Genetic Algorithms

December 19-21, 2012,
Svetlin Penkov
The Main Evolutionary Computing Metaphor

**EVOLUTION**

- Environment
- Individual
- Fitness

**PROBLEM SOLVING**

- Problem
- Candidate Solution
- Quality

Fitness → chances for survival and reproduction

Quality → chance for seeding new solutions
Brief History: the ancestors

• 1948, Turing: proposes “genetical or evolutionary search”

• 1962, Bremermann: 
  optimization through evolution and recombination

• 1964, Rechenberg: 
  introduces evolution strategies (online parameter optimization)

• 1965, L. Fogel, Owens and Walsh: 
  introduce evolutionary programming (optimize a program’s values)

• 1975, Holland: 
  introduces genetic algorithms

• 1992, Koza: 
  introduces genetic programming (optimize a program structure)
Darwinian Evolution: Survival of the Fittest

- All environments have finite resources
  (i.e., can only support a limited number of individuals)

- Life forms have basic instinct lifecycles geared towards reproduction

- Therefore some kind of selection is inevitable

- Those individuals that compete for the resources most effectively have increased chance of reproduction
Darwinian Evolution: Summary

- Population consists of diverse set of individuals

- Combinations of traits that are better adapted tend to increase representation in population

  Individuals are “units of selection”

- Variations occur through random changes yielding constant source of diversity, coupled with selection means that:

  Population is the “unit of evolution”
Adaptive Landscape Metaphor

- Can envisage population with $n$ traits as existing in an $n+1$-dimensional space (landscape) with height corresponding to fitness

- Each different individual (phenotype) represents a single point on the landscape

- Population is therefore a “cloud” of points, moving on the landscape over time as it evolves - adaptation
Example with two traits
Example with two traits

Trait 1

Trait 2

Fitness
Example with two traits
Example with two traits
Adaptive Landscape Metaphor

• Selection “pushes” population up the landscape

• Genetic drift:
  • random variations in feature distribution
  • can cause the population “melt down” hills, thus crossing valleys and leaving local optima
Demo BoxCar2D

BoxCar2D
Genes and the Genome

- Genes are encoded in strands of DNA called chromosomes

- The complete genetic material in an individual’s genotype is called the Genome

- Within a species, most of the genetic material is the same
Example: Homo Sapiens

- Human DNA is organised into chromosomes
- Human body cells contain 23 pairs of chromosomes which together define the physical attributes of the individual:
Genetic code

• All proteins in life on earth are composed of sequences built from 20 different amino acids

• DNA is built from four nucleotides in a double helix spiral: purines A,G; pyrimidines T,C

• Triplets of these from codons, each of which codes for a specific amino acid

• Much redundancy:
  • purines complement pyrimidines
  • the DNA contains much rubbish
  • \(4^3 = 64\) codons code for 20 amino acids
  • genetic code = the mapping from codons to amino acids

• For all natural life on earth, the genetic code is the same!
Transcription, translation

A central claim in molecular genetics: only one way flow

Genotype \[\rightarrow\] Phenotype
Genotype \[\rightarrow\] Phenotype
Applications: Optimal Pipe Shape

- Rechenberg, 1960
Applications: Vibration Minimization

- NASA
- Launched as a functional part of the ST-5 satellite in 2006
Applications: FPGA Oscillator

- John Bird and Paul Layzell, Sussex University, 2002
- Exploit unpredicted features of the environment

Evolutionary “creativity”
Applications: The Eyebot
Applications: The Eyebot
Neuro-Evolving Robotic Operatives

Control Point Bravo is being Challenged!
Robocode

Video
Walking Hexapod

- Video
Evolving Faces

- Video
Artificial Life

- Life as it could be...
- Video
What is an Evolutionary Algorithm?
A population of individuals exists in an environment with limited resources.

Competition for those resources causes selection of those fitter individuals that are better adapted to the environment.

These individuals act as seeds for the generation of new individuals through recombination and mutation.

The new individuals have their fitness evaluated and compete (possibly also with parents) for survival.

Over time Natural selection causes a rise in the fitness of the population.
General Scheme of EAs

- Initialization
- Population
- Parent Selection
- Parents
- Recombination
- Mutation
- Offspring
- Survivor Selection
- Termination
BEGIN

INITIALISE population with random candidate solutions;
EVALUATE each candidate;
REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO
  1 SELECT parents;
  2 RECOMBINE pairs of parents;
  3 MUTATE the resulting offspring;
  4 EVALUATE new candidates;
  5 SELECT individuals for the next generation;
OD
END
Components of EAs

- Representation
- Evaluation
- Population
- Parent Selection
- Recombination
- Mutation
- Survivor Selection
- Termination
Representations

- Candidate solutions (individuals) exist in phenotype space

- They are encoded in chromosomes, which exist in genotype space
  - Encoding: phenotype => genotype
  - Decoding: genotype => phenotype

- Chromosomes contain genes
Evaluation (Fitness) Function

- Represents the environment requirements that the population should adapt to

- Assigns a single real-valued fitness to each phenotype which forms the basis for selection
  - So the more discrimination (different values) the better

- Typically we talk about fitness being maximised
  - Some problems may be best posed as minimisation problems, but conversion is trivial
Population

- Holds representations of possible solutions
- Usually has a fixed size and is a multiset of genotypes
- Selection operators usually operate on the whole population
- **Diversity** of a population refers to the number of different fitnesses / phenotypes / genotypes present
Assigns variable probabilities of individuals acting as parents depending on their fitnesses

Usually probabilistic

- high quality solutions more likely to become parents than low quality but not guaranteed
- even worst in current population usually has non-zero probability of becoming a parent

This stochastic nature can aid escape from local optima
Variation Operators

- Role is to generate new candidate solutions

- Usually divided into two types according to their **arity** (number of individuals):
  - Arity 1: mutation operators
  - Arity >1: Recombination operators
  - Arity = 2 typically called **crossover**

- There has been much debate about relative importance of recombination and mutation
  - Nowadays most EAs use both

*Choice of particular variation operators is representation dependant!*
Mutation

- Acts on one genotype and delivers another
- Element of randomness is essential
- Provides “fresh blood” in the population
Recombination

- Merges information from parents into offspring
- Choice of what information to merge is stochastic
- Most offspring may be worse, or the same as the parents
- Hope is that some are better by combining elements of genotypes that lead to good traits
Most EAs use fixed population size so need a way of going from (parents + offspring) to next generation

Often deterministic
- Fitness based: e.g., rank parents + offspring and take best
- Age based: make as many offspring as parents and delete all parents

Sometimes do combination (elitism): the best individuals are always retained independent
Initialisation / Termination

- Initialisation usually done at random,
  - Need to ensure even spread and mixture of possible allele values
  - Can include existing solutions, or use problem-specific heuristics, to “seed” the population

- Termination condition checked every generation
  - Reaching some (known/hoped for) fitness
  - Reaching some maximum allowed number of generations
  - Reaching some minimum level of diversity
  - Reaching some specified number of generations without fitness improvement
Example: The 8 Queens Problem

Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other.
Example: Representation

Phenotype:
a board configuration

Genotype:
a permutation of the numbers 1 - 8
Example: Fitness Evaluation

- Penalty of one queen: the number of queens she can check.
- Penalty of a configuration: the sum of the penalties of all queens.
- Note: penalty is to be minimized
- Fitness of a configuration: inverse penalty to be maximized
Example: Mutation

- Small variation in one permutation, e.g.
  - swapping values of two randomly chosen positions,
Example: Recombination

- Combining two permutations into two new permutations:
  - choose random crossover point
  - copy first parts into children
  - create second part by inserting values from other parent:

```
<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>5</th>
<th>2</th>
<th>6</th>
<th>4</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>5</th>
<th>4</th>
<th>2</th>
<th>8</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
```
Example: Selection

- Parent selection:
  - Pick 5 parents and take best two to undergo crossover

- Survivor selection (replacement)
  - When inserting a new child into the population, choose an existing member to replace by:
    - sorting the whole population by decreasing fitness
    - enumerating this list from high to low
    - replacing the first with a fitness lower than the given child
**Example: Summary**

<table>
<thead>
<tr>
<th>Representation</th>
<th>Permutations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recombination</td>
<td>“Cut-and-crossfill” crossover</td>
</tr>
<tr>
<td>Recombination probability</td>
<td>100%</td>
</tr>
<tr>
<td>Mutation</td>
<td>Swap</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>80%</td>
</tr>
<tr>
<td>Parent selection</td>
<td>Best 2 out of random 5</td>
</tr>
<tr>
<td>Survival selection</td>
<td>Replace worst</td>
</tr>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Number of Offspring</td>
<td>2</td>
</tr>
<tr>
<td>Initialisation</td>
<td>Random</td>
</tr>
<tr>
<td>Termination condition</td>
<td>Solution or 10,000 fitness evaluation</td>
</tr>
</tbody>
</table>

Note: That is **only one possible** set of choices of operators and parameters.
Typical Behaviour of an EA

Phases in optimizing on a 1-dimensional fitness landscape

Early phase:
quasi-random population distribution

Mid-phase:
population arranged around/on hills

Late phase:
population concentrated on high hills
Typical run of an EA shows so-called “anytime behavior”
Are long runs beneficial?

- Answer:
  - it depends how much you want the last bit of progress
  - it may be better to do more shorter runs
EAs as Problem Solvers

- Evolutionary algorithm
- Random search
- Special, problem tailored method
Hybrid EAs as Problem Solvers

- Evolutionary algorithm
- Special, problem tailored method
- EA enriched with knowledge
ANNs guarantee that best point found is *locally optimal*, but problems often exhibit many local optima.

EAs do not guarantee local (nor global) minimum and the fitness does not necessarily improve at every step.

**No Free Lunch Theorem (Wolpert and Macready, 1997):**

“any two optimization algorithms are equivalent when their performance is averaged across all possible problems”
It is important to choose the “right” representation for the problem we are trying to solve.

Getting the representation right is one of the most difficult tasks and this comes only with practice.

Possible representations:
- Binary
- Integer
- Float
- Permutations
Mutation for Integer Representations

- Random Resetting — select a new valid value for a gene with probability $p_m$

- Creep Mutation — randomly add a small perturbation $\alpha$ to the gene where $\alpha$ is usually drawn from a zero mean normal distribution with controlled variance
Mutation for Permutation Representations

- **Swap**
  
  Input: 1 2 5 7 6 9 3 0
  
  Output: 1 2 3 7 6 9 5 0

- **Insert**
  
  Input: 1 2 5 7 6 9 3 0
  
  Output: 1 2 5 3 6 9 7 0

- **Scramble**
  
  Input: 1 2 5 7 6 9 3 0
  
  Output: 1 2 6 3 7 5 9 0

- **Inverse**
  
  Input: 1 2 5 7 6 9 3 0
  
  Output: 1 2 3 9 6 7 5 0
Recombination for Binary Representations

- 1-Point Crossover

Parents:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Children:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```
Recombination for Binary Representations

- **N-Point Crossover**

Parents:
- 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Children:
- 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1
- 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0
Recombination for Binary Representations

Uniform Crossover

Parents

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Children

0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 1
1 0 1 1 0 0 0 0 1 1 1 0 1 0 0 1 1 0
Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

There is co-operation AND competition between them

- Crossover is explorative, it makes a big jump to an area somewhere “in between” two (parent) areas
- Mutation is exploitative, it creates random small diversions, thereby staying near (in the area of ) the parent

- To hit the optimum you often need a ‘lucky’ mutation
Implementing Sudoku Solver

- Sudoku solving is a combinatorial problem, so a genetic algorithm is not the most elegant way to solve it - mainly because we do know how to solve a Sudoku puzzle.

- Use classic problem state space search and propagate constraints to minimize the search space.

- Anyway, it is an interesting application that works well in most of the cases.
GA Sudoku Solver: The Problem
GA Sudoku Solver: Representation

Phenotype:  
a puzzle configuration represented as a 2D 9x9 array

Genotype (chromosome):  
A 1D array with 81 cells that contain 9 permutations of the digits from 1 to 9
**GA Sudoku Solver: Representation**

- **Blueprint chromosome:**
  A 1D array with 81 cells that contain the initially given digits in each cell or 0 if the cell’s digit is not provided in the beginning.

*Note:* It is possible to use a more general genome containing only 81 digits (ignoring the givens as well), but the search space becomes too large.
The first 9 elements of the chromosome:

\[
\begin{array}{ccccccc}
7 & 1 & 6 & 3 & 4 & 8 & 9 \\
\end{array}
\]

... 

The first 9 elements of the blueprint chromosome:

\[
\begin{array}{ccccccc}
0 & 0 & 6 & 3 & 0 & 0 & 0 \\
\end{array}
\]

...
Single Column Fitness:
The number of cell pairs with different digits in a single column. Maximum value is:
$8 + 7 + \ldots + 1 = 36$

Total Column Fitness:
The sum of all columns’ fitness estimates. Maximum value is:
$9 \times 36 = 324$
**Single Sub-box Fitness:**
The number of cell pairs with different digits in a single 3x3 sub-box.
Maximum value is:
\[8 + 7 + \ldots + 1 = 36\]

**Total Sub-box Fitness:**
The sum of all 3x3 sub-boxes’ fitness estimates. Maximum value is:
\[9 \times 36 = 324\]

**Total Fitness:**
The sum of the total column and sub-box fitness estimates. Maximum value is:
\[2 \times 324 = 648\]
In order to preserve the permutations in each row use a swap mutation.
• Use single point crossing over where the splitting point could be only between entire rows

• Thus we ensure the integrity of the row permutations
Population:
- Use a population of size $N = 10 \times \text{SUDOKU\_ORDER}$

Parent selection:
- Randomly pick two different parents
- Generate $N$ more children

Survivor selection (replacement)
- Pick the first $N$ individuals from the $2N$ population

Reset the population every 10k iterations
class Sudoku

//Constructor that creates a random sudoku
Sudoku(const std::vector<int>& digits);

//Constructor that creates a sudoku individual
//by using two parents.
Sudoku(const Sudoku& dad, const Sudoku& mom);

//Calculates the fitness of the individual.
inline void CalcFitness()

//Less than operator used to sort the population
bool operator < (const Sudoku& other) const;
class Sudoku

//Randomly mutates the individual by swapping
//digits in a row with given probability.
void Mutate(double prob);

//The function performs crossover
//between two individuals
void CrossOver(const Sudoku& dad,
               const Sudoku& mom);

//Updates the phenotype according to the chromosome
void UpdatePhenotype();
class GASolver

//Solve a Sudoku using a genetic algorithm
static Sudoku Solve(std::vector<int> digits);
Conclusions

- Experiment with the presented methods and code

- Plenty of information on the Internet:
  - Google scholar
  - Free journals
  - Online books
  - Online courses:
    - Coursera
    - Udacity
Questions & Discussions