**Beam Search**

Beam search is a heuristic search algorithm that explores a graph by expanding the most optimistic node in a limited set. Beam search is an optimization of *best-first search* that reduces its memory requirements.

Best-first search is a graph search that orders all partial solutions according to some heuristic. But in beam search, only a predetermined number of best partial solutions are kept as candidates. Therefore, it is a *greedy* algorithm.

Beam search uses *breadth-first search* to build its search tree. At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost. However, it only stores a predetermined number (β), of best states at each level called the *beamwidth*. Only those states are expanded next.

The greater the beam width, the fewer states are pruned. No states are pruned with infinite beam width, and beam search is identical to breadth-first search. The beamwidth bounds the memory required to perform the search. Since a goal state could potentially be pruned, beam search sacrifices completeness (the guarantee that an algorithm will terminate with a solution if one exists). Beam search is not optimal, which means there is no guarantee that it will find the best solution.

In general, beam search returns the first solution found. Once reaching the configured maximum search depth (i.e., translation length), the algorithm will evaluate the solutions found during a search at various depths and return the best one that has the highest probability.

The beam width can either be *fixed* or *variable.* One approach that uses a variable beam width starts with the width at a minimum. If no solution is found, the beam is widened, and the procedure is repeated.

Components of Beam Search

A beam search takes three components as its input:

1. A problem to be solved,
2. A set of heuristic rules for pruning,
3. And a memory with a limited available capacity.

The problem is the problem to be solved, usually represented as a graph, and contains a set of nodes in which one or more of the nodes represents a goal. The set of heuristic rules are rules specific to the problem domain and prune unfavorable nodes from memory regarding the problem domain.

The memory is where the *"beam"* is stored, memory is full, and a node is to be added to the beam, the most costly node will be deleted, such that the memory limit is not exceeded.

Beam Search Algorithm

The following algorithm for a beam search, as a modified best-first search, is adapted from Zhang's 1999:

1. beamSearch(problemSet, ruleSet, memorySize)
2. openMemory = new memory of size memorySize
3. nodeList = problemSet.listOfNodes
4. node = root or initial search node
5. Add node to openMemory;
6. while (node is not a goal node)
7. Delete node from openMemory;
8. Expand node and obtain its children, evaluate those children;
9. If a child node is pruned according to a rule in ruleSet, delete it;
10. Place remaining, non-pruned children into openMemory;
11. If memory is full and has no room for new nodes, remove the worst
12. node, determined by ruleSet, in openMemory;
13. node = the least costly node in openMemory;

Uses of Beam Search

A beam search is most often used to maintain tractability in large systems with insufficient memory to store the entire search tree. For example,

* It has been used in many machine translation systems.
* Each part is processed to select the best translation, and many different ways of translating the words appear.
* According to their sentence structures, the top best translations are kept, and the rest are discarded. The translator then evaluates the translations according to a given criterion, choosing the translation which best keeps the goals.
* The first use of a beam search was in the *Harpy Speech Recognition System*, CMU 1976.

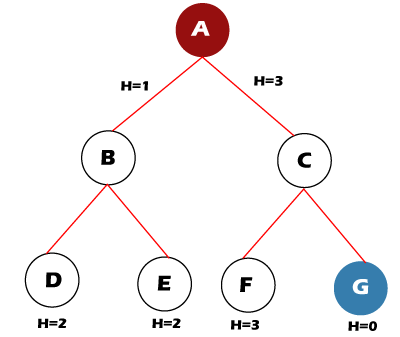
Drawbacks of Beam Search

Here is a drawback of the Beam Search with an example:

* In general, the Beam Search Algorithm is not *complete*. Despite these disadvantages, beam search has found success in the practical areas of *speech recognition, vision, planning,* and *machine learning.*
* The main disadvantages of a beam search are that the search may not result in an optimal goal and may not even reach a goal at all after given unlimited time and memory when there is a path from the start node to the goal node.
* The beam search algorithm terminates for two cases: a required goal node is reached, or a goal node is not reached, and there are no nodes left to be explored.
* A more accurate heuristic function and a larger beam width can improve Beam Search's chances of finding the goal.

AD

For example, let's take the value of ß = 2 for the tree shown below. So, follow the following steps to find the goal node.



Step 1: OPEN= {A}

Step 2: OPEN= {B, C}

Step 3: OPEN= {D, E}

Step 4: OPEN= {E}

AD

Step 5: OPEN= { }

The open set becomes empty without finding the goal node.

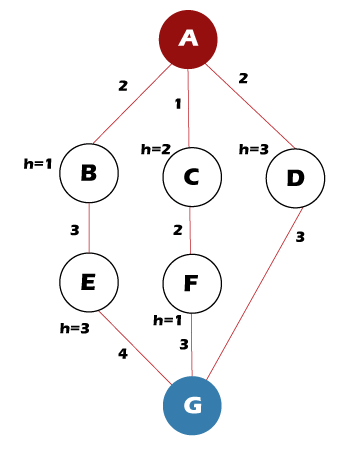
NOTE: But with the value of ß = 3, the algorithm succeeds in finding the goal node.

Beam Search Optimality

The Beam Search algorithm is not complete in some cases. Therefore it is also not guaranteed to be optimal. It can happen because of these reasons:

* The beam width and an inaccurate heuristic function may cause the algorithm to miss expanding the shortest path.
* A more precise heuristic function and a larger beam width can make Beam Search more likely to find the optimal path to the goal.

For example, we have a tree with heuristic values shown below:



Follow the following steps to find the path for the goal node.

AD

Step 1: OPEN= {A}

Step 2: OPEN= {B, C}

Step 3: OPEN= {C, E}

Step 4: OPEN= {F, E}

Step 5: OPEN= {G, E}

Step 6: Found the goal node {G}, now stop.

Path: A-> C-> F-> G

But the *Optimal Path* is A-> D-> G

Time Complexity of Beam Search

The time complexity of the Beam Search algorithm depends on the following things, such as:

* The accuracy of the heuristic function.
* In the worst case, the heuristic function leads Beam Search to the deepest level in the search tree.
* The worst-case time = *O(B\*m)*

*B* is the beam width, and *m* is the maximum depth of any path in the search tree.

Space Complexity of Beam Search

The space complexity of the Beam Search algorithm depends on the following things, such as:

* Beam Search's memory consumption is its most desirable trait.
* Since the algorithm only stores B nodes at each level in the search tree.
* The worst-case space complexity = *O(B\*m)*

*B* is the beam width, and m is the maximum depth of any path in the search tree.

* This linear memory consumption allows Beam Search to probe very deeply into large search spaces and potentially find solutions that other algorithms cannot reach.

Variants of Beam Search

Beam search has been made complete by combining it with *depth-first search*, resulting in *beam stack search* and *depth-first beam search.* With limited discrepancy search, *beam search results in limited discrepancy backtracking* (BULB).

The resulting search algorithms are anytime algorithms that find reasonable but likely sub-optimal solutions quickly, like beam search, then backtrack and continue to find improved solutions until convergence to an optimal solution.

In the context of a local search, we call *local beam search* a specific algorithm that begins selecting β generated states. Then, for each level of the search tree, it always considers β new states among all the possible successors of the current ones until it reaches a goal.

Since local beam search often ends up on local maxima, a standard solution is to choose the next β states in a random way, with a probability dependent on the heuristic evaluation of the states. This kind of search is called *stochastic beam search.*

# Genetic Algorithm in Machine Learning

A genetic algorithm is an adaptive heuristic search algorithm inspired by "Darwin's theory of evolution in Nature." It is used to solve optimization problems in machine learning. It is one of the important algorithms as it helps solve complex problems that would take a long time to solve.

Genetic Algorithms are being widely used in different real-world applications, for example, **Designing electronic circuits, code-breaking, image processing, and artificial creativity.**

In this topic, we will explain Genetic algorithm in detail, including basic terminologies used in Genetic algorithm, how it works, advantages and limitations of genetic algorithm, etc.

## What is a Genetic Algorithm?

Before understanding the Genetic algorithm, let's first understand basic terminologies to better understand this algorithm:

* **Population:** Population is the subset of all possible or probable solutions, which can solve the given problem.
* **Chromosomes:** A chromosome is one of the solutions in the population for the given problem, and the collection of gene generate a chromosome.
* **Gene:** A chromosome is divided into a different gene, or it is an element of the chromosome.
* **Allele:** Allele is the value provided to the gene within a particular chromosome.
* **Fitness Function:** The fitness function is used to determine the individual's fitness level in the population. It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function.
* **Genetic Operators:** In a genetic algorithm, the best individual mate to regenerate offspring better than parents. Here genetic operators play a role in changing the genetic composition of the next generation.
* **Selection**

After calculating the fitness of every existent in the population, a selection process is used to determine which of the individualities in the population will get to reproduce and produce the seed that will form the coming generation.

Types of selection styles available

* **Roulette wheel selection**
* **Event selection**
* **Rank- grounded selection**

AD

So, now we can define a genetic algorithm as a heuristic search algorithm to solve optimization problems. It is a subset of evolutionary algorithms, which is used in computing. A genetic algorithm uses genetic and natural selection concepts to solve optimization problems.

## How Genetic Algorithm Work?

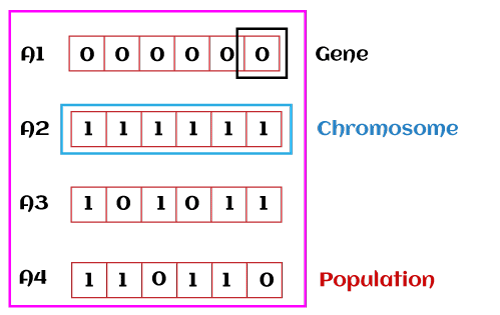
The genetic algorithm works on the evolutionary generational cycle to generate high-quality solutions. These algorithms use different operations that either enhance or replace the population to give an improved fit solution.

It basically involves five phases to solve the complex optimization problems, which are given as below:

* **Initialization**
* **Fitness Assignment**
* **Selection**
* **Reproduction**
* **Termination**

### 1. Initialization

The process of a genetic algorithm starts by generating the set of individuals, which is called population. Here each individual is the solution for the given problem. An individual contains or is characterized by a set of parameters called Genes. Genes are combined into a string and generate chromosomes, which is the solution to the problem. One of the most popular techniques for initialization is the use of random binary strings.



### 2. Fitness Assignment

Fitness function is used to determine how fit an individual is? It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function. The fitness function provides a fitness score to each individual. This score further determines the probability of being selected for reproduction. The high the fitness score, the more chances of getting selected for reproduction.

### 3. Selection

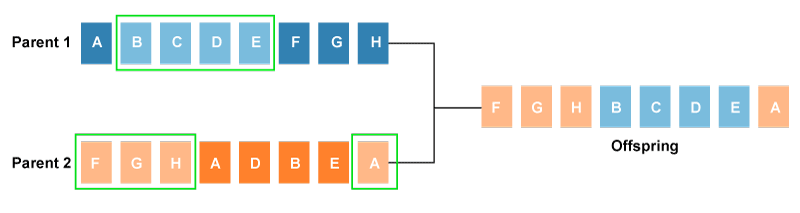
The selection phase involves the selection of individuals for the reproduction of offspring. All the selected individuals are then arranged in a pair of two to increase reproduction. Then these individuals transfer their genes to the next generation.

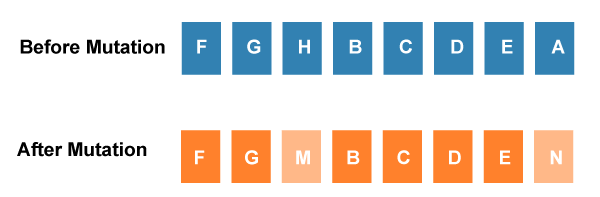
There are three types of Selection methods available, which are:

* Roulette wheel selection
* Tournament selection
* Rank-based selection

### 4. Reproduction

After the selection process, the creation of a child occurs in the reproduction step. In this step, the genetic algorithm uses two variation operators that are applied to the parent population. The two operators involved in the reproduction phase are given below:

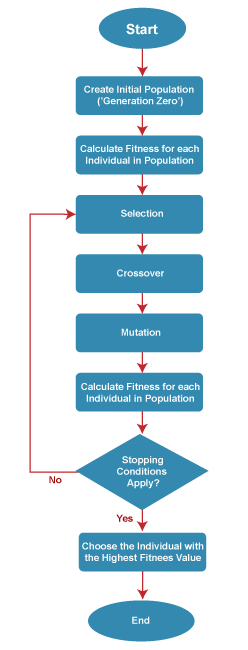
* **Crossover:** The crossover plays a most significant role in the reproduction phase of the genetic algorithm. In this process, a crossover point is selected at random within the genes. Then the crossover operator swaps genetic information of two parents from the current generation to produce a new individual representing the offspring.  
    
  The genes of parents are exchanged among themselves until the crossover point is met. These newly generated offspring are added to the population. This process is also called or crossover. Types of crossover styles available:
  + One point crossover
  + Two-point crossover
  + Livery crossover
  + Inheritable Algorithms crossover
* **Mutation**  
  The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes.  
  Mutation helps in solving the issue of premature convergence and enhances diversification. The below image shows the mutation process:  
  Types of mutation styles available,
  + **Flip bit mutation**
  + **Gaussian mutation**
  + **Exchange/Swap mutation**



### 5. Termination

After the reproduction phase, a stopping criterion is applied as a base for termination. The algorithm terminates after the threshold fitness solution is reached. It will identify the final solution as the best solution in the population.

## General Workflow of a Simple Genetic Algorithm



AD

## Advantages of Genetic Algorithm

* The parallel capabilities of genetic algorithms are best.
* It helps in optimizing various problems such as discrete functions, multi-objective problems, and continuous functions.
* It provides a solution for a problem that improves over time.
* A genetic algorithm does not need derivative information.

## Limitations of Genetic Algorithms

* Genetic algorithms are not efficient algorithms for solving simple problems.
* It does not guarantee the quality of the final solution to a problem.
* Repetitive calculation of fitness values may generate some computational challenges.

## Difference between Genetic Algorithms and Traditional Algorithms

* A search space is the set of all possible solutions to the problem. In the traditional algorithm, only one set of solutions is maintained, whereas, in a genetic algorithm, several sets of solutions in search space can be used.
* Traditional algorithms need more information in order to perform a search, whereas genetic algorithms need only one objective function to calculate the fitness of an individual.
* Traditional Algorithms cannot work parallelly, whereas genetic Algorithms can work parallelly (calculating the fitness of the individualities are independent).
* One big difference in genetic Algorithms is that rather of operating directly on seeker results, inheritable algorithms operate on their representations (or rendering), frequently appertained to as chromosomes.
* One of the big differences between traditional algorithm and genetic algorithm is that it does not directly operate on candidate solutions.
* Traditional Algorithms can only generate one result in the end, whereas Genetic Algorithms can generate multiple optimal results from different generations.
* The traditional algorithm is not more likely to generate optimal results, whereas Genetic algorithms do not guarantee to generate optimal global results, but also there is a great possibility of getting the optimal result for a problem as it uses genetic operators such as Crossover and Mutation.
* Traditional algorithms are deterministic in nature, whereas Genetic algorithms are probabilistic and stochastic in nature.