Chapter3

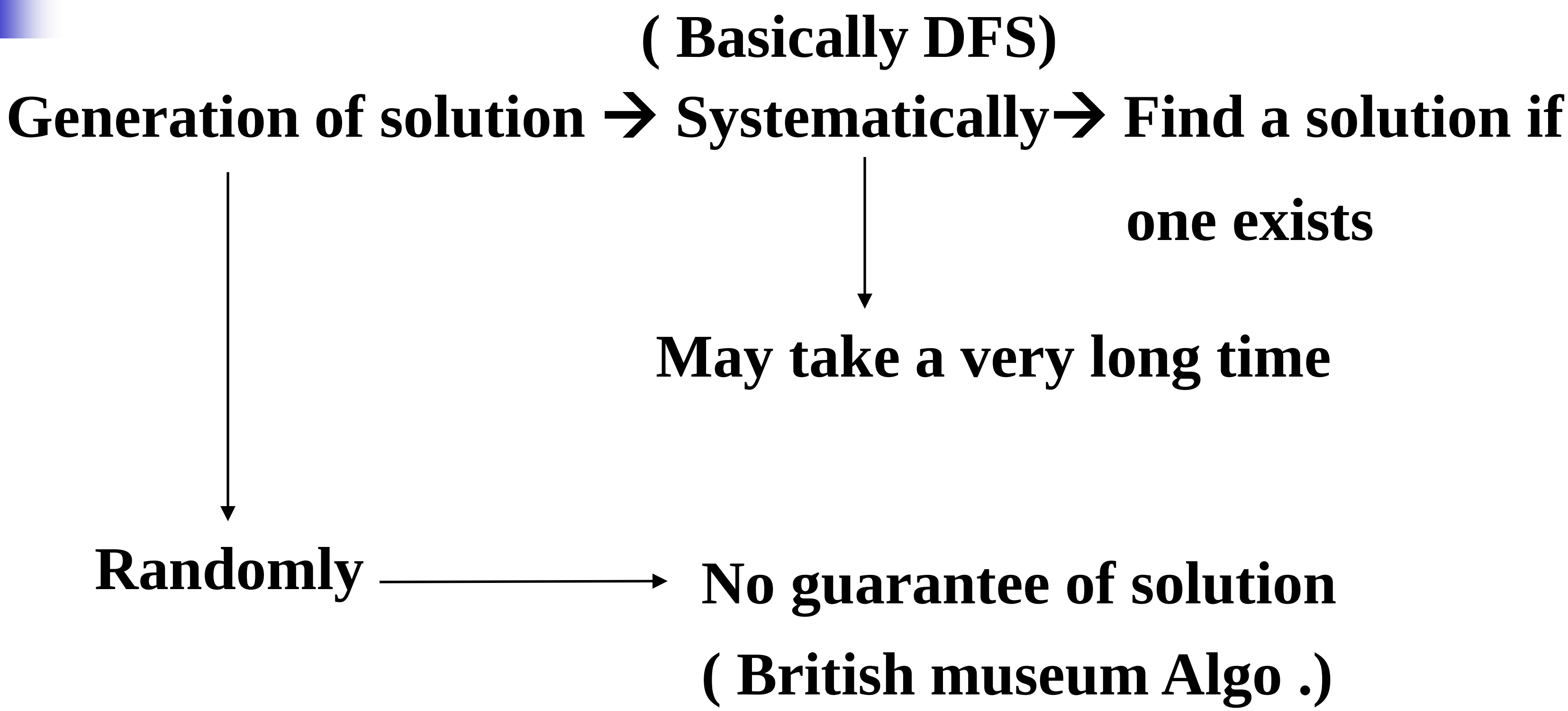
Heuristic Search Techniques

Generate-and-Test

- ❑ **Algorithm:**

  1. **Generate a possible solution(i.e. path/state)**
  2. **Test to see if this is actually a solution by comparing to a state/ a path**
  3. **If a solution has been found, quit. otherwise return to step 1**

Generate-and-Test



(Basically DFS)

Generation of solution → Systematically→ Find a solution if one exists

May take a very long time

Randomly

No guarantee of solution

(British museum Algo .)

Generate-and-Test

- ❑ Search process proceeds systematically, but some paths are not considered because they seem unlikely to lead to a solution. This evaluation is performed by a heuristic function

- ❑ For simple problems, exhaustive generate-and-test is often a reasonable technique

eg. Puzzle of four six-sided cubes,with each side of each cube painted one of four colors.arrangement of the cubes in a row such that on all four sides of the row one block face of each color is showing

- solved using exhaustively

- Using generate-and-test several configuration can be avoided

Generate-and-Test

- ❑ Observation: more red faces, then don’t use red color for cube face initially

- ❑ Generate-and-test not useful much for harder application but when combined with other techniques to restrict the space in which to search even further, the techniques can be very effective

- ❑ eg. AI→DENDERAL→Plan-Generate-Test

  ↓

  Used Constrain satisfaction technique

  Lists out recommended and contraindicated substructures

- ❑ Limitation of planning: Produce somewhat inaccurate solutions as no feedback is available after plan

- ❑ Still planning is used to avoid trying unnecessary exploration and avoid combinatorial explosion

Hill Climbing

- ❑ Variant of Generate-and-Test
- ❑ Feedback is used to decide which direction to move in search space
- ❑ Generate-and-Test: test responds: yes or no
- ❑ Hill Climbing: test $\xrightarrow{\text{has}}$ Heuristic function

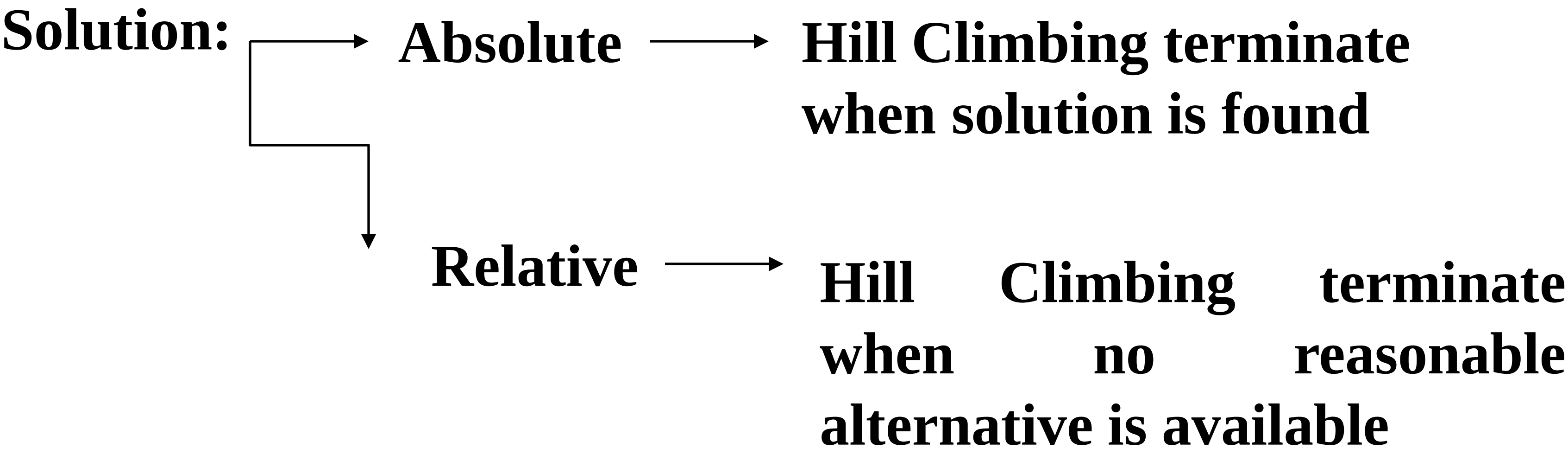
↓

Provides an estimation how close a given state is from goal state

- ❑ Hill Climbing is useful when good heuristic function is available for evaluating states but no other useful knowledge is available

Hill Climbing

eg. Finding/Getting downtown by searching and following high rise building

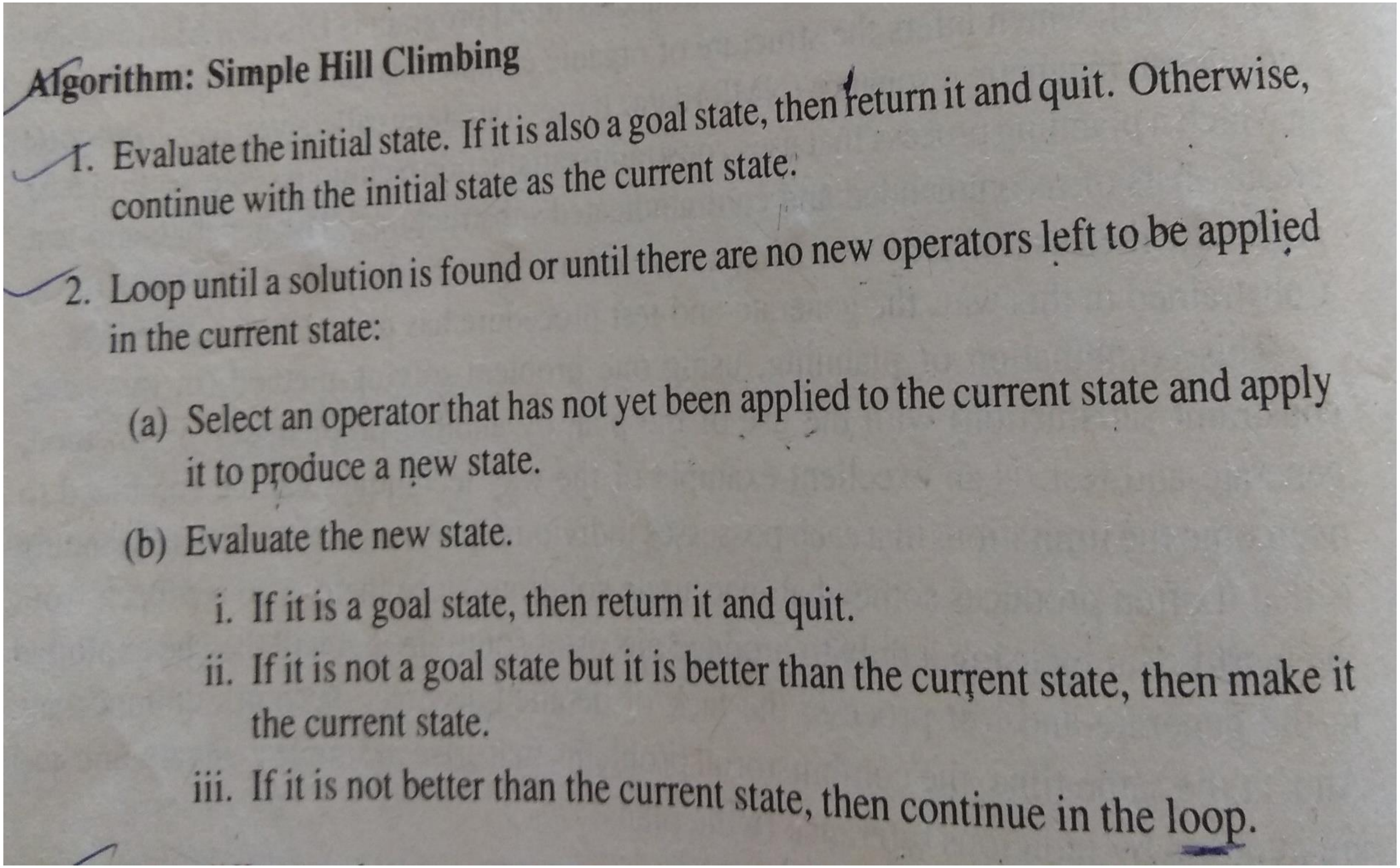


Solution:

Absolute → Hill Climbing terminate when solution is found

Relative → Hill Climbing terminate when no reasonable alternative is available

Hill Climbing



Algorithm: Simple Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   (a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.
   (b) Evaluate the new state.
      i. If it is a goal state, then return it and quit.
      ii. If it is not a goal state but it is better than the current state, then make it the current state.
      iii. If it is not better than the current state, then continue in the loop.

Hill Climbing

5		8
4	2	1
7	3	6

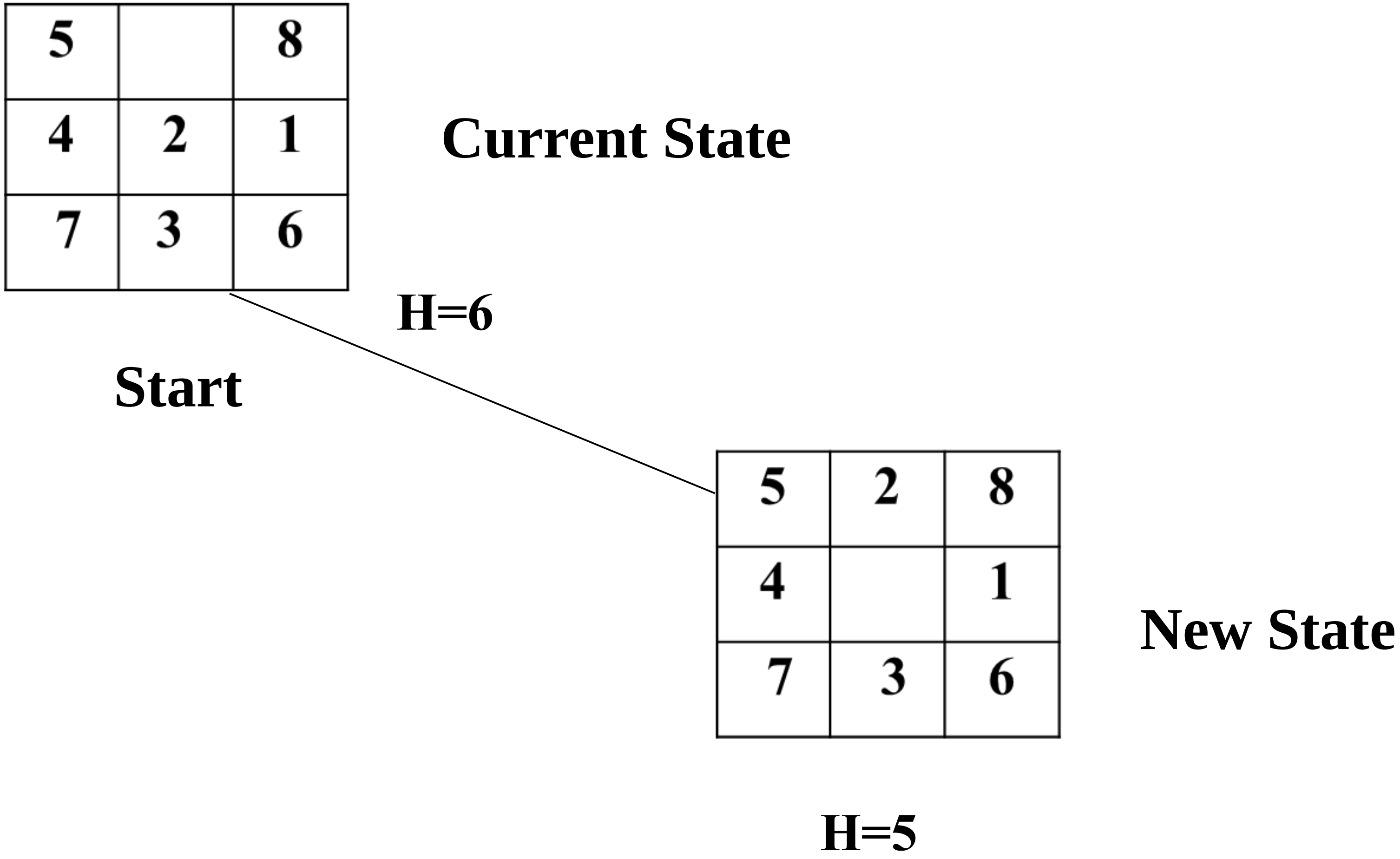
Start

1	2	3
4	5	6
7	8	

Goal

Heuristic1 = number of misplaced numbered tiles

Hill Climbing



5		8
4	2	1
7	3	6

Current State

H=6

Start

5	2	8
4		1
7	3	6

New State

H=5

New State better than current state

Hill Climbing

- ❑ Difference between hill climbing and generate-and-test is the use of evaluation function to inject task specific knowledge into the control process
- ❑ Knowledge gives power to solve some interactable problems
- ❑ heuristic function=evaluation function
- ❑ How to decide new state is better than current state?
  - Value returned by evaluation function
  - It can be higher the better/lower the better