Chapter 3: Local and Heuristic Search Methods

In this chapter we move away from “blind” search methods (such as DFS and BFS) and explore search techniques that work in the solution (or plan) space. We begin by formulating problems so that every candidate configuration is a potential solution. Then we study local search methods—including hill climbing and its variants—and finally introduce heuristic search methods that use domain-specific information to guide the search more directly toward the goal.

3.1 Introduction

When solving complex problems in Artificial Intelligence, it is often inefficient or impractical to exhaustively explore the entire state space. In many configuration problems (e.g., SAT, N-Queens, TSP, Blocks World), every candidate configuration may itself be a solution if it satisfies the constraints. This leads us to the idea of **solution space search**—searching directly among candidate solutions.

Local search methods work by starting with an initial candidate and then “moving” to a neighbor that appears better according to some evaluation or heuristic function. However, a key drawback is that pure local search (or hill climbing) may become trapped in a local optimum or plateau. To overcome these challenges, several deterministic techniques have been developed that combine exploitation (following the gradient) with exploration (searching for new promising regions).

In this chapter we describe two major strands:

- **Solution Space Search & Perturbation:** How to represent candidates and generate neighbors.
- **Local Search & Heuristic Search:** How to guide the search using evaluation functions (heuristics) and techniques to escape local optima.

3.2 Formulating the Problem: Solution Space Search

3.2.1 What Is a Solution Space?

In solution space search the focus is not on finding a path from an initial state to a goal state but on finding a candidate configuration that satisfies the problem’s constraints. Such problems include:

- **Configuration Problems:** e.g., N-Queens, SAT, Sudoku, map colouring.
- **Planning Problems:** In some cases the plan (the sequence of actions) itself is the candidate, and search is done in the plan space.

Every node in the solution space is a complete candidate solution. When a candidate satisfies the goal description, the search is complete—we need not reconstruct an action sequence.

3.2.2 Synthesis Versus Perturbation

There are two broad ways to generate candidates:

- **Synthesis (Constructive Methods):** Start from the initial state and build the solution piece by piece. For example, when solving the N-Queens problem, you might place queens one row at a time.
- **Perturbation:** Start with a complete candidate (which may not be a solution) and modify it to obtain a new candidate. For example, representing an N-Queens solution as a one-dimensional array (where the (i)th element denotes the column position of the queen in row (i)), a permutation of that array produces a new candidate. Similarly, in SAT each candidate can be represented as an N-bit string (with 0/1 for false/true), and a neighbor may be generated by flipping one or more bits.

3.2.3 Examples

The SAT Problem

- **Problem Statement:**
  Given a Boolean formula (typically in Conjunctive Normal Form, CNF) over variables ({a, b, c, \dots}), find an assignment (a candidate, represented as an N-bit string) that makes the formula true.
- **Candidate Space:**
  With (N) variables there are (2^N) possible candidates.
- **Neighbourhood Function:**
  For example, a simple neighborhood function (N_1) might flip exactly one bit, so every candidate has (N) neighbours.

The Travelling Salesman Problem (TSP)

- **Problem Statement:**
  Given a set of cities and a cost (distance) between every pair, find a Hamiltonian cycle (a tour visiting each city exactly once) of minimum total cost.
- **Candidate Representation:**
  Every permutation of the cities represents a candidate tour. (Note that because of symmetry, the number of distinct tours is ((N-1)!/2).)
- **Complexity:**
  The candidate space grows factorially with (N) (i.e., (N!)), which is much larger than the exponential growth of SAT.
- **Constructive Methods:**
  Greedy methods are often used to construct a tour:
  - **Nearest Neighbour Heuristic:** Start at a city and always move to the nearest city that does not close the tour prematurely.
  - **Greedy Heuristic:** Sort all edges by cost and add the shortest available edge that does not create a cycle (until the tour is complete).
  - **Savings Heuristic:** Begin with (N-1) subtours (each of length 2, anchored at a base city) and then merge them by selecting pairs of edges to remove that yield maximum “savings” in total cost.

3.3 Local Search Algorithms

3.3.1 Hill Climbing

Hill Climbing is the simplest local search algorithm. It starts with an initial candidate (or solution) and repeatedly moves to the “best” neighbor—that is, the one with the best (lowest, if minimizing) evaluation—provided that the neighbor is an improvement over the current candidate.

Pseudocode: Hill Climbing

```
HillClimbing(S):
  N ← S
  do:
      bestEver ← N
      newNode ← best(sorth(MoveGen(N)))
      if h(newNode) < h(N):  // For minimization; use > for maximization
         N ← newNode
      else:
        break
  while true
  return N
```

Notes:

- **Space Efficiency:** Hill Climbing requires only constant space as it only keeps the current candidate and generates a small, fixed number of neighbors.
- **Time Efficiency:** It is linear in the number of steps taken.
- **Drawback:** It may become trapped in a local optimum (or plateau) where no neighbor is better, even if the global optimum is far away.

3.3.2 An Example: The 8-Puzzle

For the 8-puzzle, we often use heuristic functions such as:

- **Hamming Distance:** The number of tiles out of place.
- **Manhattan Distance:** The sum of the distances (in grid moves) of the tiles from their goal positions.

As the puzzle is solved, the heuristic value should ideally decrease to zero. However, it is possible for hill climbing to encounter a state where every move increases the heuristic value (a local minimum) even though the goal is not reached.

3.4 Escaping Local Optima

Because hill climbing can get stuck in local optima, several deterministic methods have been developed to introduce exploration. These methods modify the basic hill climbing approach so that the algorithm can “escape” from local optima.

3.4.1 Variable Neighbourhood Descent

This method uses multiple neighborhood functions ordered from sparse (few neighbors) to dense (many neighbors). The idea is to perform hill climbing using the sparse neighborhood first; if no improvement is possible, switch to a denser neighborhood to explore further.

Pseudocode: Variable Neighbourhood Descent

```
VariableNeighbourhoodDescent(S):
  bestNode ← S
  for i ← 1 to n do:
    moveGen ← MoveGen_i   // Neighborhood function that perturbs i bits (or elements)
    bestNode ← HillClimbing(bestNode, moveGen)
  return bestNode
```

3.4.2 Best Neighbour Search

Rather than only moving if a neighbor is strictly better, this variant always moves to the best neighbor found, even if it isn’t an improvement. An external termination criterion (e.g., maximum iterations) must be used.

Pseudocode: Best Neighbour Search

```
BestNeighbourSearch(S):
  N ← S
  bestSeen ← S
  while not terminated do:
      N ← best(MoveGen(N))
      if h(N) < h(bestSeen):
          bestSeen ← N
  return bestSeen
```

3.4.3 Tabu Search

Tabu Search introduces memory (a “tabu list”) that prevents the algorithm from immediately reversing recent moves. This helps the search escape cycles and local optima.

- **Tabu Tenure:** A fixed number of iterations during which a recently changed component (e.g., a bit in a SAT candidate) is forbidden from being changed again.
- **Aspiration Criterion:** Even if a move is tabu, it may be allowed if it results in a candidate better than any seen before.

Pseudocode: Tabu Search (Outline)

```
TabuSearch(S):
  N ← S
  bestSeen ← S
  Initialize TabuList (memory array) to zeros
  while not terminated do:
      allowedNeighbours ← { x in MoveGen(N) | x is not tabu or satisfies aspiration }
      N ← best(allowedNeighbours)
      Update TabuList (e.g., set tabu tenure for changed components)
      if h(N) < h(bestSeen):
         bestSeen ← N
  return bestSeen
```

3.4.4 Iterated Hill Climbing

This method runs hill climbing repeatedly from different random starting points. The best overall solution is returned after a fixed number of iterations.

Pseudocode: Iterated Hill Climbing

```
IteratedHillClimbing(numIterations):
  bestNode ← randomCandidate()
  for i ← 1 to numIterations do:
      current ← HillClimbing(randomCandidate())
      if h(current) < h(bestNode):
         bestNode ← current
  return bestNode
```

3.4.5 Neighbourhood Operators in TSP

For the Travelling Salesman Problem, candidate tours are represented as sequences of cities. Common perturbation operators include:

- **2-City Exchange:** Swap the order of two cities.
- **2-Edge Exchange:** Remove two edges and reconnect the tour (aiming to remove long edges).

- **3-Edge Exchange:** Remove three edges and reconnect the resulting segments in one of several possible ways.

These operators define the neighborhood and can be applied in a local search framework to improve a candidate tour.

3.5 Heuristic Search

While local search methods such as hill climbing use a heuristic evaluation to guide improvements, they typically focus only on the local neighborhood. In contrast, **heuristic search** methods use the heuristic function to globally direct the search toward the goal.

3.5.1 The Heuristic Function

A heuristic function, denoted (h(n)), is an estimate of the distance or cost from node (n) to the goal. Properties include:

- (h(\text{goal}) = 0)
- The function is defined by the user and depends on the problem domain.
- Examples:
  - **8-Puzzle:** Hamming distance (number of tiles out of place) or Manhattan distance.
  - **Geographical Routing:** Euclidean or Manhattan (city-block) distance between coordinates.

3.5.2 Best First Search

Best First Search uses a priority queue (the OPEN list) ordered by the heuristic value (h(n)). At each step, it selects the candidate with the lowest (h(n)) value.

Pseudocode: Best First Search

```
BestFirstSearch(S):
  OPEN ← [(S, null, h(S))]
  CLOSED ← []
  while OPEN is not empty do:
      nodeTriple ← head(OPEN)    // nodeTriple = (N, parent, h(N))
      if GoalTest(N) is true then:
          return ReconstructPath(nodeTriple, CLOSED)
      else:
          CLOSED ← CLOSED ∪ {nodeTriple}
          neighbours ← MoveGen(N)
          newNodes ← RemoveSeen(neighbours, OPEN, CLOSED)
          newTriples ← MakeTriples(newNodes, N, h(·))
          // Merge newTriples into OPEN so that OPEN is sorted by h-value (lowest
first)
          OPEN ← MergePriority(newTriples, tail(OPEN))
  return failure
```

Notes:

- **Priority Queue:** In practice, OPEN is maintained as a priority queue.
- **Direction:** Best First Search “has a sense of direction” because the heuristic steers it toward nodes estimated to be closer to the goal.

- **Completeness & Quality:** If the heuristic is admissible and the search space is finite, Best First Search will eventually find a solution. However, it may not always find the optimal solution unless additional cost information (edge costs) is incorporated (as in A* search).

3.5.3 Comparison of Search Methods

- **Depth First Search (DFS):**
  Explores deeply, uses linear space, but has no sense of direction and may get trapped in infinite paths.
- **Breadth First Search (BFS):**
  Explores level by level, finds the shortest path (in terms of number of moves) but requires exponential space.
- **Best First Search:**
  Uses heuristic guidance to drive the search toward the goal. Its performance heavily depends on the quality of the heuristic. It typically requires less space than BFS but more than hill climbing.
- **Local Search (Hill Climbing):**
  Constant space and fast but vulnerable to local optima.

3.6 Demos and Practical Observations

In practice, algorithm behavior can be visualized on a demo platform. For example:

- **DFS Demo:**
  The algorithm expands nodes in a predetermined order (often following the order given by the MoveGen function) regardless of the goal's location.
- **BFS Demo:**
  The algorithm systematically expands nodes level by level from the start state until it finds the goal. It always returns the shortest path in terms of hops.
- **Best First Search Demo:**
  The algorithm "homes in" on the goal by selecting nodes with the smallest heuristic value. As the search progresses, nodes that seem closer to the goal (by (h(n))) are expanded first.
- **Hill Climbing Demo:**
  The algorithm moves from the current candidate to the best neighbor (following the gradient) but may become stuck on a local optimum. Different runs from different starting points can yield different outcomes.

Such demos illustrate that while blind search methods (DFS, BFS) follow fixed trajectories, heuristic search methods (Best First, hill climbing) adapt their paths based on the problem's "landscape" as defined by the heuristic function.

3.7 Summary and Conclusion

In this chapter we explored search methods that work directly in the solution space. We covered:

- **Solution Space Search:**
  How to formulate problems where every candidate is a complete configuration. We contrasted

synthesis (constructive) methods with perturbation (neighborhood-based) methods, using examples such as SAT and TSP.

- **Local Search Algorithms:**
  Hill Climbing is a constant-space, linear-time method that quickly moves “up the hill” (or down in minimization problems) but risks getting stuck in local optima.

- **Escaping Local Optima:**
  Deterministic methods—such as Variable Neighbourhood Descent, Best Neighbour Search, Tabu Search, and Iterated Hill Climbing—introduce exploration to help escape local maxima or plateaus. In TSP and SAT, specialized perturbation operators (e.g., 2-city exchange, 2-edge exchange) define the neighborhood.

- **Heuristic Search:**
  By incorporating a heuristic function (h(n)) (which estimates distance or cost to the goal), algorithms like Best First Search guide the search toward the goal. The quality of the heuristic strongly influences the performance, completeness, and solution quality.

These methods offer trade-offs in terms of space, time, and solution quality. Local search methods are fast and use little memory but may not always reach the global optimum, while heuristic search can provide better guidance at the cost of increased memory usage.

In the next chapter we will explore randomized methods (such as simulated annealing) which further enhance local search by introducing probabilistic decisions to escape local optima.

3.8 Exercises

1. **SAT Candidate Generation:**
   Implement a neighborhood function (N_1) for a SAT problem with (N) variables (i.e., flip one bit) and test your hill climbing algorithm on small CNF formulas.

2. **TSP Greedy Construction:**
   Implement the Nearest Neighbour and Savings Heuristic algorithms for the Travelling Salesman Problem. Compare the tours found by each method on small city sets.

3. **Local Search Variants:**
   For a given configuration problem (e.g., 8-puzzle or Blocks World), implement hill climbing, variable neighbourhood descent, and tabu search. Analyze in which cases each method escapes local optima.

4. **Heuristic Function Analysis:**
   For the 8-puzzle, compare the performance of Best First Search using the Hamming distance versus the Manhattan distance as the heuristic function. Discuss how each heuristic affects search order and solution quality.

This concludes Chapter 3 on Local and Heuristic Search Methods. The techniques presented here form the basis for many real-world optimization and planning problems in Artificial Intelligence.