Machine Learning

Evolutionary Algorithms

- genetic representation of candidate solutions
- genetic operators
- selection scheme
- problem domain
WWW-Resources

- Genetic Programming Notebook
  - http://www.geneticprogramming.com
    software, people, papers, tutorial, FAQs

- Hitch-Hiker’s Guide to Evolutionary Computation
  - http://alife.santafe.edu/~joke/encore/www
    FAQ for comp.ai.genetic

- Genetic Algorithms Archive
    Repository for GA related information, conferences, etc.

- EVONET European Network of Excellence on Evolutionary Comp.
  : http://www.dcs.napier.ac.uk/evonet

Literature

- Goldberg, D. “Genetic Algorithms in Search and Optimization”
  Addison-Wesley, Reading MA, 1989

- Mitchell, M. “An Introduction to Genetic Algorithms”

- Koza, J. “Genetic Programming II”

- Holland, J. “Adaptation in Natural and Artificial Systems”
  Univeristy of Michigan Press, Ann Arbor, 1975

- Bäck, Th. “Evolutionary Algorithms in Theory and Practice”
  Oxford University Press, New York, 1996
**Outline**

- simple genetic algorithm (SGA)
- examples
  - brachystrochone problem
  - lens optimization
  - prisoner’s dilemma
  - traveling salesman problem
- schema theorem
- evolution strategies
- genetic programming

**Biological Terminology**

- gene
  - functional entity that codes for a specific feature e.g. eye color
  - set of possible alleles
- allele
  - value of a gene e.g. blue, green, brown
  - codes for a specific variation of the gene/feature
- locus
  - position of a gene on the chromosome
- genome
  - set of all genes that define a species
  - the genome of a specific individual is called genotype
  - the genome of a living organism is composed of several chromosomes
- population
  - set of competing genomes/individuals
Genotype versus Phenotype

- **genotype**
  - blue print that contains the information to construct an organism e.g. human DNA
  - genetic operators such as mutation and recombination modify the genotype during reproduction
  - genotype of an individual is immutable (no Lamarckian evolution)

- **phenotype**
  - physical make-up of an organism
  - selection operates on phenotypes (Darwin's principle: “survival of the fittest”)

Evolutionary Algorithm

- population of genotypes
- coding scheme
- selection
- phenotype space
- fitness
- mutation
- recombination
Pseudo Code of an Evolutionary Algorithm

Create initial random population
Evaluate fitness of each individual
Termination criteria satisfied?
Select parents according to fitness
Recombine parents to generate offspring
Mutate offspring
Replace population by new offspring

A Simple Genetic Algorithm

- optimization task: find the maximum of \( f(x) \)
  for example \( f(x) = x \cdot \sin(x) \quad x \in [0,?] \)
- genotype: binary string \( s \in \{0,1\}^5 \) e.g. 11010, 01011, 10001
- mapping: genotype - phenotype
  binary integer encoding: \( x = \sum_{i=1}^{n=5} s_i \cdot \frac{2^{n-i-1}}{(2^n-1)} \)

<table>
<thead>
<tr>
<th>genotype</th>
<th>integ.</th>
<th>phenotype</th>
<th>fitness</th>
<th>prop. fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>11010</td>
<td>26</td>
<td>2.6349</td>
<td>1.2787</td>
<td>30%</td>
</tr>
<tr>
<td>01011</td>
<td>11</td>
<td>1.1148</td>
<td>1.0008</td>
<td>24%</td>
</tr>
<tr>
<td>10001</td>
<td>17</td>
<td>1.7228</td>
<td>1.7029</td>
<td>40%</td>
</tr>
<tr>
<td>00101</td>
<td>5</td>
<td>0.5067</td>
<td>0.2459</td>
<td>6%</td>
</tr>
</tbody>
</table>
Roulette Wheel Selection

- selection is a stochastic process
- probability of reproduction \( p_i = \frac{f_i}{\sum_k f_k} \)

- intermediate parent population:
Genotype Operators

- recombination (crossover)
  - combines two parent genotypes into a new offspring
  - generates new variants by mixing existing genetic material
  - stochastic selection among parent genes

- mutation
  - random alteration of genes
  - maintain genetic diversity

- in genetic algorithms crossover is the major operator whereas mutation only plays a minor role

Crossover

- crossover applied to parent strings with probability $p_c \in [0.6..1.0]$?
- crossover site chosen randomly

- one-point crossover
  
  parent A: 1 1 0 1 0
  parent B: 1 0 0 0 1

  offspring A: 1 1 0 1
  offspring B: 1 0 0 0

- two-point crossover
  
  parent A: 1 1 0 1 0
  parent B: 1 0 0 0 1

  offspring A: 1 1 0 0 1
  offspring B: 1 0 0 1
Mutation

- mutation applied to allele/gene with probability $P_m$ [0.001..0.1]
- role of mutation is to maintain genetic diversity

offspring: 1 1 0 0 0
Mutate 4th gene (bit flip)
mutated offspring: 1 1 0 1 0

Evolution of a Brachystrochrone

- objective: optimize track between start and end point such that a frictionless point mass travels the path in minimal time
- phenotype parameters $y_i$: height $h$ of track at point $x_i$
- fitness: time expired to get from start to end point
- parameter encoding: gray-encoded bit strings
Brachystrochrone with a GA

Evolution of Fitness
Brachystrochrone Best Solution

![Graph showing the Brachystrochrone solution over generations with travel time of 0.3579s]

Evonet Flying Circus Brachy.

[http://www.wi.leidenuniv.nl/~gusz/Flying_Circus](http://www.wi.leidenuniv.nl/~gusz/Flying_Circus)

- Brachystrochrone
- Prism Lens
- Traveling Salesman
- Symbolic Regression
- ...

..\Demos\index.html
..\Demos\index_prism.htm
Traveling Salesman Problem

- objective: find the shortest tour that visits each city (NP-hard)
- a tour is encoded by a sequence AEFCBGD which specifies the order in which cities are visited
- fitness: overall length of the tour
- phenotype (tour) is defined by the permutation of genes rather than their values

• usual crossover results in invalid tours
  AEF|CBGD  AEFDAB
  ECG|DFAB  ECGCBGD
• edge recombination operator
  parent 1 AEFCBDG
  parent 2 ECGDFAB

1. Select a start city
2. Select the neighbor with the minimum number of connections (break ties randomly)
3. Repeat 2 until no more cities are left

\Demos\index_salesman.htm
Prisoners Dilemma

<table>
<thead>
<tr>
<th></th>
<th>cooperate</th>
<th>defect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cooperate</strong></td>
<td>3,3</td>
<td>0,5</td>
</tr>
<tr>
<td><strong>defect</strong></td>
<td>5,0</td>
<td>1,1</td>
</tr>
</tbody>
</table>

- objective: maximize payoff in prisoners dilemma
- encode next action (d=0, c=1) for next move depending on the possible outcome (5-bits) in the previous round against the same opponent
- Last move: cc cd dc dd initial move
- Action: 1 0 1 0 1 Tit for Tat
- 0 0 0 0 0 always defect
- fitness: payoff obtained in a tournament against the other members in the current population (dynamic fitness function)

Extensions to the Simple GA

- Encoding schemes
  - gray encoding
  - messy genetic algorithms
- Replacement schemes
  - generational replacement
  - steady state replacement
- Fitness scaling
  - linear scaling
  - ? - truncation
  - ranking
- Selection schemes
  - stochastic sampling
  - tournament selection
Fitness Scaling

• linear scaling: $f_k^* = af_k + b$
  • choose $a, b$ such that the best individual gets $m$ (m ~ two) offspring and the average individual gets one offspring

• $\sigma$-truncation: $f_k^* = f_k - (f_{\text{avg}} - \sigma)$
  • selection pressure related to the fitness distribution

• normalizing: $f_k^* = (f_k - f_{\text{min}} + ?) / (f_{\text{max}} - f_{\text{min}} + ?)$
  • dynamic scaling

• ranking: $p_k = q - (k-1)r$
  • choose $q$ and $r$ such that $p_k = 1$
  • sort the population from the best ($k=1$) to the worst ($k=n$)
  • assign selection probability according to ranking

Replacement Schemes

• generational replacement
  • entire population is replaced each generation
  • non-overlapping population

• steady state replacement
  • a single individual (worst, random) is replaced by one offspring
  • overlapping population
Selection Schemes

- stochastic sampling
  - roulette wheel selection
  - spin wheel $N$ times

- stochastic universal sampling
  - roulette wheel selection
  - single spin, wheel has $N$ equally spaced markers

- tournament selection
  - choose $k$ candidates at random with uniform probability
  - pick best one for reproduction
  - expected number of offspring
    - best : $k$, average : $k^{1/2}$, worst : $k/N$

Gray-Coding

- proper genetic representation of candidate solutions is crucial for the success of an evolutionary algorithm

- in general similar phenotypes should correspond to similar genotypes

- Hamming distance : number of different bits among two binary strings

- Standard base two encoding : e.g. 1000 and 0111 have maximal Hamming distance but correspond to the adjacent integers 7 and 8 so called Hamming-cliff
Gray-Coding

- adjacent integers are represented by bit strings that only differ in one bit

- given a binary string $s_1, \ldots, s_n$, coding the integer in the standard way the conversion to a Gray coded string $g_1, \ldots, g_n$ is:

$$g_k = \begin{cases} 
    s_1 & : \text{if } k=1 \\
    s_{k+1} \oplus s_k & : \text{if } k>1
  \end{cases}$$

? Denotes addition modulo 2:
0 ? 0 = 0, 0 ? 1 = 1, 1 ? 0 = 1, 1 ? 1 = 0

<table>
<thead>
<tr>
<th>Integer</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>Gray</td>
<td>000</td>
<td>001</td>
<td>011</td>
<td>010</td>
<td>110</td>
<td>111</td>
<td>101</td>
<td>100</td>
</tr>
</tbody>
</table>

Concept of a Schema

Schema
- template of ‘1’, ‘0’ and ‘*’ (wild cards)
- instances of a schema are strings that match the template
- a schema corresponds to a subspace in genotype space

Schema $h=**1$ : \{001, 011, 101, 111\}
Concept of a Schema

- order $o(H)$ of a schema $H$: #defined (0,1) bits
- defining length $d(H)$ of a schema $H$: distance between outermost defined bits

<table>
<thead>
<tr>
<th>Schema H</th>
<th>$o(H)$</th>
<th>$d(H)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1**0*</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>**<em>0</em></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1**10</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Schema Theorem

- $m(h,t)$: #instances of schema $h$ in the population at generation $t$
- $f(h,t)$: observed average fitness of schema $h$ at generation $t$
  
  $f(h) = \sum_{x} P(t) f(x)/m(h,t)$

- $E(m(h,t))$: expected #instances of schema $h$

- selection:
  
  $E(m(h,t+1)) = \sum_{x} P(t) f(x)/<f> = m(h,t) f(h,t)/<f>$

- crossover: probability that a schema $h$ of defining length $d(h)$ is not destroyed by crossover: $(1 - p_c d(h)/(l-1))$

- mutation: probability that schema $h$ of order $o(h)$ is not destroyed by mutation: $(1 - p_m)^{o(h)}$
Schema Theorem

\[ E(m(h,t+1)) = m(h,t) f(h,t)/\langle f \rangle (1 - p_c d(h)/(l-1)) (1 - p_m)^{o(h)} \]

- simple GA increases the number of schemata with
  - low order
  - short defining length
  - above average fitness
- implicit parallelism
  - simultaneous evaluation of a large number of schemata in a population of n strings
  - one string implicitly samples \( 2^l \) different schemata
- building block hypothesis
  - GA works by recombining instances of good schemata to form instances of equally good or better higher-order schemata by means of crossover

Genetic Programming

- automatic generation of computer programs by means of natural evolution see Koza 1999
- programs are represented by a parse tree (LiSP expression)
- tree nodes correspond to functions:
  - arithmetic functions \{+,-,*,/\}
  - logarithmic functions \{\sin,\exp\}
- leaf nodes correspond to terminals:
  - input variables \{X_1, X_2, X_3\}
  - constants \{0.1, 0.2, 0.5\}

Tree is parsed from left to right:

\[ (+ X_1 (* X_2 X_3)) \rightarrow X_1 + (X_2 * X_3) \]
Genetic Programming: Crossover

parent A

parent B

offspring A

offspring B

Block-Stacking Problem

- objective: place the blocks in the correct order such that the stack forms the word universal
- functions: set of actions, logical operators, do-until loop
- terminals: set of sensors that indicate top block on stack, next suitable block on table etc.
- each program tree is tested on 166 different initial configurations
- fitness: #configurations for which the stack was correct after program execution
Sensors:
- CS: current stack, name of the top block of the stack
- TB: top correct block, name of the topmost block on the stack such that it and all blocks underneath are in correct order
- NN: next block needed, name of the block needed above TB

Functions:
- MS(X): move block X to the top of the stack, return value X
- MT(X): moves the block on top of the stack to the table if X is anywhere in the stack returns X
- DU(exp1, exp2): execute exp1 until the predicate exp2 becomes true
- NOT(exp1): negation of expression exp1
- EQ(exp1, exp2): returns true if exp1 and exp2 are equal
Block-Stacking Problem

• (EQ (MS NN) (EQ (MS NN) (MS NN)))
  move next needed block to the stack three times in a row
  (succeeds with a stack VERSAL and U N I on the table)
• (DU (MS NN) (NOT NN))
  move next needed block to the stack until no more blocks
  are needed
• (EQ (DU (MT CS) (NOT CS)) (DU (MS NN) (NOT NN)))
  empty the stack on the table and then build the stack in
  the correct order
• (EQ (DU (MT CS) (EQ (CS TB))) (DU (MS NN) (NOT NN)))

Evolution Strategy vs.
Genetic Algorithms

<table>
<thead>
<tr>
<th>genetic representation:</th>
<th>evolutionary strategy</th>
<th>genetic algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>real valued</td>
<td>1. mutation</td>
<td>1. recombination</td>
</tr>
<tr>
<td>binary</td>
<td>2. recombination</td>
<td>2. mutation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>importance of genetic operators:</th>
<th>self adaptation:</th>
<th>selection:</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
<td>deterministic</td>
</tr>
<tr>
<td>no</td>
<td></td>
<td>stochastic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>problem domain:</th>
</tr>
</thead>
<tbody>
<tr>
<td>continuous optimisation problems</td>
</tr>
<tr>
<td>discrete, combinatorial optimisation problems</td>
</tr>
</tbody>
</table>