Habanero Extreme Scale Software Research Project
Comp215: Genetic Algorithms

Zoran Budimlić (Rice University)
One general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die
Darwin’s Theory of Evolution

Individual organisms in nature differ from one another. Some of this variation is inherited.

Organisms in nature produce more offspring than can survive, and many of those that survive do not reproduce.

Because each organism is unique, each has different advantages and disadvantages in the struggle for existence:
- competing for resources
- avoiding predators

Individuals best suited to their environment survive and reproduce most successfully.

Species change over time:
- Mutation
- New species arise, and other species disappear
A population of organisms
Each with a DNA

Cytosine, Thymine, Guanine, Adenine
Some are more fit than others

1 mph

4 mph

3 mph

9 mph

3 mph

5 mph
Those that survive, reproduce

Note: COMP215 does not endorse polygamy
Sometimes, mutations occur

TTGGCTAGC
TTGACATGC
GCCCTcATAC
CGTATATGC
AGAACGACAT
Sometimes, mutations occur

TTGCTAGC
TTGACATGC
GCTCAATCC
CGTATATGC
AGAACGCAT
Sometimes, with beneficial results
Great, but what’s this got to do with COMP215?

Genetic Algorithms
A class of probabilistic optimization algorithms
Inspired by the biological evolution process
Uses concepts of “Natural Selection” and “Genetic Inheritance” (Darwin 1859)
Originally developed by John Holland (1975)

Rely on
A population of solutions
Survival of the fittest
Variations generated by sexual reproduction and mutation
Genetic Algorithms

Particularly well suited for hard problems where little is known about the underlying search space

Widely-used in business, science and engineering

Binary Switches

Find a combination of binary switches that maximizes the reward
Optimization and search

Looking for a solution
Not the solution
A good solution

Approach
Maintain a population of candidate solutions for the problem at hand
Make it evolve by iteratively applying a set of stochastic operators:
  - **Selection** replicates the most successful solutions found in a population at a rate proportional to their relative quality
  - **Recombination** decomposes two distinct solutions and then randomly mixes their parts to form novel solutions
  - **Mutation** randomly perturbs a candidate solution
## The Metaphor

<table>
<thead>
<tr>
<th>Nature</th>
<th>Genetic Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>Optimization problem</td>
</tr>
<tr>
<td>Individuals living in that environment</td>
<td>Feasible solutions</td>
</tr>
<tr>
<td>Individual’s degree of adaptation</td>
<td>Fitness function</td>
</tr>
<tr>
<td>A population of organisms (species)</td>
<td>A set of feasible solutions</td>
</tr>
<tr>
<td>Selection, reproduction, mutation</td>
<td>Selection, recombination, mutation</td>
</tr>
<tr>
<td>Evolution of populations to suit their environment</td>
<td>Iteratively applying a set of stochastic operators on a set of feasible solutions</td>
</tr>
</tbody>
</table>
Example: Traveling Salesman Problem

TSP: visit **every** city in his territory **exactly once** and then return to the starting point; given the cost of travel between all cities, how to plan the itinerary for **minimum total cost** of the entire tour?

TSP: NP-Complete

---

BRUTE-FORCE SOLUTION:

\[ O(n!) \]

DYNAMIC PROGRAMMING ALGORITHMS:

\[ O(n^2 2^n) \]

SELLING ON EBAY:

\[ O(1) \]

STILL WORKING ON YOUR ROUTE?

SHUT THE HELL UP.

http://xkcd.com/399
A vector \( v = (i_1 \ i_2 \ \ldots \ i_n) \) represents a tour (\( v \) is a permutation of \( \{1,2,\ldots,n\} \)).

Fitness \( f \) of a solution is the inverse cost of the corresponding tour.

Initialization: either some heuristics, or a random sample of permutations of \( \{1,2,\ldots,n\} \).

Selection:

- Fitness-proportionate
TSP: Reproduction

Build offspring by choosing a sub-sequence of a tour from one parent and preserving the relative order of cities from the other parent.

Example:

\[ p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \text{ and } p_2 = (4 \ 5 \ 2 \ 1 \ 8 \ 7 \ 6 \ 9 \ 3) \]

Copy the segments between cut points into offspring:

\[ o_1 = (x \ x \ x \ 4 \ 5 \ 6 \ 7 \ x \ x) \text{ and } o_2 = (x \ x \ x \ 1 \ 8 \ 7 \ 6 \ x \ x) \]
TSP: Reproduction

Starting from the second cut point of one parent, the cities from the other parent are copied in the same order, skipping the ones already in the offspring

\[ p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \]
\[ p_2 = (4 \ 5 \ 2 \ 1 \ 8 \ 7 \ 6 \ 9 \ 3) \]

\[ o_1 = (x \ x \ x \ 4 \ 5 \ 6 \ 7 \ x \ x) \]
\[ o_2 = (x \ x \ x \ 1 \ 8 \ 7 \ 6 \ x \ x) \]
TSP: Reproduction

Starting from the second cut point of one parent, the cities from the other parent are copied in the same order, skipping the ones already in the offspring.

\[ p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) \]
\[ p_2 = (4\ 5\ 2\ 1\ 8\ 7\ 6\ 9\ 3) \]
\[ o_1 = (x\ x\ x\ 4\ 5\ 6\ 7\ x\ x) \]
\[ o_2 = (x\ x\ x\ 1\ 8\ 7\ 6\ x\ x) \]
TSP: Reproduction

Starting from the second cut point of one parent, the cities from the other parent are copied in the same order, skipping the ones already in the offspring.

\[ p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \]
\[ p_2 = (4 \ 5 \ 2 \ 1 \ 8 \ 7 \ 6 \ 9 \ 3) \]

\[ o_1 = (x \ x \ x \ 4 \ 5 \ 6 \ 7 \ x \ x) \]
\[ o_2 = (x \ x \ x \ 1 \ 8 \ 7 \ 6 \ x \ x) \]
TSP: Reproduction

Similarly for the other offspring

\[ p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \]
\[ p_2 = (4 \ 5 \ 2 \ 1 \ 8 \ 7 \ 6 \ 9 \ 3) \]

\[ o_1 = (x \ x \ x \ 4 \ 5 \ 6 \ 7 \ x \ x) \quad \rightarrow \quad (2 \ 1 \ 8 \ 4 \ 5 \ 6 \ 7 \ 9 \ 3) \]
\[ o_2 = (x \ x \ x \ 1 \ 8 \ 7 \ 6 \ x \ x) \quad \rightarrow \quad (3 \ 4 \ 5 \ 1 \ 8 \ 7 \ 6 \ 9 \ 2) \]
TSP: Mutation

The sub-string between two randomly selected points in the path is reversed.

Example:

(1 2 3 4 5 6 7 8 9) is changed into (1 2 7 6 5 4 3 8 9)

This simple inversion guarantees that the resulting offspring is also a legal tour.
GA’s are appropriate if:

- You have no clue where to start
- There is more than one good answer
- Your algorithm doesn’t need to guarantee the best solution before the house burns down

GA’s are not appropriate:

- When exact global optima are needed.
- When any guarantee on quality of solution or convergence time is needed
- When “appropriate” representations of solutions are not available
Some existing GA applications

Designing jet engines
Designing walking strategies for legged robots.
Scheduling job shop
Classifying news stories for Dow Jones
Creating art, jazz improvisations
TSP
Drug design
Compiler code generation
Genetic music
Facial image creation
Acoustics
Genetic Algorithms: Pretty Pictures

A picture is really just a function: \((x, y) \rightarrow (R, G, B)\)

Set of intrinsic functions: \(|x|, \sin(y), +, -, *, /, \text{Perlin noise, Dissolve, e.t.c.}\)

Complex function: a tree of intrinsic functions

Simplification: scale everything to the \((-1, -1)\) to \((1, 1)\) range

**Selection:** User picks the “prettiest” ones

**Reproduction:** Combine two function trees to create a new tree

**Mutation:** Randomly change a node in the tree to something else

Genetic Algorithms: Pretty Pictures
Genetic Algorithms: Pretty Pictures
Genetic Algorithms: Pretty Pictures
Live Coding: Pretty Pictures on the Web

http://gregstoll.dyndns.org/ppga/#details