Search is inherent to the problems and methods of Artificial Intelligence (AI). That is because AI problems are intrinsically complex. Efforts to solve problems with computers which humans can routinely solve by employing innate cognitive abilities, pattern recognition, perception and experience, invariably must turn to considerations of search.

4.1 Introduction
Search for solutions in large problem spaces is a characteristic of almost all Artificial Intelligence problems. Problems are typically defined in terms of states and solutions correspond to goal states. All search algorithms have a problem space which consists of a set of states of the problem and set of operators that can alter the state. Problem space graph is used to represent the problem space where nodes represent the states and edges represent the operators. Searching through a state space represents a primary means of problem solving. The two fundamental qualities of search are complexity and completeness. Complexity refers to the amount of time and space required to search through a given search space. Completeness refers to the likelihood that a search will be successful given that a solution exists. Promising search strategies tend to have balance between complexity and completeness. There are number of solutions available for the problem. Every state is possible solution. Collection of states is called state space (Luger, 2001). A state space may be searched in two directions: from the given data of a problem instance towards a goal or vice versa.

4.2 Search Terminologies
Search terminology includes many definitions but most important terms are defined as below:

- States: places the search can visit.
- Search Space / State Space: The set of possible states.
- Search Path: Sequence of states the agent actually visits.
- Solution: One or more states which solves the given problem.
- Strategy: How to choose the next state in the path at any given state.
4.3 State Space Representation
The first question is how to represent a problem that should be solved by computer. After developing the state space one can create algorithms that work on this representation. A state space representation is a 4-tuple (A,S,G,O), where:

- A: the set of states, $A \neq \emptyset$.
- S: the initial state, $S \in A$.
- G: the set of goal states, $G \subseteq A$.
- O: the set of operators, $O \neq \emptyset$.

4.4 Specifying a Search Problem
To specify a search problem in state space representation following things are required:

- Initial State: The state where the search starts.
- Operators: Functions which can be applied on states and will produce new states.
- Goal States: The states by which the solution will be identified and search can be stopped.

Every search technique will apply operators on the chosen states and search for the goal states in order to solve a particular problem.

4.5 Types of Search Techniques
Search methods include both blind exhaustive methods and informed heuristic methods. Blind search is characterization of all search techniques that are heuristically uninformed. They are commonly known as Brute Force algorithms. They require only a starting state, state description, set of legal operators and the goal state (Korf and Reid, 1998). They do not require domain specific knowledge. In such cases, when no information regarding the problem is available, then lots of exploration of search space has to be done. They run out of space, time or both very quickly in maximum real life problems.

In an informed search, a heuristic function helps to decide which nodes are to be expanded next. The goal of heuristic search is to greatly reduce the number of nodes searched in seeking a goal. Heuristic is operationally effective information on how to direct search in a problem space. Domain specific knowledge is incorporated in heuristic to improve search efficiency. It leads to a solution along the most probable path, omitting the least promising ones. Heuristic search methods are expected to bridge the gap between the completeness of algorithms and their optimal complexity (Romanycia and
Peletier, 1985). Heuristics are criteria for deciding which among a set of alternatives promises to be the best. This approach is related to the Principle of Limited Rationality proposed by Herbert Simon. Time spent evaluating the heuristic function is recovered by reduction in search time. Through knowledge, information, rule’s insights, analogies and simplification, heuristic search aims to reduce the number of objects that must be examined. When a heuristic function is applied on a state, it returns a value indicting how close this state is to the goal. If the heuristic is good, informed search algorithms may dramatically outperform uninformed strategies. They require only little exploration of search space. Comparing the blind and informed strategies and tell which one is better depends on amount of exploration and usage of information.

4.5.1 Blind or Exhaustive or Uninformed Search Techniques
There are two important strategies that fall under the category blind search, namely Depth First Search & Breadth First Search.

4.5.1.1 Depth First Search (DFS)
Depth First Search is a search technique that first visits each node as deeply and to the one side as possible. DFS is a good idea when all partial parts either reach dead ends or become complete paths after a reasonable number of steps. This search advances in depth until the current state cannot be further expanded. Memory consumption of DFS is linear (Weise, 2007). It is neither complete nor optimal. For DFS, time and space complexity is $O(b^d)$ and $O(b^d)$ respectively where $b$ is branching factor and $d$ is depth of tree. Its major drawback is the determination of cut-off depth essential to avoid infinite looping of search procedure. It has low space complexity. DFS shall be used in cases when goal states tend to have large depths since it has been seen that DFS has the tendency to miss short solutions in some cases.

![Figure 4.1: Depth First Search in a Tree](image-url)
Depth Limited Search is DFS technique with depth limit L. Depth-limited search avoids going down an infinite branch by imposing a depth limit that effectively terminates the search at that depth. It is complete if L >= d. The choice of the depth parameter can be an important factor. It is neither complete nor optimal search technique. If maximum depth of possible solutions is known, then it may be sufficient.

4.5.1.2 Breadth First Search (BFS)
BFS is uninformed search technique that proceeds level-by-level visiting all of the nodes at each level before proceeding to the next level. It pushes uniformly into the search tree (Tucker and Allen, 1996). It is complete, since it will always find a solution if there exists one. BFS being an exhaustive search generates all the nodes for identifying the goal. Unlike DFS, BFS has to remember all the paths it has generated so far. Time complexity and space complexity of BFS is O(b^d) and O(b^d) respectively. Its major drawback is large amount of space needed. It is used when priority is given to finding minimal paths to a goal. It may be ineffective for problems whose goal depths are sizable. If the number of expansion steps needed from the origin to a state is a measure for the costs, BFS is optimal. It is an optimal search technique as nodes are expanded in increasing order.

![Breadth First Search in a Tree](image)

**Figure 4.2: Breadth First Search in a Tree**

4.5.2 Informed or Heuristic Search Techniques
Heuristic methods allow us to exploit uncertain and imprecise data in natural way. Heuristic is problem specific knowledge that reduces expected search effort. Heuristic is an approach that promotes discovery by experimentation. Heuristic techniques can be effective if applied correctly on right kind of tasks. The goal of a heuristic search is to reduce the number of nodes searched in seeking a goal. In other words, problems which grow combinatorially large may be approached. Through knowledge, information, rules,
insights, analogies, and simplification in addition to a host of other techniques heuristic search aims to reduce the number of objects examined. Heuristics do not guarantee the achievement of a solution, although good heuristics should facilitate this. Heuristic search is defined by authors in many different ways:

- It is a practical strategy increasing the effectiveness of complex problem solving (Feigenbaum and Feldman, 1963).
- It leads to a solution along the most probable path, omitting the least promising ones (Amarel, 1968).
- It should enable one to avoid the examination of dead ends, and to use already gathered data (Lenat, 1983).

The points at which heuristic information can be applied in a search include:

- Deciding which node to expand next, instead of doing the expansions in a strictly breadth-first or depth-first order.
- In the course of expanding a node, deciding which successor or successors to generate -- instead of blindly generating all possible successors at one time.
- Deciding that certain nodes should be discarded, or pruned, from the search tree.

Bolc and Cytowski (1992) add:

> ... use of heuristics in the solution construction process increases the uncertainty of arriving at a result ... due to the use of informal knowledge (rules, laws, intuition, etc.) whose usefulness have never been fully proven. Because of this, heuristic methods are employed in cases where algorithms give unsatisfactory results or do not guarantee to give any results. They are particularly important in solving very complex problems (where an accurate algorithm fails), especially in speech and image recognition, robotics and game strategy construction. ...

Heuristic methods allow us to exploit uncertain and imprecise data in a natural way. ... The Main objective of heuristics is to aid and improve the effectiveness of an algorithm solving a problem. Most
important is the elimination from further consideration of some subsets of objects still not examined. ...

Most modern heuristic search methods are expected to bridge the gap between the completeness of algorithms and their optimal complexity (Romanycia and Peletier, 1985). Strategies are being modified in order to arrive at a quasi-optimal -- instead of an optimal -- solution with a significant cost reduction (Pearl, 1984).

4.5.2.1 Generate and Test
It is an exhaustive search of problem space that proposes possible solutions randomly and then tests them for their feasibility. It uses technique called split and prune (Tucker and Allen, 1996). It defines a set of legal transformations of a state space from which those that are more likely to bring us closer to a goal state are selected. These algorithms require very little information. They were guaranteed to succeed but had no time boundation. Search in this case is not biased and solution lies in search space. Exhaustive search of problem space is often not feasible or practical due to size of problem space. Good generators are complete and will eventually produce all possible solutions and will not suggest redundant solutions.

4.5.2.2 Hill Climbing
It is a local search algorithm that estimates how far away the goal is. It is a Depth First Search with a heuristic measure that orders choices as nodes are expanded. The heuristic measure is the estimated remaining distance to the goal. The effectiveness of the hill climbing is completely dependent on accuracy of heuristic measure. It is based on the value assigned to states by the heuristic function. Heuristic does not need to be static function of a single state. Heuristic has the capability to look ahead many states.

Hill climbing is associated with certain potential problems. The major problem of hill climbing is that it very easily gets stuck on a local optimum also known as Foothill. It refers to a state where all neighboring states are worse or same. Plateau problem occurs when heuristic measure does not hint toward any significant gradient of proximity to a goal. A plateau is a flat area of the search space in which whole set of neighboring states have the same value. Ridge problem gives impression that search is taking closer to goal state but it’s not so. It is a special kind of local maximum but not the goal state. This algorithm requires more information and specifically local information. The order of operators makes big difference on performance of hill climbing. This method cannot see
past a single move in state space. It is neither complete nor optimal. It is a local search algorithm rather than global Optimization algorithm (Patterson, 1998).

Hill Climbing has various variations.

- Steepest ascent hill climbing considers all moves from current state and selects the best one as the next. It focuses not just climbing to better state but to the steepest slope. The order of operators does not affect its performance.
- In stochastic hill climbing, selection is done randomly among uphill moves and the selection probability can vary with the steepness of the uphill move.
- First choice hill climbing performs stochastic hill climbing by generating successors randomly until a better one is found.
- Random restart hill climbing tries to avoid getting stuck in local maximum.

4.5.2.3 Branch and Bound
It is an algorithmic technique to find the optimal solution by keeping the best solution found so far. If a partial solution cannot improve on the best, it is abandoned. It generates the search tree. If a tree lying at a node of this search tree has a length that exceeds the current lower bound on the optimal tree length, this path of the search tree is terminated and the algorithm backtracks and takes the next available path (Weise, 2007). When a tip of the search tree is reached, the tree is either optimally retained or sub-optimally rejected.

4.5.2.4 Best First Search
It combines the advantages of breadth first search and depth first search. It is similar to steepest ascent hill climbing but does not throw away states that are not chosen. It explores the most promising path seen so far. It is a general algorithm for heuristically searching any state-space graph that includes initial states, intermediate states and goal states. It explores the best state being in a while loop generating all successor states and adding them to the list. It then searches for the best state from the list. This process of searching continues until the goal state is reached. Best First Search is equally applicable to data and goal driven searchers and supports the use of heuristic evaluation functions. It can be used with a variety of heuristics ranging from a state’s goodness to sophisticated measures based on the probability of a state leading to a goal. It is heuristic search technique that finds the most promising node to explore next by maintaining and
exploring and ordered open node list. It does not get caught in loops or dead end paths (Weise, 2007).

4.5.2.5 Greedy Search
It is a best first search where the currently known solution candidate with the lowest heuristic value is investigated next (Weise, 2007). It internally sorts the list of currently known states in descending order according to comparator function. It is neither complete nor optimal as it can get stuck in loops. Time and space complexity for this algorithm is O(b^m). Greedy search is usually performed with a number of random restarts, such that multiple peaks are reached. The highest peak reached over all restarts is presented as the solution.

4.5.2.6 A* Algorithm
A* is an algorithm for finding an optimal solution with minimal effort. It employs an ordered state space search and an estimated heuristic cost to a goal state f*, but is unique in how it defines f* so that it can guarantee an optimal solution path. A* is regarded as branch and bound algorithm augmented by dynamic programming principle. A* is a best first procedure that uses an admissible heuristic estimating function to guide the search process to an optimal solution. The basic idea of A* is to always extend the partial path with the cheapest estimated path length. A* uses estimation function f(s) which is the sum of a heuristic function h(s) that estimates the cost needed to get from s to a valid solution and a function g(s) that computes the costs to find state s. f(s) = g(s) + h(s) where h(s) <=h*(s) where h*(s) is the exact cost of getting from s to goal. A* search proceeds exactly like greedy search, if h* is used instead of plain h. A* search will definitely find a solution if there exists one. A* is optimal if heuristic function h used is admissible.

A* uses modified evaluation function and Best first search. A* minimizes the total path cost and can provide cheapest cost solution in optimal time under right conditions. A* avoid being stuck in loops as it does not expand the same state twice and retains the information about path to goal. A* algorithm combines features of uniform cost search and pure heuristic search to efficiently compute solutions. Value of each state is combination of cost of path to the state and estimated cost of reaching a goal from state. It guides an optimal path to a goal if the heuristic function is admissible. It never overestimates actual cost. Main drawback of A* is its memory requirement because A* is severely space limited in practice and at least the entire open list must be saved (Patterson, 1998).
4.5.2.7 AO* Algorithm
A* algorithm is not adequate for AND/OR tree. AO* is general optimal strategy. It is a variation of original A* algorithm. AO* finds a minimum cost solution tree if one exists.

4.5.2.8 Means End Analysis (MEA)
MEA is state space technique that centers on the detection of differences between current state and goal state and reduce them. MEA is a strategy to control search in problem solving. Given a current state and goal state an action is chosen which will reduce the difference between the two. The action is performed on the current state to produce a new state and process is recursively applied to this new state and the goal state. Determining distance between any state and a goal state can be facilitated by difference procedure tables that depict what the next state might be (Luger, 2001). Means End Analysis allows both backward and forward searching which helps to solve major parts of the problem first and then return to smaller problems when assembling the final solution.

4.5.2.9 Problem Reduction
It solves a complex or larger problem by identifying smaller manageable problems that can be solved in fewer steps like AND/OR trees that tackle games, puzzles and well defined state space goal oriented problems like robot planning. When a problem can be divided into a set of sub-problems where each sub-problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND-OR trees are used for representing the solution. The decomposition of problem or problem reduction generates AND arcs.

4.5.2.10 Constraint Satisfaction (CS)
CS is a search procedure that operates in a space of constraint sets that impose conditions that the variables must satisfy. Heuristics are used to estimate the distance to the goal but to decide what node to expand next. It is a two-step process in which first constraints are discovered and propagated as far as possible throughout the system. Then if there is still no solution, search begins. A guess about something is made and added as new constraint.

4.5.2.11 Simulated Annealing (SA)
SA is based on physical process of annealing a metal to get the minimal energy (best) state. It was first introduced by Metropolis et al. (Metropolis et al., 1953). Metropolis simulated the cooling of material in a heat bath and called it as annealing. Metropolis’s algorithm simulated the material as a system of particles. The algorithm simulates the cooling process by gradually lowering the temperature of the system until it converges to
a steady frozen state. Kirkpatrick et al. applied this simulated annealing to Optimization problems to search for feasible solutions and converge to an optimal solution. Simulated Annealing is a global Optimization algorithm inspired by the manner in which metals crystallize in the process of annealing or in which liquids freeze (Kirkpatrick, Gelatt Jr. and Vecchi, 1983). Simulated annealing is well suited for solving combinatorial Optimization problems. Simulated annealing is empirically much better at avoiding local minima than hill climbing. Unlike hill climbing, it chooses random moves from neighborhood and if the move is better then simulated annealing would take it, otherwise, move would be accepted on some probability. Simulated annealing is a kind of quasi-local algorithm as it relies on sampling the space of models by following along a path from nearest neighbor to nearest neighbor. For its physical foundation, simulated annealing is often applied in physics or chemistry (Starkweather et al., 1991). Simulated algorithms have an associated proof of asymptotic convergence that means that they will actually find the global optimum. Such algorithms are very slow (Lester, 1998). It allows some downhill moves in its search. It initially jumps a lot to explore many regions of state space and gradually the jumping from state to state is reduced. Simulated Annealing can converge to best solution but it may take more time than exhaustive search. Simulated Annealing has the ability of escaping local minima by incorporating a probability function in accepting or rejecting new solutions. A main advantage of the SA method is that it does not need large computer memory. Figure 4.3 shows the flowchart of the process of Simulated Annealing. Pseudo code for Simulated Annealing is as follows:

\[
x = \text{initial random solution}
\]

\[
i = 0
\]

while (global stop condition not satisfied)  
    begin  
        \[ t = \text{coolingschedule}(i) \]  
        \[ i = i+1 \]  
        while (local condition not satisfied)  
            begin  
                \[ x1 = \text{pickrandomneighbour}(x) \]  
                \[ \Delta E = F(x1) - F(x) \]  
                if \( \Delta E < 0 \) then  
                    \[ x = x1 \]

```
else

\[ r = \text{random uniform number between } (0,1) \]

if \( r < e^{-\Delta E/(k-t)} \) then

\[ x = x_1 \]

endif

display solution

---

**Figure 4.3:** Flowchart of Simulated Annealing algorithm

4.5.2.12 Tabu Search

Search techniques are exploratory in nature. In order to improve efficiency of exploration, local information and information related to exploration process must be tracked. Tabu search is based on local search (Glover and Laguna, 1998). The word Tabu comes from Tongan-language of Polynesia, where it is used to indicate things that cannot be touched.
because they are sacred. Tabu search is a higher level heuristic procedure for solving Optimization problems to escape the trap of local optimality. Tabu search has obtained optimal and near optimal solutions to a wide variety of classical and practical problems. It improves the search performance and decreases the probability of getting stuck at local optima by using internal memory. The basic concept of a Tabu search is a meta-heuristic superimposed on another heuristic. Tabu Search begins by marching to local minima. To avoid retracing the steps used, the method records recent moves on one or more Tabu lists. The Tabu lists are historical in nature and form the Tabu search memory. The role of memory can change as the algorithm proceeds. The length of the Tabu list is limited to $n$ and therefore the list must be truncated surpassing that maximum length (Weise, 2007). Tabu search must incorporate adaptive memory and responsive exploration in order to solve the problem intelligently. Adaptive memory of Tabu search aids in searching the problem space economically and effectively. Responsive exploration integrates the principles of intelligent search by exploiting good solution features while exploring new promising regions. The memory structures in Tabu search operate by reference to four principle dimensions – recency, frequency, quality and influence. The core of Tabu search is embedded in its short term memory process. Tabu search has found solutions superior to the best previously obtained by alternative methods. Pseudo code for Tabu search is as follows:

$x$ is an array to store the Tabu list and $m$ is number of elements in $x$

for $i = 1$ to $m$

begin

$x(i) = 0$

end

repeat

repeat

find best facility $j$ to open/close according to local search

until facility $j$ is not Tabu

if $x(j) = 0$ then

$x(j) = 1$

else

$x(j) = 0$

endif

make $j$ Tabu
update frequencies
until no solution improvement for a fixed number of iterations
open facility f with lowest frequency
make f Tabu
until stopping condition

4.6 Characteristics of Search Algorithms
Each of the problem solving methods will differentiate from each other in the composition of their databases, in their operations and the functioning of their controllers. These differences will result in methods with different features. Some of these features are discussed as below:

- Completeness: A search algorithm is said to be complete if it is guaranteed to terminate with a solution whenever a solution exists.
- Admissibility: A search algorithm is said to be admissible it is guaranteed to terminate with an optimal solution whenever a solution exists.
- Dominance: If A and B are two admissible algorithms with heuristic functions h_1 and h_2, then A is said to be more informed than B whenever h_1(n) > h_2(n) and Algorithm A is said to dominate algorithm B.
- Optimality: Algorithm A is optimal over a class of algorithms if A dominates all the members of the class. In other words, if a problem has more than one solution, does the problem-solving method produce the solution with the lowest cost?
- Monotonicity: A search algorithm is said to be monotone if it uses best first search with f(n)=h(n)+g*(n) where h(n_i)-h(n_j) is no larger than actual cost of moving from n_i to n_j and hg=0 for every g e G (Patterson, 1998).

<table>
<thead>
<tr>
<th>SEARCH</th>
<th>COMPLETE</th>
<th>OPTIMAL</th>
<th>ADMISSIBLE</th>
<th>TIME</th>
<th>SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>b&quot;d</td>
<td>b*d</td>
</tr>
<tr>
<td>Depth Limited</td>
<td>YES, if l&gt;=d</td>
<td>NO</td>
<td>b!l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFS</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>b*d</td>
<td>b^d</td>
</tr>
<tr>
<td>Iterative Deepening</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>b^d</td>
<td>b^d</td>
</tr>
<tr>
<td>Hill Climbing</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A*</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>Exponential with path length</td>
<td>Keeps all nodes in memory</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of Different Search Techniques

SEARCH ALGORITHMS
4.7 Genetic Algorithm (GA)

Charles Darwin has formulated the fundamental principle of natural selection as the main evolutionary tool. He formulated this principle without the knowledge of basic hereditary principles. In 1865, Gregor Mendel discovered these hereditary principles by the experiments he carried out on peas. After Mendel’s experiments genetic was developed. In *The Origin of Species*, Charles Darwin stated the theory of natural evolution.

GA’s are adaptive search algorithms based on the evolutionary ideas of natural selection and genetics. The first glimpses of the ideas underlying GA are found in Holland’s papers in the early 1960s (Holland, 1962). In them Holland set out a broad and ambitious agenda for understanding the underlying principles of adaptive systems—systems that are capable of self-modification in response to their interactions with the environments in which they must function. Such a theory of adaptive systems should facilitate both the understanding of complex forms of adaptation as they appear in natural systems and our ability to design robust adaptive artifacts. As such GA represents an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random; instead they exploit historical information to direct the search into the region of better performance within the search space. The basic technique of GA is designed to simulate processes in natural systems necessary for evolutions, specifically those that follow the *survival of the fittest*. As in nature, competition among individuals for scarce resources results in the fittest individuals dominating over the weaker ones. It samples paths from the search space and chooses new paths from the fit paths already considered. It simultaneously examines and manipulates a set of possible solutions. It is not guaranteed to find the global optimum solution to a problem but is good at finding acceptable good solutions to problems. It is not completely blind search technique and not even completely hill climbing technique. It is a combination of both. It analyses the information and evaluates the solution for its optimality and considers the solution only when it is found to be optimal and better than the previous situation. It can be a type of hill climbing or it can be much more effective depending on the operators used. These operators are heuristics of a sort. GA is better than conventional algorithms in that it is more robust.

GA’s are probabilistic search approach based on ideas of evolutionary process (Chakraverty, Ravikumar and Choudhuri, 2002). Due to probabilistic development of solution, genetic algorithm does not guarantee optimality. They are likely to be close to
global optimum. This probabilistic nature of the solution is also the reason they are not contained by local optima. Genetic algorithm is a search procedure that uses random choice as a tool to guide a highly exploitative search through a coding of parameter space (Goldberg, 1989).

4.7.1 Biological Background

The science that deals with the mechanism responsible for similarities and differences in a species is called genetics. The word genetic is derived from the Greek word genesis meaning “to grow” or “to become”. The concepts of GA are directly derived from natural evolution. The terminologies involved in the biological background are as under:

Cell: Every human/animal cell is a complex of many small factories that work together. The centre of it is the cell nucleus. The genetic information is contained in cell nucleus.

Chromosome: All the genetic information gets stored in the chromosomes. Each chromosome is build of deoxyribonucleic acid (DNA). In humans, chromosomes exist in pairs (23 pairs). The chromosomes are divided into several parts called genes. Genes code the properties of species, in other words the characteristics of an individual. The possibilities of combination of the genes for one property are called alleles. A gene can take different alleles. For example, there is a gene for eye color which can take possible values as black, brown, blue and green. The set of all possible alleles present in a particular population forms a gene pool. This gene pool can determine all the different possible variations for the future generations. The set of all the genes of a specific species is called genome. Each gene has a unique position on the genome called locus. In fact, most of living organisms store their genome on several chromosomes but in GA all the genes are usually stored on the same chromosomes. Thus chromosome and genome are synonyms with one other in case of GA.

Genetics: For a particular individual the entire combination of genes is called genotype. The phenotype describes the physical aspect of decoding a genotype to produce the phenotype. One interesting point of evolution is that selection is always done on the phenotype whereas the reproduction recombines genotypes. In higher life forms, chromosomes contain two sets of genes known as diploids. In the case of conflicts between two values of the same pair of genes, the dominant one will determine the phenotype whereas the other one called recessive will still be present and can be passed onto the offspring. Diploidy allows a wider diversity of alleles. Most GAs concentrate on
Haploid as it is easy to construct and the decision of dominant and recessive can be avoided.

Reproduction: reproduction of species is carried out by two ways Mitosis and Meiosis. In Mitosis the same genetic information is copied to new offspring. No exchange of information is happened. Meiosis forms the basis sexual reproduction. In this two gametes appear in the process and these gametes conjugate to a zygote which becomes the new individual. In other words, in meiosis the genetic information is shared between parents in order to create new offsprings.

Natural Selection: The Origin of Species based on “Preservation of favorable variations and rejection of unfavorable variations.” The variation refers to the difference shown by the individuals and the offspring’ of the species. There are more individuals born than can survive, so there is a continuous struggle for life. Individuals with an advantage have more chances to survive called survival of the fittest. For example, Giraffe with long necks have food from tall trees as well from the ground; on the other hand, goat and deer having smaller neck can have food only from the ground. Table 4.2 gives a list of different expressions which are common in natural evolution and GA.

<table>
<thead>
<tr>
<th>Natural Evolution</th>
<th>Genetic Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome</td>
<td>String</td>
</tr>
<tr>
<td>Gene</td>
<td>Feature or character</td>
</tr>
<tr>
<td>Allele</td>
<td>Value of the feature</td>
</tr>
<tr>
<td>Locus</td>
<td>Position of string</td>
</tr>
<tr>
<td>Genotype</td>
<td>Structure or coded string</td>
</tr>
<tr>
<td>Phenotype</td>
<td>Decoded structure</td>
</tr>
</tbody>
</table>

4.7.2 Terminology of Genetic Algorithm
Various terms used in GA are described as below:

Individuals: An individual is a single solution. It groups together two forms of solutions. Genotype is the raw genetic information that the GA deals with and Phenotype is the expressive of the chromosome in the terms of the model. So individual in GA is the synonym of chromosome in natural evolution.
Genes: Genes are the basic material for building a GA. These may describe a possible solution to the problem without actually being the solution. Genotype and phenotype are created with the help of genes in chromosomes. In other words, creating genotype is defined as encoding a solution set into a chromosome and creating phenotype is defined as decoding a chromosome to a solution set.

Fitness: The fitness of individual in GA is the value of an objective function for its phenotype. So for calculating the fitness, the chromosome is first decoded into phenotype and the objective function has to be evaluated on this phenotype. Fitness indicates how good the solution is and how close is it to the optimal one. In case of single objective function, fitness function is simple in comparison to the multi-objective optimization where fitness function is definitely more difficult to determine.

Populations: A population is a collection of individuals. A population consists of a number of individuals being tested, the phenotype parameters defining the individuals and some information about the search space. The main aspects related to population are generation of initial population and the size of population. Generally the initialization of population is done randomly, and size of population is decided by the complexity of the problem. Sometimes a kind of heuristic can be used to seed the initial population. Thus the mean fitness of the population is already high and it may help the GA to find good solutions faster. This approach is tried on set of problems and explained in chapter 7. The size of population has its own considerations like the larger the population is, the easier it is to explore the search space. However it has been established that the time required by the GA to converge is O(n log(n)) function evaluations where n is the population size. The meaning of converge is that all the individuals are very much alike. Goldberg also shows that efficiency of GA to reach global optimum is largely determined by the size of the population.

4.7.3 Simple Genetic Algorithm
The simple form of GA is given by the following:

I. Start with a randomly generated population.
II. Calculate the fitness of each chromosome in the population.
III. Repeat the following steps until n offsprings have been created:
    a. Select a pair of parent chromosomes from the current population.
    b. With probability \( p_c \) crossover the pair to form two offsprings.
c. Mutate the offspring’s at each locus with probability \( p_m \).

IV. Replace the current population with the new offspring’s.

V. If termination criterion then exit else Go to step II.

This Simple GA can also be summarized by the flow chart depicted in figure.

![Flowchart of Simple Genetic Algorithm](image)

**Figure 4.4: Flowchart of Simple Genetic Algorithm**

### 4.7.3 Termination Criterion for Genetic Algorithm

The termination or convergence criterion finally brings the search to a halt state. Various stopping condition are available for stopping the GA simulation. Some of them are as below:

- Maximum Generations: The GA stops when the specified number of generations has evolved.
- Elapsed Time: The GA stops when a specific time has elapsed.
- No change in fitness: The GA will end if there is no change to the populations best fitness for a specified number of generations.
- Stall generations: The GA stops if there is no improvement in the fitness for a sequence of consecutive generations of length “stall generations”.
- Best individual: The GA stops once the optimum fitness in the population meet the convergence value. It brings the search to faster convergence and guaranteeing at least one good solution.
- Worst individual: The GA stops when the least fit individual in the population have fitness worse than the convergence criterion.
- Sum of Fitness: The GA stops when the sum of the fitness in the entire population is less than or equal to the convergence value in the population record. It will guarantee that virtually all the individuals will be within a particular fitness range.

4.7.4 Operators in GA

4.7.4.1 Encoding

Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees or any other objects. The encoding mainly depends on problem to be optimized. Although GAs typically use a bit string representation, GAs are not restricted to bit strings. A number of early proponents of GAs developed GAs that use other representations, such as real-valued parameters (Davis, 1991) (Janikow and Michalewicz, 1991) (Wright, 1991), permutations (Davis, 1985) (Goldberg and Lingle, 1985) (Grefenstette et al., 1985), and treelike hierarchies (Antonisse and Keller, 1987). Koza’s genetic programming (GP) paradigm (Koza, 1992) is a GA-based method for evolving programs, where the data structures are LISP S-expressions, and crossover creates new LISP S-expressions (offspring) by exchanging sub trees from the two parents.

Various Encoding schemes are listed as below:

Binary Encoding: The most common way to encode the individuals. Each chromosome encodes a binary string. Each bit can represent some characteristics of the solution. The way bit strings can code differs from problem to problem. Example chromosomes in binary encoding:

Chromosome 1: 0 0 1 1 1 0 0 1 1 0

Chromosome 2: 1 0 0 1 0 0 1 1 1 0
Holland (Holland, 1975) gave a theoretical justification for using binary encodings. He compared two encodings with roughly the same information–carrying capacity, one with a small number of alleles and long strings (e.g., bit strings of length 100) and the other with a large number of alleles and short strings (e.g., decimal strings of length 30). He argued that the former allows for a higher degree of implicit parallelism than the latter, since an instance of the former contains more schemas than an instance of the latter ($2^{100}$ versus $2^{30}$). Based on it two principles are suggested (Goldberg, 1989): -

- **Principle of minimal alphabets**: The smallest alphabets should be used for encoding that permits a natural expression of the problem, as to increase the similarity pattern. It means that by reducing the cardinality of alphabets, one can increase the potential solutions.

- **Principle of meaningful building blocks**: The user should select a coding so that the schemata should be short, of low order and relatively unrelated to schemata over other fixed positions, still relevant to the underlying problem.

**Octal Encoding**: This encoding uses string made up of octal numbers (0-7) instead of binary number. Examples are: -

Chromosome 1: 0 4 6 4 2 3 5 7 2 1  
Chromosome 2: 1 4 3 6 7 5 2 4 5 3

**Hexadecimal Encoding**: This encoding uses string made up of hexadecimal numbers (0-9, A-F). Examples are: -

Chromosome 1: 9 C B 6 5  
Chromosome 2: A 7 B 4 3

**Value Encoding**: Every chromosome is a string of values and these values can be anything connected to the problem. Direct value encoding can be used in problems, where some complicated values, such as real numbers are used. In this, the values can be real numbers, characters or some complicated objects. This scheme required new crossover operators to be designed as these values are problem specific. Examples are:

Chromosome 1: 1.54 3.46 7.24 18.45  
OR
Chromosome 1: ABCDGHIFGHEYHDJEJKB

OR

Chromosome 1: (right), (back), (forward), (back), (back), (jump)

Tree Encoding: This encoding is mainly used for evolving program expressions for genetic programming. Each chromosome is a tree of some objects such as functions commands of a programming language.

4.7.4.2 Selection
Selection is the process of choosing parents from the population for crossing. After encoding one has to think about how to select better individuals that will create offspring’s for next generations. According to Darwin’s *Survival of the fittest* the best one survive to create new offspring’s. Various selection operators are discussed in more detail in chapter 5.

The convergence rate of GA is largely determined by the magnitude of selection pressure, with higher selection pressures resulting in higher convergence rates. However, if the selection pressure is too low, then GA will take unnecessarily longer to find the optimal solution. In addition to selection pressure, selection schemes should also preserve the diversity of population to avoid the premature convergence.

Selection has to be balanced with variation from crossover and mutation. Too strong selection means sub-optimal highly fit individuals will take over the population, reducing the diversity needed for change and progress; too weak selection will result in too slow evolution.

4.7.4.3 Crossover (Recombination)
Crossover is the process of taking parent solutions and producing from them new child. After the selection, mating pool is enriched with better individuals. Various crossover operators are discussed in chapter 6 with the proposed crossover operators.

The intuitive idea behind crossover is easy to state: given two individuals who are highly fit, but for different reasons, ideally a new individual is created that combines the best features from both parents. Of course, since it is not known that which features account for the good performance (if it is known then there is no need of a search algorithm), the best one can do is to recombine features at random. This is how crossover operates. It treats these features as building blocks scattered throughout the population and tries to recombine them into better individuals via crossover. Sometimes crossover will combine
the worst features from the two parents, in which case these children will not survive for long. But sometimes it will recombine the best features from two good individuals, creating even better individuals, provided these features are compatible.

The basic parameter in crossover technique is the crossover probability \( (P_c) \). This parameter will describe how often crossover will be performed. If there is no crossover then offspring is exact copy of the parents. If there is crossover then offspring’s are made from parts of the parents. So this parameter varies from 0% to 100%. Crossover is done with the intention that good parents will be combined to produce more better childs, but it is good to leave some part of old population survive to next generation, so a crossover probability near about 80% to 90% is commonly used.

**4.7.4.4 Mutation**

After Crossover, the strings are subjected to mutation. It prevents the algorithm to be trapped in local optima. It plays the role of recovering the lost genetic material. Traditionally mutation is termed as simple search operator. There are many different kinds of mutation for the different kinds of encodings like flipping any bit, inverting a substring part from the string, exchanging any two bits etc.

Although most GAs use mutation along with crossover, mutation is sometimes treated as if it were a background operator for assuring that the population will consist of a diverse pool of alleles that can be exploited by crossover. For many optimization problems, however, an evolutionary algorithm using mutation without crossover can be very effective (Mathias and Whitley, 1994). This is not to suggest that crossover never provides an added benefit, but only that one should not criticize mutation.

A parameter in mutation technique is mutation probability \( (P_m) \). It decides how often mutation is carried out. If there is no mutation then same offspring is returned. And if there is mutation then one or more parts of the chromosome is changed. If mutation probability is 100% then whole chromosome is changed. Mutation should not occur very often, because then GA will in fact turn to random search.

**4.7.4.5 Replacement**

After Reproduction of chromosomes decision is made to replace the current population from the new ones. Two common approaches used are \( (\mu+\lambda) \) and \( (\mu, \lambda) \) (Back, Hoffmeister and Schwefel, 1991). Where first one is defined as, From \( \mu \) parents \( \lambda \) offspring are produced; the \( \mu \) parents and \( \lambda \) offspring are merged; and the best \( \mu \)
individuals are chosen to form the new parent population. The other Replacement method, places all the bias in the child selection stage. In this case, $\mu$ parents produce $\lambda$ offspring ($\lambda > \mu$), and the best $\mu$ offspring are chosen to replace the parent population.

4.8 Comparing Genetic Algorithm with Other Search Techniques

Since, according to the no-free-lunch (NFL) theorem (Wolpert and Macready, 1996), there cannot exist any algorithm for solving all (e.g. optimization) problems that is generally (on average) superior to any competitor, the question of whether Genetic Algorithm is inferior/superior to any alternative approach is senseless. What could be claimed solely is that GA behaves better than other methods with respect to solving a specific class of problems—with the consequence that they behave worse for other problem classes.

The NFL theorem can be corroborated in the case of GA versus many classical optimization methods insofar as the latter are more efficient in solving linear, quadratic, strongly convex, unimodal, separable, and many other special problems. On the other hand, GA does not give up so early when discontinuous, nondifferentiable, multimodal, noisy, and otherwise unconventional response surfaces are involved. Their effectiveness (or robustness) thus extends to a broader field of applications, of course with a corresponding loss in efficiency when applied to the classes of simple problems classical procedures have been specifically devised for.

On comparing various search techniques used in AI with GA, it can be concluded that GAs are flexible and robust as a global search method. They are good at taking potentially huge search spaces and navigating them, looking for optimal solution. They are directed search algorithms based on mechanics of biological evolution. They can deal with highly nonlinear problems and non-differentiable functions as well as functions with multiple local optima. They are also readily amenable to parallel implementation, which renders them usable in real-time. They can be used in both unconstrained and constrained Optimization problems and can be applied to non-linear programming, stochastic programming, and combinatorial Optimization problems. It works well for global Optimization especially where the objective function is discontinuous or with several local minima. Multi-point search can be done in GAs unlike other search techniques. These advantages lead to potential disadvantages also. Since it does not use extra
information such as gradients, the Genetic Algorithm has a slow convergence rate on well-behaved objective functions.

The best one can say about GAs, therefore, is that they present a methodological framework that is easy to understand and handle, and is either usable as a black-box method or open to the incorporation of new or old recipes for further sophistication, specialization or hybridization. They are applicable even in dynamic situations where the goal or constraints are moving over time or changing, either exogenously or self-induced, where parameter adjustments and fitness measurements are disturbed, and where the landscape is rough, discontinuous, multimodal, even fractal or cannot otherwise be handled by traditional methods, especially those that need global prediction from local surface analysis.

4.9 Summary
Solving Search problems is a fundamental methods in AI. Search problems are either blind or informed. Genetic algorithm is a search technique that mimics nature and evolution process. Traditional search techniques make local moves around a single solution in order to sample the search space. In genetic algorithms, moves between one population and the next under selection are not determined by a local sampling procedure. The important difference between the GA and traditional search heuristics is the use of a crossover operator, which produces new individual solutions by mixing existing parent solutions. Crossover allows non-local moves within the population, because offspring may have very different combinations from either parent. Genetic algorithms can be categorized as search technique but it is powerful technique than traditional ones.