CPS331 Lecture: Genetic Algorithms last revised October 11, 2018

Objectives:

1. To explain the basic ideas of GA/GP: evolution of a population; fitness, crossover, mutation

Materials:

1. Genetic NIM learner demo
2. Projectable of wall-follower robot problem and a solution
3. Projectable of Nilsson figures 4.2 .. 4.9
4. Mitchell pp. 17-18 and 19-21 to read
5. Projectable of payoff matrix - p. 18
6. Floreano §1.12.1 to read
7. Projectable of bottom two sections of Floreano figure 1.16

I. Introduction

A. Genetic algorithms represent an attempt to imitate the architecture of intelligence present in nature - in this case, the "intelligence" exhibited by a species (not individuals) as it evolves to better fit its niche in the ecosystem. That is, genetic algorithms draw their inspiration from biological evolution.

Some key biological concepts they draw on are:

1. The notion of a population, which is a collection of inter-breeding individuals.

2. Diversity within a population. Though the individuals in a population are similar, they are not identical. Some are more "fit" than others.

3. Selection. Evolution does not operate to improve individuals; rather, it operates to improve populations by increasing the

proportion of individuals exhibiting more fit characteristics over time. This happens because more fit individuals are more likely to reproduce, either because of being better able to survive, or because of other characteristics that increase their likelihood of reproduction (e.g. the fancy plumage of male birds in breeding season).

4. Heredity. An individual's physical characteristic are encoded in its DNA in the form of genes. (Humans, for example, have 20,000-25,000 genes). An individual's genes are copied from the genes of its parents. In the case of sexual reproduction, half the genes of an offspring come from its father and half from its mother.

   Because more fit individuals are more likely to reproduce, the proportion of their genes in the population tends to increase over time.

5. Mutation. Occasionally, a gene will be slightly altered by random processes as it is transmitted from parent to child, Such mutations are often harmful, resulting in a child that is less fit than its parents; but sometimes a mutation is beneficial, makes the child more fit, and is passed on from that child to its offspring.

   Mutation plays an important role in biological evolution, because without it evolution can only rearrange existing genes, but cannot discover new ones.

B. For a problem to be a good candidate for using a genetic algorithm or genetic programming, several things need to be true.

1. The problem can't be "all or nothing" - that is, it must be meaningful to talk about "solutions" which are less than perfect, just as there can be diversity of fitness among individuals in a biological population.

a) This does not preclude the possibility of there being a perfect solution - but it is to say that a solution that is less than perfect must still be a viable solution.

b) Moreover, given a set of proposed solutions, there must be some straightforward way to evaluate their relative fitness, so that it is meaningful to talk about "better" solutions.

2. It must be possible to break a solution up into "genes" - each of which represents part of the solution - which are, at least to some extent, independent of each other.

a) Frequently, genetic algorithms combine two "parent" individuals to produce an "offspring" individual by using crossover . Crossover consists of taking some genes from one "parent" and the remaining genes from the other "parent". (In fact, often the crossover of two parents is used to produce two offspring, with the second offspring having the reverse pairing of genes.)

b) Frequently, genetic algorithms do mutation by randomly selecting a gene of an offspring and changing is value to some other random - though legal - value.

c) Actually, there is another similar approach - called evolutionary computation - which does not entail doing this. (This is actually the approach Fogel used).

3. Some examples of problems which lend themselves to this approach.

a) Development of a spam filter.

(1)This problem could use supervised learning to learn from a database of emails have been characterized as spam or non-spam.

(2)The measure of fitness is how accurately it recognizes spam while not rejecting non-spam

(3)The genes may correspond to the presence or absence of certain words in the email.

b) A two-player game, such as the checkers program which is the basis of Fogel's book.

(1)This problem could use reinforcement learning.

(2)Checkers can be played by individuals at a wide variety of skill levels. The obvious measure of fitness is how well the program plays against players of various ability levels.

(3)A major factor in how well a game player program performs is its static evaluation function. This, in turn, can be decomposed into genes representing the weight assigned to various features being considered.

(4)We will look at a simple example of breaking a solution into genese for a simple version of the game of NIM.

(5)Fogel's book describes a program which used evolutionary computation without breaking up the solution into genes to learn the weights to use for the static evaluation function by playing games against human players under the pseudonym "Blondie24".

c) Control problems (like teaching a robot to walk).

Several years ago, we had a speaker here who discussed work he was doing on using genetic programming to evolve a program to enable a hexapod robot to walk (actually a quite non-trivial task if the terrain is uneven)

(1)A perfect solution would keep the system behaving in the desired way endlessly, but a solution that keeps it behaving the desired way for a long time is still useful. (Even we sometimes fall down while walking!).

(2)Solutions can be compared based on how long they keep the system behaving in the desired way before failing.

(3)The problem can be decomposed into "genes" representing the relationship between various percepts and actions.

d) Optimization problems like traveling salesman.

(1)For large problems, we generally have to accept a good solution, even if it is not possible to find a provably optimal one. (Indeed, what we think to be a good solution may turn out to be the best possible, even if we can't prove that it is!)

(2)But solutions can be compared on the basis of total cost.

(3)The problem can be decomposed into "genes" representing the relative order of visiting cities.

C. Genetic algorithms/programming are an approach to problem solving in which a population of potential solutions is evolved to produce increasingly better solutions to the problem.

1. Genetic algorithms/programming, though inspired by biological evolution, differ from it in one important respect - the existence of a goal (other than simple survival) by which fitness can be measured.

   When evolution has been successful, the most fit individual in the population is taken as being the solution to the problem

2. There are two related concepts: genetic algorithms and genetic programming.

a) Sometimes, the goal is to find a solution to a problem, with fitness being determined by measuring some quality of the solution (e.g. traveling salesman)

b) At other times, the goal is to to evolve a computer program that effectively solves a given problem, in which case, the approach is called genetic programming, and fitness is measured by measuring the quality of the resultant solution (e.g. checkers playing or controlling a hexapod robot).

We will look at examples of both approaches.

3. As we have noted, there is a similar approach called evolutionary computation that evolves a program without breaking the solution up into genes.

D. A genetic algorithm or genetic programming is best suited to problems where a solution is not known ahead of time and cannot be found by more traditional means.

II. Structure of a Genetic Algorithm

A. A GA proceeds by evolving a population of individuals, each of which represents a different possible solution to the problem at hand. (In the case of GP, each individual represents a possible program.)

B. Possible solutions to a given problem are encoded as a sequence of "genes", each of which may be (depending on the problem) a value from a discrete set of possible values, or a real number.

The initial population is constructed by choosing values for each gene at random. It is therefore unlikely that any individual in the population constitutes a good solution to the problem.

C. A fitness function measures the extent to which each individual in the population represents a good solution to the problem. Initially, given random individuals, the fitness function for each individual will be small; but there will be some that are better than others, and the GA will attempt to evolve their good points into the next generation of possible solutions.

That is, the critic in the learning system evaluates the overall fitness of each individual for the given problem.

D. Evolution of the population consists of a series of generations.

1. In each generation, the individuals in the population are tested and the most fit are allowed to reproduce.

2. Typically, reproduction is done by crossing two fit individuals, in the hope that their offspring will inherit the good features of each and thus be even more fit (though, of course, some inherit bad features from each parent and end up less fit.)

   Crossing is handled as follows:

   a) Typically two individuals are crossed to produce two new individuals, each having some genes from each parent.

   b) If each individual has the same number of genes arranged in some kind of sequence - we can pick a crossover point in the sequence at random, generating two offspring - e.g.

   Individual A: A1 A2 A3 A4 A5 A6 A7 A8
   Individual B: B1 B2 B3 B4 B5 B6 B7 B8

   Offspring if we cross between genes 2 and 3:

   A1 A2 B3 B4 B5 B6 B7 B8
   B1 B2 A3 A4 A5 A6 A7 A8

   c) Usually, both offspring go into the next generation.

3. A small amount of random mutation is also often used

   Mutation is done by randomly altering an individual gene. This may result in a solution that is less fit, more fit, or having the same fitness as the original.

   a) Mutation is often important, because it may be that no individual in the initial population contains the "correct" value of some gene, or perhaps the "correct" value of a gene is lost early due to incompatibility with some other genes that are selected away later.

   b) Of course, mutation can also be harmful, causing a "correct" value that was discovered by selection to be lost.

   c) Mutation is usually done with a fairly small probability - e.g. (say) 1% of the individuals in the new generation may undergo mutation.

4. Each generation may consist of a completely new collection of individuals created by crossover and/or mutation from the individuals in the previous generation - i.e. individuals "live" for only one generation. However, it is also possible to allow a subset of the most fit individuals in one generation to survive unchanged to the next. (This is a difference between algorithm evolution and biological evolution, of course.)

E. Everything is done randomly, often with probabilities determined by fitness:

1. We have already noted that the initial population is generated randomly.

2. Some implementations may allow some individuals to survive unchanged to the next generation. In this case, the individuals that survive can be selected randomly, with probability based on fitness - i.e. the more fit individuals have a higher probability of survival. (In some implementations, the most fit individuals may be guaranteed the right to survive unconditionally).

3. The individuals that reproduce may be selected randomly, with a probability based on fitness - i.e. the more fit individuals are given a higher probability of selection and the less fit ones a lower probability. (In some implementations, the most fit individuals may be guaranteed an opportunity to reproduce.)

4. The crossover point used when crossing parents is chosen randomly.

5. Whether or not a given gene is mutated is determined randomly with a predetermined - usually quite low - probability - typically independent of fitness - and if it is mutated, the change is determined randomly.

F. Depending on the nature of the problem, repetition of the process of creating new generations may continue until an individual is found that is adjudged to be perfectly fit, or until fitness stops improving, or after a predetermined number of generations.

G. Simulated evolution of this sort differs from biological evolution in several ways:

1. We have already noted one - the notion of an explicit goal used as a measure of fitness.
2. Another difference is the use of discrete generations.
3. A third difference may be the possibility that a very fit individual might survive for many generations.

III. **We will use a very simple variant of the game of NIM as an example.**

A. This is <u>not</u> a good example of a problem where a genetic approach is really useful (since we know an algorithm for perfect play), but it does provide a simple illustration of how developing an algorithm using a genetic approach can be done.

B. (Explain the one-pile variant of the game, then play a few demo games, using a maximum move of 3)

C. For this game (and indeed for all variants of NIM) there is an algorithm for perfect play, such that a player who will almost always win.

The algorithm for perfect play is this:

1. Define a "safe" state of the board as one where the number of pieces is a multiple of the maximum move+1 (e.g. a multiple of 4 in this case).

   By this definition, the winning state is a safe state.

2. An unsafe state is any state that is not safe.

3. Observe the following

   a) If a player is confronted with a board that is an unsafe state, there is always a legal move that will put it in a safe state.

   (Take number of pieces % (maximum move + 1))

   b) If a player is confronted with a board that is in a safe state, any legal move will result in putting it in an unsafe state. (Since taking 0 - the move given by the above formula - is not allowed)

4. The algorithm is this - always make the move that leaves the board in a safe state, if possible. If a player once leaves the board in a safe state, his opponent will be forced to leave it an unsafe state, and the knowledgable player can always put it back into a safe state, ultimately putting the board in the empty state.

5. (If two players both use the algorithm to play, the winner will be determined by whether the initial state of the game is safe or unsafe; but a player who uses the algorithm can always capitalize on just a single mistake by his opponent.)

D. Representing NIM as a candidate for a genetic solution

1. As we already have noted, NIM is not a problem one would normally solve using a GA since there is an algorithm for perfect play. However, it does serve as a simple example of the concepts.

2. We can represent a strategy for playing this variant of NIM as a vector of moves corresponding to each possible state of the pile - each of which we will consider to be a gene.

   For example, with a pile size limited to 10 and moves limited to taking 3 items, there would be 10 genes, each a number in the range 1 .. 3 (except that the first gene would have to be 1, and the second would have to be either 1 or 2 to comply with the rules of the game.)

   a) In this case, one possible solution might be

      1 1 3 2 2 2 1 3 1 3

      This says "if the pile contains 1 item, take 1; if it contains 2, take 1; if it contains 3, take 3; if it contains 4 take 2 ..."

   b) Obviously, the above is far from a perfect solution. However, it is still meaningful to call it a solution.

      Note that it will win in some cases - even if playing against an algorithmic player - e.g. game starts out with 8 items; opponent takes 2 (there is no algorithmically correct choice) leaving 6; program takes 2 leaving 4; opponent takes 1 leaving 3; program takes 3 and wins.

   c) It is, of course, easily possible to create a random population of solutions by randomly choosing values in the range 1 .. maximum move for each gene. (Except that gene 1 must be a 1, gene 2 must be a 1 or a 2 ...)

d) For the NIM Example, solutions can be compared by having each play against the pool of others and measuring fitness as percentage of wins.

For the initial, random population, we would expect the average fitness of an individual to be 50%. However, it is likely that some individuals will be more fit than this, while others will be less.

(1)For the example we will use for demonstration, it turns out that, with an initial population of 500, the most fit individual initially will have a fitness of over 90%, while the least fit will be under 10%. (Of course, since fitness is measured relative to other individuals in the population, even an individual that scores very high may not really be very good!)

(2)Actually, in the program we will use for demonstrations, we measure the fitness of an individual by having it play the other individuals in the population, with the individual whose fitness we are measuring playing first.

In the case of a population of perfectly fit individuals, we would, of course, expect all individuals to have the same fitness. What would that fitness be?

ASK

75%. A player that follows the NIM algorithm in a game with a maximum move of 3 will always win any game he moves first in if the initial state of the pile is unsafe. But if the initial state is safe, and the opponent also follows the NIM algorithm, then the player who moves first will lose. If the initial pile size is chosen randomly, then 1/4 of the games will start out with the pile in a safe state - and even a perfectly fit player will lose these.

(3)Note that the critic does not attempt to evaluate fitness in terms of the correctness of individual moves, but in terms of overall performance. (To make the problem interesting, we have to assume we don't actually know what the correct move is!)

(4)DEMO: genetic NIM - show initial random population., noting fitness evaluation for each.

E. Crossing in the NIM Example

1. Suppose we want to cross the solutions

   1 1 3 2 2 2 1 3 1 3 and 1 2 3 3 2 1 1 2 3 2

   just after the fifth gene

   The "children" of this cross are

   1 1 3 2 2 1 1 2 3 2 and 1 2 3 3 2 2 1 3 1 3

2. Given that we know the NIM algorithm, comparing the expected fitness of the "children" to that of the parents is instructive

   a) In the case of each of the parents, half of the genes for which we know an algorithmically correct value are correct

1 1 3 2 2 2 1 3 1 3	1 2 3 3 2 1 1 2 3 2
R W R - W R W - R W	R R R - W W W - W R
4/(4+4) = 0.5	4/(4+4) = 0.5

   b) In the case of the children, one is better and one is worse.

1 1 3 2 2 1 1 2 3 2	1 2 3 3 2 2 1 3 1 3
R W R - W W W - W R	R R R - W R W - R W
3/(3+5) = 0.375	5/(5+3) = 0.625

   c) Of course, in a real problem we wouldn't be able to make this sort of comparison!

3. Mutation in the NIM Example:

   a) Suppose we mutate 1 1 3 2 2 2 1 3 1 3 at the third gene. Any change we make will produce a less fit individual, since that gene was "right"

   b) OTOH, if we mutate this individual at the fifth gene, a change to 1 will likely produce a more fit individual, while a change to 3 will likely produce no fitness change.

F. Demonstrate NIM Example

1. Observe initial population. Note how much of the game the best individual has learned (first place where it has a "wrong" value)
2. Observe population as evolution with population 500 is done through 100 generations. Note how much of the game has been learned now.
3. Demonstrate games at this point
4. Learn for another 100 generations and look at learner.

G. Interestingly, if we change the rules of the game the learner can easily learn a new version of the game (which in effect means it has to forget the old version)

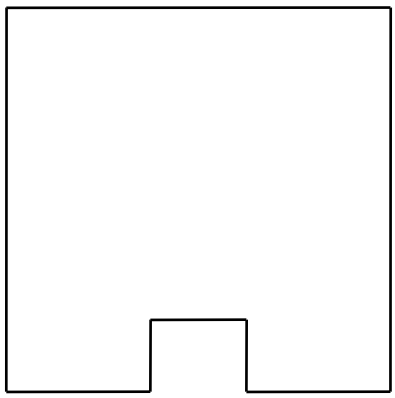
1. Start with learner that has learned 200 generations above.
2. Change the game to use a maximum move of 4 (optimum move is size of pile % 5).
3. Play a game - note how badly it does initially.
4. Learn through 200 generations.
5. Note how much of the game - but not all yet - has now been mastered.
6. Learn for another 200 generations - note improvement.

IV. A Genetic Programming Example

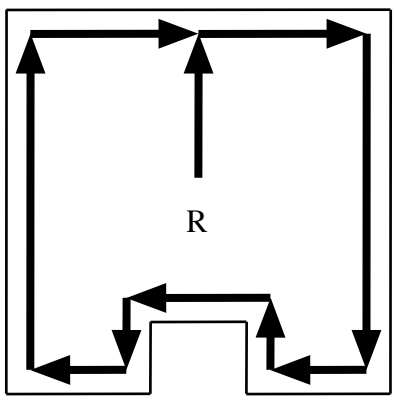
A. A book that was used in a previous version of the course includes a nice example of genetic programming, where a computer program to solve a problem is evolved by genetic means.

B. The problem is to evolve a program for a robot such that, when it is placed in an enclosed room, it moves to a wall and follows the wall around the room.

E.g. Given a room like this: (PROJECT)



We want the robot to do something like this (though we don't care how it gets to the wall initially or whether it moves clockwise or counterclockwise). (PROJECT)



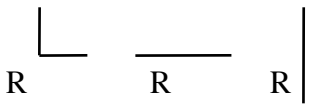
C. The primitives from which the program is to be constructed are the following.

1. Actions: north, east, south, west - move one block in the specified direction (if possible - otherwise do nothing)

2. Tests: PROJECT NILSSON FIGURE 4.2

   a) n, e, s, w - return true (1) if movement in the specified direction is blocked by a wall

   b) ne, se, sw, nw - return true (1) if there is a wall in the specified direction - including possibly a corner that doesn't actually block one of the robot's moves - e.g. ne would be true in all the following cases:



   (Note that these tests do not correspond directly to possible moves - e.g. there is no northeast move, though it is possible (if there are no walls in the way) to move northeast by moving one block north then one block east - or vice-versa.

3. Boolean connectives:

   AND (X, Y): if X == 0 then 0 else Y
   OR (X, Y): if X == 1 then 1 else Y
   NOT (X): if X then 0 else 1
   IF (X, Y, Z): if X then Y else Z

4. Example of a program that would solve the problem:

   NILSSON FIGURE 4.3 - PROJECT

   Trace through how this program works in the example problem

D. Applying Genetic Programming to this problem

1. A population of random programs is created

2. For each generation, the fitness of each program in the current population is evaluated. Fitness is measured as "number of squares next to the the wall that the robot visits in some number of moves"

   a) In the particular case Nilsson used, the room had 32 squares next to the wall.

   b) Fitness was measured as the number of these squares the robot visited in 60 moves from ten different random starting positions. [A fitness score of 320 would be perfection].

3. Crossover is handled by switching subtrees between parents

   NILSSON FIGURE 4.4 - PROJECT

4. Mutation could be handled by selecting a random subtree and replacing it with a new randomly-grown subtree.

E. Experimental results reported by Nilsson

1. The most fit individual in Generation 0

   NILSSON FIGURE 4.5 - PROJECT

2. The most fit individual in Generation 2

   NILSSON FIGURE 4.6 - PROJECT

3. The most fit individual in Generation 6

   NILSSON FIGURE 4.7 - PROJECT

4. The most fit individual in Generation 10 - a program that is actually a 100% solution.

   NILSSON FIGURE 4.8 - PROJECT

5. Evolution of fitness over the generations

   NILSSON FIGURE 4.9 - PROJECT

V. **Another Genetic Algorithm Example (omit if insufficient time)**

A. Melanie Mitchell's book - cited earlier - discussed some experiments with using GA's to evolve a strategy for a game known as "the prisoner's dilemma".

   1. READ Mitchell pp 17-18; PROJECT payoff matrix (p 18)

   2. Mitchell's book discusses experiments done by Axelrod on this game.

B. Any one game can be categorized in one of 4 ways (CC - both players cooperated; CD A cooperated and B defected; DC; DD)

   1. Because the programs in the tournament based their strategy on the last three games played with the same player, each with a move by each player, and with each move having two possible values, a strategy must be able to cope with 64 possible histories:

      3 games/history
      1 move for each player/game = 2 moves
      2 choices/player move

2. For each history, it must make a choice to either defect or cooperate on the next game. Thus, a strategy may be encoded as a 64 genes, each of which is either a C or a D, each representing the choice called for by the strategy for one possible history - e.g.

choice to make if all 3 games were CC	choice to make if first two games were CC and last was CD	....	choice to make if all 3 games were DD

3. For example, TIT-FOR-TAT for Player A would be encoded as

   CDCDCDCDCD ... CD (i.e. A always does what Player B did on the last game)

C. Results: READ Mitchell page 19 bottom - 21 top

VI. A genetic solution to a "real" problem

A. Another book develops an example where a genetic strategy was used to actually design an antenna for use in a space probe.

B. READ Floreano section 1.12.1 (pp. 40-42); PROJECT bottom two parts of figure 1.16