BITS Pilani
Pilani Campus

Contents:RBFS Analysis and SMA*
Local Search

Analysis of RBFS

Optimal if h(n) is admissible.

Space is O(bm)

Potentially exponential time in cost of solution

Keeps more information than IDA*, but may benefit from storing even more information

SMA* (Simplified Memory A*)

- MA* and SMA* are restricted memory best first search algorithms that utilize all the memory available.
- The algorithm executes best first search while memory is available.
- When the memory is full the worst node is dropped but the value of the forgotten node is backed up at the parent.

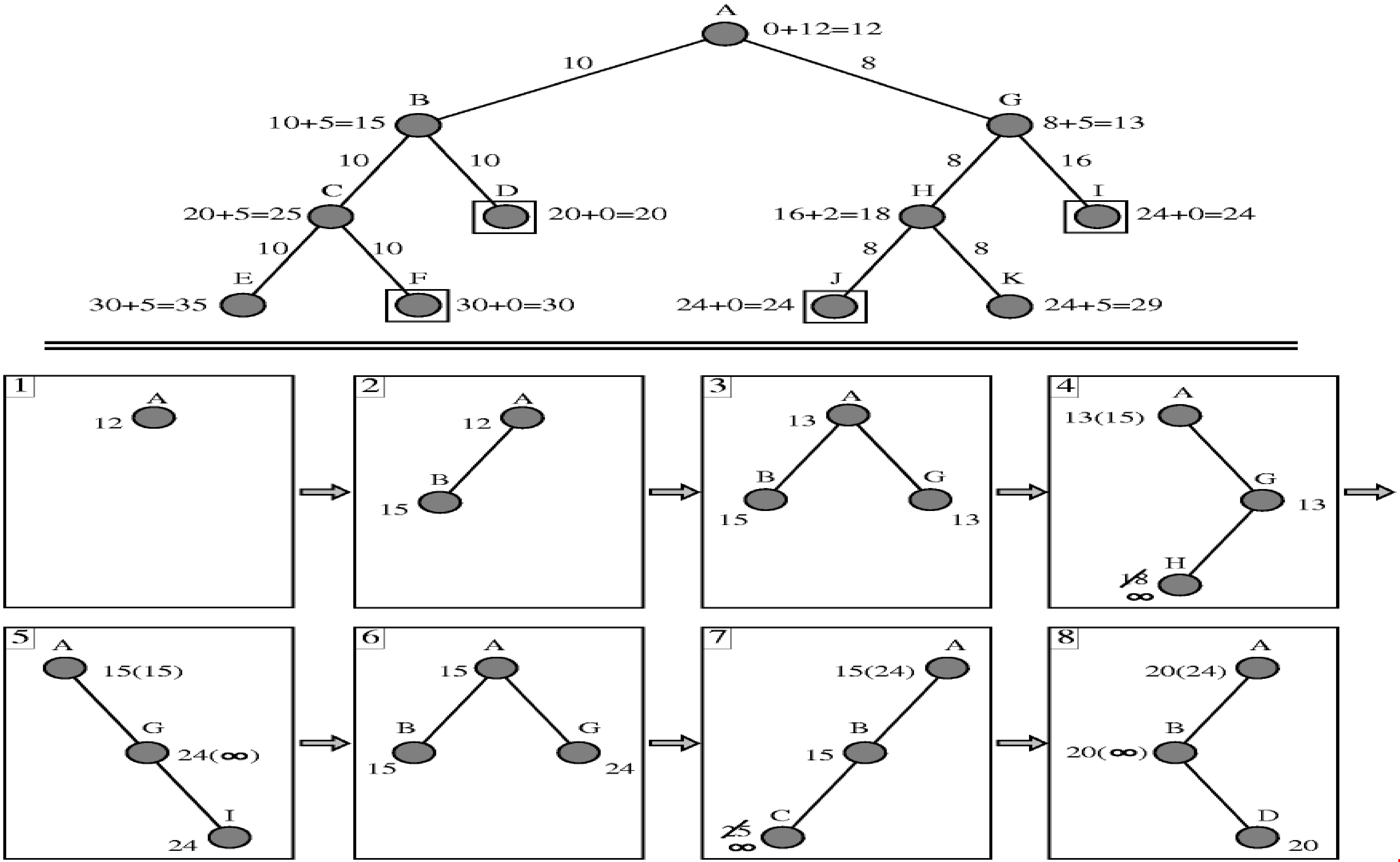
SMA*

- Expand deepest lowest f-cost leaf-node
- Best first search on f-cost
- Update f-cost of nodes whose successors have higher f-cost than it as:

F(node)=min(f(child1), f(child2),…..f(childn))

- Drop shallowest & highest f-cost leaf node
- Remember best forgotten descendant
- Node at the maximum depth get ∞ cost if its not a goal node.

SMA*



A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29
1
A
12
2
A
12
B
15
3
A
13
B
15
G
13
4
A
13(15)
G
13
H
18
∞
5
A
15(15)
G
24(∞)
I
24
6
A
15
B
15
G
24
7
A
15(24)
B
15
C
25
∞
8
A
20(24)
B
20(∞)
D
20

Analysis of SMA*

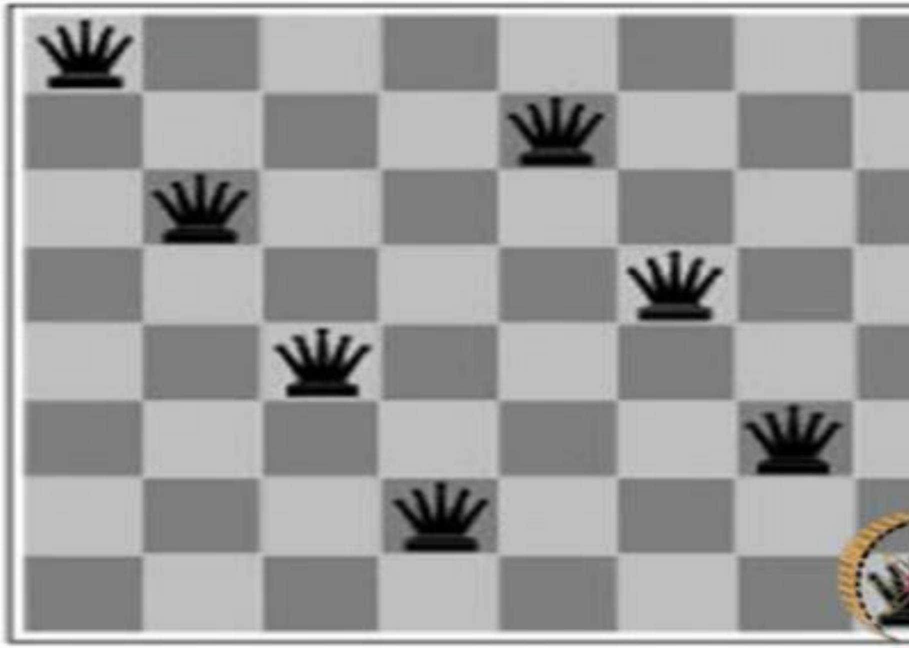
Optimal if solution fits in memory.

Constant memory requirements

Path vs. State Optimization

Previous lecture: path to goal is solution to problem
- systematic exploration of search space.

This lecture: a state is solution to problem
- for some problems path is irrelevant.
- E.g., 8-queens



Different algorithms can be used
- Depth First Branch and Bound
- Local search

Local search and optimization

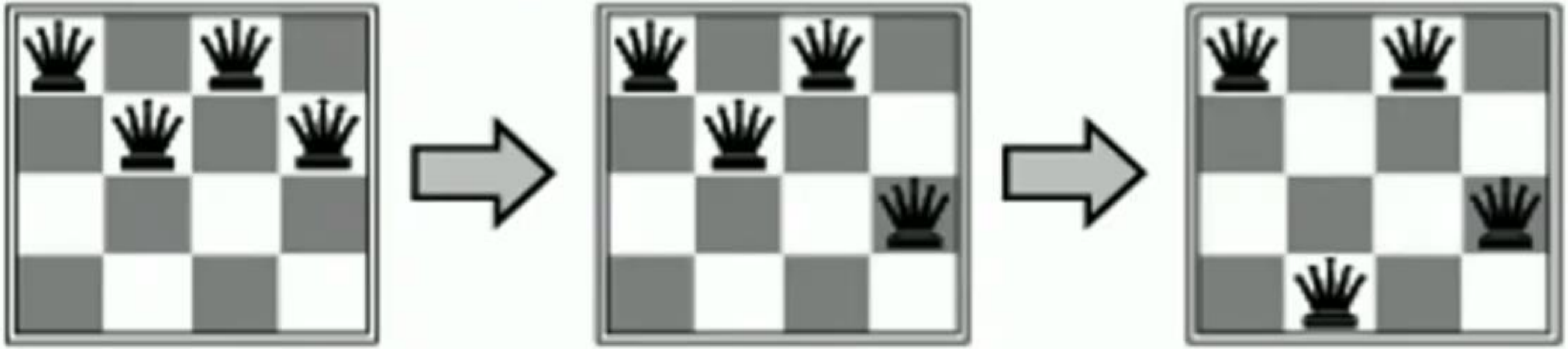
- Local search
  - Keep track of single current state
  - Move only to neighboring states
  - Ignore paths

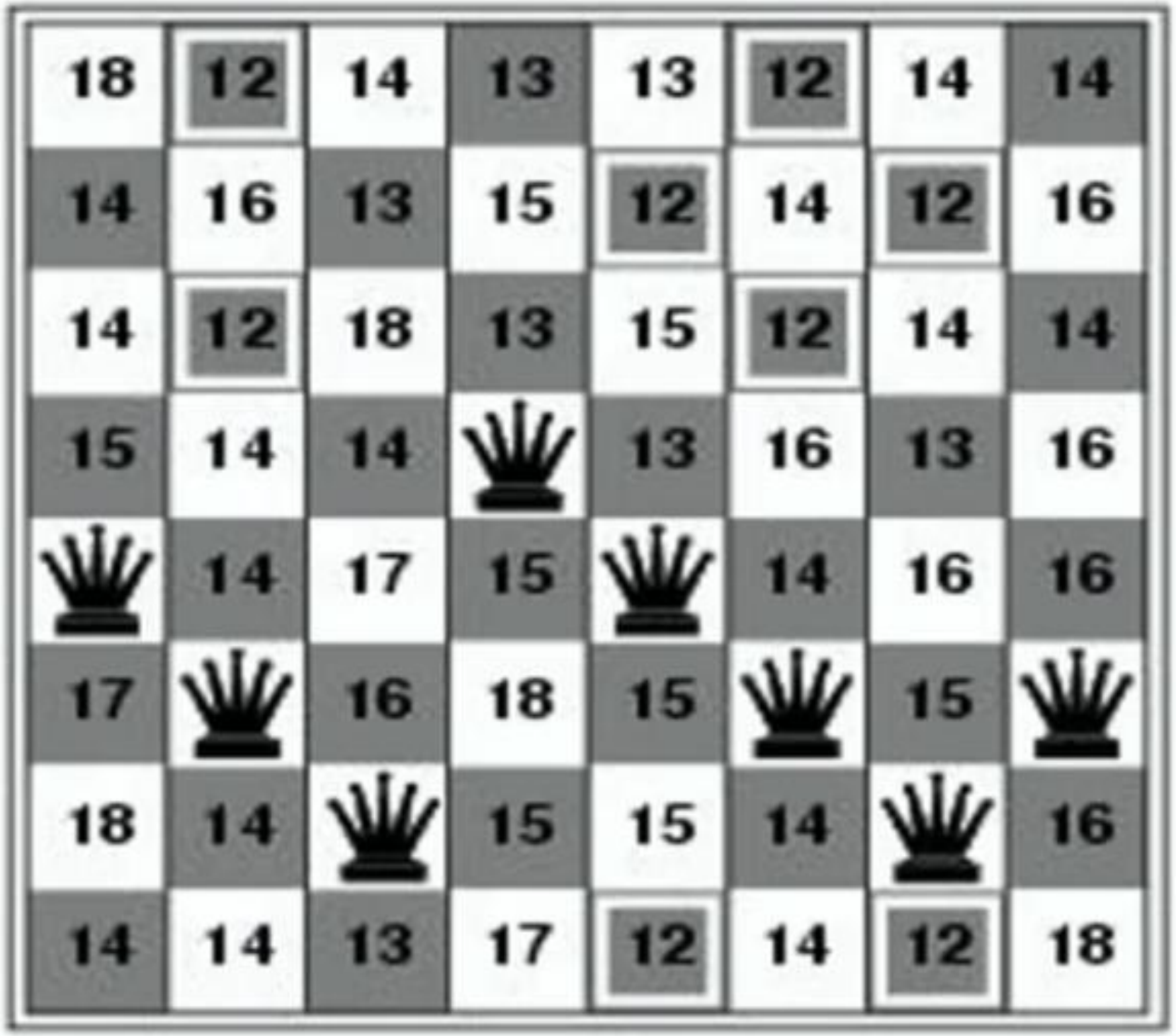
- Advantages:
  - Use very little memory
  - Can often find reasonable solutions in large or infinite (continuous) state spaces.

- "Pure optimization" problems
  - All states have an objective function
  - Goal is to find state with max (or min) objective value
  - Does not quite fit into path-cost/goal-state formulation
  - Local search can do quite well on these problems.

Example: *n*-queens

- Put *n* queens on an *n* x *n* board with no two queens on the same row, column, or diagonal





- Need to convert to an optimization problem
- *h* = number of pairs of queens that are attacking each othe
- *h = 17* for the above state

Search Space

- State
  - All 8 queens on the board in some configuration
- Successor function
  - move a single queen to another square in the same column.
- Example of a heuristic function *h(n)*:
  - the number of pairs of queens that are attacking each other
  - (so we want to minimize this)

Hill-climbing (Greedy Local Search) max version

```
function HILL-CLIMBING( problem) return a state that is a local maximum
    input: 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
```

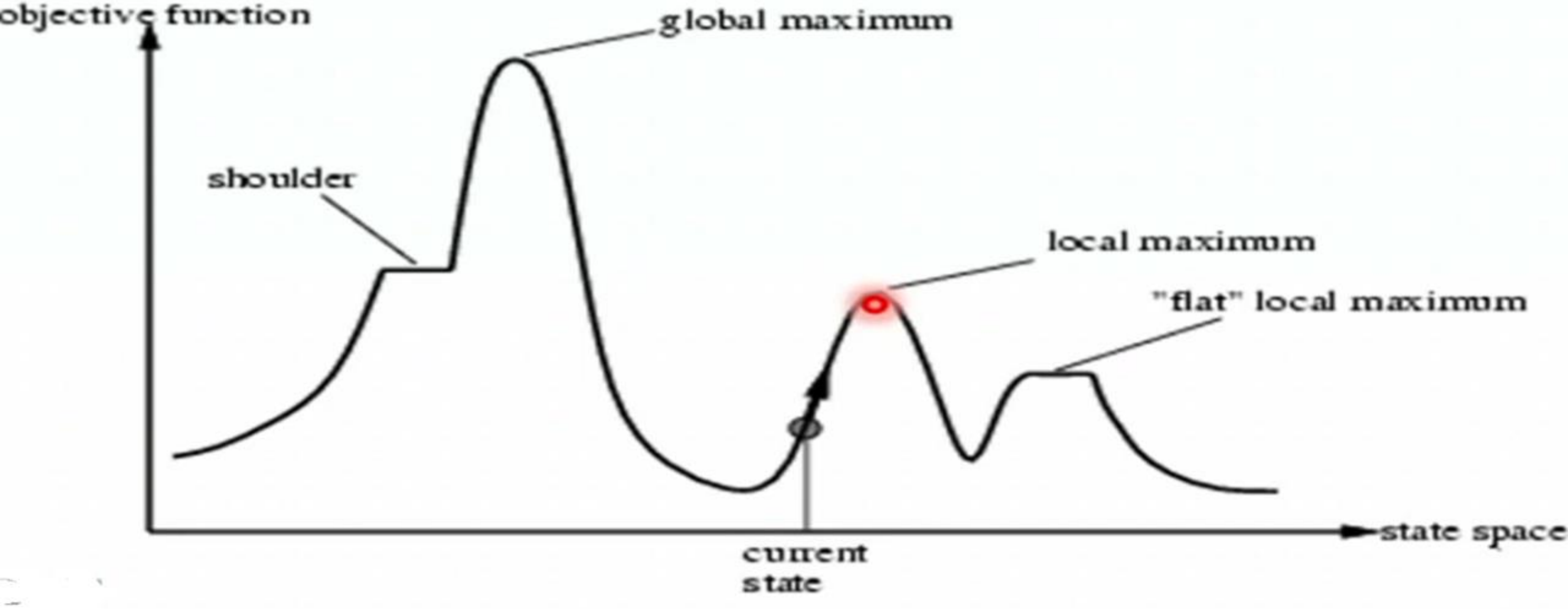
Hill-climbing search

- "a loop that continuously moves towards increasing value"
  - terminates when a peak is reached
  - Aka greedy local search
- Value can be either
  - Objective function value
  - Heuristic function value (minimized)

- Hill climbing does not look ahead of the immediate neighbors
- Can randomly choose among the set of best successors
  - if multiple have the best value

HILL CLIMBING GETS STUCK IN LOCAL MAXIMUM

“Landscape” of search



Hill-climbing on 8-queens

- Randomly generated 8-queens starting states…
- 14% the time it solves the problem
- 86% of the time it get stuck at a local minimum

- However…
  - Takes only 4 steps on average when it succeeds
  - And 3 on average when it gets stuck
  - (for a state space with 8^8 =~17 million states)

Hill Climbing Drawbacks

- Local maxima

- Plateaus

Escaping Shoulders: Sideways Move

- If no downhill (uphill) moves, allow sideways moves in hope that algorithm can escape
  - Need to place a limit on the possible number of sideways moves to avoid infinite loops
- For 8-queens
  - Now allow sideways moves with a limit of 100
  - Raises percentage of problem instances solved from 14 to 94%
  - However....
    - 21 steps for every successful solution
    - 64 for each failure

Trivial Algorithms

- Random Sampling
  - Generate a state randomly
- Random Walk
  - Randomly pick a neighbor of the current state
- Both algorithms asymptotically complete.

Hill-climbing: stochastic variations

- Stochastic hill-climbing
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.

- To avoid getting stuck in local minima
  - Random-walk hill-climbing
  - Random-restart hill-climbing
  - Hill-climbing with both

Hill Climbing with random walk

→When the state-space landscape has local minima, any search that moves only in the greedy direction cannot be complete

→Random walk, on the other hand, is asymptotically complete

Idea: Put random walk into greedy hill-climbin

- At each step do one of the two
  - Greedy: With prob p move to the neighbor with largest
  - Random: With prob 1-p move to a random neighbor

Hill-climbing with random restarts

- If at first you don't succeed, try, try again!
- Different variations
  - For each restart: run until termination vs. run for a fixed time
  - Run a fixed number of restarts or run indefinitely
- Analysis
  - Say each search has probability p of success
    - E.g., for 8-queens, p = 0.14 with no sideways moves
  - Expected number of restarts?
  - Expected number of steps taken?

Simulated Annealing

Simulated Annealing = physics inspired twist on random wa

Basic ideas:

- like hill-climbing identify the quality of the local improvements
- instead of picking the best move, pick one randomly
- say the change in objective function is δ
- if δ is positive, then move to that state
- otherwise:
  - move to this state with probability proportional to δ
  - thus: worse moves (very large negative δ) are executed less often
- however, there is always a chance of escaping from local maxima
- over time, make it less likely to accept locally bad moves

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 prob. of downward steps

current ← MAKE-NODE(INITIAL-STATE[*problem*])
for t ← 1 to ∞ do
T ← *schedule*[*t*]
if *T = 0* **then return** *current*
next ← a randomly selected successor of *current*
ΔE ← VALUE[*next*] - VALUE[*current*]
if $\Delta E > 0$ **then** *current* ← *next*
else *current* ← *next* only with probability $e^{\Delta E/T}$

Physical Interpretation of Simulated Annealing

- A Physical Analogy:
  - imagine letting a ball roll downhill on the function surface
    – this is like hill-climbing (for minimization)
  - now imagine shaking the surface, while the ball rolls, gradually reducing the amount of shaking
    – this is like simulated annealing

- Annealing = physical process of cooling a liquid or metal until particles achieve a certain frozen crystal state
  - simulated annealing:
    – free variables are like particles
    – seek "low energy" (high quality) configuration
    – slowly reducing temp. T with particles moving around randomly

Simulated Annealing

Simulated Annealing = physics inspired twist on random wa

Basic ideas:

- like hill-climbing identify the quality of the local improvements
- instead of picking the best move, pick one randomly
- say the change in objective function is δ
- if δ is positive, then move to that state
- otherwise:
  - move to this state with probability proportional to δ
  - thus: worse moves (very large negative δ) are executed less often
- however, there is always a chance of escaping from local maxima
- over time, make it less likely to accept locally bad moves

Simulated Annealing in Practice

- method proposed in 1983 by IBM researchers for solving VLSI layout problems (Kirkpatrick et al, *Science*, 220:671-680, 1983).
  - theoretically will always find the global optimum

- Other applications: Traveling salesman, Graph partitioning, Graph coloring, Scheduling, Facility Layout, Image Processing, ...

- useful for some problems, but can be very slow
  - slowness comes about because T must be decreased very gradually to retain optimality