UiO : Department of Informatics
University of Oslo

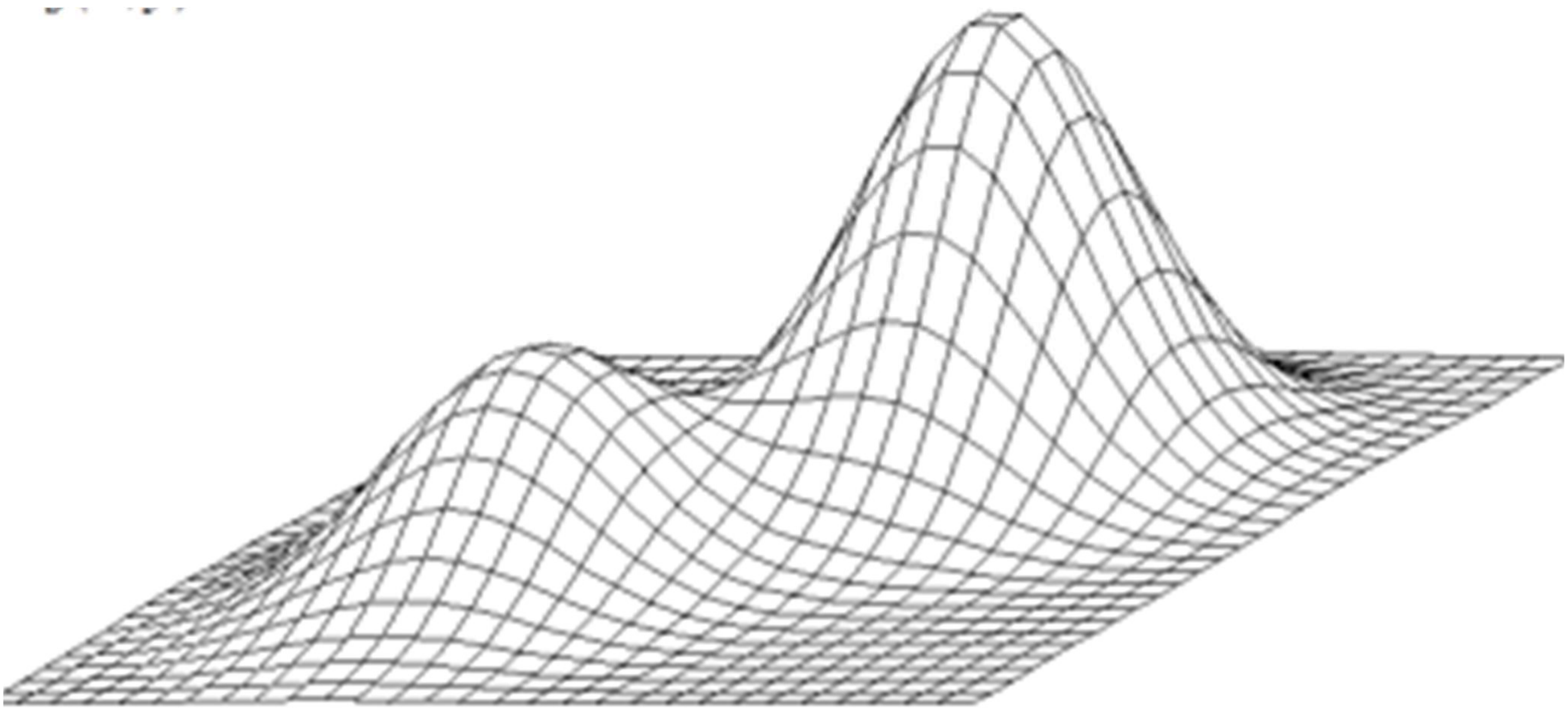
INF3490 - Biologically inspired computing

Lecture 1: Marsland chapter 9.1, 9.4-9.6

2017

Optimization and Search

Kai Olav Ellefsen



Optimization

We need

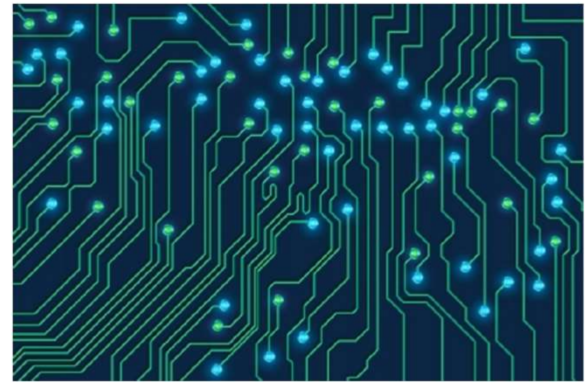
- A numerical representation x for all possible solutions to the problem
- A function $f(x)$ that tells us how good solution x is
- A way of finding
  - $\max_x f(x)$ if bigger $f(x)$ is better (benefit)
  - $\min_x f(x)$ if smaller $f(x)$ is better (cost)

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.

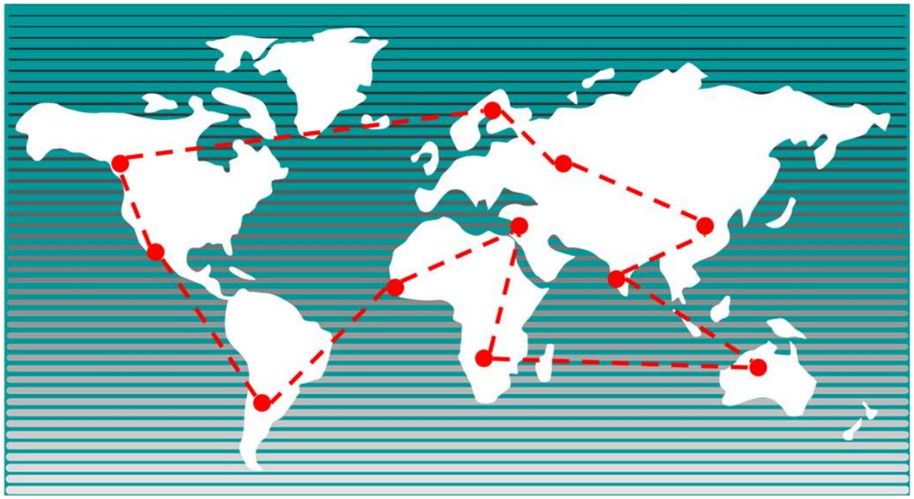
Discrete optimization



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

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?

- 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

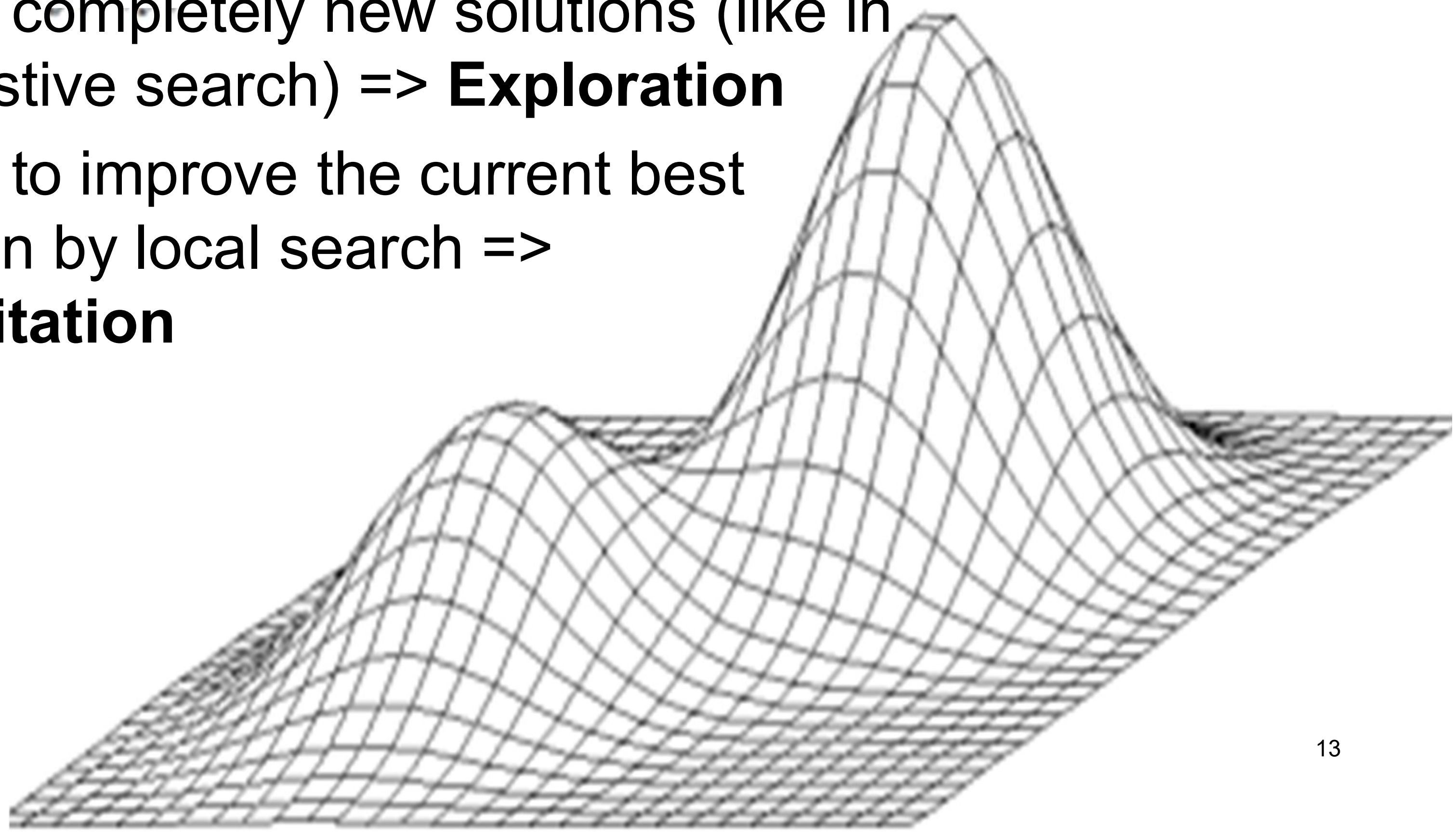
- Only generates and evaluates a single solution
- Makes several locally optimal choices, hoping the result will be near a global optimum
- Details depend on the problem being solved

Hill climbing

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

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

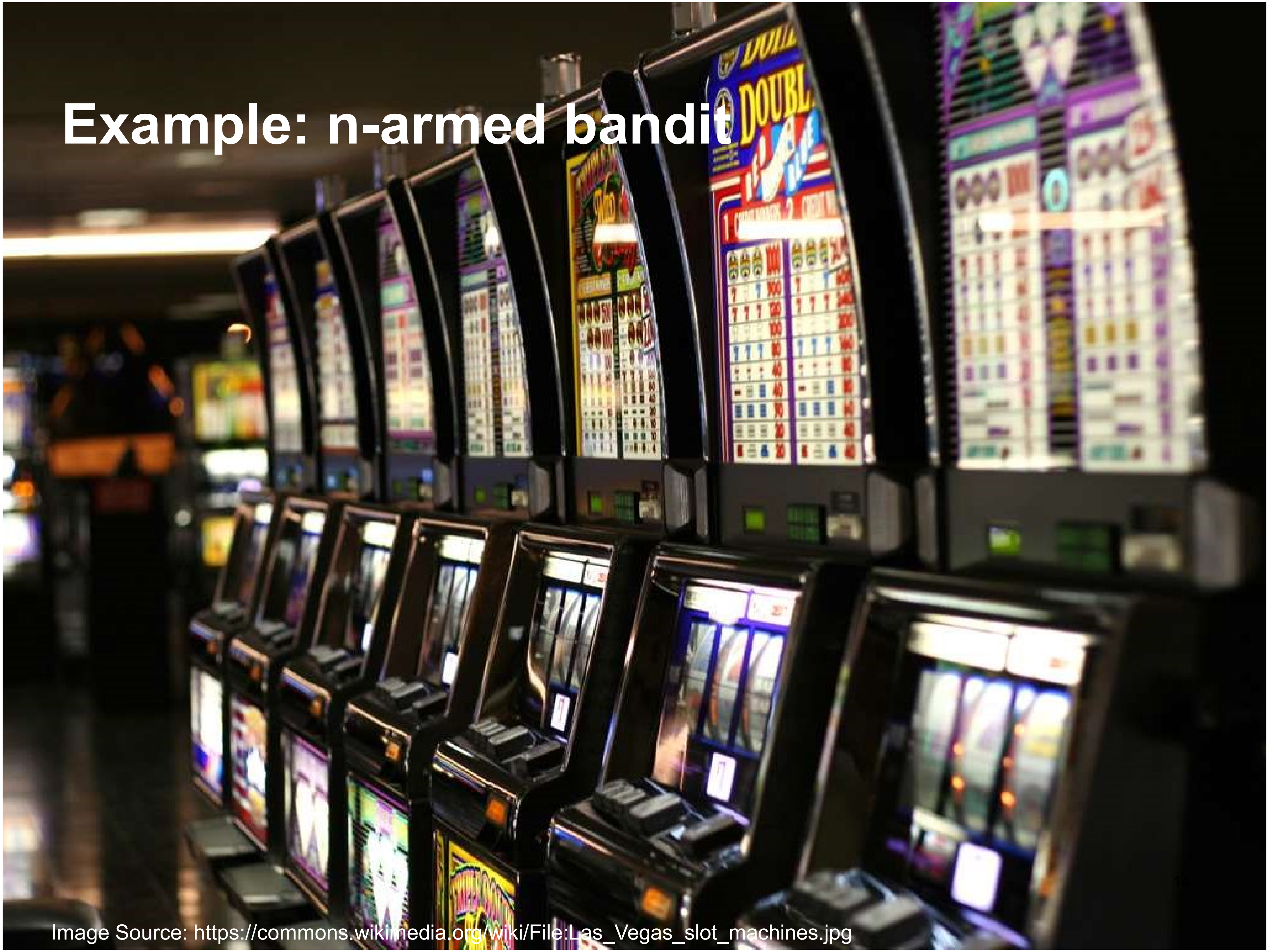


Image Source: https://commons.wikimedia.org/wiki/File:Las_Vegas_slot_machines.jpg

Example: n-armed bandit

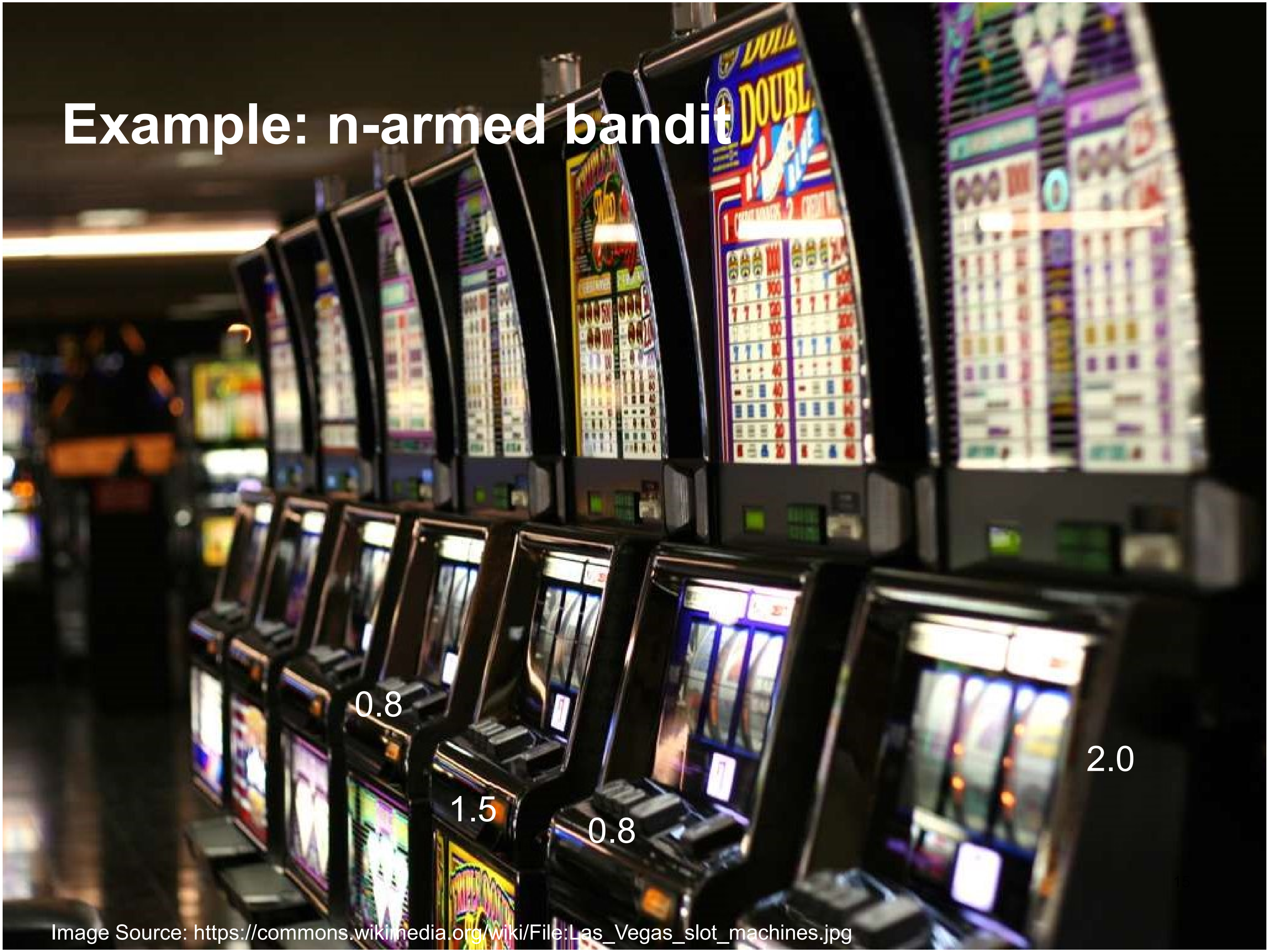
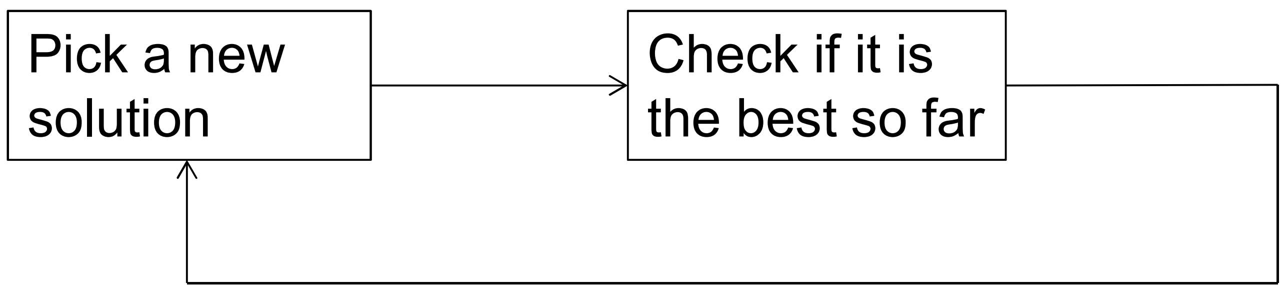
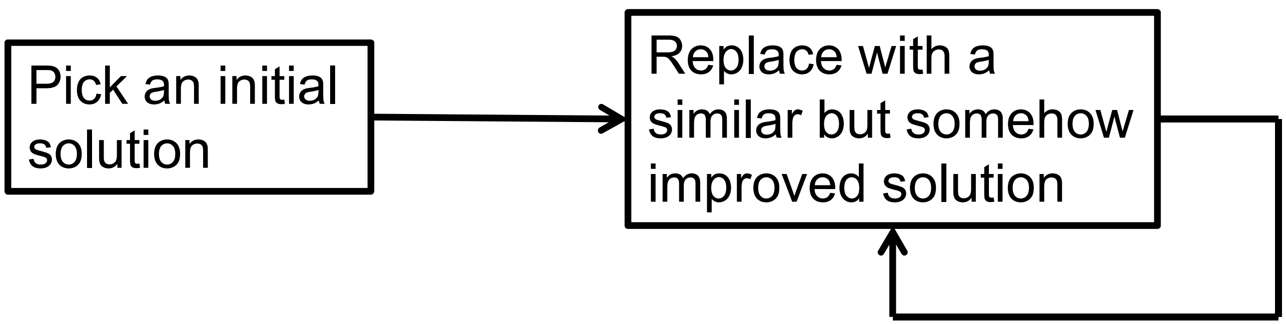


Image Source: https://commons.wikimedia.org/wiki/File:Las_Vegas_slot_machines.jpg

Exhaustive search – pure exploration

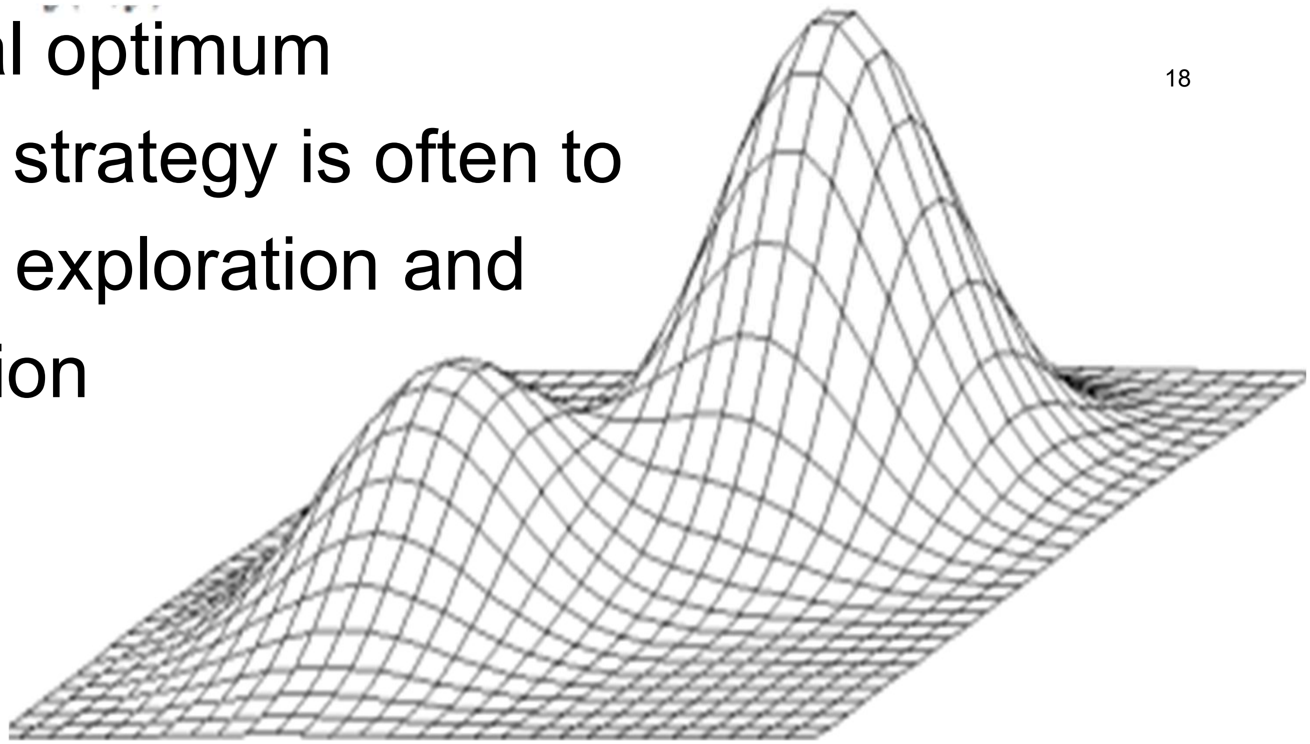


Hill Climbing – pure exploitation



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



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

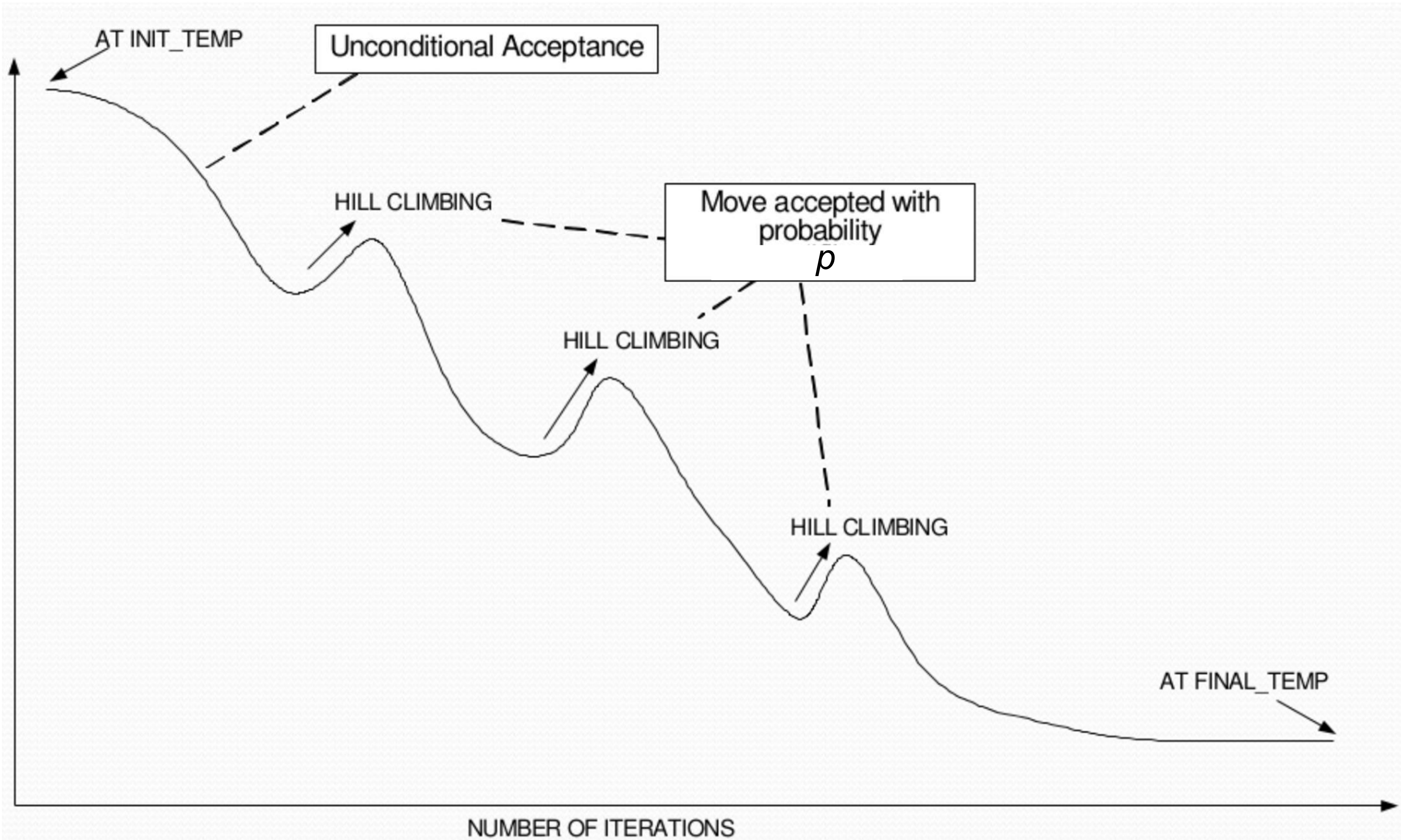
3. Annealing

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

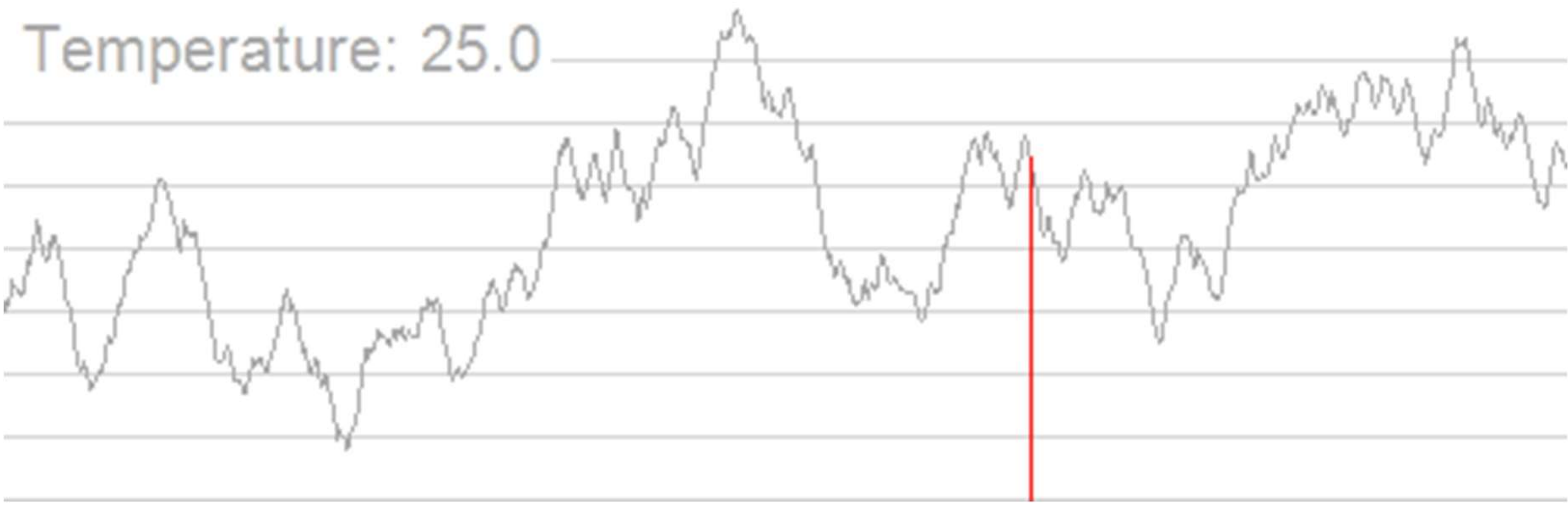
Simulated annealing

- 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



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 options, risk management etc.
- **Control engineering**
  - Process engineering, robotics etc.

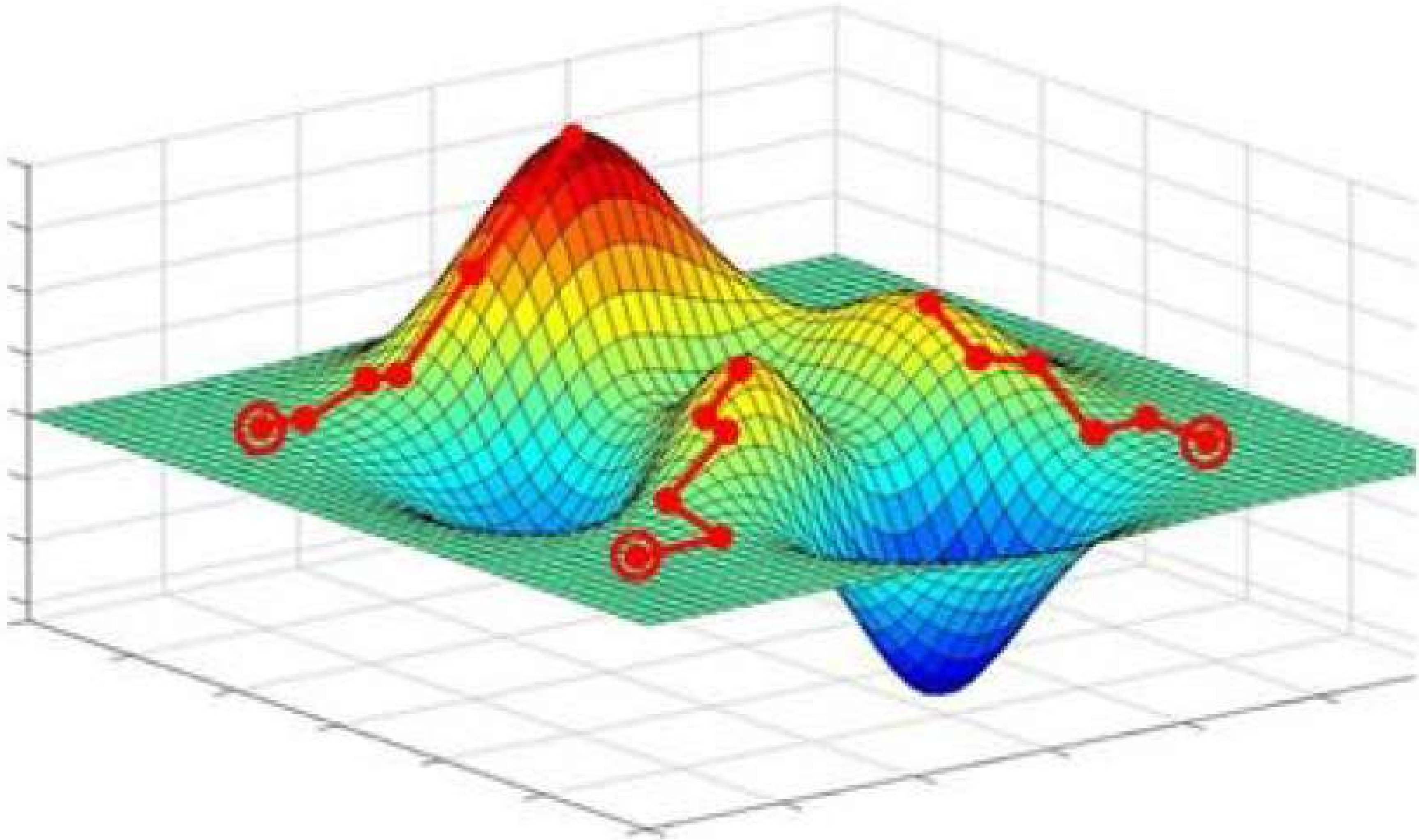
4. Gradient ascent / descent

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

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

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

4. Gradient ascent / descent



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 $x^{(k)}$:

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

https://thenextweb.com/contributors/2018/08/04/ai-experts-favorite-algorithms-siraj-raval/

Gradient Descent: Algorithm

Start with a point (guess)

Repeat

Determine a descent direction

Choose a step

Update

Until stopping criterion is satisfied

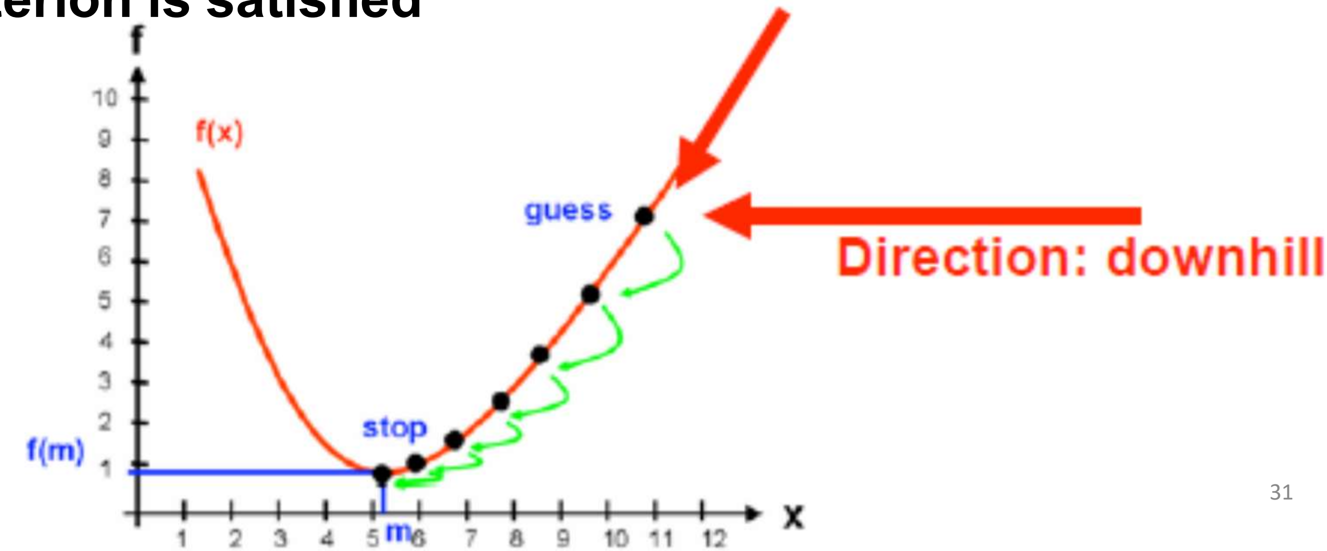


Figure from
http://bayen.eecs.berkeley.edu/sites/default/files//webfm/uploads/class_assets/ce191/lecture10v01_descent2.pdf

Gradient Descent: Algorithm

Start with a point (guess)

Repeat

Determine a descent direction

Choose a step (using gradient)

Update

Until stopping criterion is satisfied

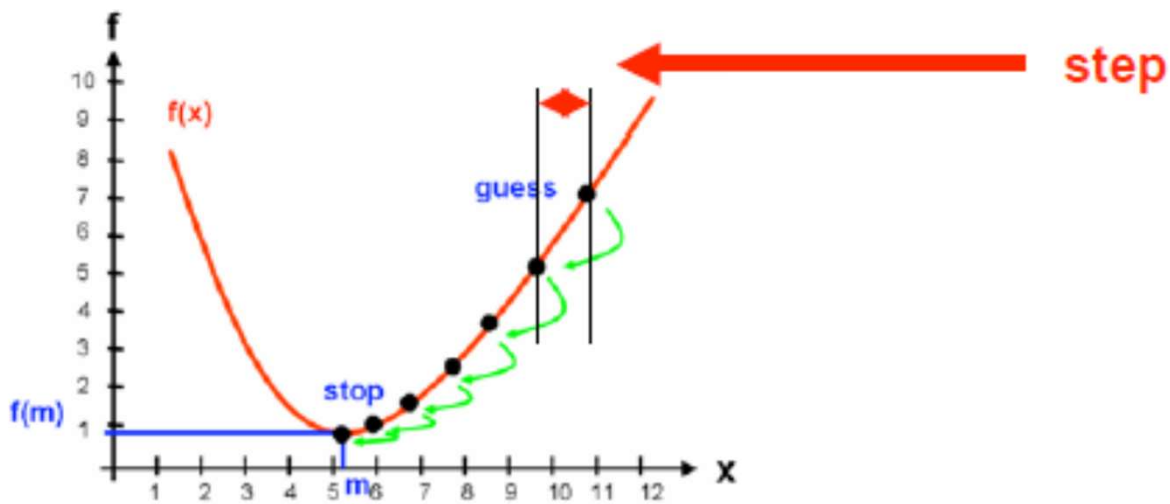


Figure from
http://bayen.eecs.berkeley.edu/sites/default/files//webfm/uploads/class_assets/ce191/lecture10v01_descent2.pdf

Gradient Descent: Algorithm

Start with a point (guess)

Repeat

Determine a descent direction

Choose a step

Update

Until stopping criterion is satisfied

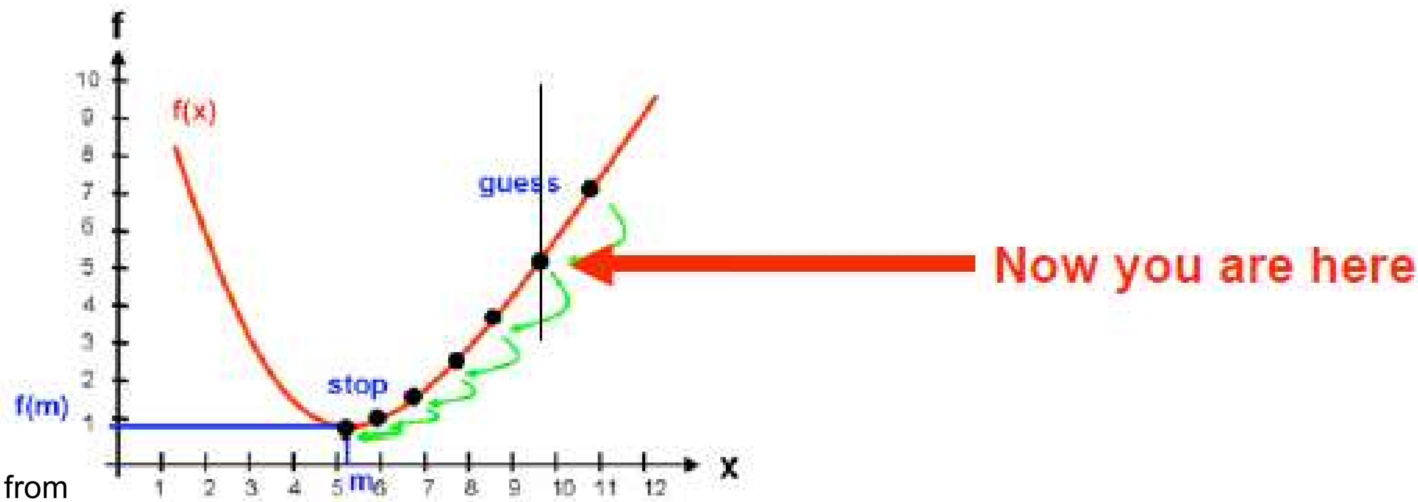


Figure from
http://bayen.eecs.berkeley.edu/sites/default/files//webfm/uploads/class_assets/ce191/lecture10v01_descent2.pdf

Gradient Descent: Algorithm

Start with a point (guess)

Repeat

Determine a descent direction

Choose a step

Update

Until stopping criterion is satisfied

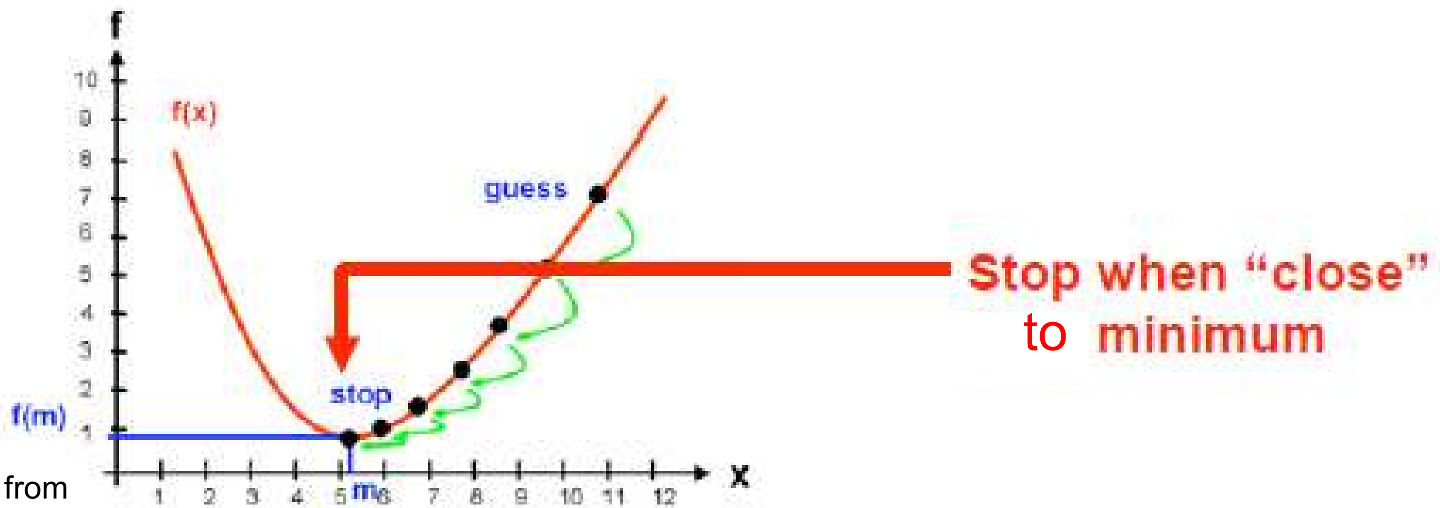


Figure from
http://bayen.eecs.berkeley.edu/sites/default/files//webfm/uploads/class_assets/ce191/lecture10v01_descent2.pdf

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