Evolutionary Algorithms II

Fitness Landscapes, Demes, Genotype-to-Phenotype Maps

CS369 // Computational Science

Matthew Egbert
m.egbert@auckland.ac.nz

Last lecture

Basics of evolutionary algorithms

- random mutation
- directed selection

Terminology

- genotype / phenotype
- fitness

Q: What are some problems and advantages associated with having a low mutation rate?

Q: What are some problems and advantages associated with having a high mutation rate?

Q: When using creep mutation of real valued genes, why a mutation amount from a Gaussian distribution better than selecting from a flat distribution?

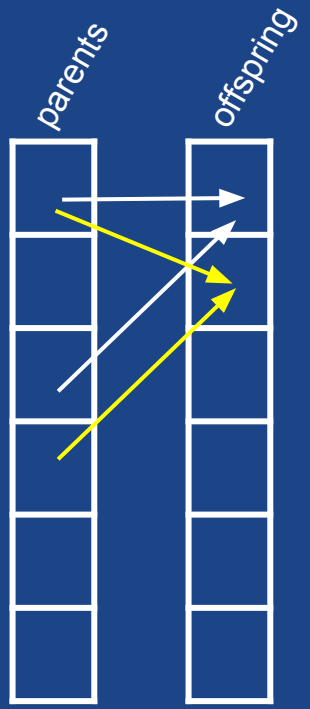
Q: Why when mutating a bitstring is it better to mutate each bit with p=1/N, rather than selecting one bit at random to flip?

Q: What exactly is the disadvantage of "wrapping" gene values to keep them within bounds?

Q: What happens when the evaluation of fitness is a bit random (e.g. different wind conditions in the paper-airplane example)?

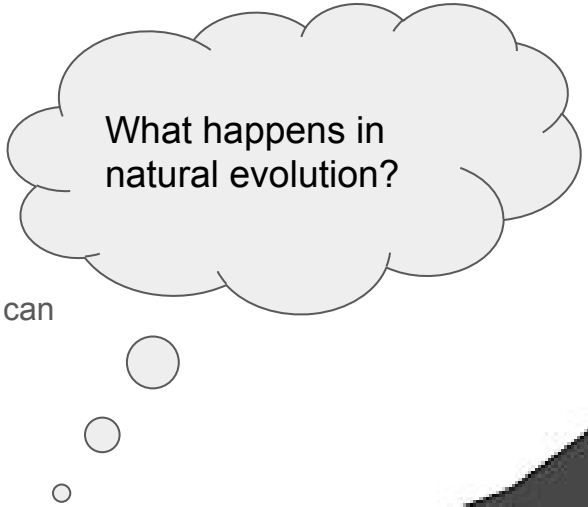
Genotype to Phenotype Map

GA pseudocode




```
POP_SIZE = 30
N_GENES  = 10
N_GENERATIONS = 100

parents[POP_SIZE,N_GENES];
offspring[POP_SIZE,N_GENES];

## INITIALIZE THE POPULATION
for i in range(0,POP_SIZE) :
    for j in range(0,N_GENES) :
        parents[i,j] = appropriate_random_value()

## EVOLVE. Each time around the outer loop is one "generation"
for generation_i in range(0,N_GENERATIONS) :

    ## evaluate the fitness of every individual
    fitnesses = [eval_fitness(indiv) for indiv in parents]

    for offspring_i in range(0,N_GENES) :
        ## randomly pick 2 individuals from parents
        ## selecting preferentially for the fitter ones
        parent_a, parent_b = biased_random_selection_of_two_parents()

        ## recombine to make 1 offspring, and mutate the offspring
        offspring[offspring_i] = mutate( recombine(parent_a,parent_b) )

    ## overwrite the parents with the offspring, (throwing
    ## away the previous generation in the process)
    parents[:] = offspring[:]
```

Evaluating Fitness



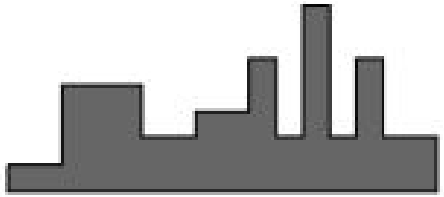
1. Given a genotype, generate a phenotype that can be evaluated for fitness.
2. Evaluate fitness.

What is the best way to set up the encoding of the phenotype known as the **genotype-to-phenotype map**?

We have seen in the demo that evolutionary algorithms can be thought of as "climbing hills" in the fitness landscape.

Some landscapes are more easily climbed than others. smooth hill vs. a "manhattan skyline"

A rule of thumb: a small change in the genotype should create a small change in fitness



Example G→P Mapping

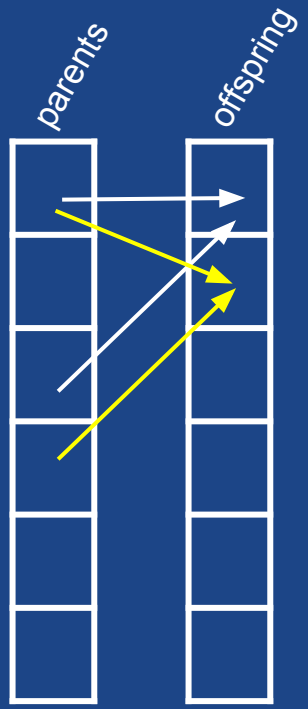
Let's imagine that the phenotype is an integer between 0 and 7, and the genotype is three bits...

To mutate from phenotype 3→4, you would need three mutations!

Binary	Phenotype
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Selection

GA pseudocode




```
POP_SIZE = 30
N_GENES  = 10
N_GENERATIONS = 100

parents[POP_SIZE,N_GENES];
offspring[POP_SIZE,N_GENES];

## INITIALIZE THE POPULATION
for i in range(0,POP_SIZE) :
    for j in range(0,N_GENES) :
        parents[i,j] = appropriate_random_value()

## EVOLVE. Each time around the outer loop is one "generation"
for generation_i in range(0,N_GENERATIONS) :

    ## evaluate the fitness of every individual
    fitnesses = [eval_fitness(indiv) for indiv in parents]

    for offspring_i in range(0,N_GENES) :
        ## randomly pick 2 individuals from parents
        ## selecting preferentially for the fitter ones
        parent_a, parent_b = biased_random_selection_of_two_parents()

        ## recombine to make 1 offspring, and mutate the offspring
        offspring[offspring_i] = mutate( recombine(parent_a,parent_b) )

    ## overwrite the parents with the offspring, (throwing
    ## away the previous generation in the process)
    parents[:] = offspring[:]
```

Selection Methods

Truncation selection uses the top X% of the population as parents.

1. Remove the poorest performing Y% of the population.
2. From the remaining population, select parents at random, and generate enough offspring to replace all of the removed individuals.

Tournament selection is just as simple, but maybe it can better take advantage of any information the population has about a local gradient.

1. Select two individuals at random. The fitter of the two gets to be a parent.
2. Repeat N-times to pick N different parents.

The random selection of individuals for the tournament may sometimes miss information about the local gradient...**fitness proportionate selection** is one attempt to take advantage the entire population's "knowledge about the gradient"

Q: What if fitnesses are e.g. 1020, 1010, 1025, 1017?

A: then there will be very little selection pressure to improve (roughly even chances of reproduction for everyone)

Rank selection ignores absolute differences in scores, focusing purely on the rank of the individual in the population. Even below average in population has a chance of reproduction.

Fitness proportionate selection

If the fitnesses of an example small population were 2, 5, 3, 7 and 4, then you would select parents with the probability 2/21, 5/21, 3/21, 7/21, 4/21 (as 2+5+3+7+4 = 21).

Rank selection

If the fitnesses of an example small population were 2, 5, 3, 7, 4 then you might select parents with probabilities 0/10, 3/10, 1/10, 4/10, 2/10 (as the sum of all of the possible ranks, 0+3+1+4+2 = 10)

Elitism

Elitism is an optional property of GAs, where the best-performing individual found so far is guaranteed to remain in the population (until there exists a more fit individual.)

Sometimes elitism is implicit in the GA. Sometimes it is an explicitly added feature.

Assuming that fitness evaluations are a deterministic function of the genotype...

Q: Which of the selection methods just presented implicitly include elitism?

Q: ...and how does this change if fitness evaluations include some stochasticity?

Convergence and Demes

Convergence

Q: How does the shape of the fitness landscape affect population convergence?

A: A local peak causes the population to converge (this is sometimes used to signal the end of the evolution), and a flat or "neutral" area causes the population to become more diverse. When the variety of the population is too low, evolution can be slow, and it can more easily get stuck in local optima.

Q: What other factors influence population convergence?

DEMO

Demes

For GAs to work well, you want some variety, but not total randomness…

...not so small you get stuck in **local optima.**

...not so big that the search becomes a **random search**. (i.e. that you are not responding to patterns / gradients in the fitness landscape).

… not so small that the algorithm takes forever.

Demes are a way to increase / maintain some variety in the population.

		9	10	**11**	12	13			

A "deme" (in evolutionary biology) refers to an **isolated sub-population**.

Generally individuals within a deme will interbreed with other individuals within that deme, but occasionally there will be some intermixing between demes.

There are a variety of ways one can include demes in GAs. Perhaps the simplest is to store your individuals as a list, and to only let individuals reproduce with individuals that are nearby on that list +/- 1 or +/- 2.

Generational vs. Steady State GAs

The pseudocode we looked at above is a **generational** GA, as it replaces one entire generation with the generation of its offspring.

Some natural systems work like this, but other have co-existent generations / no-clear distinction between generations.

We can create **steady-state** GAs that have coexisting (and potentially competing) offspring and parents.

Let's look at an example...

Microbial GA

```
## INITIALIZE THE POPULATION
population = np.random.rand(POP_SIZE,N_GENES)

## EVOLVE. Each time around the outer loop is one "tournament"
for tournament_i in range(N_TOURNAMENTS) :
    ## parent selection (with DEMES)
    parent_a_index = get_random_index()
    parent_b_index = ( parent_a_index + random.choice([-1,1]) ) % POP_SIZE

    # evaluate both parents for fitness and identify a winner and a loser
    a_fitness = evaluate(population[parent_a_index])
    b_fitness = evaluate(population[parent_b_index])

    if a_fitness > b_fitness :
        winner_i = parent_a_index
        loser_i  = parent_b_index
    else :
        winner_i = parent_b_index
        loser_i  = parent_a_index

    ## overwrite some of the loser's genome with some of the winner's genome
    ## an alternative form of recombination
    for g_i in range(N_GENES) :
        if np.random.rand() < 0.7 :
            population[loser_i,g_i] = population[winner_i,g_i]
        ## mutate the loser
        mutate(loser)
```

DEMO

Optional Exercises

Available on cs369 website...

CS369 Computational Science
Evolutionary Algorithms
Optional Exercise
Matthew Egbert 2017

Hands-on experience can really help you to remember what you have learned. For this reason I recommend that you gain some experience working with genetic algorithms by writing one of your own. This exercise is optional and not graded, but if you have questions along the way, feel free to email me or swing by my office.

Task #1. Write a genetic algorithm that maximises the sum of a set of N real-valued genes that are capped between (say) 0 and 1. You could start with the a minimal version of the microbial genetic algorithm and extend it with demes, or modify it to use alternative forms of sexual recombination rather than the microbial mechanism of overwriting the loser's genome with part of the parent's genome. This exercise is also a good place to investigate the different methods for limiting the allowed values of genes (clamping, wrapping and bouncing). In particular, what are the effects of wrapping?

Task #2. A different task is to evolve a solution to the following card problem.

You have 10 cards numbered from 1 to 10. You have to choose a way of dividing them into 2 piles, so that the cards in Pile_0 **sum** to a number as close as possible to 36, and the product of the remaining cards in Pile_1 is a number as close as possible to 360.

Why would no-body ever actually use a GA to solve this particular problem in real life?

Can you conduct any experiments that give you insight into the fitness landscape of this problem?

Thank you!