

31YB: Biologically Inspired Computing

Lectures 13/14: (15 Nov & 29 Nov)
Evolutionary Computation:
Genetic Algorithms

Dr. Amir Hussain

Reading

- ☞ **Goldberg, D.** "Genetic Algorithms in Search, Optimization and Machine Learning", Addison Wesley, 1989

- ☞ Further Reading: Holland, J. "Adaptation in Natural and Artificial Systems", MIT Press, 1992

- ☞ GA Archives
<http://www.aic.nrl.navy.mil:80/galist/>

Two Perspectives:
Optimisation
& Biological/Evolutionary

- **(I) Optimisation Perspective:**
- these systems try to optimise something under a set of constraints
 - often need to find an optimal solution to a problem
 - e.g. find max $f(x)$ over some domain of x ,
 - produce a timetable for university so that lectures do not clash,.. etc.
 - for simple problems gradient descent techniques (like DR) work well
 - but, for difficult e.g. problems with discontinuities, they fail
 - additionally, random search too fails when input space is high dimensional

• (II) Biological/Evolutionary Perspective

- these systems are inspired by the Darwinian evolutionary view, and Mendelian genetics
 - ‘Darwinian selection’: One can view suitability of living creatures to their evolutionary niche as successful optimisation
 - that is, creatures ‘are selected’ because those that fit their niche live long enough to breed
 - thus, those with characteristics which make them fit are replicated in the next generation (direct contrast with Lamarckian concepts)
- Basic idea of Evolutionary Computing/Genetic Algorithms is a mixture of above form of Darwinian selection, and Mendelian genetics (which governs the expression of genes - see later)

A General Optimisation Strategy:

1. generate a population
2. evaluate each member of the population
3. generate a new population based on those which had a “good” evaluation
4. Go back to 2 until the criteria we seek to fulfil are fulfilled

Above is not a new idea for optimisation (God had it in mind a long time ago)! Muhlenbein traces it back to the early 1960’s

Historical Background

- Until 1990, there were two schools studying this area, under different names:
 - European: Evolution Strategy
 - US: Genetic Algorithm
- They even emphasise different aspects, but are essentially studying the same things.
- Lets look at both:

Evolution Strategy (ES) [NOT Examinable]

- S1. Create an initial population, x_i $i=1,\dots,M$
(M =population size)
 - S2. Compute the *fitness* of each x_i , $F(x_i)$ $i=1,\dots,M$
 - S3. Select the N best individuals, $N < M$
 - S4. Create M/N "offspring" of these N individuals
 - S5. If not finished, go back to S2
- The new "generation" in S4 above, is created by small variations (like *mutations*) on the N selected best individuals chosen in 3.
 - In biological terms, ES model natural evolution by *asexual* reproduction with mutation & selection

Genetic Algorithm

- For GA which are search algorithms modeling *sexual* reproduction, need to consider genetic representations.
- Any population member will have a no. of parameters, and these are generated (are expression of) a set of genes. In living creatures, these are A/C/T/G.
- In GA, could be any finite set - but usually $\{0,1\}$
- In simplest case, the genetic representation is just a bitstring of length n , the *chromosome*. The position of the strings are called *loci* of the chromosome. The variable at a locus is called a *gene*, its value *allele*.

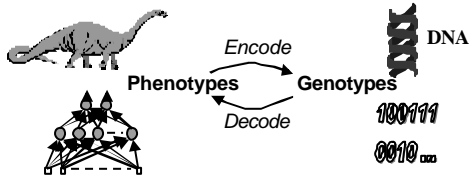
GA: Analogies with Biology

Genotype (set of chromosomes) → Structures and strings (usually bit strings)
define a

Phenotype (organism/individual formed with a certain **fitness**) → A solution (a set of parameter values)

Representation

- Encode the problem's parameters as a string



- Reproduction and mutation operate on the genotype
- Fitness evaluation operates on the phenotype (survival)

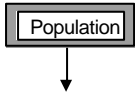
Fitness function

- Choose the fitness function f that we want to maximise through evolution
- f determines which members of the population are likely to survive to reproduce

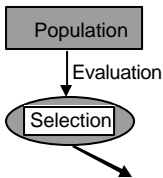
Genetic Algorithm

- S1: Define a genetic representation of problem
- S2: Create an initial population with genes chosen (randomly),
 $P(t=0) = \{ x_1^0, \dots, x_M^0 \}$ where x_i^0 has some sequence of genes
- S2: Compute the fitness of $P(t)$ ($=F(x_1^t), \dots, F(x_M^t)$) and also compute the mean (average fitness): $\mu = \text{sum}_i (F(x_i^t))/M$. Assign normalized fitness value, $F(x_i^t)/\mu$ to each individual.
- S3: Assign each x_i^t a probability $p(x_i^t)$ proportional to its normalized fitness. Using this distribution, select M vectors from $P(t)$ using these probabilities. These will be the selected parents of the next generation.
- S4: Pair all selected parents (from S3) at random (giving $\rightarrow M/2$ pairs). From each pair generate two new "children" using genetic operators. Apply crossover with a certain probability to each pair & other genetic operators such as mutation etc. forming a new population (for next time step $P(t+1)$)
- S5: Set $t=t+1$, If not finished, return to S2

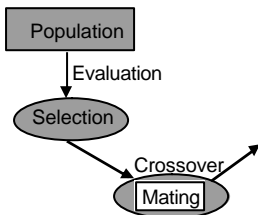
Genetic algorithm: Diagrammatically



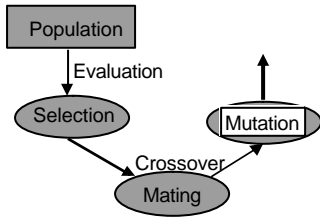
Genetic algorithm



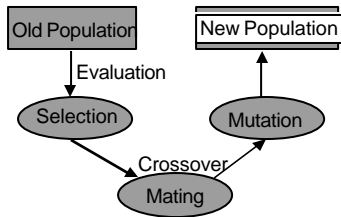
Genetic algorithm



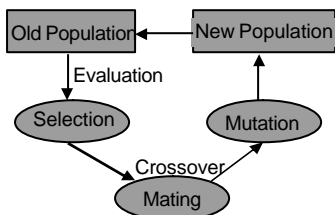
Genetic algorithm



Genetic algorithm



Genetic algorithm

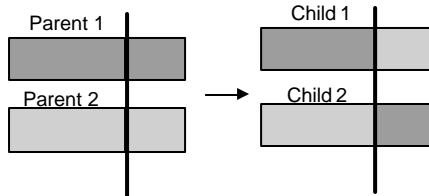


Mating/Crossover

- We want the children of parents to have some characteristics of each parent.
- Crossover is used to ensure that each child has genetic material from both parents
- Single-point crossover: genetic material of child is made up of a string from 1 parent followed by rest of string from 2nd parent:
- parent 1: $(x_i^1): g_1, g_2, \dots, g_r, g_{r+1}, \dots, g_k$
- parent 2: $(x_j^2): h_1, h_2, \dots, h_r, h_{r+1}, \dots, h_k$
- Children: $g_1, g_2, \dots, g_r, h_{r+1}, \dots, h_k$
 $h_1, h_2, \dots, h_r, g_{r+1}, \dots, g_k$

Mating / Crossover

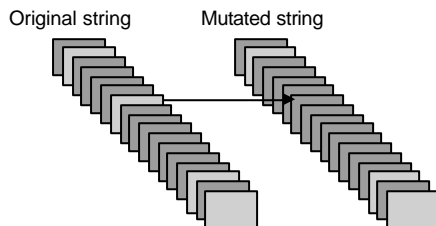
Graphically:



Note: Ordering of information on genome is important, why?

Mutation

Each element of genetic string can be altered with small probability, p



A simple example [NOT Examinable]

Problem: Search the integers $x=0..31$ for the x that maximises the function $f(x) = x^2$

☞ **Fitness function:** $f(x)$

☞ **Representation:**

Encode x as a 5-bit string in the obvious way

0 \rightarrow 00000, 1 \rightarrow 00001, 2 \rightarrow 00010, ...

31 \rightarrow 11111

(adapted from Goldberg, 1989)

Initial random population

#	
1	01101
2	11000
3	01000
4	10011

Fitness evaluation

#	x	$f(x)$
1	01101	13 169
2	11000	24 576
3	01000	8 64
4	10011	19 361
		Sum = 1170
		Avg. = 293

Selection

#	x	f(x)	P(select)	Range
1	01101	13 169	0.14	$0.00 \leq r < 0.14$
2	11000	24 576	0.49	$0.14 \leq r < 0.63$
3	01000	8 64	0.06	$0.63 \leq r < 0.69$
4	10011	19 361	0.31	$0.69 \leq r < 1.00$

Sum = 1170

Avg. = 293

$$P(\text{select}) = \frac{f}{\sum f}$$

Random numbers:	0.31	0.82	0.09	0.59
Mating Pool #:	2	4	1	2

Crossover

Mating Pool	New Popn.
11 000	11011
10 011	10000
0110 1	01100
1100 1	11001

Fitness evaluation

Mating Pool	New Popn.	x	f(x)
11 000	11011	27	729
10 011	10000	16	256
0110 1	01100	12	144
1100 1	11001	25	625
		Sum =	1754
		Avg. =	439 (formerly 293)

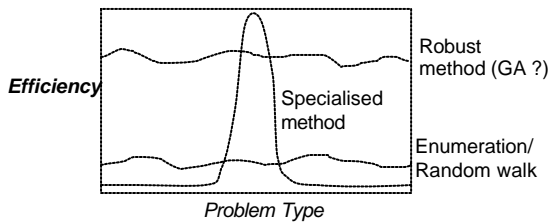
⇒ So this generation is fitter than previous generation !

What about Mutation ?

[NOT Examinable]

- Probability of mutating a bit:
Typically $P(\text{mutation}) = 0.001$
- Population size = 4
- Expected mutations = $4 \times 5 \times 0.001 = 0.02$
- Hence, no mutations occurred in the simulated generation

Are GA's useful ?



- GA's are quite robust: always reasonably efficient
- GA's are good for 'tricky' error functions with many local minima (see also simulated annealing)
- For a *specific* problem, a better algorithm usually exists

Problems with GA

- 'Crowding'
 - A fit individual quickly reproduces. Copies of it and similar individuals become very common.
 - Diversity is reduced & evolution becomes slower
- Ways of avoiding crowding:
 - Alter the selection function
 - tournament selection, rank selection
 - Fitness sharing
 - Fitness reduced by presence of other similar individuals
 - Restrict recombination (*biologically inspired*)
 - Allow only similar individuals to recombine - encourages clusters or 'sub-species'
 - Spatially distribute individuals and allow only nearby individuals to recombine.

Problems with GA

- ☞ 'Premature Convergence'
 - ☞ all population converge to non-optimal solution
 - ☞ most likely to occur due to some population scoring much more on fitness than others
 - ☞ solution: scale fitness function so that more of population gets permitted to be a parent
- ☞ 'WYTIWYG'
 - ☞ what you test is what you get
 - ☞ more of a warning that GA system may not do what you want
 - ☞ instead, will tend to do exactly what will lead to it achieving a high fitness score
 - ☞ hence, care needs to be taken in designing fitness fn.

GA's v. Gradient Descent

- ☞ Gradient Descent (e.g. Back-Prop)
 - ☞ move smoothly from one hypothesis to a new hypothesis that is very similar
 - ☞ Can be 'trapped' in local minima
- ☞ Genetic Algorithm
 - ☞ search can move abruptly, replacing a parent by a radically different offspring
 - ☞ less likely to be 'trapped' by local minima
 - ☞ parallel search using a population of hypotheses

GA's and Neural Nets

- Case 1: Use GA to learn weights
- Case 2: GA learns weights and network structure
- Case 3: GA evolves structure/connectivity, NN (e.g. BP) trains each individual for evaluation

- ☞ e.g. in financial forecasting
 - ☞ Given an initial *general* (feedforward+recurrent) NN structure and a training set: GAs have been recently used to evolve/determine optimal (feedforward or recurrent) NN predictor models

Summary

- Optimisation: Evolutionary Computation & Genetic algorithms
- Biological analogy
- GA - Example, Efficiency, Problems
- GA & Neural Networks

- **Note: NO Lectures next week, as I am away**
- **This Thursday, 18 Nov: (final) Tutorial 3**
- **Remaining Lectures (14 & 15): week starting 29 Nov (Last topic to be covered: Neuromorphic Systems)**
