

Informed Search Algorithms

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

What are Heuristic Search Algorithms?

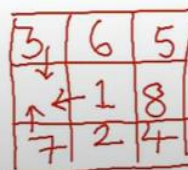
- The **Heuristic** is any rule/device that is often effective but will not guarantee work in every case.
- **Heuristic search** refers to a search strategy that attempts to optimize a problem by iteratively improving the solution based on a given **heuristic function** or a cost measure. A classic example of applying **heuristic search** is the **traveling salesman problem**.
- A **Heuristic** is a technique to **solve a problem faster** than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed. A Heuristic (or a heuristic function) takes a look at each branching step, and **evaluates the available information** and makes a decision on which branch to follow. It does so by **ranking alternatives**.

Characteristics of Heuristic Search

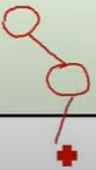
- ✓ Heuristics are knowledge about domain, which help search and reasoning in its domain.
- ✓ Heuristic search incorporates domain knowledge to improve efficiency over blind search. +
- Heuristic is a function that, when applied to a state, returns value as estimated merit of state, with respect to goal.
 - Heuristics might (for reasons) underestimate or overestimate the merit of a state with respect to goal.
 - Heuristics that underestimate are desirable and called admissible.
- Heuristic evaluation function estimates likelihood of given state leading to goal state.
- Heuristic search function estimates cost from current state to goal, presuming function is efficient.

Some of the Heuristic Functions

- ✓ • 8-Puzzle : Number of misplaced tiles
- TSP : Sum of the distance so far
Nearest neighbor
- Tic-Tac-Toe : Most wins
- Chess : Material advantage of our side over the opponent



Compare: Blind Search with Heuristic Search

Blind Search/Brute Force	Heuristic Search
Can only <u>search</u> what it has knowledge about	Estimates ' <u>distance</u> ' to goal state through <u>explored nodes</u>
No knowledge about how far a node from goal state	Guides search process towards goal
	Prefers states (nodes) that lead close to and not away from goal state


Features of Hill Climbing Search

- ✓ It is a **variant** of the generate-and-test algorithm.


- It makes use of the **greedy approach**.
- Hill Climbing is a technique to solve certain **optimization problems**. In this technique, we start with a sub-optimal solution and the solution is improved repeatedly until some condition is maximized.
- The idea of starting with a sub-optimal solution is compared to starting from the base of the hill, improving the solution is compared to walking up the hill, and finally maximizing some condition is compared to reaching the top of the hill.
- Hill Climbing technique is mainly used for **solving computationally hard problems**. It looks only at the current state and immediate future state. Hence, this technique is memory efficient as it does not maintain a search tree.



Hill Climbing Search Algorithm

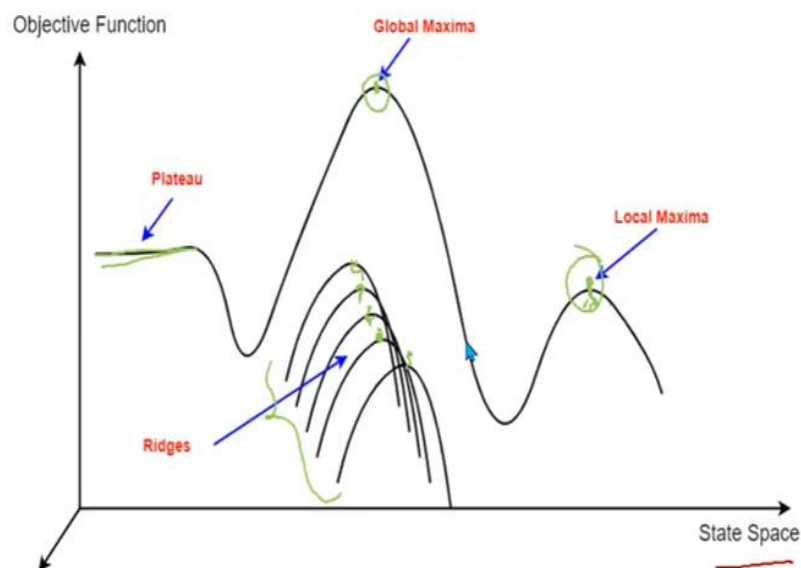
- 1) **Evaluate initial state**- If goal state, stop and return success. Else, make initial state  current.
- 2) **Loop until** the solution reached or until no new operators left to apply to current state:
 - a) **Select** new operator to apply to the current producing new state.
 - b) **Evaluate** new state:
 - If a goal state, stop and return success.
 - If better than the current state, make it current state, proceed.
 - Even if not better than the current state, continue until the solution reached.
- 3) **Exit.**

Steepest-Ascent Hill Climbing Search Algorithm

- 1) **Evaluate initial state**- If goal state, stop and return success. Else, make initial state current state.
- 2) **Loop until** the solution reached or a complete iteration produces no change to the current state:
 - a) Let SUCC be a state such that any possible successor of the current state will be better than SUCC.
 - b) **For each operator** that applies to the current state do:
 - **Apply** the operator and generate the new state..
 - **Evaluate** the new state. If it is a goal state then return and quit. If not, compare it to SUCC. If it  better, then set SUCC to this state. If it is not, then leave SUCC alone.
 - c) If the SUCC is better than current state, then set current state to SUCC. .

Problems with Hill Climbing Search

- **Local Maximum**- All neighboring states have values worse than the current. The greedy approach means we won't be moving to a worse state. This terminates the process even though there may have been a better solution. As a workaround, we use backtracking.
- **Plateau**- All neighbors to it have the same value. This makes it impossible to choose a direction. To avoid this, we randomly make a big jump.
- **Ridge**- At a ridge, movement in all possible directions is downward. This makes it look like a peak and terminates the process. To avoid this, we may use two or more rules before testing.



Solutions to the Problems

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

- ✓ • **Local maximum** : At a local maximum all neighboring states have a values which is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.

To overcome local maximum problem : Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

- ✓ • **Plateau** : On plateau all neighbors have same value . Hence, it is not possible to select the best direction.

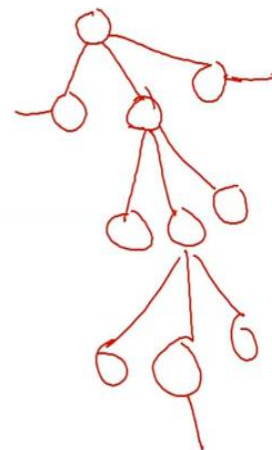
To overcome plateaus : Make a big jump ➡ Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region

- **Ridge** : Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.

To overcome Ridge : In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

Analysis of Hill Climbing Search

- | | | |
|----------------------|---|--------------------------------------|
| ✓ • Time Complexity | - | <u>$O(d)$ i.e. Linear</u> |
| ✓ • Space complexity | - | Constant |
| ✓ • Optimal | - | No |
| • Complete | - | No |



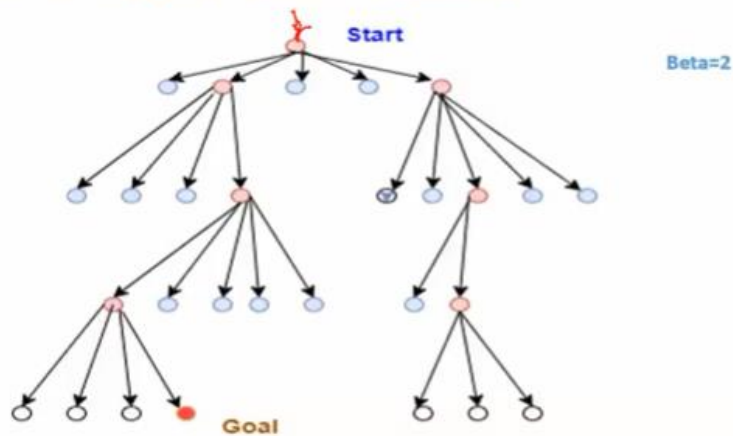
Beam Search

- Beam search is a **heuristic search** algorithm that explores a graph by expanding the most promising 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 which orders all partial solutions (states) according to some heuristic. But in beam search, only a predetermined number of best partial solutions are kept as candidates. It is thus a greedy algorithm.

Beam Search 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 beam width). Only those states are expanded next. The greater the beam width, the fewer states are pruned.
- If any one of the successor state is a goal node, beam search returns the **first solution** found.
- 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.

Working of Beam Search




Analysis of Beam Search

- Time Complexity - $O(\beta m)$ (linear even in worst case)
- Space complexity - $O(\beta m)$ (linear even in worst case)
- Optimal - No (may not find the best solution)
- Complete - No (goal state may be pruned)

Analysis of Beam Search

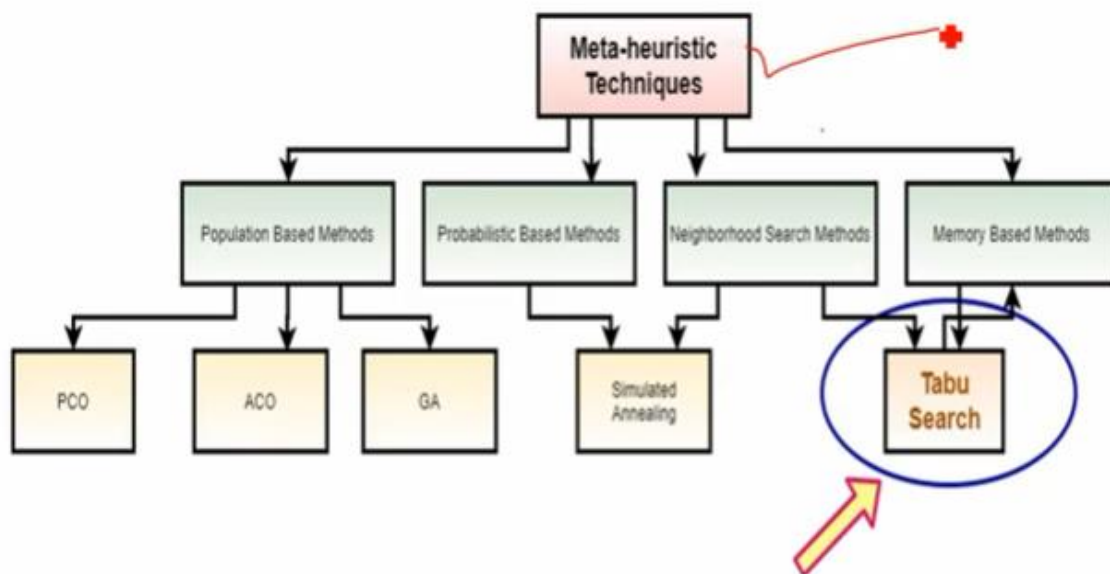
- A more **accurate heuristic function** and a **larger beam width** can **improve** Beam Search's chances of **finding the goal**.
- A more **precise heuristic function** and a **larger beam width** can make Beam Search more likely to find the **optimal path** to the goal.

Advantages

- ✓ Linear memory consumption allows Beam Search to probe very deeply into large search spaces and potentially **find solutions that other algorithms** cannot reach. 
- The **speed** with which this algorithm executes is one of its **greatest strengths**.

Applications

- A beam search is most often used to maintain tractability in **large systems** with insufficient amount of memory to store the entire search tree.
- The first use of a beam search was in the **Harpy Speech Recognition System**, CMU 1976.
- It has been used in many **machine translation systems**. Now a days, beam search is the most popular search strategy for the sequence to sequence Deep NLP algorithms like Neural Machine Translation, Image captioning, Chatbots, etc.
- To select the best translation, each part is processed, and many different ways of translating the words appear. The **top best translations** according to their sentence structures 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.



Tabu Search

- Fred Glover proposed Tabu Search (TS) in 1986, to allow Local Search (LS) methods to **overcome local optima**.
- TS can be considered as a **generalization** of iterative improvements like **Simulated Annealing** (SA). It is an adaptive procedure having the ability to use many methods, such as linear programming algorithms and specialized heuristics, which it guides to overcome the limitations of local optimality.
- TS is based on concepts that can be used in both **artificial intelligence** and optimization fields.
- TS has **memory property** that distinguishes it from other search designs.

Tabu List

- TS uses memory to keep track of solutions previously visited so that it can prevent revisiting that solution (to avoid cycles), called as **Tabu list** (Short term memory).
- The **length of the memory** (tabu list) control the search process.
- A **high length** of the tabu list will explore the larger regions and forbids revisiting high number of solutions.
- A **low length** of the tabu list will concentrates the search on the small area of the search space.

Aspiration Criteria

- In TS, tabus are sometimes too powerful: they may **prohibit attractive moves**, even when there is no danger of cycling, or they may lead to an overall **stagnation** of the searching process.
- It is thus necessary to use algorithmic devices that will allow one to **revoke** (cancel) tabus. These are called **aspiration criteria**.
- The simplest and most commonly used aspiration criterion (found in almost all TS implementations) consists in **allowing a move**, even if it is **tabu**, if it results in a solution with an objective value better than that of the current best-known solution (since the new solution has obviously not been previously visited).
- Much more complicated aspiration criteria have been proposed and successfully implemented, but they are rarely used. The **key rule** in this respect is that if cycling cannot occur, tabus can be **ignored**.

Termination Criteria

- The most commonly used stopping criteria in TS are:
 - ✓ 1) After a **fixed number of iterations** (or a fixed amount of CPU time).
 - 2) After some number of iterations **without an improvement** in the objective function value (the criterion used in most implementations).
 - 3) When the objective reaches a **pre-specified threshold** value.
- In complex tabu schemes, the search is usually stopped after **completing a sequence of phases**, the duration of each phase being determined by one of the above criteria.

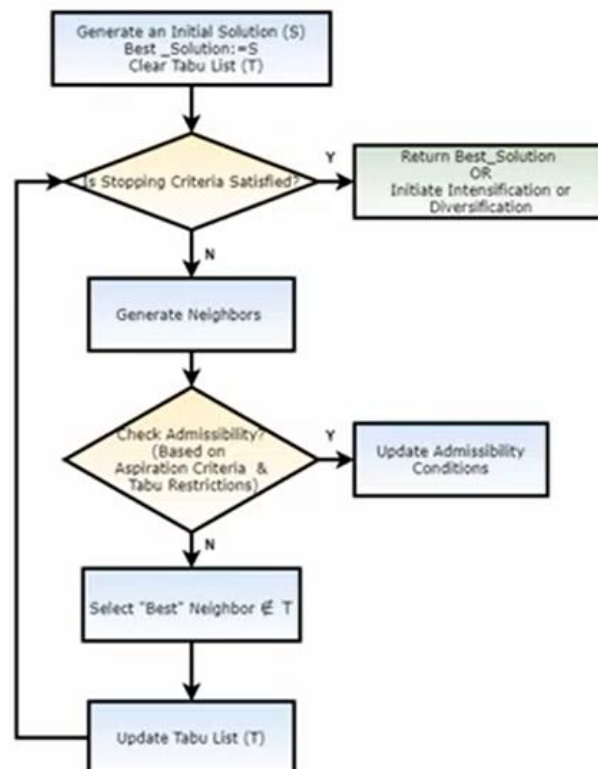
✓ Intensification

- The idea behind the concept of search intensification is that, as an intelligent human being would probably do, one should **explore** more thoroughly the portions of the search space that seem “**promising**” in order to make sure that the best solutions in these areas are indeed found.
- From time to time, one would thus stop the normal searching process to perform an intensification phase. In general, intensification is based on some **intermediate-term memory**, such as a **recency memory**, in which one records the number of consecutive iterations that various “solution components” have been present in the current solution without interruption.
- **Intensification** is used in many TS implementations, but it is **not always necessary**. This is because there are many situations where the search performed by the normal searching process is thorough enough. There is thus no need to spend time exploring more carefully the portions of the search space that have already been visited, and this time can be used more effectively.

Diversification

- One of the **main problems** of all methods based on Local Search approaches, and this includes TS in spite of the beneficial impact of tabus, is that they tend to be too “local” (as their name implies), i.e., they tend to **spend most**, if not all, of their **time** in a **restricted portion** of the search space.
- The **negative consequence** of this fact is that, although good solutions may be obtained, one may **fail to explore** the most **interesting parts** of the search space and thus end up with solutions that are still pretty far from the optimal ones.
- **Diversification** is an algorithmic mechanism that tries to alleviate this problem by **forcing** the search into **previously unexplored** areas of the search space. It is usually based on some form of **long-term memory** of the search, such as a **frequency memory**, in which one records the total number of iterations (since the beginning of the search) that various “solution components” have been present in the current solution or have been involved in the selected moves.

Generic Flowchart of Tabu Search Algorithm (Glover, 1990, Zang et al 2007)



A Simple Tabu Search Algorithm

Notations:

T :Tabu List,
N(S) :Set of Neighborhood Solutions,
S* : Best-So-Far Solution,
f(S*) :Value of S*

Algorithm Tabu Search

Begin

T := ϕ
S := Initial Solution
S* := S

Repeat

Find the BEST Admissible Solution $S' \in N(S)$;
If $f(S') > f(S^*)$ Then $S' := S^*$
S := S';
Update Tabu List T;

Until Stopping Criteria;

End;

Applications

- Tabu search (TS) is a **metaheuristic algorithm** that can be used for solving **combinatorial optimization problems**.
- **Current applications** of TS span the areas of resource planning, car pooling, telecommunications, VLSI design, financial analysis, scheduling, space planning, energy distribution, molecular engineering, logistics, vehicle routing, pattern classification, flexible manufacturing, waste management, mineral exploration, biomedical analysis, environmental conservation