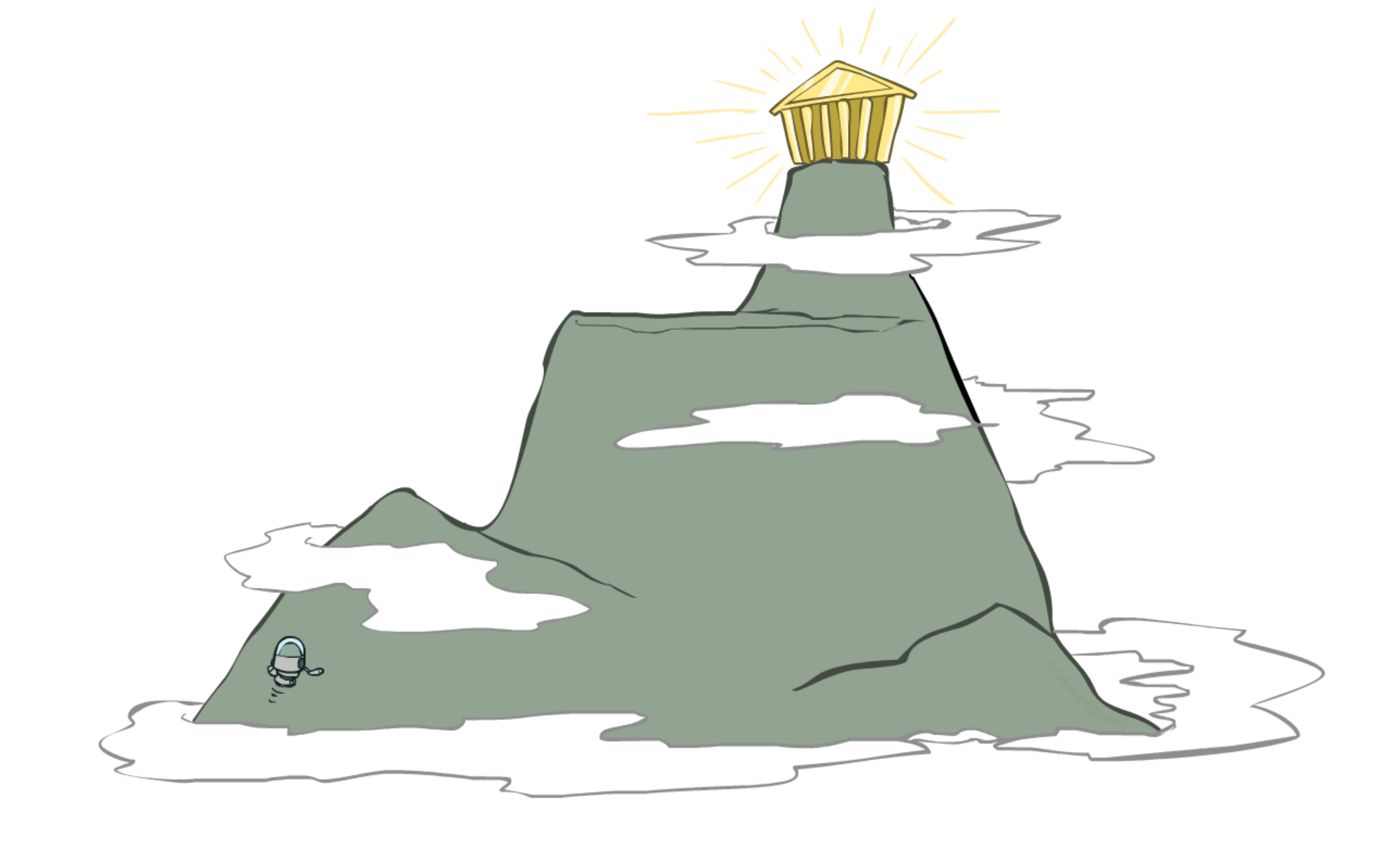
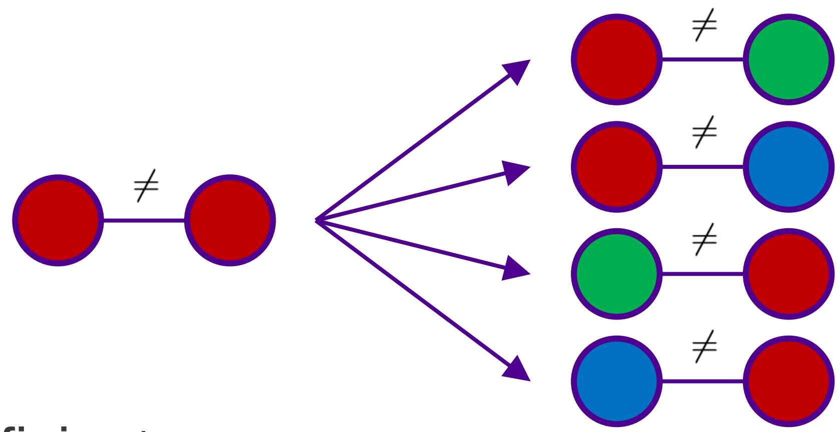
Beyond Classical Search

CHAPTER 4 IN THE TEXTBOOK



Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can't make it better (no fringe!)
- New successor function: local changes



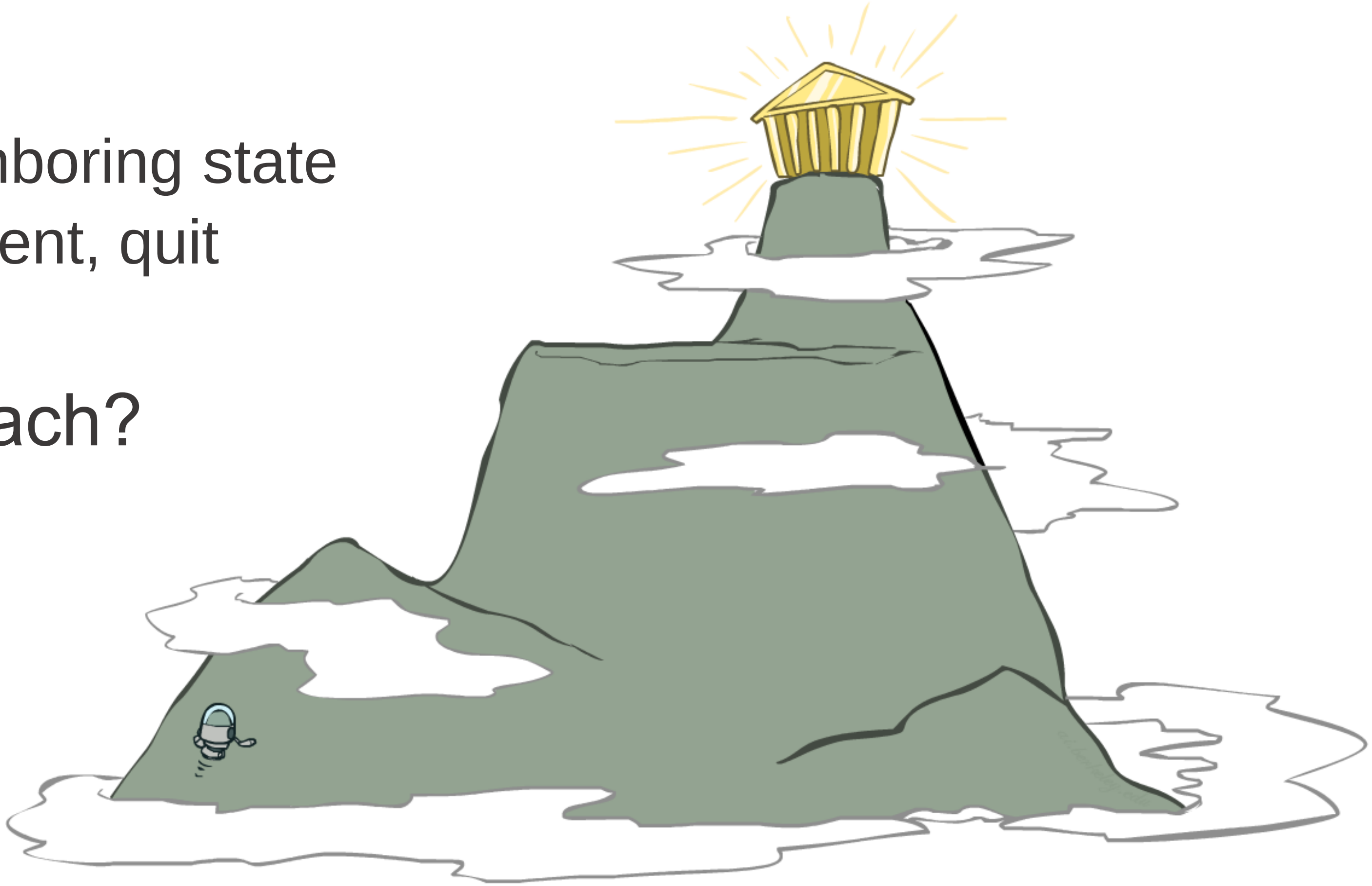
- Generally much faster and more memory efficient (but incomplete and suboptimal)

Hill Climbing

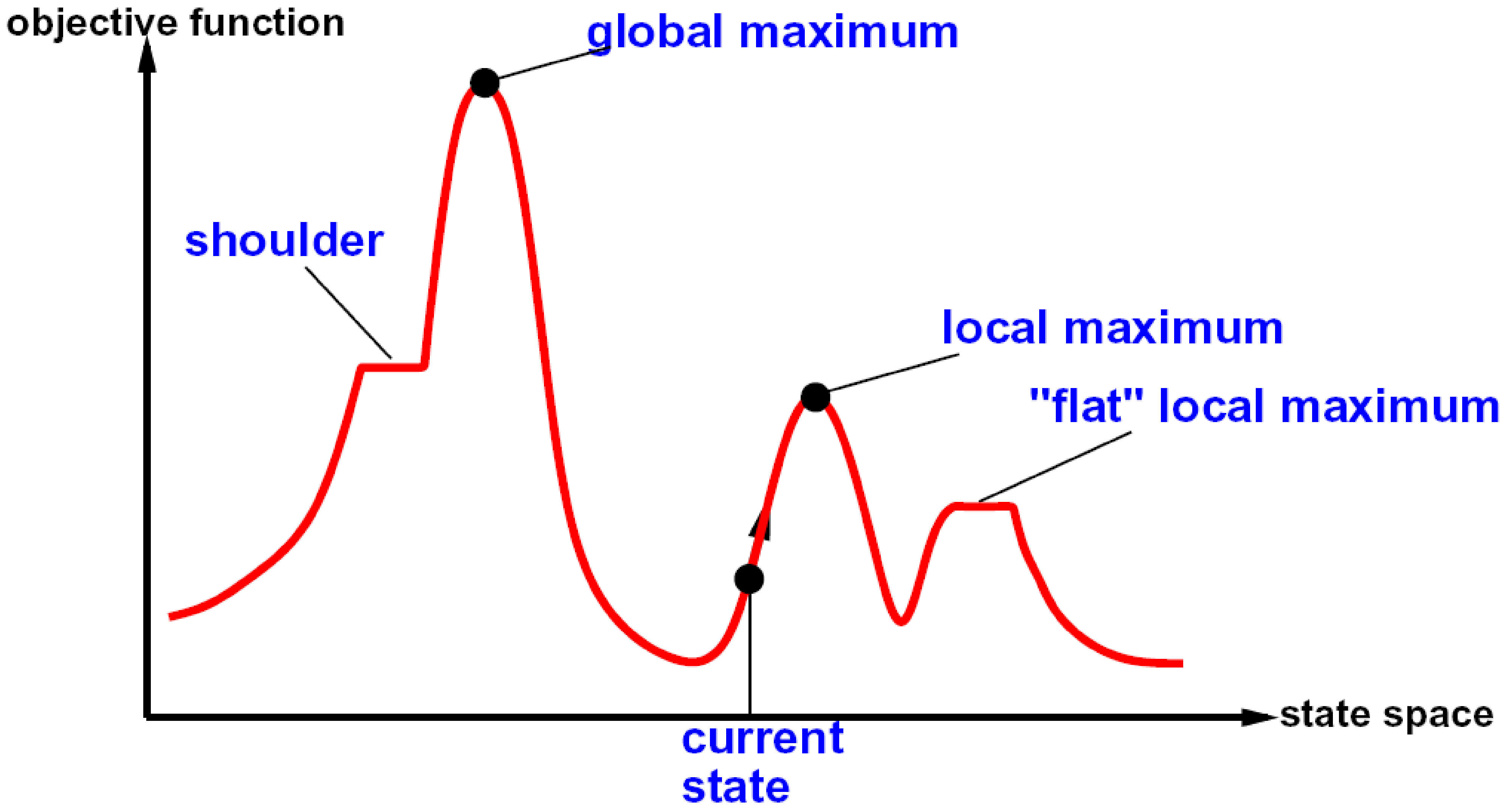
- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- What's bad about this approach?
  - Complete?
  - Optimal?

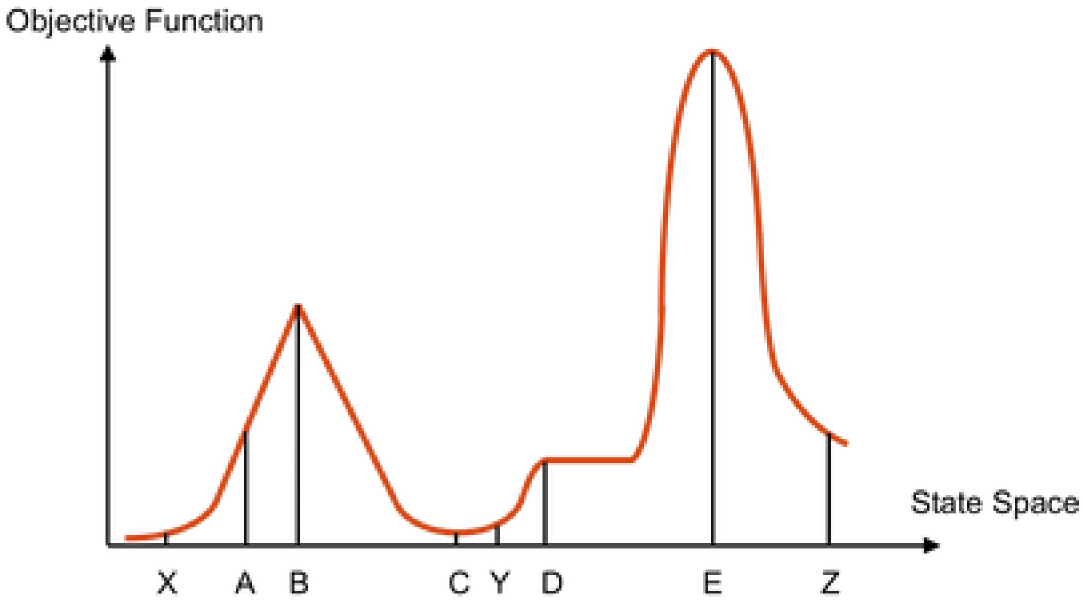
- What's good about it?



Hill Climbing Diagram



Hill Climbing Quiz



Starting from X, where do you end up ?

Starting from Y, where do you end up ?

Starting from Z, where do you end up ?

Tampere University

Hill climbing search

- Here one always chooses a successor $s' \in S(s)$ of the current state s that has the highest value for the objective function f

$$\max_{s' \in S(s)} f(s')$$

- Search terminates when all neighbors of the state have a lower value for the objective function than the current state has
- Most often search terminates in a local maximum, sometimes by chance, in a global maximum
- Also plateaux cause problems to this greedy local search
- On the other hand, improvement starting from the initial state is often very fast

- Sideways moves can be allowed when search may proceed to states that are as good as the current one
- **Stochastic** hill-climbing chooses at random one of the neighbors that improve the situation
- Neighbors can, for example, be examined in random order and choose the first one that is better than the current state
- Also these versions of hill-climbing are incomplete because they can still get stuck in a local maximum
- By using **random restarts** one can guarantee the completeness of the method
  - Start from a random initial state until a solution is found

Local beam search

- The search begins with k randomly generated states
- At each step, all the successors of all k states are generated
- If any one of the successors is a goal, the algorithm halts
- Otherwise, it selects the k best successors from the complete list and repeats
- The parallel search leads quickly to abandoning unfruitful searches and moves its resources to where the most progress is being made
- In stochastic beam search the maintained successor states are chosen with a probability based on their goodness

Beam Search

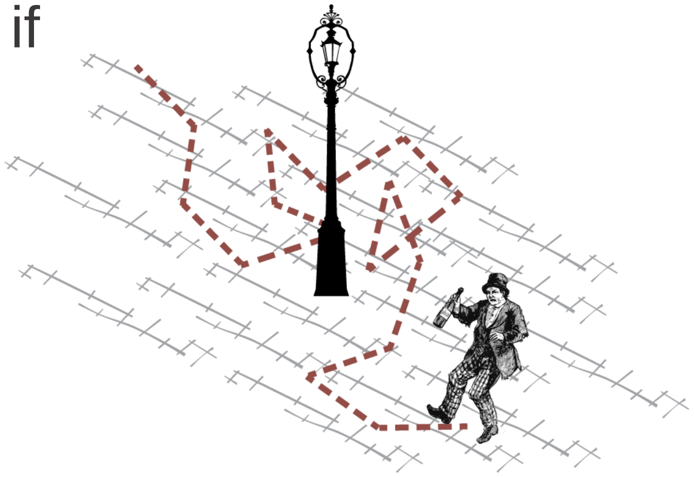
- Like greedy hill climbing search, but keep K states at all times:

Greedy Search   Beam Search

- Variables: beam size, encourage diversity?
- The best choice in MANY practical settings
- Complete? Optimal?
- Why do we still need optimal methods?

Random walk

- A ***random walk*** moves to a successor chosen uniformly at random independent of whether it is better than the current state
  - a complete search algorithm, but when unsupervised also extremely inefficient
- Let us allow “bad" moves with some probability p
- The probability of transitions leading to worse situation decreases exponentially with time (“temperature")
- We choose a candidate for transition randomly and accept it if
  - the objective value improves;
  - otherwise with probability p
- If temperature is lowered slowly enough, this method converges to a global optimum with probability $\rightarrow 1$

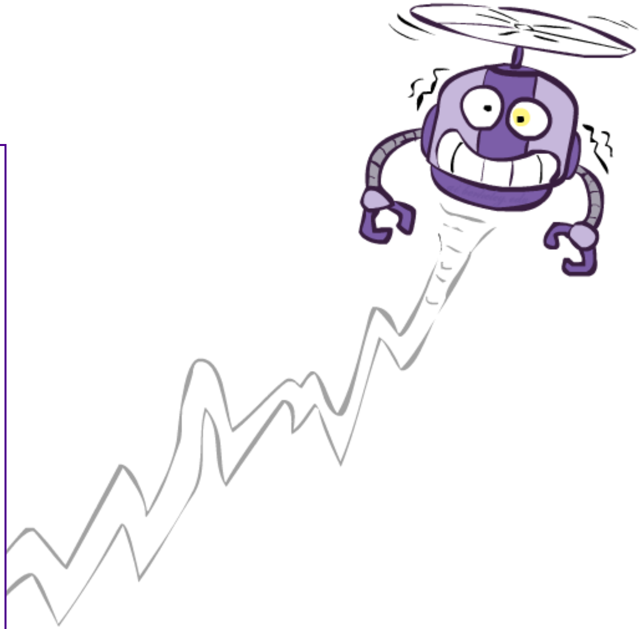


Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

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



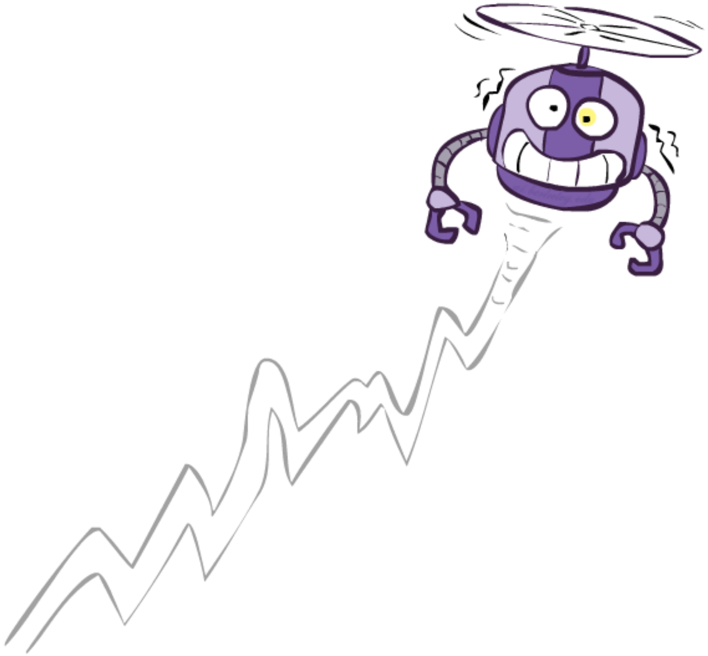
Tampere University

Simulated Annealing

- Theoretical guarantee:
  - Stationary distribution:

$$p(x) \propto e^{\frac{E(x)}{kT}}$$

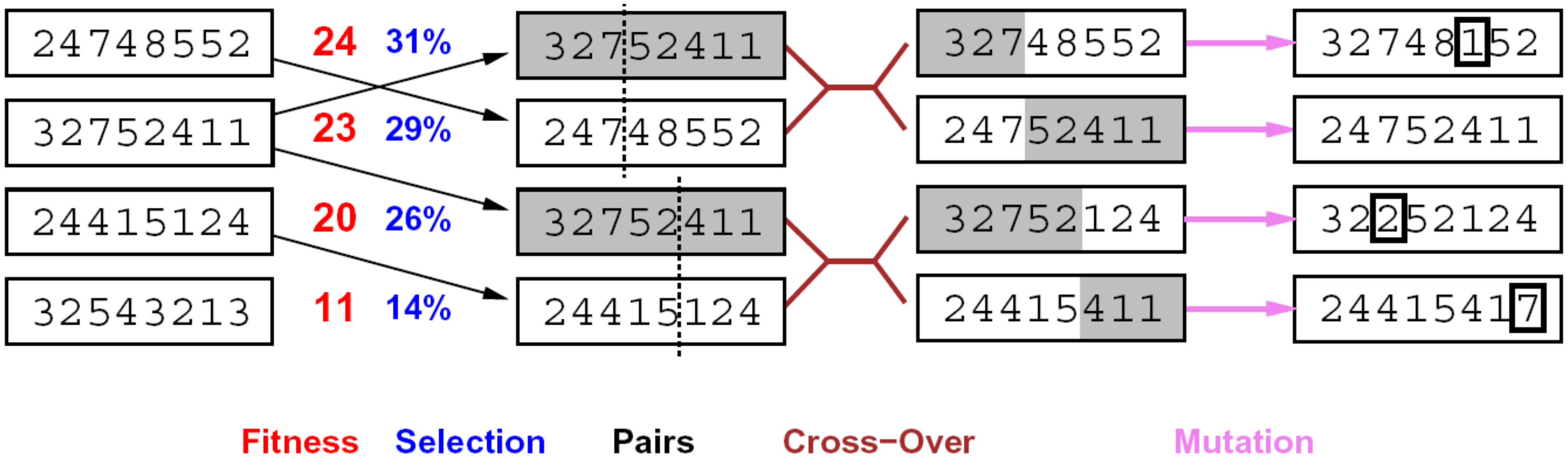
  - <u>If T decreased slowly enough</u>, will converge to optimal state!

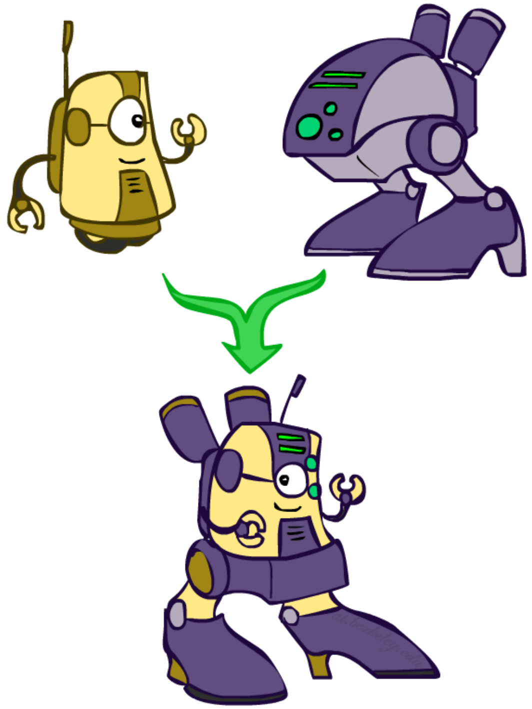


- Is this an interesting guarantee?

- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways

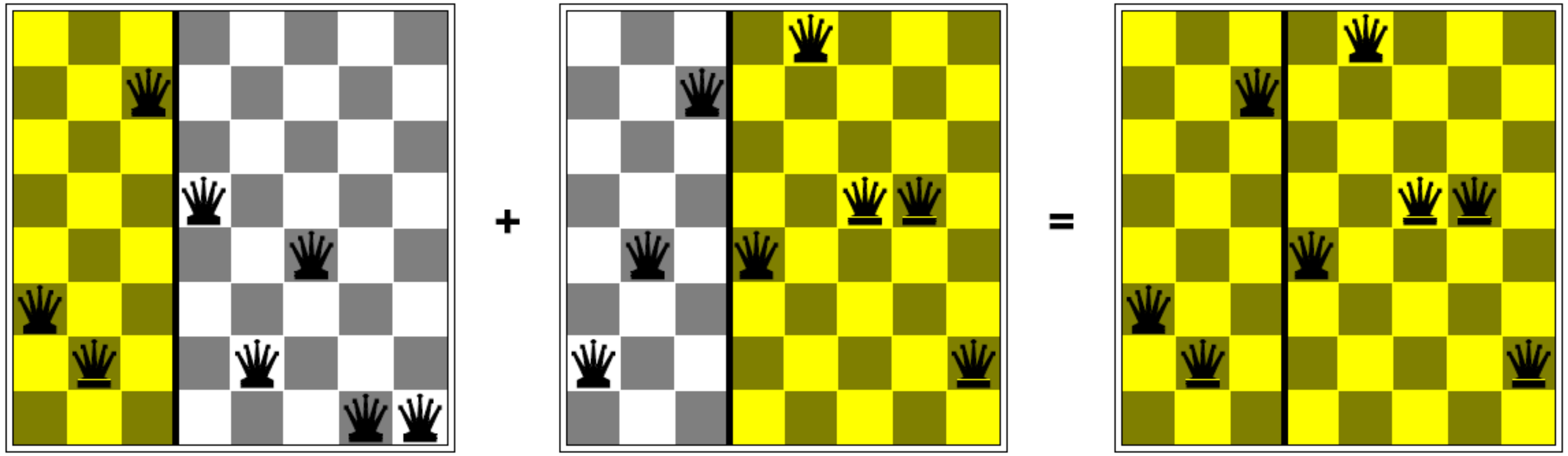
Genetic Algorithms





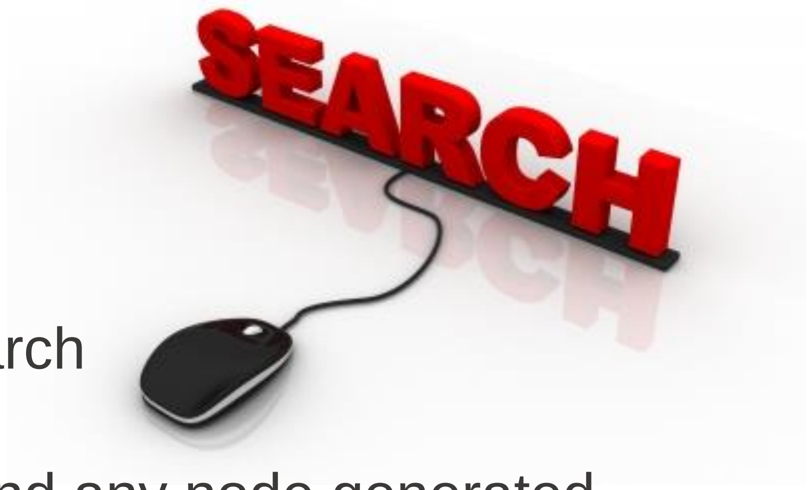
- Genetic algorithms use a natural selection metaphor
  - Keep best N hypotheses at each step (selection) based on a fitness function
  - Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around

Example: N-Queens



- Why does crossover make sense here?
- When wouldn’t it make sense?
- What would mutation be?
- What would a good fitness function be?

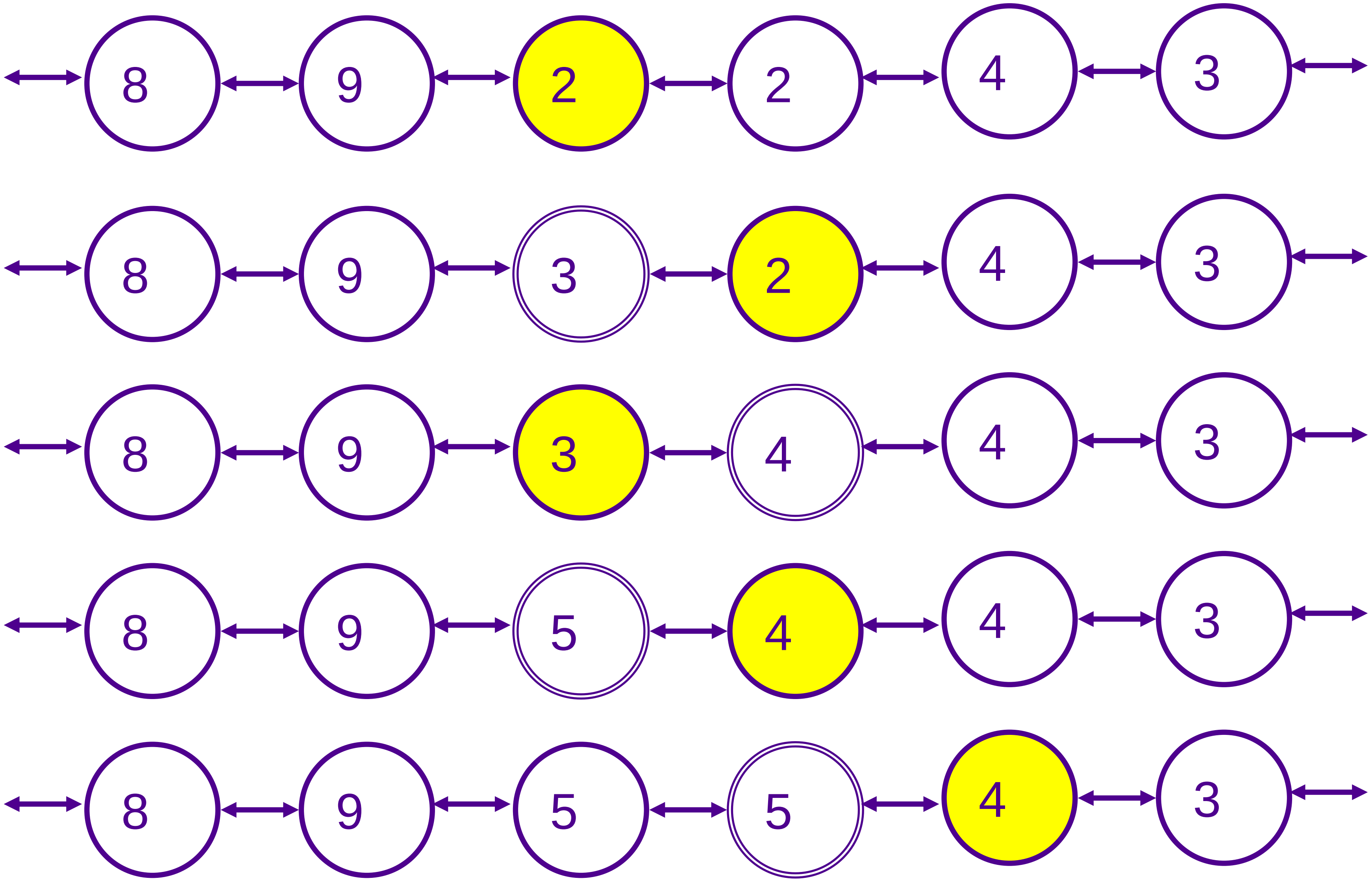
Online Search



- Not all offline search algorithms are suitable for online search
- E.g., A* is essentially based on the fact that one can expand any node generated to the search tree
- An online algorithm can expand only a node that it physically occupies
- DFS only uses local information, except when backtracking
- Hence, it is usable in online search (if actions can physically be undone)
- DFS is not competitive: one cannot bound the competitive ratio

- Hill-climbing search already is an online algorithm, but it gets stuck at local maxima
- Random restarts cannot be used:
  - the agent cannot transport itself to a new state
- Random walks are too inefficient
- Using extra space may make hill-climbing useful in online search

- We store for each state s visited our current best estimate $h(s)$ of the cost to reach the goal
- Rather than staying where it is, the agent follows what seems to be the best path to the goal based on the current $h(s)$ cost estimates for its neighbors
- At the same time the value of a local minimum gets flattened out and can be escaped


8
9
2
2
4
3
8
9
3
2
4
3
8
9
3
4
4
3
8
9
5
4
4
3
8
9
5
5
4
3