Artificial Intelligence Programming
Genetic Algorithms

Chris Brooks

Department of Computer Science
University of San Francisco
4-0: Genetic Algorithms

- States (potential solutions) are encoded as bitstrings
- Partial solutions are combined using crossover
- More fit strings are more likely to reproduce
- Local search - past history is not retained
- Probabilistic form of hill-climbing search
4-1: Genetic Algorithms

```plaintext
pop = makeRandomPopulation
while (not done)
    foreach p in pop
        p.fitness = evaluate(p)

for i = 1 to size(pop) by 2
    # select parents for reproduction
    parent1, parent2 = select two random solutions from pop
    [child1, child2] = crossover (parent1, parent2)
    mutate child1, child2

    replace old population with new population
```
How to select parents for reproduction?

Roulette selection weights the probability of each individual by its relative fitness.

**Algorithm:**

- Sum all fitnesses
- Foreach i in population
- RelativeFitness(i) = fitness(i) / sum

**Normalizing fitnesses**

- Relative fitnesses sum to 1.
- Can be used as probabilities.
Suppose we want to optimize $f(x) = x^2$ on $[0, 31]$

5 bits used to encode solution

Generate initial population

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
<th>relative fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td>169</td>
<td>0.144</td>
</tr>
<tr>
<td>11000</td>
<td>576</td>
<td>0.492</td>
</tr>
<tr>
<td>01000</td>
<td>64</td>
<td>0.055</td>
</tr>
<tr>
<td>10011</td>
<td>361</td>
<td>0.309</td>
</tr>
<tr>
<td>Total</td>
<td>1170</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Select parents using roulette selection
crossover at a random point.

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
<th>relative fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>169</td>
<td>0.144</td>
</tr>
<tr>
<td>1100</td>
<td>576</td>
<td>0.492</td>
</tr>
<tr>
<td>01000</td>
<td>64</td>
<td>0.055</td>
</tr>
<tr>
<td>10011</td>
<td>361</td>
<td>0.309</td>
</tr>
<tr>
<td>Total</td>
<td>1170</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Children:
- 01100 and 11001
Select parents using roulette selection and crossover at a random point.

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
<th>relative fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td>169</td>
<td>0.144</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>000</td>
</tr>
<tr>
<td>01000</td>
<td>64</td>
<td>0.055</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>011</td>
</tr>
<tr>
<td>Total</td>
<td>1170</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Children:
- 01100 and 11001
- 11011 and 10000
4-6: An Example

- Replace old population with new population.

- Apply mutation to the new population
  - With small population and low mutation rate (0.1%), no mutation occurs.

- New generation:
  - 01100
  - 11001
  - 11011
  - 10000

- Average fitness has increased (293 to 439)

- Maximum fitness has increased (576 to 729)
Subsolutions 11*** and ***11 are recombined to produce a better solution.

Correlation between strings and fitness:

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td>169</td>
</tr>
<tr>
<td>11000</td>
<td>576</td>
</tr>
<tr>
<td>01000</td>
<td>64</td>
</tr>
<tr>
<td>10011</td>
<td>361</td>
</tr>
</tbody>
</table>

A 1 in the first position seems correlated with fitness.
- This shouldn’t be very surprising.
- We’ll call a 1 in the first position a building block.
We need a way to talk about strings that are similar to each other.

Add ‘*’ (a ‘don’t care’ symbol) to \{0,1\}

A *schema* is a template that describes a set of strings, using \{0,1,*\}

- 111** matches 11100, 11101, 11110, 11111
- 0*11* matches 00110, 00111, 01110, 01111
- 0***1 matches 00001, 00011, 00101, 00111, 01001, 01011, 01101, 01111

Premise: Schema are correlated with fitness
GAs actually process schemata, rather than strings.

Crossover may or may not damage a schema

**11* vs 0***1

Short, highly fit schema of low order are more likely to survive without damage.

Order: The number of fixed bits in a schema.

- 1**** - order 1
- 0*1*1 - order 3
Defining length: The distance between the first and last specific bits.
- 011*1 - defining length 4
- **1*1 - defining length 2
- 0**** - defining length 0

Building block hypothesis: GAs recombine fit, short, low-order schema to produce progressively more fit solutions.
Suppose there are \( m \) examples of a schema \( H \) in a population of size \( n \) at time \( t \). \( m(H, t) \)

Strings are selected according to relative fitness. \( p = \frac{f}{\sum f} \)

At time \( t + 1 \), we will have \( m(H, t + 1) = m(H, t) n \frac{f(H)}{\sum f} \)
where \( f(H) \) is the average fitness of strings representing this schema.

Schema grow or decay according to their fitness relative to the rest of the population.

- Above average schemata will receive more samples
- Below average schemata will receive fewer samples

But how much above/below average?
Assume that schema $H$ remains above average by an amount $c$.

We can rewrite the schema difference equation as:

$$m(H, t + 1) = (1 + c)m(H, t)$$

Starting at $t = 0$, we get

$$m(H, t) = m(H, 0)(1 + c)^t$$

This is a geometric progression. (also compound interest formula)

Reproduction selects exponentially more (fewer) above (below) average schemata.
Selection is only half the story.

Schemata with longer defining length are more likely to be damaged by crossover.

\[ P(\text{survival}) \geq 1 - \frac{\text{defining length}}{\text{Strlen} - 1} \]

We can combine this with the previous equation.

\[ m(H, t + 1) \geq M(H, t) \frac{f(H)}{\Sigma f} 1 - \frac{\text{defining length}}{\text{Strlen} - 1} \]

We find that short, above-average fitness schemata are sampled at exponentially increasing rates.

This is known as the Schema Theorem.
Consider $H_1 = 1^{****}$, $H_2 = *10^{**}$, $H_3 = 1^{***0}$ in our previous example.

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
<th>relative fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td>169</td>
<td>0.144</td>
</tr>
<tr>
<td>11000</td>
<td>576</td>
<td>0.492</td>
</tr>
<tr>
<td>01000</td>
<td>64</td>
<td>0.055</td>
</tr>
<tr>
<td>10011</td>
<td>361</td>
<td>0.309</td>
</tr>
<tr>
<td>Total</td>
<td>1170</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Parents: 2 (twice), 3, 4
- 2 and 4 are both instances of $H_1$.

Children: 01100, 11001, 11011 and 10000
- 3 instances of $H_1$.
- Schema theorem predicts $m \frac{f(H)}{\Sigma f}$ copies
- $f(H_1) = 468.5$, pop average = 293, expected number: $\frac{468.5}{293} = 3.2$
4-15: Schema in our example

- Children: 01100, 11001, 11011 and 10000
- \( e(H_2) = 2 * \frac{320}{293} * 0.75 = 1.64 \)
- Actual \( H_2 \): 2
- \( e(H_3) = 1 * \frac{576}{293} * 0.2 = 0.39 \)
- Actual \( H_3 \): 1
- \( H_2 \) does well because of relatively good fitness and short defining length.
- Long defining length means that \( H_3 \) is usually destroyed.
4-16: Schemata and the Bandit Problem

- Imagine an $n$-armed slot machine.
- Problem: Choose a sequence of arm pulls that maximizes payoff.
- Solution: try the 'best' arm exponentially more often.
  - But you don’t know which one is best ahead of time.
- Schemata are “arms”
- It turns out that GAs generate a near-optimal solution to the $n$-armed bandit problem.
- Near-optimal sampling of space of schemata.
4-17: Theory vs. Implementation

6 The Schema Theorem tells us *why* GAs work in theory.

6 In practice, details of implementation can make a difference on the effectiveness of a GA.

6 Much of Project 1 deals with ways to implement GAs to solve practical problems more efficiently.
Roulette selection is effective, but can be computationally expensive.
- Every individual must be evaluated
- two iterations through entire population.

*Tournament selection* is a less expensive selection mechanism.

For each parent, choose two individuals at random.

Higher fitness gets to reproduce.
- To do crossover, choose two pairs and crossover the more fit of each pair.

Step 3 of your project asks you to implement tournament selection.
4-19: Elitism

- In practice, discarding all solutions from a generation can slow down a GA.
  △ Bad draw on the RNG can destroy progress.

- **Elitism** is the practice of keeping a fraction of the population from the previous generation.

- Use Roulette selection to choose a fraction of the population to carry over without crossover. (this is fractionRetained in your project)

- Varying the fraction you retain produces a tradeoff between current performance and learning rate.
Elitism is an example of a larger issue in learning.

Exploration is the process of gathering information or trying new solutions.

Exploitation is the process of choosing the currently best-seeming solution.

Issue: New solutions might be worse than what you’ve seen so far.

But, your best solution to date may not be the global optimum.

Should you learn more (how much?) or stop?
One challenge when working with GAs is in setting up the problem.

- Schema theorem tells us that short parameter encodings are good.
- Parameters that are dependent on each other should be located nearby each other.

N-queens: Assume that each queen will go in one column.

Problem: Find the right row for each queen.

$N$ rows requires $\log_2 N$ bits.

Entire string is of length $N \times \log_2 N$. 
Some problems don’t have a natural bitstring representation.

We may need to ensure that crossover produces valid solutions.

Traveling Salesperson Problem - find the shortest tour through cities (1, 2, ..., n) such that each city is visited once.

Alternatively, find the permutation of (1, 2, ..., n) with the minimum tour length.

Representation: List of cities
   ▲ 3, 1, 2, 4, 5

Fitness: TOURMAX - tour length (to turn minimization into maximization)
4-23: Partially Matched Crossover (PMX)

- How can we do crossover in this case?
- Exchange *positions* rather than substrings.
- Example:
  - $t_1$: 3 5 4 6 1 2 8 7
  - $t_2$: 1 3 6 5 8 7 2 4
- First, pick two points (loci) at random.
4-24: Partially Matched Crossover (PMX)

- t1: 3 5 | 4 6 1 2 | 8 7
- t2: 1 3 | 6 5 8 7 | 2 4

Use pairwise matching to exchange the corresponding cities in each tour.

△ In each string, 4 exchanges places with 6, 6 with 5, 1 with 8, and 2 with 7.
△ t1: 3 6 5 4 8 7 1 2
△ t2: 8 3 4 6 1 2 7 5

Intuition: Building blocks here are sections of a tour that should remain together.

This is Step 5 in your project.