Genetic Algorithms and Genetic Programming Lecture 1

Gillian Hayes

22nd September 2006

Admin

Lecturer: Gillian Hayes, IPAB, School of Informatics

Email: gmh@inf.ed.ac.uk

Office: JCMB room 2107C, ext. 513440

Course Activities:

- Lectures: Tuesday 12:10 (JCMB LTB), Friday 12:10 (Daniel Rutherford Building LT1),
- Tutorials: Mon 10:00 (JCMB 6324), 15:00 (DHT FRN), Wed 13:00 (JCMB 6324), Fri 15:00 (JCMB 4310 and AT M3). Weeks 3–10

Admin

- Reading: From supplied course notes and set book (*An Introduction to Genetic Algorithms* by Melanie Mitchell, MIT Press 1998, £20.85 on Amazon, also available on MIT CogNet)
  See `http://www.lib.ed.ac.uk/resbysub/info/ebooks.shtml`
- Assignments: a single assignment worth 25% of the course mark, to be handed in at the start of Week 11.
- Exam: worth 75% of the course mark, taken at the end of Semester 2.

Syllabus

Part 1: Introduction

- Genetic Algorithms: biological inspiration

Part 2: Genetic Algorithms (GAs)

- The canonical genetic algorithm
- The schema theorem and building block hypothesis
- Formal analysis of genetic algorithms
- Methodology for genetic algorithms
- Designing real genetic algorithms

continued....

Syllabus

Part 3: Optimisation Problems

- Solving optimisation problems
- Swarm intelligence: ant colony optimisation (ACO)
- Adding local search: hybrid GAs and hybrid ACO
- Other methods: simulated annealing, tabu search

Part 4: Evolving Programs and Intelligent Agents

- Evolving programs: genetic programming
- Evolving controllers: neural networks and robots
- Evolving intelligence: agents that play games
- Evolving intelligence: programs that can plan

Recommended Books

- [set book] Mitchell, Melanie (1998). An Introduction to Genetic Algorithms. MIT Press.
  – *A very good introduction with a scientific flavour.*
- Michalewicz, Zbigniew (1996). Genetic Algorithms + Data Structures + Evolution Programs. Springer.
  – *An alternative to Mitchell, with more emphasis on problems from Computer Science.*
- Banzhaf, Wolfgang, et al. (1998). Genetic Programming: An Introduction. Morgan Kaufmann.
  – *An excellent introduction to Genetic Programming.*
- Bonabeau, Dorigo and Theraulaz (1999). Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press.
  – *Introduces Ant Colony Optimisation.*

A Simple Example

Consider the Tutor Allocation Problem

Jobs: Job1, Job2, . . . Job$m$

Job$_i$ is a single tutorial to be taught:

– subject, e.g. Java, GAGP

– slot, e.g. Thu 2–3

– place, e.g. A10, 5 Forrest Hill

– knowledge, skills required, e.g. strong at Java, some knowledge of AI techniques useful

One tutor teaches each tutorial.
We have a pool of tutors to choose from:

Tutors: TutorA, TutorB, TutorC . . .

Properties of tutors:

– knowledge/skills

– cost per hour

– time preferences

– room preferences

– optimal number of jobs

Solutions

A **solution** is an allocation of tutors to jobs:

Job:	1	2	3	4	5	6	7	8	9	10
Tutor:	A	B	C	D	E	F	G	H	I	J

Each job-tutor pairing can be given a **score**, based on how good the knowledge/skills match is:

Tutor A: some C++, strong at AI
Job 1: strong Java, some AI useful
– a reasonable match, though not perfect

A function $f(\text{job}, \text{tutor})$ calculates a numerical score for us for any pairing.

The **whole** solution can be given a score, based on:

– scores for job-tutor pairings

– total cost of solution

– hard constraints

– tutor preferences

The total score will be calculated from the scores for the individual parts.

The **problem** is to find the solution with the **best** score.

Possible Methods

Use **exhaustive** search?

– 5 tutors, 10 jobs $= 9.8 \times 10^6$ solutions

– 10 tutors, 20 jobs $= 1.0 \times 10^{20}$ solutions

– 15 tutors, 30 jobs $= 1.92 \times 10^{35}$ solutions

– . . .

Use **greedy** search?

Job 1 – find best tutor

Job 2 – find best tutor to give best combined score with the choice for Job 1

Job 3 – etc.

Almost certain to be **sub-optimal** since it commits to choices too early.

Use **Hillclimbing Local** Search?

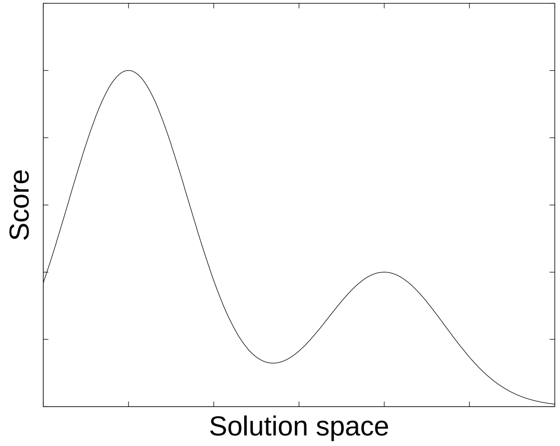
Solution$_i$: A B E A B B [D] C E D

Suppose [D] is the worst scorer. Try A, B, C, E

Solution$_{i+1}$: A B E A B B [A] C E D

Continue until no improvement possible.
Prone to local maxima.

Local Maxima



Genetic Algorithms

How about trying a biologically inspired solution based on genetics?

1. Generate a **population** of solutions:

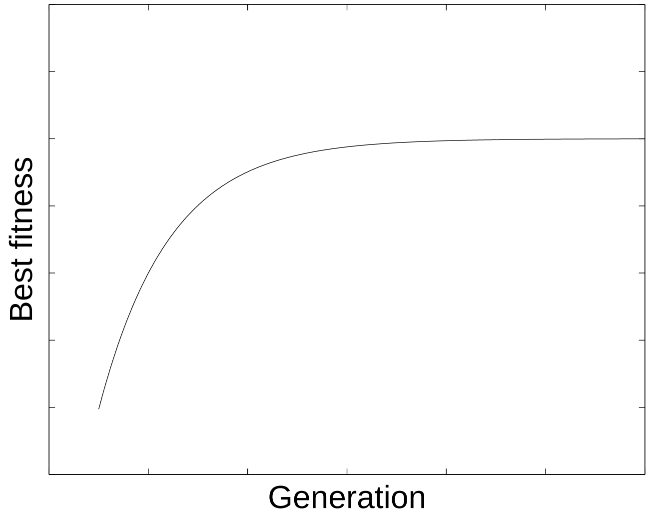
Generation$_i$:

Solution1:	A	B	C	A	B	C	D	D	E	E
Solution2:	B	C	E	A	B	D	E	C	A	D
⋮										
Solution$n$:	E	D	A	C	C	D	A	D	B	A

Note that the ordering of jobs is **implicit** in this representation.

2. Give each solution a score, called a **fitness**.

3. Create a **new generation** of solutions by:
(a) selecting fit solutions
(b) breeding new solutions from old ones and add to generation$_{i+1}$.
4. When a sufficiently good solution has been found, stop.



"Breeding"

How does breeding work?

1. Reproduction:

Copy solution$_i$ unchanged into the next generation.

2. Crossover:

Parent1: ABCABCDDEE

⇒

Parent2: BAEDCADCBA

Exchange of genetic material to form children.

3. Mutation:

(a) change one value in a solution to a random new value:

AEBCABDDCE ⇒

(b) swap two values:

AEBCABDDCE ⇒

(c) lots of others!

Mutation is usually done after reproduction/crossover, with low probability (1%).

How Well Does This Work?

- small problems: optimal solutions
- larger problems: optimal or near optimal given enough time
- anytime behaviour
- runs on parallel machines
- adding constraints is very easy
- used in a multitude of real applications
- wide applicability to problems in search, optimisation, machine learning, automatic programming, A-life, . . .

Next lecture: Introduction to Genetics