Genetic Algorithms

An Introduction

Benjamin Kudria
CSE 352

This Presentation

What are Genetic Algorithms?

When can we use them?

How do they work?

An Example – Binary Numbers

Why should we use them?

Why shouldn't we use them?

Related techniques

This Presentation

What are Genetic Algorithms?

When can we use them?

How do they work?

An Example – Binary Numbers

Why should we use them?

Why shouldn't we use them?

Related techniques

What are Genetic Algorithms?

Evolutionary Search optimization algorithms

Techniques inspired by Biology, such as:

- Evolution (Fitness, Selection)
- Mutation (Crossover, etc)

Can search large spaces somewhat intelligently and quickly

This Presentation

What are Genetic Algorithms?

When can we use them?

How do they work?

An Example – Binary Numbers

Why should we use them?

Why shouldn't we use them?

Related techniques

When can we use them?

Large complex search space

Many levels of correctness for a potential solution

We can encode a solution with a small amount of data

We can quickly and precisely, tell how good a potential solution is.

This Presentation

What are Genetic Algorithms?

When can we use them?

How do they work?

An Example – Binary Numbers

Why should we use them?

Why shouldn't we use them?

Related techniques

General Technique

- Encode the problem, and select an initial population
- Select the most fit of each generation, create an offspring population
- Replace unselected solutions with the new offspring to obtain a new population.
- Repeat until:
  - There is a suitably-fit solution
  - A certain number of generations or computational time elapse
  - Successive repetitions reach a plateau and no better solutions are found

Implementing

Define the problem, and decide how to encode a potential solution

Write a fitness function, to determine the degree of "correctness" for any solution

Define how we select the most fit solutions:

- Usually top X% percent, but there are other strategies

Determine how to breed individual solutions:

- **Crossover:** selecting large sections of a solution from one parent, and others from another
- **Mutation:** randomly changing the elements of the children, with some probability, to avoid local optima

Select a termination condition

This Presentation

What are Genetic Algorithms?

When can we use them?

How do they work?

An Example – Binary Numbers

Why should we use them?

Why shouldn't we use them?

Related techniques

Example – Binary Numbers

Problem: Which bitstring encodes a specific number in binary?

Each solution (genotype) is a string of bits

Our fitness function converts the bitstring into decimal, and subtracts it from the goal

We stop when we have found the bitstring, i.e., difference is 0.

I used a library called Charlie, written by Sander Land

`http://charlie.rubyforge.org`

Code Example

```
SIZE = 30
MAX = 2**SIZE
MIN = MAX / SIZE
N = rand(MAX - MIN) + MIN

class Number < BitStringGenotype(size)
    def fitness
        -(number - N).abs
    end

    def number
        bitstring.to_i(2)
    end

    def bitstring
        genes.map(&:to_s).join
    end

    def to_s
        "#{bitstring} (#{number.to_s})"
    end
end
```

We find a random number, and how big it might be.

We define the genotype

The fitness function

Convert it to a number

Benchmarking

We can also specify multiple strategies to test, and compare with mutation, crossover, and selection strategies are best for our problem.

```
GABenchmark.benchmark(Number, 'output.html') do
    selection \
    RandomSelection,
    TruncationSelection,
    TruncationSelection(0.5),
    TruncationSelection(0.9),
    BestOnlySelection,
    ScaledRouletteSelection,
    TournamentSelection

    crossover \
    SinglePointCrossover, TwoPointCrossover, ThreePointCrossover, NPointCrossover(10),
    UniformCrossover,
    BlendingCrossover,
    BlendingCrossover(0.2, :cube),
    BlendingCrossover(0.5, :cube),
    BlendingCrossover(0.9, :cube),
    BlendingCrossover(0.2, :line),
    BlendingCrossover(0.5, :line),
    BlendingCrossover(0.9, :line)

    mutator \
    ListMutator(:expected_n[1], :flip),
    ListMutator(:expected_n[5], :flip),
    ListMutator(:expected_n[15], :flip)

    repeat 20
    generations 100
end
```

This Presentation

What are Genetic Algorithms?

When can we use them?

How do they work?

An Example – Binary Numbers

Why should we use them?

Why shouldn't we use them?

Related techniques

Why should we use GAs?

Sometimes, depending on the problem, they can find a solution **very fast** in a large problem space.

Implementing a GA is not too difficult.

Your other option is exhaustive search.

This Presentation

What are Genetic Algorithms?

When can we use them?

How do they work?

An Example – Binary Numbers

Why should we use them?

Why shouldn't we use them?

Related techniques

Why shouldn't we use GAs?

Writing a good fitness function for your problem may be hard.

The fitness "landscape" may cause a population to converge on a local optima, and thus miss a global optimum.

If your problem can only tell you if a solution is either right or wrong, GAs cannot search effectively.

> (However, if the test can be repeated with varying results, a ratio of right to wrong can be used.)

Computationally expensive, although easily parallelizable.

This Presentation

What are Genetic Algorithms?

When can we use them?

How do they work?

An Example – Binary Numbers

Why should we use them?

Why shouldn't we use them?

Related techniques

Related Techniques

Simulated Annealing

Useful when the search space is discrete

Can, to a degree, avoid local optima

Genetic Programming

Use a GA to evolve a program to solve instances of your problem efficiently

Memetic Algorithms

New technique, individuals undergo self-improvement in each generation.

Swarm Intelligence

Ant-colony Optimization

Individuals leave "pheromones" to direct later iterations in the proper direction.

Bees Algorithm

Mimics honey-bee foraging behavior, teaches other individuals where "food" (optima/ridge) is.

Particle Swarm Optimization

Each individual is given a velocity, heading is adjusted towards particles that have performed better

Often are able to adapt to a changing problem space, and can thus run continually.

Applications in network routing, urban traffic routing, etc.

Sources

Fraser, Alex (1957). "Simulation of genetic systems by automatic digital computers. I. Introduction". Aust. J. Biol. Sci. 10: 484–491.

Fraser, Alex; Donald Burnell (1970). Computer Models in Genetics. New York: McGraw-Hill.

Crosby, Jack L. (1973). Computer Simulation in Genetics. London: John Wiley & Sons.

Fogel, David B. (editor) (1998). Evolutionary Computation: The Fossil Record. New York: IEEE Press.

Koza, J.R. (1992), Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press

V. Cerny, A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. Journal of Optimization Theory and Applications, 45:41-51, 1985

M. Dorigo, 1992. Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italy.

J. Kennedy, and R. Eberhart, Particle swarm optimization, in Proc. of the IEEE Int. Conf. on Neural Networks, Piscataway, NJ, pp. 1942–1948, 1995.

Land, Sander, 2008, Charlie – A Genetic Algorithms Library For Ruby, `http://charlie.rubyforge.org`