Course 16 :198 :440 : Introduction To Artificial Intelligence
Lecture 4

Solving Problems by Searching: Beyond Classical Search

Kowndinya Boyalakuntla

Tuesday, September 15, 2026

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

These slides are adapted from Prof. Abdeslam Boularias' CS440 course.

Outline

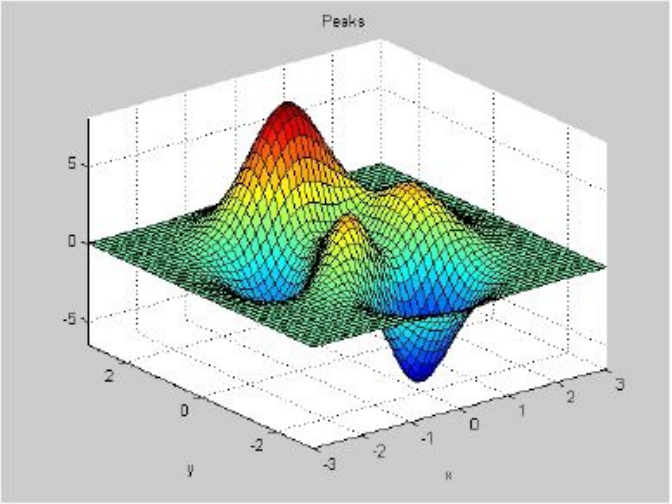
We relax the simplifying assumptions of the previous lectures, and we get closer to real-world applications

1. Local search
2. Continuous state spaces
3. Stochastic actions
4. Partial observations

Local search

- So far, we have focused on searching for the best path (sequence of state-actions) that leads to a goal.
- In many problems, such as the 8-queens problem, we do not care about the path to the goal. We only care about finding the goal.
- If paths are not relevant, one can start with an initial state and move only to **neighbors** of that state.

Local search is suitable for **optimization problems**, where we aim to find the best state according to a given **objective function**.



Local search

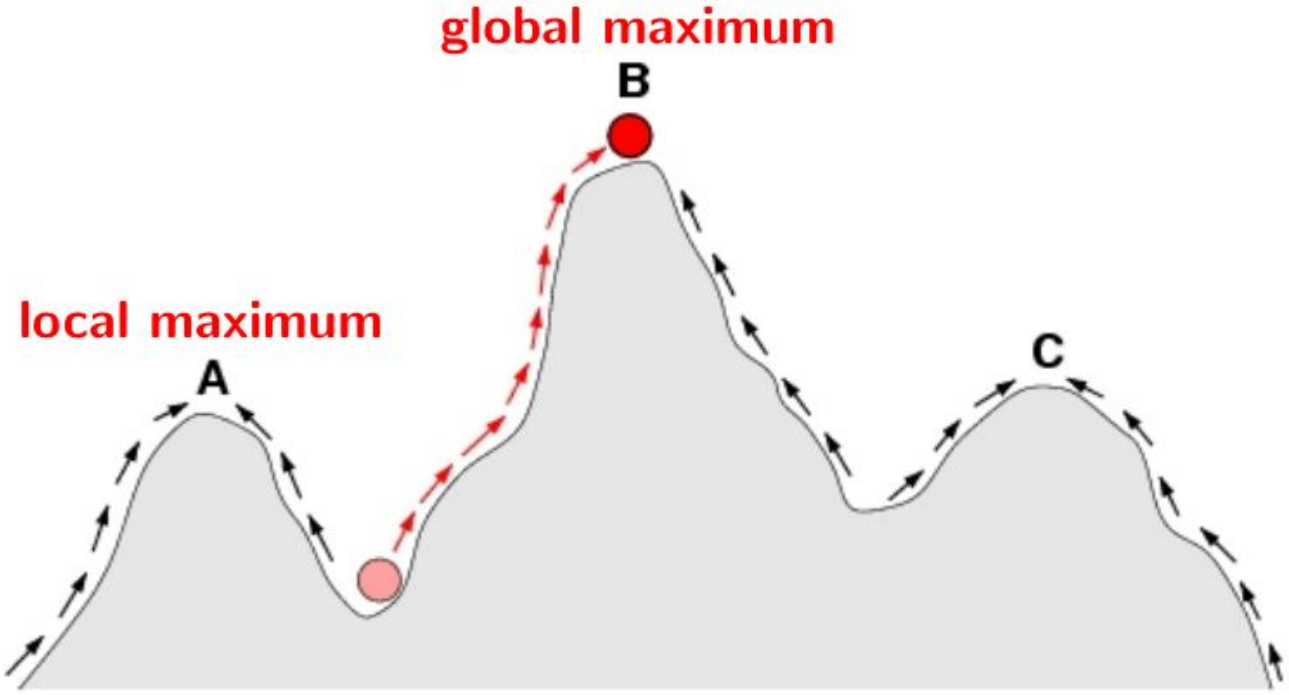
- So far, we have focused on searching for the best path (sequence of state-actions) that leads to a goal.
- In many problems, such as the 8-queens problem, we do not care about the path to the goal. We only care about finding the goal.
- If paths are not relevant, one can start with an initial state and move only to **neighbors** of that state.

Local search is suitable for **optimization problems**, where we aim to find the best state according to a given **objective function**.

Local search algorithms have two key advantages :

1. They use little memory as they usually do not need to remember paths.
2. They can often find reasonable solutions in large (or infinite) state spaces where systematic search is unsuitable.

Local search



$$\text{Optimum} = \begin{cases} \text{Maximum} & \text{if the function is an objective function (reward),} \\ \text{Minimum} & \text{if the function is a cost function (penalty).} \end{cases}$$

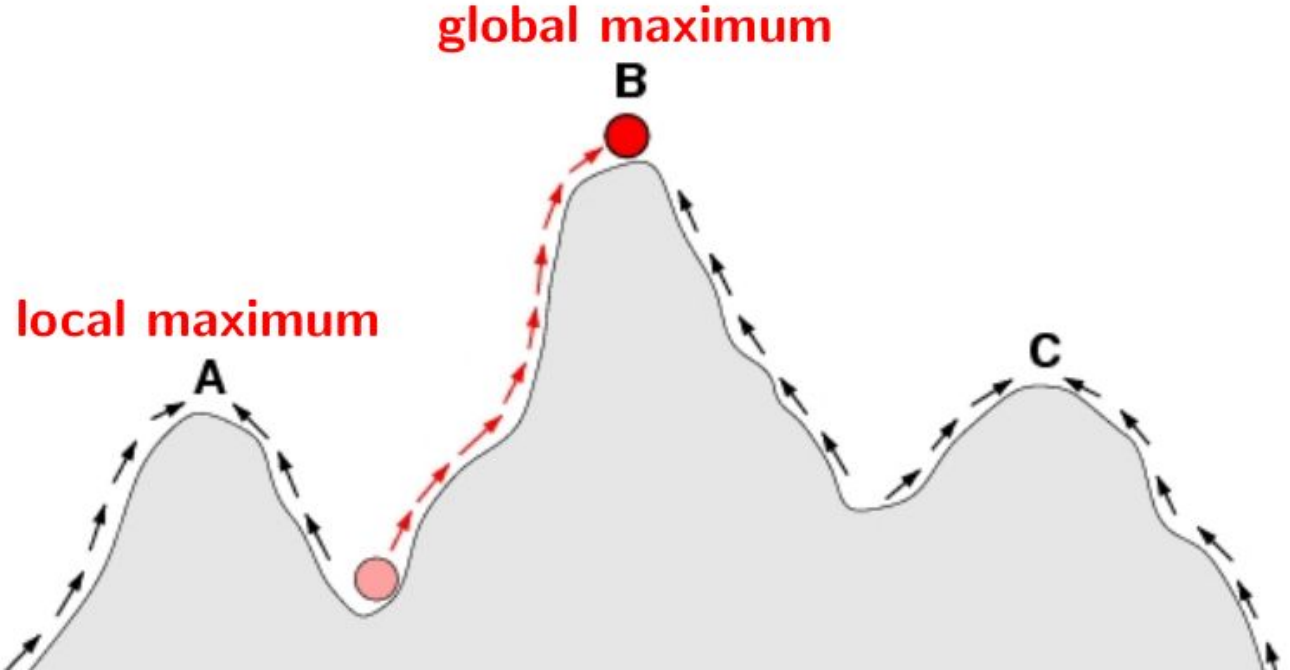
Every global optimum is a local optimum, the inverse is not always correct.

Hill-climbing search (a.k.a. greedy local search)

Continually move in the direction of increasing value (uphill).

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

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

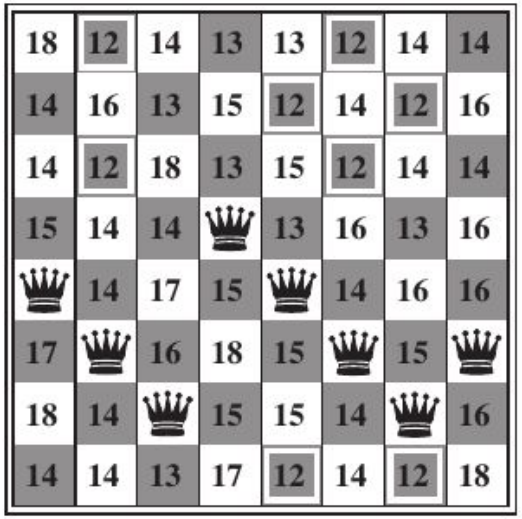


An example of hill-climbing search

Cost function : number of pairs of attacking queens.

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

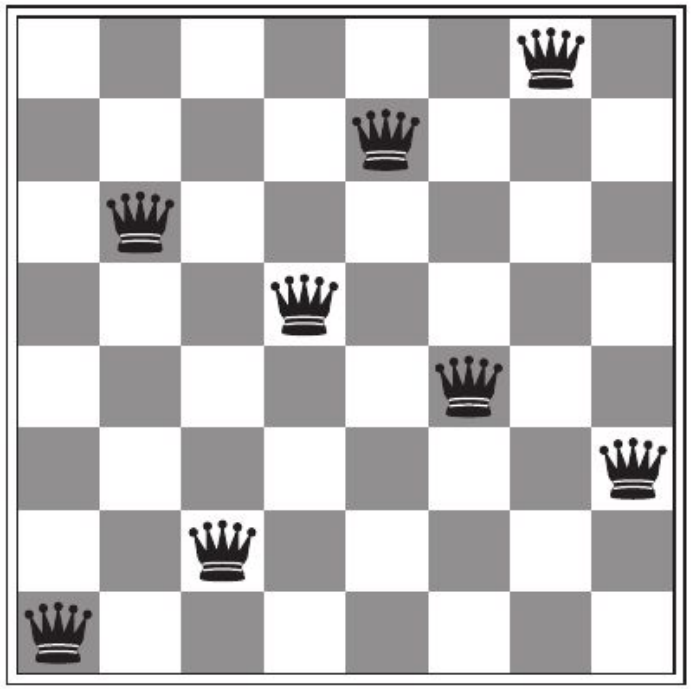
current ← MAKE-NODE(*problem*.INITIAL-STATE)
loop do
neighbor ← a highest-valued successor of *current*
if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
current ← *neighbor*



A state with cost function equal to 17. Numbers indicate the value of moving each queen vertically.

An example of hill-climbing search

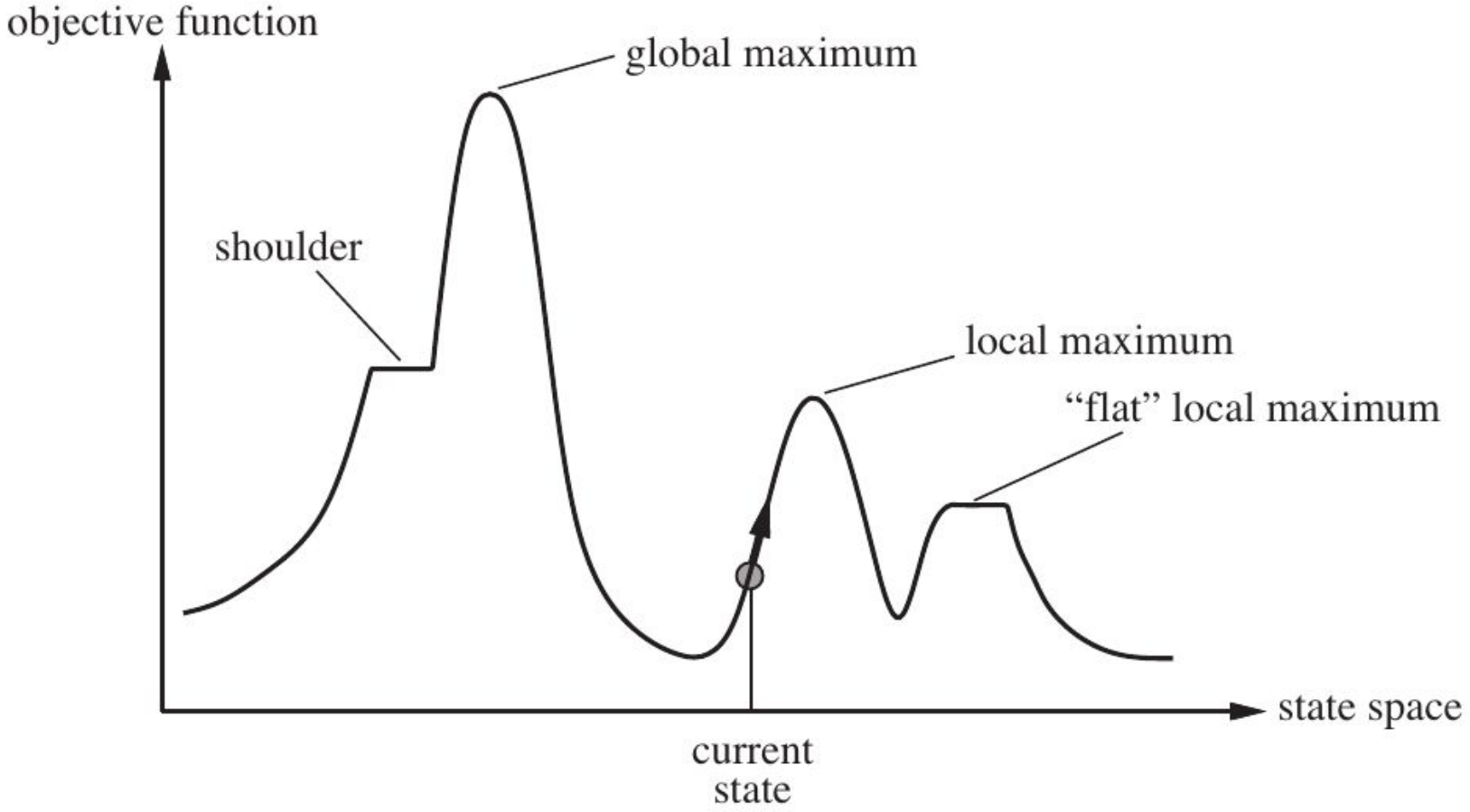
Cost function : number of pairs of attacking queens.



Local minimum with cost function equal to 1. It takes just five steps to reach this state from the previous one. Not bad !

Hill-climbing : the trouble with plateaus

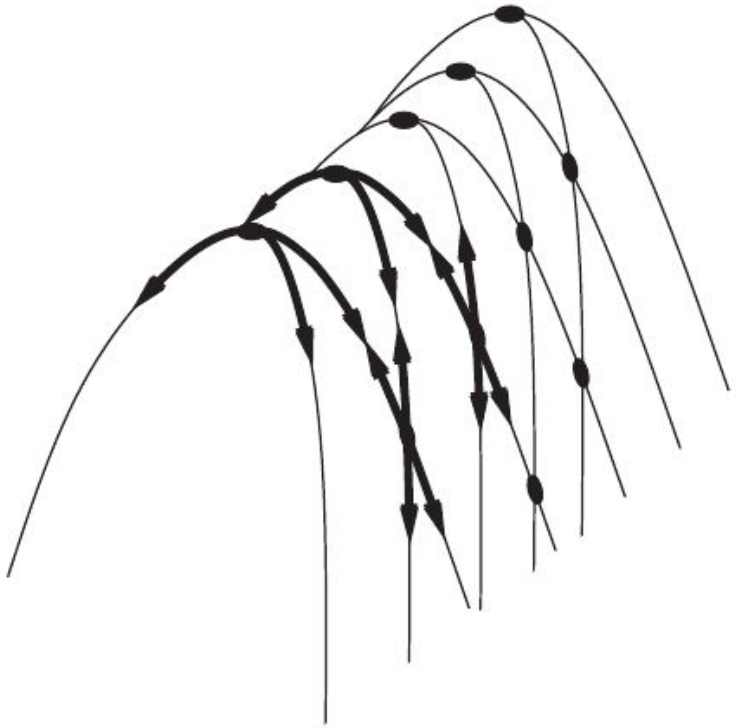
Plateau : a flat area of the state-space landscape.



Hill-climbing gets stuck in plateaus.

Hill-climbing : the trouble with ridges

Ridges : a sequence of local maxima that is very difficult for greedy algorithms to navigate.



Sequence of local maxima that are not directly connected to each other.

Hill-climbing : how to avoid shoulders ?

Starting from a random initial 8-queens state, hill-climbing gets stuck in a local maximum 86% of the time (with an average of 3 steps) and finds the global maximum only 14% (with an average of 4 steps).

Sideways moves

- To avoid staying stuck in a shoulder plateau, we allow hill-climbing to move to neighbor states that have the same value as the current one.
- However, infinite loops may occur. We avoid infinite loops by limiting the number of sideways moves.

By allowing up to 100 sideways moves in the 8-queens problem,

- We increase the percentage of problem instances solved by hill climbing from 14% to 94%.
- However, each successful instance is solved in 21 steps (on average) and failures (local maxima) are reached after 64 steps (on average).

Random-restart hill-climbing

- Repeats the hill-climbing from a randomly generated initial state, until a solution is found.
- Guaranteed to eventually find a solution if the state space is finite.
- The expected number of restarts is $\frac{1}{p}$ if hill-climbing succeeds with probability p at each iteration.

Example of random-restart hill-climbing in the 8-queens problem

- Without sideways moves : $p = 0.14$, we need about 7 iterations on average to find a solution. The average number of steps is 22.
- With sideways moves : $p = 0.94$, we need about 1.06 iterations on average to find a solution. The average number of steps is 68.

Simulated annealing

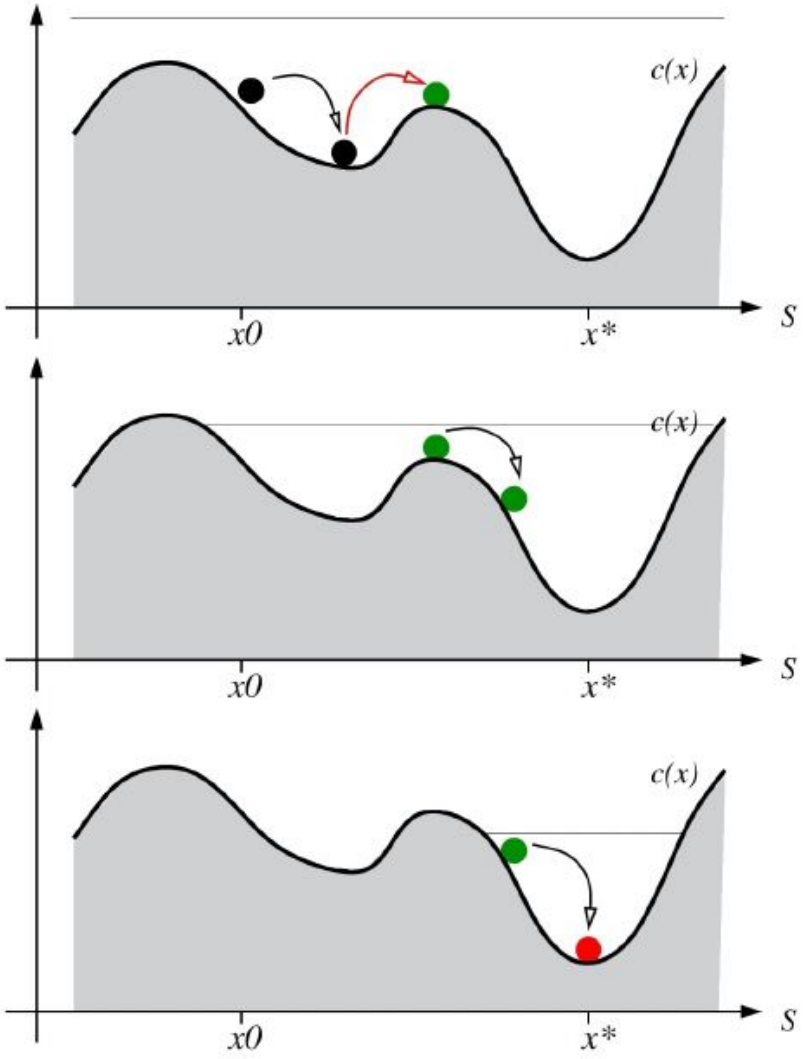
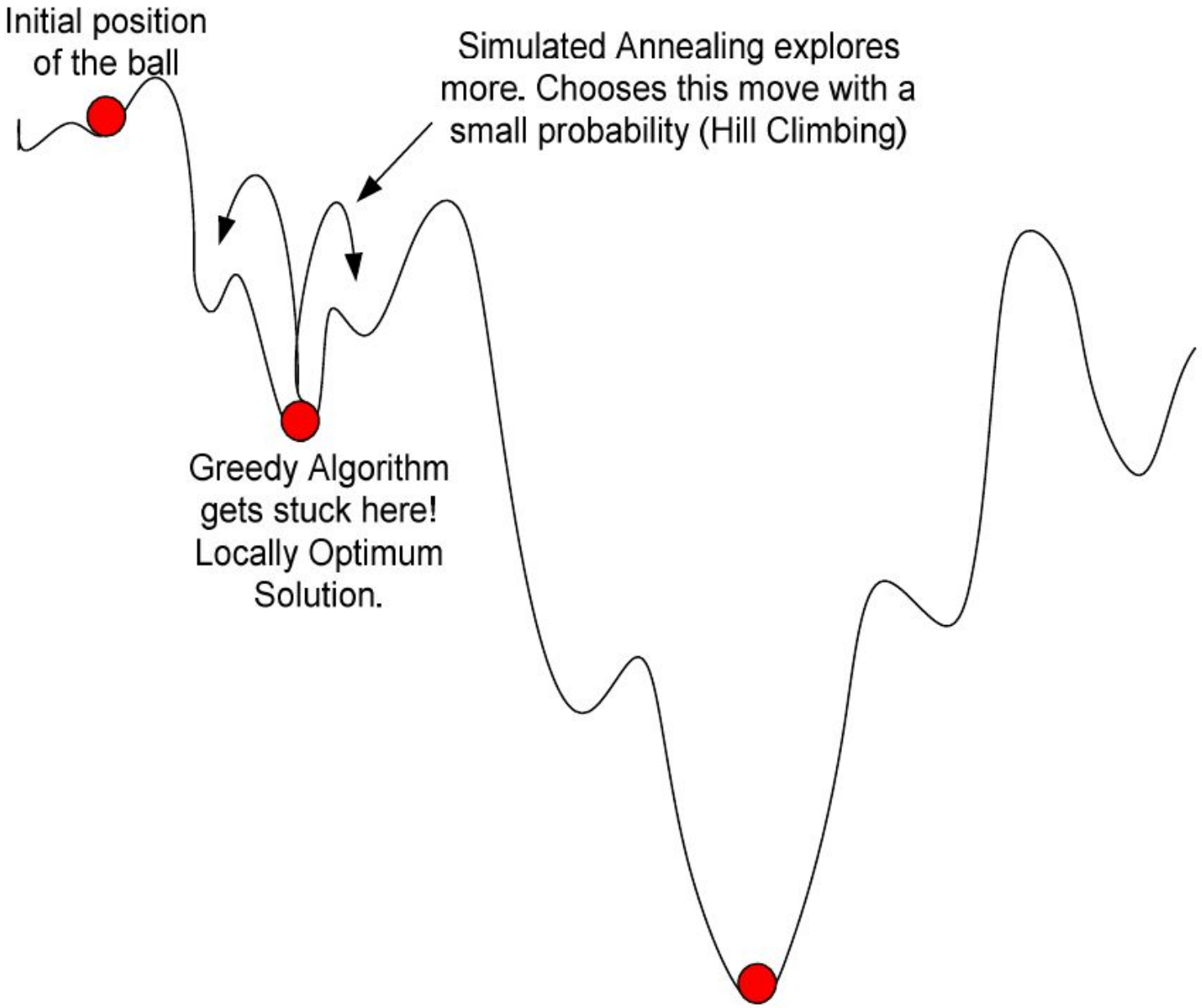


image from cmbi.ru.nl

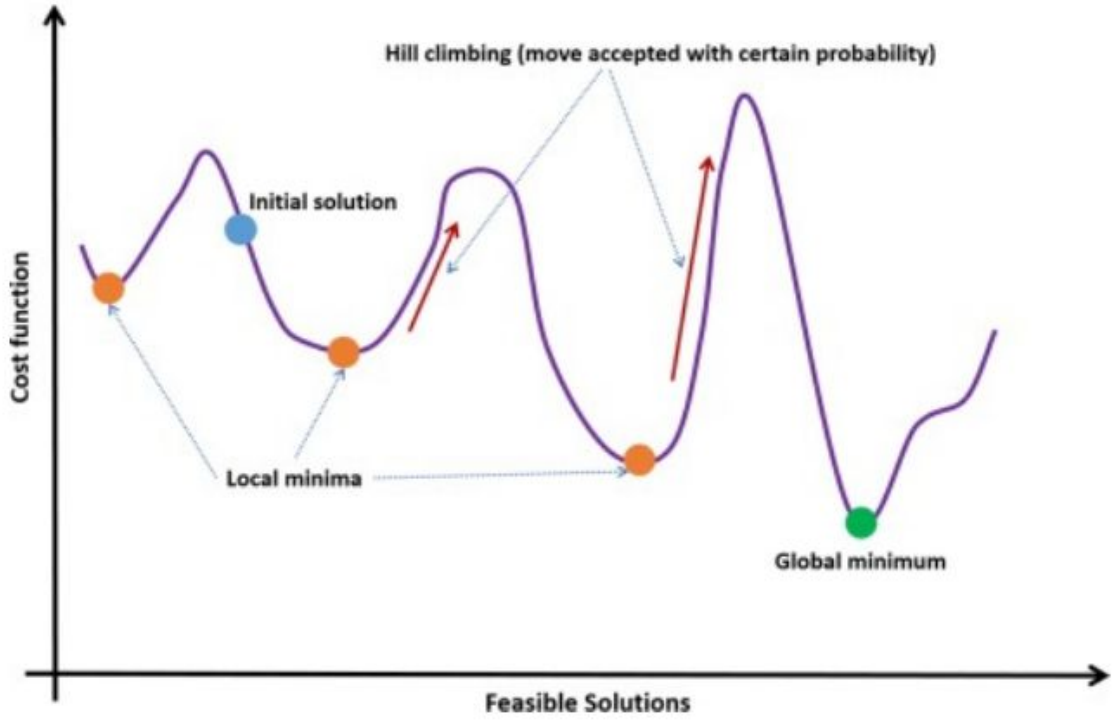
Simulated annealing



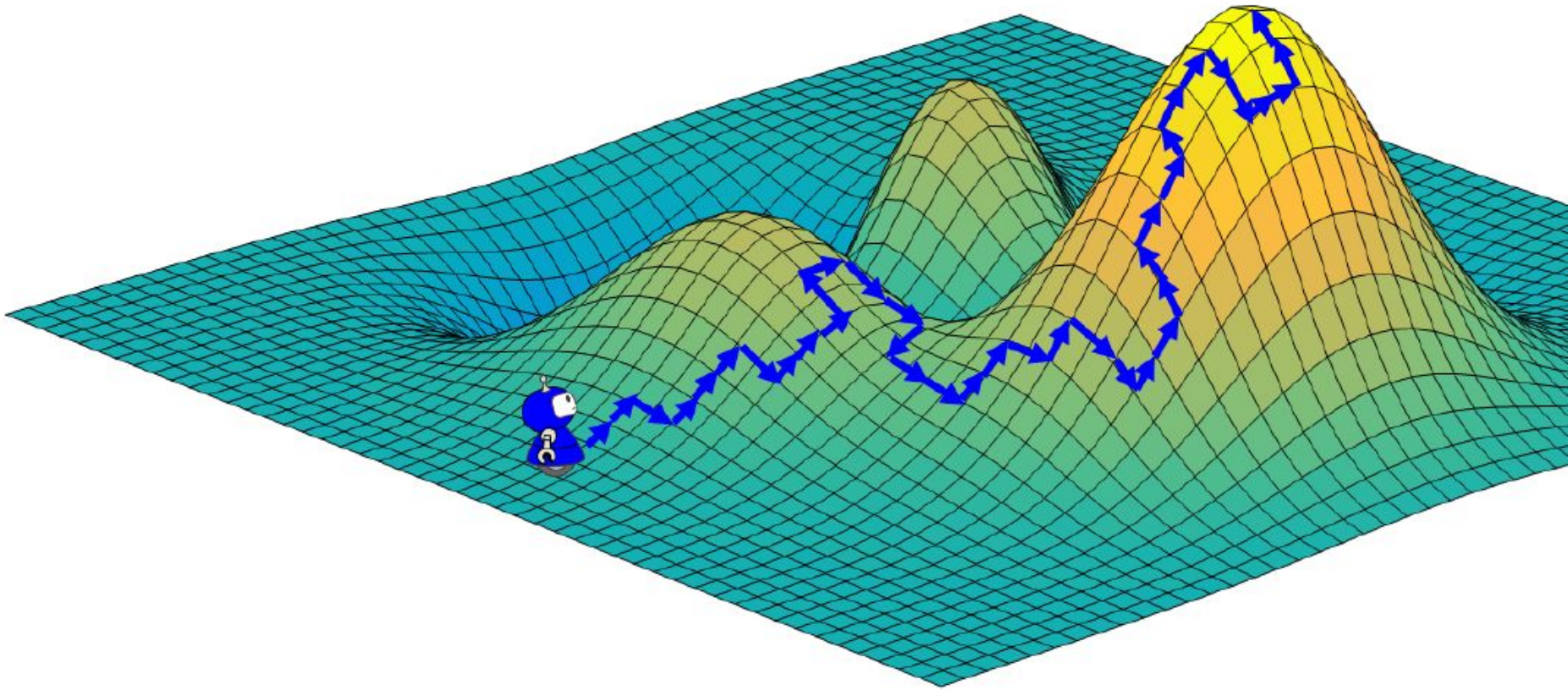
Simulated annealing

1. Step 1 : Initialize the search with a randomly chosen state, and set the temperature T to a high value.
2. Step 2 : Move in a random direction for a fixed distance.
3. Step 3 : Calculate the value of the objective function in the new state and compare it to the value in the old state.
4. Step 4 : If the value has increased, then stay in the new state. Else, stay in the new state with a probability that is proportional to the change in value, otherwise move back to the old state.
5. Step 5 : Decrease the temperature and go back to step 2.

Simulated annealing



Simulated annealing



Simulated annealing

function SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  inputs: $problem$, a problem
    $schedule$, a mapping from time to "temperature"

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

Example of temperature function (schedule) : $T = \frac{100}{t^2}$. The key thing is that the temperature starts high and decreases over time.

Local beam search

- Start with k randomly chosen initial states.
- At each step, generate all the successors of the k states, and retain the k best ones.
- Stop when a goal state is reached or when no further local improvement is possible.

- Local beam search seems similar to running hill-climbing k times, but it is different.
- In a local beam search, the k search paths are not independent. The successors of the k states are compared with each other.
- States with many "good" successors dominate local beam searches.

Stochastic local beam search

- Local beam search suffers from a lack of diversity among the k paths. The search often ends up concentrated in a small region.
- Stochastic local beam search overcomes this problem by randomly generating the successors and keeping them with probabilities proportional to their values.