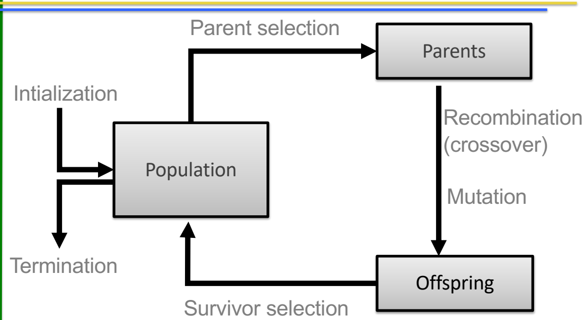
Genetic Algorithms

CS 419/519

Representation, Mutation, and Recombination Outline:

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

Scheme of an EA: General scheme of EAs



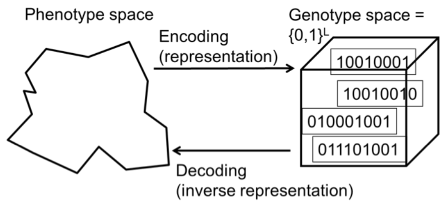
Role of representation and variation operators

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

- TSP problem
  - What are possible representations?

Binary Representation

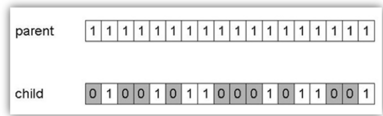
- One of the earliest representations
- Genotype consists of a string of binary digits



- Generally a mistake to use binary representation of non-binary information

Binary Representation: Mutation

- Alter each gene independently with a probability p_m



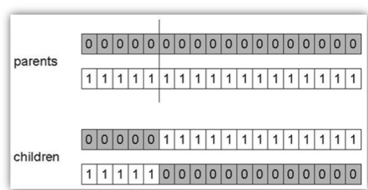
- Mutation can cause variable effect – bits have different significance so some cause bigger jumps in phenotype than others.
- Hamming cliff – using Gray code representation overcomes this.

Binary Representation: Mutation rate

- p_m is called the mutation rate
  - Typically between 1/(pop_size * genome_length) and 1/genome_length
- 1/pop_size * genome_length: about one mutation per generation over entire population
  - Less likely to disrupt good individuals
- 1/genome_length: about one mutation per member in each generation
  - More likely to result in larger number of highly fit individuals

Binary Representation: 1-point crossover

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

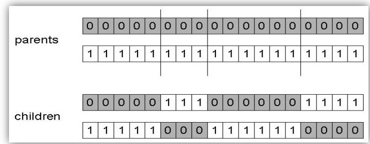


Binary Representation: Alternative Crossover Operators

- Why do we need other crossover(s)?
- 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*

Binary Representation: n-point crossover

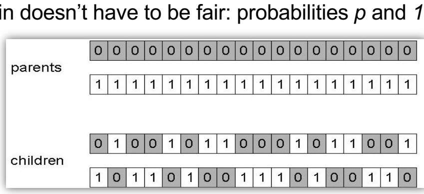
- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalization of 1-point (still some positional bias)



- What if we take n-point crossover to its logical limit?

Binary Representation: Uniform crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position – no positional bias
- Coin doesn't have to be fair: probabilities *p* and *1-p*



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

- Decade long debate: which one is better / necessary
- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role
  - mutation-only-EA is possible, xover-only-EA unlikely to work

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

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

Exploitation: Optimizing 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

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

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)

- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, ?% after performing n crossovers)
- To hit the optimum you often need a 'lucky' mutation

Integer Representation

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

Real-Valued or Floating-Point Representation

- Many problems occur as real-valued problems, *e.g.* continuous parameter optimization $f: \mathbb{R}^n \longrightarrow \mathbb{R}$
- Genotype is a vector $\langle x_1, \ldots, x_k \rangle,\ x_i \in \mathbb{R}$

- Examples:
  - Satellite boom design: angles and spar lengths are real-valued
  - Neural network training: weights are real-valued
  - Problems in *k*-dimensional space

Real-Valued or Floating-Point Representation: Mapping real values onto 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)
- $\Gamma: \{0,1\}^L \rightarrow [x,y]$ defines the representation

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

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

Real-Valued or Floating-Point Representation: Uniform Mutation

- General scheme of floating point mutations

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

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

- Uniform Mutation

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

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

Real-Valued or Floating-Point Representation: Nonuniform Mutation

- Non-uniform mutations:
  - Many methods proposed, such as time-varying range of change etc.
  - Most schemes are probabilistic but usually only make a small change to value
  - Most common method is to add random deviate to each variable separately, taken from $N(0, \sigma)$ Gaussian distribution and then curtail to range

    $x'_i = x_i + N(0,\sigma)$
  - Standard deviation σ, *mutation step size*, controls amount of change (2/3 of drawings will lie in range ($-\sigma$ to $+\sigma$))

Real-Valued or Floating-Point Representation: Self-Adaptive Mutation

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

Real-Valued or Floating-Point Representation: Self-Adaptive Mutation

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

Real-Valued or Floating-Point Representation: Crossover operators

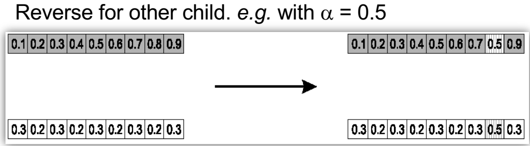
- Discrete:
  - each allele value in offspring *z* comes from one of its parents *(x,y)* with equal probability: $z_i = x_i$ or y_i
  - Could use n-point or uniform
- Arithmetic
  - exploits idea of creating children "between" parents (*a.k.a. intermediate* recombination)
  - $z_i = \alpha\, x_i + (1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

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

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

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

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



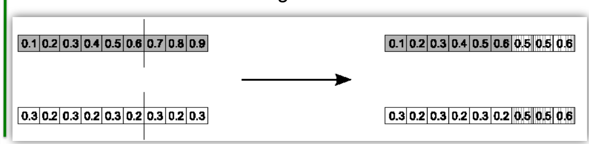
- How does this differ from mutation?

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

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

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

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

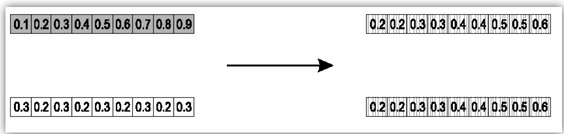


Real-Valued or Floating-Point Representation: Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Child$_1$ is:

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

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



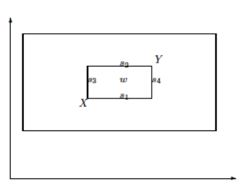
- α != 0.5 avoids the duplication seen above

Real-Valued or Floating-Point Representation: Blend Crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- $d_i = \text{abs}(y_i - x_i)$
- Random sample $z_i = [\min(x_i, y_i) - \alpha d_i, \min(x_i, y_i) + \alpha d_i]$
- Original authors had best results with α = 0.5

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

Parents: X, Y



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

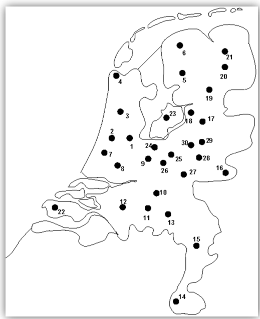
Permutation Representations

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

Permutation Representation: TSP example

- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (*e.g.* for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:

  for 30 cities there are 30! ≈ 10^{32} possible tours



Permutation Representations: Mutation

- Normal mutation operators lead to inadmissible solutions
  - e.g. bit-wise mutation: let gene *i* have value *j*
  - changing to some other value *k in [1..n]* would mean that *k* occurred twice and *j* no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

Permutation Representations: Swap mutation

- Choose two alleles at random and swap their positions

1	2	3	4	5	6	7	8	9	→	1	5	3	4	2	6	7	8	9

- **Variation**: choose two adjacent alleles at random and swap their positions

Permutation Representations: Insert Mutation

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

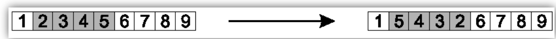
1	2	3	4	5	6	7	8	9	→	1	2	5	3	4	6	7	8	9

Permutation Representations: Scramble mutation

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

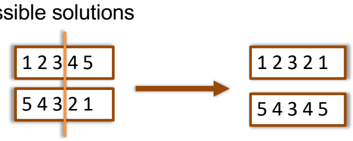
Permutation Representations: Inversion mutation

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



Permutation Representations: Crossover operators

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



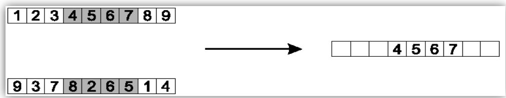
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Permutation Representations: Order 1 crossover

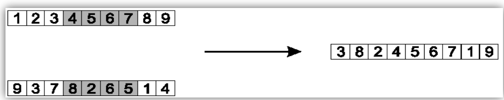
- Idea is to preserve relative order that elements occur
- Informal procedure:
  - 1. Choose an arbitrary part from the first parent
  - 2. Copy this part to the first child
  - 3. Copy the numbers that are not in the first part, to the first child:
    - starting right from cut point of the copied part,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

Permutation Representations: Order 1 crossover

- Copy randomly selected set from first parent



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



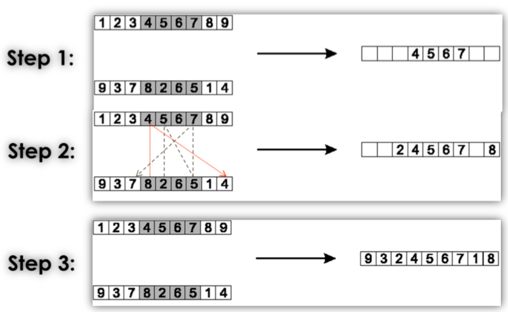
Permutation Representations: Partially Mapped Crossover (PMX)

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 by *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 by *k*, put *i* in the position occupied by *k* in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

Permutation Representations: Partially Mapped Crossover (PMX)



Permutation Representations: Cycle crossover

Basic idea:

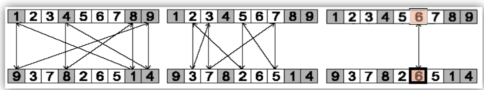
Each allele comes from one parent *together with its position*.

Informal procedure:

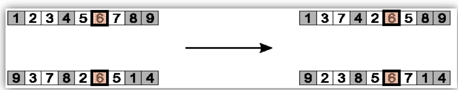
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the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

Permutation Representations: Cycle crossover

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Permutation Representations: 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+		

Permutation Representations: 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

Permutation Representations: Edge Recombination

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*]