CS 581

Advanced Artificial Intelligence

January 29, 2024

Announcements / Reminders

- **Please follow the Week 03 To Do List instructions (if you haven't already)**
- **Written Assignment #01: to be posted soon**
- **Programming Assignment #01: to be posted soon**

Teaching Assistants

Name	e-mail	Office hours
Gawade, Vishal	vgawade@hawk.iit.edu	Tuesdays 12:30 PM - 01:30 PM CST in SB 108
Zhou, Xiaoting	xzhou70@hawk.iit.edu	Thursdays: 10:00 AM - 11:00 AM CST

TAs will:

- assist you with your assignments,
- hold office hours to answer your questions,
- grade your assignments (**a specific TA will be assigned to you**).

Take advantage of their time and knowledge!

DO NOT email them with questions unrelated to lab grading.

Make time to meet them during their office hours.

Add a **[CS581 Spring 2024]** prefix to your email subject when contacting TAs, please.

Plan for Today

- **Solving problems by Searching**
  - **Local Search Algorithms**

“Hill Climbing” (Greedy Local) Search and Romanian Roadtrip Example

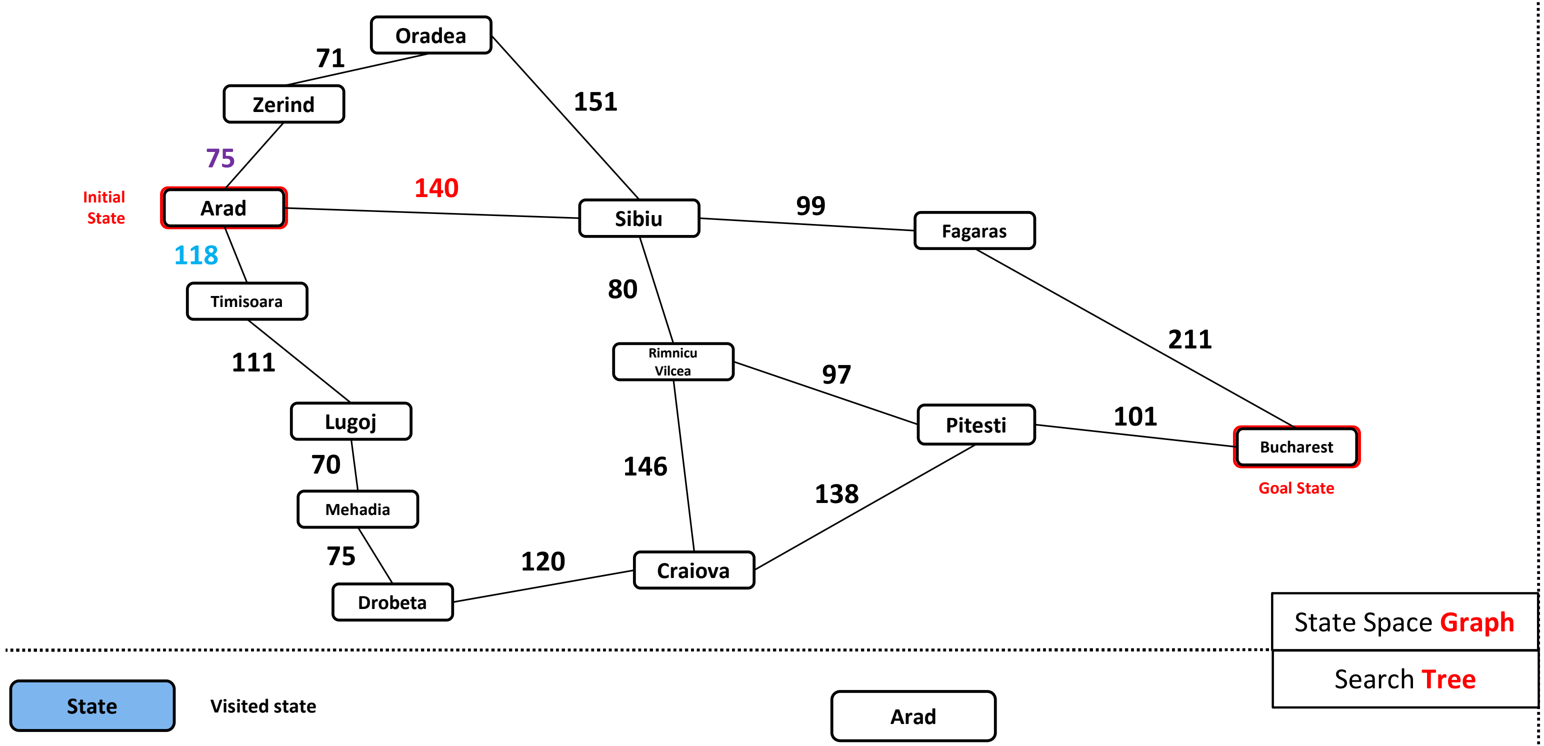
Greedy Local: Evaluation Function

Calculate / obtain:

f(n) = **ACTION-COST($State_a$, $toState_n$, $State_n$)**

A state n with minimum (or maximum) f(n) should be chosen for expansion

Romanian Roadtrip: Greedy Local



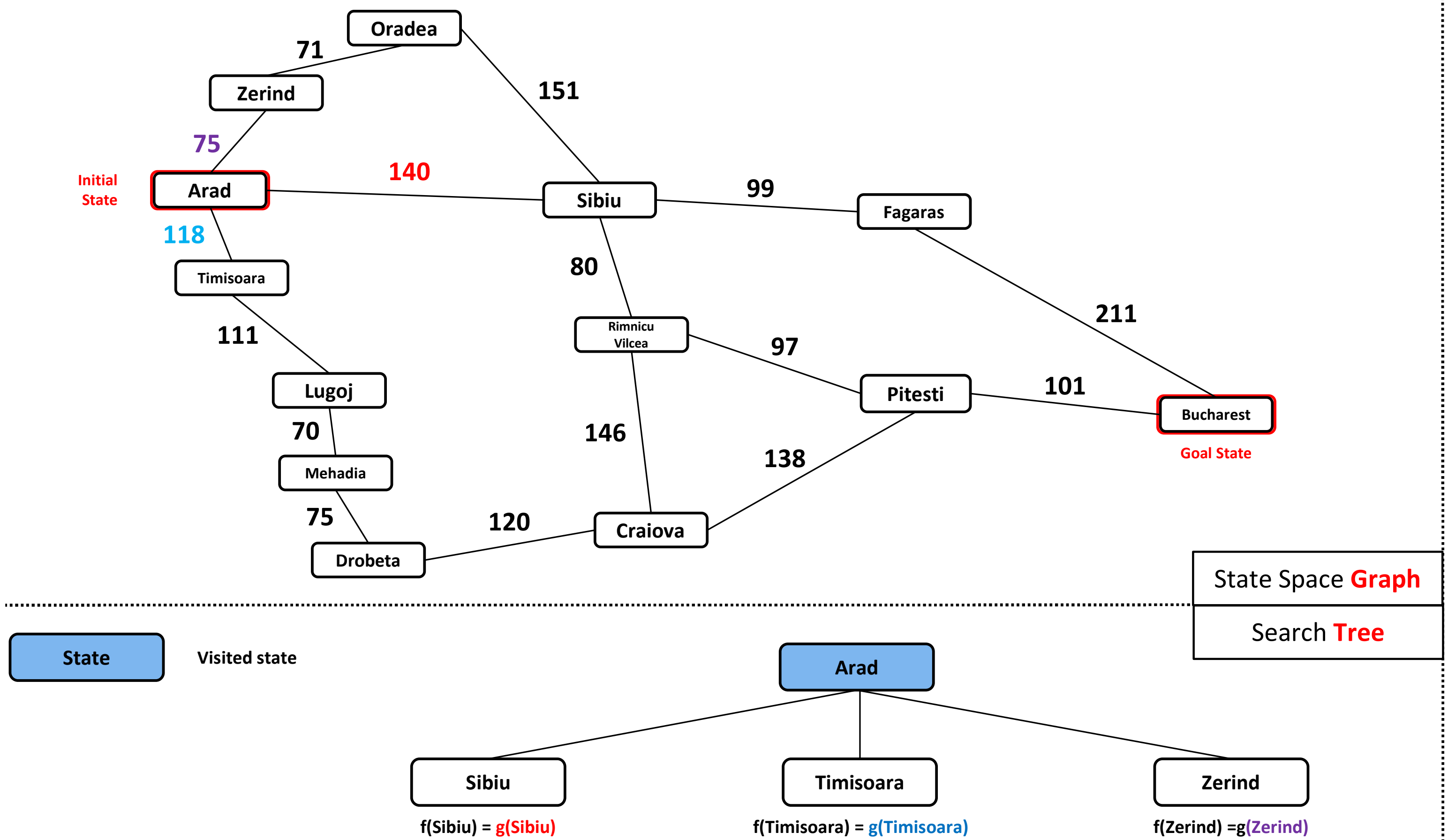
Assumption:
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
  - get stuck there

  or
  - Infinite loop

Romanian Roadtrip: Greedy Local



Assumption:

We don't "go" to a repeated state

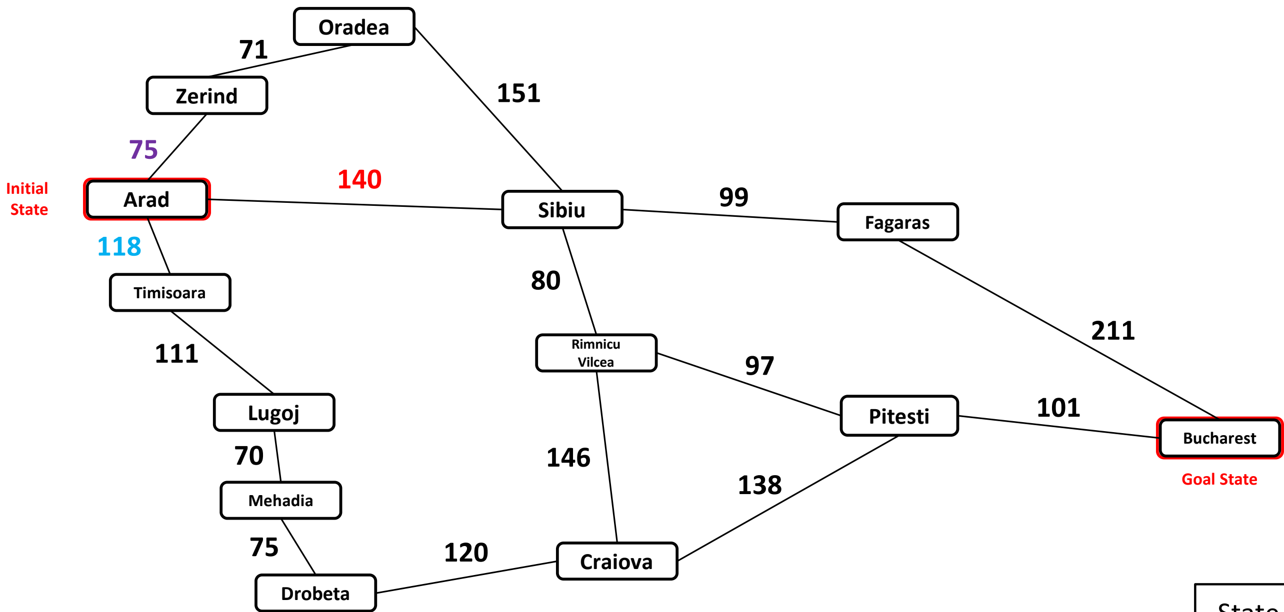
Alternatively:

- We could go to a repeated state end
  - get stuck there

  or

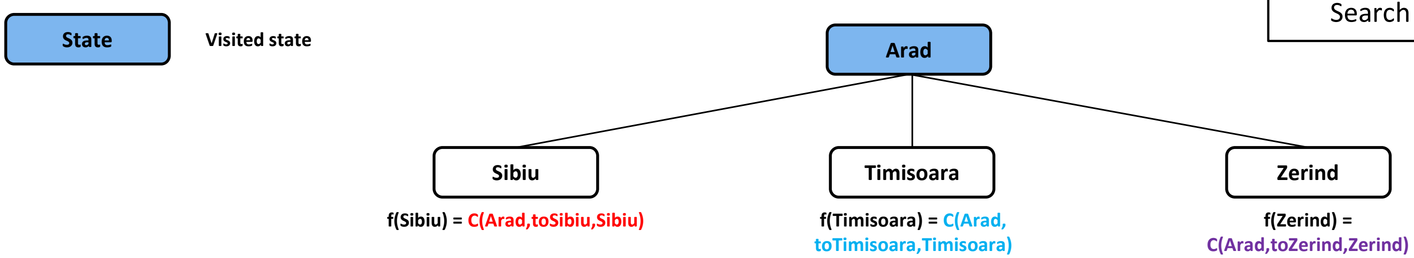
  - Infinite loop

Romanian Roadtrip: Greedy Local



State Space **Graph**

Search **Tree**

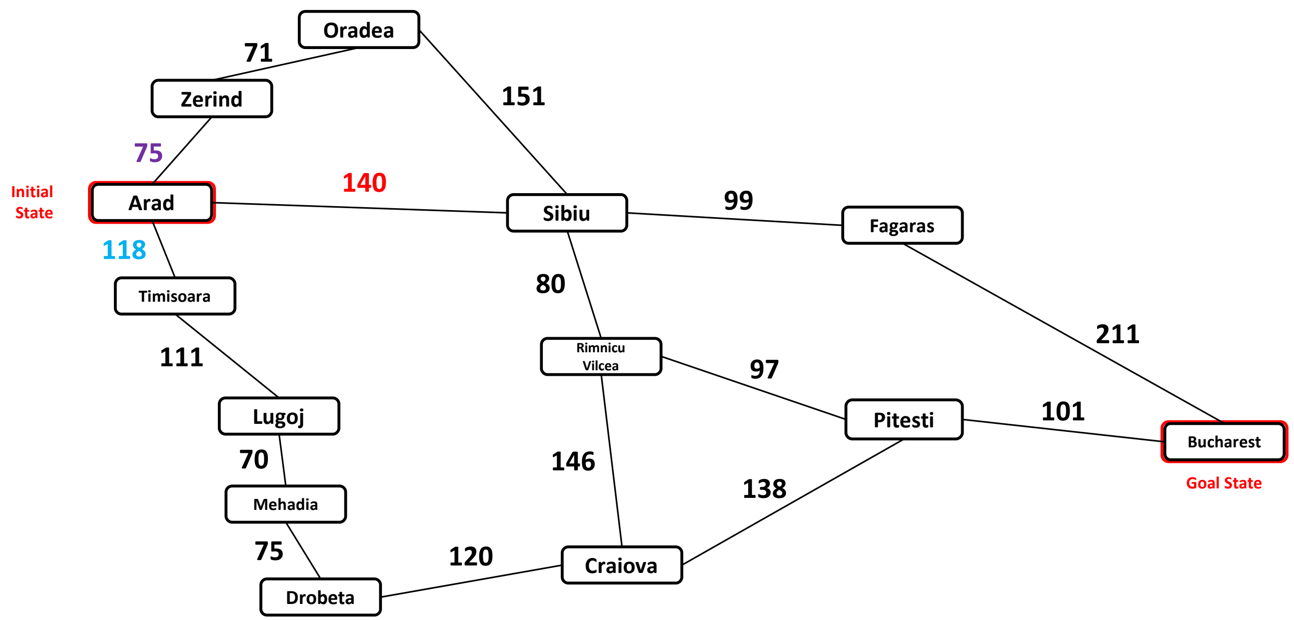


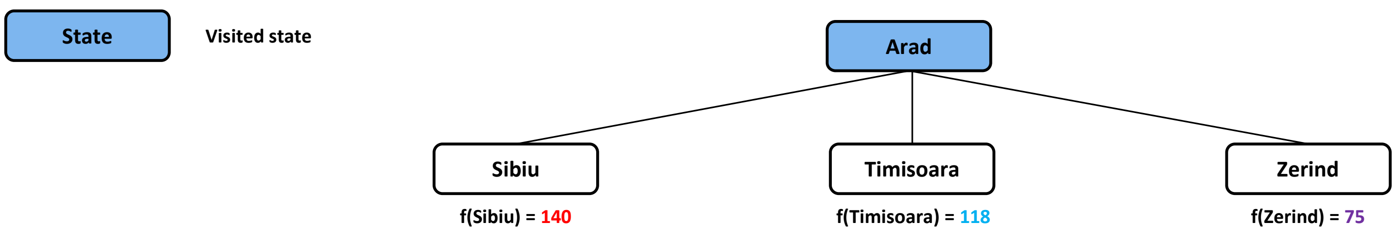
Assumption:
We don't "go" to a repeated state

Alternatively:
- **We could go to a repeated state end**
  - **get stuck there**

  or
  - **Infinite loop**

Romanian Roadtrip: Greedy Local



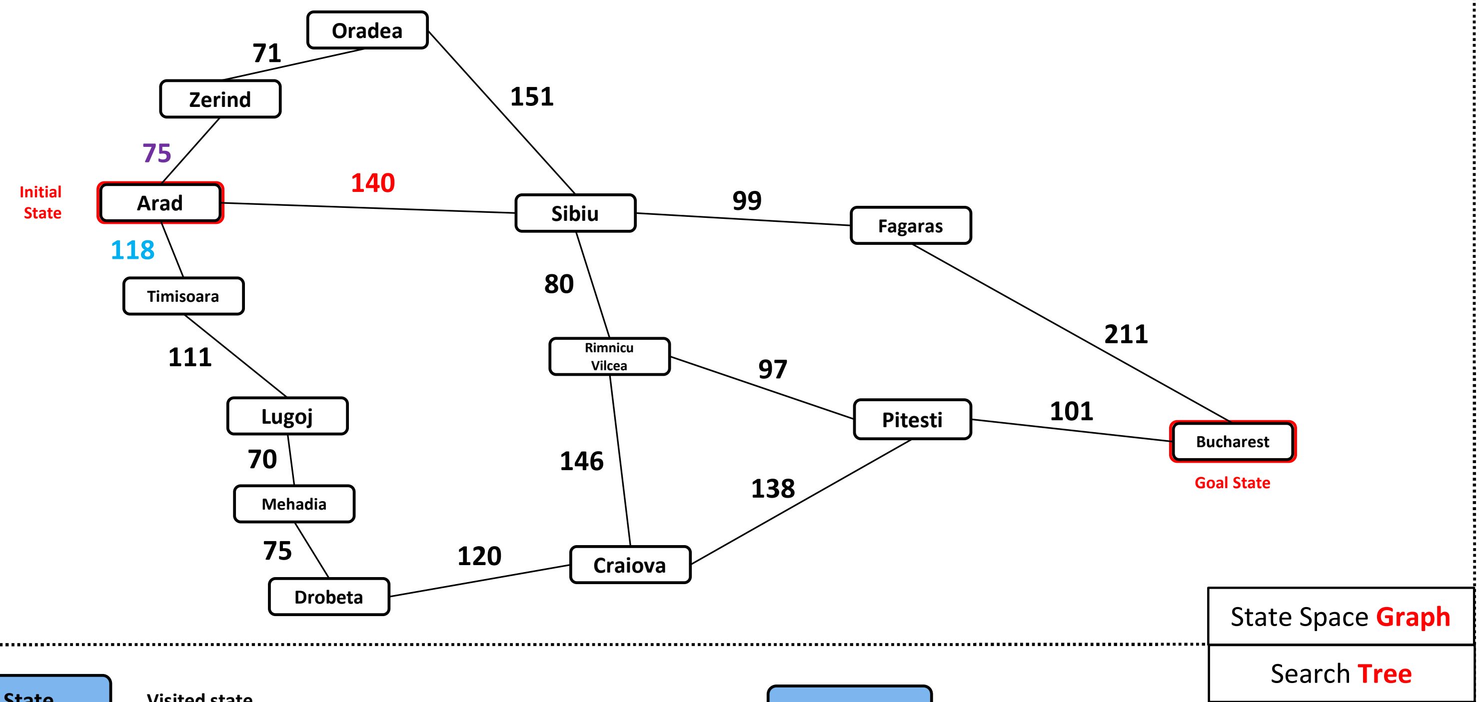


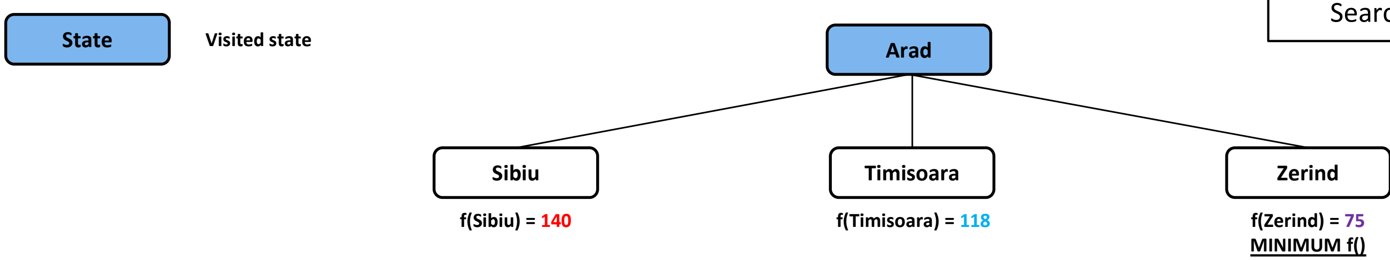
Assumption:
We don't "go" to a repeated state

Alternatively:
- We could go to a repeated state end
  - get stuck there

  or
  - Infinite loop

Romanian Roadtrip: Greedy Local





Assumption:
We don't "go" to a repeated state

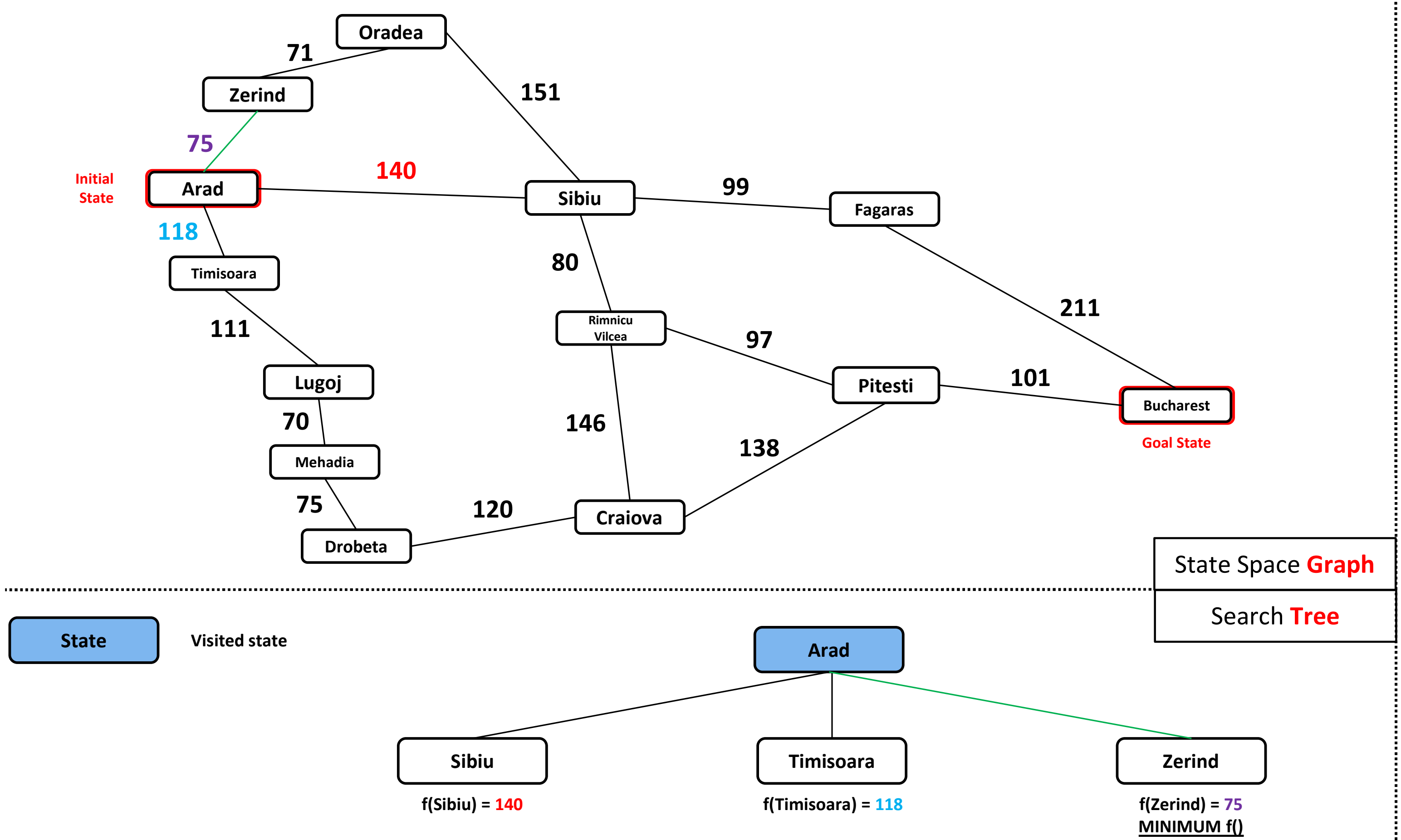
Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or

  - **Infinite loop**

Romanian Roadtrip: Greedy Local



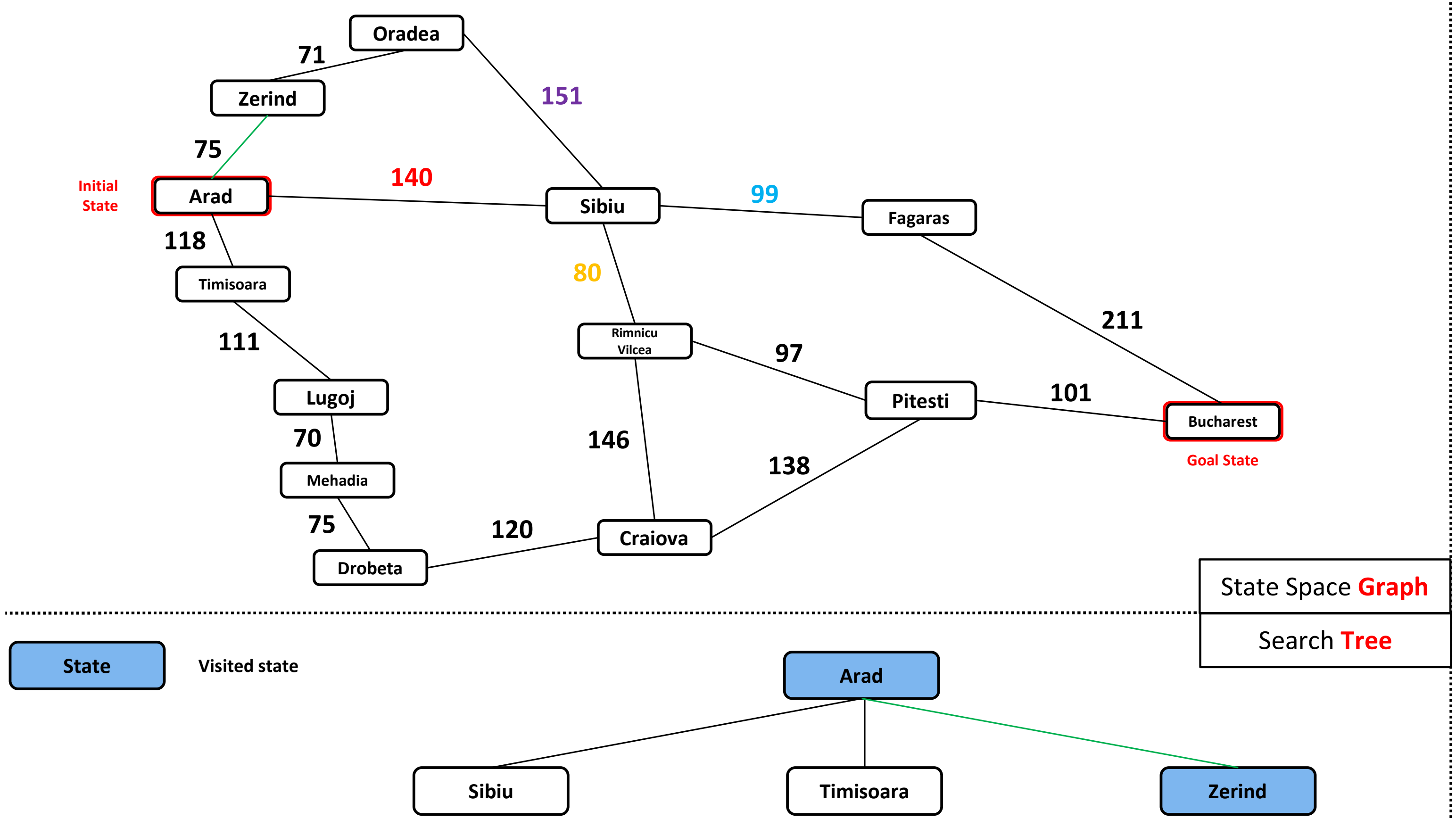
Assumption:
We don't "go" to a repeated state

Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or
  - **Infinite loop**

Romanian Roadtrip: Greedy Local

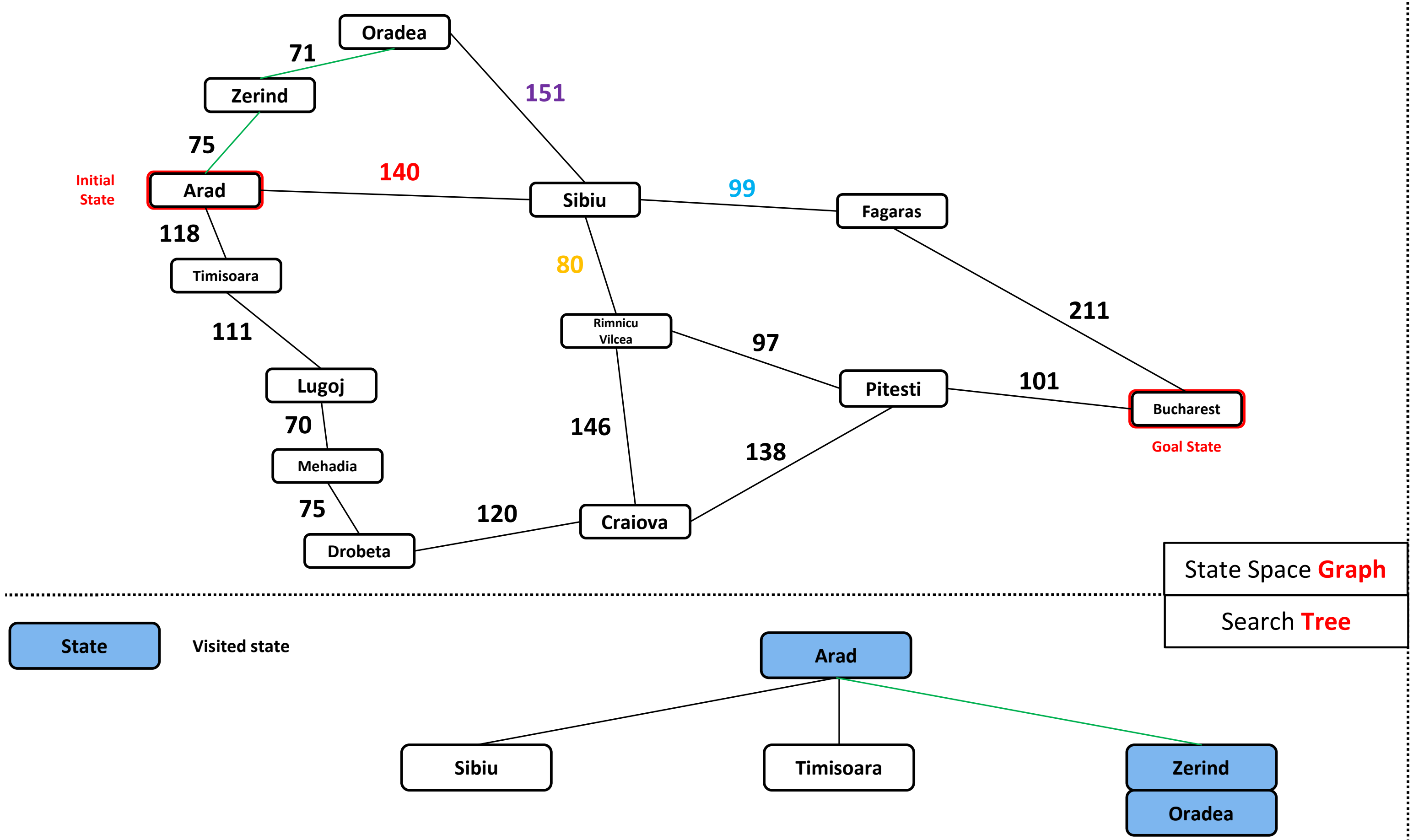


Assumption:
We don't "go" to a repeated state

Alternatively:
- **We could go to a repeated state end**
  - **get stuck there**

  or
  - **Infinite loop**

Romanian Roadtrip: Greedy Local

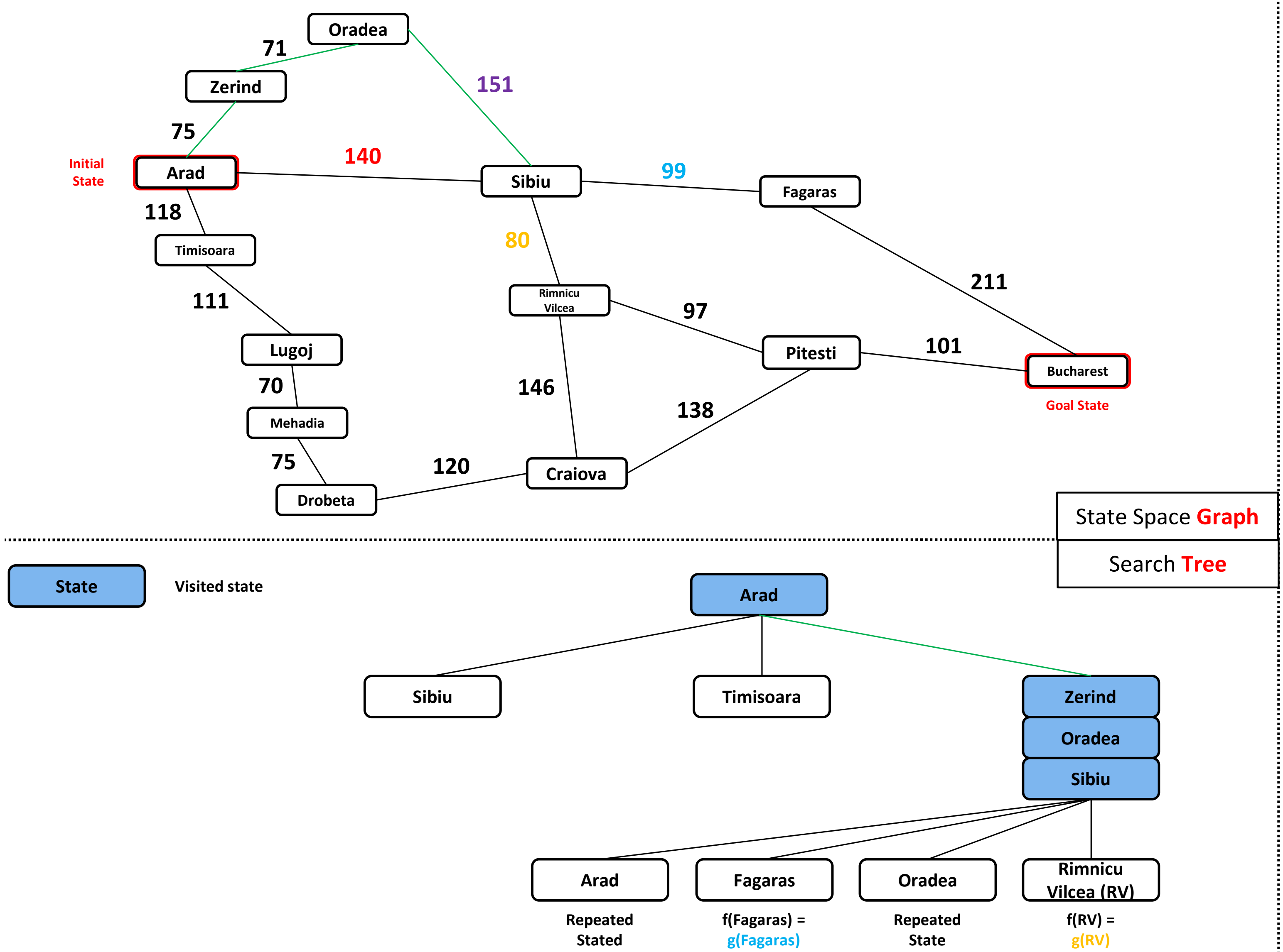


Assumption:
We don't "go" to a repeated state

Alternatively:
- We could go to a repeated state end
  - get stuck there

  or
  - Infinite loop

Romanian Roadtrip: Greedy Local



Assumption:
We don't "go" to a repeated state

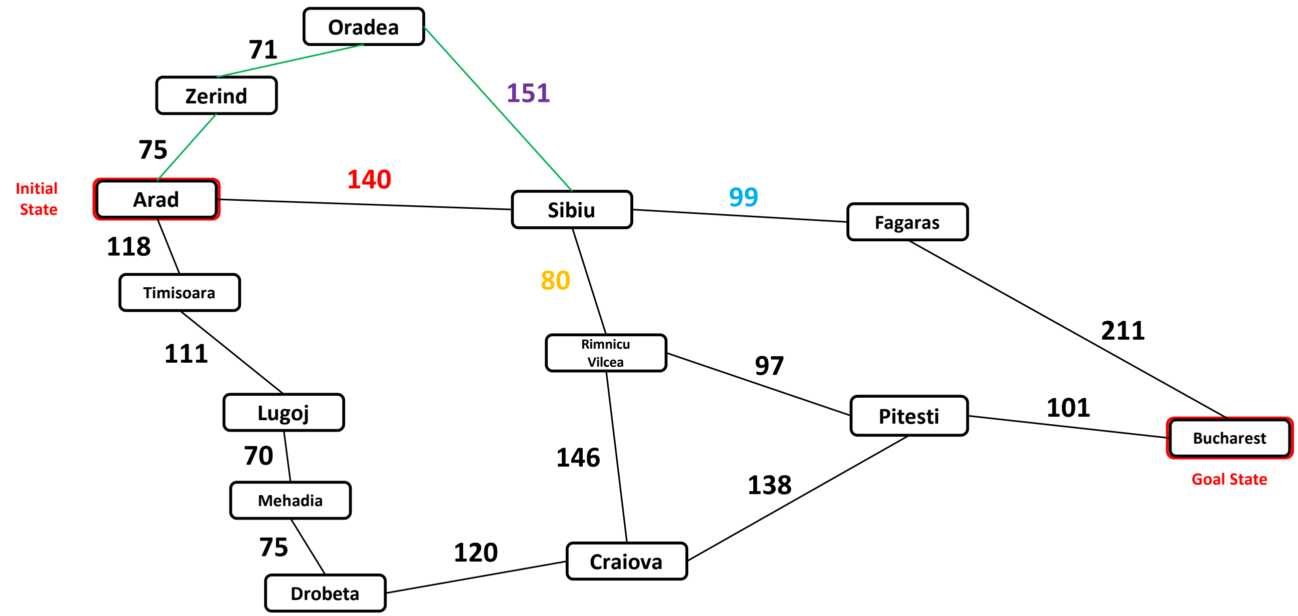
Alternatively:

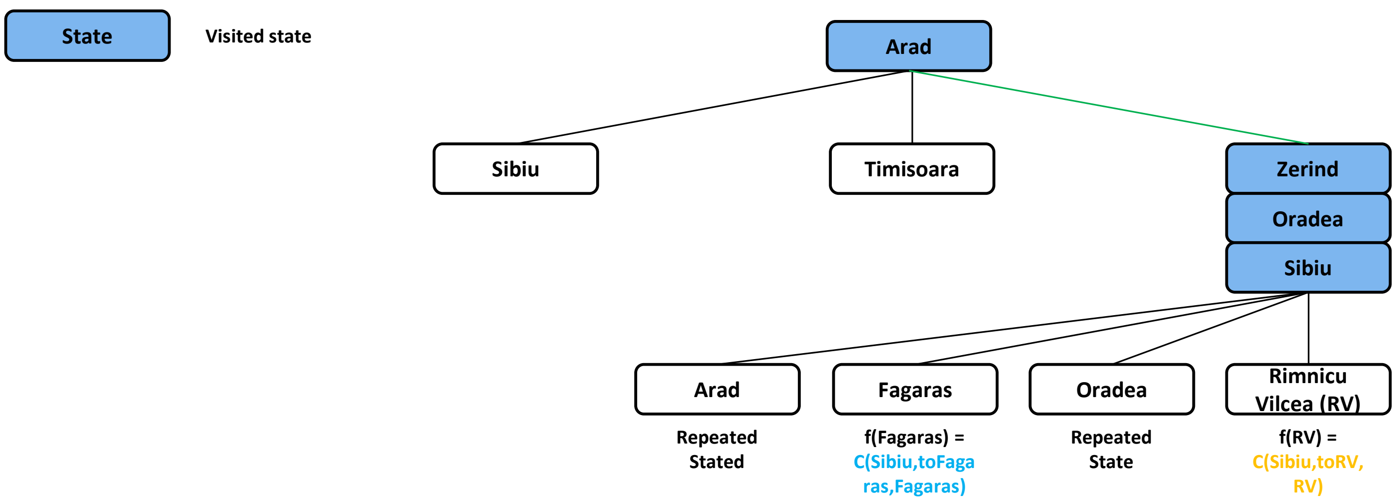
- **We could go to a repeated state end**
  - **get stuck there**

  or

  - **Infinite loop**

Romanian Roadtrip: Greedy Local





Assumption:

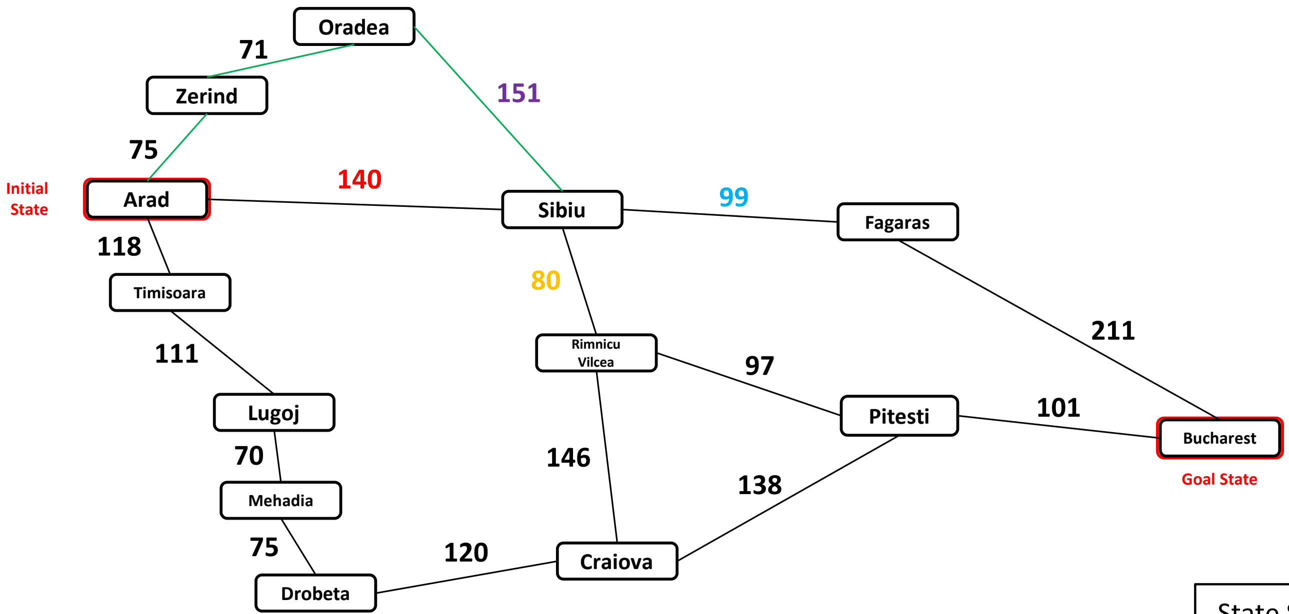
We don't "go" to a repeated state

Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

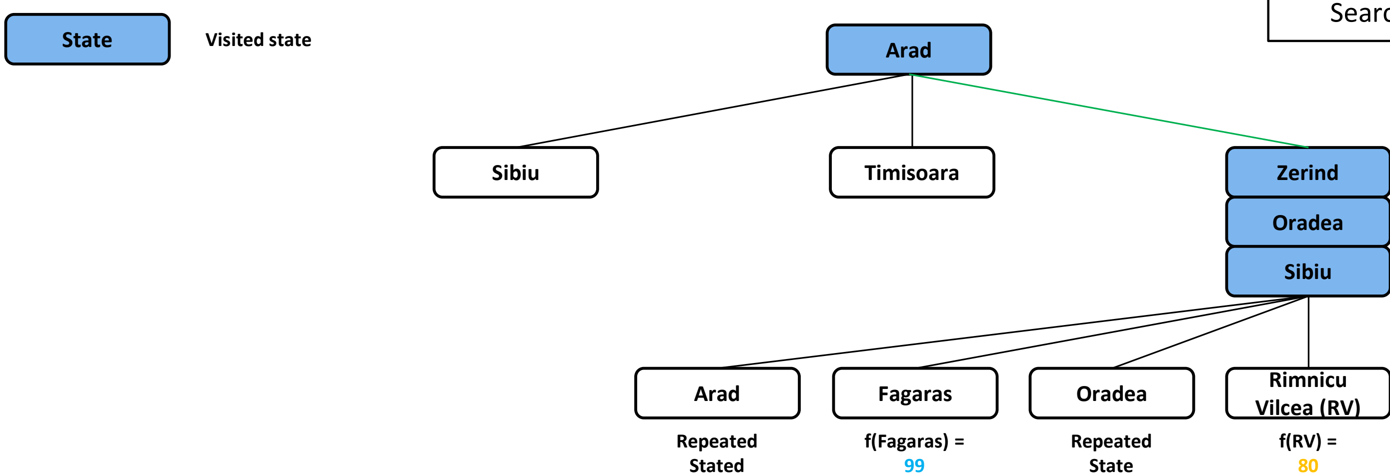
  or
  - **Infinite loop**

Romanian Roadtrip: Greedy Local



State Space **Graph**

Search **Tree**



Assumption:

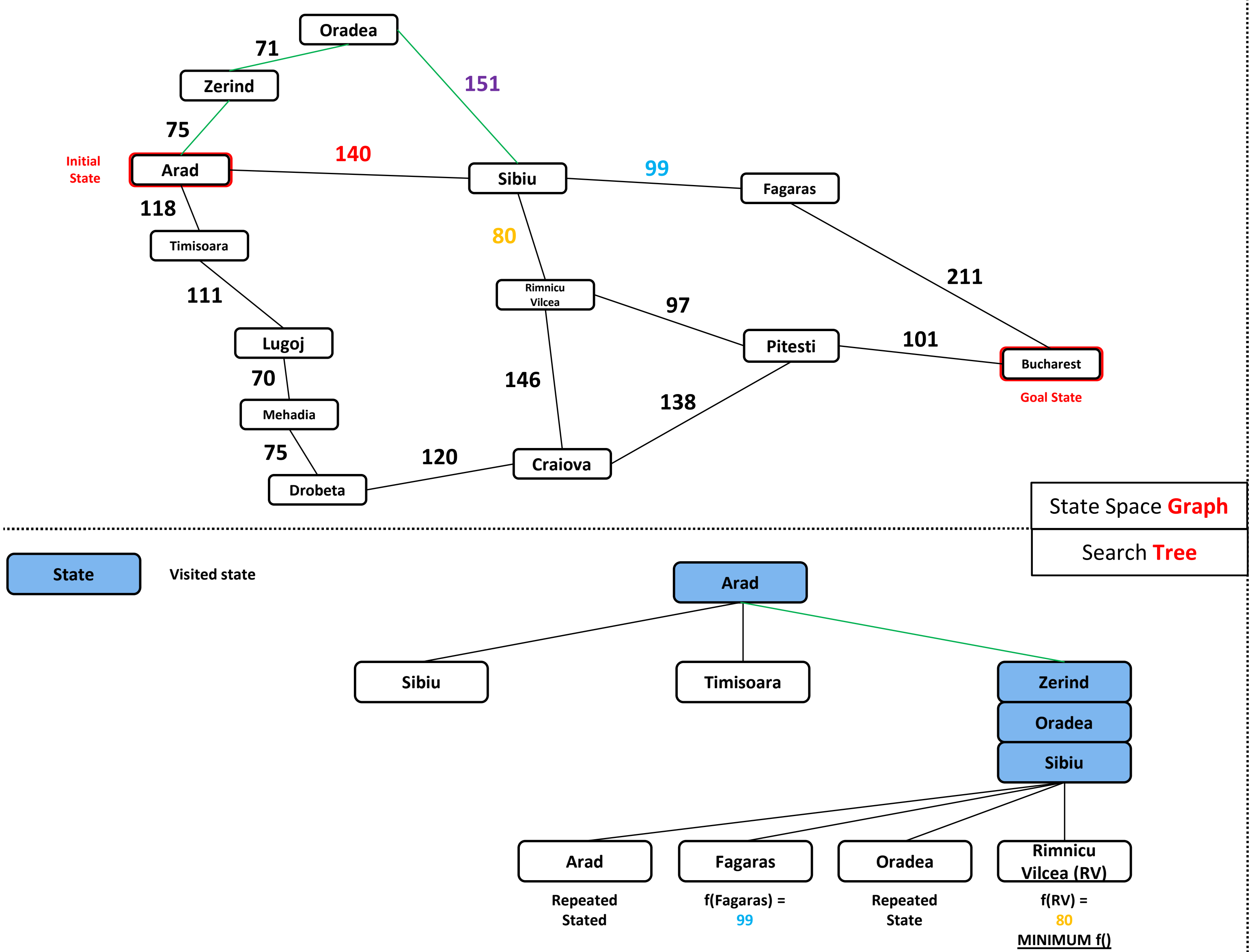
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
  - get stuck there

  or
  - Infinite loop

Romanian Roadtrip: Greedy Local



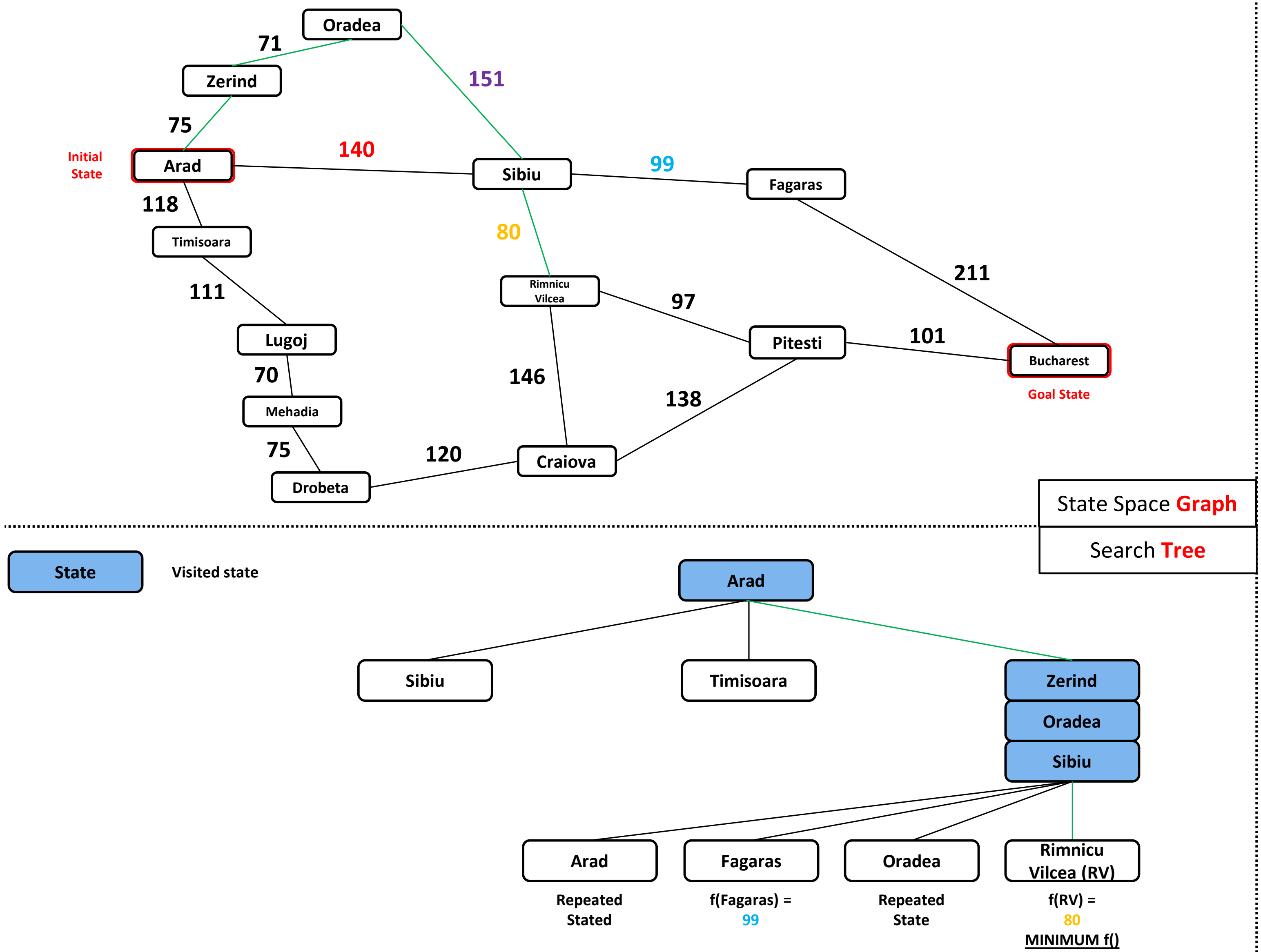
Assumption:
We don't "go" to a repeated state

Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or
  - **Infinite loop**

Romanian Roadtrip: Greedy Local



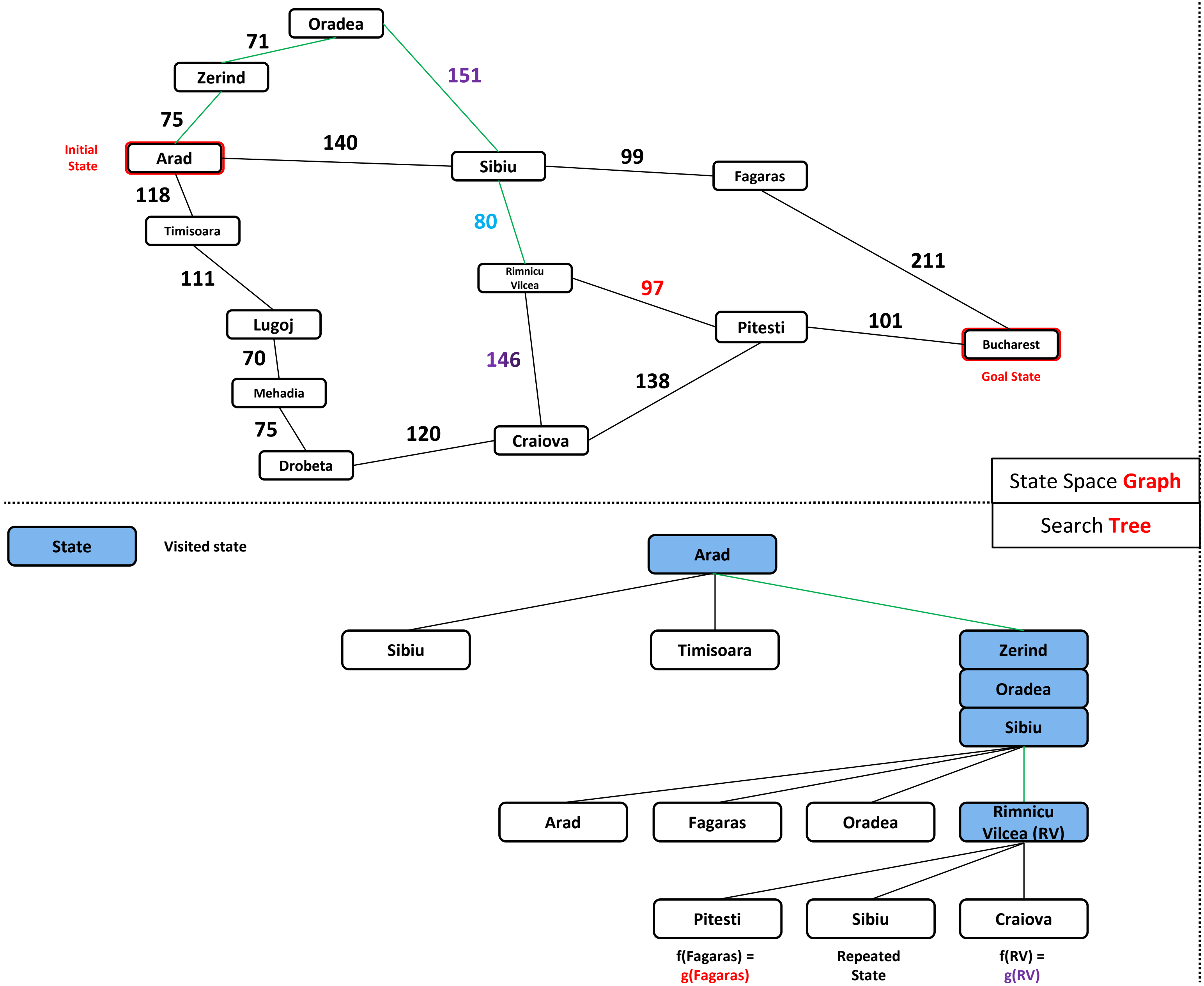
Assumption:
We don't "go" to a repeated state

Alternatively:
- We could go to a repeated state end
  - get stuck there

  or

  - Infinite loop

Romanian Roadtrip: Greedy Local



Assumption:

We don't "go" to a repeated state

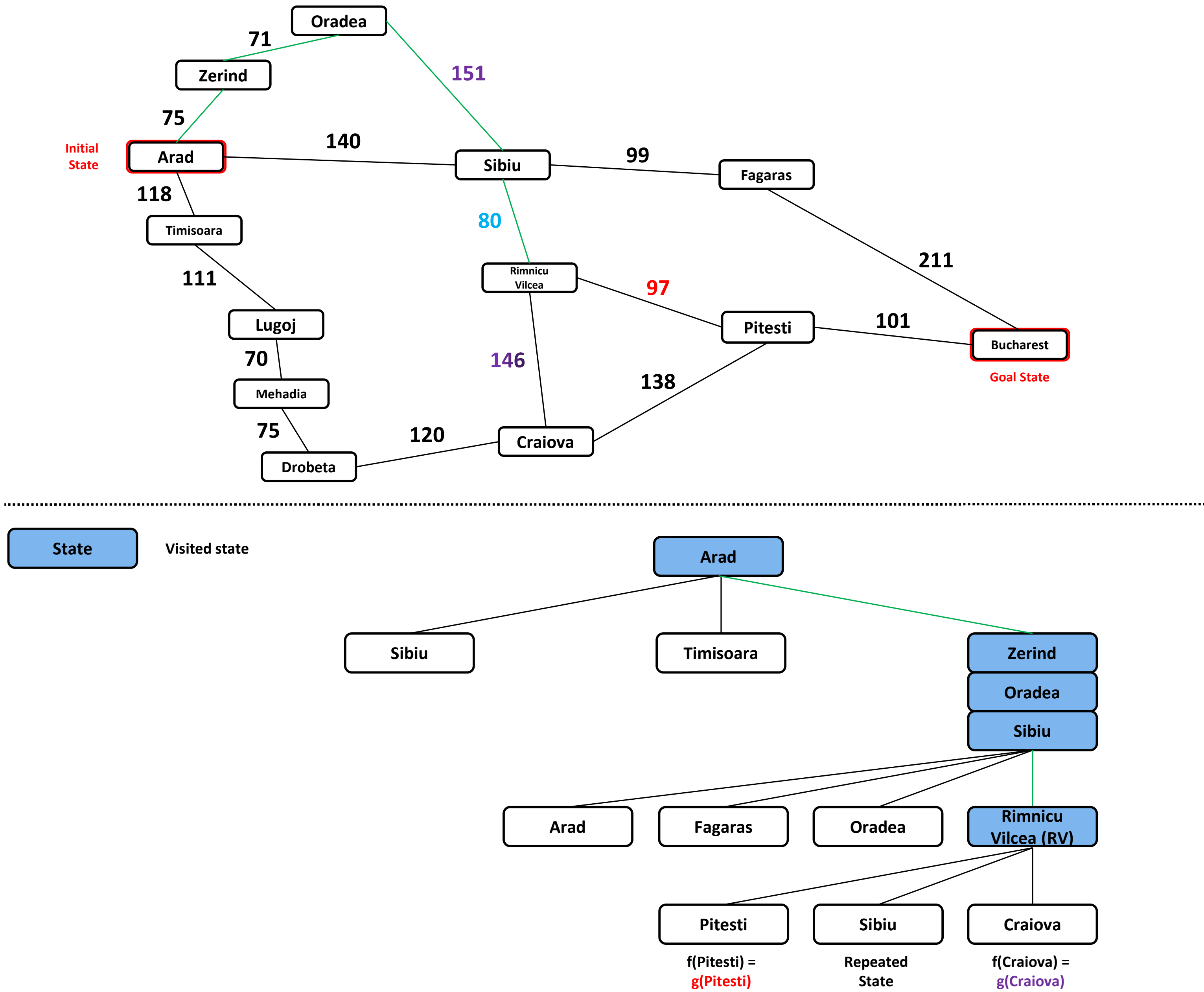
Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or

  - **Infinite loop**

Romanian Roadtrip: Greedy Local



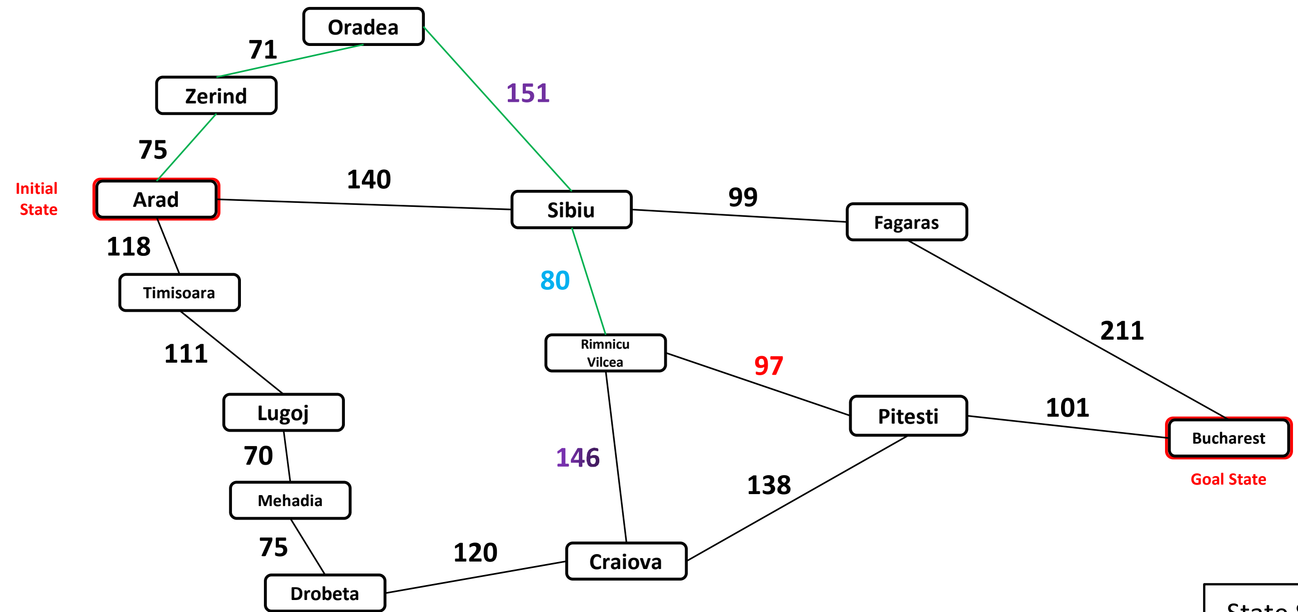
Assumption:
We don't "go" to a repeated state

Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

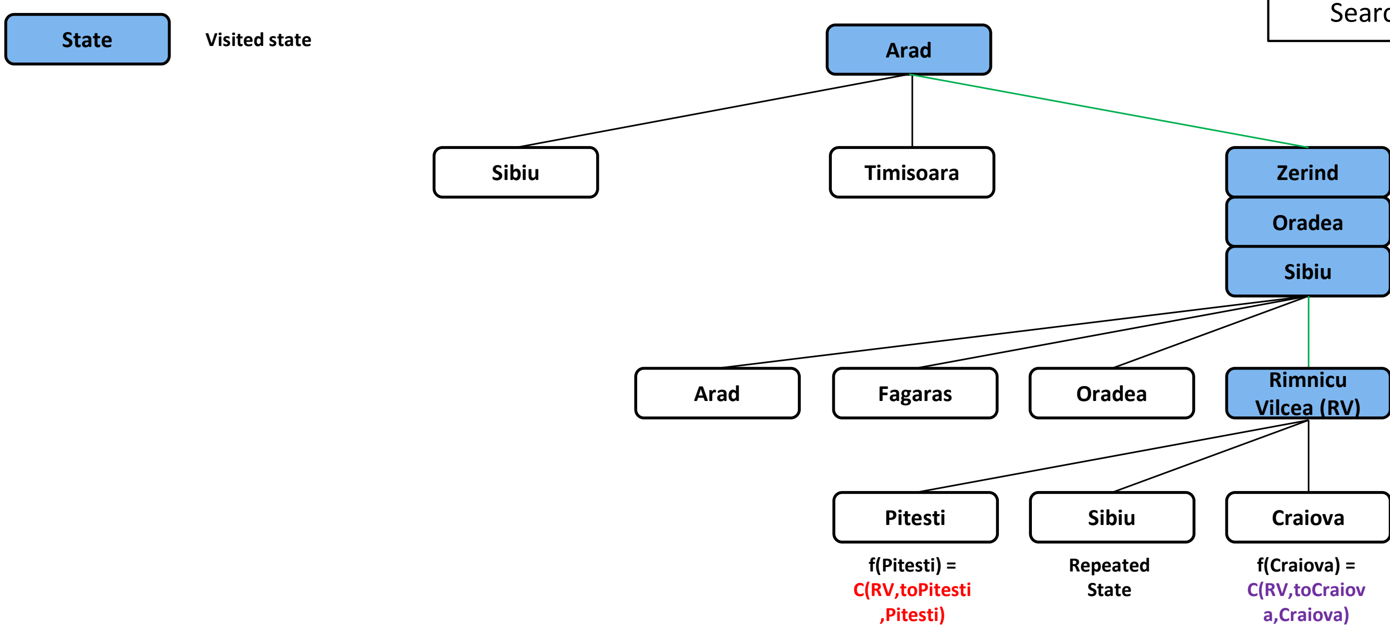
  or
  - **Infinite loop**

Romanian Roadtrip: Greedy Local



State Space Graph

Search Tree



Assumption:

We don't "go" to a repeated state

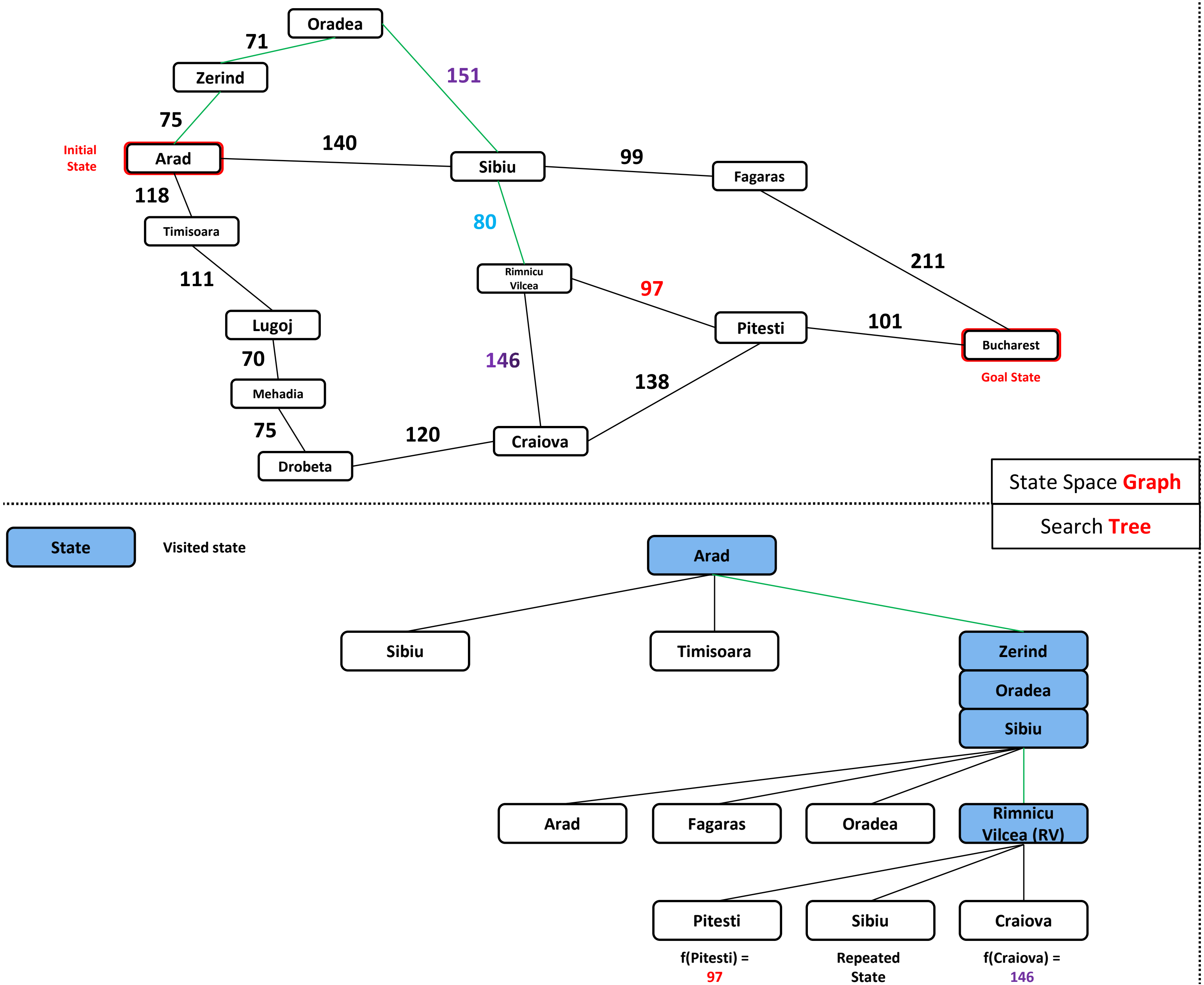
Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or

  - **Infinite loop**

Romanian Roadtrip: Greedy Local



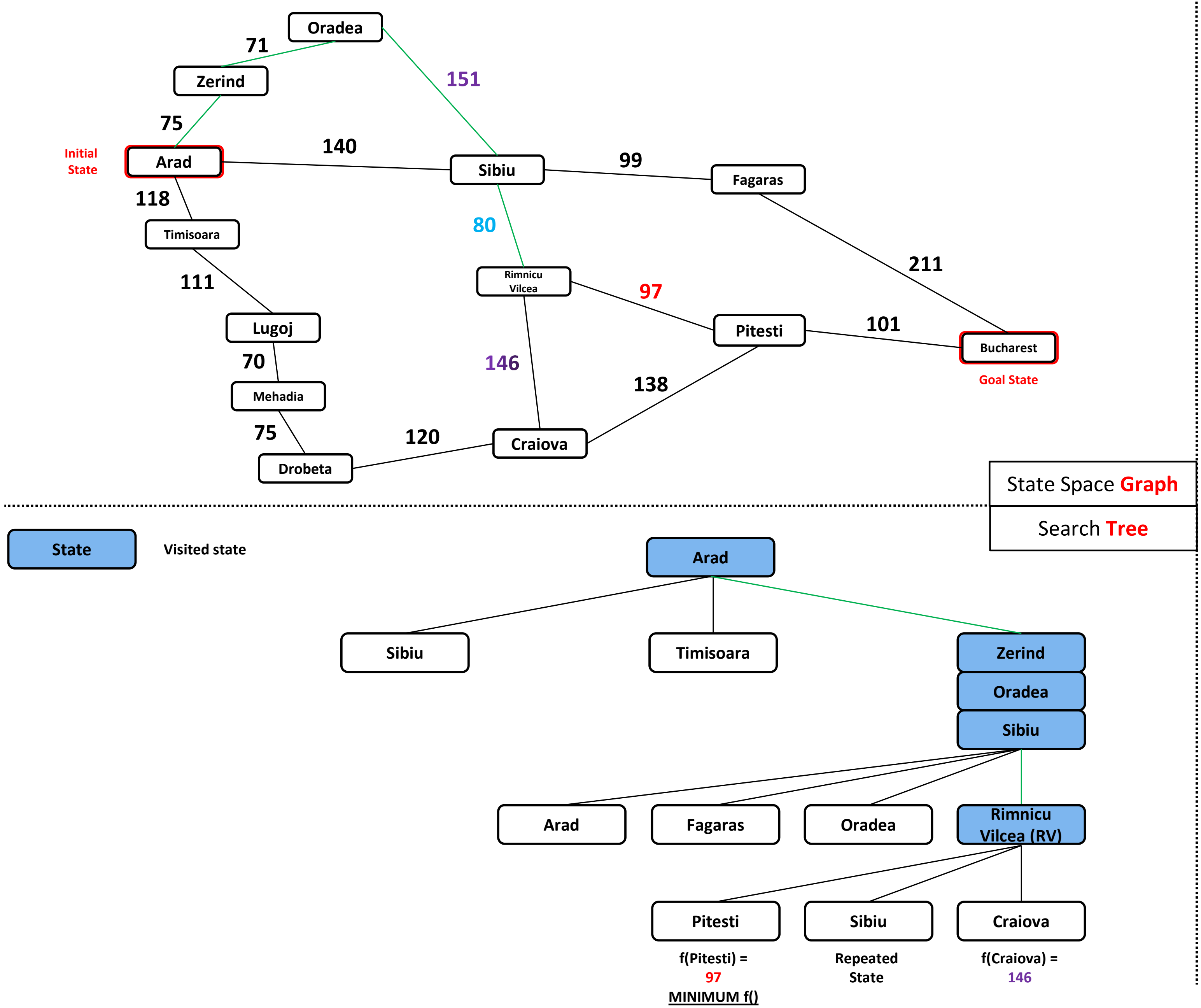
Assumption:
We don't "go" to a repeated state

Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or
  - **Infinite loop**

Romanian Roadtrip: Greedy Local



Assumption:

We don't "go" to a repeated state

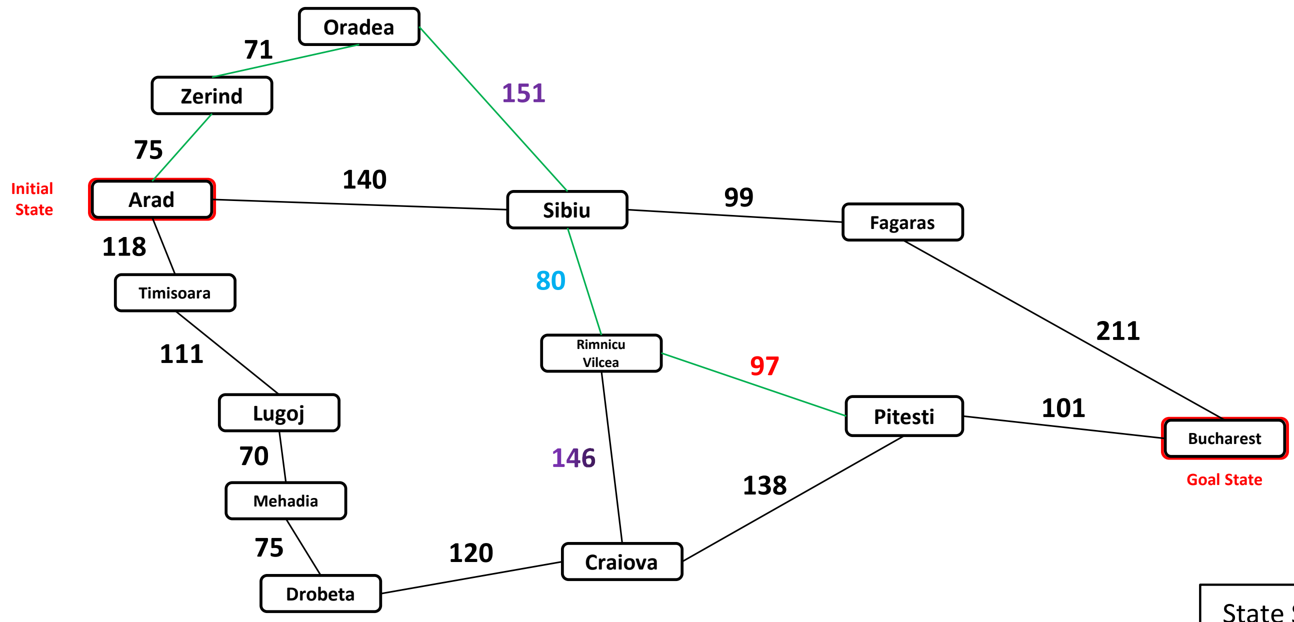
Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or

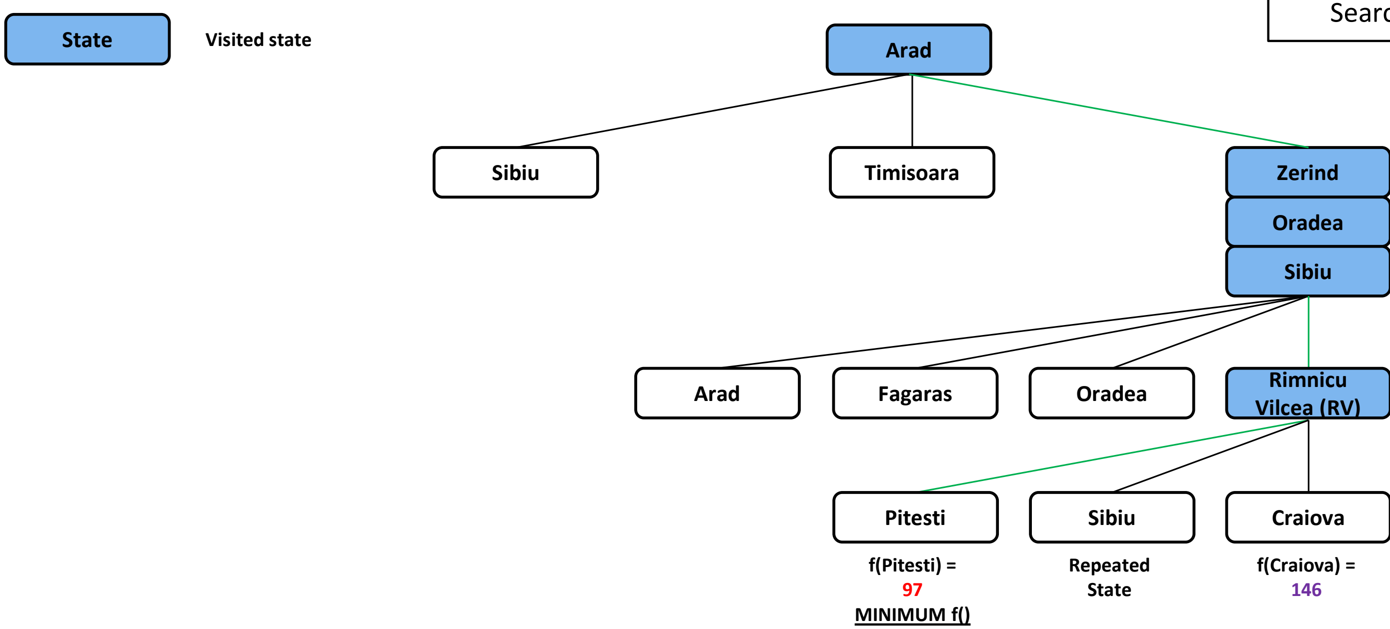
  - **Infinite loop**

Romanian Roadtrip: Greedy Local



State Space Graph

Search Tree



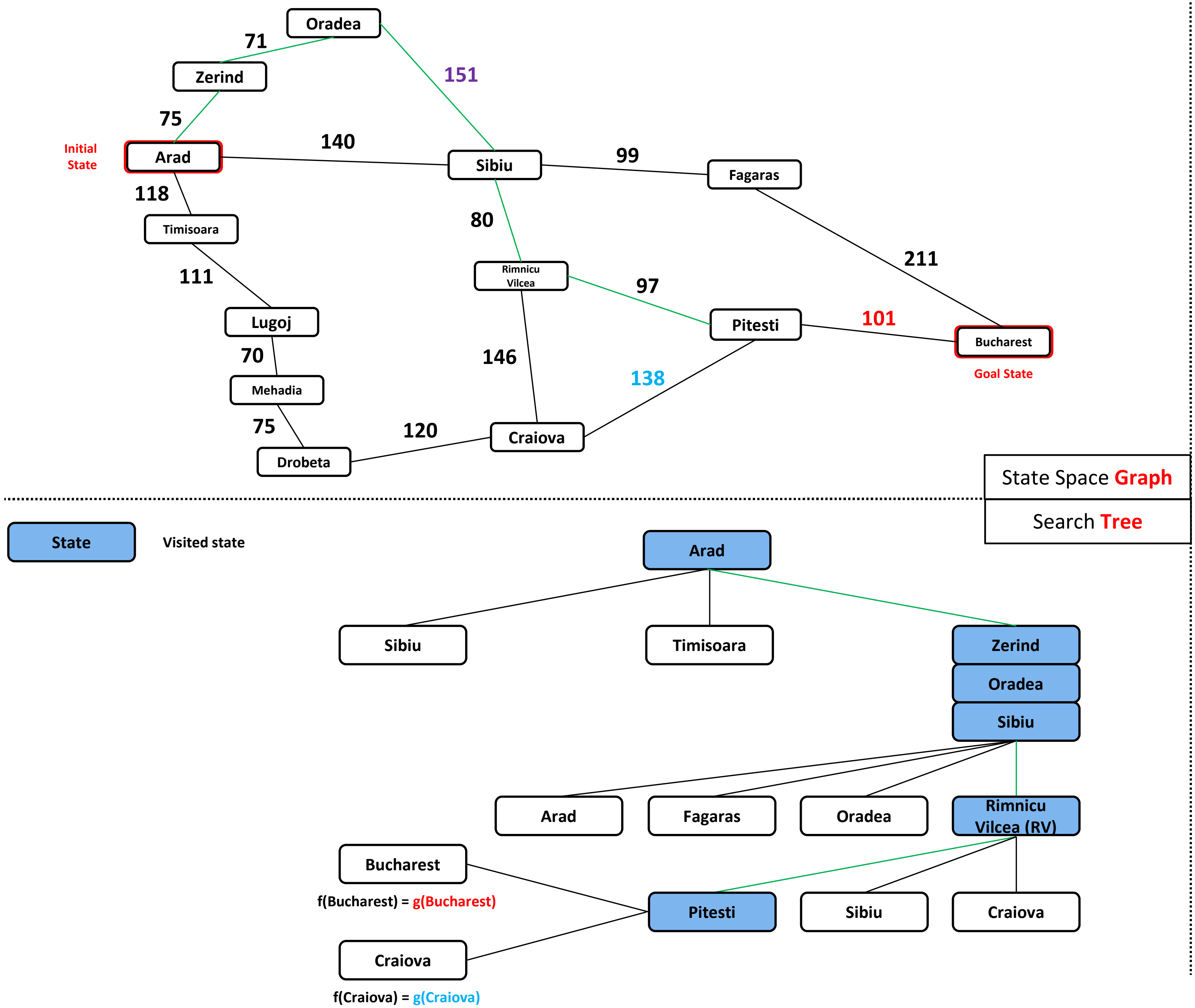
Assumption:
We don't "go" to a repeated state

Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or
  - **Infinite loop**

Romanian Roadtrip: Greedy Local



Assumption:

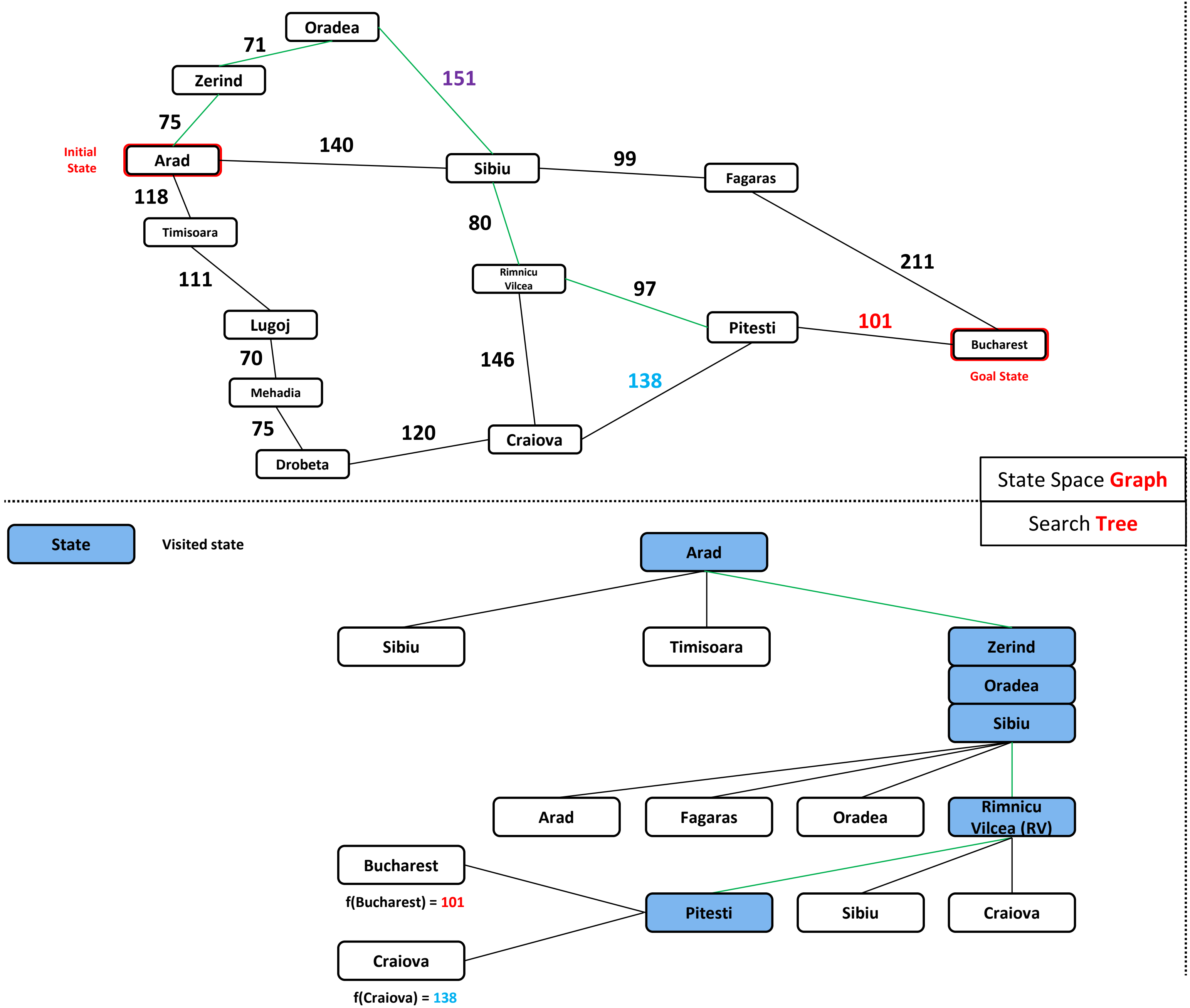
We don't "go" to a repeated state

Alternatively:

- We could go to a repeated state end
  - get stuck there

  or
  - Infinite loop

Romanian Roadtrip: Greedy Local



Assumption:

We don't "go" to a repeated state

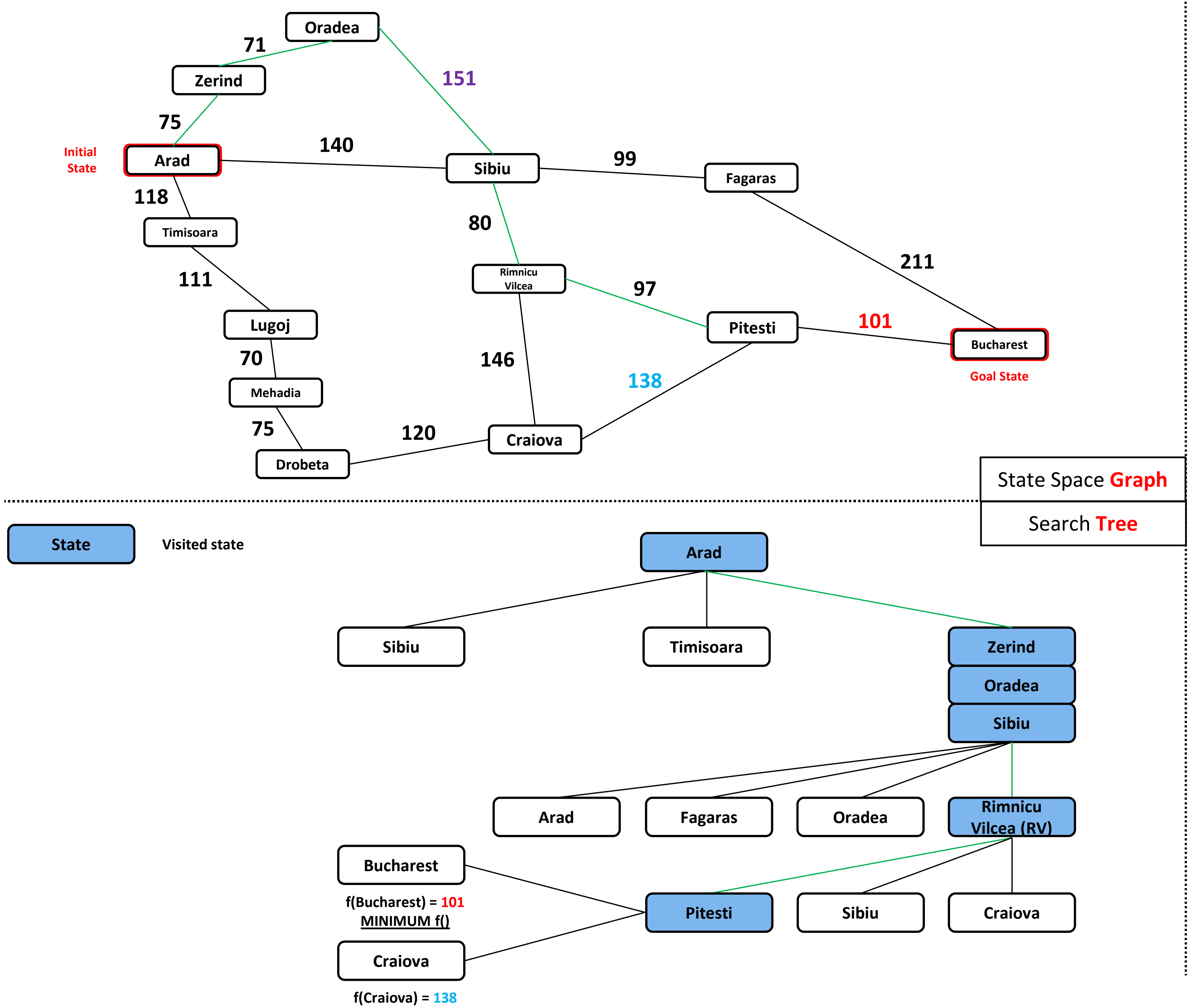
Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or

  - **Infinite loop**

Romanian Roadtrip: Greedy Local

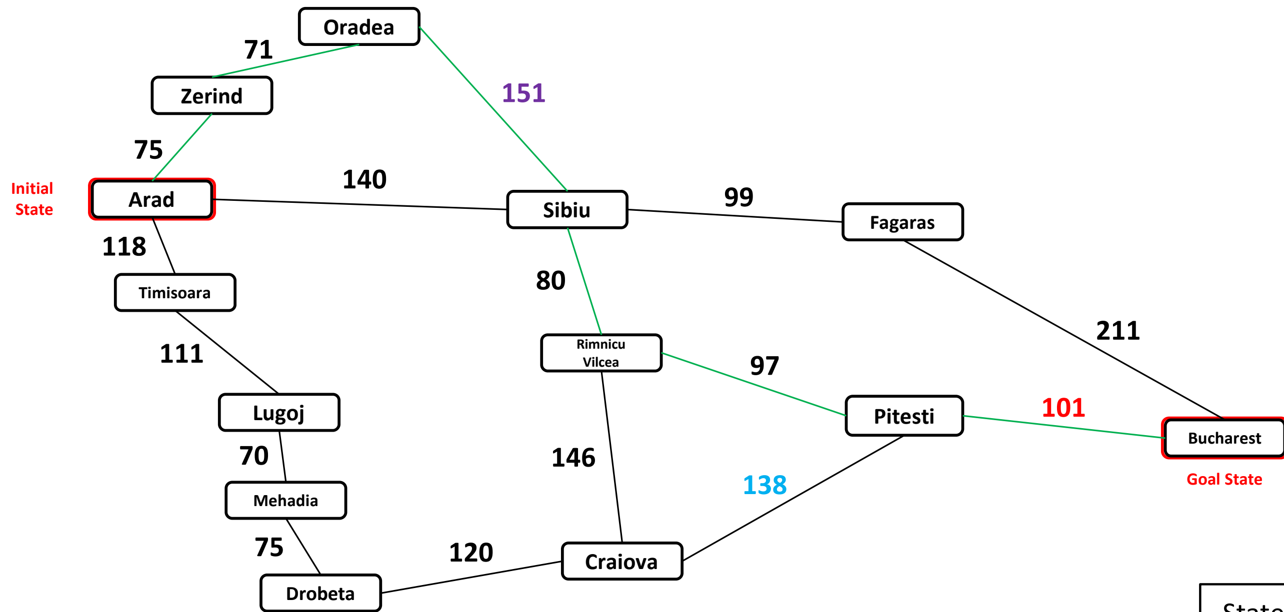


Assumption:
We don't "go" to a repeated state

Alternatively:
- We could go to a repeated state end
  - get stuck there

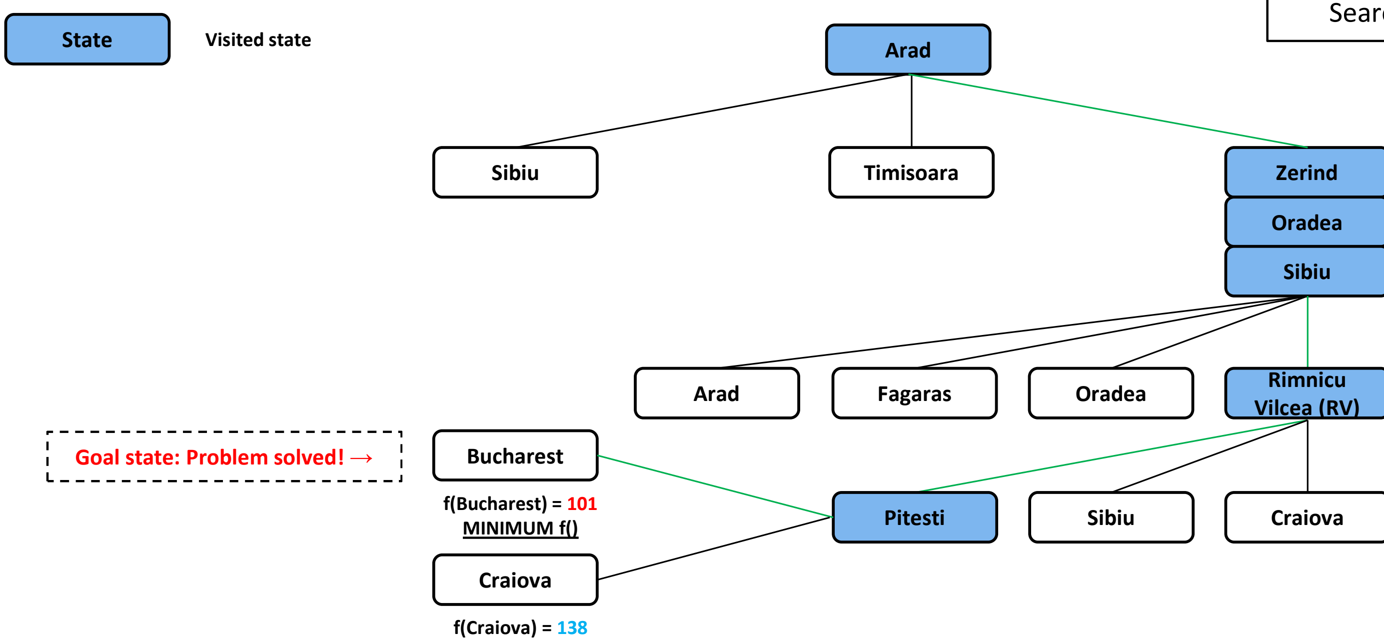
  or
  - Infinite loop

Romanian Roadtrip: Greedy Local



State Space **Graph**

Search **Tree**



Assumption:

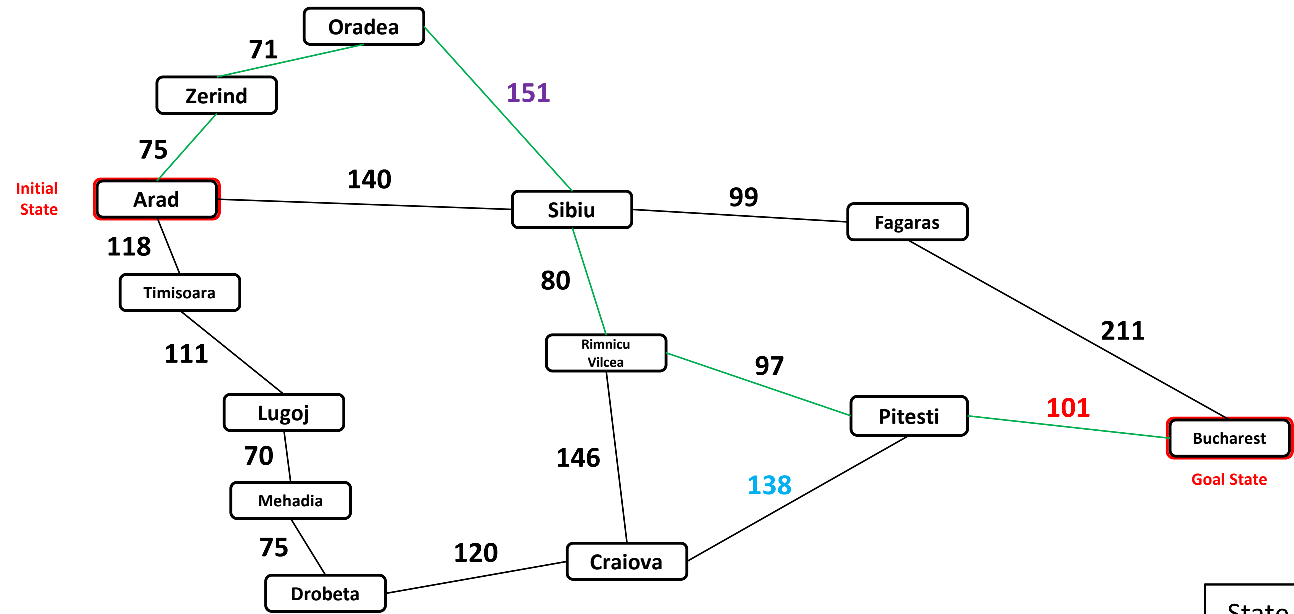
We don't "go" to a repeated state

Alternatively:

- **We could go to a repeated state end**
  - **get stuck there**

  or
  - **Infinite loop**

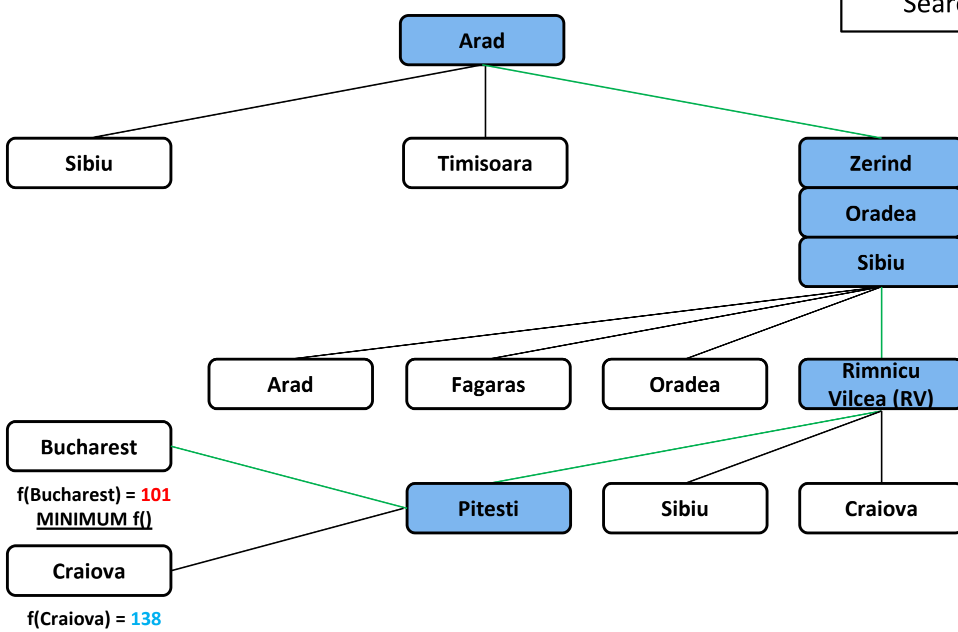
Romanian Roadtrip: Greedy Local



State Space **Graph**

Search **Tree**

State Visited state



Goal state: Problem solved! →

but we were "lucky"
It could have been unreachable using Hill Climbing

Assumption:
We don't "go" to a repeated state

Alternatively:
- We could go to a repeated state end
  - get stuck there

  or
  - Infinite loop

Do we always need to care about the path to the goal?

Informed Search: the Idea

When traversing the search tree use domain knowledge / heuristics to avoid search paths (moves/actions) that are likely to be fruitless

Informed Search and Heuristics

Informed search relies on domain-specific knowledge / hints that help locate the goal state.

$$h(n) = h(\text{State } n)$$

$$h(n) = n(\text{relevant information about State } n)$$

h(n): heuristic function - estimated cost of … what exactly?

Evaluation function

Calculate / obtain:

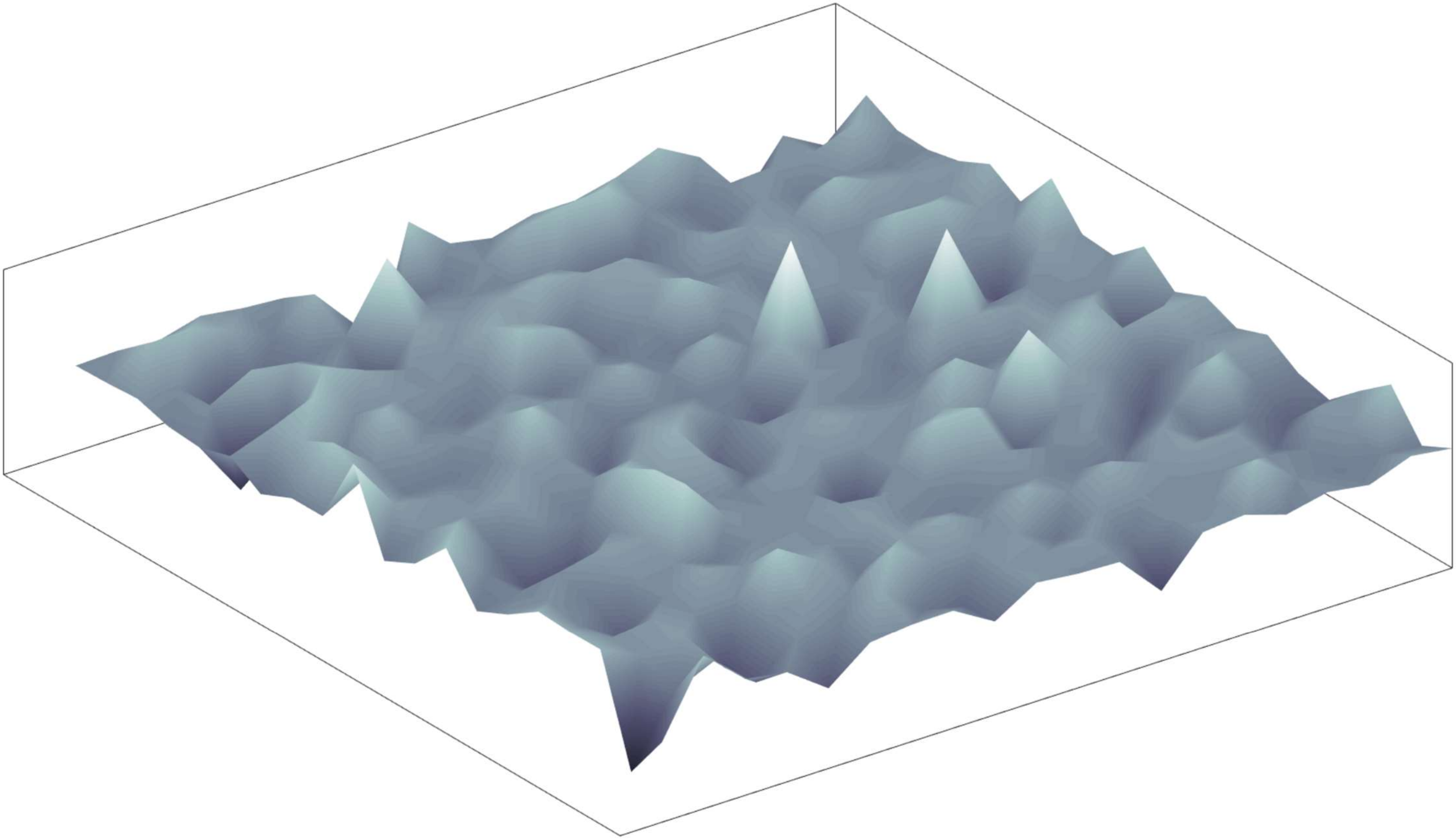
f(n) = f(State n)

f(n) = f(relevant information about State n)

A state n with minimum (or maximum) f(n) should be chosen for … what exactly?

Search In Complex Environments

Difficult Environment / State Space

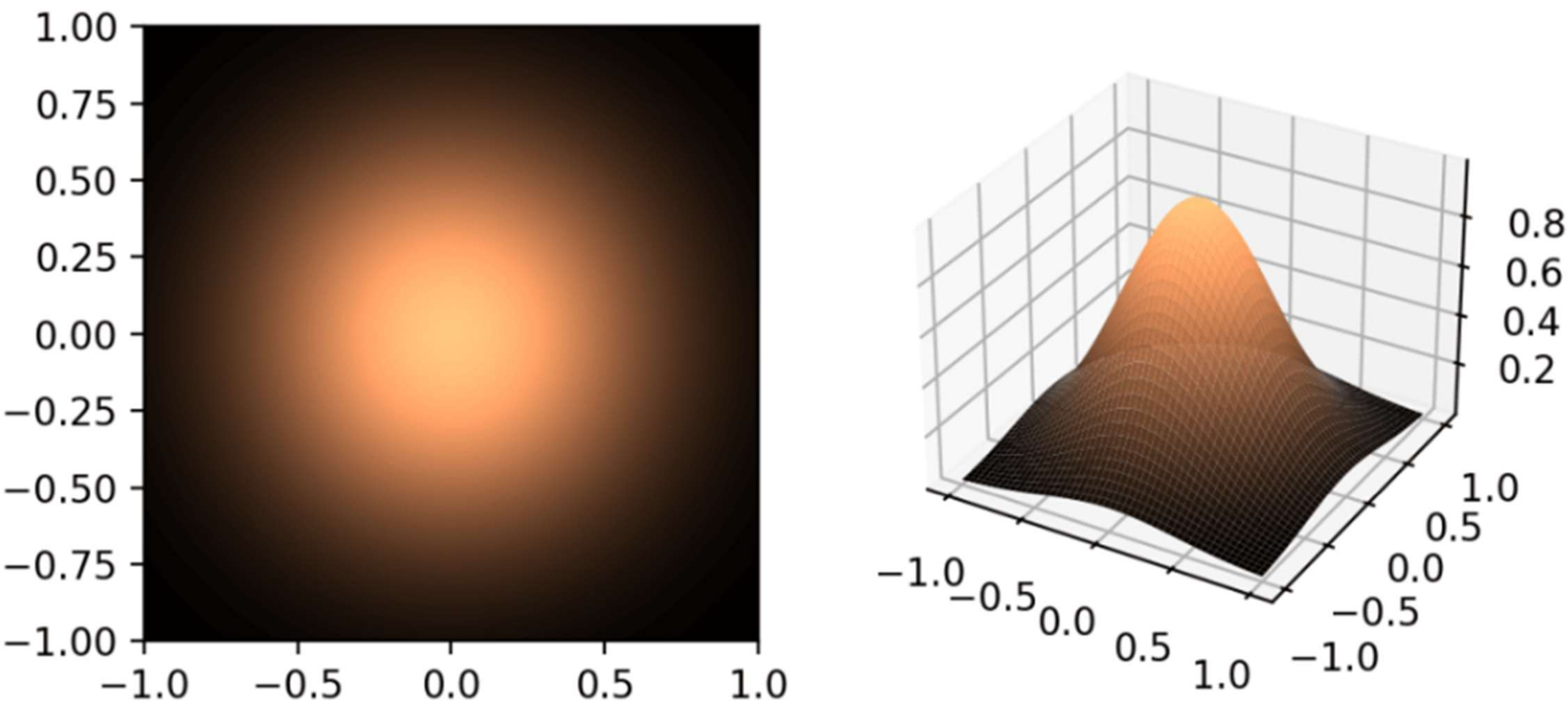


Reality: Environment Assumptions

Reality is not simple. What to relax?

- Fully observable?
- Single agent?
- Deterministic?
- Static?
- Episodic or sequential?
- Discrete?
- Known to the agent?

Discrete vs. Continuous Spaces



Hard Problems

- **Many important problems are provably not solvable in polynomial time (NP and harder)**
- **Results based on the worst-case analysis**

- **In practice: instances are often easier**
- **Approximate methods can often obtain good solutions**

Local Search:

When we can't/don't care about the path to the goal (that much).

We just want to reach the goal.

Local Search

- Moves between configurations by performing local moves
- Works with complete assignments of the variables
- Optimization problems:
  - Start from a suboptimal configuration
  - Move towards better solutions
- Satisfaction problems:
  - Start from an infeasible configuration
  - Move towards feasibility
- No guarantees
- Can work great in practice!

Local Search Algorithms

If the path to the goal does not matter, we might consider a different class of algorithms.

Local Search Algorithms

- do not worry about paths at all.
- Local search algorithms operate:
  - using a single current state (rather than multiple paths) and generally move only to neighbors of that state.
  - typically, the paths followed by the search are not retained

Selecting Neighbor

- How to select the neighbor?
  - exploring the whole or part of the neighborhood
- Best neighbor
  - select "the" best neighbor in the neighborhood
- First neighbor
  - select the first "legal" neighbor
  - avoid scanning the entire neighborhood
- Multi-stage selection
  - select one "part" of neighborhood and then
  - select from the remaining "part" of neighborhood

Local Search Algorithms

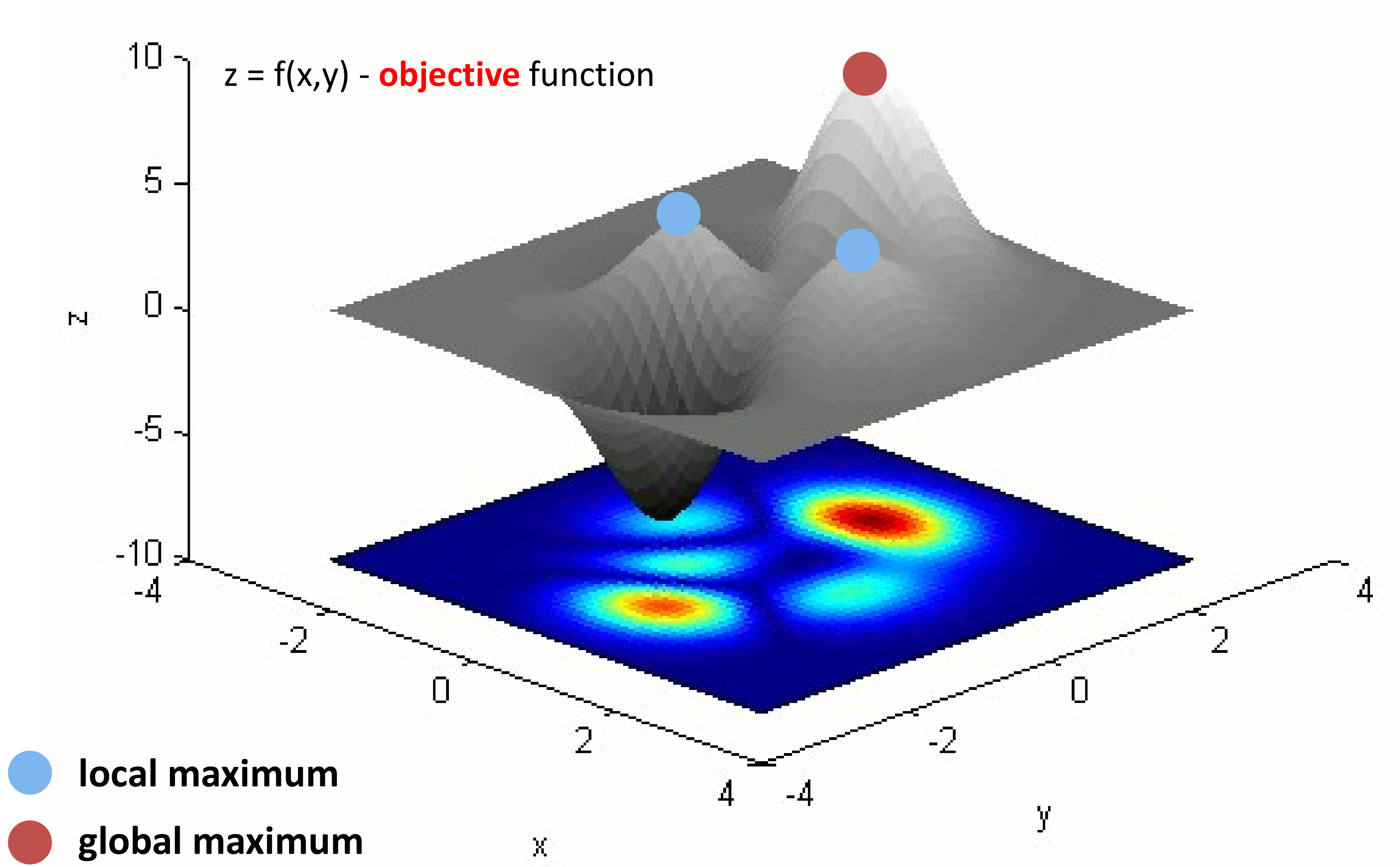
- **Hill-climbing search**
  - **Gradient descent in continuous state spaces**
  - **Can use e.g. Newton's method to find roots**
- **Simulated annealing search**
- **Tabu search**
- **Local beam search**
- **Evolutionary/genetic algorithms**

Local Search Algorithms

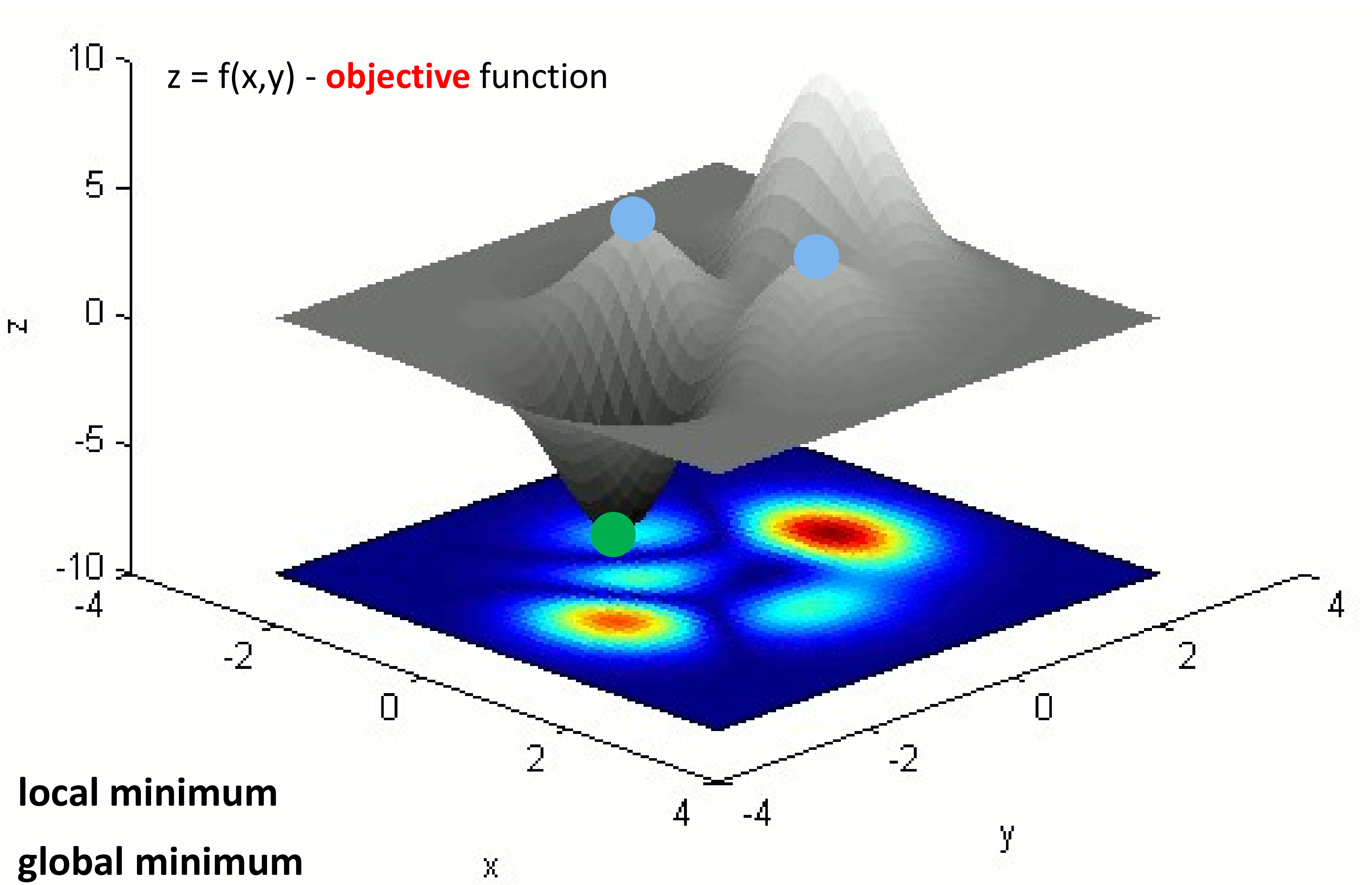
Although local search algorithms are **not systematic**, they have two key advantages:

- they **use very little memory**—usually a constant amount; and
- they can often **find reasonable solutions in large or infinite (continuous) state spaces** for which systematic algorithms are unsuitable.

Local and Global Maxima



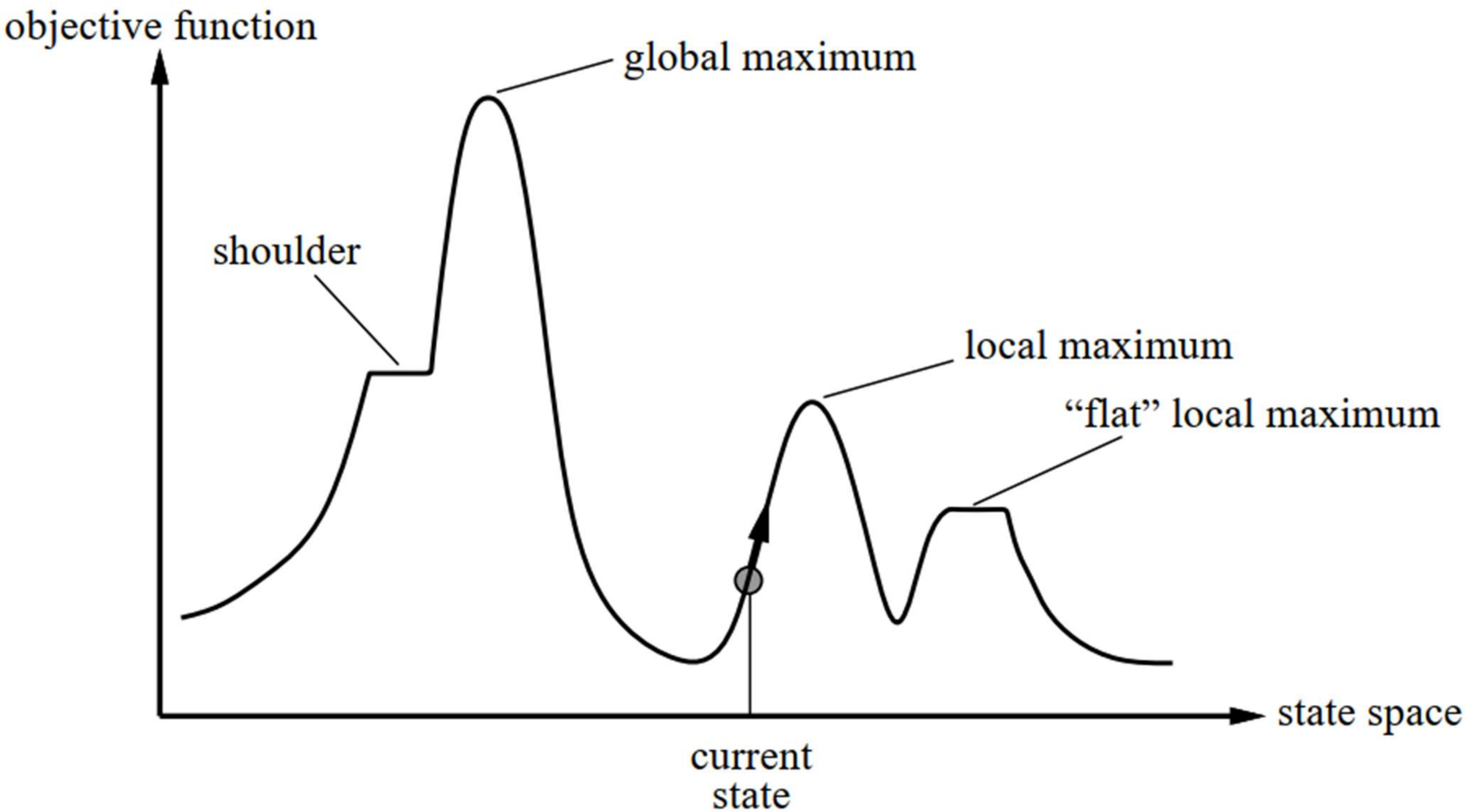
Local and Global Minima



local minimum

global minimum

1D State Space Landscape



Local search algorithms are useful for solving pure <u>optimization</u> problems, in which the aim is to find the best state according to an objective function.

Evaluation / Objective Function

Calculate / obtain:

f(n) = f(State n)

f(n) = f(relevant information about State n)

A state n with minimum (or maximum) f(n) should be chosen

Evaluation / Objective Function

Evaluation function f(n) provides information on the "cost" of getting from node n to the goal state to help decide where to go next

Three ways to think about f() so far:

- **f() - the value of the state (its "goodness" or "fitness")**
- **f() - the estimated cost of getting to the goal from the current state:**

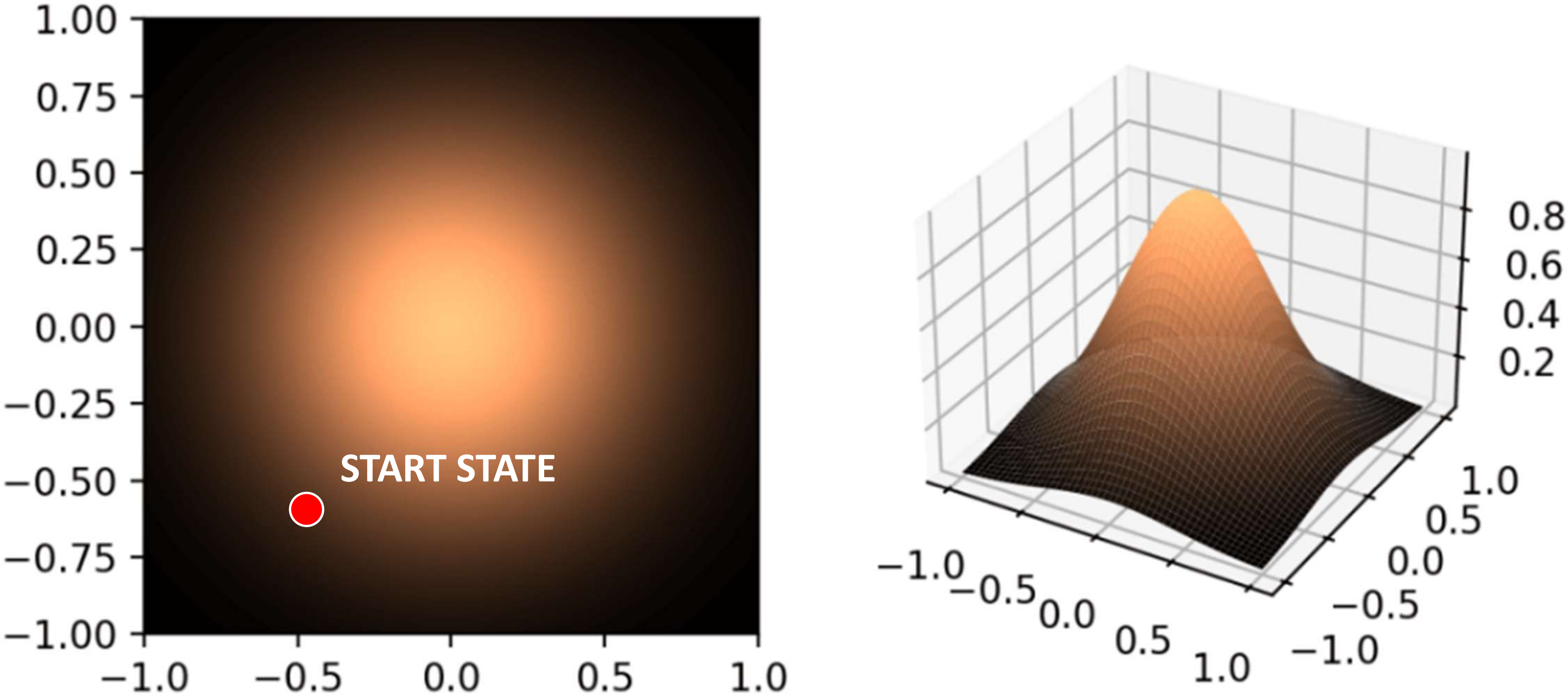
  $f(n) = h(n)$ where $h(n)$ – heuristic

- **f() - the estimated cost of getting to the goal state from the current state and the cost of the existing path to it.**

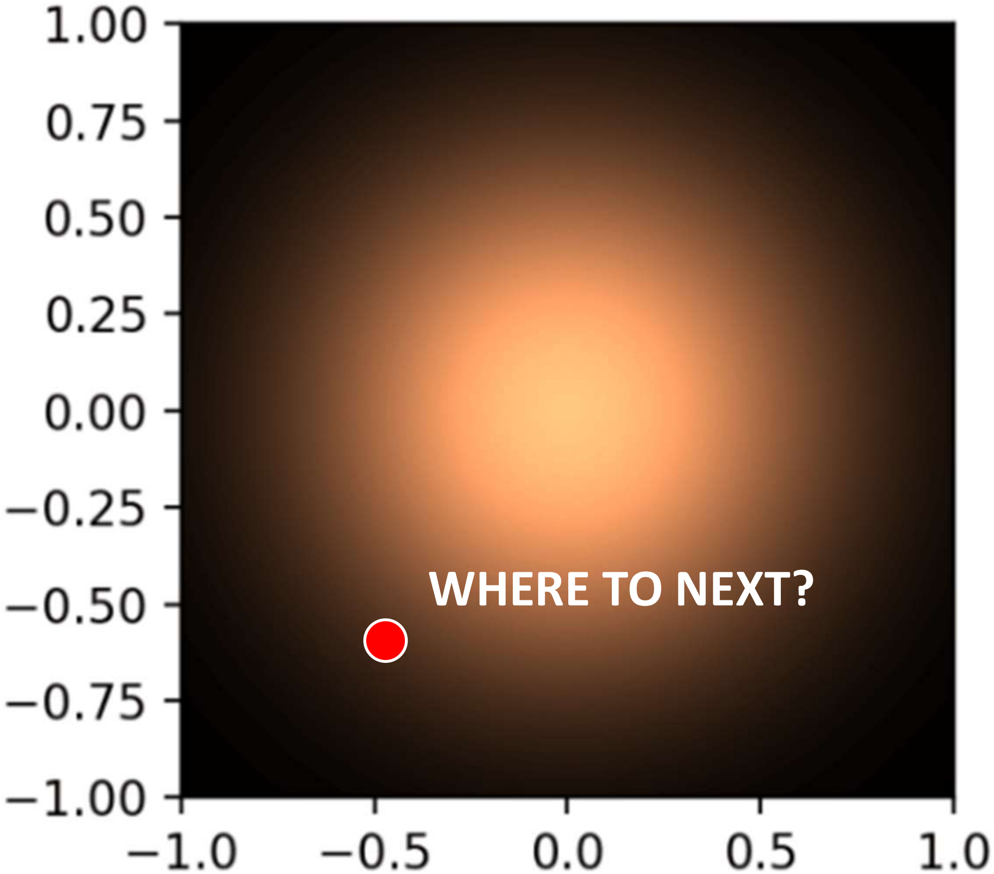
  $f(n) = g(n) + h(n)$

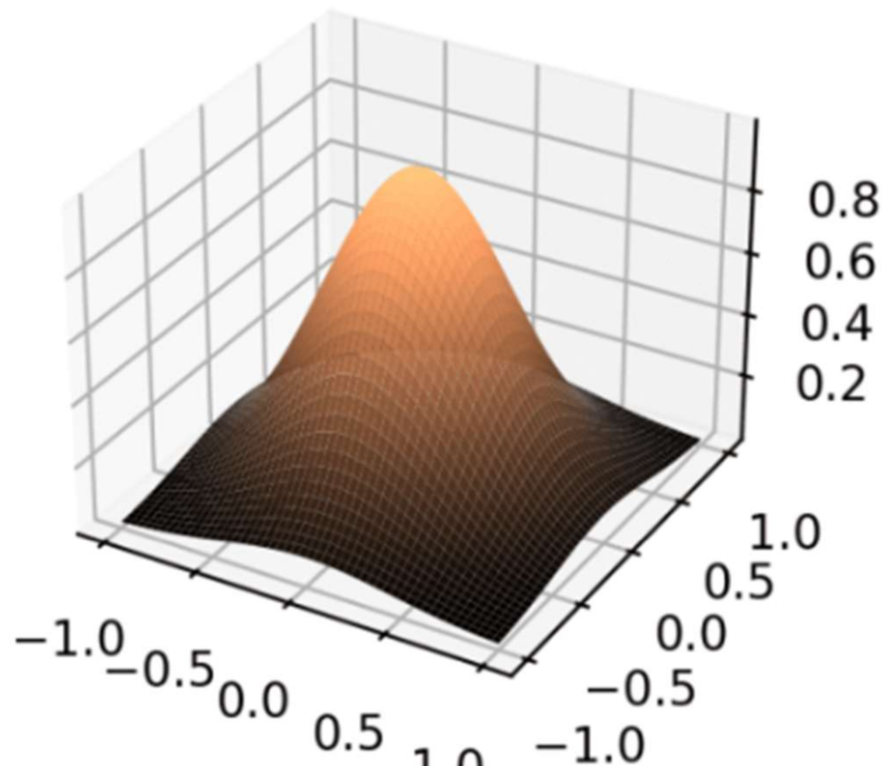
  where: $g(n)$ - the cost to get to n (from initial state), $h(n)$ - heuristic

Local Search: Start "Somewhere"

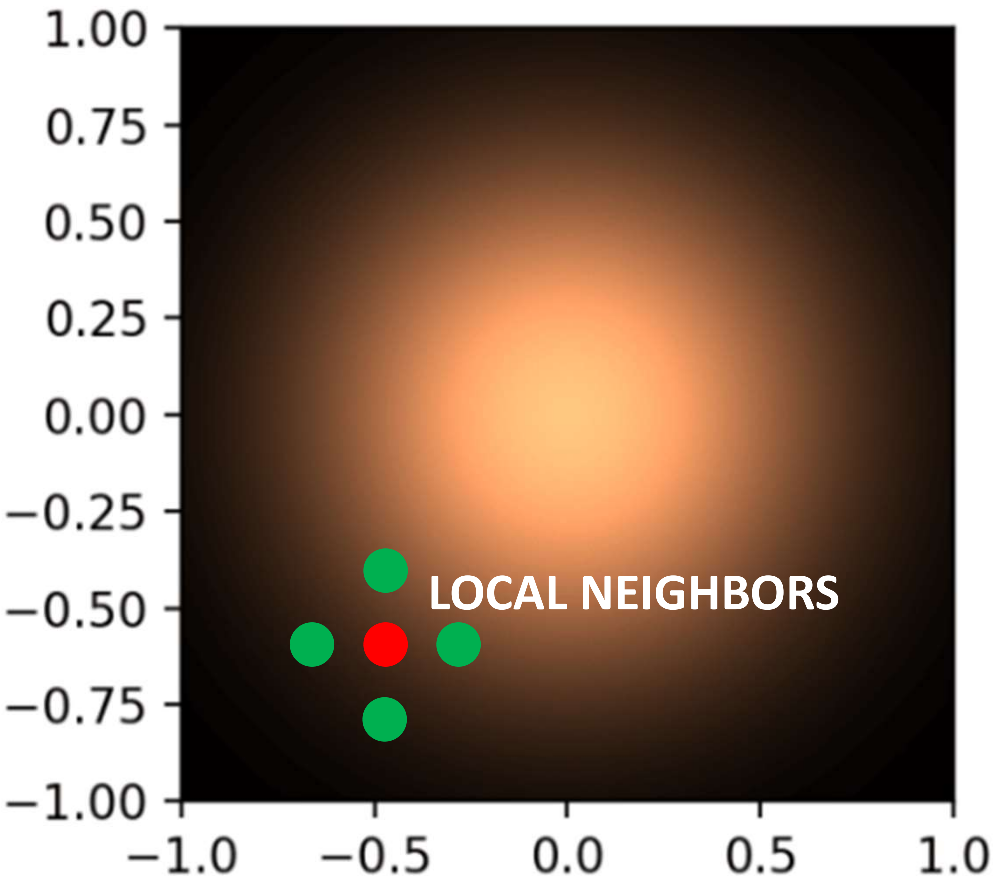


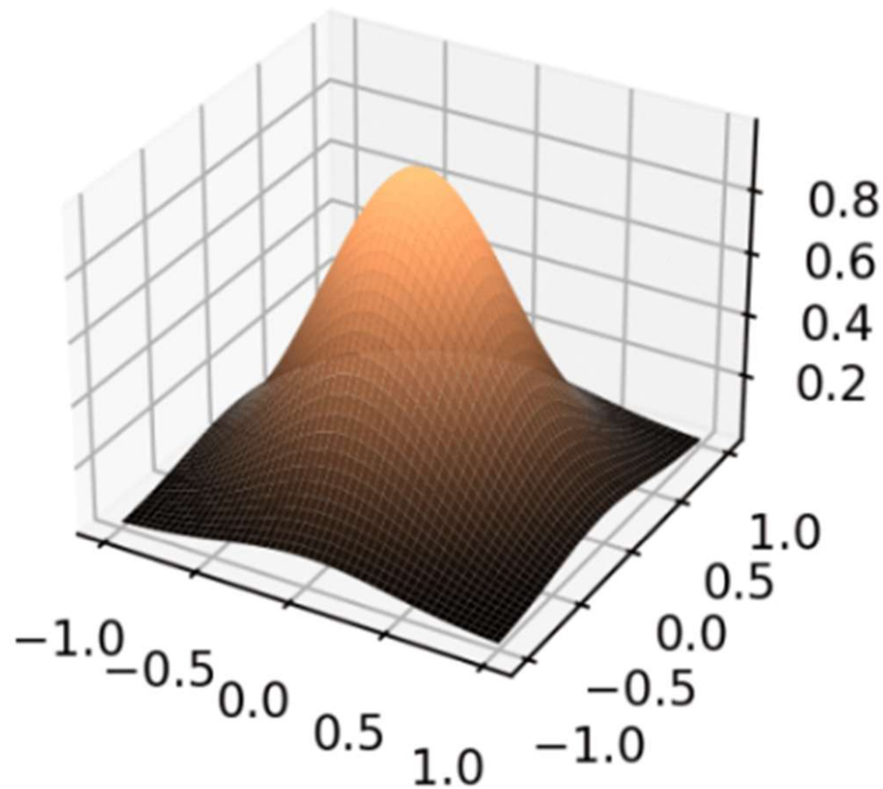
Local Search: Start "Somewhere"



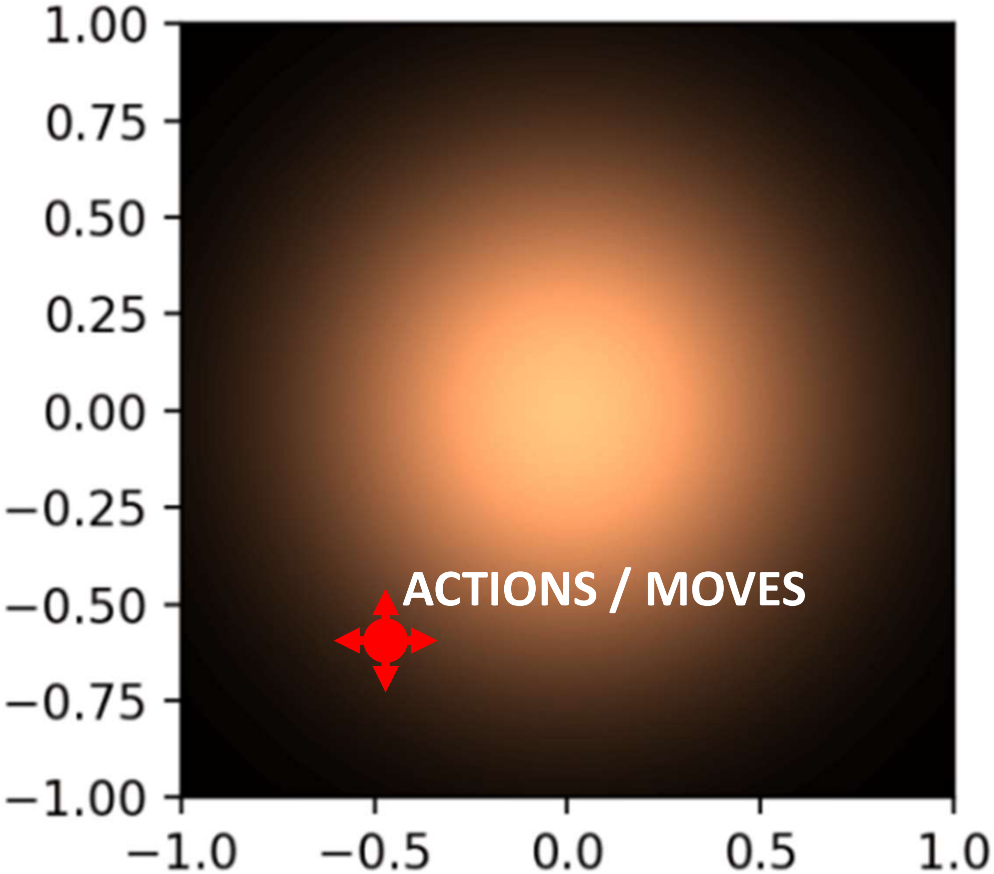


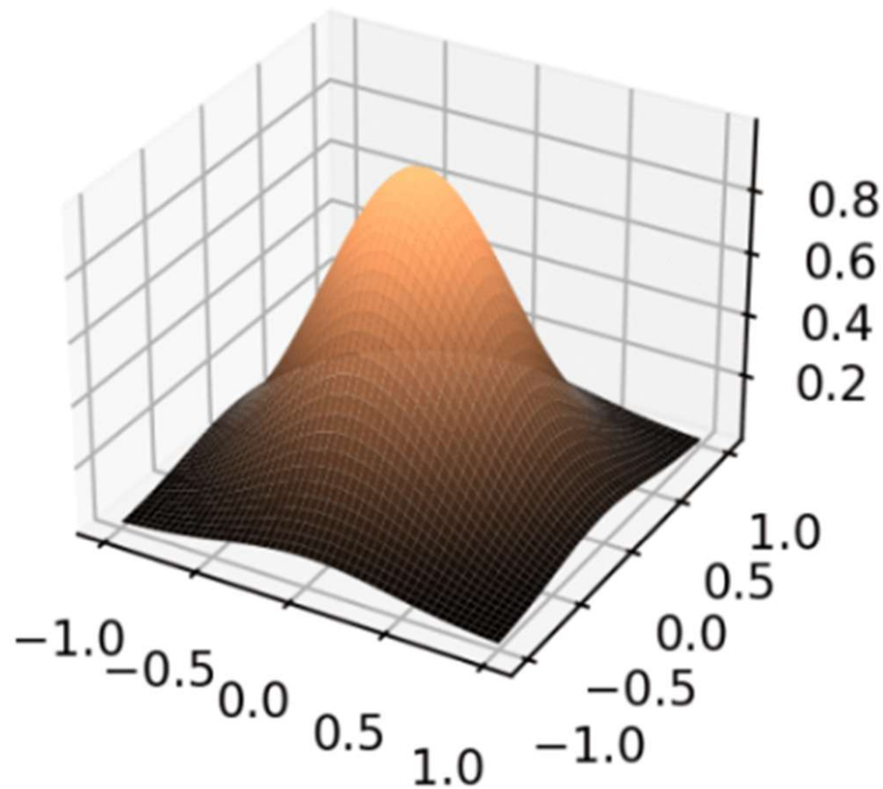
Local Search: Neighbors





Traverse/Explore Space With “Actions”

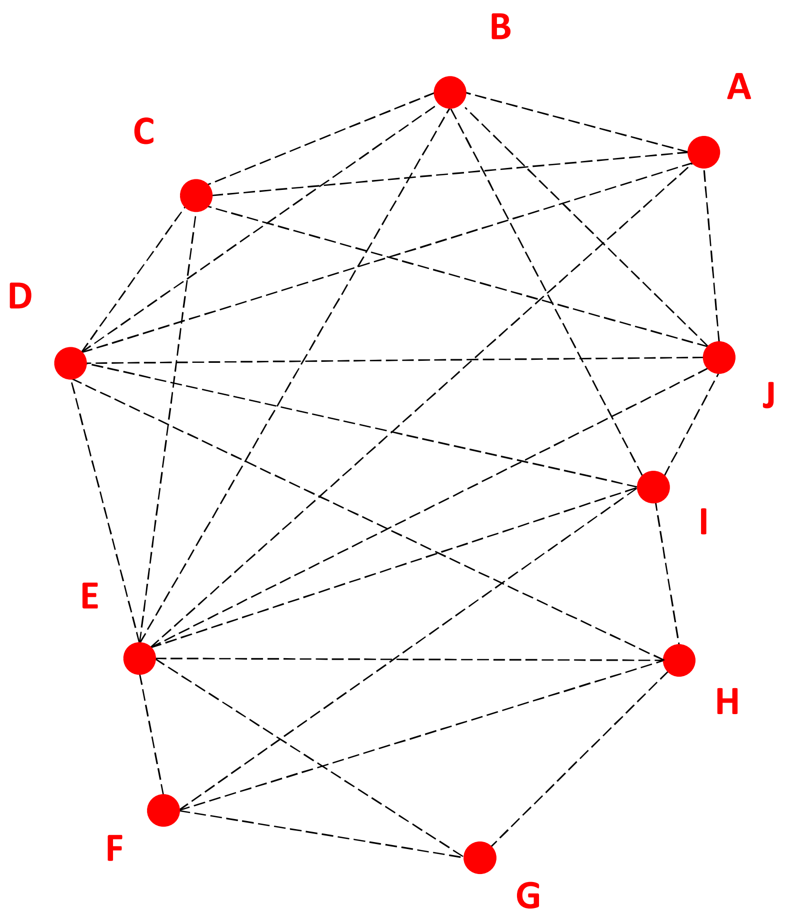




When we can’t/don’t care about the path to the goal (that much)

Selected Problems

Travelling Salesman Problem



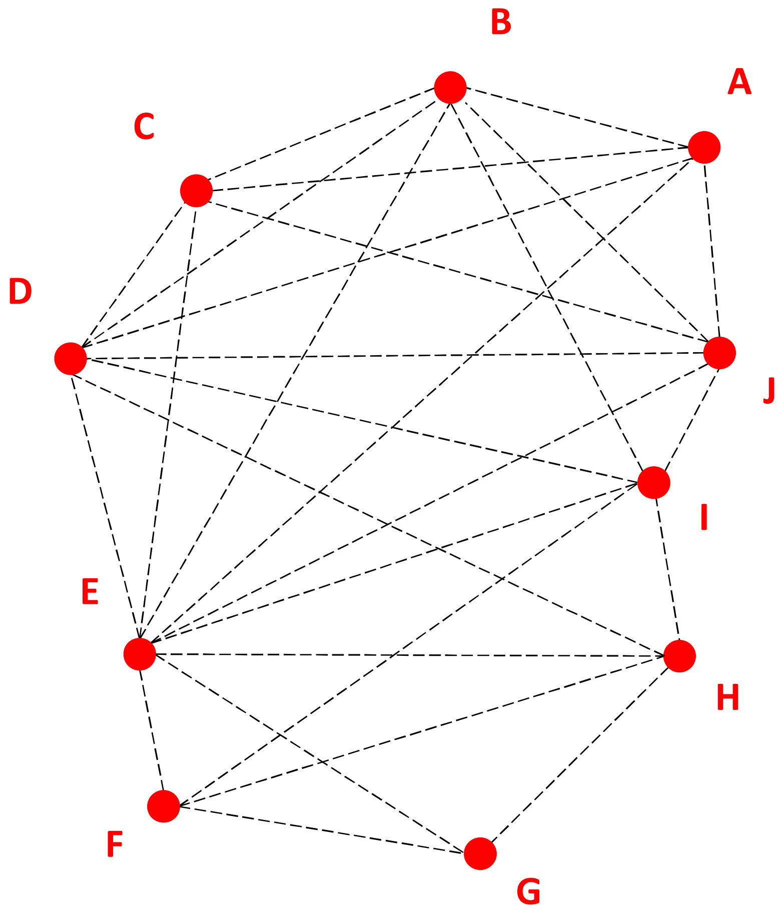
Problem:

A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once.

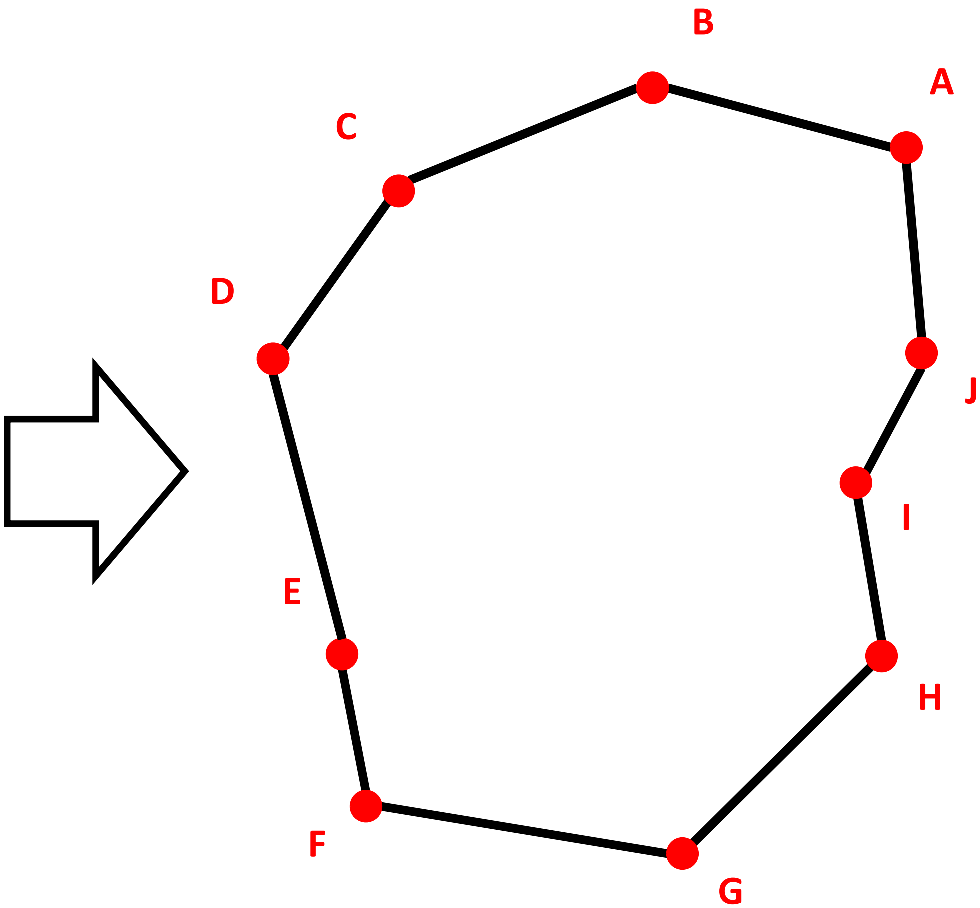
Solution:

Shortest possible path/route such that he visits each city exactly once and returns to the origin city.

Travelling Salesman Problem

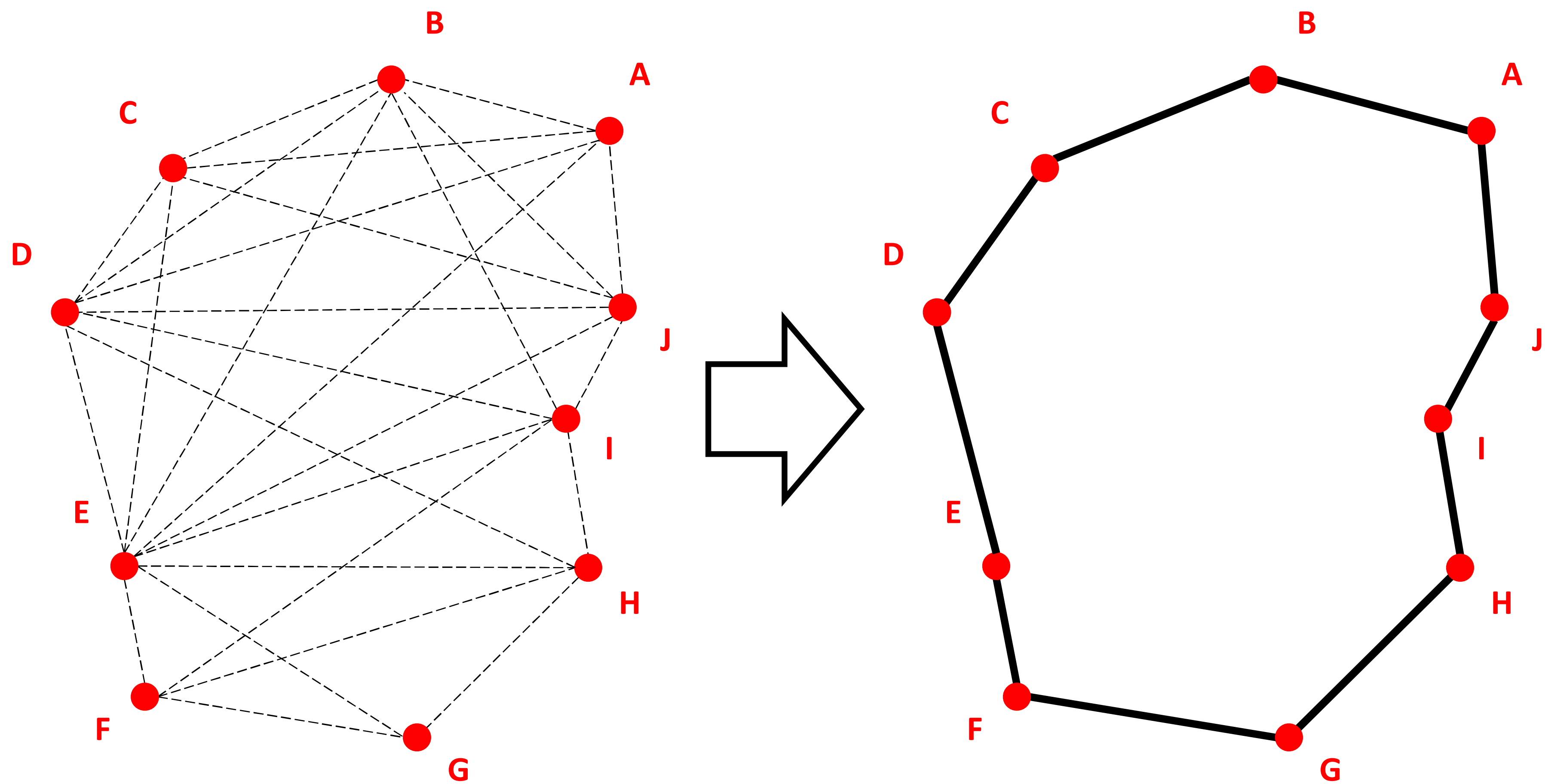


PROBLEM



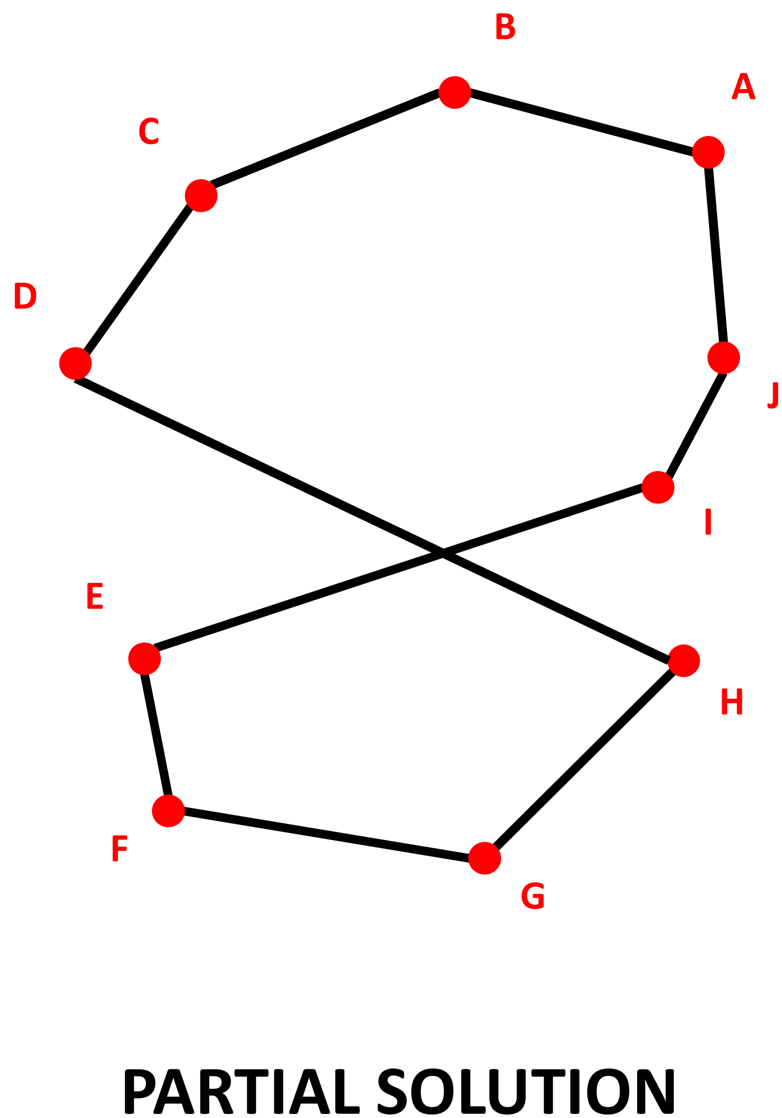
SOLUTION

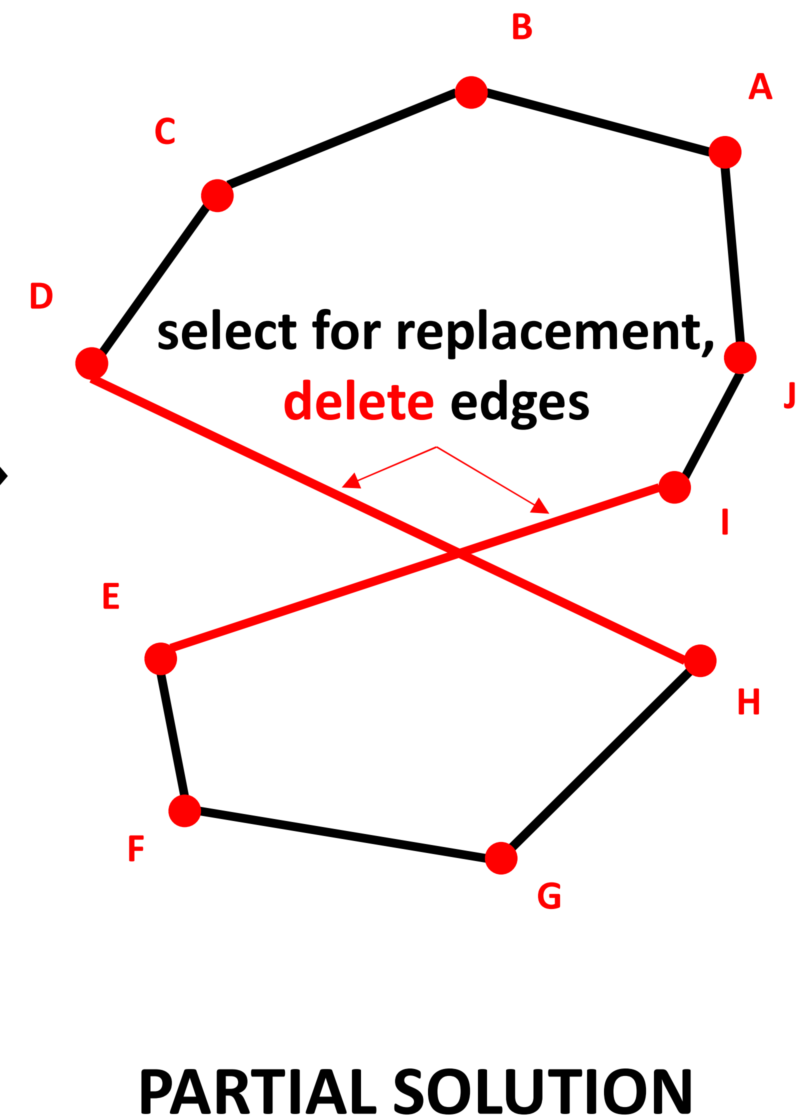
Travelling Salesman Problem



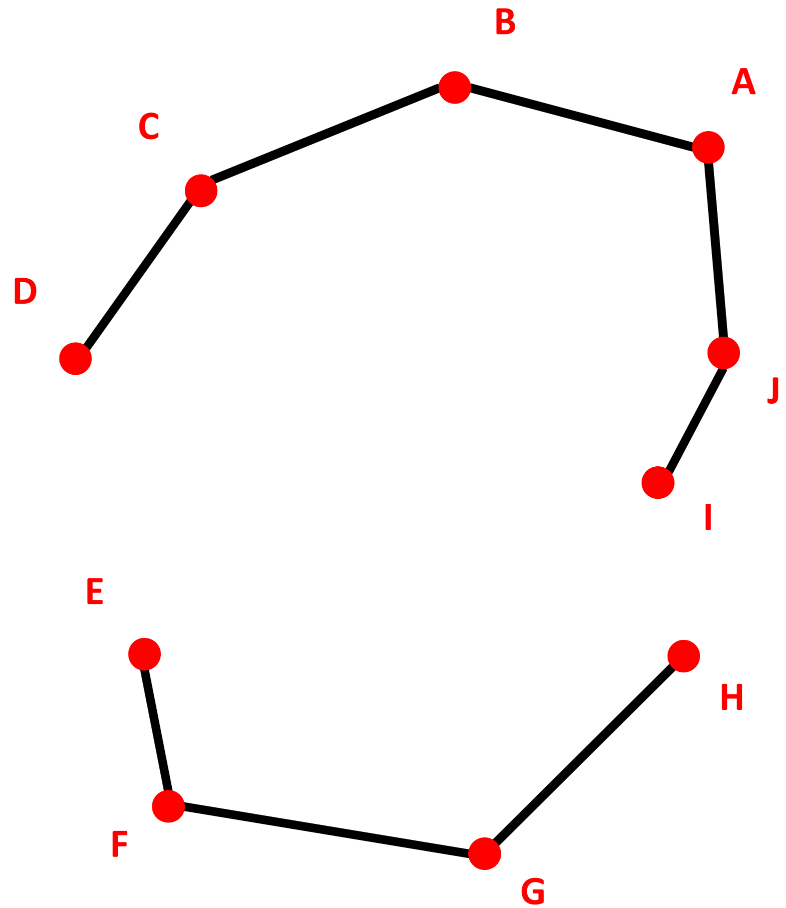
N cities → (N-1)!/2 paths | 15 cities → 43589145600 paths

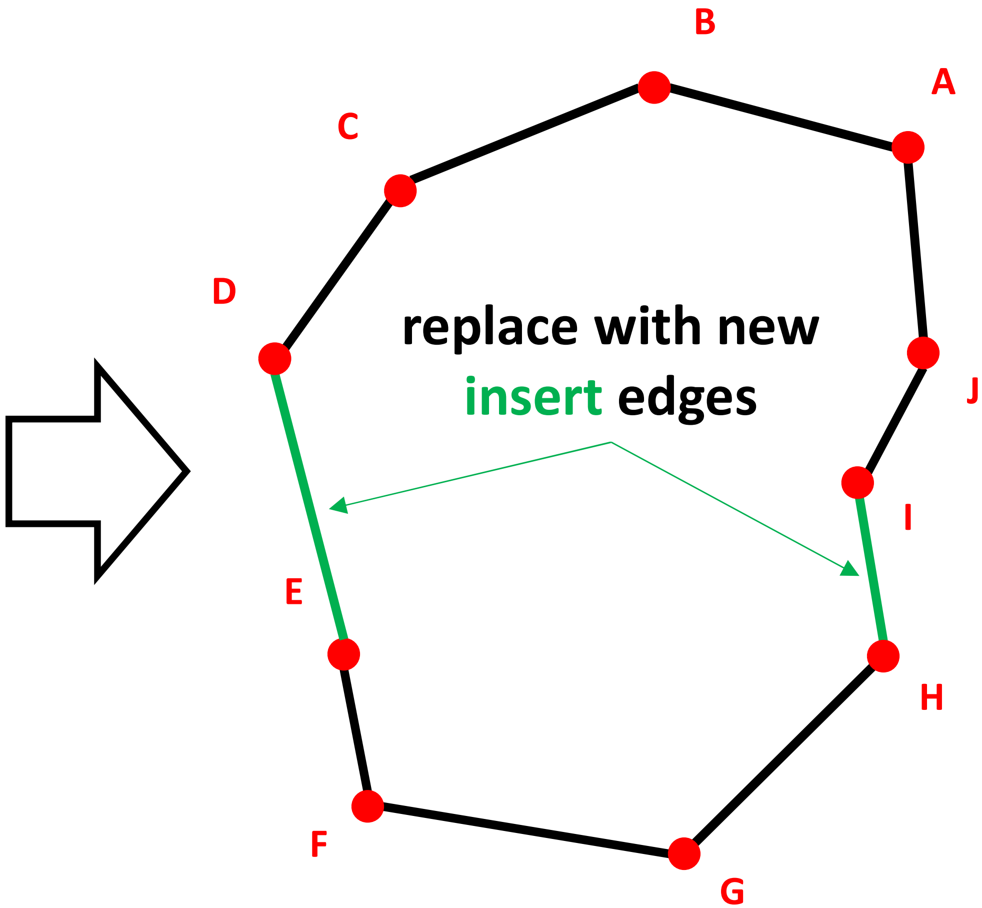
Travelling Salesman Problem



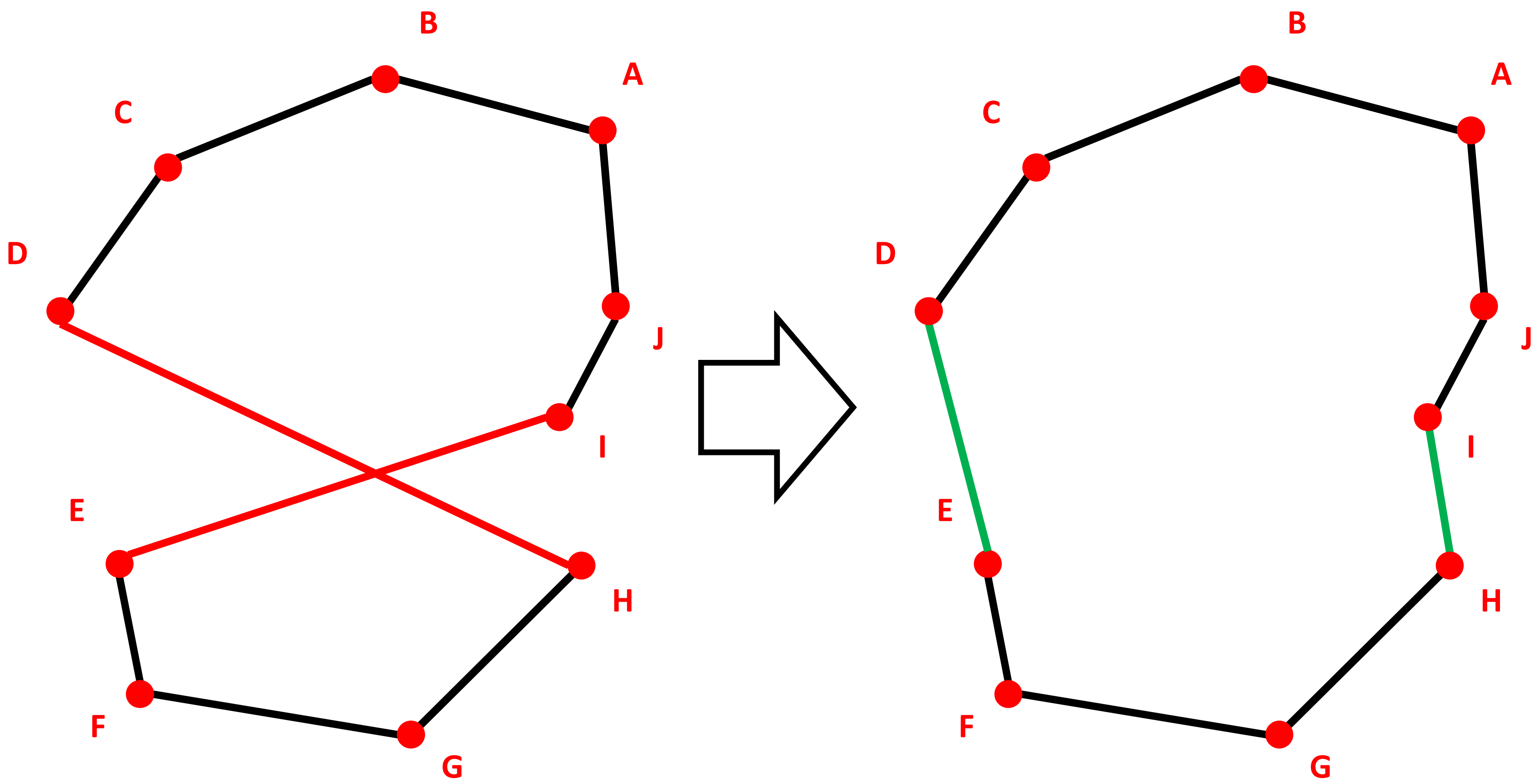


Travelling Salesman Problem



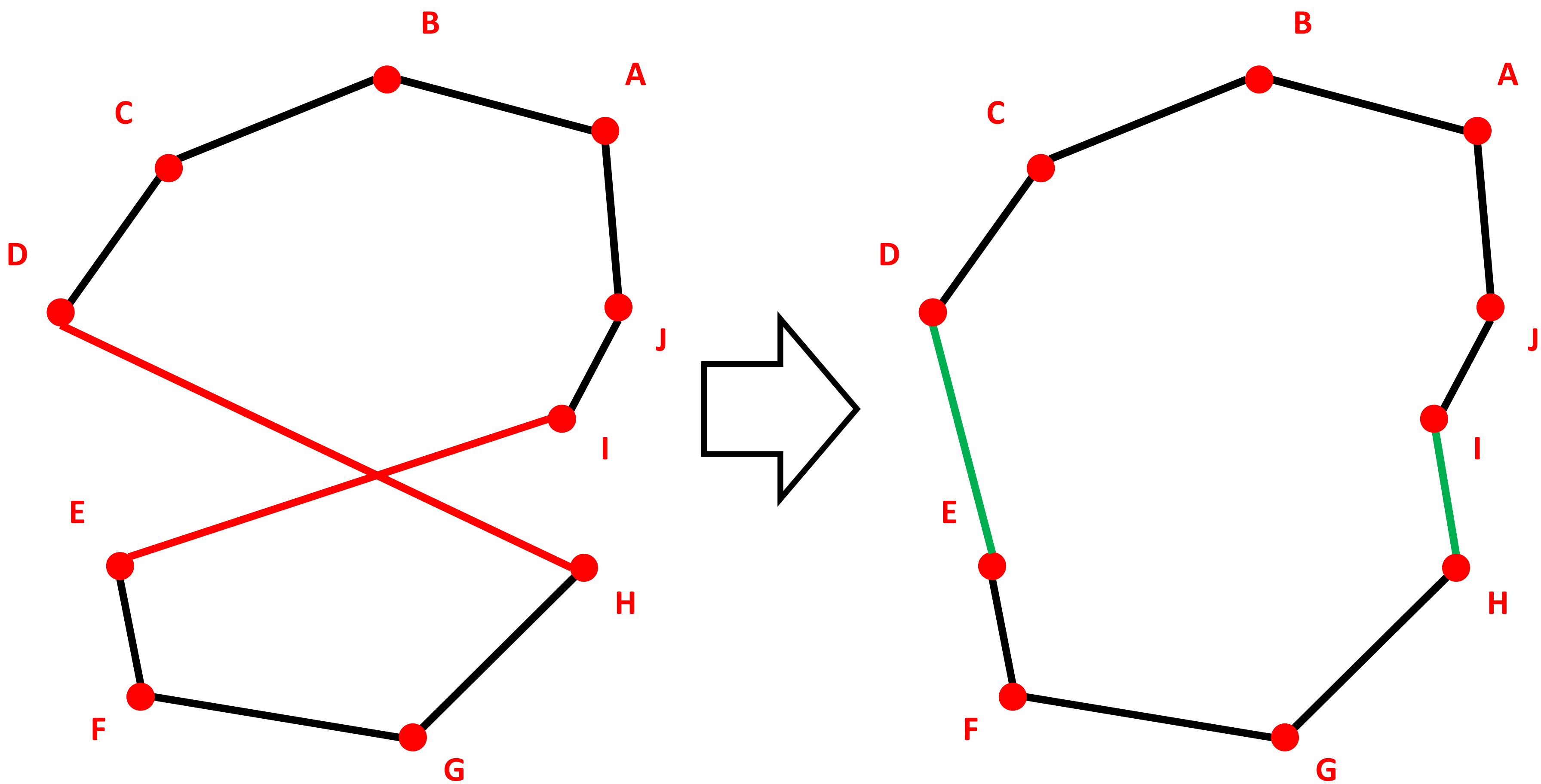


Travelling Salesman Problem



State i **State j: State i's neighbor**

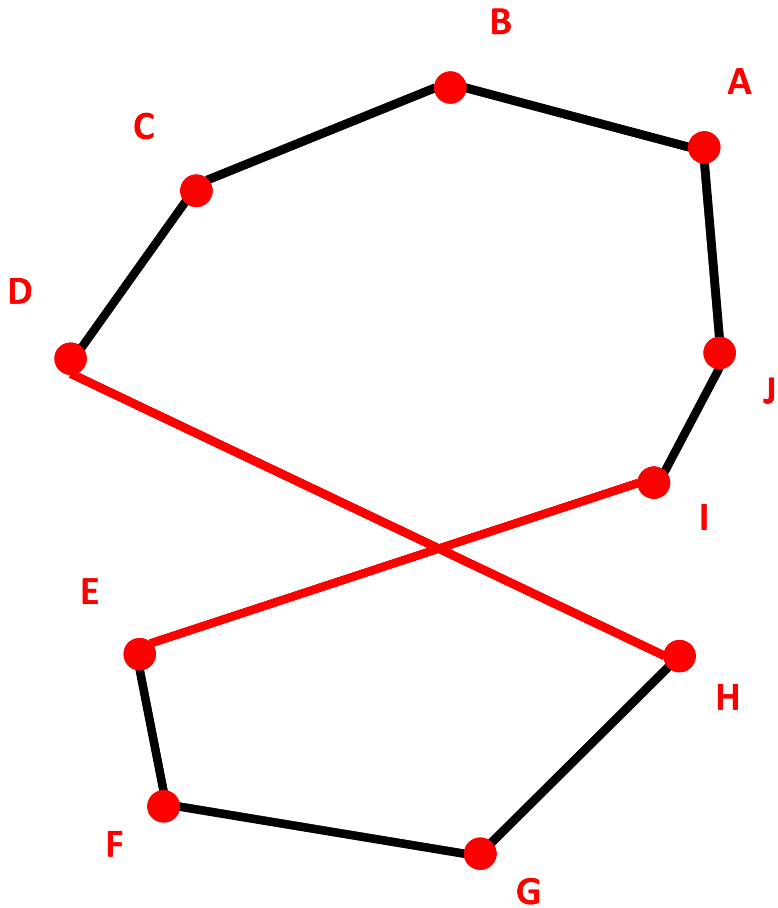
Travelling Salesman Problem

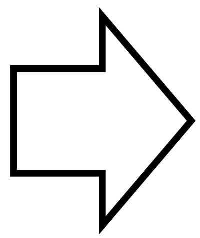


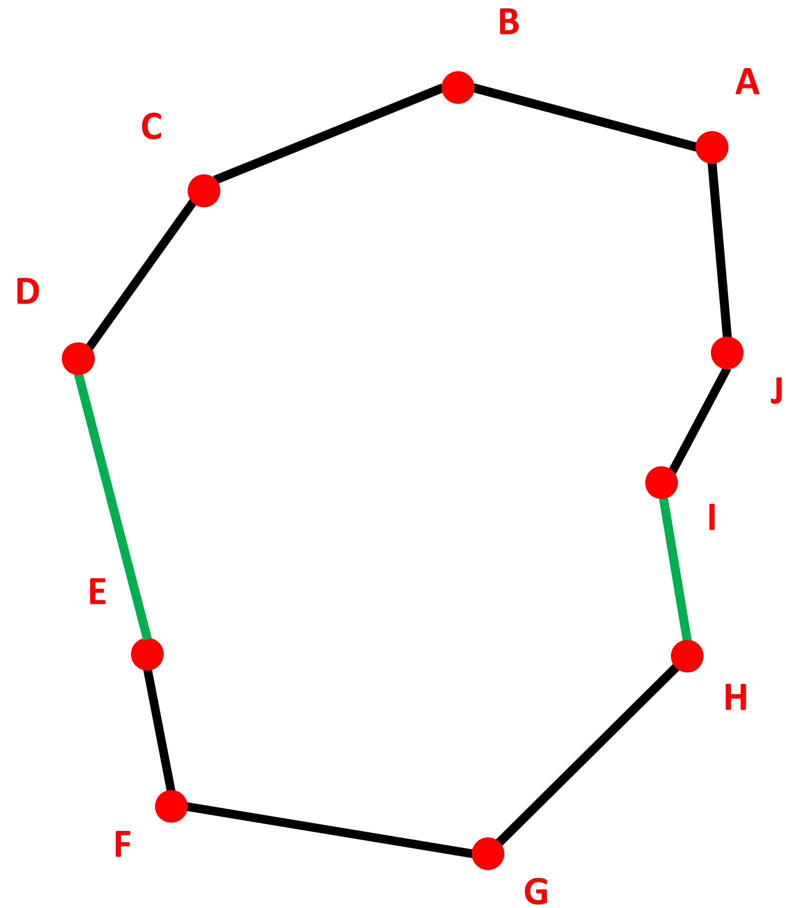
Neighbor: A state one "2-edge" (or N-edge) swap away

Travelling Salesman Problem

$f(State_i) > f(State_j)$



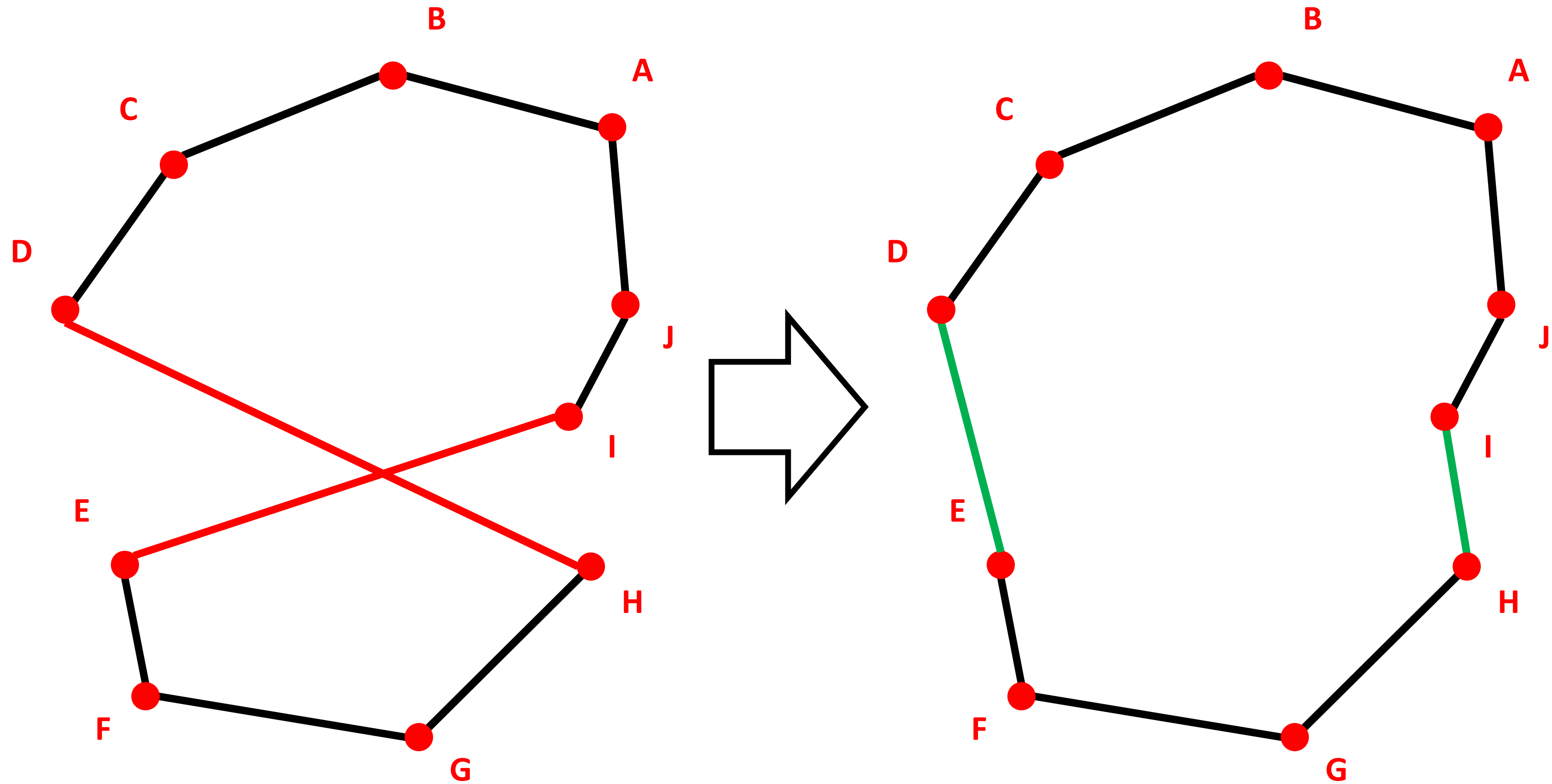




LOCAL MINIMUM **GLOBAL MINIMUM**

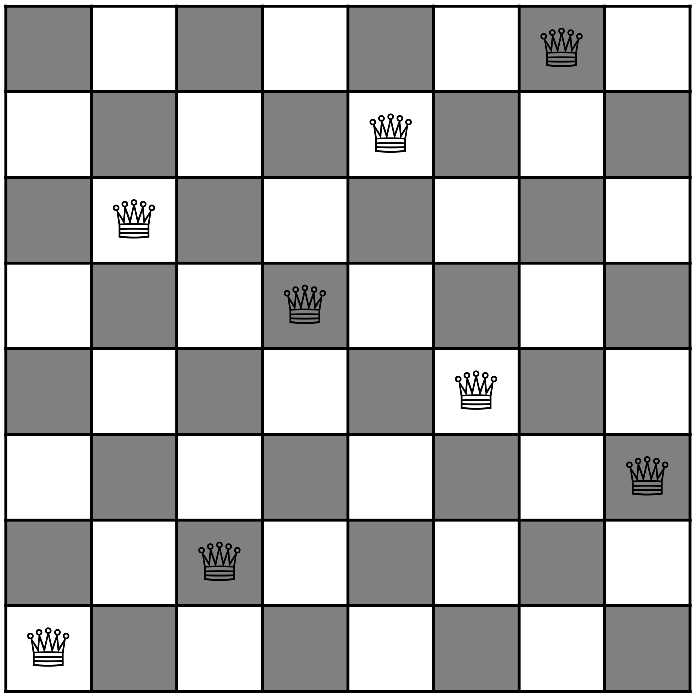
Travelling Salesman Problem:

$f(\text{State}_i) = \text{path cost} \quad > \quad f(\text{State}_j) = \text{path cost}$



LOCAL MINIMUM **GLOBAL MINIMUM**

N-Queen Problem



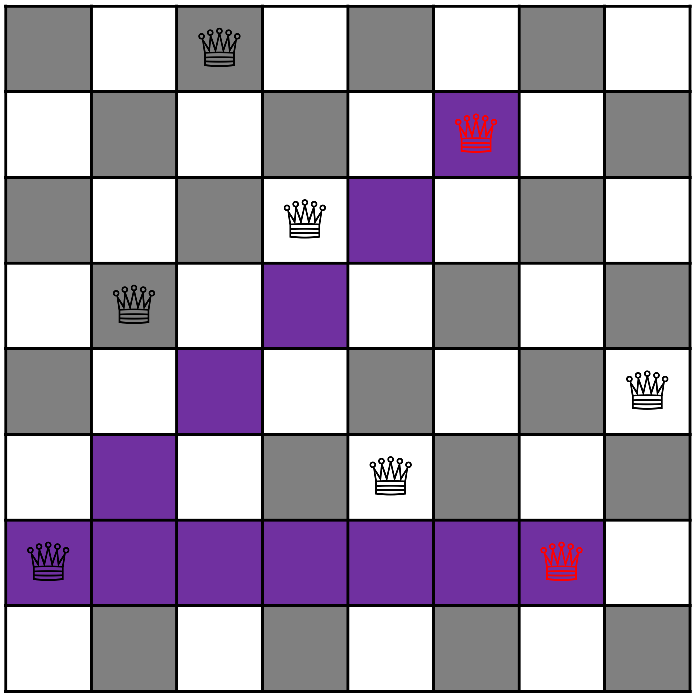
Problem:

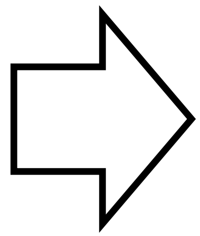
The N-Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.

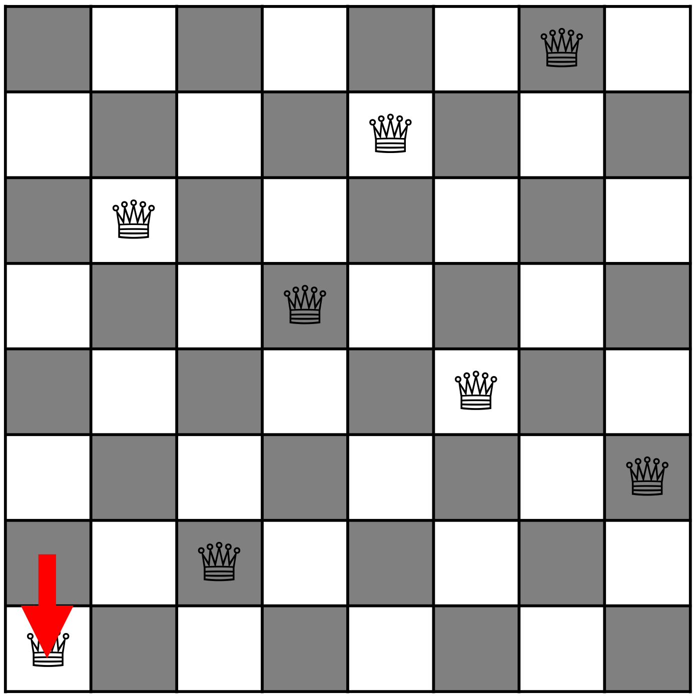
Solution:

N chess queens arrangement on the chessboard in such a way that no two queens attack (diagonally, horizontally, and vertically).

N-Queen Problem



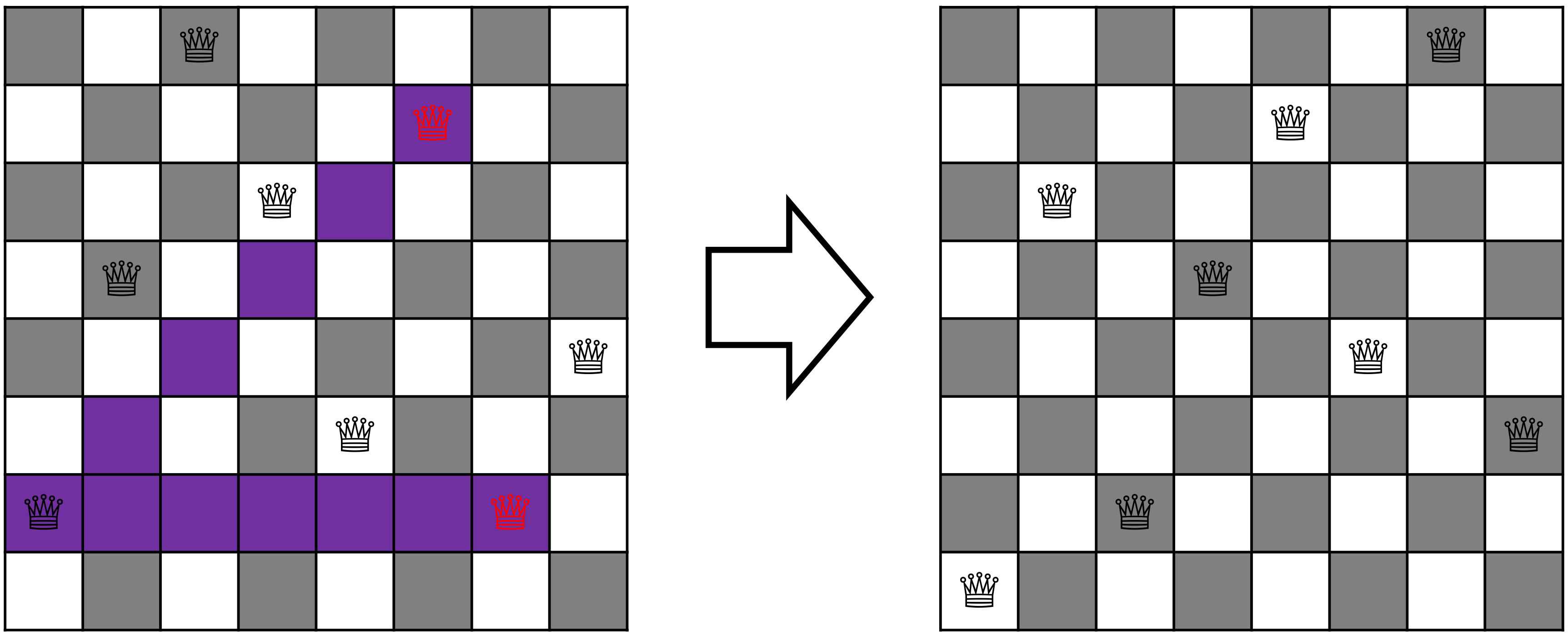




PARTIAL SOLUTION **SOLUTION**

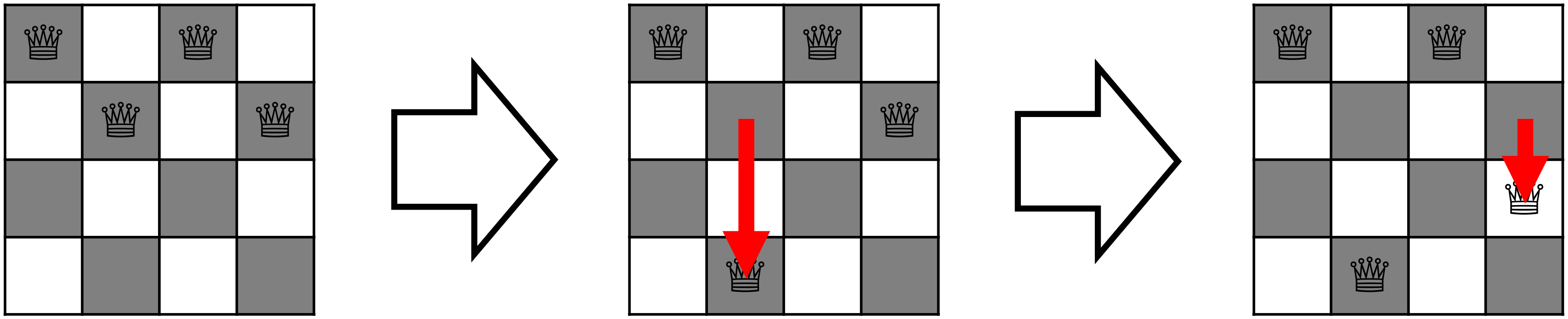
N-Queen Problem

$f(State_i) = \textcolor{red}{2} \quad > \quad f(State_j) = \textcolor{green}{0}$



State i | **State j: State i's neighbor**

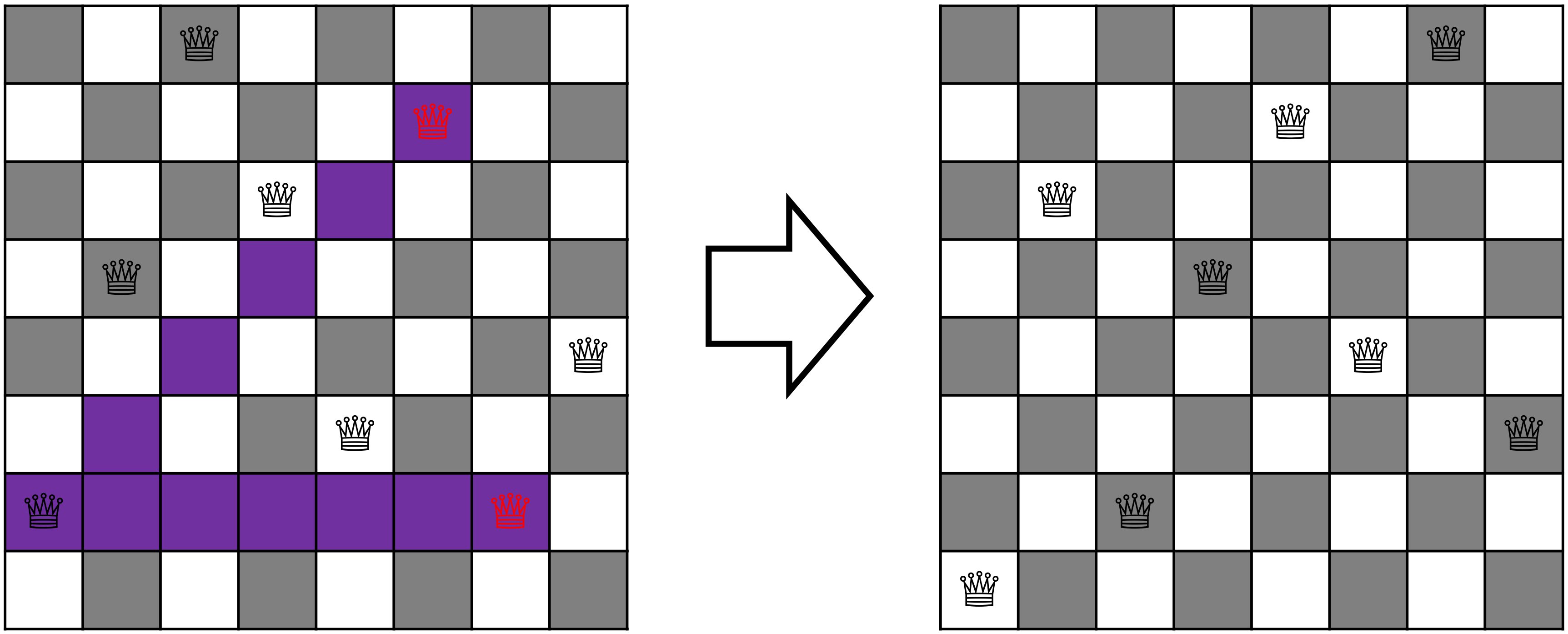
N-Queen Problem



Neighbor: a state one "queen move" away

N-Queen Problem

$f(State_i) = \color{red}{2}$ $>$ $f(State_j) = \color{green}{0}$



LOCAL MINIMUM **GLOBAL MINIMUM**

N-Queen Problem: Heuristic

$f(State_i) = h(State_i) =$ **17 conflicts**

			♛				
♛				♛			
	♛				♛		♛
		♛				♛	

CURRENT STATE

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

POTENTIAL “MOVES”

Local Search:
Hill Climbing

Hill Climbing (Greedy Local) Search

- **The most primitive informed search approach**
  - **a naive greedy algorithm**
  - **~~evaluation~~ objective function: value of next state**
  - **does not care about the "bigger picture" (for example: total search path cost)**

- **Practicalities:**
  - **does not keep track of search history**

Hill Climbing Search: Pseudocode

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*

"…like trying to find the top of Mount Everest in a thick fog while suffering from amnesia"

Hill Climbing Search: Pseudocode

```
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
```

VALUE[] → OBJECTIVE FUNCTION → f()

Hill Climbing Search: Pseudocode

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] $\leq$ VALUE[current] **then return** STATE[*current*]
    current ← *neighbor*

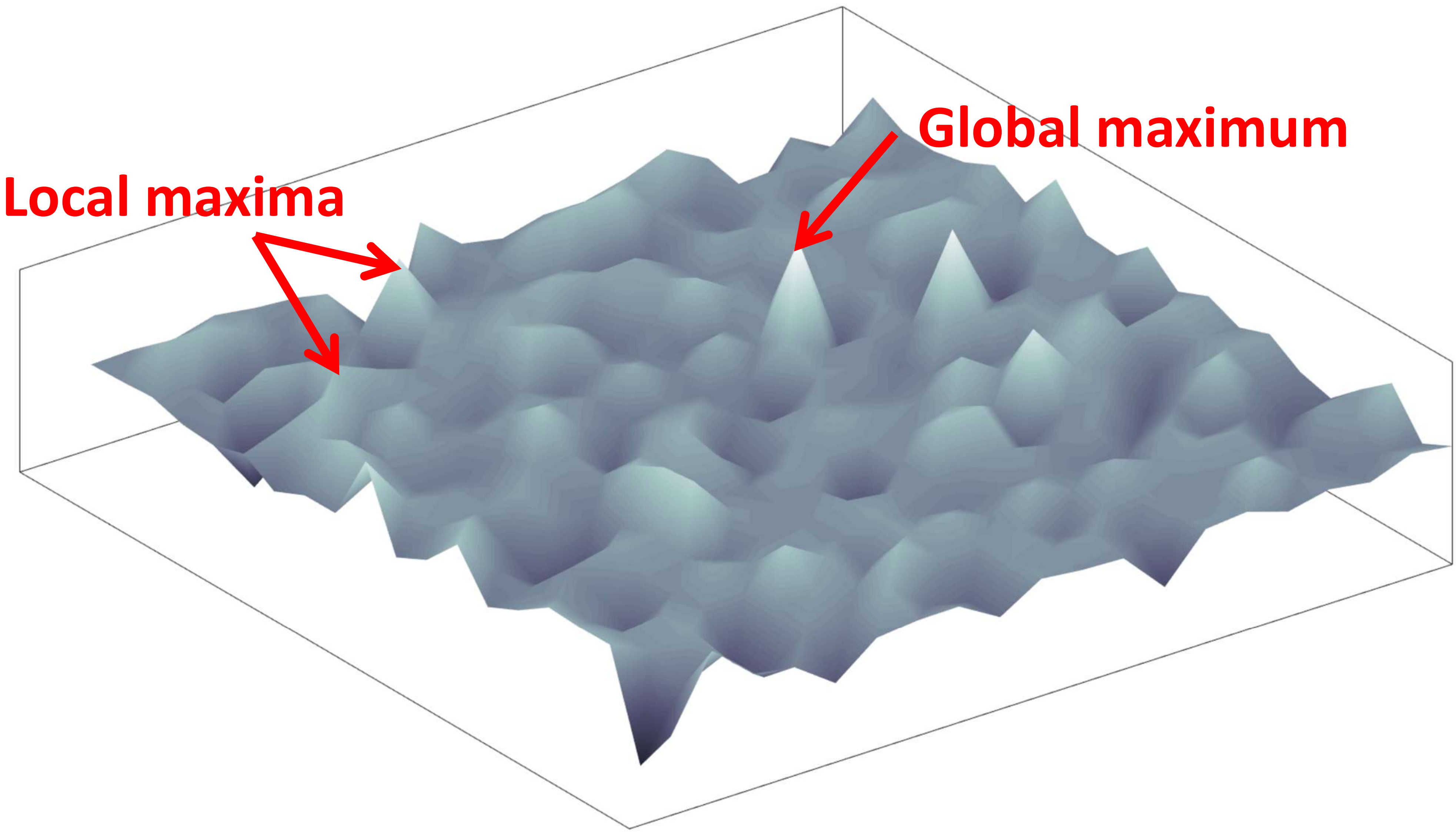
Change comparison operator if minimizing

Hill Climbing and Difficult State Spaces / Environments

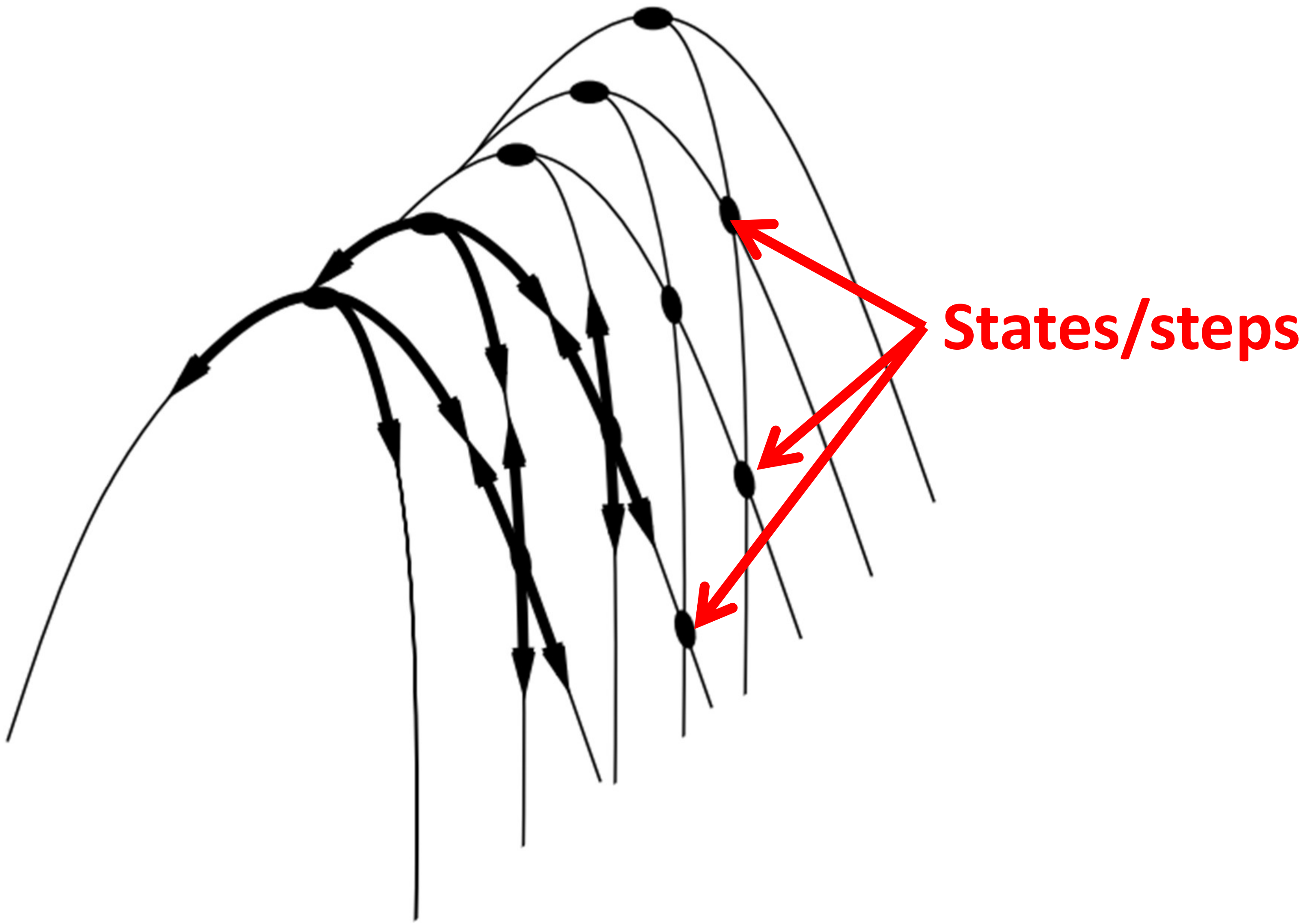
“Getting Stuck”

- **Local maxima**: a local maximum is a peak that is higher than each of its neighboring states, but lower than the global maximum.
  - Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upwards towards the peak, but will then be stuck with nowhere else to go
- **Ridge**: ridges result in a sequence of local maxima that is
  - very difficult for greedy algorithms to navigate.
- **Plateau**: a plateau is an area of the state space landscape where the evaluation function is “flat”. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which it is possible to make progress.
  - A hill-climbing search might be unable to find its way off the plateau

Hill Climbing Problems: Local Maxima

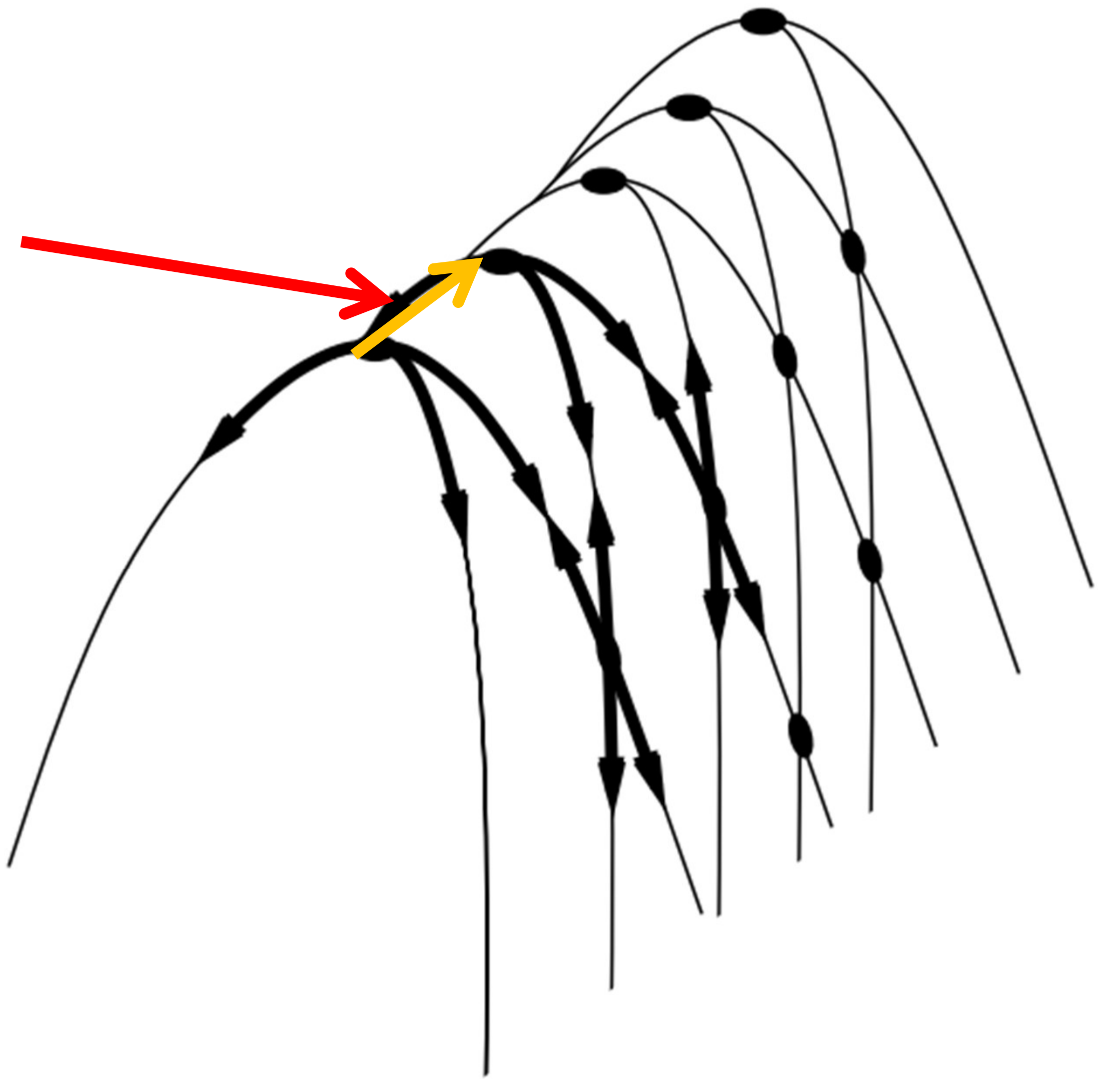


Hill Climbing Problems: Ridges

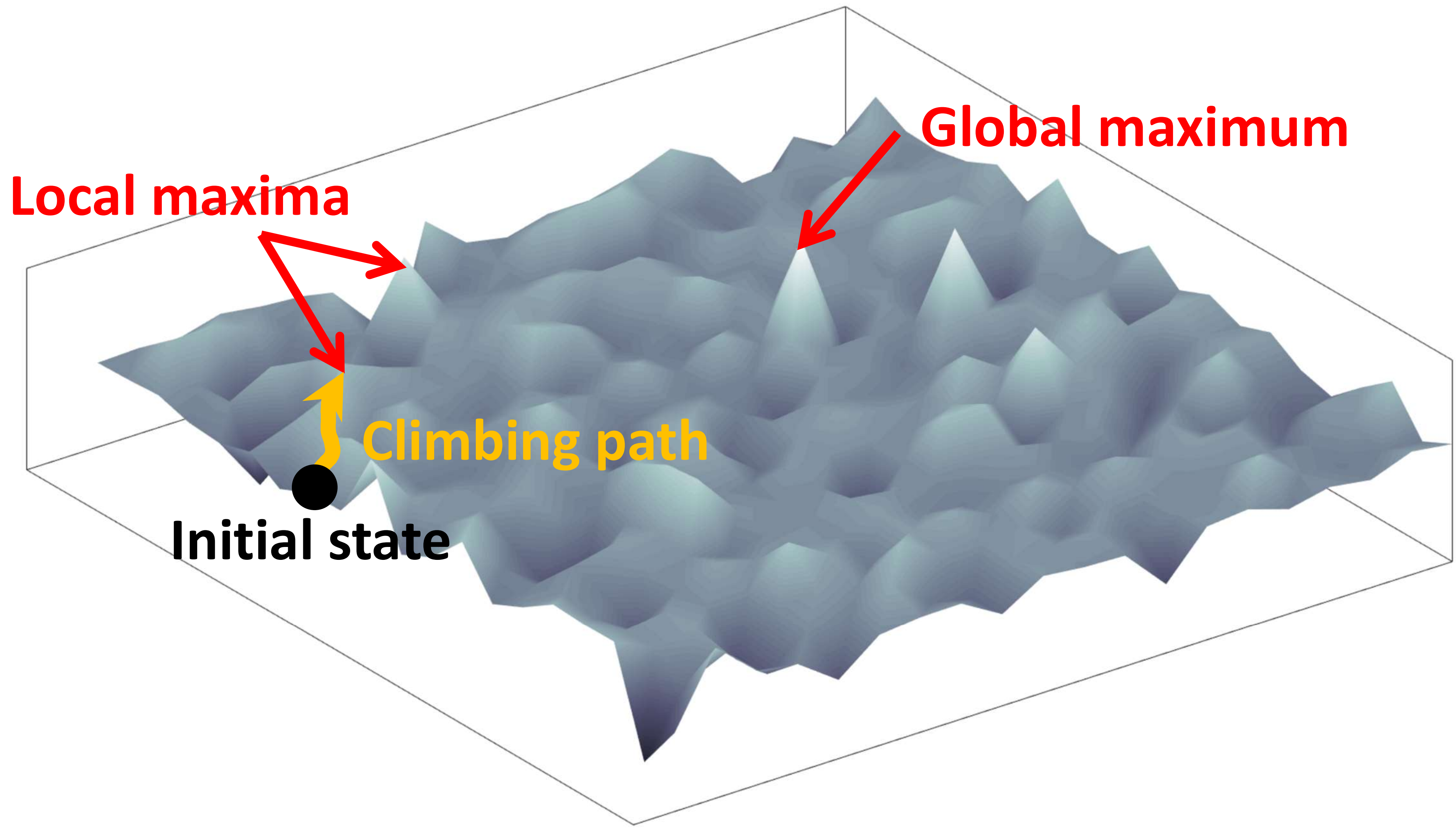


Hill Climbing Problems: Ridges

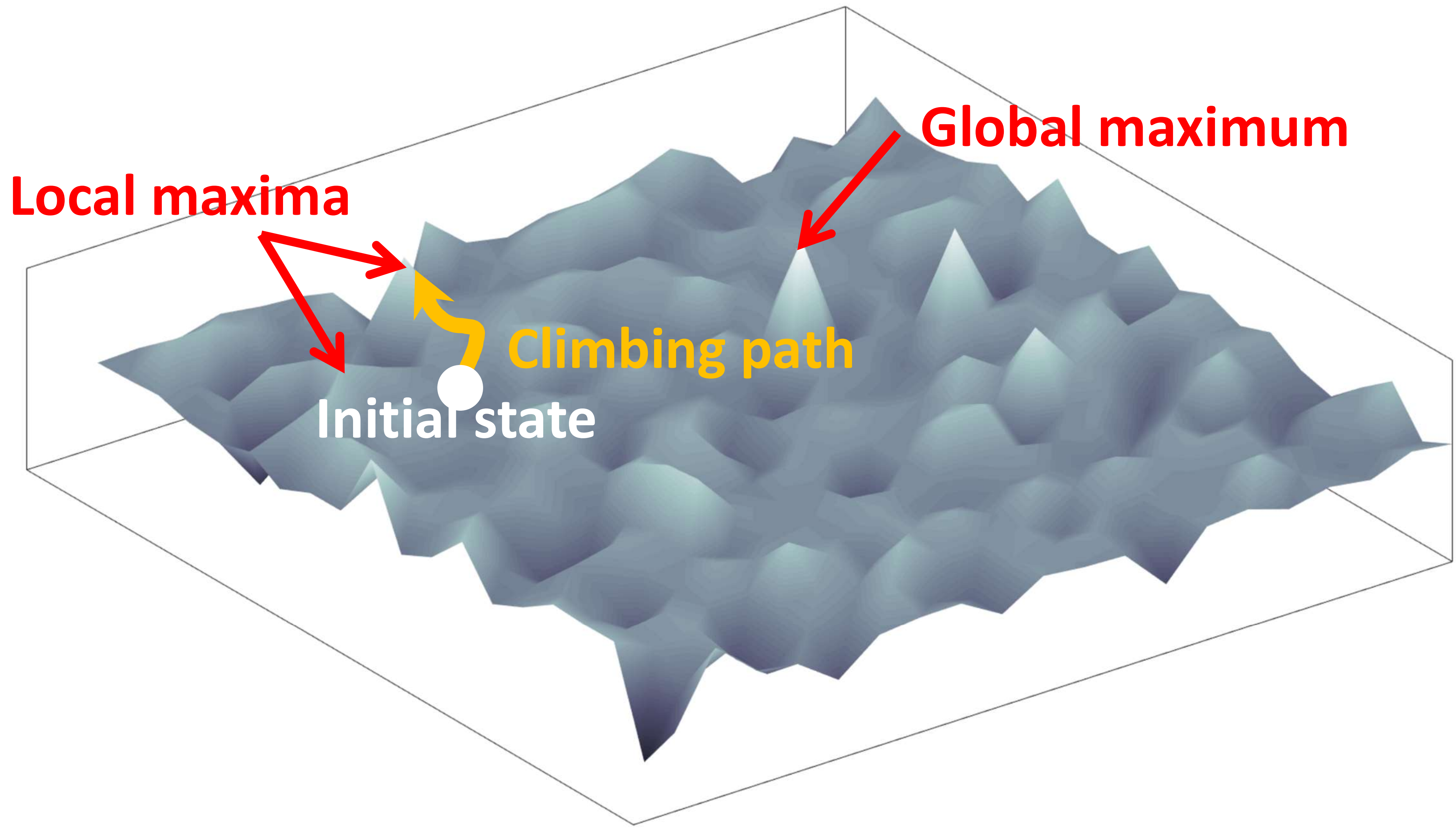
No such move!



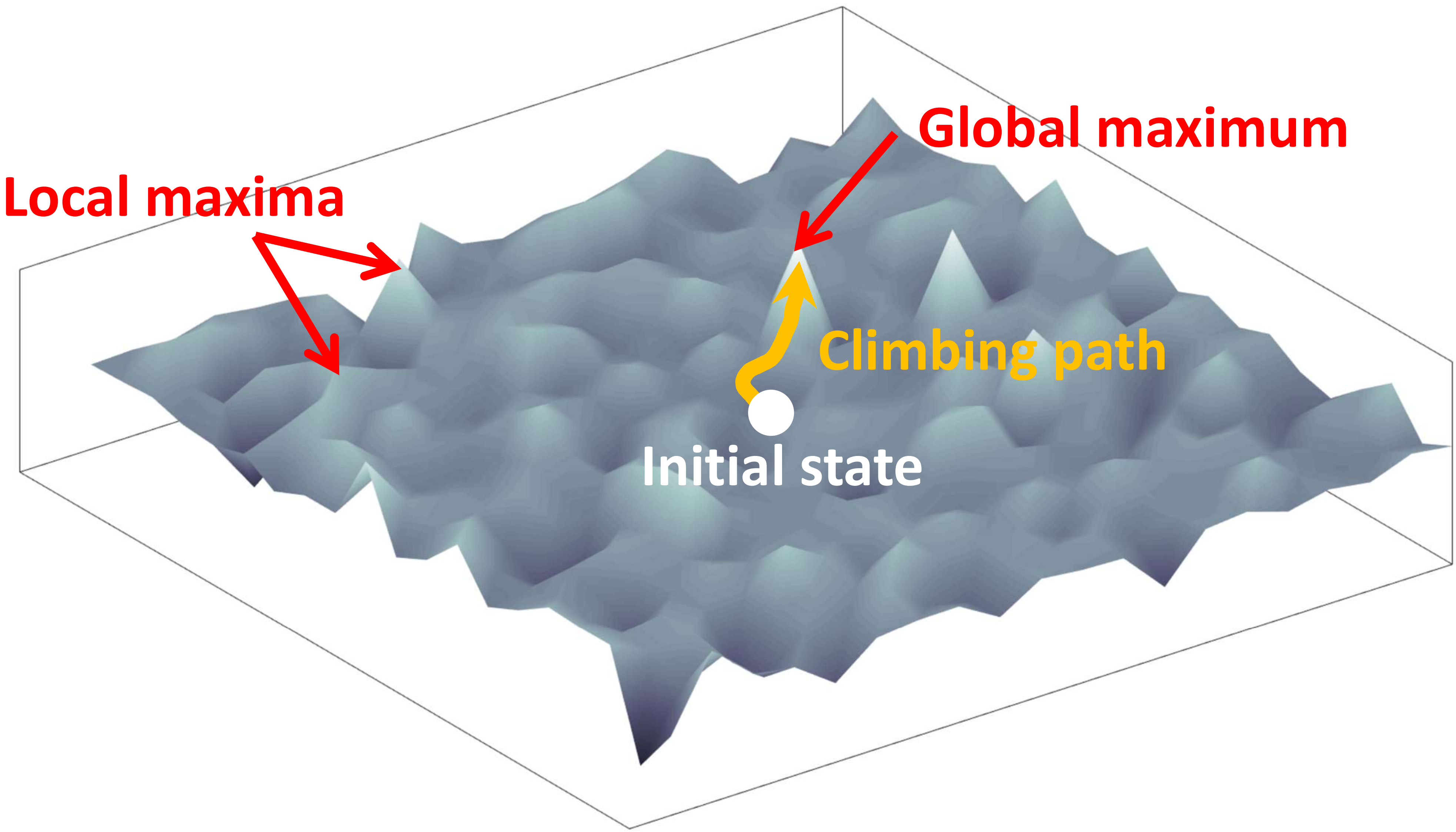
Hill Climbing Problems: Local Maxima



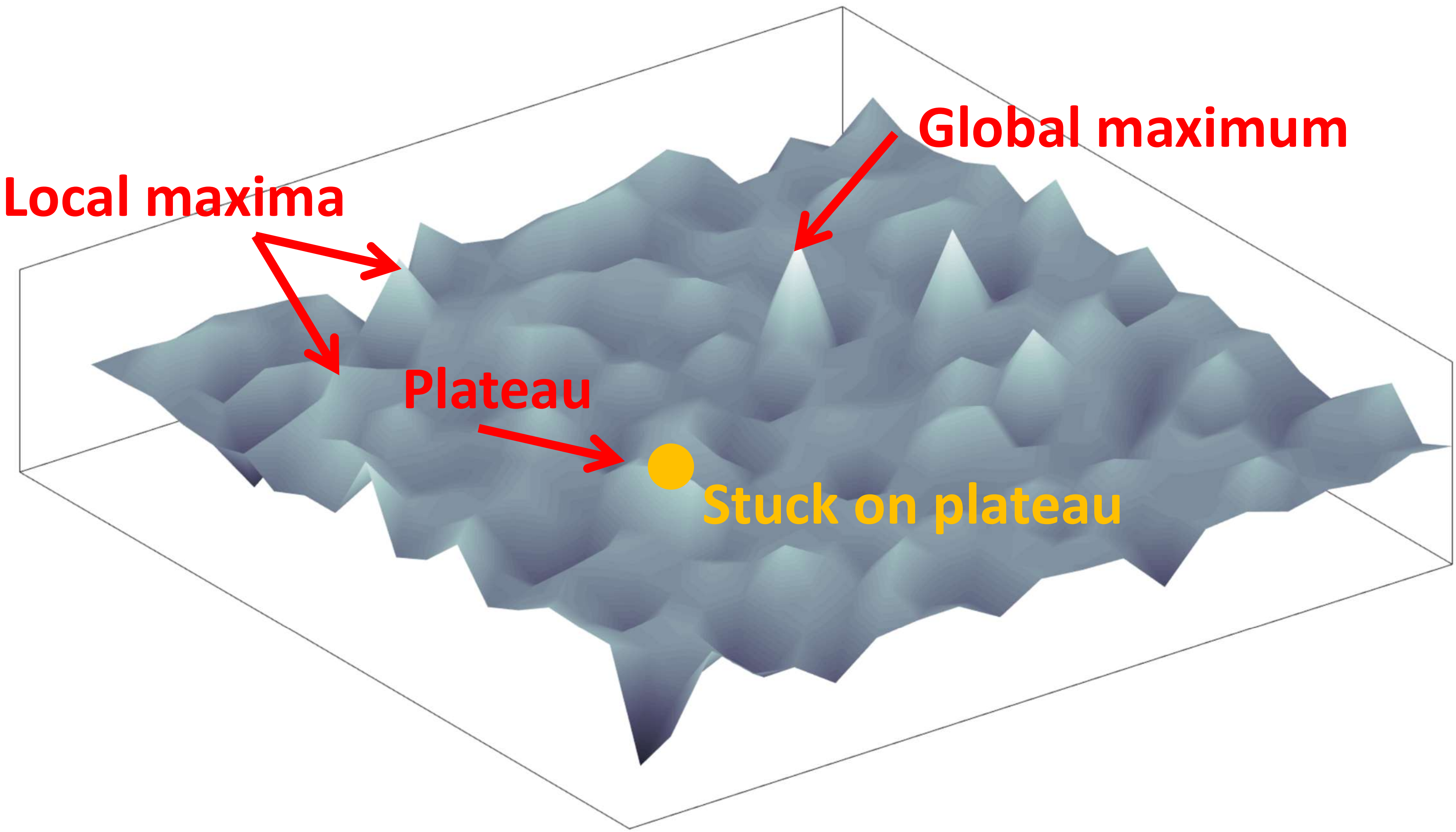
Hill Climbing Problems: Local Maxima



Hill Climbing Problems: Local Maxima



Hill Climbing Problems: Plateaus

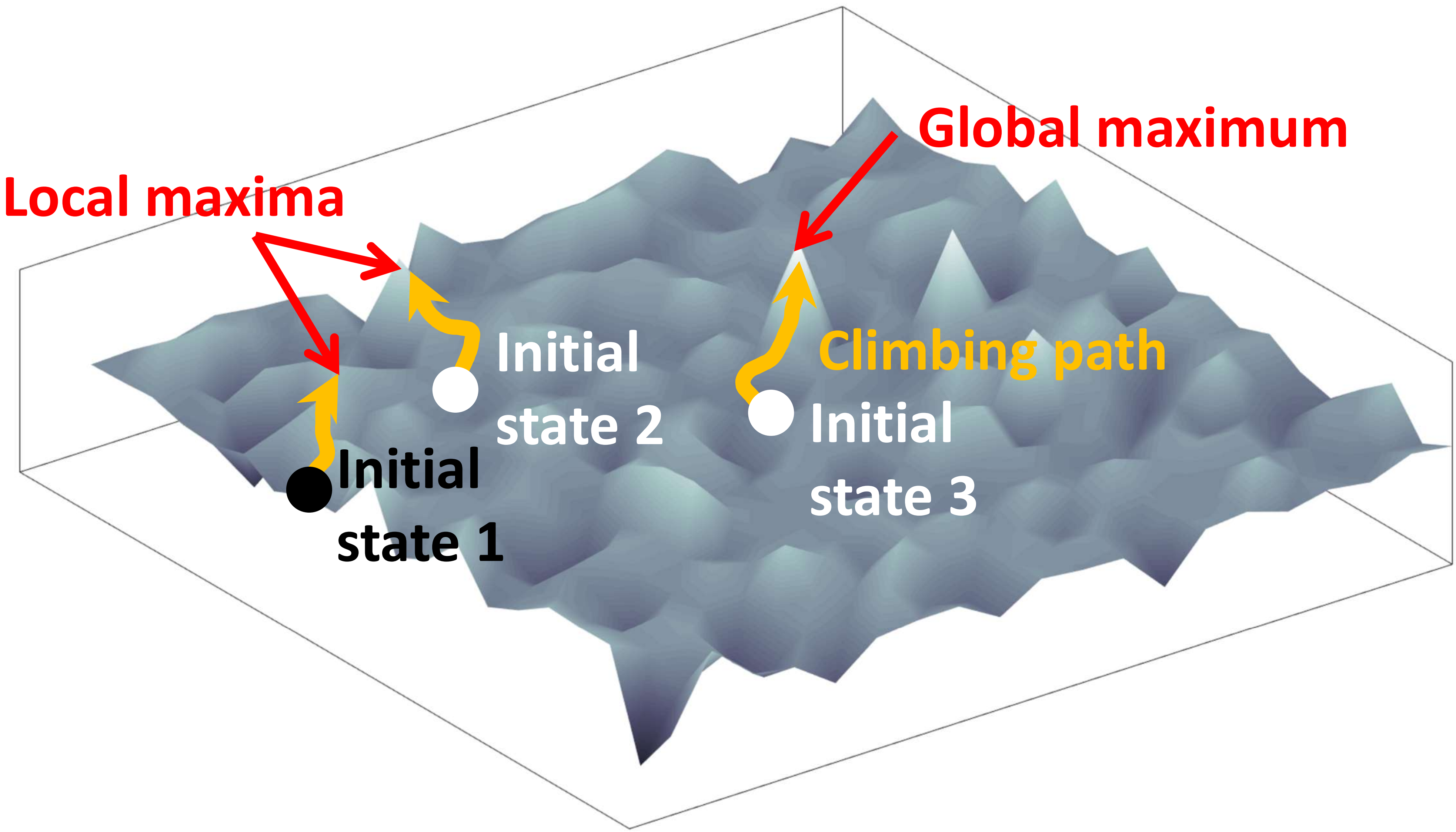


Hill Climbing:

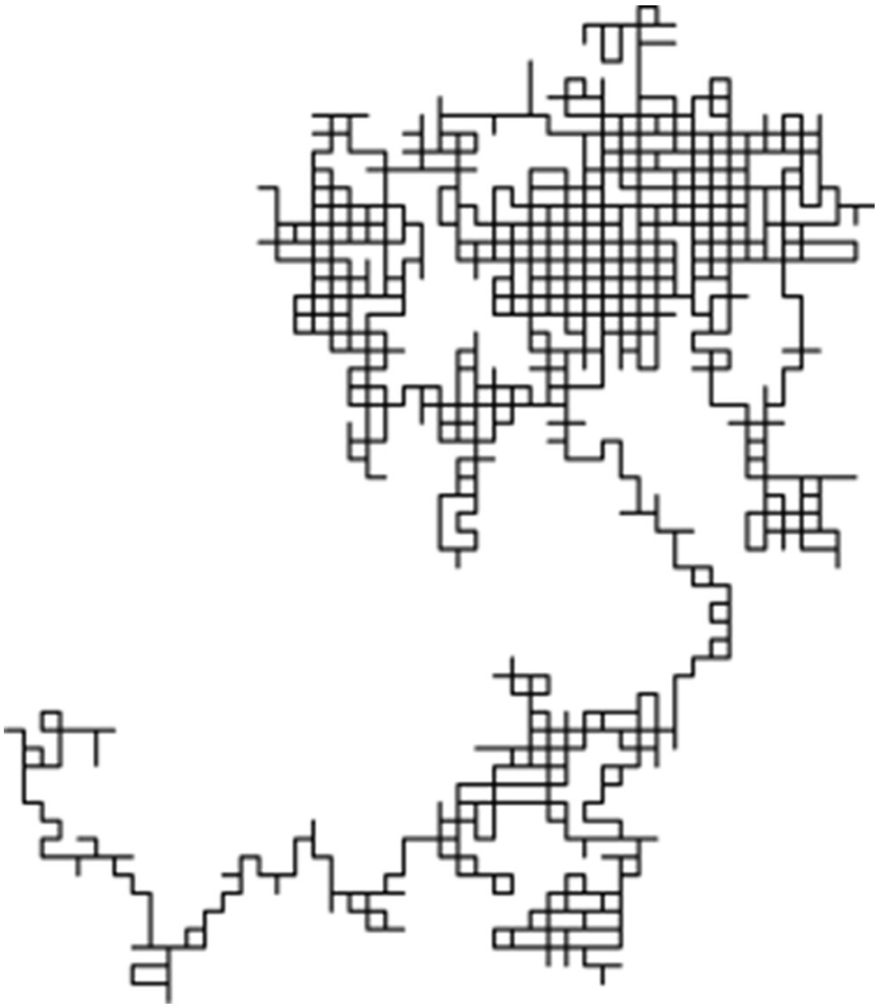
Optimality not Guaranteed

→ Stochastic Hill Climbing Can Help

Hill Climbing: Random “Restarts”



Random Walk



In mathematics, **a random walk**, sometimes known as **a drunkard's walk**, is a random process that describes a path that consists of a succession of random steps on some mathematical space.

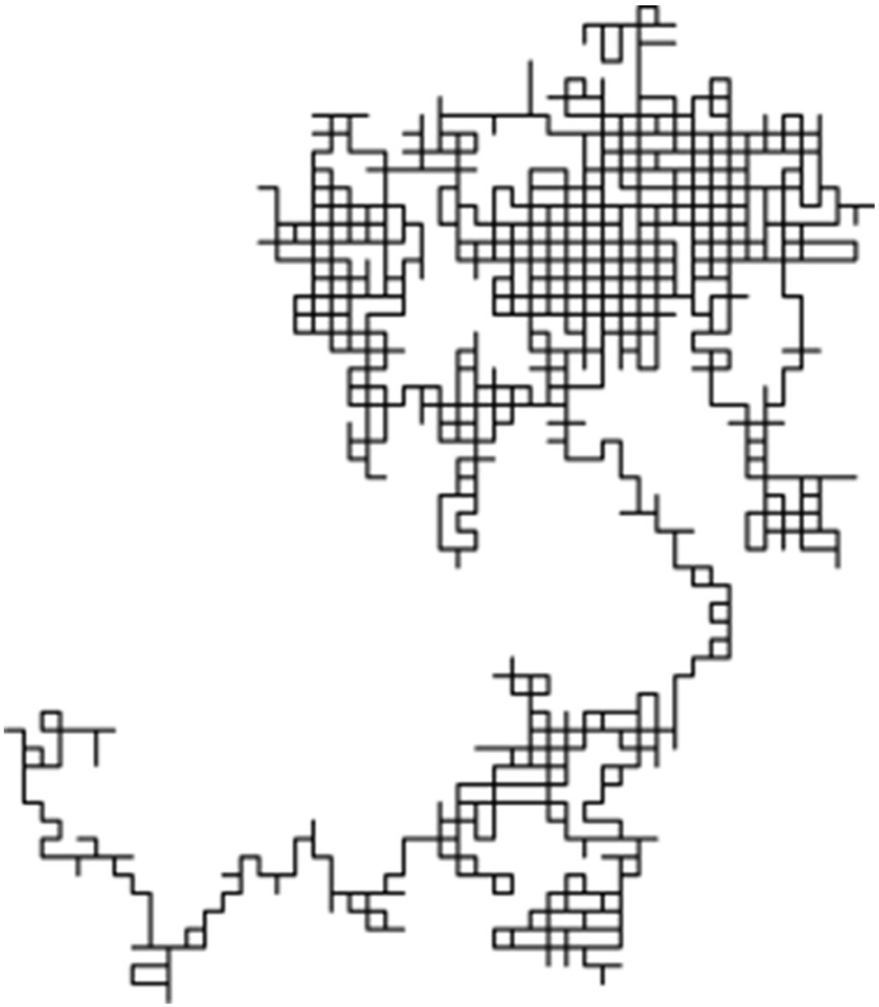
Source: https://en.wikipedia.org/wiki/Random_walk

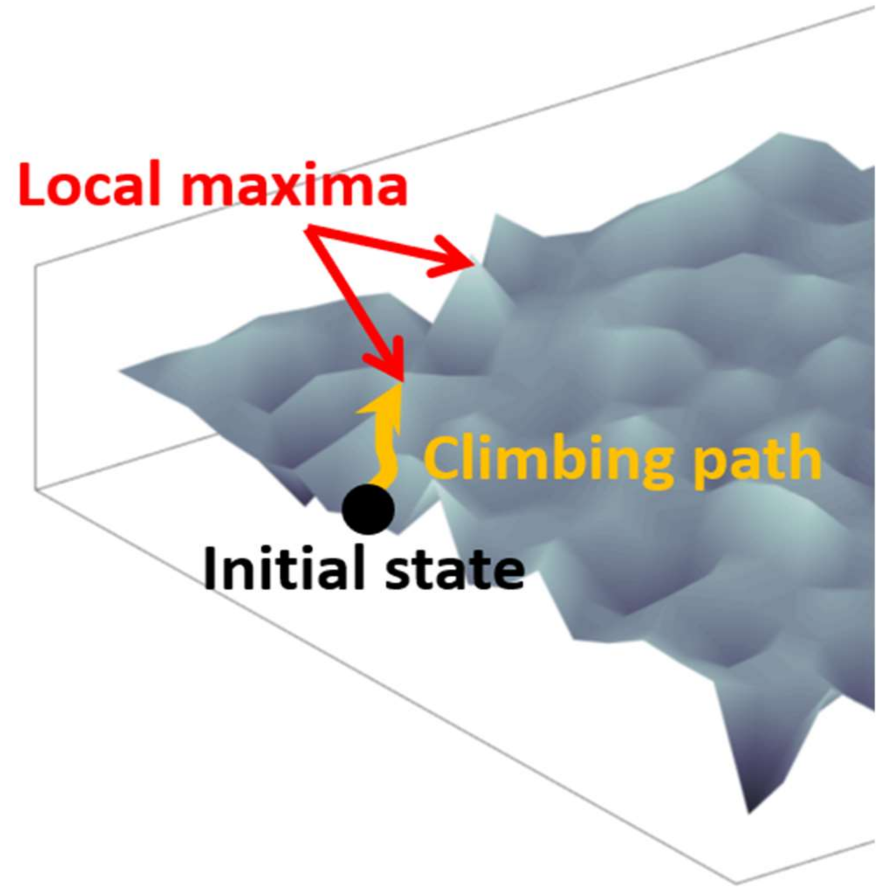
Measuring Searching Performance

Search algorithms can be evaluated in four ways:

- **Completeness: Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?**
- **Cost optimality: Does it find a solution with the lowest path cost of all solutions?**
- **Time complexity: How long does it take to find a solution? (in seconds, actions, states, etc.)**
- **Space complexity: How much memory is needed to perform the search?**

Random Walk vs. Hill Climbing





Local Search:
Simulated Annealing

Metropolis Heuristics

- **Basic idea**
  - **accept a move if it improves the objective value**
  - **accept "bad moves" as well with some probability**
  - **the probability depends on how "bad" the move is**
  - **inspired by statistical physics**
- **How to choose the probability?**
  - **t is a scaling parameter (called temperature)**
  - **Δ is the difference f(n) – f(s)**
  - **a degrading move is accepted with probability**

$$\exp\left(\frac{-\Delta}{t}\right)$$

Metropolis Heuristics: Fixed T

- What happens for a large T?
  - probability of accepting a degrading move is large
- What happens for a small T ?
  - probability of accepting a degrading move is small

- Finding the correct temperature T is hard

- Let us gradually change the temperature
  - simulated annealing

Simulated Annealing: What Is It?

In metallurgy, annealing is the process used to temper or harden metals and glass by heating them to a high temperature T and then gradually cooling them, thus allowing the material to coalesce into a low-energy (E) crystalline state (less or no defects).

Key ideas:

- Use Metropolis algorithm but adjust the temperature dynamically
- Start with a high temperature (random moves)
- Decrease the temperature
- When the temperature is low becomes a local search

Simulated Annealing: Pseudocode

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

Similar idea to Hill Climbing but with "downward moves" (really "upward" here) allowed

Simulated Annealing: Pseudocode

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

Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

Simulated Annealing: Pseudocode

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

"Cooling" "plan" or function

Temperature / Cooling Schedule

Idea: start with Large T and "slowly" decrease it

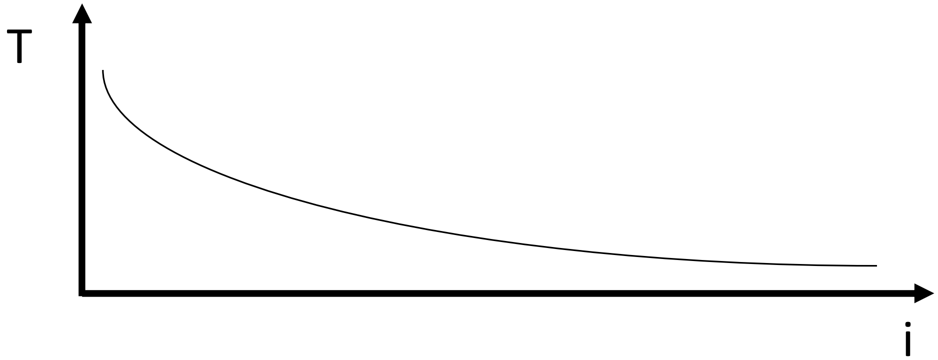
- **linear T(i)**

$$T_i = T_{INITIAL} - i * \delta$$

- **Exponential**

$$T_i = T_{INITIAL} * e^{-i * \lambda}$$

- **other**



Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  $\boxed{current \leftarrow problem.\text{INITIAL}}$
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    $next \leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow \text{VALUE}(current) - \text{VALUE}(next)$
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

***current*: initial state → start "somewhere"**

Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
    current ← *problem*.INITIAL
    $\boxed{\textbf{for } t = 1 \textbf{ to } \infty \textbf{ do}}$
        $T \leftarrow schedule(t)$
        if $T = 0$ **then return** *current*
        next ← a randomly selected successor of *current*
        $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
        if $\Delta E > 0$ **then** *current* ← *next*
        else *current* ← *next* only with probability $e^{-\Delta E/T}$

Technically a while loop

Simulated Annealing: Pseudocode

```
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)
```

Update T according to the "cooling" "plan"

Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING(*problem*, *schedule*) $\boxed{\textbf{returns}\text{ a solution state}}$
    current ← *problem*.INITIAL
    for $t = 1$ **to** ∞ **do**
        $T \leftarrow schedule(t)$
        $\boxed{\textbf{if } T = 0 \textbf{ then return } current}$
        next ← a randomly selected successor of *current*
        $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
        if $\Delta E > 0$ **then** *current* ← *next*
        else *current* ← *next* only with probability $e^{-\Delta E/T}$

If completely "cooled"
(T=0) → stop

Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    $\boxed{next \leftarrow \text{a randomly selected successor of } current}$
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

Pick a neighbor of *current* at random

Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for t = 1 **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\boxed{\Delta E \leftarrow \text{VALUE}(current) - \text{VALUE}(next)}$
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

Calculate the "energy level" change between the two states

Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  $current \leftarrow problem$.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    $next \leftarrow$ a randomly selected successor of *current*
    $\boxed{\Delta E \leftarrow \text{VALUE}(current) - \text{VALUE}(next)}$
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

VALUE[State] is the evaluation / objective function f(State)

Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    $\boxed{\textbf{if } \Delta E > 0 \textbf{ then } current \leftarrow next}$
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

If *next* state leads to lowering of "energy level", go there

Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  $current \leftarrow problem.$INITIAL
  for t = 1 **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if T = 0 **then return** *current*
    $next \leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    $\boxed{\textbf{if } \Delta E > 0 \textbf{ then } current \leftarrow next}$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

We could be maximizing here as well. Just change > to <

Simulated Annealing: Pseudocode

ACCEPTANCE CRITERIA

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  $current \leftarrow problem$.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    $next \leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

Simulated Annealing: Pseudocode

function SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  $current \leftarrow problem$.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow$ VALUE($current$) – VALUE($next$)
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

If *next* state does NOT lead to lowering of "energy level", go there, but ONLY with certain probability

Simulated Annealing: Pseudocode

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

ACCEPT move with certain probability

If exp(-ΔE/T) > random[0,1)
ACCEPT

Simulated Annealing: Pseudocode

```
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)
```

If *next* state does NOT lead to lowering of "energy level" → use this option to sometimes "escape local minimum"

Simulated Annealing: Pseudocode

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

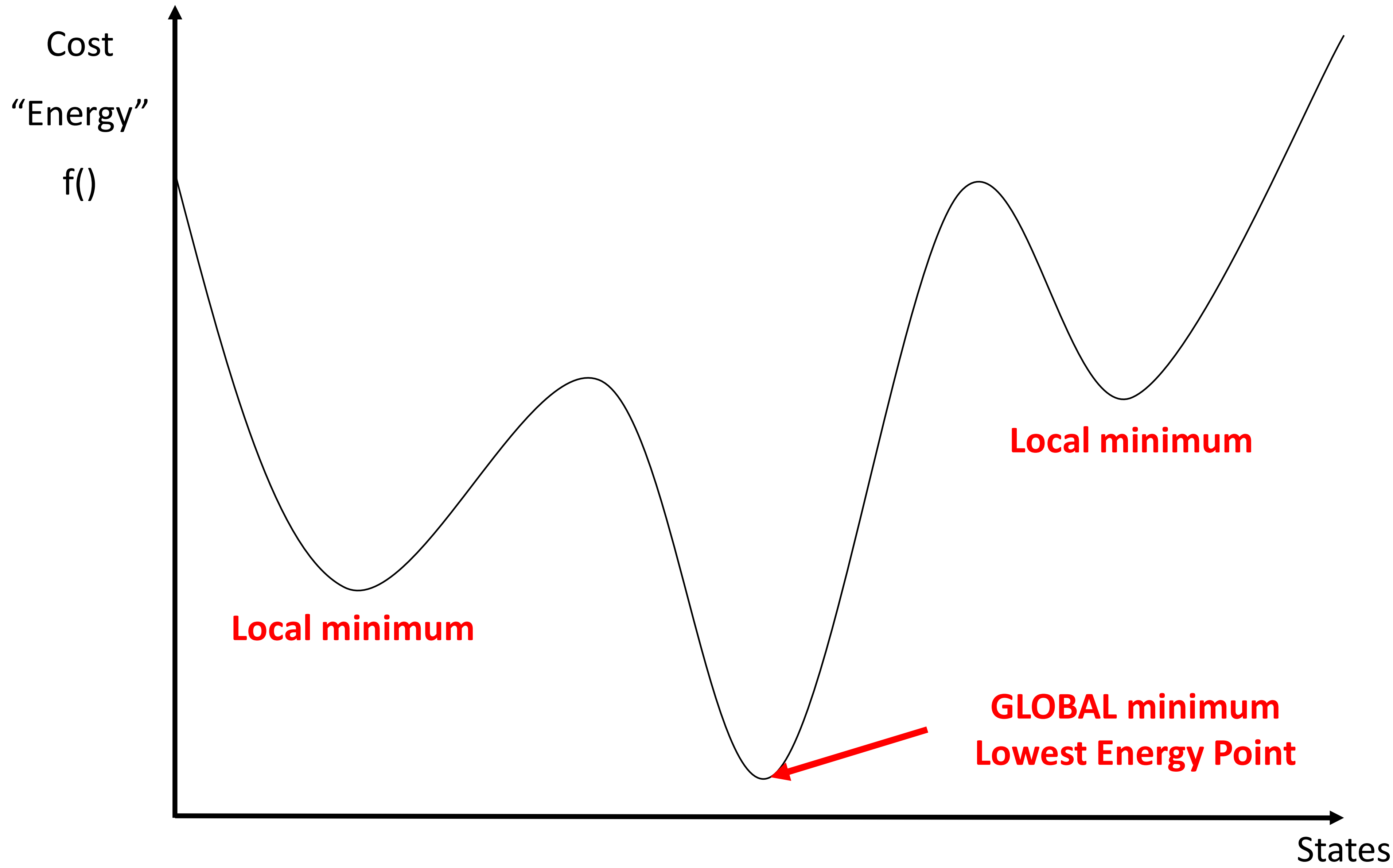
If *next* state <u>does NOT</u> lead to lowering of "energy level" → use this option to sometimes "escape local minimum"

Simulated Annealing: Article

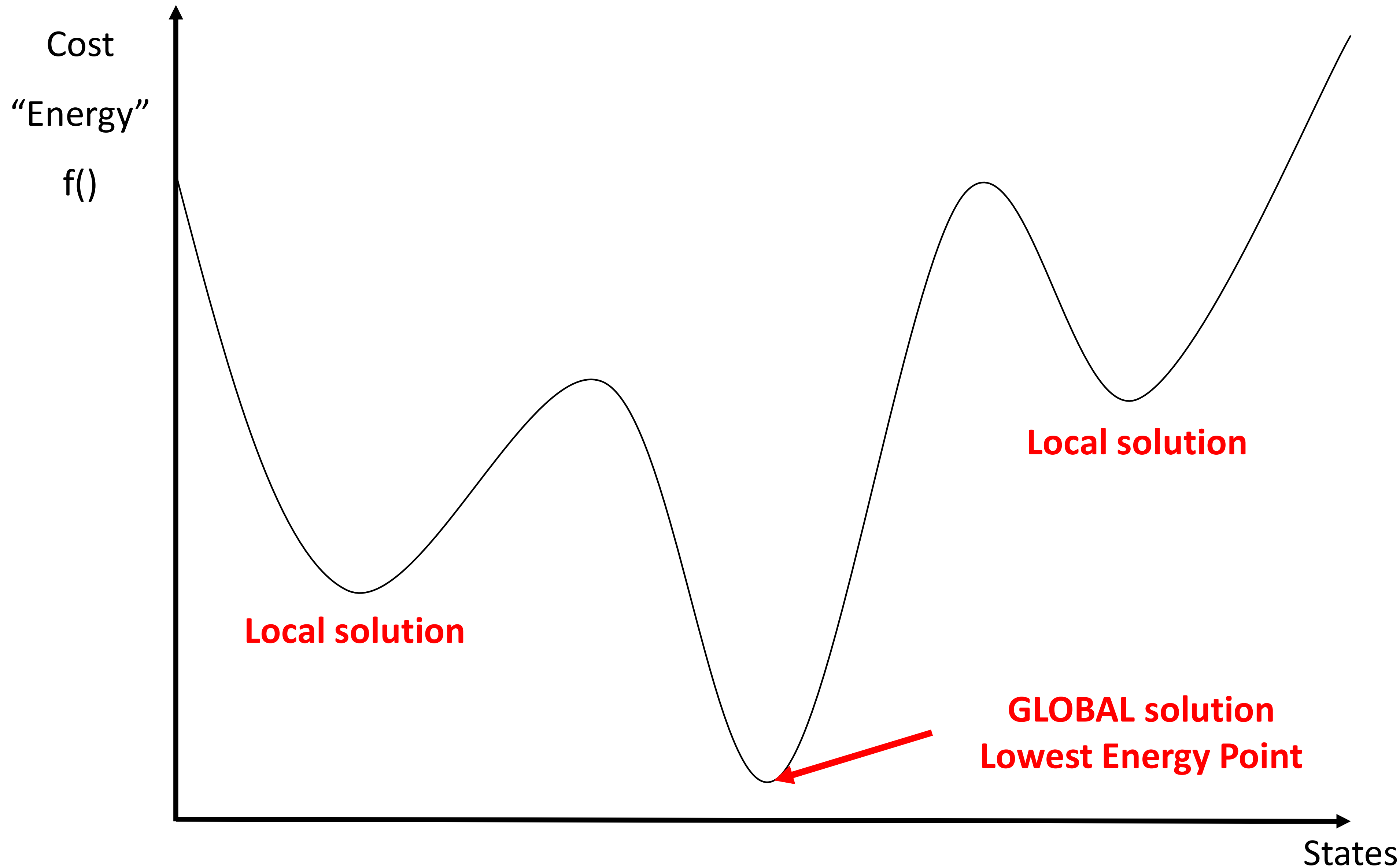
Optimization by Simulated Annealing

- S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi
- *Science* 13 May 1983:
- Vol. 220, Issue 4598, pp. 671-680
- https://science.sciencemag.org/content/220/4598/671

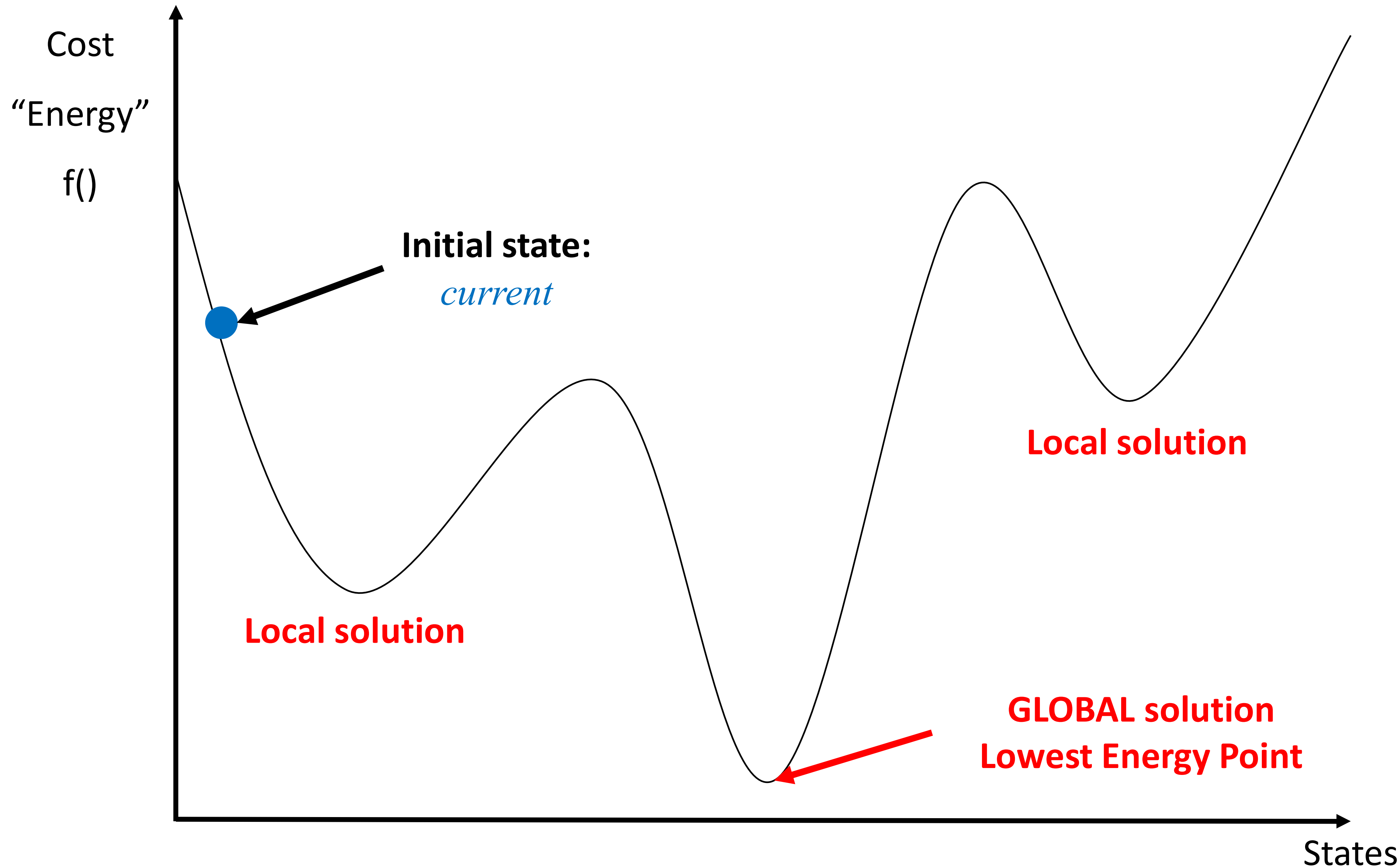
Simulated Annealing: Progress



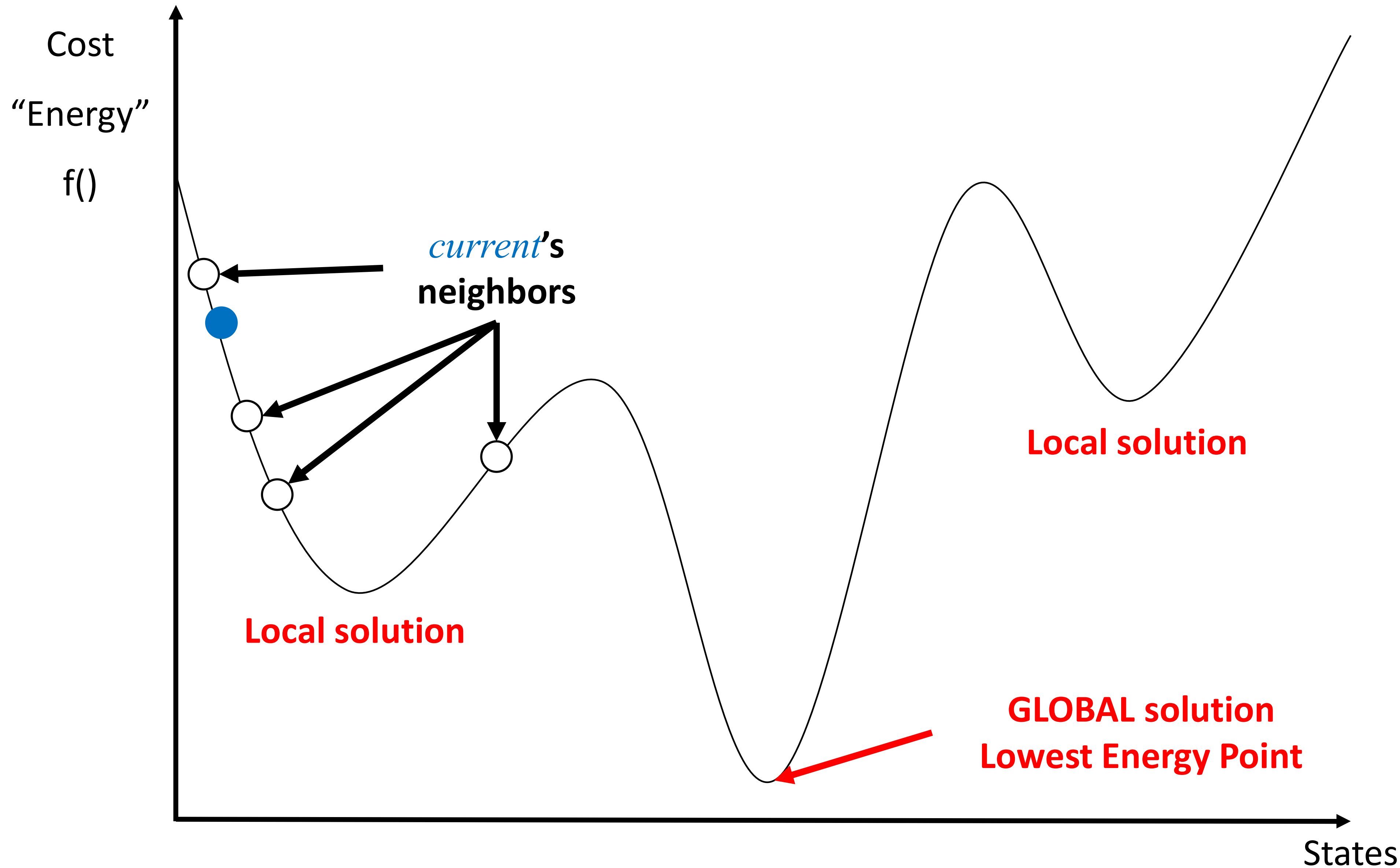
Simulated Annealing: Progress



Simulated Annealing: Progress

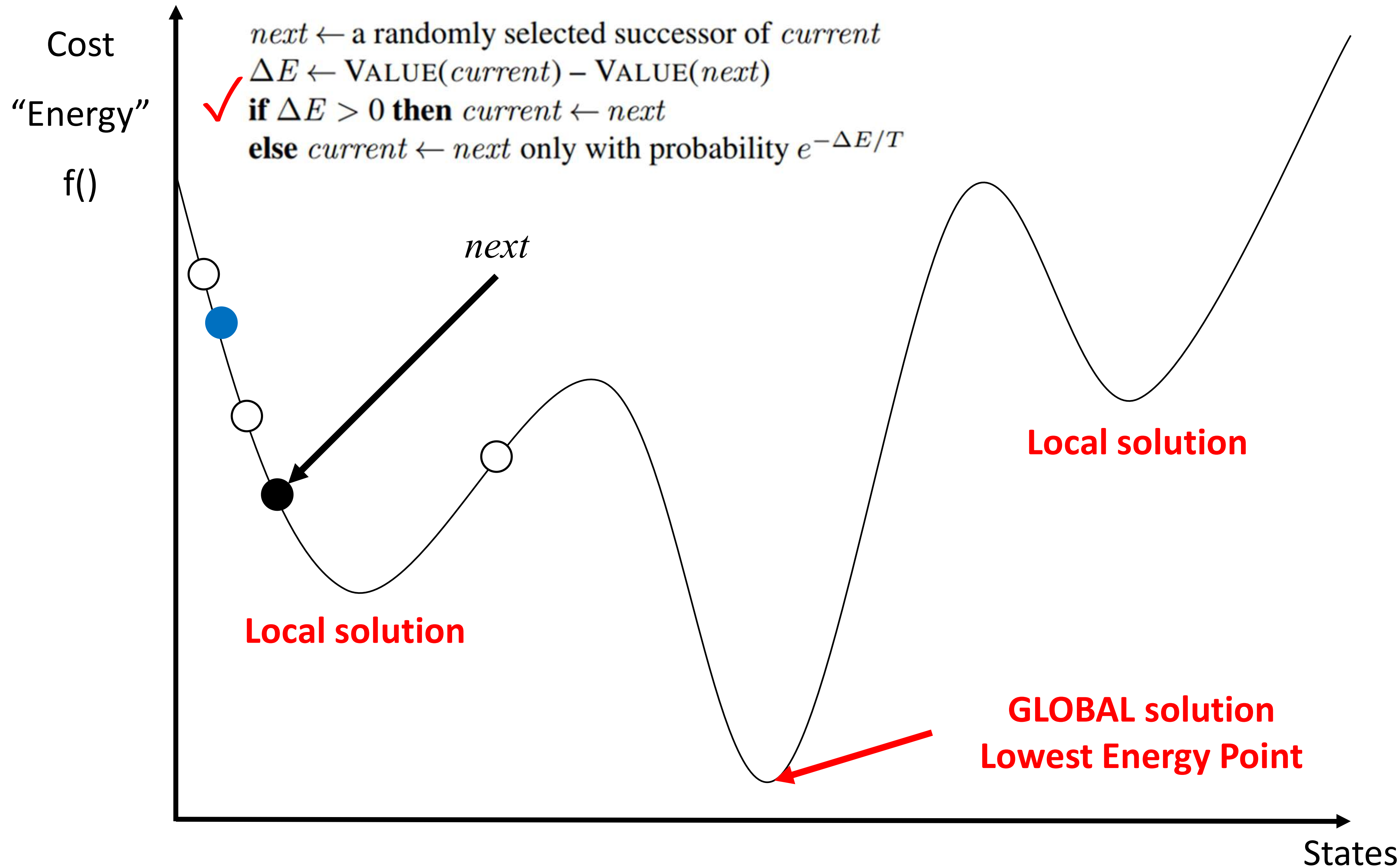


Simulated Annealing: Progress

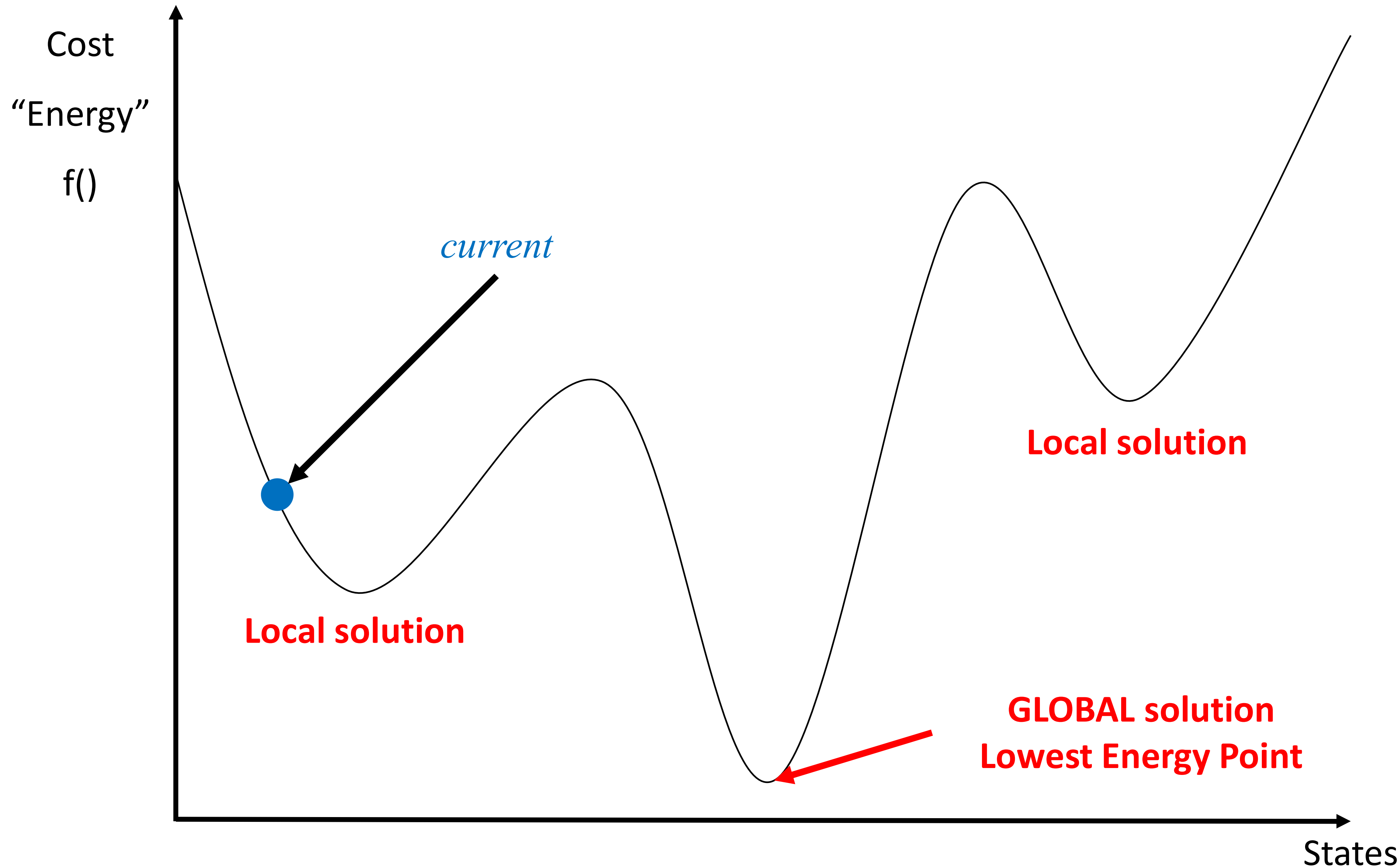


Simulated Annealing: Progress

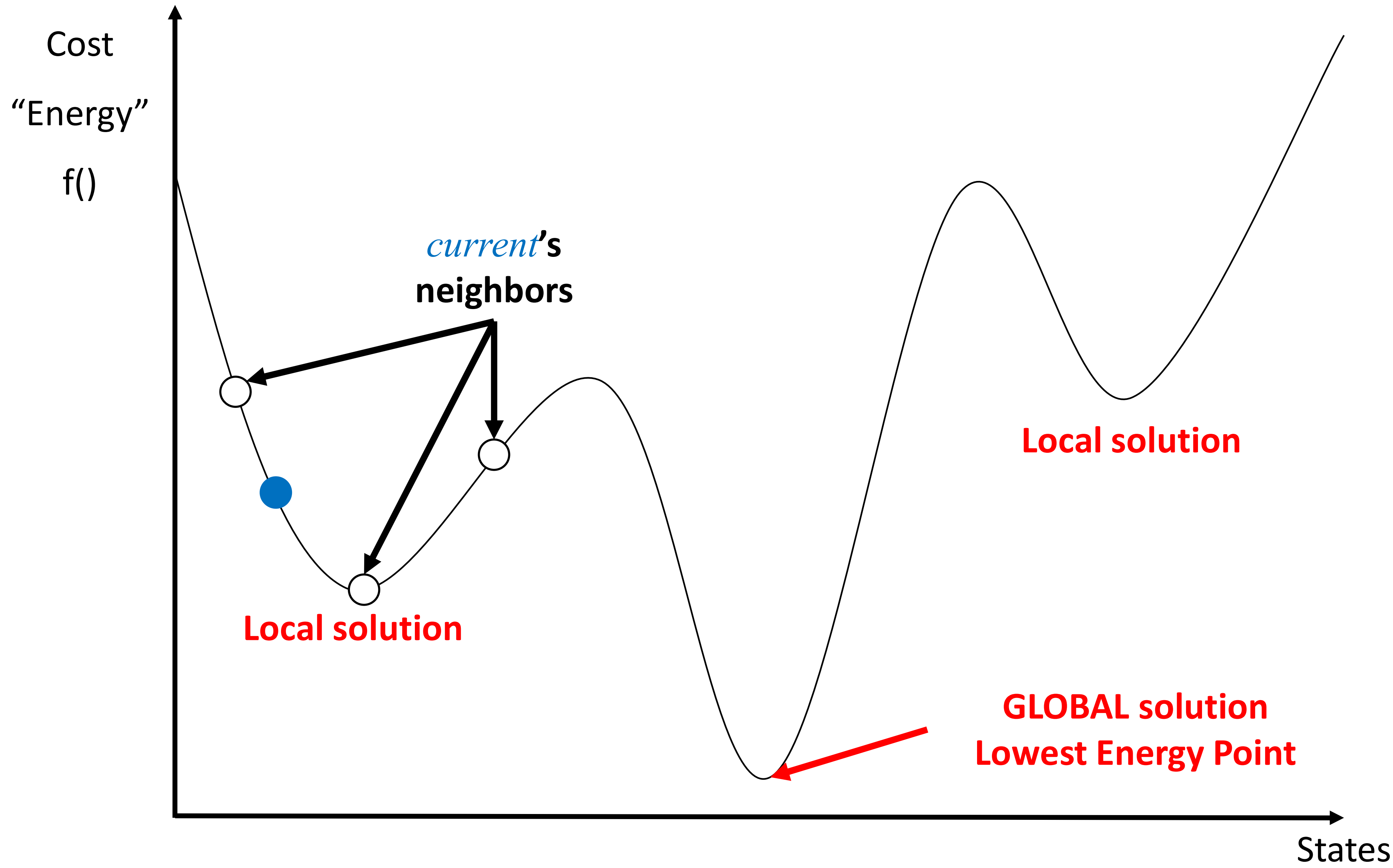
$next \leftarrow$ a randomly selected successor of $current$
$\Delta E \leftarrow$ VALUE($current$) – VALUE($next$)
✓ **if** $\Delta E > 0$ **then** $current \leftarrow next$
else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$



Simulated Annealing: Progress

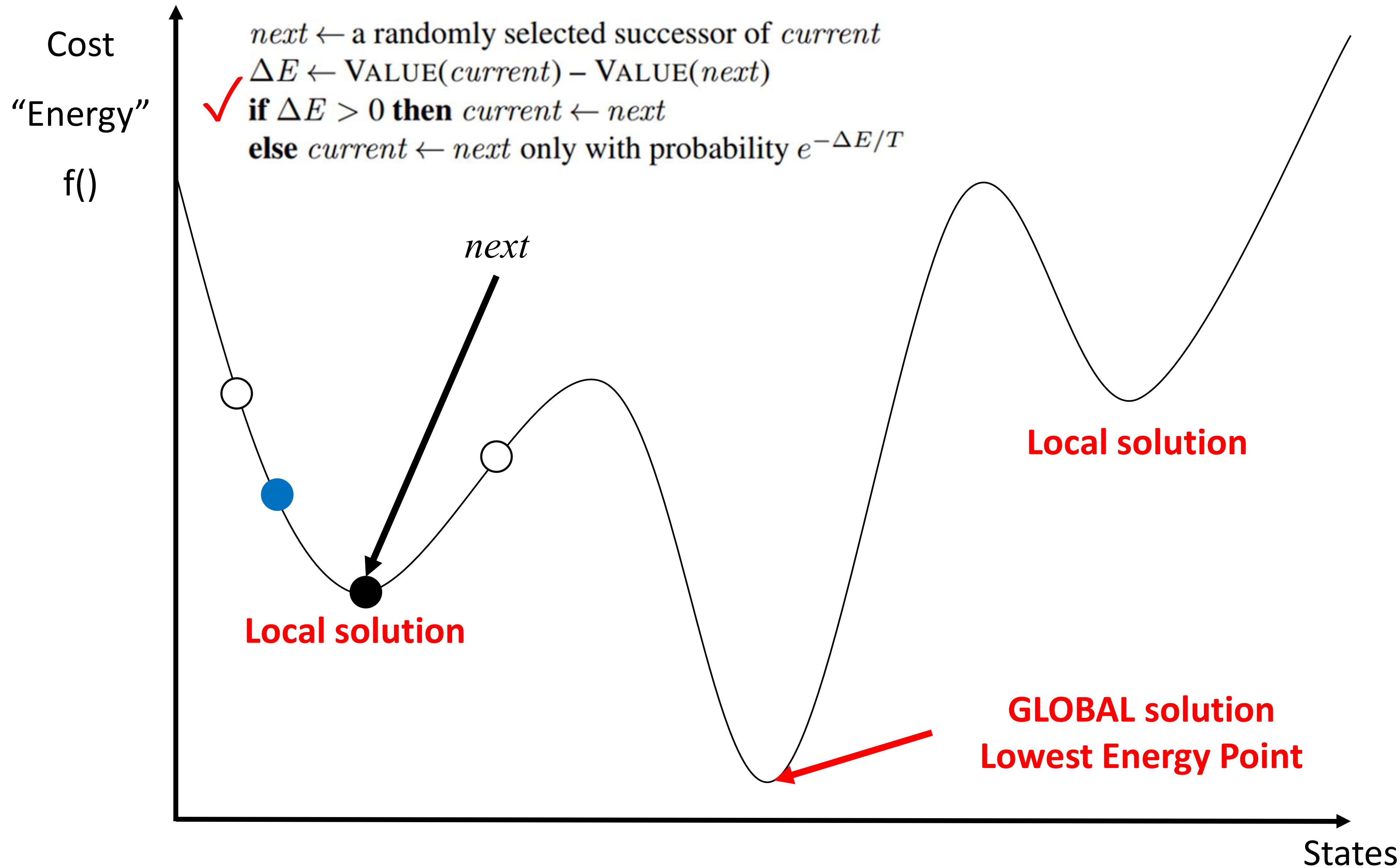


Simulated Annealing: Progress

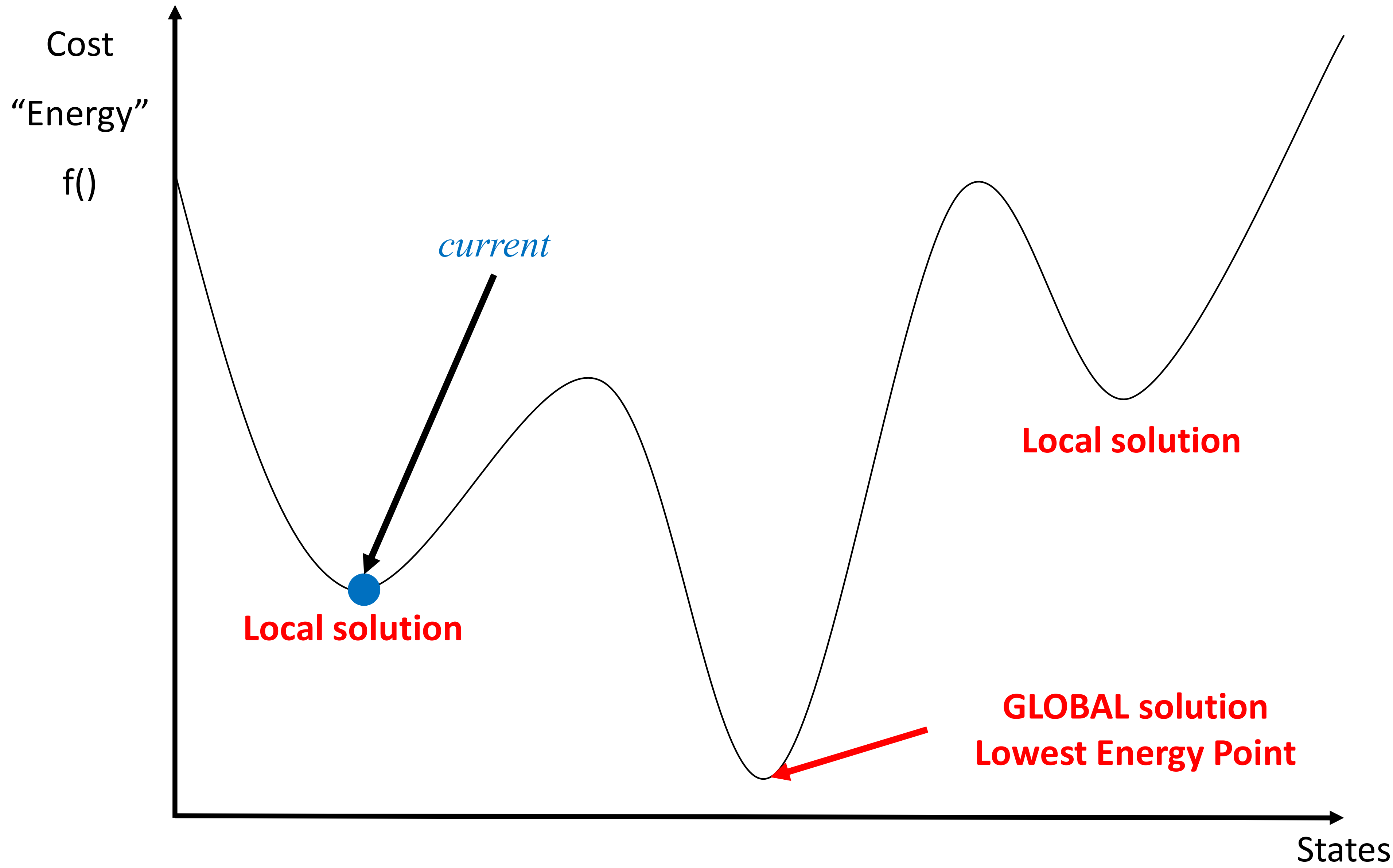


Simulated Annealing: Progress

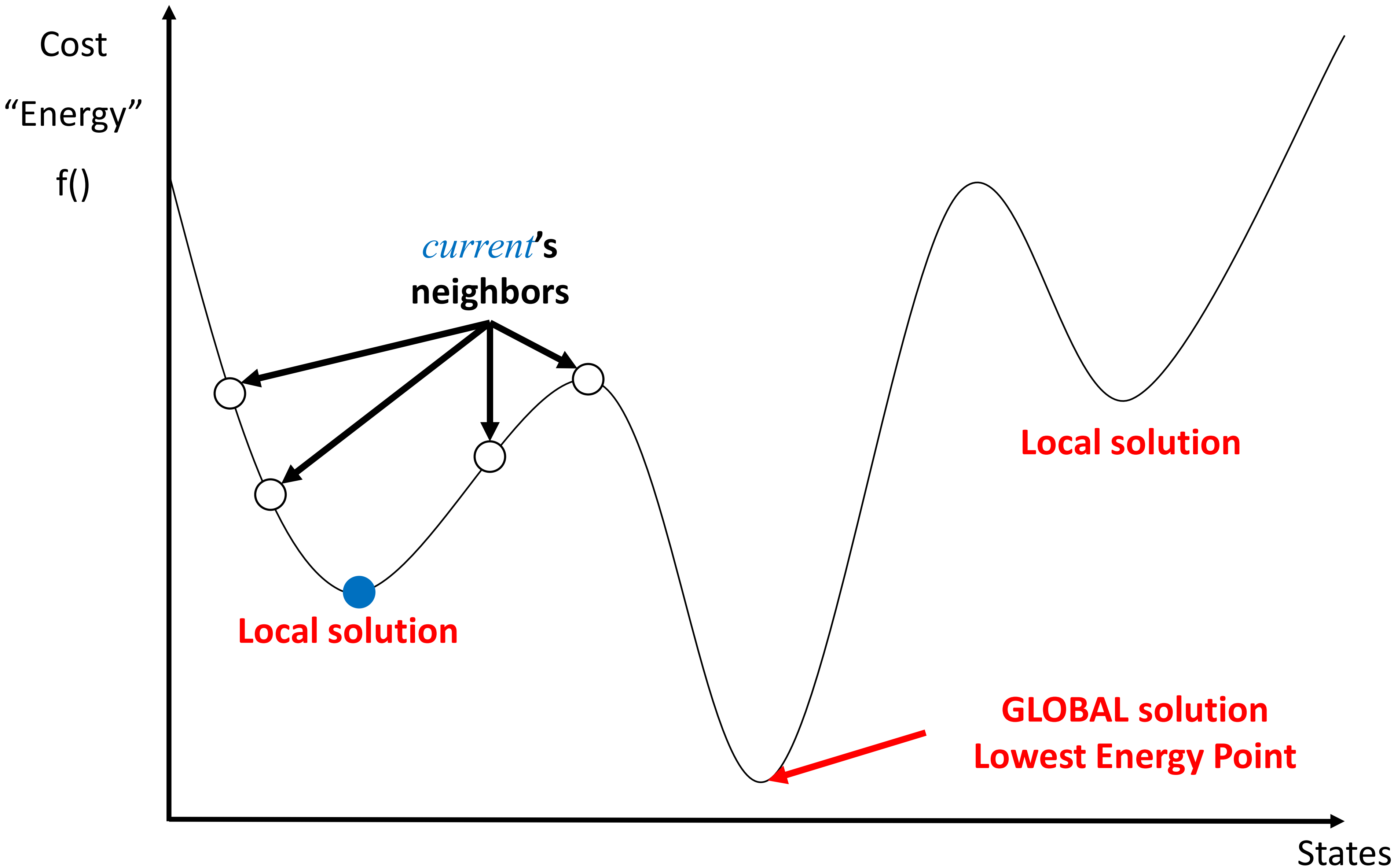
next $\leftarrow$ a randomly selected successor of *current*
$\Delta E \leftarrow \text{VALUE}(current) - \text{VALUE}(next)$
✓ **if** $\Delta E > 0$ **then** $current \leftarrow next$
else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$



Simulated Annealing: Progress

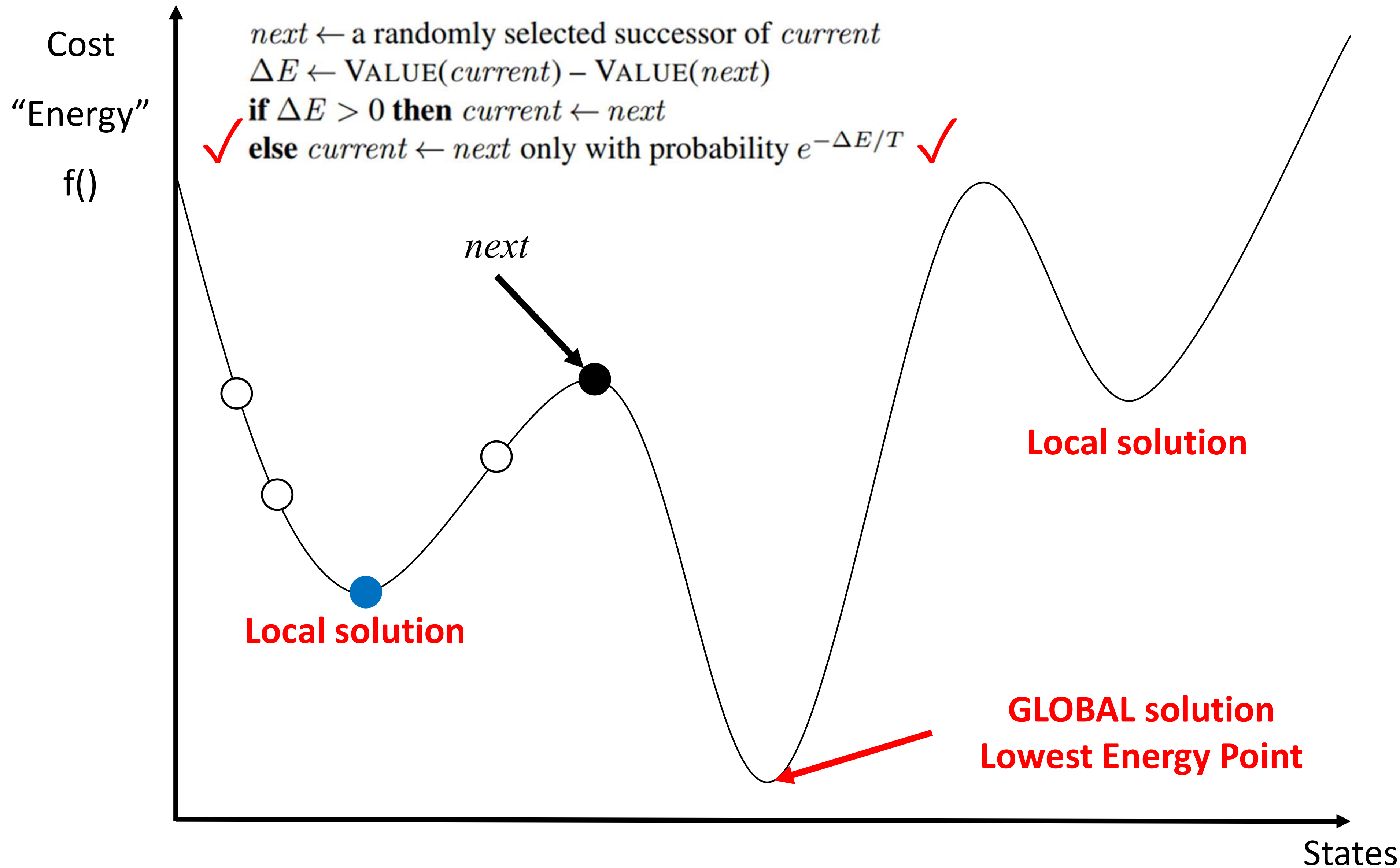


Simulated Annealing: Progress

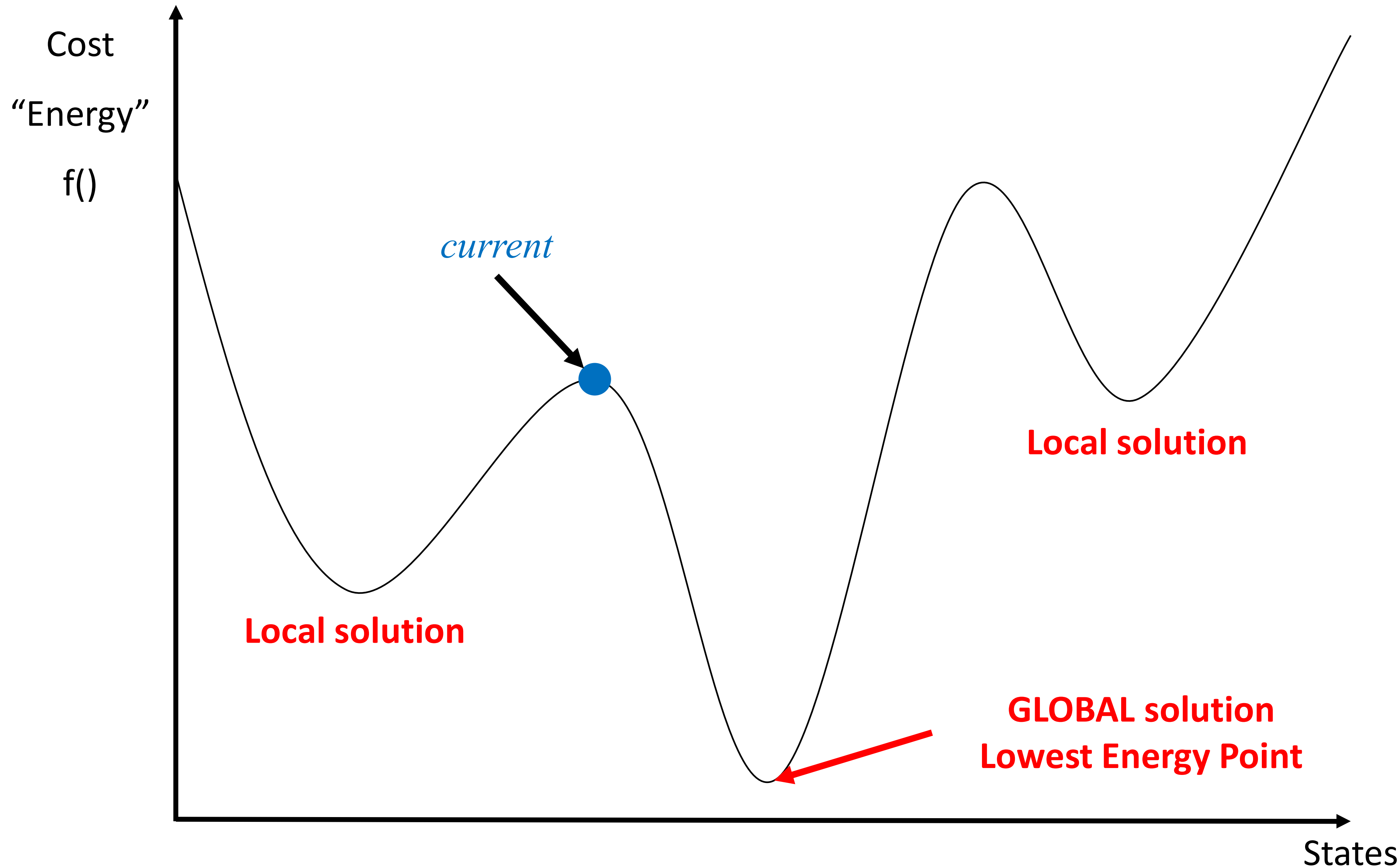


Simulated Annealing: Progress

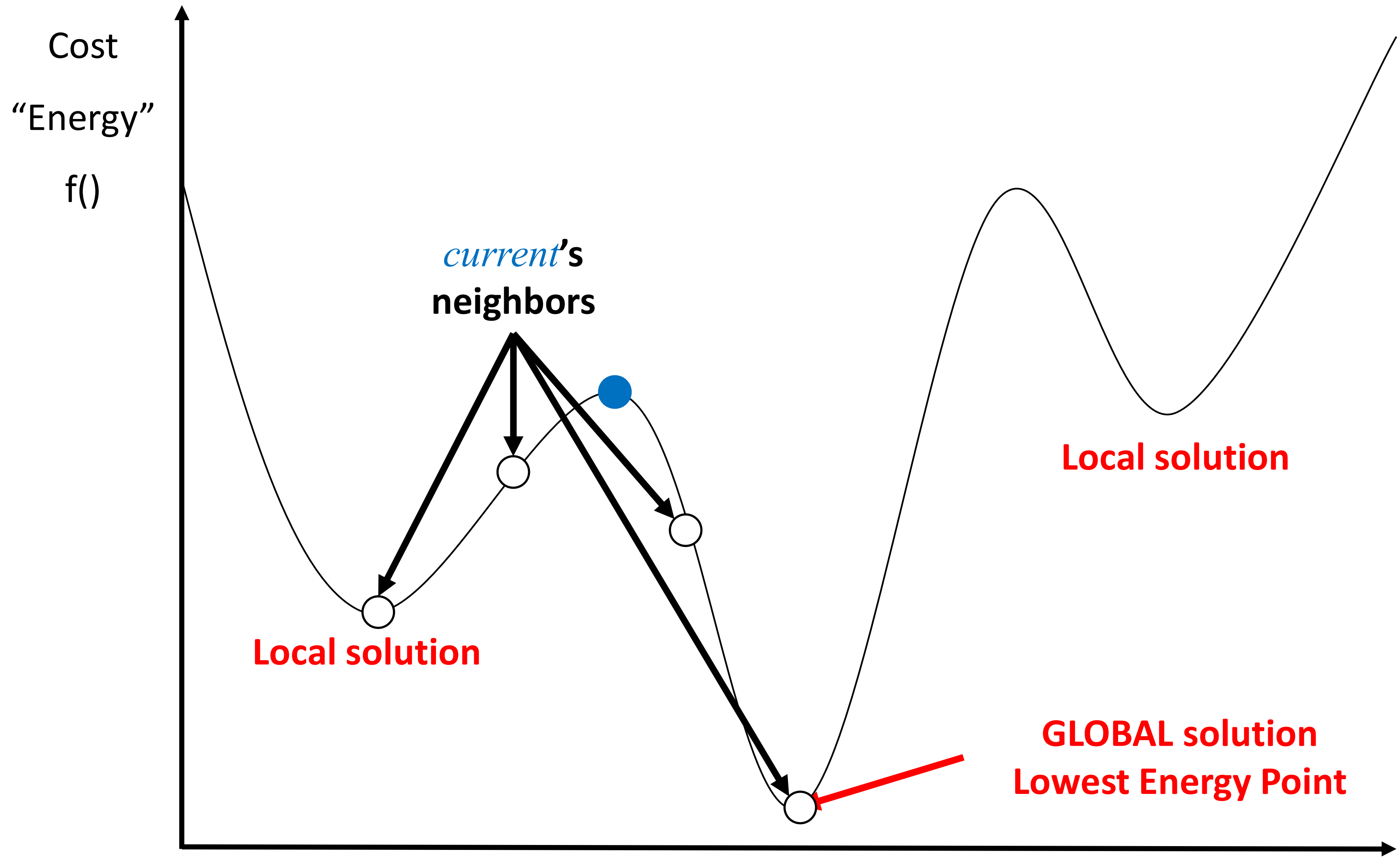
$next \leftarrow$ a randomly selected successor of $current$
$\Delta E \leftarrow \text{VALUE}(current) - \text{VALUE}(next)$
if $\Delta E > 0$ **then** $current \leftarrow next$
else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$ ✓



Simulated Annealing: Progress

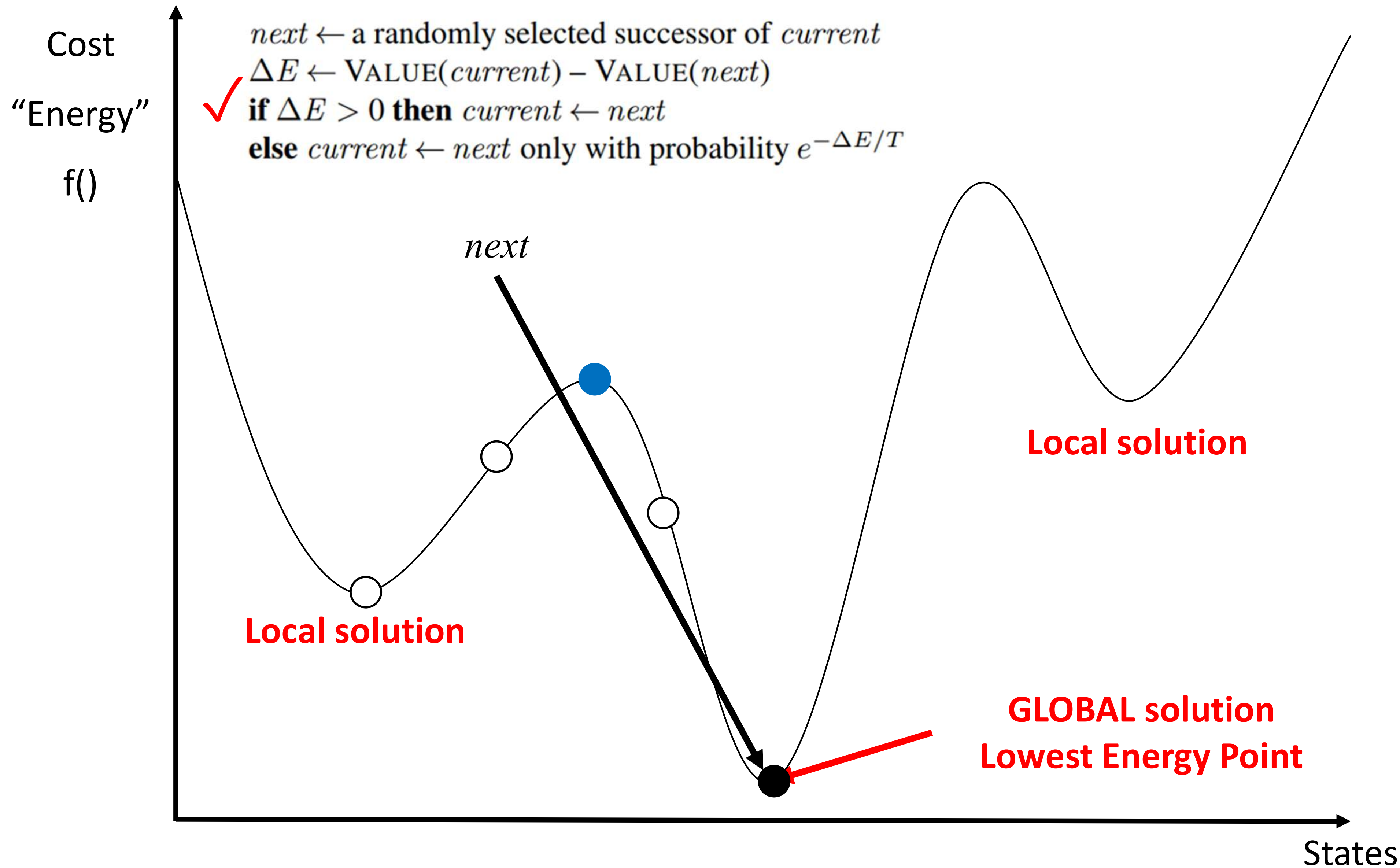


Simulated Annealing: Progress

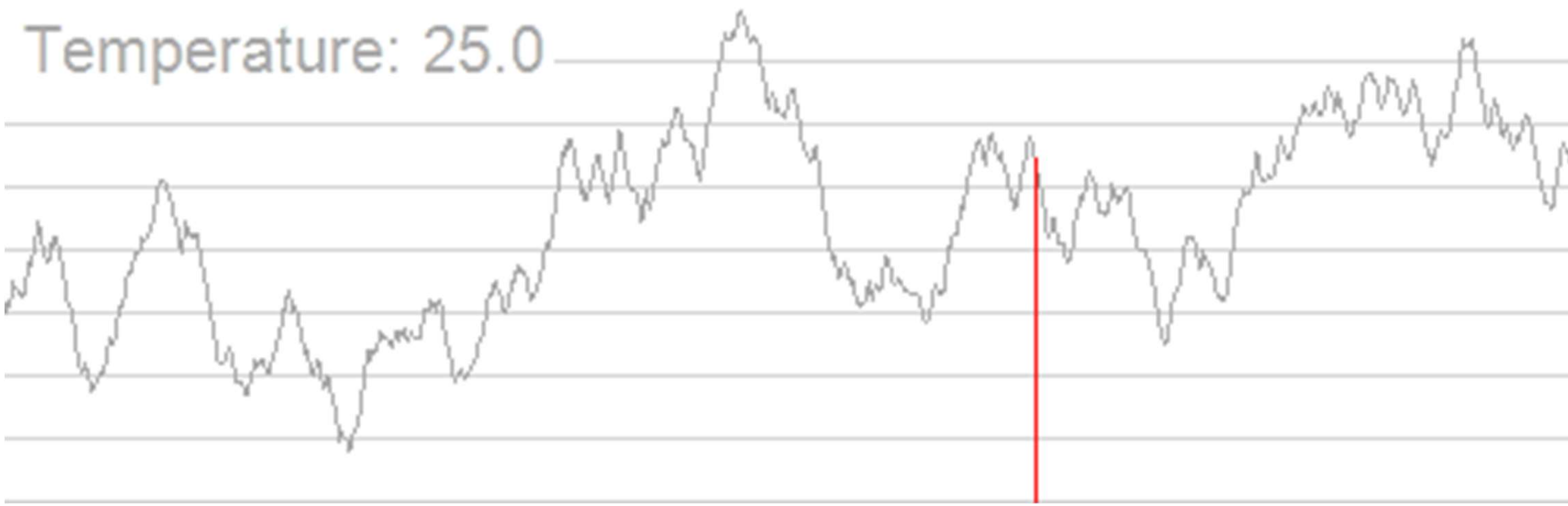


Simulated Annealing: Progress

$next \leftarrow$ a randomly selected successor of $current$
$\Delta E \leftarrow \text{VALUE}(current) - \text{VALUE}(next)$
✓ **if** $\Delta E > 0$ **then** $current \leftarrow next$
else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$



Simulated Annealing: Progress



Source: https://en.wikipedia.org/wiki/Simulated_annealing

Simulated Annealing: Summary

- Converges to a global optimum
  - connected neighborhood
  - slow cooling schedule
    - slower than the exhaustive search
- In practice
  - can give excellent results
  - need to tune a temperature schedule
  - default choice: $t_{k+1} = \alpha\, t_k$
- Additional tools
  - restarts and reheats

Simulated Annealing: Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Engineering
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory

Heuristics and Metaheuristics

- **Heuristics:**
  - how to choose the next neighbor?
  - use local information (state and its neighborhood)
  - direct the search towards a local min/maximum

- **Metaheuristics:**
  - how to escape local minima?
  - direct the search towards a global min/maximum
  - typically include some memory or learning