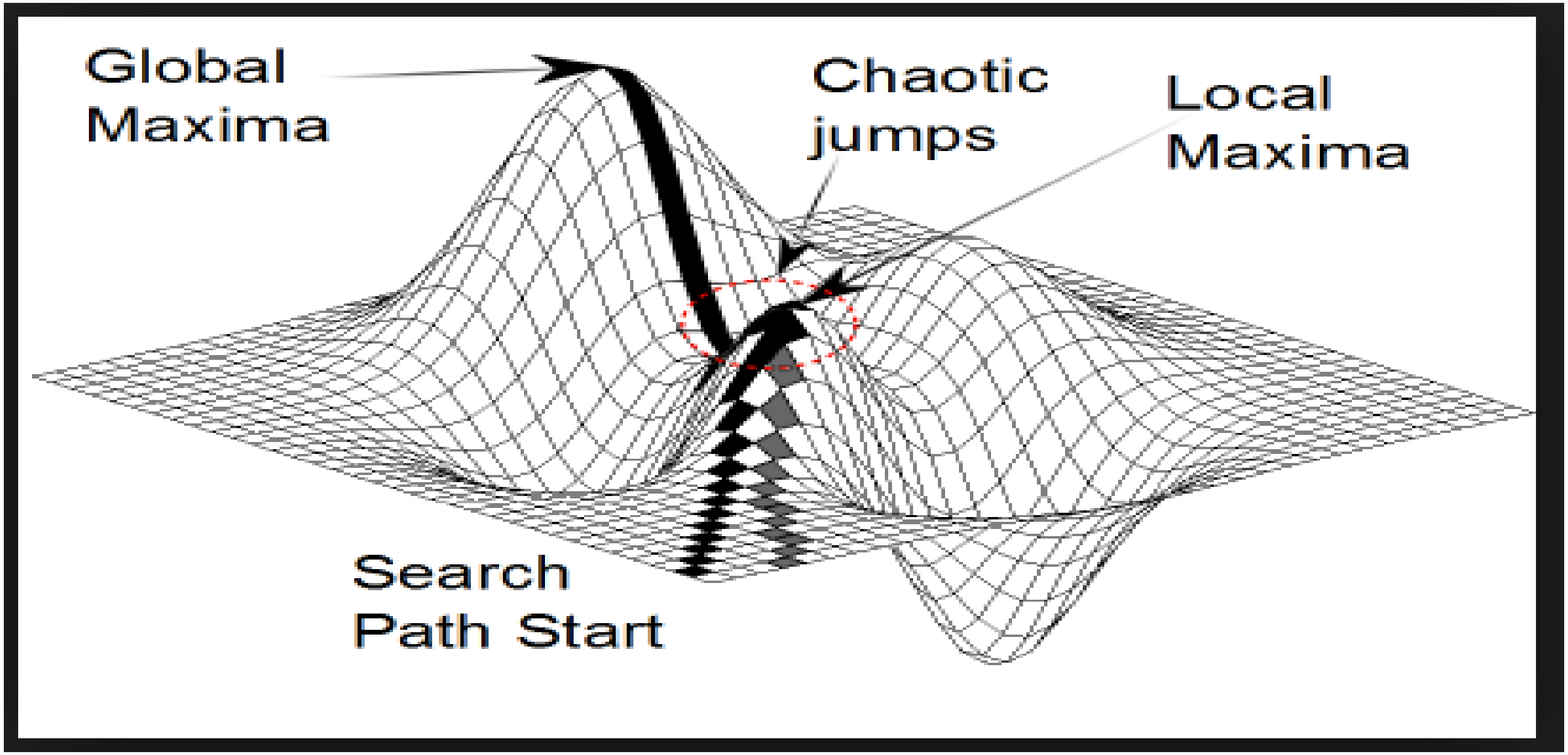
Local Search Algorithms & Optimization Problems

Beyond Classical Search – **Hill Climbing & Simulated Annealing**



Local Search Algorithms (LSA):

Previous search algorithms (classical search algorithms) use to explore search space systematically to find the optimal path to the goal

- ***Ex: TSP, Route finding problems etc.***

But, in many large & complex search problems, the path to the goal is irrelevant; the goal state itself is the solution

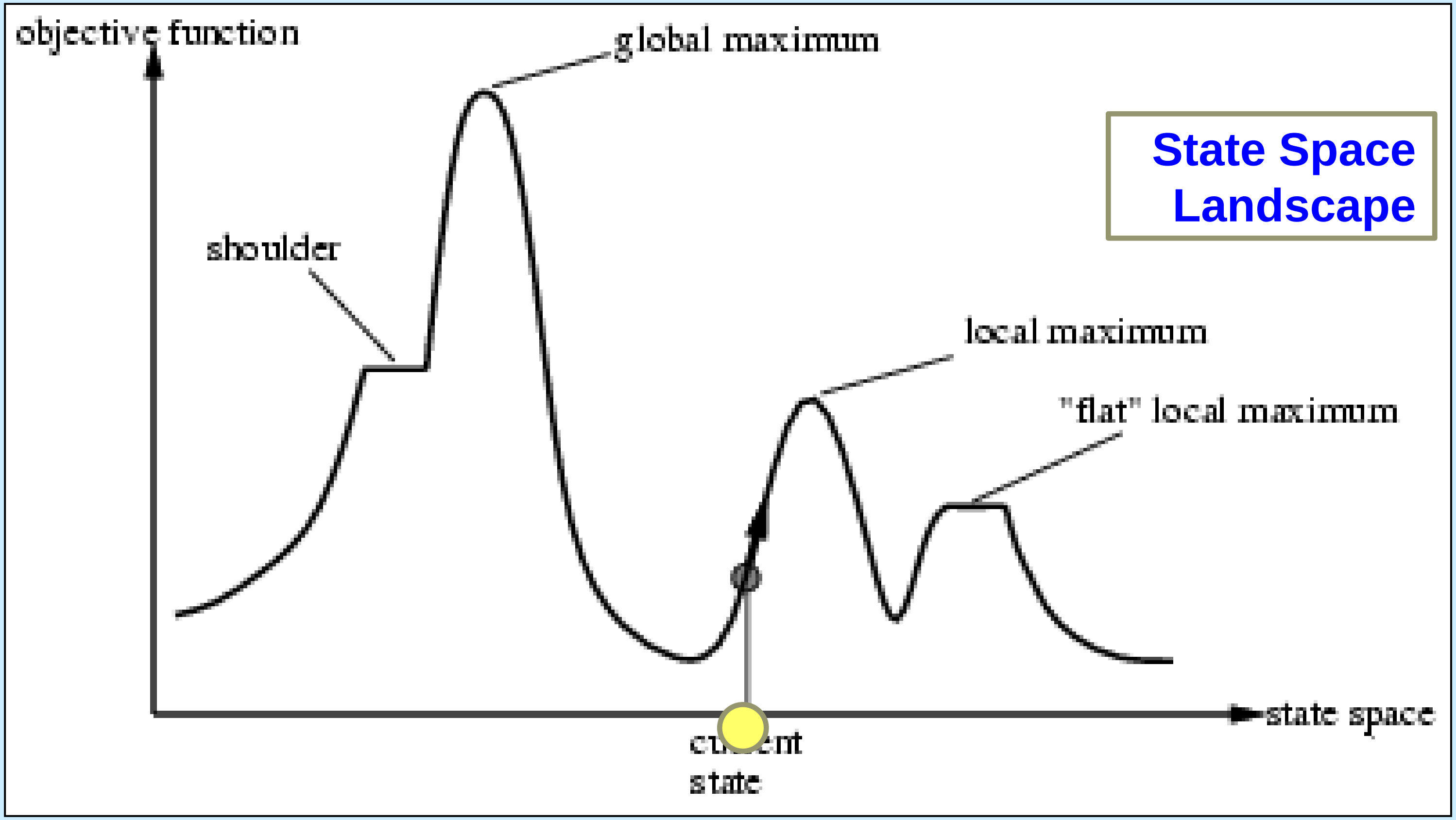
Ex: ***8-puzzle, n-queens, Chess etc.***

we have some "local search algorithms" where the path cost does not matters, and only focus on solution-state needed to reach the goal node. A local search algorithm completes its task by traversing on a single current node rather than multiple paths and following the neighbors of that node generally.

Optimization Problems:

Optimization problems implies to find the best state according to the objective function

- LSA are useful for solving **large search space problems fast by** finding the best state according to "Objective Function"
  - **Ex**: Darwinian Evolution, Fitness Function – Reproductive fitness



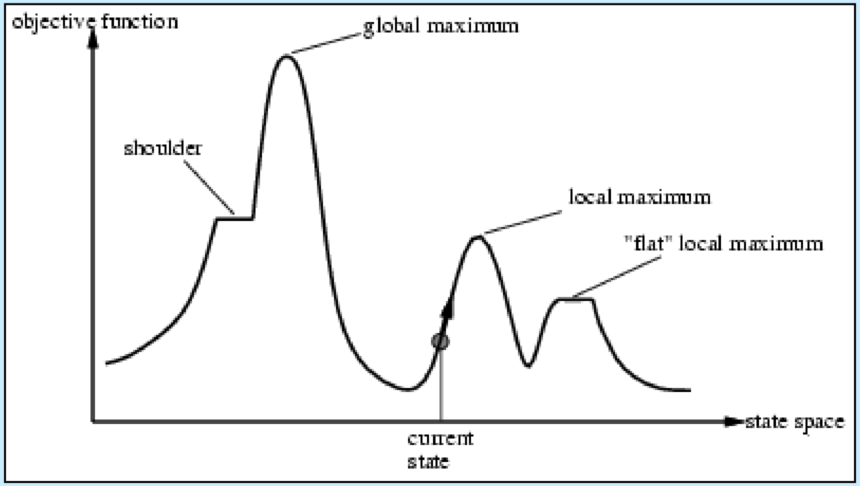
- **LSAs** explore the "State-space landscape" containing 2 things

Which has both **location** (state) & **elevation** (cost defined by objective function)

Aim – to find **lowest valley** (global minimum) or **highest peak** (global maximum)

Complete LSA – always finds a **goal**

Optimal LSA - always finds a **global maximum/minimum**



Local search algorithms

- In such cases: we can use **local search algorithms**
  - **LCA**'s keep track of the "**current**" state
  - Then try to **improve it** by comparing it with it's **neighbors** or **successors** ***(if current state is better, it is a peak)***
  - Comparison is done based on "**Objective Function**" *(elevation – similar to evaluation function)*
    - ***Ex***: *No. of conflicts for n-Queen, Manhattan distance for 8-Puzzle…*
- **Advantages** –

  1. Requires small memory – a constant amount *(to store states & values)*
  2. Can find a reasonable solution in large search space where systematic algorithms are unsuitable

Topics

(1) Hill-climbing search (Local Greedy Search)

- **HCS: Algorithm**
- **HCS: TSP, n-queens problem, 8-Puzzle Problems**

(2) Simulated Annealing

- **HCS: n-queens problem**
- **HCS: Algorithm**

Hill-climbing search (steepest ascent)

- **HCS** algorithm is a **loop** that continuously moves in the direction of increasing value (uphill)
- **Terminates** when reaches a "**Peak**" where **no neighbor** has a higher value
  - *Like climbing Everest in thick fog*
- **Data structure:** *Stores **State** & **Value** of objective function (not search tree)*
- Also called "**Greedy Local Search**"
- Makes rapid progress towards a solution by improving on previous state
  - *Ex: n-Queen problem*

HCS: Algorithm

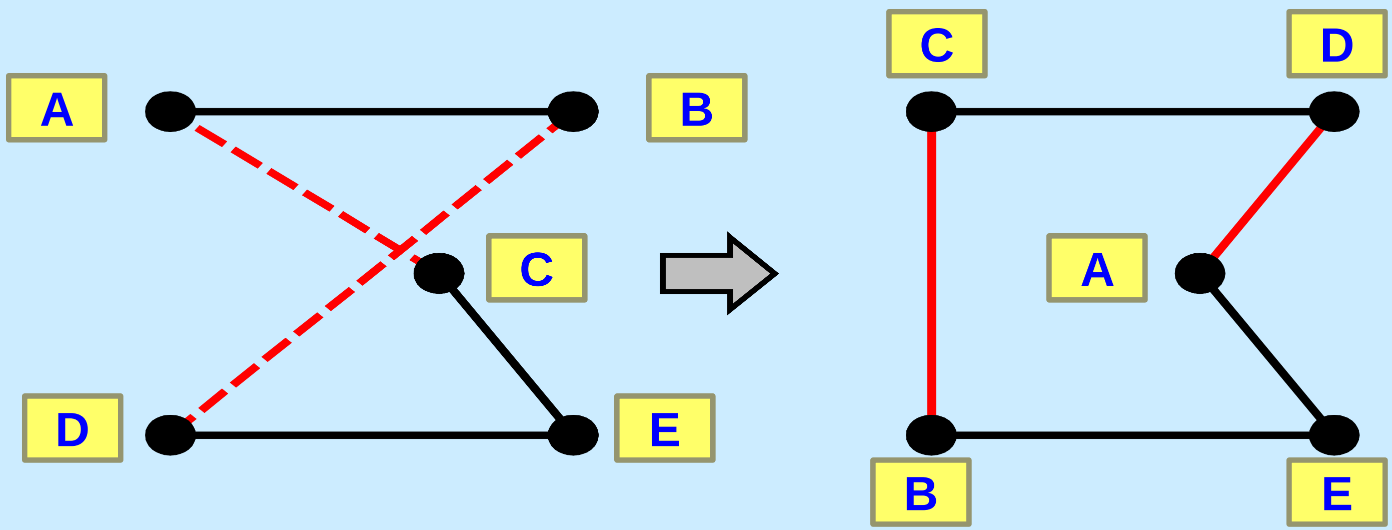
- At each step current node is replaced by it’s best neighbor (with highest/lowest VALUE)

```
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])   \\ Current node set to start node
   loop do   \\ Loop through each neighbor
      neighbor ← a highest-valued successor of current   \\ Select the nbr with best obj .function
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor        \\ If current node’s obj function is higher, return current node
                                     else set nbr node as the current node
```

Example: TSP

- Start with any complete tour, perform pair-wise exchange



- Variants of this approach get within 1% of optimal very quickly with thousands of cities

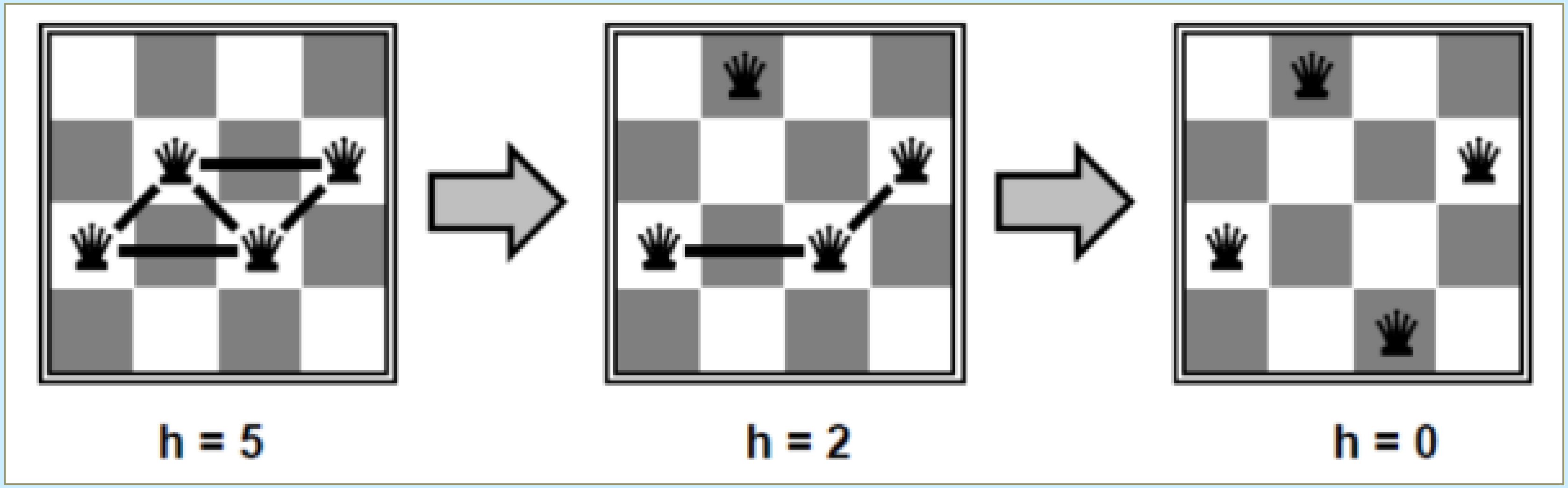
Example: n-queens

- Put **n queens** on an **n × n board** with no two queens attacking each other *(i.e not on the same row, column or diagonal)*
- **Objective function**: **Number of conflicts** *(solution is global minimum – preferably 0 conflicts(perfect solution))*

  or Manhattan distance (how many positions away)
- **Conflict** => the number of pairs of queens that are attacking each other

Example: n-queens

- Reached a solution in 2 steps

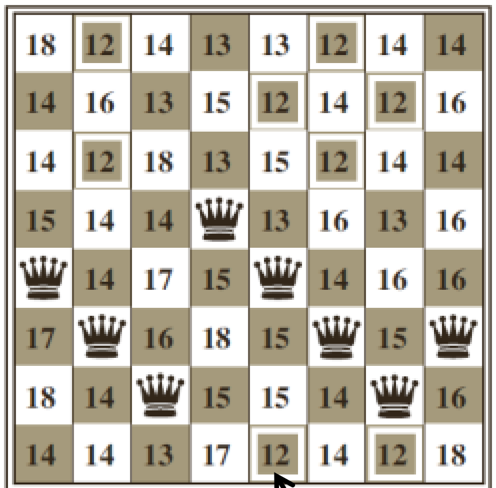


HCS: 8-queens problem

- Complete state formation
  - *Each state has 8 queens, one per column*
  - ***Successor states*** *– All possible states by moving a queen to another square in the same column (8X7=56 successors)*
- Heuristic cost function (h) – No. of queen pairs that attack each other
- Global minimum = 0 (perfect solutions)
- **Fig-1** in next page shows initial state with **h=17**
- Value of best successors **h=12**
- It takes only **5 steps** to reach the state in **Fig-2** with **h=1** (very nearly a solution)
- – HCS makes rapid progress towards solution

HCS: 8-queens problem

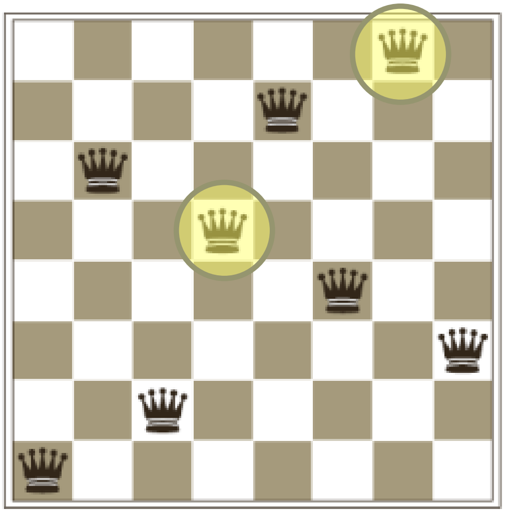
- Hill Climbing may NOT reach to a goal state for n-queens problem.



Min h

five steps to reach this state

local minimum with *h = 1*

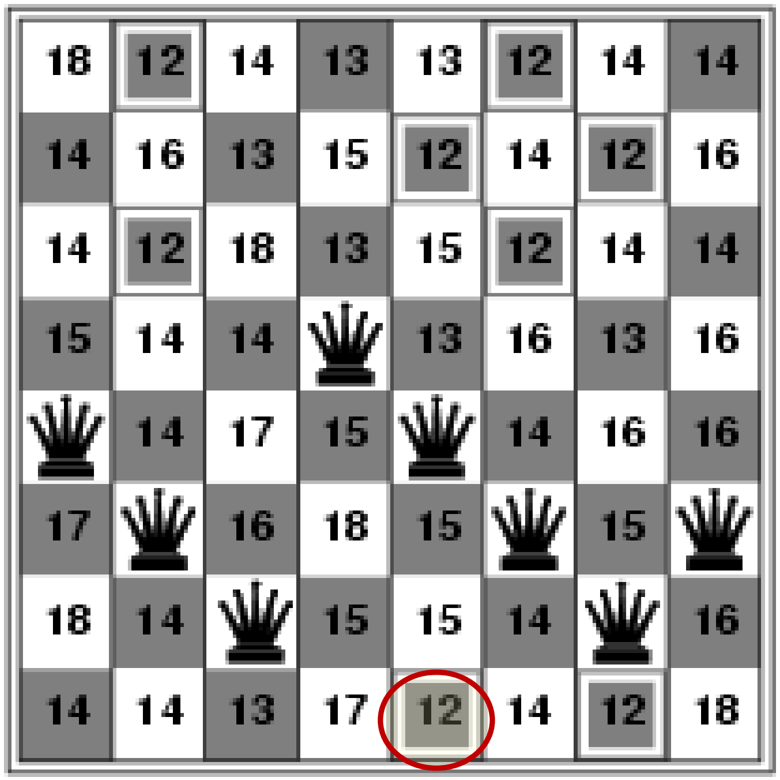


- A local minimum in the 8-queens state space; the state has h=1
- but every successor has a higher cost.
- Hill Climbing will stuck here

- ***h* =** no. of queen pairs attacking each other, either directly or indirectly
- ***h* = *17*** for the above state
- *Each square contains h values of the successors (h=12 is next best state)*

HCS: 8-queens problem

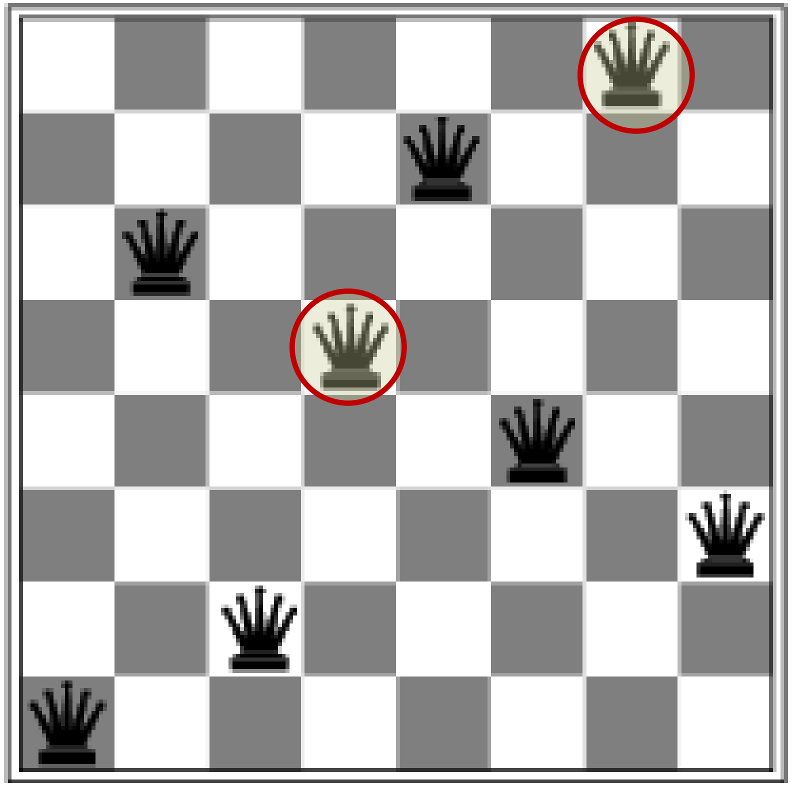
Fig-1



- ***h* =** no. of queen pairs attacking each other, either directly or indirectly
- ***h* = *17*** for the above state
- *Each square contains h values of the successors (h=12 is next best state)*

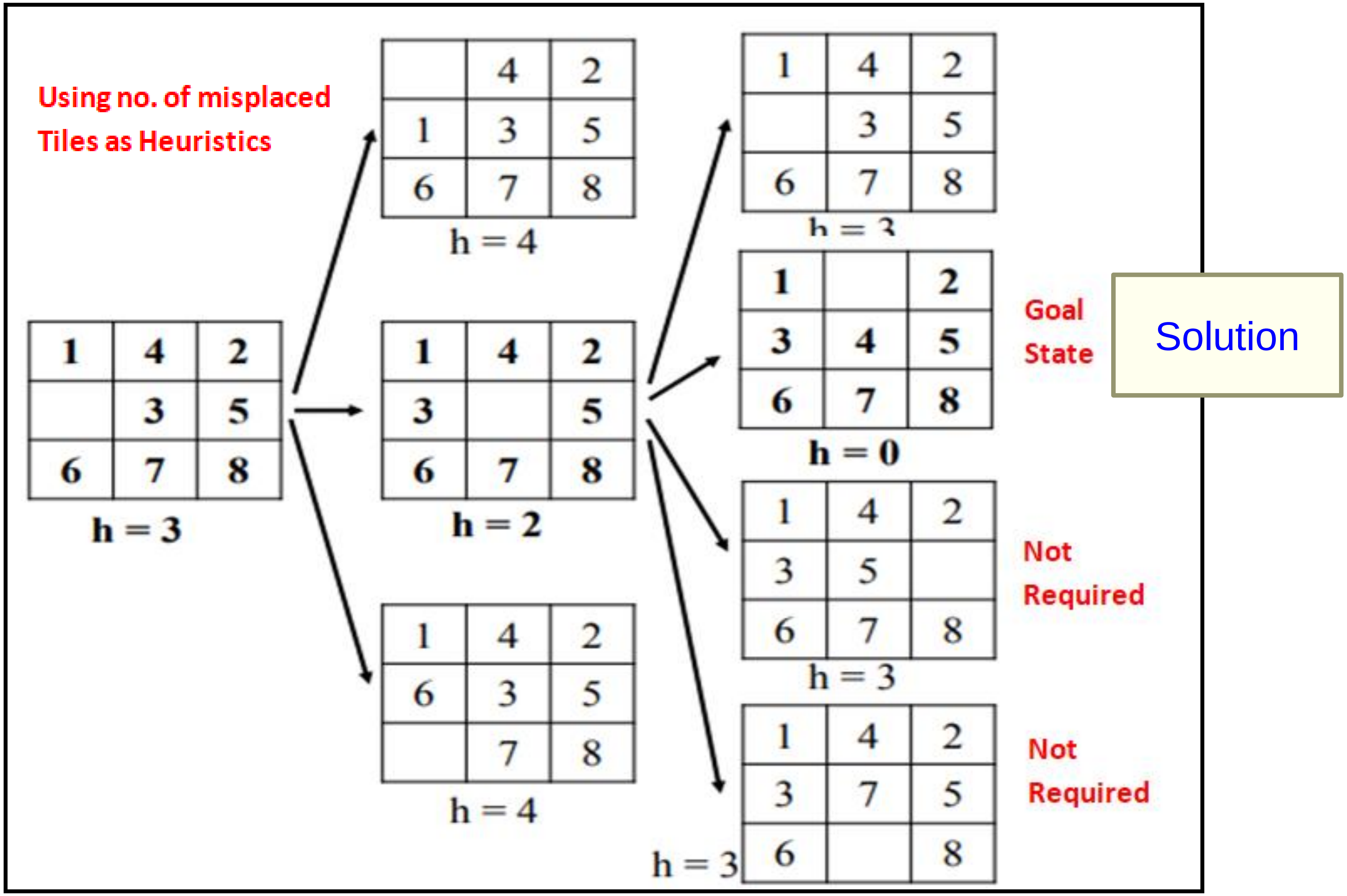
HCS: 8-queens problem

Fig-2



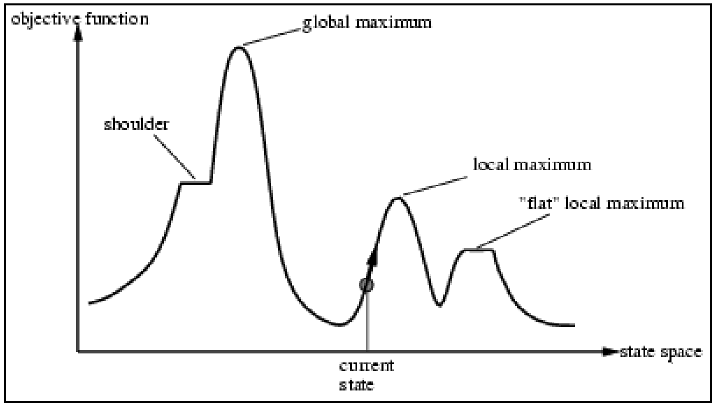
- A local minimum with *h = 1*
- Achieved in 5 steps

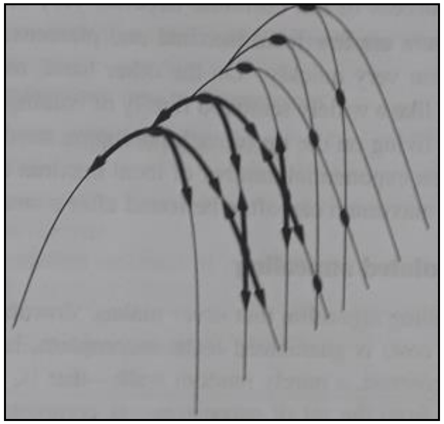
An 8-puzzle problem solved by a HCS



HCS-Problems

- Local maxima: is a peak that is higher than it's neighboring states but lower than global maximum
  - HCS can get stuck in local maxima





- Ridges – Sequence of local maxima that are difficult to navigate
- Plateaux – Flat local maximum from which no uphill exit exists or a shoulder.

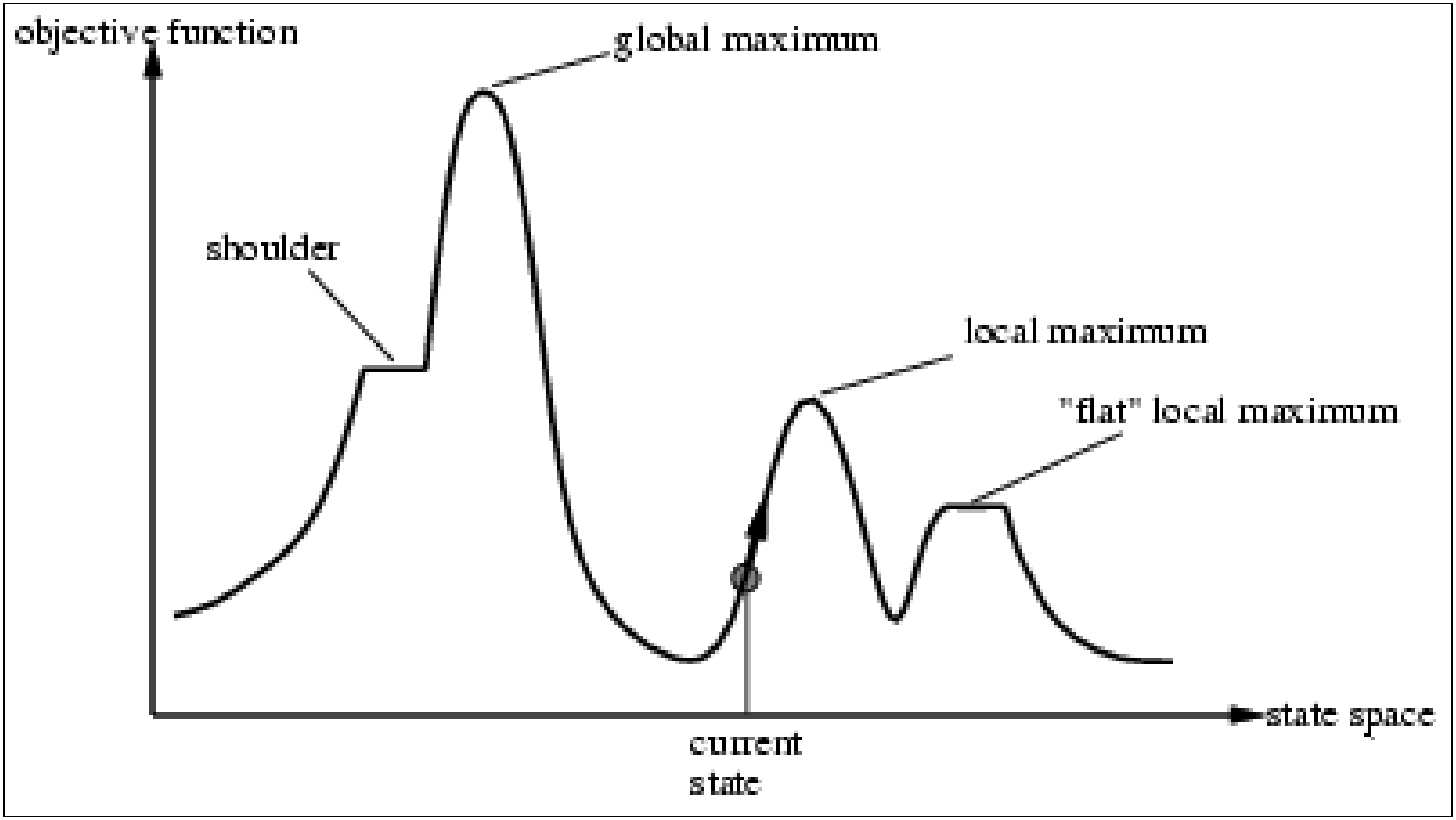
Simulated Annealing

HCS algorithm - can “get stuck in local maxima” as it never makes “downhill” moves *(hence remains incomplete)*

Random walk – “Randomly selects a successor” from the neighborhood

Is complete but inefficient

Simulated Annealing algorithm combines both **HCS** & **Random Walk**


objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

Simulated Annealing

- **Gradient descent:**

*Roll a **ping pong ball** on **bumpy surface** it will **rest at a local minimum**. Then **shake the ball hard** enough to dislodge it from **local minimum** to **global minimum***

- **Simulated Annealing –**
- Shake the ball hard (by increasing the temp in the beginning) & then gradually reduce the intensity of shaking (reduce the temp)

Simulated Annealing Algorithm

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
    T ← schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE(current) – VALUE(next)
    if ΔE > 0 then current ← next
    else current ← next only with probability e^(ΔE/T)
```

Thanks!!!