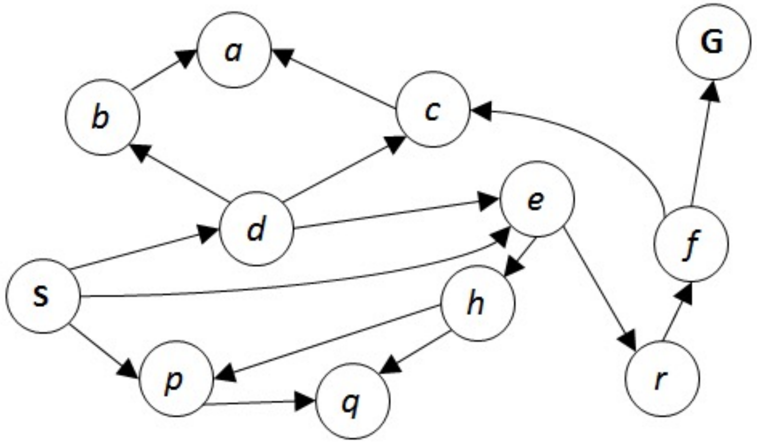
COMP 341 Intro to AI
Local Search

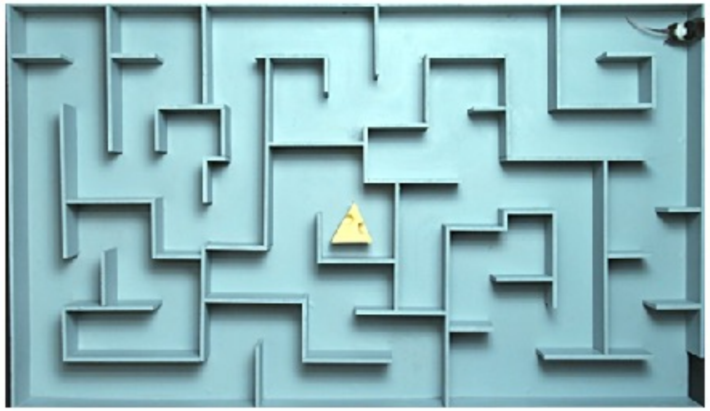
Asst. Prof. Barış Akgün

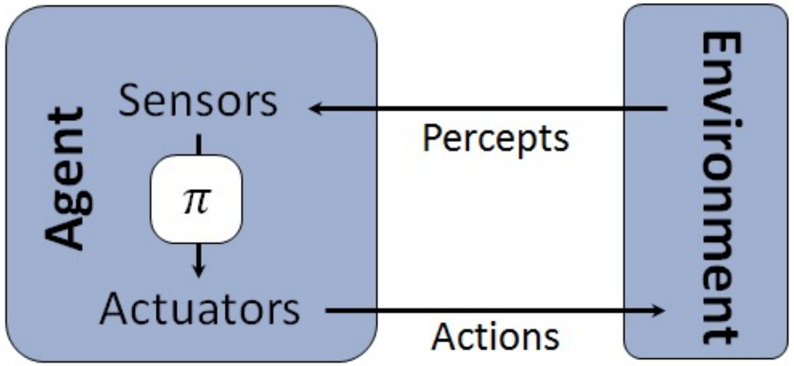
Koç University

Today

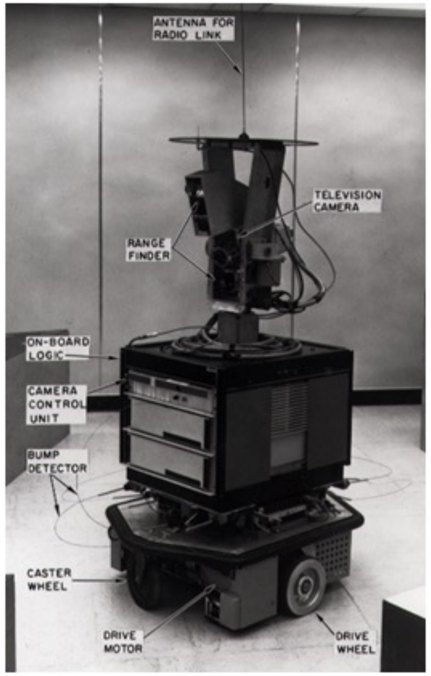
- Recap
- Local Search
  - “Hill-Climbing”
  - Simulated Annealing
  - Local Beam Search
  - Genetic Algorithms
  - Gradient Ascent/Descent
- Online Search
- Beyond Classical Search: Some Topics







Previously on Intro to AI



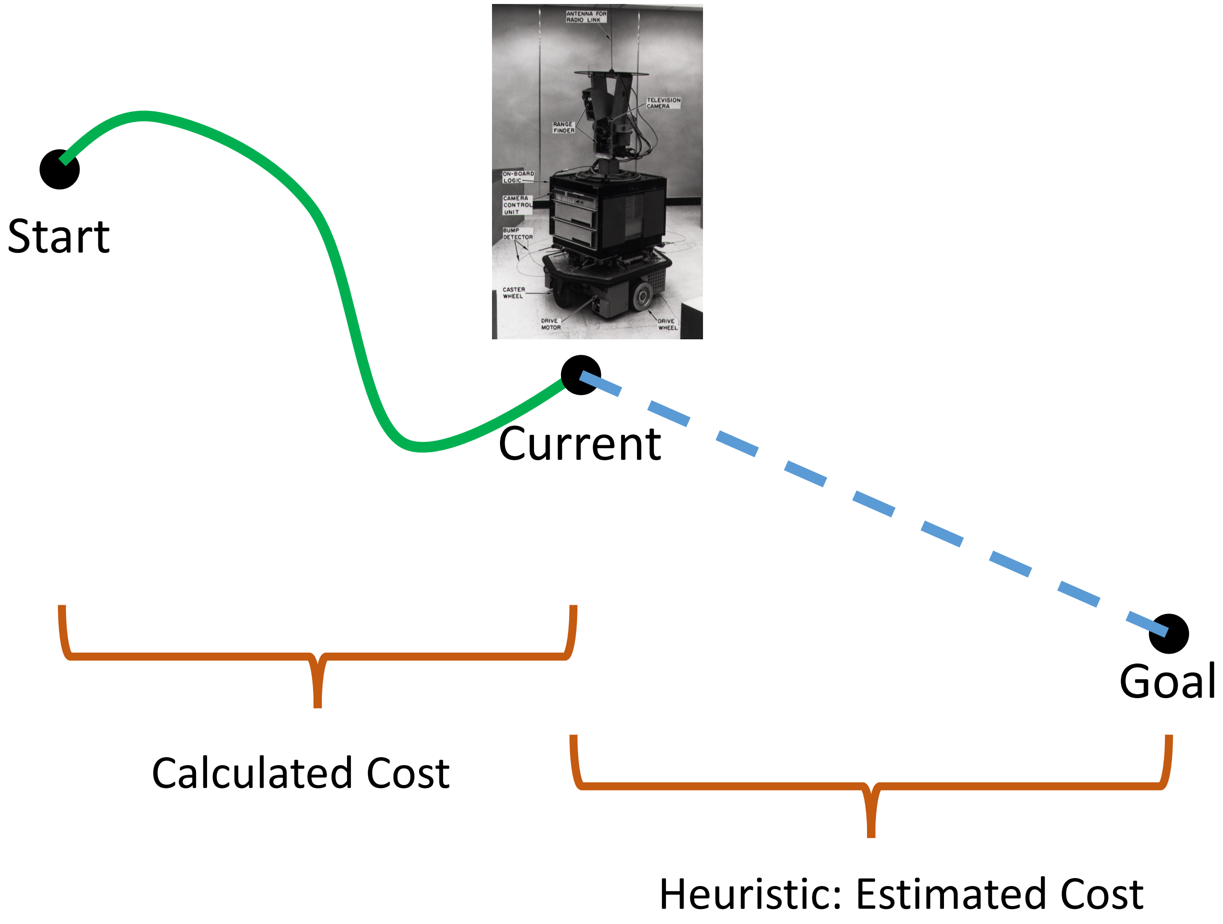
Search

- Uninformed
  - DFS, BFS, UCS
  - No domain knowledge
- Informed
  - Greedy, A*
  - Heuristics based on domain/problem knowledge
- Solution is a path to goal

Heuristics

- How promising is a given state?
  - Admissibility/Consistency
  - Dominance

- Design:
  - Solution Costs to Relaxed Problems (e.g., 8 puzzle)
  - Geometric Limits (e.g., Euclidean/Manhattan)
  - Creativity ☺



Local Search

Local Search

- Classical Search:
  - Solution is path to a goal state

- Local Search:
  - Solution is the goal state itself
  - Search for a solution when the path doesn’t matter

- Optimization: Get to a “better” state (best state if possible!)

Local Search Applications

- Integrated circuit design
- Factory floor layout
- Scheduling
- Routing
- Portfolio Management
- Network optimization
- …

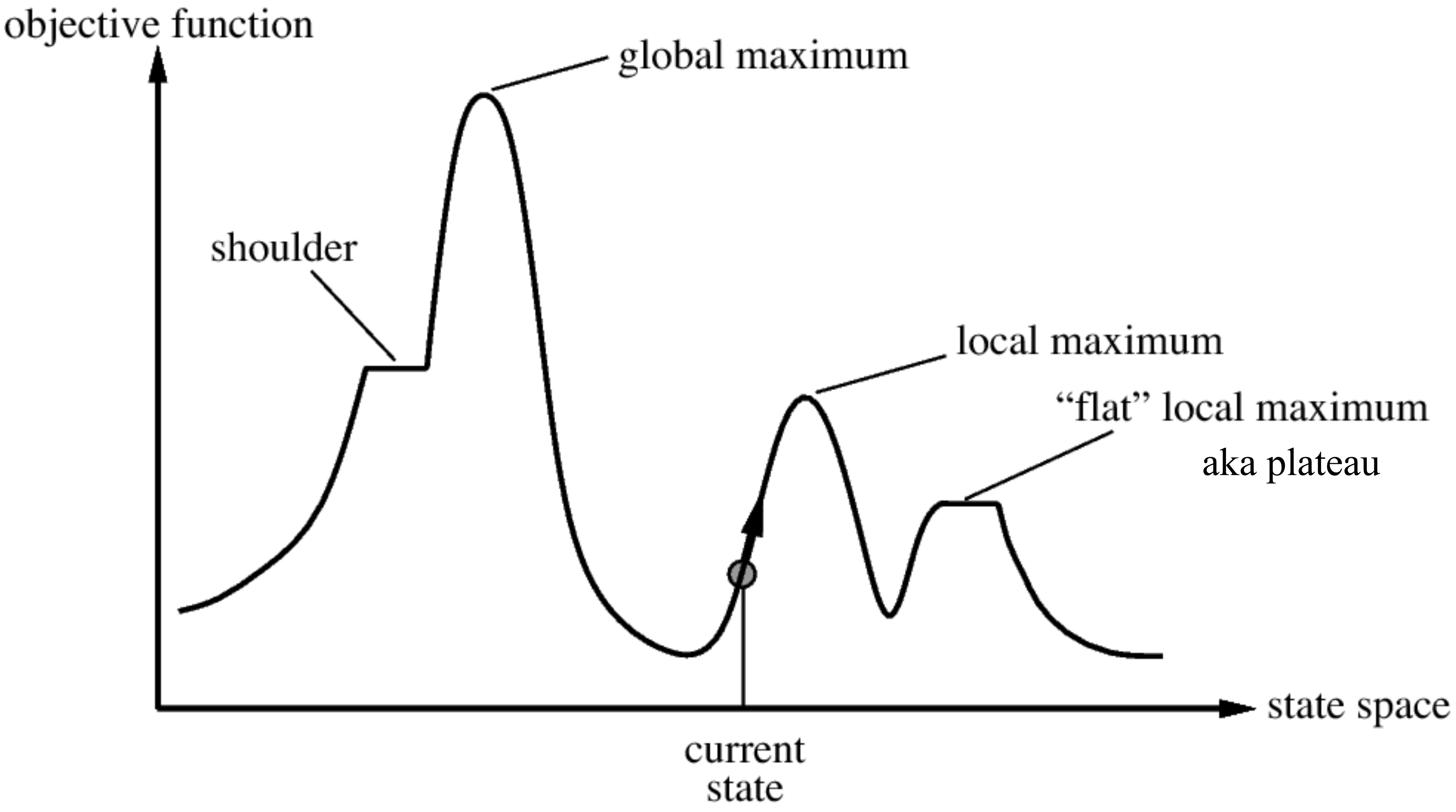
Local Search

- Formulation:
  - Current State
  - Transition Function
  - Evaluation Function

- Algorithms: Move towards Better States
  - Complete: Find a solution if one exists
  - Optimal: Find the best state
- New Concept: State Space “Landscape”

- Usually easy to code!

1D State Space Example

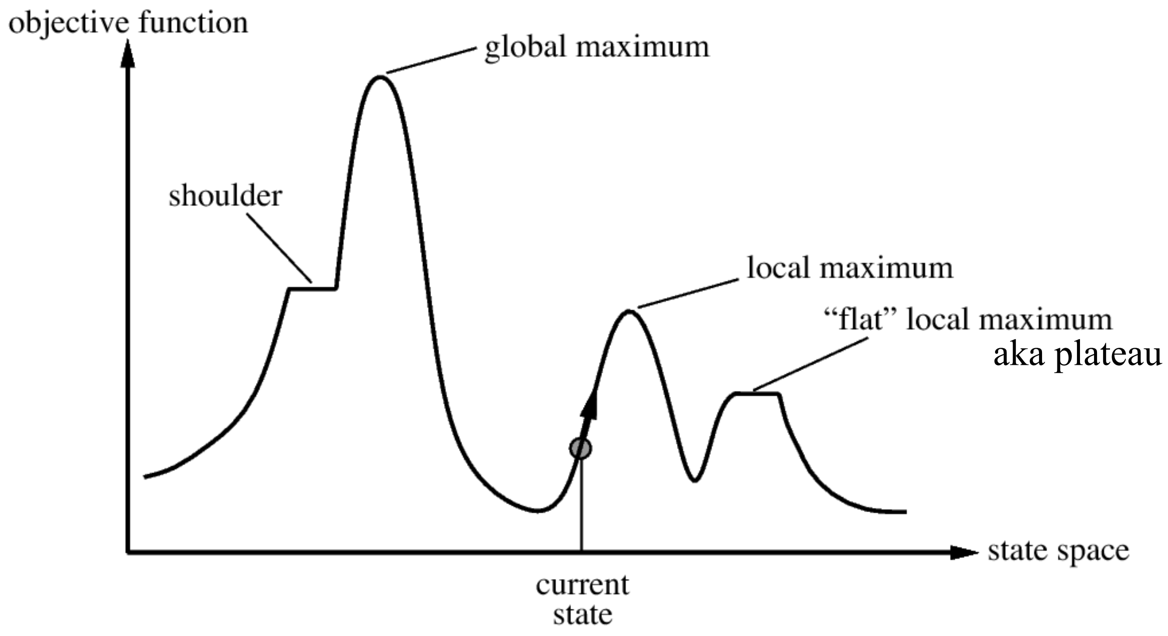


Hill-Climbing (in Heavy Fog with Amnesia)

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

Hill Climbing Properties

- Complete
  - No
- Optimal
  - No
- Time Complexity
  - O(d) – d: longest path to solution (could be infinite!)
- Space Complexity
  - Constant



Drawbacks:

- Plateaus
- Local Minima
- No memory

Hill Climbing – Avoiding Drawbacks?

- Get out of Plateaus:
  - Random walk among equally valued states – can escape “shoulders”
  - Infinite Loop, e.g. local flat maxima: Move thresholding

- Avoid Local Minima:
  - Random restart: Start the problem at different states after termination
  - Probabilistically complete but not efficient

- Add Memory:
  - Hill climbing from multiple states in parallel
  - Exchange information

Some Randomized Variants

- Stochastic HC: choose randomly from uphill moves
- First choice Stochastic HC: generate successors randomly, pick 1st uphill
- Random Restart HC: try and try HC until a goal is found
- What is left?
  - Maybe sometimes move downhill?

Reality Check for HC

- HC is too greedy
  - Never moving downhill is incomplete
  - Real problems have lots of local maxima

- Randomness can bring completeness, but it is inefficient

- So?
  - Combine them!

Simulated Annealing

- Idea:
  - Sometimes move downhill to escape local maxima
  - i.e. Combine random walk with hill climbing
  - Gradually decrease the size and frequency of the random walk

- T:
  - Probability of picking a random neighbor instead of best
  - “Temperature parameter”

- Slowly decrease the temperature – hence annealing!

Simulated Annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
#*schedule*, a mapping from time to "temperature"
current ←MAKE-NODE(*problem*.INITIAL-STATE)
while true **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 $e^{\Delta E/T}$

reduce T over time

Select a random neighbor and calculate the change in value

If the change is positive, move to the next state
else move with some probability

If T is decreased "slow enough", then SA is probabilistically complete and globally optimal!

SA Temperature Scheduling Examples

$$T(t) = \frac{d}{\log(t)}, d > 0, t > 1$$

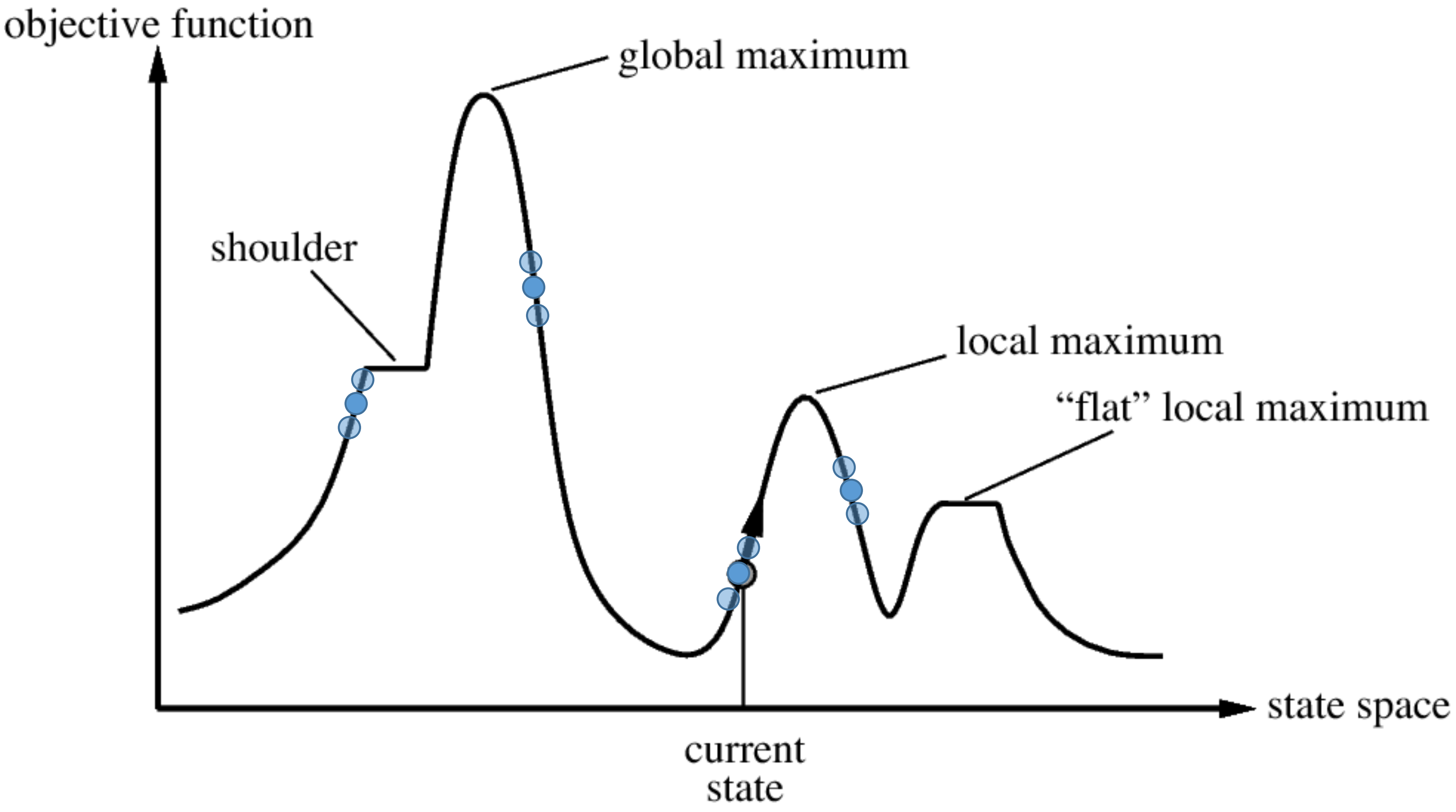
$$T(t) = \alpha T(t-1), 1 > \alpha > 0$$

$$T(t) = T(t-1) - k, k \; > 0$$

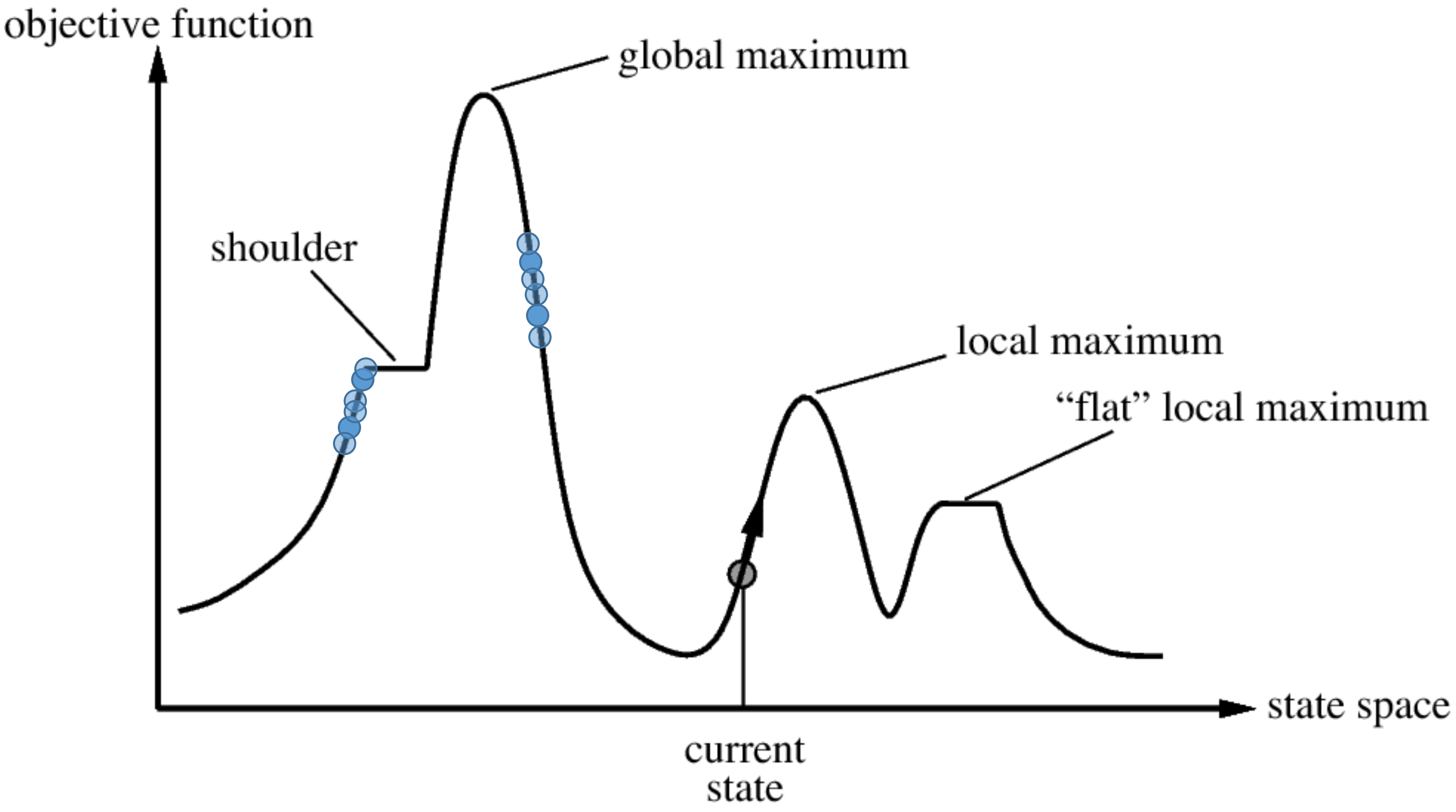
Local Beam Search - Memory

- **Idea**: Keep k states instead of 1 and chose top k of all their successors
  - Not the same as k searches run in parallel! Searches that lead to good states recruit other searches to join them

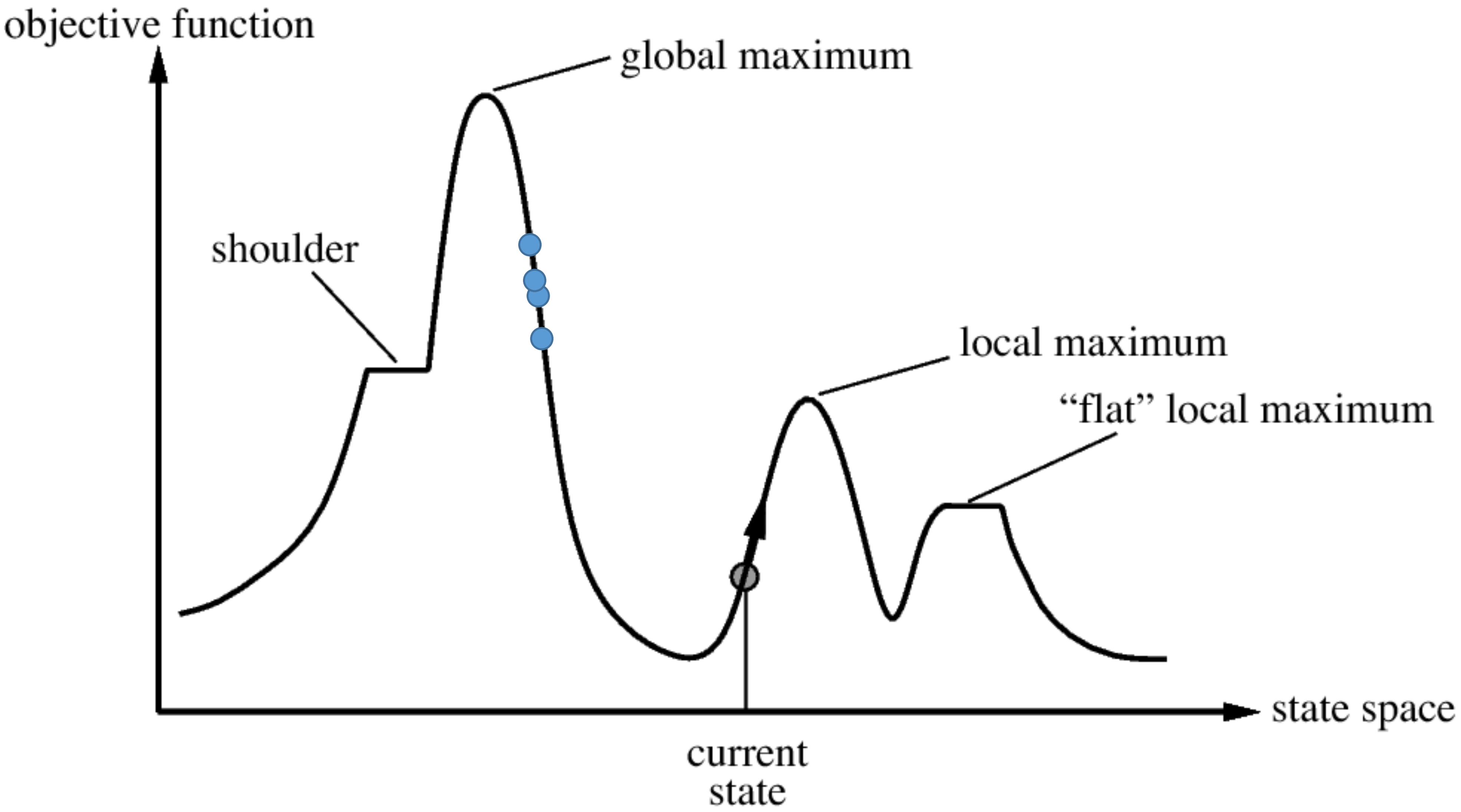
Local Beam Search - Memory



Local Beam Search - Memory



Local Beam Search - Memory



Local Beam Search - Memory

- **Idea**: Keep k states instead of 1 and chose top k of all of their successors
  - Not the same as k searches run in parallel! Searches that and good states recruit other searches to join them
- **Problem**: quite often, all k states end up on same local hill
- **Idea 2**: choose k successors randomly, <u>biased</u> towards good ones
  - Stochastic Beam Search
  - Natural selection anyone?

Genetic Algorithms

- Stochastic local beam search + generate successors from **pairs** of states

- Formulation
  - state: *finite alphabet*
  - k states: *population*
  - Evaluation function called *fitness function*
  - Successors from two parents

Genetic Algorithms

24748552

32752411

24415124

32543213

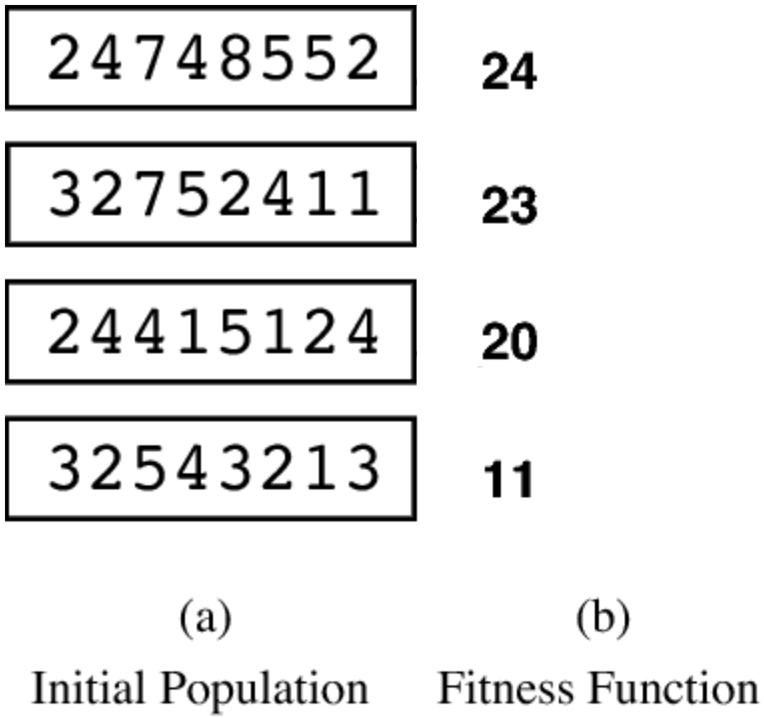
(a)

Initial Population

states encoded from a finite alphabet

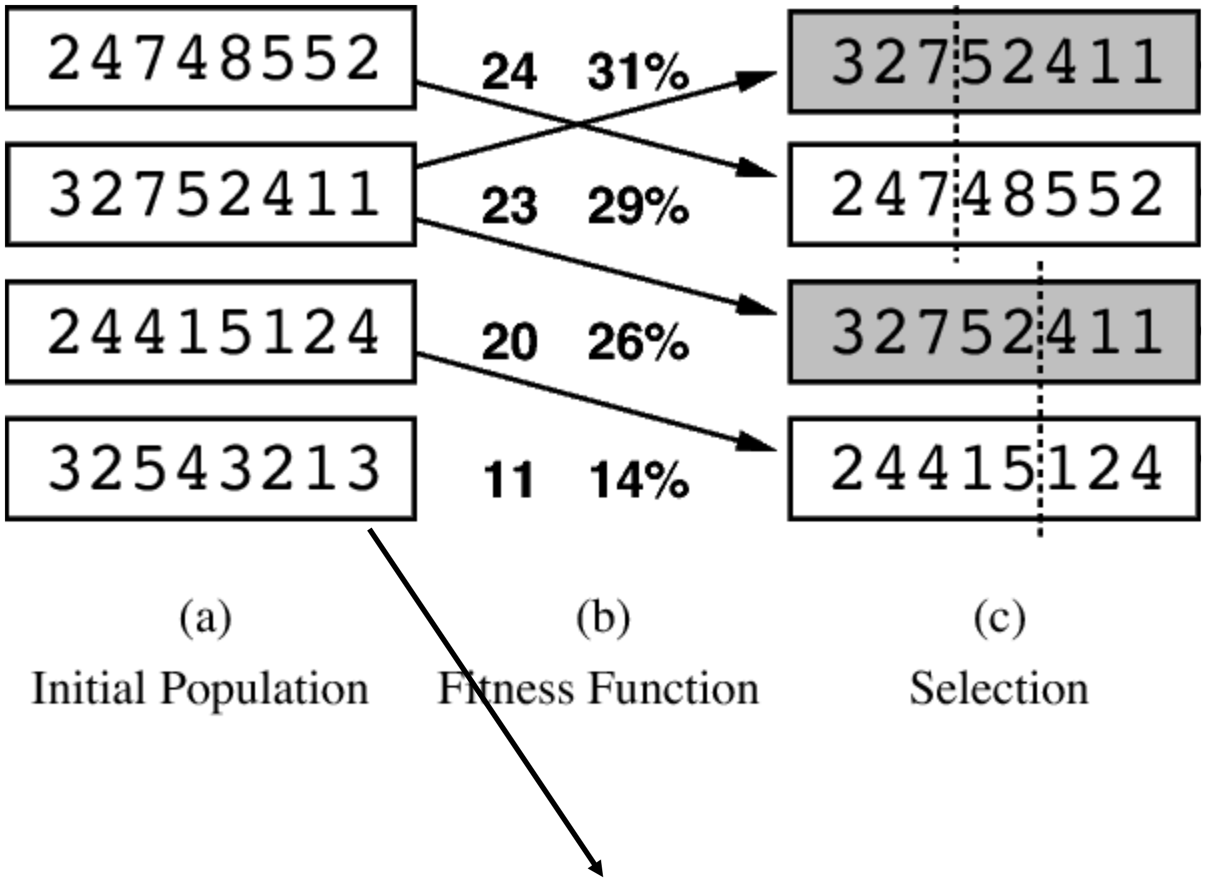
k initial states, usually randomly selected

Genetic Algorithms



Each state has a "fitness" which is given by the fitness/evaluation function

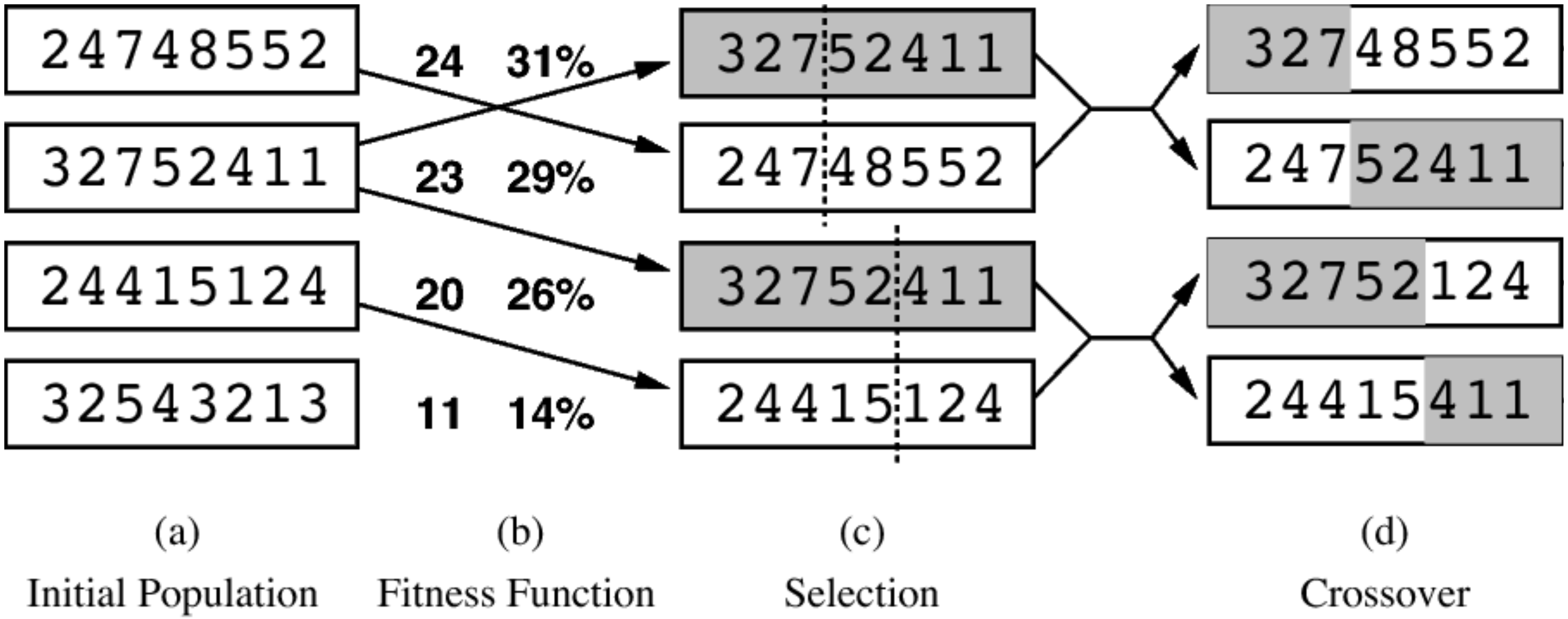
Genetic Algorithms



Select pairs of states for **reproduction**, weighted by their fitness

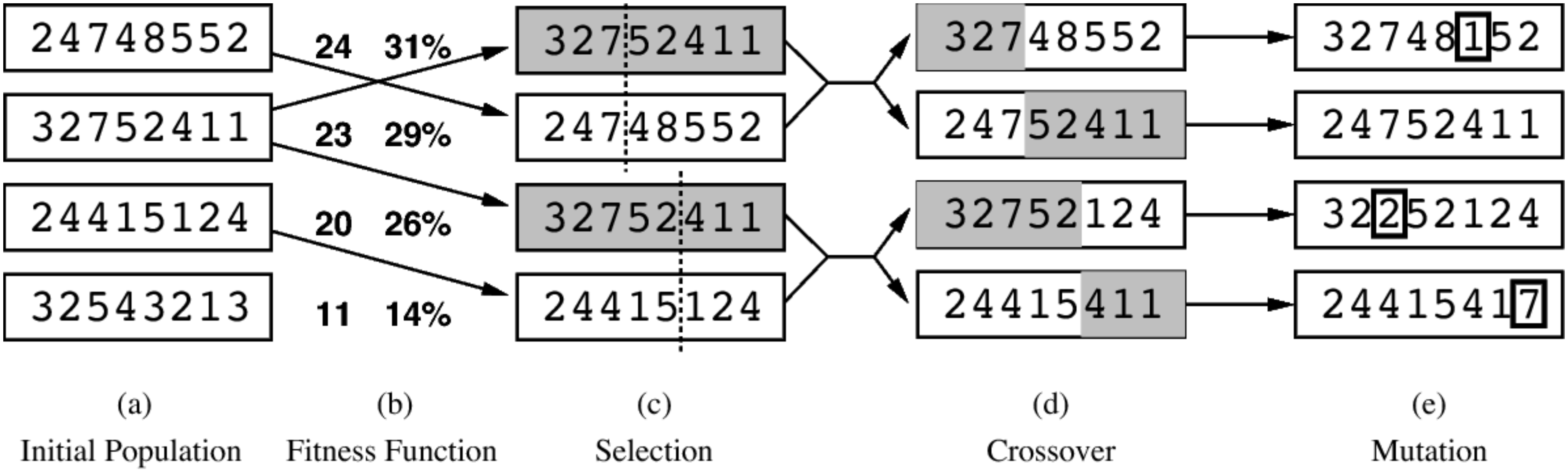
This state was not lucky enough

Genetic Algorithms



Reproduction: randomly choose a cross-over point, create two new states by mixing the pair

Genetic Algorithms



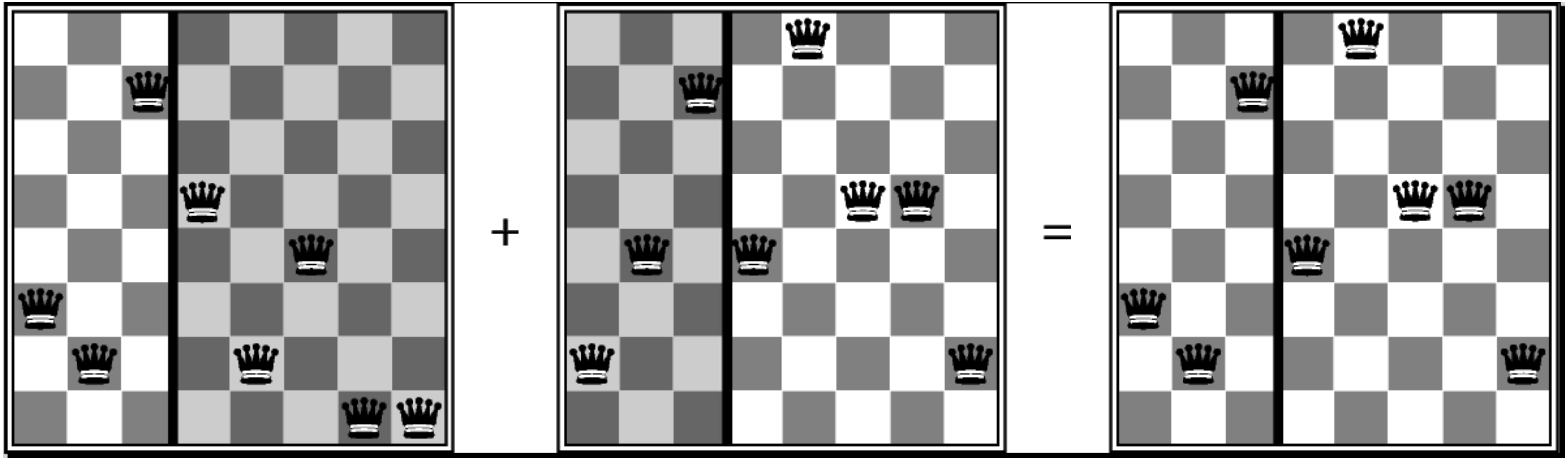
Mutation: these new states are also subject to random mutations of single digits in the state

Genetic Algorithms – 8 Queens

- 8 Queens Puzzle: placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other
- Formulate:
  - State?
  - Fitness Function?
  - Do we need a transition function?

Genetic Algorithms – 8 Queens

- 8 Queens Puzzle: placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other



Continuous State - Gradient Descent/Ascent

- State x: multivariate and continuous
- Objective function f(x): Differentiable around x

- **Idea**: Move x in the direction of decreasing/increasing f
- **How**: Derivatives!

- Need to calculate the **gradient:**

Getting into the realm of optimization!

$$\nabla f(x) = \left(\frac{\partial f}{x_1}, \frac{\partial f}{x_2}, \ldots, \frac{\partial f}{x_n}\right)$$

Can be computed *analytically* or *numerically*

Gradient Descent/Ascent

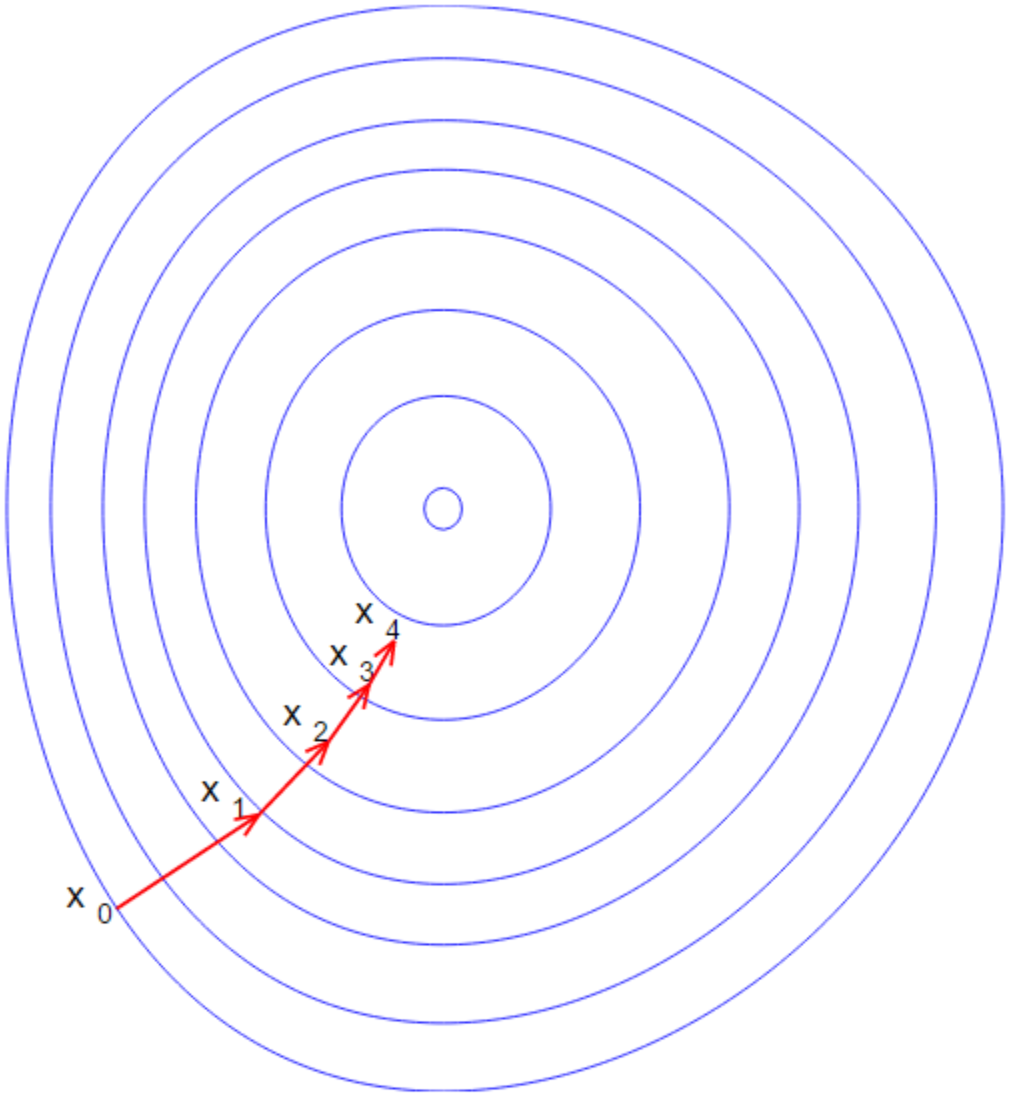
- Implementation:

While SomeCondition:

$$x(t+1) = x(t) \pm \alpha \nabla f(x)$$

- SomeCondition:
  - Maximum iterations
  - |x(t+1) – x(t)| < small
  - …

- More complicated versions exists such as calculating higher order derivatives (e.g. *Newton-Raphson*), adaptive step size (e.g. momentum) etc.



A few Words on Gradient Descent/Ascent

- This family of algorithms is used in **a lot of** applications!
- Very simple to code the base version
- Works in any number of dimensions
    - Even in infinite-dimensions! Just need to change the derivative

- Inefficient
- Some state-space landscapes wreak havoc

- In any case, a good first tool to tackle an applicable problem!

Online Search

- **Offline**: simulate the world and reason about a plan to get to a goal
- **Online**: solve the search problem while executing actions
- Interleaves search and execution

Remaining parts of this slide deck will not be in the exams. Similar topics introduced later will be.

Why Online Search?

- **Dynamic environment**, things change too quickly to plan
- **Nondeterministic environment**, deal with what happens rather than planning for all contingencies
- **Hard to model**, cannot get a reasonable model to do search on

- **Problem**: a shortsighted view may lead the agent astray

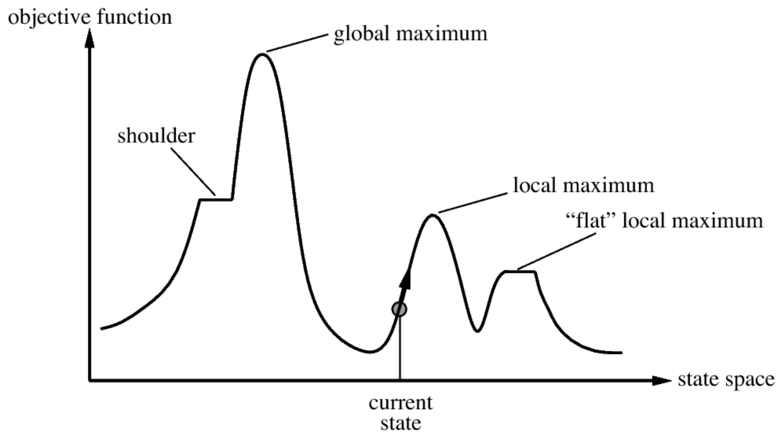
Online Search

- *Actions*(*s*) = actions allowed in state *s*
- *c*(*s*,*a*,*s'*) = cost of action going from state *s* to state *s'* with action *a*
- *Goal-test*(*s*)
- No transition model! Do things and see what happens
  - You know *s'* when you get there!

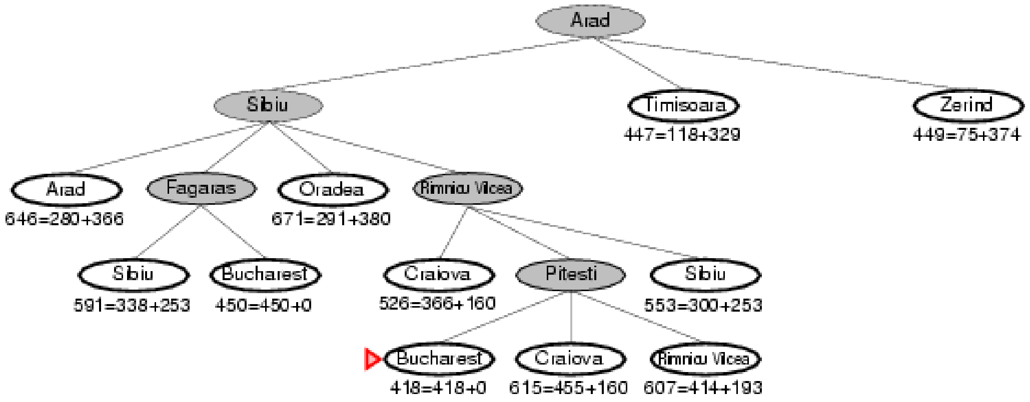
- Issues:
  - Irreversible actions
  - Dead-ends
  - Is environment **safe to explore**?

Online Searching with Algos Learned so Far

- Hill-Climbing?
  - Yes!
  - In fact, most local search methods
  - Obviously, no random restarts



- A*?
  - No
  - Cannot jump around the state space!



But I really like A*!

- Learning Real-Time A*

- Follow f(n) locally but note that initial state does not matter!

- Learning: update h(s) with experience to keep from getting stuck in local minima

- If interested read:

Richard E. Korf, *Real-time heuristic search*, Artificial Intelligence 1990

Beyond Classical Search

- What if actions are non-deterministic?
    - Contingency Plans, plan for multiple outcomes

- What if percepts do not give a whole state? (Partially observable)
    - Reason about **belief states**
    - Things get more complex
    - The full version is not in the scope of this course