Chapter 10

Genetic Algorithms and Genetic Programming

Preview

- Overview
- Lecture Notes
  - Genetics Review
  - Genetic Algorithms
    - Algorithm
    - Illustrative Example
    - The Travelling Salesperson Problem
      - Order Crossover
      - Nearest Neighbor Crossover
      - Greedy Edge Crossover
      - Evaluation
  - Genetic Programming
    - Illustrative Example
    - Artificial Ant
    - Application to Financial Trading
      - Developing the Trading System
      - Evaluation
- Quick Check
- Classroom Discussion
- Homework
- Keywords
- Important Links

Overview

Evolution is the process of change in the genetic makeup of populations. **Natural selection** is the process by which organisms possessing traits that better enable them to adapt to environmental pressures survive and reproduce in greater numbers than other similar organisms thereby increasing he existence of those favorable traits in future generations.

Evolutionary computation endeavors to obtain approximate solutions to problems such as optimization problems using the evolutionary mechanisms involved in natural selection as its paradigm. The four areas of evolutionary computation are **genetic algorithms**, **genetic programming**, **evolutionary programming**, and **evolution strategies**. The first two areas are discussed in detail in the chapter. First, we briefly review genetics to provide a proper context for these algorithms.

Lecture Notes

Genetics Review

An **organism** is an individual form of life. A **cell** is the basic structural and functional unit of an organism. **Chromosomes** are the carriers of biologically expressed hereditary characteristics. Chromosomes consist of the polymer **deoxyribonucleic acid (DNA)**, which is composed of monomers (building blocks) called **nucleotides**. The nucleotides (a.k.a. bases) occurring in chromosomes are **adenine** (A), **guanine** (G), **cytosine** (C), and **thymine** (T). The **purines** A and G are structurally similar to each other, as are the **pyrimidines** C and T. Each chromosome consists of two complementary strands of DNA joined by hydrogen bonds between base pairs – corresponding pairs of nucleotides (A with T and G with C).

[See figure 10.1 on page 443]

A **gene** is a section of a chromosome that codes for the protein responsible for a particular biological function. An **allele** is any of several forms of a gene. Alleles are responsible for hereditary variation. A **genome** is a complete set of chromosomes in an organism. The **genotype** of an organism is the set of genes in its genome. The **phenotype** of an organism is the expression of its genes (e.g. certain aspects of appearance).

A **haploid** (**diploid**) cell contains one (two) genome(s). Each matched pair of chromosomes in a diploid cell is called a **homologous pair**, and each of the chromosomes is a **homolog**. A **somatic cell** is a cell in the body of an organism. Haploid (diploid) organisms have haploid (diploid) somatic cells.

Most diploid organisms reproduce sexually. In a cell division process called **meiosis**, each chromosome of a germ cell is duplicated to form a pair of **chromatids**. The chromatids may then exchange corresponding sections of genetic material in a process called **cross(ing-)over**. The chromatids are then separated into (diploid) daughter cells, which are further divided into (haploid) gametes. Thus, a **gamete** is a mature sexual reproductive cell produced through meiosis. In animals, gametes produced by male (female) organisms are **sperm (egg) cells**. One gamete from each of two parents unite to form a (diploid) **zygote**, which grows into a new organism (through chromosome replication, another cell division process called mitosis, and cell differentiation). Most haploid organisms reproduce asexually through **binary fission**. Some reproduce sexually through **fusion**.

Sometimes during cell division, errors called **mutations** occur in the DNA replication process. In a **substitution mutation**, one nucleotide is replaced by another. An **insertion (deletion) mutation** occurs when a section of DNA is added to (removed from) a chromosome.

Genetic Algorithms

Algorithm

Genetic algorithms are modeled on fusion in haploid organisms. Candidate solutions to a problem are represented by haploid individuals in a population, each having a single chromosome. The **fitness** of an individual is determined by the quality of the solution it represents. In each generation, individuals are selected for reproduction based on fitness. Genetic material from reproduced individuals is recombined in cross(ing-)over and randomly altered in mutation. Together, the processes of selection, cross(ing-)over and mutation produce the next generation of the population. The process is repeated until some termination condition is satisfied.

Selection of individuals employs both **exploitation** (concentrating on regions that look good) and **exploration** (looking for new regions without regard for how good they currently appear).

Illustrative Example

Suppose we seek to maximize f(x) = sin(x * pi / 256) over the interval [0,255] with x an integer. The solution x = 128 can be obtained analytically, so there is no practical reason to use a genetic algorithm to solve this problem. However, for purposes of illustration, these would be the steps:

Step 1. Choose an alphabet to represent solutions to the problem. All candidate solutions can be represented as 8-bit unsigned integers, so we choose an alphabet of {0, 1} and a chromosome length of 8.

Step 2. Decide how many individuals make up a population. For this example, we use a population size of 8.

Step 3. Decide how to initialize the population. As is often the case, we choose to initialize the population randomly. (Table 10.1, corrected below, shows one possible initial population. [The sixth individual should be 01001010 to be consistent with the rest of the table and the remainder of the example. Also, the fitness of the last individual should be .606, making the stated total fitness, normed fitnesses, and cumulative normed fitnesses incorrect.] The total fitness of the population is 5.040.)

Individual	x	f(x)	Normed f(x)	Cumulative Normed f(x)
1 0 1 1 1 1 0 1	189	0.733	0.145	0.145
1 1 0 1 1 0 0 0	216	0.471	0.094	0.239
0 1 1 0 0 0 1 1	99	0.937	0.186	0.425
1 1 1 0 1 1 0 0	236	0.243	0.048	0.473
1 0 1 0 1 1 1 0	174	0.845	0.168	0.641
0 1 0 0 1 0 1 0	74	0.788	0.156	0.797
0 0 1 0 0 0 1 1	35	0.416	0.083	0.880
0 0 1 1 0 1 0 1	53	0.606	0.120	1.000

Step 4. Decide how to evaluate fitness. Our goal is to maximize f(x) given above, so we use f(x) as the fitness function, i.e. the fitness of an individual representing the integer x is f(x).

Step 5. Decide which individuals to select for reproduction. In this example, we use “roulette wheel selection” [although it has serious disadvantages in real applications]. Conceptually, each individual is assigned to a wedge of a roulette wheel that is sized in proportion to the individual’s fitness. [What if smaller fitnesses are better? What if some individuals have negative fitness? What if all individuals had similar fitness? Selection operators exist that avoid all of these

difficulties.] Each individual in the offspring population is generated by spinning the roulette wheel and selecting the corresponding individual. (Table 10.2 shows a possible offspring population. The average fitness after selection is 0.776, compared to 0.630 in the initial population.)

Individual	x	f(x)
0 1 1 0 0 0 1 1	99	.937
0 0 1 1 0 1 0 1	53	.650
1 1 0 1 1 0 0 0	216	.471
1 0 1 0 1 1 1 0	174	.845
0 1 0 0 1 0 1 0	74	.788
1 0 1 0 1 1 1 0	174	.845
0 1 1 0 0 0 1 1	99	.937
1 0 1 1 1 1 0 1	189	.733

Step 6. Determine how to perform crossover and mutation. For this example, we randomly pair the individuals in the offspring population and perform "**two-point crossover**." We randomly choose two points within each pair and exchange the genetic material of the paired individuals between the two points [The text addresses the question of what happens if the second crossover point appears before the first crossover point, but the question is moot since both offspring are included in the population.] (Table 10.3, corrected below, shows possible results. [There are typos in the fifth and sixth lines. The last digit of each individual should be bold, and the sixth child should begin with 0 1.] The average fitness after crossover is 0.792, which is higher than the average fitness before crossover [but this is atypical in real applications; some improved individuals may be found, but many other crossovers are unsuccessful].

For this example, mutation could be performed by flipping each bit of each individual with probability somewhere in the range of .01 to .001 [some researchers argue that the probability should be inversely proportional to the length of the chromosome].

Parents	Children	x	f(x)
0 1 1 ***0 0 0*** 1 1	0 1 1 ***1 0 1*** 1 1	119	0.994
0 0 1 ***1 0 1*** 0 1	0 0 1 ***0 0 0*** 0 1	33	0.394
1 ***1 0 1 1*** 0 0 0	1 ***0 1 0 1*** 0 0 0	168	0.882
1 ***0 1 0 1*** 1 1 0	1 ***1 0 1 1*** 1 1 0	222	0.405
0 1 0 0 1 0 1 ***0***	***1 0*** 0 0 1 0 1 ***0***	138	0.992
1 0 1 0 1 1 1 ***0***	**0 1** 1 0 1 1 1 ***0***	110	0.976
0 1 1 0 0 ***0 1 1***	0 1 1 0 0 ***1 0 1***	101	0.946
1 0 1 1 1 ***1 0 1***	1 0 1 1 1 ***0 1 1***	187	0.749

Step 7. Decide when to terminate. Possible termination conditions include attainment of a maximum number of generations, expiration of allotted time, attainment of prescribed fitness level, [near-homogeneity of the population], and others. For this example, the condition could be either 10,000 generations or fitness in excess of 0.999.

The Travelling Salesperson Problem

Given a weighted directed graph of n vertices (“cities”) find the Hamiltonian cycle (“tour”) with minimum total weight (“tour length”). A Hamiltonian cycle is a permutation (v_1, v_2, …, v_n) of the vertices such that all of the edges $v_i \rightarrow v_{(i \bmod n)+1}$ are in the graph. The weight of a Hamiltonian cycle is the sum of the weights of those edges (“road lengths”). Genetic algorithms for the TSP based on three different crossover operators are presented. Only those steps that differ from the illustrative example are presented. In each case, Steps 1 and 4 are as follows.

> Step 1. Choose a representation scheme. Each chromosome will consist of a permutation (v_1, v_2, …, v_n) of the vertex indices representing the tour $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n \rightarrow v_1$.
>
> Step 4. The fitness of an individual is the length of the tour represented by its chromosome. Shorter tours are considered more fit.

The primary challenge is to ensure that the results of crossover represent valid tours. They must first of all be permutations of the vertex indices. Furthermore, all of the edges in the tours they represent must be in the graph, and this restriction is also relevant to mutation.

Order Crossover

> Step 6. Determine how to perform crossovers and mutations. As in the illustrative example, individuals are randomly paired, and two crossover points are selected. The first (second) child is constructed by copying the genes of the first (second) parent between the crossover points and filling in the remaining vertex indices in the order that they appear in the second (first) parent. This process guarantees that the child will be a permutation of the vertex indices. If the tour it represents includes edges that are not in the graph, the child is rejected. See Table 10.4. To perform mutation, either exchange two randomly selected vertices or reverse a randomly selected subsequence.

Nearest Neighbor Crossover

The Nearest Neighbor Algorithm constructs candidate TSP solutions by starting at an arbitrary vertex to initiate a tour and then repeatedly adds the closest unvisited vertex to the partial tour until a tour is completed. See Algorithm 10.1 and Figure 10.3. The text attributes Nearest Neighbor Crossover (NNX) to Süral, et al. (2010) and describes their experiments.

> Step 1. The representation scheme is the same as described for the order crossover version.
>
> Step 2. The researchers used population sizes of 50 and 100.
>
> Step 3. One technique was to generate the entire initial population randomly. Another was to generate half of it randomly and the other half using a hybrid of NNX and Greedy Edge (described next).
>
> Step 5. Decide which individuals to select for reproduction. Four copies of each of the individuals in the top 50% according to fitness are put in a pool. Pairs of parents are selected from the pool randomly without replacement.
>
> Step 6. Determine how to perform crossovers and mutations. For crossover, parents are combined to form a graph containing exactly the union of the edges in the parents (see Figure 10.4). NNA is applied to the resulting graph, choosing starting vertices until a valid tour results (adding edges and restarting if no starting vertex yields a valid tour). A stochastic variation is to choose the next edge from the set of possible edges with probability inversely proportional to its length. For mutation, the subpath between two randomly selected vertices is reversed (Figure

> 10.5). In version M1, mutation is applied only to the best offspring in each generation. In M2, it is applied to all offspring.

Greedy Edge Crossover

The Greedy Edge Algorithm (GEA) begins by sorting the edges in non-decreasing order, and then greedily adds edges while ensuring that no vertex exceeds degree 2 and no cycle of less than n edges is created (Algorithm 10.2 and Figure 10.6).

The Greedy Edge Crossover Algorithm (GEX) as also described by Süral, et al., is identical to NNX except:

> Step 6. Determine how to perform crossovers and mutations. For crossover, parents are combined to form the union graph as in NNX. In the more effective of two variations, the first half of the new tour is constructed by applying GEA to the union graph, and the remainder is constructed by continuing with all the edges in the original graph. For mutation, the subpath between two randomly selected vertices is reversed (Figure 10.5). In version M1, mutation is applied only to the best offspring in each generation. In M2, it is applied to all offspring.

Evaluation

Like other heuristic algorithms, genetic algorithms are not generally guaranteed to obtain optimal solutions within finite time [special cases do exist]. Thus, their effectiveness is typically evaluated through computational experimentation.

Süral, et al. report a series of experiments evaluating NNX and GEX on a set of 10 real-world (and therefore symmetric) TSP instances from TSPLIB ranging in size up to 226 cities. Their experiments included, for each of the 10 instances, 30 independent executions of each of 30 possible combinations of

- crossover mixtures (pure NNX, pure GEX, 50/50 NNX/GEX, 90/10 NNX/GEX, 95/5 NNX/GEX),
- mutation (none, M1, M2), and
- initialization (random, hybrid).

For each combination, the text reports (Table 10.5) the average over all executions on all instances of

- the percent deviation of the final best solution from the optimal solution,
- the number of generations required to reach convergence, and
- the time in seconds required to reach convergence on the platform used in the experiments.

In those experiments, the following were observed:

- pure NNX performed better than pure GEX,
- M2 performed the best of the mutations,
- hybrid initialization did not result in much better performance than random initialization,
- M2 performed slightly better with crossover mixtures with high rates of NNX usage.

Further experiments were performed using pure NNX, each of the mutation strategies, random initialization, and a population size of 100 (results in Table 10.6), resulting in improved effectiveness at the cost of marginally increased execution time. These combinations were further evaluated on TSP instances with between 318 and 1748 cities (results in Table 10.7) with the results that M2 was only slightly more effective M1 at a substantially increased execution time.

The researchers then compared the effectiveness of NNX-M1, NNX-M2, and the previously existing heuristic algorithms Meta-RaPS and ESOM on ten benchmark TSP instances. In each case, either NNX-M1 or NNX-M2 exhibited the best performance (Table 10.8).

Genetic Programming

In **genetic programming**, chromosomes represent candidate programs to solve a problem. The fitness function in some way measures how well the program solves the problem. Keeping this is mind, the high-level algorithm for genetic programming is identical to that of the genetic algorithm. Chromosomes are trees. Leaf nodes contain terminal symbols. Internal nodes contain function symbols that operate on the node's children (see Figure 10.7 for an example). [Variations on this scheme abound.]

Illustrative Example

Due to Banzhaf, et al. (1998). Given a set of points (x_i, y_i), learn a function $y = f(x)$ that fits the points. Ten points satisfying $y = x^2/2$ (the "correct" answer we hope to find in this example) are given in Table 10.9.

Step 1. Decide on the terminal set T. In this example, T = {x} U {-5, -4, …, 5}.

Step 2. Decide on the function set F. In this example, F = {+, -, *, /}.

Step 3. Decide on the population size. In this example, it is 600.

Step 4. Decide how to initialize the population. In this example, each individual is "grown" by randomly choosing a symbol in T U F. If it is a terminal symbol, we stop. Otherwise, we recursively generate two subtrees.

Step 5. Decide on a fitness function. In this example, it is the sum of the squared errors $(f(x_i)-y_i)^2$. Smaller errors are more fit.

Step 6. Decide which individuals to select for reproduction. In this example, a four-individual tournament selection process is used. Four individuals are selected randomly [the text does not state whether they are selected with or without replacement]. The two most fit produce children which replace the two least fit.

Step 7. Decide how to perform crossovers and mutations. In this example, randomly selected subtrees are exchanged between parents (see Figure 10.8). Mutation is performed by replacing a randomly selected subtree by a new randomly grown subtree. Mutations are performed on 5% of the offspring.

Step 8. Decide when to terminate. In this example, the algorithm terminates when the sum of the squared errors is zero or when 100 generations have been produced.

The text reports that the experiment of Banzhaf, et al. obtained the "correct" solution in four generations.

Artificial Ant

Consider the problem of developing a "robotic" ant that navigates along a trail of "food" in a two-dimensional grid (e.g. the Santa Fe Trail shown in Figure 10.9). The ant has one sensor that allows it to determine whether or not the square directly ahead of it contains food. At each time step, the ant can (1) turn 90 degrees right, (2) turn 90 degrees left, or (3) move forward one square. The goal of the ant is to use the minimum number of time steps to move from the start square to the final square after passing through each food-containing square.

Koza (1992) [by far the best known GP evangelist] developed the following genetic programming algorithm for this problem:

Step 1. Decide on the terminal set T. T = {right, left, move}.

Step 2. Decide on the function set F. F consists of

- if_food_ahead(instruction1, instruction2)
- do2(instruction1, instruction2)
- do3(instruction1, instruction2, instruction3)

Step 3. Decide on the population size. 500.

Step 4. Decide how to initialize the population. As described above.

Step 5. Decide on a fitness function. Fitness is the number of food pellets consumed within 400 time steps.

Step 6. Decide which individuals to select for reproduction. [The text does not specify].

Step 7. Decide how to perform crossovers and mutations. As described above.

Step 8. Decide when to terminate. When an individual succeeds in maximizing fitness (i.e. "eating" all 89 food pellets) or some maximum number of iterations is reached.

In one particular run, Koza obtained an optimal ant in the 22nd generation (see Figure 10.10). In that run, average fitness was 3.5 in generation 0 and 32 in the final generation [the text contains a typo].

Application to Financial Trading

There have been many efforts to develop automated systems to make the decision whether to buy, hold, or sell a stock on a given day. Farnsworth, et al. (2004) applies genetic programming to this problem, using the daily values of various market indicators as input.

Developing the Trading System

Step 1. Decide on the terminal set T. Values of several market indicators based on preliminary experimentation, normalized to the interval [-1,1] as well as real constants in that interval.

Step 2. Decide on the function set F. F consists of

- +, -, and *;
- IF: If x > 0 then y else z; and
- IFGT: If x > w then y else z.

Step 3. Decide on the population size. Various sizes were evaluated. Sizes smaller than 500 were ineffective. Those around 2500 had good results.

Step 4. Decide how to initialize the population. As described above, with a maximum number of levels (4) and maximum number of nodes (24) to prevent overfitting.

Step 5. Decide on a fitness function. Each individual was evaluated against actual S&P closing prices over a period of 4750 days as follows:

- The individual starts with $1.
- If the individual returns x > 0 on a given day, it buys as much as it can. Otherwise, it sells everything.
- Every transaction incurs a fee (to prevent overfitting).
- At the end of the period, the initial $1 is subtracted.

Step 6. Decide which individuals to select for reproduction. The top 75% of the population survives, which the remainder are replaced by more fit individuals.

Step 7. Decide how to perform crossovers and mutations. Crossover is as described above. Tree mutation is as described above. Node mutation randomly selects a few nodes and replaces them with random nodes that take the same arguments. The top 10% of the population is left unchanged; 50% are randomly selected for crossover with the top 10%; 20% are selected for node mutations; 10% are selected for tree mutations; and 10% are randomly replaced.

Step 8. Decide when to terminate. The maximum number of generations was in the 300-500 range.

The best individual obtained in one run is shown in Figure 10.11.

Evaluation

The best individual was evaluated using the 500 days of data following the training data. It had an evaluation fitness of 0.397, compared to 0.1098 for the buy-and-hold strategy.

Quick Check

1. ____ endeavors to obtain approximate solutions to problems such as optimization problems using the evolutionary mechanisms involved in natural selection as its paradigm.
   Answer: Evolutionary computation
2. In a genetic algorithm, the ____ of an individual is a measure of how well it solves the given problem.
   Answer: Fitness
3. A significant challenge in applying genetic algorithms to the Traveling Salesperson Problem is ensuring that offspring represent ____.
   Answer: Valid tours
4. In Genetic Programming, individuals are trees that represent candidate ____ to solve a problem.
   Answer: Programs
5. In the Artificial Ant problem, the goal is to develop a robotic ant that eats all the food pellets while using the smallest possible number of ____.
   Answer: Time steps.

Classroom Discussion

- Discuss a genetic algorithm to minimize f(x,y) = sin(x)cos(y)/(xy) over (0,2*pi].

Homework

Assign Exercises 5, 6, 9, 13, and 14

Keywords

- **Allele** – any of several forms of a gene.
- **Cell** – the basic structural and functional unit of an organism.
- **Chromosome** – the carrier of biologically expressed hereditary characteristics.
- **Crossover** – the exchange of corresponding sections of genetic material.
- **Deletion mutation** – mutation in which a nucleotide is removed from a chromosome.
- **Diploid** – containing one two genomes.
- **Evolution** – the process of change in the genetic makeup of populations.

- **Evolution strategies** – one of the four main areas of evolutionary computation.
- **Evolutionary computation** – endeavors to obtain approximate solutions to problems such as optimization problems using the evolutionary mechanisms involved in natural selection as its paradigm.
- **Evolutionary programming** – one of the four main areas of evolutionary computation.
- **Exploitation** – concentrating on regions of the search space that look good.
- **Exploration** – looking for new regions of the search space without regard for how good they currently appear.
- **Fitness** (in a genetic algorithm) – a value determined by the quality of the solution an individual represents.
- **Gene** – a section of a chromosome that codes for the protein responsible for a particular biological function.
- **Genetic algorithms** – one of the four main areas of evolutionary computation.
- **Genetic programming** – form of evolutionary computation in which chromosomes represent candidate programs to solve a problem.
- **Genetic programming** – one of the four main areas of evolutionary computation.
- **Genome** – a complete set of chromosomes in an organism.
- **Genotype** – the set of genes in an organism's genome.
- **Haploid** – containing one genome.
- **Insertion mutation** – mutation in which a nucleotide is added to a chromosome.
- **Mutations** – error in the DNA replication process.
- **Natural selection** – the process by which organisms possessing traits that better enable them to adapt to environmental pressures survive and reproduce in greater numbers than other similar organisms thereby increasing he existence of those favorable traits in future generations.
- **Organism** – an individual form of life.
- **Phenotype** – the expression of an organism's genes (e.g. certain aspects of appearance).
- **Substitution mutation** – mutation in which one nucleotide is replaced by another.

Important Links

- http://www.mitpressjournals.org/loi/evco (Evolutionary Computation journal)
- http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=4235 (IEEE Transactions on Evolutionary Computation)