



Local search algorithms

Local search algorithms

- In many optimization problems, the **path** to the goal is not relevant; finding the goal state is sufficient
 - State space = set of "complete" configurations
 - Example: n-queens
- In such cases, we can use **local search algorithms**
 - keep a single "current" state, tries to improve it
 - all previous states can be discarded
 - since only information about the current state is kept, such methods are called local

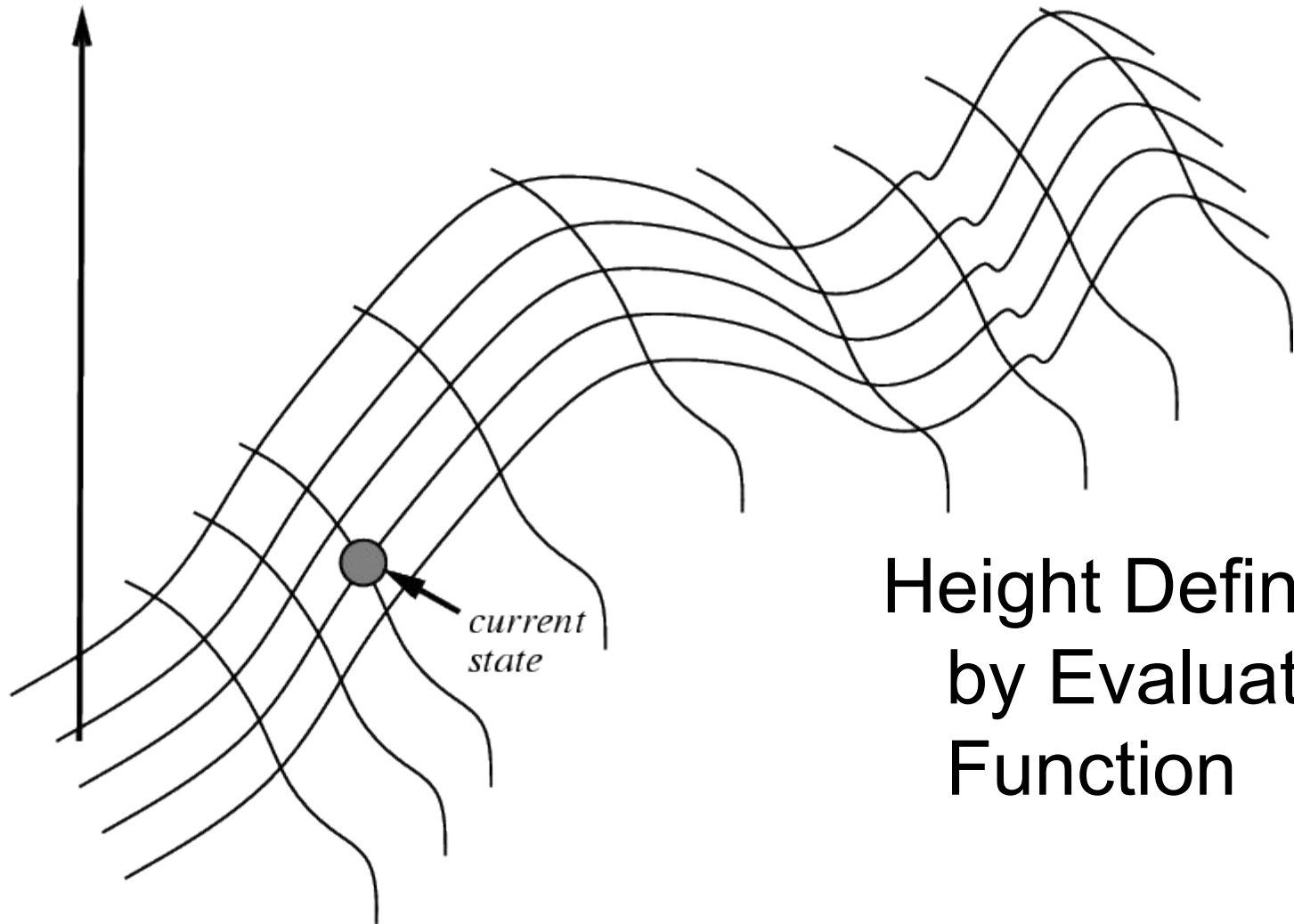


Local search algorithms

- Local search = use single current state and move to neighboring states.
- Some examples:
 - Hill climbing
 - Simulated annealing
 - Constraint satisfaction
- Advantages:
 - Use very little memory
 - Find often reasonable solutions in large or infinite state spaces.

Hill Climbing on a Surface of States

evaluation



Height Defined
by Evaluation
Function

Hill Climbing Search

- Search technique that continuously moves uphill (increasing value of an evaluation function h)
 - Terminates when a peak is reached
- If there exists a neighbor s for the current state n such that
 - $h(s) < h(n)$
 - $h(s) \leq h(t)$ for all the neighbors t of n ,then move from n to s . Otherwise, halt at n .
- Looks one step ahead to determine if any neighbor is better than the current state; if there is, move to the best neighbor.
 - Does not look “beyond the neighbors”
- Similar to Greedy search in that it uses h , but does not allow backtracking or jumping to an alternative path since it doesn’t “remember” where it has been.

Hill-climbing search

- "Like a blind man climbing mount Everest"

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node

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

8-puzzle (here we go again)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

[Note: optimal solution of n -Puzzle family is NP-hard]

8-puzzle (here we go again)

Remember the heuristics?

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)

7	2	4
5		6
8	3	1

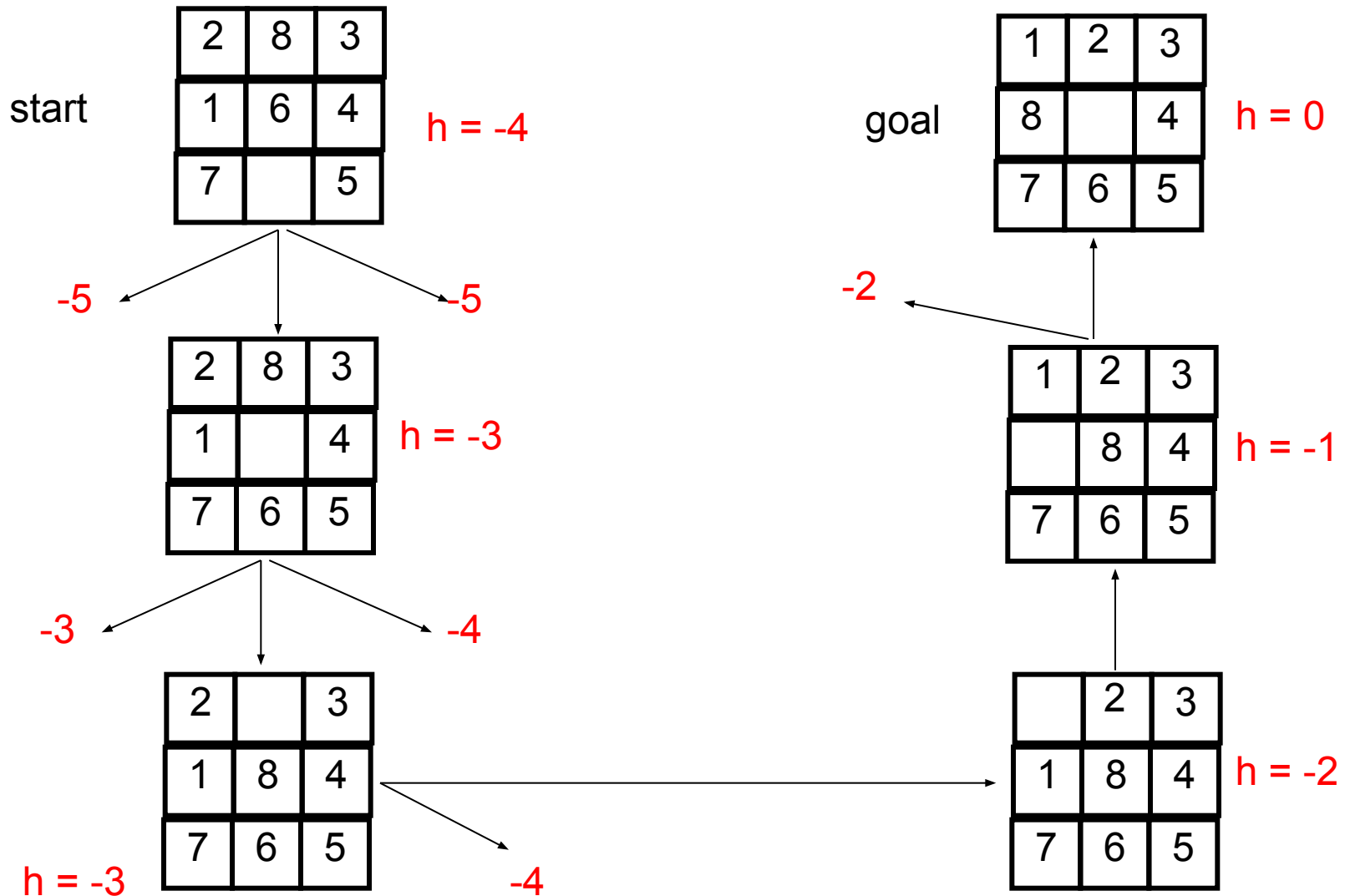
Start State

	1	2
3	4	5
6	7	8

Goal State

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

Hill Climbing Example



$$f(n) = -(\text{number of tiles out of place})$$

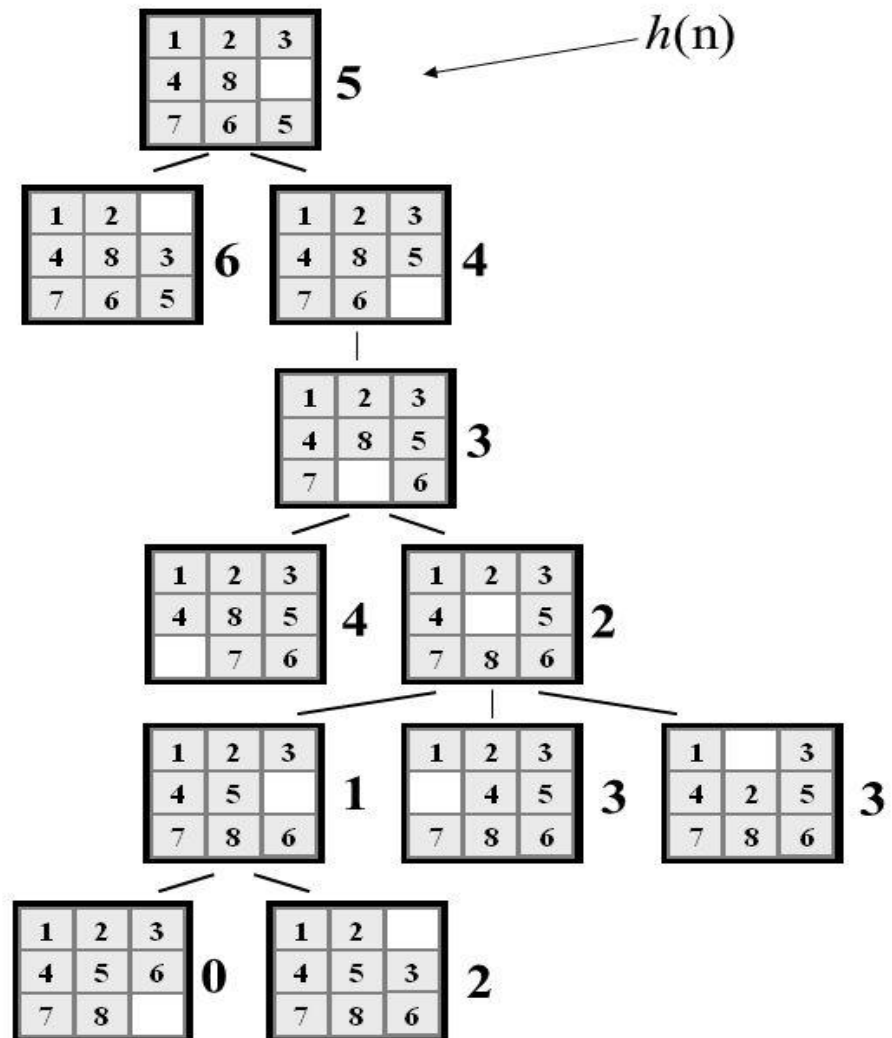
Hill Climbing Example

Hill climbing with minimization goal:
Here, the objective function is
 $\min f$ Where, $f = h(n) = (\text{manhattan distance})$

We can use heuristics
to guide “hill climbing”
search.

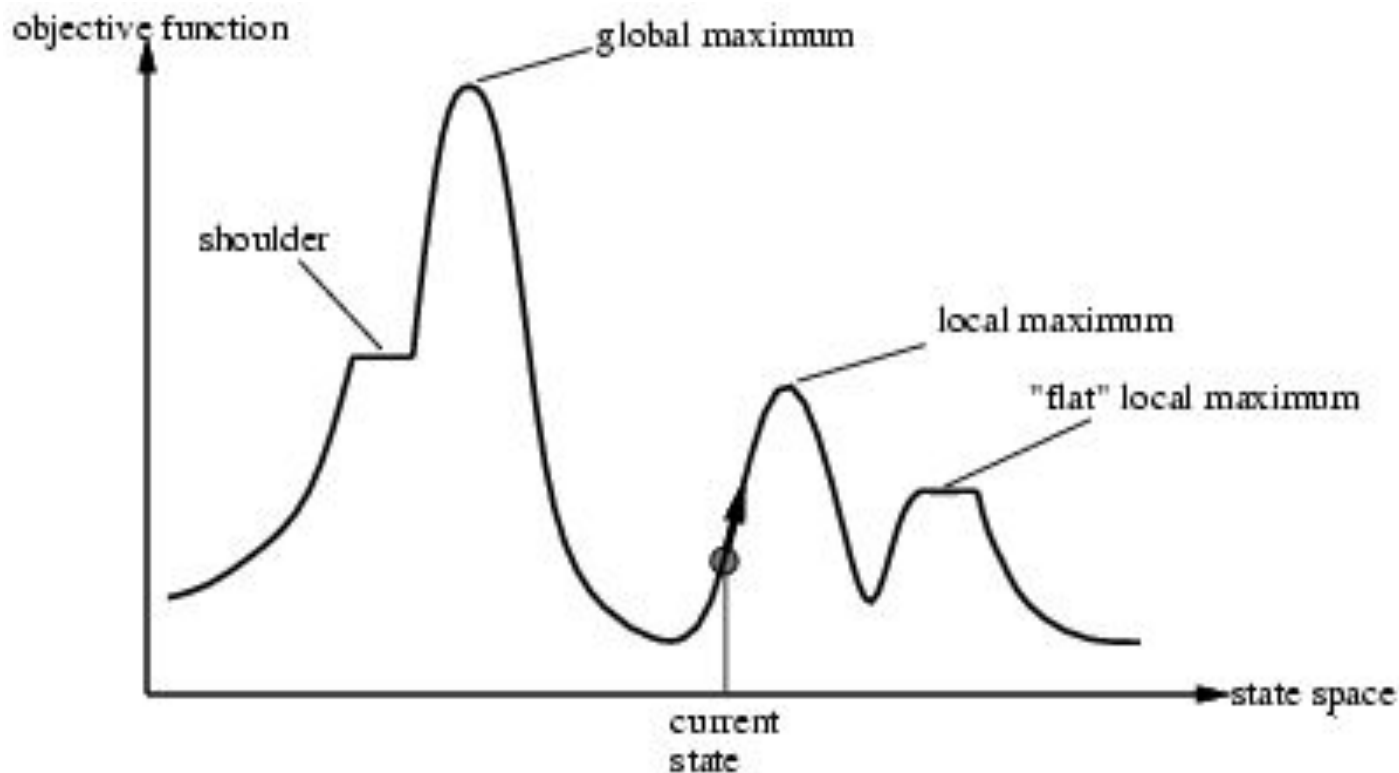
In this example, the
Manhattan Distance
heuristic helps us
quickly find a solution
to the 8-puzzle.

But “hill climbing has
a problem...”



Drawbacks of Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima



Exploring the Landscape

- **Local Maxima:** peaks that aren't the highest point in the space
- **Plateaus:** the space has a broad flat region that gives the search algorithm no direction (random walk)
- **Ridges:** flat like a plateau, but with drop-offs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.

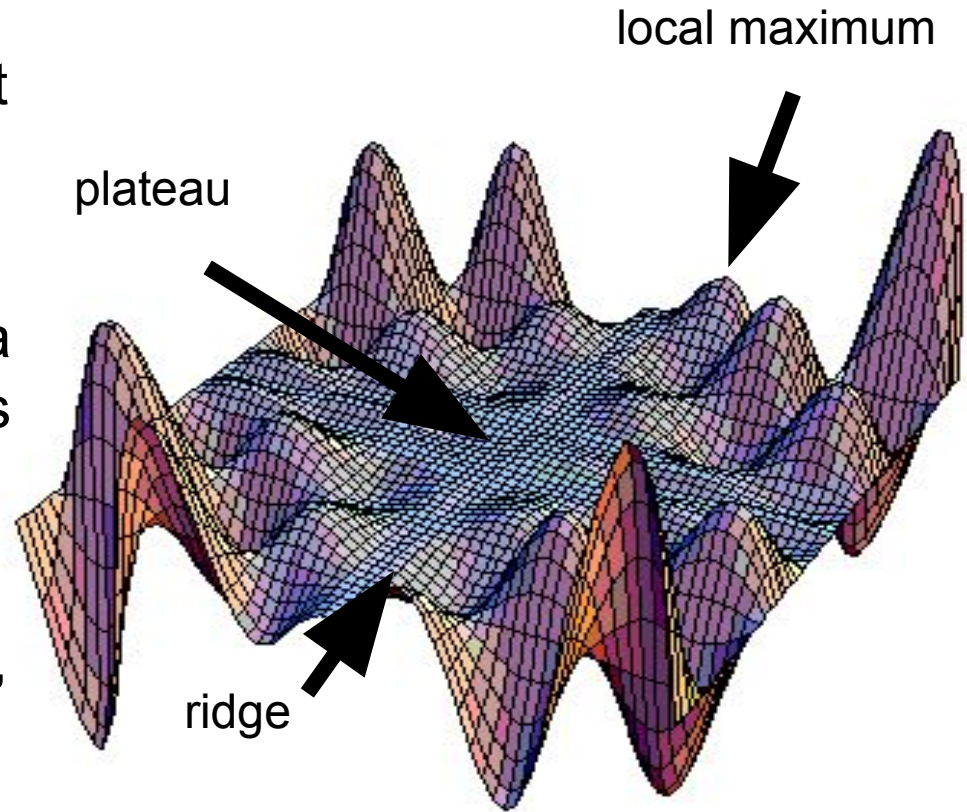
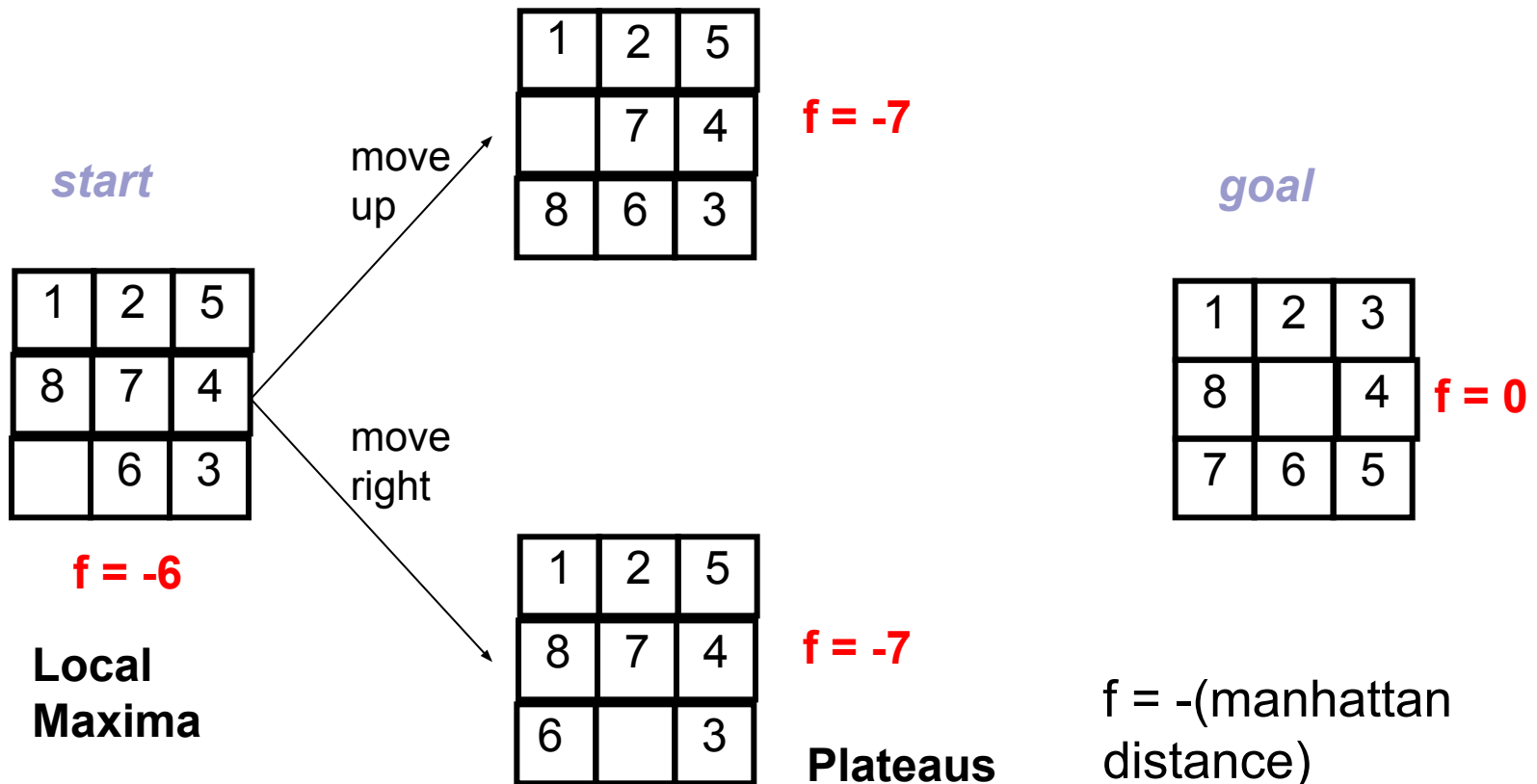


Image from:
<http://classes.yale.edu/fractals/CA/GA/Fitness/Fitness.html>

Example of a Local Optimum



Example: n -queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

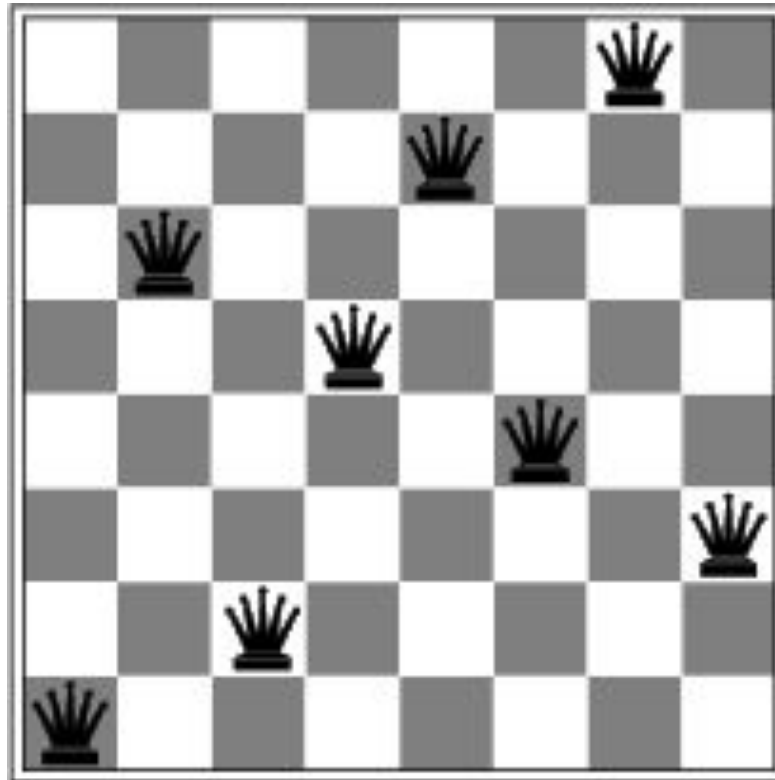


Hill-climbing search: 8-queens problem

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♚	13	16	13	16
♚	14	17	15	♚	14	16	16
17	♚	16	18	15	♚	15	♚
18	14	♚	15	15	14	♚	16
14	14	13	17	12	14	12	18

- h = number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state

Hill-climbing search: 8-queens problem



A local minimum with $h = 1$



Remedies of Hill Climbing Search

- Problems: local maxima, plateaus, ridges
- Remedies:
 - **Random restart:** keep restarting the search from random locations until a goal is found.
 - **Accept worse neighbours:** even if neighbour is worse, accept with some fixed probability p
 - **Simulated Annealing**
- Some problem spaces are great for hill climbing and others are terrible.

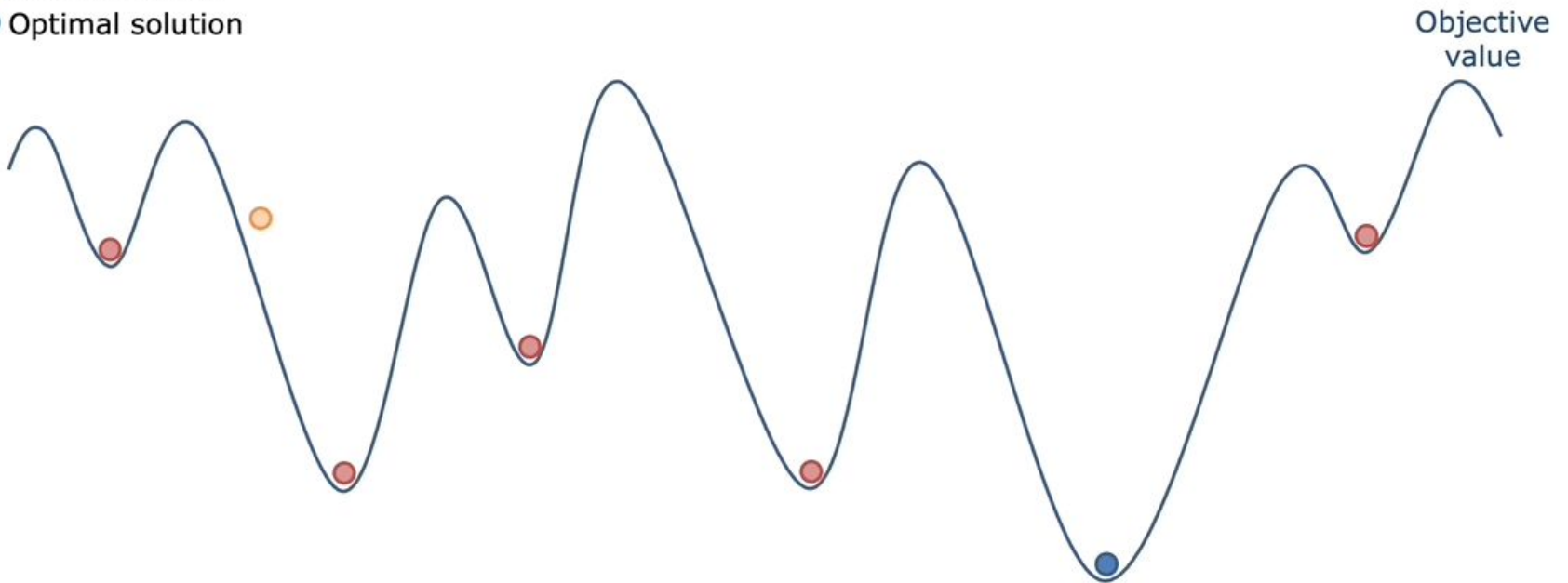
Simulated Annealing

- Idea: escape local maxima by **allowing some "bad" moves** but **gradually decrease** their frequency
- The algorithm is based on the metallurgical process of **annealing**, which is used to strengthen metals or remove internal stresses
- Metal is first heated to a high temperature so that its atoms can move freely
- Then the metal is slowly cooled down so that the atoms gradually settle into a stable, low-energy crystal structure
- Bouncing ball analogy
 - Shaking hard (= high temperature)
 - Shaking less (= lower the temperature)

Simulated Annealing

Escape local minima

- Current solution
- Local minimum
- Optimal solution



Simulated Annealing

- SA uses a random search that accepts changes that increase objective function f , as well as some that decrease it
- SA uses a control parameter T , which by analogy with the original application is known as the system “temperature”
- T starts out high and gradually decreases toward 0.
- If T decreases slowly enough, best state is reached

Simulated annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **return** a solution state

input: *problem*, a problem

schedule, a mapping from time to temperature

local variables: *current*, a node.

next, a node.

T, a “temperature” controlling the probability of downward steps

current \leftarrow MAKE-NODE(INITIAL-STATE[*problem*])

for *t* \leftarrow 1 **to** ∞ **do**

T \leftarrow *schedule*[*t*]

if *T* = 0 **then return** *current*

next \leftarrow a randomly selected successor of *current*

$\Delta E \leftarrow$ VALUE[*next*] - VALUE[*current*]

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E / T}$

Simulated annealing

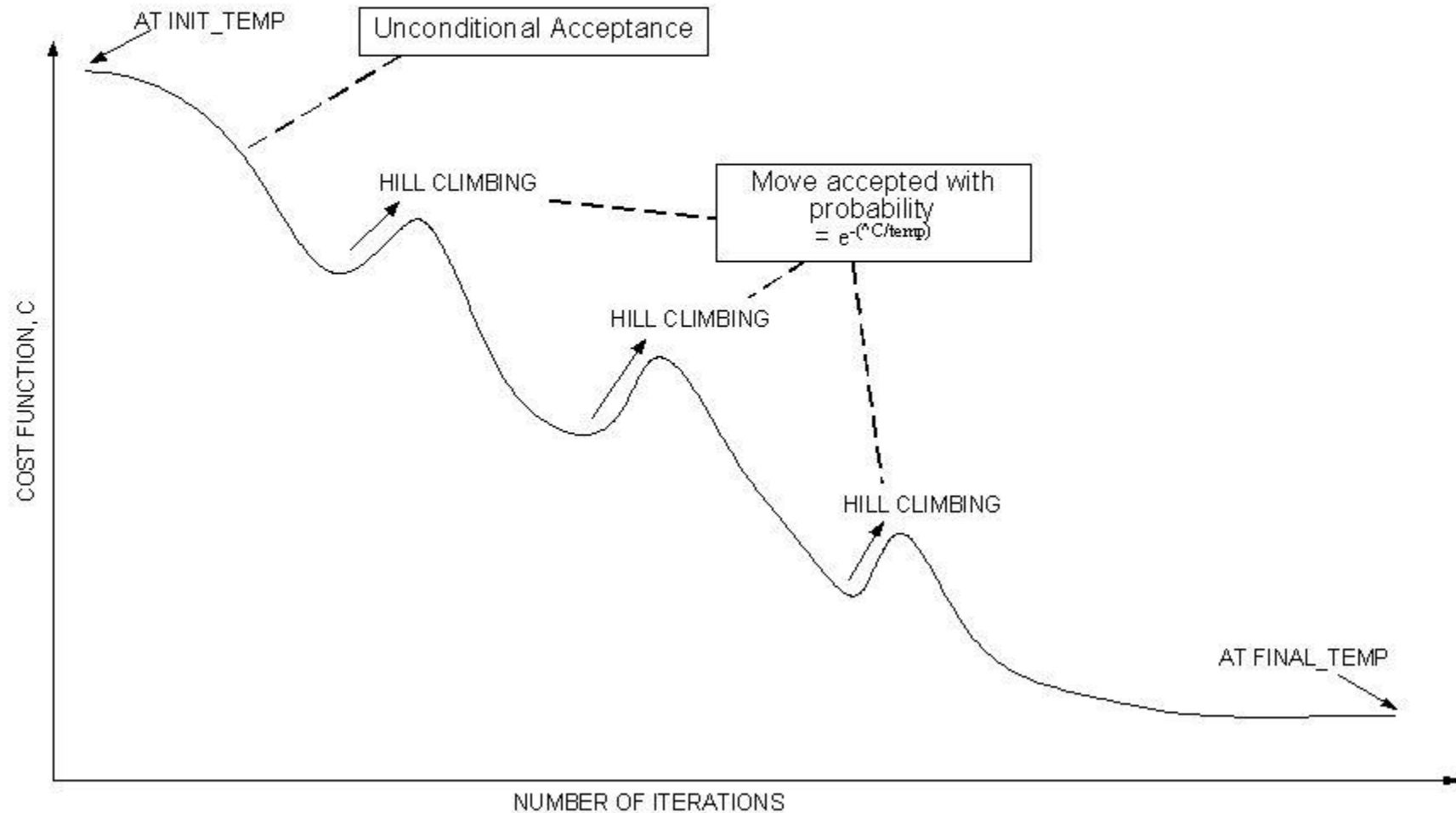
Time	Temperature	ΔE	Probability= $e^{\Delta E/T}$
1	30	-2	0.93551
2	20	-2	0.90484
3	10	-2	0.81873
4	9	-2	0.80074
5	8	-2	0.7788
6	7	-2	0.75148
7	6	-2	0.71653

Time	Temperature	ΔE	Probability= $e^{\Delta E/T}$
8	5	-2	0.67032
9	4	-2	0.60653
10	3	-2	0.51342
11	2	-2	0.36788
12	1	-2	0.13534
13	0	-2	0

Simulated Annealing (cont.)

- Note that for a “bad” move ΔE will be negative, so bad moves always have a relatively probability less than one.
- Good moves, for which ΔE is positive, have a relative probability greater than one.
- The higher the temperature, the more likely it is that a bad move can be made.
- As T tends to zero, this probability tends to zero, and SA becomes more like hill climbing
- If T is lowered slowly enough, SA is complete and admissible.

Convergence of simulated annealing



Properties of simulated annealing search

- One can prove: If T decreases slowly enough (infinitely slowly), then simulated annealing search will find a global optimum with probability approaching 1 [complete and optimal]
 - This would require letting SA run for infinite (very long) time – defeats the purpose
- Widely used in VLSI layout, airline scheduling, etc

Local search vs A* search

- A* search needs to know the goal state, local search does not
- A* finds the path to goal, local search does not
- What else?
 - Think about search strategy, memory usage, completeness, optimality