Genetic Algorithms

Chapter 4.1.4

Introduction to Genetic Algorithms

• Another Local Search method
• Inspired by natural evolution:
  – Living things evolved into more successful organisms
  – offspring exhibit some traits of each parent
  – hereditary traits are determined by genes
  – genetic instructions are contained in chromosomes
  – chromosomes are strands of DNA
  – DNA is composed of base pairs (A,C,G,T), when in meaningful combinations, encode hereditary traits

Introduction to Genetic Algorithms

• Keep a population of individuals that are complete solutions (or partial solutions)
• Explore solution space by having these individuals interact and compete
  – interaction produces new individuals
  – competition eliminates weak individuals
• After multiple generations a strong individual (i.e., solution) should be found

Evaluation to Genetic Algorithms

• Mechanisms of evolutionary change:
  – **Crossover**: the (random) exchange of 2 parents’ chromosomes during reproduction resulting in offspring that have some traits of each parent

• Crossover requires genetic diversity among the parents to ensure sufficiently varied offspring
Introduction to Genetic Algorithms

- Mechanisms of evolutionary change:
  - **Mutation**: the rare occurrence of errors during the process of copying chromosomes resulting in
    - changes that are nonsensical/deadly, producing organisms that can’t survive
    - changes that are beneficial, producing “stronger” organisms
    - changes that aren't harmful or beneficial, producing organisms that aren't improved

- Natural selection: the fittest survive in a competitive environment resulting in better organisms
  - individuals with better survival traits generally survive for a longer period of time
  - this provides a better chance for reproducing and passing the successful traits on to offspring
  - over many generations the species improves since better traits will out number weaker ones

Representation of Individuals

- Solutions that can be represented as a vector of values
  - Satisfiability problem (SAT)
    - determine if a statement in propositional logic is satisfiable, for example:
      \[(P_1 \lor P_2) \land (P_1 \lor P_3) \land (P_1 \lor P_4)\]
    - each element corresponds with a proposition having a truth value of either true (i.e., 1) or false (i.e., 0)
    - vector: \[P_1 \ P_2 \ P_3 \ P_4\]
    - values: \[1 \ 0 \ 1 \ 1\] represents 1 individual
  - Traveling salesperson problem
    - Tour can be represented as a sequence of cities visited

Genetic Algorithm

1. Create initial random population
2. Evaluate fitness of each individual
3. Select parents according to fitness
4. Recombine parents to generate offspring
5. Mutate offspring
6. Replace population by new offspring
7. Termination criteria satisfied?
   - yes stop
   - no

- If yes, stop
- If no, repeat steps 2-6

- The Genetic Algorithm works by mimicking the process of natural selection, where the fittest individuals are more likely to pass their traits on to the next generation.
**Genetic Algorithm (1 version*)**

1. Let \( s = \{ s_1, \ldots, s_N \} \) be the current population
2. Let \( p[i] = f(s_i)/\text{SUM}_{j} f(s_j) \) be the fitness probabilities
3. for \( k = 1; \; k < N; \; k += 2 \)
   - Parent1 = randomly pick \( s_i \) with prob. \( p[i] \)
   - Parent2 = randomly pick another \( s_j \) with prob. \( p[j] \)
   - Randomly select 1 crossover point, and swap strings of parents 1 and 2 to generate children \( t[k] \) and \( t[k+1] \)
4. for \( k = 1; \; k \leq N; \; k++ \)
   - Randomly mutate each position in \( t[k] \) with a small prob.
5. New generation replaces old generation: \( s = t \)

*different than in book

**Initialization: Seeding the Population**

- Initialization sets the beginning population of individuals from which future generations are produced
- Concerns:
  - size of the initial population
    - experimentally determined for problem
  - diversity of the initial population (genetic diversity)
    - a common issue resulting from the lack of diversity is premature convergence to non-optimal solution

**Evaluation: Ranking by Fitness**

- Evaluation ranks the individuals by some fitness measure that corresponds with the quality of the individual solutions
- For example, given individual \( i \):
  - classification: \((\text{correct}(i))^2\)
  - TSP: \(1/distance(i)\)
  - SAT: \#ofTermsSatisfied(i)

**Initialization: Seeding the Population**

- How is a diverse initial population generated?
  - uniformly random: generate individuals randomly from a solution space with uniform distribution
  - grid initialization: choose individuals at regular "intervals" from the solution space
  - non-clustering: require individuals to be a predefined "distance" away from those already in the population
  - local optimization: use another technique (e.g. HC) to find initial population of local optima; doesn't ensure diversity but guarantees solution to be no worse than the local optima
Selection: Finding the Fittest

- **Choose which individuals survive and possibly reproduce in the next generation**
- Selection depends on the evaluation/fitness function
  - if too dependent, then, like greedy search, a non-optimal solution may be found
  - if not dependent enough, then may not converge to a solution at all
- Nature doesn't eliminate all "unfit" genes; they usually become recessive for a long period of time, and then may mutate to something useful

Selection Techniques

- **Proportional Fitness Selection**
  - each individual is selected proportionally to their fitness score
  - even the worst individual has a chance to survive
  - this helps prevent stagnation in the population
- Two approaches:
  - **rank selection**: individual selected with a probability proportional to its rank in population sorted by fitness
  - **proportional selection**: individual selected with a probability:
    \[
    \frac{\text{Fitness}(\text{individual})}{\text{SUM Fitness for all individuals}}
    \]

**Proportional selection** example:

- Given the following fitness values for population:
  - **Sum the Fitness**
    \[
    5 + 20 + 11 + 8 + 6 = 50
    \]
  - **Determine probabilities**
    \[
    \frac{\text{Fitness}(i)}{50}
    \]

<table>
<thead>
<tr>
<th>Individual</th>
<th>Fitness</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>10%</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>40%</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>22%</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>16%</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>12%</td>
</tr>
</tbody>
</table>

Selection Techniques

- **Tournament Selection**
  - randomly select two individuals and the one with the highest rank goes on and reproduces
  - cares only about the one with the higher rank, not the spread between the two fitness scores
  - puts an upper and lower bound on the chances that any individual has to reproduce for the next generation equal to \((2s - 2r + 1) / s^2\)
  - \(s\) is the size of the population
  - \(r\) is the rank of the "winning" individual
  - can be generalized to select best of \(n\) individuals
Selection Techniques

**Tournament selection** example:
- Given the following population and fitness:
  - Select B and D
  - B wins
  - Probability: \( \frac{2s - 2r + 1}{s^2} \)

<table>
<thead>
<tr>
<th>Individual</th>
<th>Fitness</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1/25 = 4%</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>9/25 = 36%</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>7/25 = 28%</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>5/25 = 20%</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>3/25 = 12%</td>
</tr>
</tbody>
</table>

B: \( s=5, r=1 \)  D: \( s=5, r=3 \)

Alteration: Producing New Individuals

- Alteration is used to produce new individuals

- **Crossover** for vector representations:
  - pick one or more pairs of individuals as parents and randomly swap their segments
  - also known as "cut and splice"

- Parameters:
  - crossover rate
  - number of crossover points
  - positions of the crossover points

Selection Techniques

- **Crowding**
  - a potential problem associated with the selection
    - occurs when the individuals that are most-fit quickly reproduce so that a large percentage of the entire population looks very similar
    - reduces diversity in the population
    - may hinder the long-run progress of the algorithm

Alteration: Producing New Individuals

- **1-point crossover**
  - pick a dividing point in the parents' vectors and swap the segments

- **Example**
  - given parents: 1101101101 and 0001001000
  - crossover point: after the 4th digit
  - children produced are:
    - \( 1101 + 001000 \) and \( 0001 + 101101 \)
Alteration: Producing New Individuals

- **N-point crossover**
  - generalization of 1-point crossover
  - pick n dividing points in the parents' vectors and splice together alternating segments
- **Uniform crossover**
  - the value of each element of the vector is randomly chosen from the values in the corresponding elements of the two parents
- Techniques also exist for permutation representations

Alteration: Producing New Individuals

- Alteration is used to produce new individuals
- **Mutation**
  - randomly change an individual
  - e.g. TSP: two-swap, two-interchange
  - e.g. SAT: bit flip
- Parameters:
  - mutation rate
  - size of the mutation

Genetic Algorithms as Search

- **Problem of Local Maxima**
  - individuals get stuck at pretty good but not optimal solutions
  - any small mutation gives worse fitness
  - crossover can help them get out of a local maximum
  - mutation is a random process, so it is possible that we may have a sudden large mutation to get these individuals out of this situation

Genetic Algorithms as Search

- GA is a kind of hill-climbing search
- One significant difference between GAs and HC is that, it is generally a good idea in GAs to “fill the local maxima up with individuals”
- Overall, GAs have less problems with local maxima than back-propagation neural networks
Summary

• Easy to apply to a wide range of problems
  – optimizations like TSP
  – inductive concept learning
  – scheduling
  – layout
• The results can be very good on some problems, and rather poor on others
• GA is very slow if only mutation is used; crossover makes the algorithm significantly faster