Artificial Intelligence

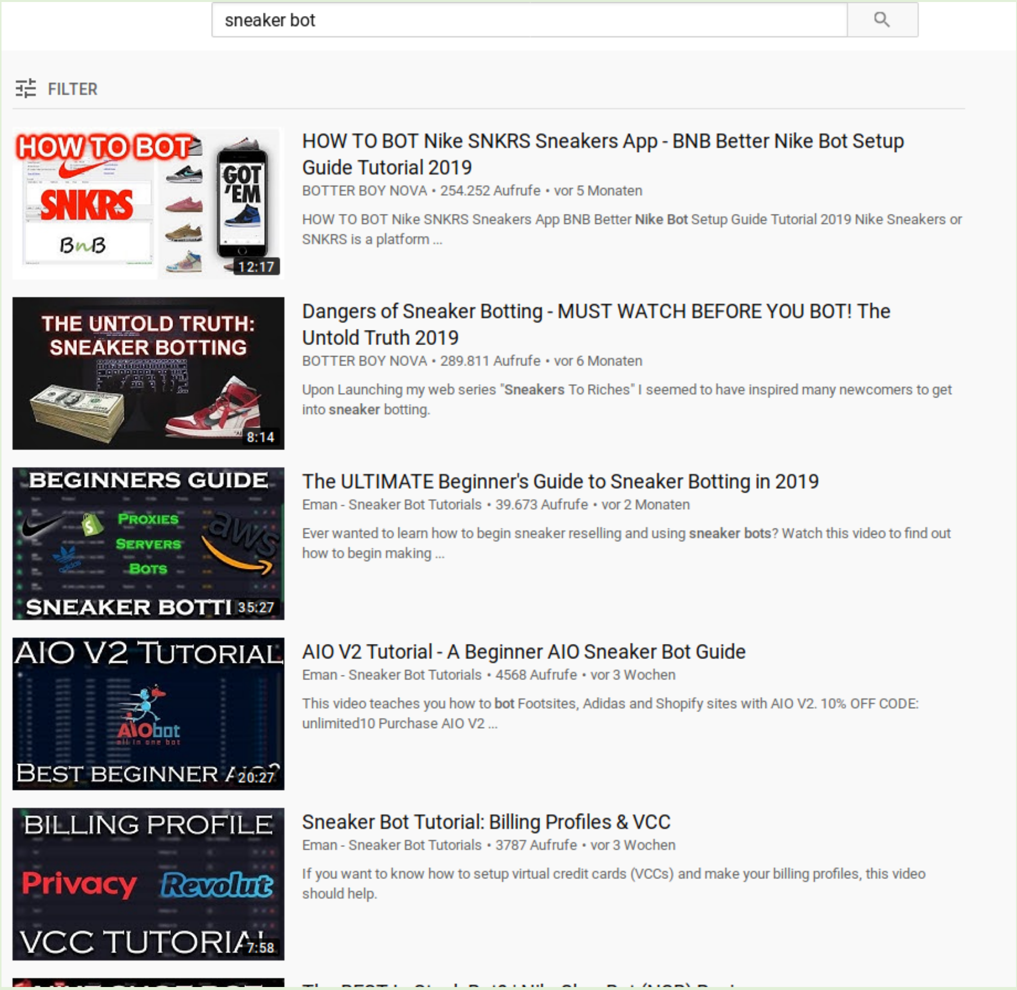
Algorithms and Applications with Python

Chapter 2

Dr. Dominik Jung
dominik.jung42@gmail.com

python

Business Case: Stop Automated Nike Shopping Agents



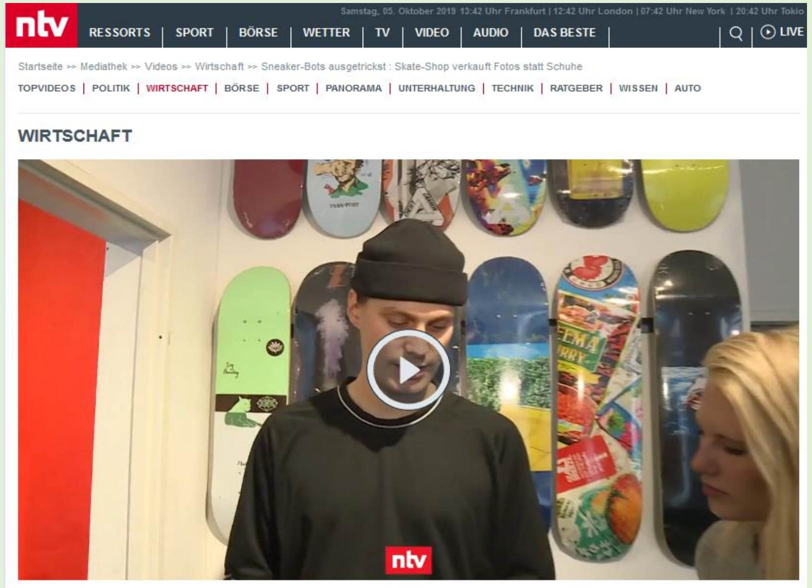
Business Case: Stop Automated Nike Shopping Agents

01 Executive Summary

Limited sneakers can be resold profitably. Professional resellers use AI-based agents that can make automated purchases. Today, there are many AI-based agents and web-crawler that are able to detect and buy limited sneakers or other limited offers in web-shops, while normal customers often go away empty-handed. Furthermore, these kind of information systems produce a lot of web traffic (especially if they are very bad designed), and most shop-owners do not want them. A sneaker shop in Frankfurt has now outwitted the software. With a simple but ingenious idea.

02 Solution

- Most simple AI agents use simple decision rules for decision-making.
- Hence, they lack of “true” intelligence and can be fooled easily by targeting on the decision-rules of such systems



03 References

- https://www.n-tv.de/mediathek/videos/wirtschaft/Skate-Shop-verkauft-Fotos-statt-Schuhe-article21229466.html

Take-Aways

- Shopping agents can be used to automate human tasks (find cheap offers)
- However, they can not replace humans if it gets difficult

Outline

2 Problem-Solving Agents

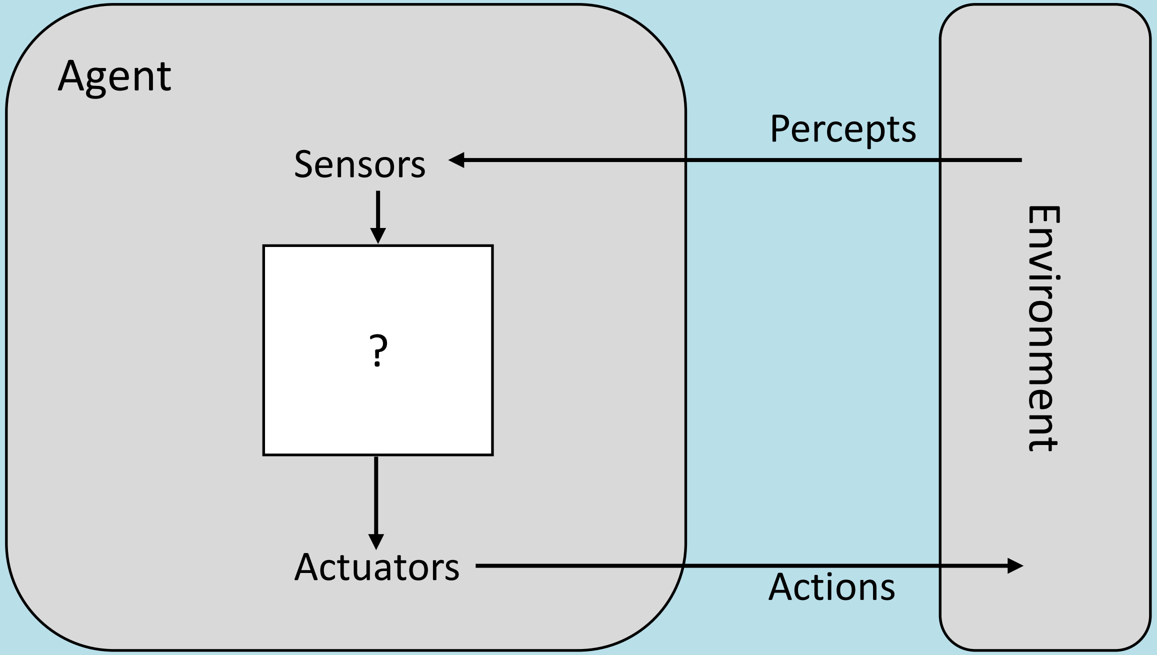
Image source: ↗ Pixabay (2019) / ↗ CC0

▶ **What we will learn:**
- We define the concept of rational agents (≈ intelligent agents)
- Characteristics of artificial agents (perfect or otherwise), the diversity of environments, and the resulting menagerie of agent types
- We discuss how AI problems can be modelled as search-problems, and how they can be solved by searching

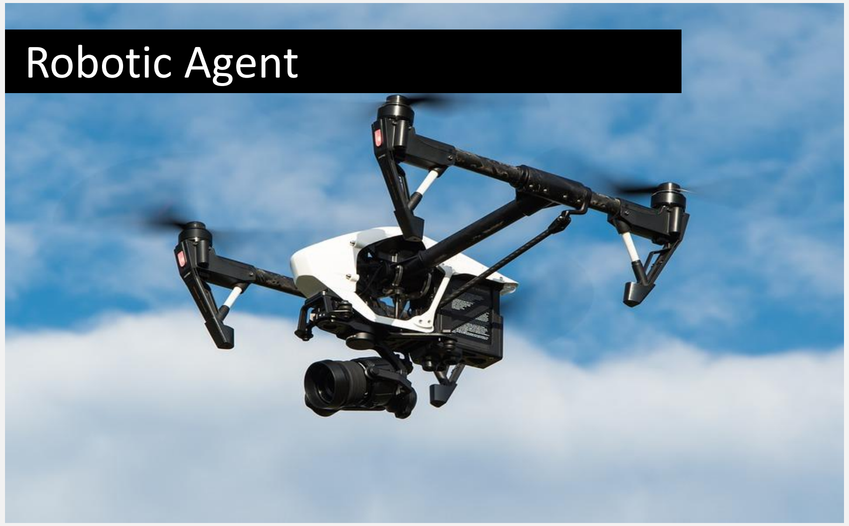
▶ **Duration:**
- 225 min

▶ **Relevant for Exam:**
- 2.1 – 2.5

2.1 „Intelligent“ Agents (Russel, Norvig)

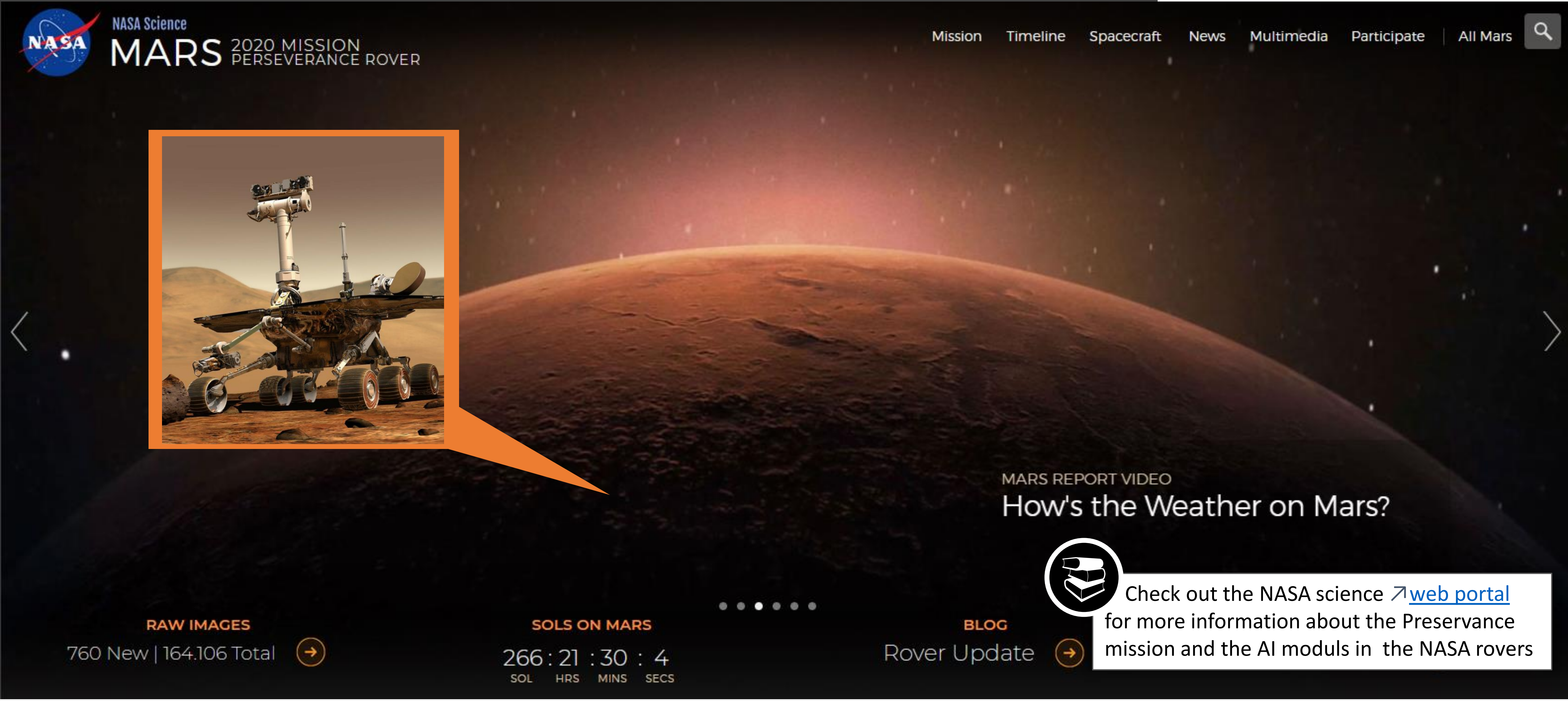


> **D** *An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators (*Rusell & Norvig, 2016*)*



Adapted from Rusell, S., & Norvig, P. (2016) | Image source: ↗ [Pixabay](https://pixabay.com) (2019) / ↗ CC0

2.1 NASA Perseverance Rover



Check out the NASA science ↗[web portal](https://mars.nasa.gov/mars2020) for more information about the Preservance mission and the AI moduls in the NASA rovers

Image source: NASA (2022) ↗ https://mars.nasa.gov/mars2020 ; ↗ Pixabay (2019) / ↗ CC0

2.1 Agent in this Lecture

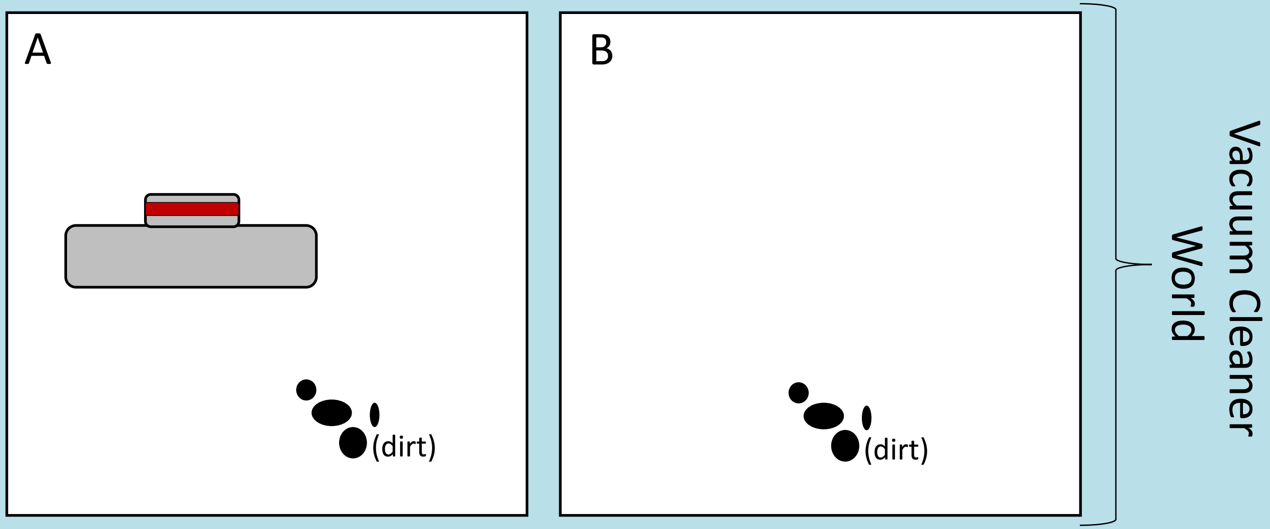
- Examples of agents
  - A web shopping program
  - An automated factory module
  - A traffic control system
  - NASA's Perseverance Rover

- Focus in this lecture: **Software agents** that gathers information about an environment and takes actions based on that

Image source: ↗ Pixabay (2019) / ↗ CC0

How do you design an intelligent agent?

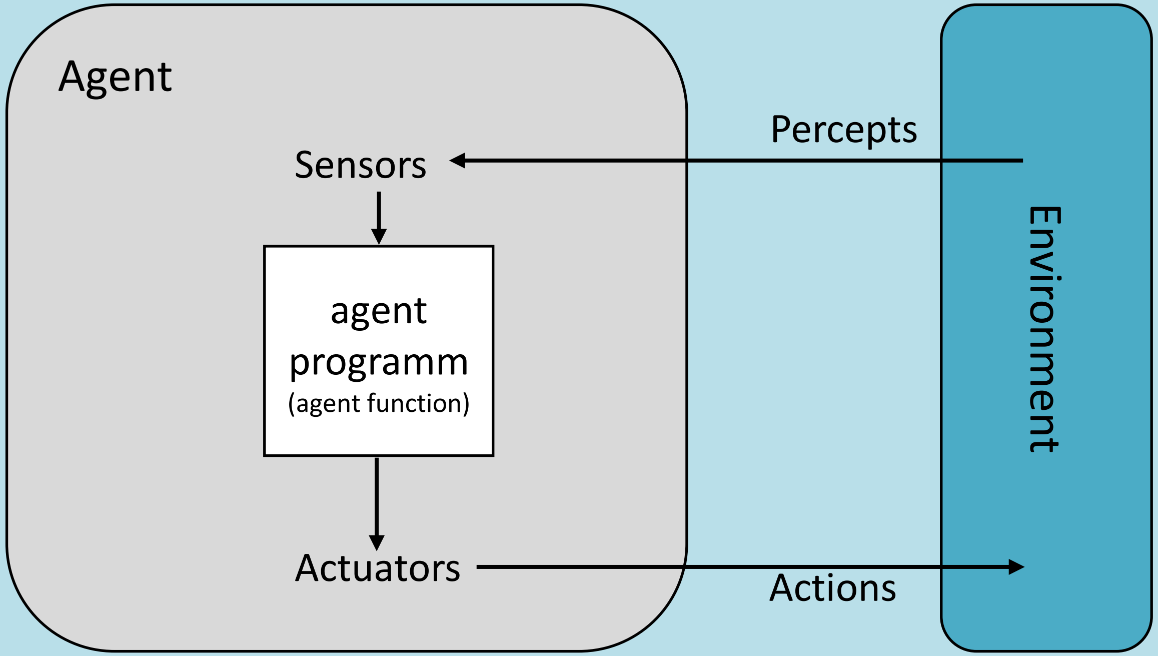
2.1 Example: Vacuum Cleaner World - Tabulation



- We use a very simple example: the vacuum-cleaner world with just two squares (A and B)
- The vacuum cleaner agent perceives which square it is and whether there is dirt or not
- It can move (left or right), suck up the dirt or do nothing.
- One simple function is: If the current square is dirty, then suck

Adapted from Rusell, S., & Norvig, P. (2016) | Image sources: Dominik Jung (2019)

2.1 Step 1 – Specifying the Task Environment



- How does the task and the environment in which the task should be solved look like?

Adapted from Rusell, S., & Norvig, P. (2016)

2.1 Specifying the Task Environment

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

- Norvig and Russel propose to characterize the task environment based on the four characteristics: Performance, Environment, Actuators and Sensors (PEAS)
- In designing an agent, the first step must always be to specify the task environment as fully as possible.

Table and example adapted from Rusell, S., & Norvig, P. (2016)

2.1 Further Properties of Task Environments

- Fully observable or partially observable
- Single agent or multiagent
- Deterministic or stochastic
- Episodic or sequential
- Static or dynamic
- Discrete or continuous
- Known or unknown

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle						
Chess with a clock						
Poker						
Backgammon						
Taxi driving						
Medical diagnostics						
Image Analysis						
Part-Picking Robot						
Refinery Controller						
Interactive English Tutor						

Adapted from Rusell, S., & Norvig, P. (2016)

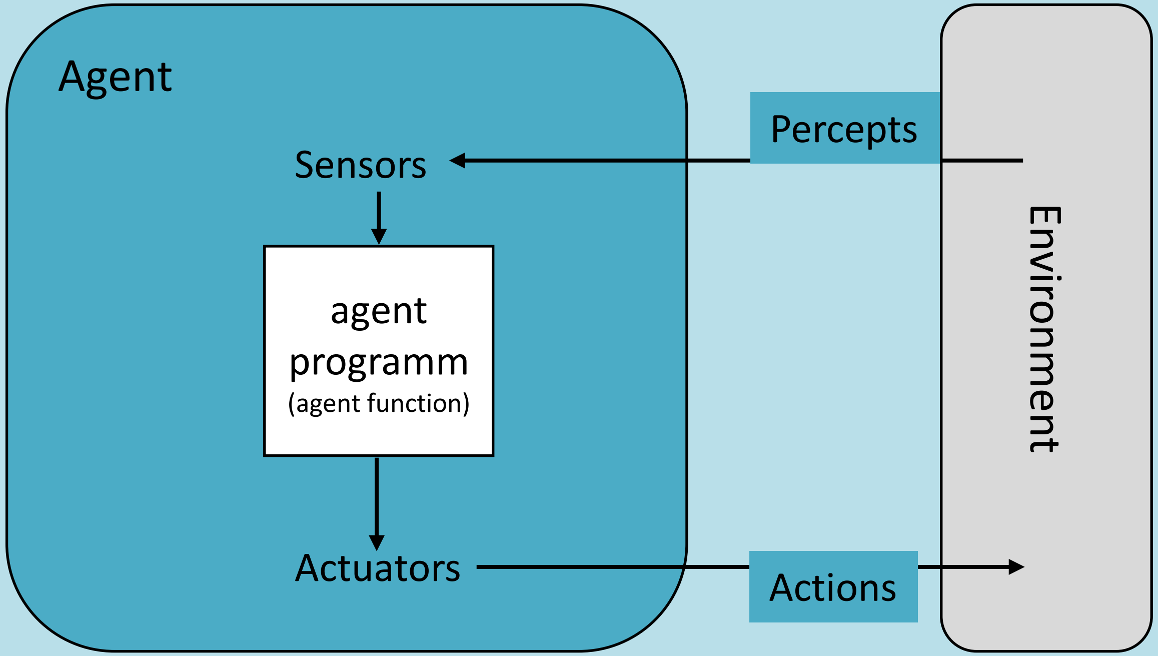
2.1 How to Get Task Environments in Development?

- We will use simulators that provide different environments to test our agents (see lectorials)

- The simulator takes one or more agents as input, provides each agent with the correct perceptions, accepts the agent's actions, and updates the environment based on the actions and possibly other influences

Adapted from Rusell, S., & Norvig, P. (2016)

2.1 Step 2 – Specifying the Agent



Agent = architecture + program

- How should the agent act, what would be „intelligent behaviour"?

Adapted from Rusell, S., & Norvig, P. (2016)

2.1 What Makes Agents Intelligent?

Percept Sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
...	...
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck

- An agent's behavior is described by the agent function that maps any given percept sequence to an action.
- We could imagine this as a table matching each possible percepts and actions (remember chinese room argument?)
- Internally, the agent function is implemented by an agent program

Adapted from Rusell, S., & Norvig, P. (2016)

2.1 Good Behaviour: Conceptionalizing Rationality in AI

- For a vacuum cleaner: What does it mean to do the right thing?

- What is rational, at any given time, depends on four things:
  - The performance measure that defines the criterion of success.
  - The agent’s prior knowledge of the environment.
  - The actions that the agent can perform.
  - The agent’s percept sequence to date

D

Rational Agent
*For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has (*Rusell & Norvig, 2016, p.37*)*

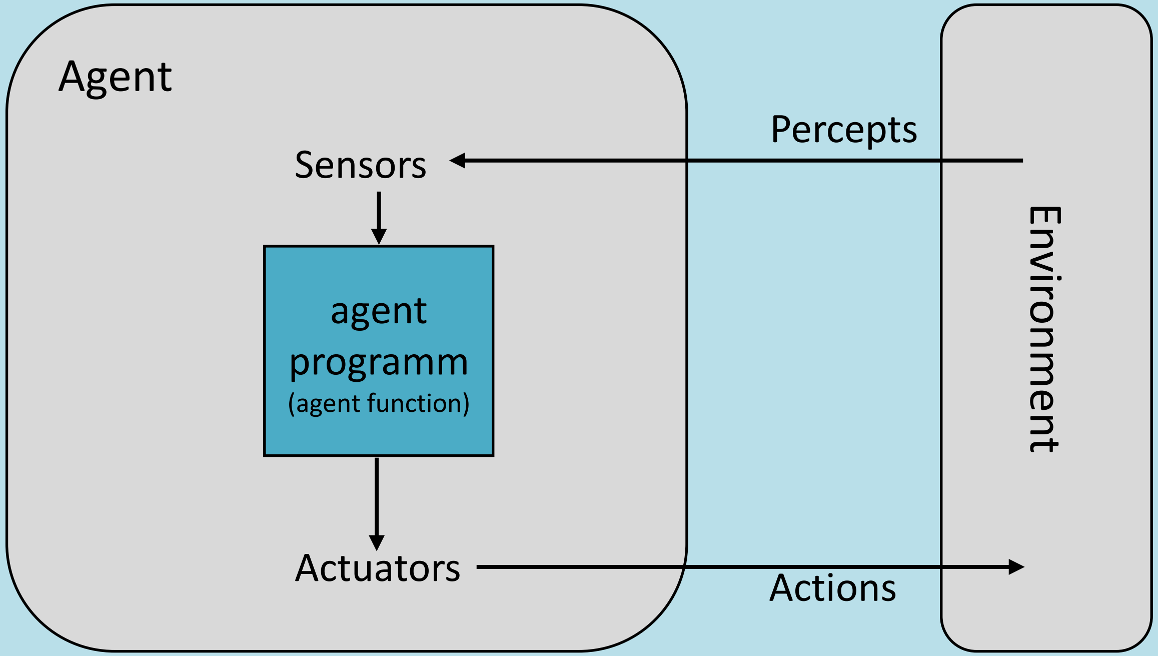
Adapted from Rusell, S., & Norvig, P. (2016)

2.1 Omniscience, Learning, and Autonomy

- We need to be careful to distinguish between rationality and omniscience
- Rationality maximizes **expected** performance, while perfection maximizes **actual** performance
- Norvig & Russels' definition of rationality does not require omniscience, because the rational choice depends only on the percept sequence **to date**
- Their definition requires a rational agent not only to gather information but also to **learn** as much as possible from what it perceives.

Adapted from Rusell, S., & Norvig, P. (2016)

2.1 Step 3 – Specifying the Agents Behaviour



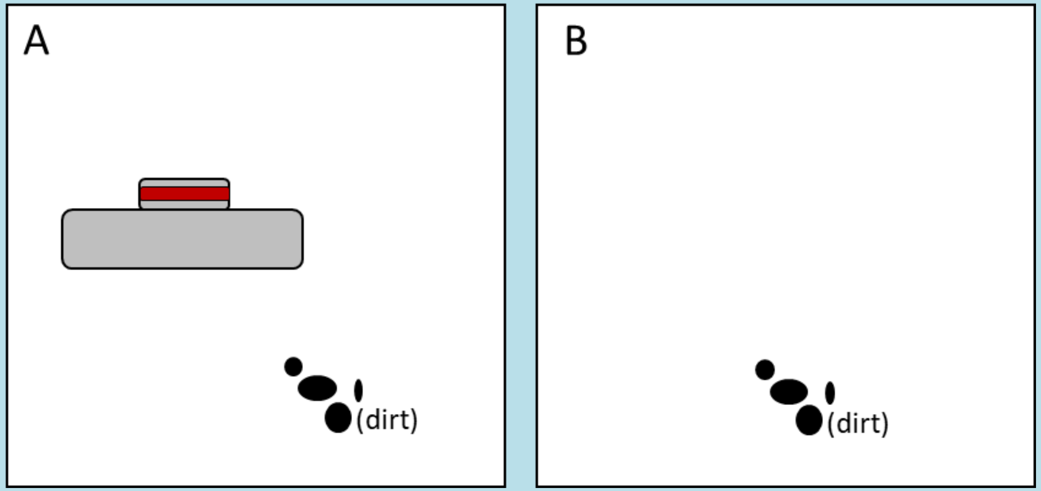
Agent = architecture + program

- How can we implement the „intelligent“ behaviour?

Adapted from Rusell, S., & Norvig, P. (2016) | Image source: ↗ [Pixabay](#) (2019) / ↗ [CC0](#)

2.1 Structure of Agents (Architecture)

- The job of AI specialists is to design an agent program that implements the agent function/the mapping from percepts to actions
- This program runs on some sort of computing device with physical sensors and actuators (architecture)
- *Agent = architecture + program*



Adapted from Rusell, S., & Norvig, P. (2016)

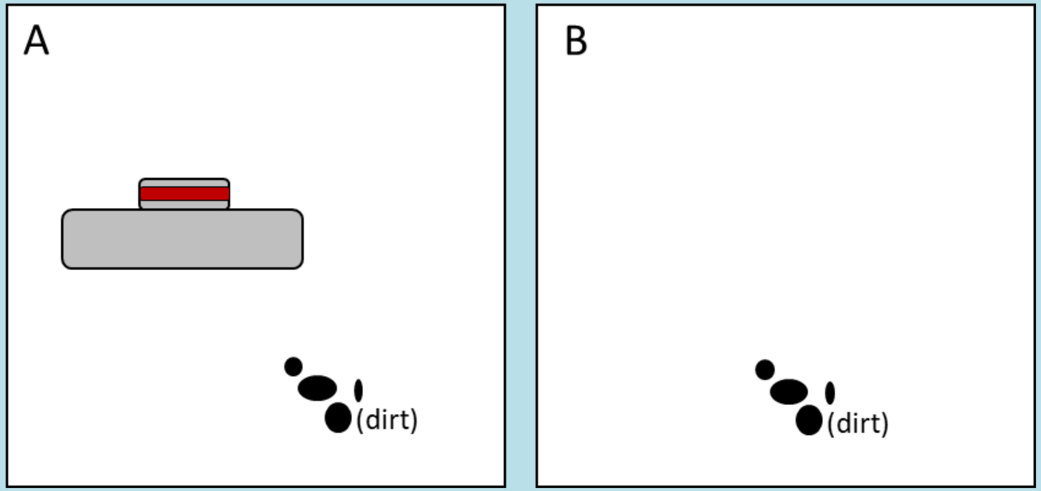
2.1 Structure of Agents (Architecture)

Algorithm: Reflex-Vacuum Agent

$\textit{if}\ status = dirty\ \textbf{then}$
    $return\ suck$
$\textbf{end}$

$\textbf{else}\ \textit{if}\ location = A\ \textbf{then}$
    $return\ right$
$\textbf{end}$

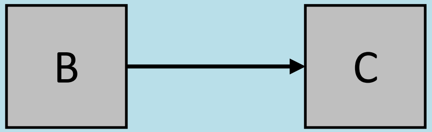
$\textbf{else}\ \textit{if}\ location = B\ \textbf{then}$
    $return\ left$
$\textbf{end}$



Adapted from Rusell, S., & Norvig, P. (2016)

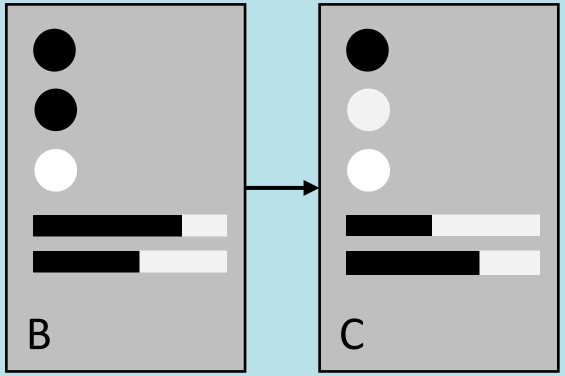
2.1 Components of Agents

Atomic representation



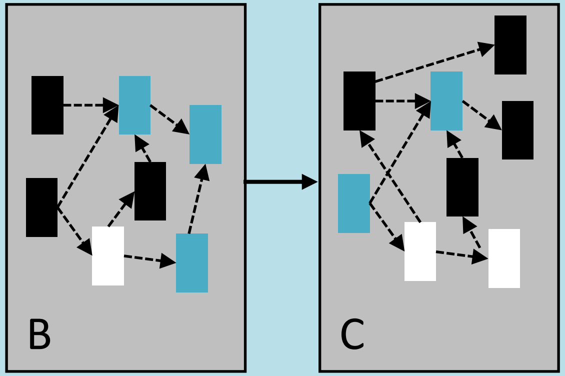
- a state (such as B or C) is a black box with no internal structure

Factored representation



- a state consists of a vector of attribute values; values can be Boolean, realvalued, or one of a fixed set of symbol

Structured representation



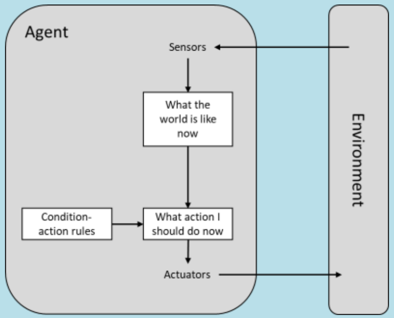
- a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

Adapted from Rusell, S., & Norvig, P. (2016)

2.1 Subtypes of Agents (Russell & Norvig, 2016)

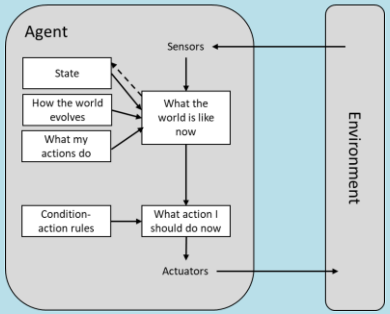
Rational Agents

Simple Reflex Agent



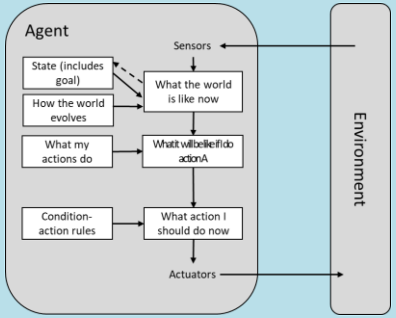
- Agents select actions on the basis of the current percept
- Work only when the correct decision can be made on the basis of the current percept

Model-based Agent



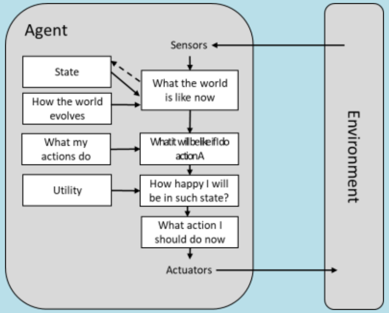
- Agents select actions on the basis of the current state-model / state description

Goal-based Agent



- Agents select actions on the basis of as well current state, and some sort of goal information that describes situations that are desirable

Utility Agent



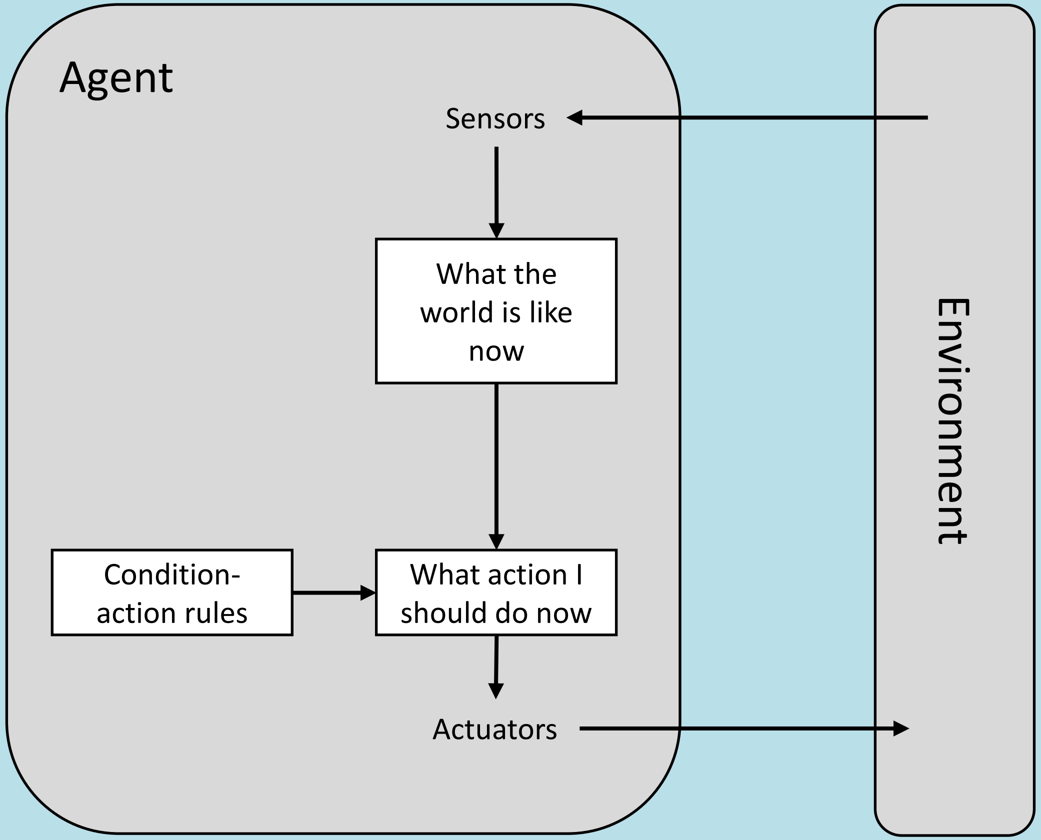
- Agents select actions by maximizing the utility function (if the external performance measure and the utility function are in agreement)

Learning Agent	All the agents can be generalized as learning agent

Adapted from Rusell, S., & Norvig, P. (2016)

2.1 Simple Reflex Agent

- Select actions on the basis of the current percept, ignoring the rest of the percept history



Adapted from Rusell, S., & Norvig, P. (2016)

Algorithm: Simple Reflex Agent

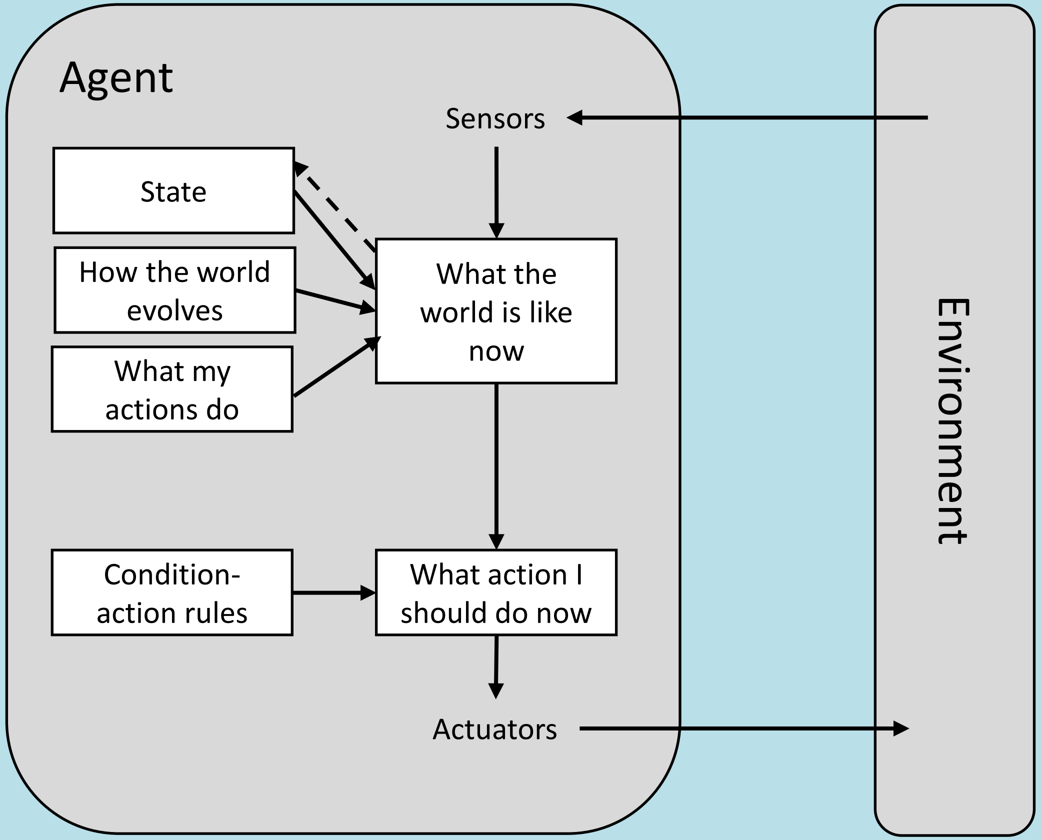
persistent: *rules*, a set of condition–action rules

state ← *INTERPRET-INPUT(percept)*
rule← *RULE-MATCH(state, rules)*
action ← *rule.ACTION*

return *action*

2.1 Model-based Reflex Agent

▶ Keep track of the part of the world it can't see now



Adapted from Rusell, S., & Norvig, P. (2016)

Algorithm: Model-based Reflex Agent

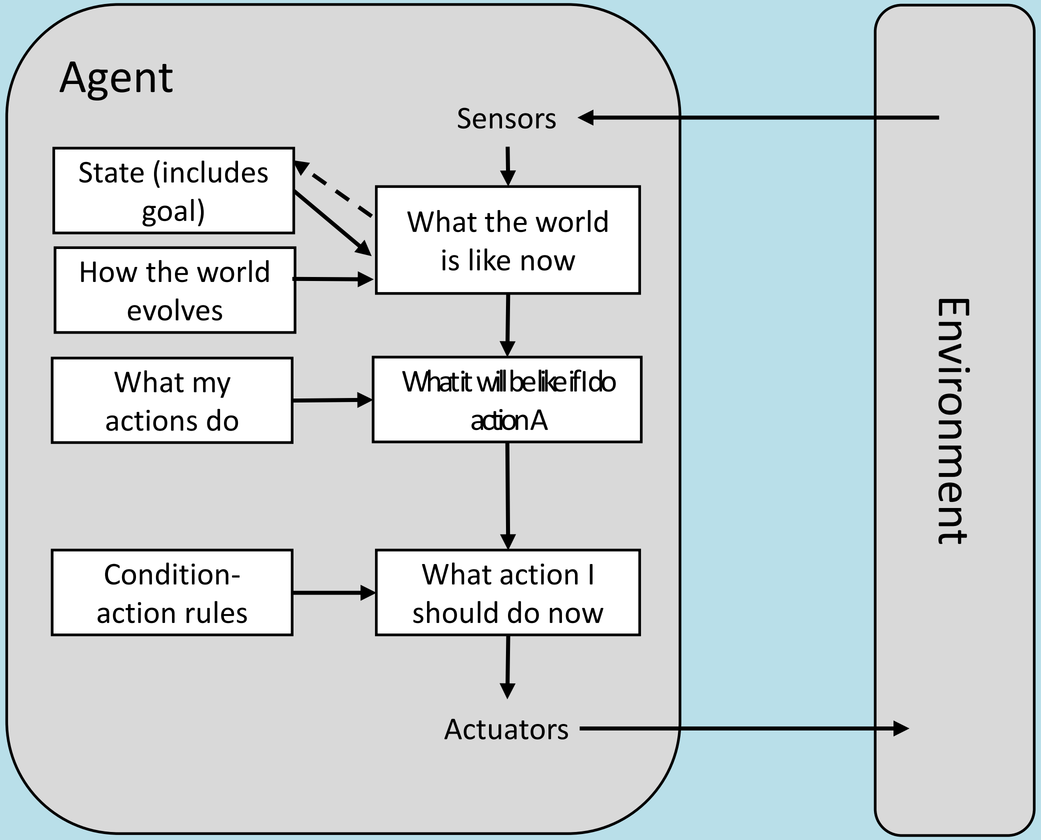
persistent:
state, the agent's current conception of the world state
model, a description of how the next state depends on current state and action
rules, a set of condition–action rules
action, the most recent action, initially

state ← UPDATE-STATE(state, action, percept, model)
rule← RULE-MATCH(state, rules)
action ← rule.ACTION

return *action*

2.1 Goal-based Agents

- Besides state description the agent uses goal information in its decision process



Adapted from Rusell, S., & Norvig, P. (2016)

Algorithm: Goal-based Agent

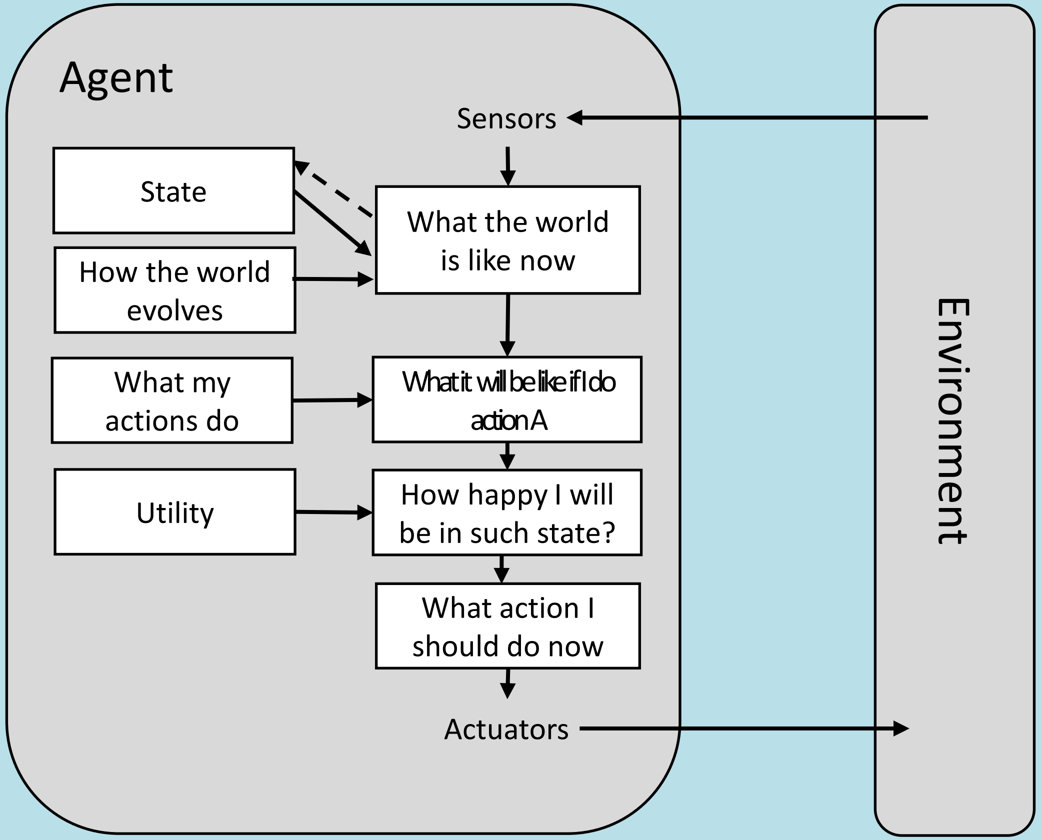
***persistent**:*
state, the agent's current conception of the world state
model, a description of how the next state depends on current state and action
rules, a set of condition–action rules
action, the most recent action, initially
goal, desired result of the agents behaviour

state ← UPDATE-STATE(state, action, percept, model, goal)
rule← RULE-MATCH(state, rules)
action ← rule.ACTION

***return** action*

2.1 Utility-based Agent

▶ Utility function to describe desired behavior



Adapted from Rusell, S., & Norvig, P. (2016)

Algorithm: Goal-based Agent

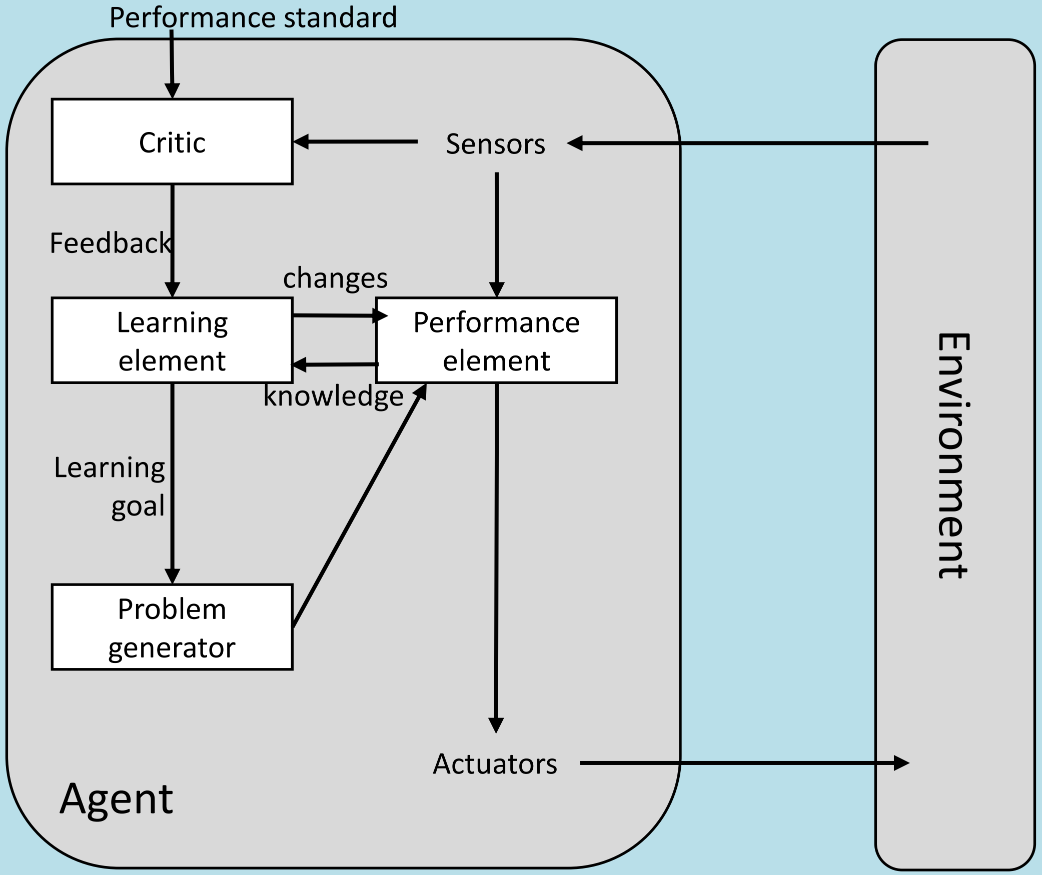
persistent:
state, the agent’s current conception of the world state
model, a description of how the next state depends on current state and action
rules, a set of condition–action rules
action, the most recent action, initially
Utility function, performance measure

state ← UPDATE-STATE(state, action, percept, model, utility function)
rule← RULE-MATCH(state, rules)
action ← rule.ACTION

return *action*

2.1 Conclusion: The Learning Agent

▶ Agent improves its performance by learning optimal outputs



Adapted from Rusell, S., & Norvig, P. (2016)

- A learning agent can be divided into four conceptual components: Learning element, performance element, critic, and problem generator
- The design of the learning element depends very much on the design of the performance element.
- The critic tells the learning element how well the agent is doing with respect to a fixed performance standard

Your turn!

Task

Please explain:

- What is the difference between an agents program and function?
- Could you model a hand-held calculator as an agent that chooses the action of displaying “`3`” when given the percept sequence “`1 + 2 =`“ ? Is this correct based on the definitions of Russel & Norvig?

Outline

Image source: ↗ Pixabay (2019) / ↗ CC0

▶ **What we will learn:**

- We define the concept of rational agents (≈ intelligent agents)
- Characteristics of artificial agents (perfect or otherwise), the diversity of environments, and the resulting menagerie of agent types
- We discuss how AI problems can be modelled as search-problems, and how they can be solved by searching

▶ **Duration:**

- 225 min

▶ **Relevant for Exam:**

- 2.1 – 2.5

2.2 Why We Need Goals: Example Roadtrip Planning

Adapted from Rusell, S., & Norvig, P. (2016) | Image source: ↗ Pixabay (2019) / ↗ CC0

- **Example**: US Nationalpark roadtrip with your new Porsche
- Next step: How to formulate AI Problems and solve them

2.2 Why We Need Goals: Example Roadtrip Planning

D *We will consider a **goal** to be a set of world states—exactly those states in which the goal is satisfied* (Rusell & Norvig, 2016, p.65*)*

Our problem can be defined formally by different aspects like:

- Initial state or start
- Driving costs
- **Visit Top 5 parks**
- **Music playlist**
- Possible driving routes, allowed streets
- What each action does
- List with all parks to check if we got them all

→ **Simplify Goal**: Be in Shenandoah

Image sources: ↗ US NationalParks (2015) by Mwierschkec↗ CC BY-SA 4.0; ↗ View from Skyline Drive (2019) by Steevven1 ↗ CC BY-SA 4.0; ↗ Schluchten des Grand Canyon (2006) by Tenji ↗ CC BY-SA 3.0
And yes I am working on a roadtrip playlist, you can check it out here (↗ Spotify), feel free to give any suggestions

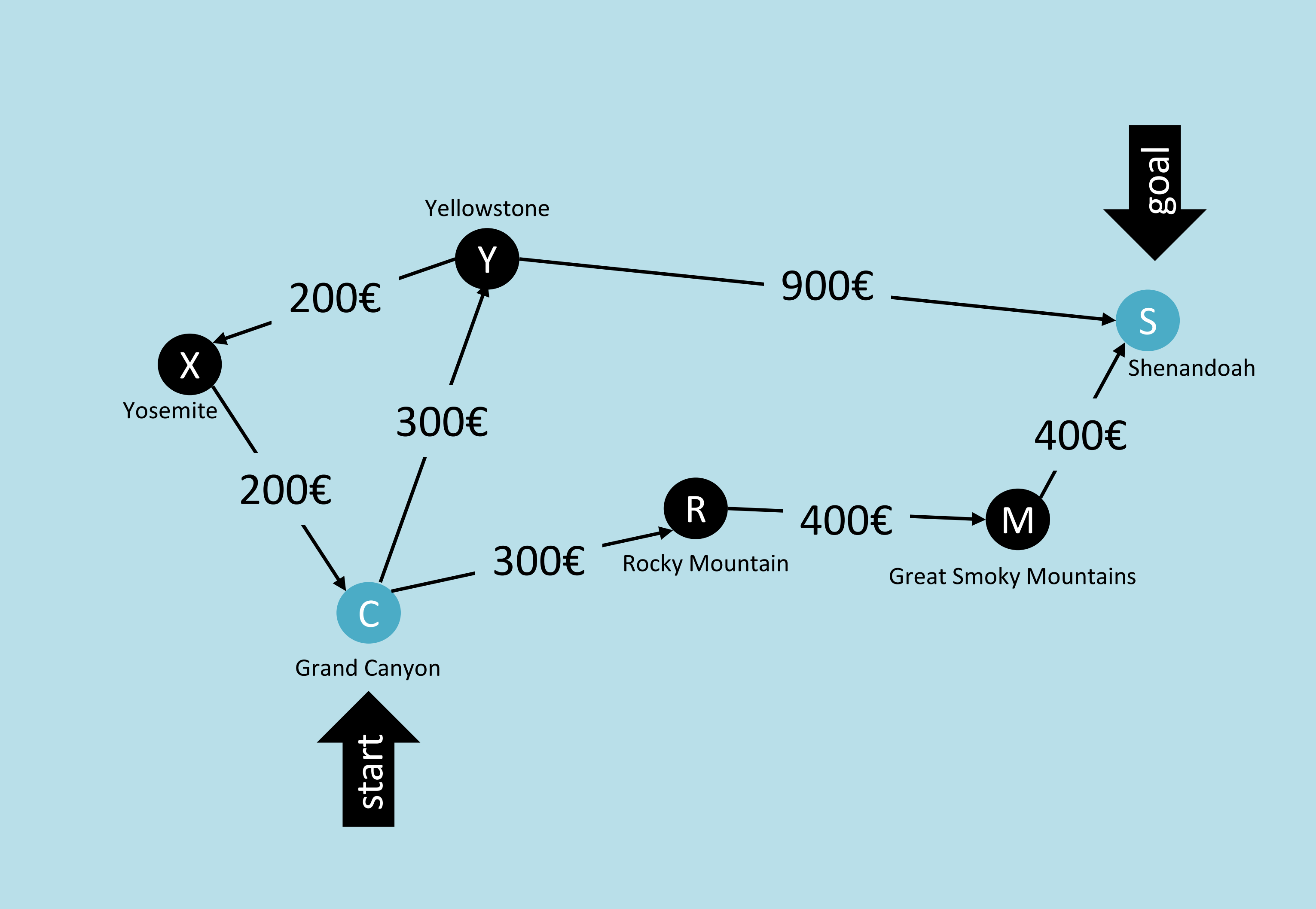
2.2 Well-defined Problems and Solutions

Adapted from Rusell, S., & Norvig, P. (2016)

Let us now try to model our problem in a way a computer can solve it:

- In a **first step**, we want to build an agent that finds the **optimal route from start to end** (route-finding problem)
- Our **second step** is to have an agent that finds the **optimal route for all/Top 5 mainland US parks** (touring-problem)

2.2 Model the Roadtrip as an AI problem



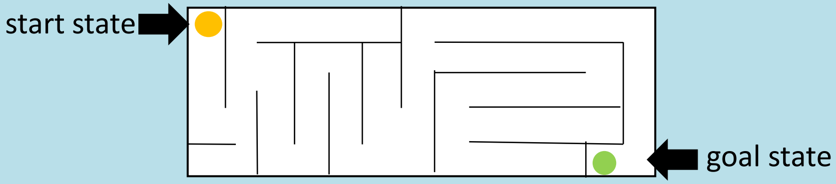
Goal: Be in Shenandoah

Other aspects:

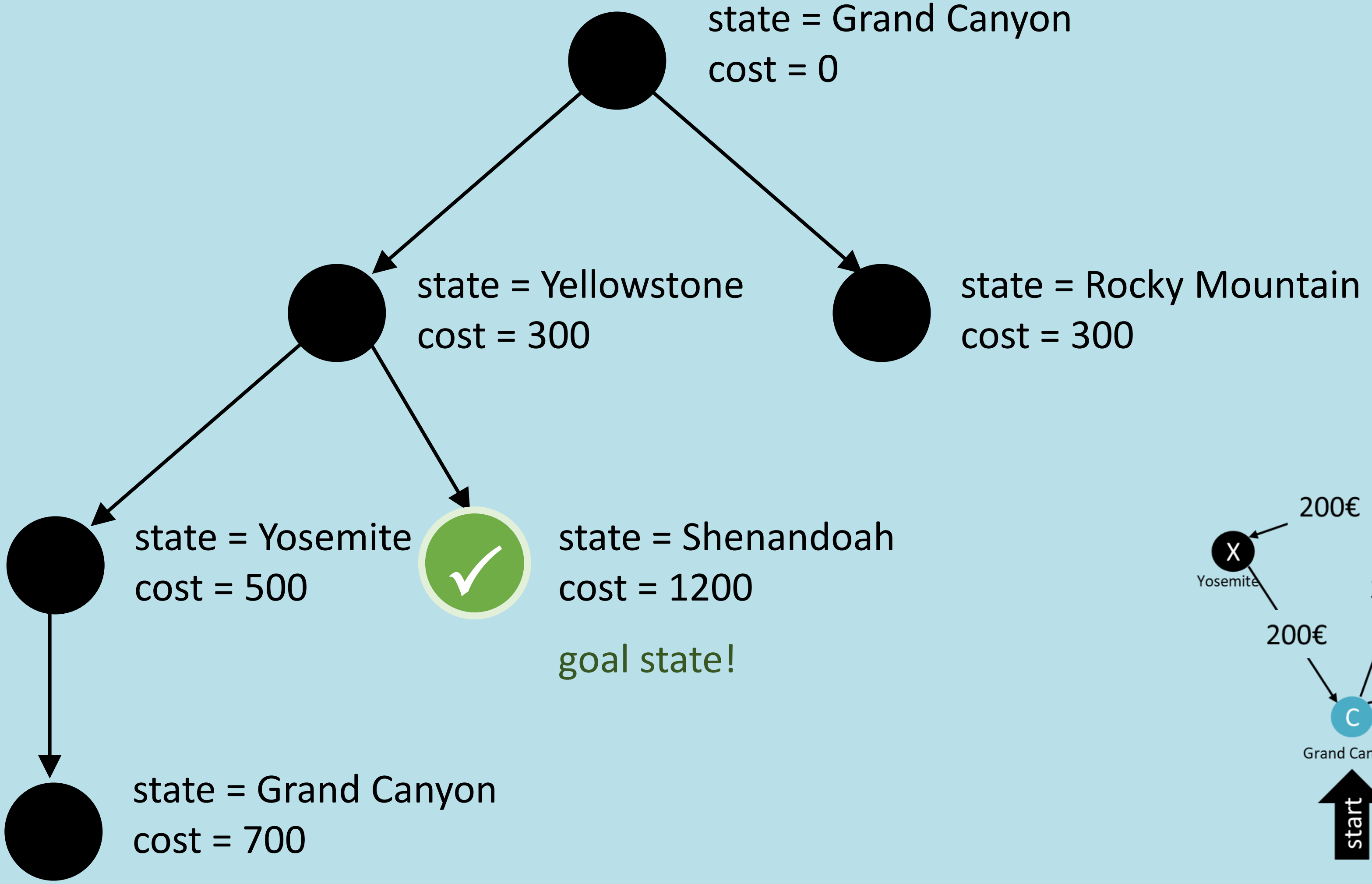
- Initial state or start: Grand Canyon
- Consider Driving costs
- Visit Top 5 parks

2.2 Possible Solution: Model Roadtrip-Problem as Search

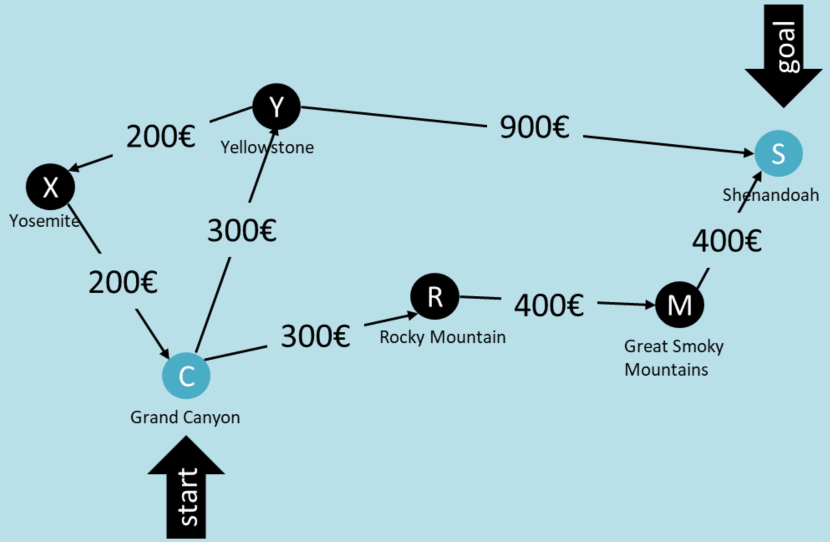
- Model problems as search problems
- Our agent has not to execute all actions in real life while searching for solution
- We want to find a sequence of actions that will lead us to a desired state
  - We want to minimize number of actions
  - We want to minimize total cost of actions (more general speaking)



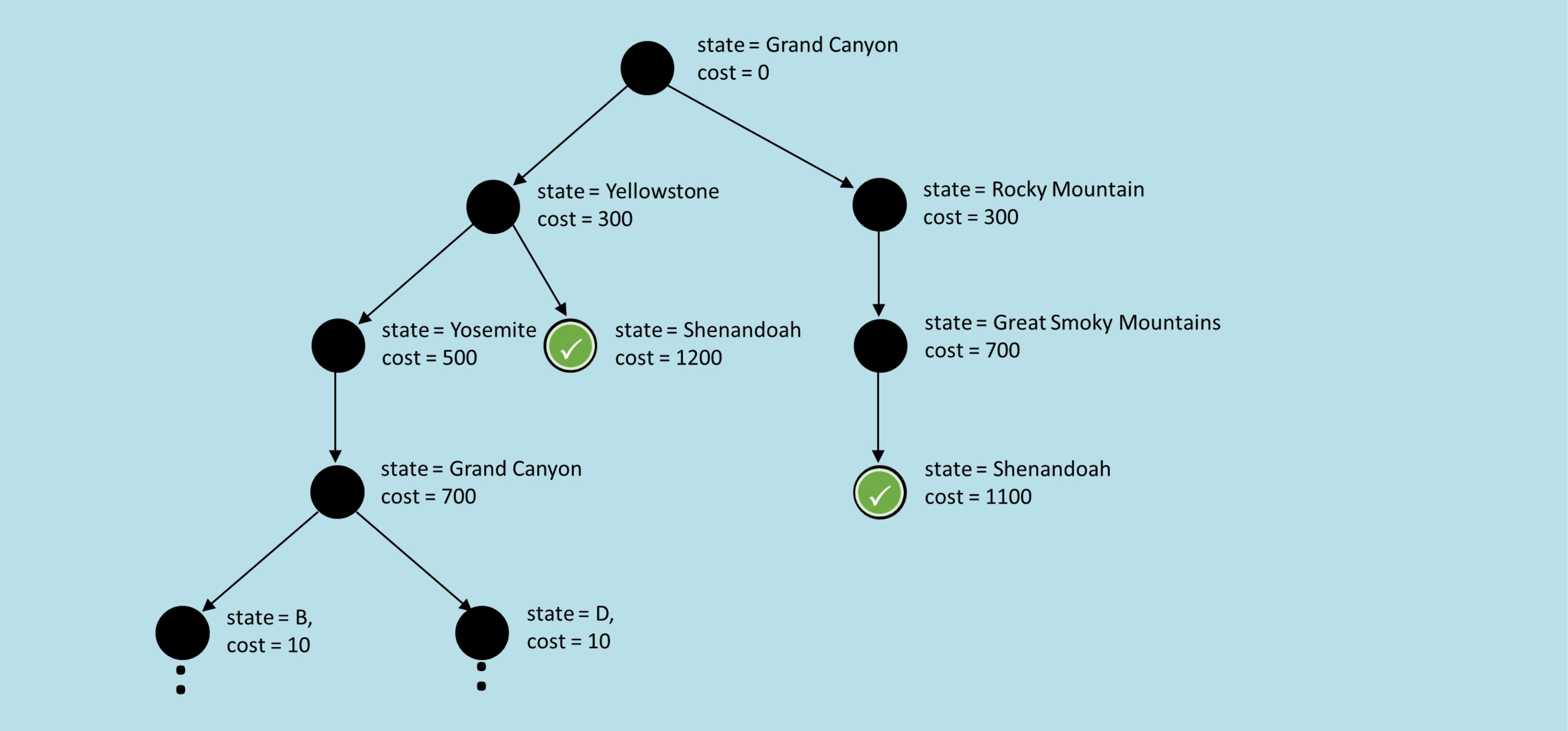
2.2 Model Problem as Search-tree ▶ Step 1



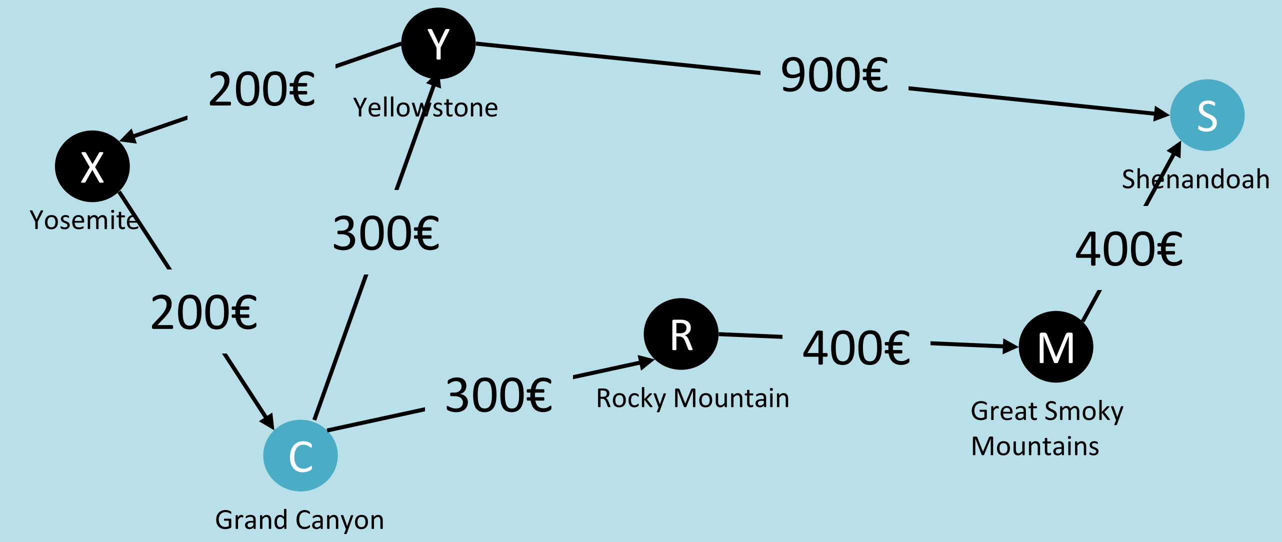
Search tree nodes and states are not the same thing!



2.2 Model Problem as Search-tree ▶ Step 2



2.2 From Rout-Finding to Touring-Problems: Visit TOP 5 Nationalparks

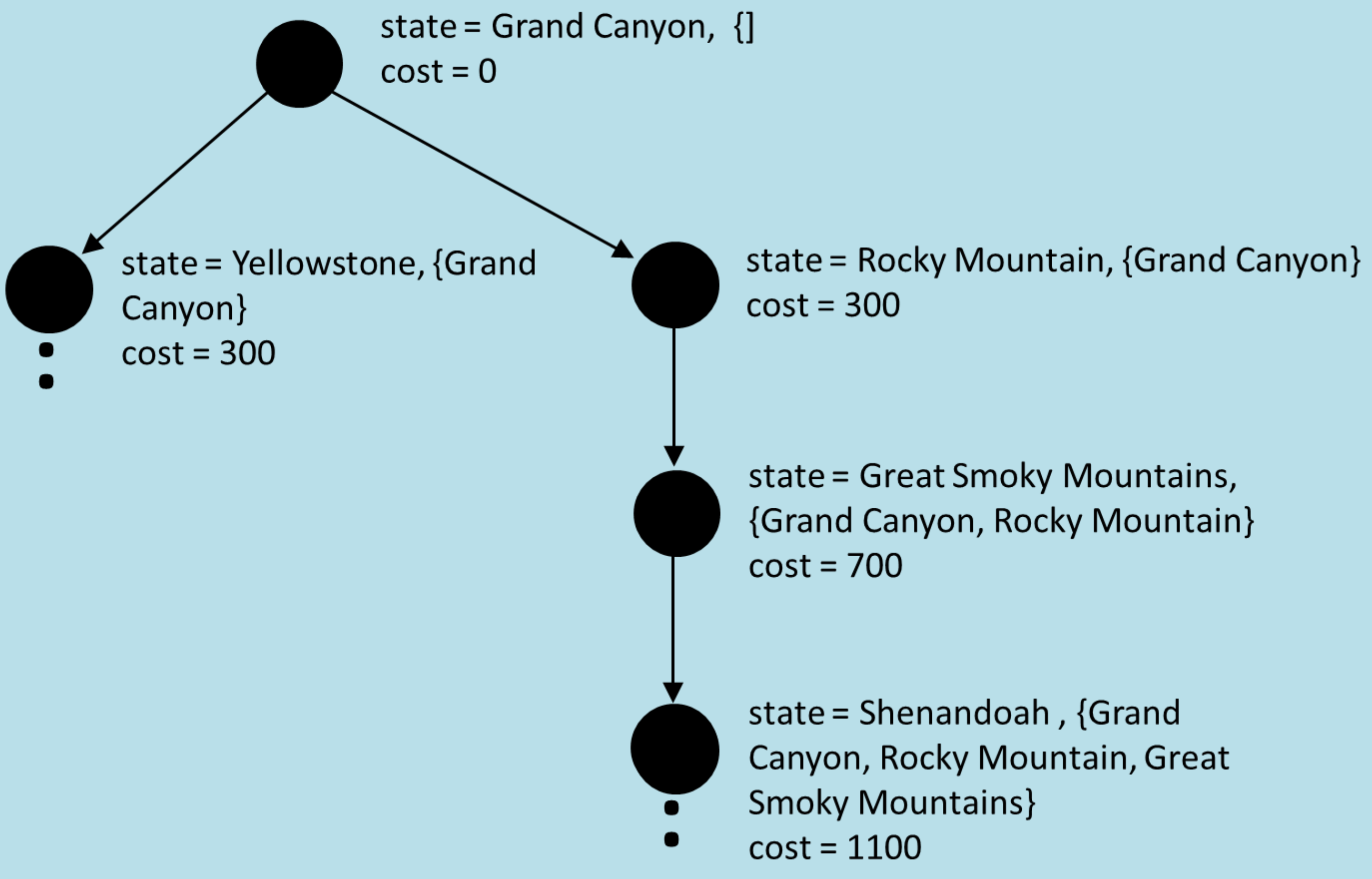


Goal: Visit all vertices on the graph

- As with before, the actions correspond to trips between adjacent parks.
- The state space is quite different: Each state must include not just the current location but also the set of parks the agent has visited.

- **Problem**: large number of states

2.2 Full Search Tree



Tree gets incredible big, due to large number of states: $n \cdot 2^{n-1}$

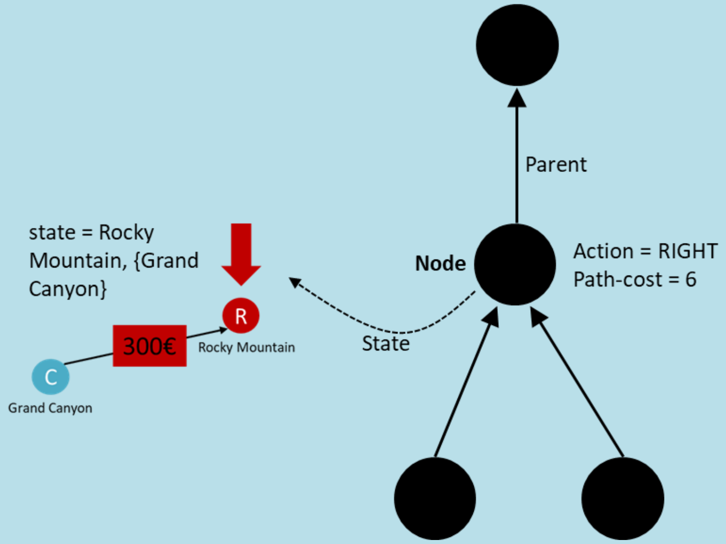
2.2 Summary Key Concepts in Search

Set of **states** that we can be in

- Including an initial state
- and goal states (equivalently, a goal test)

For every state, a set of **actions** that we can take

- Each action results in a new state
- Typically defined by successor function



Transition model: Given a state, produces all states that can be reached from it

Cost function that determines the cost of each action (or path = sequence of actions)

Solution: path from initial state to a goal state (optimal solution: solution with minimal cost)

Adapted from Rusell, S., & Norvig, P. (2016)

2.2 Problem-Solving Agents

- We can formalize this concepts into a general concept of an simple problem-solving agent
- For that purpose, we say that AI problems should be defined formally by five components:
  - initial state
  - description of actions
  - what each action does (transition model)
  - goal test
  - path cost function

operators

Algorithm: Simple Problem-Solving Agent

persistent:
seq, an action sequence, initially empty
state, a description of the current world state
goal, a goal, initially null
problem, a problem formulation

state ← *UPDATE-STATE(state, percept)*
If *seq* is empty ***then***
    goal← *FORMULATE-GOAL(state)*
    problem← *FORMULATE-PROBLEM(state, goal)*
    seq← *SEARCH(problem)*
    If *seq = failure* ***then return*** null action
action ← *FIRST(seq)*
seq← *REST(seq)*

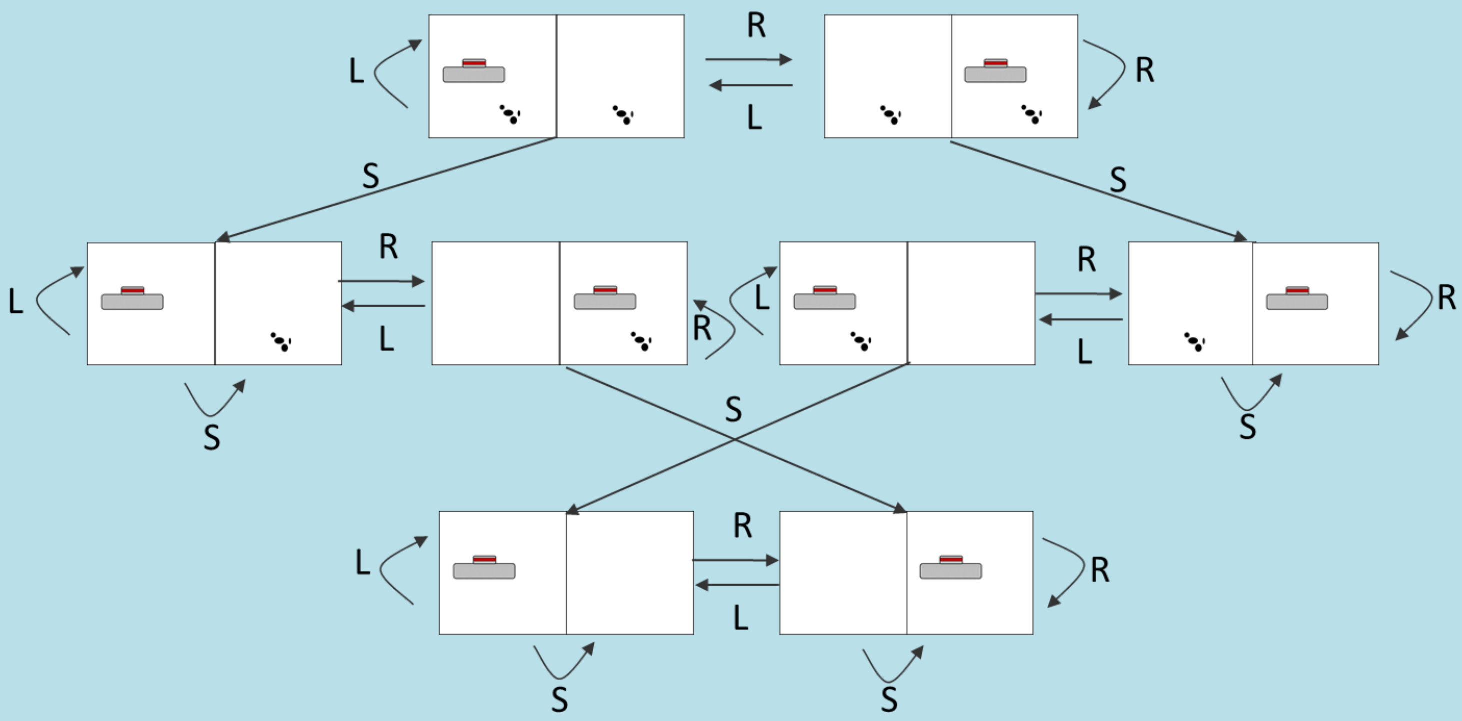
return *action*

2.2 What We Did so Far - Formulating Problems for AI

- In the preceding section we proposed a simple formulation of the problem of planning an US nationalpark trip
- This formulation seems reasonable, but it is still an abstract mathematical simplification of a much more complex problem
- Many considerations are left out of our state descriptions because they are irrelevant to the problem of finding a route.
- The process of removing detail from a representation is called abstraction.
- Toy vs. real-world problem

Adapted from Rusell, S., & Norvig, P. (2016) | And yes I am working on a roadtrip playlist, you can check it out here (↗ Spotify), feel free to give any suggestions

2.2 Example: Vacuum-Cleaner



Adapted from Rusell, S., & Norvig, P. (2016);

- **States**: The state is determined by both the agent location and the dirt locations.
- **Initial state**: Any state can be designated as the initial state.
- **Actions**: *Left*, *Right*, and *Suck*.
- **Transition model**: The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Suck*ing in a clean square have no effect.
- **Goal test**: This checks whether all the squares are clean.
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

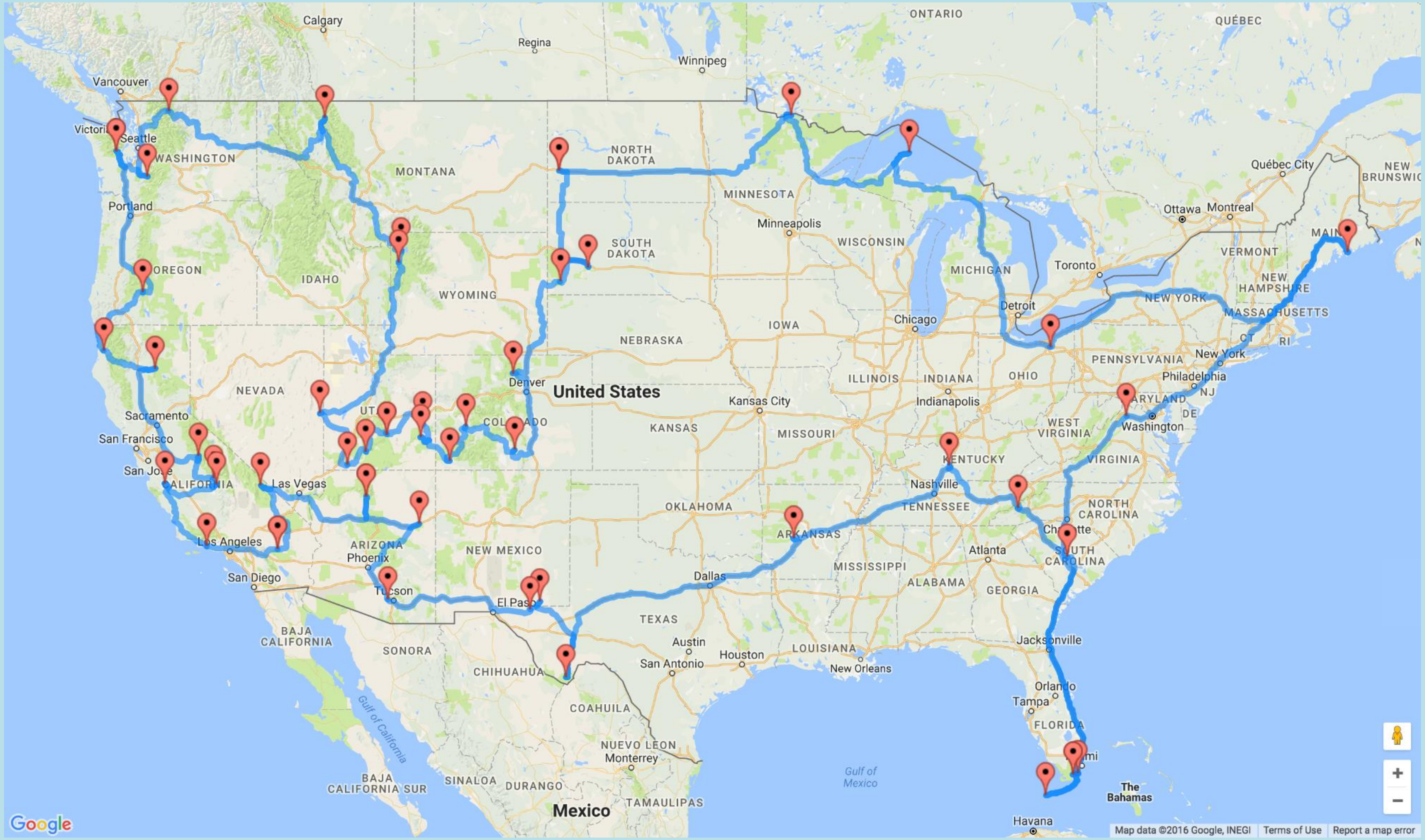
2.2 Other Popular Real-world Problems

- We tried to find
  - the **optimal route from start to end** (route-finding problem)
  - **An optimal route for to visit all TOP 5 US parks** (touring-problem)

- The **traveling salesperson problem** (TSP) is a touring problem in which each city must be visited exactly once. The aim is to find the *shortest* tour.

Adapted from Rusell, S., & Norvig, P. (2016);

2.2 Traveling Salesperson Problem of all 47 Mainland US Nationalparks



Credits for this route and Image source: ↗ [Dr. Randal Olson](#) (2019)

2.2 Other Popular Real-world Problems

- The **traveling salesperson problem** (TSP) is a touring problem in which each city must be visited exactly once. The aim is to find the *shortest* tour.

- A **VLSI layout** problem requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield

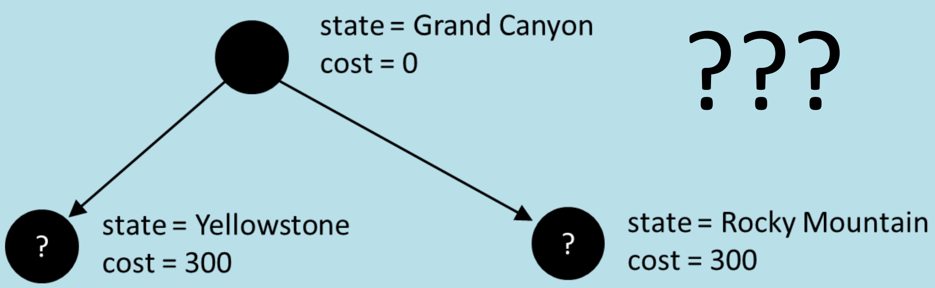
- **Protein design**: in which the goal is to find a sequence of amino acids that will fold into a three-dimensional protein with the right properties to cure some disease.

Adapted from Rusell, S., & Norvig, P. (2016);

2.2 Generic Search Algorithm

- Recap: We will consider the problem of designing **goal-based** agents in **observable, deterministic, discrete, known** environments

- **Key question in search**: Which of the generated nodes do we expand next?



Algorithm: Tree Search

initialize the frontier using the initial state of problem

loop do
  if the frontier is empty **then return** failure
  choose a leaf node and remove it from the frontier

  if the node contains a goal state **then return** the corresponding solution

expand the chosen node, adding the resulting nodes to the frontier

Adapted from Rusell, S., & Norvig, P. (2016)

2.2 Overview: Fundamental Search Algorithms

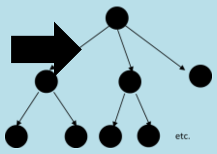
D

Uninformed Search

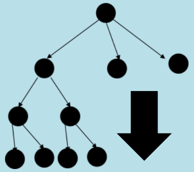
Strategies have no additional information about states beyond that provided in the problem definition

Uninformed Search

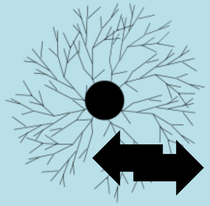
- Breadth-first (BFS)
- Depth-first search (DFS)
- Bidirectional



- Root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.



- Always expands the deepest node in the current frontier of the search tree.



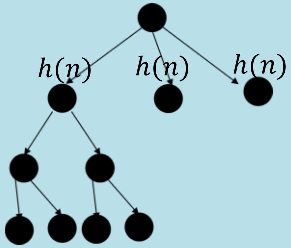
- Always expands the deepest node in the current frontier of the search tree.

Informed Search

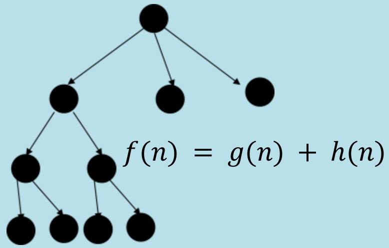
Uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy.

Informed Search

- Greedy best-first
- A-Star (A*)



- Expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.



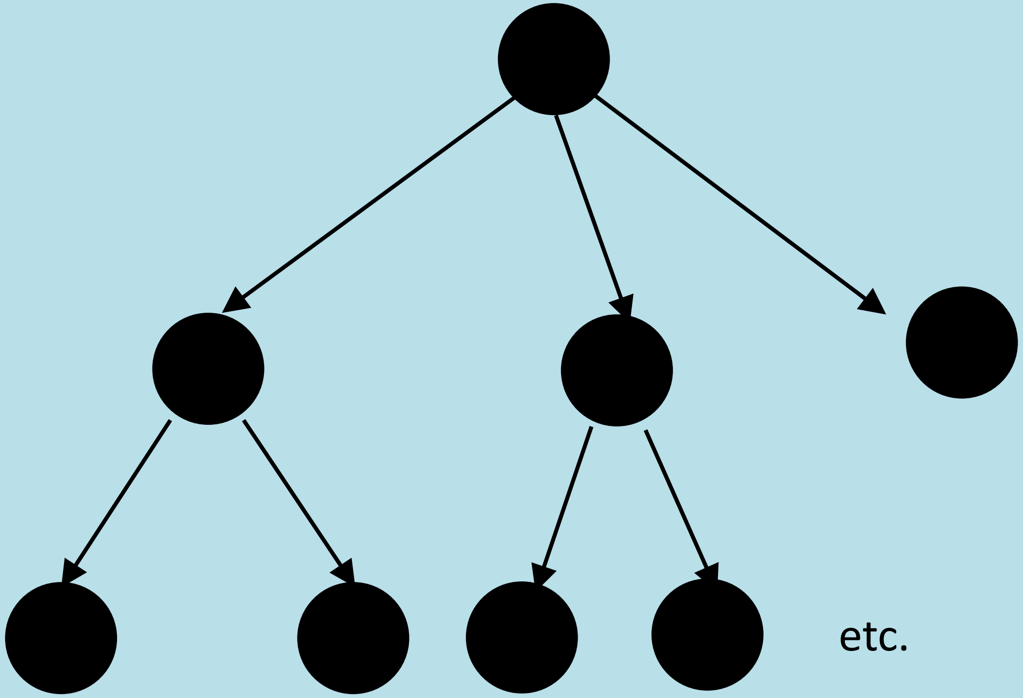
- Cluster observations into (distinct/different) groups

2.2 Uninformed Search

- Given a state, we only know whether it is a goal state or not
- Cannot say one non-goal state looks better than another non-goal state
- Can only traverse state space blindly in hope of somehow hitting a goal state at some point
  - Also called blind search
  - Blind does **not** imply unsystematic!

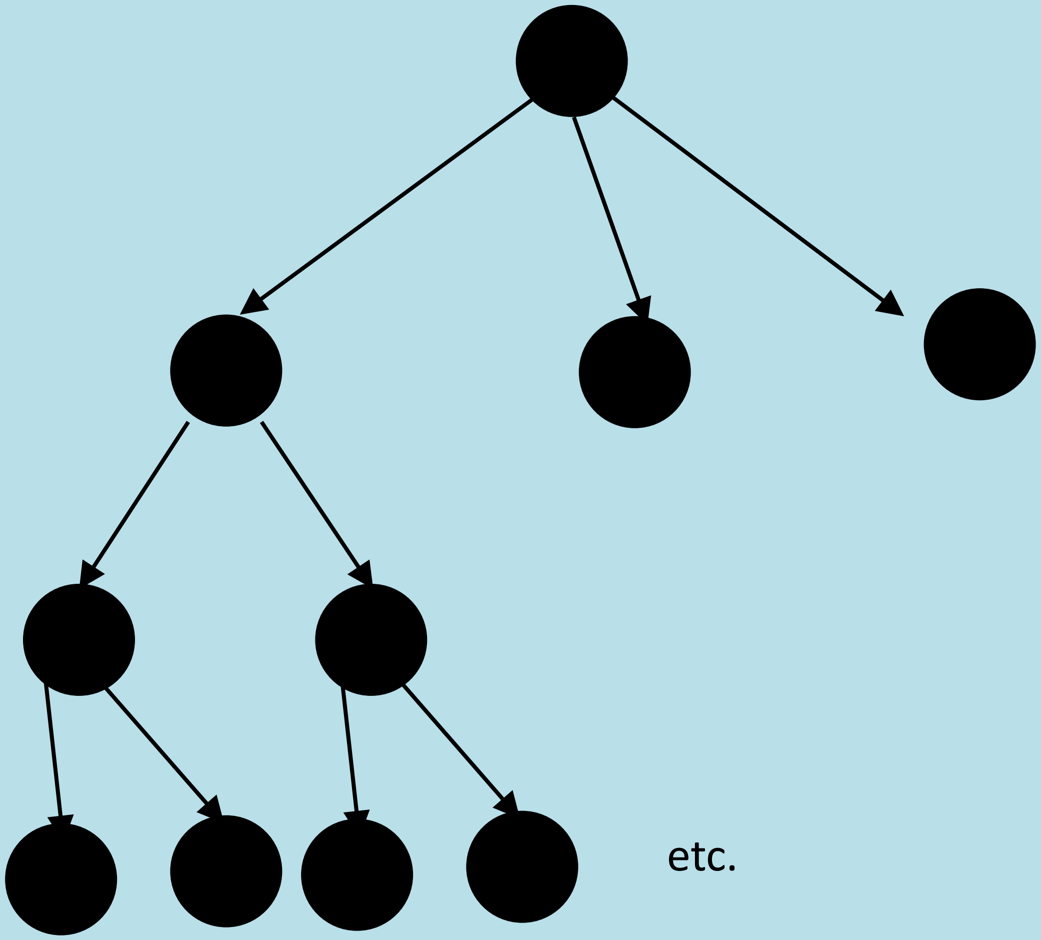
Adapted from Rusell, S., & Norvig, P. (2016);

2.2 Breadth-first Search



- Nodes are expanded in the same order in which they are generated
- Fringe can be maintained as a First-In-First-Out (FIFO) queue
- BFS is complete: if a solution exists, one will be found
- BFS finds a shallowest solution
- Not necessarily an optimal solution
- If every node has b successors (the branching factor), first solution is at depth d, then fringe size will be at least bd at some point
- This much space (and time) required !!!

2.2 Depth-first Search



- Always expand node at the deepest level of the tree, e.g. one of the most recently generated nodes
- When hit a dead-end, backtrack to last choice
- Fringe can be maintained as a Last-In-First-Out (LIFO) queue (aka. a stack)

2.2 Expansion: Combining Properties of BFS and DFS

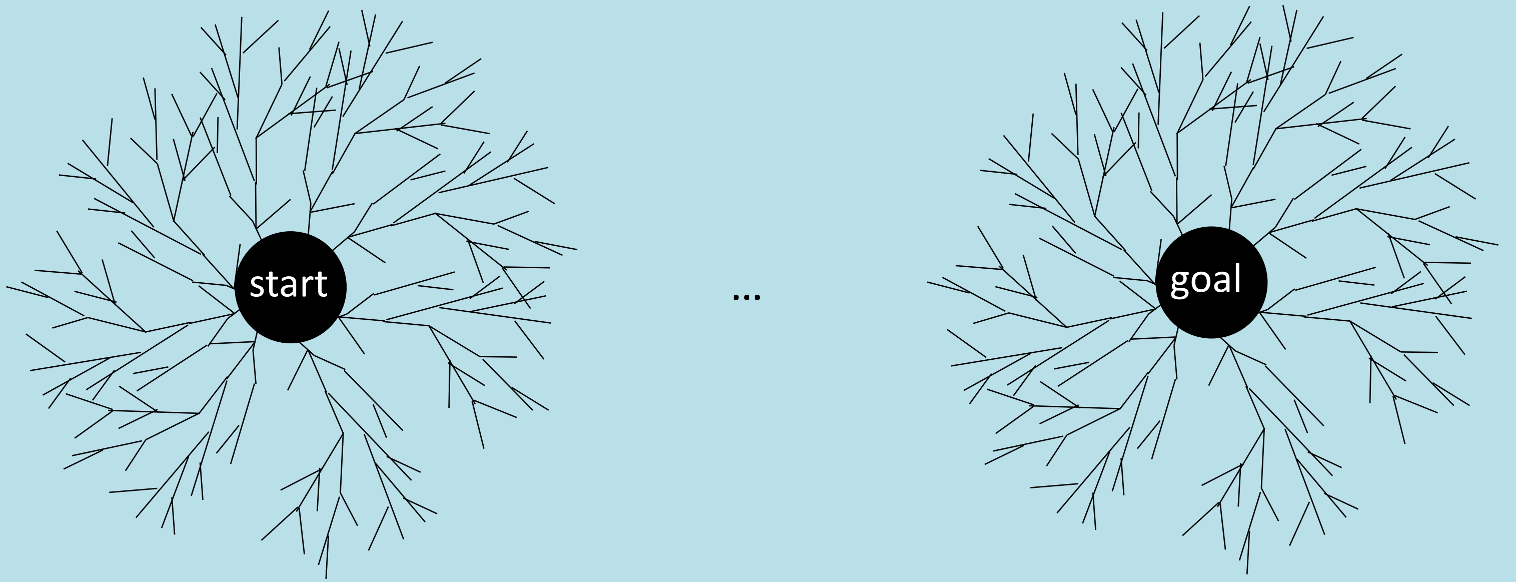
- Limited depth DFS: just like DFS, except never go deeper than some depth d
- Iterative deepening DFS:
  - Call limited depth DFS with depth 0;
  - If unsuccessful, call with depth 1;
  - If unsuccessful, call with depth 2;
  - etc.
- Complete, finds shallowest solution
- Space requirements of DFS
- May seem wasteful timewise because replicating effort
- Really not that wasteful because almost all effort at deepest level
- $db + (d-1)b^2 + (d-2)b^3 + \ldots + 1bd \text{ is } O(b^d) \text{ for } b > 1$

2.2 Let's Start Thinking About Cost

- **Path costs**: a function that assigns a cost to path, typically by summing the costs of the individual operators in the path. We want to minimize the cost.
- **Search costs**: The computational time and space (memory) required to find the solution
- There is a trade-off between path costs and search cost, in real-world-problems we can not build full search trees, we have to find best solution in the time available

2.2 Bidirectional Search

- Even better: search from both the start and the goal, in parallel!



If the shallowest solution has depth d and branching factor is b on both sides, requires only $O(b^{d/2})$ nodes to be explored!

Adapted from Rusell, S., & Norvig, P. (2016);

2.2 Informed Search

- So far, have assumed that no non-goal state looks better than another

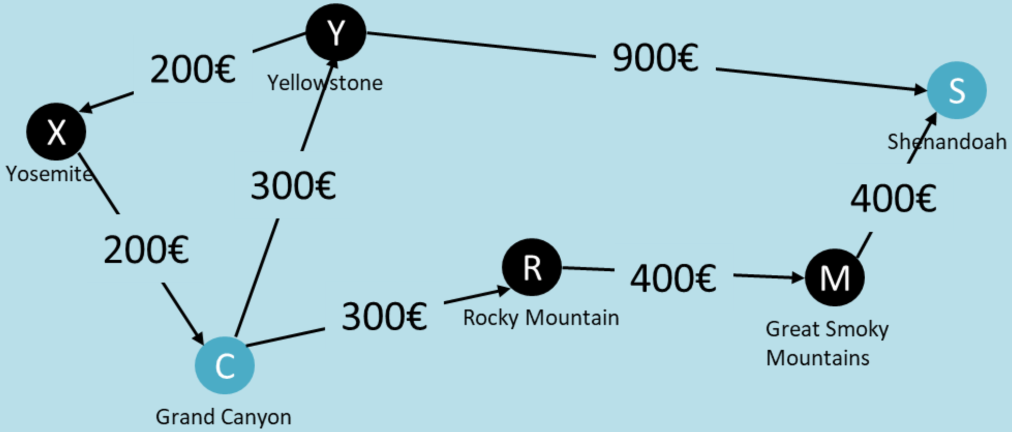
- Use knowledge to build search trees:
  - Even without knowing the road structure, some locations seem closer to the goal than others
  - Some states of a problem seem closer to the goal than others

- Makes sense to expand closer-seeming nodes first

Adapted from Rusell, S., & Norvig, P. (2016);

2.2 Idea: Use an Criterion to Identify which Node to Expand First

- Heuristic function h(n) gives an estimate of the distance from n to the goal (with h(n)=0 for goal nodes)
- E.g. straight-line distance for traveling problem (less costs for fuel etc.)



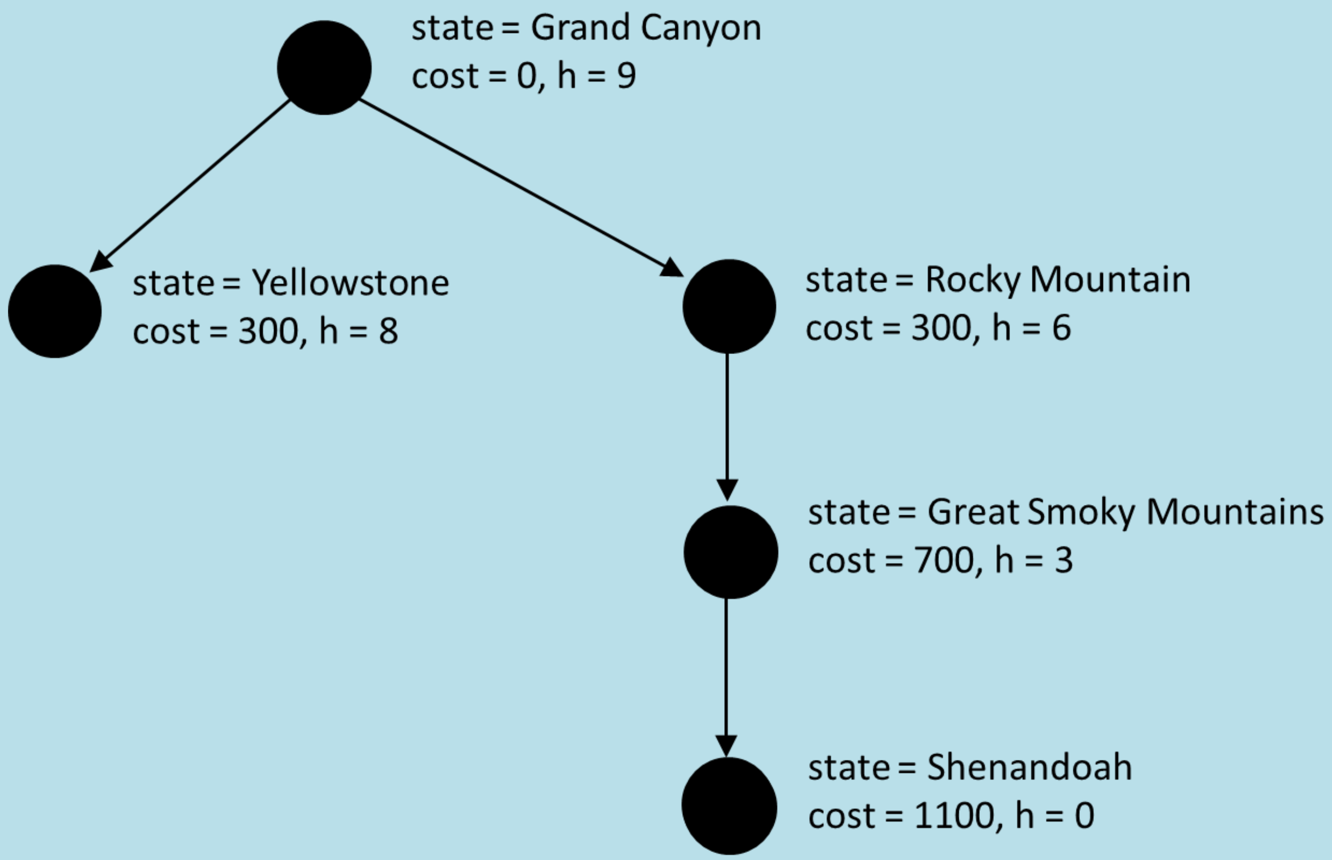
- We assume: h(C) = 9, h(Y) = 8, h(X) = 9, h(R) = 6, h(M) = 3, h(S) = 0
- Can use heuristic to decide which nodes to expand first

Adapted from Rusell, S., & Norvig, P. (2016) | Image sources: ↗ US NationalParks (2015) by Mwierschkec↗ CC BY-SA 4.0;

2.2 Greedy Best-First Search

- Greedy best-first search: expand nodes with lowest h values first

- Can find fast an optimal soultion



2.2 A* search: Minimizing the Total Estimated Solution Cost

- Evaluate nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

$$f(n) = g(n) + h(n)$$

- Since $g(n)$ gives the path cost from the start node to node n, and $h(n)$ is the estimated cost of the cheapest path from n to the goal, we have:

$$f(n) = estimated\ cost\ of\ the\ cheapest\ solution\ through\ n$$

2.2 A* Search and Admissibility

- A heuristic is admissible if it never overestimates the distance to the goal
- If n is the optimal solution reachable from n', then $g(n) \geq g(n') + h(n')$
- Straight-line distance is admissible: can't hope for anything better than a straight road to the goal
- Admissible heuristic means that A* is always optimistic

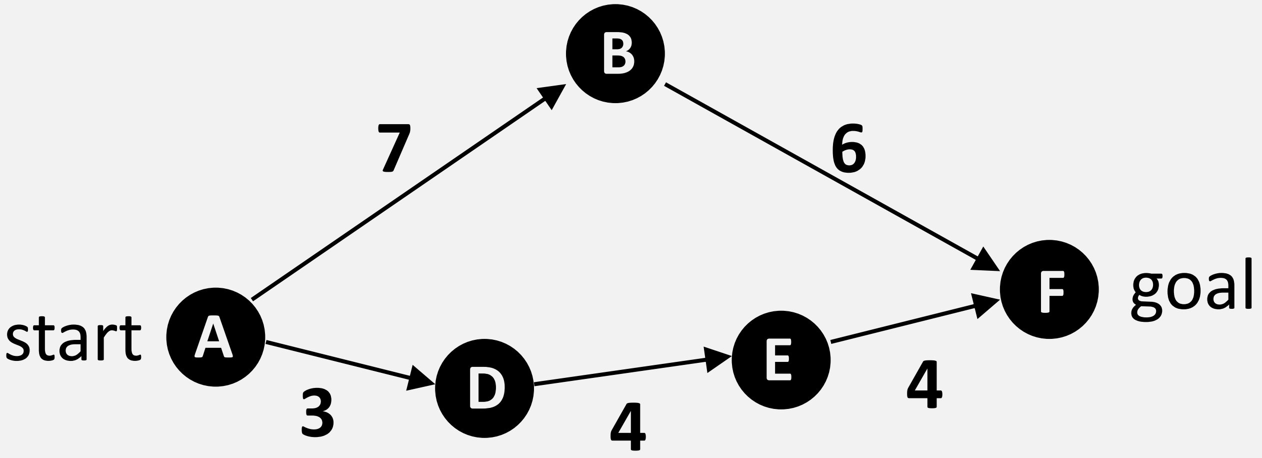
2.2 Optimality of A*

- If the heuristic is admissible, A* is optimal (in the sense that it will never return a suboptimal solution)
- Proof:
  - Suppose a suboptimal solution node n with solution value C > C* is about to be expanded (where C* is optimal)
  - Let n* be an optimal solution node (perhaps not yet discovered)
  - There must be some node n’ that is currently in the fringe and on the path to n*
  - We have $g(n) = C > C* = g(n*) \geq g(n') + h(n')$
  - But then, n’ should be expanded first (contradiction)

Your turn!

Task

Given the following route map with the following distance heuristics h(A) = 9, h(B) = 5, h(D) = 6, h(E) = 3, h(F) = 0. Try to solve this map with the greedy algorithm and discuss the results with your neighbors!



Outline

▶ **What we will learn:**
- We define the concept of rational agents (≈ intelligent agents)
- Characteristics of artificial agents (perfect or otherwise), the diversity of environments, and the resulting menagerie of agent types
- We discuss how AI problems can be modelled as search-problems, and how they can be solved by searching

Image source: ↗ Pixabay (2019) / ↗ CC0

▶ **Duration:**
- 225 min

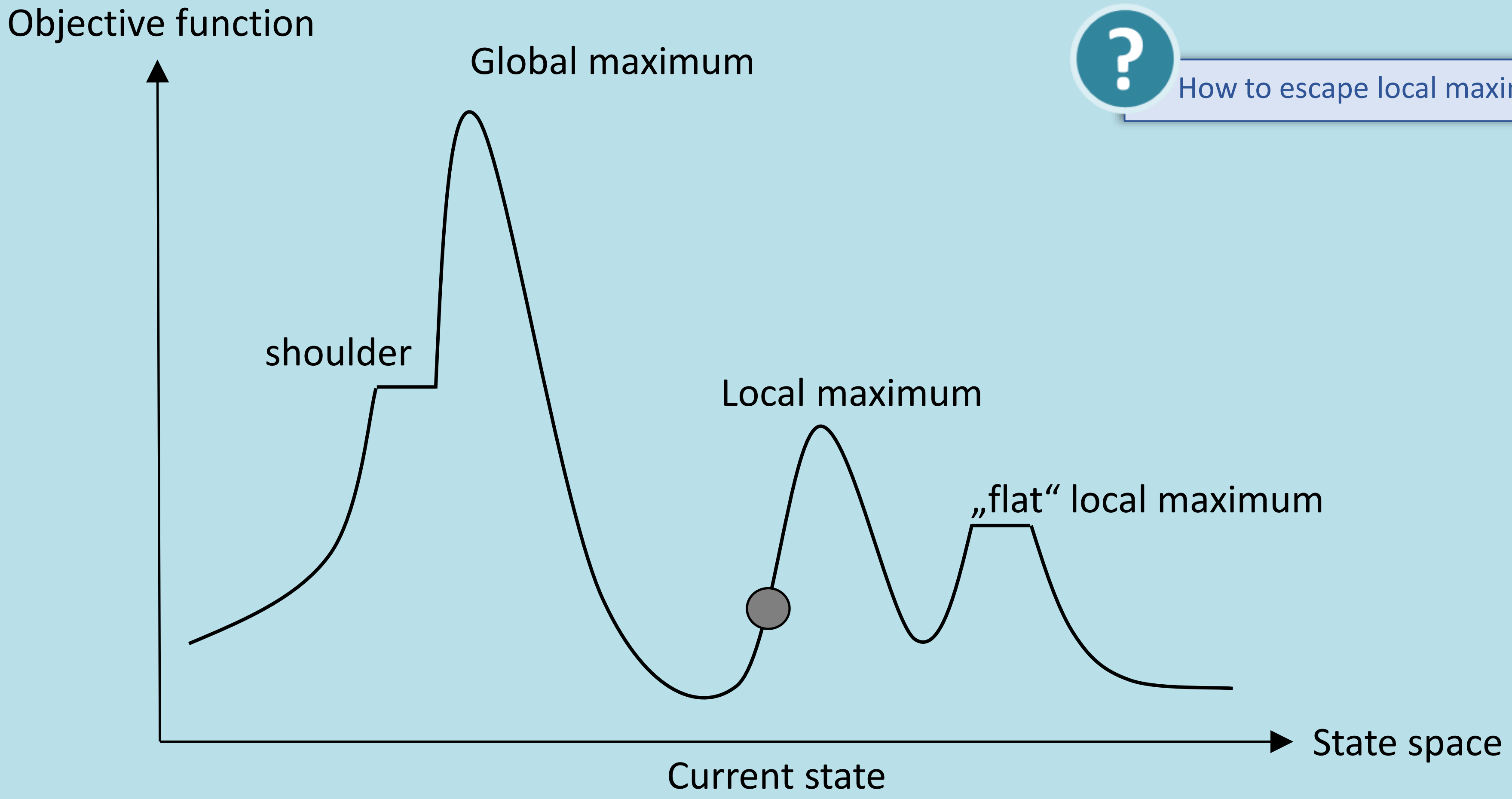
▶ **Relevant for Exam:**
- 2.1 – 2.5

- **Previous chapter**: Path to goal is solution to problem

- But sometimes…
  - The start state may not be specified
  - The path to the goal doesn’t matter

- In such cases, we can use local search algorithms that keep a single “current” state and gradually try to improve it

2.3 The State Space “Landscape”



How to escape local maxima?

Image adapted from Rusell, S., & Norvig, P. (2016);

2.3 How to Escape Local Maxima: Trivial Algorithms

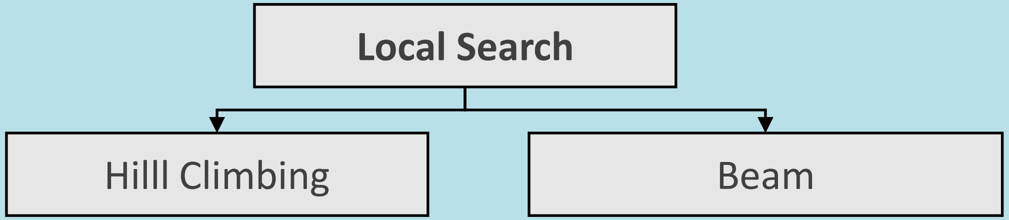
- Random Sampling
  - Generate a state randomly
- Random Walk
  - Randomly pick a neighbor of the current state
- Both algorithms asymptotically complete

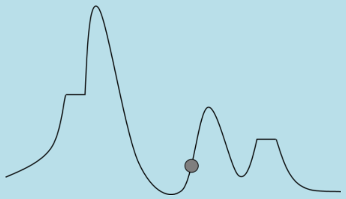
2.2 Overview: Fundamental Search Algorithms

D

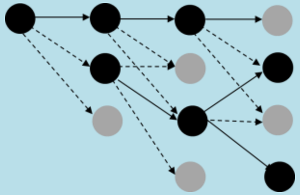
Local Search

Local search algorithms are such algorithms, they use a single current node (rather than multiple paths) and generally move only to neighbors of that node



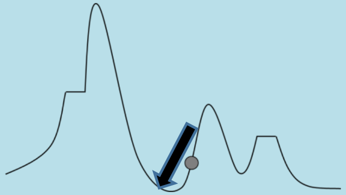


- Continually move in the direction of increasing value



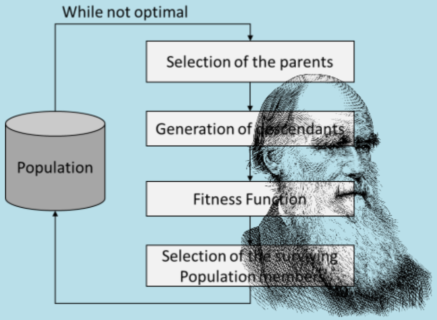
- expanding the most promising node in a limited set

Simulated Anhealing



- Escape local maxima by allowing some "bad" moves but gradually decrease their frequency

Genetic Algorithms



- Always expands the deepest node in the current frontier of the search tree.

Adapted from Rusell, S., & Norvig, P. (2016) | | Image source: ↗ Pixabay (2019) / ↗ CC0

2.3 Local Search Algorithms

- If the path to the goal does not matter → simplify algorithms and ignore paths
- **Local search** algorithms are such algorithms, they use a single **current node** (rather than multiple paths) and generally move only to neighbors of that node
- Typically, the paths followed by the search are not retained. Although local search algorithms are not systematic, they have two key advantages:
  - + they use very little memory - usually a constant amount
  - + they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable

Adapted from Rusell, S., & Norvig, P. (2016);

2.3 Hill-Climbing Search

Algorithm: Hill Climbing

current ← *MAKE-NODE(problem.INITIAL-STATE)*

While STOP != TRUE
  $neighbor \leftarrow a\ highest\text{−}valued\ succesor\ of\ current$
  If *neighbor.value* ≤ *current.VALUE*
    Then return *current. STATE*
  current ← *neighbor*

- **Idea**: simply a loop that continually moves in the direction of increasing value—that is, uphill
- The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.
- Hill climbing does not look ahead beyond the immediate neighbors of the current state

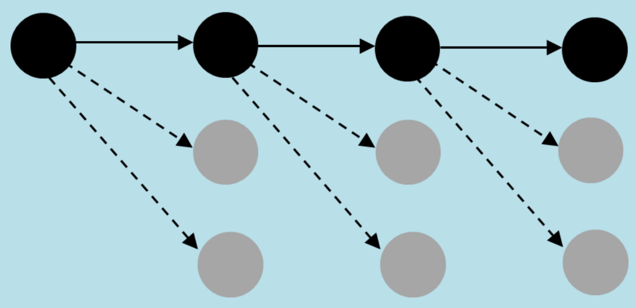
Adapted from Rusell, S., & Norvig, P. (2016)

2.3 Local Beam Search

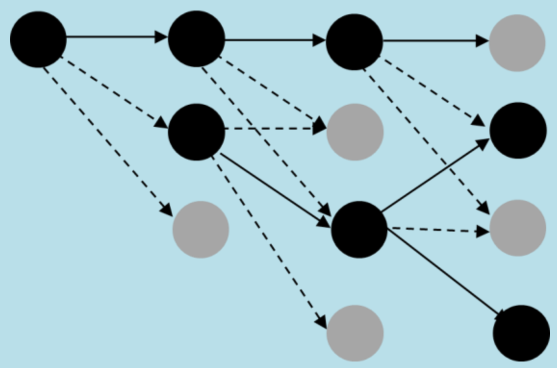
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat

Is this the same as running k greedy searches in parallel?

Greedy search



Beam search



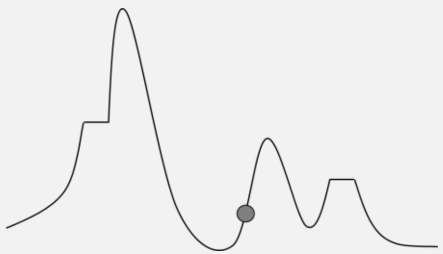
Adapted from Rusell, S., & Norvig, P. (2016);

2.3 Simulated Annealing Search

Algorithm: Simulated Annealing

Initialize current to starting state

For i = 1 to ∞
  If T(i) = 0 *return current*
  Let *next = random successor of current*
  Let *Δ = value(next) – value(current)*
  If *Δ > 0* ***then let*** *current = next*
  Else *let current = next with probability exp(Δ/T(i))*



Adapted from Rusell, S., & Norvig, P. (2016)

- **Idea**: Escape local maxima by allowing some "bad" moves but gradually decrease their frequency
- Probability of taking downhill move decreases with number of iterations, steepness of downhill move
- Controlled by annealing schedule
- Inspired by tempering of glass, metal

2.3 Conclusion: Simulated Annealing Search

- We can mathematically prove that a slow temperature decrease will find a global optimum with probability approaching one

- However:
  - This usually takes impractically long
  - The more downhill steps you need to escape a local optimum, the less likely you are to make all of them in a row

- State-of-the-Art: General family of Markov Chain Monte Carlo (MCMC) algorithms for exploring complicated state spaces

Adapted from Rusell, S., & Norvig, P. (2016);

2.3 Genetic/Evolutionary Algorithms

- Biological evolutionary model according to Darwin:
  Selection = driving force of evolution
- Transfer to Computer Science: Evolution as optimization of complex, artificial systems
- Build machines that adapt to an defined working environment

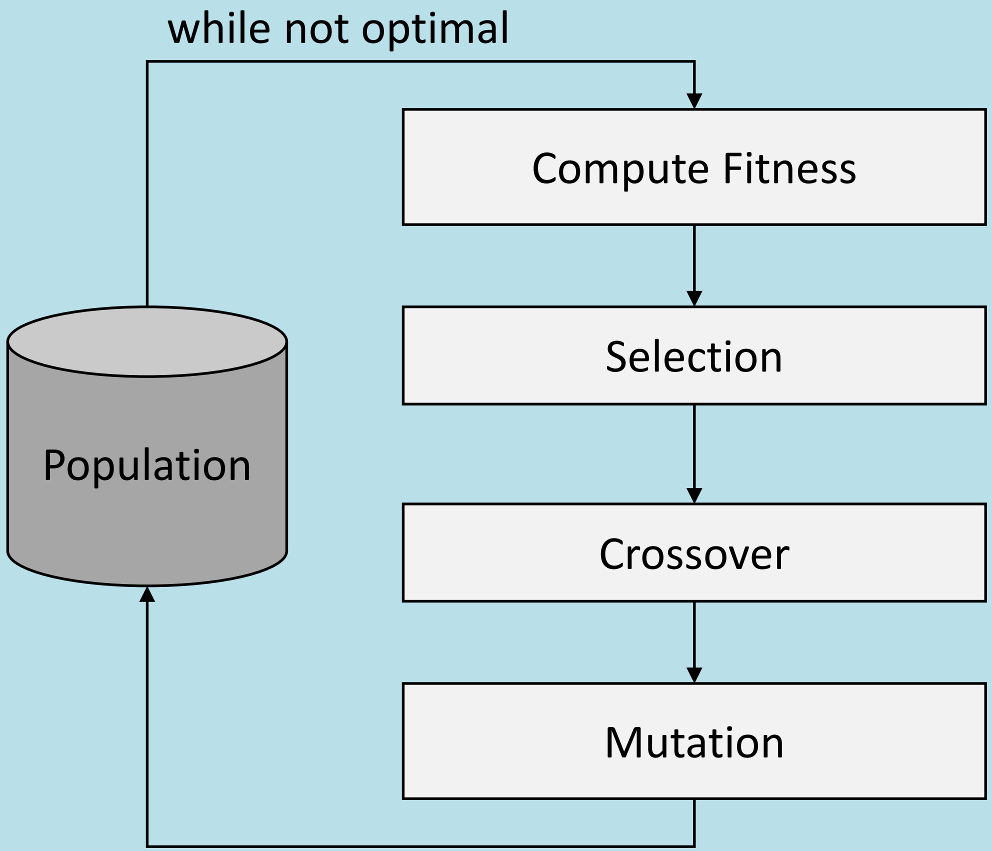
Adapted from Rusell, S., & Norvig, P. (2016) | | Image source: ↗ Pixabay (2019) / ↗ CC0

2.3 Wording in Evolutionary Algorithms

Individual	Possible solution, hypothesis, or configuration
Population and generation	Set of solutions or hypothesis
Generation of descendants	Generation of new hypotheses. Methods: recombination (Cross-over) and mutation
Changed successor, child offspring	New hypothesis or configuration
Fitness function	Hypothesis quality, criterion to be optimized
Selection of the best	Selection of the hypotheses that the create the best problem solution

Adapted from Rusell, S., & Norvig, P. (2016);

2.3 Basic Algorithm



Algorithm: Genetic Algorithm

Inputs: population, fitness_func

new_population <- empty set
Repeat
    For i = 1 to** SIZE(population) **do
        X ← RANDOM-SELECTION(population, fitness_func)
        Y ← RANDOM-SELECTION(population, fitness_func)
        Child ← REPRODUCE(x,y)
        ***If(random prob) then** child ← MUTATE(child)*
            Add child to new_population
    Population ← new_population
***Until** some individuals optimal or time elapse*

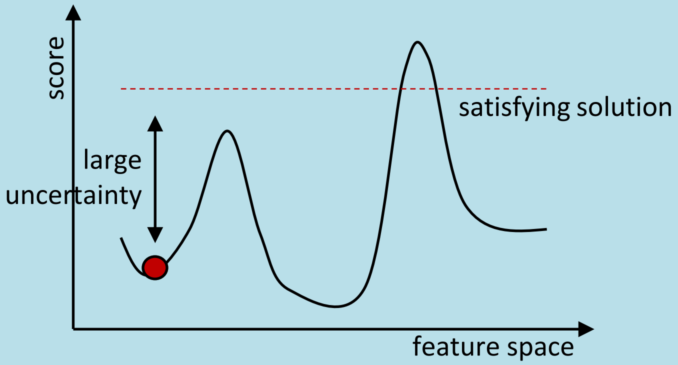
***Return** the best individuals in pop, according fitness_func*

*Reproduce returns an new individual

Adapted from Rusell, S., & Norvig, P. (2016);

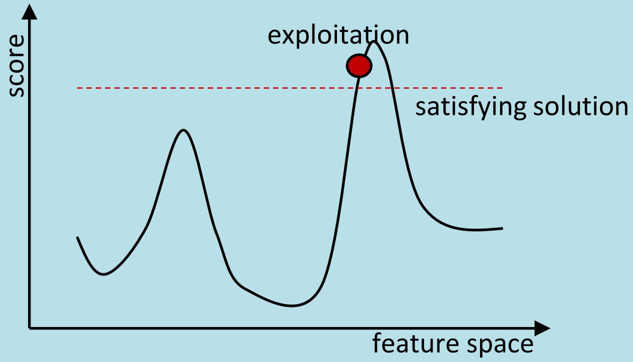
2.3 Generation of descendants

Exploration



- Exploring the local space, local optimization

Exploitation



- Exploitation of the hypothesis space

- The stronger and more random changes are, the lower the Probability to produce better offsprings
- With local improvement methods, the risk of local Minima given
- level of exploration must be in accordance with the current fitness of the generation can be selected (e.g.: initially high then falling)

Adapted from Rusell, S., & Norvig, P. (2016);

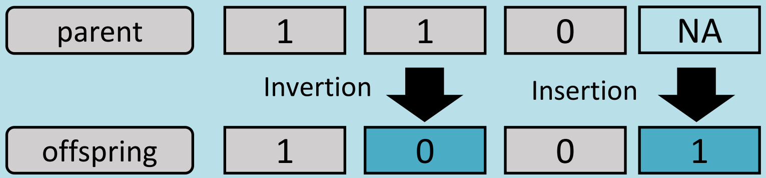
2.3 Mutation

Mutation

The offspring is descended from one parent, with mutation of single gene

<u>Mutation</u>:

- All bits of a sequence are independently inverted with a certain probability
- For a certain (or random) number of bits the indices are selected randomly
- Remove a partial sequence and insert it at another Place
- Inverted insertion of the partial sequence

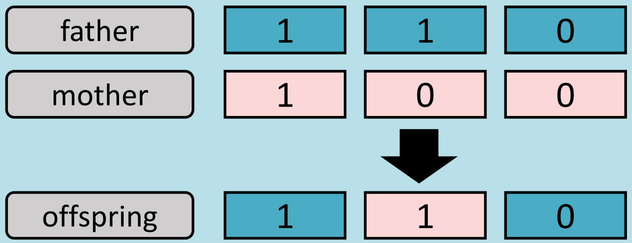


Crossover / Recombination

Mix properties of two or more parents

<u>Crossover</u>:

- Discrete recombination



- Intermediate recombination (continuous representation)

$$parent_1 := x \text{ and } parent_2 := y,$$
$$offspring_i := (x_i + y_i)\,/\,2$$

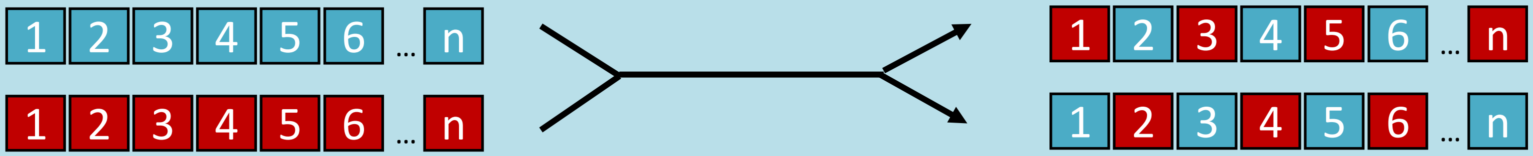
Adapted from Rusell, S., & Norvig, P. (2016);

2.3 Crossover/Recombination

Single-point Crossover

Two-point Crossover

Uniform Crossover



2.3 Selection

Two different types of selection:

- of the parents for respective production of offspring (Mating)
- of the population in each iteration

Problems:

- Genetic drift: Individuals reproduce more than others by chance
- Crowding, outlier problem: "fit" individuals and similar offspring dominate the Population

→Development of individuals is slowed down

→Diversity of the population is restricted

Solution:

- Different population models and selection methods
- Optimize population size

2.3 Population Models

- Island model (local)
  The evolution runs largely separately, only sometimes individuals exchanged

- Neighborhood model (near surroundings)
  Descendants may only be produced by individuals who are have the best fitness in their neighborhood

- A simple set (global)
  The best in the world are developing rapidly, others Lines of development are suppressed

2.3 Population Members

Population size:

- Should it remain constant? (μ)
- How many newly produced offspring? (λ)
- How many parents should be used? (ρ)
- How are they determined?

Member selection:

stochastically selected ⇒ the best μ individuals

- (μ, λ) Strategy: Selection refers only to the Offspring (better exploration)
- ($\mu + \lambda$) Strategy:

  Selection also involves parents (the Best are considered, search for Elites → Exploitation, cheap with good calculable fitness functions)

2.3 Substitute Population Members

Substitution rule for members:

- Descendants replace all parents (Generation mode)
- Offspring replaced a part of the parents
- Offspring replaced parents that are most similar to them
- Geographical Replacement
- Best individual survives (Elitist - Mode)

Rule of thumb:

The best quarter of the population should be three quarters of the descendants

2.3 Selection Methods - Fitness Based Selection

Fitness Based Selection: $P(X) \approx \frac{f(x)}{\sum_{x' \in Pop.} f(x')}$ exactly $P(X) = \frac{\lambda}{\mu} \cdot \frac{f(x)}{\sum_{x' \in Pop} f(x')}$

P(x) : Probability of selection of individual x
λ : Number of descendants
μ : Population size
f : Fitness – Function

- depending on the value of the fitness function
- e.g. during Evolution only minor changes in $f(x)$ and thus in *P(x)*

2.3 Selection Methods - Ranking Based Selection

Ranking Based Selection : $P(x) \approx \frac{g(r(x))}{\sum_{x' \epsilon Pop.} g(r(x'))}$ with

$P(x)$: Probability of selection of individual x
$\mathrm{r}(x)$: Ranking of x in the current population according to fitness -Function
g : function increasing monotonically with the quality of the rank greater than 0

- Exponential: $\boxed{g(x) = a^{-x}}$
- Hyperbolic: $\boxed{g(x) = x^{-a}}$
- the best k: $\mathrm{g}(x) = \begin{cases} {}^{1}/_{k}, & x \leq 0 \\ 0, & else \end{cases}$

- less dependent on the amount of fitness
- better adaptation of exploration / exploitation

2.3 Selection Method – Tournament Selection

Tournament Selection (tournament)

- select n (=2) individuals for each individual to be created
- reward (increase rating) of it, according to the fitness the best individual
- select individuals with highest rating

- little dependent on the amount of fitness

Choosing the selection method

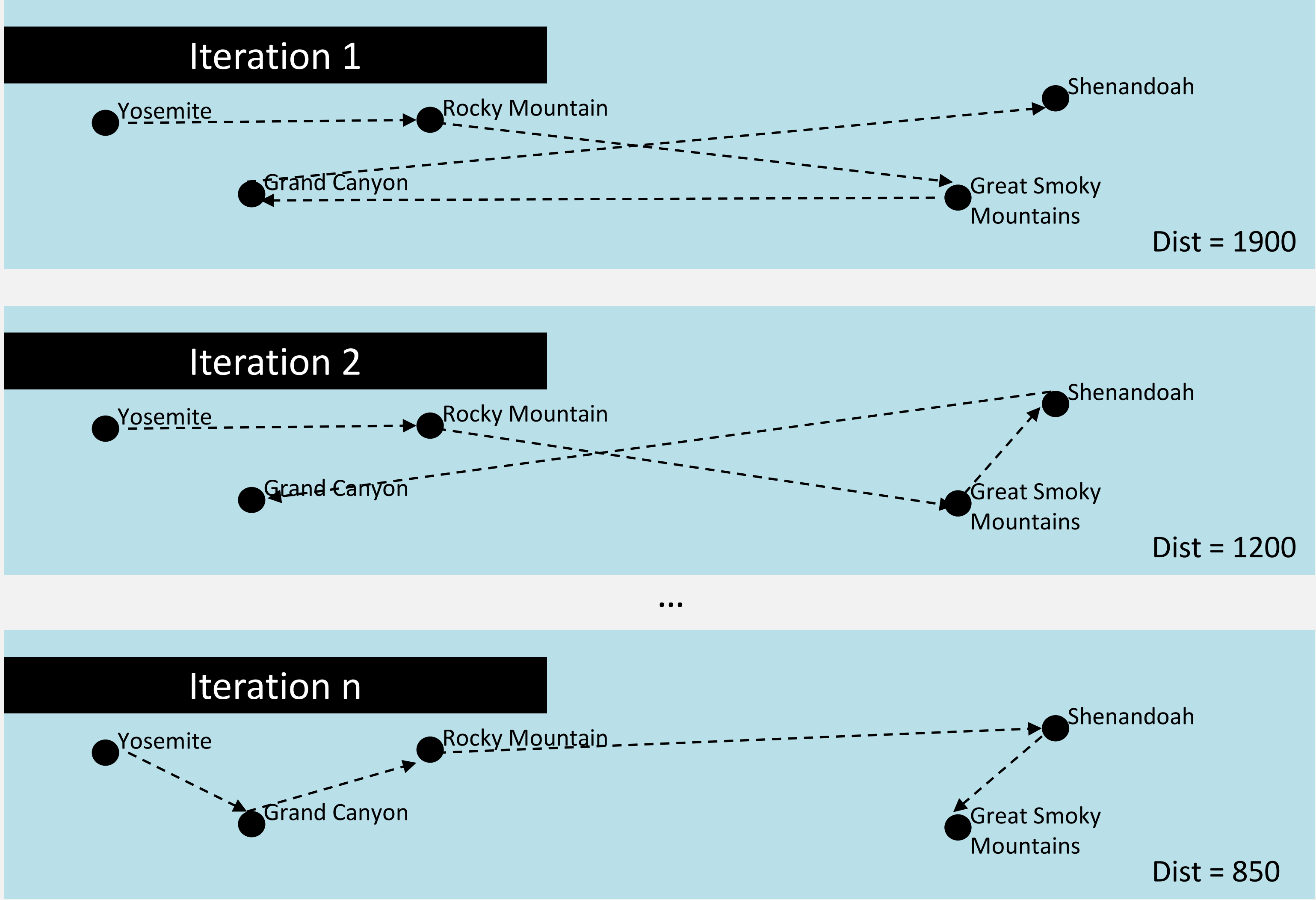
- often application-specific

2.3 Use Genetic Algorithms to Solve our Travelling Salesman Problem

Find a path:

- start in Yosemite
- each national park is visited exactly once
- the travelled distance is minimal

2.3 Use Genetic Algorithms to Solve our Travelling Salesman Problem I



Find a path:

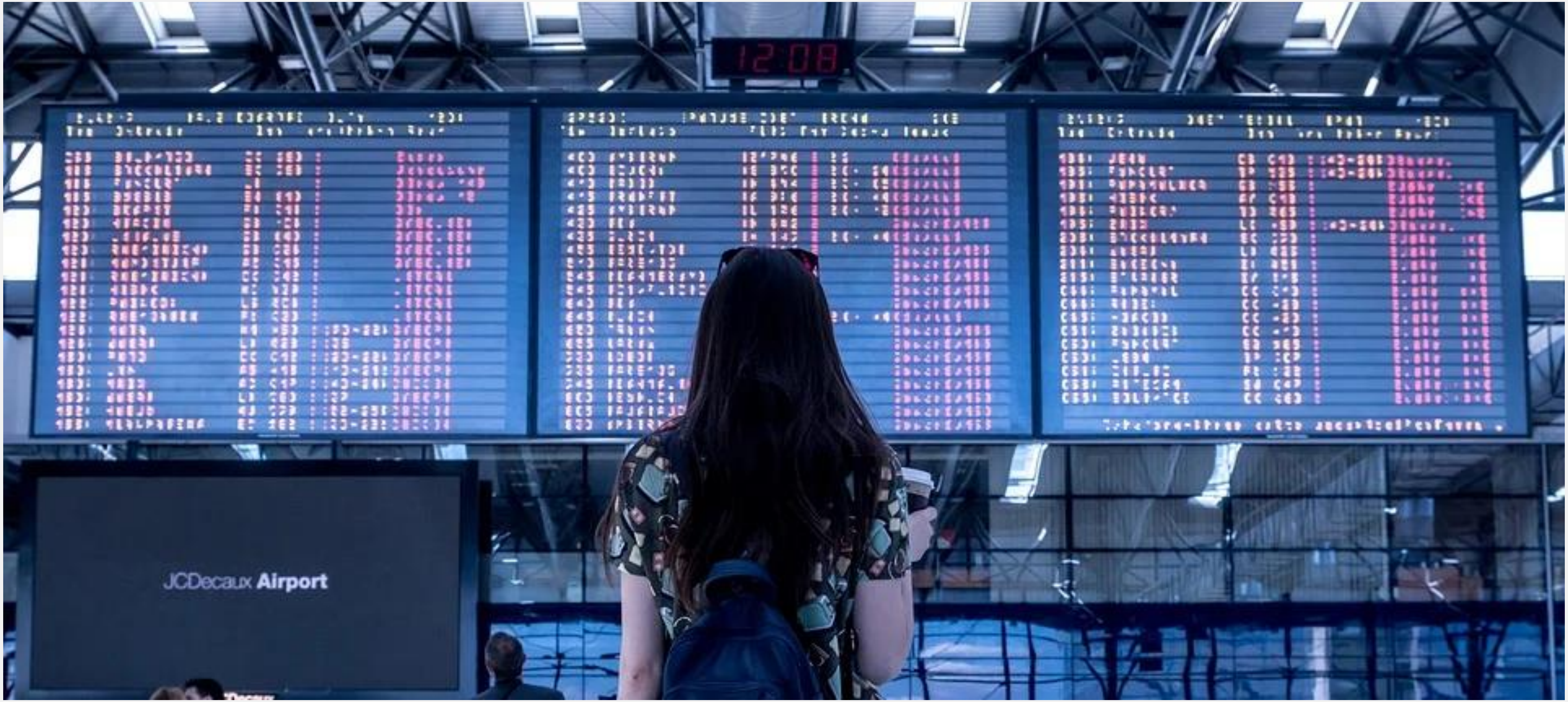
- each national park is visited exactly once
- the travelled distance is minimal

[YOS ROC GRE GRA SHE]

↓ Inversion

[YOS ROC GRE SHE GRA]

2.3 Application of Genetic Algorithms



Your turn!

Task

Please explain

- What is the difference between the steps mutation and crossover in the context of genetic algorithms?

Exercises

Workbook Exercises

- Please read the chapters 2-4 from Rusell, S. & Norvig, P. (2016). Work through the exercises of the related chapters. Start with the exercises related to algorithms we discussed during lecture.

Coding Exercises

- *Coding exercises start after lecture 3*

References

Literature

1. Rusell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach.* Global Edition.

News articles

1. NTV (2019): Sneaker-Bots ausgetrickst - Skate-Shop verkauft Fotos statt Schuhe. Online available at: https://www.n-tv.de/mediathek/videos/wirtschaft/Skate-Shop-verkauft-Fotos-statt-Schuhe-article21229466.html

Images

All images that were not marked other ways are made by myself, or licensed ↗CC0 from ↗Pixabay.

Further reading

- If you are also interested in visiting all US national parks, I can recommend to take a look at the blog article from *University of Penn data scientist Dr. Randal Olson* (↗http://www.randalolson.com). He created a roadtrip of the optimal way to visit all 47* U.S. National Parks in the mainland United States. Take a look at his Python projects (↗Github).

**before Gateway Arch in 2018 became a national park*

Glossary

Agent	*An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators (Rusell & Norvig, 2016)*
Agent program	*Implements the agents function (Rusell & Norvig, 2016, p.59)*
Performance measure	*Evaluates the behavior of the agent in an environment. A rational agent acts so as to maximize the expected value of the performance measure, given the percept sequence it has seen so far (Rusell & Norvig, 2016, p.59)*
Rationality/Rational agent	*For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has (Rusell & Norvig, 2016, p.37)*
Path costs	A function that assigns a cost to path, typically by summing the costs of the individual operators in the path
Search costs	The computational time and space (memory) required to find the solution
Task environment	*External environment of an agent including the performance measure, the external environment, the actuators, and the sensors (Rusell & Norvig, 2016, p.59)*