Why Genetic Algorithms?

It is better than conventional AI in that it is more robust. Unlike older AI systems, they do not break easily even if the inputs changed slightly, or in the presence of reasonable noise. Also, in searching a large state-space, multi-modal state-space, or n-dimensional surface, a genetic algorithm may offer significant benefits over more typical search of optimization techniques. (linear programming, heuristic, depth-first, breath-first, and praxis.)

Genetic Algorithms Overview

GAs simulate the survival of the fittest among individuals over consecutive generation for solving a problem. Each generation consists of a population of character strings that are analogous to the chromosome that we see in our DNA. 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.

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.

Search Space

A population of individualsare is maintained within **search space** for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of components, or variables, in terms of some alphabet, usually the **binary** alphabet **{0,1}**. To continue the genetic analogy these individuals 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. Individuals in the population die and are replaced by the new solutions, eventually creating a new generation once all mating opportunities in the old population have been exhausted. In this way it is hoped that over successive generations better solutions will thrive while the least fit solutions die out.

New generations of solutions are produced containing, on average, more good genes than a typical solution in a previous generation. Each successive generation will contain more good `partial solutions' than previous generations. Eventually, once the population has converged and is not producing offspring noticeably different from those in previous generations, the algorithm itself is said to have converged to a set of solutions to the problem at hand.

Implementation Details

Based on Natural Selection

After an initial population is randomly generated, the algorithm evolves the through three operators:

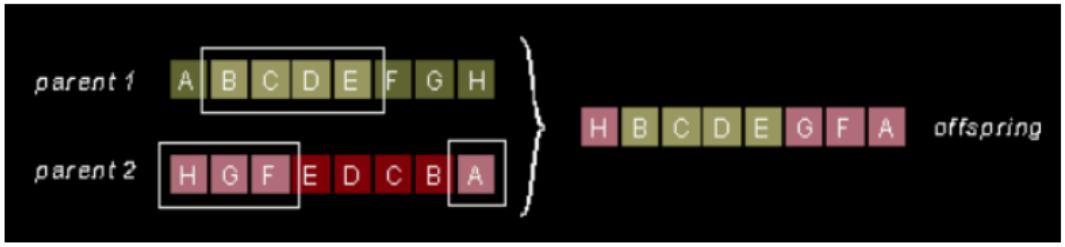
1. **selection** which equates to survival of the fittest;
2. **crossover** which represents mating between individuals;
3. **mutation** which introduces random modifications.

1. Selection Operator

- key idea: give prefrence to better individuals, allowing them to pass on their genes to the next generation.
- The goodness of each individual depends on its fitness.
- Fitness may be determined by an objective function or by a subjective judgement.

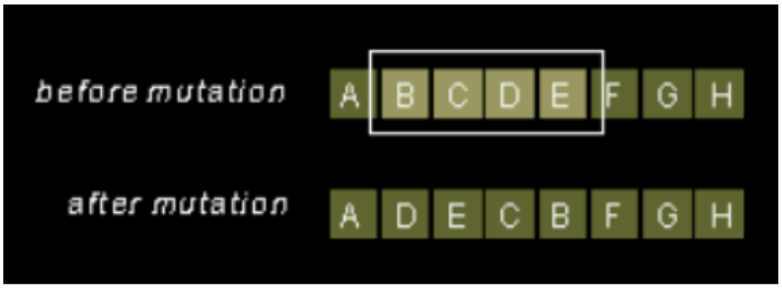
2. Crossover Operator

- Prime distinguished factor of GA from other optimization techniques
- Two individuals are chosen from the population using the selection operator
- A crossover site along the bit strings is randomly chosen
- The values of the two strings are exchanged up to this point
- If S1=000000 and s2=111111 and the crossover point is 2 then S1'=110000 and s2'=001111
- The two new offspring created from this mating are put into the next generation of the population
- By recombining portions of good individuals, this process is likely to create even better individuals



3. Mutation Operator

- With some low probability, a portion of the new individuals will have some of their bits flipped.
- Its purpose is to maintain diversity within the population and inhibit premature convergence.
- Mutation alone induces a random walk through the search space
- Mutation and selection (without crossover) create a parallel, noise-tolerant, hill-climbing algorithms



Effects of Genetic Operators

- Using selection alone will tend to fill the population with copies of the best individual from the population
- Using selection and crossover operators will tend to cause the algorithms to converge on a good but sub-optimal solution
- Using mutation alone induces a random walk through the search space.
- Using selection and mutation creates a parrallel, noise-tolerant, hill climbing algorithm

The Algorithms

1. randomly initialize population(t)
2. determine fitness of population(t)
3. repeat
    1. select parents from population(t)
    2. perform crossover on parents creating population(t+1)
    3. perform mutation of population(t+1)
    4. determine fitness of population(t+1)
4. until best individual is good enough

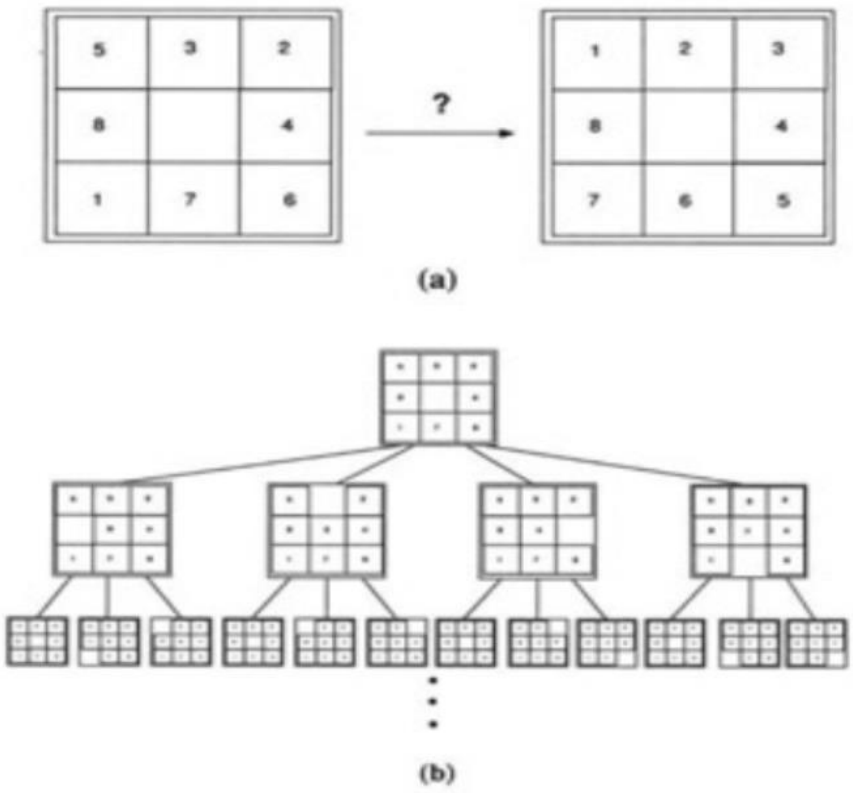
GENETIC ALGORITHMS AND TRADITIONAL SEARCH METHODS

In the preceding sections I used the word "search" to describe what GAs do. It is important at this point to contrast this meaning of "search" with its other meanings in computer science. There are at least three (overlapping) meanings of "search":

Search for stored data Here the problem is to efficiently retrieve information stored in computer memory. Suppose you have a large database of names and addresses stored in some ordered way. What is the best way to search for the record corresponding to a given last name? "Binary search" is one method for efficiently finding the desired record. Knuth (1973) describes and analyzes many such search methods.

Search for paths to goals Here the problem is to efficiently find a set of actions that will move from a given initial state to a given goal. This form of search is central to many approaches in artificial intelligence. A simple example—all too familiar to anyone who has taken a course in AI—is the "8−puzzle," illustrated in figure . A set of tiles numbered 1–8 are placed in a square, leaving one space empty. Sliding one of the adjacent tiles into the blank space is termed a "move." Figure illustrates the problem of finding a set of moves from the initial state to the state in which all the tiles are in order. A partial search tree corresponding to this problem is illustrated in figure 1.2b The "root" node represents the initial state, the nodes branching out from it represent all possible results of one move from that state, and so on down the tree. The search algorithms discussed in most AI contexts are methods for efficiently finding the best (here, the shortest) path in the tree from the initial state to the goal state. Typical algorithms are "depth−first search," "branch and bound," and "A*."

Figure 1: The 8−puzzle. (a) The problem is to find a sequence of moves that will go from the initial state to the state with the tiles in the correct order (the goal state). (b) A partial search tree for the 8−puzzle.



Search for solutions This is a more general class of search than "search for paths to goals." The idea is to efficiently find a solution to a problem in a large space of candidate solutions. These are the kinds of search problems for which genetic algorithms are used.

There is clearly a big difference between the first kind of search and the second two. The first concerns problems in which one needs to find a piece of information (e.g., a telephone number) in a collection of explicitly stored information. In the second two, the information to be searched is not explicitly stored; rather, candidate solutions are created as the search process proceeds. For example, the AI search methods for solving the 8−puzzle do not begin with a complete search tree in which all the nodes are already stored in memory; for most problems of interest there are too many possible nodes in the tree to store them all. Rather, the search tree is elaborated step by step in a way that depends on the particular algorithm, and the goal is to find an optimal or high−quality solution by examining only a small portion of the tree. Likewise, when searching a space of candidate solutions with a GA, not all possible candidate solutions are created first and then evaluated; rather, the GA is a method for finding optimal or good solutions by examining only a small fraction of the possible candidates.

"Search for solutions" subsumes "search for paths to goals," since a path through a search tree can be encoded as a candidate solution. For the 8−puzzle, the candidate solutions could be lists of moves from the initial state to some other state (correct only if the final state is the goal state). However, many "search for paths to goals" problems are better solved by the AI tree−search techniques (in which partial solutions can be evaluated) than by GA or GA−like techniques (in which full candidate solutions must typically be generated before they can be evaluated).

However, the standard AI tree−search (or, more generally, graph−search) methods do not always apply. Not all problems require finding a path