Introduction to Genetic Algorithms

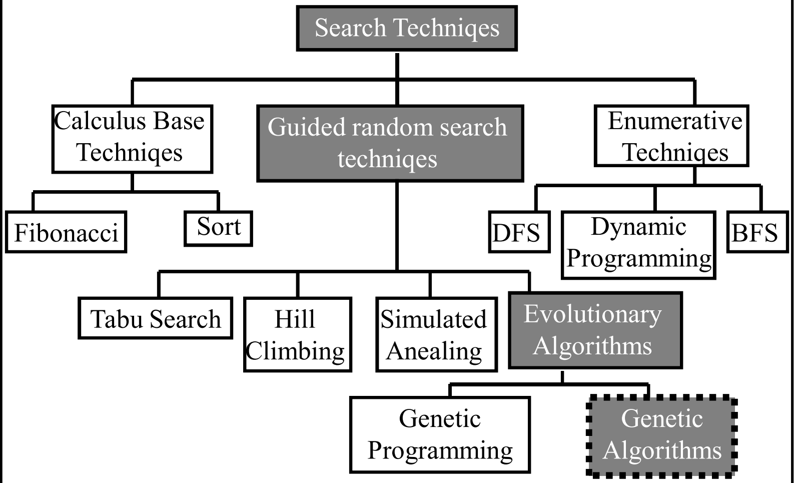
Genetic Algorithms (GA) OVERVIEW

- A class of probabilistic optimization algorithms
- Inspired by the biological evolution process
- Uses concepts of “Natural Selection” and “Genetic Inheritance” (Darwin 1859)
- Originally developed by John Holland (1975)

GA overview (cont)

- Particularly well suited for hard problems where little is known about the underlying search space
- Widely-used in business, science and engineering

Classes of Search Techniques



A genetic algorithm maintains a population of candidate solutions for the problem at hand, and makes it evolve by iteratively applying a set of stochastic operators

Stochastic operators

- ***Selection*** replicates the most successful solutions found in a population at a rate proportional to their relative quality
- ***Recombination*** decomposes two distinct solutions and then randomly mixes their parts to form novel solutions
- ***Mutation*** randomly perturbs a candidate solution

The Metaphor

Genetic Algorithm	Nature
Optimization problem	Environment
Feasible solutions	Individuals living in that environment
Solutions quality (fitness function)	Individual's degree of adaptation to its surrounding environment

The Metaphor (cont)

Genetic Algorithm	Nature
A set of feasible solutions	A population of organisms (species)
Stochastic operators	Selection, recombination and mutation in nature's evolutionary process
Iteratively applying a set of stochastic operators on a set of feasible solutions	Evolution of populations to suit their environment

The Metaphor (cont)

The computer model introduces simplifications (relative to the real biological mechanisms),

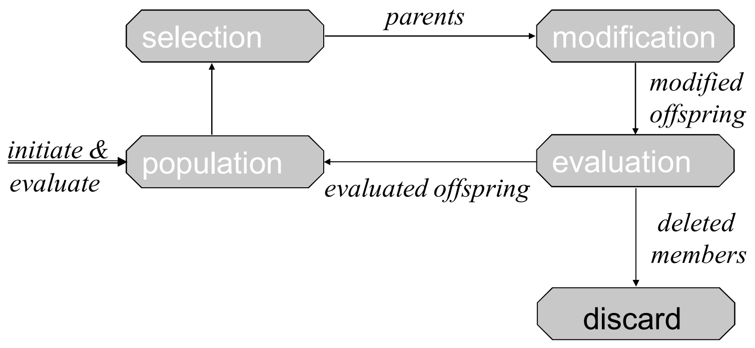
BUT

surprisingly complex and interesting structures have emerged out of evolutionary algorithms

Simple Genetic Algorithm

```
produce an initial population of individuals
evaluate the fitness of all individuals
while termination condition not met do
        select fitter individuals for reproduction
        recombine between individuals
        mutate individuals
        evaluate the fitness of the modified individuals
        generate a new population
End while
```

The Evolutionary Cycle



Example: the MAXONE problem

Suppose we want to maximize the number of ones in a string of l binary digits

Is it a trivial problem?

It may seem so because we know the answer in advance

However, we can think of it as maximizing the number of correct answers, each encoded by 1, to l yes/no difficult questions`

Example (cont)

- An individual is encoded (naturally) as a string of l binary digits
- The fitness f of a candidate solution to the MAXONE problem is the number of ones in its genetic code
- We start with a population of n random strings. Suppose that l = 10 and n = 6

Example (initialization)

We toss a fair coin 60 times and get the following initial population:

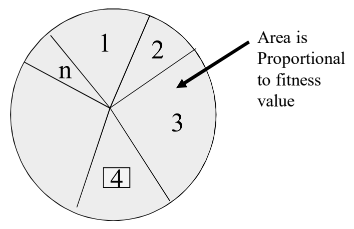
$s_1 = 1111010101$	$f(s_1) = 7$
$s_2 = 0111000101$	$f(s_2) = 5$
$s_3 = 1110110101$	$f(s_3) = 7$
$s_4 = 0100010011$	$f(s_4) = 4$
$s_5 = 1110111101$	$f(s_5) = 8$
$s_6 = 0100110000$	$f(s_6) = 3$

Example (selection1)

Next we apply fitness proportionate selection with the roulette wheel method:

Individual i will have a $\frac{f(i)}{\sum_i f(i)}$ probability to be chosen

We repeat the extraction as many times as the number of individuals we need to have the same parent population size (6 in our case)



Example (selection2)

Suppose that, after performing selection, we get the following population:

$s_1` = 1111010101$ (s_1)

$s_2` = 1110110101$ (s_3)

$s_3` = 1110111101$ (s_5)

$s_4` = 0111000101$ (s_2)

$s_5` = 0100010011$ (s_4)

$s_6` = 1110111101$ (s_5)

Example (crossover1)

Next we mate strings for crossover. For each couple we decide according to crossover probability (for instance 0.6) whether to actually perform crossover or not

Suppose that we decide to actually perform crossover only for couples ($s_1`$, $s_2`$) and ($s_5`$, $s_6`$). For each couple, we randomly extract a crossover point, for instance 2 for the first and 5 for the second

Example (crossover2)

Before crossover:

$s_1` = 1111010101$ $s_5` = 0100010011$
$s_2` = 1110110101$ $s_6` = 1110111101$

After crossover:

$s_1`` = 1110110101$ $s_5`` = 0100011101$
$s_2`` = 1111010101$ $s_6`` = 1110110011$

Example (mutation1)

The final step is to apply random mutation: for each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)

Before applying mutation:

$s_1`` = 1110110101$

$s_2`` = 1111010101$

$s_3`` = 1110111101$

$s_4`` = 0111000101$

$s_5`` = 0100011101$

$s_6`` = 1110110011$

Example (mutation2)

After applying mutation:

$$s_1^{```} = 1110100101 \quad f(s_1^{```}) = 6$$
$$s_2^{```} = 1111110100 \quad f(s_2^{```}) = 7$$
$$s_3^{```} = 1110101111 \quad f(s_3^{```}) = 8$$
$$s_4^{```} = 0111000101 \quad f(s_4^{```}) = 5$$
$$s_5^{```} = 0100011101 \quad f(s_5^{```}) = 5$$
$$s_6^{```} = 1110110001 \quad f(s_6^{```}) = 6$$

Example (end)

In one generation, the total population fitness changed from 34 to 37, thus improved by ~9%

At this point, we go through the same process all over again, until a stopping criterion is met

Components of a GA

A problem definition as input, and

- Encoding principles (gene, chromosome)
- Initialization procedure (creation)
- Selection of parents (reproduction)
- Genetic operators (mutation, recombination)
- Evaluation function (environment)
- Termination condition

Representation (encoding)

Possible individual's encoding

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of element (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- ... any data structure ...

Representation (cont)

When choosing an encoding method rely on the following key ideas

- Use a data structure as close as possible to the natural representation
- Write appropriate genetic operators as needed
- If possible, ensure that all genotypes correspond to feasible solutions
- If possible, ensure that genetic operators preserve feasibility

Initialization

Start with a population of randomly generated individuals, or use
- A previously saved population
- A set of solutions provided by a human expert
- A set of solutions provided by another heuristic algorithm

Selection

- Purpose: to focus the search in promising regions of the space
- Inspiration: Darwin's "survival of the fittest"
- Trade-off between *exploration* and *exploitation* of the search space

Next we shall discuss possible selection methods

Fitness Proportionate Selection

- Derived by Holland as the optimal trade-off between exploration and exploitation

Drawbacks

- Different selection for $f_1(x)$ *and* $f_2(x) = f_1(x) + c$
- *Superindividuals* cause convergence (that may be premature)

Linear Ranking Selection

Based on sorting of individuals by decreasing fitness

The probability to be extracted for the ith individual in the ranking is defined as

$$p(i) = \frac{1}{n}\left[\beta - 2(\beta - 1)\frac{i-1}{n-1}\right], 1 \le \beta \le 2$$

where β can be interpreted as the expected sampling rate of the best individual

Local Tournament Selection

Extracts k individuals from the population with uniform probability (without re-insertion) and makes them play a “tournament”, where the probability for an individual to win is generally proportional to its fitness

Selection pressure is directly proportional to the number k of participants

Recombination (Crossover)

* Enables the evolutionary process to move toward promising regions of the search space

* Matches good parents’ sub-solutions to construct better offspring

Mutation

Purpose: to simulate the effect of errors that happen with low probability during duplication

Results:

- Movement in the search space
- Restoration of lost information to the population

Evaluation (fitness function)

- Solution is only as good as the evaluation function; choosing a good one is often the hardest part
- Similar-encoded solutions should have a similar fitness

Termination condition

Examples:

- A pre-determined number of generations or time has elapsed
- A satisfactory solution has been achieved
- No improvement in solution quality has taken place for a pre-determined number of generations

Another Example: The Traveling Salesman Problem (TSP)

The traveling salesman must visit every city in his territory exactly once and then return to the starting point; given the cost of travel between all cities, how should he plan his itinerary for minimum total cost of the entire tour?

TSP $\in$ NP-Complete

Note: we shall discuss a single possible approach to approximate the TSP by GAs

TSP (Representation, Evaluation, Initialization and Selection)

A vector $v = (i_1\, i_{2\ldots}\, i_n)$ represents a tour (v is a permutation of {1,2,…,n})

Fitness f of a solution is the inverse cost of the corresponding tour

Initialization: use either some heuristics, or a random sample of permutations of {1,2,…,n}

We shall use the fitness proportionate selection

TSP (Crossover1)

OX – builds offspring by choosing a sub-sequence of a tour from one parent and preserving the relative order of cities from the other parent and feasibility

Example:

p_1 = (1 2 3 4 5 6 7 8 9) and

p_2 = (4 5 2 1 8 7 6 9 3)

First, the segments between cut points are copied into offspring

o_1 = (x x x 4 5 6 7 x x) and

o_2 = (x x x 1 8 7 6 x x)

TSP (Crossover2)

Next, starting from the second cut point of one parent, the cities from the other parent are copied in the same order

The sequence of the cities in the second parent is
9 – 3 – 4 – 5 – 2 – 1 – 8 – 7 – 6

After removal of cities from the first offspring we get
9 – 3 – 2 – 1 – 8

This sequence is placed in the first offspring

o_1 = (2 1 8 4 5 6 7 9 3), and similarly in the second

o_2 = (3 4 5 1 8 7 6 9 2)

TSP (Inversion)

The sub-string between two randomly selected points in the path is reversed

Example:

(1 2 3 4 5 6 7 8 9) is changed into (1 2 7 6 5 4 3 8 9)

Such simple inversion guarantees that the resulting offspring is a legal tour

GAs: Why Do They Work?

In this section we take an in-depth look at the working of the standard genetic algorithm, explaining why GAs constitute an effective search procedure

For simplicity we discuss binary string representation of individuals

Notation (schema)

{0,1,#} is the symbol alphabet, where # is a special *wild card* symbol

A *schema* is a template consisting of a string composed of these three symbols

Example: the schema [01#1#] matches the strings: [01010], [01011], [01110] and [01111]

Notation (order)

The *order* of the schema S (denoted by $o(S)$) is the number of fixed positions (0 or 1) presented in the schema

Example: for S_1 = [01#1#], $o(S_1)$ = 3
for S_2 = [##1#1010], $o(S_2)$ = 5

The order of a schema is useful to calculate survival probability of the schema for mutations

There are $2^{l-o(S)}$ different strings that match S

Notation (defining length)

The *defining length* of schema S (denoted by $\delta(S)$) is the distance between the first and last fixed positions in it

Example: for S_1 = [01#1#], $\delta(S_1) = 4 – 1 = 3$,
for S_2 = [##1#1010], $\delta(S_2) = 8 – 3 = 5$

The defining length of a schema is useful to calculate survival probability of the schema for crossovers

Notation (cont)

$m(S,t)$ is the number of individuals in the population belonging to a particular schema S at time t (in terms of generations)

$f_S(t)$ is the average fitness value of strings belonging to schema S at time t

$f(t)$ is the average fitness value over all strings in the population

The effect of **Selection**

Under fitness-proportionate selection the expected number of individuals belonging to schema S at time (t+1) is $m(S,t+1) = m(S,t)\,(f_S(t)/f(t))$

Assuming that a schema S remains above average by $0 \le c$, (i.e., $f_S(t) = f(t) + c\,f(t)$), then

$$m(S,t) = m(S,0)\,(1 + c)^t$$

Significance: “above average” schema receives an exponentially increasing number of strings in the next generation

The effect of **Crossover**

The probability of schema S ($|S| = l$) to survive crossover is $p_s(S) \ge 1 - p_c(\delta(S)/(l-1))$

The combined effect of selection and crossover yields

$$m(S,t+1) \ge m(S,t)\,(f_S(t)/f(t))\,[1 - p_c(\delta(S)/(l-1))]$$

Above-average schemata with short defining lengths would still be sampled at exponentially increasing rates

The effect of **Mutation**

The probability of S to survive mutation is:

$$p_s(S) = (1 - p_m)^{o(S)}$$

Since $p_m \ll 1$, this probability can be approximated by:

$$p_s(S) \approx 1 - p_m \cdot o(S)$$

The combined effect of selection, crossover and mutation yields

$$m(S,t+1) \ge m(S,t)\,(f_S(t)/f(t))\,[1 - p_c(\delta(S)/(l-1)) - p_m o(S)]$$

Schema Theorem

Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm

Result: GAs explore the search space by short, low-order schemata which, subsequently, are used for information exchange during crossover

Building Block Hypothesis

A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the building blocks

The building block hypothesis has been found to apply in many cases but it depends on the representation and genetic operators used

Building Block Hypothesis (cont)

It is easy to construct examples for which the above hypothesis does not hold:

S_1 = [111#######] and S_2 = [########11]

are above average, but their combination

S_3 = [111#####11] is much less fit than S_4 = [000#####00]

Assume further that the optimal string is S_0 = [1111111111]. A GA may have some difficulties in converging to S_0, since it may tend to converge to points like [0001111100].

Some building blocks (short, low-order schemata) can mislead GA and cause its convergence to suboptimal points

Building Block Hypothesis (cont)

Dealing with deception:

Code the fitness function in an appropriate way (assumes prior knowledge)

or

Use a third genetic operator, *inversion*

A brief introduction to Co-evolution

- Two or more species evolve and interact through the fitness function
- Closer to real biological systems
- Competitive (prey-predator) or Cooperative
- Examples: Hillis (competitive), my work (cooperative)