Genetic Algorithms: An introductory Overview

References: An introduction to Genetic Algorithms by M. Mitchell

Genetic Algorithms + Data Structures = Evolution programs by Z. Michalewicz

Biological evolution I

- Long molecules known as DNA (**D**eoxyribo**n**ucleic **A**cid) are the physical carrier of genetic information that define us.
- Fragments of DNA, known as **genes** produce chemicals called *proteins*.
  - **Gene** = basic functional block of inheritance (encoding and directing the synthesis of a protein)
- The proteins activate or suppress other genes in other cells, they cause cells to multiply, move, change, extrude substances, grow and even die.
- Genes are a little like parameters. They control our development. The different values a gene can take are called ***alleles*.**
- And **evolution** creates the genes and specifies the alleles.

Biological evolution II

- Our genes define how we develop from a single cell into the **complex organisms** that we are.
- A **cell** consists of a single nucleus containing **chromosomes** which are large chains of **genes**
  - Chromosomes = a single, very long molecule of DNA
- **Genome** is the complete collection of genetic material (all chromosomes together)
- **Genotype** is the particular set of genes contained in a genome.
- **Phenotype** is the manifested characteristics of the individual; determined by the genotype

Biological evolution III

- Charles Darwin's 1859 "The Origin of Species" proposed **evolution through natural selection**
- According to **Universal Darwinism**, the following things are needed in order for evolution to occur:
  - **Reproduction with inheritance**
    - Individuals make copies of themselves
    - Copies should resemble their parents
      - organisms pass traits to offspring
  - **Variation**
    - Ensure that copies are not identical to parents
      - **mutations**, **crossover** produces individuals with different traits
  - **Selection**
    - need method to ensure that some individuals make more copies of themselves than others.
    - **fittest individuals** (favorable traits) have more offspring than unfit individuals, and population therefore has those traits

over time, changes will cause new species that can specialize for particular environments

Evolutionary Computation

- Study of computational systems that use ideas inspired from natural evolution
  - **Survival of the fittest**

Main Branches of EC

- Genetic algorithms (GA)
- Genetic programming (GP)
- Evolution strategies (ES)
  - Evolving evolution
- Evolutionary Programming (EP)
  - Simulation of adaptive behaviour in evolution
  - Emphasizes the development of behavioural models and not genetic models

----

- Evolutionary computation
  - Reproduction
  - Random variation
  - Competition
  - Selection

Other Branches of EC

- **Differential evolution**
  - Similar to GA, differing in the reproduction mechanism used
- **Co-evolution**
  - Initially “dumb” individuals evolve through cooperation or in competition with one another, acquiring the necessary characteristics to survive
- **Cultural evolution**
  - Models the evolution of culture of a population
- Modelling of other aspects of natural evolution exists

- Evolutionary computation
  - Reproduction
  - Random variation
  - Competition
  - Selection

Other population based techniques: Swarm Intelligence

- Motivated by the study of colonies, or swarms of social organisms.
- **Ant colony optimization**
  - Modeling of social ants behaviour
- **Particle swarm Optimization**
  - A global optimization approach modeled on the social behaviour of bird flocks or fish schooling
  - Individuals (particles) grouped a swarms
- **Bees Algorithms**
  - Foraging behaviour of swarms of bees (2005)

Other algorithm exist..

Inventors

- GAs, ES and EP were all independently 'discovered' by researchers in the 1960s. GP was created in the early 1990s, as a specialised type of GA.
- John Holland "Adaptation in Natural and Artificial Systems", University of Michigan Press (1975)- Genetic Algorithms.
- Lawrence Fogel, M. Evans, M. Walsh "Artificial Intelligence through Simulated Evolution", Wiley, 1966 - Evolutionary programming.
- Ingo Rechenburg, 1965 - Evolutionary Strategies.
- Marco Dorigo, 1992, ACO.
- Kennedy, 1995, PSO.

What are Genetic Algorithms ?

- **Genetic algorithms (GAs):** a search technique that incorporates a simulation of evolution as a search heuristic when finding a good solution
  - akin to Darwinians theory of natural selection
  - recent years have seen explosion of interest in genetic algorithm research and applications
  - a practical, dynamic technique that applies to many problem domains
  - can "evolve" unique, inventive solutions
  - can search potentially large spaces.
- Related areas:
  - **genetic programming**: applying GA towards the evolving of programs that solve desired problems
  - **artificial life**: simulations of "virtual" living organisms
    - doesn't necessarily use GA, but commonly does

Comparison of Natural and GA Terminology

Natural	Genetic Algorithm
chromosome	string
gene	feature, character or detector
allele	feature value
locus	string position
genotype	structure, or population
phenotype	parameter set, alternative solution, a decoded structure

Genetic algorithms

- Formally introduced in the US in the 70s by John Holland
  - Early names: J. Holland, K. DeJong, D. Goldberg
- Holland's original GA is usually referred to as the simple genetic algorithm (SGA)
- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

Main components of SGA reproduction cycle

- Select parents for the mating pool (equal to population size)
- Apply crossover with probability p_c, otherwise, copy parents
- For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
- Replace the whole population with the resulting offspring
  - Generational population model

A General Simple GA

i=0	set generation number to zero
initpopulation P(0)	initialise usually random population of individuals
evaluate P(0)	evaluate fitness of all initial individuals of population

while (not done) do test for termination criterion (time,fitness, etc.)

begin

i = i + 1	increase the generation number
select P(i) from P(i-1)	select a sub-population for offspring reproduction
recombine P(i)	recombine the genes of selected parents
mutate P(i)	perturb the mated population stochastically
evaluate P(i)	evaluate its new fitness

end

Figure 1 Basic general GA

Genetic Algorithms

Before we can apply Genetic Algorithm to a problem, we need to answer:

- How is an individual represented
- What is the fitness function?
- How are individuals selected?
- How do individuals reproduce?

Genetic Algorithms

- To use a GA, the first-step is to identify and define the characteristics of the problem domain that you need to search
- This information encoded together defines an individual referred to as genetic string or chromosome (genome).
  - –the chromosome is all you need to uniquely identify an individual
  - chromosome represents a solution to your problem

The genetic algorithm then creates a population of solutions

•finally, need a way to compare individuals (i.e., rank chromosomes)

- --> Fitness measure
- –a type of heuristic

Representation of individuals

- Remember that each individual must represent a complete solution (or partial solution) to the problem you are trying to solve by GAs.
- Recall that Holland worked primarily with strings of bits, where a chromosome consists of genes i.e., binary chromosome.
- But we can use other representations such as arrays, trees, lists or integers, floating points or any other objects.
- However, remember that you will need to define genetic operators (mutation, crossover etc) for any representation that one decides on.

Initial Population

- Initialization sets the beginning population of individuals from which future generations are produced
- Concerns:
  - size of initial population
    - empirically determined for a given problem
  - genetic **diversity** of initial population
  - a common problem resulting from the lack of diversity is the **premature convergence** on non-optimal solution

Simple Vs Steady-state

Population creation: two most commonly used; Simple/Steady-state

simple: it is a generational algorithm in which entire population is replaced at each generation

steady-state: only a few individuals are replaced at each 'generation'

examples of replacement schemes

- **replace worst**
- **replace most similar (crowding)**

Other population schemes exists e.g

- **Parallel population**
- **co-evolution**

Evaluation: ranking by Fitness

- **Evaluation ranks the individuals** by some fitness measure that corresponds with the individual solutions
- **For example, given an individual** i:
  - classification: (correct(i))2
  - TSP: distance (i)
  - walking animation: subjective rating

Selection scheme

- **determines which individuals survive and possibly mate and reproduce in the next generation**
- **selection depends on the evaluation function**
  - if too dependent then a non optimal solution maybe found
  - if not dependent enough then may not converge at all to a solution
  - selection method that picks only best individual => population converges quickly (to a possibly local optima)
- **Nature doesn't eliminate all "unfit" genes. They may usually become recessive for a long period of time and then may mutate to something useful**
  - Hence, selector should be biased towards better individuals but should also pick some that aren't quite as good (with hopes of retaining some good genetic material in them).

Selection Techniques

examples of selections schemes

- Fitness-Proportionate Selection
- **rank selection**
- **tournament selection (select K individuals, and keep best for reproduction)------we will focus on this**
- **roulette wheel selection (probabilistic selection based on fitness)**
- **other probabilistic selection**

Fitness-Proportionate Selection

- Concerns include
  - One highly fit member can rapidly take over if rest of population is much less fit: **Premature Convergence**
  - At the end of runs when fitnesses are similar, lose selection pressure

Rank - Based Selection

- Attempt to remove problems of FPS by basing selection probabilities on relative rather than absolute fitness
- Rank population according to fitness and then base selection probabilities on rank where fittest has rank m and worst rank 1
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Tournament Selection

- Idea:
  - Pick *k* members at random then select the best of these
  - Repeat to select more individuals

Tournament Selection 2

- Probability of selecting *i* will depend on:
  - Rank of *i*
  - Size of sample *k*
    - higher *k* increases selection pressure
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
  - Whether fittest contestant always wins (deterministic) or this happens with probability *p*

Roulette Wheel Selection (Read for knowledge)

- adapted from “GeneticAlgorithms + Data Structures = Evolution Programs, 3rd ed., Z.Michalwicz, p.34
- A. Fitness evaluation
  - Calculate fitness value vi for each individual i: v(i) (i=1,...,pop_size)
    - if smaller score is better, and 0 is perfect, then go on
    - else if higher is better, set v(i) = MAX - v(i) , where MAX is the maximum best score
  - Adjusted score: set adj_v(i) = 1 / (1 + v(i))
    - this sets best to 1, and worst scores approach 0
    - also exaggerates small differences in raw scores
  - **Find total adjusted fitness for pop.**:

$$F = \sum_{i=1}^{pop\_size} adj\_V(i)$$

  - Calculate probability for each indiv: p(i) = adj_v(i) / F
  - Calculate cumulative probability for each indiv in pop.:

Roulette wheel selection

- After step A, the cumulative probabilities q(i) from 1 to pop size are fractions that range from about 0 to 1 for last individual q(pop_size)
  - analogous to a Roulette wheel, in which the whole wheel is of circumference 1, and each indiv. has space in proportion to its fitness

B. Selection

  - Generate random number R between 0 and 1
  - if $R < q(1)$ then select individual 1
  - else select individual i if: $q(i-1) < R <= q(i)$

Premature convergence

- **If the population consists of similar individuals, it reduces likelihood of finding new solutions**
  - **for example, crossover operator and selection method may drive GA to create population of individuals that are similar**

De Jong-style crowding using replacement schemes: when creating new individuals, replace individuals in the population that are most similar to them

Goldberg style Fitness scaling: delete scores of similar individuals to reduce chances of similar individuals being selected for mating

Genetic Operators

(1) Crossover: provides a method of combining two candidates from the population to create new candidates

- Swaps pieces of genetic material between two individuals; represents mating

-Typically crossover defined such that two individuals (the parents) combine to produce two more individuals (children). But one can define asexual or single-child crossover as well.

(2) Mutation: changing gene value(s)

–lets offspring evolve in new directions; otherwise, population traits may become fixed ; introduces a certain amount of randomness to the search.

(3) Replication: copy an individual without alteration

Genetic Operators

- In terms of search, effects of crossover and mutation are problem dependent:
  - some problems with a single global maxima perform well with incremental mutation
  - crossover and mutation can let search carry on both at current local maxima, as well as other undiscovered maxima

Genetic operators: Crossover

- Selecting a genetic operator:
  - if Pc is the probability of using crossover, then if R is a random number between 0 and 1,   then do crossover if R < Pc

Crossover Operators

- 1-point, n-point crossover
- Uniform order crossover (UOX)
- Order (OX) crossover
- Partially mapped (PMX)
- Cycle (CX) crossover

…..many variations exist

Crossover Operators

1-point crossover

P1:	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
P2:	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**

C1:	**1**	**0**	**0**	**1**	**1**	**0**	<u>1</u>	<u>0</u>	<u>1</u>	<u>0</u>
C2:	**0**	**1**	**1**	**0**	**1**	**1**	<u>0</u>	<u>1</u>	<u>0</u>	<u>0</u>

crossover

N-point crossover:

generalization of 1-point crossover

- **e.g., Two-point crossover**

P1: 1 0 0 1 1 0 0 1 0 0

P2: 0 1 1 0 1 1 1 0 1 0

C1: 1 0 0 0 1 1 1 **1 0 0**

C2: 0 1 1 1 1 0 0 **0 1 0**

Techniques exist for permutation representations

Learning illegal structures

Consider the TSP where an individual represents a potential solution. The standard crossover operator can produce illegal children:

Parent A: Thorold Catharines Hamilton Oakville Toronto
Parent B: Hamilton Oakville Toronto Catharines Thorold

Child AB: Thorold Catharines Hamilton Catharines Thorold
Child BA: Hamilton OakVille Toronto Oakville Toronto

2 possible solutions:

- Define special genetic operators that only produce syntactically and semantically **legal/feasible** hypotheses (a.ka. solutions).
- ensure that the fitness function returns extremely low fitness values to illegal hypotheses **(penalty functions)**

Uniform-Order crossover (UOX)

P1:	6	**2**	**1**	4	**5**	7	**3**
Mask :	0	1	1	0	1	0	1
P2:	4	**3**	**7**	2	**1**	6	**5**
C1:	**4**	2	1	**7**	5	**6**	3
C2:	**6**	3	7	**2**	1	**4**	5

Order crossover (OX)

- Main idea: preserve relative order that elements occur
  - e.g for the TSP, chooses a subsequence of a tour from one parent and preserves the relative order of cities from the other parent.

OX example

- Copy randomly selected set from first parent

p1: 1 2 3 4 5 6 7 8 9 c1: * * * 4 5 6 7 * *

p2: 9 3 7 8 2 6 5 1 4 c2: * * * 8 2 6 5 * *

- Copy rest from second parent in order 1,9,3,8,2

C1: 3 8 2 4 5 6 7 1 9

C2: ?

OX example (2)

- Copy randomly selected set from first parent

p1: 1 2 3 4 5 6 7 8 9   c1: * * * 4 5 6 7 * *

P2:4 5 2 1 8 7 6 9 3

- Copy rest from second parent in order 9,3,2, 1, 8

C1: 2 1 8 4 5 6 7 9 3

Cycle crossover (CX)

Basic idea:

Each element comes from one parent together with its position.
e.g for TSP, each city (and its position) comes from one of the parents

Example: Cycle crossover

- Step 1: identify cycle

p1: 1 2 3 4 5 6 7 8 9

p2: 9 3 7 8 2 6 5 1 4

c1: 1 * * 4 * * * 8 9

- Step 2: Fill the remaining cities from the other parent

c1: 1 3 7 4 2 6 5 8 9

Example:Partially mapped crossover (PMX)

- Step 1: identify arbitrary cut points
  p1: 1 2 3 4 5 6 7 8 9
  p2:4 5 2 1 8 7 6 9 3

- Step 2: copy & swap
  c1: * * * 1 8 7 6 * *   Note: 1<->4, 8<->5, 7<->6, 6<->7
  c2: * * * 4 5 6 7 * *

- Step 3:fill cities where no conflict
  c1: * 2 3 1 8 7 6 * 9
  c2: * * 2 4 5 6 7 9 3

- Step 4: Fill the remaining cities
  c1: 4 2 3 1 8 7 6 5 9
  c2: 1 8 2 4 5 6 7 9 3

Example:Partially mapped crossover (PMX)

- Step 1: identify arbitrary cut points

  p1: 1 2 3 4 5 6 7 8 9

  p2: 9 3 7 8 2 6 5 1 4

- Step 2: copy & swap

  c1: * * * 8 2 6 5 * *

  c2: * * * 4 5 6 7 * *

- Step 3:fill cities where no conflict

  c1: 1 * 3 8 2 6 5 * 9

  c2: 9 3 * 4 5 6 7 1 *

- Step 4: Fill the remaining cities

NOTE: pay attention to this example

Mutation

Alteration is used to produce new individuals

- **Mutation: various strategies e.g., for TSP**
  - **Inversion**
  - **Insertion**, select a city & insert it in random place
  - **Displacement** – selects a subtour and inserts it in a random place
  - **Reciprocal exchange** – swaps two cities
  - **Scramble mutation -** Pick a subset of genes at random

    Randomly rearrange the alleles in those positions

Mutation

- The mutation operator introduces random variations, allowing hypotheses to jump to different parts of the search space.
- What happens if the mutation rate is too low?
- What happens if the mutation rate is too high?
- A common strategy is to use a high mutation rate when learning begins but to decrease the mutation rate as learning progresses. **(Adaptive mutation)**

Crossover Vs mutation

Exploration: How to discover promising areas in the search space, i.e. gaining information on the problem

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

Crossover is explorative: makes a big jump to an area somewhere “in between” two (parent) areas

Mutation is exploitative: creates random small diversions, thus staying near (within the area of) the parent

A balance between Exploration and Exploitation is necessary. Too much exploration results in a pure random search whereas too much exploitation results in a pure local search.

Parameter Control

A GA/EA has many strategy parameters, e.g.

- mutation operator and mutation rate
- crossover operator and crossover rate
- selection mechanism and selective pressure (e.g. tournament size)
- population size

Good parameter values facilitate good performance, but how do we find good parameter values ?

Setting GA parameters

- parameters (selected according to problem)
  - how many individuals (chromosomes) will be in population
    - **too few: soon all chromosomes will have same traits & little crossover effect; too many: computation time expensive**
  - mutation rate
    - **too low: slow changes; too much: desired traits are not retained**
  - how are individuals selected for mating? crossover points?
  - what are the probabilities of operators are used?
  - Should a chromosome appear more than once in a population?
  - fitness criteria
- genetic algorithm can be computationally expensive = > need to keep bounds on GA parameters and GA analysis

Why do GA's work?

- GA offers a means of searching a broad search space
- different features of problem are represented in search space by DNA representation
  - parallel nature: often many solutions to a problem
  - different "good" characteristics represented by particular gene settings
  - some combinations of these genes are better than others
- evolution creates new gene combinations --> new areas of search space to try
- Key to success: individuals that are fitter are more likely to be retained and mated; poorer individuals are more likely discarded
- global search technique, unlike other search techniques that use heuristics to prune the search space

Genetic Algorithms as search

GAs differ from more normal optimization and search procedures in 4 ways:

- GAs work with a coding of the parameter set, not the parameters themselves.
- GAs search from a population of points, not a single point
- GAs use payoff (objective function) information, not derivatives or other auxiliary knowledge.
- GAs use probabilistic transition rules, not deterministic rules.

Summary: GAs

- Easy to apply to a wide range of problems
  - optimization like TSP, VRP
  - inductive concept learning
  - scheduling
  - Layout
  - Evolving art
  - Game playing
- network design etc
- The results can be very good on some problems, and rather poor on others
- GA can be very slow if only mutation applied, crossover makes the algorithm significantly faster

Summary:GAs

-GA better than gradient methods if search space has many local optima

-various data representation, one algorithm

-no gradients or fancy math, however, designing an objective function can be difficult

-computationally expensive (how so, do we care?)

-can be easily parallelized

- can be easily customized (question is, is it GA anymore?)

Artificial Life

- Artificial life (ALife): simulate desired aspects of biological organisms on computers
  - AI's focus on intelligence is just one aspect of organism behavior
  - others: sight, movement (robotics), hearing, morphology, adaptation to environment, behavior, ...
- Practical use of ALife: model realistic theories of vision, robotics
  - traditional vision, robotics theories are constrained by hardware limitations
  - hence theories of vision, movement are necessarily primitive
  - virtual life permits theories of unlimited complexity to be used: physical, real-time constraints are removed

ALife

- Another use: simulate complex behaviours for use in graphics and animation
  - manual reproduction of realistic movement, animal behaviour is too complicated and time-consuming
  - let systems evolve themselves, and/or react according to their virtual definitions
- ALife is a testbed for many areas of AI research:
  - GA to simulate population evolution
  - robotics
  - vision
  - machine learning

Summary

- Research on the latest applications of evolutionary computation & AI----lots of them
- Readings
  - Handbooks of Genetic Algorithms
  - Genetic Algorithms + Data structures = Evolutionary Programs (3ed Z. Michalewicz)
  - Koza:vol 1

Applications of GAs/EC

- Discussed in class as examples only based on vehicle routing and job shop scheduling—not included in exam: If interested in my research papers on this, email me.