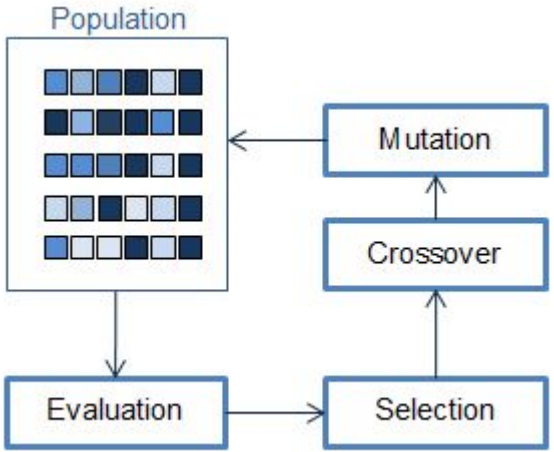
Genetic Algorithms

A.I.

Outline

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population
3. **[New population]** Create a new population by repeating following steps until the new population is complete
    1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
    2. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
    3. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
    4. **[Accepting]** Place new offspring in a new population
4. **[Replace]** Use new generated population for a further run of algorithm
5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6. **[Loop]** Go to step **2**



Terms

- The *physical expression* of the genotype is called the *phenotype*. The genotype is the actual genes.
- Allele, in biology, is the term given to the appropriate range of values for genes. In genetic algorithms, an allele is the value of the gene (or genes).
- A generation is an iteration of the genetic algorithm. Conventionally, the initial random generation is known as generation zero.
- Convergence is a reduction in the diversity of genes available in the population due to fitness proportionate operators.
- The *fitness function* is the function you want to optimize. For standard optimization algorithms, this is known as the objective function.
- Fitness scaling converts the raw fitness scores that are returned by the fitness function to values in a range that is suitable for the selection function.

Genetic Algorithms

Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics.

Unlike older AI systems, they do not break easily even if the inputs changed slightly, or in the presence of reasonable noise.

GAs simulate the survival of the fittest among individuals over consecutive generations. Each generation consists of a population of character strings that are analogous to chromosomes. Each individual represents a point in a search space and a possible solution. The individuals in the population are then made to go through a process of evolution.

The usual representation is {0,1}.

Genetic Algorithms

Individual “strands” are likened to chromosomes and the variables are analogous to genes. Thus a chromosome (solution) is composed of several genes (variables). A **fitness score** is assigned to each solution representing the abilities of an individual to `compete'. The individual with the optimal (or generally near optimal) fitness score is sought. The GA aims to use **selective `breeding'** of the solutions to produce `offspring' better than the parents by combining information from the chromosomes.

The GA maintains a population of n chromosomes (solutions) with associated fitness values. Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan. Consequently highly fit solutions are given more opportunities to reproduce, so that offspring inherit characteristics from each parent. As parents mate and produce offspring, room must be made for the new arrivals since the population is kept at a static size.

Each successive generation will contain more good `partial solutions' than previous generations. Eventually, there is convergence to a “best solution”.

Genetic Algorithms

GAs are based on an analogy with the genetic structure and behaviour of chromosomes within a population of individuals using the following foundations:

- Individuals in a population compete for resources and mates.
- Those individuals most successful in each **'competition'** will produce more offspring than those individuals that perform poorly.
- Genes from `good' individuals propagate throughout the population so that two good parents will sometimes produce offspring that are better than either parent.
- Thus each successive generation will become more suited to their environment.

Steps

Encode many “random” solutions to the problem as bit strings. At first the strings should be entirely random, made up of 0s and 1s.

1. **Fitness function** which measures the chromosomes
2. **Selection** which equates to survival of the fittest
3. **Crossover** which represents mating between individuals
4. **Mutation** which introduces random modifications

Fitness Function

Fitness Function

The fitness function:

- Fitness computation time of a single solution should be very fast
- Precise model for the “correct answer” is not always necessary
- The fitness function works well under uncertain or noisy.

Two main classes of fitness functions exist:

- Where the fitness function does not change, as in optimizing a fixed function or testing with a fixed set of test cases
- Where the fitness function is mutable, as in niche differentiation or co-evolving the set of test cases.

Selection

Genetic Algorithms

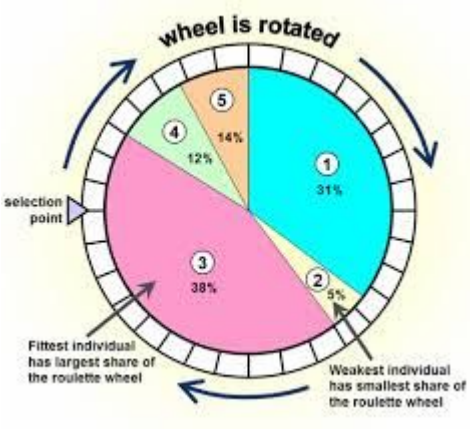
Selection after a fitness test does not guarantee that the fittest member goes through to the next generation, merely that it has a very good chance of doing so.

How should we choose who gets to go to the next generation? Choosing too many “just fit” solutions may converge quickly but lack enough diversity to get to the actual solution. It may also miss answers that need a small bit of tweaking to get there, but don’t look that way at first glance.

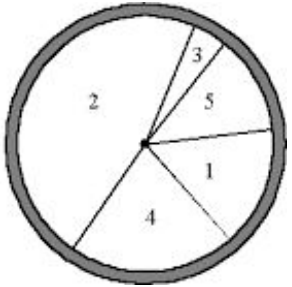
How many should we choose to create the next generation?

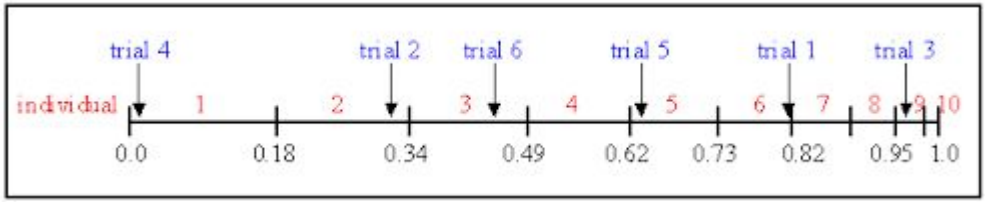
Roulette Wheel

The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness. A random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained (called mating population). This technique is analogous to a roulette wheel with each slice proportional in size to the fitness.



	String s_i	Fitness $f(s_i)$	Relative Fitness
s_1	10110	2.23	0.14
s_2	11000	7.27	0.47
s_3	11110	1.05	0.07
s_4	01001	3.35	0.21
s_5	00110	1.69	0.11





Methods of Selection

***Elitist selection*:** The most fit members of each generation are guaranteed to be selected.

***Fitness-proportionate selection*:** More fit individuals are more likely, but not certain, to be selected.

***Roulette-wheel selection*:** A form of fitness-proportionate selection in which the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness. (Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. The wheel is then spun, and whichever individual "owns" the section on which it lands each time is chosen.)

***Scaling selection*:** As the average fitness of the population increases, the strength of the selective pressure also increases and the fitness function becomes more discriminating. This method can be helpful in making the best selection later on when all individuals have relatively high fitness and only small differences in fitness distinguish one from another.

Selection

***Tournament selection*:** Subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce.

***Rank selection*:** Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking rather than absolute differences in fitness. The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones, which would reduce the population's genetic diversity and might hinder attempts to find an acceptable solution.

***Steady-state selection*:** The offspring of the individuals selected from each generation go back into the pre-existing gene pool, replacing some of the less fit members of the previous generation. Some individuals are retained between generations.

***Hierarchical selection*:** Individuals go through multiple rounds of selection each generation. Lower-level evaluations are faster and less discriminating, while those that survive to higher levels are evaluated more rigorously. The advantage of this method is that it reduces overall computation time by using faster, less selective evaluation to weed out the majority of individuals that show little or no promise, and only subjecting those who survive this initial test to more rigorous and more computationally expensive fitness evaluation.

Crossover and Mutation

Crossover Rate

This is simply the chance that two chromosomes will swap their bits. A good value for this is around 0.7. Crossover is performed by selecting a random gene along the length of the chromosomes and swapping all the genes after that point.

e.g. Given two chromosomes

10001001110010010
01010001001000011

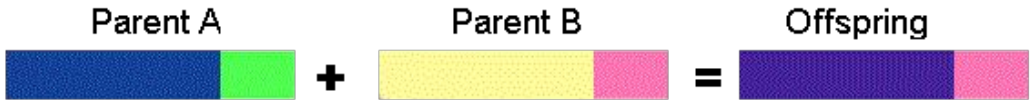
Choose a random bit along the length, say at position 9, and swap all the bits after that point

so the above become:

10001001101000011
01010001010010010

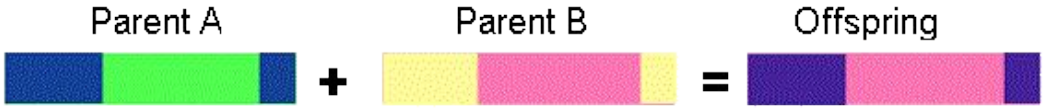
Crossover and Mutation

Single Point Crossover:one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, the rest is copied from the second parent



11001011+11011**111** = **11001111**

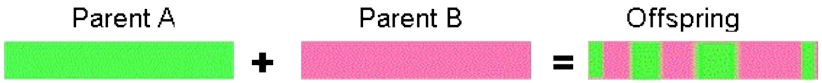
Two point crossover - two crossover point are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent



11001011 + 11**011**111 = **11011111**

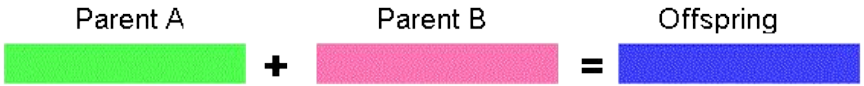
Crossover and Mutation

Uniform crossover - bits are randomly copied from the first or from the second parent



1**1**0010**11** + **1**10**111**01 = 11011111

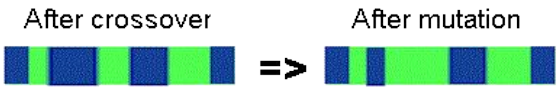
Arithmetic crossover - some arithmetic operation is performed to make a new offspring



11001011 + 11011111 = 11001001 (AND)

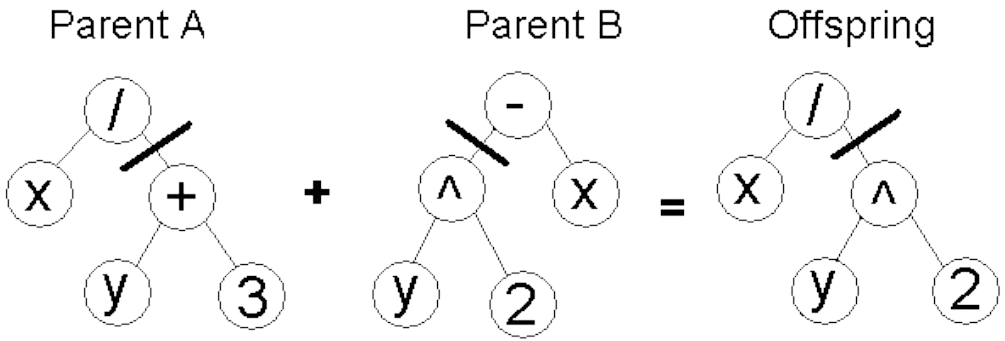
Mutation: Bit inversion - selected bits are inverted

1**1**001001 => 1**0**001001



Tree Crossover

Tree crossover - in both parent one crossover point is selected, parents are divided in that point and exchange part below crossover point to produce new offspring



Termination

Termination

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

TSP Example

Travelling salesman problem (TSP) has been already mentioned in one of the previous chapters. To repeat it, there are cities and given distances between them.Travelling salesman has to visit all of them, but he does not to travel very much. Task is to find a sequence of cities to minimize travelled distance. In other words, find a minimal Hamiltonian tour in a complete graph of ***N*** nodes.

TSP Example (http://www.obitko.com/tutorials/genetic-algorithms/tsp-example.php)

Population of n chromosomes is used.

Crossover

- One point - part of the first parent is copied and the rest is taken in the same order as in the second parent
- Two point - two parts of the first parent are copied and the rest between is taken in the same order as in the second parent
- None - no crossover, offspring is exact copy of parents

Mutation

- Normal random - a few cities are chosen and exchanged
- Random, only improving - a few cities are randomly chosen and exchanged only if they improve solution (increase fitness)
- Systematic, only improving - cities are systematically chosen and exchanged only if they improve solution (increase fitness)
- Random improving - the same as "random, only improving", but before this is "normal random" mutation performed
- Systematic improving - the same as "systematic, only improving", but before this is "normal random" mutation performed
- None - no mutation

TSP Example - Permutation Encoding

Crossover

Single point crossover - one crossover point is selected, till this point the permutation is copied from the first parent, then the second parent is scanned and if the number is not yet in the offspring it is added
Note: there are more ways how to produce the rest after crossover point

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

Mutation

Order changing - two numbers are selected and exchanged

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

Recommendations and Other Approaches

Recommendations for Parameters

Recommendations are often results of some empirical studies of GAs:

- **Crossover rate:** Crossover rate generally should be 3-5%.
- **Mutation rate:** On the other side, mutation rate should be very low. Best rates reported are about **1%-3%**.
- **Population size:** It may be surprising, that very big population size usually does not improve performance of GA (in meaning of speed of finding solution). Good population size is about **20-30**, however sometimes sizes 50-100 are reported as best. Some research also shows, that best population size depends on encoding, on **size of encoded string**. It means, if you have chromosome with 32 bits, the population should be say 32, but surely two times more than the best population size for chromosome with 16 bits.
- **Selection:** Basic **roulette wheel selection** can be used, but sometimes rank selection can be better. There are also some more sophisticated method, which changes parameters of selection during run of GA. Basically they behaves like simulated annealing.

Hints

When bit-string representations of integers are used, Gray coding is often employed. Small changes in the integer can be readily effected through mutations or crossovers. This has been found to help prevent premature convergence.

A **Gray code** is a code assigning to each of a contiguous set of integers, or to each member of a circular list, a word of symbols such that each two adjacent code words differ by one symbol.

Dezimal	Binär	Gray-Code
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

Other (Related) Approaches

Hill-climbing

Similar to genetic algorithms, though more systematic and less random, a hill-climbing algorithm begins with one initial solution to the problem at hand, usually chosen at random. The string is then mutated, and if the mutation results in higher fitness for the new solution than for the previous one, the new solution is kept; otherwise, the current solution is retained. The algorithm is then repeated until no mutation can be found that causes an increase in the current solution's fitness, and this solution is returned as the result (Koza et al. 2003, p. 59). (To understand where the name of this technique comes from, imagine that the space of all possible solutions to a given problem is represented as a three-dimensional contour landscape. A given set of coordinates on that landscape represents one particular solution. Those solutions that are better are higher in altitude, forming hills and peaks; those that are worse are lower in altitude, forming valleys. A "hill-climber" is then an algorithm that starts out at a given point on the landscape and moves inexorably uphill.)

Hill-climbing is what is known as a *greedy algorithm*, meaning it always makes the best choice available at each step in the hope that the overall best result can be achieved this way. By contrast, methods such as genetic algorithms and simulated annealing, discussed below, are not greedy; these methods sometimes make suboptimal choices in the hopes that they will lead to better solutions later on.

Other (Related) Approaches

Simulated annealing

The idea borrows its name from the industrial process of *annealing* in which a material is heated to above a critical point to soften it, then gradually cooled in order to erase defects in its crystalline structure, producing a more stable and regular lattice arrangement of atoms (Haupt and Haupt 1998, p. 16).

In simulated annealing, as in genetic algorithms, there is a fitness function that defines a fitness landscape; however, rather than a population of candidates as in GAs, there is only one candidate solution. Simulated annealing also adds the concept of "temperature", a global numerical quantity which gradually decreases over time. At each step of the algorithm, the solution mutates (which is equivalent to moving to an adjacent point of the fitness landscape). The fitness of the new solution is then compared to the fitness of the previous solution; if it is higher, the new solution is kept. Otherwise, the algorithm makes a decision whether to keep or discard it based on temperature. If the temperature is high, as it is initially, even changes that cause significant decreases in fitness may be kept and used as the basis for the next round of the algorithm, but as temperature decreases, the algorithm becomes more and more inclined to only accept fitness-increasing changes. Finally, the temperature reaches zero and the system "freezes"; whatever configuration it is in at that point becomes the solution. Simulated annealing is often used for engineering design applications such as determining the physical layout of components on a computer chip (Kirkpatrick, Gelatt and Vecchi 1983).

Benefits of GAs

1. Inherently parallel
2. Natural low-level coding implementation
3. Multiple parameters are easy to model and measure
4. Complex fitness functions do not impede performance

Difficulties with GAs

1. The language used to specify candidate solutions must be robust; i.e., it must be able to tolerate random changes such that fatal errors or nonsense do not consistently result.
2. The problem of how to write the fitness function must be carefully considered so that higher fitness is attainable and actually does equate to a better solution for the given problem.
3. In addition to making a good choice of fitness function, the other parameters of a GA - the size of the population, the rate of mutation and crossover, the type and strength of selection - must be also chosen with care. If the population size is too small, the genetic algorithm may not explore enough of the solution space to consistently find good solutions.
4. P*remature convergence*: If an individual that is more fit than most of its competitors emerges early on in the course of the run, it may reproduce so abundantly that it drives down the population's diversity too soon, leading the algorithm to converge on the local optimum that that individual represents rather than searching the fitness landscape thoroughly enough to find the global optimum (Forrest 1993, p. 876; Mitchell 1996, p. 167).