CS369 Computational Science
Evolutionary Algorithms II
Matthew Egbert 2017

A First Evolutionary Algorithm

This is the same pseudocode as last lecture.

```
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[:]
```

The example code above had the magic function `eval_fitness(indiv)`, which takes an individual's genes as an argument, generates a phenotype from that individual, and then evaluates the fitness of that phenotype. Sometimes the phenotype is essentially the same thing as the phenotype, but more frequently, the genotype encodes the phenotype and there are various ways that the encoding can be accomplished. This encoding, the **genotype to phenotype map** heavily influences the shape of the fitness landscape, and the shape of the fitness landscape heavily influences the likelihood of your GA finding a good solution. A good rule of thumb is that where possible, *a small change in the genotype should create a small change in fitness*. When we view evolution as a "hill-climbing" search of parameter space, we can see that GAs take advantage of regularities in the search space.

Selection

In the pseudocode above, we had a "magic" function, `biased_random_selection_of_two_parents()`. This function determines which individuals get to reproduce. This **selection** process can be accomplished in a variety of different ways.

- **Tournament selection** is perhaps the simplest method. Two individuals are selected at random. The fitter of the two gets to be a parent (this can be done N-times to pick N parents for).
- **Truncation selection** uses the top X% of the population as parents.
- **Fitness proportionate selection.** Say the fitnesses of an example small population are 2, 5, 3, 7 and 4, then select parents with the probability 2/21, 5/21, 3/21, 7/21, 4/21 (as 2+5+3+7+4 = 21).
    - **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.

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.)

Q: Which of the above inherently accomplish elitism (i.e. without any additional code to do so) (a) when fitness is a deterministic function of the genotype and (b) when the evaluation of fitness includes stochastic terms?

Population Convergence and Demes

Another dynamic that we can observe in the simulation is **population convergence**, where the variety of the population goes down. **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?

It is often the case that the population quickly becomes fairly genetically converged, and thereafter tends to search just one small part of the search space.

Multiple populations may search multiple corners, and a common approach when using GAs is to run multiple independent evolutionary runs (each is hopefully searching a different part of the genotype space).

But maybe it would be better if these different populations could interbreed a little bit, to share the beneficial mutations when they have found them, etc.

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. One way to do so is to use **tournament selection** to evaluate two parents, and to replace the less fit parent with the offspring of the two. The population size remains constant, but over time everyone is replaced -- *or are they?*

The following is pseudocode for the Microbial Genetic Algorithm, an algorithm that was designed with simplicity in mind. It incorporates **demes, tournament selection,** and **elitism**, but the generation of offspring is a bit different than we have seen above -- in that it is inspired by the transferral of genetic material between living organisms as observed in some bacteria.

```
POP_SIZE = 30
N_GENES  = 8
N_TOURNAMENTS = 500

## 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)
```

Further reading

David Goldberg 1989 "Genetic Algorithms in Search, Optimization and Machine Learning" Addison Wesley

Melanie Mitchell and Stephanie Forrest "Genetic Algorithms and Artificial Life". Artificial Life v1 no3 pp 267-289, 1994.

Melanie Mitchell "An Intro to GAs" MIT Press 1998

Z Michalewicz "GAs + Data Structures = Evolution Programs" Springer Verlag 1996

Harvey I. (2011) The Microbial Genetic Algorithm. In: Kampis G., Karsai I., Szathmáry E. (eds) Advances in Artificial Life. Darwin Meets von Neumann. ECAL 2009. Lecture Notes in Computer Science, vol 5778. Springer, Berlin, Heidelberg