Search In Complex Environments

Search in AI – Basic Idea

- In AI, **search** refers to systematically exploring possible actions and states in order to find a solution (e.g., shortest path, best strategy, optimal plan).

- Example: In a maze, search means looking for a path from the start to the goal.

Simple vs. Complex Environments

A search problem becomes **complex** when the environment has challenging characteristics.

Simple Environment

- Fully observable (you can see the whole map).
- Deterministic (actions always work as expected).
- Static (doesn't change while you are thinking).
- Discrete (finite states).

→ Example: Solving a crossword puzzle.

Complex Environment

The environment may have one or more of these:

- **Large or infinite state space** – e.g., chess, where there are ~10^120 possible states.
- **Uncertainty / nondeterminism** – actions may not always lead to expected outcomes (e.g., robot moving on slippery floor).
- **Partial observability** – agent has limited information (e.g., self-driving car can't see around corners).
- **Dynamic** – environment changes while the agent is searching (e.g., stock market, real-time strategy games).
- **Continuous spaces** – positions, velocities, probabilities are continuous, not discrete (e.g., robot navigation in real world).
- **Multi-agent environments** – presence of other agents with possibly competing goals (e.g., autonomous cars in traffic, board games like Go).

What “Search in Complex Environments” Means

It refers to designing **search algorithms** that can handle these real-world complications.
Instead of just exploring a neat graph/tree of possibilities, the AI must account for **uncertainty, incomplete info, huge state spaces, and dynamic changes.**

Examples of Search in Complex Environments

- **Path planning for robots/drones** in unpredictable terrain.
- **Autonomous driving** where traffic and pedestrian behavior are uncertain.
- **Game playing** like chess, Go, or StarCraft with massive state spaces and opponents.
- **Medical decision making** under incomplete patient data.
- **Resource allocation in networks** with changing conditions.

Local Search and Optimization

General Idea

- When we think of the word local search in the modern-day age, we think about something like this, right.
  - Tell me all restaurants near IIT right, or consult Google first
  - Local search, as it says, is any search aimed at finding something within a specific geographical area.
- General search algorithms: Any model can be, any problem can be formulated as a search problem and is solved using one algorithm, one search algorithm called the Local search algorithm. It is a series, a set of algorithms.
- There are some differences in the way we have formulated any given problem earlier, and how we are going to formulate it now.
- Previous lectures:
  - Path to goal is solution to problem,
  - we had start state, goal state, objective was to find what sequence of actions to apply
  - State/node was some partial information of the search so far
  - Systematic exploration of search space.
- From now on:
  - We will now be in a setting where solution is not the path but the state itself.
  - In other words, we will search in solution space.
  - Each search node would be a good/bad solution

Why Local Search in Complex Environments?

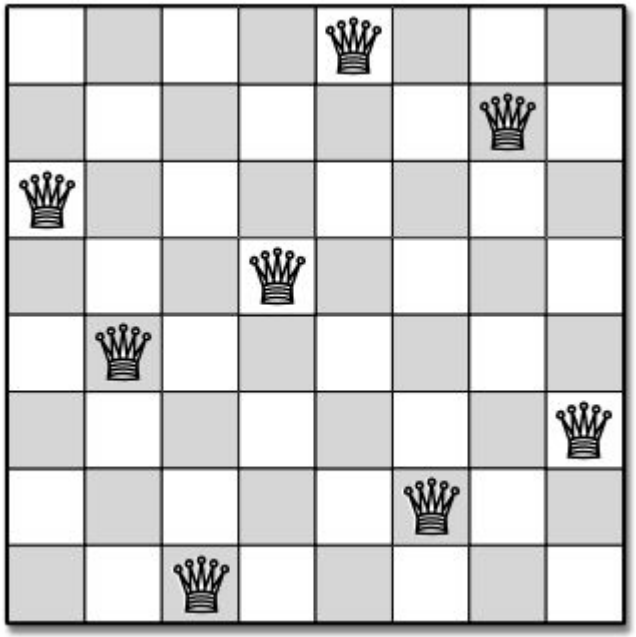
In complex environments:

- The **state space is huge or infinite** → exploring everything (like BFS/DFS) is impossible.
- We can’t store all explored states (memory explosion).
- Often, we don’t need the *entire path*, only a *good solution* (sometimes optimal, sometimes “good enough”).

Local search fits well here because:

- It works with a **single current state** rather than an entire search tree.
- It improves the state iteratively using an **evaluation/fitness function**.
- It uses **limited memory**.
- It can handle **continuous or very large state spaces**.

N Queens Problem



The **N-Queens problem** is a classic combinatorial puzzle where the objective is to place **N queens** on an **N x N chessboard** in such a way that no two queens threaten each other. This means no two queens can share the same row, column, or diagonal. The challenge is to find all valid configurations or a specific solution where these conditions are met.

Initial State: An empty chessboard (no queens placed yet), or a partial configuration with some queens already placed in the first few columns.

Goal State: A configuration where N queens are placed on the board such that no two queens threaten each other (i.e., no two queens share the same row, column, or diagonal).

Action: Add one queen.

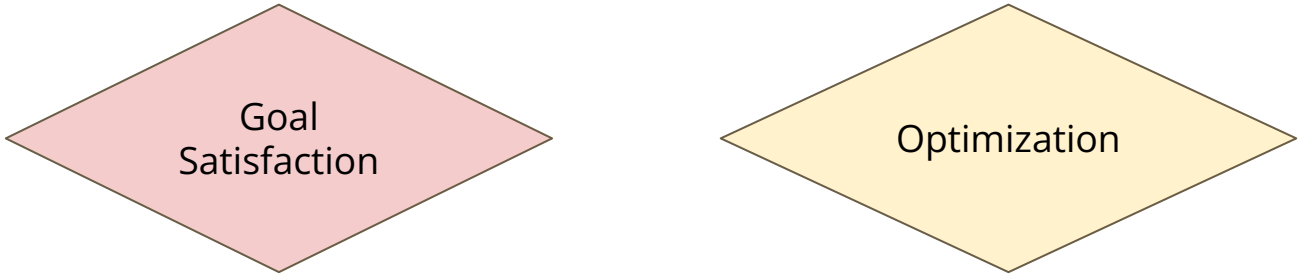
Search Algorithms: Informed and Uninformed

- Trying to find the shortest path to the goal.

Now, in general we are interested in the final state, satisfying all the constraints.

This can be modelled as a Local search problem.

Satisfaction vs. Optimization



- Up until now, we had been looking at satisfaction problems. Satisfaction problem is to find the path to the goal, and once we find the path, we want the best path to the goal.
- So, there is some optimisation going also.
- But suppose we are only interested in finding a path to the goal. That is what we are going to call a satisfaction problem.
- Optimisation problems, is like finding the optimal path to optimising the objective function in general.
- In the satisfaction world, we have to find a solution that satisfies all constraints.
- In the optimization world, we have an objective function and want to minimise or maximise.
- In satisfaction, I have given just several constraints. In Optimisation, I have given the number of constraints and an objective function.

Example

- N Queens problem is a satisfaction problem or an optimization problem?
- It is a satisfaction problem, but we will formulate it as an optimization problem now, and a state would be a solution.
- We have to develop an objective function that is analogous to the original objective function of the satisfaction problem when optimized fully.

Optimization Problems

Pure Optimization Problems

- **Definition:**
  - In a pure optimization problem, the objective is to find the best possible state (or configuration) that maximizes or minimizes a given objective function. The problem is usually defined in terms of:
    - **Objective Function:** A function that assigns a value (fitness, cost, utility, etc.) to each possible state. The goal is to optimize this function.
    - **State Space:** All possible configurations or states that the variables of the problem can take.
    - **Constraints:** Conditions that the solutions must satisfy (in constrained optimization problems).
- **No Goal Test or Path Cost:**
  - Unlike "standard" search problems (like those introduced in previous slides), optimization problems often do not have a **goal test**—there is no predefined target state to reach. Instead, the search is about finding the best state according to the objective function.
  - Additionally, there is typically no **path cost** associated with reaching the solution. The focus is on the quality of the final state, not on how it was reached.

Examples of Pure Optimization Problems

- **Engineering Design:** Finding the optimal design parameters for a bridge that minimizes cost while maximizing strength.
- **Portfolio Optimization:** Selecting a combination of financial assets that maximizes return for a given level of risk.
- **Traveling Salesperson Problem (TSP):** Finding the shortest possible route that visits each city exactly once and returns to the origin city.
- **Machine Learning Model Tuning:** Selecting hyperparameters that minimize the error on validation data.

Technical Description: Local Search

Local search is a type of optimization technique used to solve complex computational problems, particularly in large and combinatorial search spaces where finding an exact solution might be computationally infeasible. Instead of exhaustively exploring the entire search space, local search focuses on iteratively improving a single solution by making small, incremental changes, often referred to as "moves." The goal is to find a solution that is better than the current one by exploring its "neighborhood."

1. Key Concepts of Local Search

1. **Search Space:**
    - The search space represents all possible solutions to a problem. In local search, the focus is on exploring a small portion of this space around a current solution rather than attempting to traverse the entire space.
2. **Current Solution:**
    - Local search begins with an initial solution, which can be generated randomly or through some heuristic method. This solution is not necessarily optimal but serves as a starting point.

3. **Neighborhood:**

   - The neighborhood of a solution consists of all possible solutions that can be reached by making a small, defined change to the current solution. The exact nature of this change depends on the problem being solved. For example, in a traveling salesperson problem (TSP), the neighborhood might include all possible routes that can be created by swapping the order of two cities.

4. **Objective Function:**

   - The objective function measures the quality or "fitness" of a solution. Local search algorithms aim to optimize this function, either by maximizing or minimizing its value, depending on the problem.

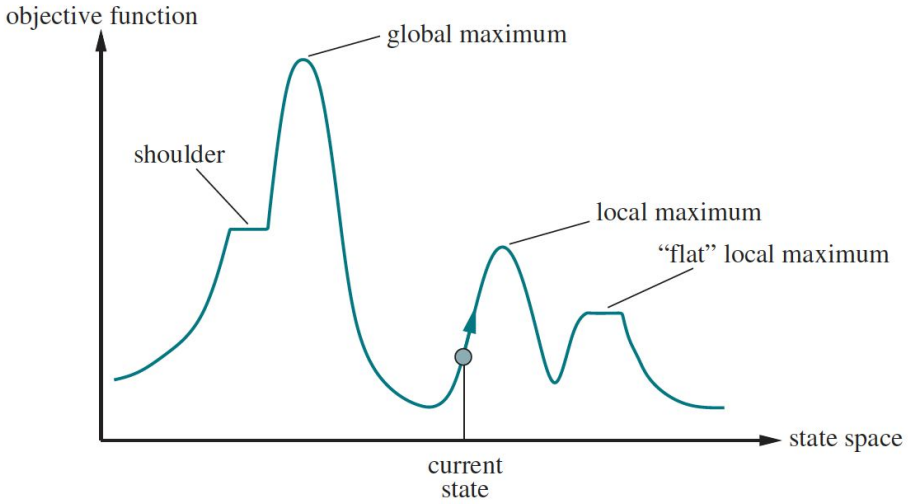
5. **Move:**

   - A move refers to the transition from one solution to another within its neighborhood. This is achieved by making a small change to the current solution.

6 **Termination Criteria:**

   - Local search algorithms typically terminate when a predefined condition is met, such as reaching a maximum number of iterations, achieving a solution of acceptable quality, or when no further improvement is possible (a local optimum is reached).

State-Space Landscape



A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum

Objective Function: The vertical axis represents the **objective function**, which could be a cost or reward function depending on the problem. The goal of optimization is to maximize or minimize this function.

State Space: The horizontal axis represents the **state space**, which consists of possible solutions or configurations in the problem domain.

Global Maximum: This is the highest point in the search space, representing the **best possible solution**. The goal of many optimization algorithms is to reach this point.

Local Maximum: A **local maximum** is a point where the objective function reaches a peak, but it is **not the highest point** in the entire search space. The algorithm can get stuck here, thinking it has found the best solution when it hasn't.

"Flat" Local Maximum: This is a region where the objective function is **flat** or constant over multiple states. It can prevent the algorithm from making progress because it doesn't have a clear direction to improve the solution.

Shoulder: The shoulder is a relatively flat area on the way to the global maximum. This flatness can temporarily deceive the algorithm into thinking it's reached a local maximum, though improvement may still be possible.

Current State: The arrow starting from the **current state** shows the algorithm's position in the search space. The image suggests that the algorithm is currently at a local maximum or is stuck on a flat region, potentially preventing it from reaching the global maximum.

Advantages of Local Search

- **Scalability:** Local search methods can handle very large search spaces because they focus on a small neighborhood around the current solution rather than attempting to explore the entire space.
- **Simplicity:** Many local search algorithms, like hill climbing, are conceptually simple and easy to implement.
- **Flexibility:** Local search can be applied to a wide range of problems, including those where other algorithms may be impractical due to the size or complexity of the search space.
- **Efficiency:** For many practical problems, local search can quickly find good (if not optimal) solutions, making it useful in real-time applications where time is a critical factor.

Limitations of Local Search

Local Optima: Local search algorithms can get stuck in local optima—solutions that are better than neighboring solutions but not the best overall. Overcoming this requires additional strategies, like simulated annealing or tabu search.

No Guarantee of Global Optimality: Because local search focuses on improving a single solution incrementally, it does not guarantee finding the global optimum unless the problem has a particular structure that allows it.

Problem-Specific Design: The effectiveness of a local search algorithm often depends on the careful design of the neighborhood structure and move operators, which can be problem-specific.

No Full Exploration: Local search does not explore the entire search space, so it might miss better solutions located far from the current solution.

Types of Local Search Algorithms

1. **Hill Climbing**
2. **Simulated Annealing**
3. **Tabu Search**
4. **Genetic Algorithms (GA)**
5. **Local Beam Search**
6. **Iterated Local Search**

Summary

- Local Search
  - Keep track of single current state
  - Move only to neighbouring states
  - Ignore paths

- “Pure Optimization” Problems
  - All states have an objective function
  - Goal is to find state with max (or min) objective value
  - Does not quite fit into path-cost/ goal-state formulation
  - Local search can do quite well on these problems.

Hill Climbing

Hill Climbing is a type of **local search algorithm** used for solving **optimization problems**, where the goal is to find the best solution according to an objective function. It is particularly useful for problems where the search space is vast and traditional exhaustive search methods are computationally infeasible.

- **Local Search Algorithm:**
  - In contrast to global search algorithms, which attempt to explore the entire search space, local search algorithms focus on finding solutions by exploring the local neighborhood of a given solution. The term "local" implies that the algorithm's perspective is limited to the immediate vicinity of the current solution.
- **Mathematical Optimization:**
  - Hill climbing is used in various mathematical optimization problems, where the objective is to either maximize or minimize a specific function. These problems can range from simple linear functions to complex, multi-dimensional spaces.
- **Iterative Process:**
  - Hill climbing starts with an arbitrary solution, which can be chosen randomly or based on a heuristic. From this initial solution, the algorithm iteratively makes small changes, or "moves," to the solution. Each move involves evaluating the neighboring solutions (states) and selecting the one that improves the objective function the most. This process continues until no better neighboring solution can be found.
- **Neighboring Solutions:**
  - A neighboring solution is one that is reachable from the current solution by making a small change. For example, in the case of a numerical optimization problem, a neighboring solution could be obtained by slightly increasing or decreasing the value of a variable.

Basic Idea of Hill Climbing

The name "hill climbing" comes from the metaphor of climbing a hill: just as a climber takes steps in the direction that leads upward to reach the top of a hill, the algorithm takes steps in the direction that improves the objective function. The "hill" in this metaphor represents the landscape of the objective function, where higher points correspond to better (higher) values of the function in the case of maximization, and lower points correspond to better (lower) values in the case of minimization.

- **Direction of Movement:**
  - **Maximization Problems:** In these problems, the goal is to find the maximum value of the objective function. The algorithm moves in the direction of increasing the function's value, akin to climbing up a hill.
  - **Minimization Problems:** Here, the goal is to find the minimum value of the objective function. The algorithm moves in the direction of decreasing the function's value, which can be thought of as descending a hill or valley to reach the lowest point.
- **Local Optimum:**
  - The algorithm stops when it reaches a point where no neighboring solution has a better value than the current solution. This point is called a **local optimum**. It is important to note that a local optimum is not necessarily the best possible solution (global optimum) but is the best in the immediate vicinity of the current solution.
- **No Exploration of the Entire Search Space:**
  - Hill climbing does not attempt to explore the entire search space. Instead, it focuses on improving the current solution by evaluating and selecting among its neighbors. This makes the algorithm efficient in terms of time and space, as it only needs to consider a small portion of the search space at each step.

Hill Climbing Algorithm

1. Start with an Initial Solution

- **Objective:** Establish a starting point for the algorithm.
- **Process:**
  - **Initial State Selection:**
    - The algorithm begins by choosing an initial state or solution. This initial state can be selected in various ways:
      - **Random Selection:** In many cases, the initial solution is chosen randomly from the search space. This randomness helps ensure that the algorithm starts from a diverse range of points if multiple runs are performed.
      - **Heuristic-Based Selection:** Alternatively, the initial solution can be chosen based on some heuristic that suggests a potentially good starting point. For example, in a scheduling problem, you might start with a schedule that you know satisfies some key constraints.
- **Importance:**
  - The choice of the initial solution can significantly impact the algorithm's performance. A good initial solution may lead to quicker convergence to an optimal or near-optimal solution, while a poor initial choice might result in the algorithm getting stuck in suboptimal areas of the search space.

2. Evaluate the Objective Function

- **Objective:** Measure the quality of the current solution.
- **Process:**
  - **Objective Function Computation:**
    - The algorithm evaluates the objective function at the current solution. The objective function is a mathematical expression that quantifies how good or bad a solution is with respect to the problem being solved.
    - **For Maximization Problems:** The objective function's value is higher for better solutions. The goal is to maximize this value.
    - **For Minimization Problems:** The objective function's value is lower for better solutions. The goal is to minimize this value.
- **Importance:**
  - The objective function is central to the hill climbing algorithm. It determines the direction of the search by guiding the algorithm towards better solutions.
- **Example:**
  - If the problem is to maximize a function $f(x) = -x^2+4x$, and the initial solution is $x=1$, the objective function value would be $f(1) = -1^2 + 4(1) = 3$.

3. Generate Neighboring Solutions

- **Objective:** Explore the local area around the current solution to find potential improvements.
- **Process:**
  - **Defining the Neighborhood:**
    - The algorithm identifies all neighboring solutions. A neighbor is a solution that can be reached from the current solution by making a small, defined change.
    - **Neighborhood Definition:** The definition of a neighborhood depends on the problem. For example:
      - **In a Numerical Optimization Problem:** A neighbor might be a solution obtained by slightly increasing or decreasing the value of a variable.
      - **In a Combinatorial Problem (e.g., TSP):** A neighbor might be generated by swapping two cities in the tour.
- **Importance:**
  - The size and structure of the neighborhood significantly impact the algorithm's effectiveness. A well-defined neighborhood ensures that the algorithm can explore the search space effectively and avoid getting stuck in suboptimal regions.
- **Example:**
  - For x=1, possible neighbors might be x= 0.9 and x=1.1. The objective function values at these neighbors would be computed next.

4. Select the Best Neighbor

- **Objective:** Choose the most promising neighboring solution.
- **Process:**
  - **Evaluate Neighboring Solutions:**
    - The algorithm computes the objective function values for all the neighboring solutions identified in the previous step.
  - **Best Neighbor Selection:**
    - Among the neighboring solutions, the one that offers the greatest improvement in the objective function is selected. This move ensures that the algorithm is always progressing towards a better solution.
    - **For Maximization Problems:** The neighbor with the highest objective function value is chosen.
    - **For Minimization Problems:** The neighbor with the lowest objective function value is chosen.
- **Importance:**
  - This step ensures that the algorithm makes the best possible move at each iteration, steadily improving the solution. However, this greedy approach can sometimes lead the algorithm to get stuck in local optima.
- **Example:**
  - If x = 0.9 yields a value of 2.81 and x=1.1 yields a value of 2.99, the algorithm selects x=1.1 as the next solution because it has the higher value.

5. Move to the Neighboring Solution

- **Objective:** Update the current solution to the selected neighbor.
- **Process:**
  - **Transition to the New Solution:**
    - The algorithm moves to the best neighboring solution selected in the previous step. This solution becomes the new "current solution" for the next iteration.
  - **Repeat the Process:**
    - The algorithm then repeats the process, starting from evaluating the objective function at the new current solution, generating its neighbors, and selecting the best one.
- **Importance:**
  - This step ensures the algorithm is always progressing towards better solutions, provided such solutions exist in the neighborhood.
- **Example:**
  - The algorithm updates the current solution from x=1 to x=1.1 and repeats the process from this new starting point.

6. Termination Criteria

- **Objective:** Determine when the algorithm should stop running.
- **Process:**
  - **No Improvement Found:**
    - The most common termination criterion is when no neighboring solution improves the objective function. This situation indicates that the algorithm has reached a **local optimum**—a solution that is better than all its neighbors but may not be the best overall solution (global optimum).
  - **Other Termination Criteria:**
    - **Maximum Iterations:** The algorithm stops after a predefined number of iterations, regardless of whether an optimal solution has been found.
    - **Threshold Value:** The algorithm stops if the objective function value reaches or exceeds (for maximization) or falls below (for minimization) a certain threshold.
    - **Time Limit:** The algorithm stops after running for a certain amount of time.
- **Importance:**
  - Proper termination criteria are crucial to ensure that the algorithm does not run indefinitely and that it finds a solution in a reasonable time. However, it also determines how close the algorithm gets to the global optimum.
- **Example:**
  - If the algorithm reaches a point where increasing or decreasing **x** no longer improves the objective function, it stops and returns the current solution as the final result.

Algorithm

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

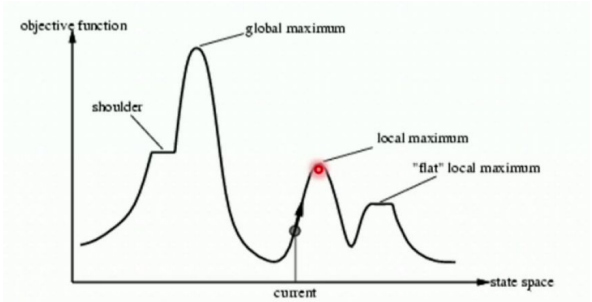
Hill Climbing is Called Greedy Local Search

Focus on Immediate Gains: Hill climbing's "greediness" comes from its strategy of always selecting the best immediate move that improves the current solution's value according to the objective function. It makes decisions based on the current, local information without considering the global context or potential long-term consequences.

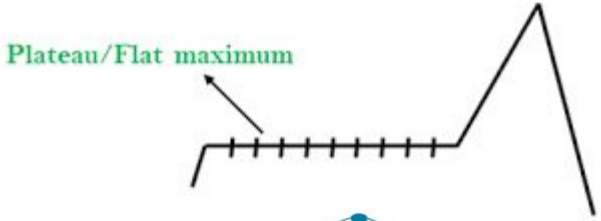
Limited Perspective: As a local search method, hill climbing only explores the immediate neighborhood of the current solution, further emphasizing its greedy approach by restricting its focus to nearby, small-scale improvements.

Drawbacks of Hill Climbing

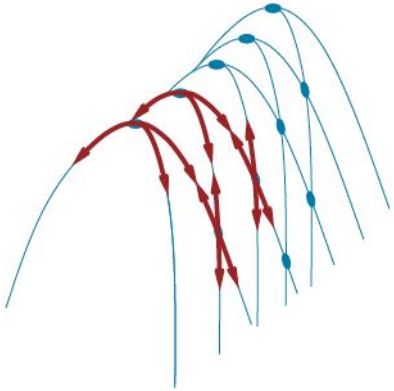
1. Stuck in Local Optima



2. Plateau Problem



3. Ridges: A **ridge** is a narrow path of optimal or near-optimal solutions surrounded by regions of lower values.



Variants of Hill Climbing

- **Steepest-Ascent Hill Climbing**
- **Stochastic Hill Climbing**
- **First-Choice Hill Climbing:**
- **Random-Restart Hill Climbing**

Steepest-Ascent Hill Climbing

Steepest-Ascent Hill Climbing is a variant of the hill climbing algorithm that seeks to address some of the limitations of basic hill climbing by systematically and comprehensively evaluating all possible moves (neighbors) before making a decision. This approach ensures that the algorithm always chooses the most significant improvement available in the current neighborhood, hence the name "steepest-ascent."

- **Basic Idea:**
  - Steepest-Ascent Hill Climbing is a local search optimization algorithm where the goal is to iteratively move towards the best solution by making the steepest possible ascent (or descent in the case of minimization problems) at each step.
  - Unlike the simpler "greedy" hill climbing that might choose the first neighbor that improves the objective function, steepest-ascent hill climbing evaluates all neighboring solutions and selects the one that offers the greatest improvement.
- **Application:**
  - This algorithm is useful in scenarios where the search space is relatively smooth and the best local decision at each step is likely to lead towards the global optimum. It is commonly applied in optimization problems, including numerical optimization, combinatorial optimization, and machine learning hyperparameter tuning.

Step-by-Step Process of Steepest-Ascent Hill Climbing

Step 1: Start with an Initial Solution

- **Initial State Selection:**
  - As with other hill climbing algorithms, steepest-ascent hill climbing starts with an initial solution. This solution can be chosen randomly or based on some heuristic.
- **Importance of Initial Solution:**
  - The initial solution serves as the starting point for the search. Its quality can affect the speed of convergence and the likelihood of finding a global optimum.

Step 2: Evaluate the Objective Function

- **Objective Function:**
  - The algorithm evaluates the objective function for the current solution to determine its fitness or quality. The objective function is the metric that the algorithm seeks to optimize (either maximize or minimize).
- **Importance:**
  - The value of the objective function provides a baseline against which all neighboring solutions will be compared.

Step 3: Generate and Evaluate All Neighboring Solutions

- **Neighborhood Generation:**
  - The algorithm identifies all possible neighboring solutions of the current solution. A neighbor is typically defined as a solution that can be reached by making a small, specific change to the current solution.
  - **Example in Numerical Optimization:** If the solution is represented by a vector of variables, a neighbor might be generated by slightly perturbing one of the variables.
- **Comprehensive Evaluation:**
  - **Evaluate Each Neighbor:** The algorithm calculates the objective function value for each neighboring solution.
  - **Compare Neighbors:** After evaluating all neighbors, the algorithm compares their objective function values to determine which one offers the steepest ascent (or descent).
- **Importance:**
  - This comprehensive evaluation step is what distinguishes steepest-ascent hill climbing from simpler variants. By considering all possible neighbors, the algorithm ensures that it is making the best possible move at each step, avoiding the pitfalls of making a suboptimal choice due to limited exploration.

Step 4: Select the Best Neighbor

- **Best Neighbor Selection:**
  - The neighbor with the highest improvement (for maximization problems) or the greatest reduction (for minimization problems) in the objective function is selected as the next solution.
- **No Improvement Handling:**
  - If no neighboring solution is better than the current solution, the algorithm recognizes that it has reached a local optimum, where no further improvements are possible within the current neighborhood.

Step 5: Move to the Neighboring Solution

- **Update the Current Solution:**
  - The algorithm moves to the selected neighboring solution, which becomes the new current solution. This move represents the steepest possible ascent in the objective function based on the local neighborhood.
- **Iterative Process:**
  - The process then repeats from the new solution, with the algorithm generating and evaluating its neighbors, selecting the best one, and moving again.
- **Importance:**
  - By always choosing the steepest ascent, the algorithm maximizes the likelihood of progressing towards the global optimum or, at the very least, a better local optimum.

Step 6: Termination Criteria

- **Local Optimum Recognition:**
  - The algorithm terminates when it reaches a point where no neighboring solution improves the objective function. This situation indicates that the algorithm has reached a local optimum, and further exploration in the current neighborhood will not yield better results.
- **Alternative Termination Conditions:**
  - **Maximum Iterations:** The algorithm might stop after a predefined number of iterations.
  - **Objective Threshold:** The algorithm could also stop when the objective function value reaches a certain target or threshold.

Advantages of Steepest-Ascent Hill Climbing

Systematic Exploration:

- By evaluating all neighbors, steepest-ascent hill climbing avoids the risk of missing a good move due to a hasty decision, making it more reliable in finding the best possible local solution within the current neighborhood.

Efficiency in Smooth Search Spaces:

- This algorithm performs well in smooth, well-behaved search spaces where local improvements consistently lead toward the global optimum.

Fewer Local Optima Issues:

- While still prone to local optima, the systematic approach reduces the chances of getting stuck early in suboptimal regions of the search space.

Limitations of Steepest-Ascent Hill Climbing

- **Computational Cost:**
  - Evaluating all neighboring solutions can be computationally expensive, especially in high-dimensional search spaces where the number of neighbors is large.
- **Local Optima:**
  - Despite its systematic approach, steepest-ascent hill climbing can still get stuck in local optima, particularly in complex, multi-modal search spaces where the global optimum is not easily reachable from any local optimum.
- **No Global Perspective:**
  - The algorithm only considers local information and does not have any mechanism to escape local optima or explore more distant parts of the search space.

Stochastic Hill Climbing

Stochastic Hill Climbing is a variant of the hill climbing algorithm that introduces an element of randomness into the search process. Unlike traditional hill climbing methods, which deterministically select the best neighbor at each step, stochastic hill climbing randomly selects a neighbor to move to, with a preference for better neighbors. This randomness helps the algorithm explore the search space more broadly and potentially avoid getting stuck in local optima.

- **Basic Idea:**
  - Stochastic Hill Climbing modifies the standard hill climbing approach by introducing randomization into the selection of the next move. Instead of always choosing the best possible neighbor (as in steepest-ascent hill climbing), stochastic hill climbing may choose a neighbor that is not necessarily the best, based on a probabilistic criterion.
  - This approach helps the algorithm escape local optima by allowing it to explore less optimal regions of the search space that could potentially lead to better solutions in the long run.
- **Application:**
  - Stochastic hill climbing is useful in scenarios where the search space is complex and contains many local optima. It is often applied in optimization problems, particularly in domains like machine learning, game AI, and operations research.

Step-by-Step Process of Stochastic Hill Climbing

Step 1: Start with an Initial Solution

- **Initial State Selection:**
  - The algorithm begins with an initial solution, which can be selected randomly or based on a heuristic. The initial solution serves as the starting point for the search.
- **Importance of Initial Solution:**
  - The quality of the initial solution can influence the speed and success of the search, but due to the random nature of stochastic hill climbing, the algorithm is less dependent on finding a good initial solution compared to deterministic methods.

Step 2: Evaluate the Objective Function

- **Objective Function:**
  - The algorithm evaluates the objective function at the current solution. This function measures the quality of the solution and provides a basis for comparing neighboring solutions.
- **Importance:**
  - The objective function's value guides the decision-making process, helping the algorithm determine whether a move is beneficial or not.

Step 3: Generate Neighboring Solutions

- **Neighborhood Generation:**
  - The algorithm identifies neighboring solutions by making small changes to the current solution. The definition of a neighbor depends on the problem being solved.
- **Importance:**
  - The structure of the neighborhood impacts the algorithm's ability to explore the search space. A well-defined neighborhood allows the algorithm to consider a diverse set of potential moves.

Step 4: Select a Neighbor Stochastically

- **Stochastic Selection Process:**
  - Unlike deterministic hill climbing, which selects the best neighbor, stochastic hill climbing selects a neighbor randomly. However, this selection is often biased towards better neighbors—those that improve the objective function.
  - **Probability Distribution:** The likelihood of selecting a particular neighbor can be determined by a probability distribution. For instance, neighbors that offer greater improvements might be selected with higher probability, while less favorable neighbors are less likely but still have a chance of being selected.
  - **Example Probability Distribution:**
    - A common approach is to assign probabilities based on the difference in objective function values. For example, if one neighbor has a significantly better objective function value than others, it might be assigned a higher probability of being chosen, but others are not entirely excluded.
- **Importance:**
  - The stochastic nature of this step allows the algorithm to explore the search space more freely, reducing the risk of getting stuck in local optima. It also introduces a level of unpredictability, which can be beneficial in complex landscapes.

Step 5: Move to the Selected Neighbor

- **Update the Current Solution:**
  - The algorithm moves to the selected neighboring solution. This solution becomes the new current solution for the next iteration.
- **Iterative Process:**
  - The process then repeats, with the algorithm generating new neighbors, selecting one stochastically, and moving to it.
- **Importance:**
  - By continuously moving to new solutions, the algorithm explores the search space in a non-deterministic manner, which can help in finding better solutions that might be missed by more rigid methods.

Step 6: Termination Criteria

- **Termination Conditions:**
  - The algorithm can terminate under several conditions:
    - **No Improvement:** If after several iterations no significant improvement is found, the algorithm may stop.
    - **Maximum Iterations:** The algorithm stops after a predefined number of iterations.
    - **Time Limit:** The algorithm stops after a certain amount of computational time has passed.
    - **Objective Threshold:** The algorithm stops if the objective function reaches a desired target value.
- **Importance:**
  - Proper termination criteria ensure that the algorithm does not run indefinitely and can provide a solution within a reasonable time frame.

Advantages of Stochastic Hill Climbing

Escaping Local Optima:

- The primary advantage of stochastic hill climbing is its ability to escape local optima. By allowing moves that are not strictly the best, the algorithm can explore regions of the search space that deterministic hill climbing might overlook.

Broad Exploration:

- The random selection process enables the algorithm to explore the search space more broadly, potentially leading to better overall solutions.

Simplicity and Flexibility:

- Stochastic hill climbing is relatively simple to implement and can be easily adapted to different types of optimization problems by adjusting the probability distribution for selecting neighbors.

Limitations of Stochastic Hill Climbing

Potentially Slower Convergence:

- Because the algorithm may sometimes choose suboptimal moves, it can take longer to converge to a solution compared to deterministic methods. This slower convergence is the trade-off for potentially escaping local optima.

No Guarantee of Global Optimum:

- Like other hill climbing methods, stochastic hill climbing does not guarantee finding the global optimum. It may still get stuck in local optima, especially if the search space is highly irregular or if the probability distribution is not well-tuned.

Dependence on Probability Distribution:

- The effectiveness of stochastic hill climbing can heavily depend on how the probability distribution is defined. Poorly chosen distributions may lead to ineffective exploration or excessive randomness, reducing the algorithm's efficiency.

First-Choice Hill Climbing

First-Choice Hill Climbing is a variant of the hill climbing algorithm designed to balance between the thoroughness of steepest-ascent hill climbing and the efficiency of simple hill climbing. In this approach, the algorithm randomly generates neighboring solutions and selects the first one that offers an improvement over the current solution. This method is particularly useful when the neighborhood of each solution is large, and evaluating all neighbors would be computationally expensive.

- **Basic Idea:**
  - First-Choice Hill Climbing combines elements of randomness and greediness. Instead of systematically evaluating all neighbors or choosing the best one from a pre-generated set, the algorithm evaluates neighbors one by one in a random order. It accepts the first neighbor that improves the objective function, which makes it faster than steepest-ascent hill climbing while still capable of escaping local optima more effectively than simple hill climbing.
- **Application:**
  - This algorithm is useful in large search spaces where generating and evaluating all possible neighbors is impractical. It is often applied in optimization problems where quick, incremental improvements are desirable without the need for an exhaustive search at each step.

Step-by-Step Process of First-Choice Hill Climbing

Step 1: Start with an Initial Solution

- **Initial State Selection:**
  - The algorithm starts with an initial solution, which can be chosen randomly or based on some heuristic. This initial solution serves as the starting point for the search process.
- **Importance of Initial Solution:**
  - The initial solution influences the starting position in the search space and can affect the speed and effectiveness of the algorithm. However, because of the randomized nature of neighbor selection, the algorithm is relatively robust to different starting points.

Step 2: Evaluate the Objective Function

- **Objective Function:**
  - The algorithm evaluates the objective function at the current solution to determine its quality or fitness. The objective function quantifies how good or bad the current solution is with respect to the problem's goals.
- **Importance:**
  - The objective function provides the baseline for evaluating neighboring solutions. Improvements are determined by comparing the objective function values of the current solution and potential neighbors.

Step 3: Generate and Evaluate Neighboring Solutions One by One

- **Neighborhood Generation:**
  - The algorithm generates neighboring solutions of the current solution by making small, specific changes. These changes define the neighborhood around the current solution.
- **Randomized Evaluation:**
  - **Random Selection:** The algorithm does not generate all neighbors at once. Instead, it generates neighbors one at a time in a random order.
  - **Immediate Evaluation:** Each time a neighbor is generated, the algorithm immediately evaluates its objective function value.
- **Importance:**
  - This randomized, one-by-one approach allows the algorithm to find an improving solution without the need to exhaustively evaluate all neighbors. It saves computational resources, particularly in large search spaces where the number of neighbors can be very large.

Step 4: Select the First Improving Neighbor

- **Immediate Selection:**
  - The algorithm selects the first neighbor that offers an improvement over the current solution. Once a better neighbor is found, the search stops evaluating other neighbors and moves to this new solution.
- **Greedy Acceptance:**
  - The acceptance criterion is greedy in nature: the algorithm only accepts moves that improve the objective function. This ensures that each move is a step towards a better solution.
- **Importance:**
  - By selecting the first improving neighbor, the algorithm quickly progresses towards better solutions without wasting time on unnecessary evaluations. This makes it faster than methods that evaluate all neighbors but still capable of finding good solutions.

Step 5: Move to the New Solution

- **Update the Current Solution:**
  - Once a better neighbor is found, the algorithm moves to this new solution. This new solution becomes the current solution for the next iteration of the algorithm.
- **Iterative Process:**
  - The process repeats from this new solution, with the algorithm generating new neighbors, evaluating them one by one, and moving to the first one that improves the objective function.
- **Importance:**
  - This step ensures that the algorithm is always progressing towards better solutions, while the randomized approach allows it to explore the search space in a flexible manner.

Step 6: Termination Criteria

- **Local Optimum Recognition:**
  - The algorithm terminates when it can no longer find a neighbor that improves the objective function. This indicates that a local optimum has been reached, where no further improvements are possible within the current neighborhood.
- **Alternative Termination Conditions:**
  - **Maximum Iterations:** The algorithm might stop after a predefined number of iterations.
  - **Objective Threshold:** The algorithm could also stop if the objective function value reaches a certain target or threshold.
  - **Time Limit:** The algorithm might stop after a certain amount of computational time has passed.

Advantages of First-Choice Hill Climbing

Efficiency:

- First-Choice Hill Climbing is more efficient than steepest-ascent hill climbing because it does not require evaluating all neighbors. By stopping as soon as an improvement is found, the algorithm reduces the computational burden, especially in large search spaces.

Escape from Local Optima:

- The randomized nature of the neighbor selection process helps the algorithm escape local optima. By not always selecting the globally best neighbor, the algorithm can explore more diverse areas of the search space.

Simplicity and Flexibility:

- The algorithm is straightforward to implement and can be easily adapted to various types of optimization problems. Its flexibility comes from the randomization in neighbor selection, which can be adjusted based on the problem's requirements.

Limitations of First-Choice Hill Climbing

Potential Suboptimal Choices:

- Because the algorithm selects the first improving neighbor rather than the best possible neighbor, it may make suboptimal moves. This could lead to slower convergence to the optimal solution or settling for a less-than-optimal solution.

Still Prone to Local Optima:

- Although more flexible than simple hill climbing, First-Choice Hill Climbing can still get stuck in local optima, particularly if the search space is highly irregular or contains many deceptive local peaks.

Dependence on Randomization:

- The performance of the algorithm can be sensitive to the randomization process. Poorly designed randomization may lead to inefficient exploration or excessive cycling between similar solutions.

Random-Restart Hill Climbing

Random-Restart Hill Climbing is a variant of the hill climbing algorithm that seeks to overcome one of the most significant limitations of basic hill climbing—getting stuck in local optima. This method enhances the standard hill climbing approach by running the algorithm multiple times from different random starting points in the search space. The idea is that if one run gets stuck in a local optimum, subsequent runs from different starting points may find better solutions, potentially leading to the global optimum.

- **Basic Idea:**
  - Random-Restart Hill Climbing is a meta-algorithm, meaning it wraps around the basic hill climbing algorithm. Instead of relying on a single run of hill climbing, which might get trapped in a local optimum, the algorithm runs hill climbing multiple times from different randomly chosen initial solutions. The best solution found across all these runs is then chosen as the final solution.
- **Application:**
  - Random-Restart Hill Climbing is useful in optimization problems where the search space is complex, with many local optima. It is widely used in problems like function optimization, neural network training, and combinatorial optimization, where finding the global optimum is crucial.

Step-by-Step Process of Random-Restart Hill Climbing

Step 1: Choose the Number of Restarts

- **Determining the Number of Restarts:**
  - Before starting the algorithm, decide how many times the hill climbing algorithm will be restarted. This number can be fixed or adaptive, depending on the problem's complexity and the computational resources available.
  - **Fixed Number of Restarts:** Predefine a specific number of restarts based on experience or trial-and-error.
  - **Adaptive Restarts:** Continue restarting until a satisfactory solution is found or until computational resources (e.g., time) are exhausted.
- **Importance:**
  - The number of restarts balances the trade-off between search thoroughness and computational efficiency. More restarts increase the chances of finding the global optimum but also require more computational time.

Step 2: Start the First Hill Climbing Run

- **Initial Random Solution:**
  - Randomly select an initial solution from the search space. This solution serves as the starting point for the first run of the hill climbing algorithm.
- **Run Hill Climbing:**
  - Perform the standard hill climbing algorithm starting from this initial solution. The algorithm will iterate through generating and evaluating neighbors, selecting the best one, and moving to it until a local optimum is reached.
- **Importance:**
  - The first run establishes a baseline solution. Even if this solution is not globally optimal, it provides a starting point for comparison with solutions found in subsequent restarts.

Step 3: Evaluate and Record the Result

- **Objective Function Evaluation:**
  - Once the hill climbing algorithm converges to a local optimum, evaluate the quality of this solution using the objective function.
- **Record the Best Solution:**
  - Keep track of the best solution found so far. If this solution is better than any previously recorded solutions, update the best solution.
- **Importance:**
  - Recording the best solution ensures that the algorithm can ultimately return the best possible result found across all restarts.

Step 4: Repeat for Additional Restarts

- **Subsequent Random Solutions:**
  - For each subsequent restart, choose a new random initial solution from the search space.
- **Run Hill Climbing Again:**
  - Perform the hill climbing algorithm from this new starting point, allowing it to find another local optimum.
- **Evaluate and Record:**
  - After each restart, evaluate the resulting solution and update the best solution if necessary.
- **Importance:**
  - Multiple restarts increase the algorithm's coverage of the search space, reducing the likelihood of missing the global optimum.

Step 5: Termination Criteria

- **Fixed Restarts:**
  - If the algorithm is using a fixed number of restarts, it will stop after the predefined number of runs. The best solution found across all runs is returned as the final solution.
- **Adaptive Restarts:**
  - If the algorithm is adaptive, it may stop when a satisfactory solution is found or when a predefined computational budget (e.g., time or iterations) is exhausted.
- **Importance:**
  - Proper termination criteria ensure that the algorithm provides a solution within a reasonable time while maximizing the chances of finding the global optimum.

Advantages of Random-Restart Hill Climbing

- **Increased Probability of Finding Global Optimum:**
  - By running hill climbing multiple times from different starting points, Random-Restart Hill Climbing significantly increases the chances of finding the global optimum, even in complex search spaces with many local optima.
- **Flexibility:**
  - The algorithm can be adapted to different problem complexities by adjusting the number of restarts. It can be fine-tuned to balance between thoroughness and computational efficiency.
- **Simplicity:**
  - Random-Restart Hill Climbing is straightforward to implement. It leverages the simplicity of the basic hill climbing algorithm while adding a layer of robustness against local optima.

Limitations of Random-Restart Hill Climbing

Computational Cost:

- Running the hill climbing algorithm multiple times can be computationally expensive, especially if each run requires significant time or if many restarts are necessary to find a good solution.

No Guarantee of Global Optimum:

- While Random-Restart Hill Climbing increases the chances of finding the global optimum, it does not guarantee it. If the search space is vast or highly complex, even multiple restarts may not be sufficient to find the best solution.

Dependence on the Number of Restarts:

- The effectiveness of the algorithm heavily depends on the number of restarts. Too few restarts may lead to poor performance, while too many may result in unnecessary computational overhead.

N Queen Formulation as Local Search

Problem Definition (State, Objective, and Cost Function):

1. **State**:
   - A state in the N-Queens problem is defined by the positions of N queens on an N×N board, where each queen is in a unique column (a common constraint to simplify the problem).
   - For each column, we store the row position of the queen.
2. **Objective Function**:
   - The objective is to minimize the number of conflicts between queens. A conflict is defined as two queens attacking each other (sharing the same row or diagonal).
3. **Initial State**:
   - You start with a random initial configuration, where each queen is placed in a random row within its column. This configuration may have many conflicts initially.
4. **Neighboring States (Set of Actions)**:
   - A neighboring state is generated by moving a single queen to a different row within the same column. The set of possible actions is thus defined as moving any queen from its current row to any other row in the same column.
5. **Cost Function**:
   - The cost function C(S) calculates the number of queens attacking each other. The lower the cost, the better the state.

Note:

Objective = "minimize" or "maximize" something (direction of search).
Cost = the actual numerical value we're minimizing/maximizing.

N Queens- Hill Climbing

Greedy Hill Climbing: **Possible Set of Actions for Greedy:** For each queen, consider moving it to a different row in its column. The first move that reduces conflicts will be chosen.

Example:

- Assume you have a queen in column 1, row 3.
- Check each possible row (1, 2, 4, 5, etc.) in column 1. As soon as you find a row that reduces the total number of conflicts (e.g., moving the queen to row 2 reduces conflicts), make that move and stop further exploration.
- This approach **stops at the first improvement**.

Distinctive Feature: It **doesn't explore all possible moves**. It just makes the **first move that leads to a better configuration**, without further comparisons.

Steepest-Ascent Hill Climbing: Possible Set of Actions for Steepest-Ascent: For each queen, consider all possible row moves in its column. Compare all possible moves and choose the one that gives the largest reduction in conflicts.

Example:

- Assume you have a queen in column 2, row 5.
- Evaluate every possible move of every queen across all columns (e.g., column 1, column 2, etc.) to see which move results in the maximum reduction in conflicts.
- If moving the queen in column 3 from row 4 to row 1 results in the **largest decrease in conflicts**, you select this move.

Distinctive Feature: The algorithm looks for the **best overall move** across the entire board, rather than stopping at the first improvement. It evaluates the entire board before deciding on the best move.

First-Choice Hill Climbing: Possible Set of Actions for First-Choice: Randomly select a move (new row for a queen within its column). If it improves the solution (i.e., reduces conflicts), make the move.

Example:

- Start with any random initial configuration.
- Randomly choose a queen (say the one in column 4) and randomly move it to a different row (row 1, row 2, etc.).
- If this random move reduces conflicts, make the move immediately and continue the process.

Distinctive Feature: Moves are **randomly selected** without evaluating all possible neighbors. The algorithm makes the first move it finds that improves the configuration.

Stochastic Hill Climbing: Possible Set of Actions for Stochastic Hill Climbing: For each queen, randomly choose a neighboring position but with **higher probability for moves that reduce conflicts more**.

Example:

- Assume you have a queen in column 3, row 4.
- Evaluate all possible moves in its column (rows 1, 2, 3, 5, etc.), assigning higher probabilities to moves that reduce conflicts more. For example:
  - Moving the queen to row 1 reduces conflicts by 3 (higher probability of being chosen).
  - Moving the queen to row 2 reduces conflicts by 1 (lower probability).
  - Moving the queen to row 3 increases conflicts (very low probability).
- The algorithm **randomly selects a move based on these probabilities**.

Distinctive Feature: Moves are selected **probabilistically**, with better moves having higher chances, but **there's still a possibility of choosing a suboptimal move**.

Random-Restart Hill Climbing: Possible Set of Actions for Random-Restart Hill Climbing:

Randomly restart the hill climbing process with a completely new random board configuration when a local minimum is reached.

Example:

- Start with a random configuration and apply **steepest-ascent hill climbing**.
- If you reach a local minimum (a state where no further improvements can be made), **restart** with a completely new random configuration.
- Repeat the process until you find a solution with 0 conflicts.

Distinctive Feature: The algorithm **restarts from scratch** when stuck in a local minimum, avoiding the possibility of staying stuck in a suboptimal configuration.