Genetic Algorithms

Chapter 3

Contents of this Chapter

- Introductory example.
- Representation of individuals:
  - Binary, integer, real-valued, and permutation.
- Mutation operator.
  - Mutation for binary, integer, real-valued, and permutation representations.
- Recombination Operator:
  - Recombination for binary, integer, real-valued, and permutation representations.
  - Multiparent recombination.
- Models of population.
- Parent selection:
  - Types of selection: fitness proportional, ranking, selection probabilities, and tournament.
- Survivor selection
  - Age-based, fitness based.

GA Quick Overview

- The most widely known and used EA.
- Developed: USA in the 1970’s.
- Early names: John Holland, Kenneth A. DeJong, David E. Goldberg.
- Typically applied to:
  - Discrete optimization.
- Attributed features:
  - Not too fast, actually, very slow.
  - Good heuristic for combinatorial problems.
- Special Features:
  - Traditionally emphasizes combining information from good parents (crossover).
  - Many variants, e.g., reproduction models, operators.

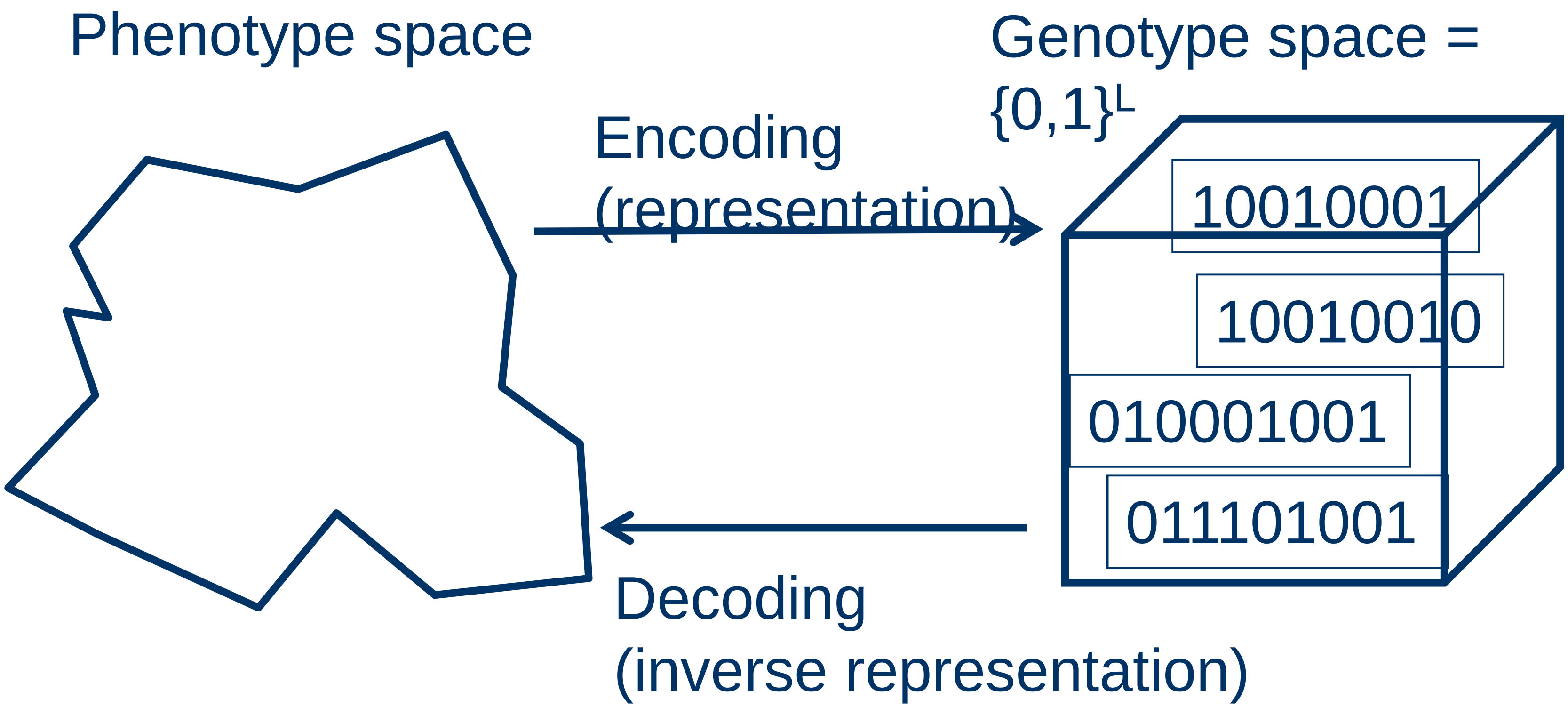
Genetic Algorithms

- Holland original GA is currently known as the Simple Genetic Algorithm (SGA).
- Features of the SGA:
  - Binary representation;
  - Parent selection proportional to the fitness.
  - Low probability of mutation.
  - Genetically inspired recombination.
  - Generational scheme for selection of survivors.
- Other GAs use different:
  - Representations.
  - Mutations.
  - Crossovers.
  - Selection mechanisms.

SGA Technical Summary Table

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

SGA Representation

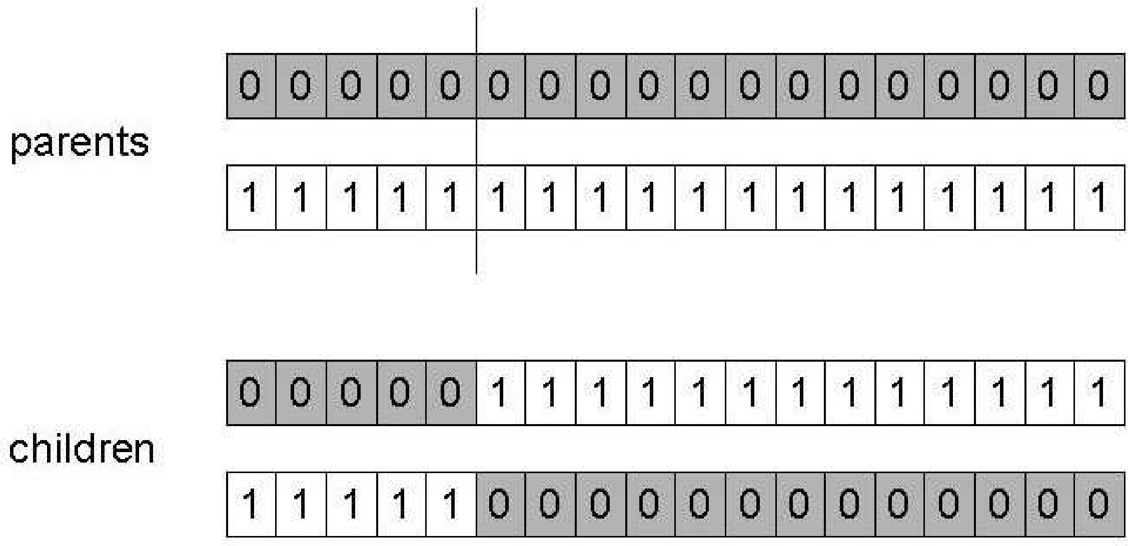


SGA Evolution Cycle

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

SGA Operators: 1-point Crossover

- Choose a random point in a pair of parents.
- Split parents at this crossover point.
- Create children by exchanging tails of the string.
- p_c typically in range (0.6, 0.9).



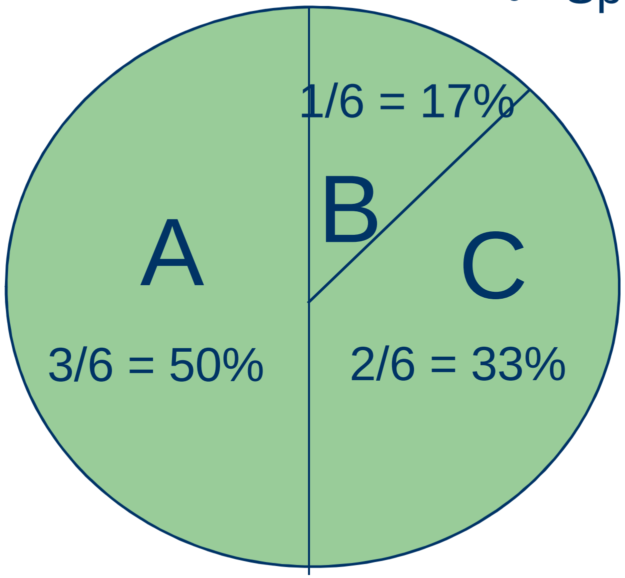
SGA Operators: Mutation

- Alter each gene independently with a probability p_m .
- p_m is called the mutation rate
  - Typically between 1/(population size) and 1/(chromosome length).

parent	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
child	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

SGA Operators: Selection

- Main idea: fitter individuals have higher chance of being selected
  - 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) = 1

fitness(C) = 2

Example after Goldberg 1989: x^2 Example

- Simple problem: max x^2 over {0,1,…,31}
- GA approach:
  - Representation: binary code, e.g. 01101 $\leftrightarrow$ 13.
  - Population size: 4.
  - 1-point crossover, bitwise mutation.
  - Roulette wheel selection.
  - Random initialisation.
- Next, execution of one generational cycle will be shown step by step.

x^2 Example: Selection

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

Table showing the selection operation: genotype and phenotype of the initial population, fitness, probability of becoming parent, number of expected parents (approximated and actual).

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

Table showing the crossover operation: The chosen parents, the choice of the crossover point, the offspring, the phenotype, and the fitness value.

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

Table showing the mutation operation: The offspring produced by the crossover, the offspring following the mutation, the phenotype, and the fitness value.

The Simple GA

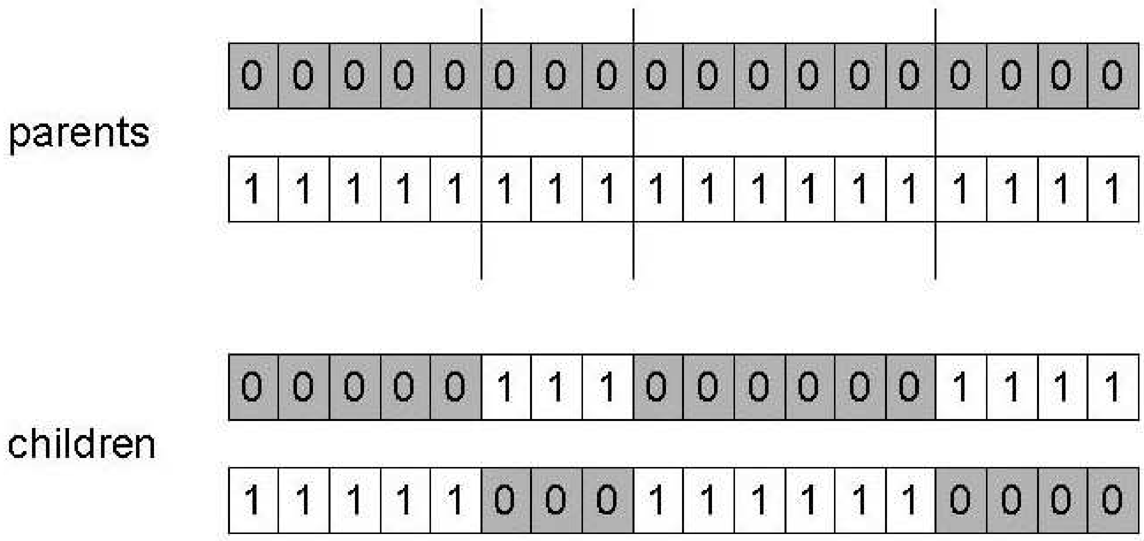
- It has been subject of many (early) studies:
  - Still often used as benchmark for novel GAs.
- It shows many limitations, such as:
  - Representation is too restrictive.
  - Mutation & crossovers only applicable for bit-string & integer representations.
  - Selection mechanism sensitive for converging populations with close fitness values.
  - Generational population model (step 5 in SGA evolution cycle) can be improved with explicit survivor selection.

Other Crossover Operators: Reasons

- 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.

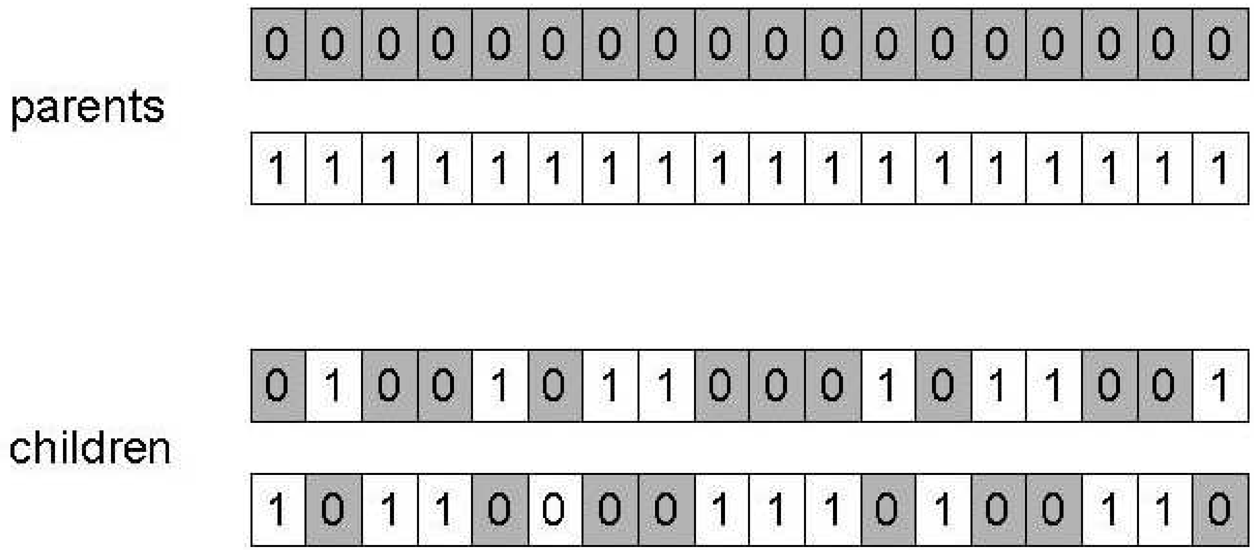
Other X Operators: n-point Crossover

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



Other X Operators: Uniform Crossover

- Assign 'heads' to one parent, 'tails' to the other: .
- ‘Flip a coin’ for each gene of each child. If the number is larger than a particular probability, take the *i-th* gene to the *i-th* child, else, choose the gene of the other parent.
- Inheritance is independent of position.



Crossover OR Mutation?

- Mutation: Variation operator that use only one individual to create another one by applying some kind of randomised change to the genotype.
- Recombination: A new individual solution is created from information contained within two or more parent solutions.
  - Crossover: Two parent recombination.
  - Crossover rate (p_c): Chance that a pair of parents creates a child.
  - Usual procedure: selection of 2 parents; comparison of a random number from [0,1) with p_c, two offsprings are created by recombination of parents or asexually (copy of the parents).
- Debate: which one is better / necessary / main-background.
- Answer (at least, rather wide agreement):
  - It depends on the problem.
  - In general, it is good to have both.
  - Mutation-only-EA is possible, xover-only-EA would not work.

Crossover OR Mutation?

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 cooperation 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?

- 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.

Other Representations

- Gray coding of integers (still binary chromosomes):
  - Gray coding is a mapping that means that small changes in the genotype cause small changes in the phenotype (unlike binary coding), i.e., generates “smoother” genotype-phenotype mapping.
- Encoding numerical variables directly as:
  - Integers: Different genes can take integers values.
  - Floating point variables: Values to be represented are generated by continuous distributions.
- Permutation Representations:
  - Suitable to decide on the order of occurrence of a sequence of events.

Integer Representations

- The integer values might be unrestricted (any integer value is allowed) or restricted to a finite set (a number of allowed values is defined).
- Some problems naturally involve integer variables, e.g. image processing parameters.
- Other problems take *categorical* values from a fixed set e.g. {blue, green, yellow, pink}.
- Natural relations (those generated by a considered problem) between the possible values that an attribute can take should be considered to design the encoding and variation operators.

Integer Representations - Operators

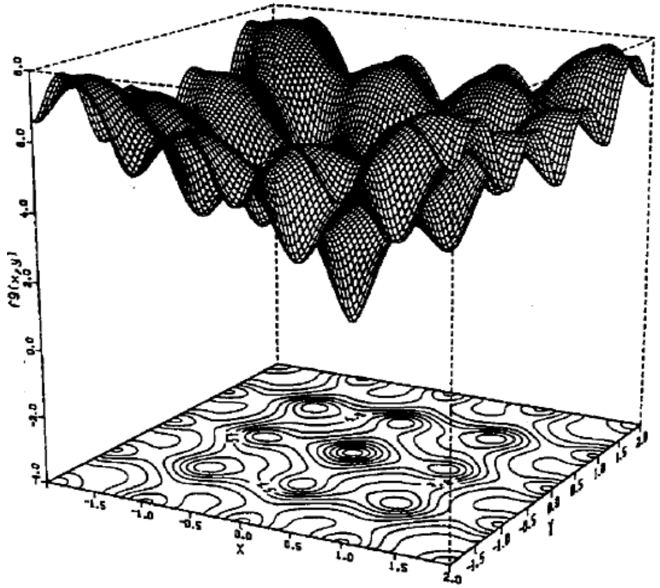
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make
  - Random choice from the set of allowed values in each gene position.
    - Suitable for cardinal attributes in which each gene value is equally likely to be chosen.
  - Creep mutation: more likely to move to similar value.
    - Suitable for ordinal attributes.
    - Obtained through a distribution symmetric about the current gene value, e.g., normal distribution.
  - For ordinal problems, it is hard to know correct range for creep mutation, so often one might use two mutation operators in tandem (at the same time).

Real Valued Problems

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

$$f(\overline{x}) = -c_1 \cdot exp\left(-c_2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} cos(c_3 \cdot x_i)\right) + c_1 + 1$$

$$c_1 = 20,\ c_2 = 0.2,\ c_3 = 2\pi$$



Mapping Real Values on Bit Strings

- $z \in [x,y] \subseteq \Re$ represented by $\{a_1,\ldots,a_L\} \in \{0,1\}^L$.
- $[x,y] \rightarrow \{0,1\}^L$ must be invertible (one phenotype per genotype).
- Γ: $\{0,1\}^L \rightarrow [x,y]$ 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)

Floating Point Mutations: 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), i.e, usually the amount of change introduced is small.

Floating Point Mutations: Nonuniform Mutation with a Fixed Distribution

- Non-uniform mutations:
  - Many methods proposed, such as time-varying range of change.
  - Most schemes are probabilistic but usually only make a small change to value.
  - The most common method is to separately generate a random amount to a gene, taken from a Gaussian distribution - N(0, σ), and add it to the such a gene.
    - Standard deviation σ controls amount of change (2/3 of deviations will lie in range (- σ to + σ).

Recombination Operators for Real Valued GAs

- Discrete:
  - 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, called *arithmetic* recombination).
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - Constant: uniform arithmetical crossover.
    - Variable (e.g. depend on the age of the population).
    - Picked at random every time.

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: $\left\langle x_1, \ldots, x_k, \alpha \cdot y_k + (1-\alpha)\cdot x_k, \ldots, x_n \right\rangle$
- Child$_2$ is the same exchanging x and y. For instance, with $\alpha = 0.5$.

Parents:

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

Children:

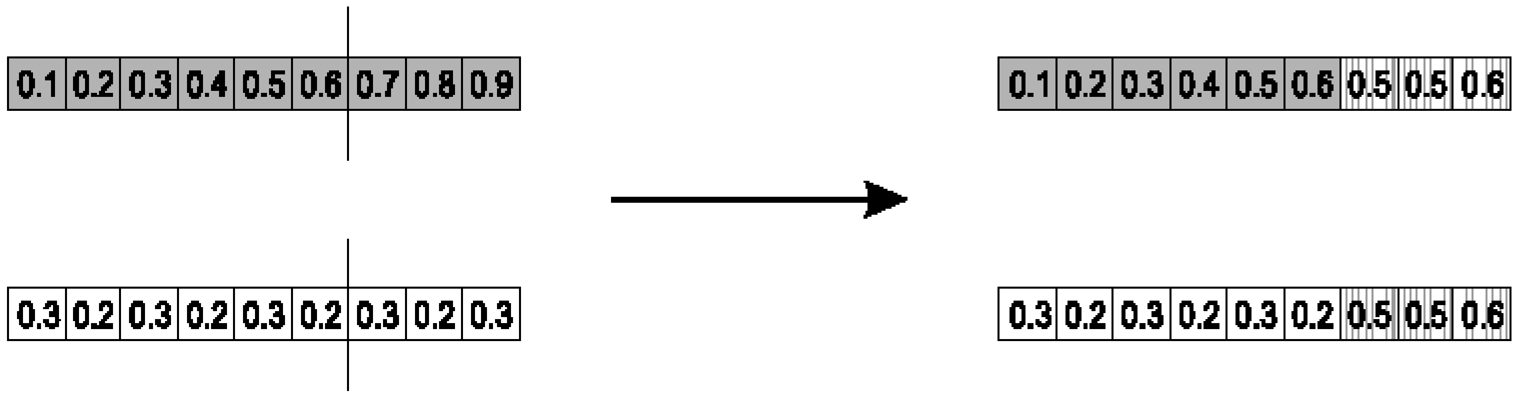
0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.5	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.5	0.3

Simple Arithmetic Crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$.
- Pick random gene *(k)* after this point mix values.
- child$_1$ is:

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

- reverse for other child. e.g. with $\alpha = 0.5$.



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 $\alpha = 0.5$.

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.6
0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.6

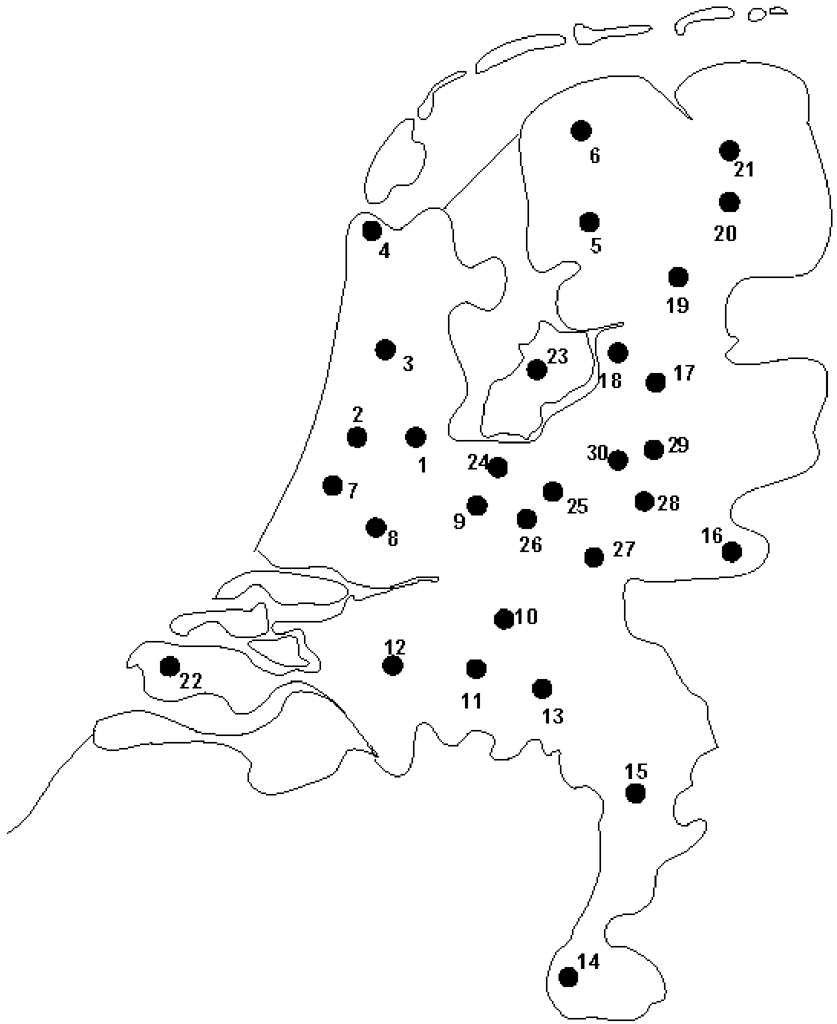
Permutation Representations

- Ordering/sequencing problems form a special type in which the task is (or can be solved by) arranging some objects in a certain order.
  - Example 1: Sort algorithm in which the central issue is to determine what elements occur before others (*order*).
  - Example 2: Travelling Salesman Problem (TSP) in which the main issue is to establish which elements occur next to each other (*adjacency*). The initial point is not important.
- The former representations allow multiple occurrence of numbers generating invalid solutions.
- 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 $30! \approx 10^{32}$ possible tours



Mutation Operators for Permutations

- Normal mutation operators lead to unviable solutions, for instance:
  - As in bit-wise mutation, let gene *i* have value *j.*
  - Changing to some other value *k* would mean that *k* could occur twice and, thus, *j* no longer occurred.
  - To satisfy the main constraint, the chromosome 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.

Swap Mutation for Permutations

- Pick two alleles at random and swap their positions.
  - Preserves most of the adjacency information, in the example only 4 links are broken.
  - Disrupts the order, i.e., chances significantly the position of each allele.

Insert Mutation for Permutations

- Pick two allele values at random.
- Move the second to follow the first, and shift right the remaining alleles.
  - Preserves most of the order and the adjacency information.

Scramble Mutation for Permutations

- Choose a subset of genes at random and randomly rearrange the alleles in those positions.
  - Loses most of the adjacency information within the subset.
  - Disrupts the order, i.e., chances significantly the position of each allele, within the subset.

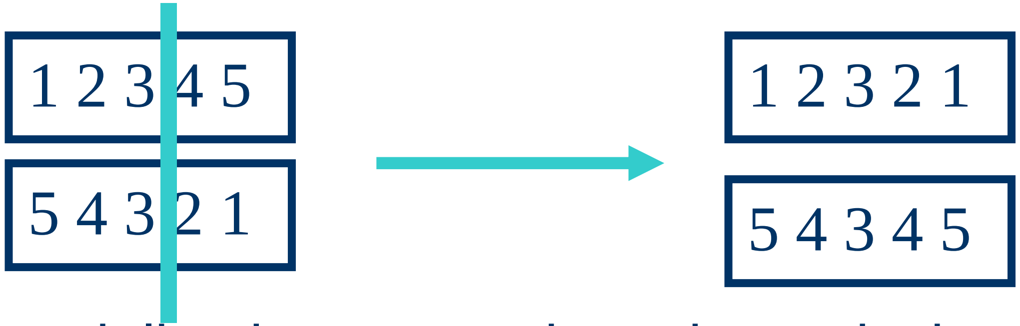
(note subset does not have to be contiguous)

Inversion Mutation for Permutations

- Pick two alleles at random and then invert the substring between them.
  - Preserves most adjacency information, in the example, only two links are broken.
  - Disruptive of order information.

Crossover Operators for Permutations

- “Normal” crossover operators will often lead to inadmissible solutions, repeating gene values.



- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents.
- Aims to transmit as much as possible information contained in the pairs, in particular, the common genes.
- Recombination operators: For adjacency problems: Partially Mapped Crossover and Edge Crossover; For order problems: Order Crossover and Cycle Crossover.

Order 1 Crossover

- 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

Order 1 Crossover Example

- Copy randomly selected set from first parent

1	2	3	4	5	6	7	8	9

→

			4	5	6	7		

9	3	7	8	2	6	5	1	4

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

1	2	3	4	5	6	7	8	9

→

3	8	2	4	5	6	7	1	9

9	3	7	8	2	6	5	1	4

Partially Mapped Crossover (PMX)

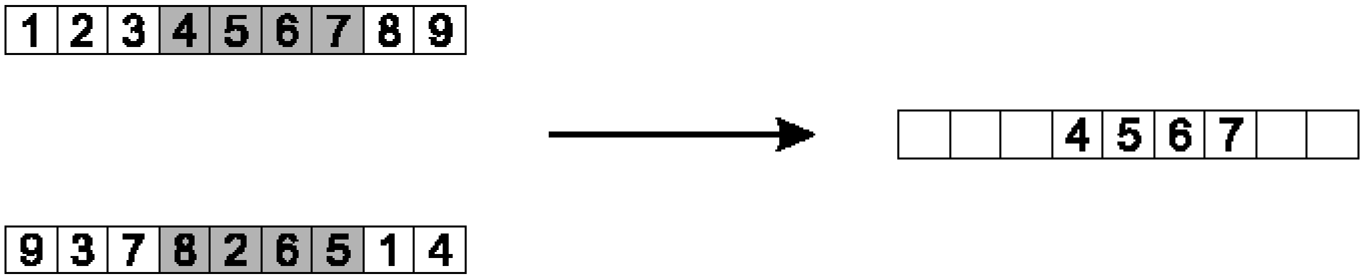
Informal procedure for parents P1 and P2:

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.

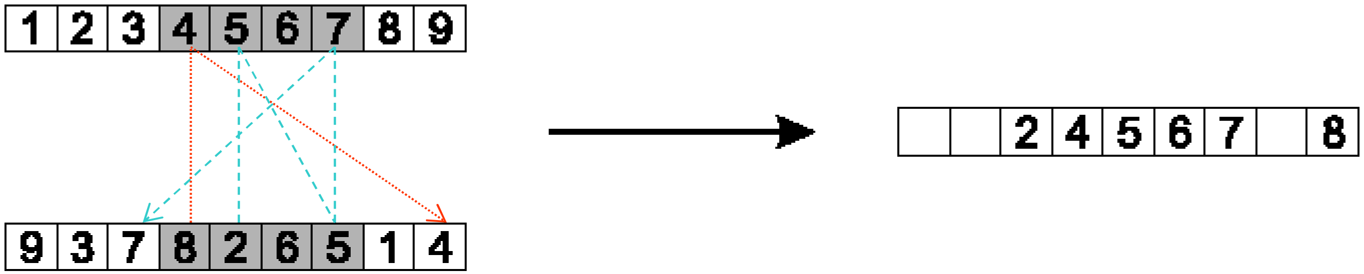
Second child is created analogously

PMX Example

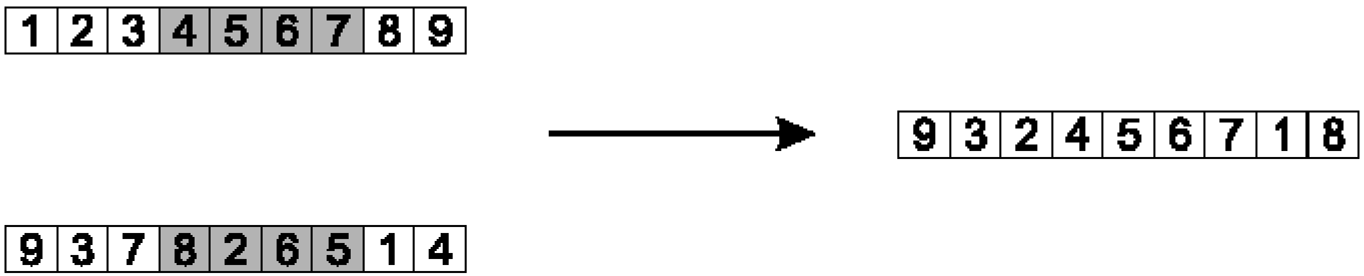
- Step 1



- Step 2



- Step 3



Cycle Crossover

Basic idea:

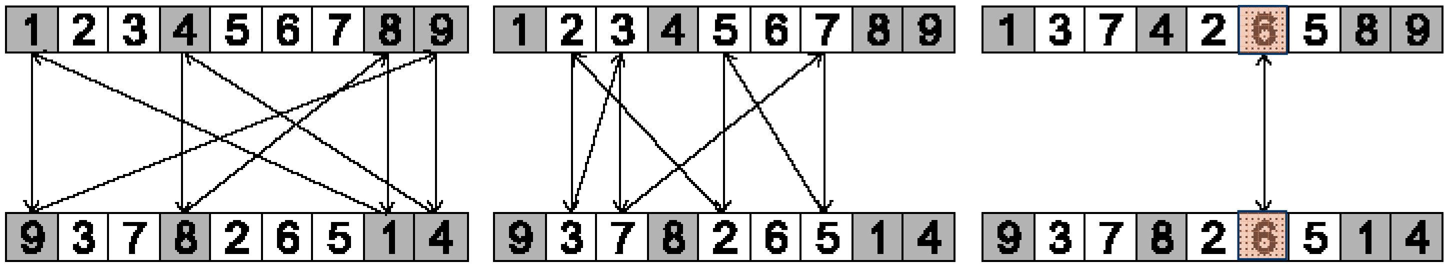
Each allele comes from one parent *together with its position*.

Informal procedure:

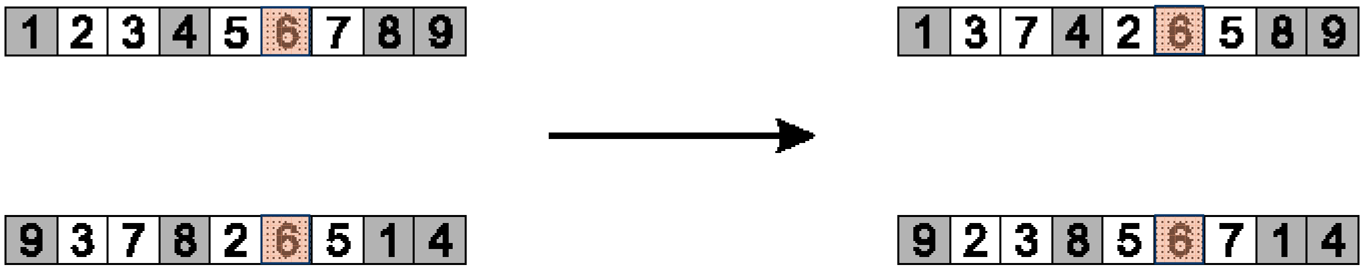
1. Make a cycle of alleles from P1 in the following way.
   - (a) Start with the first allele of P1.
   - (b) Look at the allele at the *same position* in P2.
   - (c) Go to the position with the *same allele* in P1.
   - (d) Add this allele to the cycle.
   - (e) Repeat step (b) through (d) until you 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

Cycle Crossover Example

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Edge Recombination

- 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+		

Edge Recombination 2

Informal procedure once edge table is constructed

1. Pick an initial element 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:
   - Examine the other end of the offspring is for extension
   - Otherwise a new element is chosen at random

Edge Recombination example

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*]

Multiparent Recombination

- Recall that we are not constricted by the practicalities of nature.
- Noting that mutation uses 1 parent, and "traditional" crossover 2, the extension to *a*>2 is natural to examine.
- Been around since 1960s, still rare but studies indicate useful.
- Three main types:
  - Based on allele frequencies, e.g., p-sexual voting generalising uniform crossover.
  - Based on segmentation and recombination of the parents, e.g., diagonal crossover generalising n-point crossover.
  - Based on numerical operations on real-valued alleles, e.g., center of mass crossover, generalising arithmetic recombination operators.

Population Models

- SGA uses a Generational Genetic Algorithm (GGA):
  - 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 Genetic Algorithms (SSGAs):
  - One offspring is generated per generation.
  - One member of the population is replaced.
- Generation Gap
  - Definition: The percentage of the population that is replaced.
  - 1.0 for GGA, 1/pop_size for SSGA.

Population Models - Fitness Based Competition

- Selection often can occur in two cases:
  - Selection from current generation to take part in mating (*parent selection*).
  - Selection from parents + offspring to compose the next generation (*survivor selection*).
- Selection operators work on the whole individual:
  - Such operators are representation-independent.
- Distinction between selection:
  - Operators: define selection probabilities.
  - Algorithms: define how probabilities are implemented.

Population Models - Implementation Example: SGA

- Expected number of copies of an individual *i*:

$$E(n_i) = \mu\, f_i \Big/ \sum_{j=1}^{\mu} f_j$$

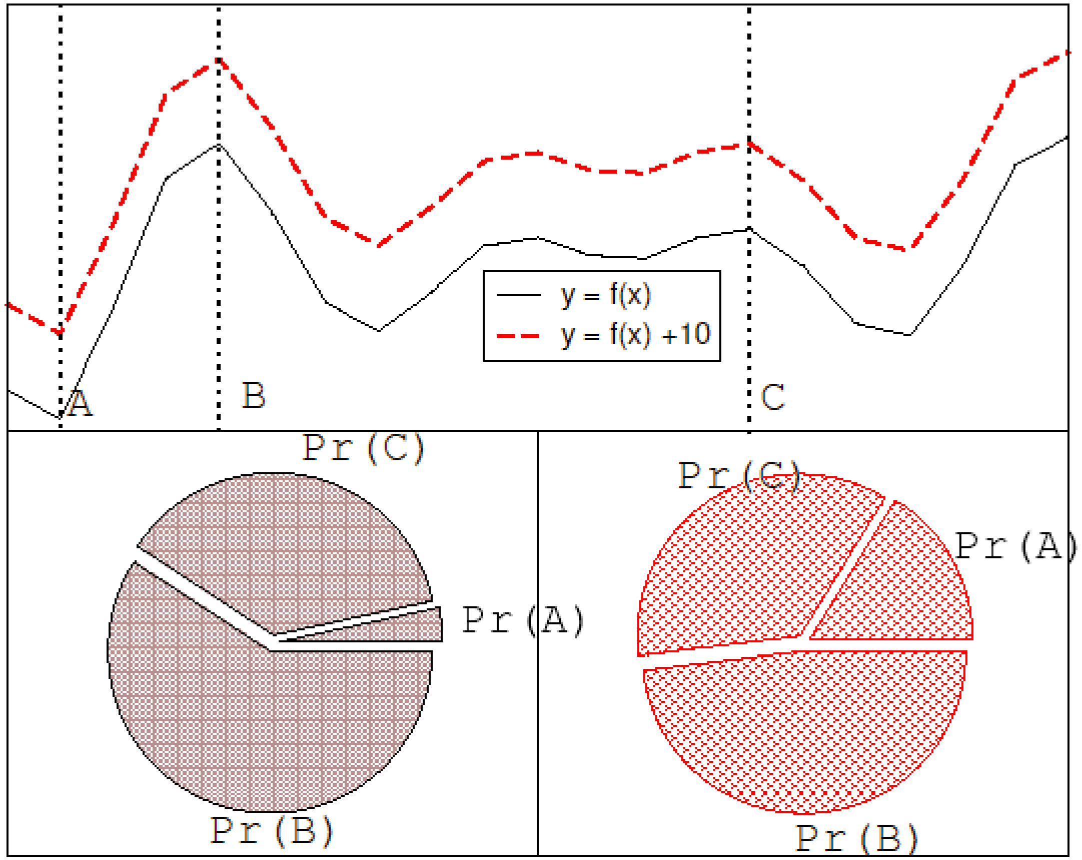
(μ = pop.size, f_i fitness of i, $\sum_{j=1}^{\mu} f_j$ = total fitness in pop.)

- Roulette wheel algorithm:
  - Given a probability distribution, spin a 1-armed wheel *n* times to make *n* selections.
  - No guarantees on actual value of n_i .
- Baker's SUS algorithm:
  - *n* evenly spaced arms on wheel and spin once.
  - Guarantees $floor(E(n_i)) \leq n_i \leq ceil(E(n_i))$.

Fitness-Proportional Selection (FPS)

- Limitations of the strategy:
  - One highly fit member can rapidly take over the process if the rest of population is much less fit: p*remature convergence*.
  - At end of runs when fitnesses are very similar, lose *selection pressure*, i.e., there are little differences between fitnesses of individuals, hence, the selection probabilities are about the same.
  - Highly susceptible to function transposition, e.g. addition of fixed values to fitnesses disrupts the functions.
- Scaling can fix last two problems:
  - Windowing: $f'(i) = f(i) - \beta^t$
    - where β is worst fitness in this (last n) generations.
  - Sigma Scaling: $f'(i) = max(f(i) – (\langle f \rangle - c \bullet \sigma_f), 0.0)$
    - where c is a constant, usually 2.0.

Function Transposition for FPS



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 where fittest has rank μ (population size) 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

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

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

Exponential Ranking

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

- Linear Ranking is limited to selection pressure.
- Exponential Ranking can allocate more than 2 copies to fittest individual.
- Normalisation factor *c*: calculated according to the population size, i.e., the sum of the probabilities must be equal to one.

Tournament Selection

- All methods above rely on global population statistics, then
  - Might yield bottlenecks especially on parallel machines.
  - Relies on the presence of “global” fitness function which might not exist: e.g. evolving game players.
- Informal Procedure:
  - Pick *k* members at random then select the best of these.
  - Repeat to select more individuals.

Tournament Selection

- Probability of selecting *i* will depend on:
  - Rank of *I*: does not need to sort the whole population.
  - 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*.
- For $k = 2$, time for fittest individual to take over population is the same as linear ranking with $s = 2 \cdot p$.

Survivor Selection

- Most of methods used for parent selection are also useful.
- This selection can be divided in two approaches:
  - Age-Based Selection:
    - Each individual exists in the population for the same number of GA interactions.
    - Examples:
      - SGA in which each individual exists for one generation.
      - SSGA can implement as “delete-random” (not recommended) or as first-in-first-out (a.k.a. delete-oldest) .
  - Fitness-Based Selection
    - Select individuals from the set composed by parents and offspring.
    - Possible methods: Fitness proportional, rank-based, tournament, replace worst, and elitism.

Two Special Cases

- Elitism:
  - At least one copy of the current fittest member is kept in a population.
  - Often used in conjunction with age-based and stochastic fitness-based replacement schemes.
  - Widely used in both population models (GGA, SSGA).
- GENITOR: a.k.a. “delete-worst”
  - The worst member of the population is replaced.
  - Improves quickly the mean population fitness and may converge prematurely.
  - 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.

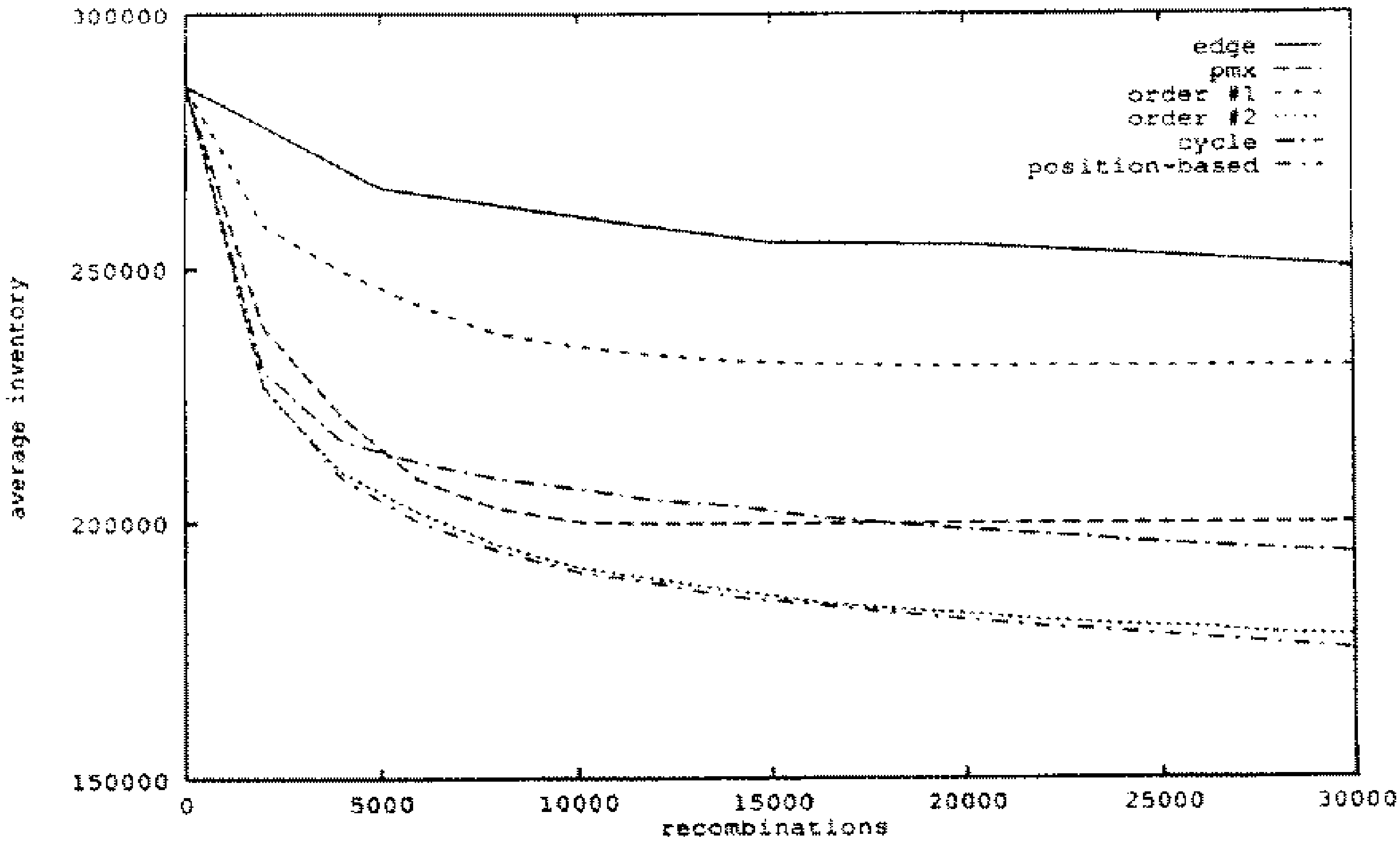
Example Application of Order Based GAs: Job Shop Scheduling Problem - JSSP

- Precedence constrained job shop scheduling problem:
  - J is a set of jobs.
  - O is a set of operations.
  - M is a set of machines.
  - *Able* $\subseteq$ O $\times$ M defines which machines can perform particular operations .
  - *Pre* $\subseteq$ O $\times$ O defines which operation should precede another one.
  - *Dur* : $\subseteq$ O $\times$ M $\rightarrow$ IR defines the duration of o $\in$ O on m $\in$ M.
- Scheduling an operation is understood as assignment of a starting time to it and a schedule is a collection of these assignments containing each operations at most once.
- The goal is now to find a schedule that is:
  - Complete: All jobs are scheduled.
  - Correct: All constraints defined by *Able* and *Pre* are satisfied.
  - Optimal: The total duration of the schedule is minimal.

Precedence Constrained GA

- Representation: individuals are permutations of operations.
- Permutations are decoded to schedules by a decoding procedure:
  - Take the first (next) operation from the individual.
  - Look up its machine (here we assume there is only one).
  - Assign the earliest possible starting time on this machine, subject to:
    - Machine occupation.
    - Precedence relations holding for this operation in the schedule so far.
- Fitness of a permutation is the duration of the corresponding schedule (to be minimized).
- Variation operators: Any suitable mutation and crossover.
- Parent selection: Roulette wheel applied on inverse fitness.
- Survivor selection: Generational GA model.
- Random initialisation and maximum number of fitness evaluations.

JSSP Example: Operator Comparison



Some GAs Interesting Sites

- http://www-2.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/top.html
- http://cs.gmu.edu/research/gag/
- http://www-illigal.ge.uiuc.edu/index.php3
- http://www.arch.columbia.edu/DDL/cad/A4513/S2001/r7/
- http://www.aic.nrl.navy.mil/galist/
- http://www.aaai.org/AITopics/html/genalg.html
- http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/0.html
- http://psychology.about.com/od/companies/
- http://www.nutechsolutions.com/
- http://www.autonomoussolutions.com/
- http://www.palisade.com/
- http://www.optisyn.com/