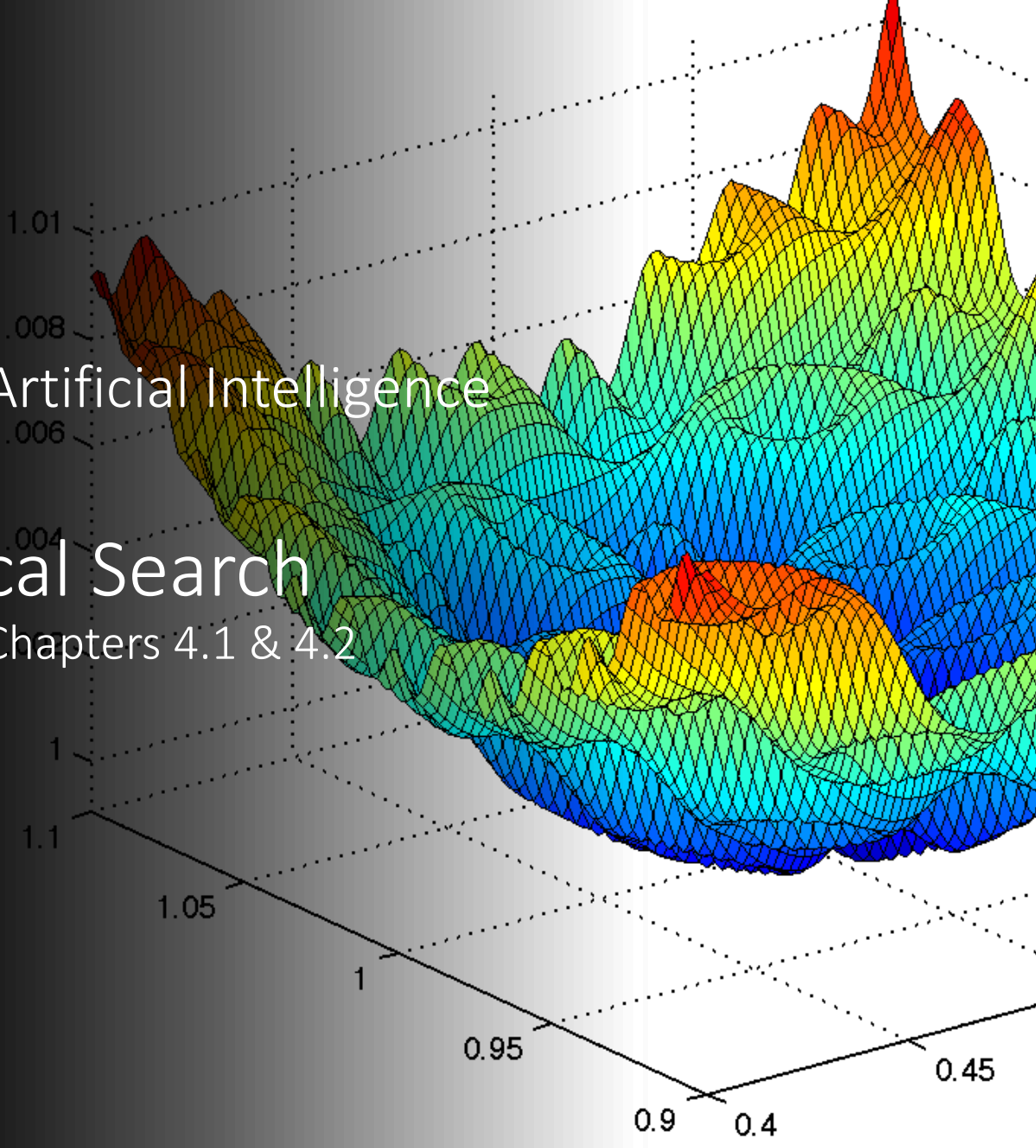
Discussion

Advanced Artificial Intelligence

Local Search

AIMA Chapters 4.1 & 4.2



Recap: Uninformed and Informed Search

Tries to **plan** the

best path

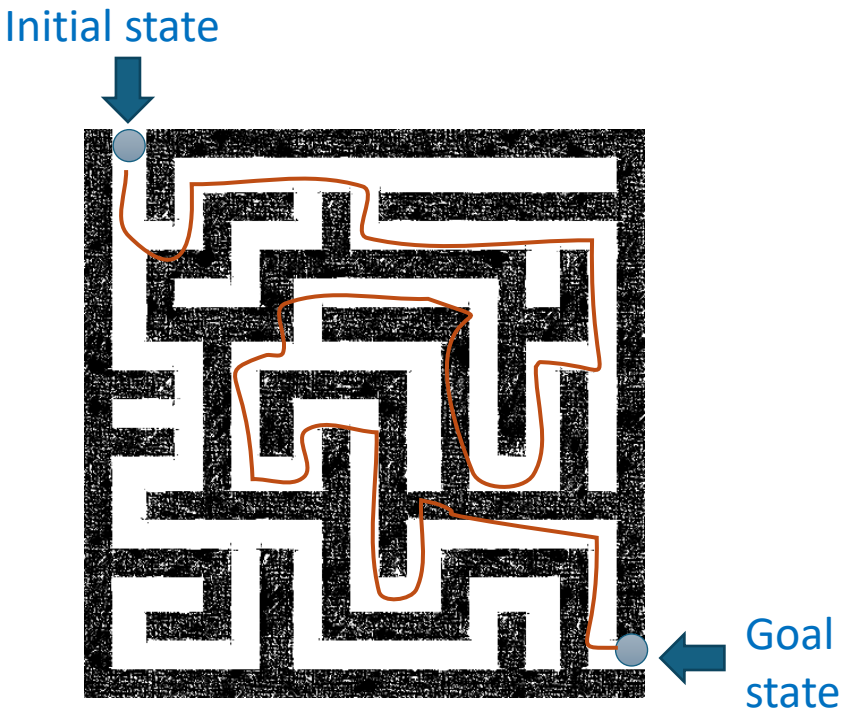
from a

given initial state

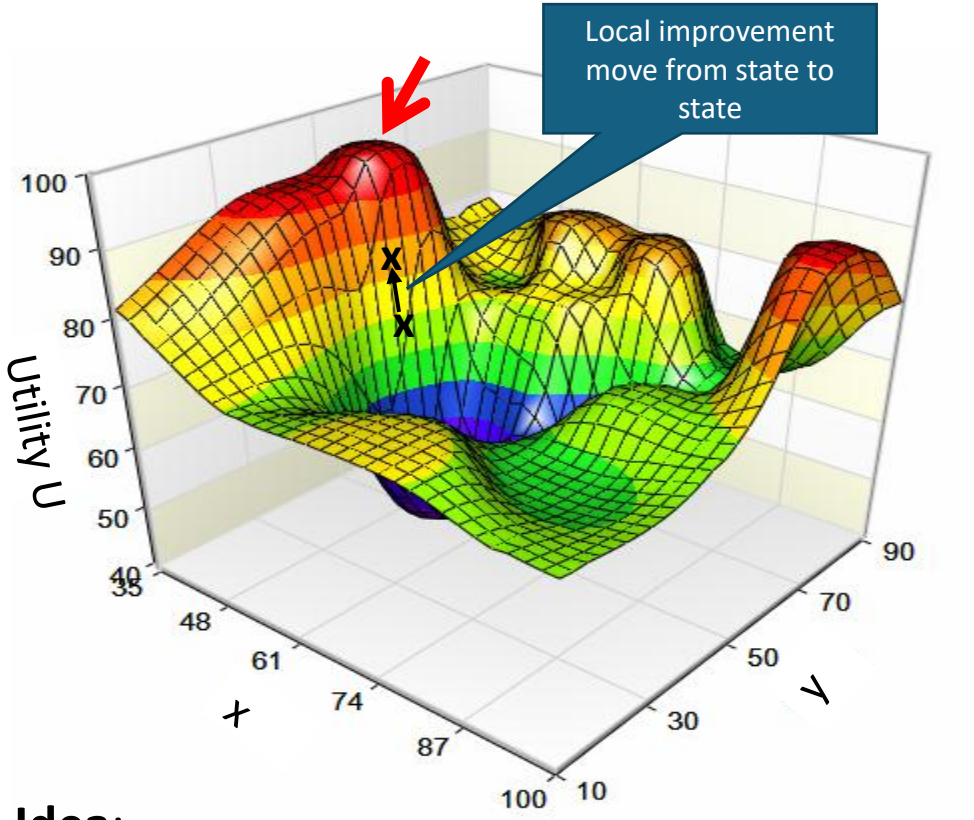
to a

given goal state.

- Often comes with completeness and optimality guarantees (BFS, A* Search, IDS).
- **Issue**: Typically have to search a large portion of the search space and therefore need a lot of time and memory.



Local Search



- Assume we know the utility of each possible state given by a utility function
$$U = u(s)$$
- We use a factored state description. Here $s = (x, y)$
- How can we identify the best or at least a "good" state?
- This is the **optimization problem**:
$$s^* = \underset{s \in S}{\operatorname{argmax}}\, u(s)$$
- We need a fast and memory-efficient way to find the best/a good state.

Idea:

Start with a current solution (a state) and improve the solution by moving from the current state to a "neighboring" better state (a.k.a. performing a series of **local moves**).

Hill-Climbing Search (Greedy Local Search)

Maximization

```
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
```

We often start with a random state

Use ≥ for minimization

Variants:

- **Steepest ascent hill climbing:** Check all possible successors and choose the highest-valued successor.
- **Stochastic hill climbing:** Choose randomly among **all** uphill (improvement) moves.
- **First-choice stochastic hill climbing**: Generate randomly **one** new successor at a time and only move to better ones. This is what people often mean by "stochastic hill climbing." It is equivalent to a, but computationally much cheaper.

Minimization

$h = 17$

This is what people often mean by “stochastic hill climbing.”

Stochastic Hill Climbing

- **First-choice stochastic hill climbing**: Generate randomly **one** new successor at a time and only move to better ones.
- **Steps:**

Minimization

$h = 17$

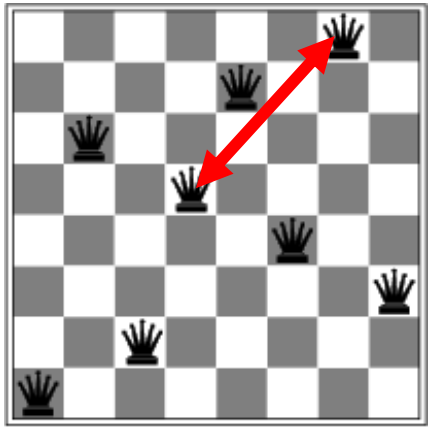
- Why is the result equivalent to stochastic hill climbing that calculates the heuristic for all moves?

Local Optima

Hill-climbing search is like greedy best-first search with the objective function as a (maybe not admissible) heuristic. It only stores the current state (has no frontier data structure) and just stops at a dead end.

Is it complete/optimal?

- No – can get stuck in local optima.



$h = 1$

Example: local optimum for the 8-queens problem. No single queen can be moved within its column to improve the objective function.

Simple approach that can help with local optima:

Random-restart hill climbing: Restart hill-climbing many times with random initial states and return the best solution. This strategy can be used for any stochastic (i.e., randomized) algorithm.

Simulated Annealing Algorithm

- Use first-choice stochastic hill climbing + escape local minima by allowing some "bad" moves but gradually decreasing their frequency as we get closer to the solution.
- Annealing tries to reach a low energy state: A negative ΔE means the solution gets better.
- The probability of accepting "bad" moves follows the annealing schedule, which reduces the temperature T over time t.

Maximization

```
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)
```

Typically, we start with a random state

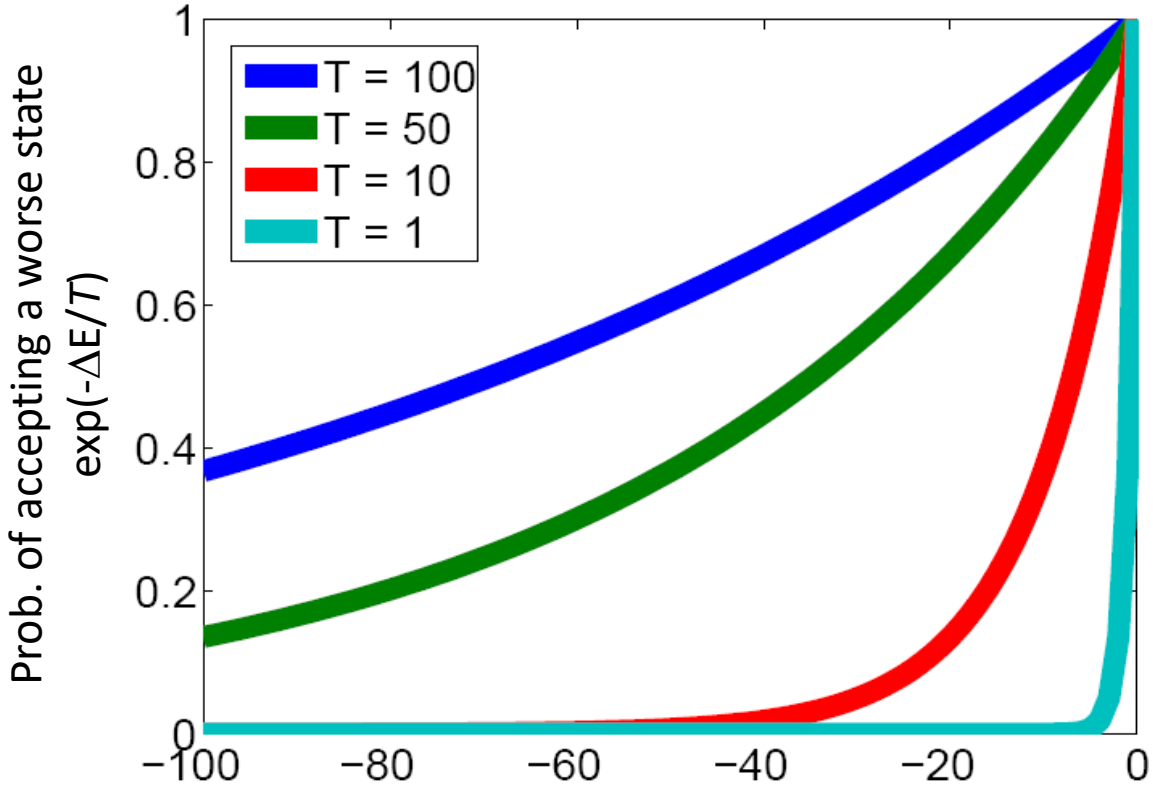
Always accept good moves that reduce the energy.

Accept "bad" moves with a probability inspired by the acceptance criterion in the Metropolis–Hastings MCMC algorithm.

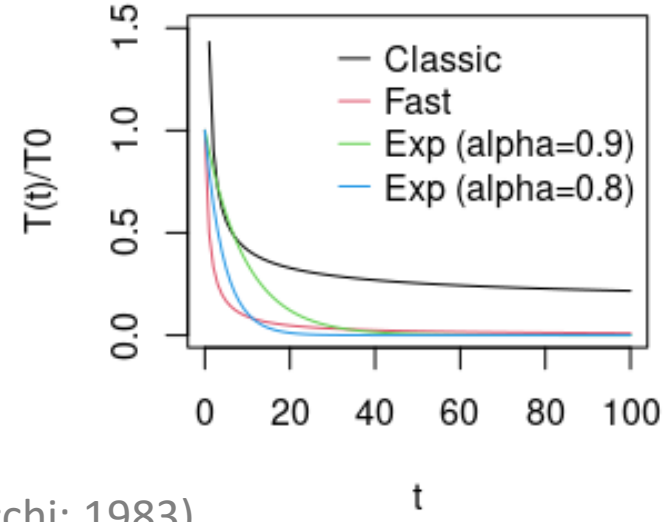
Note: Use $VALUE(next) - VALUE(current)$ for minimization

The Effect of Temperature

Convert the changes due to “bad” moves into an acceptance probability depending on the temperature. The criterion uses the negative part of the exponential function.



Cooling Schedule



The cooling schedule is very important.
Popular schedules for the temperature at time t:

- **Classic simulated annealing:** $T_t = T_0 \frac{1}{\log(1+t)}$
- **Exponential cooling** (Kirkpatrick, Gelatt and Vecchi; 1983)

$$T_t = T_0 \alpha^t \quad \text{for} \quad 0.8 < \alpha < 1$$

- **Fast simulated annealing** (Szy and Hartley; 1987)

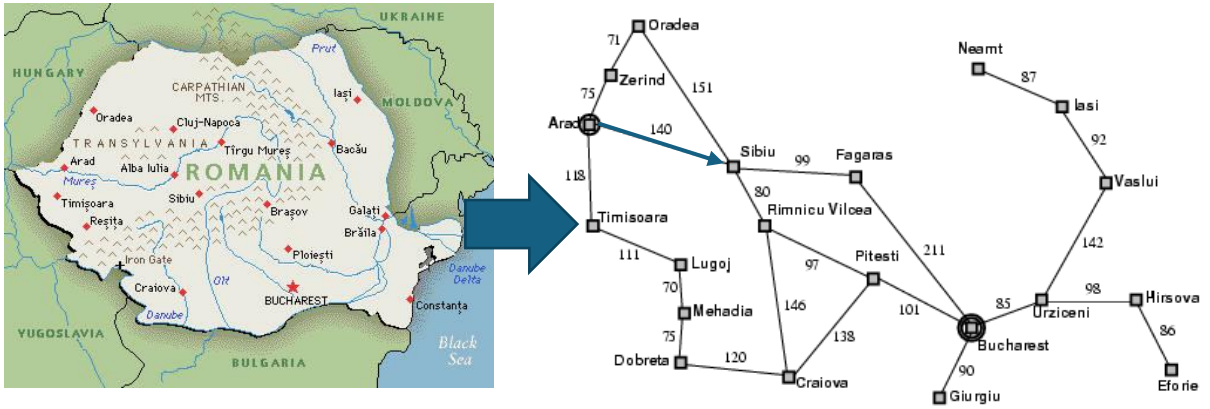
$$T_t = T_0 \frac{1}{1+t}$$

Notes:

- Choose T_0 to provide a high probability $p_0 = e^{-\frac{\Delta E}{T_0}}$ that any move will be accepted at time $t = 0$. ΔE is determined by the worst possible move.
- T_t will not become 0 but very small. Stop when $T < \epsilon$ (ϵ is a very small constant).
- The best schedule (cooling rate) is typically determined by trial-and-error. The goal is to have a low chance of getting stuck in a local optima.

Methods: Discretization of Continuous Space

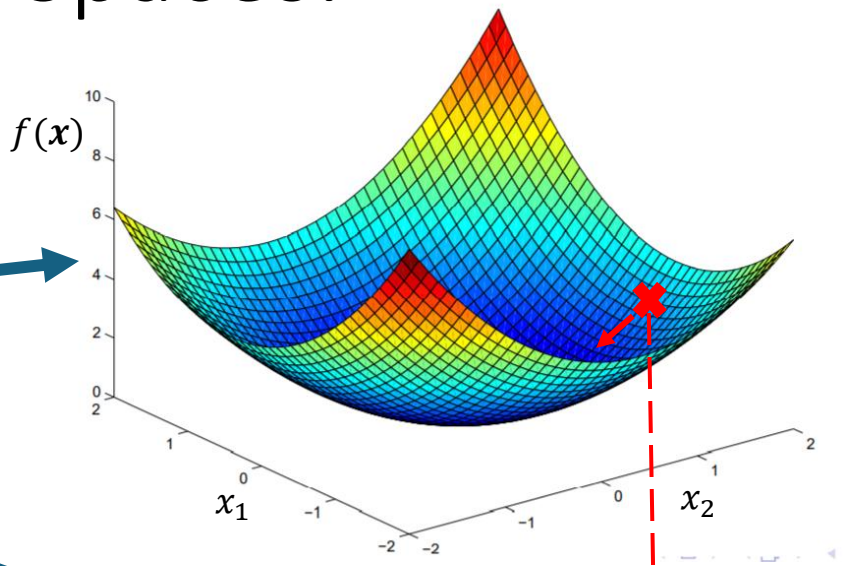
- Use atomic states and create a graph as the transition function.



- Use a grid with spacing of size δ

  Note: You probably need a way finer grid!

Search in Continuous Spaces: Gradient Descent



State representation: $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$

State space size: infinite

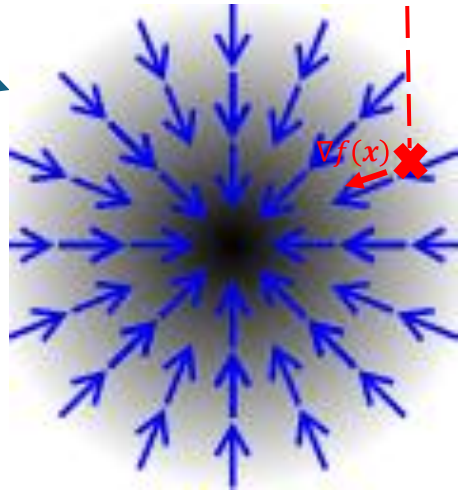
Objective function: min $f(\boldsymbol{x})$

Local neighborhood: small changes in $x_1, x_2, \ldots, x_k$

Gradient at point $\boldsymbol{x}$: $\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_k}\right)$

(=evaluation of the Jacobian matrix at $\boldsymbol{x}$)

Find optimum by solving: $\nabla f(\boldsymbol{x}) = 0$



- **Gradient descent (= Steepest-ascend hill climbing for minimization)** with step size α (typically reduced over time)

  Repeat: $\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f(\boldsymbol{x})$

- **Newton-Raphson method**
  uses the inverse of the Hessian matrix (second-order partial derivative of $f(\boldsymbol{x})$)
  $H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ as the optimal step size

  Repeat: $\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{H}_f^{-1}(\boldsymbol{x}) \nabla f(\boldsymbol{x})$

Note: May get stuck in a local optimum if the search space is non-convex! Use simulated annealing, momentum or other methods to escape local optima.