UiO : University of Oslo

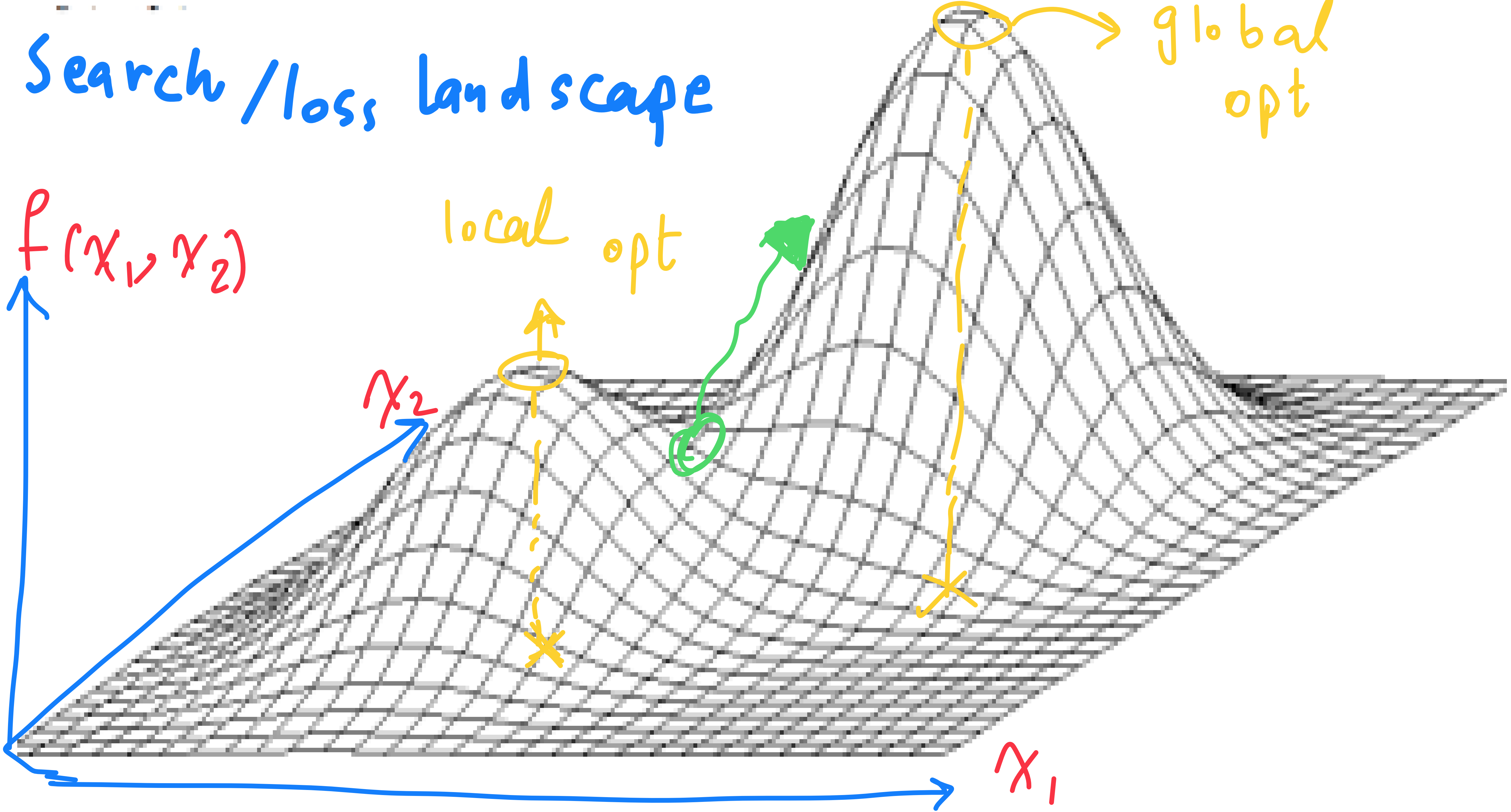
institutt for informatikk

IN3050/IN4050 - Introduction to Artificial Intelligence and Machine Learning

Optimization and Search

Ali Ramezani-Kebrya

Search/loss landscape



Optimization

We need

- A numerical representation x for all possible solutions to the problem
- A function $f(x)$ that tells us how good solution x is

  how well do we predict given ground-truth?
- A way of finding
  - $\max_x f(x)$ if bigger $f(x)$ is better (benefit)

    Knapsack problem
  - $\min_x f(x)$ if smaller $f(x)$ is better (cost)

    TSP

Optimisation and Search

- **Continous Optimization** is the mathematical discipline which is concerned with finding the maxima and minima of functions, possibly subject to constraints.

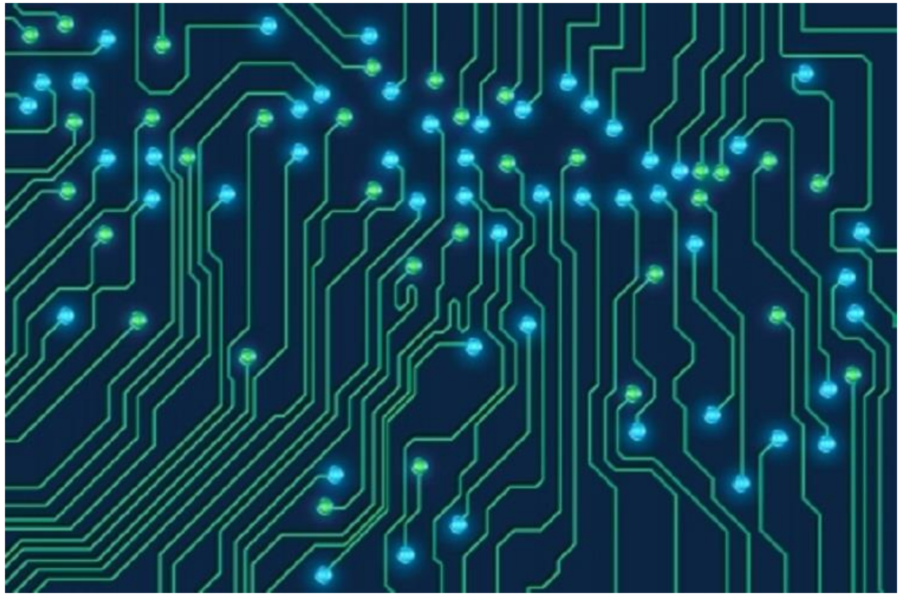
- **Discrete Optimization** is the activity of looking thoroughly in order to find an item with specified properties among a collection of items.

Discrete optimization

No intermediate sol

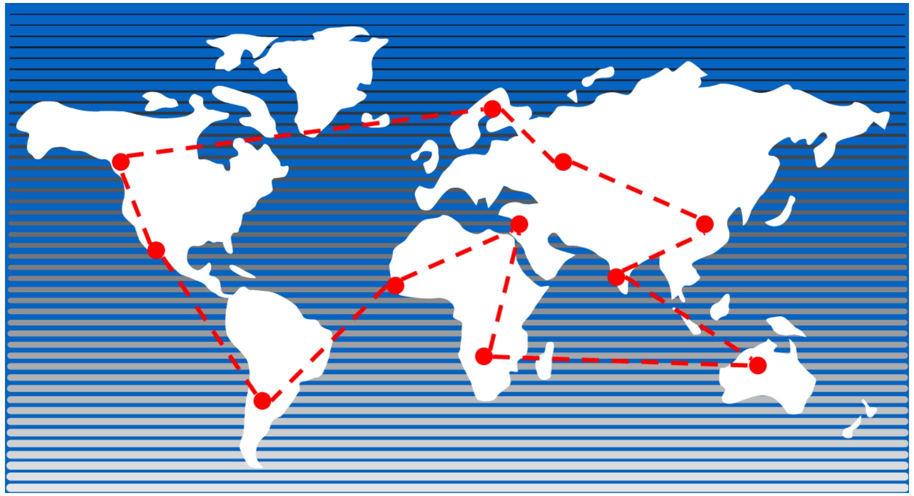
Sol space is discrete

- **Chip design**
  - Routing tracks during chip layout design
- **Timetabling**
  - E.g.: Find a course time table with the minimum number of clashes for registered students
- **Travelling salesman problem**
  - Optimization of travel routes and similar logistics problems



Example: Travelling Salesman Problem (TSP)

- Given the coordinates of n cities, find the ***shortest closed tour*** which visits each ***once and only once*** (i.e. exactly once).
- Constraint :
  - all cities be visited, once and only once.



Some Optimization Methods

1. Exhaustive search
2. Greedy search and hill climbing
3. Simulated annealing

Discrete

4. Gradient descent/ascent
   - Not applicable for discrete optimization

Continuous

Divide to buckets

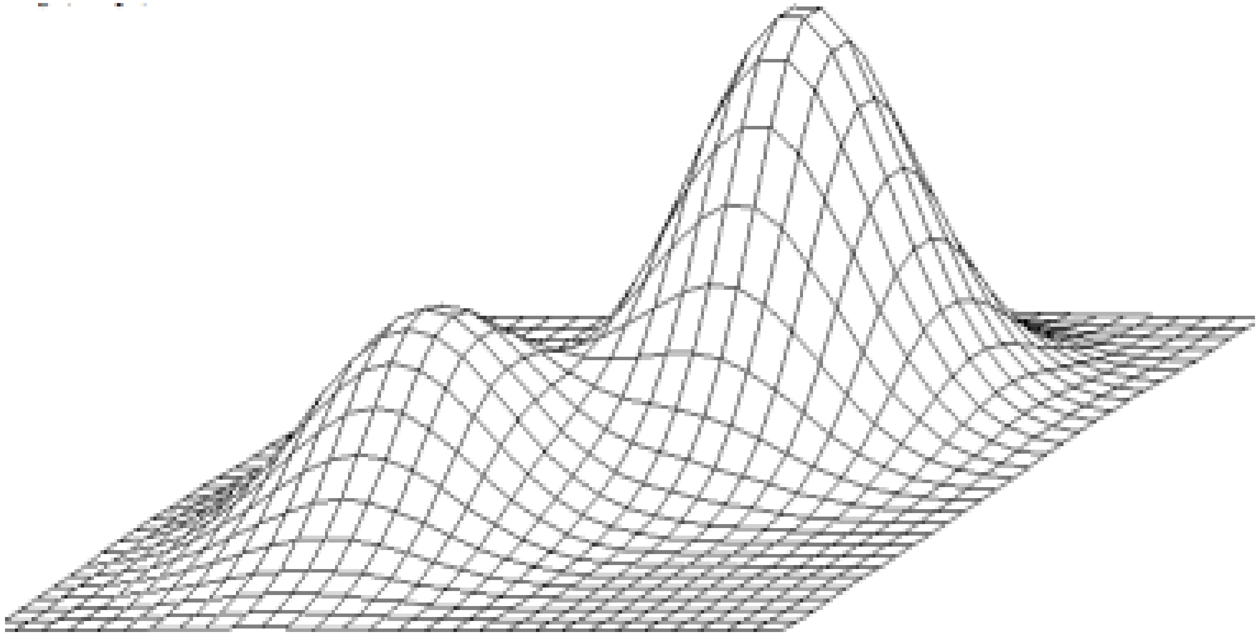
1. Exhaustive search (AKA brute-force search)

- Test all possible solutions, pick the best
- Guaranteed to find the optimal solution
- For TSP: Try every possible ordering of the cities. Need to evaluate N! different solutions
  - For 70 cities, $N! > 10^{100}$. That's more than the number of atoms in the universe.

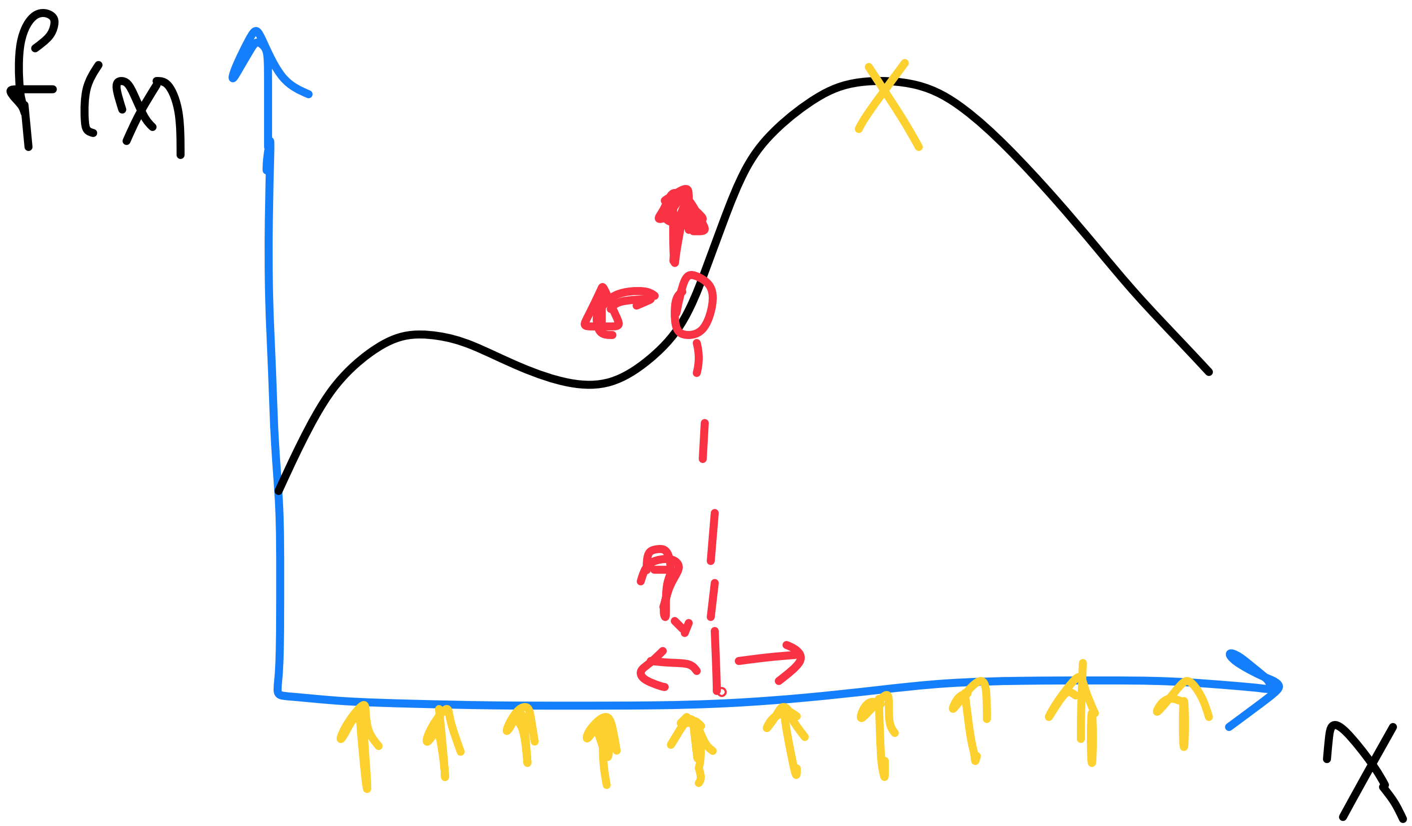
Exhaustive search

Only works for simple discrete problems, but can be approximated in continuous problems

- Sample the space at regular intervals (grid search)
- Sample the space randomly N times



How can we be smarter than exhaustive search?

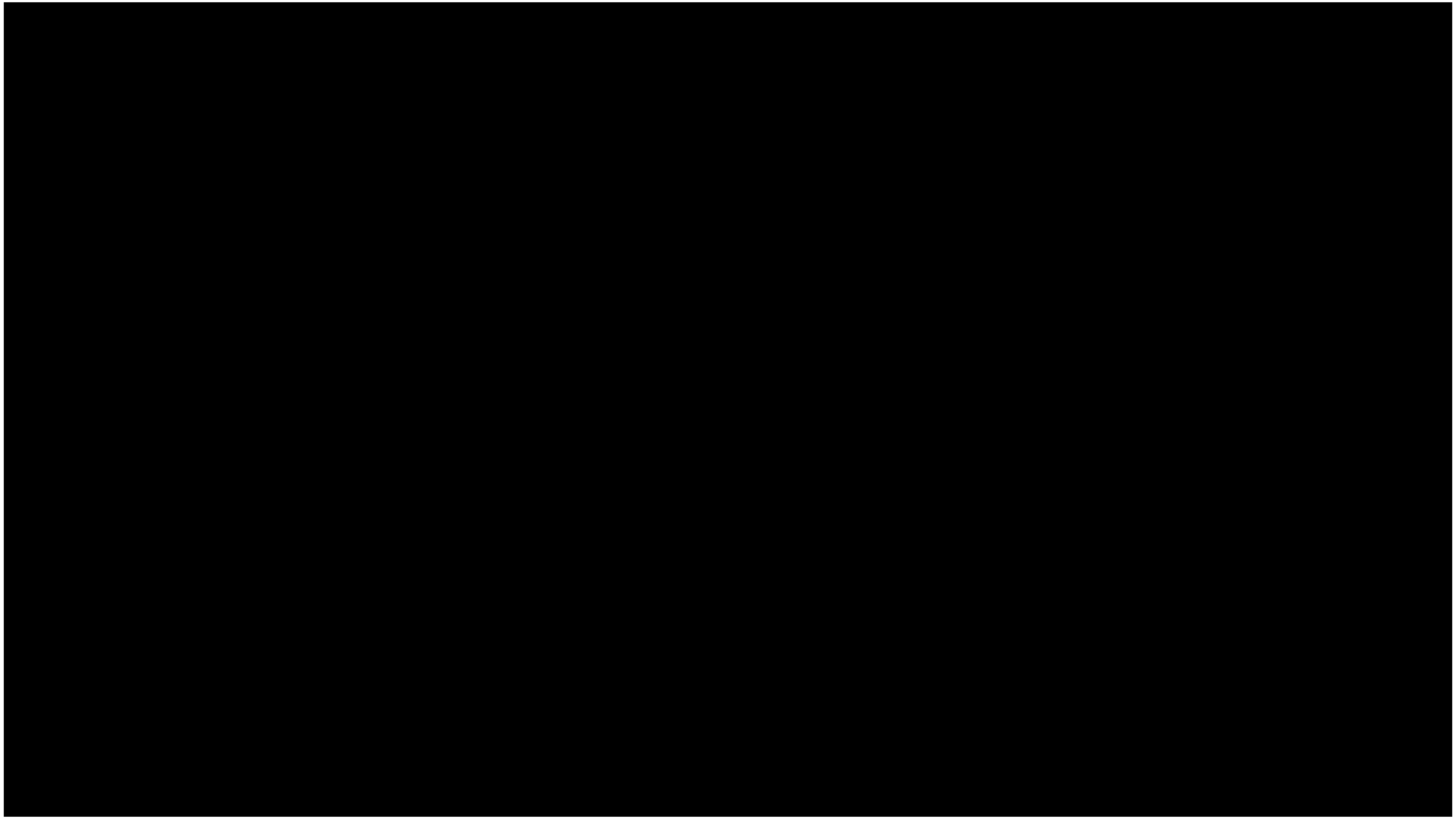


How can we be smarter than exhaustive search?

- Usually, search spaces have some local structure
- Similar solutions often have similar quality
- Making small changes to a solution, and measuring resulting quality, we can gradually move towards better solutions

2. Greedy search

- Only generates and evaluates a single solution
- Makes several locally optimal choices, hoping the result will be near a global optimum
- Details depend on the problem being solved

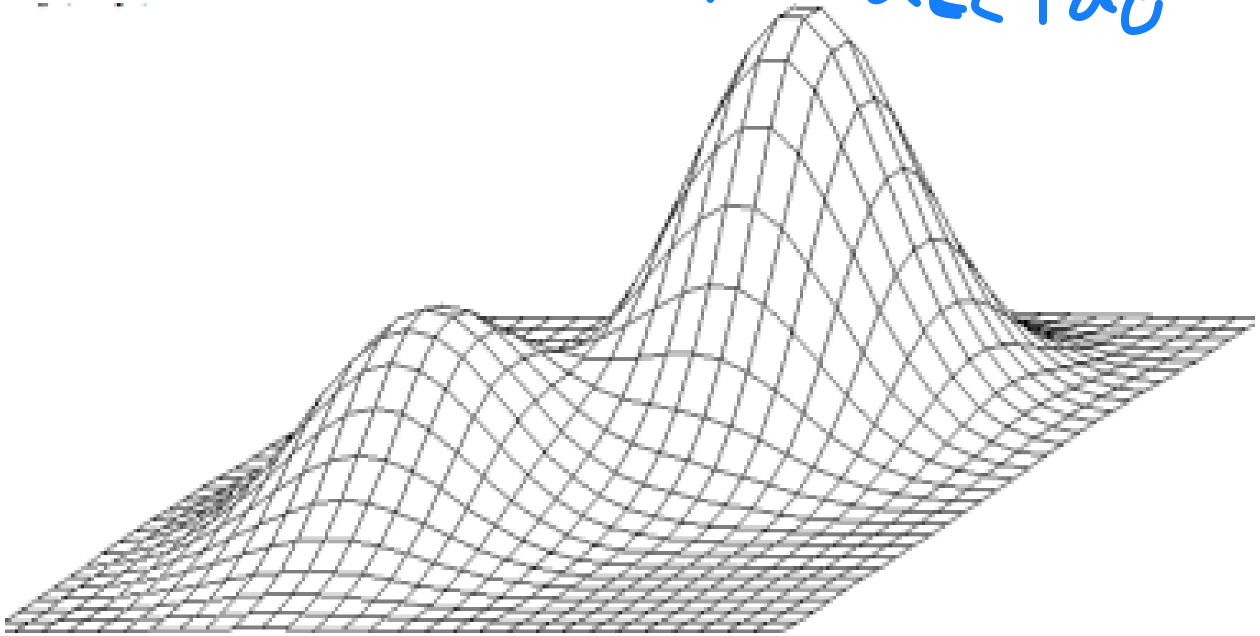


Video from poprythm, at Youtube:
https://www.youtube.com/watch?v=SC5CX8drAtU

Hill climbing

- Pick a solution as the current best (e.g. a random solution)
- Compare to neighbor solution(s) → How to find a neighbor?
  - If the neighbor is better, replace the current best
  - Repeat until we reach a certain number of evaluations

You decide

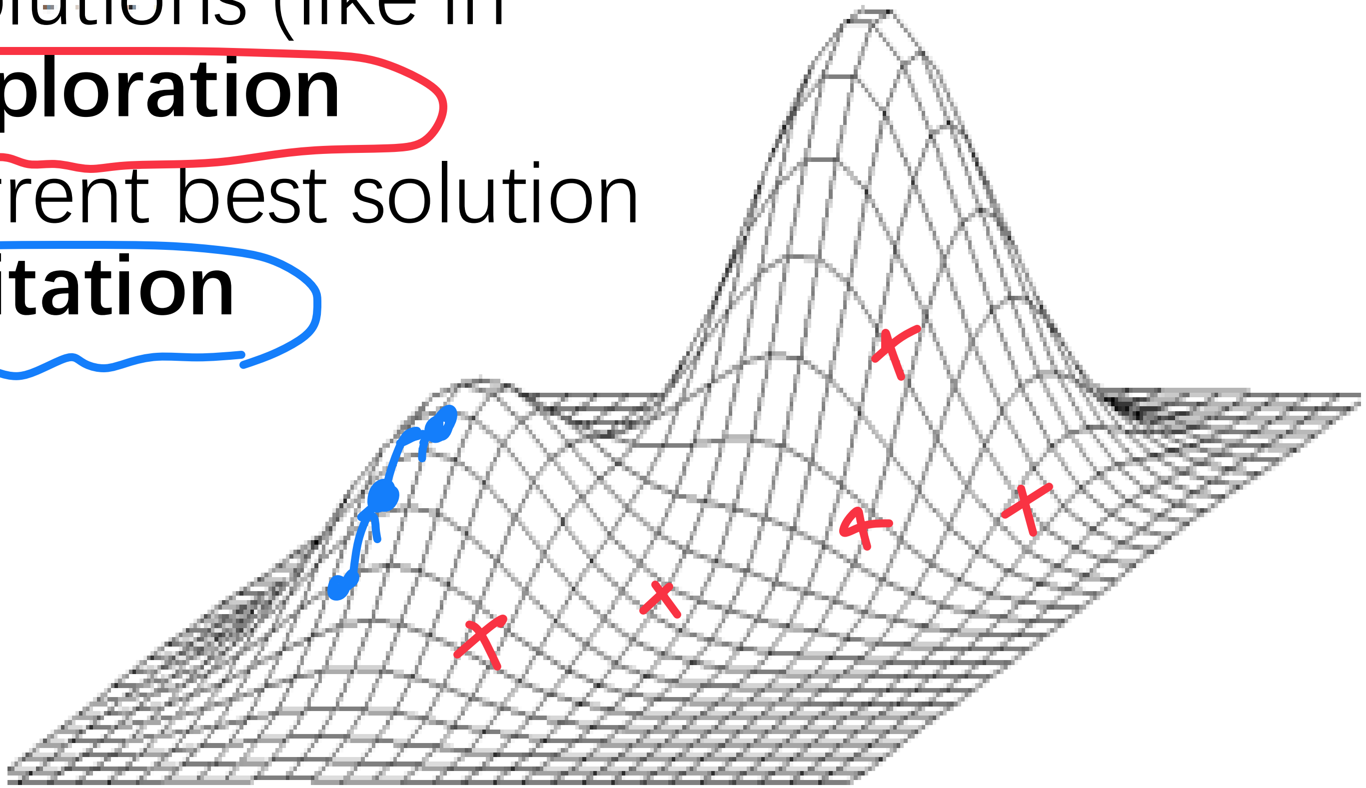


What is the disadvantage/limitation to hillclimbing?

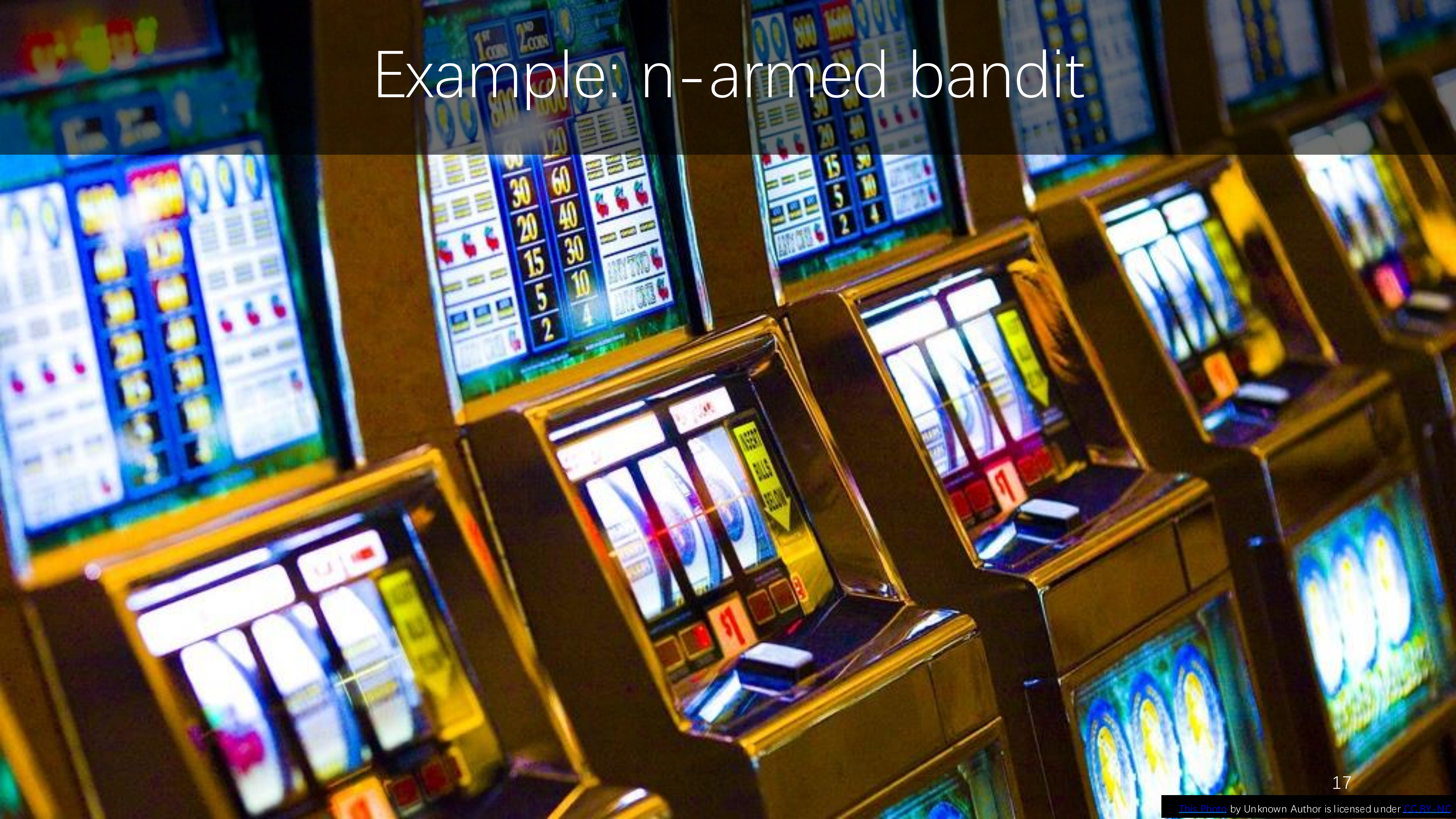
stuck in local optima

Exploitation and Exploration

- Search methods should combine:
  - Trying completely new solutions (like in exhaustive search) => **Exploration**
  - Trying to improve the current best solution by local search => **Exploitation**

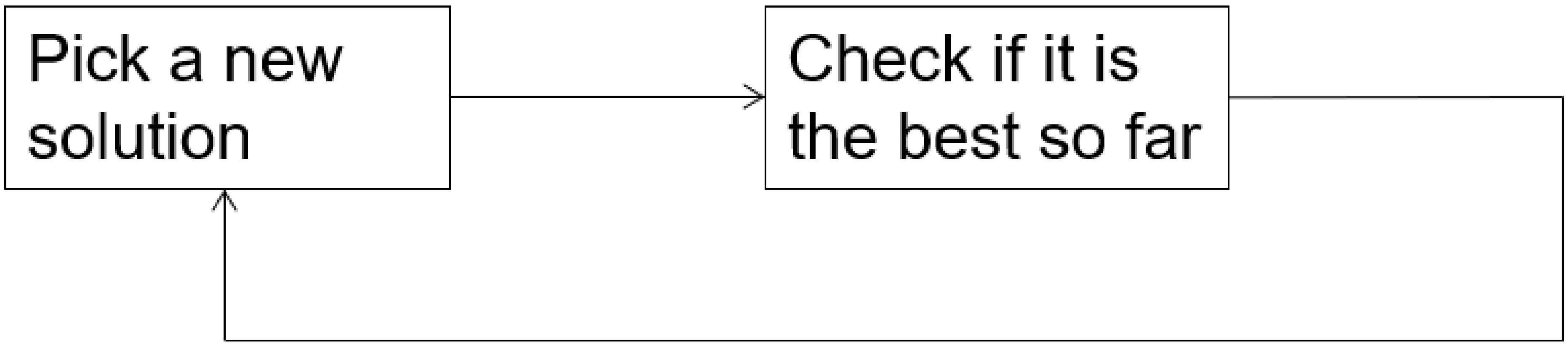


Example: n-armed bandit

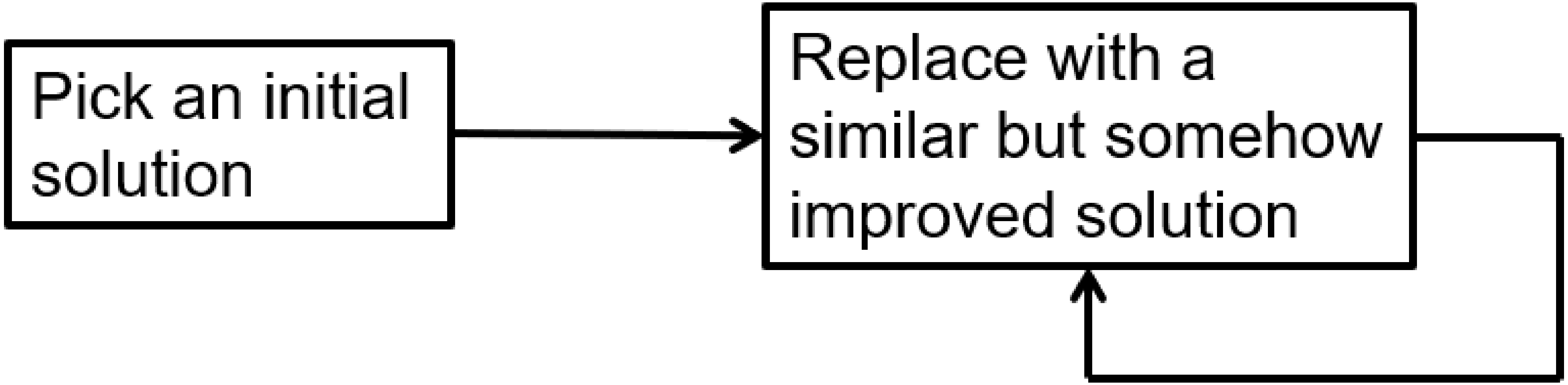



0.8
1.2
0.5
1.6

Exhaustive search – pure exploration

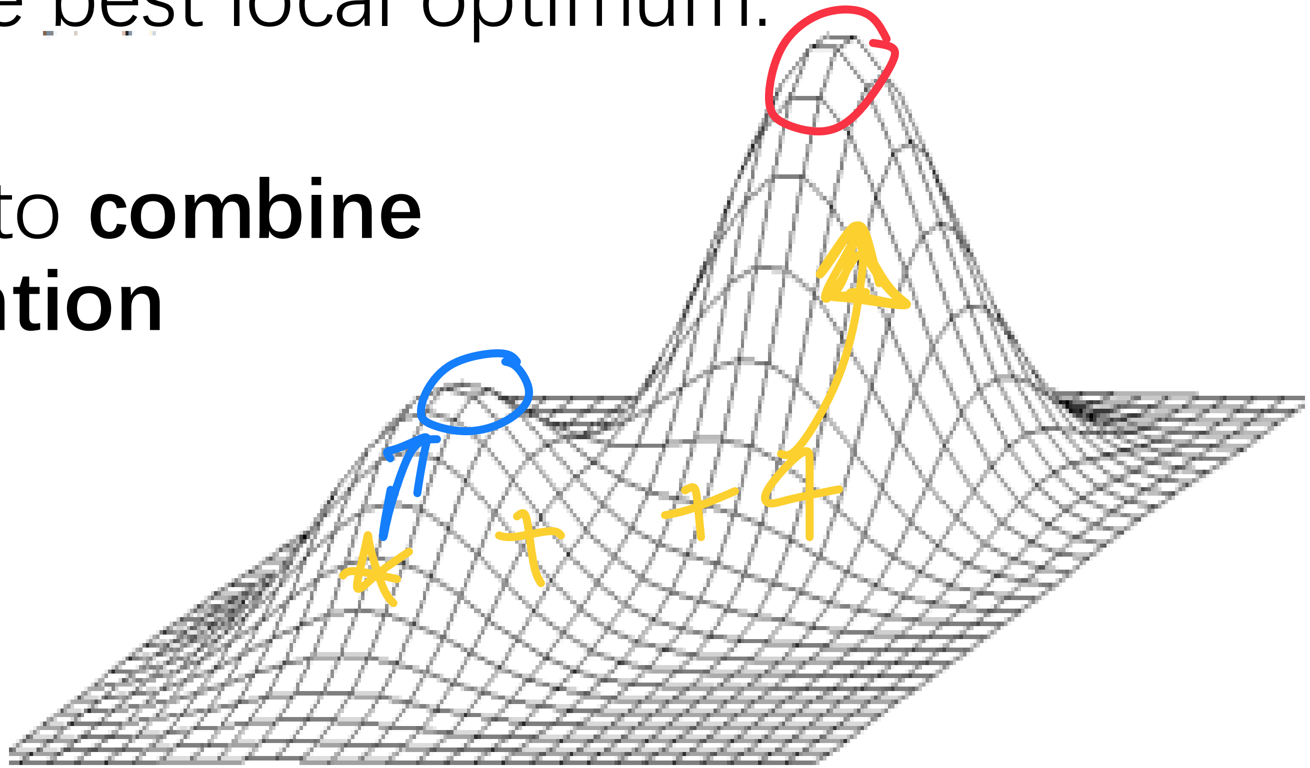


Hill Climbing – pure exploitation



Global optimization

- Most of the time, we must expect the problem to have **many local optima**
- Ideally, we want to find the best local optimum: **the global optimum**
- The best strategy is often to **combine exploration and exploitation**



How can we combine exploration and exploitation?

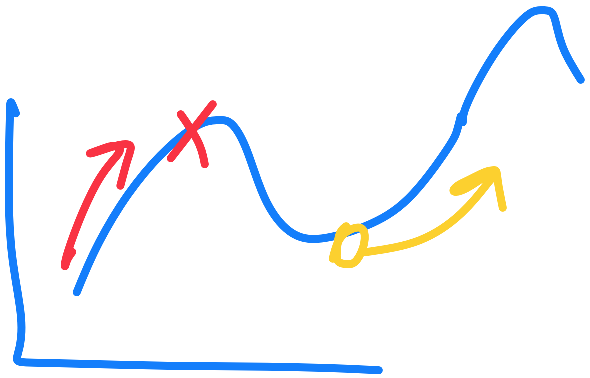
Allow missteps to explore

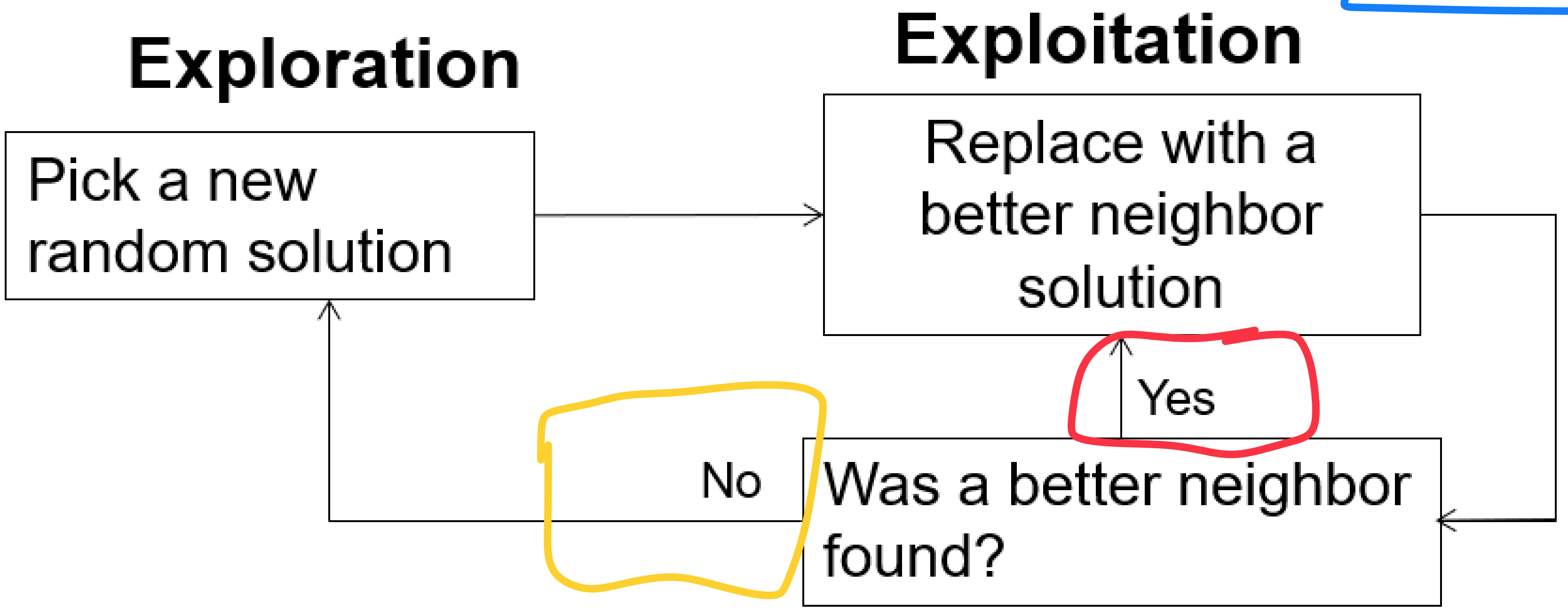
Simulated Annealing

start from a number of random initial solutions

Evolutionary Alg

Mixed solution





Local optima

Algorithms like greedy search, hill climbing and gradient ascent/descent can only find local optima:

- They will only move through a strictly improving chain of neighbors
- Once they find a solution with no better neighbors they stop

Going the wrong way

What if we modified the hill climber to sometimes choose worse solutions?

- Goal: avoid getting stuck in a local optimum
- Always keep the new solution if it is better
- However, if it is worse, we'd still want to keep it sometimes, i.e. with some probability

3. Annealing

A thermal process for obtaining low energy states of a solid in a heat bath:

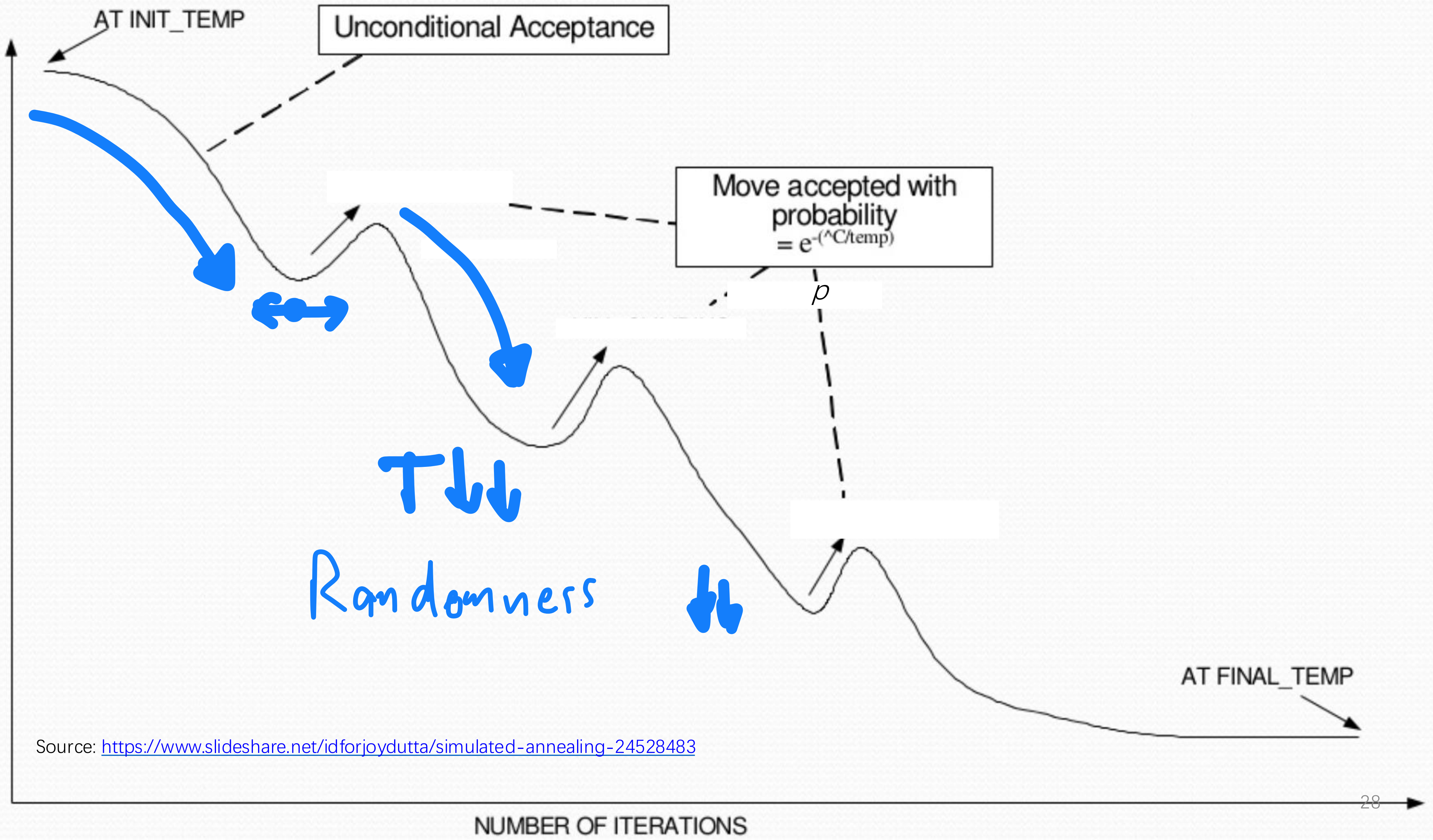
- Increase the temperature of the heat bath to a the point at which the solid melts
- Decrease the temperature slowly
- If done slowly enough, the particles arrange themselves in the minimum energy state

Simulated annealing

high T : high exploration
low T : high exploitation

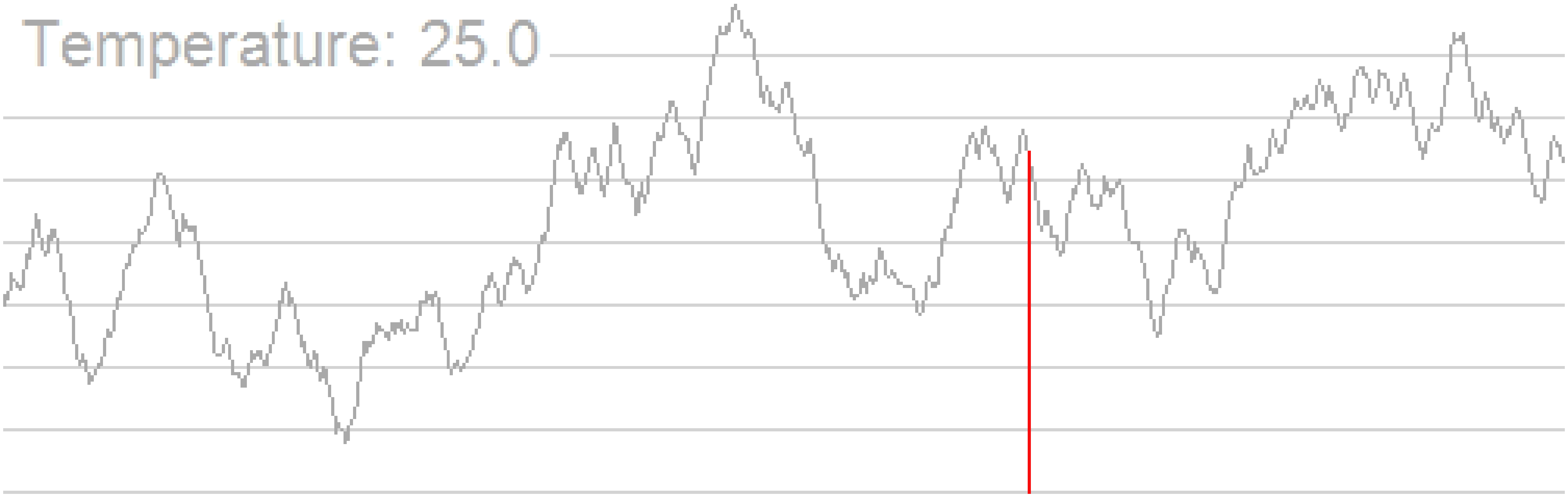
- Set an initial temperature T
- Pick an initial solution
- Repeat:
  - Pick a solution neighboring the current solution
  - If the new one is better, keep it
  - Otherwise, keep the new one with probability p
    - p depends on the **difference in quality** and the **temperature.** high temp -> high p *(more randomness)*
  - Reduce T

exploration



Source: https://www.slideshare.net/idforjoydutta/simulated-annealing-24528483

Simulated Annealing Illustrated



Continuous optimization

- **Mechanics**
  - Optimized design of mechanical shapes etc.
- **Economics**
  - Portfolio selection, pricing
    risk management etc.
- **Control engineering**
  - Process engineering, robotics etc.

4. Gradient ascent / descent

$\underline{x} \in \mathbb{R}^n$

$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

In continuous optimization we may be able to calculate the gradient of $f(x)$:

$f: \mathbb{R}^n \to \mathbb{R}$

gradient: function value change along all directions

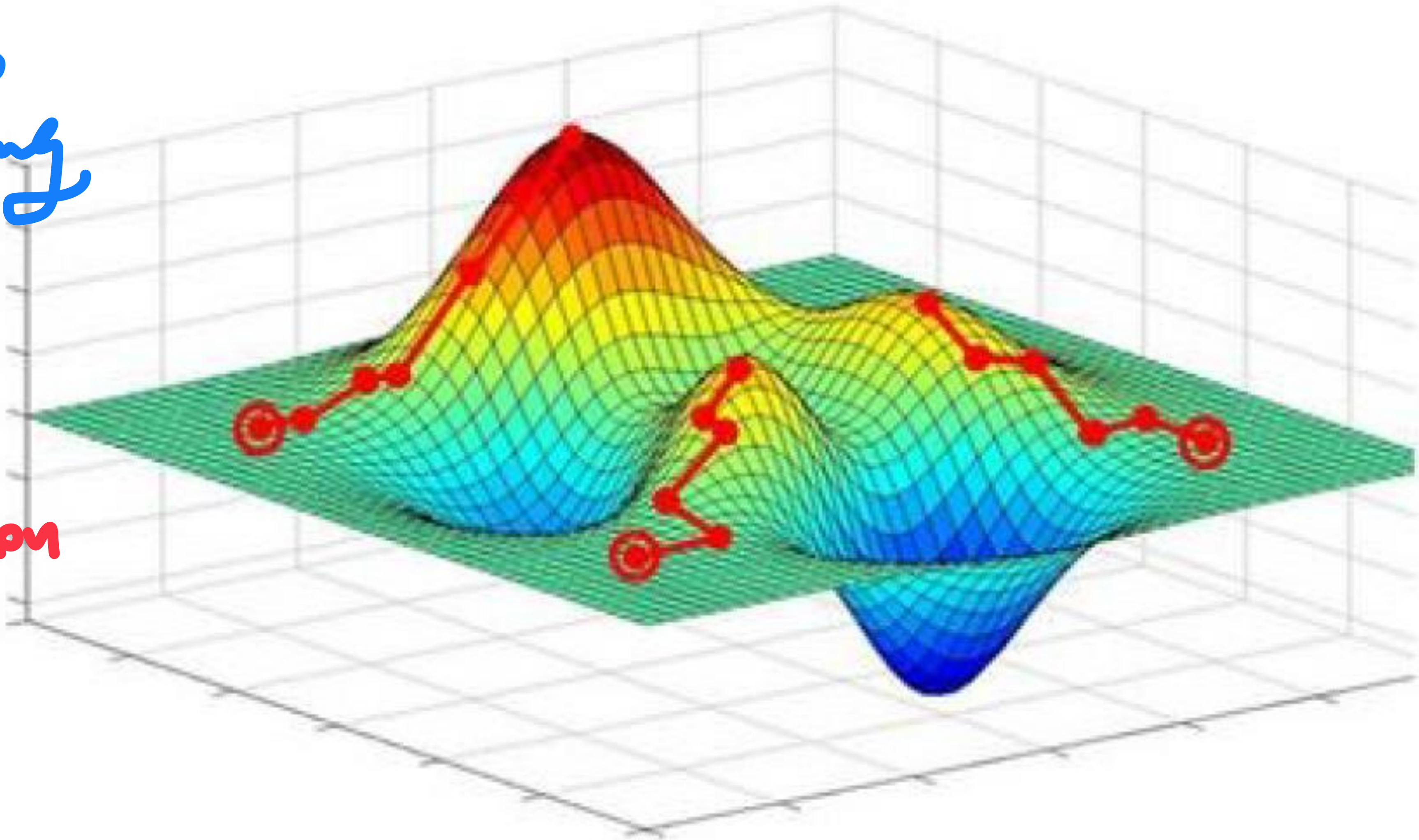
$$\nabla f(x) = \begin{bmatrix} \frac{\delta f(x)}{\delta x_0} \\ \frac{\delta f(x)}{\delta x_1} \\ \vdots \\ \frac{\delta f(x)}{\delta x_n} \end{bmatrix}$$

The gradient tells us in which direction $f(x)$ increases the most

4. Gradient ascent / descent

similar to
hill climbing

gradient
tells us
the direction



Gradient ascent / descent (subtract)

Starting from $x^{(0)}$, we can iteratively find higher $f\left(x^{(k+1)}\right)$ by adding a value proportional to the gradient to $x^{(k)}$:

GA: $$x^{(k+1)} = x^{(k)} + \gamma \nabla f\left(x^{(k)}\right)$$

GD: $x^{(k+1)} \leftarrow x^{(k)} - \gamma \nabla f(x^{(k)})$

↳ step-size or learning rate

Gradient Descent: Algorithm

Start with a point (guess)

Repeat

 Determine a descent direction

 Choose a step

 Update

Until stopping criterion is satisfied

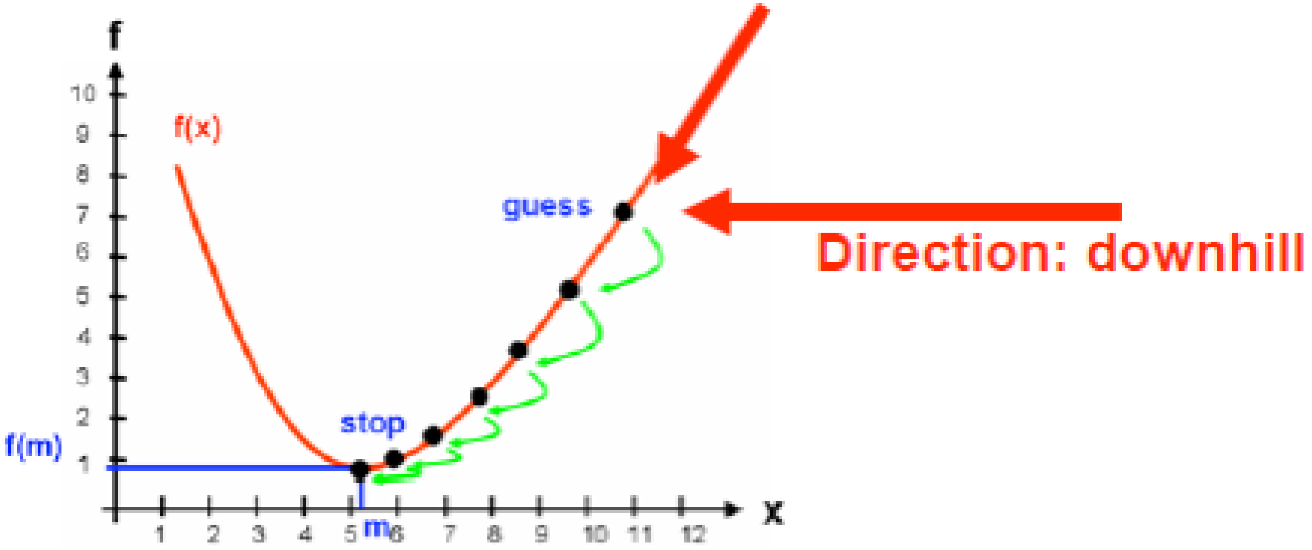


Figure from
http://bayen.eecs.berkeley.edu/sites/default/files//webfm/uploads/class_assets/ce191/lecture10v01_descent2.pdf

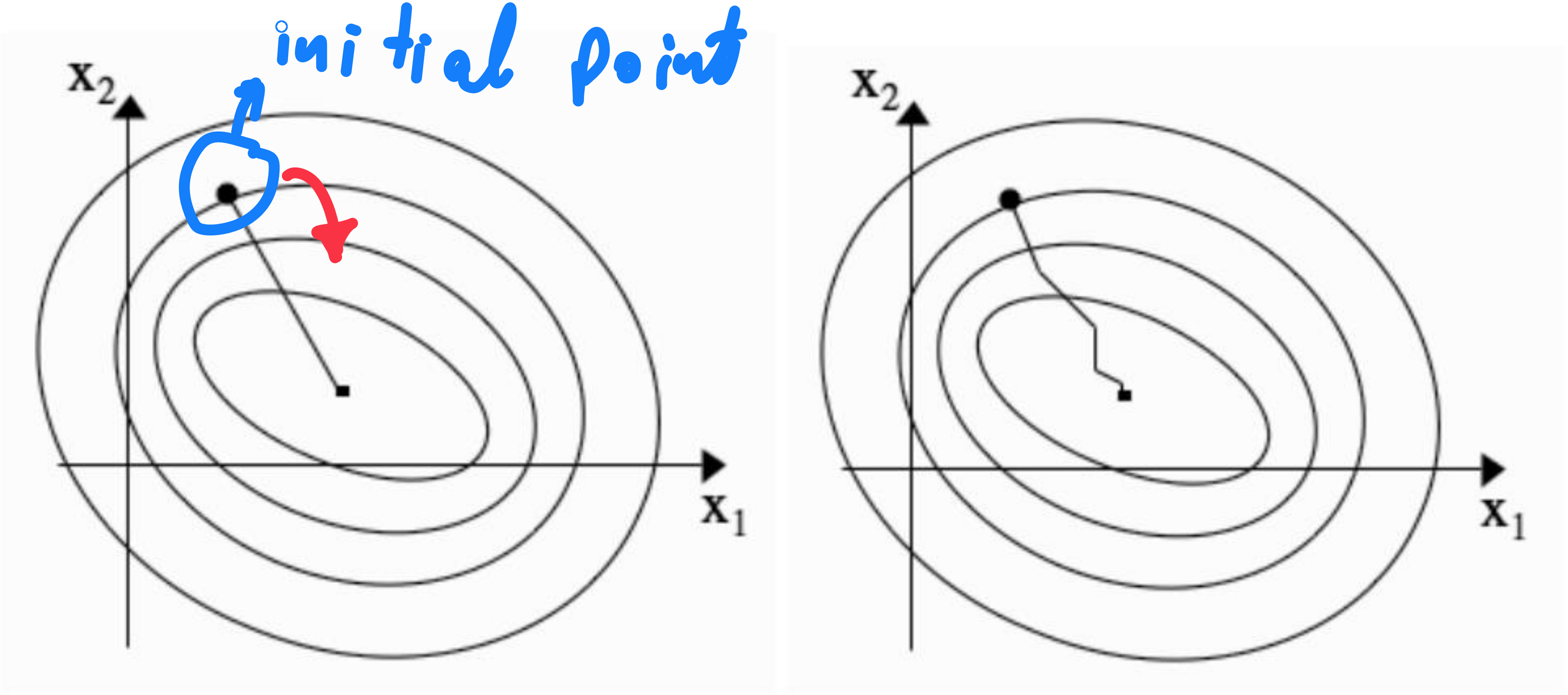


FIGURE 9.3 *Left:* In an ideal world we would know how to go to the minimum directly. In practice, we don't, so we have to approximate it by something like *right:* moving in the direction of steepest descent at each stage.

Source: Marsland

Gradient Descent: Algorithm

γ: how fast we wanna move!

Start with a point (guess)

Repeat

Determine a descent direction

Choose a step (using gradient)

Update

Until stopping criterion is satisfied

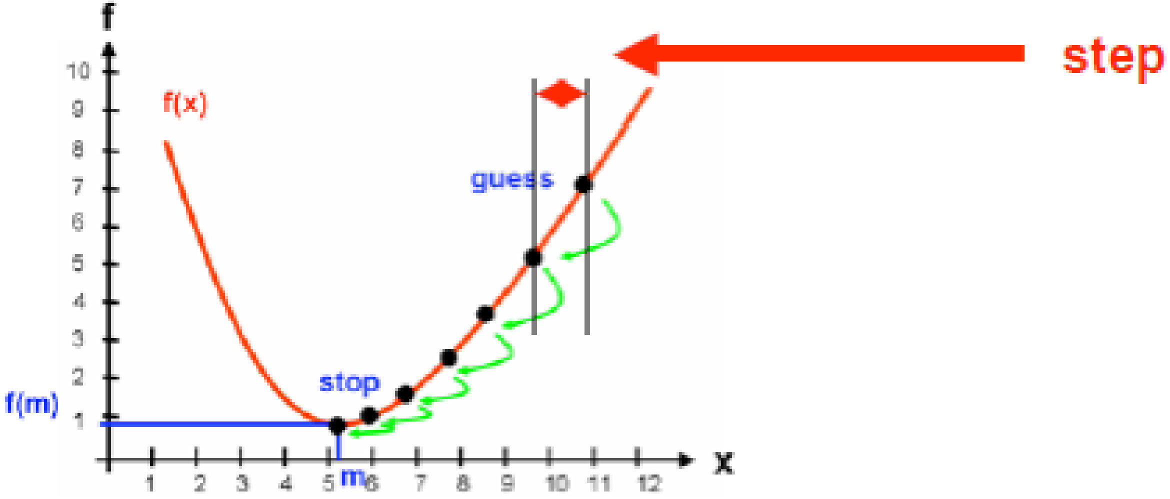


Figure from
http://bayen.eecs.berkeley.edu/sites/default/files//webfm/uploads/class_assets/ce191/lecture10v01_descent2.pdf

Gradient Descent: Algorithm

Start with a point (guess)

Repeat

Determine a descent direction

Choose a step

Update

Until stopping criterion is satisfied

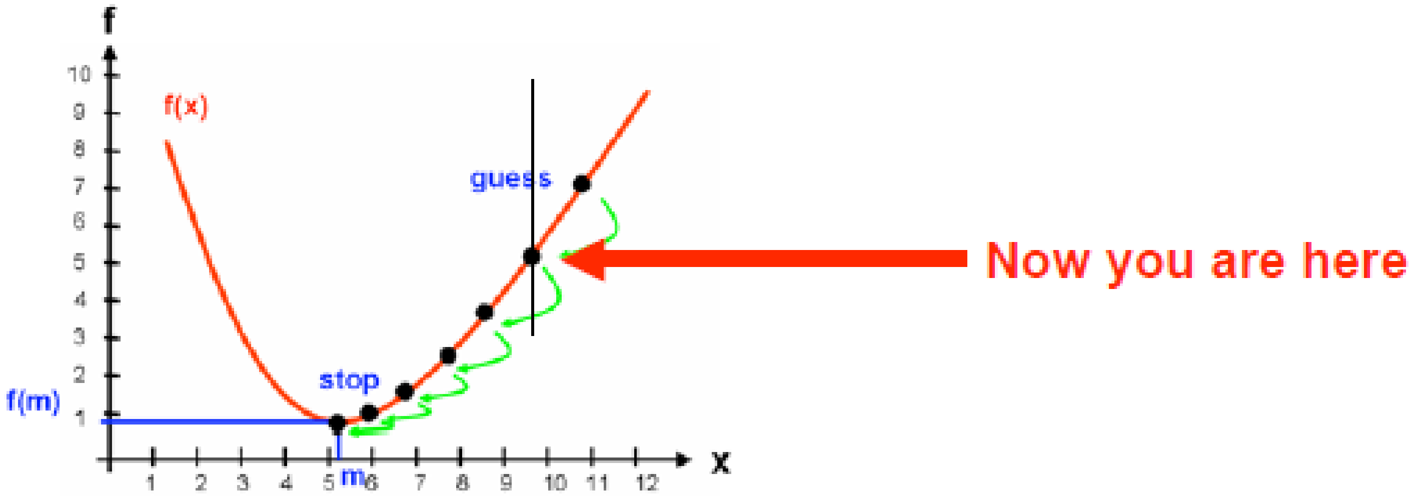


Figure from
http://bayen.eecs.berkeley.edu/sites/default/files//webfm/uploads/class_assets/ce191/lecture10v01_descent2.pdf

Gradient Descent: Algorithm

we adapt & tune γ

Start with a point (guess)

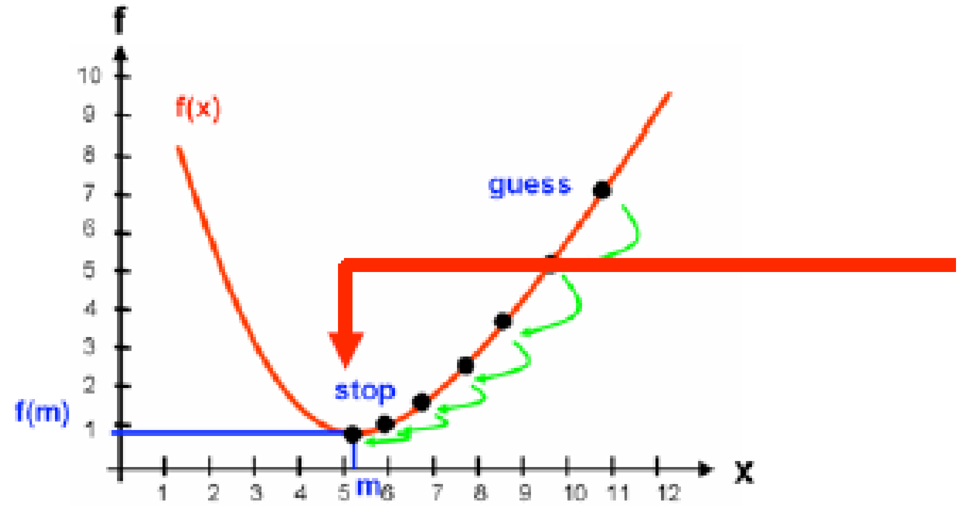
Repeat

Determine a descent direction

Choose a step

Update

Until stopping criterion is satisfied



Stop when "close" from minimum

gradient ≈ 0 & function is flatt to

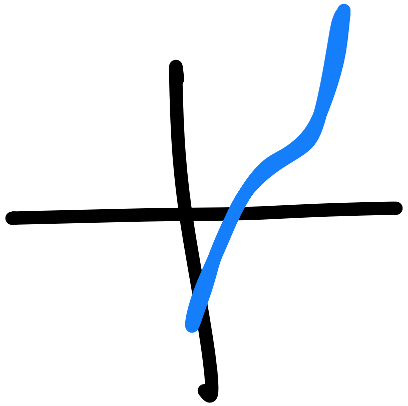
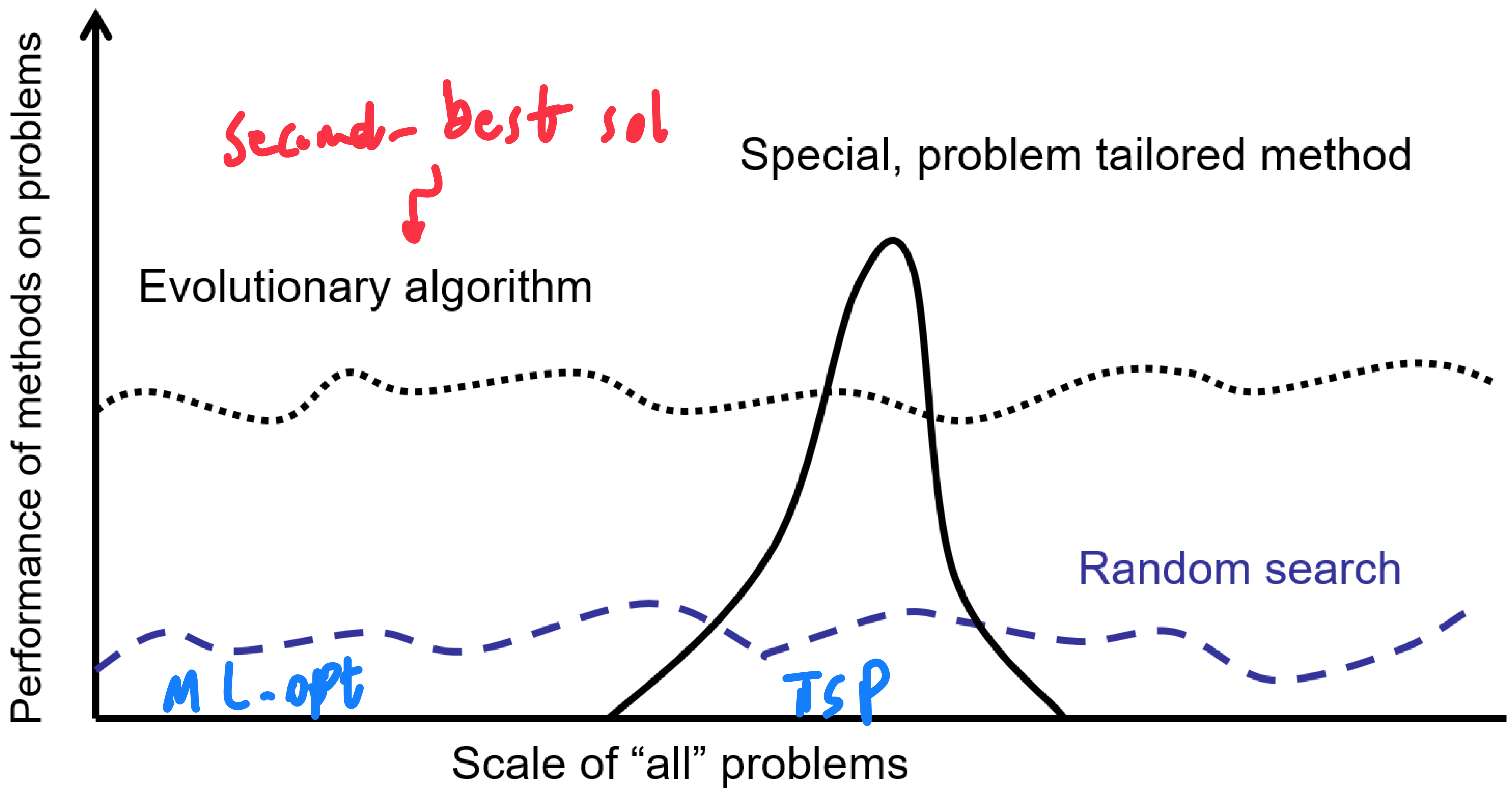


Figure from http://bayen.eecs.berkeley.edu/sites/default/files//webfm/uploads/class_assets/ce191/lecture10v01_descent2.pdf

“No Free Lunch” Theorem

- No search method is **best for all problems**
- Choose the method and **search operators** that suits your problem
- There are however some algorithms that aim to do well **across a range of problems**
  - Evolutionary algorithms are one example



Performance of methods on problems
second-best sol
Special, problem tailored method
Evolutionary algorithm
Random search
ML-opt
TSP
Scale of "all" problems

Summary

- Two classes of problems in optimization:
  - Discrete and Continuous
- Optimization methods:
  - Exhaustive search,
  - Greedy search
  - hill climbing
  - simulated annealing
  - gradient descent
- Exploration vs exploitation