University of BRISTOL

The Genetic Algorithm

Seth Bullock

A Standard GA

- Genetic algorithms share the same basic form:
  - Repeatedly: assess, breed, mutate a population of solutions
- But different types of GA vary in many different ways:
  - Selection scheme, genetic operators, population structure, etc.
  - ‘Steady state’ GA vs. ‘generational’ GA
  - + Many special tricks of various kinds...
- Here we present a very simple version
  - (and mention some of the most frequent variants)

Terminology

- *Population* – the set of individual solutions that the GA is acting on
- *Individual* – a member of the population
- *Genotype* – a string of symbols from some alphabet that encodes a particular solution (sometimes: *Genome* or *Chromosome*)
- *Phenotype* – the actual solution encoded by a genotype
- *Genotype-Phenotype Mapping* – analogous to development
- *Genes* (also *Loci*) – chunks of genome, each taking a value: “*Allele*”
- *Fitness* – the value or quality of an individual solution phenotype
    - *Fitness Function* – returns the fitness of a solution

University of BRISTOL

Terminology

- *Selection* – choosing which current individuals reproduce
- *Parents* – individuals selected to reproduce
- *Offspring* – the new individuals that result from reproduction
- *Crossover* – the recombination of alleles from multiple parents
- *Mutation* – replacing offspring alleles with random alternatives
- *Fitness Landscape* – an 'evolutionary search space' organising all possible solutions according to the neighbourhood relationships that result from the GA's Genotype Structure & Genetic Operators, with landscape 'altitude' set by each solution's Fitness.

Simple Genetic Algorithm

- A Simple GA:

```
initialise() => population
repeat:
    evaluate(population)
    select(population) => parents
    breed(parents) => offspring
    offspring => population
    if timed_out or good_enough:
        return best(population)
```

- Compare with a Hillclimber

```
initialise() => individual
repeat:
    evaluate(neighbours)
    select(neighbours) => chosen

    chosen => individual
    if timed_out or stuck:
        return(individual)
```

Generate Initial Population and Evaluate

- We start with a population of individual random genotypes
  - Each is a random string of symbols from the genetic ‘alphabet’
  - A typical (classic) GA alphabet: {0,1} – binary genotypes
- To evaluate an individual we use the fitness function:
  - How well does the robot specified by the genotype behave?
  - How good is the wing design specified by the genotype?
  - How good is the exam timetable specified by the genotype?
  - How successful is the chess strategy specified by the genotype?

University of BRISTOL

Genotype-Phenotype Mapping

- The genotype is just a way to write down a solution:
  - #1: [Thin, Cheese, NoTomato, NoOlives, NoAnchovy]
  - #2: [Deep, Cheese, Tomato, Olives, Anchovy]
- To allocate fitness to genotypes, we must decode them.
- We must build ('develop') the *phenotypes* they encode:
- The *genotype-phenotype mapping* characterises this 'developmental' encoding relationship…

University of BRISTOL

Selection

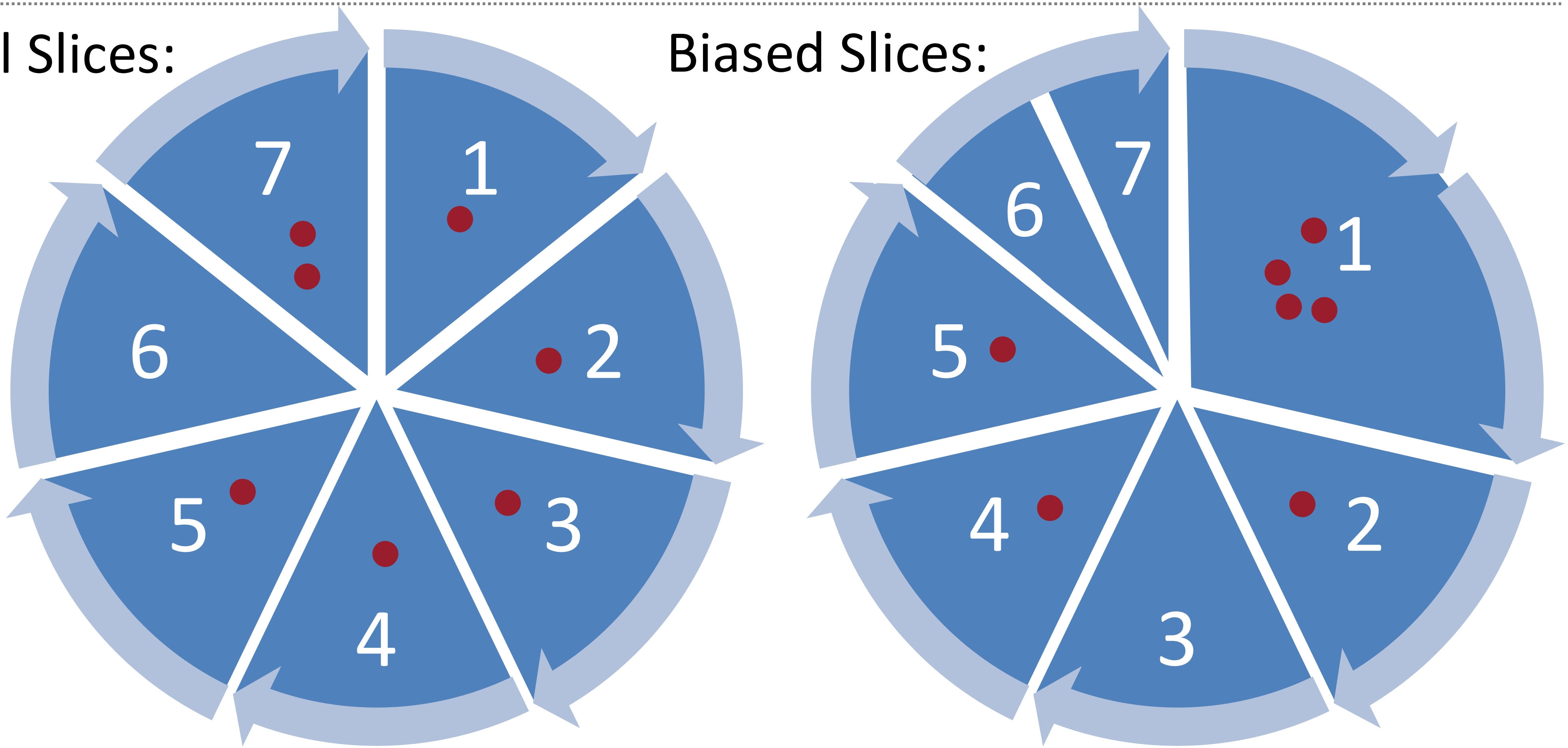
- A key idea is that we should tend to select the *fitter* individuals from the current population to become parents
- We can pick parents with a chance proportional to fitness:
  - $p_i \propto f_i / S$ where p_i is the chance that we pick individual i
  - S is the sum of fitnesses in the current population: $S = \sum_{j=1}^{N} f_j$
  - N is the number of individuals in the current population
- One way of doing this is using ‘roulette wheel’ selection:
  - Spin a wheel where: probability of landing in slice $i = p_i$

University of BRISTOL

Roulette Wheel Selection

Equal Slices:

Biased Slices:



Roulette Wheel Selection

Run a roulette wheel that selects element i with probability p_i

```
start with a population: N individuals, each with a fitness score

sum(fitnesses(population)) => S
rand(0,S) => roll

0 => running_total
loop i from 1 to N:
     running_total + fitness(i) => running_total
     if running_total >= roll:
          return(i)
```

Consider:

- *Is it worth sorting the population by fitness before using the roulette wheel to pick N parents?*

Rank-Proportionate Selection

- Alternatively we could pick each parent with a chance proportional to the *rank* of their fitness in the population
- One way of doing this is using ‘tournament’ selection:
  - Sample *k* (unique) individuals at random from the population
  - Pick a winner (e.g., the one with highest fitness) to be a parent
- To find *N* parents, run *N* independent tournaments.

Consider:
- *How is selecting on fitness* rank *different from selecting on* fitness?

Consider:
- *What is happening if k=1?*
- *What is happening is k=N?*

Reproduction

- Having selected parents, we reproduce them:
  - we make a copy of their genes and put it into the next generation of the population
- Genetic operators are applied during reproduction:
  - Mutation: each offspring gene is replaced by a random allele with probability *m*
  - Crossover: the offspring genotype inherits genes from multiple parents
- Mutation is almost always used. One mutation per offspring is a typical rate.
- Crossover is sometimes not used at all ('asexual' reproduction).
- …or is only applied to some offspring (e.g., it occurs with probability *c*)
- Elitism: the fittest current individual is copied into next generation perfectly

Consider:

- *What if all parents were just copied perfectly?*

Types of Crossover

- One-Point: pick a point along the genotype, $k \in [0, L]$
  - The offspring inherits the first *k* genes from their mother's genotype and the remainder from their father's genotype
- Two-Point: pick two points along the genotype, $j, k \in [0, L]$
  - The offspring inherits genes *j* thru *k* from their mother's genotype and the remainder from their father's genotype
- Uniform: merge the genes from both parents at random
  - Each of the offspring's genes, is inherited from their mother with probability 0.5, else from their father.

University of BRISTOL

Crossover Schemes

Consider:

- *How do different crossover schemes disrupt genotypes in different ways?*

Mum:	1	0	0	0	0	0	1	0	0
Dad:	0	1	1	1	1	1	0	0	0
One-Point Crossover Kid:	1	0	0	0	0	0	0	0	0
Two-Point Crossover Kid:	0	1	0	0	0	1	0	0	0
Uniform Crossover Kid:	1	1	1	0	0	0	0	0	0

Generational GAs vs Steady State GAs

- Typically the GA population size, *N*, is held constant...
- Two common ways to ensure this:
  - Generational Reproduction: ...we continue selecting and reproducing parents until we have a *N* new offspring. Then we discard the old population and replace it with the *N* new individuals.
  - Steady-State Reproduction: …as soon as we have generated *one* new offspring, we pick a member of the current population (either at random or biased toward the unfit) and kill it, replacing it with the new offspring.
- There are many different wrinkles on these basic schemes:
    - E.g., only ever let a new offspring displace a less fit individual

University of BRISTOL

Stopping

- Two simple stopping conditions:
  1. We've found the perfect solution, i.e., a genotype that achieves maximum fitness
  2. We've run out of time, i.e., our generation counter has reached max_generation
- A more subtle stopping condition:
  3. We think the GA has run out of steam and things are as good as they are going to get
- Checking for conditions 1 and 2 is easy. How do we check for condition 3?
- Fitness scores have *stagnated* – no improvement for many generations:
  - Current "best" genotype is (very) old…
  - The population is strongly converged…

Consider:

- *Might the population still be exploring new solutions?*
- *How could we know for sure?*

Genome Structure

- Remember: Each genotype is made of symbols from the genetic ‘alphabet’
  - {0,1} – bit strings; {A, C, G, T} – nucleic acids; {A, B, C, …Z} sentences
  - Or maybe parameters of some kind: {<integers>}, {<floats>}
  - Note that choice of alphabet has implications for mutation and crossover
  - Mutation = a bit flip; or a random letter; or a random perturbation
- And there could be constraints on some genes:
  - `pizza_radius` must be *positive* and *less than* `oven_width`
  - `num_robot_legs` must be *even*;
    - a 3-bit encoding of `exam_day` leaves `101`, `110`, and `111` unused?
    - illegal genotypes must be discarded… causing search *biases*

Simple Genetic Algorithm (Recap)

- The Simple GA:

```
initialise() => population
repeat:
    evaluate(population)
    select(population) => parents
    breed(parents) => offspring
    offspring => population
    if timed_out or good_enough:
        return best(population)
```

- Simple Example

```
# generate N random bit strings
# employ a generational GA
#  fit. func. assigns fi∈[0,1]
#  tournament selection, use k=3
#  bitflip mutation & 1-pt xover
#  elitism on: always copy best
#  gen > max_gen or fitness = 1
```

Example Questions

- Name two different kinds of cross-over operator. *[1 mark]*
- Uniform crossover of two L-bit binary genotypes can result in how many different offspring genotypes? *[2 marks]*
- What is the difference between roulette-wheel selection and tournament selection? *[4 marks]*
- Define a good fitness function for evolving a university time-table schedule. Explain your answer. *[8 marks]*

Thank you!

Fitness Landscapes

Seth Bullock