Evolutionary Computation

Introduction and overview

- EC models the process of evolution as found in nature (biological evolution).
- Charles Darwin vs Jean-Babtiste Lamarck
- Lamarckian view:
  - Inheritance
  - Adapt over lifetime
  - Use and disuse
- Darwinian view (EC is largely based on this view):
  - Natural Selection (The Origin of Species)
  - All species share common ancestors
  - Survival of the fittest (Fitness)
  - Also inheritance (because of reproduction)
  - Mutation

- Evolutionary Algorithms (EA) in AI is an umbrella term for all the algorithms that model biological evolution (evolve solutions) and is a subset of EC.
- EC Paradigms (EAs)
  - Genetic algorithms (GA) - models genetic evolution
  - Genetic programming (GP) - GA's cousin
  - Evolutionary programming (EP) - phenotypic evolution
  - Evolutionary strategies (ES) - evolution of evolution, with strategy parameters
  - Differential evolution (DE) - GA with a different reproduction system (crossover)
  - Cultural evolution - models the way in which culture (the general beliefs or preferences of a population) influences the evolution of a population (genetic and phenotypic)
  - Co-evolution - two or more "species" compete or work together, evolving towards a solution. Predator-prey relationship with competitive co-evolution vs cooperative coevolution.

Genetic Programming:

- John Holland, widely considered as the father of GA
- Holland was not the first to propose GA but he popularized the idea

- Genotypic evolution i.e. the genetic characteristics of individuals
- Key characteristics of GAs:
  - Selection
  - Reproduction (crossover)
  - Mutation
  - Fitness
- The survival of the fittest

Search/optimization method

- GAs are used to find solutions in some search space similar to the way a local search method works (How does local search work?).
- Formally: A population based, parallel search or optimization algorithm.
- We have group of proposed solutions which are then continually modified through the GA operators to obtain possibly better solutions.
- Optimization in terms of trying to "optimize" the *fitness* of proposed solutions
- Liken to function optimization (Example?)

Representation schemes

- How do we represent *solutions*?

- Solution = individual = chromosome
- Usually have more than one individual in a population
- The very first representation schemes where bitstrings (Holland)
  - 1010010111010
  - What can we do with bitstrings?
- Nowadays n-dimensional vectors with real valued elements are popular
- Can actually have n-dimensional vectors with mixed-valued elements
- Terminology:
  - *Chromosome*: A proposed solution produced and manipulated by the GA operators
  - *Gene*: An element of a chromosome. One of the variables in the problem domain.
  - *Allele*: A value for a gene at a particular point in time.
  - *Population*: A collection of chromosomes.
  - *Generation*: A population at a given point in time during the execution of the GA. Epochs vs Generations

7.10	50	1.0	5	6	0.12	0.001	0.95

Figure 1: An example chromosome

Fitness Function

- Once again: Survival of the fittest
- Fitness is a measure how good a solution is, i.e. the quality of a proposed solution.
- Almost like the heuristic function used in game trees but not quite the same thing.
- The better the solution represented by the chromosome, the fitter the chromosome is.
- Fitness is a function that we are trying to maximize or minimize.
- It is often (but not always) the distance in search space from some desired solution (e.g. Euclidean distance).
- A fitness function F takes a chromosome C and produces a fitness value for C.
  - $Fitness = F(C)$
- Often used to drive selection.
- Absolute vs Relative.
- Objective vs Subjective.
- Stopping criterion.

- Example?

Crossover

- Represents reproduction as found in nature.
- Recombines parent genes to create offspring.
- Applied at some probability of crossover.
- Offspring (usually) succeed parent chromosomes for the next generation.
- Types of crossover:
  - Asexual (one parent)
  - Sexual (two parents, most common)
  - Multi-recombination (three or more parents)
- How do we apply crossover?
  - Assume we selected 2 parents and generate 2 offspring
  - One-point crossover
  - Two-point crossover
  - Uniform (n-point) crossover
  - For bitstrings we generate a mask to do crossover.
- Example?

Mutation

- Small changes in genetic makeup to (hopefully) render an advantage to survive in the environment.
- Adaptability?
- As a rule, mutation is only applied to offspring.
- *Main purpose:* To introduce (and possibly maintain) genetic diversity in a population, i.e. explore more solutions.
- Prevents stagnation of a population.
- Exploration vs Exploitation.
- Usually applied at a low probability P, why?
- Better strategy for the probability of mutation?
- Which chromosomes do we want to mutate more?
- How is mutation applied?
  - Bitstrings.
  - Floating points.
  - Other?

Selection

- Very important, used for almost all the GA operators to decide on which chromosomes to operate.

- Very big friends with fitness function (fitness is more often than not used as a parameter in the selection process).
- Cat breeder?
- Used to:
  - Select parents to crossover.
  - Select chromosomes to mutate
  - Next population!!!
- Selective pressure:
  - The importance of fitness.
  - High vs Low selective pressure.
  - Exploration vs Exploitation.
  - Stagnation?
- Types of selection operators:
  - Random selection
  - Proportional selection
  - Roulette wheel selection (form of proportional selection)
    * Danger?
    * High selective pressure, why?
  - Tournament selection (chromosomes compete among one another)
    * High or low selective pressure?
  - Rank-based selection
    * Linear vs Nonlinear

    * Arranges chromosomes and then selects index i.
    * i $\sim U(0, U(0, n-1))$, n = the number of chromosomes in the population.
  - Elitism (best chromosomes survive to next generation).
  - Hall of fame (over all generations, best solutions thus far)

Stopping conditions

- Can't execute indefinitely, we would like a solution at some stage.
- When can we stop?
  - Fitness, we found a solution!
  - Maximum generations.
  - Stagnation.
    * No improvement in fitness (average of population or best) for a number of generations.
    * The population becomes homogenous.

Pseudocode

1. Initialize the generation counter $G = 0$
2. Create and initialize a population P_0, containing N chromosomes
3. While NOT(**stopping conditions**) do
    a) Calculate the fitness of each chromosome C_i in P_G, *f(Ci) for i = 1,..,N*
    b) Create a new population O to contain the off-spring and populate it by applying **crossover** to **selected** chromosomes from P_G
    c) Apply **mutation** at some probability to the chromosomes in O, the offspring
    d) Select the next population P_{G+1} from P_G and O, $P_{G+1} = Select(P_G, O)$
    e) Increment the generation counter, $G = G + 1$
4. end while

- Chromosome representation?
- Initialization?
- Stopping conditions?
- Fitness function?
- Selection?
- Crossover?
- Mutation?

Example application

- Function optimization.
- Finding solutions to systems of equations.
- Neural Network training.
- Combinatorial problem solving in general.
- Data clustering.
- Solve N-queens problem.
- Routing, scheduling, planning (TSP which is NP-complete!).
- Picture evolution.
- Intelligent agents.
- Code breaking.
- MANY MORE!

Genetic Programming:

Intro

- Originally proposed by John R. Koza in the late eighties (1989).
- Used to evolve computer programs (e.g. S-expressions for LISP).
- A bit more computationally expensive than other EAs.
- Almost exactly the same thing as GAs, biggest difference is the representation scheme used.

Representation

- Trees.
- What can be represented as trees?
  - Mathematical expressions and functions.
  - Boolean expressions (AND, OR, NOT).
  - Programming languages.
  - Decision trees.
  - Game trees.
  - Graphics or pictures???
  - More?

- How do we read trees?
  - AND's (down the branch to a child node) and OR's (between sibling nodes)

Grammar

- Rules used to combine the "primitives" of a language into useful and meaningful constructs.
- Defines valid or legal ways to combine simple entities (words, punctuation, variables etc.) into more complex structures.
- Terminal set, function set and semantics.
- Example: Mathematical expressions
- Terminal set (corresponds to leaf nodes) = {1,..,9, all valid variable names e.g. x, y and z}.
- Function set (corresponds to internal or non-leaf nodes) = {+,- ,* ,/ , log, ln, =, etc}.
- Then the grammar could be:
  - Expr ::= Binary | Unary | Terminal
  - Binary ::= Expr BINOP Expr
  - Unary ::= UNOP Expr
  - Terminal ::= {variables and numbers}
  - BINOP ::= {+,-,*,/}
  - UNOP ::= {log, sin, cos, tan etc.}
- Note how naturally trees and grammars fit together.

Mutation and crossover

- Used for the same purposes in GP as in GAs
- Type checking becomes an issue.
- Mutation:
  - Growing
  - Shrinking (or truncation).
  - Node swapping.
  - Randomly altering node elements (beware non-terminal and terminal).
  - Replacements of nodes.
  - Which node?
  - Which operators?
- Crossover:
  - Any crossover technique as for GA.
  - Usually one-point crossover.
  - One offspring vs two offspring.

Fitness

- Size?
- Accuracy (depends on application and if we are evolving decision trees)
- Number of patterns covered...

Example applications

- Program evolution.
- Evolving decision trees (data mining).
- Planning.
- Find the function (function learning)
- www.genetic-programming.org for some more examples

Questions to think about

1. Discuss how you would use a GP in terms of:
   - Tree representation
   - Grammar
   - Mutation
   - Crossover
   - Fitness

   To solve the following problems:
   - To find the expression (possibly polynomial) for a given graph.
   - To evolve a decision tree for some given dataset.
   - To alphabetically sort elements in some given list (hint: Binary Search Tree).
2. Propose a grammar (including terminal and non-terminal sets) to be used to evolve Boolean expressions.

3. Would it be a good idea to use GP to evolve a game tree?

EC for Data Mining:

GA and Rule Extraction:

- Representation:
- Michigan approach:
  - Individual = 1 rule.
  - Fixed length chromosomes.
  - Rule sets? Run multiple times...
- Pittsburg approach:
  - Set of rules for each individual.
  - Variable length individuals.
- Binary Strings:
- Length of string per feature = the number of values that the feature can take on.

```
Att1 = {blue,red,green}
Att2 = {yes,no}
Class = {male,female}

Att1 Att2 Class
101  10   0
```

Rule:

```
if(Att1 = blue or green)
   AND (Att2 = yes)
then
    class=male
```

- What about attributes with all 1/0?
- What about more than 2 classes?
- Mutation?
- Crossover?
- Fitness function?
- Other operations:
  - Generalizing crossover:
    - OR between crossover points.
    - Generalizes rules.
    - Overfitting?
    - Copy features...
  - Specializing crossover:
    - AND between crossover points.
    - Prevents underfitting
    - Removes feature tests.
- What about different data types?

Clustering:

- Representation 1:

  - Number of patterns = length of chromosome. Each gene corresponds to 1 pattern. Assign cluster labels to each gene.
  - No crossover, does not make sense.
  - Mutation?
  - Fitness function?
    * Find center of cluster (how)?
    * Euclidean distance of all associated patterns from cluster center.
    * Average over all clusters...
- Representation 2:
  - Chromosome = cluster centroids...
  - Know number of clusters in advance = fixed length individuals.
  - Evolve number of clusters = variable length individuals

Feature Selection:

- Each gene is one feature.
- Takes on the value 1 (use feature) or 0 (omit feature)
- Mutation?
- No crossover.
- Fitness? Determined by data mining algorithm.

GP:

- Evolve:
  - Decision Trees: Leaf nodes contain class labels.
  - Regression Trees: Leaf nodes are continuous values.
  - Model Trees: Leaf nodes contain models (usually expression trees/functions)
- What about using GP for hierarchical clustering?

References:

AP Engelbrecht, Computational Intelligence: An Introduction, Second Edition, Wiley & Sons, 2007

I De Falc, A Iazzeta, E Tarantino, AD Cioppa, An Evolutionary System for Automatic Explicit Rule Extraction, CEC2000.

CB Congdon, ClassiïňĄcation of Epidemiological Data: A Comparison of Genetic Algorithm and Deceision Tree Approaches, CEC2000.

JJ Liu, JT-Y Kwok, An Extended Genetic Rule Induction Algorithm, CEC2000.

WM Spears, DF Gordon, Adaptive Strategy Selection for Concept Learning.

KA De Jong, WM Spears, DF Gordon, Using Genetic Algorihtms for Concept Learning.