Genetic Algorithms

- John Holland, 1970’s
- Optimization Tool
- Attributed Features
  - not too fast
  - good heuristic for optimization problems
- Special Features:
  - combines information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

Phenotype and Genotype

- Nature encodes “programs” into DNA strings implicitly, describing structure and development of living organisms
- GA use binary strings to code information structures

Evolution Schemes

- Generates individuals
- Phenotypes of an individual, evolution
- Population of μ individuals
- Genetic operators:
  -Duplication
  - Mutation
  - Recombination
- Selection and evaluation:
  - Random selection
  - Selection
  - Evaluation
Evolution Schemes

\[
\left( \frac{\mu_0}{\rho_0} + \lambda_0 \| \Omega_0 \right)^{\gamma_0}
\]

\( \mu_0 \) number of parents;
\( \rho_0 \) size of the recombination pools;
\( \lambda_0 \) number of offspring;
\( \gamma_0 \) number of generations.

MH: Metaheuristics

Genetic Algorithms

- Original
- Crossover
- Mutation

1-point CrossOver

<table>
<thead>
<tr>
<th>parents</th>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</th>
<th>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

n-point CrossOver

<table>
<thead>
<tr>
<th>parents</th>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</th>
<th>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>0 0 0 0 1 1 0 0 0 0 0 0 0 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Uniform CrossOver

<table>
<thead>
<tr>
<th>parents</th>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</th>
<th>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>0 1 0 0 1 1 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Multi-Parent CrossOver
Polyploid CrossOver

Crossover

- n-point Crossover performance
- depends on the order of variables in the chromosome
- more likely to keep together genes that were closer
- Cannot bring together genes from opposite ends
- Known as Positional Bias

Can be exploited if we know about the structure of our problem ... not usually the case

Other representations

- Gray coding of integers (still binary chromosomes)
- Small changes in genotype cause small changes in phenotype
- mapping makes life easier for the GA
- Better to encode numerical variables directly as
  - integers
  - Floating point
- (generally accepted)

Real valued problems

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

$\Gamma(a_1, \ldots, a_L) \in [x_l, x_u]$

$\Gamma = \{0,1\}^L \in \{x_l, x_u\}$

defines the representation

$x \in [x_l, x_u]$ represented by $\{a_1, \ldots, a_L\} \in \{0,1\}^L$

- Mapping must be invertible (one phenotype per genotype)

Crossover for Real-Valued GAs

- Discrete:
  - each allele value in offspring $z$ comes from one of its parents $(x, y)$ with equal probability: $z = x_i$ or $y_i$
  - Could use n-point or uniform

- Intermediate
  - exploits idea of creating children “between” parents (a.k.a. arithmetic recombination)

- $z_i = \alpha x_i + (1 - \alpha) y_i \quad \alpha \in [0,1]$

- it can be:
  - constant: uniform arithmetical crossover
  - variable (e.g. depend on the age of the population)
  - picked at random every time

Real Values $\rightarrow$ Bit Strings

- Only $2^L$ values out of infinite are represented
- $L$ determines maximum precision of solution
- High precision $\rightarrow$ long chromosomes (slow evolution)
**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 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 \( \alpha = 0.5 \)

**Simple arithmetic crossover**

- Parents: \( \langle x_1, \ldots, x_n \rangle \) and \( \langle y_1, \ldots, y_n \rangle \)
- Pick random gene (\( k \)), mix values after this point
- Child 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 \( \alpha = 0.5 \)

**Whole arithmetic crossover**

- Parents: \( \langle x_1, \ldots, x_n \rangle \) and \( \langle y_1, \ldots, y_n \rangle \)
- Child is: \( \alpha \cdot \vec{x} + (1-\alpha) \cdot \vec{y} \)
- Reverse for other child. e.g. with \( \alpha = 0.5 \)

**Mutation**

- Alter each gene independently with a probability \( p_m \)
- \( p_m \) mutation rate \( \frac{1}{\text{pop_size}} \leq p_m \leq \frac{1}{\text{chromosome_length}} \)
- Most commonly used

**Floating Point Mutations**

- General scheme of floating point mutations
  \( \vec{x} = \langle x_1, \ldots, x_n \rangle \rightarrow \vec{x} = \langle x'_1, \ldots, x'_n \rangle \)
- Uniform mutation:
  \( x'_j \) drawn randomly (uniform) from \( [x_j, x_j] \)
- Analogous to bit-flipping (binary) or random resetting (integers)
**Crossover OR mutation?**

- Decade long debate: which one is better / 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
- 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

**Crossover OR mutation?**

- 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
- On the other hand
  - Only crossover can combine information from two parents
  - Only mutation can introduce new information (alleles)

**Crossover OR mutation?**

- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0’s on first bit in the population, 75% after performing n crossovers)
- To hit the optimum you often need a ‘lucky’ mutation

**Selection**

- Selection occurs in two places:
  - To take part in mating (parent selection)
  - From parents + offspring (survivor selection)
- Selection operators work on whole individual
  - i.e., they are representation-independent
- Selection
  - operators: define selection probabilities
  - algorithms: define how probabilities are implemented

**Fitness Proportionate Selection**

- Expected number of copies of $x_i$ in the mating pool
  $$E(x_i) = n \frac{f(x_i)}{\sum_{j} f(x_j)}$$
- Roulette wheel algorithm:
  - Spin a n-armed wheel n times to make n selections
  - No guarantees on actual value of $E(x_i)$
- Baker’s SUS algorithm:
  - $\mu$ evenly spaced arms on wheel - spin once
  - Guarantees
    $$\mu P(x_i) \leq E(x_i) \leq \mu P(x_i)$$

**Fitness Proportionate Selection**

- Fitness
- Probability
Fitness-Proporionate Selection

- Problems include:
  - Super-individuals rapidly take over; Premature Convergence
  - At end of runs, fitnesses are similar, lose selection pressure
  - Highly susceptible to function transposition

Scaling

- Scaling can fix last two problems
- Windowing: \( f'(x) = f(x) / \beta \)
- Sigma Scaling (Goldberg):
  - E.g. \( c = 2.0 \)
  - \( f'(x) = \max(f(x) - (f - c) \sigma_f), 0) \)

Function Transposition

Rank-based Selection

- Attempt to remove problems of FPS by basing selection probabilities on relative (position) rather than absolute fitness
- Rank population according to fitness and then base selection probabilities on rank where fittest has rank \( n \) and worst rank 1
- Sorting overhead - negligible wrt fitness evaluation

Linear Ranking

- \( p_r(x) = \frac{2 - s \cdot \frac{r(x) - 1}{\mu}}{\mu \cdot (\mu - 1)} \)
- \( s, 1.0 < s < 2.0 \)
- \( r(x) = \text{rank}(x) \)
- \( \mu = \text{offspring of best individual} \)

<table>
<thead>
<tr>
<th>Fitness</th>
<th>Rank</th>
<th>( P_{s]P} )</th>
<th>( P_{s]R} ) ( s = 2 )</th>
<th>( P_{s]R} ) ( s = 1.5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0.1</td>
<td>0</td>
<td>0.167</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>2</td>
<td>0.67</td>
<td>0.5</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>1</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>Sum</td>
<td>10</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Exponential Ranking

$$P_r(x_i) = \frac{1-e^{-r}}{c}$$

- Linear Ranking limits selection pressure
- Can allocate more than 2 copies to fittest individual
- $c$ chosen to abide by probability axioms

Tournament Selection

- Methods above rely on global population knowledge
- Could be a bottleneck esp. on parallel machines
- Absolute fitness function might not exist: e.g. game players, art design, face recognition, etc.
- Informal Procedure:
  1. Pick $k$ members at random then select the best of these
  2. Repeat to select more individuals

Tournament Selection

- Probability of selecting $x_i$ will depend on:
  1. $\text{Rank}(x_i)$
  2. Tournament size $k$
  3. Higher $k$ increases selection pressure
  4. Contestants picked with/without replacement
  5. Picking without replacement increases selection pressure
  6. 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

- Which $\lambda$ of the $\mu > \lambda$ parents should be replaced
- Two approaches:
  1. Age-Based Selection
  2. SSGA (steady state GA) implements as “delete-random” (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
  3. Fitness-Based Selection
  4. Using one of the methods above based on inverse fitness or rank

Two Special Cases

- GENITOR: a.k.a. “delete-worst”
  1. Rapid improvement in mean population fitness
  2. Rapid takeover (premature convergence)
  3. Use with large populations or “no duplicates” policy
- Elitism
  1. Widely used in both population models (GGA – Generational GA, SSGA)
  2. Keep at least one copy of the fittest solution so far

Population Models

- Generational
  1. Each individual survives for exactly one generation
  2. The entire set of parents is replaced by the offspring
- Steady-State
  1. One offspring per generation
  2. One individual replaced
- Generation Gap
  1. Proportion of population replaced
  2. $1.0$ for GGA, $1/pop\_size$ for SSGA
Example (Goldberg)

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

$x^2$ Example: Selection

<table>
<thead>
<tr>
<th>String no.</th>
<th>Initial population</th>
<th>$z$ Value</th>
<th>$f(x) = x^2$</th>
<th>$Prob$</th>
<th>Expected count</th>
<th>Actual count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1 0 1</td>
<td>13</td>
<td>141</td>
<td>0.14</td>
<td>0.58</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1 1 0 0 0</td>
<td>25</td>
<td>625</td>
<td>0.49</td>
<td>1.97</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0 1 0 0 0</td>
<td>64</td>
<td>22</td>
<td>0.06</td>
<td>0.22</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 1 1</td>
<td>361</td>
<td>1.23</td>
<td>0.31</td>
<td>1.23</td>
<td>1</td>
</tr>
<tr>
<td>Sum</td>
<td></td>
<td>1170</td>
<td>4.00</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>293</td>
<td>1.00</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td>579</td>
<td>1.97</td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

$x^2$ Example: Crossover

<table>
<thead>
<tr>
<th>String no.</th>
<th>Mating pool</th>
<th>Crossover point</th>
<th>Offspring after xover</th>
<th>$z$ Value</th>
<th>$f(x) = x^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1 0 1</td>
<td>4</td>
<td>0 1 1 0 0</td>
<td>12</td>
<td>144</td>
</tr>
<tr>
<td>2</td>
<td>1 1 0 0 0</td>
<td>2</td>
<td>1 1 0 0 1</td>
<td>25</td>
<td>625</td>
</tr>
<tr>
<td>3</td>
<td>1 0 0 0 0</td>
<td>2</td>
<td>1 1 0 1 1</td>
<td>27</td>
<td>729</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 1 1</td>
<td>2</td>
<td>1 0 0 0 0</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>Sum</td>
<td></td>
<td></td>
<td>1754</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>439</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
<td>729</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$x^2$ Example: Mutation

<table>
<thead>
<tr>
<th>String no.</th>
<th>Offspring after xover</th>
<th>Offspring after mutation</th>
<th>$z$ Value</th>
<th>$f(x) = x^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1 0 0</td>
<td>1 1 0 0 0</td>
<td>26</td>
<td>676</td>
</tr>
<tr>
<td>2</td>
<td>1 1 0 0 1</td>
<td>1 1 0 0 0</td>
<td>25</td>
<td>676</td>
</tr>
<tr>
<td>3</td>
<td>1 1 0 1 1</td>
<td>1 1 0 1 1</td>
<td>27</td>
<td>729</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 0 0</td>
<td>1 0 1 0 0</td>
<td>18</td>
<td>324</td>
</tr>
<tr>
<td>Sum</td>
<td></td>
<td></td>
<td>2354</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>588.5</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
<td>729</td>
<td></td>
</tr>
</tbody>
</table>

Simple Genetic Algorithms

<table>
<thead>
<tr>
<th>Representation</th>
<th>Binary strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recombination</td>
<td>N-point or uniform</td>
</tr>
<tr>
<td>Mutation</td>
<td>Bitwise bit-flipping with fixed probability</td>
</tr>
<tr>
<td>Parent selection</td>
<td>Fitness-Proportionate</td>
</tr>
<tr>
<td>Survivor selection</td>
<td>All children replace parents</td>
</tr>
<tr>
<td>Speciality</td>
<td>Emphasis on crossover</td>
</tr>
</tbody>
</table>