Genetic Algorithms

Chapter 20, Section 8

Introduction to Genetic Algorithms (GA)

- **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 (ACGT), when in meaningful combinations, encode hereditary traits

Introduction to Genetic Algorithms

- **Mechanisms of evolutionary change:**
  - crossover: the random exchange of parent's chromosomes during reproduction resulting in:
    - offspring 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 are beneficial producing "stronger" organisms
    - changes aren't harmful/deadly but aren't beneficial producing organisms that aren't improved

Introduction to Genetic Algorithms

- **Mechanisms of evolutionary change:**
  - natural selection: in a competitive environment the fittest survive 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

Introduction to Genetic Algorithms

- **Keep a population of individuals that are complete 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**

Representation of Individuals

- **Some problems have solutions that can be represented as a vector of values:**
  - e.g. satisfiability problem (SAT):
    determine if a statement in propositional logic is satisfiable
    $(P_1 \wedge P_2) \vee (P_1 \wedge \neg P_3) \vee (P_1 \wedge \neg P_4) \vee (\neg P_3 \wedge \neg P_4)$
    - 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`
- **Some problems have solutions that can be represented as a permutation of values:**
  - e.g. traveling sales person problem (TSP)

Algorithm for Simulated Evolution

`Individual evolutionaryAlgorithm (Problem problem)`

- **`Problem`**: object that contains
  - **`initialize:`** returns an initial population of individuals
  - a population size (e.g. 100)
  - **`evaluate:`** sorts the population of individuals by fitness
  - **`atFitnessThreshold:`** test if desired solution quality reached
  - **`select:`** chooses individuals that survive
  - **`alter:`** generates new individuals
  - method that implements crossover
  - a fraction of population to be generated by crossover (e.g. 0.6)
  - method that implements mutation
  - a mutation rate (e.g. 0.001)
- **`Individual`**: the best solution found

Algorithm for Simulated Evolution

```
Individual evolutionaryAlgorithm(Problem problem) {
  Population currGen = problem.initialize();
  problem.evaluate(currGen);
  while (!problem.atFitnessThreshold(currGen)) {
    Population nextGen = problem.select(currGen);
    problem.alter(nextGen);
    currGen = nextGen;
    problem.evaluate(currGen);
  }
  return problem.mostFitIndividual(currGen);
}
```


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 on 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

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: *(correct(i))*^{2}
  - TSP: *distance(i)*
  - SAT: *#ofTermsSatisfied(i)*
  - walking animation: *subjective rating*
- **Often fitness functions have local minima.**

Selection: Finding the Fittest

- ***Choose which individuals survive and possibly reproduce in the next generation.***
- **Selection depends on the evaluation 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
  - converges fastest
- **Two approaches:**
  - next generation is parents and their children
    - parents are the best of the current generation
    - parents are used to produce children and survive
  - 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 evaluation 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 *Fitness(individual) / sum Fitness for all individuals*

Selection Techniques

Proportional selection example:

- **Given the following population and fitness:**
- **Sum the Fitness**

  $5 + 20 + 11 + 8 + 6 = 50$
- **Determine probabilities**

  Fitness(i) / 50

Individual	Fitness	Prob.
A	5	10%
B	20	40%
C	11	22%
D	8	16%
E	6	12%

Selection Techniques

- **Tournament Selection:**
  - randomly select two individuals and the one with the highest fitness wins to go on and reproduce
  - cares only about ranking, not the spread of scores
  - puts an upper and lower bound on the chances that any individual to reproduce for the next generation equal to: $(2n - 2m + 1) / n^2$
    - n is the size of the population
    - m 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:**

  $(2n - 2m + 1) / n^2$

Individual	Fitness	Prob.
A	5	1/25 = 4%
B	20	9/25 = 36%
C	11	7/25 = 28%
D	8	5/25 = 20%
E	6	3/25 = 12%

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

∗ ***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

- **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

Alteration: Producing new Individuals

- **1-point crossover:**
  - pick a dividing point in the parents' vectors and swap the segments
- **For 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 exist for permutation representations.**

Genetic Algorithms (GA) as Search

- **The problem of local maxima:**
  individuals stuck at a pretty good but not optimal
  - 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 (GA) as Search

- **GA is a kind of hill-climbing (HC) 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 back-propagation neural networks**

Summary

- **Easy to apply to a wide range of problems:**
  - optimization 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.**