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

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
• “Simulated Evolution” via a form of Randomized Beam Search

Introduction to Genetic Algorithms

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

• Alteration 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 or 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 represented as a vector of values
- Satisfiability problem (SAT)
  - determine if a statement in propositional logic is satisfiable, for example:
    \[(P_1 \lor P_2) \land (P_3 \lor \neg P_4) \land (P_1 \lor \neg P_3) \land (\neg P_4 \lor \neg P_3)\]
  - each element corresponds to a symbol having a 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 \rightarrow rep. of 1 individual
- Traveling salesperson problem (TSP)
  - Tour can be represented as a sequence of cities visited

Genetic Algorithm

- Create initial random population
- Evaluate fitness of each individual
- Termination criterion satisfied?
- Select parents according to fitness
- Recombine parents to generate offspring
- Mutate offspring
- Replace population by new offspring
- Stop if yes, otherwise no
Gene Algorithm (1 version*)

1. Let \( s = \{ s_1, \ldots, s_N \} \) be the current population
2. Let \( p[i] = f(s_i) / \text{SUM} f(s_j) \) 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 two 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 probability
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
- Issues:
  - size of the initial population
    - experimentally determined for problem
  - diversity of the initial population (genetic diversity)
    - a problem resulting from lack of diversity is premature convergence to a non-optimal solution

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

Evaluation: Ranking by Fitness

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

• Deterministic Selection
  – relies heavily on evaluation/fitness function
  – converges fast
• Two approaches:
  – next generation is parents and their children
    • parents are the best of the current generation
    • parents produce children and survive to next generation
  – next generation is only the children
    • parents are the best of the current generation
    • parents are used to produce children only
    • parents don’t survive (counters early convergence)

Selection Techniques

• Proportional Fitness Selection
  – each individual is selected proportionally to their fitness score
  – even the worst individual has a chance to survive
  – 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: \( \frac{\text{Fitness}(\text{individual})}{\sum \text{Fitness for all individuals}} \)

Selection Techniques

• Proportional selection example:
  • Given the following fitness values for population:
  • Sum all the Fitnesses
    \[ 5 + 20 + 11 + 8 + 6 = 50 \]
  • 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 difference between the two fitness scores
  – defines a probability on the chances that any individual has to reproduce for the next generation equal to 
    \( \frac{2s - 2r + 1}{s^2} \)
  • \( s \) is the size of the population
  • \( r \) is the rank of the “winning” individual
  – can be generalized to select best \( n \) individuals

Tournament selection example:

<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>

Selection Techniques

Crowding: a potential problem associated with 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

• Alteration is used to produce new individuals
  • **Crossover** for vector representations:
    – Pick pairs of individuals as parents and randomly swap their segments
    – also known as "cut and splice"
  • Parameters:
    – number of crossover points
    – positions of the crossover points
Alteration: Producing New Individuals

• **1-point Crossover**
  – pick a dividing point in the parents’ vectors and swap their 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

• 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

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

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 \( k = 1; \ k < N; \ k += 2 \)
   • Randomly mutate each position in \( t[k] \) with a small prob.
5. New generation replaces old generation: \( s = t \)

*different than in book
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 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

• GA is a kind of hill-climbing search

• Very similar to a randomized beam 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 HC or neural networks
Summary

• Easy to apply to a wide range of problems
  – Optimization problems such as TSP
  – inductive concept learning
  – scheduling
  – Layout
• 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