ME 620: Fundamentals of Artificial Intelligence

Lecture 8: Informed Search Strategies - II

Shyamanta M Hazarika
Biomimetic Robotics and Artificial Intelligence Lab
Mechanical Engineering and M F School of Data Sc. & AI
IIT Guwahati

Admissible heuristics

- A heuristic is admissible if it never overestimates the cost of reaching a goal from any node.

  - According to this definition, even the null heuristic (which returns 0 for any node) is admissible!

- The only thing required for a heuristic to be admissible is that it never returns a value greater than the actual path cost to the nearest goal for any node.

Admissible heuristics

- A heuristic h(n) is admissible if for every node n,

$$h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to reach the goal state from n.

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic

Admissible heuristics

E.g., for the 8-puzzle

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

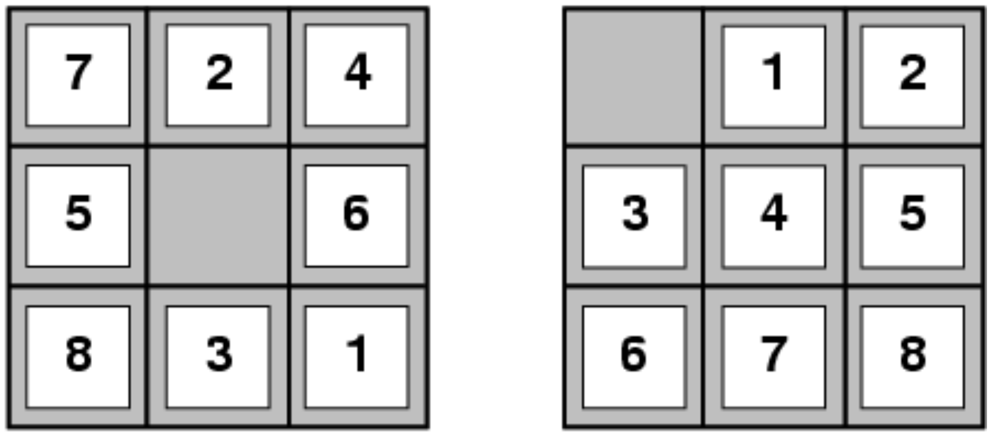
Goal State

- $h_1(S) = ?$
- $h_2(S) = ?$

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance



Start State Goal State

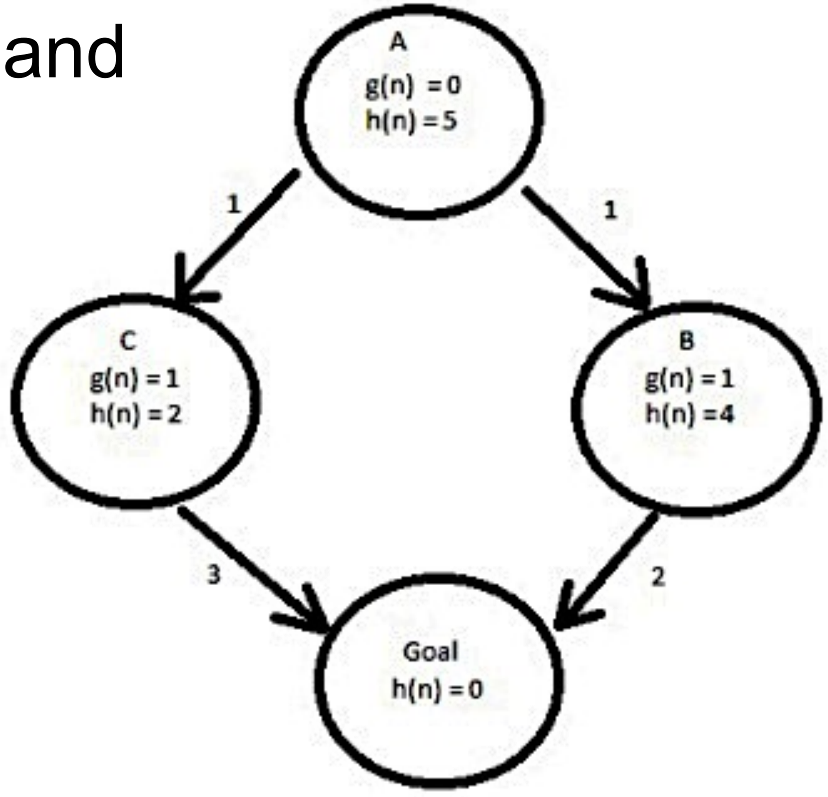
- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

Admissible vs. Inadmissible Heuristics

- Inadmissible heuristics
  - pessimistic heuristics because they overestimate the cost
  - break optimality by trapping good plans on the fringe.
- Admissible heuristics
  - optimistic heuristics because they can only underestimate the cost
  - can only slow down search by assigning a lower cost to a bad plan so that it is explored first but sooner or later search will find the optimal solution.

Inadmissibility breaks Optimality

Suppose node A has been expanded and nodes B and C are on the fringe.



Dominance

For two **admissible heuristics** h_1 and h_2

If $h_2(n) \geq h_1(n)$ for all n then h_2 **dominates** h_1

h_2 is **better** for search

A heuristic h_a is better than another h_b (i.e., h_a dominates h_b; $h_a \geq h_b$) if $\forall$ node n: $h_a(n) \geq h_b(n)$.

For example, for 8-puzzle, Manhattan distance dominates Hamming distance. Using a better heuristic means we have to explore fewer nodes before we find the solution.

Composite Heuristic

- The maximum of two admissible heuristics is also admissible!
- If we have developed two or more heuristics for the same problem and are unsure whether any of them dominates all others, we can use the maximum of them as the composite heuristic.
- This composite heuristic will take more time to compute but would be more accurate.

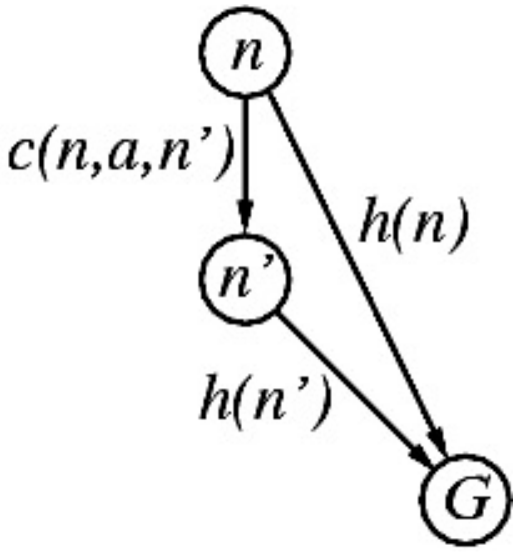
Consistent heuristics

- Consistent heuristics places even stricter constraints on the heuristic. It requires that the heuristic estimate must never be greater than the actual cost for each arc along a path to a goal.
  - One consequence of this is that the f value never decreases along a path to a goal.
- Consistency implies admissibility i.e., a consistent heuristic is also admissible (however the opposite is not necessarily true). Although, most admissible heuristics are consistent, especially if from relaxed problems.

Consistent heuristics

A heuristic is consistent if for every node n, every successor n' of n generated by any action a,

$$h(n) \leq c(n,a,n') + h(n')$$



Heuristics – a Trade-off

- Heuristics have a trade-off between quality of estimate and work per node. As heuristics get closer to the true cost, we expand fewer nodes but usually do more work per node to compute the heuristic itself.

- At the two extremes are the null heuristic (which always returns 0 and requires the least amount of work) and the exact cost (which can only be found out by conducting search itself and requires the most amount of work).

Choosing a Good Heuristic

- For a given problem, there might be many different heuristics one can choose. However, some heuristics are better than others.
  - We say the heuristic is better if it needs few nodes to examine in the search tree.
  - We also call this kinds of heuristics are better informed.
- Efficiency is Very important in Picking a Heuristic.
  - It is very important to consider the efficiency of running the heuristic itself.
  - More information about a problem may mean more time to process the information.

Admissibility of A*

- The A* algorithm, depending on the heuristic function, finds or not an optimal solution. If the heuristic function is admissible, the optimization is granted.
- A heuristic function is admissible if it satisfies the following property:

$$\forall n \; 0 \leq h(n) \leq h^*(n)$$

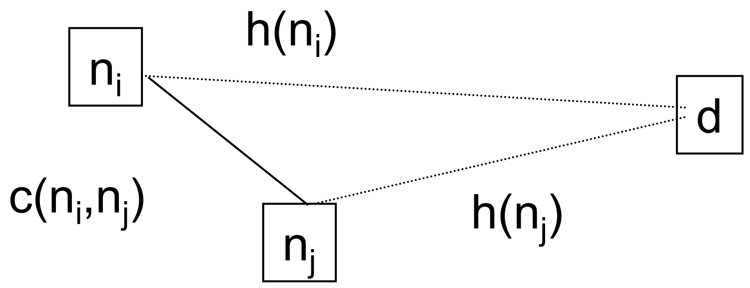
  h(n) has to be an optimistic estimator; it never has to overestimate h*(n).

- Using an admissible heuristic function guarantees that a node on the optimal path never seems too bad and that it is considered at some point in time.

Admissibility of A*

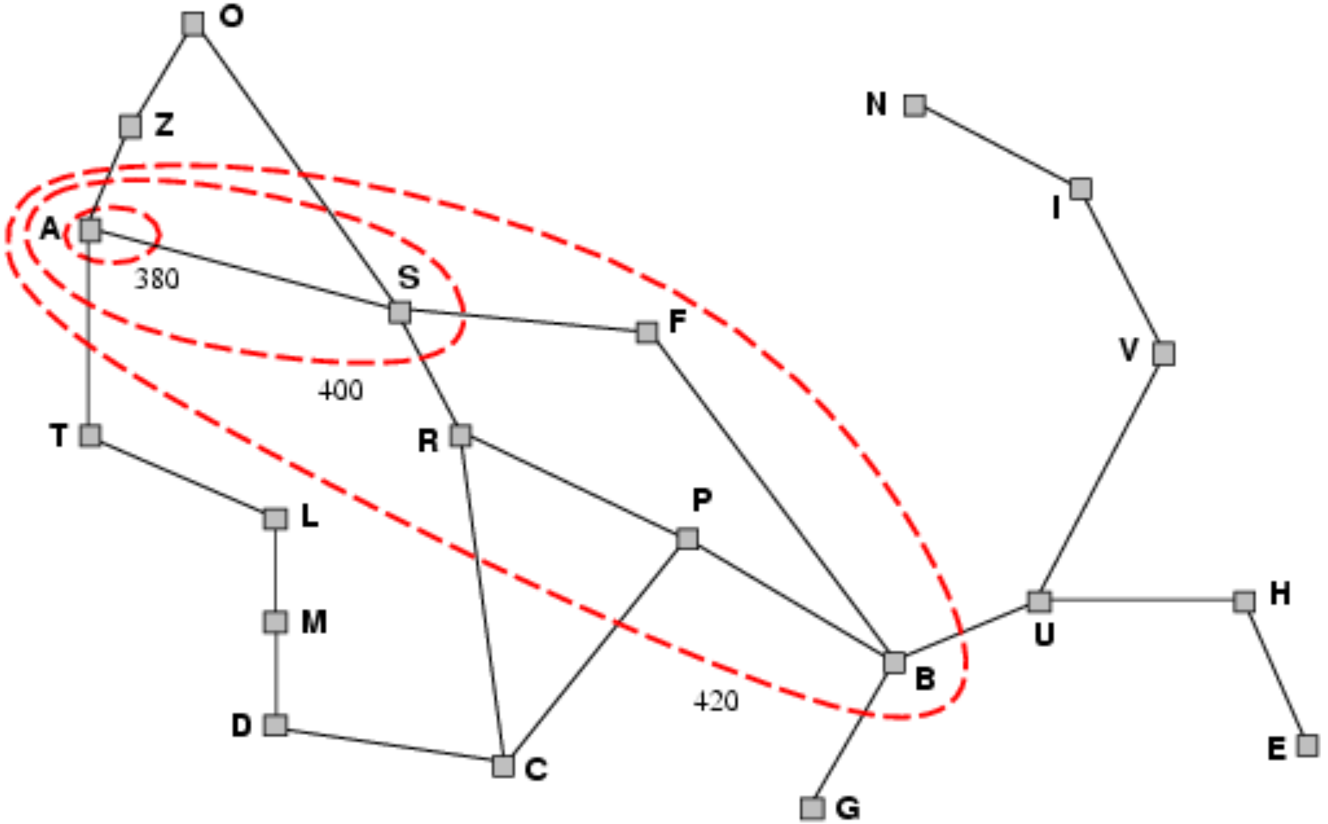
- A* generates an **optimal solution** if h(n) is an admissible heuristic and the search space is a tree:
  - h(n) is **admissible** if it never overestimates the cost to reach the destination node
- A* generates an **optimal solution** if h(n) is a consistent heuristic and the search space is a graph:
  - $h(n_i)$ is **consistent** if for every node n_i and for every successor node n_j of n_i:

$$h(n_i) \leq c(n_i,n_j) + h(n_j)$$



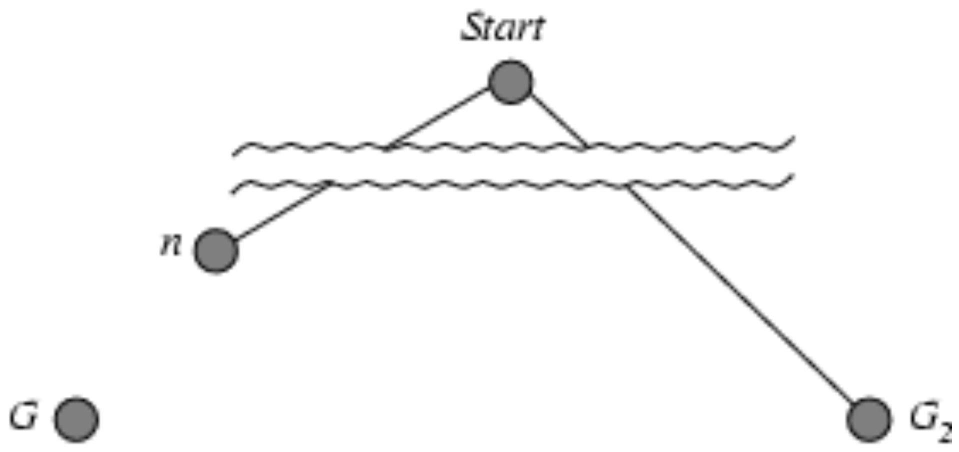
Optimality of A*

- A* expands nodes in order of increasing *f* value; Gradually adds *f*-contours of nodes.



Optimality of A*

- Suboptimal goal G_2 has been generated and is in the fringe.
- Let n be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.
- If A* is optimal, f(n) < f(G2)



$f(G_2) = g(G_2)$ since $h(G_2) = 0$

$f(G) = g(G)$ since $h(G) = 0$

$g(G_2) > g(G)$ since G_2 is suboptimal

$f(G_2) > f(G)$

$h(n) \leq h^*(n)$ since h is admissible

$g(n) + h(n) \leq g(n) + h^*(n)$

$f(n) \leq f(G)$ since $g(n) + h(n) = f(n)$ and $g(n) + h^*(n) = f(G)$

$f(n) < f(G2)$

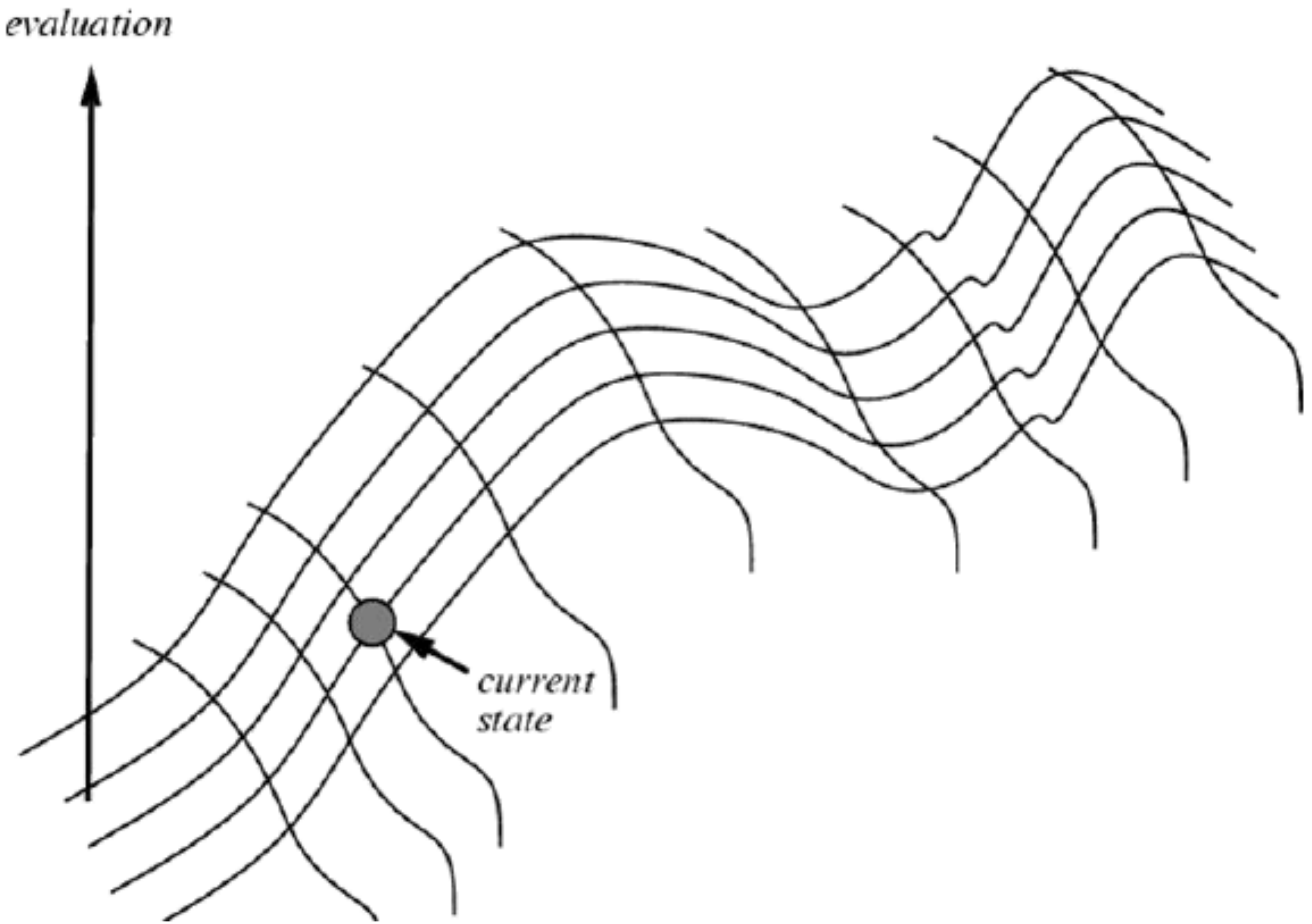
Local search algorithms

- In many optimization problems, the state space is the space of all possible complete solutions
  - Objective function tells us how “good” a given state is; Find the solution by minimizing or maximizing the value of this function
- These algorithms do **not** systematically explore all the state space.
  - Heuristic (or evaluation) function reduce the search space (not considering states which are not worth being explored).
  - Algorithms do not **usually** keep track of the path traveled.
    - The memory cost is minimal.
    - This total lack of memory can be a problem (i.e., cycles).

Local search algorithms

- Local search:
  - Use single current state and move to neighboring states.
- Idea: start with an initial guess at a solution and incrementally improve it until it is one
- Advantages:
  - Use very little memory
  - Find often *reasonable* solutions in large or infinite state spaces.
- Useful for pure optimization problems.
  - Find or approximate best state according to some objective function

Hill Climbing search



Hill Climbing search

Procedure: Hill-Climbing

Initialize current to starting state

Loop:

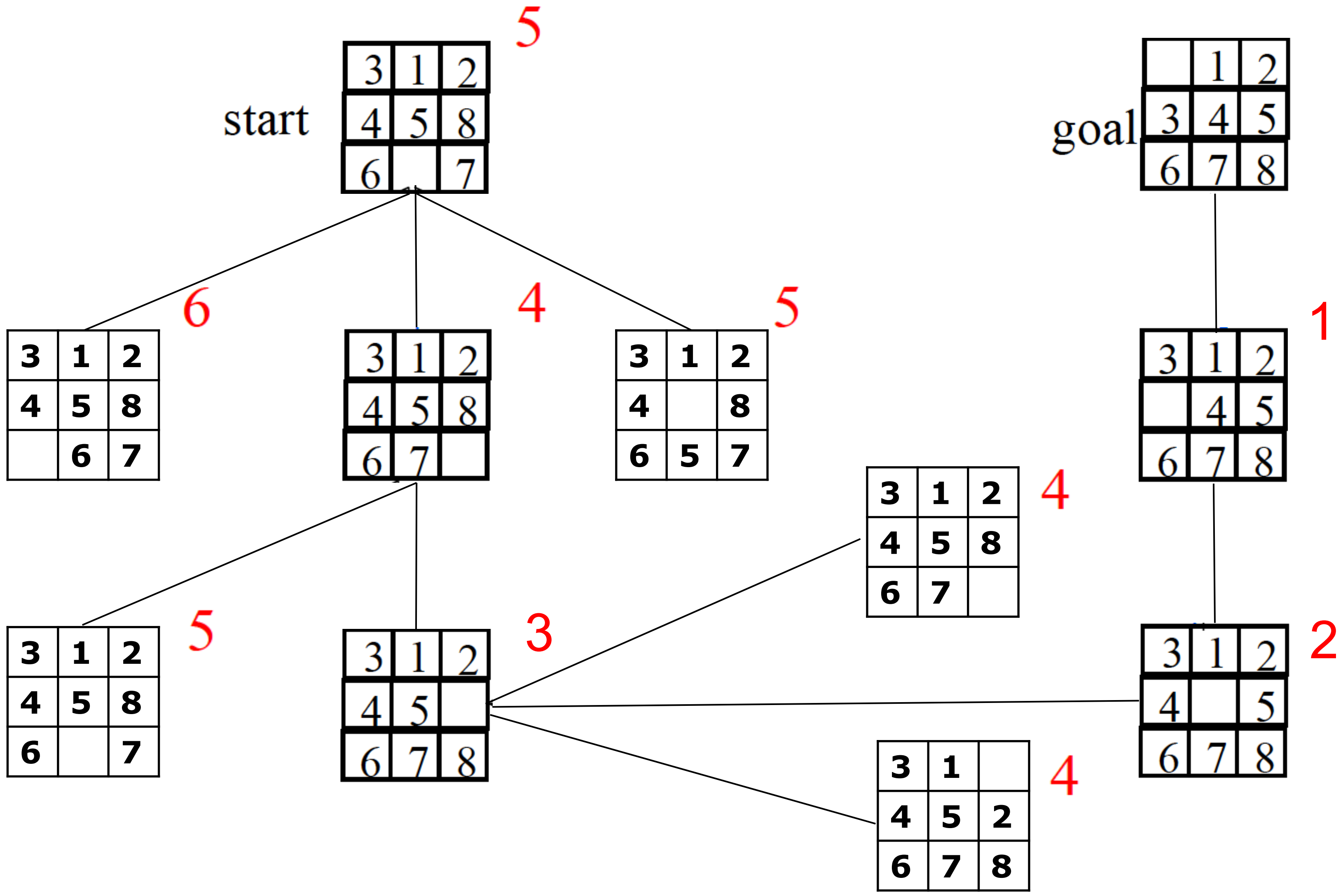
Let next = highest-valued successor of current

If value(next) < value(current) return current

Else let current = next

- Variants
  - **Simple Hill Climbing**
    - Basic method in which the first state that is better than current state is selected.
  - **Steepest-Ascent Hill Climbing**
    - Consider ALL moves from the current state and select the best one as the next state.

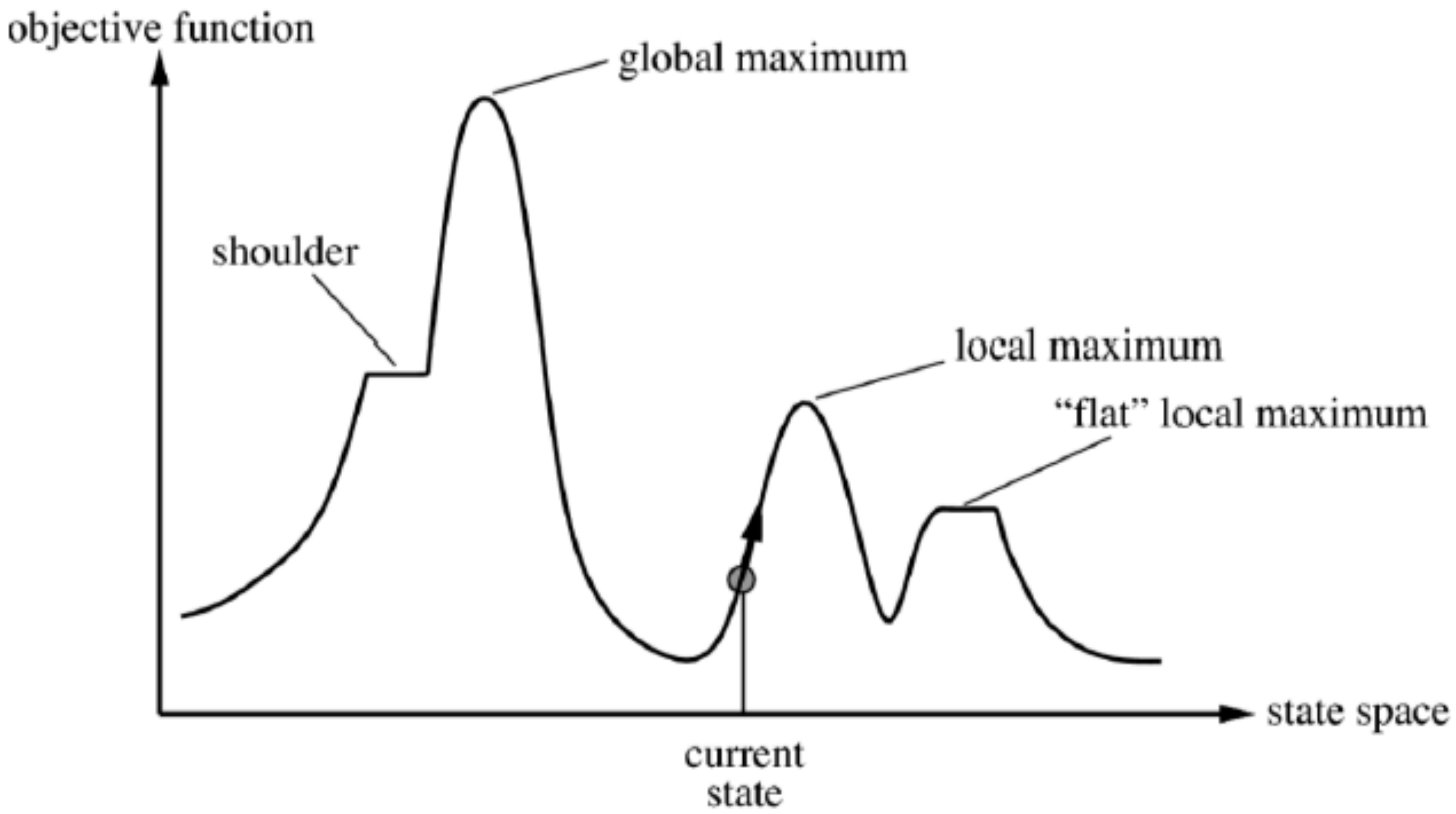
Hill Climbing Search



Hill Climbing search

- Properties:
  - Terminates when a peak is reached.
  - Does not look ahead of the immediate neighbors of the current state.
  - Chooses randomly among the set of best successors, if there is more than one.
  - Does not backtrack, since it doesn’t remember where it’s been

Search Space Features



Drawbacks of Hill Climbing

- **Local Maxima**: peaks that aren't the highest point in the space
  - State that is better than all its neighbours but is not better than some other states farther away.
- **Plateaus**: the space has a broad flat region that gives the search algorithm no direction (random walk)
  - Whole set of neighbouring states have the same value! Not possible to determine the best direction.
- **Ridges**: dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.
  - Orientation of the high region, compared to the set of available moves and directions in which they move, make it impossible to traverse ridges.