Massachusetts Institute of Technology

6.034 Artificial Intelligence

Solutions to Assignment # $\boxed{\textbf{11}}$

Goals

- You learn how quickly or slowly independent, incremental, random changes would "evolve" towards a goal.
- You learn about simulated evolution, using Genetic Algorithms (GAs), and how these differ from independent, random, incremental evolution: better changes are accumulated through time by use of a "fitness" function.
- You learn about the strengths and limitations of GAs.

Problem 1 Basic string probability calculations

Starting with one $l = 11$ string where each letter is chosen randomly (with equal probability) from the $c = 26$ alphabet:

Part A

How many possible such strings are there?

Answer: $c^l = 26^{11} \simeq 3.7 \times 10^{15}$

Part B

What is the probability that at least one of the 11 characters correctly matches the corresponding character in the target string?

Answer: The probability of an incorrect character at any one position in the string is $\frac{25}{26}$. Let C be the number of correct characters in a string.
Then $P(C \geq 1) = 1 - P(C = 0) = 1 - (25/26)^{11} \simeq 0.35$

Part C

What is the probability that exactly one of the 11 characters is correct? *(Remember: each character is generated independently and with equal likelihood).*

Answer: This requires one character correct and $l - 1 = 10$ characters wrong, for any one of 11 characters, so the answer is:
$P(C = 1) = 11 \times (\frac{25}{26}^{10} \cdot \frac{1}{26}) \simeq 0.28578$

Problem 2 Introducing mutation in copying

The original string is copied ten times, but errors creep in. For each letter copied, there is a 99% chance that everything goes OK, but 1% of the copy operations "mutate" the letter, changing it to any one of the other characters in the alphabet, with equal probability.

Part A

If none of the characters in the original string are correct, what is the probability that at least one character does match in at least one of the ten copies?

Answer: Let N be the number of copies of a totally incorrect string that have at least one correct character. Then $P(N \geq 1) = 1 - P(N = 0)$, and for 10 copies $P(N = 0) = P(I = 0)^{10}$ where I is the number of correct characters in one copy of the original string. Now at each of the l incorrect characters in the original there is a $\frac{99}{100}$ chance of it being copied without a mutation, and a $\frac{1}{100}$ probability of mutation. If a mutation occurs, then there is a $\frac{24}{25}$ probability of the original incorrect character mutating to one of the 24 remaining incorrect characters. So, for each location on the string there is a probability $i = \frac{99}{100} + \frac{1}{100} \cdot \frac{24}{25} = 0.9996$ that the copy's character at that location will still be incorrect. Thus $P(I = 0) = i^l = 0.9996^{11} \simeq 0.995609$ and hence $P(N \geq 1) = 1 - P(I = 0)^{10} \simeq 0.043054$.

Problem 3 Fitness

Part A

Approximately how many generations will pass before a string with a nonzero fitness value is created, if the original string had a fitness of zero? *(Hint: in these questions, you can estimate the number of generations before an event X occurs by calculating the reciprocal of the probability of X occurring on any one generation).*

Answer: From the hint, and the answer to the previous question, if $G_{F>0}$ is the number of generations it takes to generate an individual string with non-zero fitness, starting from an individual with zero fitness, then $G_{F>0} = 1/P(N \geq 1) \simeq 23.22$; but the number of generations must be an integer, so the number of generations will be 23 or 24.

Part B

If you waited around long enough, a string with a fitness value of 10 will be created. What is the probability that at least one of the ten "children" of such a string is correct at *all* 11 sites?

Answer: Let N_l be the number of children with all $l = 11$ characters correct. Then $P(N_l \geq 1) = 1 - P(N_l = 0)$. Now, for one child of an individual with 10 of the $l = 11$ sites correct to get *all* 11 correct, the 1 incorrect site must first be selected for mutation (there's a $\frac{1}{100}$ chance of this happening) and then must be mutated to the correct character (there's a $\frac{1}{25}$ chance of that happening). Meanwhile, the remaining $l - 1 = 10$ sites must not be mutated at all: at each site there's a $\frac{99}{100}$ chance of that happening. Therefore, the probability c_l of a child with l correct characters is $c_l = (\frac{99}{100})^{10} \cdot \frac{1}{100} \cdot \frac{1}{25} \simeq 3.6175 \times 10^{-4}$. Thus the probability that a child is *not* perfect is 1 minus this, or $1 - c_l$. Hence $P(N_l = 0) = (1 - c_l)^{10}$ and so $P(N_l \geq 1) = 1 - (1 - c_l)^{10} \simeq 0.00361$.

Part C

Roughly how many generations will it take to progress to a copy with all 11 characters correct, starting from a string with a fitness of 10?

Answer: The number of generations $G_{F=10\rightarrow 11}$ can be estimated using:
$G_{F=10\rightarrow 11} = 1/(1 - (1 - c_l)^{10}) \simeq 276.882$. So it will take roughly 277 or 278 generations.

Problem 4 Running the system: what happens?

Part A Convergence

What do you notice about the speed with which the system approaches the correct string "intelligence"?

Answer: Convergence follows the stochastic model suggested above; because we accumulate "good" mutations, the rate is faster at first than at the end, where only a few positions are left to change, and there are more chances that we might mutate away from a fitness peak; thus the convergence roughly follows a sigmoid curve.

Part B Mutation rate

Change the mutation rate, `mrate`, in `ga-multi`, from 0.09 to 0.01. Does it take more or less time to find an optimally fit string? Why or why not?

Answer: It appears that decreasing the mutation rate slows the search by about an order of magnitude: with `mrate` changed to 0.01, from 0.1, with a run of 5000 generations we can get up to a fitness level of roughly 9. This is because there's a far smaller chance to change any given letter position at a single step. It will generally take 10,000 generations to reach the fitness peak of 11 (if one doesn't run out of Scheme memory first). Note that if we were talking about the generation time in terms of humans, about 100 generations takes us back to the founding of Rome, and 10,000 generations to before the origins of modern *H. sapiens*. So this mutation rate is too slow. However, a mutation rate of 0.1 is known to be biologically implausible. So, we must do something else to boost the rate of evolutionary change. That takes us to...

Part C Sex

The current code shows only *asexual* reproduction, where copies of the best individual are made with mutation. Describe in a few sentences how you would alter the code so as to implement *sexual* reproduction, with recombination or crossover.

Answer: This is actually in the code, in the function `reproduce-sexually` — sexual recombination is introduced by using the gene-copy function, which pastes together pieces of two existing genomes (whole genes).

```
(define (reproduce-sexually pop mrate target)
  (let*
      ;; sort pop so fittest is first
      ((newpop (list->vector (sort (vector->list pop) fitter?)))
       ;; make kid a mutated crossover of the best two genomes
       (n (random 2))
       (kid (mutate-gene (gene-copy pop n (- 1 n)
                                    (random (string-length (genome pop 0))))
                         mrate)))
    (vector-set! newpop (- (vector-length newpop) 1)
                 (cons kid (eval-fit kid target)))
    newpop))

(define reproduce reproduce-sexually)
```

Evolution with sexual reproduction is much faster in this model; for example, for a "typical" set of 10 runs, the mean number of generations to reach an optimal fitness (for, say 10 runs) was 489.5, with a standard deviation of 22.5 (This excludes runs that do not reach the fitness optimum.) 9 out of 10 runs hit the optimal fitness peak. In comparison, with asexual reproduction, about 1 in 20 runs will hit the fitness peak with 1000 generations.

Problem 5 The limits of genetic algorithms

Part A

In what single major way does the evolutionary model described here *differ* from biological evolution (Hint: one does not need to be a biologist to know the answer.)

Answer: While there are many unrealistic biological assumptions in this model (such as the high mutation rate discussed above), there is one very, very large and important difference between this model considered as an optimization, goal-finding method and evolution by natural selection: in the model, we assume we *know* the target string "intelligent", and how far away we are from it (as recorded by the fitness measure). In general, biological evolution does not "know" where it is headed — it cannot have anticipatory global knowledge of some goal, but depends entirely on local gradient ascent.

(Another way that this method differs from "real" evolution is that there is an identity mapping between the set of genes, or *genotype*, and how those genes get expressed in the resulting organism's *phenotype* (literally, the "form that shows"), or what the organism's actual properties are. In general, we don't know what the possibilities for this mapping are, but it's not an identity mapping.)

Part B

Suppose our evolutionary system starts with the word string "do" and tries to evolve it into the string "by" with this additional constraint: the intermediate strings that are evolved must all be "viable" — they must all be legitimate two-letter English words. Given this constraint, is the evolution of "do" into "be" possible (given enough time)? Similarly, given this viability constraint, is it possible for *any* legitimate two-letter English word to evolve into *any other* two-letter English word? Why or why not?

Note that this corresponds to a kind of "viability" constraint on the evolutionary process (the organisms in between generations must all "live").

What implications does this have for Genetic Algorithms in general?

Answer: It's possible to go from a Consonant-Vowel pair like *at* to, say, *in*, because there is an intermediate form, *an*. Similarly, *do* could change to *be* if there were a sequence (a chain) of single-step changes in between. *Do* can change to *be* via the incremental steps *go*, *ho*, *ha*, *la*, *ma*, *my*, *by*, and *be*. For a more biologically relevant example, one might consider the amino acids, and see which can evolve to which others by single "letter" changes in the genetic code.

More generally, since Vowel-vowel 2-letter forms aren't words, one can't evolve from a Vowel-Consonant form to a Consonant-Vowel form by single-step changes alone: to get from *to* to *in* one would have to pass through a stage with 2 vowels, which is impossible.

(Note: This wasn't meant to reach into deep knowledge about English; if you simply saw the constraint operating in the second part of the question, in the case of vowel-vowel "blocks" to evolving from C-V to V-C, that was sufficient.)

For GAs, this shows that the possibilities of *incremental*, *gradual* change may depend on the domain space. If we are doing an optimization problem using GAs, then the intermediate solutions must also be solutions. This search method might therefore be limited — the density of solutions might have to be high, as suggested in lecture. Or there may have to be some other non-gradual, non-incremental change operating. If we make more realistic biological assumptions about how mutations are gained and lost, then it has been shown recently by Allan Orr in the journal *Evolution*, Sept. 1998, that very small, incremental changes do not work very well in evolution by natural selection, because they will be lost by chance. Intermediate and more macroscopic changes have a much larger chance of becoming fixed in a population.