Heuristic Optimization

Genetic Algorithms

José M PEÑA (jmpena@fi.upm.es)
(Universidad Politécnica de Madrid)
Heuristic Optimization (GA)

Overview

- A class of heuristic optimization algorithms
- Inspired by the biological evolution process
- Uses concepts of “Natural Selection” and “Genetic Inheritance” (Darwin 1859)
- Originally developed by John Holland (1975)
- Particularly well suited for hard problems where little is known about the underlying search space
- Widely-used in business, science and engineering
Natural Evolution

Darwin: The Origin of Species

- Search of *optimal* forms (fittest in the environment)
- Based on:
  - assortment: recombination of genetic material
  - randomness
  - selection: survival of the fittest
Chromosome is divided in parts: **genes**
Genes code for properties
Possible values of the genes: **allele**
Position of the gene in the chromosome: **locus**
Biological Concepts II

- The entire combination of genes: **genotype**
- A genotype is expressed as a **phenotype**
- During reproduction “errors” occur
- Due to these “errors” genetic variation exists
- Most important “errors” are:
  - Recombination (cross-over)
  - Mutation
Genetic Algorithms (GAs)

John Holland (~1973)
- GAs use the principle of natural selection to solve complex optimization problems.
- An alternative of brute-force search on a complex solution space.
  - Instead of complete exploration
  - Estocastic-driven search
- Mathematical expression of success in life: **fitness**
- “Preservation of favourable variations and rejection of unfavourable variations.”
Genetic Algorithms: Definitions

Concepts:
- **Individual**: A possible solution of the problem.
- **Gene**: A solution component (a.k.a. an attribute) [eye color]
- **Allele**: A gene value [eye color=green]
- **Population**: group of potential solutions

**Assumption**: there exists a relationship between genetic information and individual fitness.
Fitness Landscapes

**Fitness:**
Function that evaluates individual capabilities.

Graphical representation
N+1 dimensions

© L.J. Pit, '96
Natural Evolution and Operators

How new individuals are created?

- Crossover (recombination)
  - Mixing existing genetic material, assortment
  - Inheritance of useful characteristics
- Mutation:
  - Rare (very rare in biological perspective)
  - Randomness

They simulate the evolution process
The new individuals created are called offspring
Evolutionary Algorithms

\[ P = \text{GenerateInitialPopulation()} \]
\[ \text{Evaluate}(P) \]
\[ \text{while} \ \text{termination conditions not met} \ \text{do} \]
\[ P' = \text{Recombine}(P) \]
\[ P'' = \text{Mutate}(P') \]
\[ \text{Evaluate}(P'') \]
\[ P = \text{Select}(P'' \cup P) \]
\[ \text{endwhile} \]
Natural Evolution and Operators

Crossover

Variants:
- 2, 3, n-points crossover
- Uniform crossover
- Other...
Natural Evolution and Operators

Mutation

Variants:
- Mutation-by-swap
- Biased mutation
- Cataclysmic mutation
Natural Evolution and Operators

Selection:

1. Evaluation of individuals (fitness)
2. Parent selection (for each crossover operation).

Selective pressure: relationship between capacities (fitness) of the individual and its possibilities to generate new individuals (participate in the reproduction).

Alternatives:

- Tournament selection
- Rank-based: Continuous function
- Roulette wheel
Canonical Genetic Algorithm

- **Initial population** ($P_0$)
- **Selected population** ($P'_i$)
- **Fitness-function**
- **End condition?**
- **Selection**
- **Crossover/mutation**
- **Next population** ($P_{i+1}$)

Includes the chances of participation in reproduction
GA Variants (Replacement)

Elitism:
- The best(s) individuals in $P_i$ stay in population $P_i$.
- Provides monotonic fitness increment

Steady-state:
- Only one individual is created per generation.
- New one replaces worst individual.
Replacement Variants

- **Elitism:**
  - The best(s) individuals in Pi stay in population Pi.
  - Provides monotonic fitness increment

- **Steady-state:**
  - Only one individual is created per generation.
  - New one replaces worst individual

<table>
<thead>
<tr>
<th></th>
<th>Canonical</th>
<th>Steady-state</th>
<th>Elitism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offspring size</td>
<td>N</td>
<td>1</td>
<td>&gt;, = or &lt; N</td>
</tr>
<tr>
<td>Current → Next</td>
<td>0</td>
<td>N-1</td>
<td>&lt;&lt; N</td>
</tr>
<tr>
<td>Replacement</td>
<td>All</td>
<td>Worst</td>
<td>Worst</td>
</tr>
</tbody>
</table>
Parameters and Functions

- Chromosome representation
- Creation of initial population
- Fitness function
- Genetic operators
- Parameters (population size, probabilities of the genetic operators, etc.)

Typical configuration for small problems:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size:</td>
<td>50 – 100</td>
</tr>
<tr>
<td>Children per generation:</td>
<td>= population size</td>
</tr>
<tr>
<td>Crossovers:</td>
<td>&gt; 85%</td>
</tr>
<tr>
<td>Mutations:</td>
<td>&lt; 5%</td>
</tr>
<tr>
<td>Generations:</td>
<td>20 – 20,000</td>
</tr>
</tbody>
</table>
Heuristic Optimization (02 GA)

Differences between GA and Conventional Search Algorithms

- GA works on a coding of the parameters set, not the parameter themselves
- GA searches from a population of points, not a single point
- GA uses only a payoff function, and no domain knowledge
- GA uses probabilistic transition rules, not deterministic ones
- GA can provide a number of potential solutions to a given problem. The final choice is left to the user.
Conclusions

“Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime.”

Salvatore Mangano

Computer Design, May 1995
Programming with GAlib

Provides:
- Different algorithms
- Different genomes
GA Example

Programming a problem requires:
- To define the fitness function
- To define the genome format
- To setup the algorithm parameters (most of them have default values).

```cpp
float Objective(GAGenome&)
{
    /*objective function goes */
}

main()
{
    // Create a genome
    GA1DBinaryStringGenome genome(size, Objective);
    // Create the GA
    GASimpleGA ga(genome);
    // Evolve the GA
    ga.evolve();
    // Print out the results
    cout << ga.statistics() << endl;
}
```
Extra Features

Defining genetic operators (see GAGenome)
- g.corssover(...)  g.mutator(...)

Bounding gene values (see GAAlleleSet)
- GARealAlleleSet a(0,1)  GAAlleleSet<int> a(5,v)

Evolution loop control (see GAAlgorithm)
- ga.done()  ga.population()
- ga.step()  ga.generation()