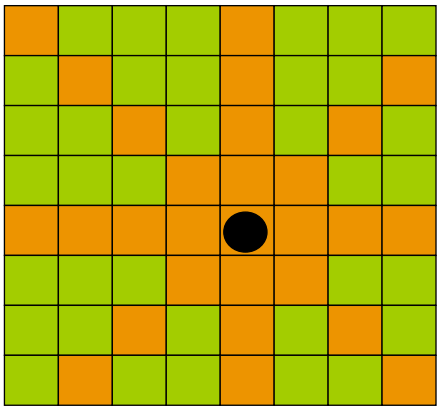


EA schema in pseudo-code

```
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
```

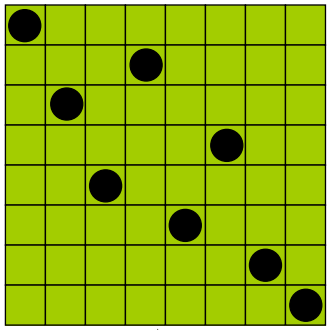
Example: the 8-queens problem

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



The 8-queens problem: representation

Phenotype:
a board configuration



Genotype:
a permutation of the numbers 1 - 8

Obvious mapping

1	3	5	2	6	4	7	8

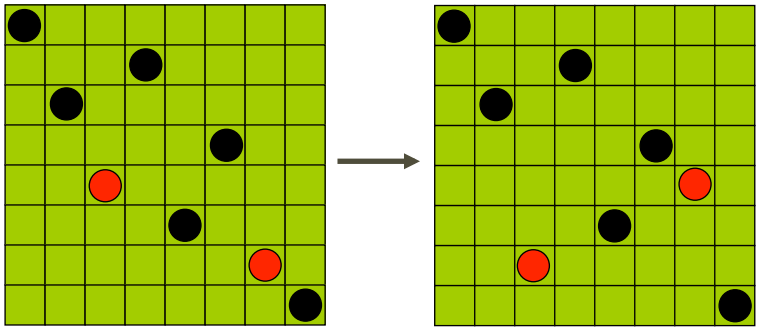
The 8-queens problem: fitness evaluation

- Penalty of one queen: the number of queens she can attack
- 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.:

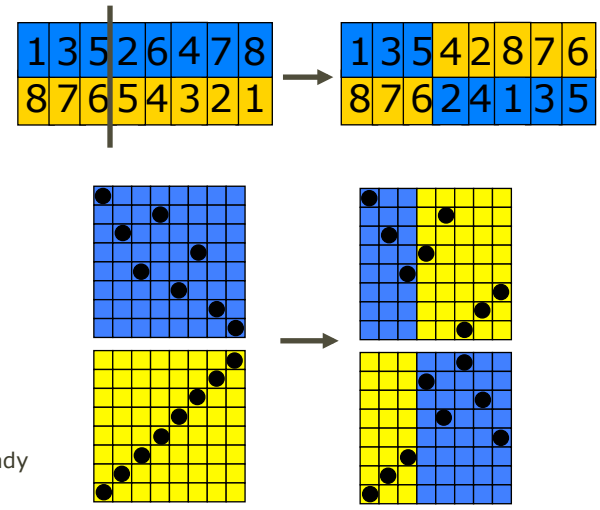
- swapping values of two randomly chosen positions



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 at random and take best two to undergo crossover
- **Survivor selection** (replacement)
  - When inserting a new child into the population, choose an existing member to replace by:
    - sorting the whole population by decreasing fitness
    - enumerating this list from high to low
    - replacing the first with a fitness lower than the given child

8 Queens Problem: summary

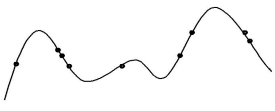
Representation	Permutations
Recombination	"Cut-and-crossfill" crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition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

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

Typical behavior of an EA

Stages in **optimizing** on a **1-dimensional** fitness landscape

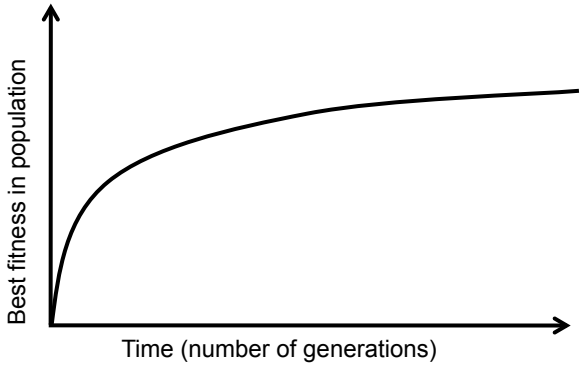
Early stage:
quasi-random population distribution



Mid-stage:
population arranged **around/on hills**

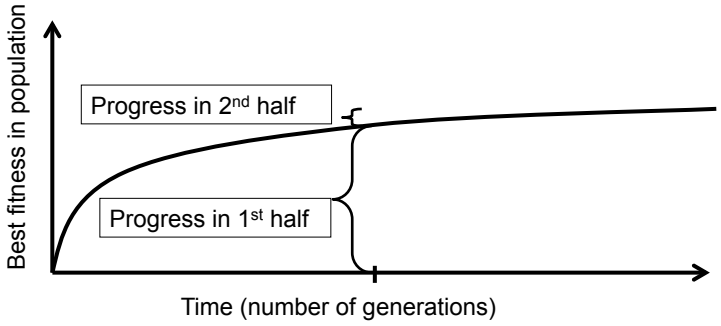
Late stage:
population concentrated on **high hills**

Typical run: progression of fitness



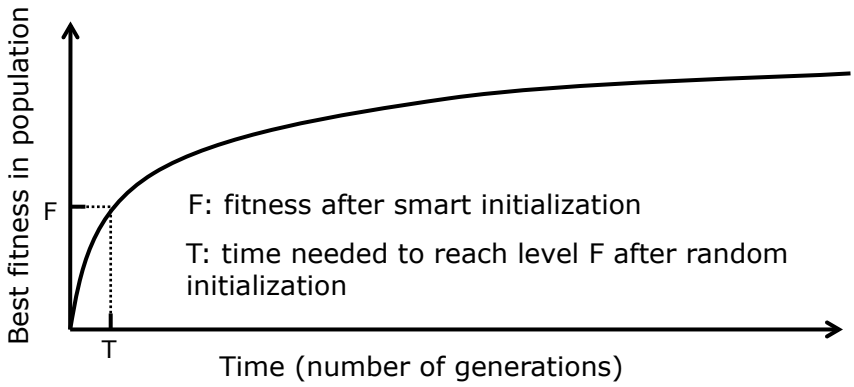
Typical run of an EA shows so-called "**anytime behavior**"

Typical run: progression of fitness



- Are long runs beneficial?
  - It depends on how much you want the last bit of progress
  - May be better to do more short runs

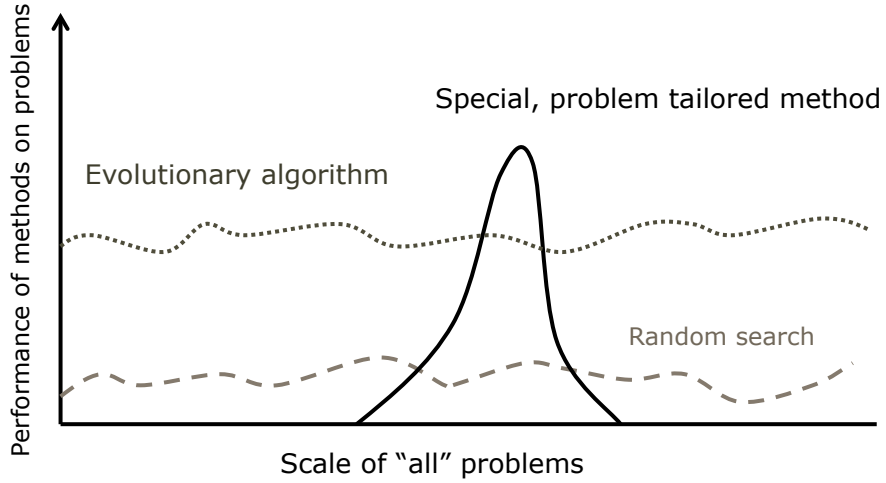
Is it worth expending effort on smart initialization?



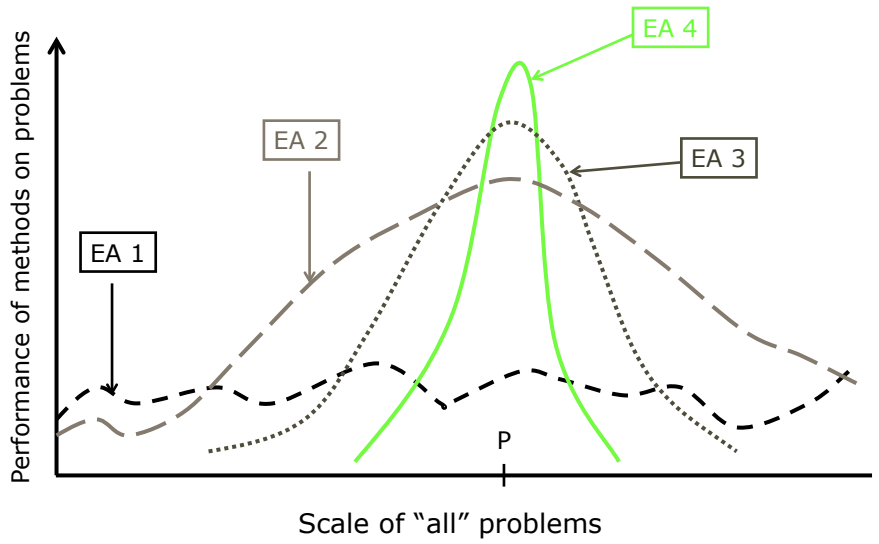
Answer: **it depends**

- Possibly good, if good solutions/methods exist
- Care is needed

EAs as problem solvers: Goldberg view (1989)



EAs as problem solvers: Michalewicz view (1996)

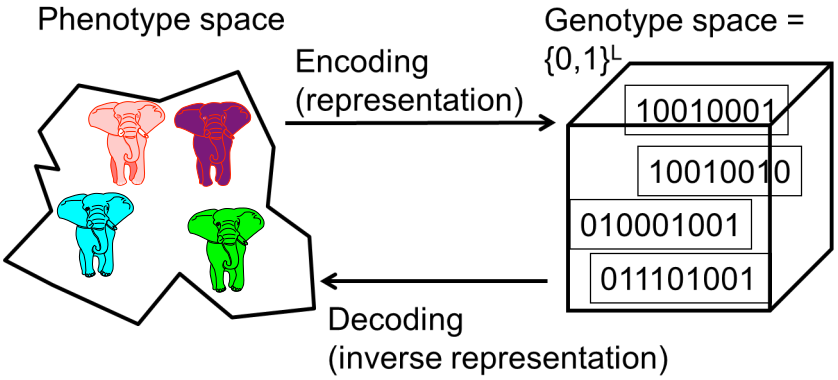


Genetic Algorithms

GA Quick Overview

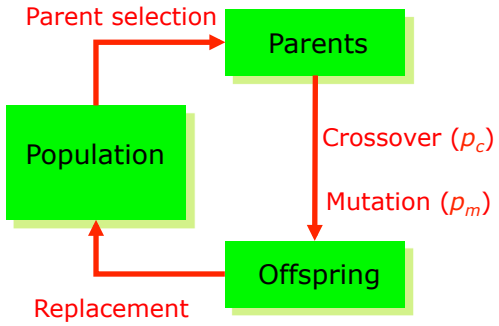
- Developed: USA in the 1970's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Holland's original GA is now known as the simple genetic algorithm (SGA)
- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

Representation



SGA reproduction cycle

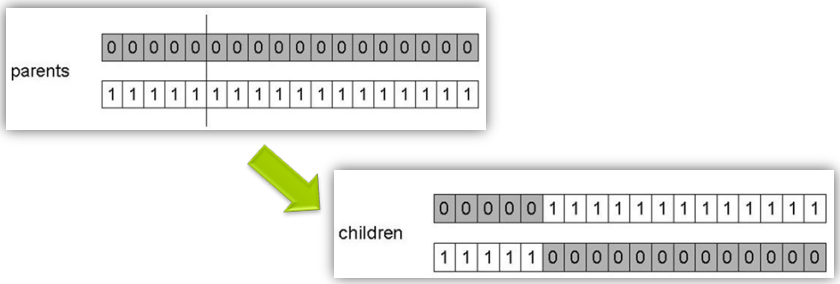
1. Select parents for the mating pool (size of mating pool = population size)
2. Shuffle the mating pool
3. Apply crossover for each consecutive pair with probability p_c, otherwise copy parents
4. Apply mutation for each offspring (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring



|Population| = |Parents| = |Offspring|

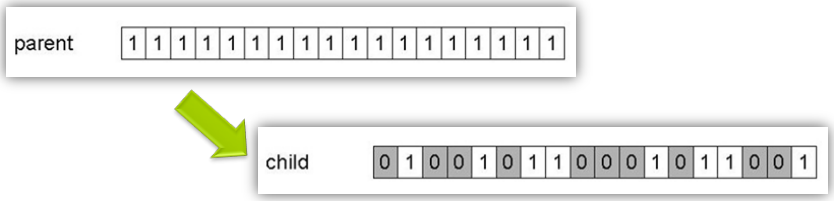
SGA operators: 1-point crossover

- Apply with probability p_c
  - p_c typically in range (0.6, 0.9)
- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails



SGA operators: mutation

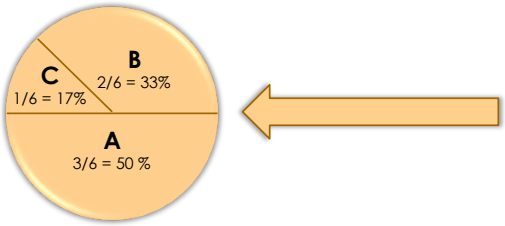
- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/chromosome_length



SGA operators: Selection

Main idea: better individuals get higher chance

- Chances proportional to fitness
- Implementation: roulette wheel technique
  - Assign to each individual a part of the roulette wheel
  - Spin the wheel *n* times to select *n* individuals



Fitness(A)	3
Fitness(B)	2
Fitness(C)	1

An example after Goldberg '89 (1)

- Simple problem: **max x^2 over {0,1,…,31}**
- GA approach:
  - Representation: binary code, e.g., 01101 ↔ 13
  - Population size: 4
  - 1-point crossover, bitwise mutation
  - Roulette wheel selection
  - Random initialization

X^2 example: selection

We show one generational cycle done by hand

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X^2 example: Crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X^2 example: Mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

The simple GA

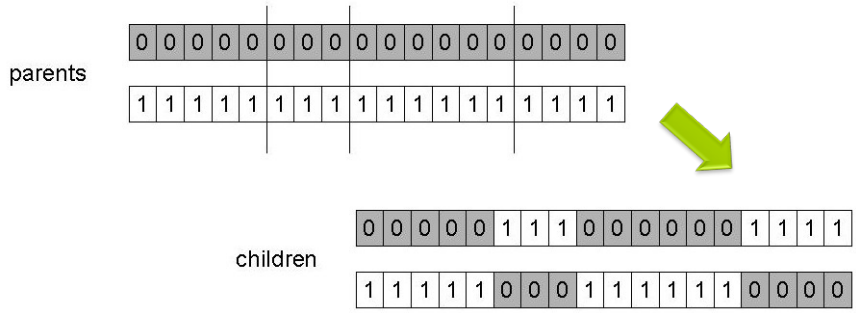
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.,
  - Representation is too restrictive
  - Mutation & crossover operators only applicable for bit-string
  - Selection mechanism sensitive for converging populations with close fitness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

Alternative Crossover Operators

- 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

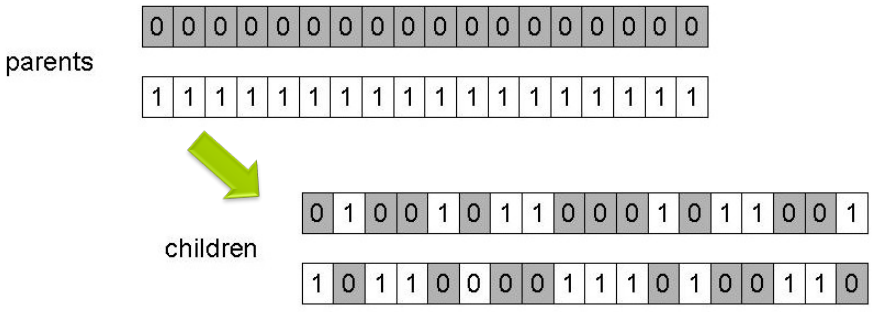
n-point crossover

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



Uniform crossover

- Flip a coin for each gene of the first child
  - Keep the value if 'heads', change the value if 'tails'
  - Make an inverse copy of the gene for the second child
- Inheritance is independent of position



Crossover OR mutation?

- Decade long debate:
  - which one is better?
  - Are both necessary?
- Answer (at least, rather wide agreement):
  - it depends on the problem
  - in general, **it is good to have both**
  - mutation-only-EA is possible, crossover-only-EA would not work

Crossover OR mutation? (cont'd)

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

There is co-operation AND competition between them

- **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

Crossover OR mutation? (cont'd)

- Only **crossover** can **combine information from two parents**
- Only **mutation** can **introduce new information** (alleles)
- To hit the **optimum** you often need a **'lucky' mutation**

Population Models

- SGA uses a **Generational model**:
  - each individual survives for exactly one generation
  - **the entire set of parents is replaced by the offspring**
- At the **other end** of the scale **are Steady-State** models:
  - one offspring is generated per generation
  - **one member of population replaced**
- **Generation Gap**
  - the **proportion** of the population **replaced**
  - makes a parameterized transition between generational and steady-state Gas
  - **gg = 1.0 for SGA, gg = 1/pop_size for SSGA**
- The name SSGA is often used for any GA with a generation gap < 1

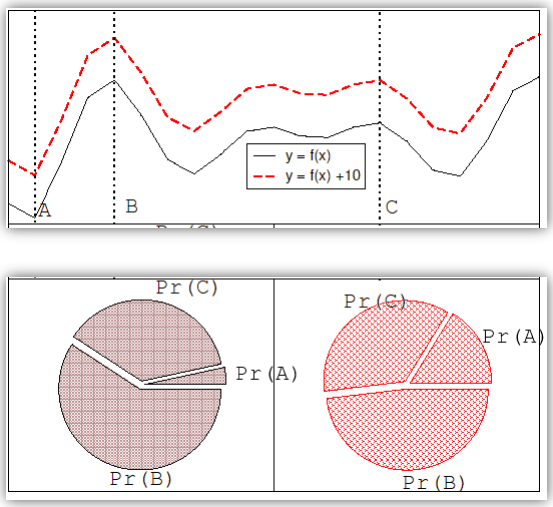
Fitness Based Competition

- **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
  - i.e. they are representation-independent !

Fitness-Proportionate Selection

- 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** (see next slide)

Function transposition for FPS



Fitness-Proportionate Selection

- 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** (see next slide)
- **Scaling** can fix the last two problems
  - **Windowing** $f'(i) = f(i) - \beta$
    - where β is worst fitness in this generation
  - **Sigma Scaling** $f'(i) = \max(f(i) - (\overline{f} - c * \sigma_f), 0.0)$
    - where c is a constant, usually 2.0

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 and worst rank 1)
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Linear Ranking

- Parameterised by factor *s:* $1.0 < s \leq 2.0$
  - measures **advantage of best individual**
  - In SGA this is the number of children allotted to it

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

- Simple 3 member example

	Fitness	Rank	P_{selFP}	P_{selLR} $(s=2)$	P_{selLR} $(s=1.5)$
A	1	1	0.1	0	0.167
B	5	3	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0

Survivor Selection

- Most of **selection methods above** are used for **parent selection**
- **Survivor selection** can be divided into two approaches:
  - **Age-Based Selection**
    - In SGA the population is fully replaced ad each generation
    - In SSGA can implement as "delete-random" (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
  - **Fitness-Based Selection**
    - Using one of the methods above

Two Special Cases of fitness-based survivor selection

- **Replace-worst**
  - The **worst** (in term of fitness) individuals are replaced and each generation by the offspring
- **Elitism**
  - Always keep at least one copy of the fittest solution so far
  - Widely used in both population models (SGA, SSGA)

SGA technical summary tableau

Representation	Binary Strings
Recombination	N-point or uniform crossover
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover