CHAPTER 2-2
GENETIC ALGORITHM

Siamak Sarmady, UUT – (Based on the slides from A.E. Eiben)

Simple GA

Overview

- Holland's original GA is now known as the simple genetic algorithm (SGA)
  - **Authors:** J. Holland, K. DeJong, D. Goldberg, 1970's, USA.
  - **Problems:** typically applied to discrete optimization
  - **Features:**
    - Not too fast
    - Good heuristic search algorithm for combinatorial problems
  - **Emphasis:**
    - Combining information from good parents (crossover)
  - **Other GAs use different:**
    - Representations
    - Mutations
    - Crossovers
    - Selection mechanisms (parent, survivor)

SGA technical summary tableau

- **Simple GA Specifications:**

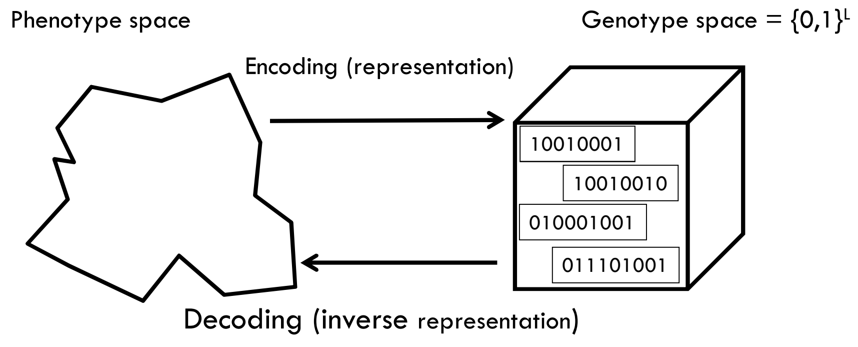
Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

SGA reproduction cycle

- **Simple GA Flow:**
  1. **Representation:** determine the genotype representation
  2. **Create pool:** create parents for the mating pool (n = population size)
  3. **Parent selection:** start selection of parents using roulette wheel
  4. **Crossover:** for each consecutive pair apply crossover with probability p_c , otherwise copy parents
  5. **Mutation:** for each offspring, for each bit, bit-flip with probability p_m independently
  6. **Survivor selection:** replace the whole population with the resulting offspring

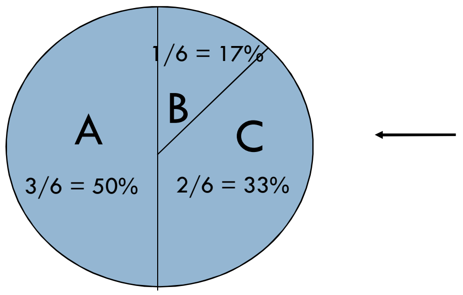
Representation

- The first step in using GA is to transfer the problem from phenotype space to genotype space.
- It means variables that represent the problem are put together into a chromosome (string) form.


Phenotype space
Genotype space = $\{0,1\}^L$
Encoding (representation)
10010001
10010010
010001001
011101001
Decoding (inverse representation)

Operators: Parent Selection

- **Main idea:** better individuals get higher chance
  - Selection 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



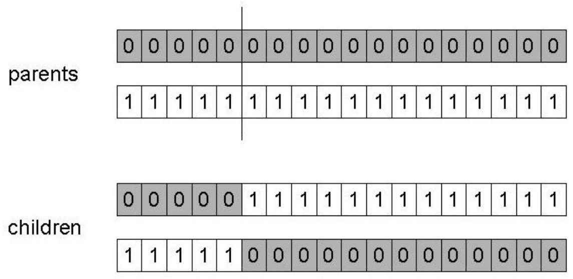
fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

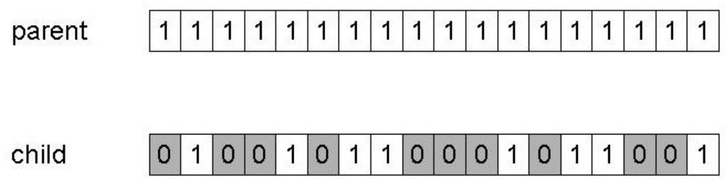
Operators: 1-point crossover

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



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



- **Note:** mutation should preferably build a valid output

Simple GA example: Max of X^2 Goldberg 89 (1)

- Simple problem: max x^2 over {0,1,…,31}
- GA approach:
  - **Representation:** binary code, e.g. 01101 ↔ 13
  - **Initialization:** Random
  - **Population size:** 4
  - **Crossover:** 1-point
  - **Mutation:** bitwise
  - **Parent Selection:** Roulette wheel selection
- We show one generation cycle done by hand

Simple GA example: parent selection

- Generation 1:

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

Simple GA example: crossover

- Generation 2:

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

Simple GA example: mutation

- Generation 2 (mutation):

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	$\boxed{1}$ 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 $\boxed{1}$ 0 0	18	324
Sum				2354
Average				588.5
Max				729

The simple GA

- Subject of many (early) studies
- Still used as benchmark for novel GAs
- **Shortcomings:**
  - **Representation:** too restrictive
  - **Mutation & crossover:** only applicable for bit-string & integer representations
  - **Parent selection:** sensitive when finesses converge and are too close
  - **Survivor selection:** a better selection (than replacement) method can improve the cycle

Building Alternative GAs

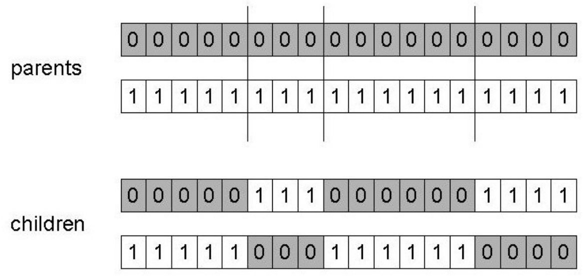
- Alternative Crossover Operators
- Integer representations
- Real valued problems
- Permutation Representations

Alternative Crossover Operators

- **One Point Crossover:**
  - **Performance:** depends on the order of variables in the representation
  - **Positional Bias:**
    - More likely to keep together genes that are near each other
    - Cannot keep genes from opposite ends of a chromosome together
  - Positional bias can be exploited if we know about the structure of our problem, but this is not usually the case

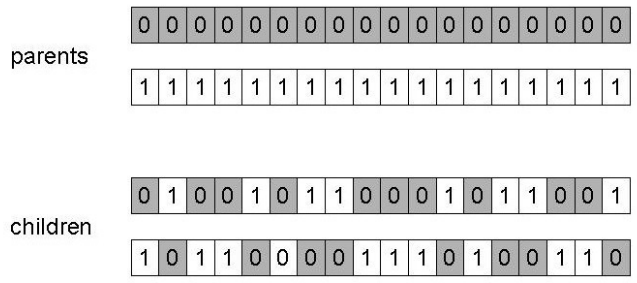
Alternative Crossover Operators: n-point crossover

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



Alternative Crossover Operators: Uniform crossover

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



Crossover vs. mutation (1)

- **Debate:** a decade long debate on which one is better / necessary
- **Widely agreed answer:**
  - Depends on the problem
  - They have different roles
  - Good to have both
  - Mutation-only-EA is possible
  - Crossover-only-EA does not work

Crossover vs. mutation (2)

- **Exploration and exploitation:** there is co-operation AND competition between these
  - **Exploration:** discovering promising areas in the search space
  - **Exploitation:** optimizing within a promising area
- **Crossover:** explorative, makes a *big* jump to an area between the two parents
  - Only crossover can combine information from two parents
  - Does not change the frequencies of gene values (allele) in the population (Experiment: 50% 0's on first bit in the population, what % after performing n crossovers)
- **Mutation:** exploitative, creates random *small* diversions, staying near (the area of) the parent
  - Only mutation can introduce new information
  - To hit the optimum you often need a 'lucky' mutation

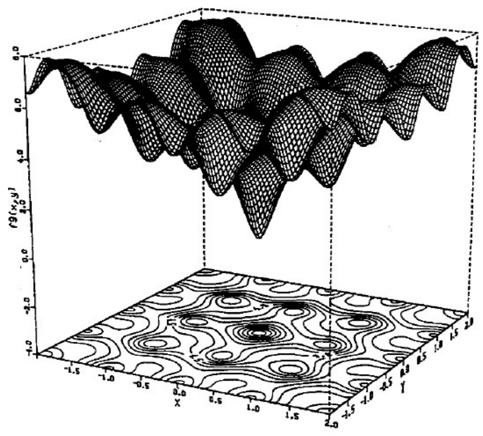
Other representations: Integer representations

- **Integer problems:** some problems involve integer variables
  - Image processing parameters
  - *Categorical* values {blue, green, yellow, pink}
- **Crossover:** both n-point and uniform crossover operators work
- **Mutation:** bit-flipping mutation can be extended
  - **Random choice:** in categorical variables, we select a random item from the fixed set of choices

Other representations: Real valued problems

- Many problems are real valued
  - continuous parameter optimizations i.e. f : $\Re^n \rightarrow \Re$
    - **Example:** 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 binary representation (Method 1)

- With real values, one options is to represent real values (in the range of [x,y]) using 0/1 bits and then use binary methods:

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

- Mapping $z \rightarrow \{0,1\}^L$ must be invertible (i.e. only one phenotype per genotype)
- Γ: $\{0,1\}^L \rightarrow [x,y]$ defines the representation mapping

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

- Only 2^L values out of infinite (or range [x,y]) are represented
- L determines possible maximum precision of solution
- High precision → long chromosomes (slow evolution)

Real valued representation (Method 2)

- It is possible to have real valued genes

$$\bar{x} = \langle x_1,\ldots,x_l \rangle \qquad x_i \in [LB_i, UB_i]$$

- However we need to determine a lower and higher band for all or individual genes
- Combination and mutation operators of binary representations can not be applied. New operators should be used…

Real value mutations 1 (Creep)

- **General scheme** of mutation for real valued chromosomes

$$\bar{x} = \langle x_1, \ldots, x_l \rangle \rightarrow \bar{x}' = \langle x_1', \ldots, x_l' \rangle \qquad x_i, x_i' \in [LB_i, UB_i]$$

- **Creep Mutation:** for real valued genes, a random gene is selected and a random value is added to it or it is changed to another random value in this range (gene should still be between a lower and an upper bound).
- **Uniform mutation:** a random number of uniform distribution replaces the old gene

$$x_i' \text{ drawn randomly (uniform) from } [LB_i, UB_i]$$

Note: The above two methods are analogous to bit-flipping (in binary representation) and random resetting/choice (in integer representation)

Real value mutations 2 (Non-Uniform)

- **Non-uniform mutations:** many methods, but most of them are probabilistic and usually make only small change to value (similar to creep)
  - **Time varying:** changes that vary based on the generation number, fitness etc.
  - **Gaussian Random Deviation (most common):** adds random deviate to each variable separately, taken from N(0, σ) Gaussian distribution and then curtail to range
    - Standard deviation σ controls amount of change (determined for each variable)
    - 2/3 of deviations will lie in range (- σ to + σ)

Real value crossover operators: Discrete, Intermediate

- **Discrete:**
  - N-point or uniform
  - Each allele value in offspring *z* comes from one of its parents (*x*,*y*) with equal probability:

$$z_i = x_i \text{ OR } y_i$$

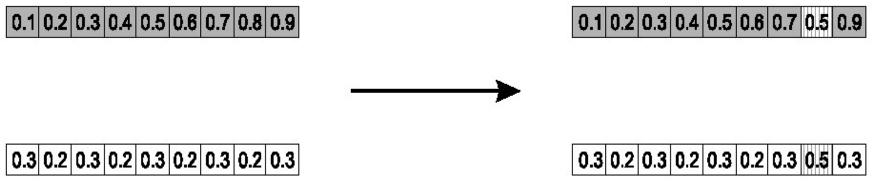
- **Intermediate:**
  - Exploits idea of creating children “between” parents (hence a.k.a. *arithmetic* recombination)
  - Practically combine two chromosomes with a weight of α
  - $z_i = \alpha\, x_i + (1 - \alpha)\, y_i$ where $\alpha : 0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - **Constant:** uniform arithmetical crossover
    - **Variable:** for example can depend on the age of the population
    - **Random:** picked at random every time

Real value crossover operators: single arithmetic

- Parents: $\langle x_1,\dots,x_n \rangle$ and $\langle y_1,\dots,y_n \rangle$
- Pick a single gene (*k*) at random, mix specific gene of two parents using α and α-1 ratios
- Child$_1$ is:

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

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

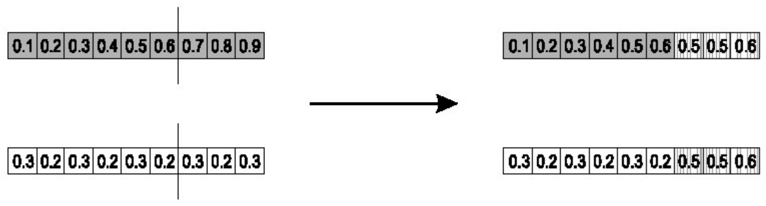


Real value crossover operators: Simple arithmetic

- Parents: ⟨x1,…,xn ⟩ and ⟨y1,…,yn⟩
- Pick random gene *(k)* after this point mix values
- Child1 is:

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

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



Real value crossover operators: Whole arithmetic

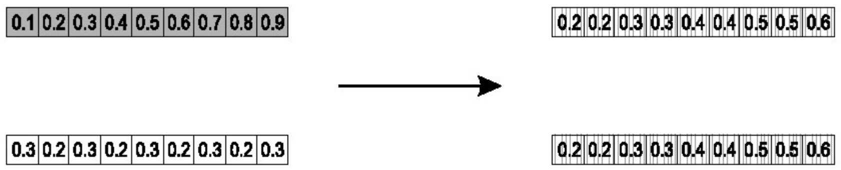
- Most common
- Parents: ⟨x1,…,xn ⟩ and ⟨y1,…,yn⟩
- Child1 is:

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

- Reverse for other child

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

- e.g. with α = 0.5

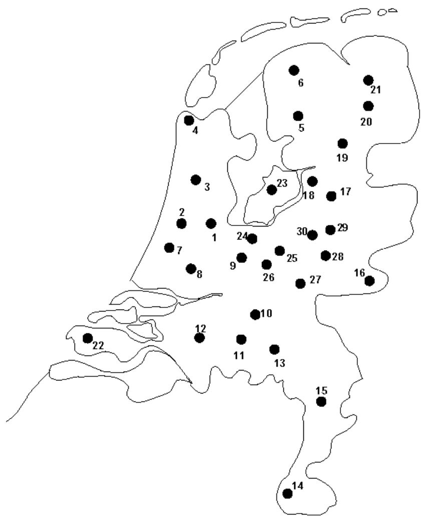


Other representations: Permutation Representations

- **Ordering/sequencing problems:** these tasks are solved by arranging some objects in a certain order
  - **Sort and ranking problems:** order of items so that they are listed by relevance or other property
  - **Travelling Salesman Problem (TSP):** order and adjacency of elements (elements occurring next to each other) is important
- **Permutation problems:** above examples are expressed as permutations, some of the permutations are optimum
  - There are *n* variables
  - The representation is as a list of *n* integers
  - Each integer 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*
  - A complete tour is a permutation
    - For n =4 answers could be [1,2,3,4] and [3,4,2,1]
- **Search space is BIG:**
  - for 30 cities there are 30! ≈ 10^{32} possible tours

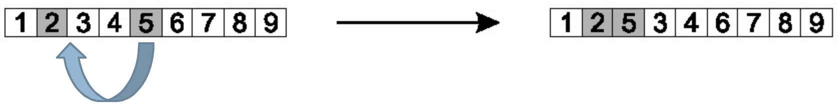


Permutation representation: Mutation operators

- Normal mutation operators lead to inadmissible solutions
  - **Bit-wise mutation:** let gene *i* have value *j*, changing to some other value *k* would mean that *k* occurred twice and *j* no longer occurred
- **Solution:** we must change at least two values
  - Some operator is applied to the whole string, rather than individually for each position
  - A probability of applying the operator is still used

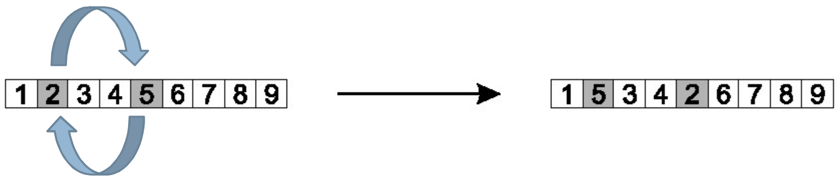
Permutation representation: Insert Mutation

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



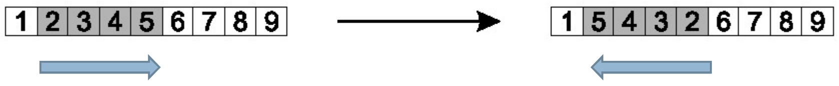
Permutation representation: Swap mutation

- Pick two genes at random and swap their positions
- Preserves most of adjacency information (4 links broken), disrupts order more



Permutation representation: Inversion mutation

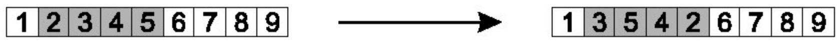
- Pick two genes at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information



Permutation representation: Scramble mutation

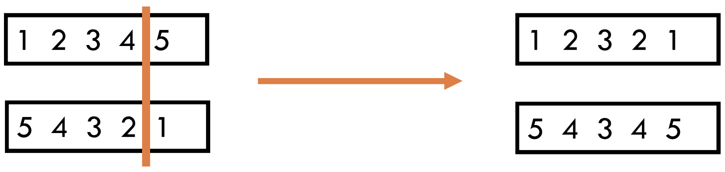
- Pick a random subset of genes
- Randomly rearrange the alleles in those positions

(note subset does not have to be contiguous)



Permutation representation: Crossover operators

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



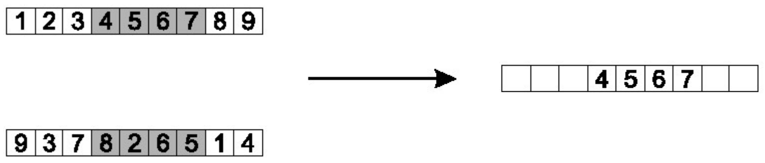
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Permutation representation: "Order1" crossover

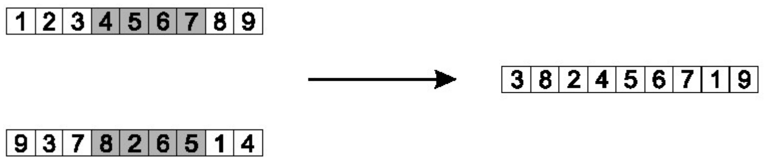
- Idea is to preserve relative order that elements occur
- Informal procedure:
  1. Choose an arbitrary part from the first parent to the first child
  2. Copy the numbers that are not in the first part, to the first child:
     - starting right from cut point of the copied part,
     - using the **order**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

Permutation representation: "Order 1" crossover example

- Copy randomly selected set from first parent



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



Permutation representation: Partially Mapped Crossover (PMX)

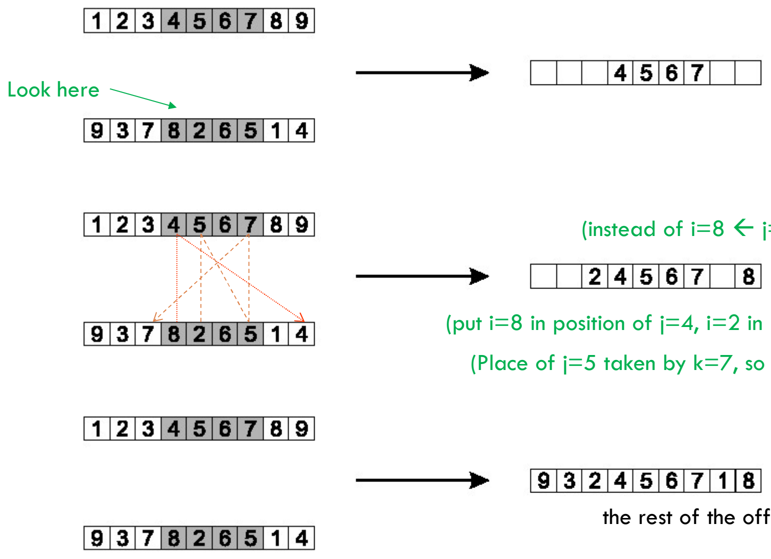
Informal procedure for parents P1 and P2:

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 (i=8,i=2)
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1 (instead of i=8 ← j=4 , instead of i=2 ← j=5)
4. Place *i* into the position occupied by *j* in P2, since we know that we will not be putting *j* there (as is already in offspring) (put i=8 in position of j=4, i=2 in position of j=5 taken)
5. If the place occupied by *j* in P2 has already been filled in the offspring by *k*, put *i* in the position occupied by *k* in P2 (Place of j=5 taken by k=7, so put i=2 in position of k=7)
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

Permutation representation: PMX example

- Step 1
- Step 2
- Step 3



Not copied (i=8,i=2)

(instead of i=8 ← j=4 , instead of i=2 ← j=5)

(put i=8 in position of j=4, i=2 in position of j=5 taken by 7)

(Place of j=5 taken by k=7, so put i=2 in position of k=7)

the rest of the offspring can be filled from P2

Permutation representation: Cycle crossover

Basic idea:

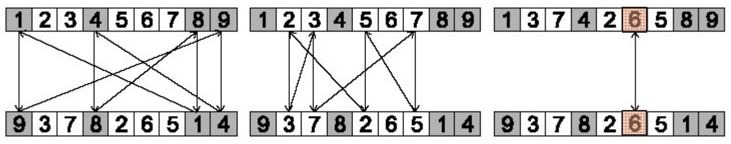
Each allele comes from one parent *together with its position*.

Procedure:

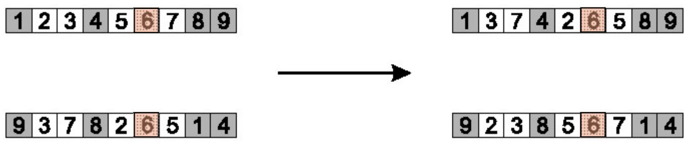
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the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

Permutation representation: Cycle crossover example

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Permutation representation: Multi-parent recombination

- **Existing operators:**
  - **Mutation:** operates on 1 parent
  - **Traditional crossover:** operates on 2 parents, makes 2 Childs

- **Multi-parent:** the extension to $a > 2$
  - Since unlike nature we are not limited to two parents..
  - Been around since 1960s
  - Still rare
  - Studies indicate it could be useful

- **Three main types:**
  - **Uniform crossover:** equal probabilities for the selection of each allele
  - **Segmentation and recombination of the parents:** for example generalising n-point crossover, diagonal crossover
  - **Numerical operations on real-valued alleles:** generalising arithmetic recombination operators

Alternative Selection Methods

Fitness Based Competition

- **Selection:** there are two major kinds of selection
  - **Parent selection:** selection from current generation to take part in mating
  - **Survivor selection:** selection from parents + offspring to go into next generation
- **Operation span:** selection operators work on all the individuals
- **Differences:** selection operators are different in two aspects
  - **Probabilities:** how selection probabilities are determined
  - **Algorithms:** how probabilities are implemented

Population Models

- **Generational models:** used by Simple GA
  - Each individual survives for exactly one generation
  - The entire set of parents is replaced by the offspring
- **Steady-state models:** population changes are very small
  - One offspring is generated per generation
  - One member of population replaced
- **Generation Gap:**
  - A specific proportion of the population replaced
  - **Proportion Bands:** between "1.0" and "1/pop_size" (i.e. between G-GA and SS-GA)

Parent Selection: SGA's fitness proportionate method

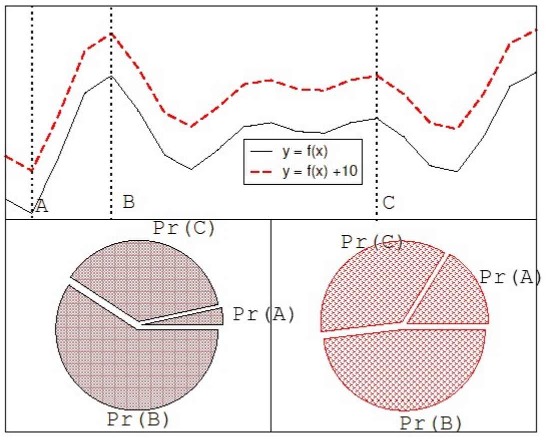
- **Probability:** expected number of copies of an individual *i*
  - $E(n_i) = N \bullet f(i)/f_{avg}$ or $E(n_i) = N \bullet f(i)/f_{all}$
    - N = population size
    - f(i) = fitness of individual I
    - F_{avg} = Average fitness in population
- **Roulette wheel Algorithm:**
  - Given a probability 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 that individuals are really selected based on the expectation.
  - $\text{floor}(E(n_i)) \leq n_i \leq \text{ceil}(E(n_i))$

Fitness-Proportionate Selection - Discussion

- **Problems:**
  - **Premature Convergence:** one highly fit member can rapidly take over if rest of population is much less fit
    - Fitnesses: 93, 21,20, 18, 13
  - **Loss of selection pressure:** at end of runs when fitnesses are similar, selection is not that useful (fails in its intended function) e.g.
    - fitnesses: 93, 95, 92, 92, 90
  - **Highly sensitive to function transposition:** if we add a constant to fitness function (i.e. use a bit different fitness function) the behaviour of the method totally changes
- **Scaling:** can fix the last two problems
  - **Windowing:** $f'(i) = f(i) - f_{worst}$
    - where f_{worst} is the worst fitness in this (or last n) generations
  - **Sigma Scaling:** $f'(i) = max(f(i) - (f_{avg} - c \bullet \sigma_f) , 0.0)$
    - where c is a constant, usually 2.0
    - Scales the fittnesses near 0 (some kind of standardization)

Function transposition for FPS

- if we add a constant to fitness function (i.e. use a bit different fitness function) the behaviour of the method totally changes



Ranking Selection

- **Ranking Selection:** attempts to solve FPS problem by basing selection probabilities on *relative* fitness rather than *absolute* fitness
  - Rank population according to fitness (Best fittest has rank *N* and worst rank 1)
  - Base selection probabilities on rank
- 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 GGA this is the number of children allotted to it
- Simple 3 member example

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

	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

- Linear Ranking is limited to selection pressure
- Exponential Ranking can allocate more than 2 copies to fittest individual
- Normalise constant factor c according to population size

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

Parent Selection: Tournament Selection

- Most methods rely on global population statistics (e.g. which are the less fit among all population)
  - Could be a bottleneck specially on parallel machines
- **Tournament Selection Procedure:**
  - Pick *k* members at random
  - Select the best of above k members
  - Repeat to select more individuals

Parent Selection: Tournament Selection 2

- Probability of selecting i depends on:
  - Rank of i (among k)
  - 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

Survivor Selection

- Survivor selection can be divided into two approaches:
  - Age-Based Selection
    - S-GA
    - SS-GA: can be implemented as "delete-random" (not recommended) or as first-in-first-out (i.e. delete-oldest)
  - Fitness-Based Selection
    - Using one of the methods above but according to fitness

Survivor Selection: Two Special Cases

- **Elitism**
  - Widely used in both population models (G-GA, SS-GA)
  - Always keep at least one copy of the fittest solution so far
- **GENITOR (delete-worst):**
  - From Whitley's original Steady-State algorithm (he also used linear ranking for parent selection)
  - **Rapid takeover:** use with large populations or "no duplicates" policy

JSSP Example

Example application of order based (permutation) GAs: 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 which operations
  - ***Pre*** $\subseteq$ **O** $\times$ **O:** defines which operation should precede which
  - ***Dur*** $\subseteq$ **O** $\times$ **M** $\to$ **IR:** defines the duration of $o \in O$ on $m \in M$

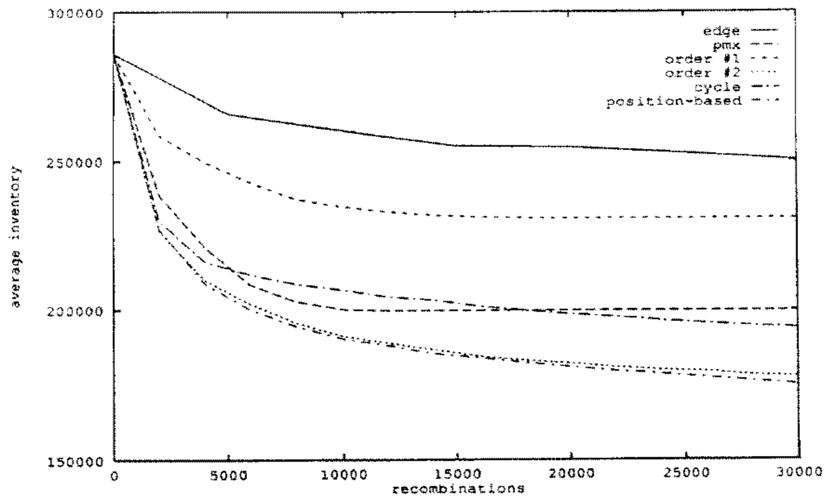
- The goal is now to find a schedule that is:
  - **Complete:** all jobs are scheduled
  - **Correct:** all conditions defined by *Able* and *Pre* are satisfied
  - **Optimal:** the total duration of the schedule is minimal

Precedence constrained job shop scheduling GA

- **Representation:** individuals are permutations of operations
- **Usage:** permutations are decoded for scheduling (by a decoding procedure)
  1. take the first (next) operation from the individual
  2. look up its machine (here we assume there is only one)
  3. assign the earliest possible starting time on this machine, subject to
     - machine occupation
     - precedence relations holding for this operation in the schedule created so far
- **Fitness of a permutation:** is the duration of the corresponding schedule (to be minimized)
- **Operators:** use any suitable mutation and crossover
- **Parent selection:** use roulette wheel on inverse fitness
- **Survivor selection:** generational GA model for survivor selection
- **Initialization:** use random initialization

JSSP example: operator comparison

- Results


edge
pmx
order #1
order #2
cycle
position-based
average inventory
recombinations