Remind All Students:

• Final Project has been posted

Review of Last Class
• None.

Today's Lecture

• Genetic Optimizations, Concepts and examples
  No HW, though.
• Final Project discussion

Reading Assignment
• None.
Crossover and Mutation

Introduction

Crossover and mutation are two basic operators of GA. Performance of GA depends on them very much. The type and implementation of operators depends on the encoding and also on the problem.

There are many ways how to perform crossover and mutation. In this chapter we briefly describe some examples and suggestions how to perform them several encoding.

Binary Encoding

Crossover

Single point crossover - one crossover point is selected, binary string from the beginning of the chromosome to the crossover point is copied from the first parent, the rest is copied from the other parent

Two point crossover - two crossover points are selected, binary string from the beginning of the chromosome to the first crossover point is copied from the first parent, the part from the
first to the second crossover point is copied from the other parent and the rest is copied from the first parent again

\[
\begin{align*}
\text{Parent A} & \quad + \quad \text{Parent B} & = \quad \text{Offspring} \\
11001011 + 11011111 &= 11011111
\end{align*}
\]

**Uniform crossover** - bits are randomly copied from the first or from the second parent

\[
\begin{align*}
\text{Parent A} & \quad + \quad \text{Parent B} & = \quad \text{Offspring} \\
11001011 + 11011101 &= 11011111
\end{align*}
\]

**Arithmetic crossover** - some arithmetic operation is performed to make a new offspring

\[
\begin{align*}
\text{Parent A} & \quad + \quad \text{Parent B} & = \quad \text{Offspring} \\
11001011 + 11011111 &= 11001001 \quad (\text{AND})
\end{align*}
\]
Mutation

**Bit inversion** - selected bits are inverted

![Mutation Diagram](forums.nvidia.com)
**Roulette Wheel Selection**

The idea behind the roulette wheel selection technique is that each individual is given a chance to become a parent in proportion to its fitness. It is called roulette wheel selection as the chances of selecting a parent can be seen as spinning a roulette wheel with the size of the slot for each parent being proportional to its fitness. Obviously those with the largest fitness (slot sizes) have more chance of being chosen. Thus, it is possible for one member to dominate all the others and get selected a high proportion of the time.

Roulette wheel selection can be implemented as follows:
1. Sum the fitness of all the population members. Call this $TF$ (total fitness).
2. Generate a random number $n$, between 0 and $TF$.
3. Return the first population member whose fitness added to the preceding population members is greater than or equal to $n$.

**Tournament Selection**

Two methods are presented:

Select a pair of individuals at random. Generate a random number, $R$, between 0 and 1. If $R < r$ use the first individual as a parent. If the $R >= r$ then use the second individual as the parent. This is repeated to select the second parent. (The value of $r$ is a parameter to this method.)

Select two individuals at random. The individual with the highest evaluation becomes the parent. Repeat to find a second parent.
Evolutionary Optimization Readings and Misc.

- A course in Evolutionary Computing for Economics
  
  http://www.econ.iastate.edu/tesfatsi/evcomp.htm

  http://www.genetic-programming.org/

  http://www.oursland.net/projects/PopulationExperiment/ experiment online

  http://www.aaai.org/AITopics/html/genalg.html#tsp

  http://www.oursland.net/projects/lakeflock/
Genetic Algorithm Experiment

From http://www.oursland.net/projects/PopulationExperiment/

Overview

This applet demonstrates a continuous value genetic algorithm on a variety of problem spaces with a variety of reproduction methods. The problem spaces were adapted from Ackley(1). The Restart button reinitializes the population with random individuals. The Run/Pause button starts and stops the simulation. The Population Count field specifies how many individuals are initialized during a restart. You must press Enter in the field to set this value. The Reproduction Method choice changes how the next generation is created from the current generation. The Fitness Function choice changes the fitness landscape. The first time you go to a function, it will take a while to generate a picture of the landscape. Reproduction Divergence specifies how close child points are to the parent points during crossover. You must press in this field for the value to take effect. Random Divergence adds a random element to the reproduction divergence.

This program is meant to help visualize the processes in a genetic algorithm. The 2D space being searched is simple and easy to understand, but may not be generate behavior representative of higher dimension spaces. The result of crossing two points is the missing corner of an isosceles triangles that can be generated by adding a point to the two points. The divergence factor determines the angles of this triangle and thus the distance of the new point from the old points.
Descriptions of Problem Spaces or Fitness Functions

Function 1: A mostly maximal surface. Most searches do well here.

Function 2: A sloped plane with no local maximum. Hillclimbing is very effective here.

Function 3: A hill. Again there is no local maximum. Hillclimbing is very effective here too.

Function 4: Two hills, one slightly higher than the other. Helps demonstrate the dangers to converging too quickly.

Function 5: A sloped plane with a local maximum.

Function 6: A space with many local minimum. A global function still specifies the global maximum. A search method needs to be able to jump between peaks.

Function 7: The same space as function 6, but the local maximums are more pronounced. It is more difficult to jump between peaks.

Function 8: A space with large local maximum surrounding the global maximum. It is difficult to get to the global maximum without finding a local maximum.

Function 9: A space with a small area with value 1.0 and everything else zero. There is nothing to learn from here. All searches have difficulty with this type of space.
Descriptions of Reproduction Methods

**Fitness Roulette:** The probability of an individual being selected in the population is equal to the fitness value normalized with respect to the total fitness of the population. This method takes advantage of individuals that make large jumps in improvement by giving them a much greater chance of reproducing. The problem is that if the slope of the problem space in the area being explored can cause this preference to be either far too great or far too small. Watch this on Function 2. The population will generally converge about halfway up the slope. At this point, the fitness values of the individuals are very close and none of them has a distinct advantage to be selected for reproduction.

**Sorted Roulette:** Sorted roulette attempts to solve the problems of fitness roulette by sorting the population by fitness, and then selecting for reproduction with some bias toward the front of the list. This applet takes a random number between 1 and 0, squares it, and multiplies it by the size of the population to get the index of the reproducing individual. This technique does not take advantage of differences between fitness values, but it makes the fitness function much easier to evaluate.

**Fitness Generational:** It is based on the idea that individuals should be mated with individuals that are "close" to them. In many cases "close" does not have an obvious meaning. I am guessing that individuals with similar parents are close to each other. The GA maintains a tree of parents. One individual is selecting using fitness roulette, but the mate is found by travelling up the family tree a random distance and then randomly travelling back down to the current generation. This results in sub-populations that explore local areas of the fitness space. This technique tends to ignore global tendencies however since it converges on several local maximums very quickly. The basic idea is to preserve diversity in the population.

**Sorted Generational:** This is the same as fitness generational, but it uses a sorted roulette method to select the first individual.

**Hillclimb Best Individual:** This select the best individual in the population and generates random points around that individual.

**Elitist Random Search:** This moves the best individual to the next population and generates random values for the remainder.
Hill climbing

attempts to maximize (or minimize) a function $f(x)$, where $x$ are discrete states. These states are typically represented by vertices in a graph, where edges in the graph encode nearness or similarity of a graph. Hill climbing will follow the graph from vertex to vertex, always locally increasing (or decreasing) the value of $f$, until a local maximum (or local minimum) $x_m$ is reached. Hill climbing can also operate on a continuous space: in that case, the algorithm is called gradient ascent (or gradient descent if the function is minimized).

How well does each method work?

(1) Fitness Roulette vs Sorted Roulette (RD values)
(2) Variations among Reproduction Method
(3) Variations among Fitness Functions
(4) Other trials?
JavaEVA: A Java based framework for Evolutionary Algorithms

http://www-ra.informatik.uni-tuebingen.de/software/JavaEvA/download.html

JavaEVA is a framework for Evolutionary and general Optimization Algorithms implemented in Java. It is developed as resumption of the former Evolutionary Algorithms (EvA) software package which can handle complex optimisation problems. JavaEVA is divided into two parts: a swing-based graphical user interface (EvAClient) and the optimization kernel (EvAServer) which contains the different optimization algorithms.

Of the algorithms included in JavaEVA is the Genetic Algorithms.

JavaEVA will be used as a teaching aid to optimize the solutions to problems. Demonstrations of applications of GA will be made in class.

Reference:
@TECHREPORT{Streichert05JavaEvA,
    author    = {Felix Streichert and Holger Ulmer},
    title     = {{JavaEvA} - A Java Framework for Evolutionary Algorithms},
    institution = {Centre for Bioinformatics Tubingen, University of Tubingen},
    year      = {2005},
    type      = {Technical Report},
    number    = {WSI-2005-06},
    doi       = {urn:nbn:de:bsz:21-opus-17022},
    url       = {http://w210.ub.uni-tuebingen.de/dbt/volltexte/2005/1702/}
}