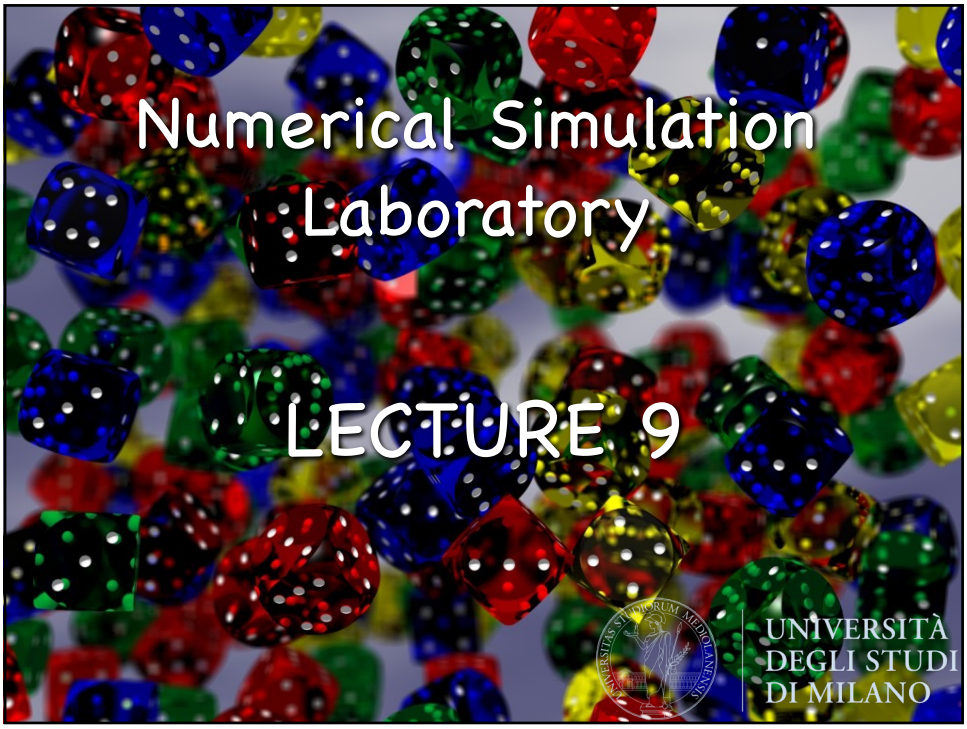


1

Outline

- Genetic algorithms

2

2

Introduction to Genetic Algorithms

- Genetic algorithms (GAs) were invented by John Holland in the 1960s and were developed by Holland and his collaborators in the 1960s and the 1970s. Their goal originally was to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which natural adaptation might be imported into computer systems.
- Holland's 1975 book *Adaptation in Natural and Artificial Systems* presented GAS as an abstraction of biological evolution. Gas soon became a general framework to solve optimization problems based on the mechanics of natural selection and natural genetics
- Why use evolution as an inspiration for solving computational optimization problems?
- As we have seen, many computational problems require searching through a huge number of possibilities for solutions. Such search problems can often benefit from an effective use of parallelism, in which many different possibilities are explored simultaneously in an efficient way.

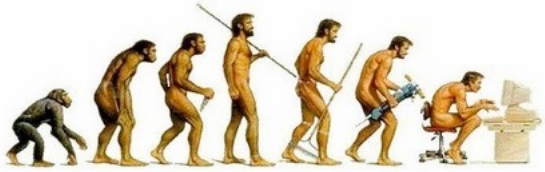
The 2007 NASA ST5 spacecraft antenna. This complicated shape was found by an evolutionary computer program to create the best radiation pattern.

3

The appeal of evolution

- Biological evolution is an appealing source of inspiration for addressing these optimization problems. Evolution is, in effect, a method of searching among an enormous number of possibilities for "solutions".
- In biology the enormous set of possibilities is the set of possible genetic sequences, and the desired "solutions" are highly fit organisms that are well able to survive and reproduce in their environments.
- Evolution can also be seen as a method for designing innovative solutions to complex problems. Seen in this light, the mechanisms of evolution can inspire computational search/optimization methods.
- Of course the fitness of a biological organism depends on many factors, for example, how well it can adapt to the environment and how well it can compete with or cooperate with the other organisms around it. The fitness criteria continually change as creatures evolve, so evolution is searching a constantly changing set of possibilities.
- Furthermore, natural evolution is a massively parallel search method at work!: rather than work on one species at a time, evolution tests and changes millions of species in parallel.

4



- The "rules" of evolution are remarkably simple: species evolve by means of random variation (via mutation, recombination, and other operators), followed by natural selection in which the fittest tend to survive and reproduce, thus propagating their genetic material to future generations. Yet these simple rules are thought to be responsible, in large part, for the extraordinary variety and complexity we see in the biosphere.
- Holland was the first to attempt to put computational evolution on a firm theoretical footing (see Holland 1975). This theoretical foundation, based on the notion of "schemas" was the basis of almost all subsequent theoretical work on genetic algorithms.
- GAs attractively contain the two major components of metaheuristic algorithms which are: intensification and diversification. What is needed to obtain these components is both computational parallelism and an intelligent strategy for choosing the next set of sequences to evaluate.

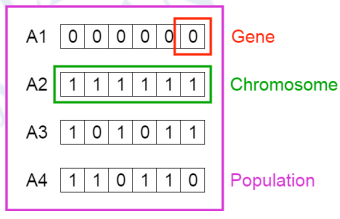
5

5

Elements of Genetic Algorithms

- In the original Holland's idea GA is a method for moving from one population of "chromosomes" (e.g., strings of ones and zeros, or "bits") to a new population by using a sort of "natural selection" together with the genetics-inspired operators of crossover, mutation, and maybe others.

- Each chromosome consists of "genes" (e.g., bits), each gene being an instance of a particular "allele" (e.g., 0 or 1).
- The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more descendants than the less fit ones.



- Crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single-chromosome organisms; mutation randomly changes the allele values of some locations in the chromosome.

6

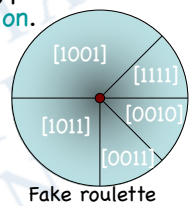
6

Population, Genes and Alleles

- Genetic algorithms are different from more normal optimization and search procedures in four ways:
  - GAs in general work with a coding of the parameter set, not the parameters themselves
  - GAs search from a population of "points", not a single point
  - GAs use direct information, not derivatives or other auxiliary knowledge
  - GAs use probabilistic transition rules, not deterministic rules
- Genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite-length string over some specific (in general finite) alphabet
- The population is the ensemble of the chromosomes existing at a given time. Chromosomes (a possible solution) could be:
  - Bit string [0101...1100] - Real numbers [43.2 -33.1 ... 0.0 89.2]
  - Program elements (genetic/evolutionary programming)
  - ...any data structure
- Genes: elements of a chromosome
  Bit string: the chromosome [1001101] has 7 genes
- Alleles: possible values of a gene
  Bit string: 2 values (0,1)

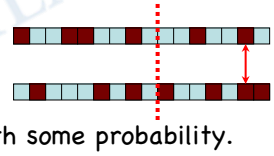
Operators and definitions

- The simplest form of genetic algorithm involves three types of operators: selection, crossover (single point), and mutation.
- Selection: this operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce (Fitness: the measure of goodness of a solution in an optimization problem).



Fake roulette wheel selection

- Crossover: when two individuals have been selected, both parents pass their chromosomes onto their offspring. The two chromosomes come together and swap genetic material. In binary GAs crossover is performed by swapping a part of binary strings between two solutions at a randomly chosen cross-site with some probability.



- Mutation: conversion of genes from one to another. In Binary GAs mutation is performed by converting some random bit of a binary string into its complementary bit (i.e. a 1 to a 0 or vice versa) with some probability. Mutation will help prevent the population from stagnating. It adds diversity.
- Note that crossover and mutation destroy old solutions.

A simple genetic algorithm

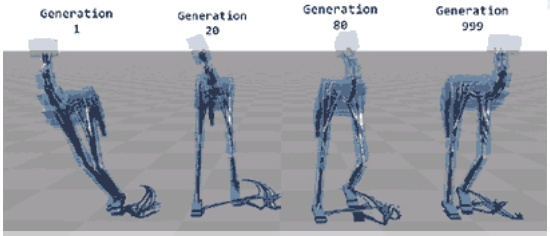
- Given a clearly defined problem to be solved and a bit string representation for candidate solutions, a simple GA works as follows:

1. Start with a randomly generated population of n l-bit chromosomes (candidate solutions to a problem).
2. Calculate the fitness f(x) of each chromosome x in the population.
3. Repeat the following steps until n offspring have been created:
   A. Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can be selected more than once to become a parent.
   B. With probability P_c crossover the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents.
   C. Mutate the two offspring at each locus with probability P_m, and place the resulting chromosomes in the new population.
4. Replace the current population with the new population.
5. Repeat: Go to step 2.

9

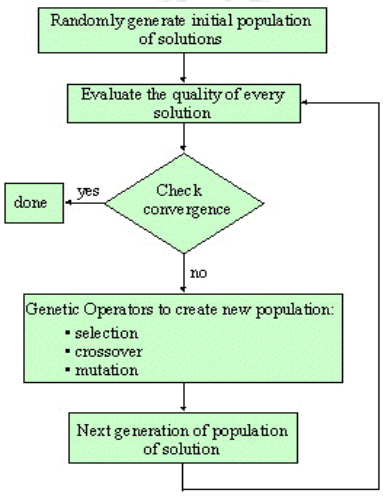
9

- Each iteration of this process is called a generation. The entire set of generations is called a run. At the end of a run there are often one or more highly fit chromosomes in the population.
- The simple procedure just described is the basis for most applications of GAs. There are a number of details to fill in, such as the size of the population and the probabilities of crossover and mutation, and the success of the algorithm often depends greatly on these details.



Flexible Muscle-Based Locomotion for Bipedal Creatures
In ACM Transactions on Graphics, Vol. 32, T. Geijtenbeek et al.

Flow chart



10

10

For example one can have:

- Solutions Generational GA: entire populations replaced with each iteration
- Steady-state GA: a few members replaced each generation
- Elitism: Some elite (good) solutions are carried onto the next generation without being destroyed.

GAs are especially useful when

- The search space is large, complex or the knowledge about it is scarce or it is difficult to encode to narrow the search space.
- Traditional search methods fail.

Moreover GAs:

- support multi-objective optimization
- give always an answer; and this answer gets better with time
- are inherently parallel

The traditional theory of Gas assumes in fact that, at a very general level of description, GAs work by discovering, emphasizing, and recombining good "building blocks" of solutions in a highly parallel fashion.

11

11

The "building block": schema

- The idea here is that good solutions tend to be made up of good building blocks—combinations of bit values that confer higher fitness on the strings in which they are present.
- Holland ('75) introduced the notion of schema to formalize the informal notion of "building blocks."
- DEFINITION: a schema is an equivalence class of chromosomes
- A schema is a set of bit strings that can be described by a template made up of ones, zeros, and asterisks, the asterisks (*) representing wild cards (or "don't cares").
- For example, the schema H=[1****1] represents the set of all 6-bit strings that begin and end with 1. (here I use Goldberg's notation in which H stands for "hyperplane." H is used to denote schemas because schemas define hyperplanes-"planes" of various dimensions in the l-dimensional space of length-l bit strings.)
- The strings that fit this template (e.g., [100111] and [110011]) are said to be instances of H (e.g. [1****1])

12

12

Instances, order and defining length

- DEFINITION: a schemas is of order n iff n is the number of genes different from an asterisks (*); formally n=o(H)
- DEFINITION: the defining length of a schemas H is the maximum distance in H between two defined genes; formally d(H)
- Thus, in the example above, the schema H=[1****1] is said to have two defined bits (non-asterisks) or, equivalently, to be of order 2. Its defining length (the distance between its outermost defined bits) is 5.

- A central belief of traditional GA theory is that schemas are (implicitly) the building blocks that the GA processes effectively under the operators of selection, mutation, and single-point crossover.
- How does the GA process schemas? In the case of a binary alphabet, any given bit string of length l is an instance of 2^l different schemas.

How do GA work?

- For example, the string [11] is an instance of [**] (all four possible bit strings of length 2), [*1] , [1*] , and [11] (the schema that contains only one string, 11).
- Thus, any given population of n strings contains instances of between 2^l (maximal homogeneity) and $n \times 2^l$ (maximal diversity) different schemas. If all the strings are identical, then there are instances of exactly 2^l different schemas; otherwise, the number is less than or equal to $n \times 2^l$.
- This means that, at a given generation, while the GA is explicitly evaluating the fitnesses of the n strings in the population, it is actually implicitly estimating the average fitness of a much larger number of schemas
- The Schema Theorem (Holland '75, see supplementary material) says that "short, low-order schemas whose average fitness remains above the mean will receive exponentially increasing numbers of samples (i.e., instances evaluated) over time". The Schema Theorem is a lower bound, since it deals only with the destructive effects of crossover and mutation.

The Schema Theorem

- Just as schemas are not explicitly represented or evaluated by the GA, the estimates of schema average fitnesses are not calculated or stored explicitly by the GA. However, as will be seen below, the GA's behaviour, in terms of the increase and decrease in numbers of instances of given schemas in the population, can be described as though it actually were calculating and storing these averages.
- We can calculate the approximate dynamics of this increase and decrease in schema instances as follows:
  - Let H be a schema with at least one instance x_i present in the population at time t, which retains N individuals.
  - Let N(H,t) (≤N) be the number of instances of H at time t, …
  - and let F(H,t) be the observed average fitness of H at time t (i.e., the average fitness of instances of H in the population at time t):

$$F(H,t) = \frac{\sum_{j=1(x_j \in H)}^{N(H,t)} f(x_j)}{N(H,t)}$$

being f the fitness function.

  - We want to calculate E[N(H,t+1)] , the expected number of instances of H at time t+1.

15

15

- Assume that the probability for a string x_i (a chromosome) to be selected is equal to

$$P(x_i) = \frac{f(x_i)}{\sum_j f(x_j)}$$

i.e. it is equal to the ratio among the fitness of x_i and the sum of the fitnesses of the population at time t.

- Then, assuming x_i is in the population at time t, and x_i is an instance of H, and (for now) ignoring the effects of crossover and mutation, we have that the expected number of instances of H at time t+1 is

$$E[N(H,t+1)] = N \times \sum_{j=1}^{N} P(x_j)\delta_{(x_j \in H)} = N \times \frac{\sum_{j=1(x_j \in H)}^{N(H,t)} f(x_j)}{\sum_{j=1}^{N} f(x_j)} =$$

$$N \times N(H,t) \frac{\frac{1}{N(H,t)} \sum_{j=1(x_j \in H)}^{N(H,t)} f(x_j)}{\sum_{j=1}^{N} f(x_j)} = N(H,t) \frac{F(H,t)}{\frac{1}{N}\sum_{j=1}^{N} f(x_j)} \qquad (*)$$

- Thus even though the GA does not calculate F(H,t) explicitly, the increases or decreases of schema instances in the population depend on this quantity: schemas with a greater average fitness will posses a greater number of instances as the generations evolve

16

16

- By assuming that $F(H,t) = [\Sigma_j f(x_j)/N](1+c) > \Sigma_j f(x_j)/N$ it follows that

$$E[N(H,t+1)] = N(H,t)\frac{\left[\frac{1}{N}\sum_{j=1}^{N} f(x_j)\right](1+c)}{\frac{1}{N}\sum_{j=1}^{N} f(x_j)} = N(H,t)(1+c)$$

- Then starting from t=0 and assuming c a constant we obtain:

$$E[N(H,t+1)] = N(H,t)(1+c)^t$$

  which is a *geometric progression*, the discrete analogous of the exponential form
- Thus, the selection operator assigns an increasing (decreasing) number of instances to schemas with high (low) idoneity following an exponential law.
- Crossover and mutation can both destroy and create instances of H. For now let us include only the destructive effects of crossover and mutation, those that decrease the number of instances of H.
- Including these effects, we modify the right side of the previous equation to give a lower bound on E[N(H,t+1)].

17

17

- Let P_c be the probability that single-point crossover will be applied to a string, and suppose that an instance of schema H is picked to be a parent.
- Schema H is said to "survive" under single-point crossover if one of the offspring is also an instance of schema H. We can give a lower bound on the probability $S_c(H)$ that H will survive to a single-point crossover:

$$S_c(H) \geq 1 - P_c \times \left(\frac{d(H)}{l-1}\right)$$

  where d(H) is the defining length of H and l is the length of bit strings in the search space.
- That is, crossovers occurring within the defining length of H can destroy H (i.e., can produce offspring that are not instances of H), so we multiply the fraction of the string that H occupies by the crossover probability to obtain an upper bound on the probability that it will be destroyed. (The value is an upper bound because some crossovers inside a schema's defined positions will not destroy it, e.g., if two identical strings cross with each other.)
- Subtracting this value from 1 gives a lower bound on the probability of survival $S_c(H)$. In short, the probability of survival under crossover is higher for shorter schemas.

18

18

- The disruptive effects of mutation can be quantified as follows: Let P_m be the probability of any bit being mutated. Then $S_m(H)$, the probability that schema H will survive under mutation of an instance of H, is equal to $(1-P_m)^{o(H)}$, where o(H) is the order of H (i.e., the number of defined bits in H).
- That is, for each bit, the probability that the bit will not be mutated is $1-P_m$, so the probability that no defined bits of schema H will be mutated is this quantity multiplied by itself o(H) times. In short, the probability of survival under mutation is higher for lower-order schemas.
- These disruptive effects can be used to amend equation (*) :

$$E[N(H,t+1)] \geq N(H,t)\frac{F(H,t)}{\frac{1}{N}\sum_{j=1}^{N} f(x_j)}\left(1-P_c\frac{d(H)}{l-1}\right)(1-P_m)^{o(H)}$$

- This is known as the Schema Theorem (Holland '75). It describes the growth of a schema from one generation to the next. The Schema Theorem is often interpreted as implying that short, low-order schemas whose average fitness remains above the mean will receive exponentially increasing numbers of samples (i.e., instances evaluated) over time.

19

19

- The Schema Theorem is a lower bound, since it deals only with the destructive effects of crossover and mutation.
- However, crossover is believed to be a major source of the GA's power, with the ability to recombine instances of good schemas to form instances of equally good or better higher-order schemas; this is known as the Building Block Hypothesis (Goldberg '89).
- In evaluating a population of n strings, the GA is implicitly estimating the average fitnesses of all schemas that are present in the population, and increasing or decreasing their representation according to the Schema Theorem.
- This simultaneous implicit evaluation of large numbers of schemas in a population of n strings is known as implicit paralelism (Holland '75).
- The effect of selection is to gradually bias the sampling procedure toward instances of schemas whose fitness is estimated to be above average.
- Mutation is what prevents the loss of diversity at a given bit position.
- In the end note that GAs can be seen as Markov processes!

20

20

Lecture 9: Suggested books

- M.P. David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*
- A. Brabazon, M. O'Neill, S. McGarraghy, *Natural Computing Algorithms*, Springer (2015)

21