Chapter 4

- Beyond Classical Search: Local Search

Local search algorithms

- In many problems, the path to the goal is irrelevant. The goal state itself is the solution
- In such problems, the goal state is described implicitly by giving constraints that need to be satisfied
- Examples: 8-queens problem, TSP, Job scheduling
- There is no need then to keep track of the path → keep only a single current node and improve it.
    - Less memory.
    - Applicable in large (even infinite) spaces.
- Such algorithms are called local search algorithms

Local search algorithms

- Local search algorithms are useful for solving pure optimization problems, where we want to find the best state according to an objective function.

- Local search algorithms use a single node and try to minimize or maximize a cost (or objective) function:
    - 8-Queens: objective = number of queens under attack.
    - TSP: objective = the total travelled distance.

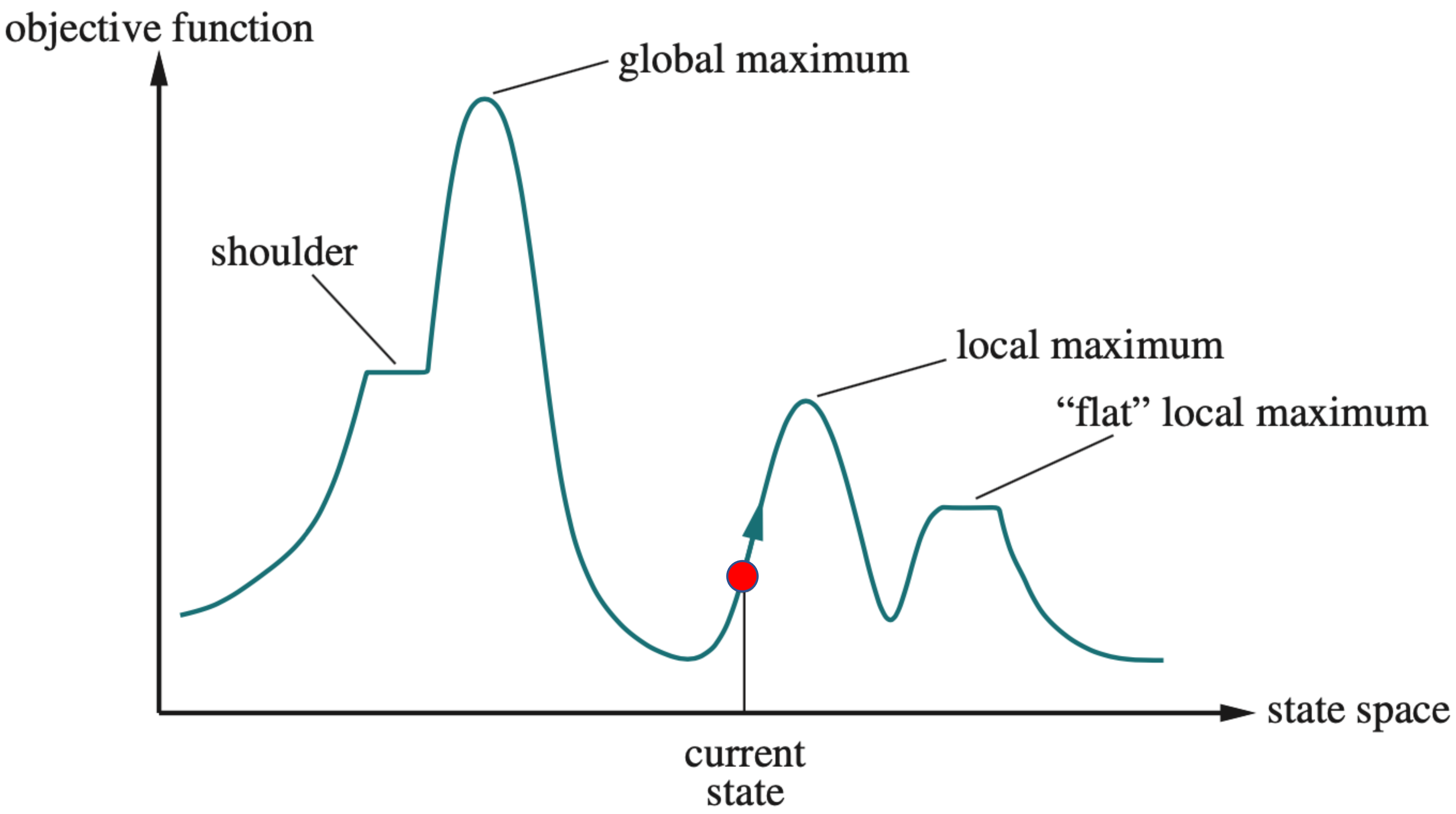
- Minimization ≡ maximization:
  - Maximizing f is equivalent to minimizing $-f$ or $constant - f$

Modeling for local search

To apply local search:

- State representation: typically use a complete-state formulation
- Initial state(s): Start with a random complete assignment (we allow for inconsistent assignments). This formulation is called Complete state formulation.
- Actions: Change the values of one variable.
- Objective function (no goal test: the algorithms search for a minimum (or the maximum) of the function).

Modeling for local search



Local Search Algorithms

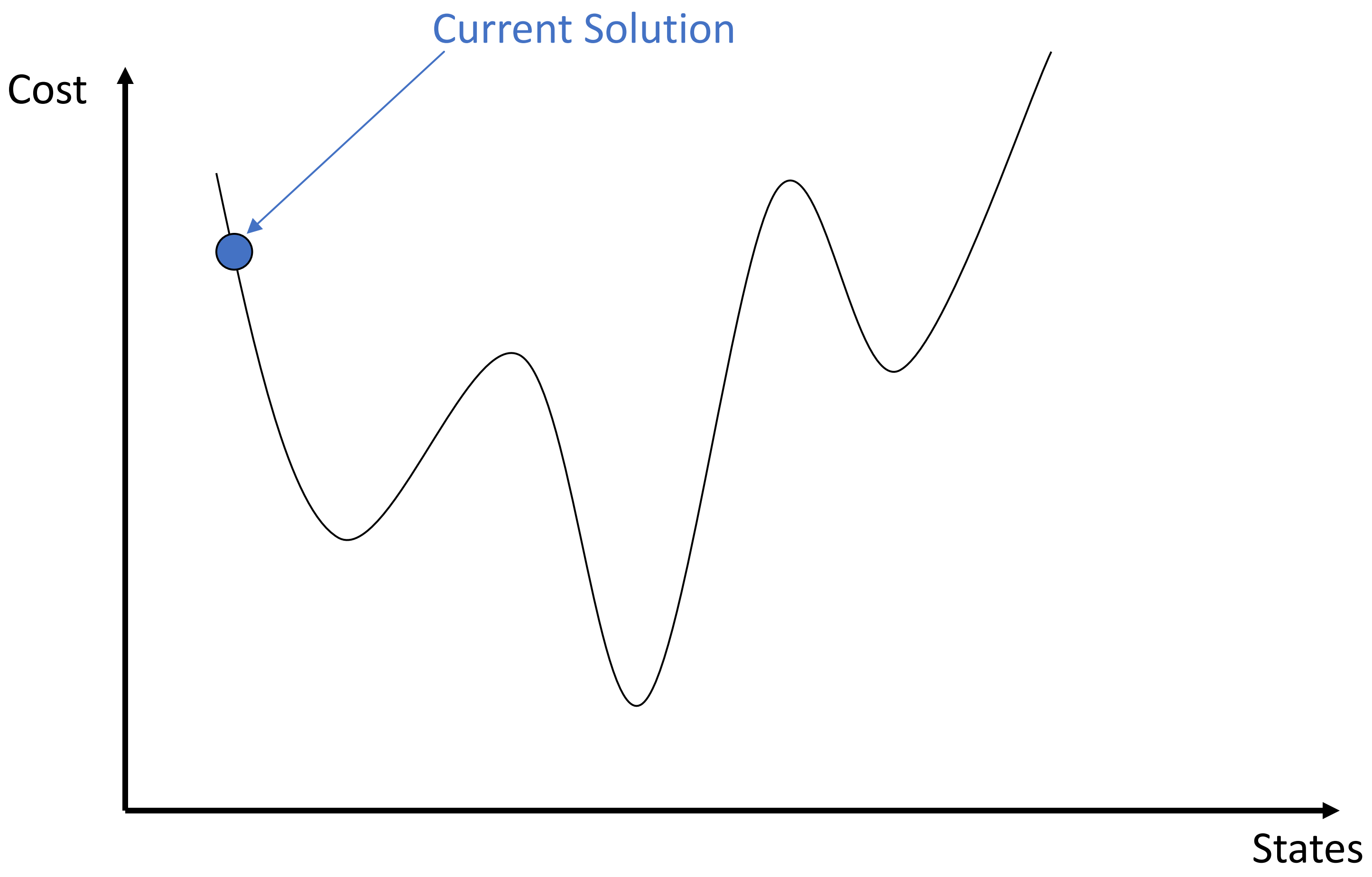
1. Hill Climbing
2. Simulated Annealing
3. Beam Search
4. Genetic Algorithm

1. Hill-climbing

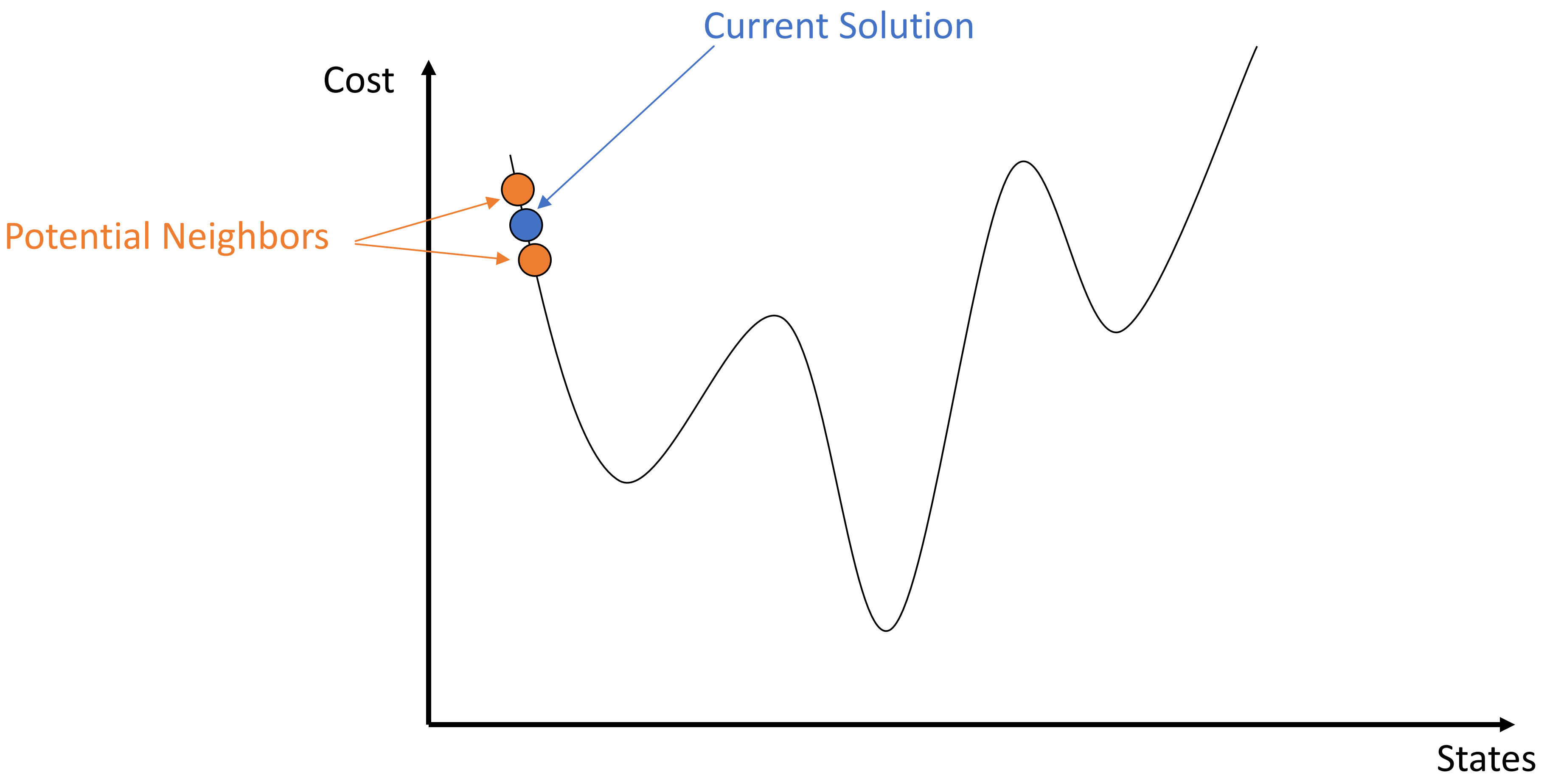
- Move to the neighbor with the best value
- Best value depends on minimization or maximization
- Keeps only node: state + value of the objective function

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

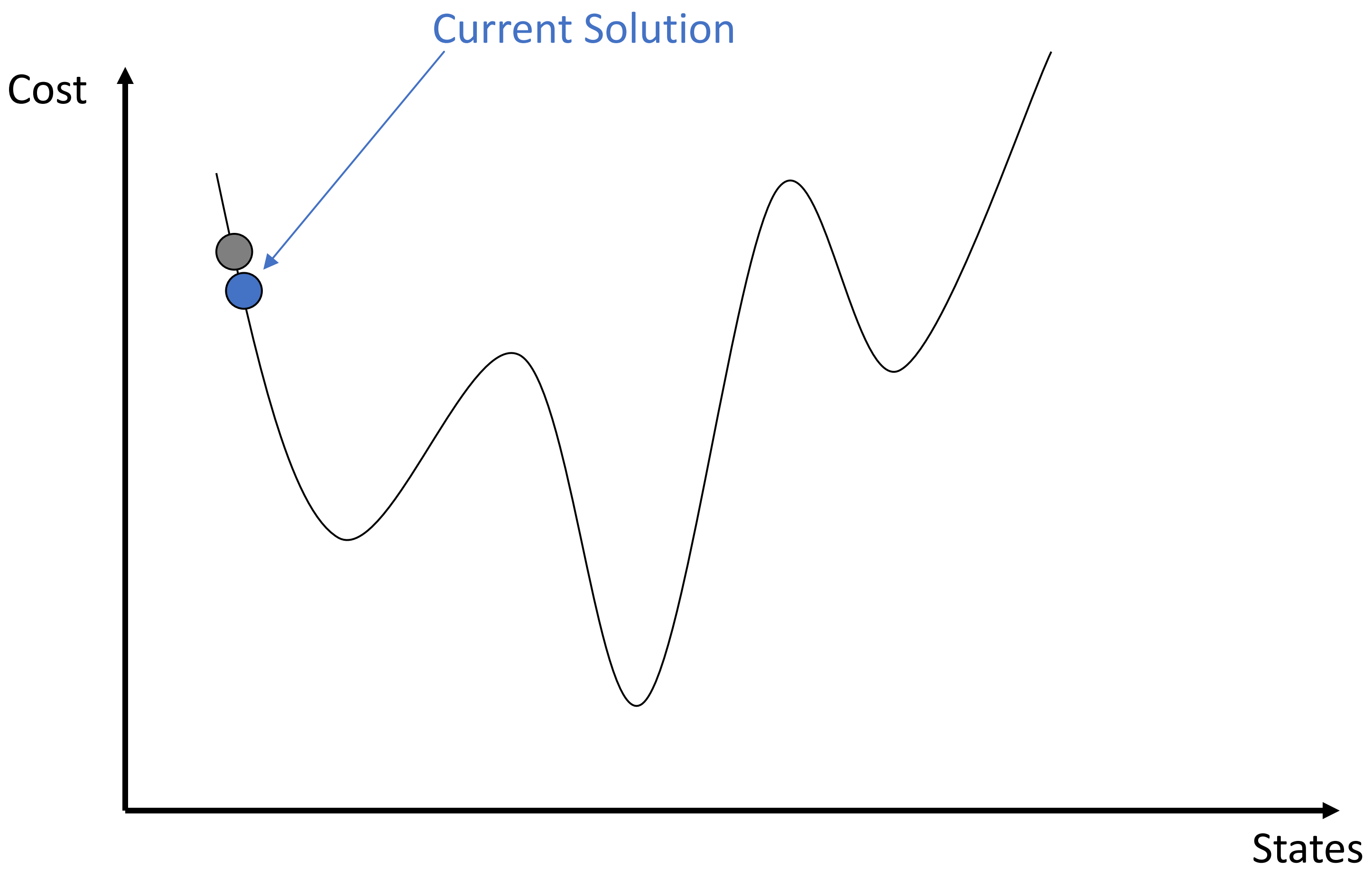
1. Hill-climbing



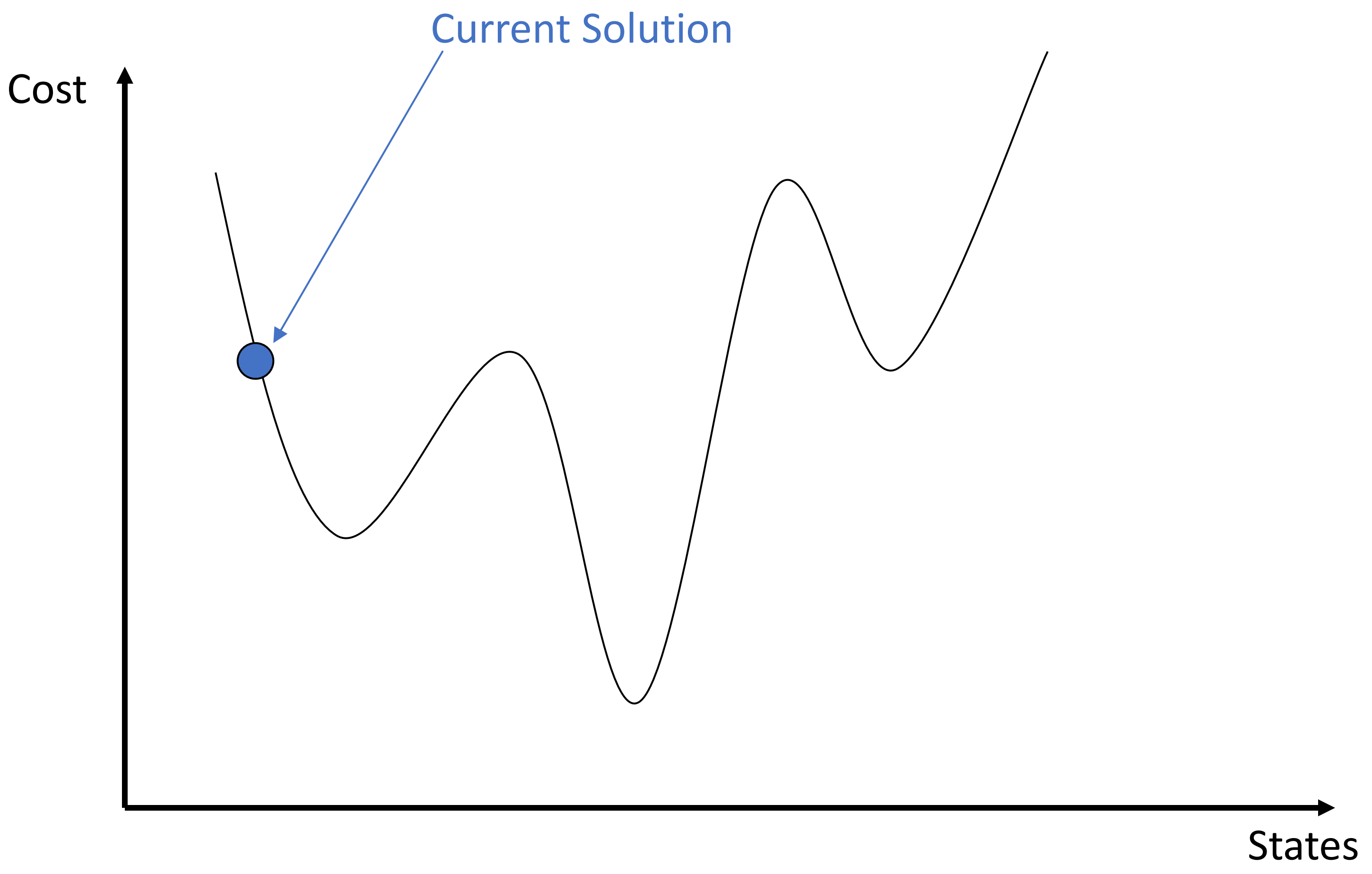
1. Hill-climbing



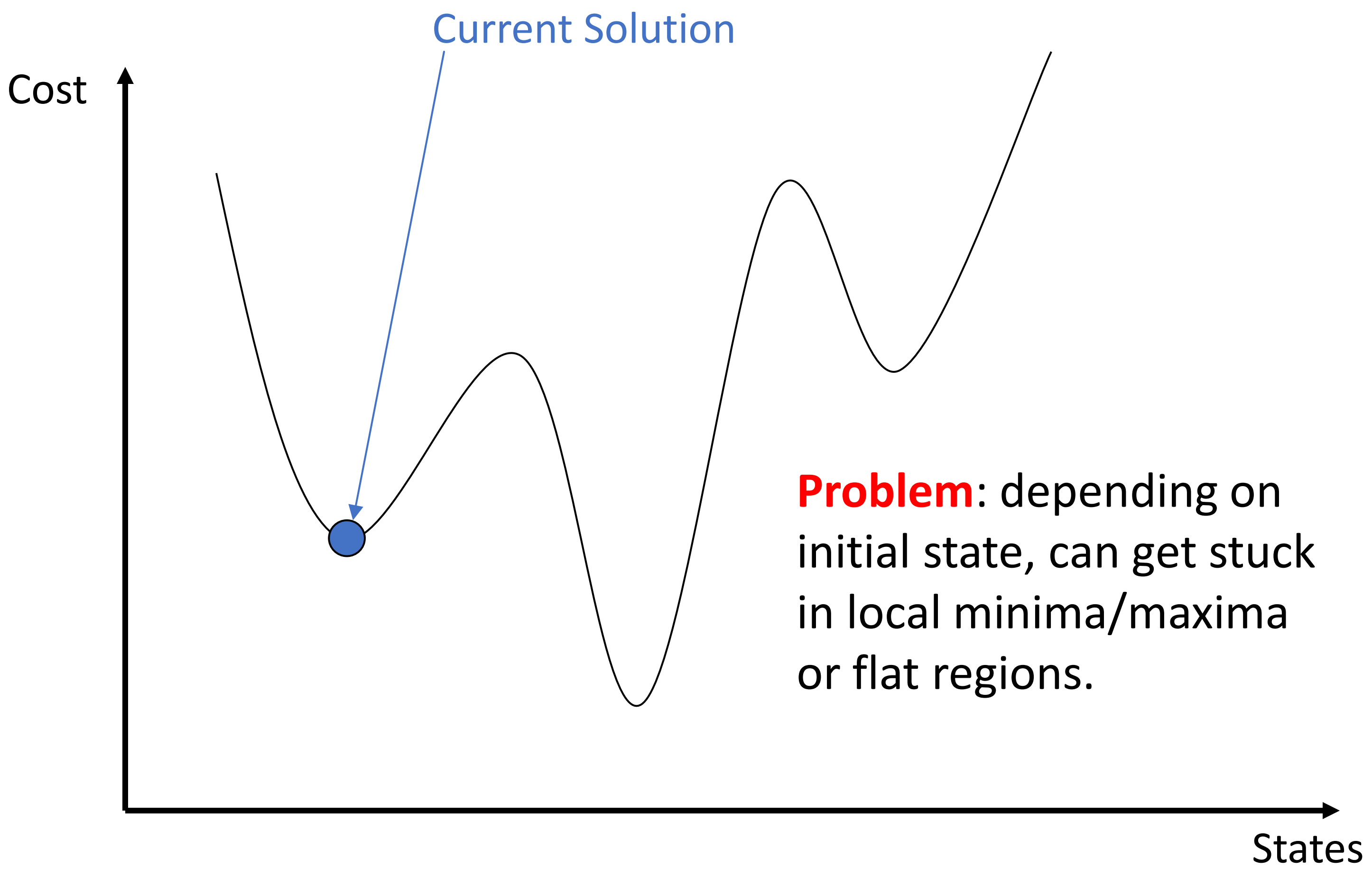
1. Hill-climbing



1. Hill-climbing

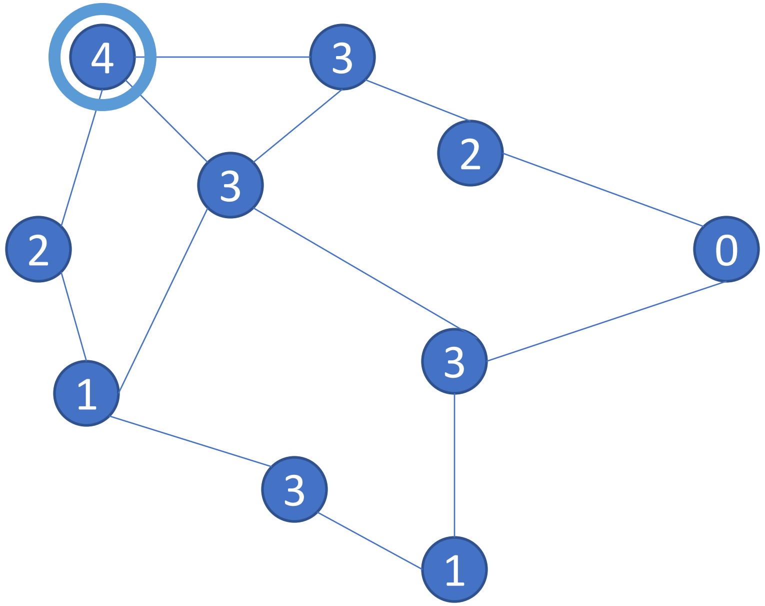


1. Hill-climbing

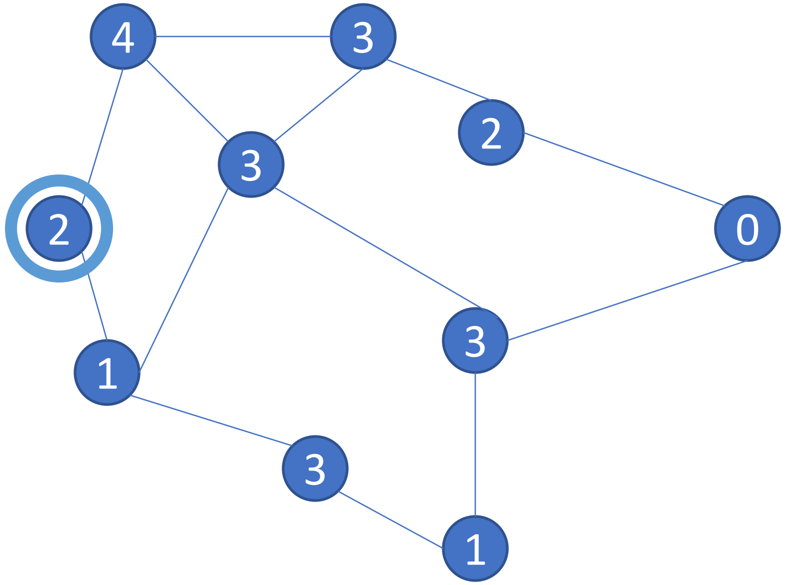


Problem: depending on initial state, can get stuck in local minima/maxima or flat regions.

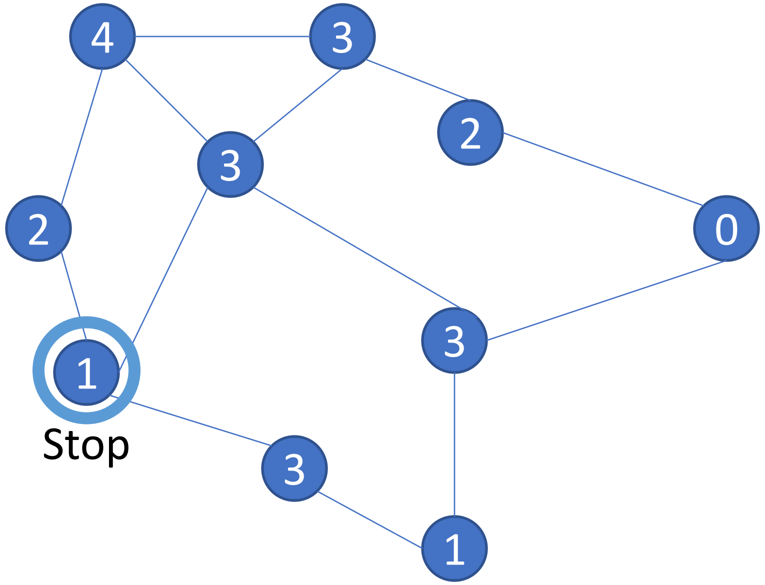
1. Hill-Climbing



1. Hill-Climbing

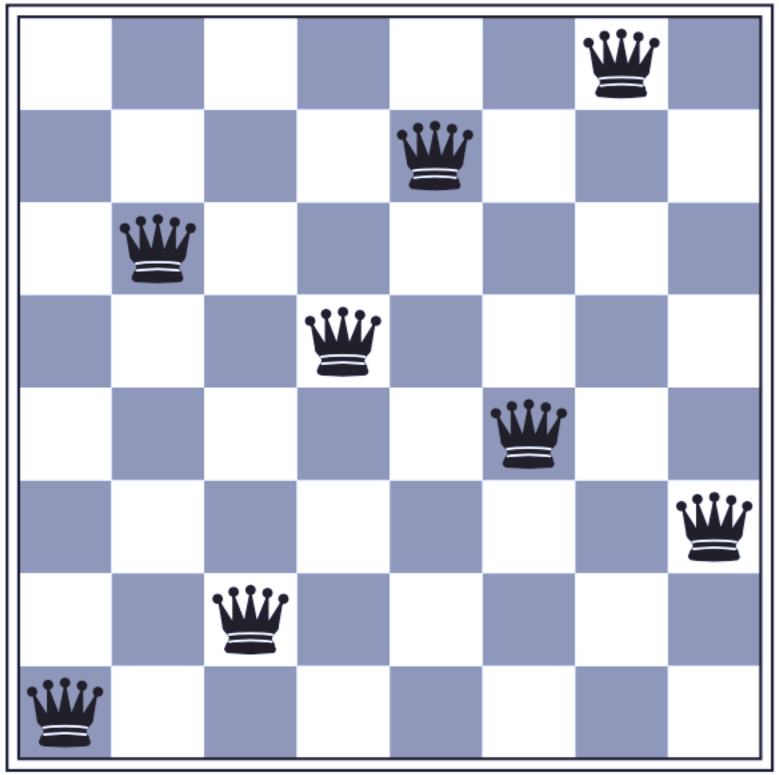


1. Hill-Climbing



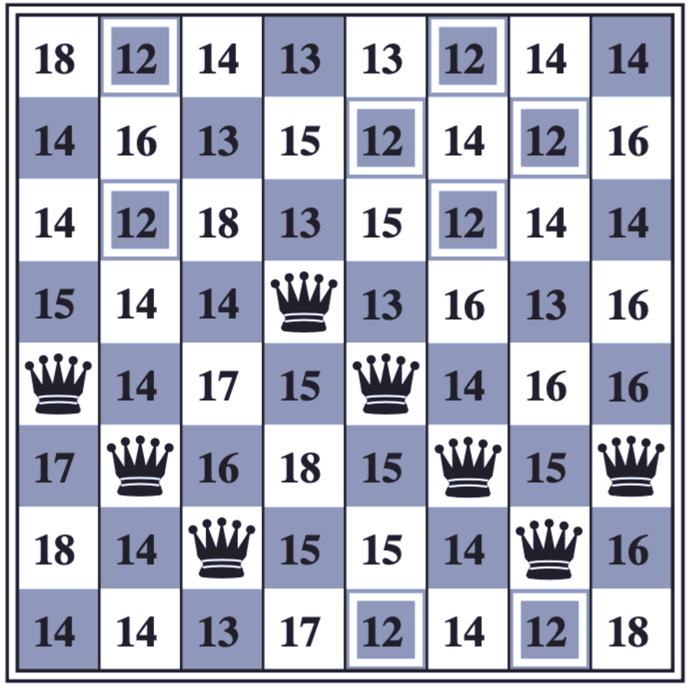
1. Hill-climbing: 8-queens problem

- Each state has 8 queens on the board, one per column.
- The successors of a state are all possible states generated by moving a single queen to another square in the same column
  - each state has 8 × 7 = 56 successors



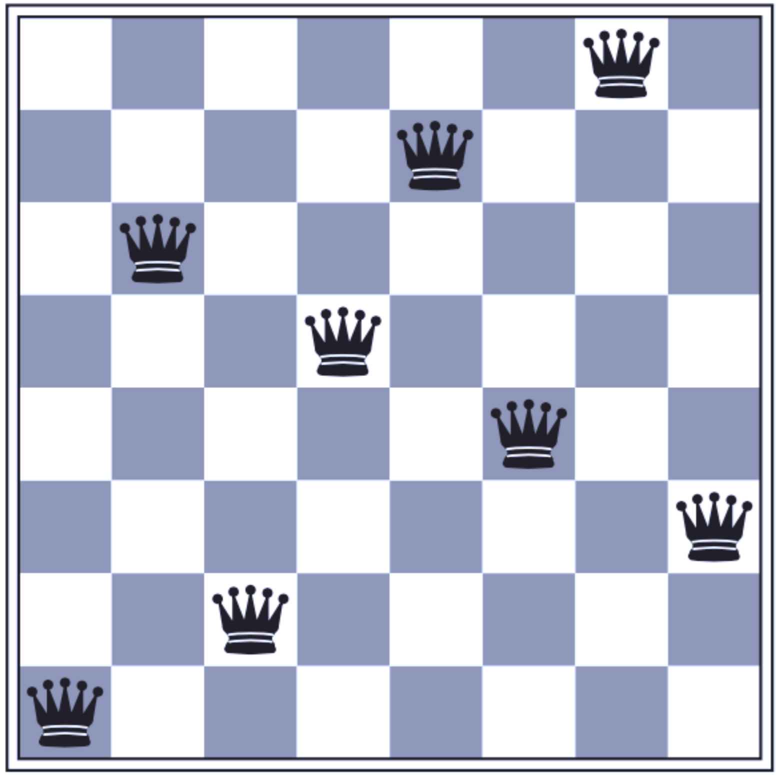
1. Hill-climbing: 8-queens problem

- h = number of pairs of queens that are attacking each other, either directly or indirectly
- $h\ =\ 17$ for the state
- Best is $h = 12$, hill climbing chooses randomly between successors set



1. Hill-climbing

- Known as greedy local search
- Can perform quite well, takes 5 steps to reach this state ($h = 1$) from previous state
- Gets stuck at local optima, every move after the current move is worse
- For a randomly generated 8-queens state, steepest-ascent hill climbing gets stuck 86% of the time
- works quickly, taking just 4 steps on average when it succeeds and 3 when it gets stuck



1. Hill-climbing

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
current ← *problem*.INITIAL
while *true* **do**
neighbor ← a highest-valued successor state of *current*
if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
current ← *neighbor*

What if we can move sideways?
i.e. remove the equal sign

1. Hill-climbing

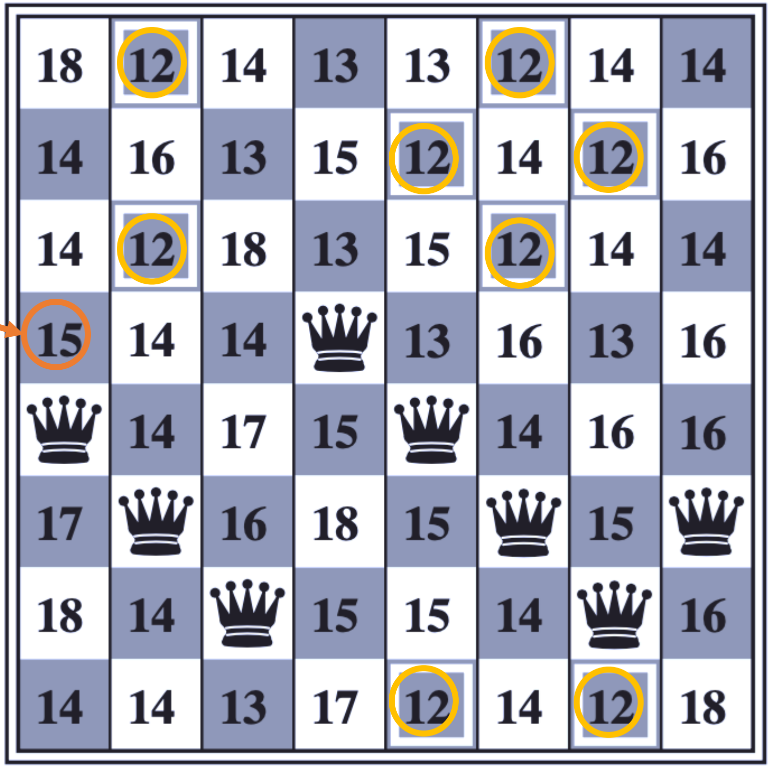
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

What if we can move sideways?
i.e. remove the equal sign

- Might get stuck in an infinite loop, better to use a limit
- Example: 8-queens problem allow 100 consecutive sideways moves
  - Problem instances solved 94%
  - Averages 21 steps for each successful instance and 64 for each failure

1. Hill-climbing variations

- Simple hill climbing: chooses the first uphill generated, which might not be the best move
  - Might pick 15, if it was generated first
- Steepest ascent hill climbing: looks at all the neighbors and then picks the best valued successor
  - Will always pick one of the 12's



1. Hill-climbing variations

- Stochastic hill climbing: chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move
- Incomplete: A hill-climbing algorithm that *never* makes “downhill” moves toward states with lower value (or higher cost) is guaranteed to be incomplete, because it can get stuck on a local maximum
- Complete version: Random-restart hill climbing: conduct a series of hill-climbing searches from randomly generated initial states until a goal is found

2. Simulated annealing

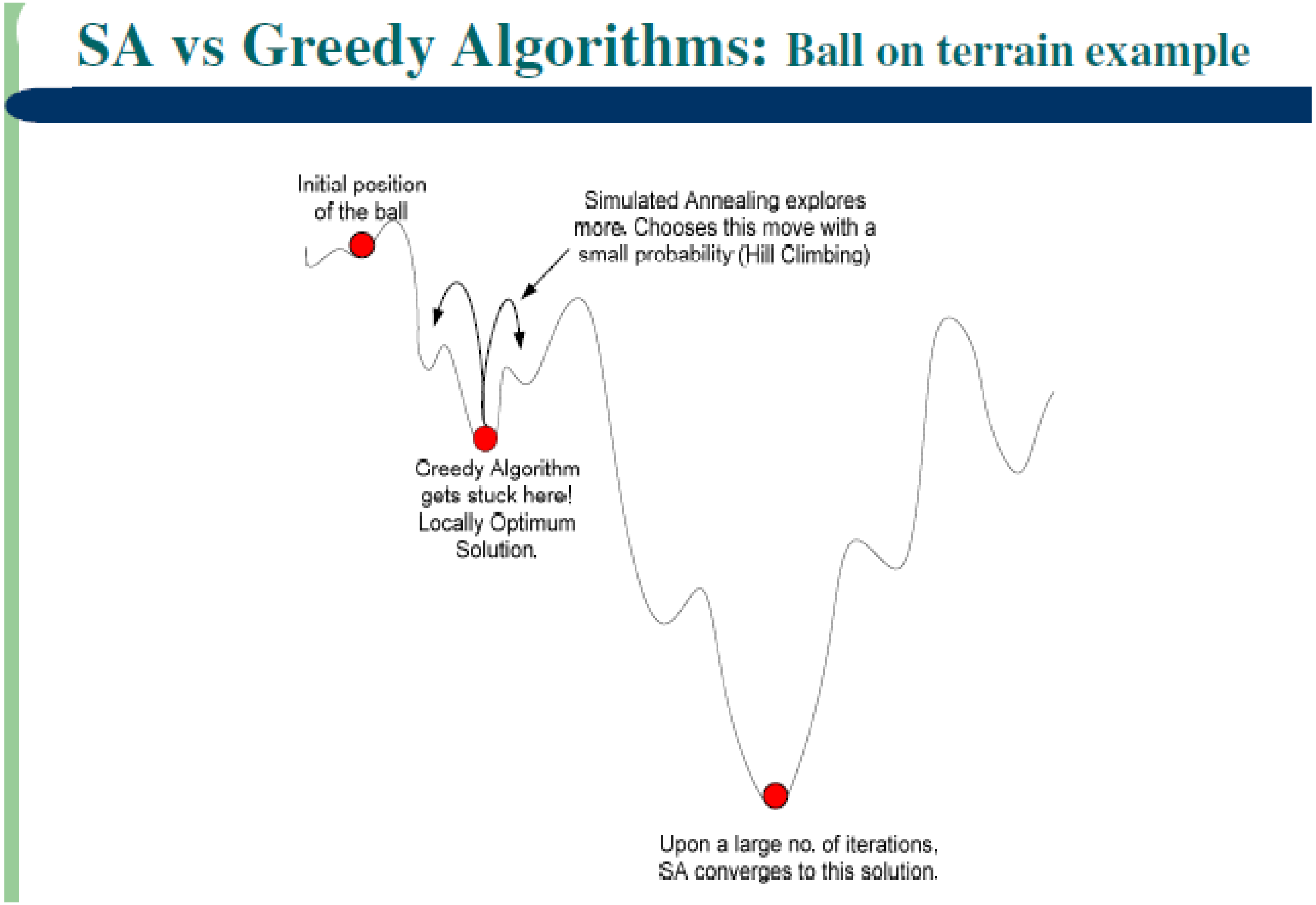
- Random walk: move to a successor chosen uniformly at random from the set of successors
    - complete but extremely inefficient ☹
- What if we combine hill climbing with a random walk in some way that yields both efficiency and completeness?

2. Simulated annealing

What Is Simulated Annealing?

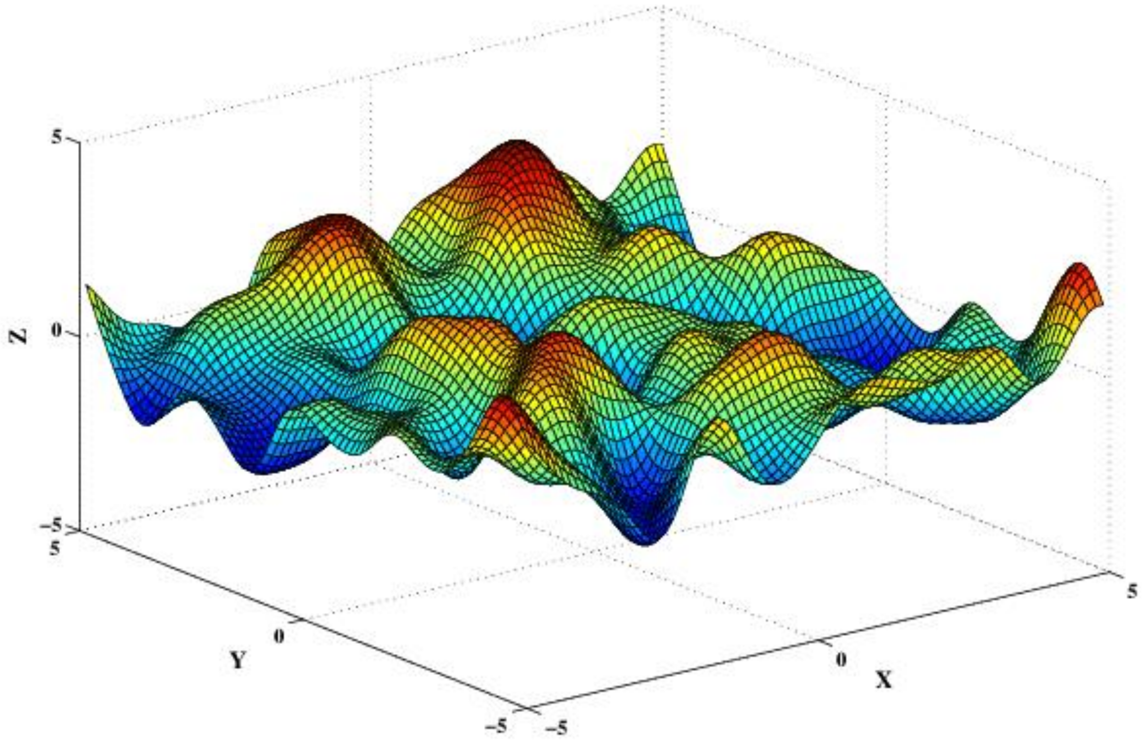
- Simulated Annealing (SA)
  - SA is applied to solve optimization problems
  - SA is a stochastic algorithm
  - SA is escaping from local optima by allowing worsening moves
  - SA is a memoryless algorithm, the algorithm does not use any information gathered during the search
  - SA is applied for both combinatorial and continuous optimization problems
  - SA is simple and easy to implement.
  - SA is motivated by the physical annealing process

2. Simulated annealing



2. Simulated Annealing: IDEA

- Shake the surface: get to a local minimum.
- Shake just hard enough to bounce the ball out of local minima but not hard enough to dislodge it from the global minimum
- Simulated-annealing starts by shaking hard (at a high temperature) and then slowly reduce the intensity of the shaking (lower the temperature).



2. Simulated annealing

Real Annealing Technique

- **Annealing Technique** is known as a thermal process for **obtaining low-energy state** of a solid in a heat bath.
- The process consists of the following two steps:
  - **Increasing temperature**: Increase the **temperature** of the heat bath to a maximum value at which the solid melts.
  - **Decreasing temperature:** Decrease carefully the temperature of the heat bath until the **particles** arrange themselves in the **ground state** of the solid.

2. Simulated annealing

Real Annealing Technique

- In the **liquid phase** all **particles** arrange themselves randomly, whereas in the ground state of the solid, the particles are arranged in a highly structured lattice, for which the corresponding energy is minimal.
- The **ground state** of the solid is obtained only if:
  - the maximum value of the temperature is sufficiently high and
  - the cooling is done sufficiently slow.
- Strong solid are grown from careful and slow cooling.

2. Simulated annealing

Real Annealing Technique

- **Metastable states**
  - If the initial temperature is not sufficiently high or a fast cooling is applied, **metastable states** (imperfections) are obtained.
- **Quenching**
  - The process that leads to metastable states is called **quenching**
- **Thermal equilibrium**
  - If the **lowering of the temperature** is done sufficiently slow, the solid can reach **thermal equilibrium** at each temperature.

2. Simulated annealing

Real Annealing and Simulated Annealing

- The analogy between the physical system and the optimization problem.

Physical System		Optimization Problem
System state	⟺	Solution
Molecular positions	⟺	Decision variables
Energy	⟺	Objective function
Minimizing energy	⟺	Minimizing cost
Ground state	⟺	Global optimal solution
Metastable state	⟺	Local optimum
Quenching	⟺	Local search
Temperature	⟺	Control parameter T
Real annealing	⟺	Simulated annealing

2. Simulated annealing

Real Annealing and Simulated Annealing

- The objective function of the problem is analogous to the energy state of the system.
- A solution of the optimization problem corresponds to a system state.
- The decision variables associated with a solution of the problem are analogous to the molecular positions.
- The global optimum corresponds to the ground state of the system.
- Finding a local minimum implies that a metastable state has been reached.

2. Simulated annealing

- Idea: escape local minima by allowing some "bad" moves but gradually decrease their frequency.

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

   current ← MAKE-NODE(problem.INITIAL-STATE)
   for t = 1 to ∞ do
       T ← schedule(t)
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← next.VALUE – current.VALUE
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(ΔE/T)
```

Pick a random, not best, move

2. Simulated annealing

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

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

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

Always accept better moves

2. Simulated annealing

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

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

   current ← MAKE-NODE(problem.INITIAL-STATE)
   for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE – current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

Accept bad moves with a probability

2. Simulated annealing

How do we move with probability?

- Compute the probability $p = \; e^{negative\,number/T}$
- Generate a uniform random number $x \in [0,1]$
  - If $(x \leq p)$: move
  - If $(x > p)$: do not move

2. Simulated annealing

Template of SA

- At high temperature, $e^{\frac{-\Delta E}{T}}$ is close to 1,
  - therefore the majority of the moves are accepted and the algorithm becomes equivalent to a simple random walk in the configuration space .
- At low temperature, $e^{\frac{-\Delta E}{T}}$ is close to 0,
  - therefore the majority of the moves increasing energy is refused.
- At an intermediate temperature,
  - the algorithm intermittently authorizes the transformations that degrade the objective function

2. Simulated annealing

Template of SA

- From an initial solution, SA proceeds in several iterations.
- At each iteration, a random neighbor is generated.
- Moves that improve the cost function are always accepted.
- Otherwise, the neighbor is selected with a given probability that depends on the current temperature and the amount of degradation ΔE of the objective function.
- ΔE represents the difference in the objective value (energy) between the current solution and the generated neighboring solution.

2. Simulated annealing

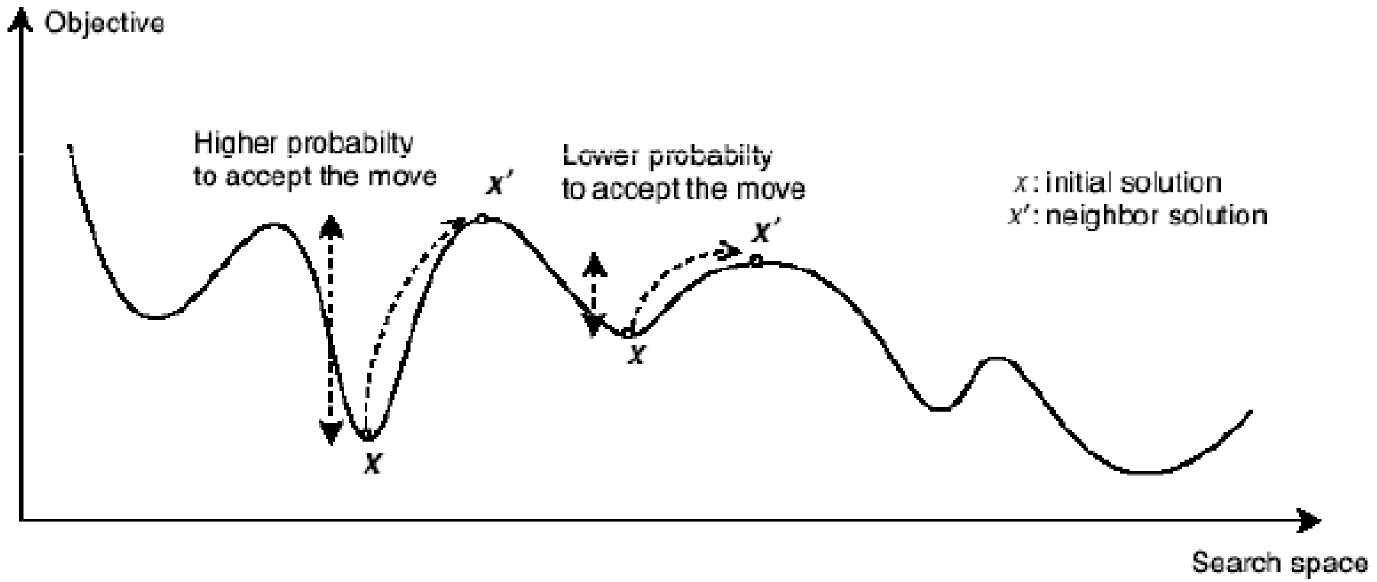
Template of SA

- The higher the temperature, the more significant the probability of accepting a worst move.
- At a given temperature, the lower the increase of the objective function, the more significant the probability of accepting the move.

2. Simulated annealing

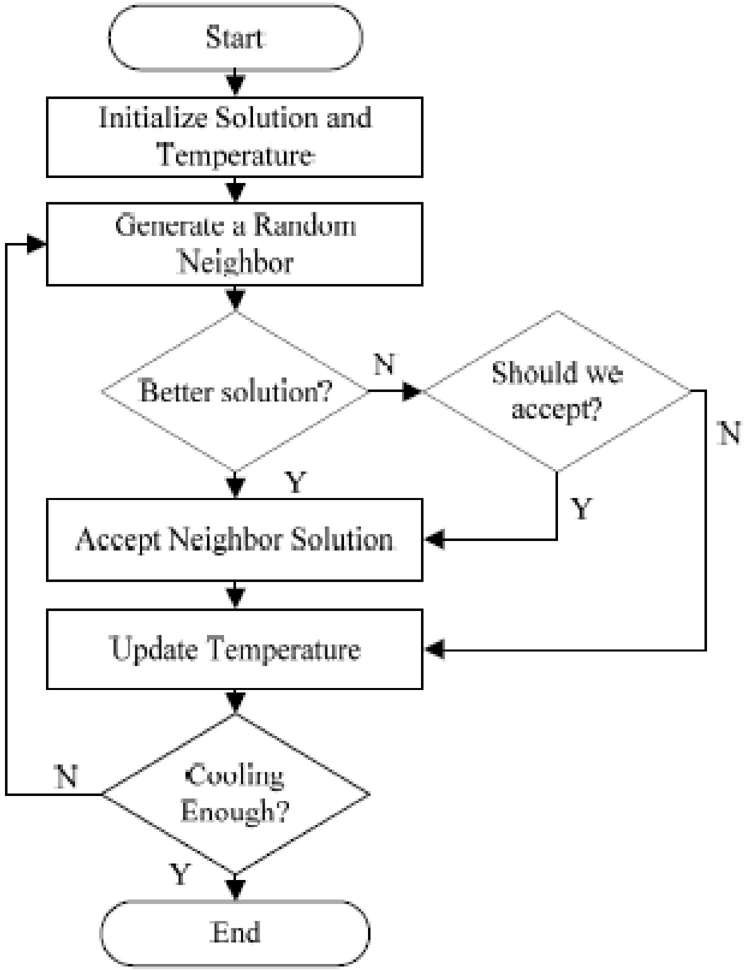
Template of SA

- As the algorithm progresses, the probability that such moves are accepted decreases.



2. Simulated annealing

Inhomogeneous variant



Input: Cooling schedule.
$s = s_0$; /* Generation of the initial solution */
$T = T_{max}$; /* Starting temperature */
Repeat
  Generate a random neighbor s' ;
  $\Delta E = f(s') - f(s)$;
  If $\Delta E \leq 0$ **Then** $s = s'$ /* Accept the neighbor solution */
  Else Accept s' with a probability $e^{\frac{-\Delta E}{T}}$;
  $T = g(T)$; /* Temperature update */
Until Stopping criteria satisfied /* e.g. $T < T_{min}$ */
Output: Best solution found.

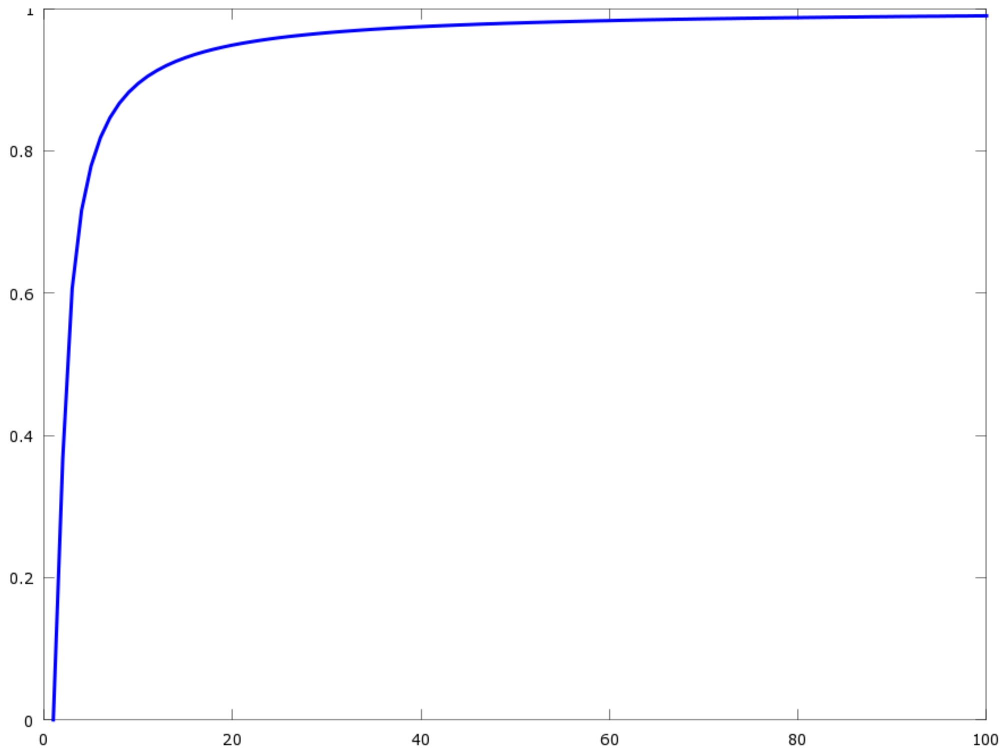
2. Simulated annealing

	Bad move		Lesser but still bad move	
T	ΔE	$p = e^{\Delta E / T}$	ΔE	$p = e^{\Delta E / T}$
1000	-100	0.9048374180359595	-50	0.951229424500714
999	-100	0.9047468482529377	-50	0.9511818166118072
…..				
700	-100	0.8668778997501816	-50	0.9310627797040227
…..				
500	-100	0.8187307530779818	-50	0.9048374180359595
…..				
200	-100	0.6065306597126334	-50	0.7788007830714
…..				
100	-100	0.36787944117144233	-50	0.6065306597126334
…..				
10	-100	0.0000453999297624849	-50	0.006737946999085469

2. Simulated annealing

Variation of the probability as a function of T when ΔE is fixed

$e^{negative\ number/T}$

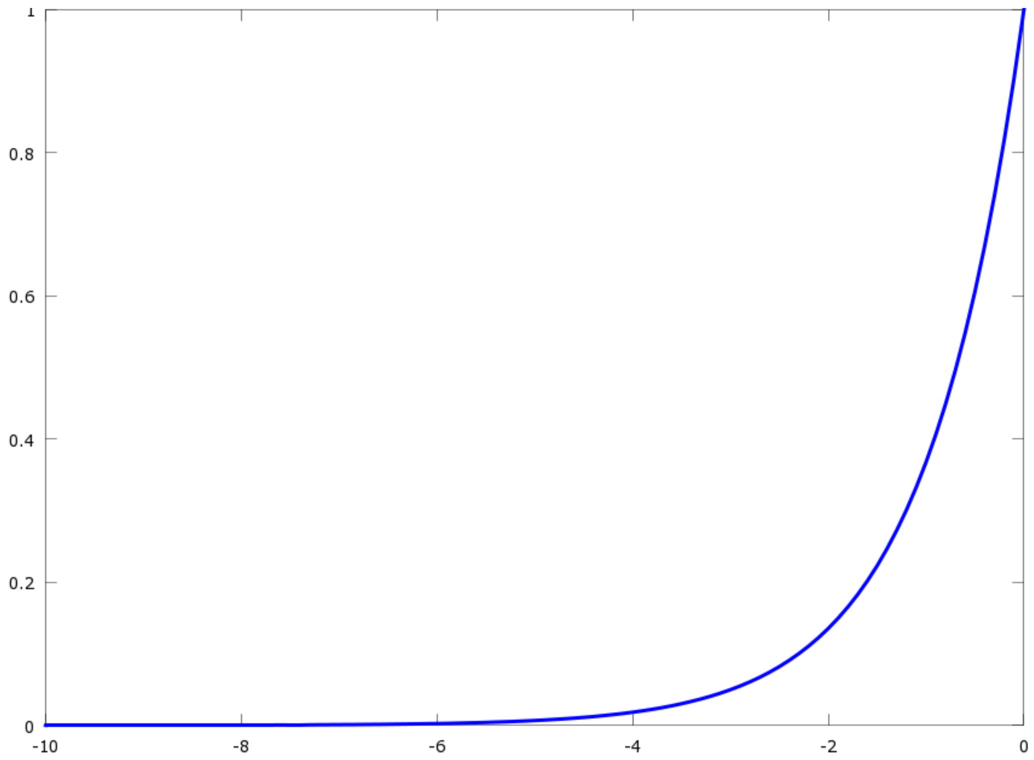


T

2. Simulated annealing

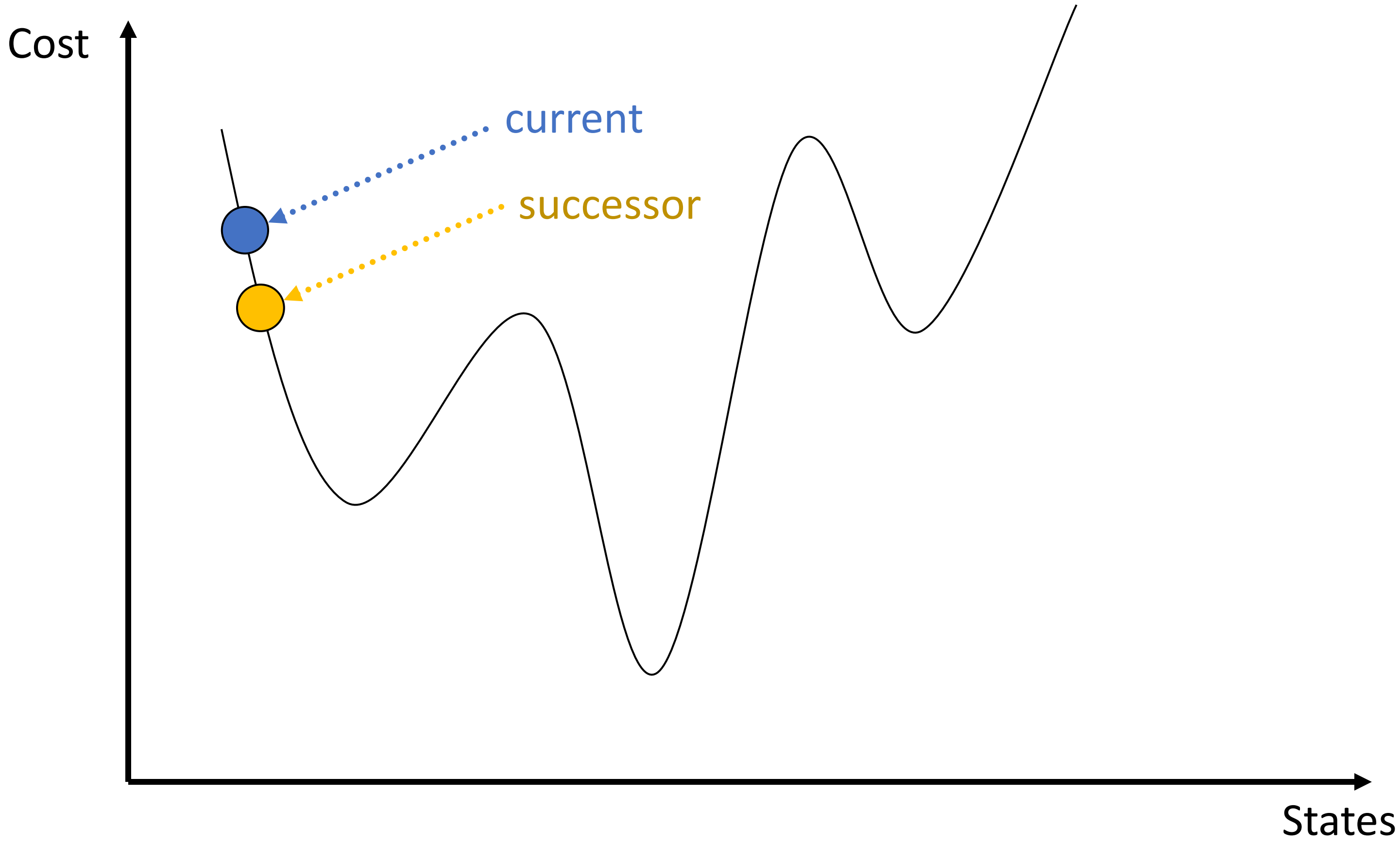
Variation of the probability as a function of ΔE when T is fixed

$e^{negative\ number/T}$

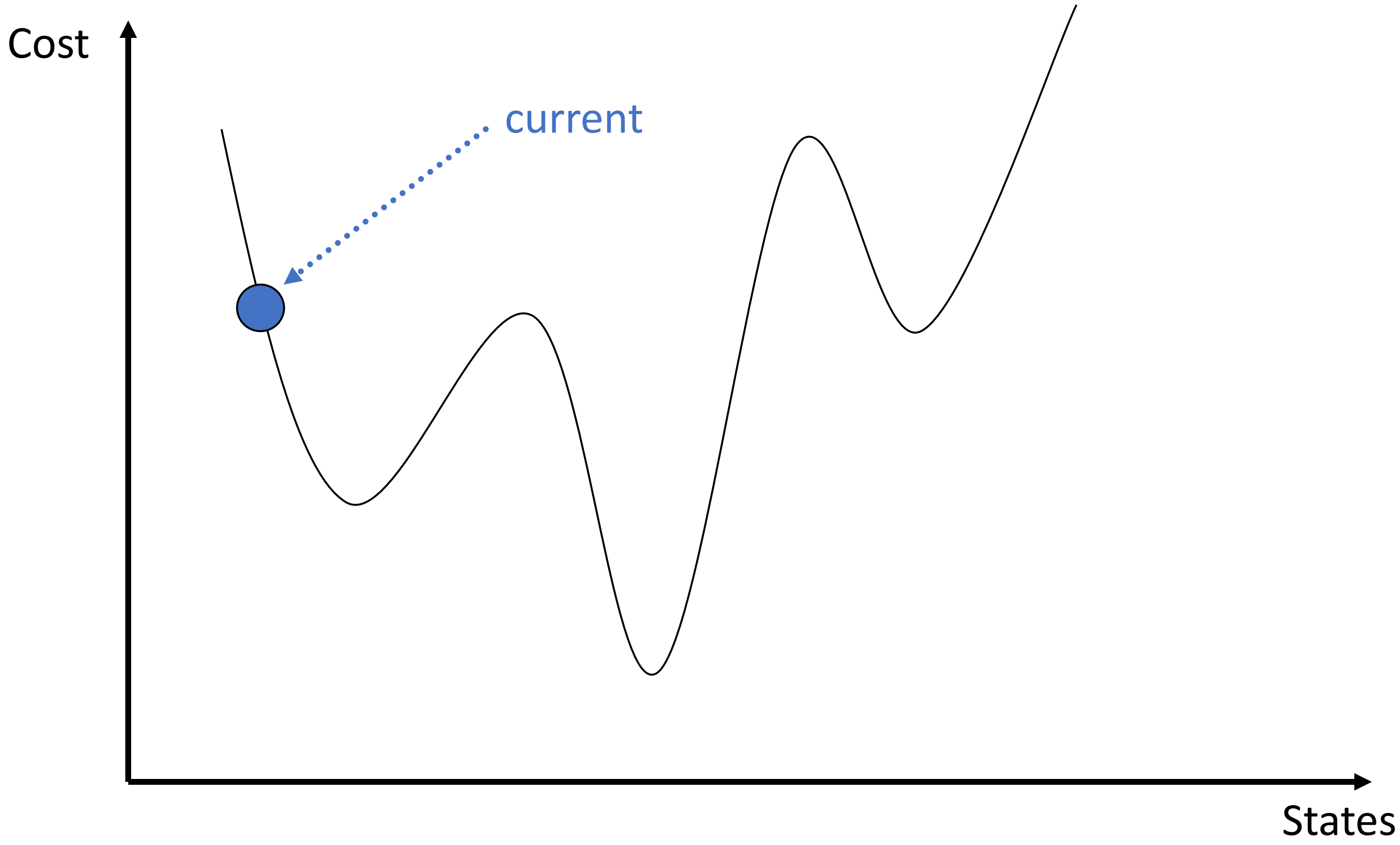


ΔE

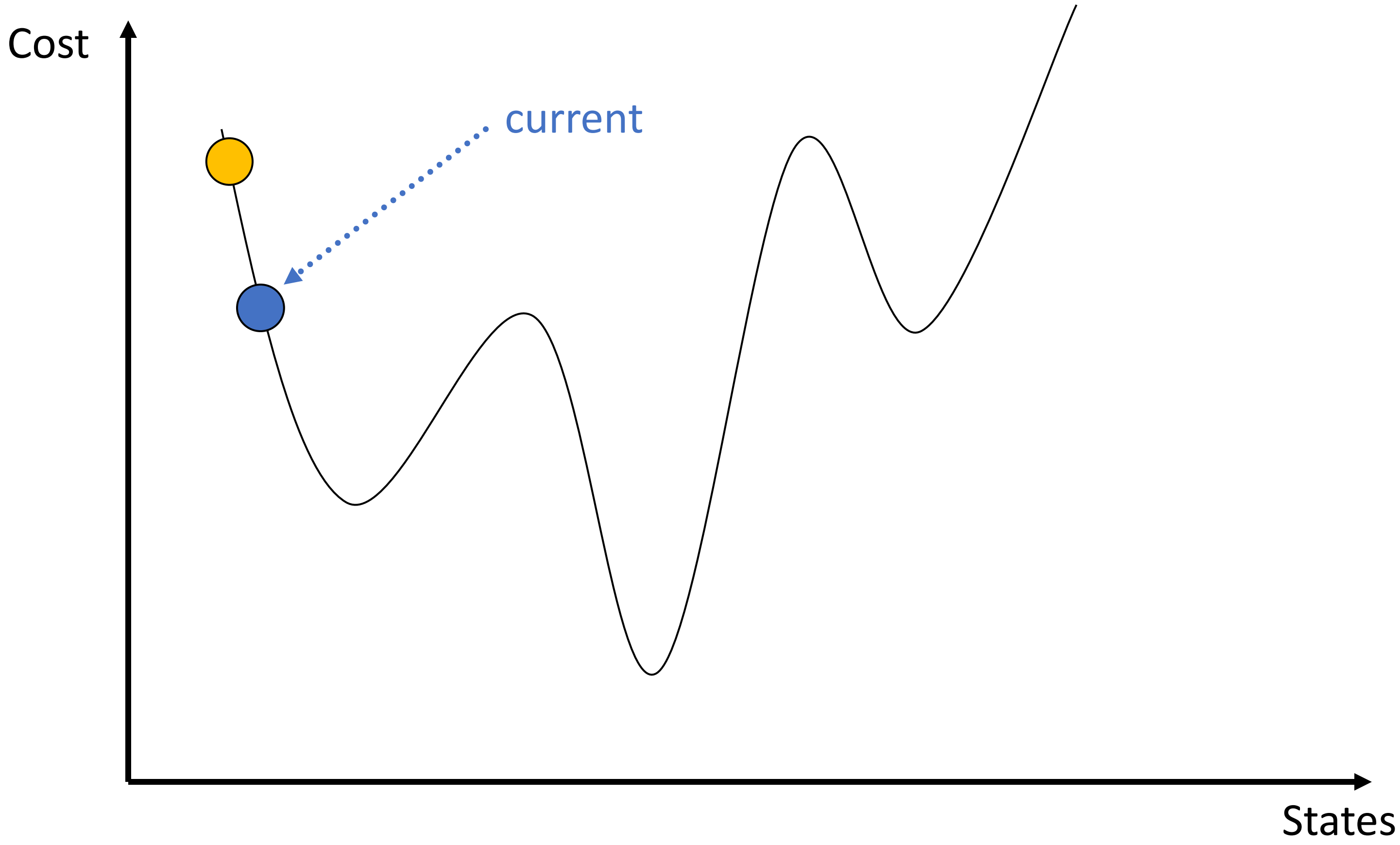
2. Simulated annealing in Action …



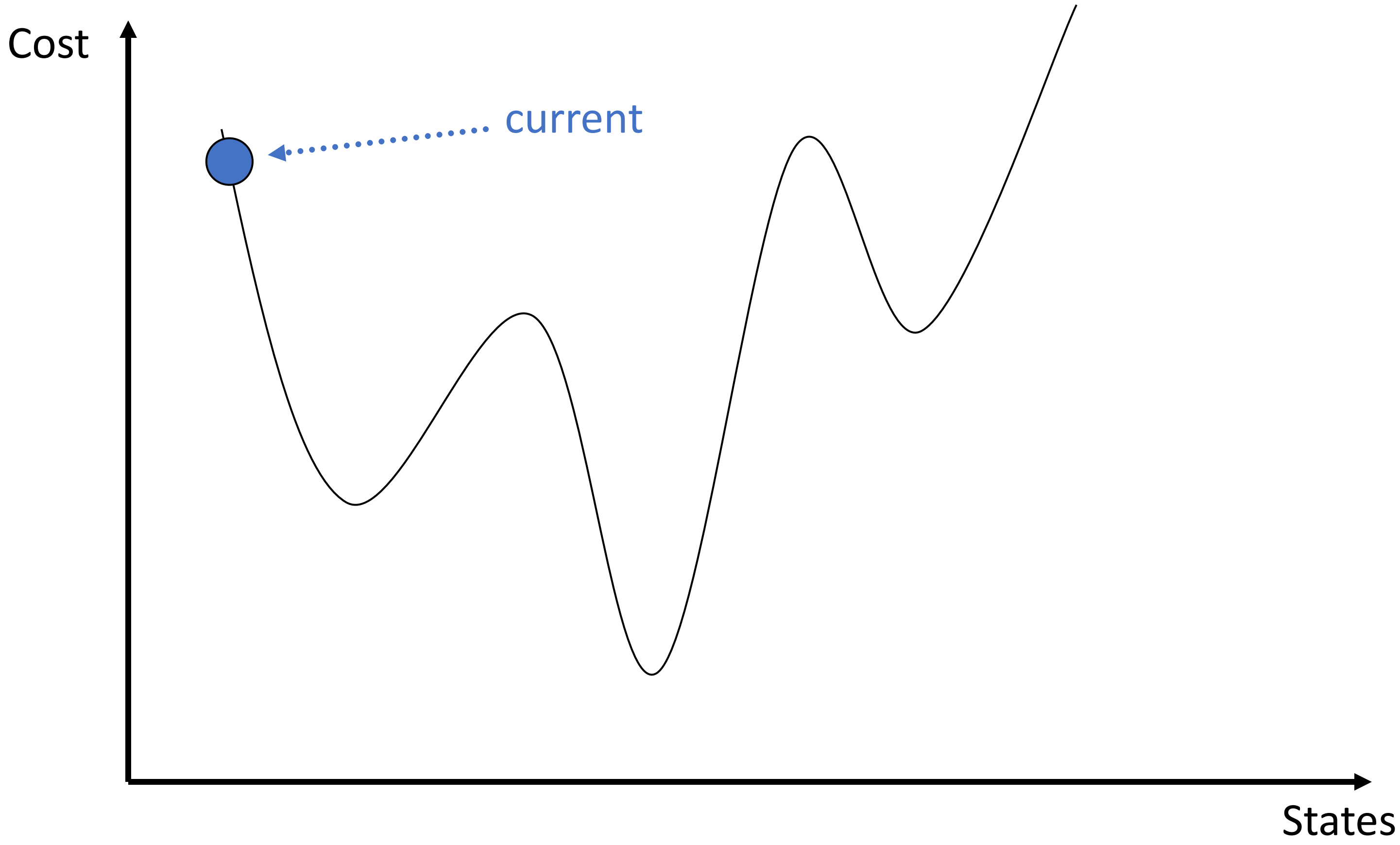
2. Simulated annealing in Action …



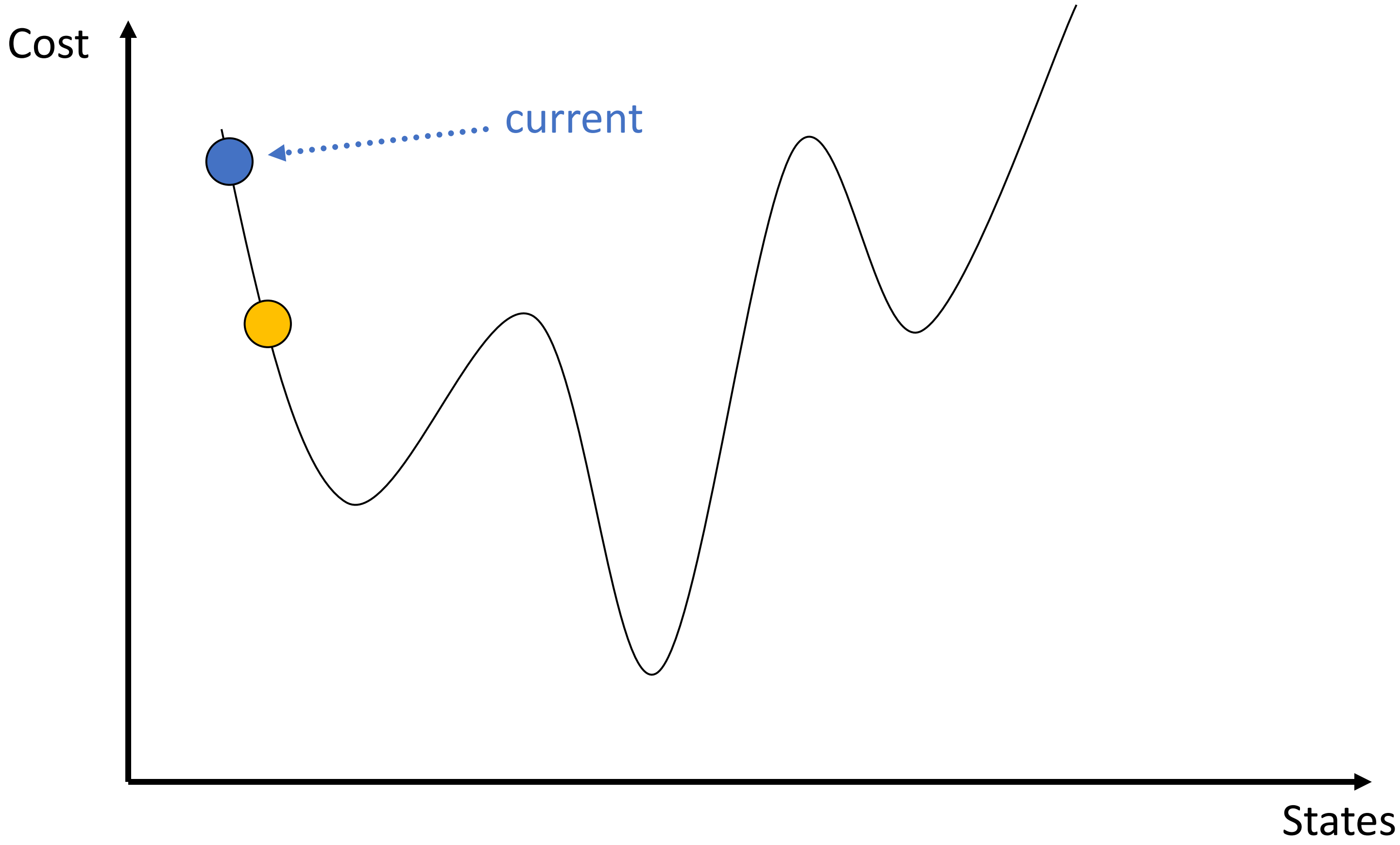
2. Simulated annealing in Action ...



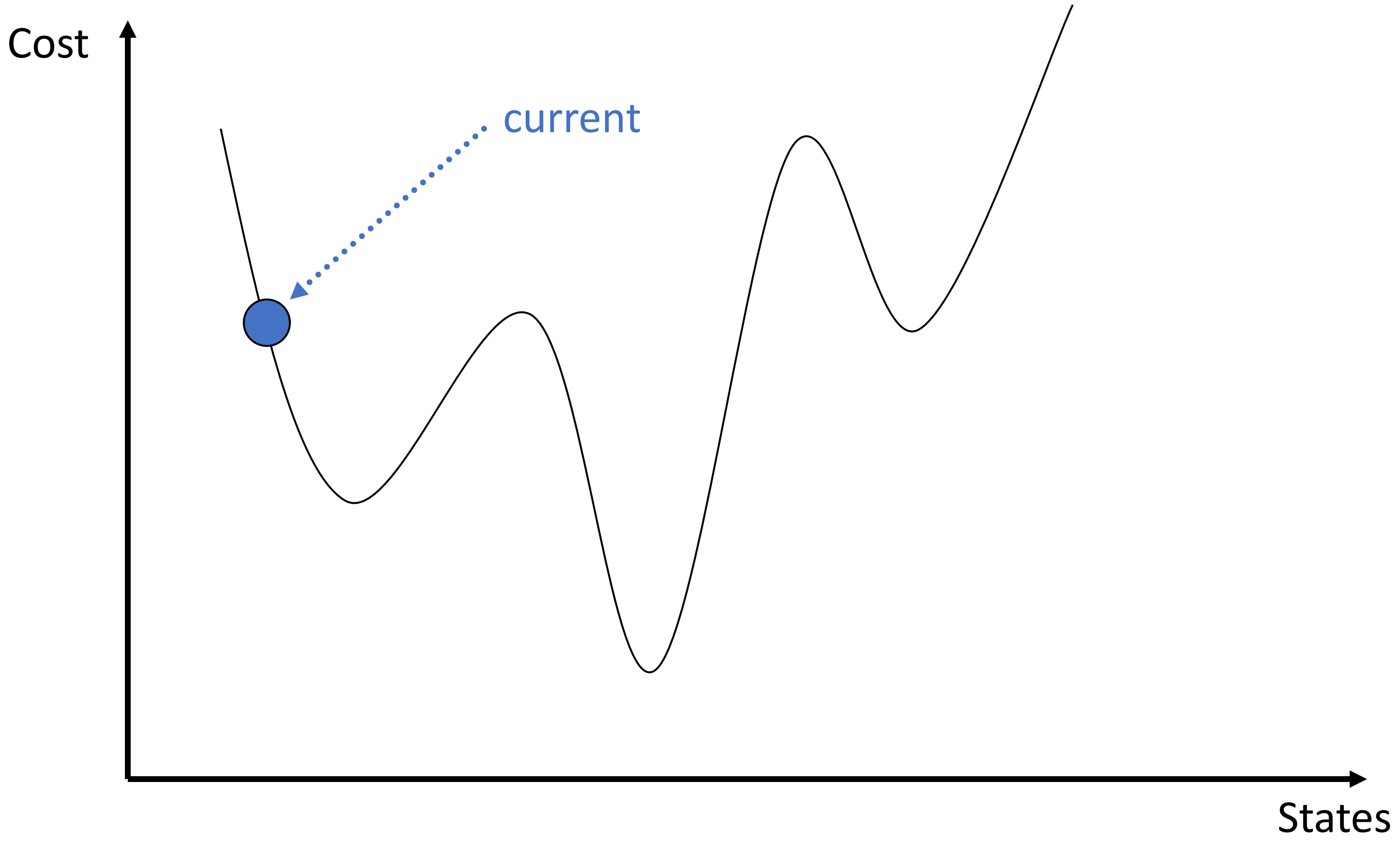
2. Simulated annealing in Action …



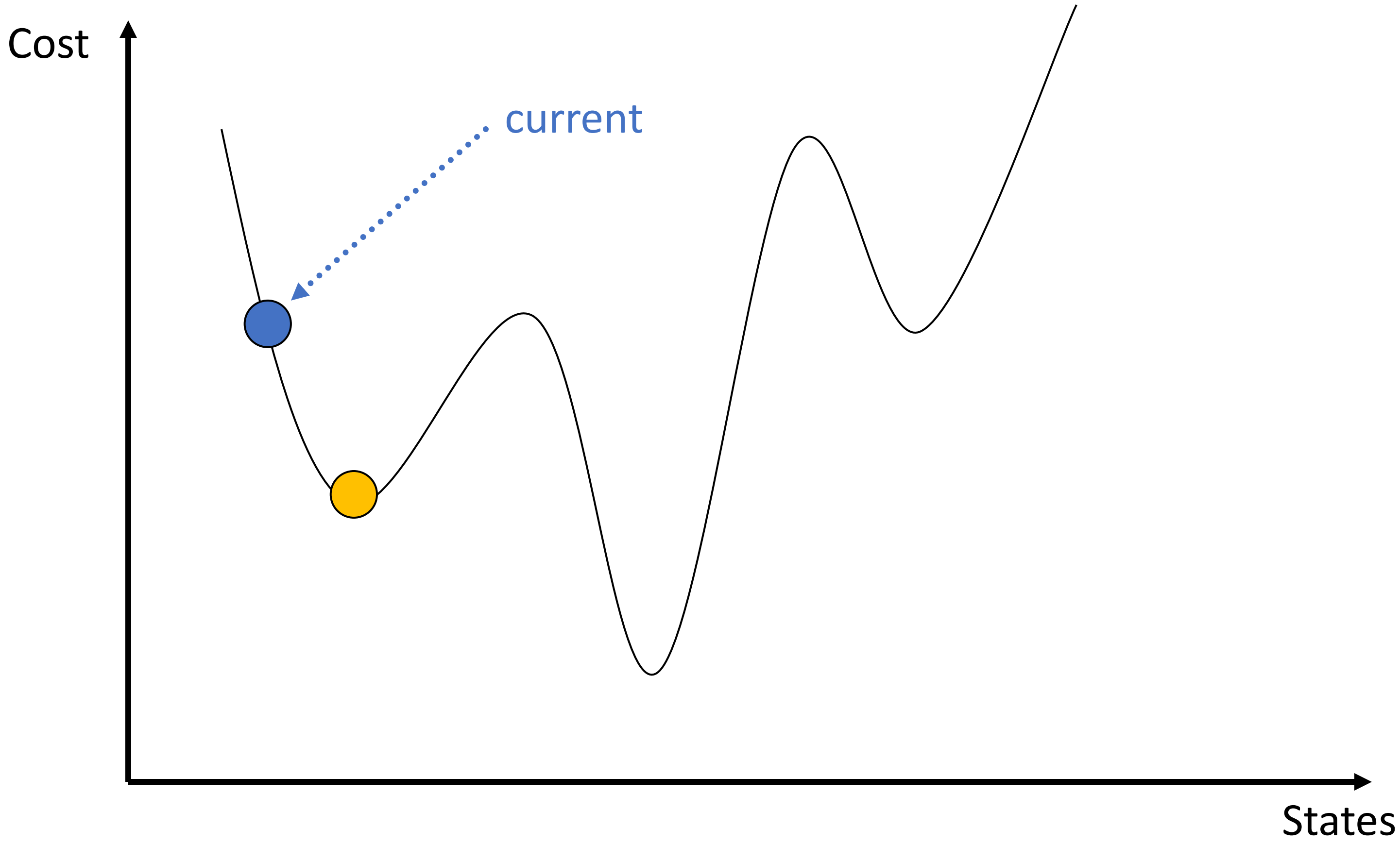
2. Simulated annealing in Action …



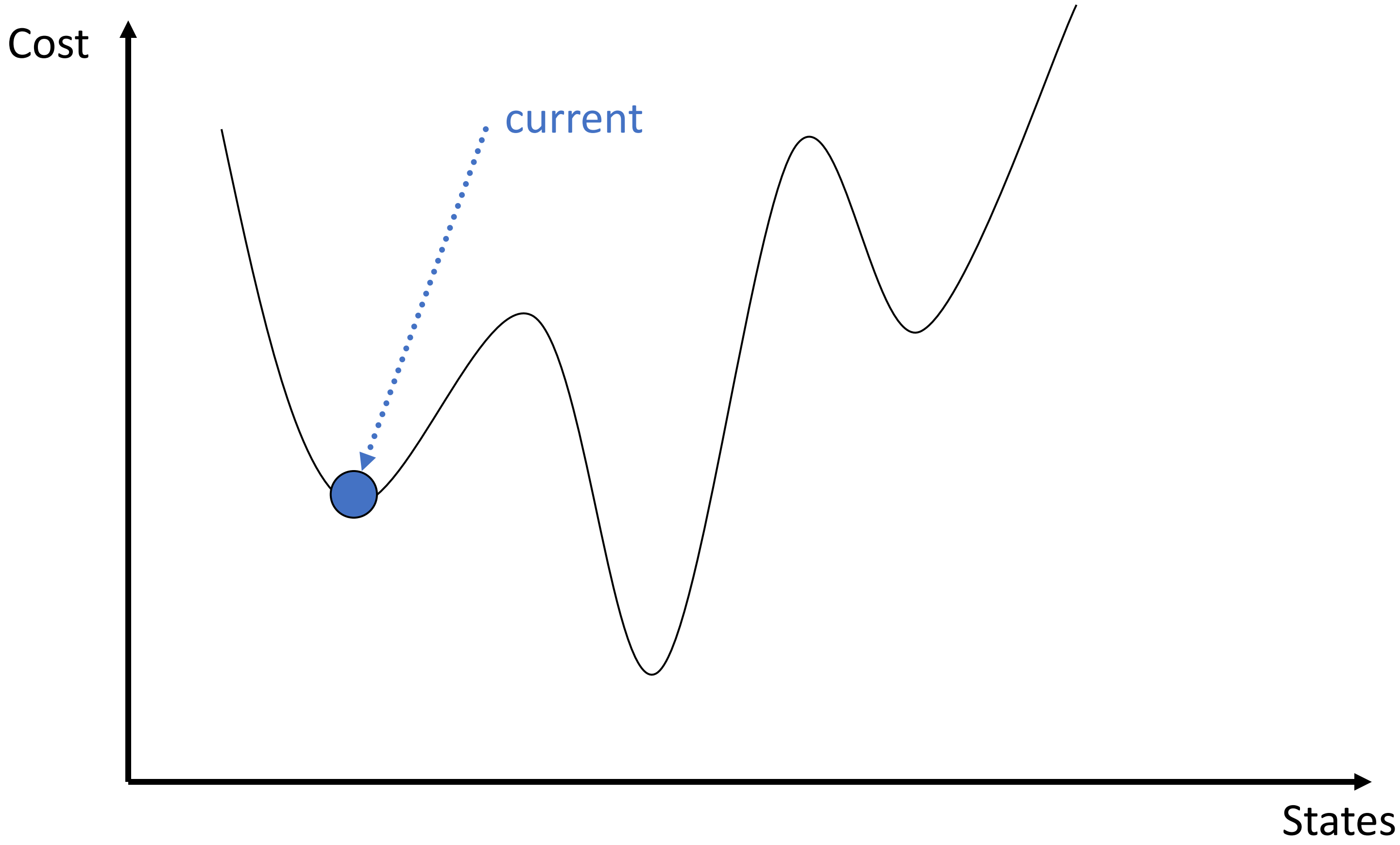
2. Simulated annealing in Action …



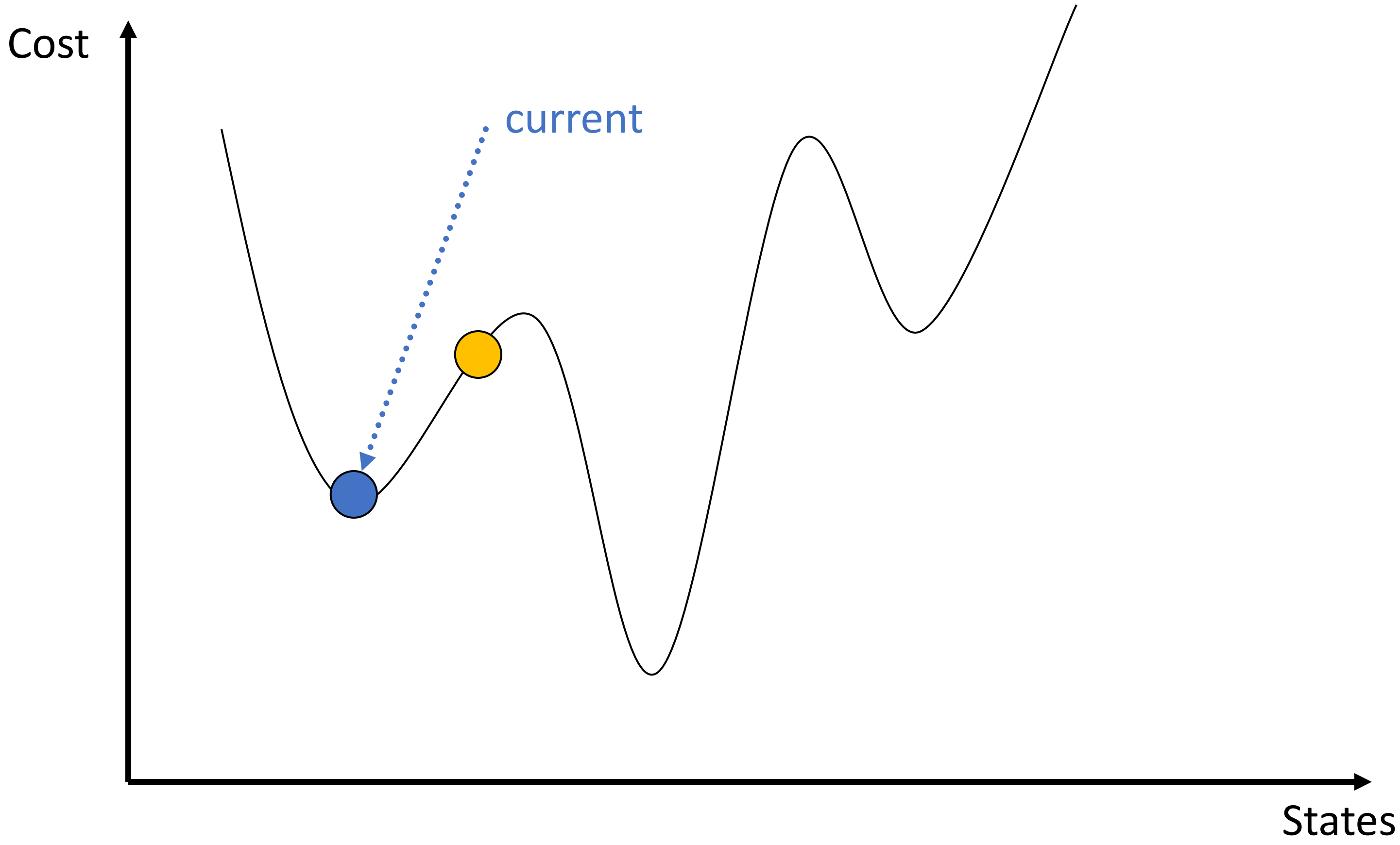
2. Simulated annealing in Action …



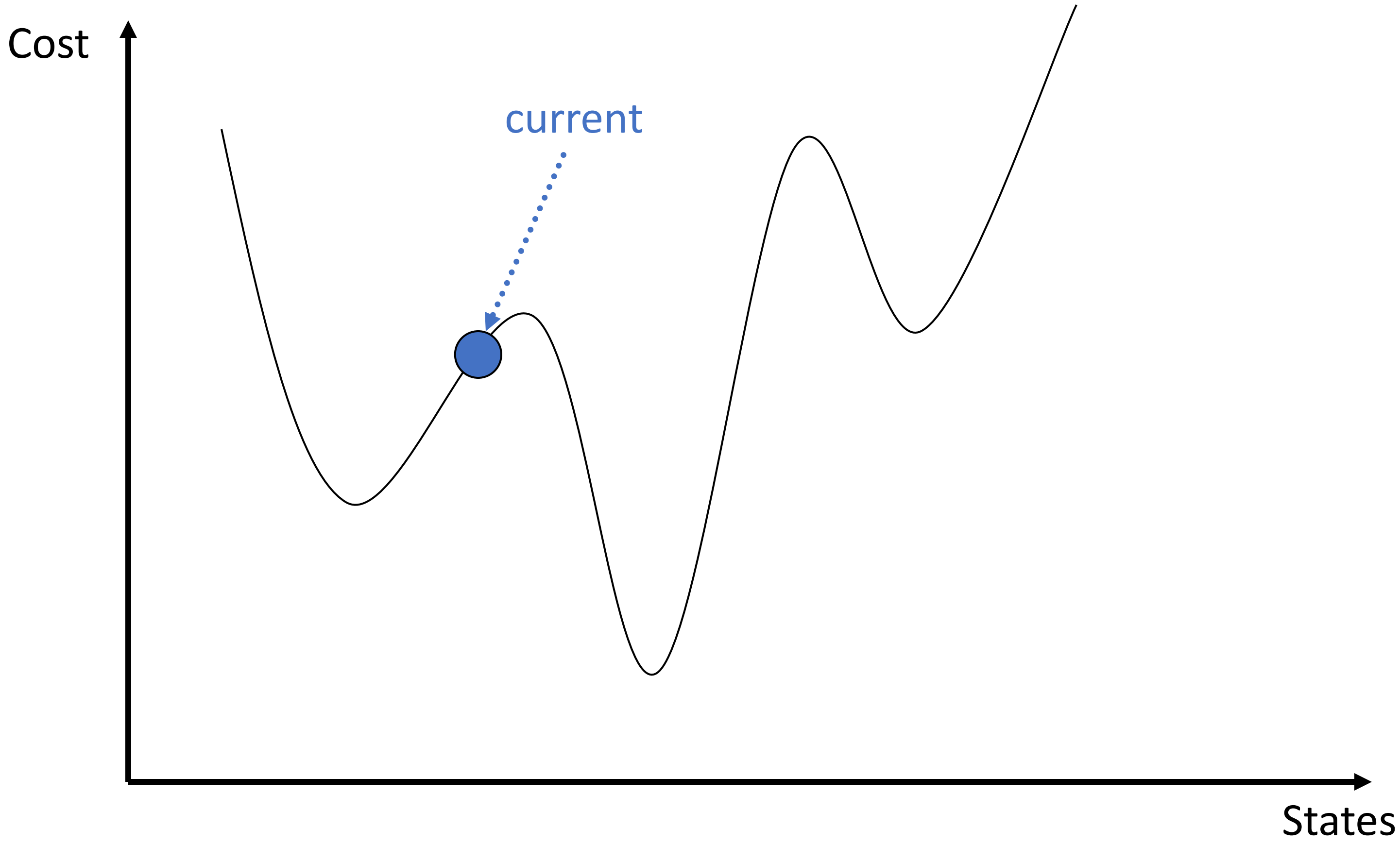
2. Simulated annealing in Action …



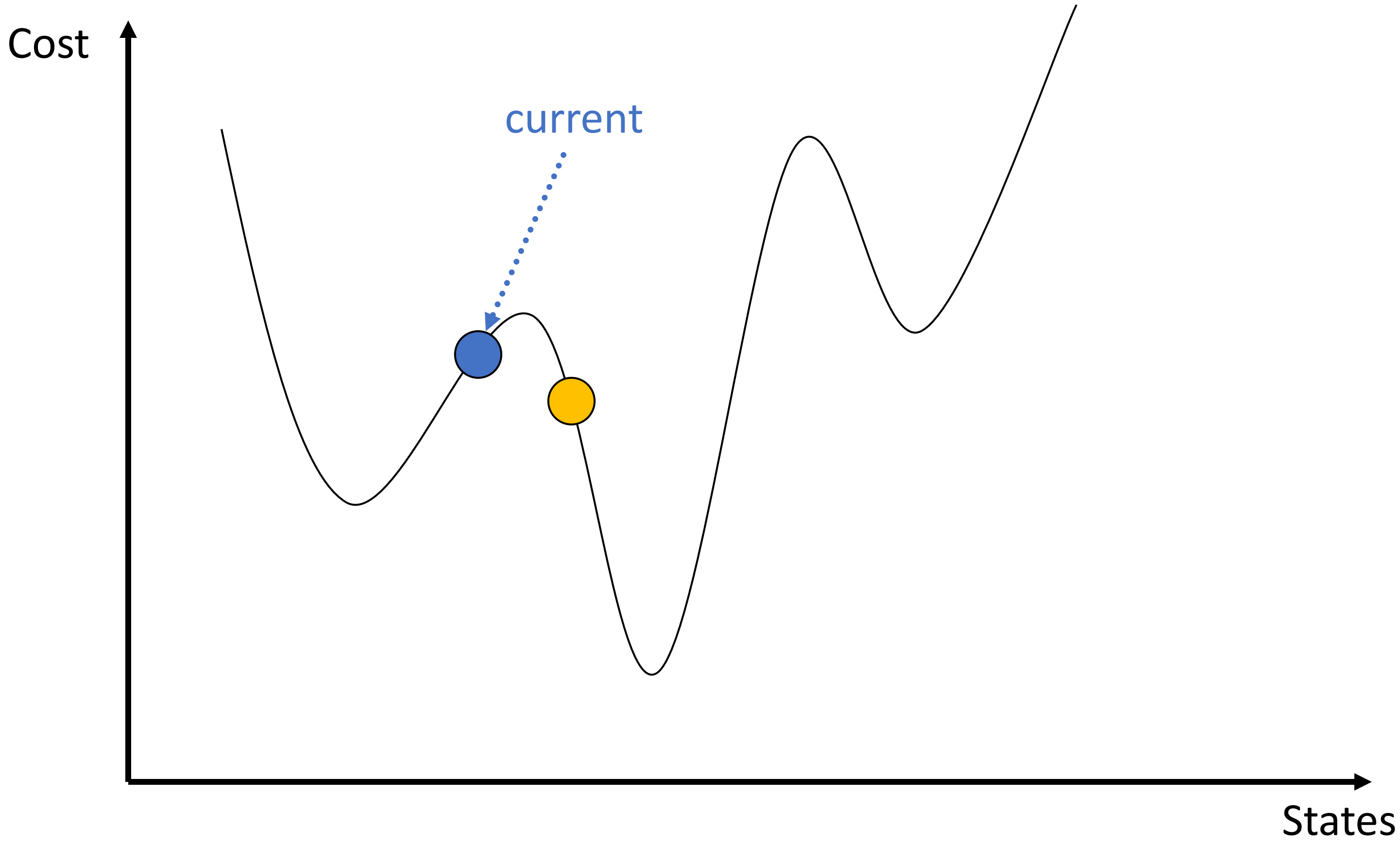
2. Simulated annealing in Action …



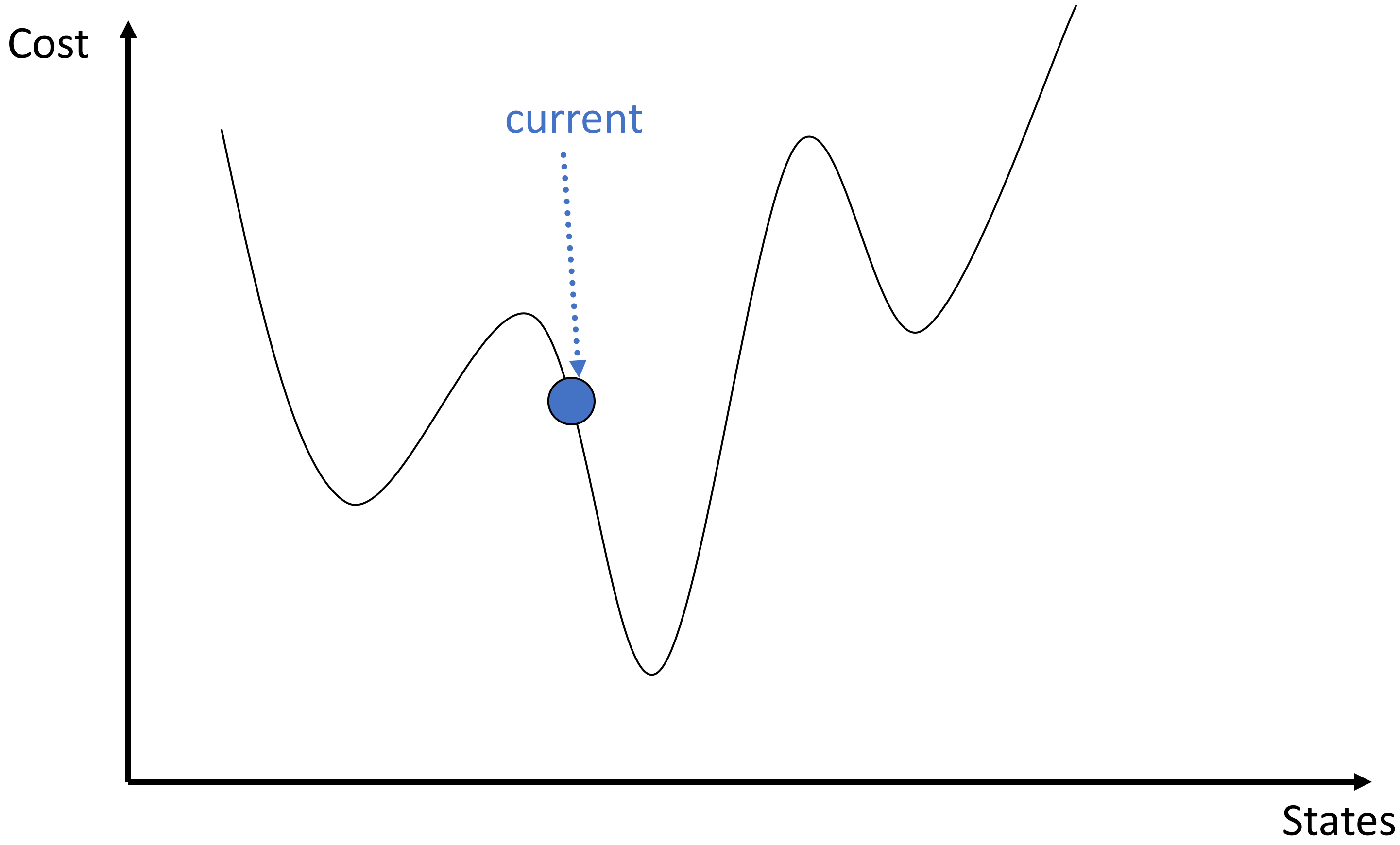
2. Simulated annealing in Action …



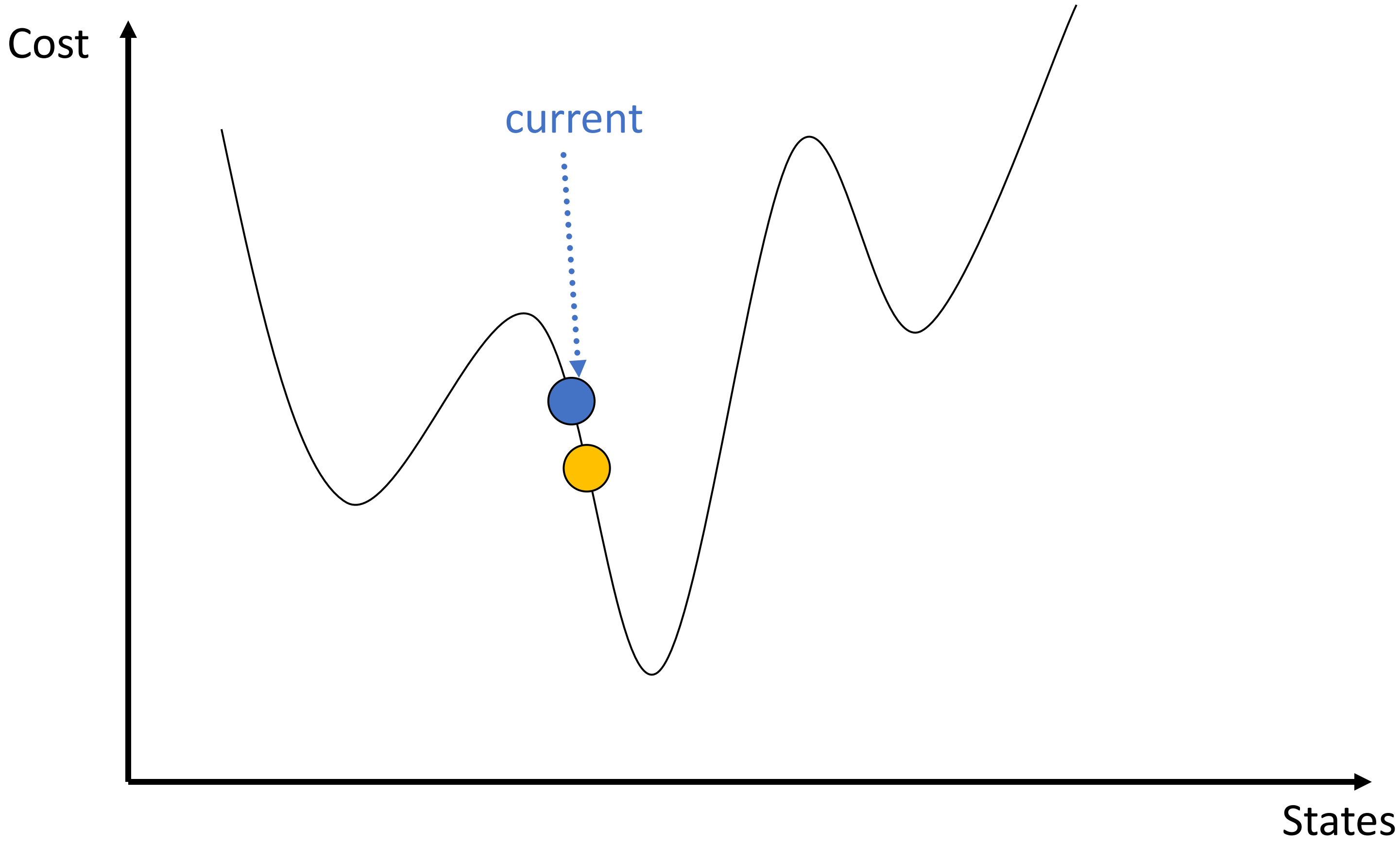
2. Simulated annealing in Action …



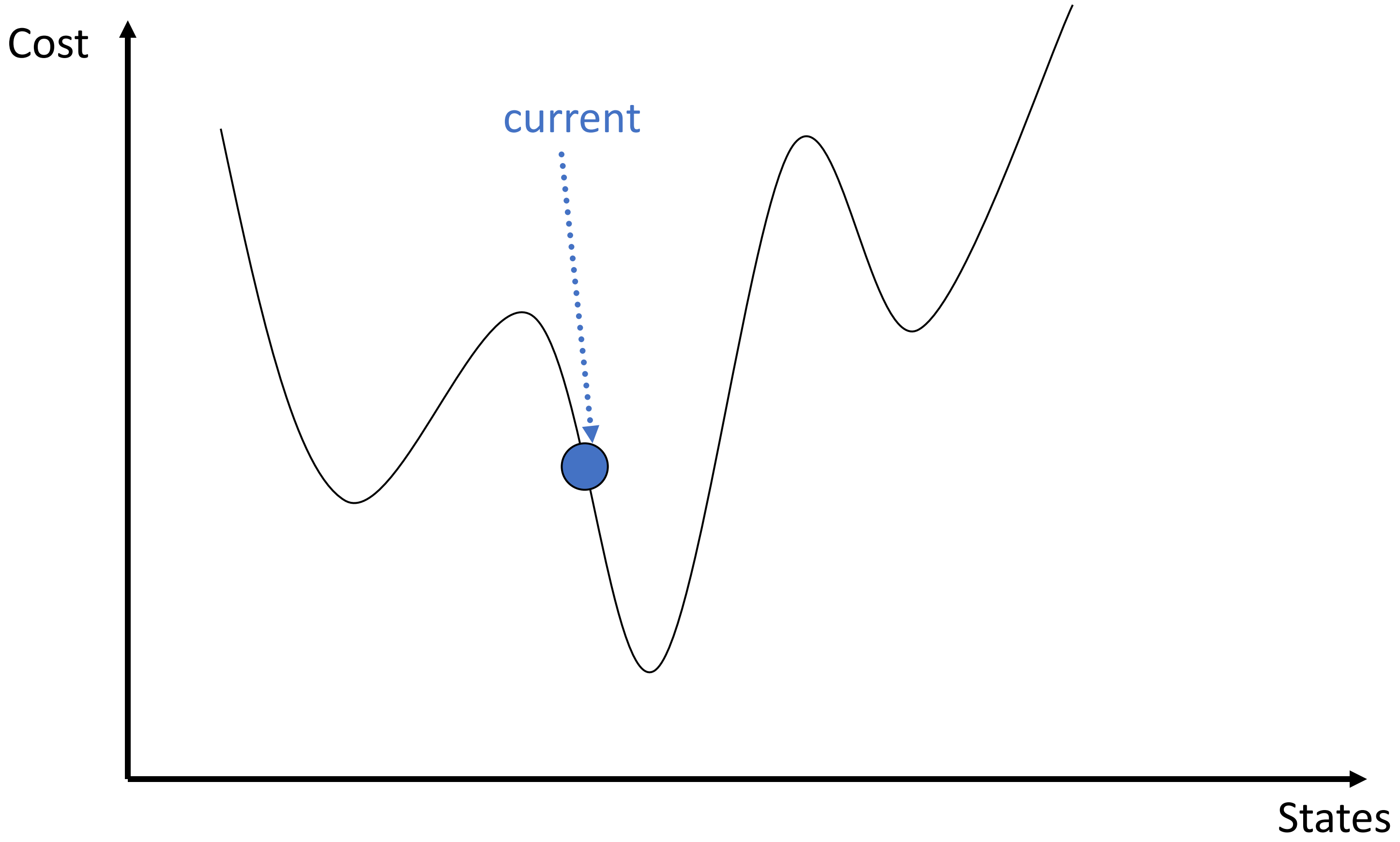
2. Simulated annealing in Action …



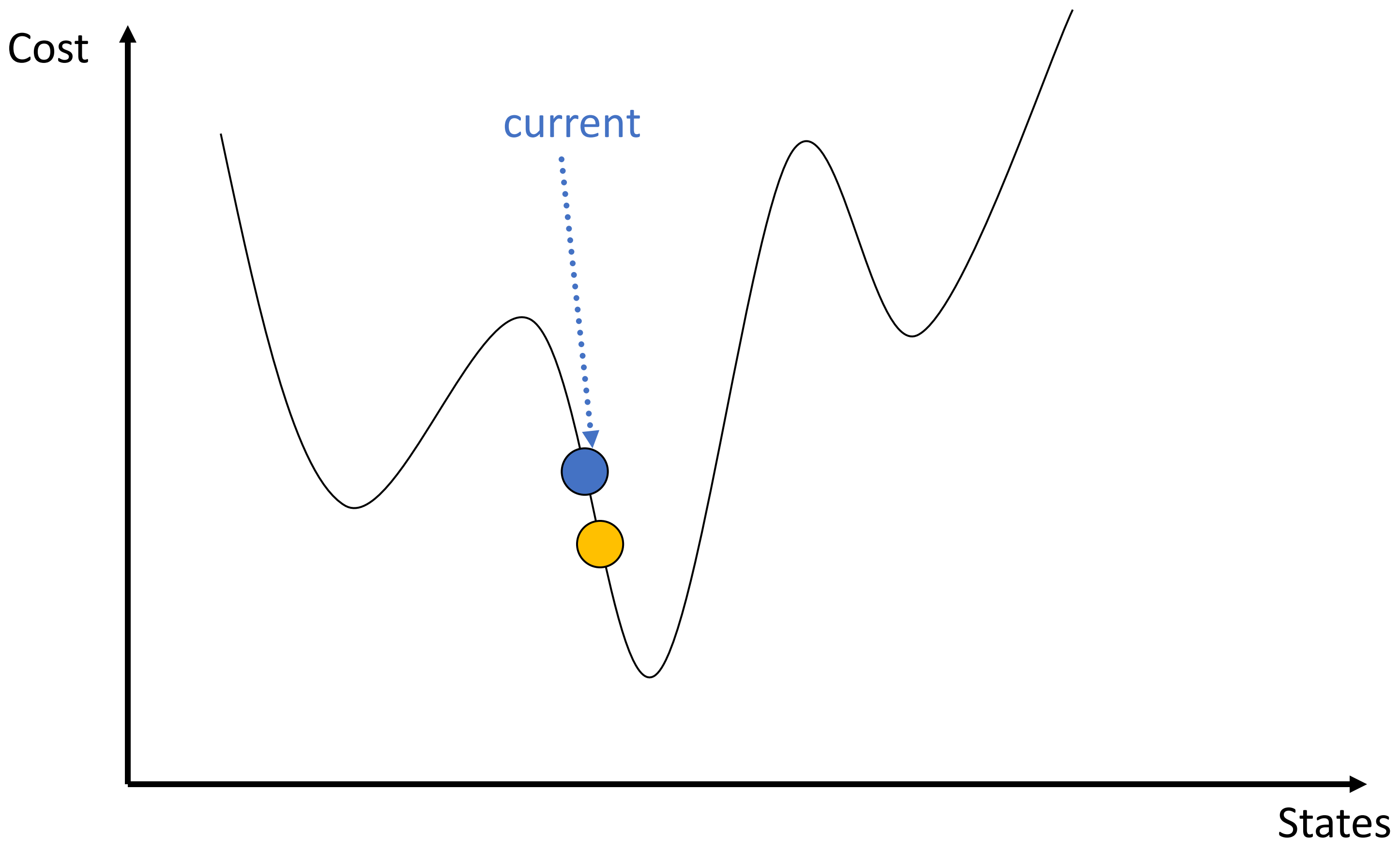
2. Simulated annealing in Action …



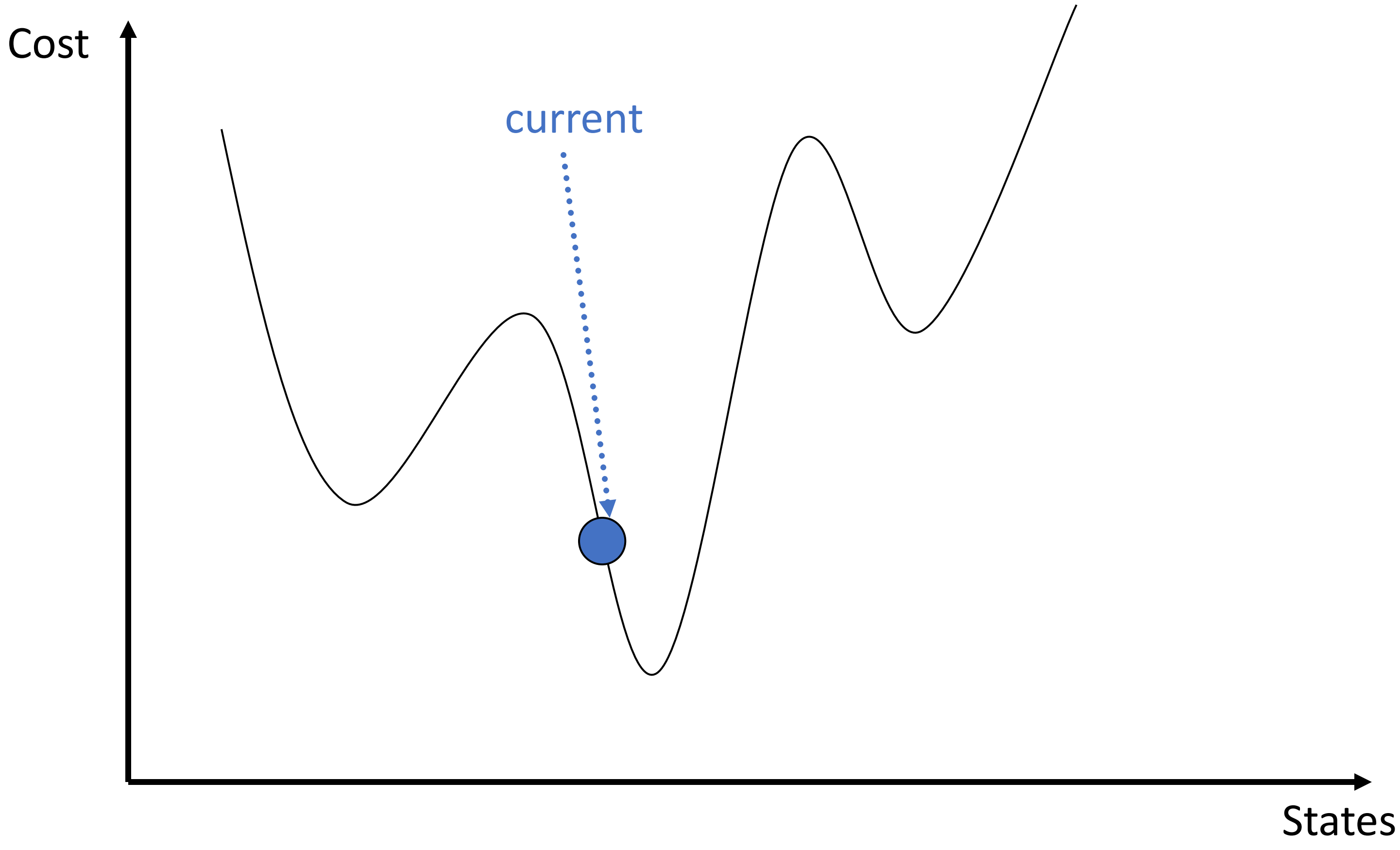
2. Simulated annealing in Action …



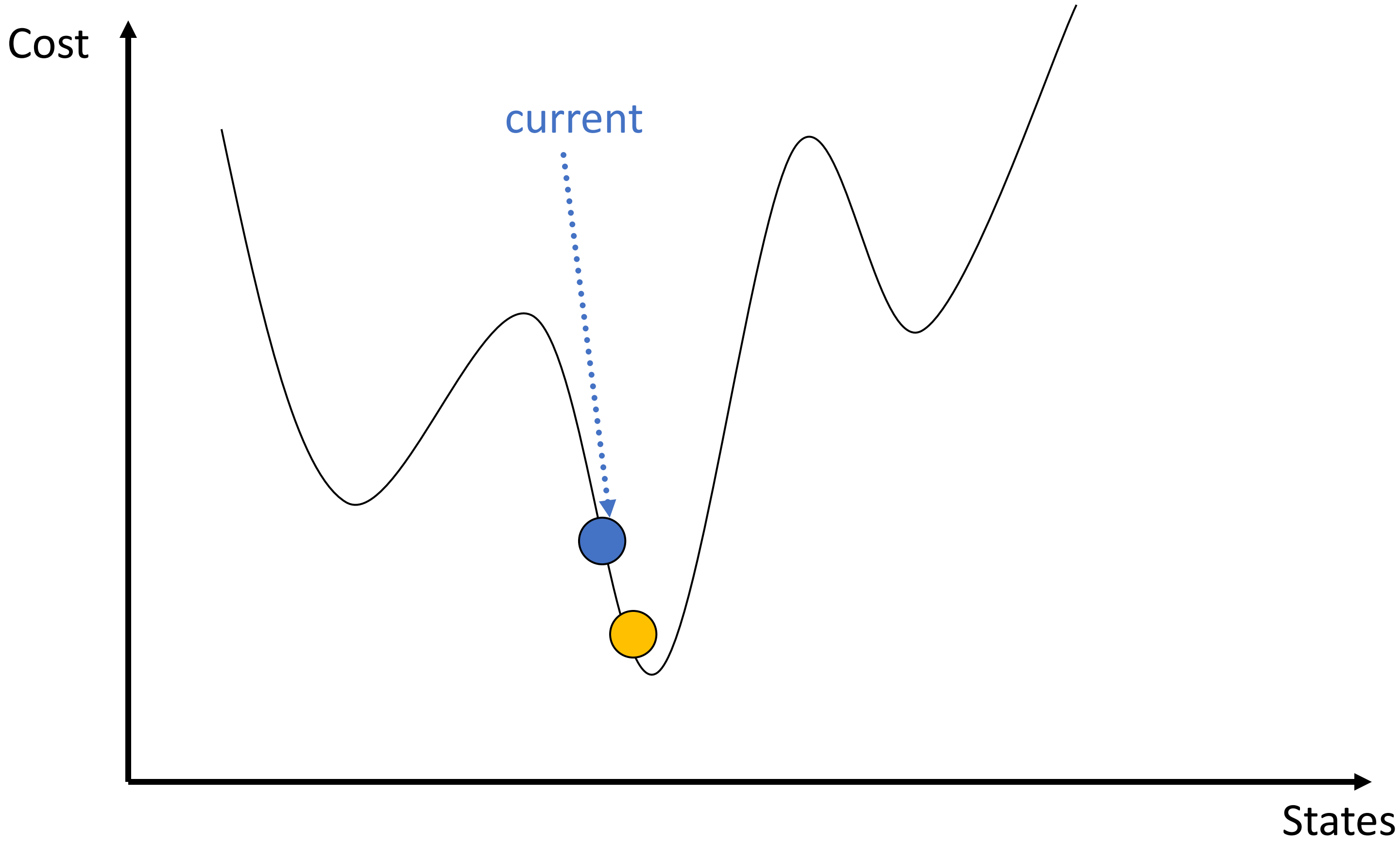
2. Simulated annealing in Action …



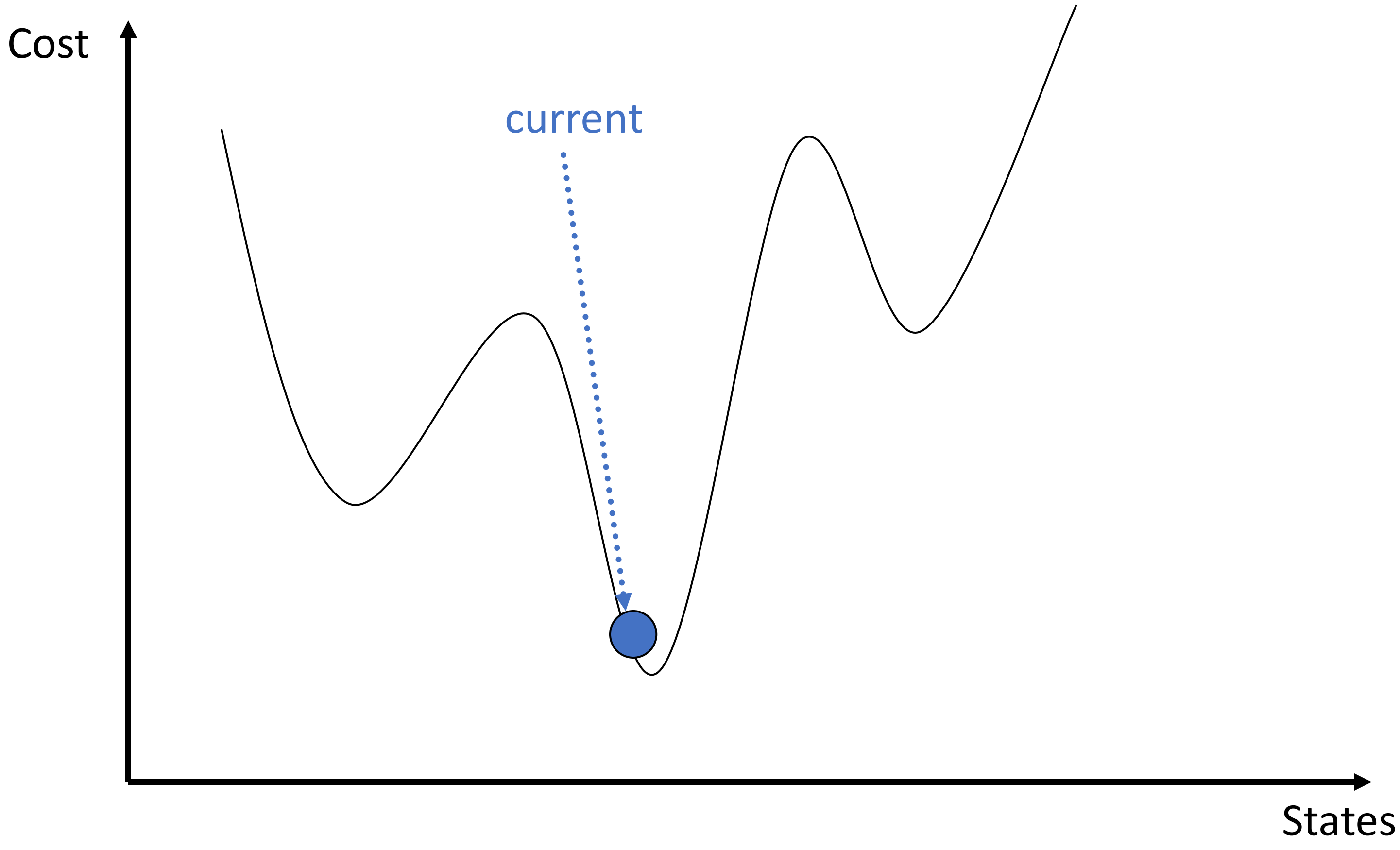
2. Simulated annealing in Action …



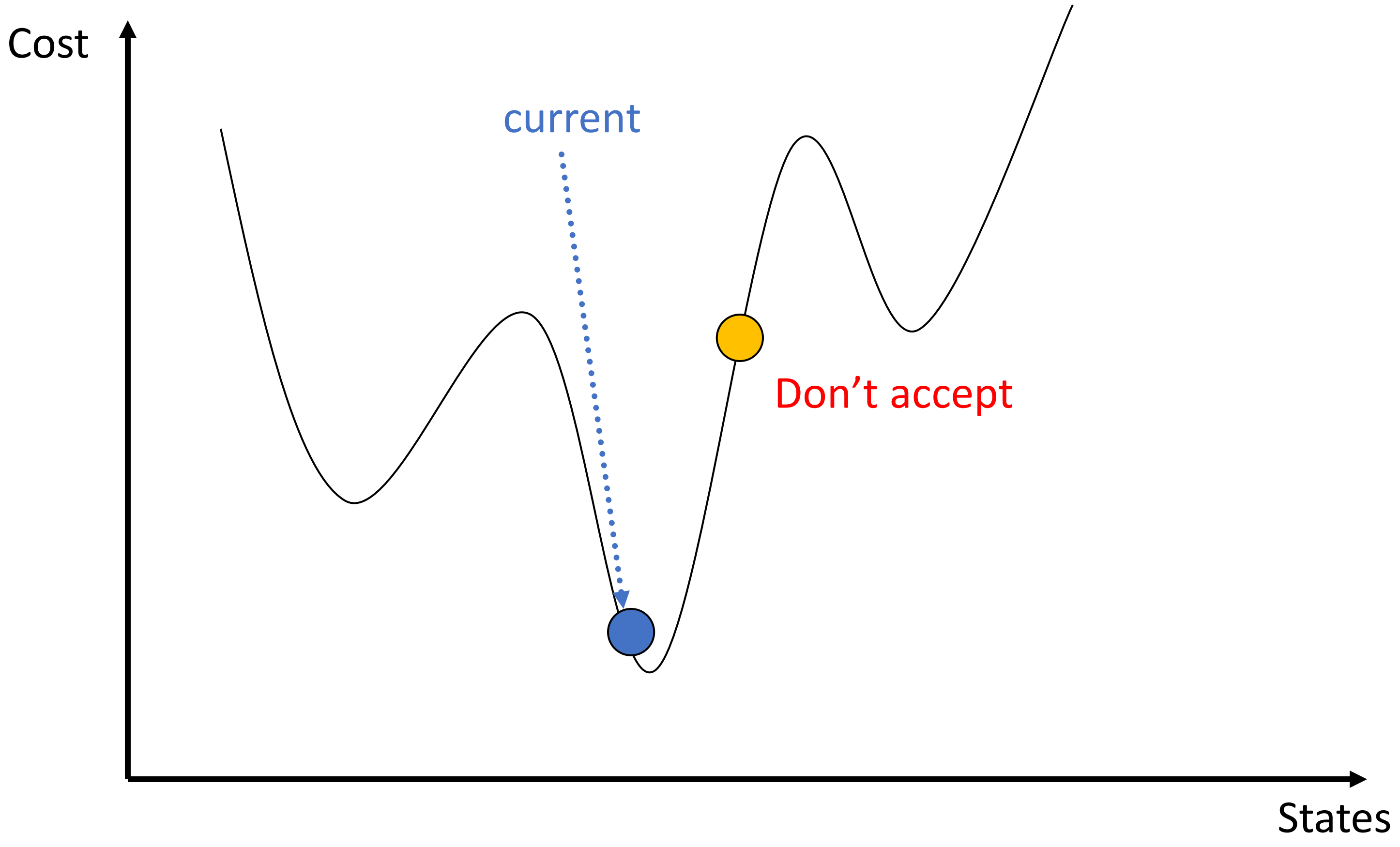
2. Simulated annealing in Action …



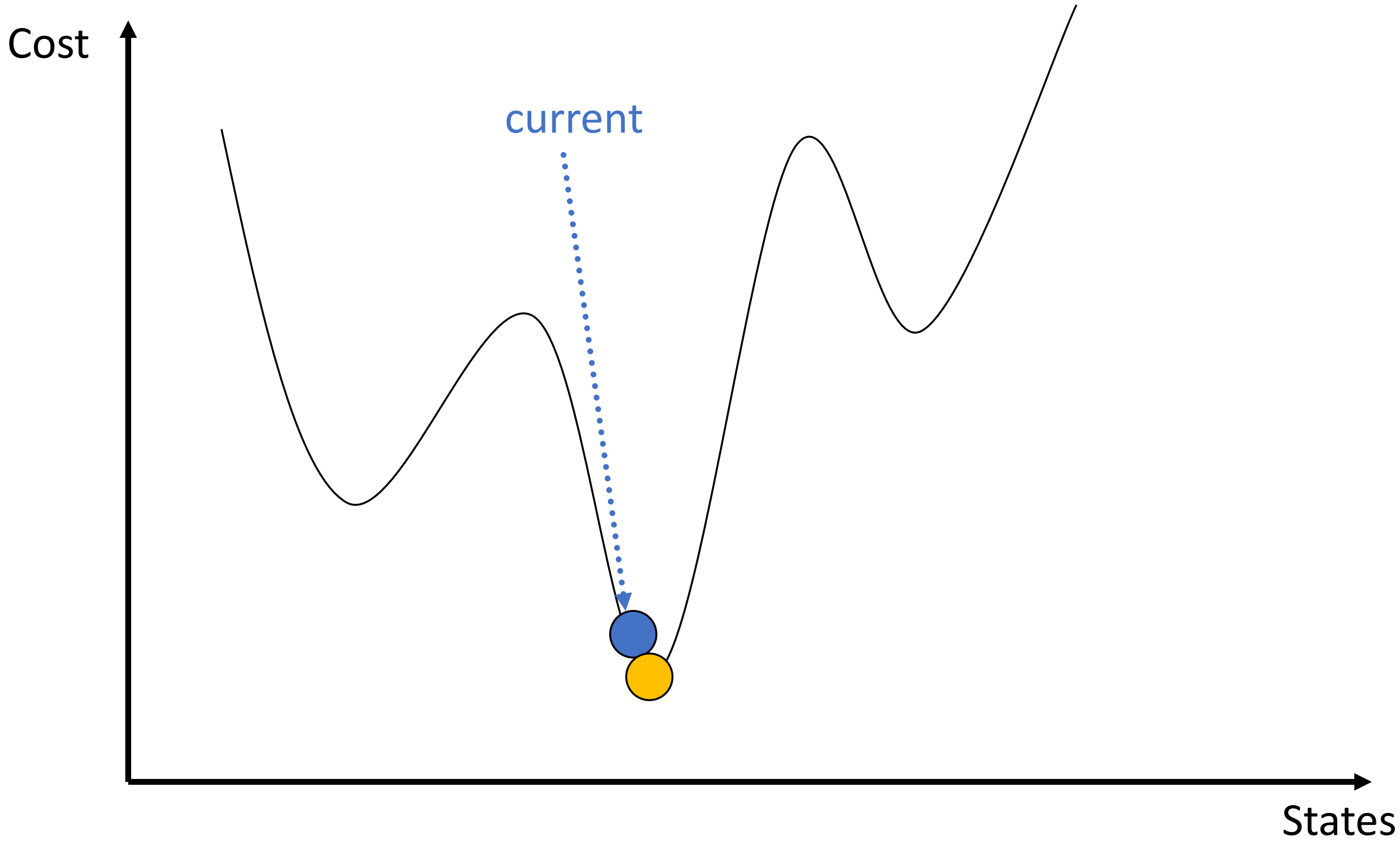
2. Simulated annealing in Action …



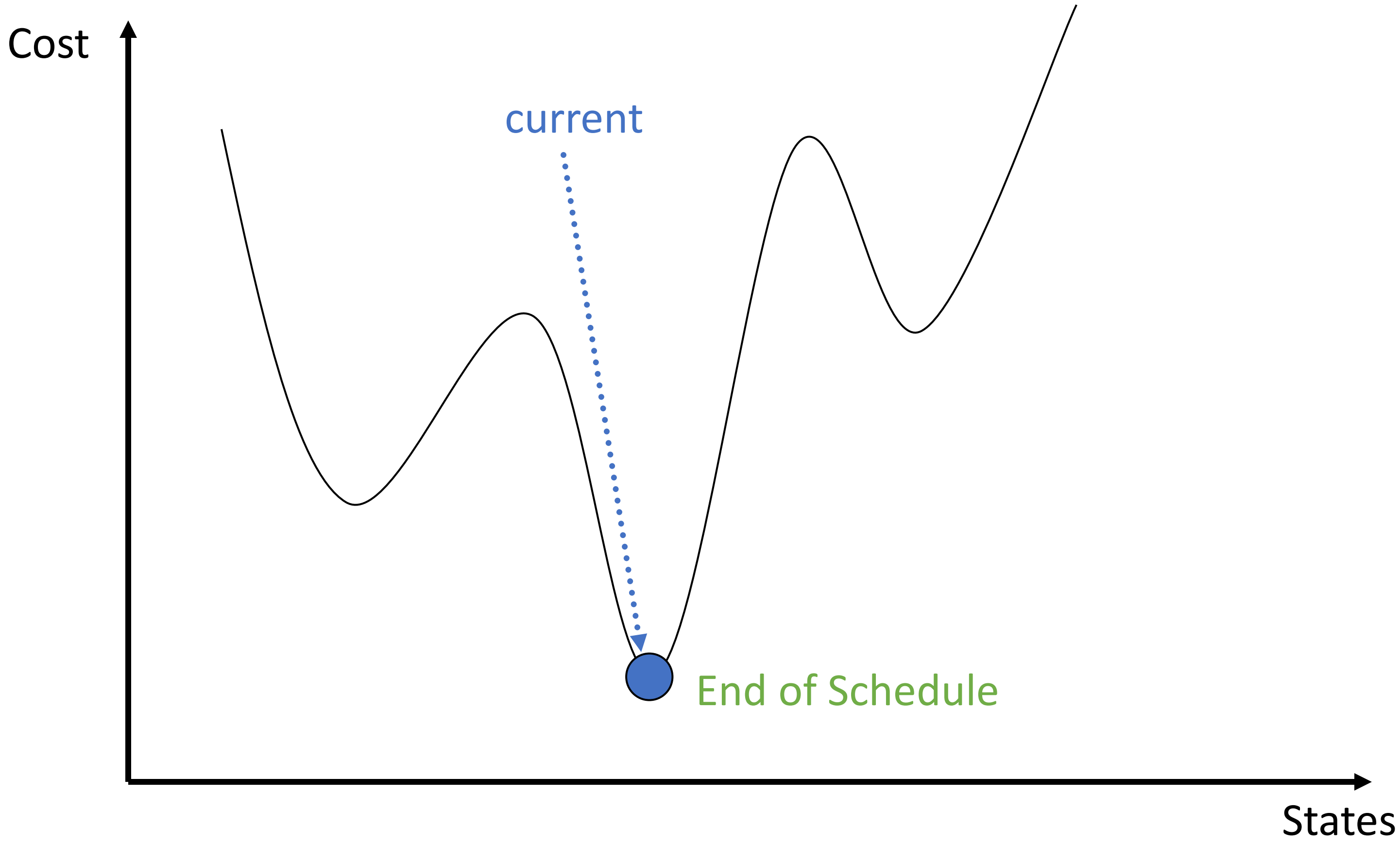
2. Simulated annealing in Action …



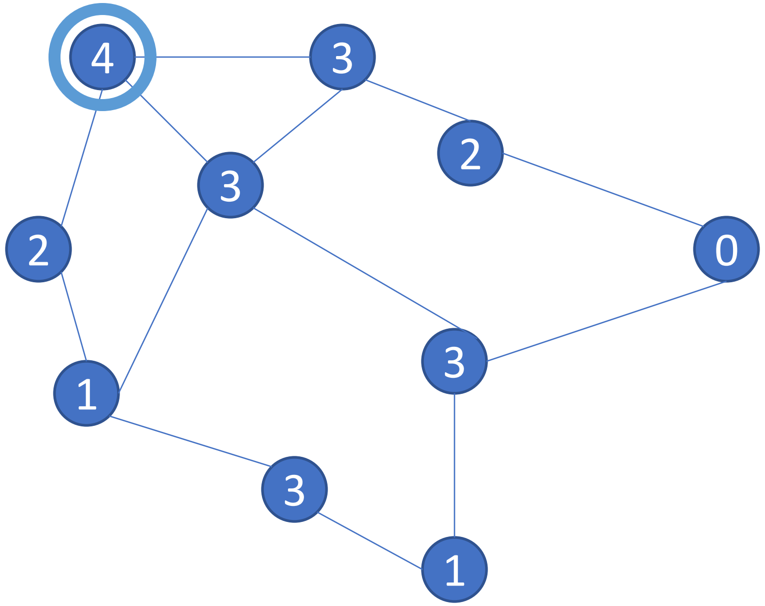
2. Simulated annealing in Action …



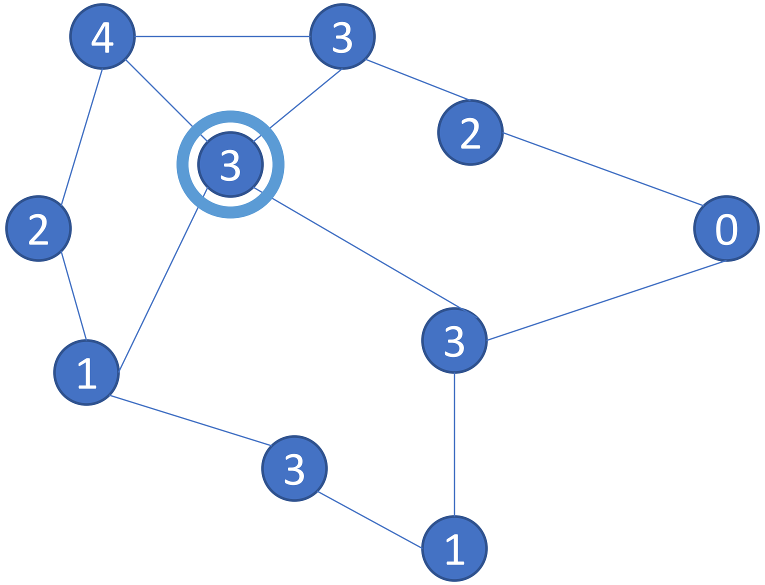
2. Simulated annealing in Action …



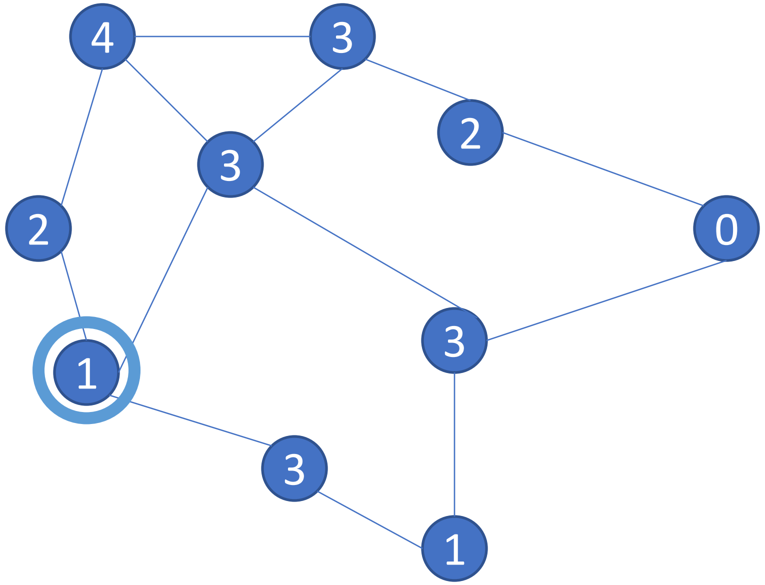
2. Simulated annealing



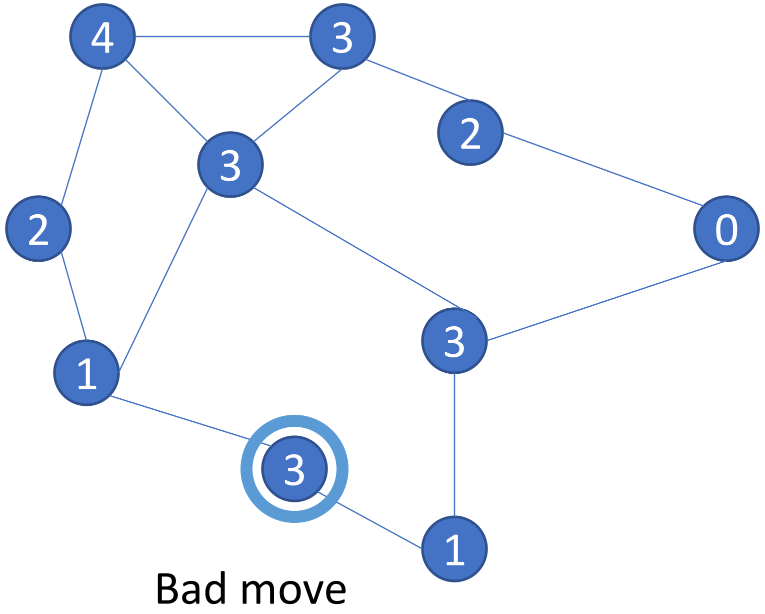
2. Simulated annealing



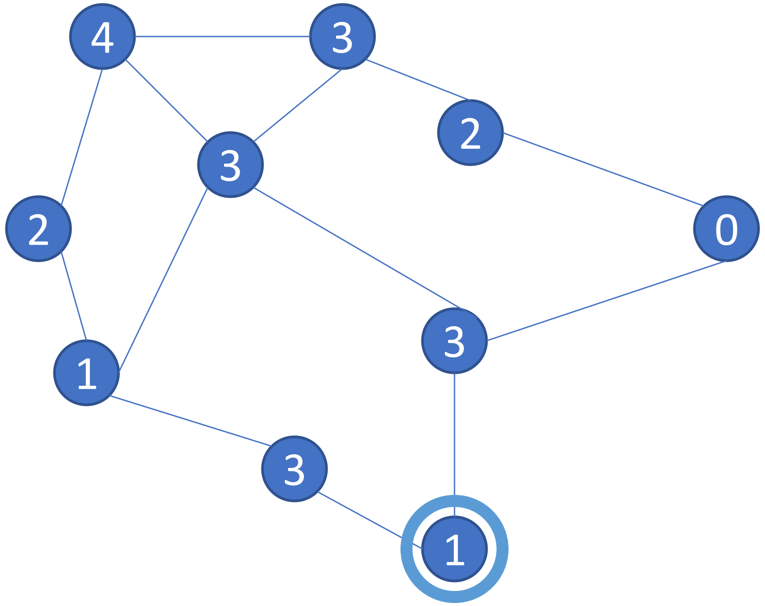
2. Simulated annealing



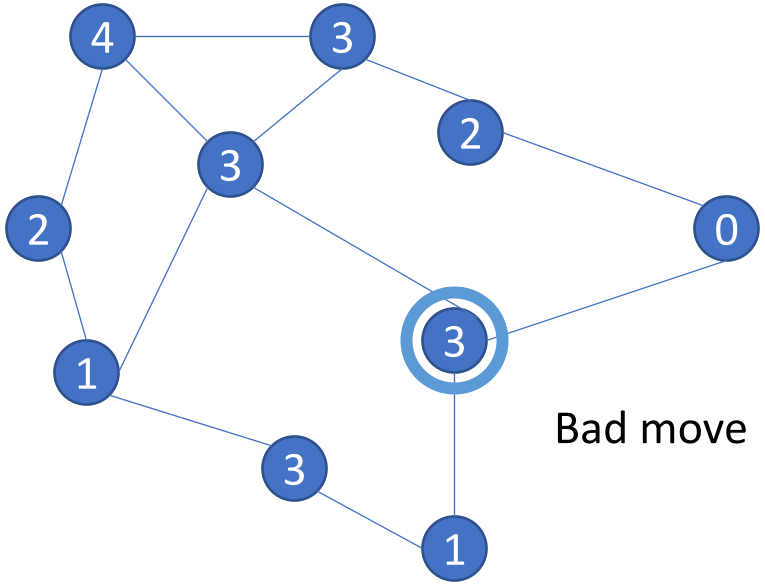
2. Simulated annealing



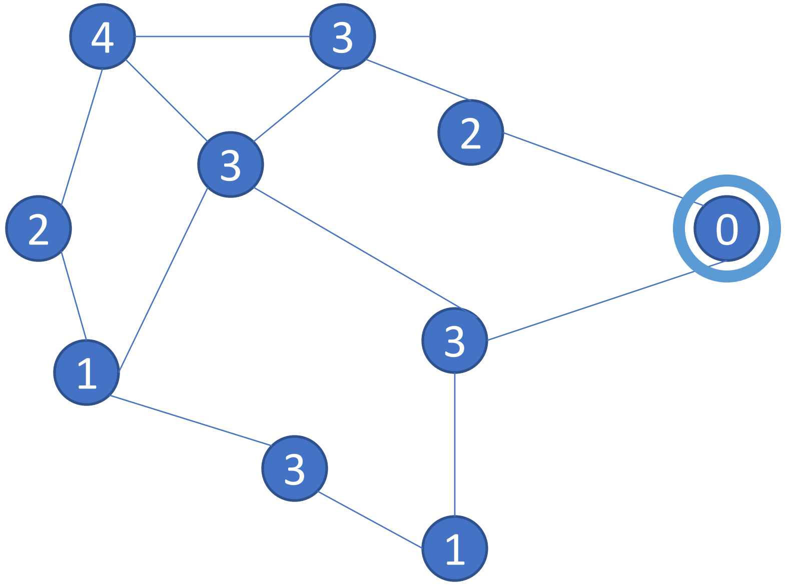
2. Simulated annealing



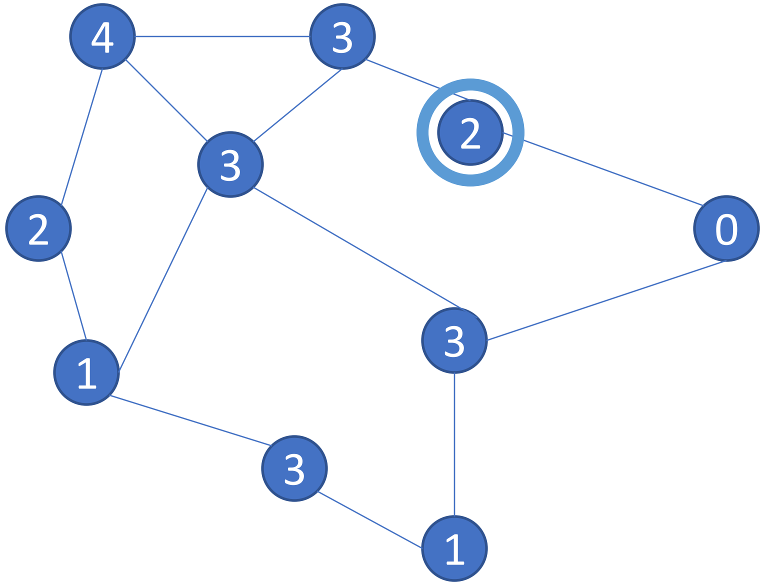
2. Simulated annealing



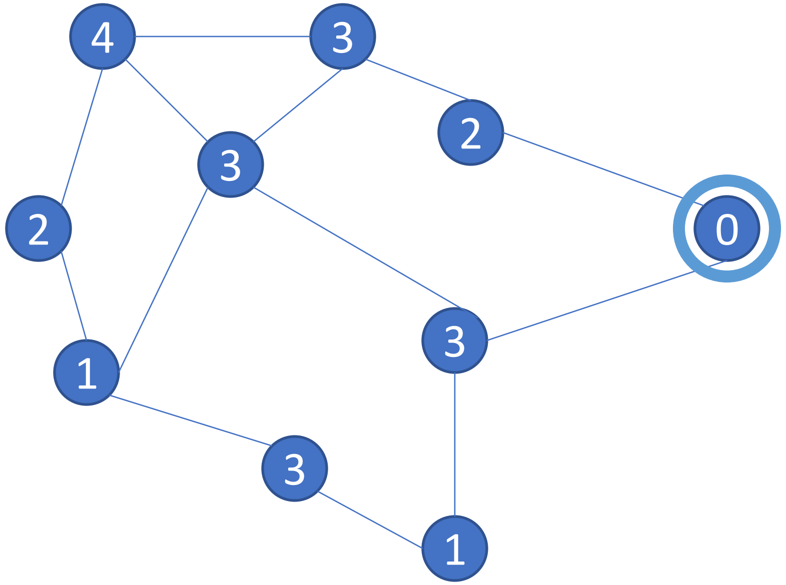
2. Simulated annealing



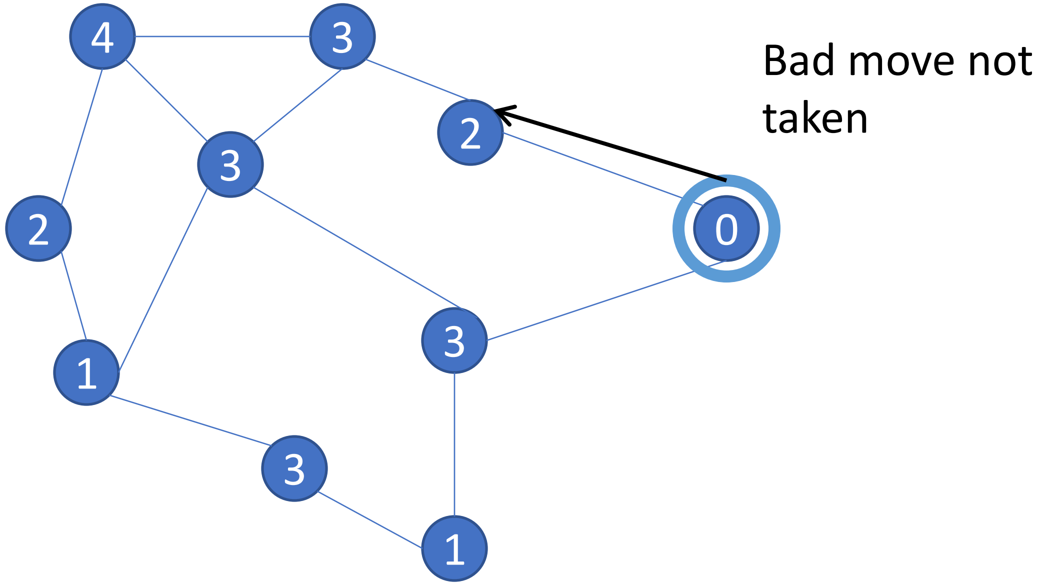
2. Simulated annealing



2. Simulated annealing



2. Simulated annealing



2. Properties of simulated annealing search

- Can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1.
- Widely used in VLSI layout, airline scheduling, etc.