ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Artificial Intelligence

Lecturer 5 - Advanced search methods

School of Information and Communication Technology - HUST

Outline

- Memory-bounded heuristic search
- Hill-climbing search
- Simulated annealing search

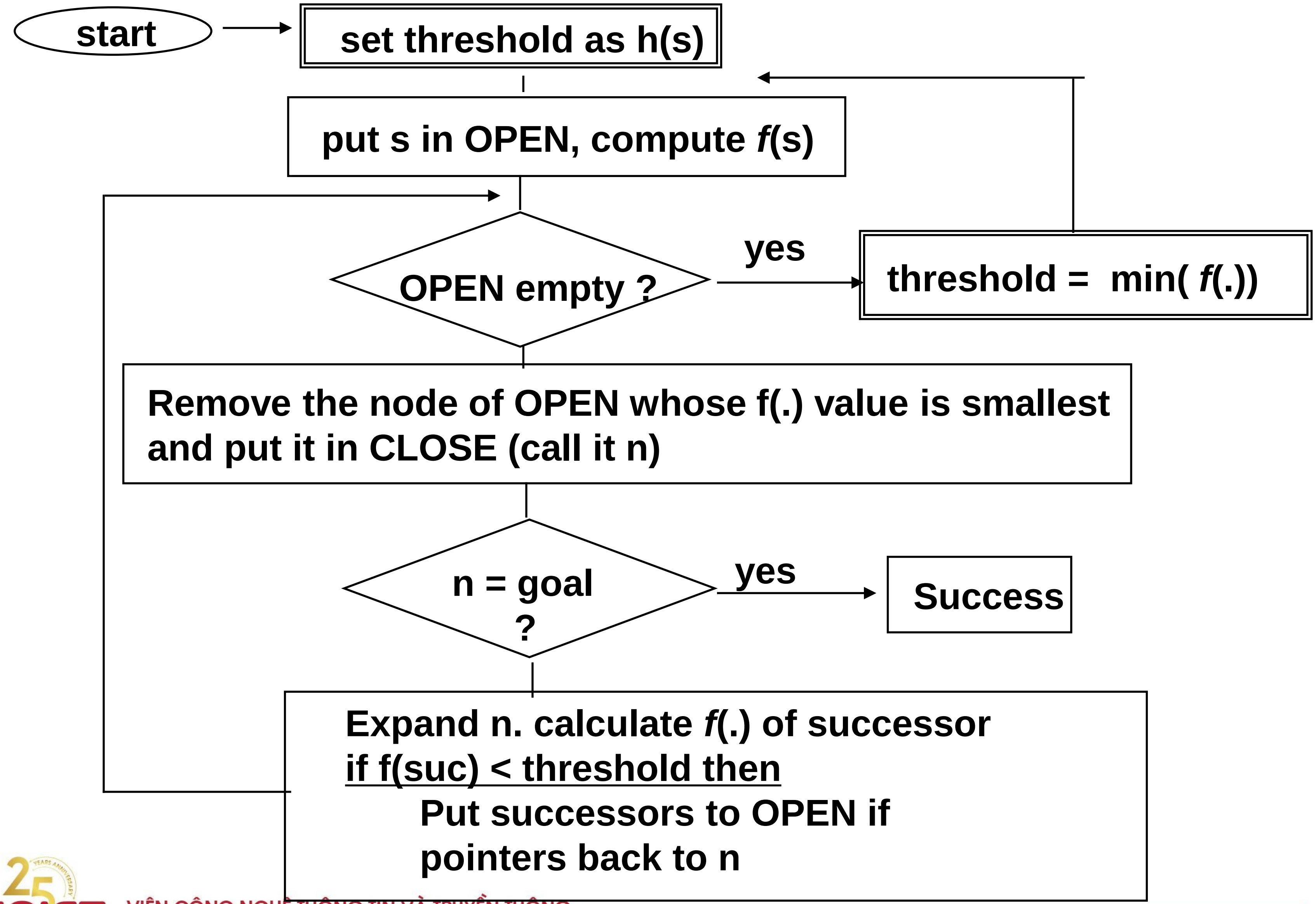
Memory-bounded heuristic search

- Some solutions to A* space problems (maintain completeness and optimality)
  - Iterative-deepening A* (IDA*)
    - Here cutoff information is the *f*-cost ($g+h$) instead of depth
  - Recursive best-first search(RBFS)
    - Recursive algorithm that attempts to mimic standard best-first search with linear space.
  - (simple) Memory-bounded A* ((S)MA*)
    - Drop the worst-leaf node when memory is full

Iterative Deeping A*

- Iterative Deeping version of A*
  - use threshold as depth bound
    - To find solution under the threshold of *f*(.)
  - increase threshold as minimum of *f*(.) of
    - previous cycle
- Still admissible
- same order of node expansion
- Storage Efficient – practical
  - but suffers for the real-valued *f*(.)
  - large number of iterations

Iterative Deepening A* Search Algorithm (for tree search)



Recursive best-first search

- A variation of Depth-first search
- Keep track of *f*-value of the best alternative path
- Unwind if *f*-value of all children exceed its best alternative
- When unwind, store *f*-value of best child as its *f*-value
- When needed, the parent regenerate its children again.

Recursive best-first search

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
   return RBFS(problem,MAKE-NODE(INITIAL-STATE[problem]),∞)

function RBFS ( problem, node, f_limit) return a solution or failure and a new f-cost limit
   if GOAL-TEST[problem](STATE[node]) then return node
   successors ← EXPAND(node, problem)
   if successors is empty then return failure, ∞
   for each s in successors do
          f [s] ← max(g(s) + h(s), f [node])
   repeat
          best ← the lowest f-value node in successors
          if f [best] > f_limit then return failure, f [best]
          alternative ←  the second lowest f-value among successors
          result, f [best] ←  RBFS(problem, best, min(f_limit, alternative))
          if result ≠ failure then return result
```

Recursive best-first search

- Keeps track of the f-value of the best-alternative path available.
  - If current f-values exceeds this alternative f-value then backtrack to alternative path.
  - Upon backtracking change f-value to best f-value of its children.
  - Re-expansion of this result is thus still possible.

RBFS evaluation

- RBFS is a bit more efficient than IDA*
  - Still excessive node generation (mind changes)
- Like A*, optimal if *h(n)* is admissible
- Space complexity is *O(bd)*.
  - IDA* retains only one single number (the current f-cost limit)
- Time complexity difficult to characterize
  - Depends on accuracy if h(n) and how often best path changes.
- IDA* and RBFS suffer from ***too little*** memory.

(simplified) memory-bounded A*

- Use all available memory.
  - I.e. expand best leafs until available memory is full
  - When full, SMA* drops worst leaf node (highest *f*-value)
  - Like RBFS, we remember the best descendant in the branch we delete

- What if all leafs have the same *f*-value?
  - Same node could be selected for expansion and deletion.
  - SMA* solves this by expanding *newest* best leaf and deleting *oldest* worst leaf.

- The deleted node is regenerated when all other candidates look worse than the node.

- SMA* is complete if solution is reachable, optimal if optimal solution is reachable.

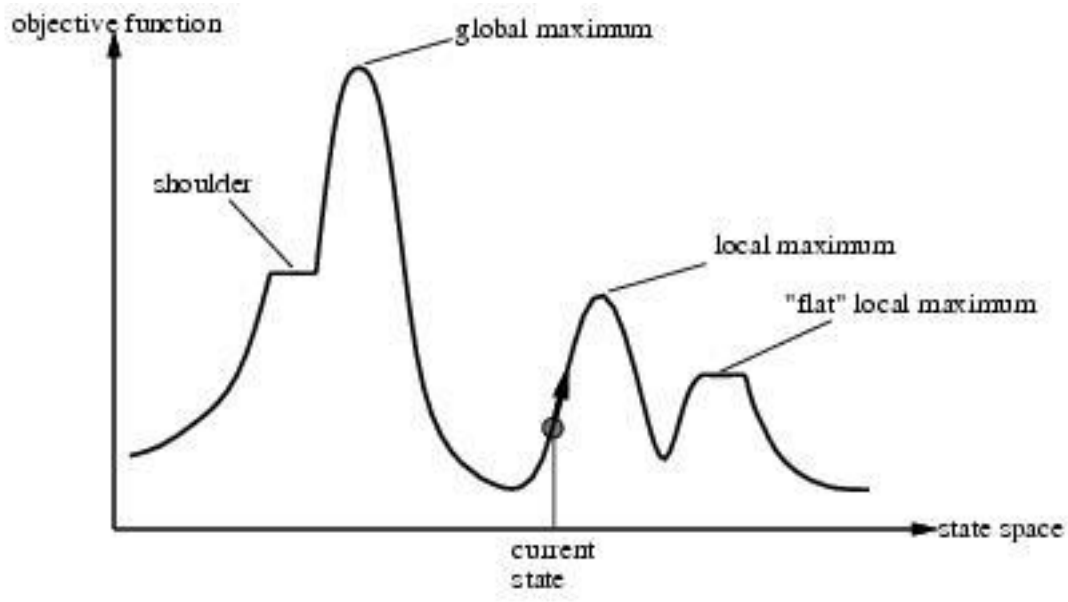
- Time can still be exponential.

Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms

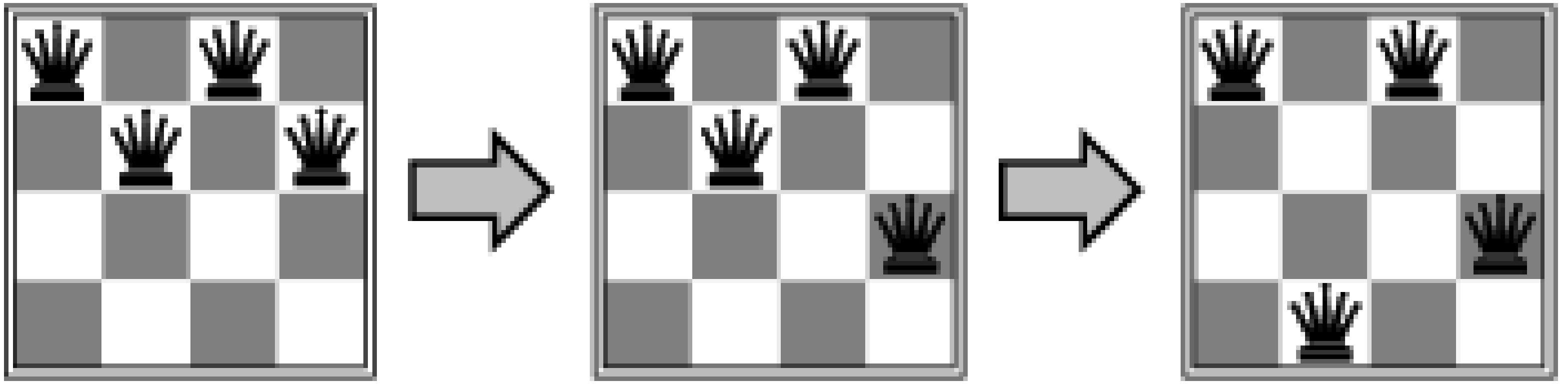
- Local search= use single current state and move to neighboring states.
- Advantages:
  - Use very little memory
  - Find often reasonable solutions in large or infinite state spaces.
- Are also useful for pure optimization problems.
  - Find best state according to some *objective function*.

Local search and optimization



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

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Hill climbing does not look ahead of the immediate neighbors of the current state.
- Hill-climbing chooses randomly among the set of best successors, if there is more than one.
- Some problem spaces are great for hill climbing and others are terrible.

Hill-climbing search

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*])

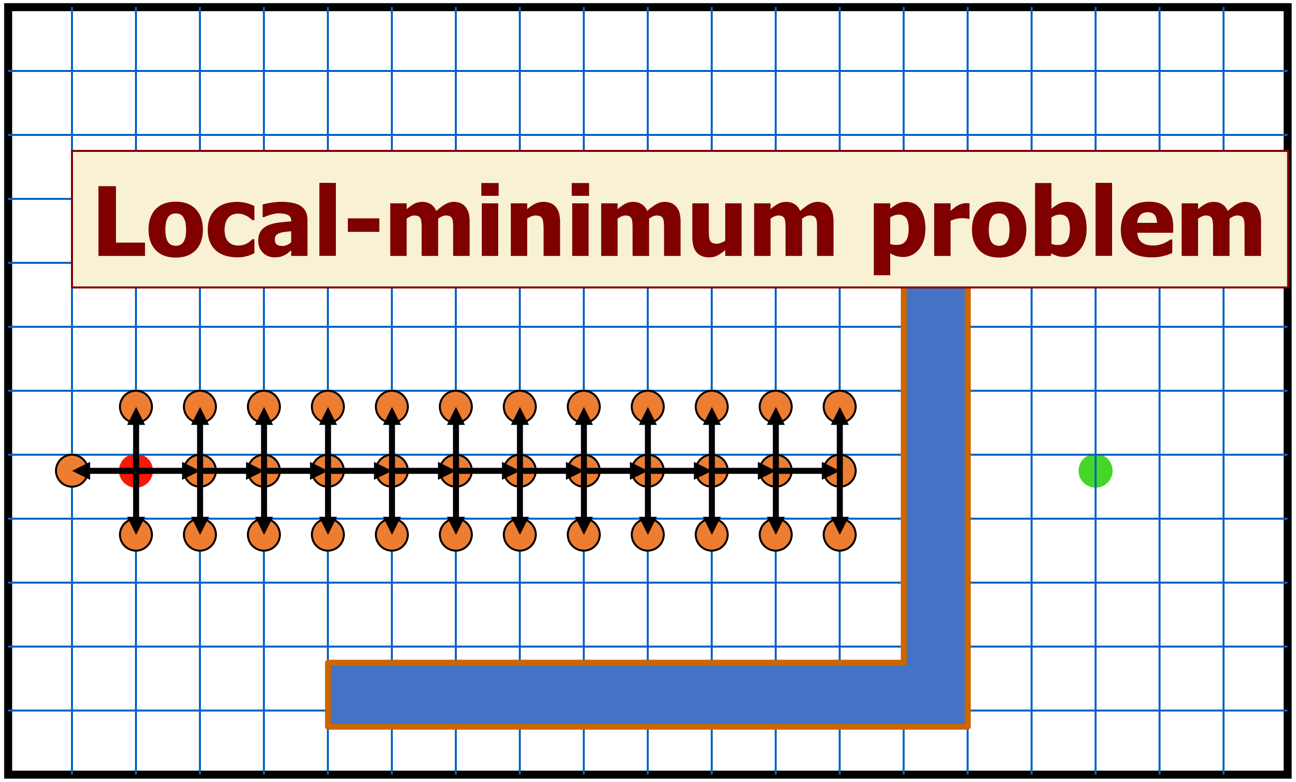
loop do

neighbor ← a highest valued successor of *current*

if VALUE [*neighbor*] < VALUE[*current*] **then return** STATE[*current*]

current ← *neighbor*

Robot Navigation

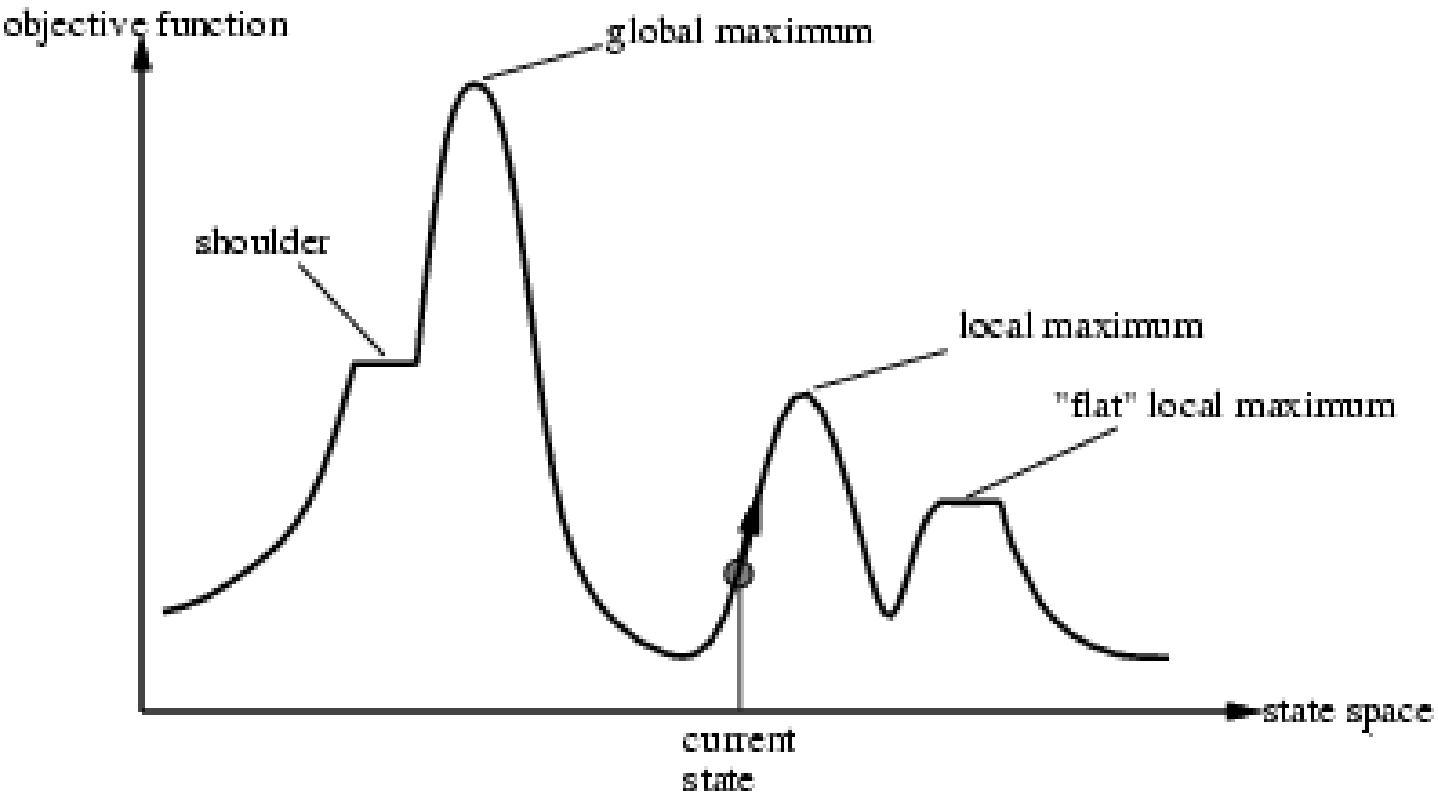


f(N) = h(N) = straight distance to the goal

Drawbacks of hill climbing

- Problems:
  - **Local Maxima**: depending on initial state, can get stuck in local maxima
  - **Plateaus:** the space has a broad flat region that gives the search algorithm no direction (random walk)
  - **Ridges:** flat like a plateau, but with dropoffs to the sides; steps to the North, East, South and West may go down, but a combination of two steps (e.g. N, W) may go up

➢ Introduce randomness



VIỆN CÔNG

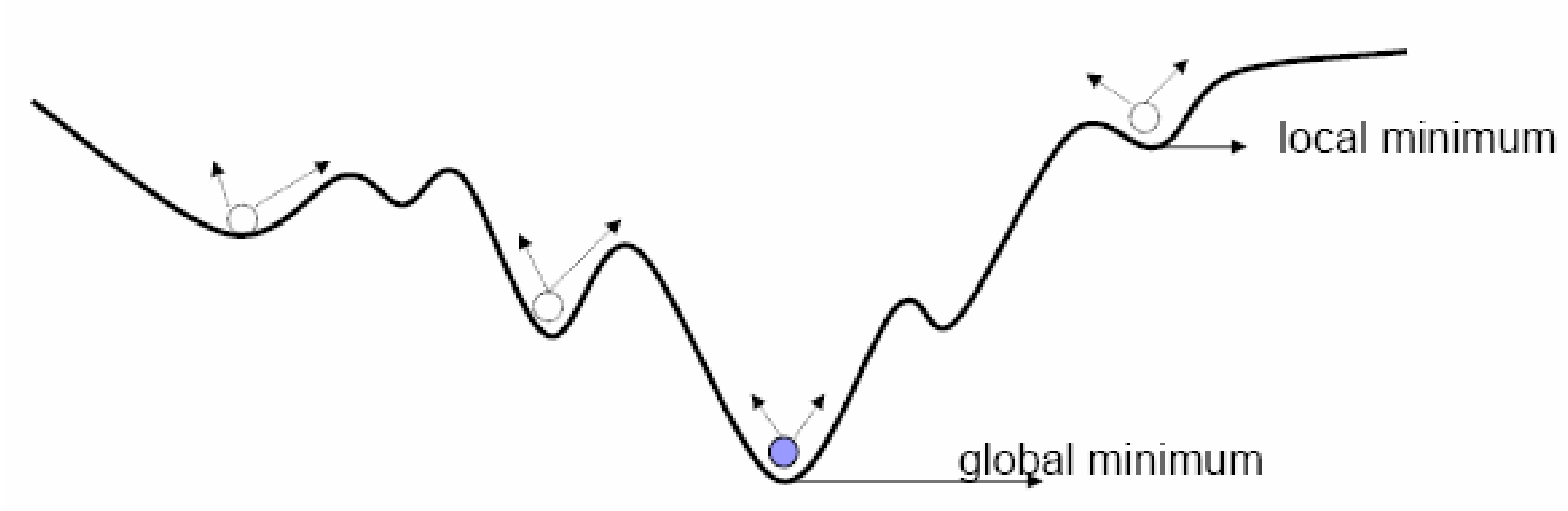
Hill-climbing variations

- Stochastic hill-climbing
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.
- First-choice hill-climbing
  - Stochastic hill climbing by generating successors randomly until a better one is found.
- Random-restart hill-climbing
  - Tries to avoid getting stuck in local maxima.
  - If at first you don’t succeed, try, try again…

Simulated Annealing

- Simulates slow cooling of annealing process
- Applied for combinatorial optimization problem by S. Kirkpatric ('83)
- **What is annealing?**
  - Process of slowly cooling down a compound or a substance
  - Slow cooling let the substance flow around → thermodynamic equilibrium
  - Molecules get optimum conformation

Simulated annealing



gradually decrease shaking to make sure the ball escape from local minima and fall into the global minimum

Simulated annealing

- Escape local maxima by allowing “bad” moves.
  - Idea: but gradually decrease their size and frequency.
- Origin; metallurgical annealing
- Implement:
  - Randomly select a move instead of selecting best move
  - Accept a bad move with probability less than 1 (p<1)
  - p decreases by time

- If T decreases slowly enough, best state is reached.
- Applied for VLSI layout, airline scheduling, etc.

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 $e^{\Delta E / T}$

Similar to hill climbing, but a **random** move instead of best move

case of improvement, make the move

Otherwise, choose the move with probability that decreases exponentially with the "badness" of the move.

What's the probability when: T → inf?

What's the probability when: T → 0?

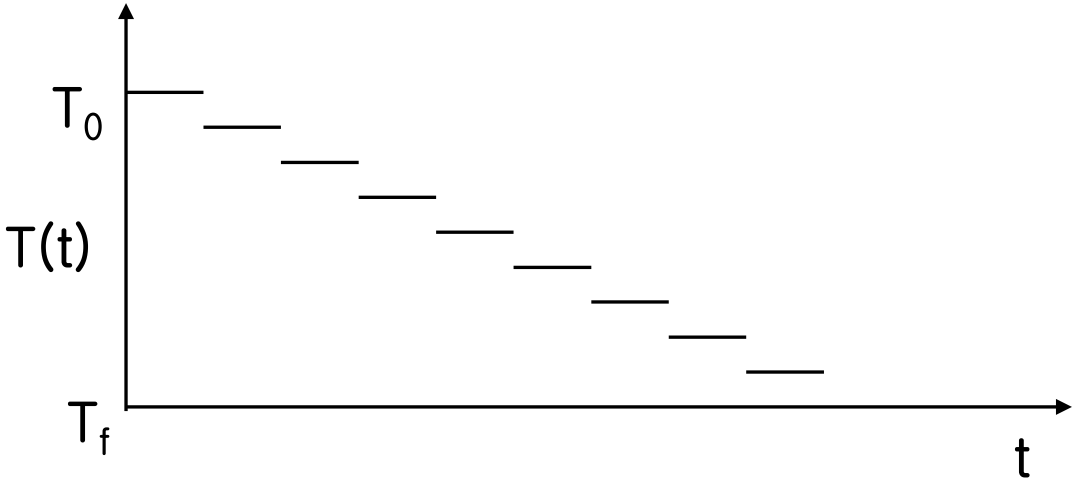
What's the probability when: Δ=0?

What's the probability when: Δ→-∞?

Simulated Annealing parameters

- Temperature T
  - Used to determine the probability
  - High T : large changes
  - Low T : small changes
- Cooling Schedule
  - Determines rate at which the temperature T is lowered
  - Lowers T slowly enough, the algorithm will find a global optimum
- In the beginning, aggressive for searching alternatives, become conservative when time goes by

Simulated Annealing Cooling Schedule



- if Ti is reduced too fast, poor quality
- if Tt >= T(0) / log(1+t) - Geman
  - System will converge to minimun configuration
- Tt = k/1+t - Szu
- Tt = a T(t-1) where a is in between 0.8 and 0.99

Tips for Simulated Annealing

- To avoid of entrainment in local minima
  - Annealing schedule : by trial and error
    - Choice of initial temperature
    - How many iterations are performed at each temperature
    - How much the temperature is decremented at each step as cooling proceeds
- Difficulties
  - Determination of parameters
  - If cooling is too slow →Too much time to get solution
  - If cooling is too rapid → Solution may not be the global optimum

Properties of simulated annealing

- Theoretical guarantee:
  - Stationary distribution: $p(x) \,\alpha\, e^{\frac{E(x)}{kT}}$
  - If T decreased slowly enough, will converge to optimal state!

- Is this an interesting guarantee?

- Sounds like magic, but :
  - The more downhill steps you need to escape, the less likely you are to every make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways