

Artificial Intelligence

Algorithms and Applications with Python

Chapter 2



Dr. Dominik Jung
dominik.jung@jung-isec.de



python

Business Case: Stop Automated Nike Shopping Agents

Sneaker bot

SEARCH

FILTER

HOW TO BOT HOW TO BOT Nike SNKRS Sneakers App - BNB Better Nike Bot Setup Guide Tutorial 2019
BOTTER BOY NOVA • 254.252 Aufrufe • vor 5 Monaten

HOW TO BOT Nike SNKRS Sneakers App BNB Better Nike Bot Setup Guide Tutorial 2019 Nike Sneakers or SNKRS is a platform ...

THE UNTOLD TRUTH: SNEAKER BOTTING Dangers of Sneaker Botting - MUST WATCH BEFORE YOU BOT! The Untold Truth 2019
BOTTER BOY NOVA • 289.811 Aufrufe • vor 6 Monaten

Upon Launching my web series "Sneakers To Riches" I seemed to have inspired many newcomers to get into sneaker botting.

BEGINNERS GUIDE The ULTIMATE Beginner's Guide to Sneaker Botting in 2019
Eman - Sneaker Bot Tutorials • 39.673 Aufrufe • vor 2 Monaten

Ever wanted to learn how to begin sneaker reselling and using sneaker bots? Watch this video to find out how to begin making ...

AIO V2 TUTORIAL AIO V2 Tutorial - A Beginner AIO Sneaker Bot Guide
Eman - Sneaker Bot Tutorials • 4568 Aufrufe • vor 3 Wochen

This video teaches you how to bot Footsites, Adidas and Shopify sites with AIO V2. 10% OFF CODE: unlimited10 Purchase AIO V2 ...

BEST BEGINNER A BEST BEGINNER A

BILLING PROFILE Sneaker Bot Tutorial: Billing Profiles & VCC
Eman - Sneaker Bot Tutorials • 3787 Aufrufe • vor 3 Wochen

If you want to know how to setup virtual credit cards (VCCs) and make your billing profiles, this video should help.

Privacy Revolut

VCC TUTORIAL

ntv Samstag, 05. Oktober 2019 13:42 Uhr Frankfurt | 12:42 Uhr London | 07:42 Uhr New York | 20:42 Uhr Tokio

RESSORTS | SPORT | BÖRSE | WETTER | TV | VIDEO | AUDIO | DAS BESTE | Q | LIVE

Startseite >> Mediathek >> Videos >> Wirtschaft >> Sneaker-Bots ausgetrickst : Skate-Shop verkauft Fotos statt Schuhe

TOPVIDEOS | POLITIK | WIRTSCHAFT | BÖRSE | SPORT | PANORAMA | UNTERHALTUNG | TECHNIK | RATGEBER | WISSEN | AUTO

WIRTSCHAFT



26.08.2019 10:43 Uhr – 01:27 min

Sneaker-Bots ausgetrickst
Skate-Shop verkauft Fotos statt Schuhe

Limitierte Sneaker lassen sich gewinnbringend weiterverkaufen. Professionelle Reseller verwenden Bots, die automatisiert Einkäufe tätigen können. Normale Kunden gehen oft leer aus. Ein Sneaker-Shop in Frankfurt hat die Software nun ausgetrickst. Mit einer simplen aber genialen Idee.

Facebook Twitter Email

PLAY

Business Case: Stop Automated Nike Shopping Agents

01 | Executive Summary

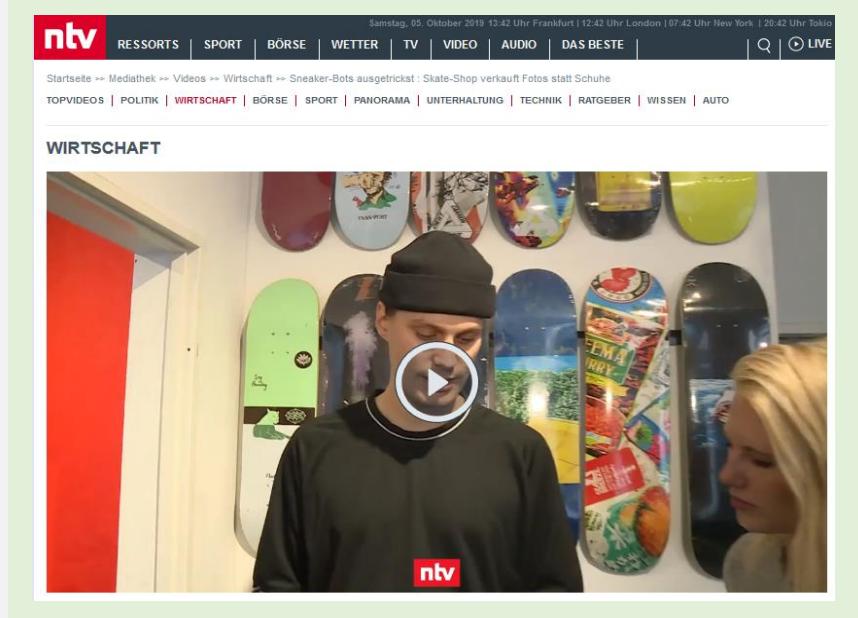
Limited sneakers can be resold profitably. Professional resellers use AI-based agents that can make automated purchases. Today, there are many AI-based agents and web-crawler that are able to detect and buy limited sneakers or other limited offers in web-shops, while normal customers often go away empty-handed. 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

Take-Aways

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



03 | References

- <https://www.n-tv.de/mediathek/videos/wirtschaft/Skate-Shop-verkauft-Fotosstatt-Schuhe-article21229466.html>

Outline

2 Search, Problem Solving, and Planning

2.1 Intelligent Agents

2.2 Solving Problems by Searching

2.3 Beyond Classical Search

2.4 Adversarial Search and Game Theory

2.5 Constraint Satisfaction Problems

► What we will learn:

- We define the concept of rational agents (\approx 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

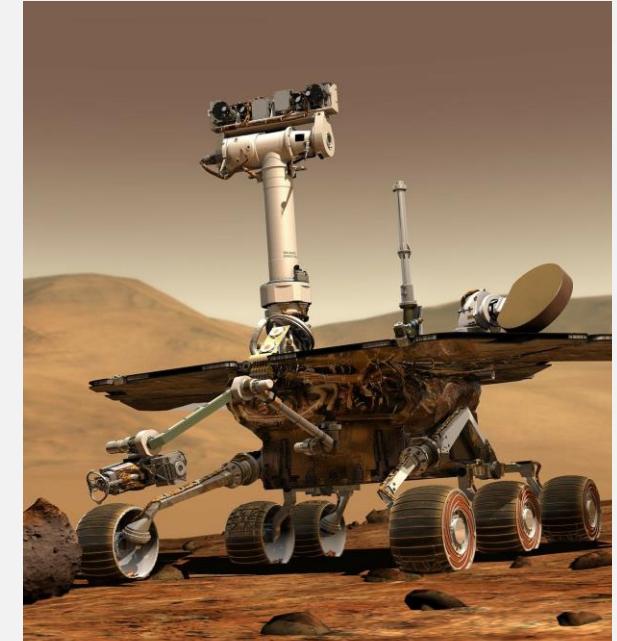


Image source: ↗ [Pixabay](#) (2019) / ↗ [CC0](#)

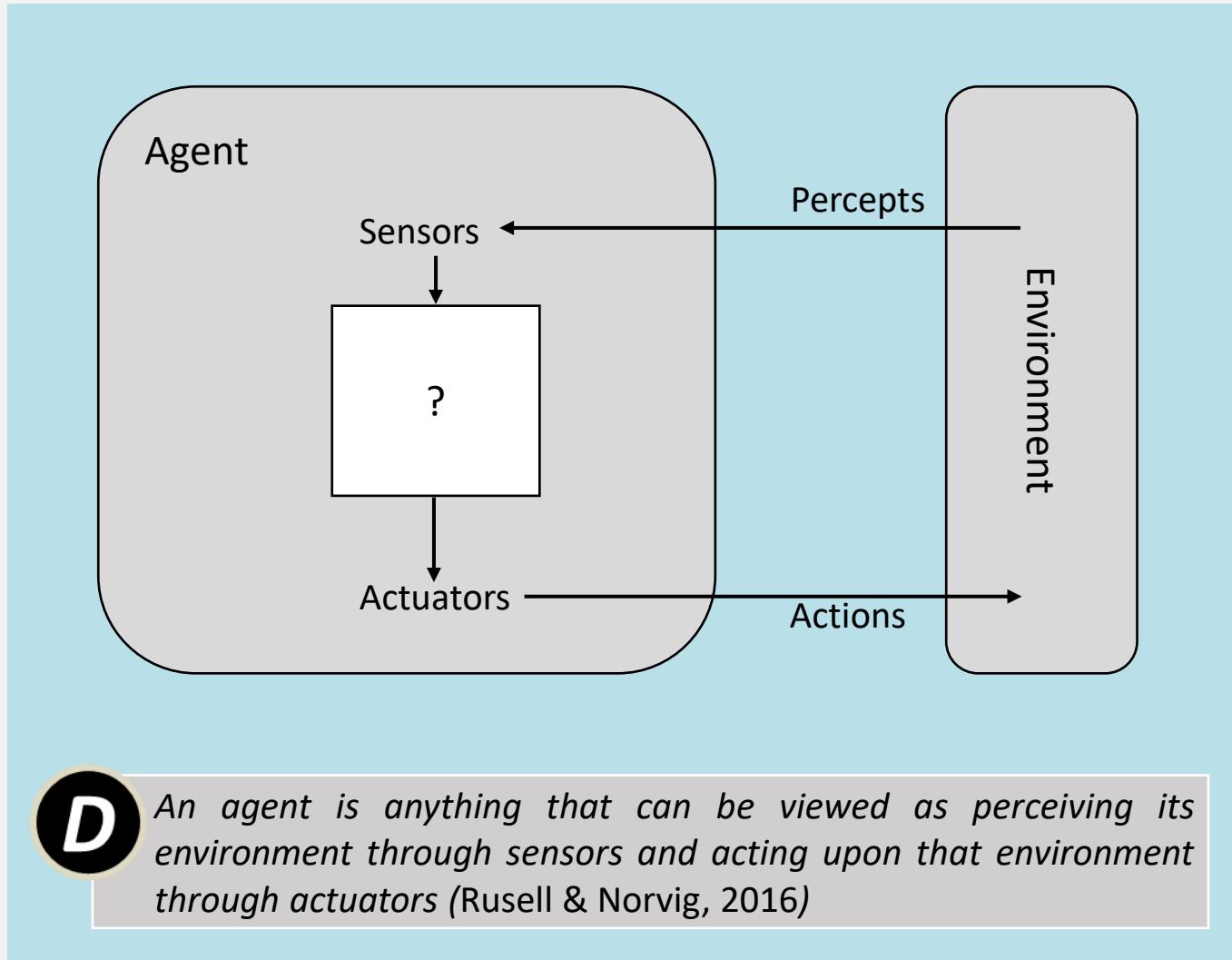
► Duration:

- 180 min

► Relevant for Exam:

- 2.1 – 2.5

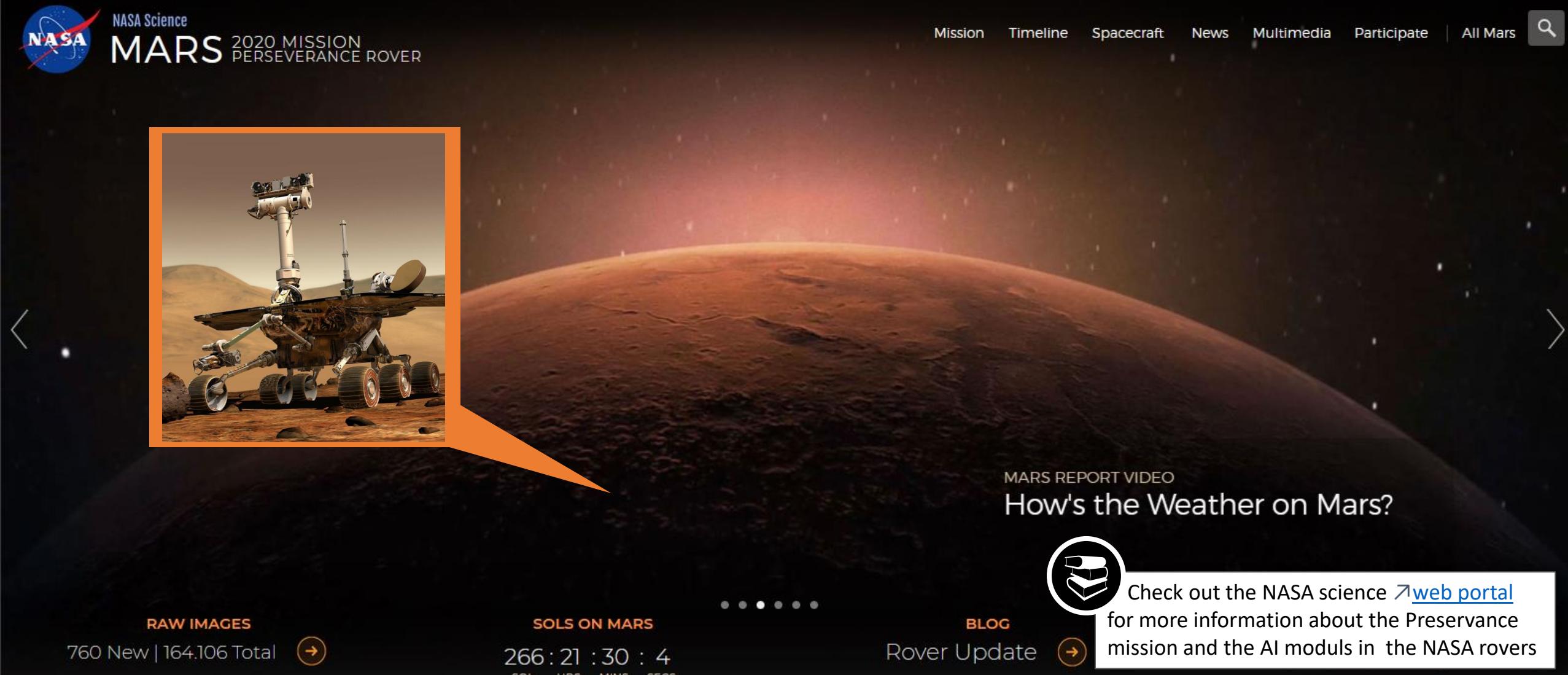
2.1 „Intelligent“ Agents (Russel, Norvig)



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



2.1 NASA Perseverance Rover



The image shows a wide-angle view of the surface of Mars, featuring a prominent reddish-orange hue. In the lower-left foreground, there is a small inset image of the Perseverance rover, which is dark-colored with various scientific instruments and wheels attached. An orange arrow points from this inset towards the main, larger image of the Martian landscape.

NASA Science MARS 2020 MISSION PERSEVERANCE ROVER

Mission Timeline Spacecraft News Multimedia Participate | All Mars

RAW IMAGES 760 New | 164,106 Total →

SOLS ON MARS
266 : 21 : 30 : 4
SOL HRS MINS SECS

... • • • •

BLOG Rover Update →

MARS REPORT VIDEO
How's the Weather on Mars?

Check out the NASA science [web portal](#) 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
- Main focus in this lecture: **Software agents** that gathers information about an environment and takes actions based on that

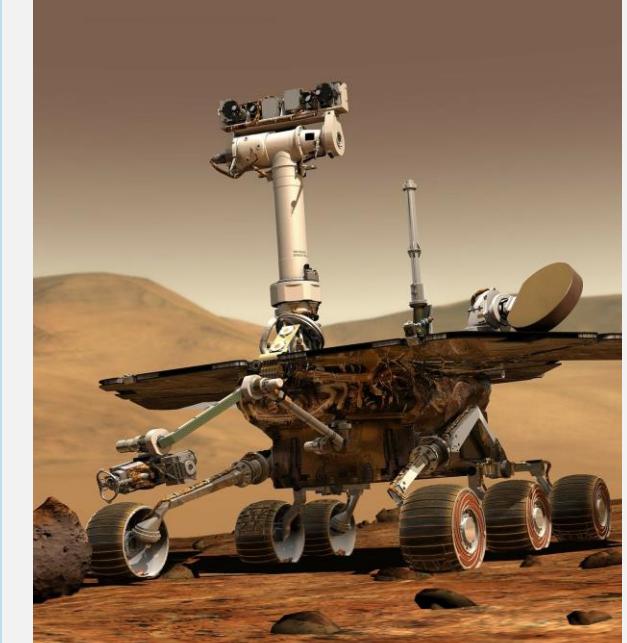
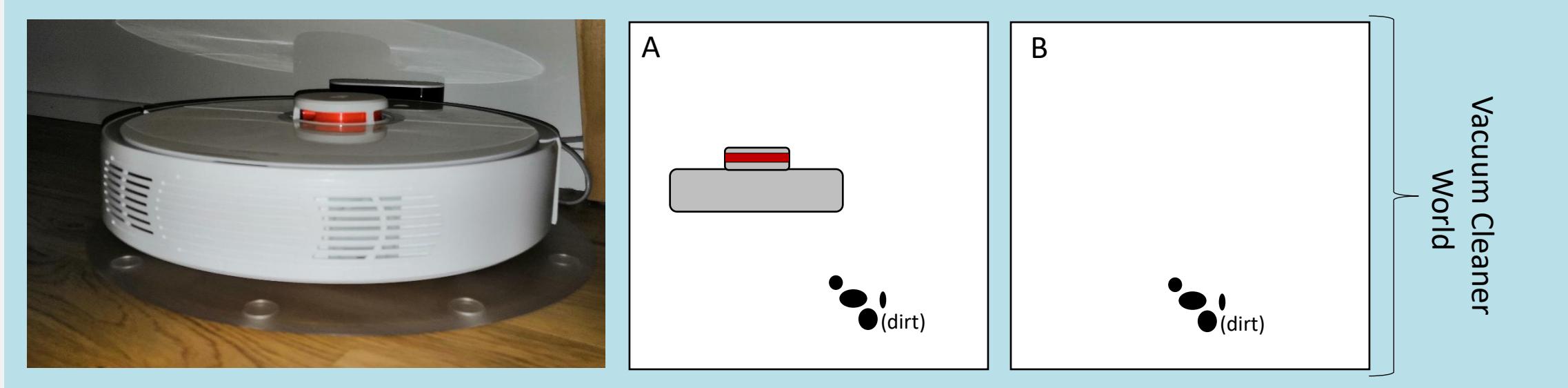


Image source: ↗ [Pixabay](#) (2019) / ↗ [CC0](#)

How do you design an intelligent agent?

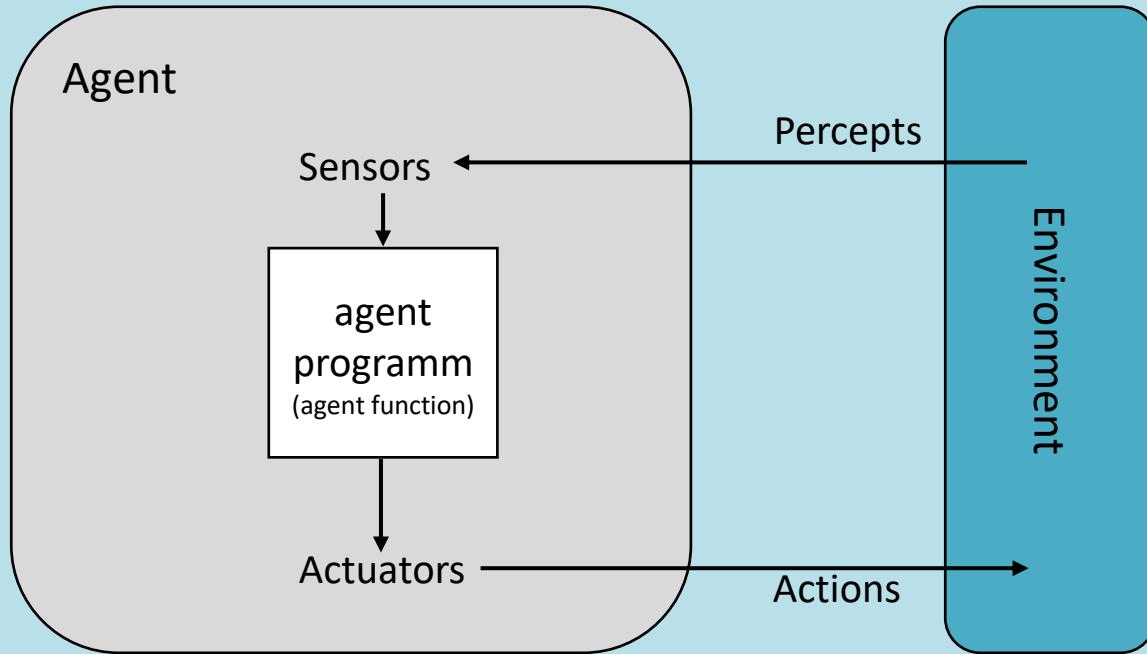
2.1 Example: Vacuum Cleaner World - Tabulation



- 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 Russell, 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 Russell, 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 Russell, 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 Russell, S., & Norvig, P. (2016)

The simplest environment is

- Fully observable
- Deterministic
- Episodic
- Static
- Discrete
- Single-agent

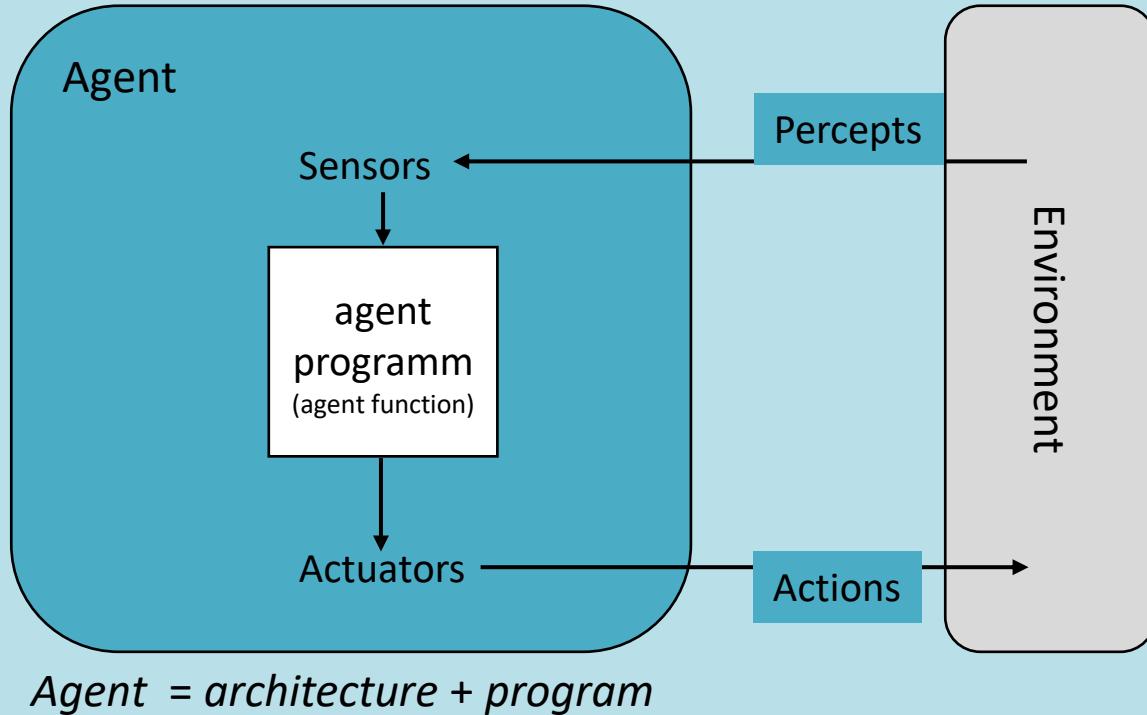
Most real situations are

- Partially observable
- Stochastic
- Sequential
- Dynamic
- Continuous
- Multi-agent

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

2.1 Step 2 – Specifying the Agent



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

Adapted from Russell, 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



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

- 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

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



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 (Russell & Norvig, 2016, p.37)

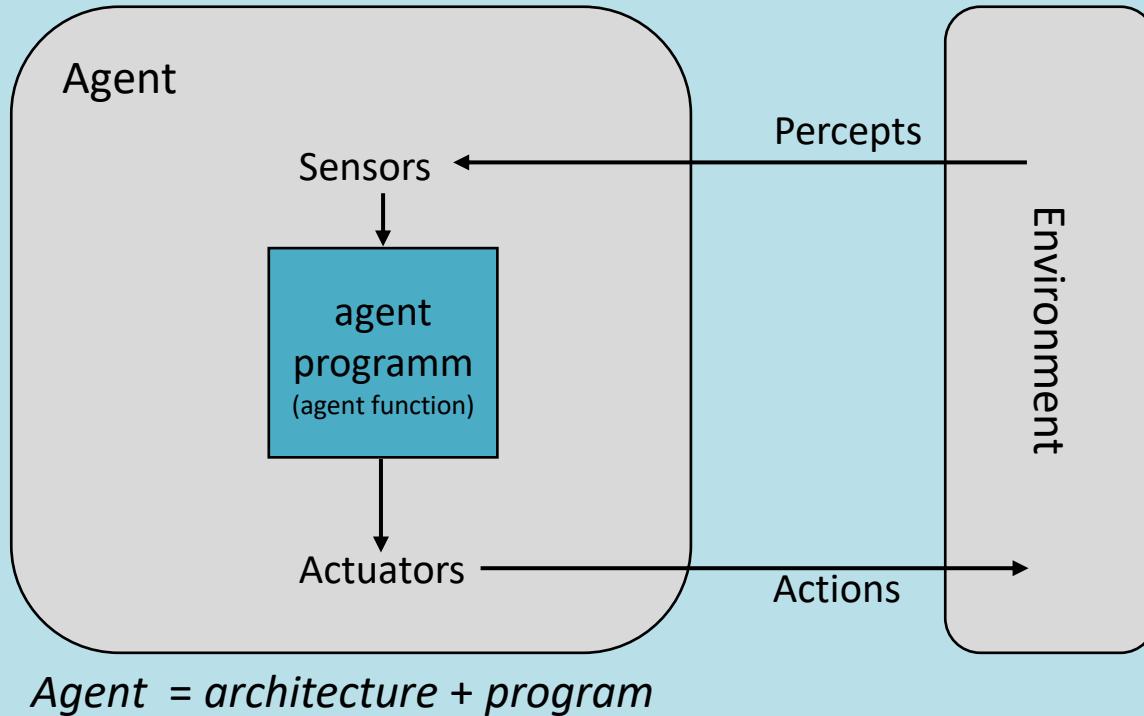
Adapted from Russell, S., & Norvig, P. (2016)

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 & Russells' 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 Russell, S., & Norvig, P. (2016)

2.1 Step 3 – Specifying the Agents Behaviour

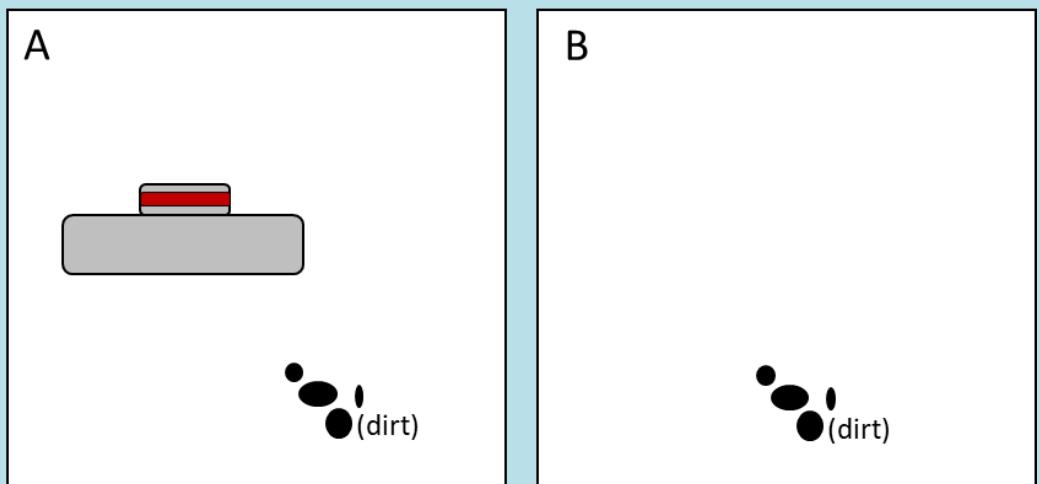


- How can we implement the „intelligent“ behaviour?

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

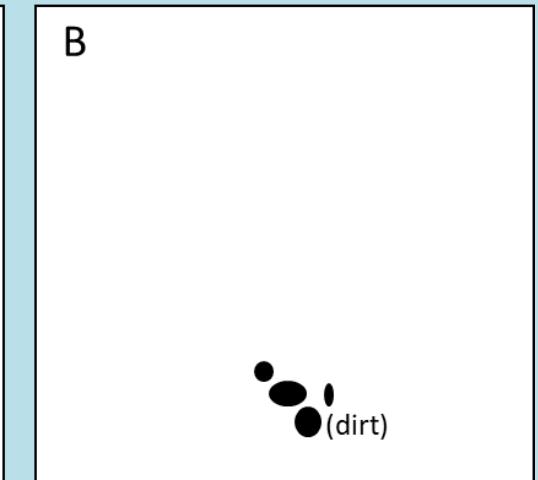
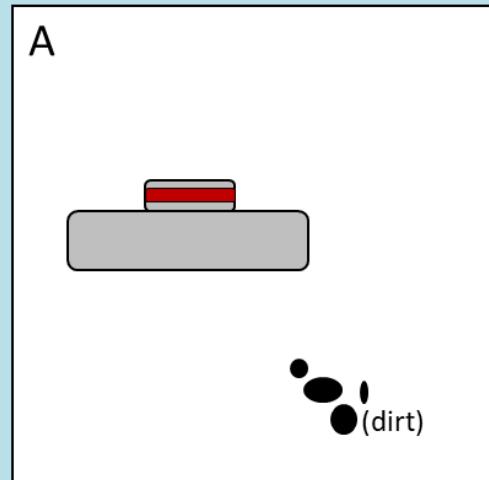
2.1 Structure of Agents (Architecture)

Algorithm: Reflex-Vacuum Agent

```
if status = dirty then  
    return suck  
end
```

```
else if location = A then  
    return right  
end
```

```
else if location = B then  
    return left  
end
```



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

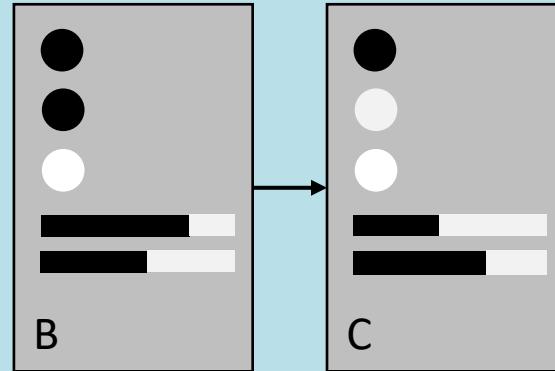
2.1 Components of Agents

Atomic representation



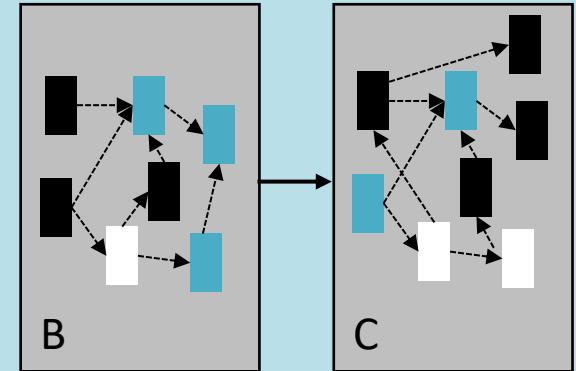
- a state (such as B or C) is a black box with no internal structure

Factored representation



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

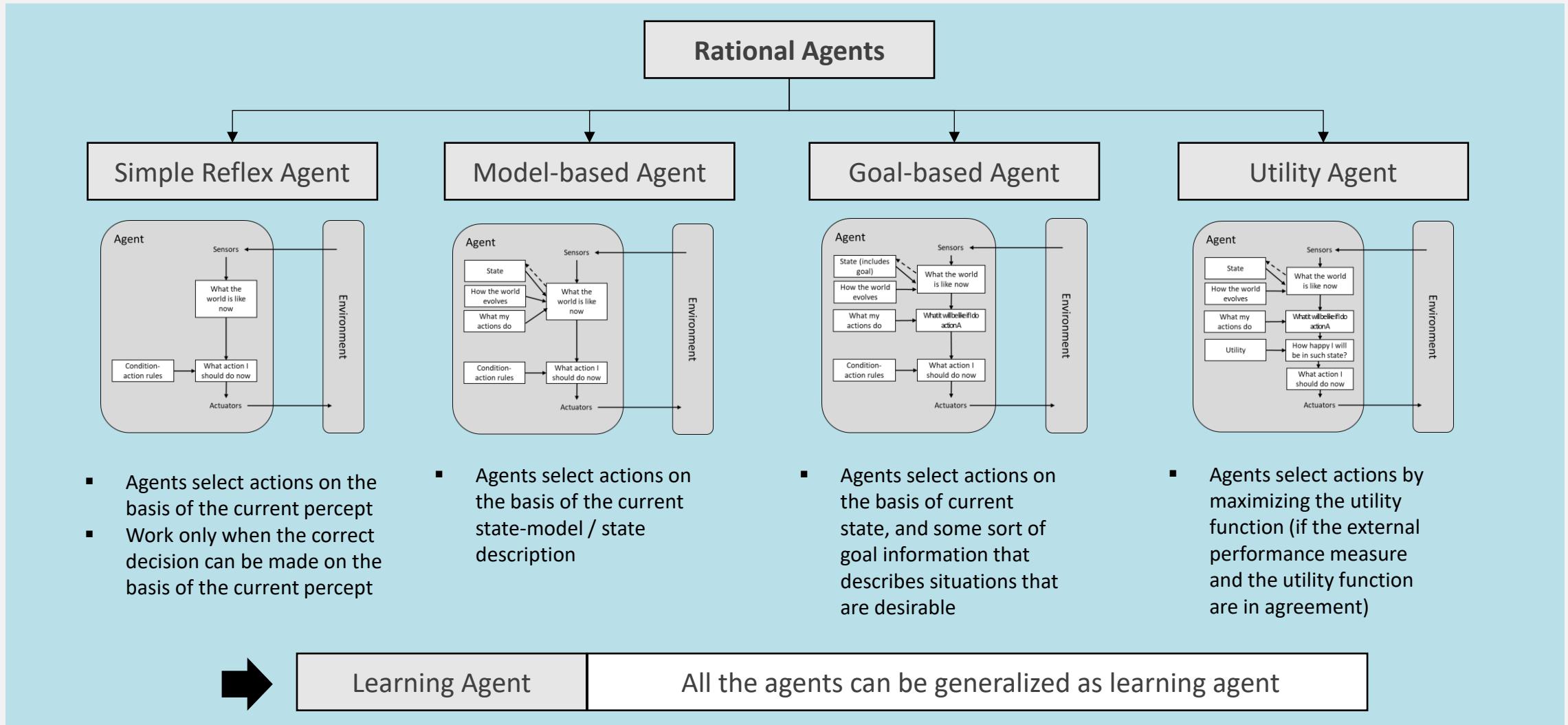
Structured representation



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

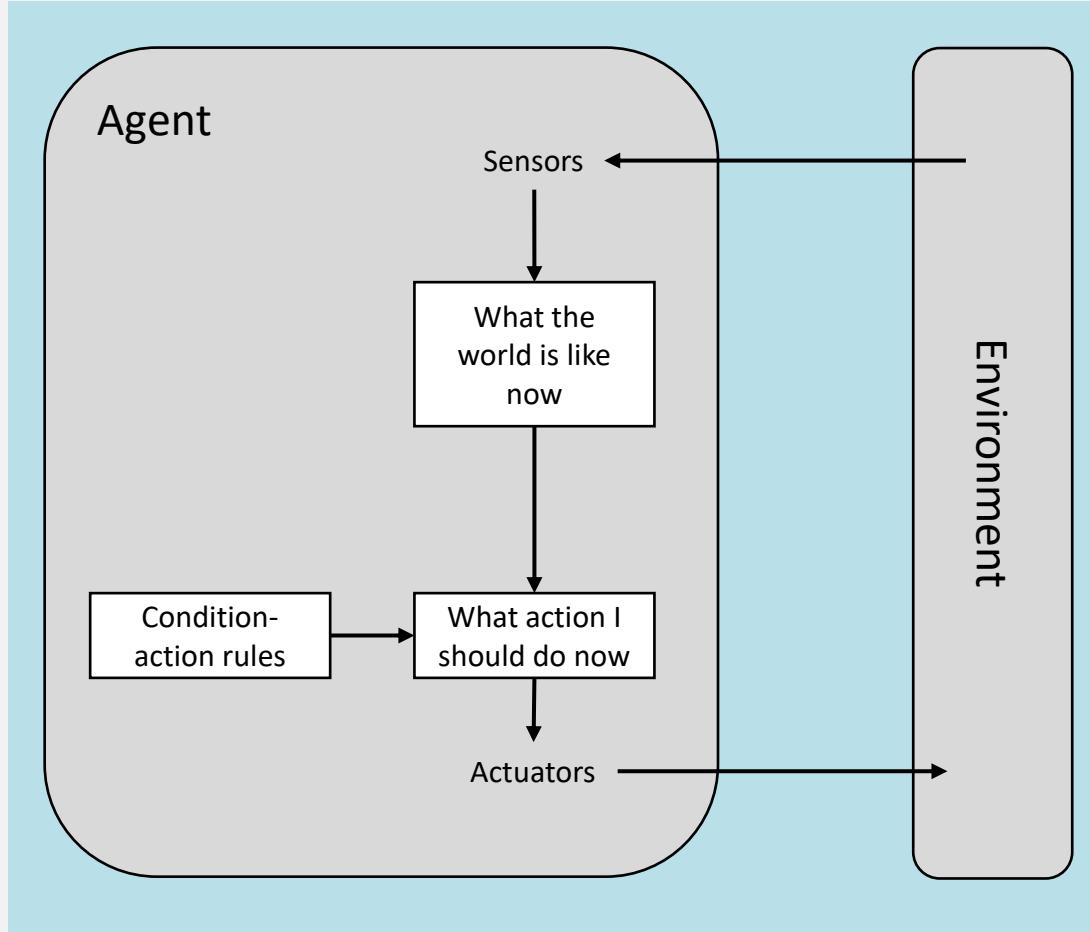
Adapted from Russell, S., & Norvig, P. (2016)

2.1 Subtypes of Agents (Russell & Norvig, 2016)



2.1 Simple Reflex Agent

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



Algorithm: Simple Reflex Agent

persistent: rules, a set of condition–action rules

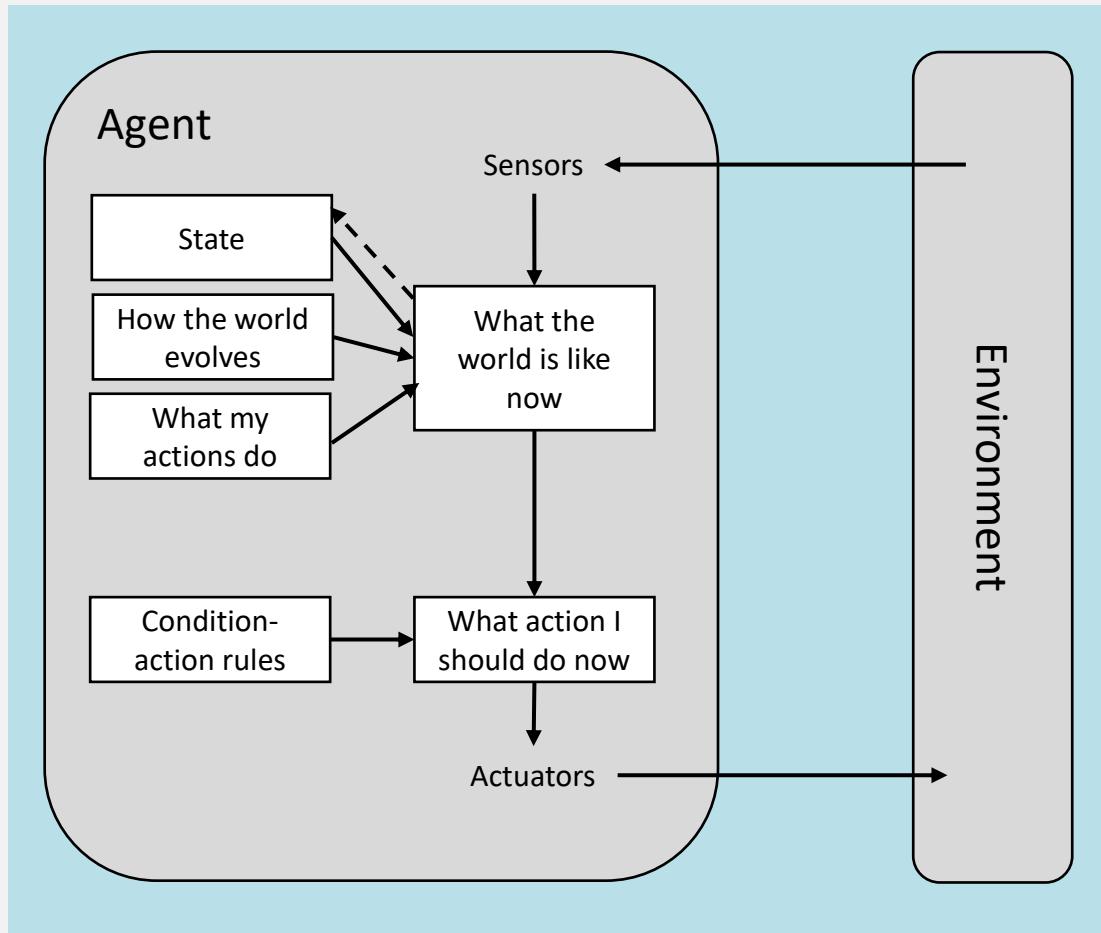
```
state  $\leftarrow$  INTERPRET-INPUT(percept)
rule  $\leftarrow$  RULE-MATCH(state, rules)
action  $\leftarrow$  rule.ACTION
```

return action

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

2.1 Model-based Reflex Agent

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



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 \leftarrow *UPDATE-STATE(state, action, percept, model)*

rule \leftarrow *RULE-MATCH(state, rules)*

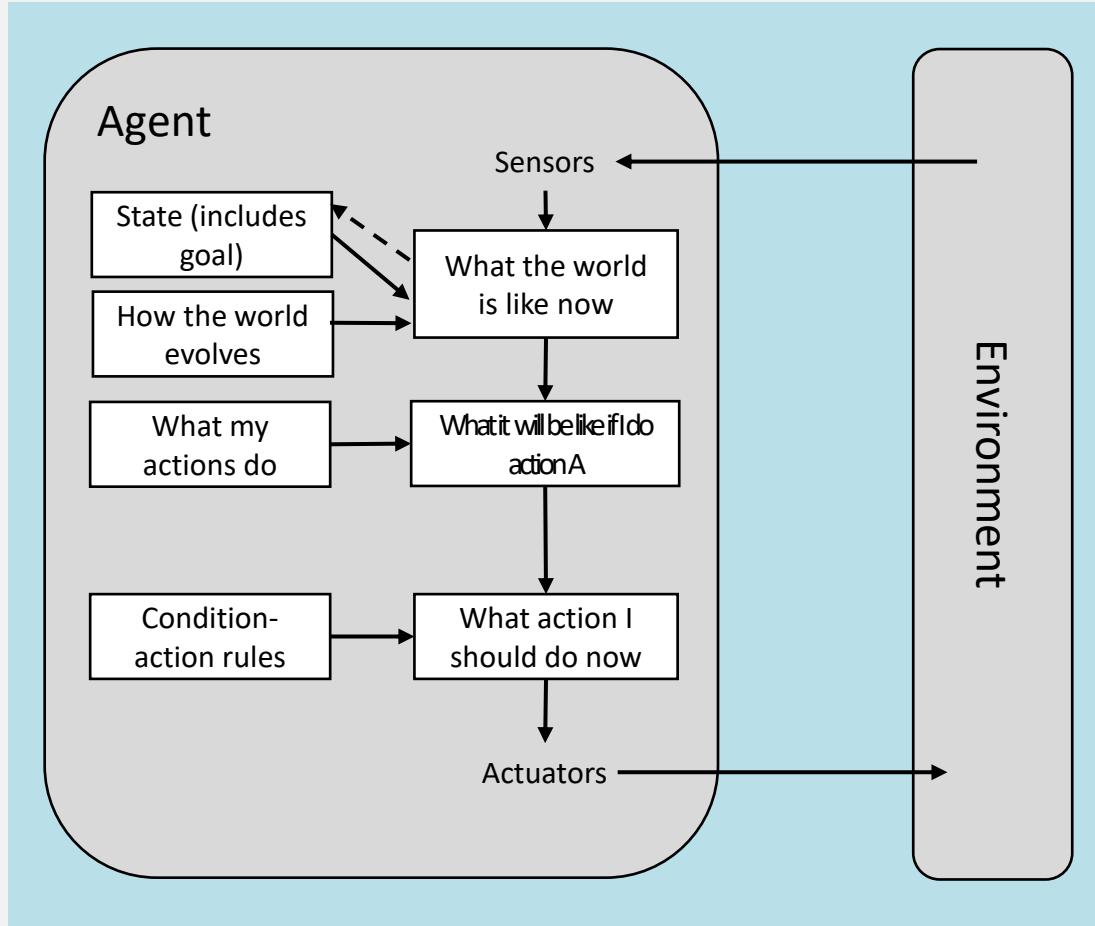
action \leftarrow *rule.ACTION*

return action

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

2.1 Goal-based Agents

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



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

$state \leftarrow UPDATE-STATE(state, action, percept, model, goal)$

$rule \leftarrow RULE-MATCH(state, rules)$

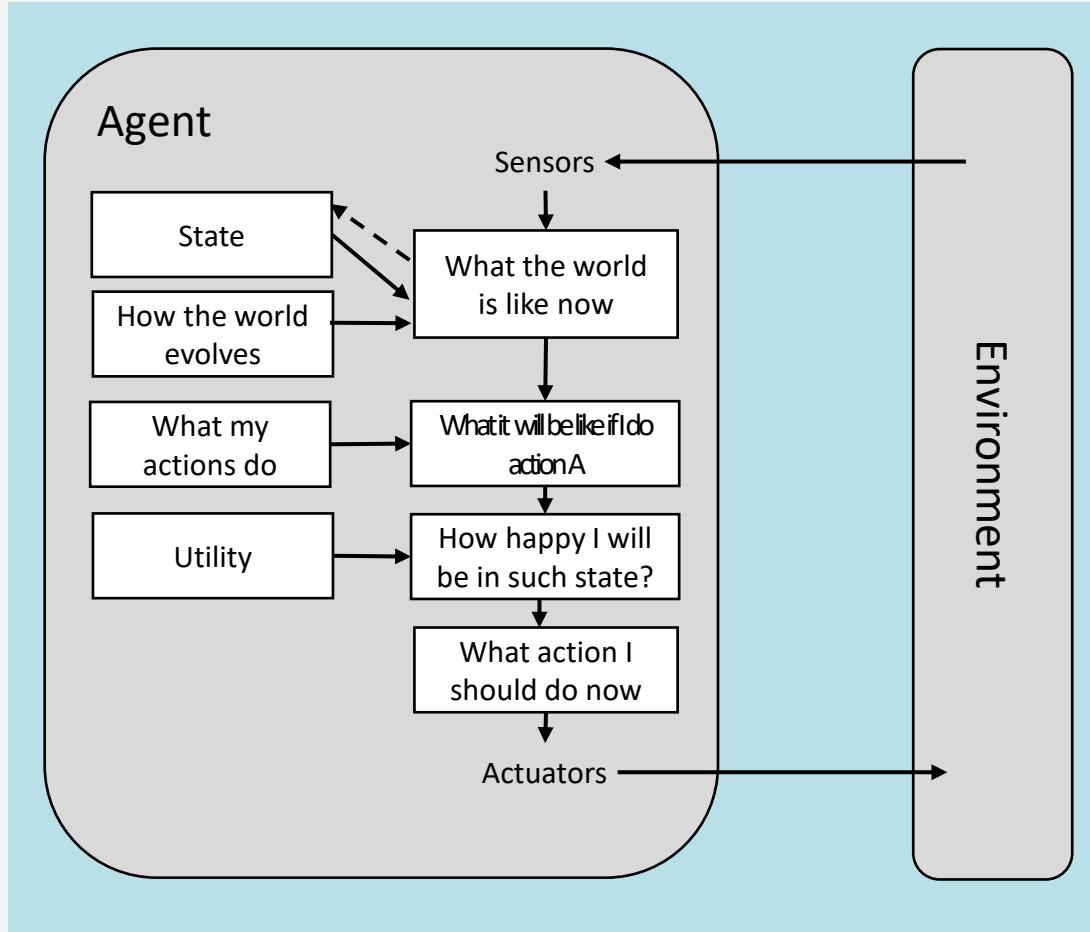
$action \leftarrow rule.ACTION$

return action

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

2.1 Utility-based Agent

- Utility function to describe desired behavior



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

$state \leftarrow UPDATE-STATE(state, action, percept, model, utility\ function)$

$rule \leftarrow RULE-MATCH(state, rules)$

$action \leftarrow rule.ACTION$

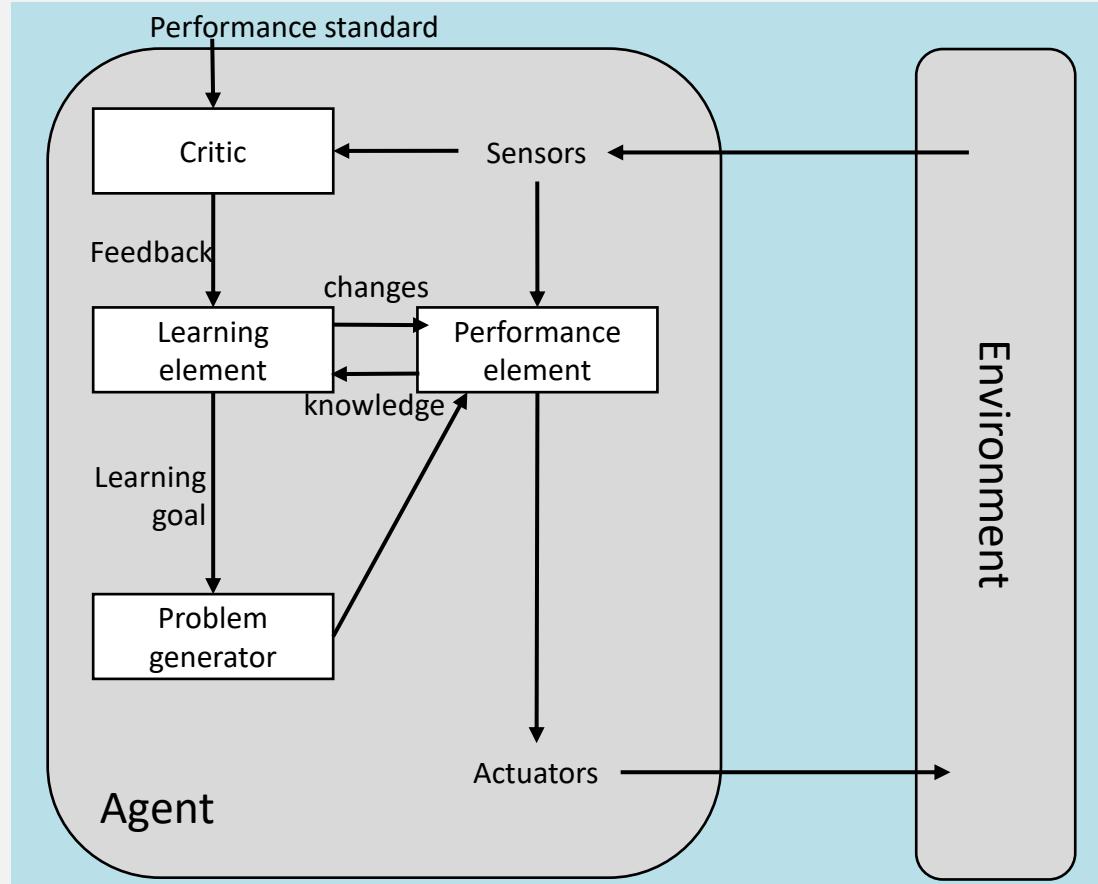
return action

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

2.1 Conclusion: The Learning Agent



- Agent improves its performance by learning optimal outputs



- Turing proposed 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

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

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

2 Search, Problem Solving, and Planning

2.1 Intelligent Agents

2.2 Solving Problems by Searching

2.3 Beyond Classical Search

2.4 Adversarial Search and Game Theory

2.5 Constraint Satisfaction Problems

► What we will learn:

- We define the concept of rational agents (\approx 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

2.2 Why We Need Goals: Example Roadtrip Planning



Adapted from Russell, 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



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

2.2 Well-defined Problems and Solutions



Adapted from Russell, 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

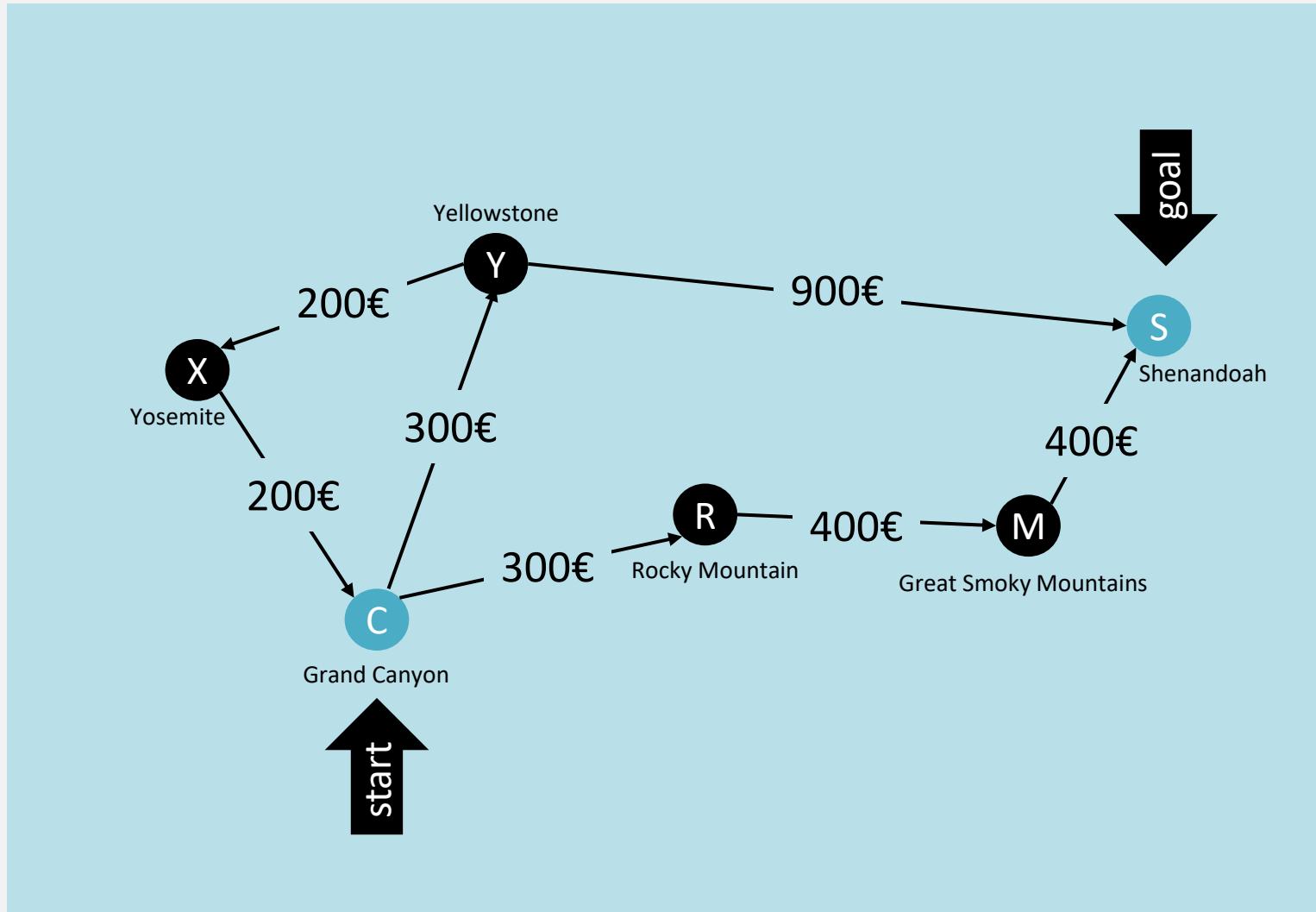


Image source ↗ [US NationalParks](#) (2015) by Mwierschkec ↗ [CC BY-SA 4.0](#)

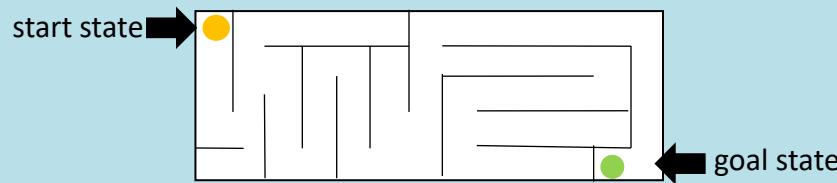
Goal: Be in Shenandoah

Other aspects:

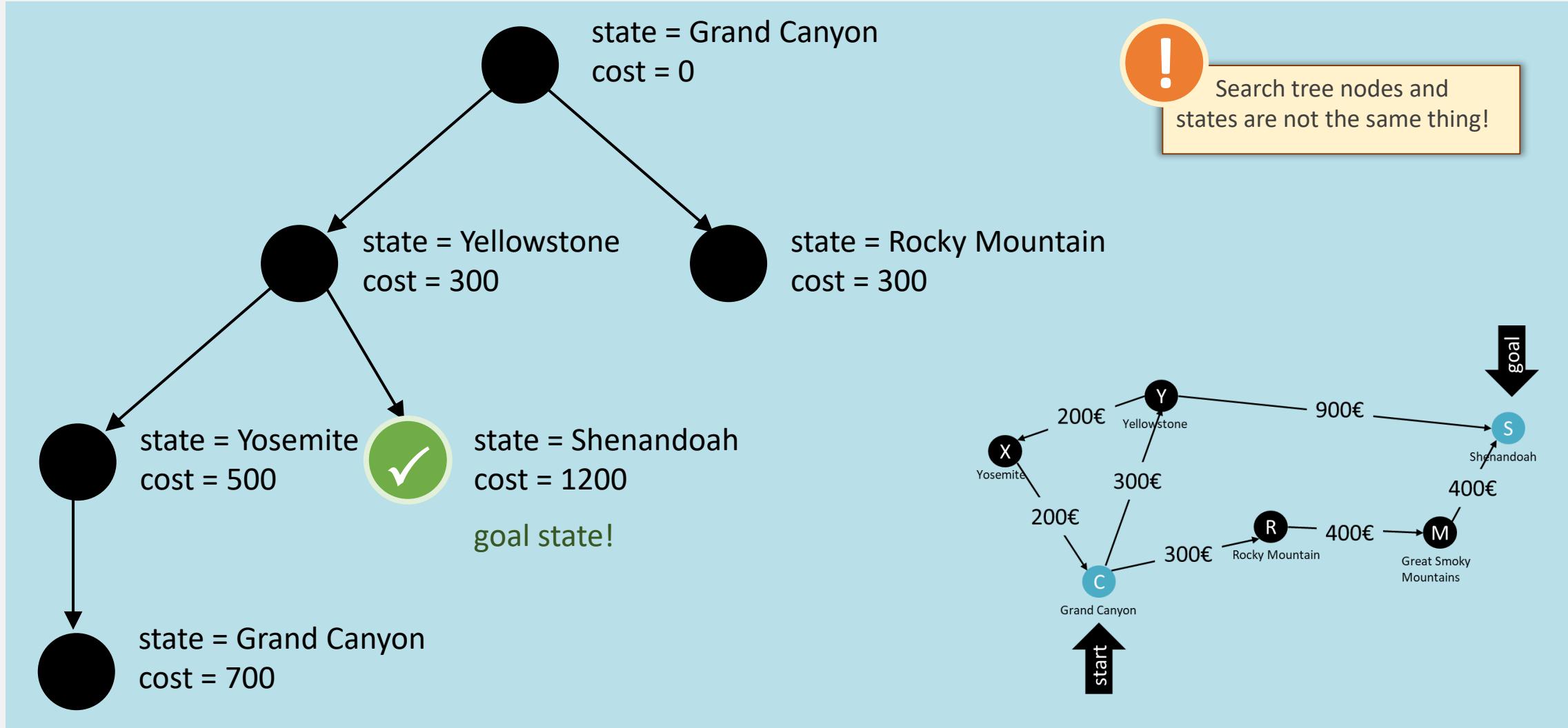
- Initial state or start: Grand Canyon
- Consider Driving costs
- Visit Top 5 parks

2.2 Possible Solution: Model Roadtrip-Problem as Search

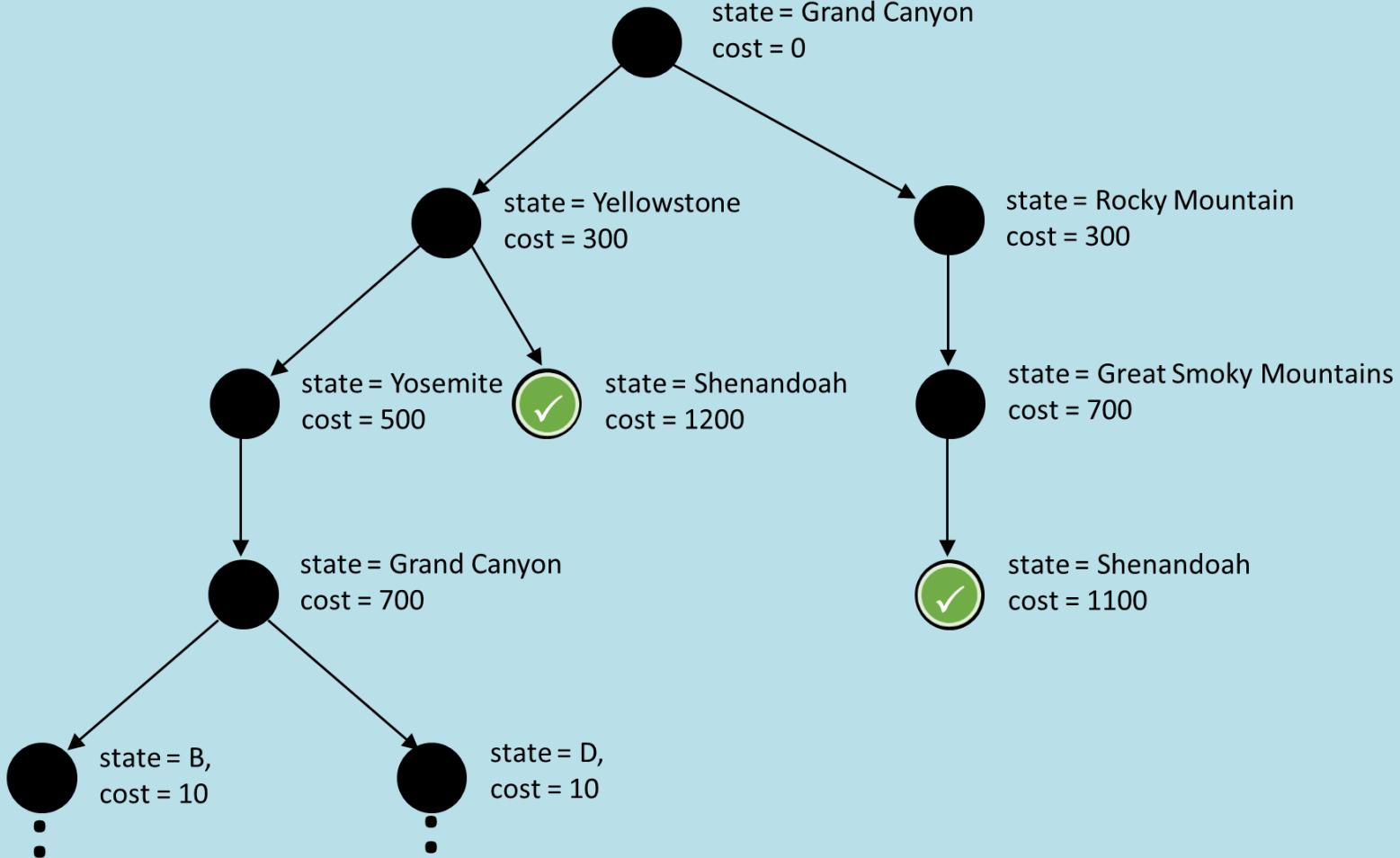
- Model problems as search problems
- Our agent does not have to execute all actions in real life while searching for solution
- We want to find a sequence of actions that will lead us to a desired state
 - We want to minimize number of actions
 - We want to minimize total cost of actions (more general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



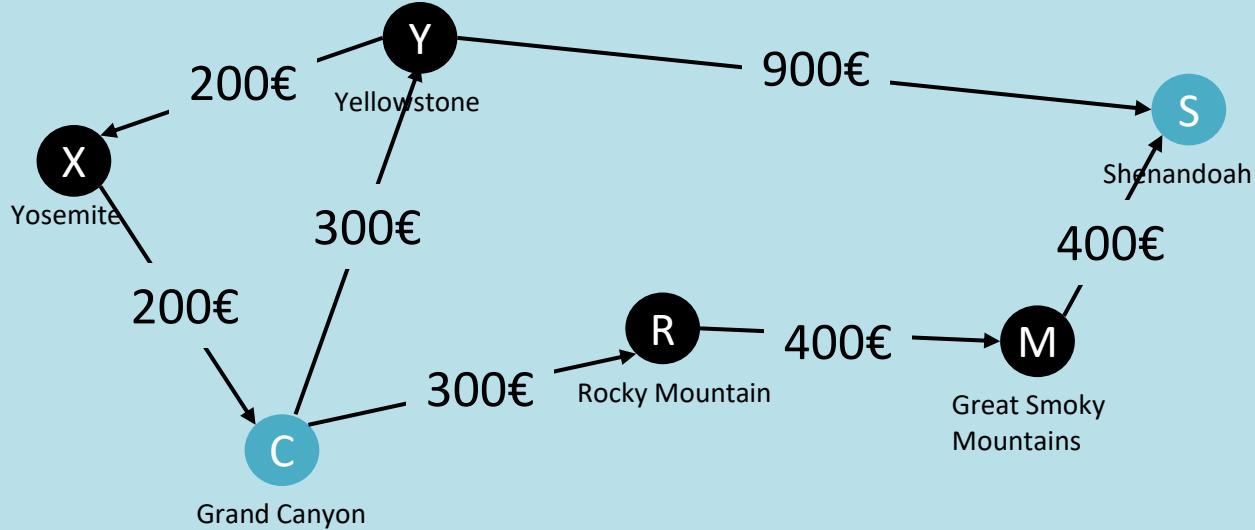
2.2 Model Problem as Search-tree ▶ Step 1



2.2 Model Problem as Search-tree ▶ Step 2



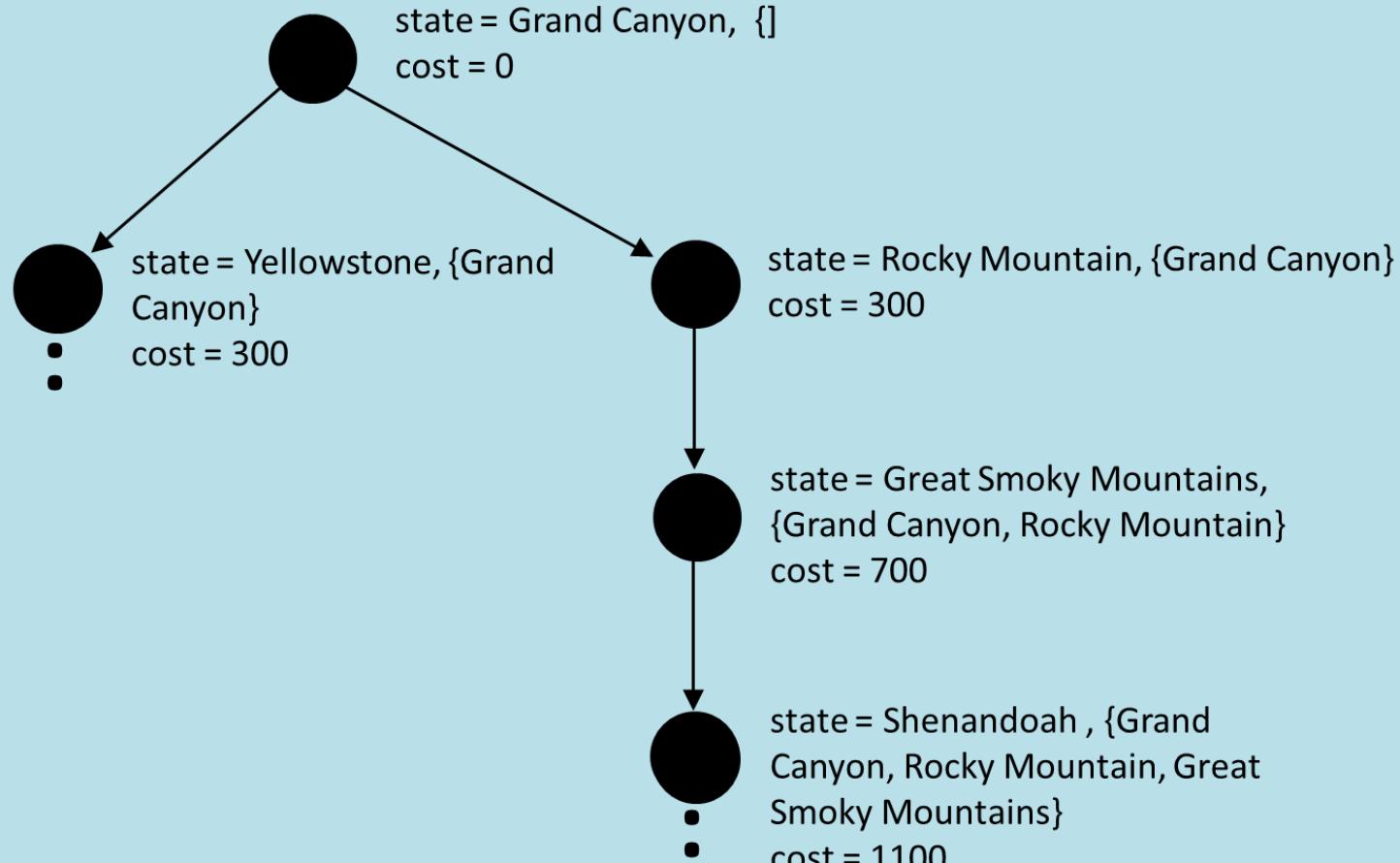
2.2 From Rout-Finding to Touring-Problems: Visit TOP 5 Nationalparks



Goal: Visit all vertices on the graph

- As 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)

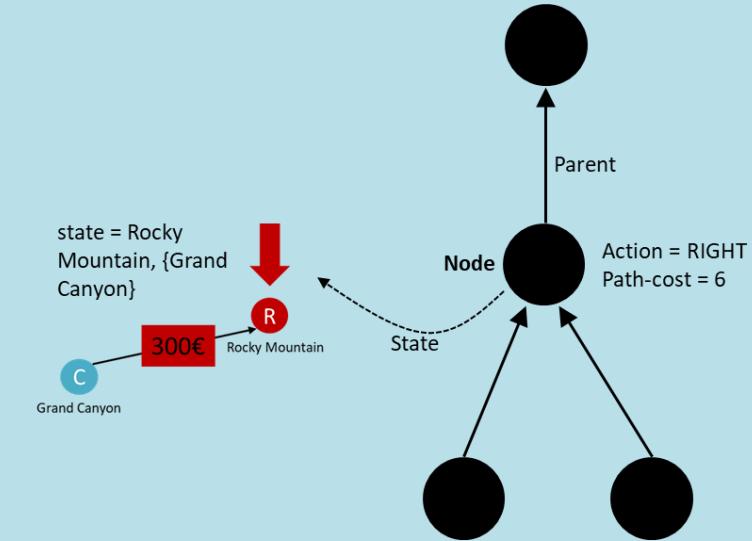
For every state, a set of **actions** that we can take

- 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 Russell, S., & Norvig, P. (2016)

2.2 Problem-Solving Agents

- We can formalize these concepts into a general concept of a 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 \leftarrow UPDATE-STATE(*state*, *percept*)

If *seq* is empty **then**

goal \leftarrow FORMULATE-GOAL(*state*)

problem \leftarrow FORMULATE-PROBLEM(*state*, *goal*)

seq \leftarrow SEARCH(*problem*)

If *seq* = failure **then return** null action

action \leftarrow FIRST(*seq*)

seq \leftarrow REST(*seq*)

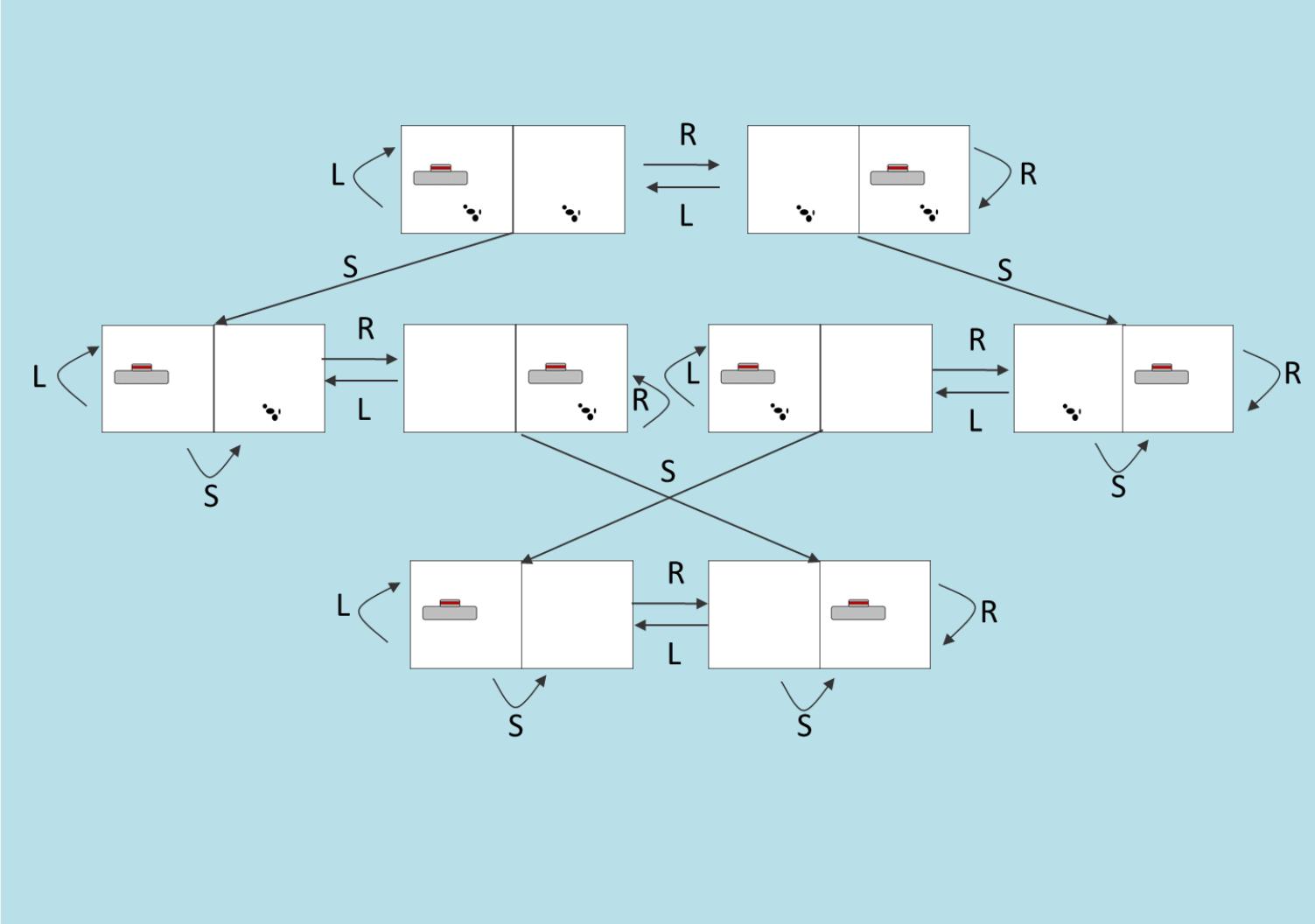
return *action*

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 not considered in 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 Russell, 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 Russell, S., & Norvig, P. (2016);

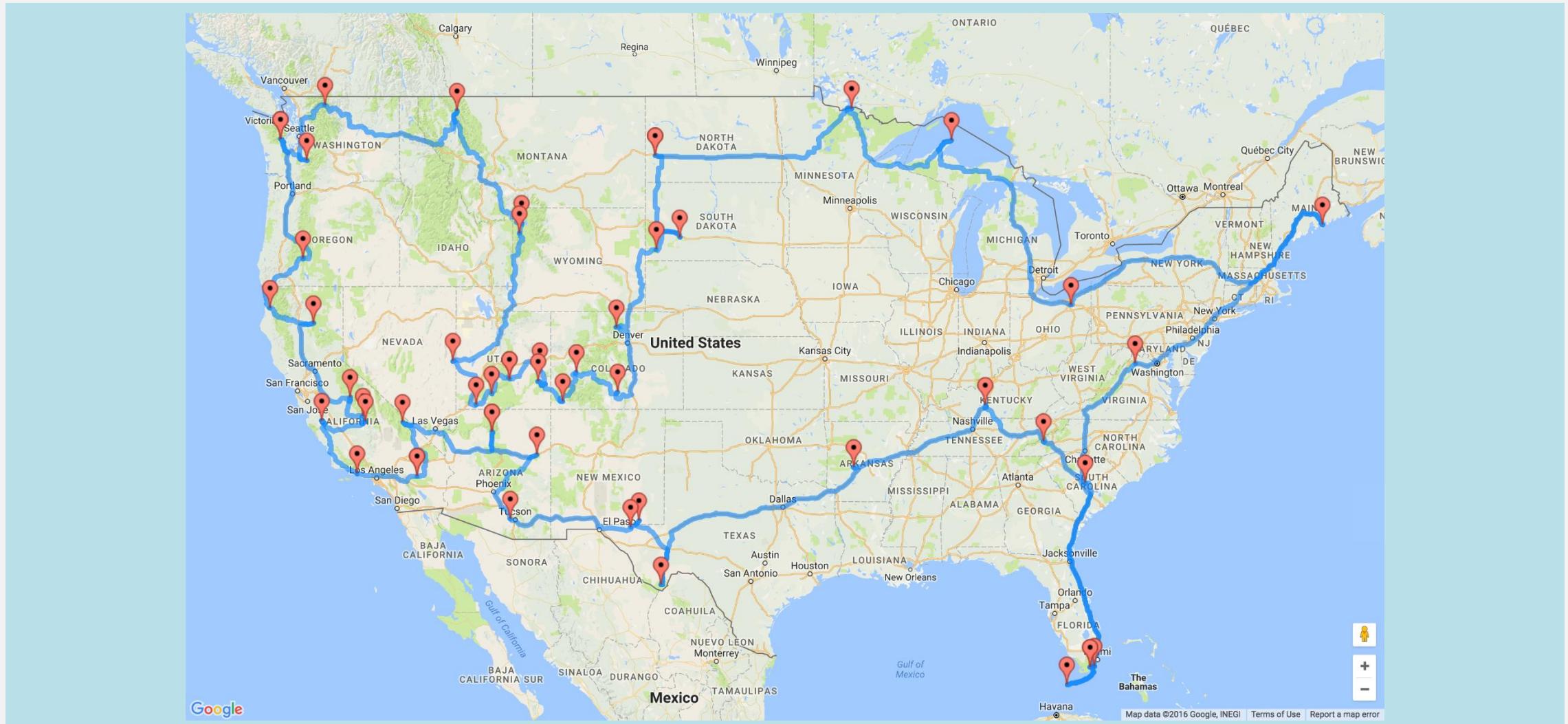
- **States:** The state is determined by both the agent location and the dirt locations.
- **Initial state:** Any state can be designated as the initial state.
- **Actions:** *Left*, *Right*, and *Suck*.
- **Transition model:** The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Sucking* 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 Russell, 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)

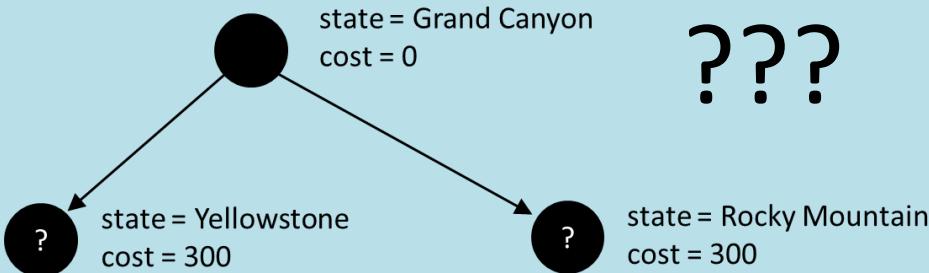
2.2 Other Popular Real-world Problems

- 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 Russell, 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

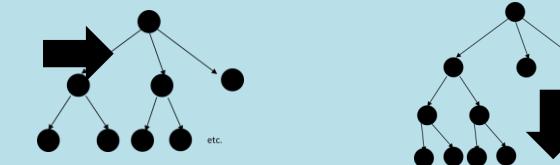
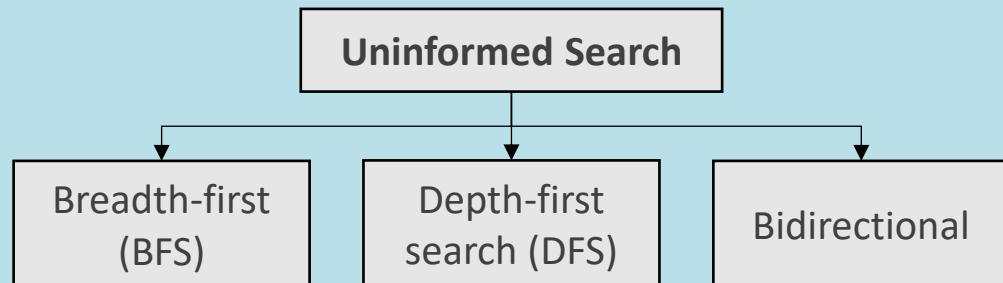
Adapted from Russell, S., & Norvig, P. (2016)

2.2 Overview: Fundamental Search Algorithms

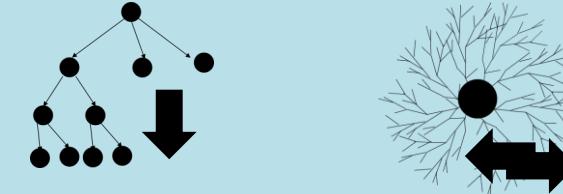
D

Uninformed Search

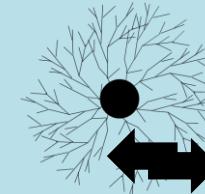
Strategies have no additional information about states beyond that provided in the problem definition



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



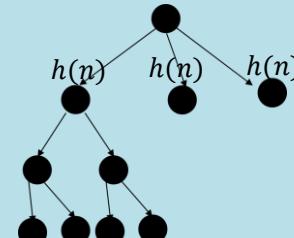
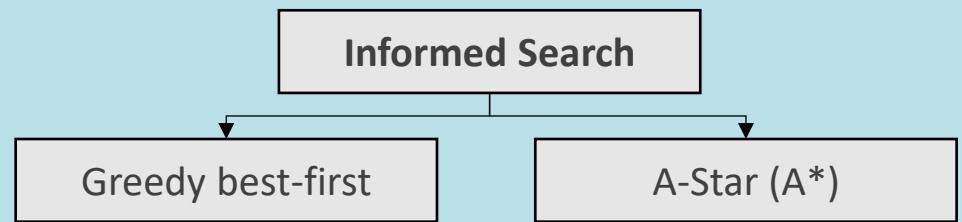
- Always expands the deepest node in the current frontier of the search tree.



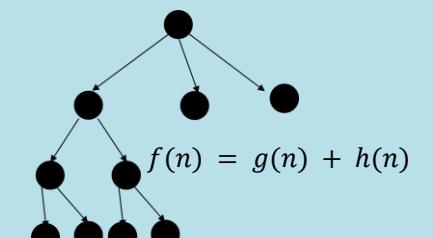
- Always expands the deepest node in the current frontier of the search tree.

Informed Search

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



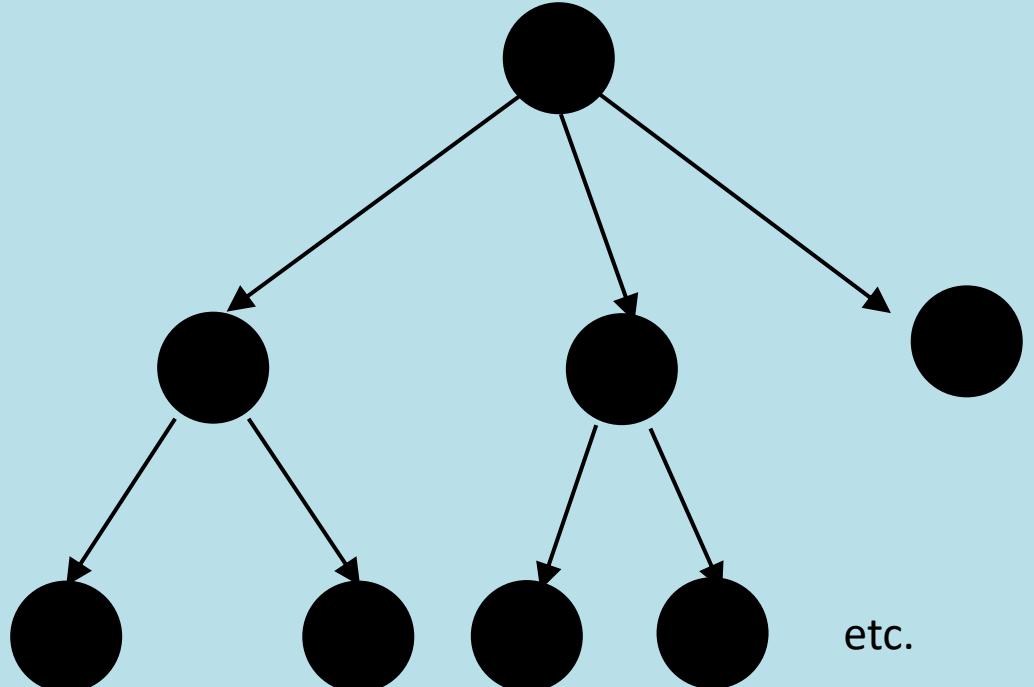
- Expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.



- Cluster observations into (distinct/different) groups

- 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!

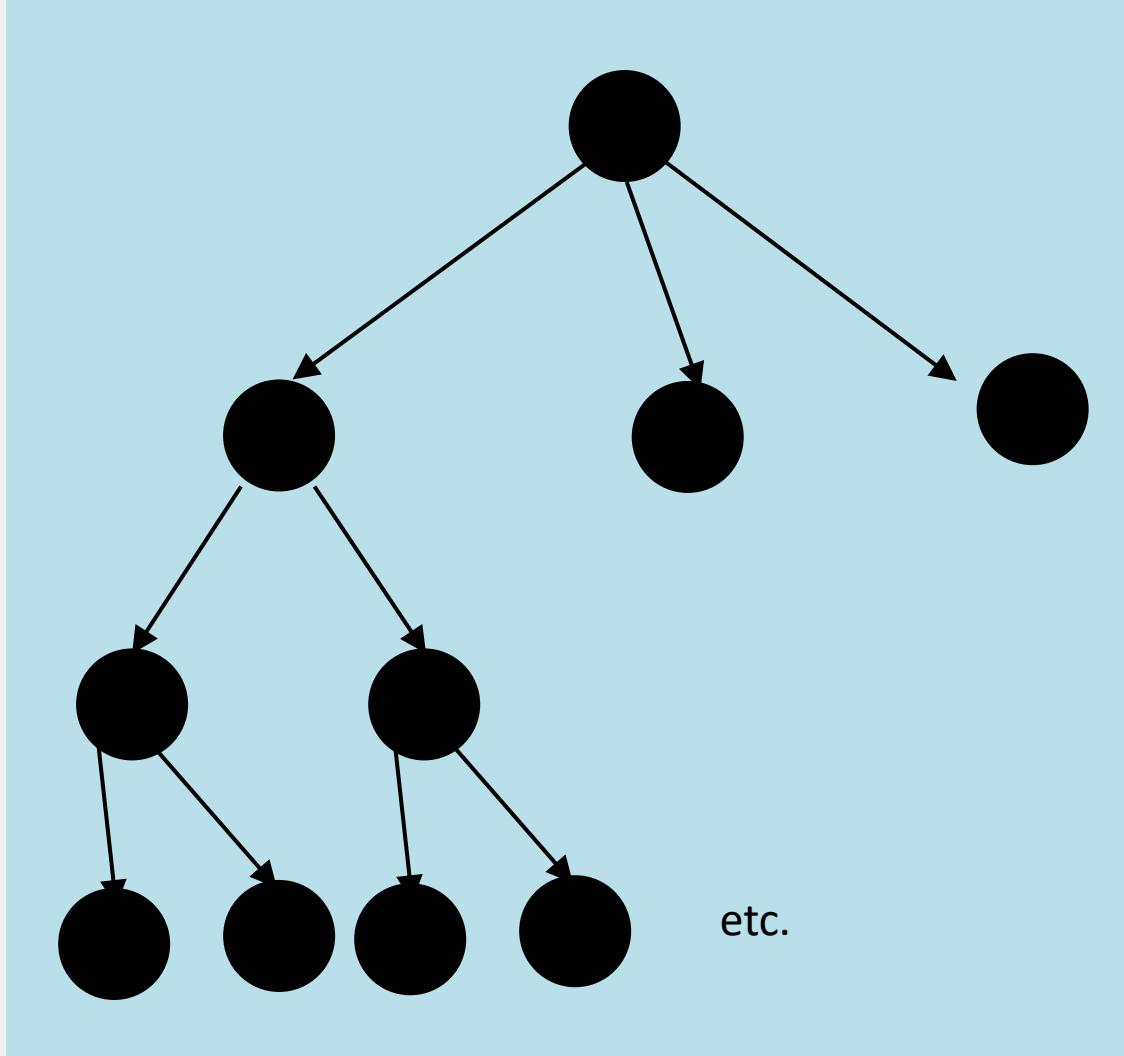
2.2 Breadth-first Search



- 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 !!!

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

2.2 Depth-first Search



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

- 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
- $db + (d - 1)b^2 + (d - 2)b^3 + \dots + 1bd$ is $O(b^d)$ for $b > 1$

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

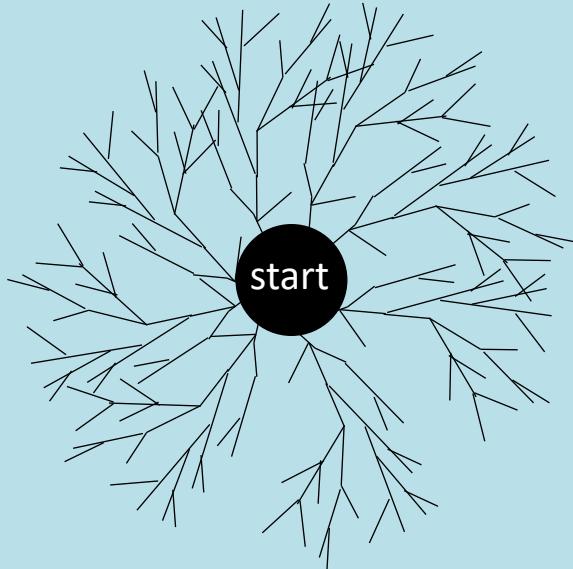
2.2 Let's Start Thinking About Cost

- **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

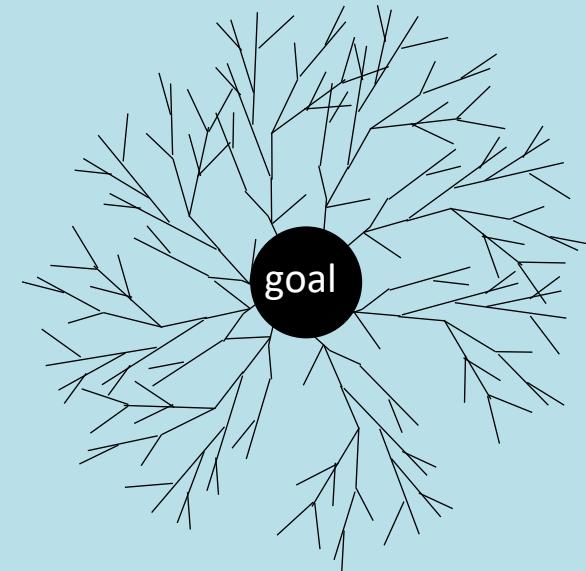
Adapted from Russell, S., & Norvig, P. (2016);

2.2 Bidirectional Search

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



...



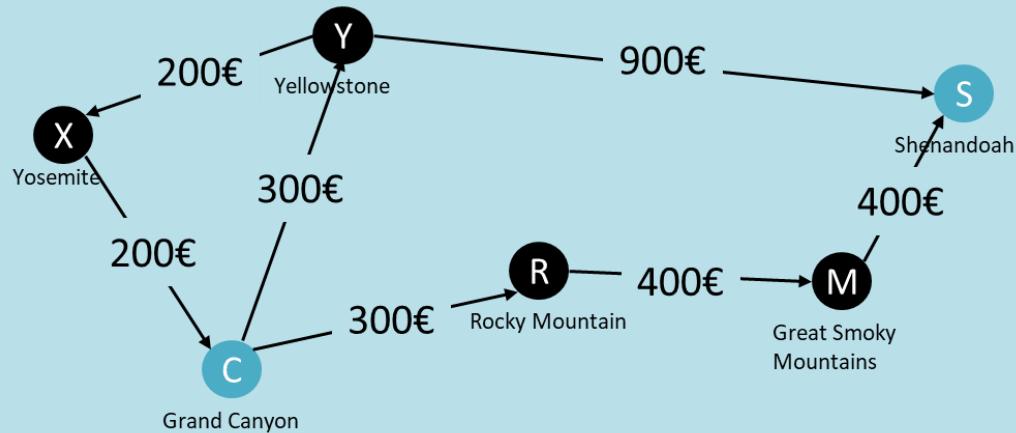
If the shallowest solution has depth d and branching factor is b on both sides, requires only $O(b^{d/2})$ nodes to be explored!

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

- 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

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.)

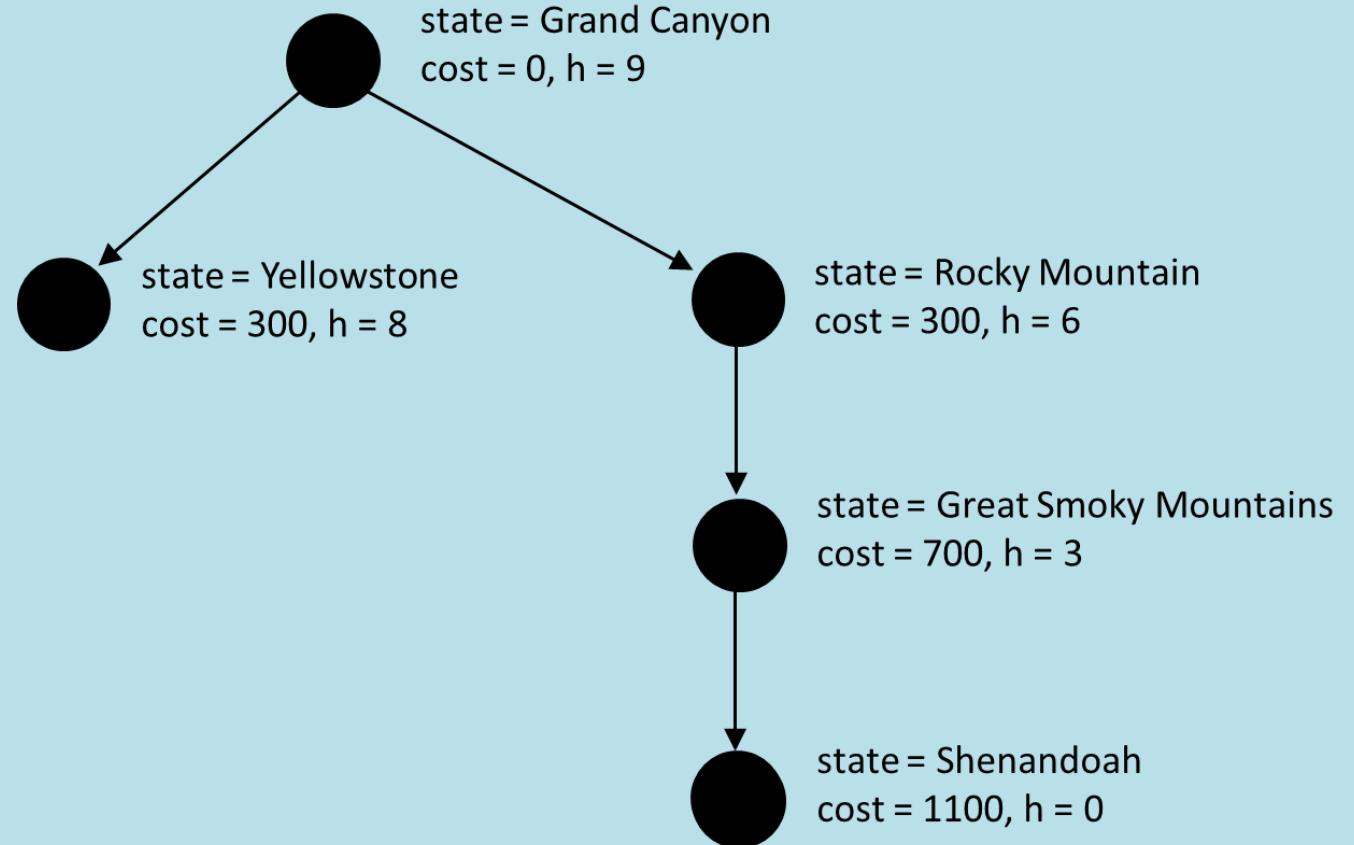


- We assume: $h(C) = 9$, $h(Y) = 8$, $h(X) = 9$, $h(R) = 6$, $h(M) = 3$, $h(S) = 0$
- Can use heuristic to decide which nodes to expand first

Adapted from Russell, S., & Norvig, P. (2016) | Image sources: ↗ [US National Parks](#) (2015) by Mwierschke ↗ [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 solution



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

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*

- 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)

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

2.2 Example: Age of Empires 2 Pathfinding

freeaoe / src / ai / location.hh

Code Blame 334 lines (285 loc) · 8.38 KB

```
120
121     /// Add \a width columns, in place.
122     inline position&
123     operator+=(position& res, position::counter_type width)
124     {
125         res.columns_ += width;
126         return res;
127     }
128
129     /// Add \a width columns.
130     inline position
131     operator+(position res, position::counter_type width)
132     {
133         return res += width;
134     }
135
136     /// Subtract \a width columns, in place.
137     inline position&
138     operator-=(position& res, position::counter_type width)
139     {
140         res.columns_ -= width;
141         return res;
142     }
143
144     /// Subtract \a width columns.
145     inline position
146     operator-(position res, position::counter_type width)
147     {
148         return res -= width;
149     }
150
151     /// Compare two position objects.
152     inline bool
153     operator==(const position& pos1, const position& pos2)
154     {
155         return (pos1.line == pos2.line
156                 && pos1.column == pos2.column
157                 && (pos1.filename == pos2.filename
158                     || (pos1.filename && pos2.filename
159                         && *pos1.filename == *pos2.filename)));
160     }
```

- The map in AOE2 is a grid of squares which computes distances as Manhattan distance
- Some of the squares are not accessible (water, trees, buildings, squares with units in them etc.)

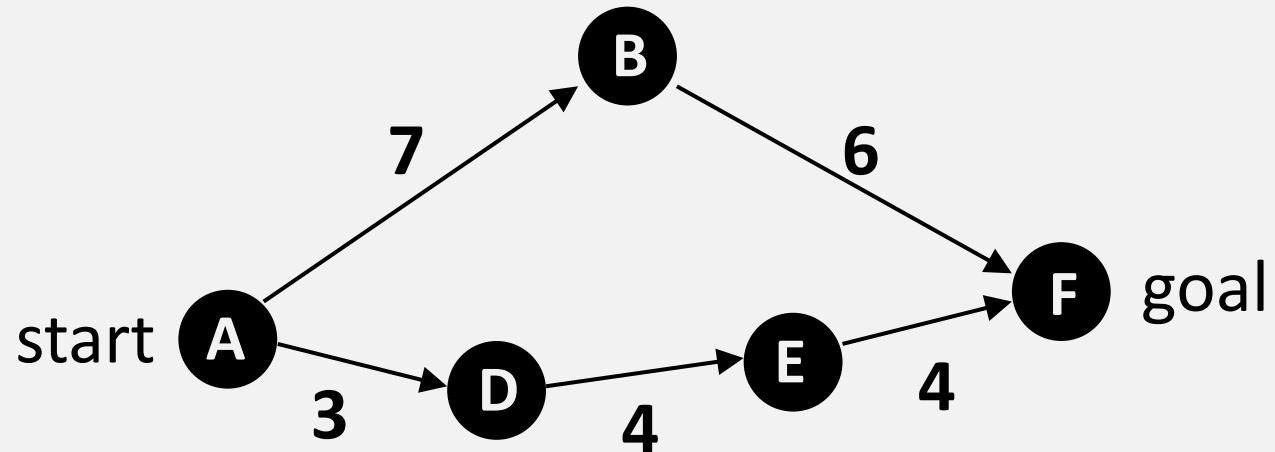


Source: <https://github.com/sandmark/freeaoe/blob/master/src/ai/location.hh>

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

2 Search, Problem Solving, and Planning

2.1 Intelligent Agents

2.2 Solving Problems by Searching

2.3 Beyond Classical Search

2.4 Adversarial Search and Game Theory

2.5 Constraint Satisfaction Problems

► What we will learn:

- We define the concept of rational agents (\approx 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

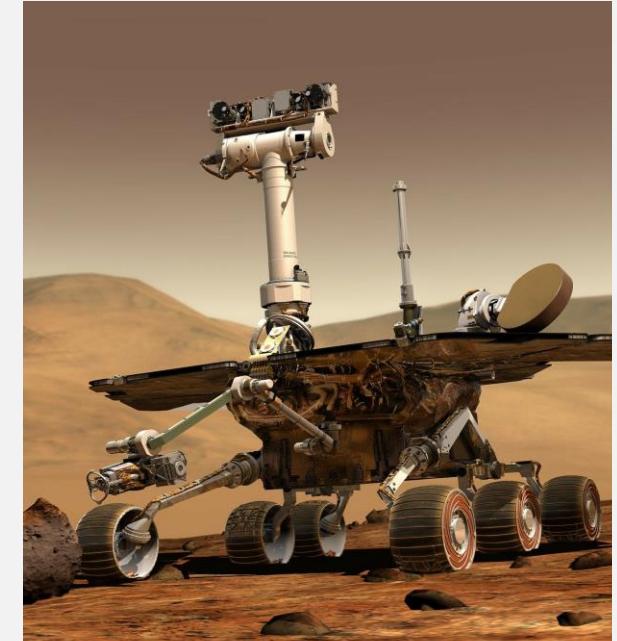


Image source: [↗ Pixabay](#) (2019) / [↗ CCO](#)

► 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”

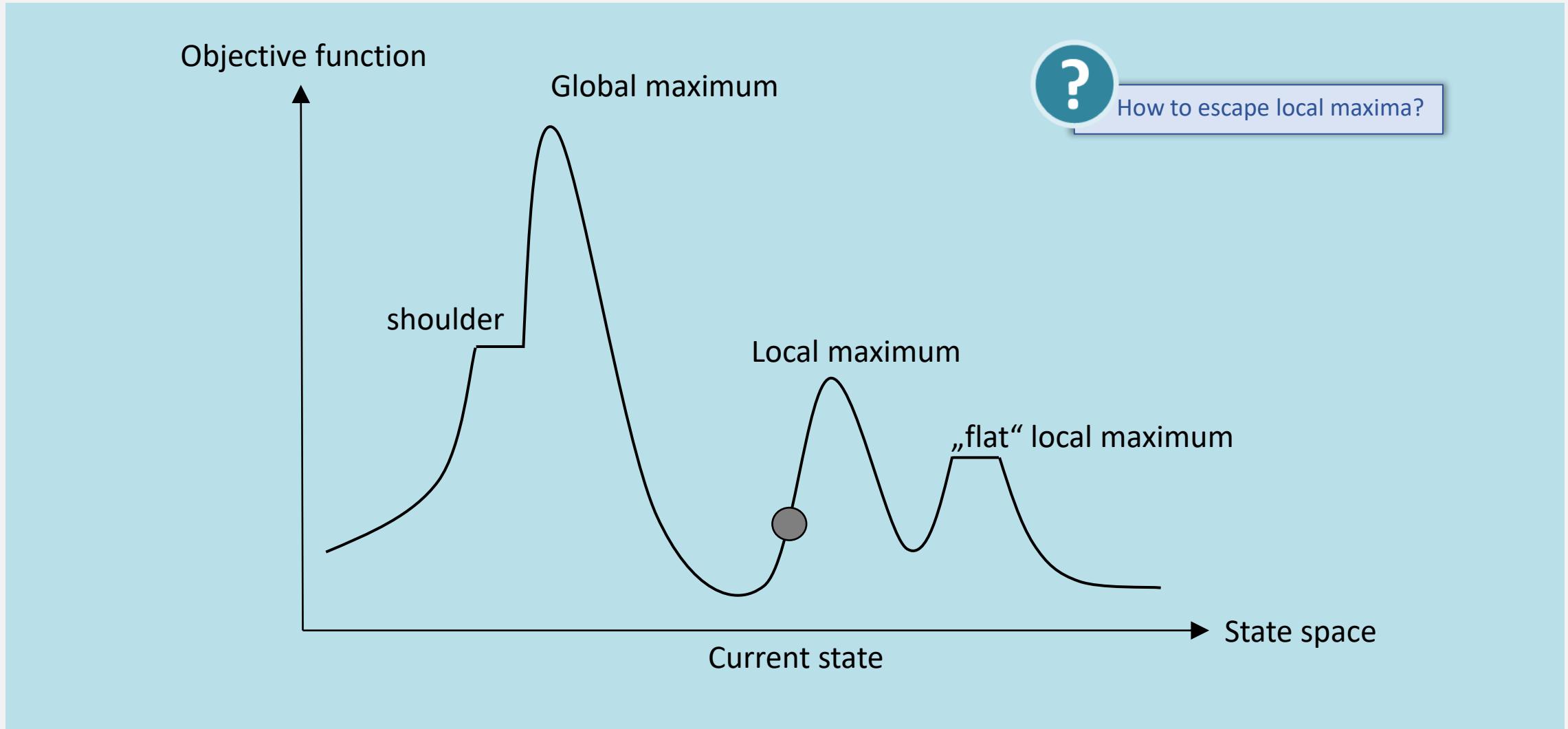


Image adapted from Russell, S., & Norvig, P. (2016);

2.3 How to Escape Local Maxima: Trivial Algorithms

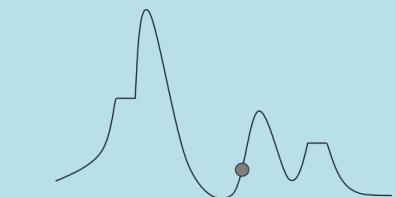
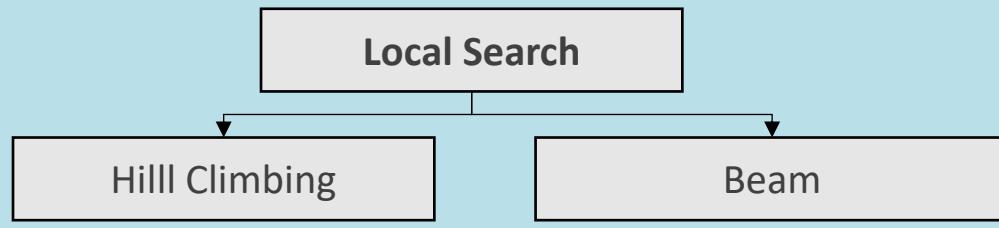
- Random Sampling
 - Generate a state randomly
- Random Walk
 - Randomly pick a neighbor of the current state
- Both algorithms asymptotically complete

2.2 Overview: Beyond Classical Search

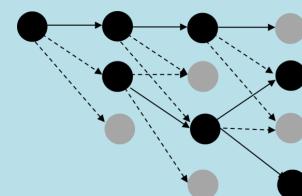
D

Local Search

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

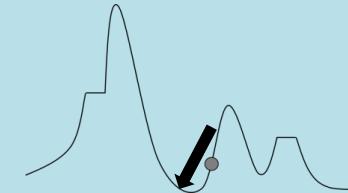


- Continually move in the direction of increasing value



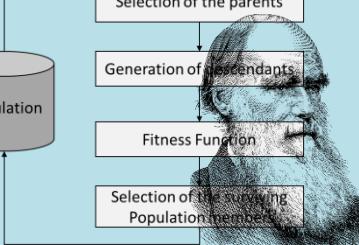
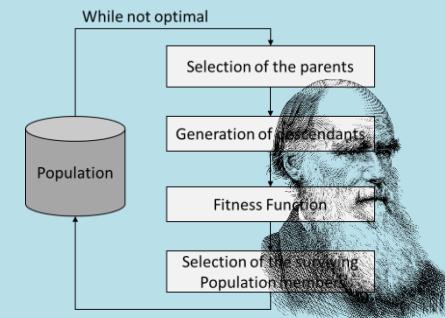
- expanding the most promising node in a limited set

Simulated Annealing



- 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 Russell, S., & Norvig, P. (2016) | | Image source: [↗ Pixabay](#) (2019) / [↗ CCO](#)

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, which 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 Russell, S., & Norvig, P. (2016);

2.3 Hill-Climbing Search

Algorithm: Hill Climbing

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

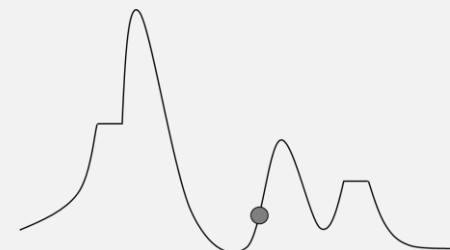
While STOP != TRUE

 neighbor ← a highest-valued successor 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 Russell, S., & Norvig, P. (2016)

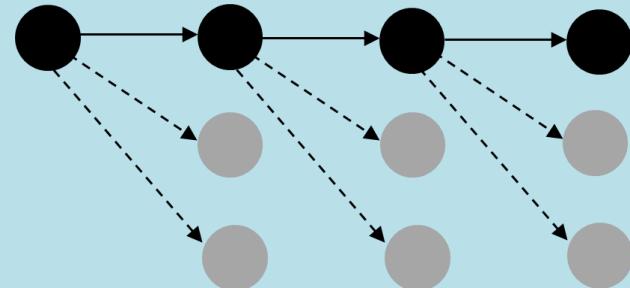
2.3 Local Beam Search

- 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

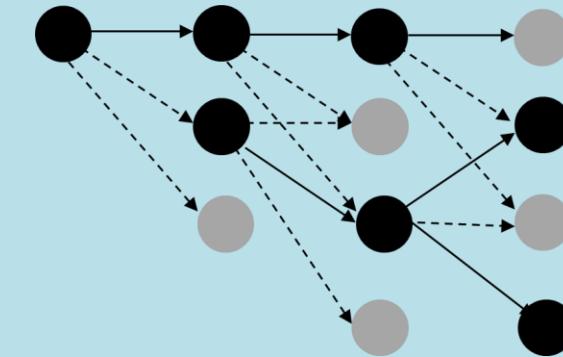


Is this the same as running k greedy searches in parallel?

Greedy search



Beam search



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

2.3 Simulated Annealing Search

Algorithm: Simulated Annealing

Initialize current to starting state

For $i = 1$ **to** ∞

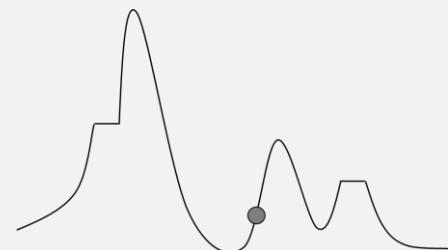
If $T(i) = 0$ **return** current

Let next = random successor of current

Let Δ = value(next) – value(current)

If $\Delta > 0$ **then let** current = next

Else let current = next with probability $\exp(\Delta/T(i))$



- **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

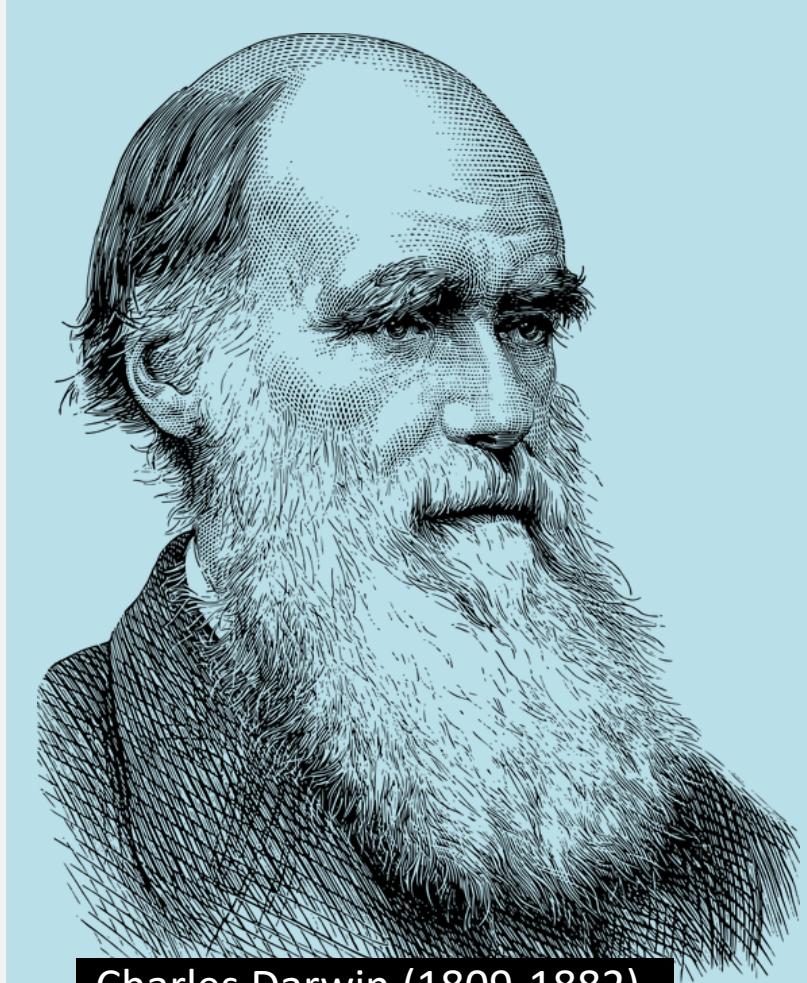
Adapted from Russell, S., & Norvig, P. (2016)

2.3 Conclusion: Simulated Annealing Search

- We can mathematically prove that a slow decrease in temperature 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 Russell, S., & Norvig, P. (2016);

2.3 Genetic/Evolutionary Algorithms



Charles Darwin (1809-1882)

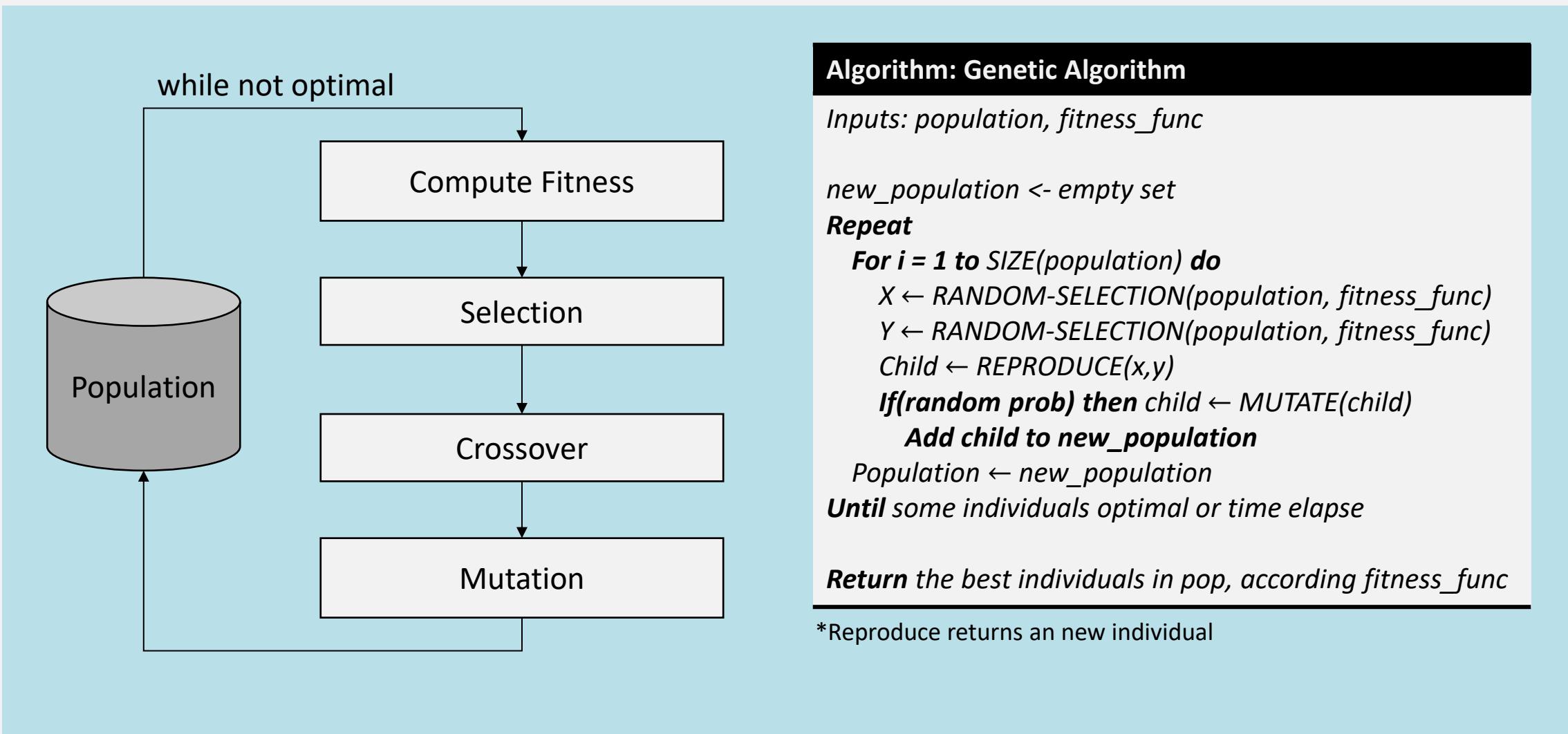
- Biological evolutionary model according to Darwin:
Selection = driving force of evolution
- Transfer to Computer Science: Evolution as optimization of complex, artificial systems
- Build machines that adapt to an defined working environment

Adapted from Russell, 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 create the best problem solution

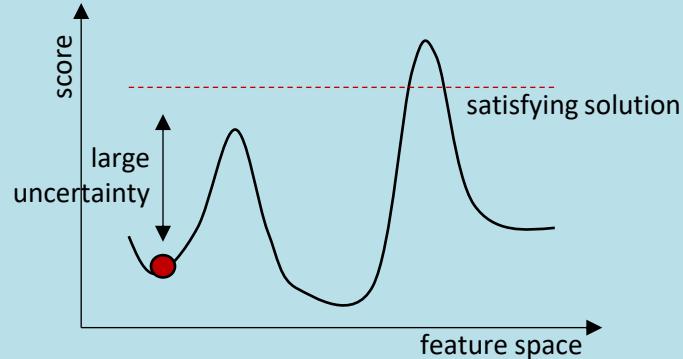
2.3 Basic Algorithm



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

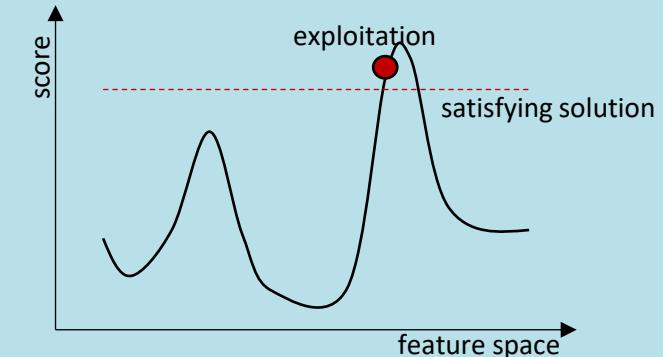
2.3 Generation of Descendants

Exploration



- Exploring the local space, local optimization

Exploitation



- Exploitation of the hypothesis space

- The stronger and more random changes are, the lower is the probability of producing better offsprings
- With local improvement methods, the risk of local minima is 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 Russell, S., & Norvig, P. (2016);

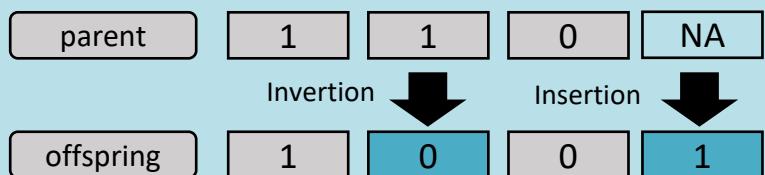
2.3 Mutation

Mutation

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

Mutation:

- 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

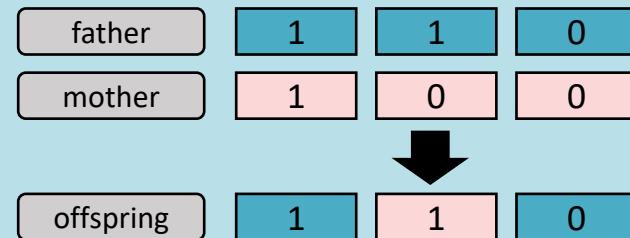


Crossover / Recombination

Mix properties of two or more parents

Crossover:

- Discrete recombination



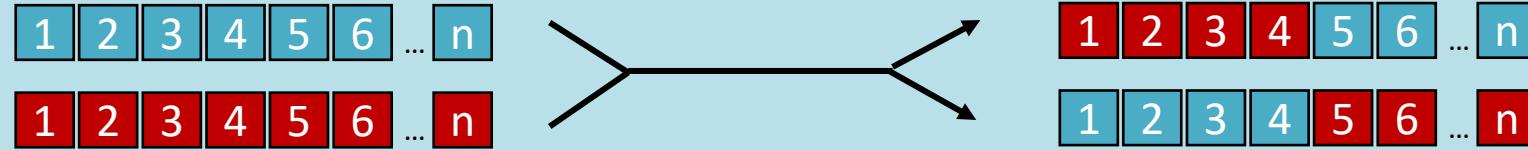
- Intermediate recombination (continuous representation)

$$\text{parent}_1 := x \text{ and } \text{parent}_2 := y, \\ \text{offspring}_i := (\textcolor{blue}{x}_i + \textcolor{orange}{y}_i)/2$$

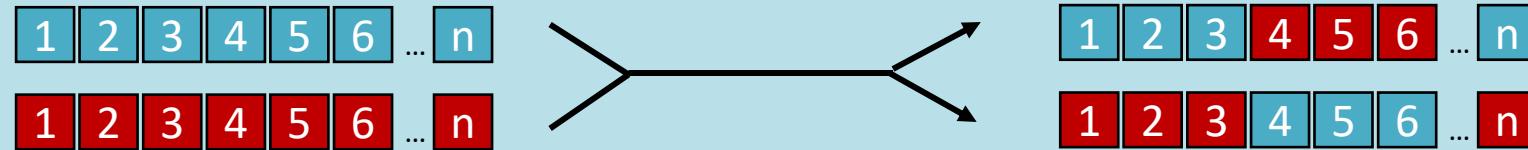
Adapted from Russell, S., & Norvig, P. (2016);

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 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

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 \Rightarrow 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 \rightarrow 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

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

$P(x)$: 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 : $P(x) \approx \frac{g(r(x))}{\sum_{x' \in Pop.} g(r(x'))}$ with

$P(x)$: Probability of selection of individual x

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

- 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

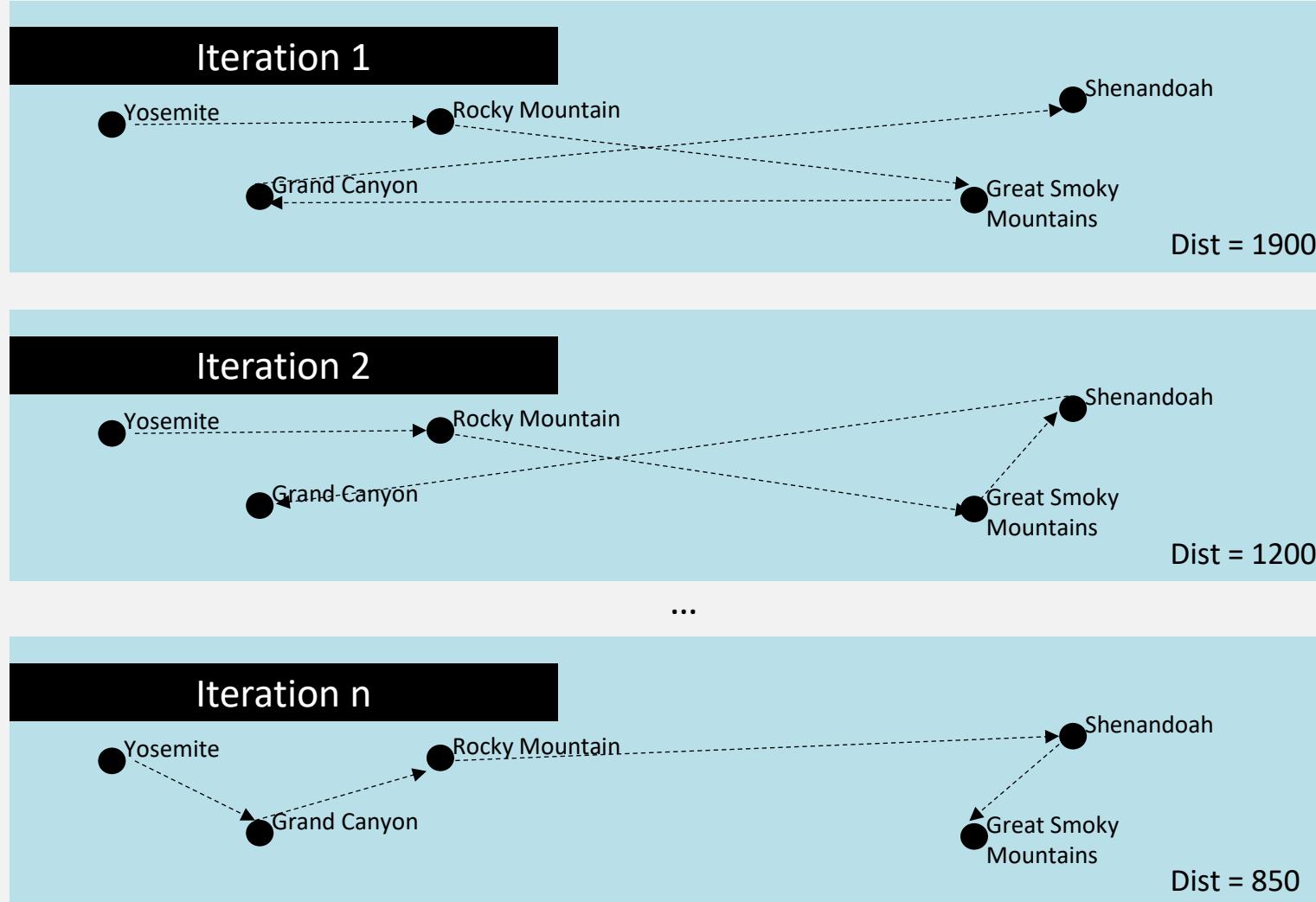


Image sources: ↗ [US NationalParks](#) (2015) by Mwierschke ↗ [CC BY-SA 4.0](#)

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:

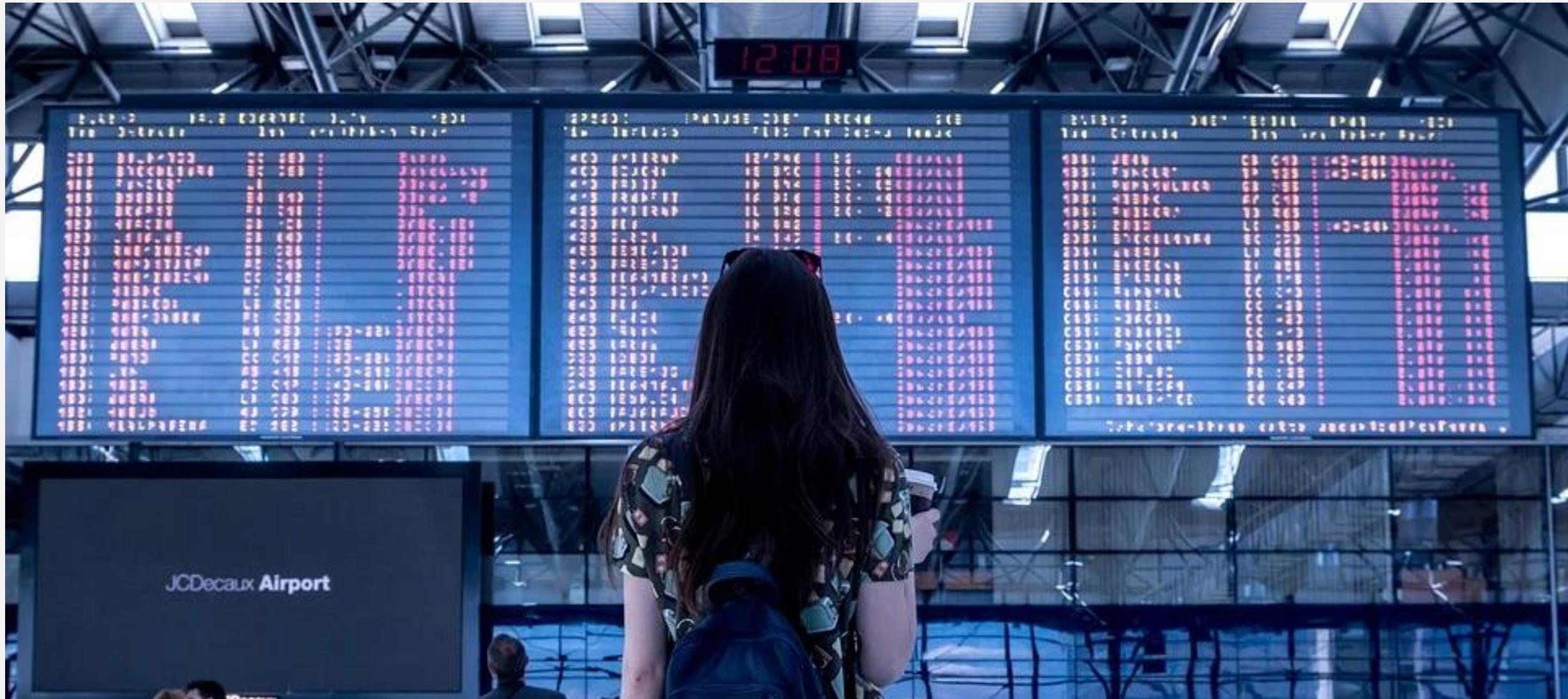
- each national park is visited exactly once
- the travelled distance is minimal

[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?

2.4 Outline

2 Search, Problem Solving, and Planning

2.1 Intelligent Agents

2.2 Solving Problems by Searching

2.3 Beyond Classical Search

2.4 Adversarial Search and Game Theory

2.5 Constraint Satisfaction Problems

► What we will learn:

- We define the concept of rational agents (\approx 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

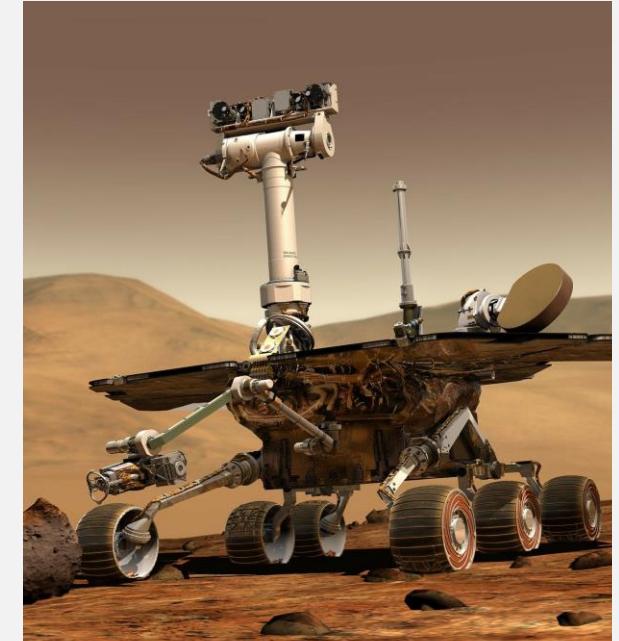


Image source: ↗ [Pixabay](#) (2019) / ↗ [CC0](#)

► Duration:

- 225 min

► Relevant for Exam:

- 2.1 – 2.5

2.4 “2/3 of the average” game

- Everyone writes down a number between 0 and 100
- Person closest to 2/3 of the average wins
- Example:
 - A says 50
 - B says 10
 - C says 90
 - Average(50, 10, 90) = 50
 - 2/3 of average = 33.33
 - A is closest ($|50-33.33| = 16.67$), so A wins

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

2.4 What is game theory?

- Game theory studies settings where multiple parties (**agents**) each have
 - different preferences (utility functions),
 - different actions that they can take
- Each agent's utility (potentially) depends on all agents' actions
 - What is optimal for one agent depends on what other agents do
 - Very circular!
- Game theory studies how agents can rationally form **beliefs** over what other agents will do, and (hence) how agents should **act**
 - Useful for acting as well as predicting behavior of others

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

2.4 Rock-paper-scissors

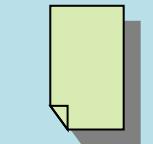
Zero-sum game: the utilities in each entry sum to 0 (or a constant)
Three-player game would be a 3D table with 3 utilities per entry, etc.

Row player aka.
player 1 chooses
a row

A row or column is called
an action or (pure)
strategy

Column player aka. player
2 (simultaneously)
chooses a column



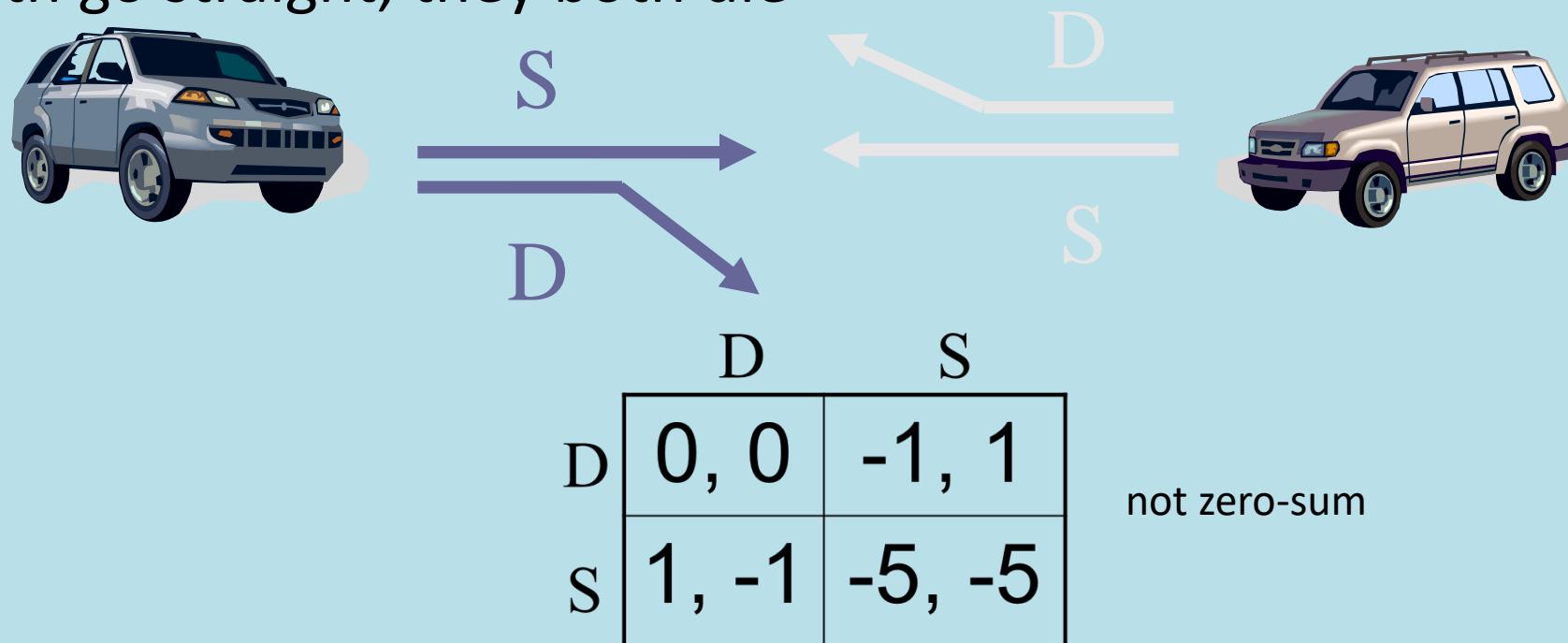
	0, 0	-1, 1	1, -1
	1, -1	0, 0	-1, 1
	-1, 1	1, -1	0, 0

Row player's utility is always listed first, column player's second

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

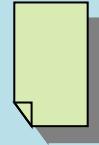
2.4 “Chicken”

- Two players drive cars towards each other
- If one player goes straight, that player wins
- If both go straight, they both die



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

2.4 Rock-paper-scissors – Seinfeld variant



MICKEY: All right, rock beats paper!
(Mickey smacks Kramer's hand for losing)
KRAMER: I thought paper covered rock.
MICKEY: Nah, rock flies right through paper.
KRAMER: What beats rock?
MICKEY: (looks at hand) Nothing beats rock.

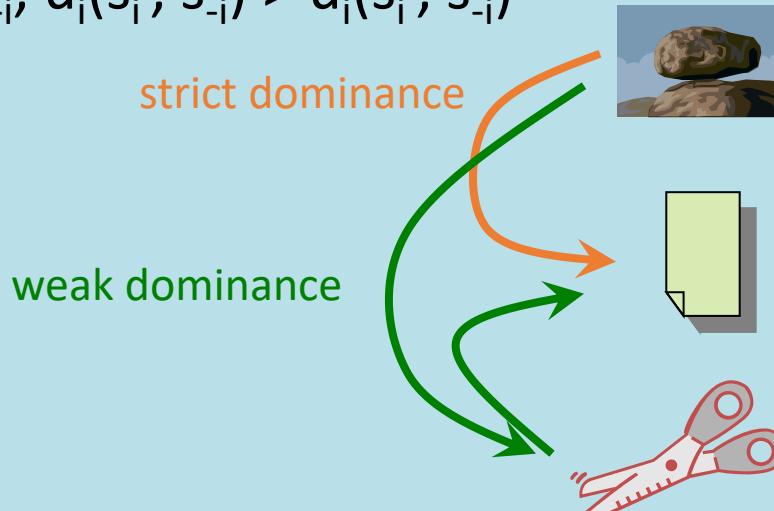
	0, 0	1, -1	1, -1
	-1, 1	0, 0	-1, 1
	-1, 1	1, -1	0, 0

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

2.4 Dominance

- Player i 's strategy s_i strictly dominates s'_i if
 - for any s_{-i} , $u_i(s_i, s_{-i}) > u_i(s'_i, s_{-i})$
- s_i weakly dominates s'_i if
 - for any s_{-i} , $u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i})$; and
 - for some s_{-i} , $u_i(s_i, s_{-i}) > u_i(s'_i, s_{-i})$

$-i = "the\ player(s)\ other\ than\ i"$



0, 0	1, -1	1, -1
-1, 1	0, 0	-1, 1
-1, 1	1, -1	0, 0

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

2.4 Prisoner's Dilemma

- Pair of criminals has been caught
- District attorney has evidence to convict them of a minor crime (1 year in jail); knows that they committed a major crime together (3 years in jail) but cannot prove it
- Offers them a deal:
 - If both confess to the major crime, they each get a 1 year reduction
 - If only one confesses, that one gets 3 years reduction

	confess	don't confess
confess	-2, -2	0, -3
don't confess	-3, 0	-1, -1

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

2.4 “Should I buy an SUV?”

purchasing cost



cost: 5



cost: 3

accident cost



cost: 5



cost: 8



cost: 5



cost: 2

cost: 5



cost: 5



-10, -10	-7, -11
-11, -7	-8, -8

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

2.4 Mixed strategies

- Mixed strategy for player i = probability distribution over player i's (pure) strategies

- E.g. $1/3$ $1/3$, $1/3$

- Example of dominance by a mixed strategy:

3, 0	0, 0
0, 0	3, 0
1, 0	1, 0

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

2.4 Checking for dominance by mixed strategies I

- Linear program for checking whether strategy s_i^* is **strictly** dominated by a mixed strategy:
- maximize ε
- such that:
 - for any s_{-i} , $\sum_{s_i} p_{s_i} u_i(s_i, s_{-i}) \geq u_i(s_i^*, s_{-i}) + \varepsilon$
 - $\sum_{s_i} p_{s_i} = 1$

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

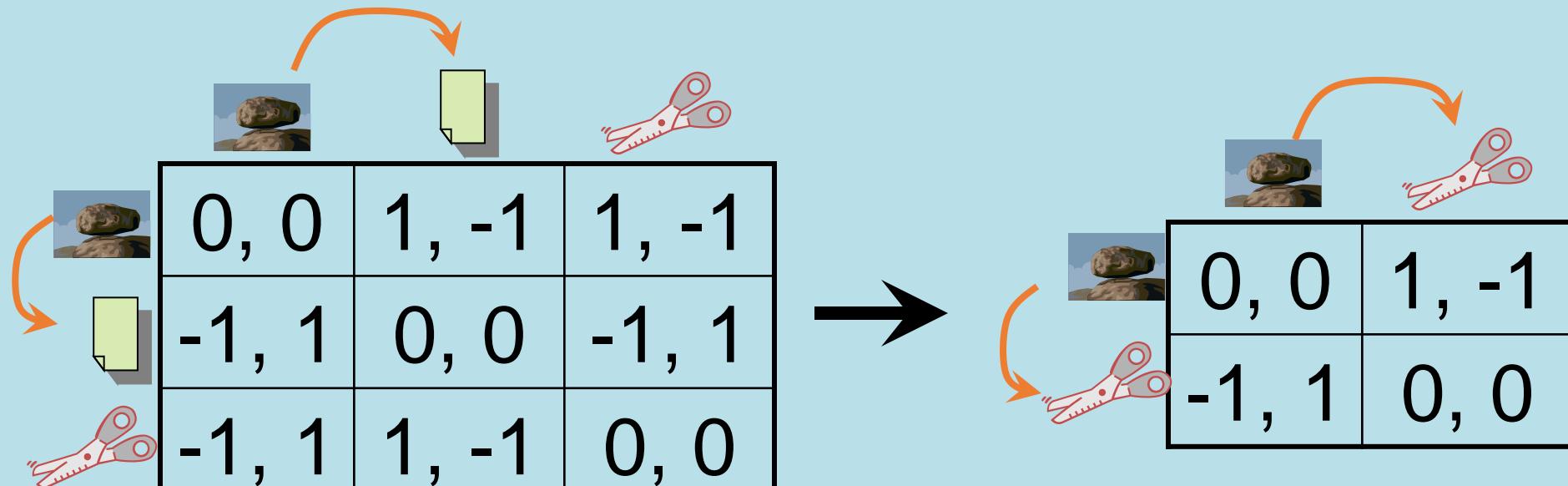
2.4 Checking for dominance by mixed strategies II

- Linear program for checking whether strategy s_i^* is **weakly** dominated by a mixed strategy:
- maximize $\sum_{s_{-i}} (\sum_{s_i} p_{s_i} u_i(s_i, s_{-i})) - u_i(s_i^*, s_{-i})$
- such that:
 - for any s_{-i} , $\sum_{s_i} p_{s_i} u_i(s_i, s_{-i}) \geq u_i(s_i^*, s_{-i})$
 - $\sum_{s_i} p_{s_i} = 1$

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

2.4 Iterated dominance

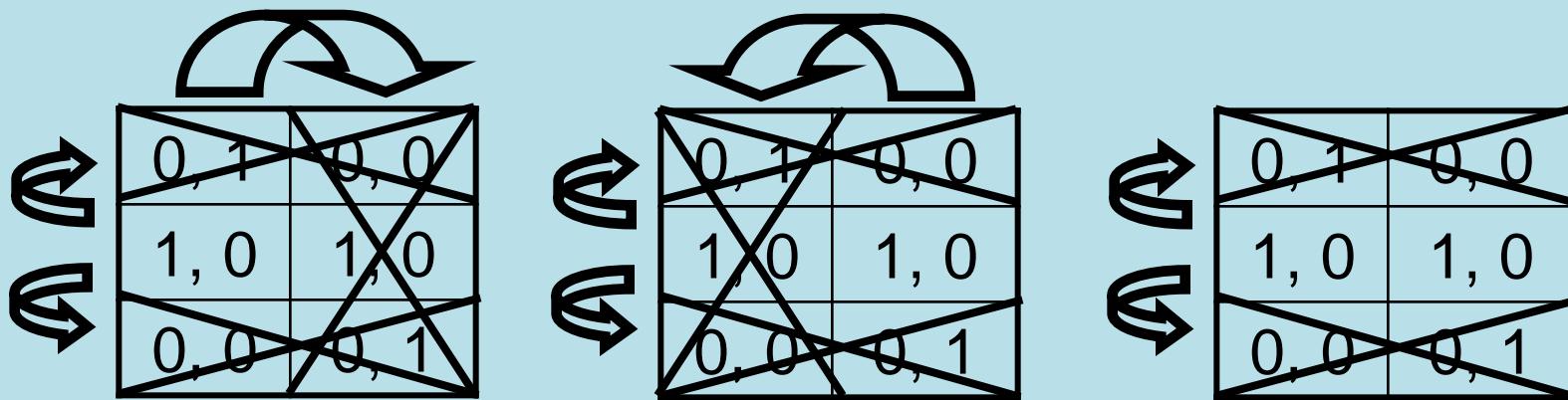
- Iterated dominance: remove (strictly/weakly) dominated strategy, repeat
- Iterated strict dominance on Seinfeld's RPS:



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

2.4 Path-Dependency

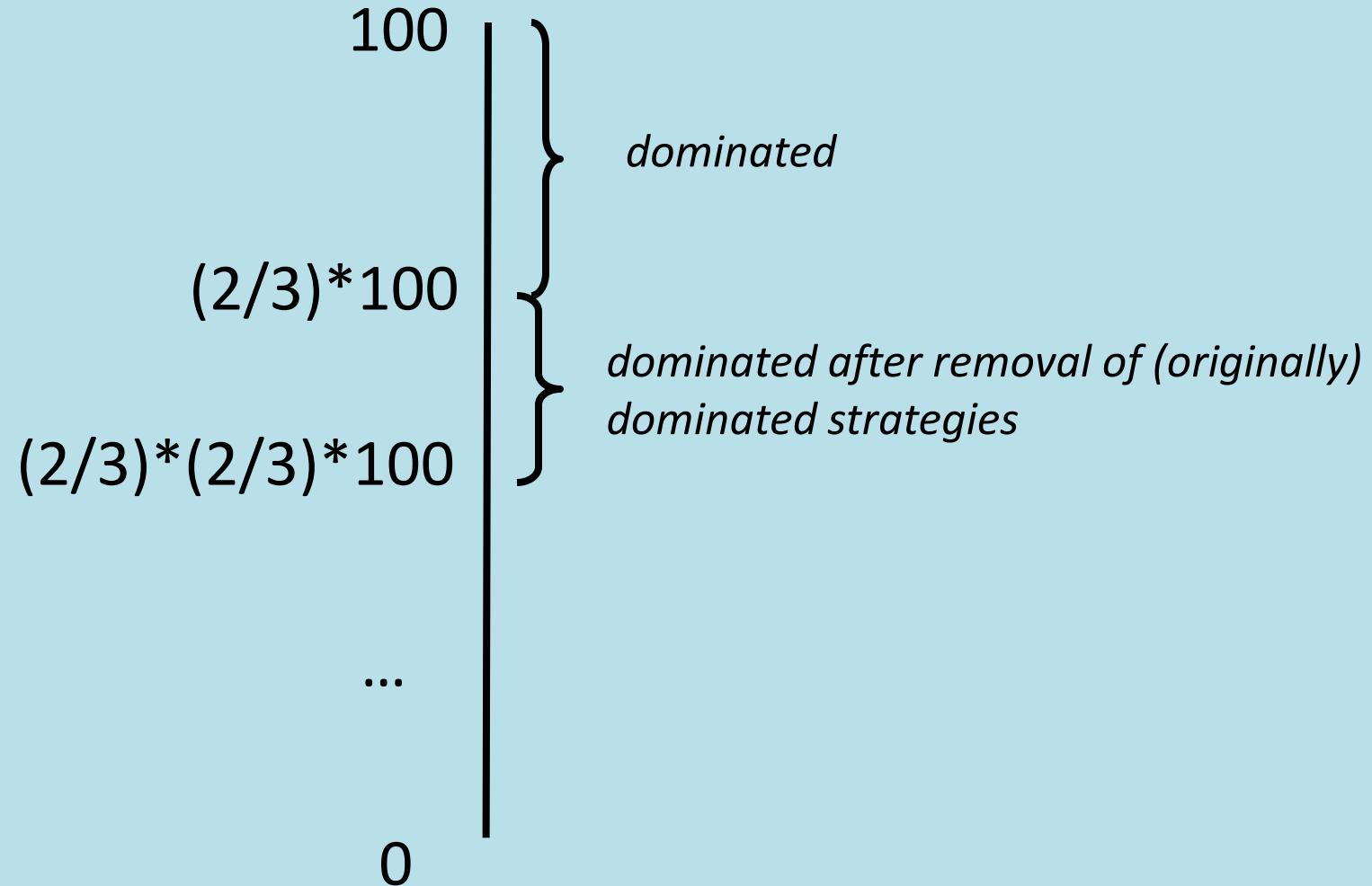
Iterated weak dominance is **path-dependent**: sequence of eliminations may determine which solution we get (if any)
(whether or not dominance by mixed strategies allowed)



Iterated strict dominance is **path-independent**: elimination process will always terminate at the same point
(whether or not dominance by mixed strategies allowed)

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

2.4 “2/3 of the average” game revisited



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

2.4 Game playing

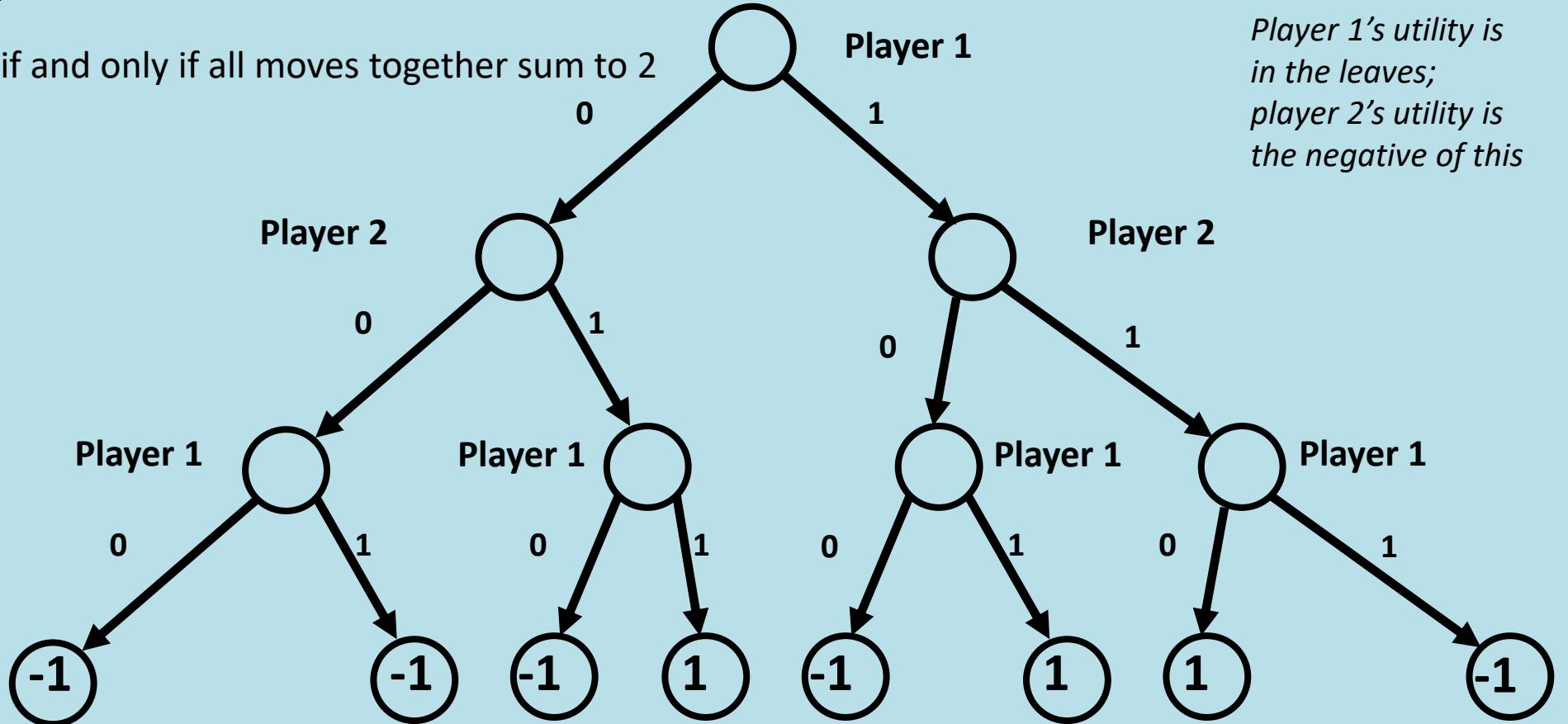
- Rich tradition of creating game-playing programs in AI
- Many similarities to search
- Most of the games studied
 - have two players,
 - are **zero-sum**: what one player wins, the other loses
 - have **perfect information**: the entire state of the game is known to both players at all times
- E.g., tic-tac-toe, checkers, chess, Go, backgammon, ...
- Will focus on these for now
- Recently more interest in other games
 - Esp. games without perfect information; e.g., poker
 - Need probability theory, game theory for such games

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

2.4 “Sum to 2” game

- Player 1 moves, then player 2, finally player 1 again
- Move = 0 or 1
- Player 1 wins if and only if all moves together sum to 2

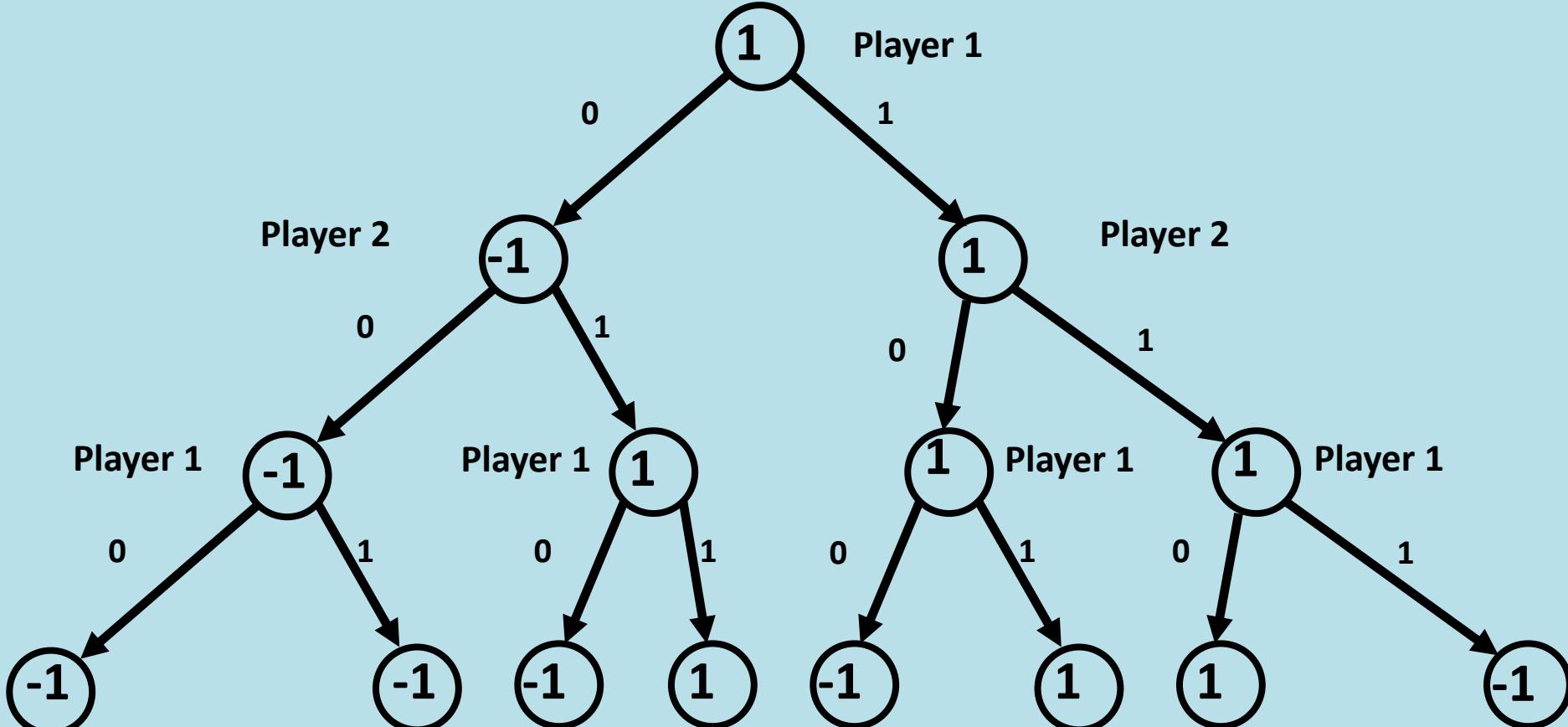
*Player 1’s utility is
in the leaves;
player 2’s utility is
the negative of this*



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

2.4 Backward induction (aka. minimax)

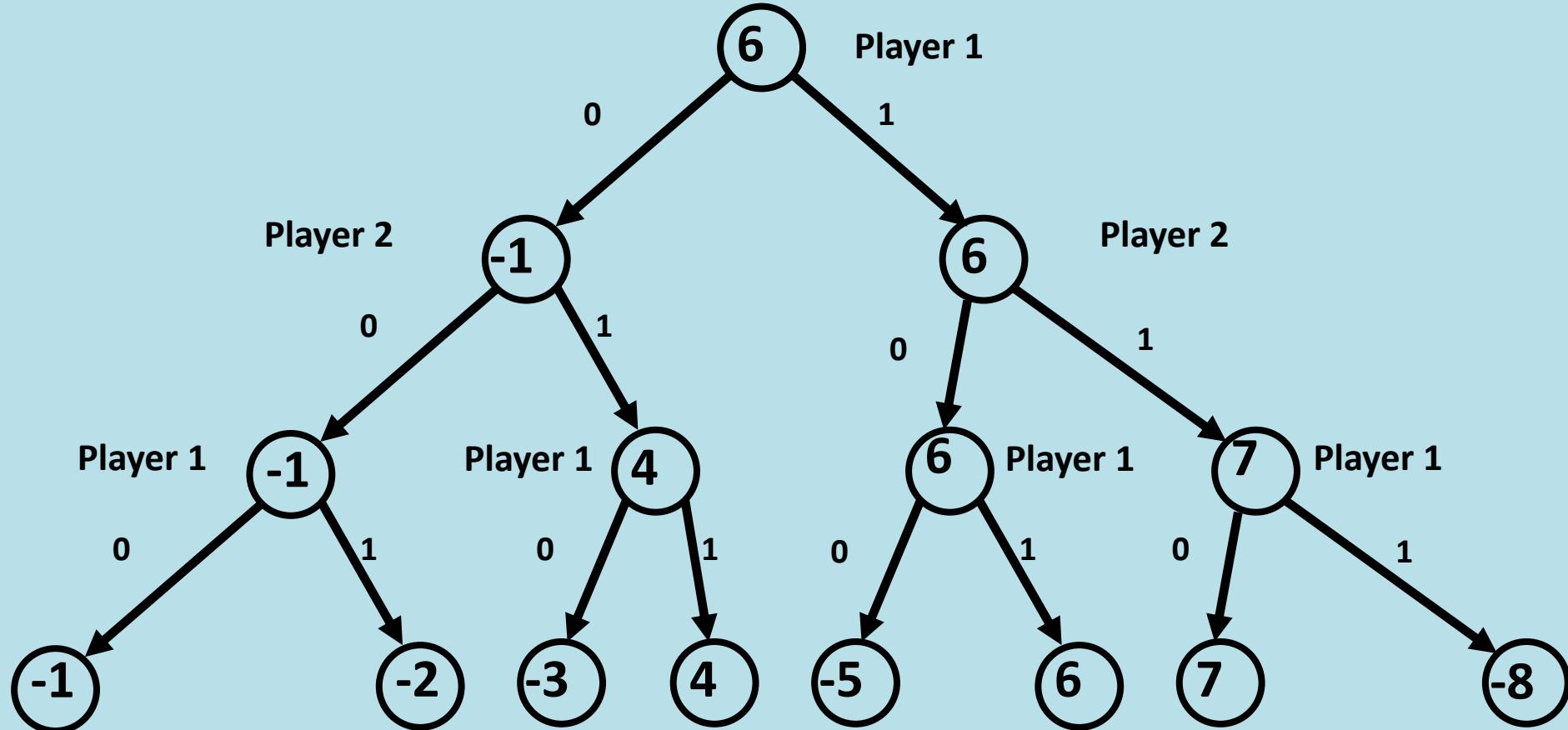
- From leaves upward, analyze best decision for player at node, give node a value
 - Once we know values, easy to find optimal action (choose best value)



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

2.4 Modified game

- From leaves upward, analyze best decision for player at node, give node a value



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

2.4 A recursive implementation

Value(state)

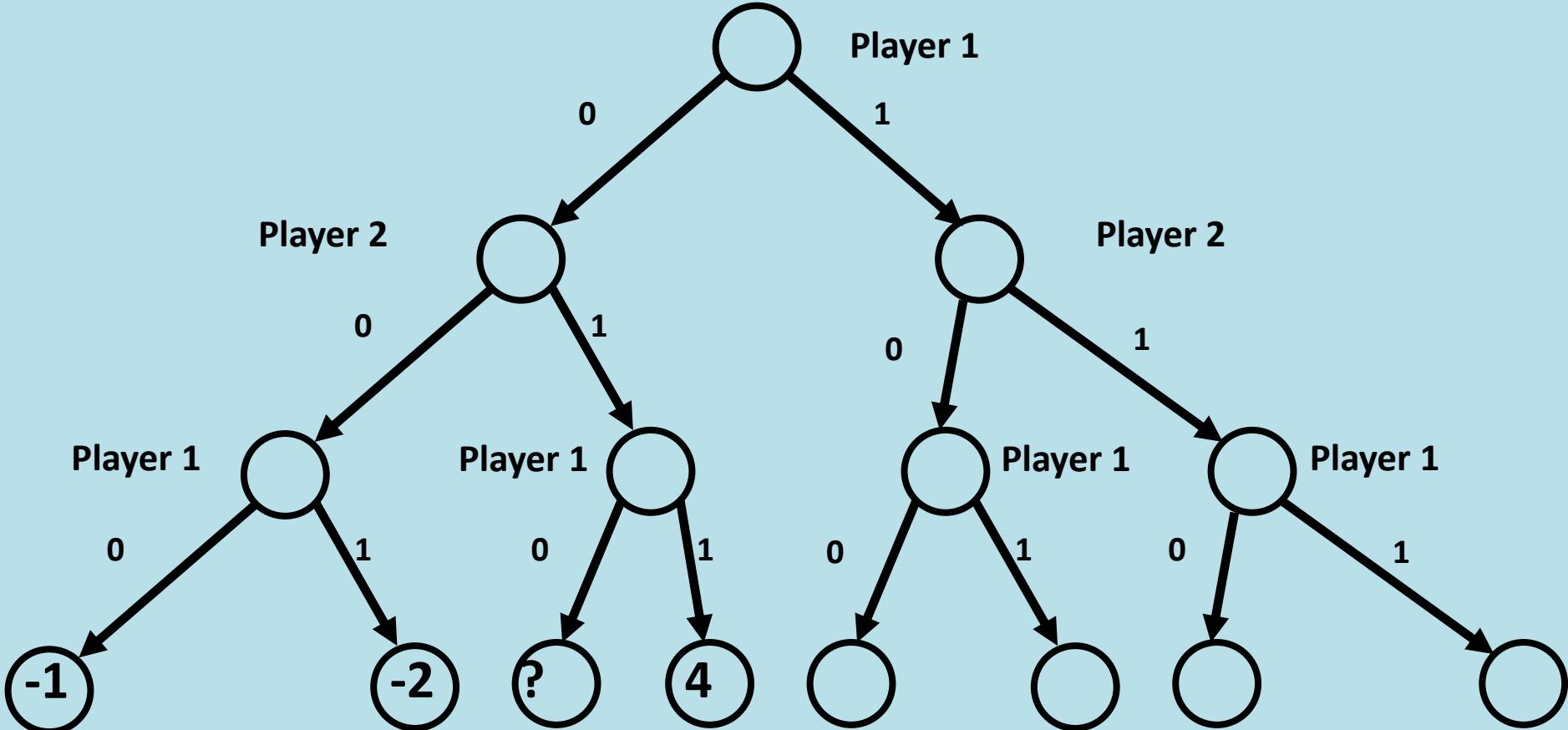
```
If state is terminal, return its value  
If (player(state) = player 1)  
    v := -infinity  
    For each action  
        v := max(v, Value(successor(state, action)))  
    Return v  
  
Else  
    v := infinity  
    For each action  
        v := min(v, Value(successor(state, action)))  
    Return v
```

Space? Time?

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

2.4 Do we need to see all the leaves?

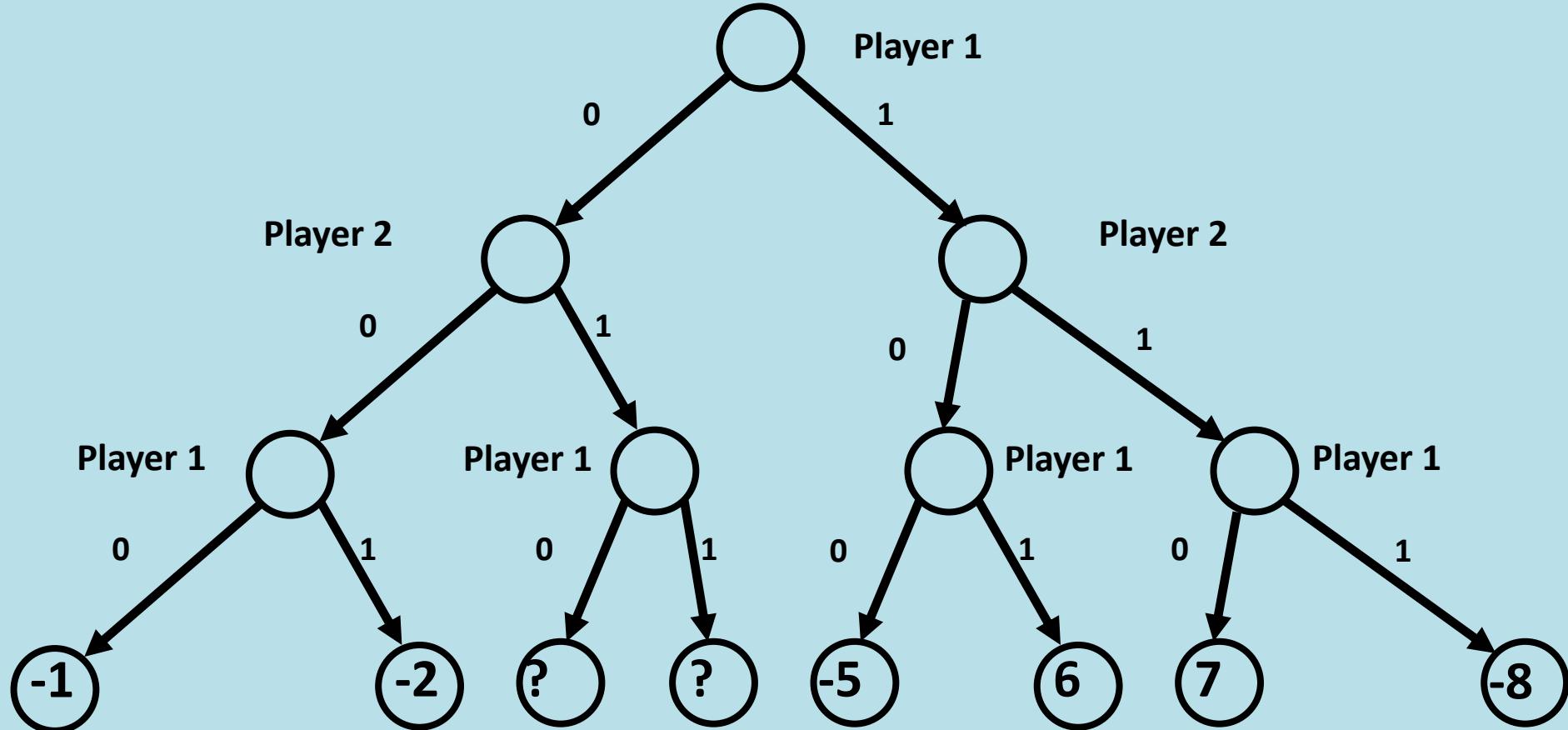
- Do we need to see the value of the question mark here?



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

2.4 Do we need to see all the leaves?

- Do we need to see the values of the question marks here?



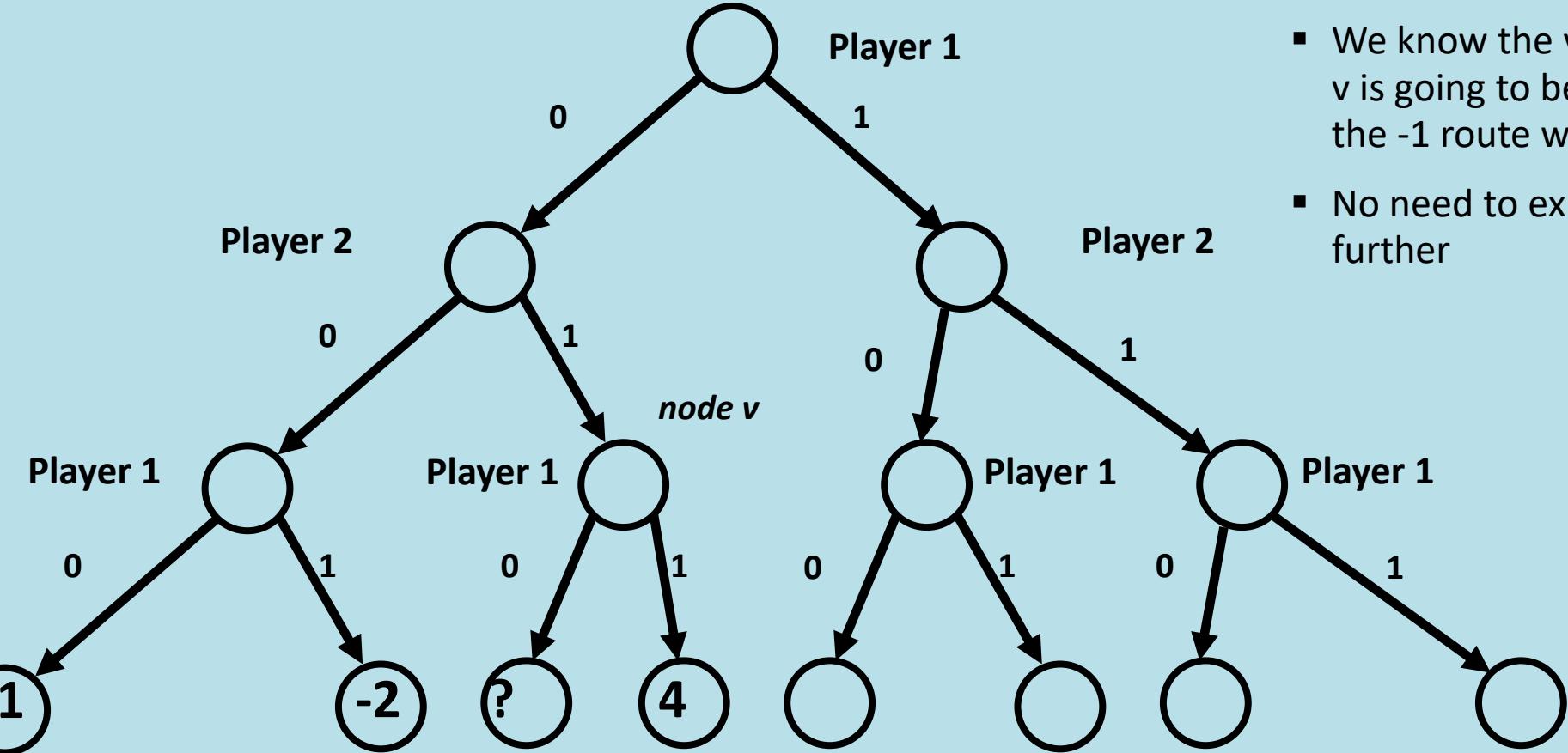
Adapted from Russell, S., & Norvig, P. (2016);

2.4 Alpha-beta pruning

- **Pruning** = cutting off parts of the search tree (because you realize you don't need to look at them)
 - When we considered A* we also pruned large parts of the search tree
- **Maintain alpha** = value of the best option for player 1 along the path so far
- **Beta** = value of the best option for player 2 along the path so far

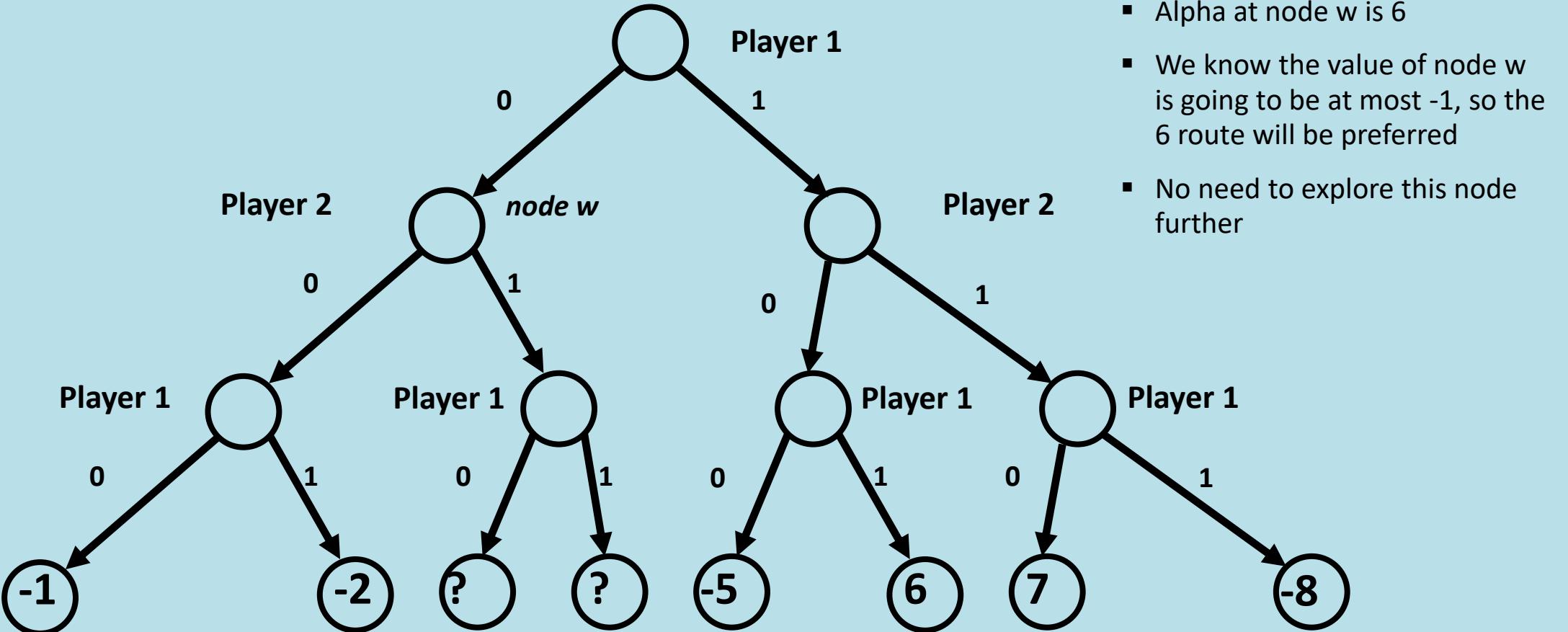
Adapted from Russell, S., & Norvig, P. (2016);

2.4 Pruning on beta



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

2.4 Pruning on alpha



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

2.4 Modifying recursive implementation to do alpha-beta pruning

```
Value(state, alpha, beta)
    If state is terminal, return its value
    If (player(state) = player 1)
        v := -infinity
        For each action
            v := max(v, Value(successor(state, action), alpha, beta))
            If v >= beta, return v
            alpha := max(alpha, v)
        Return v
    Else
        v := infinity
        For each action
            v := min(v, Value(successor(state, action), alpha, beta))
            If v <= alpha, return v
            beta := min(beta, v)
        Return v
```

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

2.4 Benefits of alpha-beta pruning

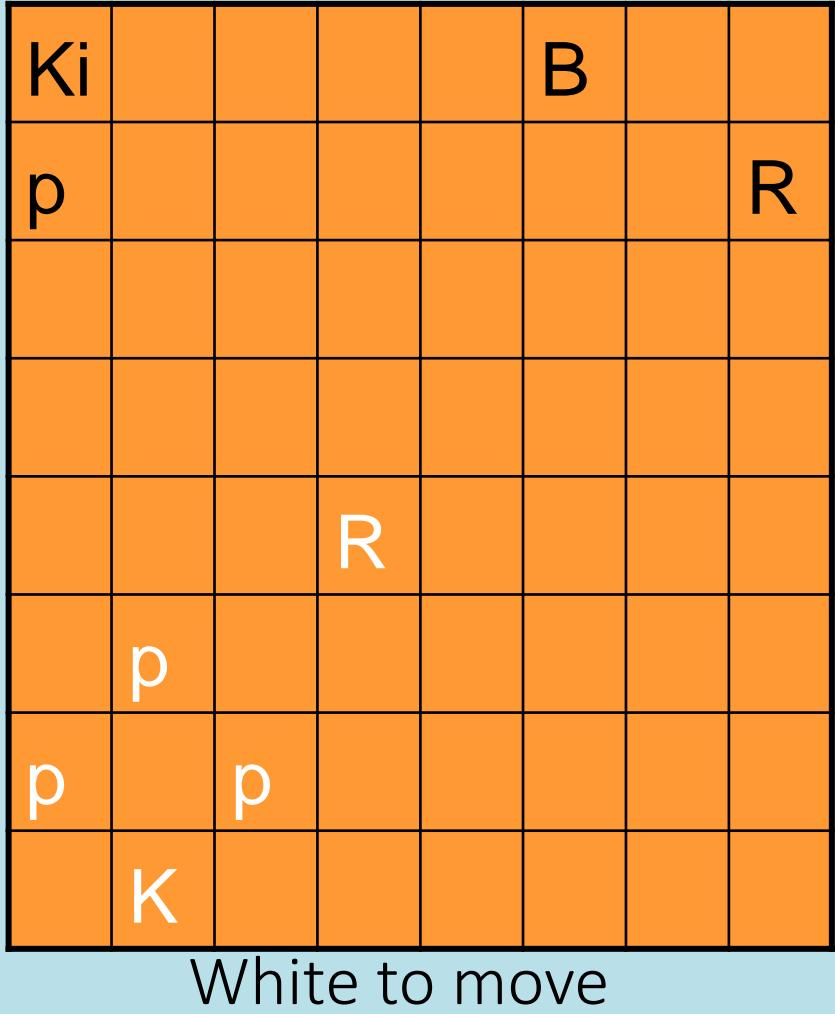
- Without pruning, need to examine $O(b^m)$ nodes
- With pruning, depends on which nodes we consider first
- If we choose a random successor, need to examine $O(b^{3m/4})$ nodes
- If we manage to choose the best successor first, need to examine $O(b^{m/2})$ nodes
 - Practical heuristics for choosing next successor to consider get quite close to this
- Can effectively look twice as deep!
 - Difference between reasonable and expert play

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

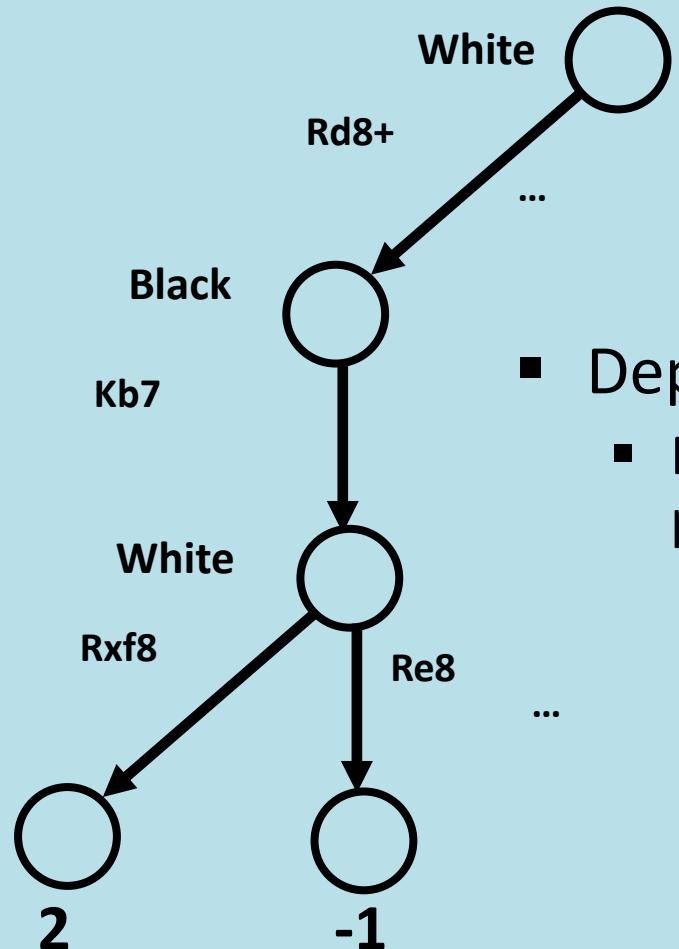
- As in search, multiple sequences of moves may lead to the same state
- Again, can keep track of previously seen states (usually called a **transposition table** in this context)
 - May not want to keep track of **all** previously seen states...

- Most games are too big to solve even with alpha-beta pruning
- Solution: Only look ahead to **limited depth** (nonterminal nodes)
- Evaluate nodes at depth cutoff by a heuristic (aka. **evaluation function**)
- E.g., chess:
 - Material value: queen worth 9 points, rook 5, bishop 3, knight 3, pawn 1
 - Heuristic: difference between players' material values

2.4 Chess example

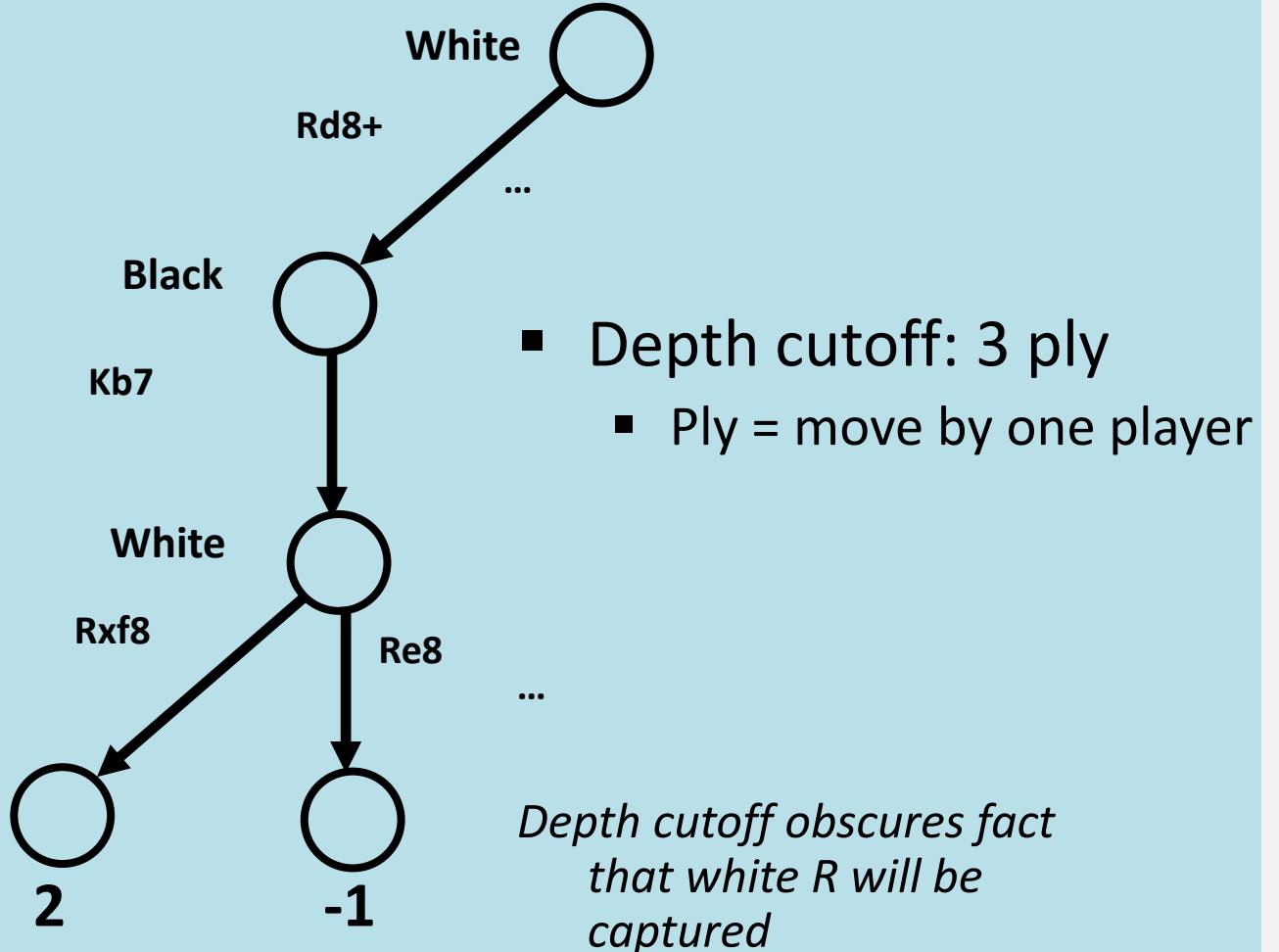
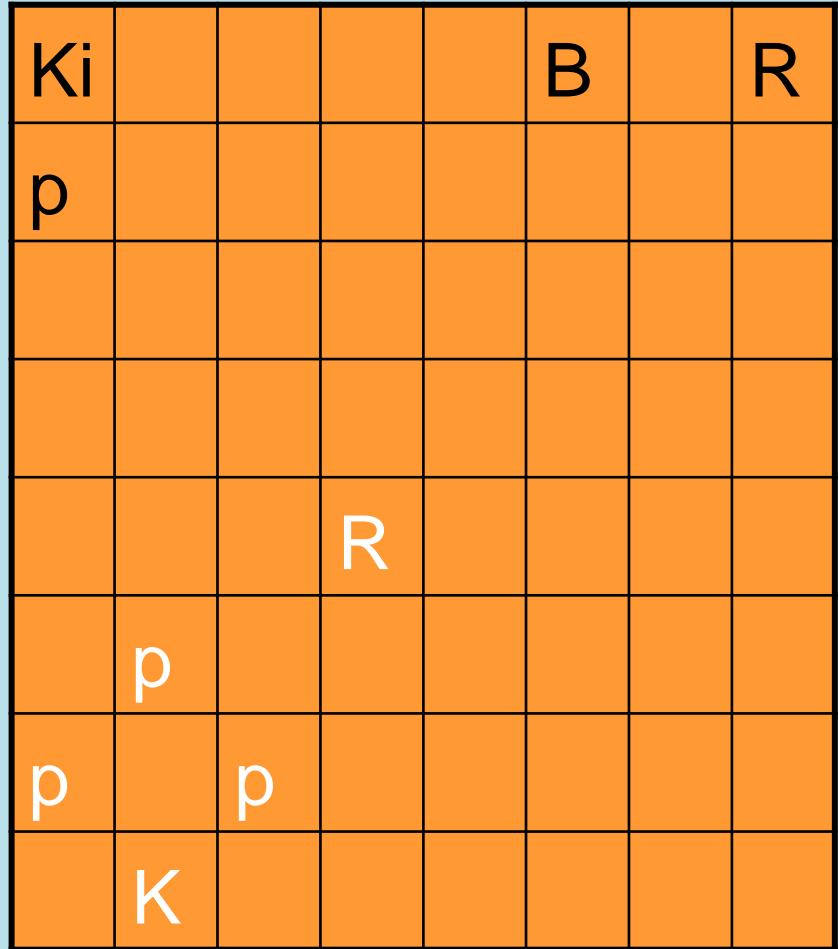


Adapted from Russell, S., & Norvig, P. (2016);



- Depth cutoff: 3 ply
 - Ply = move by one player

2.4 Chess (bad) example



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

2.4 Addressing this problem

- Try to evaluate whether nodes are **quiescent**
 - Quiescent = evaluation function will not change rapidly in near future
 - Only apply evaluation function to quiescent nodes
- If there is an “obvious” move at a state, apply it before applying evaluation function

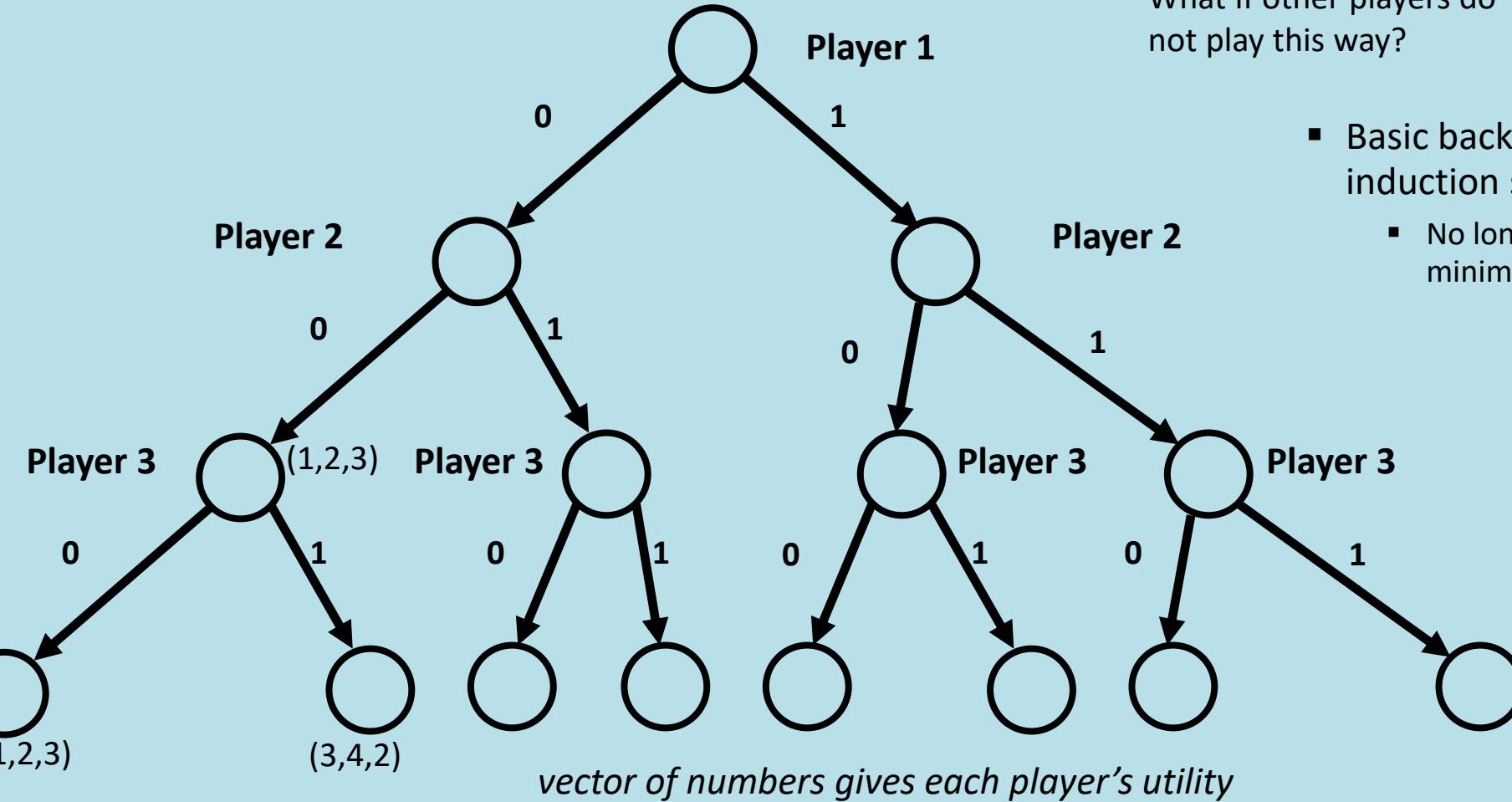
Adapted from Russell, S., & Norvig, P. (2016);

2.4 Playing against suboptimal players

- Minimax is optimal against other minimax players
- What about against players that play in some other way?

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

2.4 Many-player, general-sum games of perfect information



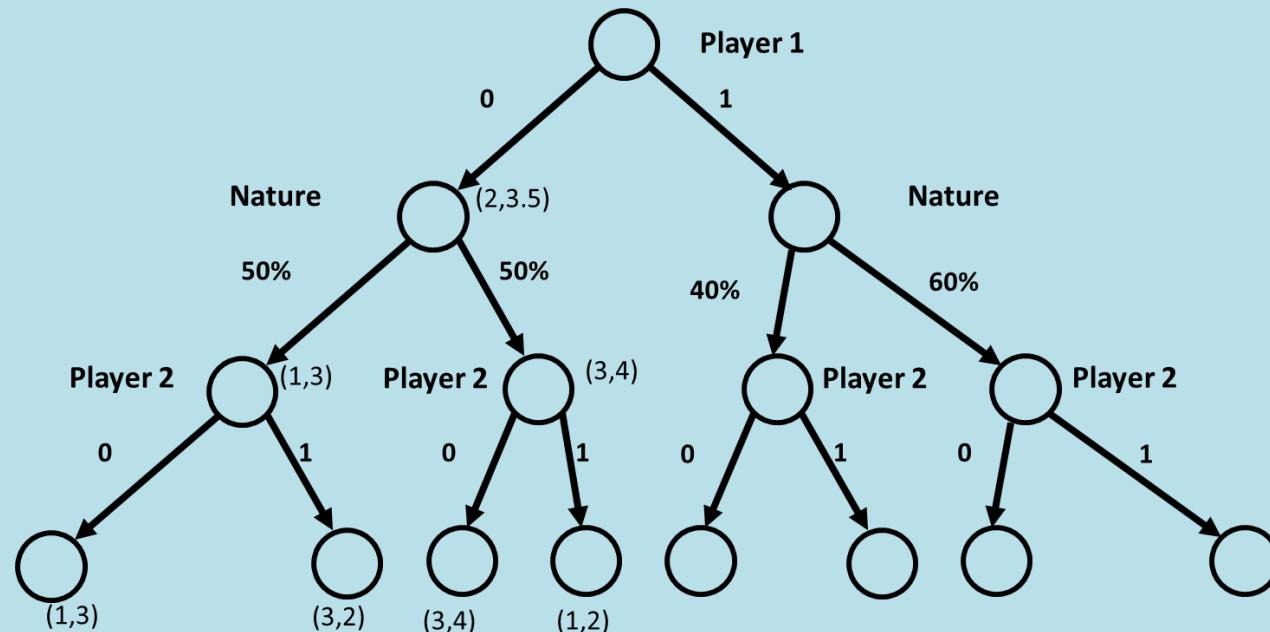
What if other players do not play this way?

- Basic backward induction still works
 - No longer called minimax

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

2.4 Games with random moves by “Nature”

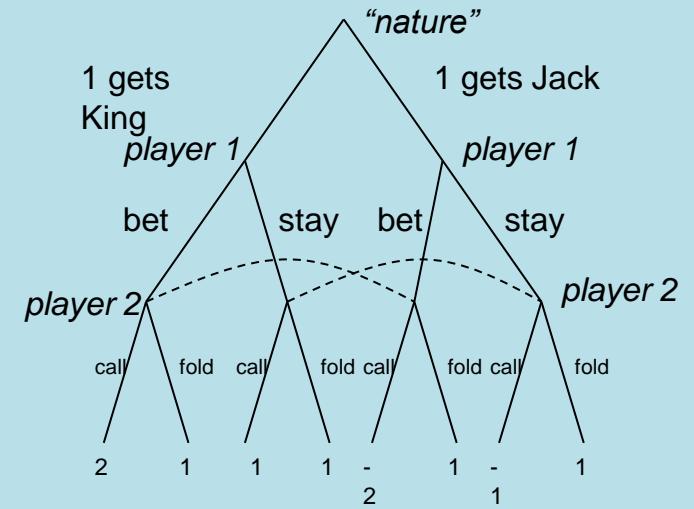
- E.g., games with dice (Nature chooses dice roll)
- Backward induction still works...
 - Evaluation functions now need to be **cardinally** right (not just **ordinally**)
 - For two-player zero-sum games with random moves, can we generalize alpha-beta? How?



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

2.4 Games with imperfect information

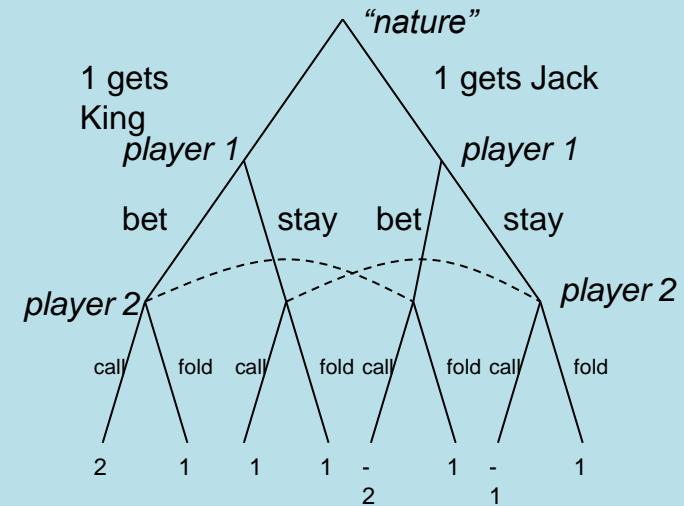
- Players cannot necessarily see the whole current state of the game
 - Card games
- Ridiculously simple poker game:
 - Player 1 receives King (winning) or Jack (losing),
 - Player 1 can bet or stay,
 - Player 2 can call or fold
- Dashed lines indicate indistinguishable states
- Backward induction does **not** work, need random strategies for optimality! (more later in course)



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

2.4 Intuition for need of random strategies

- Suppose my strategy is “bet on King, stay on Jack”
 - What will you do?
 - What is your expected utility?
- What if my strategy is “always bet”?
- What if my strategy is “always bet when given King, 10% of the time bet when given Jack”?



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

2.4 The state of the art for some games Until (2010)

- Chess:
 - 1997: IBM Deep Blue defeats Kasparov
 - ... there is still debate about whether computers are really better
- Checkers:
 - Computer world champion since 1994
 - ... there was still debate about whether computers are really better...
 - until 2007: checkers solved **optimally** by computer
- Go:
 - Computers still not very good
 - Branching factor really high
 - Some recent progress
- Poker:
 - Competitive with top humans in some 2-player games
 - 3+ player case much less well-understood

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

2.4 Is this of any value to society?

- Some of the techniques developed for games have found applications in other domains
 - Especially “adversarial” settings
- Real-world strategic situations are usually not two-player, perfect-information, zero-sum, ...
- But **game theory** does not need any of those
- Example application: security scheduling at airports

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

Your turn!

Task

Please explain

- The Seinfeld variant of rock-paper-scissors.
- how minimax algorithms are used in decision-making for two-player zero-sum games, and describe one limitation of the basic minimax approach.

Outline

2 Search, Problem Solving, and Planning

2.1 Intelligent Agents

2.2 Solving Problems by Searching

2.3 Beyond Classical Search

2.4 Adversarial Search and Game Theory

2.5 Constraint Satisfaction Problems

► What we will learn:

- We define the concept of rational agents (\approx 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

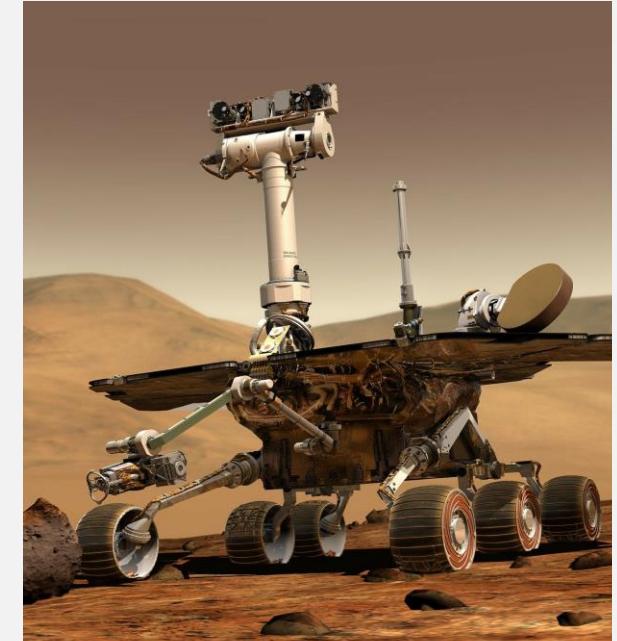


Image source: ↗ [Pixabay](#) (2019) / ↗ [CC0](#)

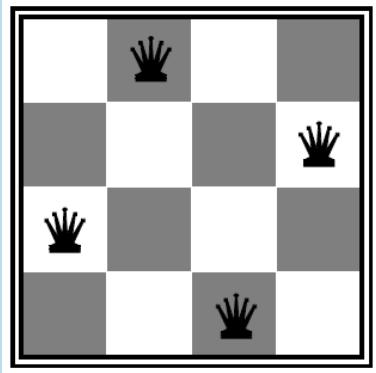
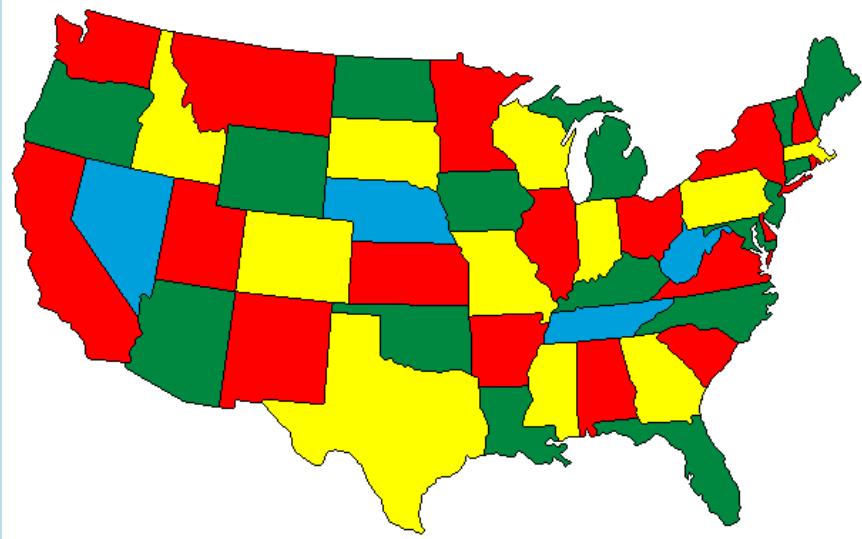
► Duration:

- 225 min

► Relevant for Exam:

- 2.1 – 2.5

2.5 Constraint Satisfaction Problems



8		4	6		7
	1			6	5
5	9	3	7	8	
		7			
4	8	2	1	3	
	5	2			9
		1			
3		9	2		5

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

2.5 Constraint satisfaction problems (CSPs)

- Standard search problem:
 - **State** is a “black box” – any data structure that supports successor function, heuristic function, and goal test
- Constraint satisfaction problem:
 - **State** is defined by **variables** X_i with **values** from **domain** D_i
 - **Goal test** is a set of **constraints** specifying allowable combinations of values for subsets of variables
- A simple example of a formal representation language
- Allows useful **general-purpose** algorithms with more power than standard search algorithms

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

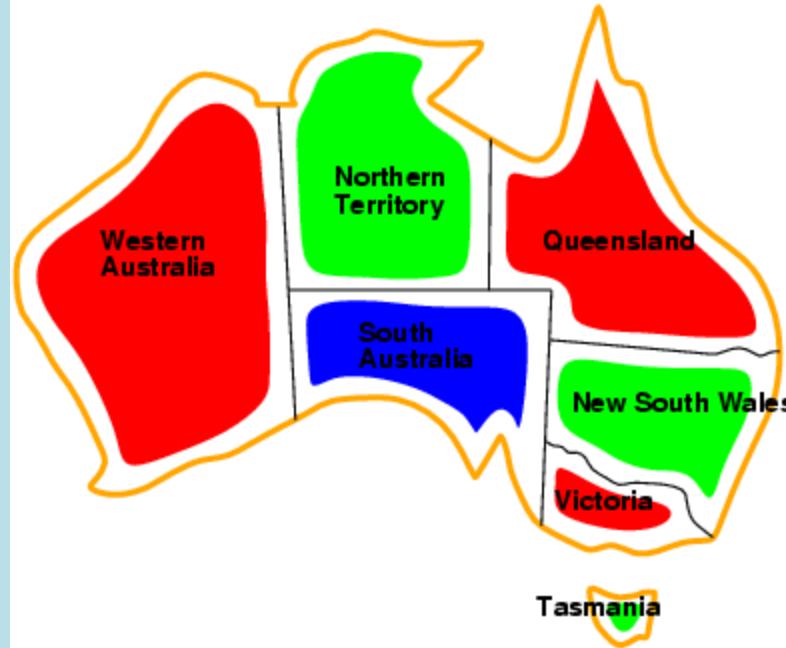
2.5 Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** {red, green, blue}
- **Constraints:** adjacent regions must have different colors
e.g., $WA \neq NT$, or $(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$



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

2.5 Example: Map Coloring



- **Solutions** are *complete* and *consistent* assignments, e.g.,
WA = **red**, NT = **green**, Q = **red**, NSW = **green**,
V = **red**, SA = **blue**, T = **green**

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

2.5 Example: N-Queens

- **Variables:** X_{ij}

- **Domains:** {0, 1}

- **Constraints:**

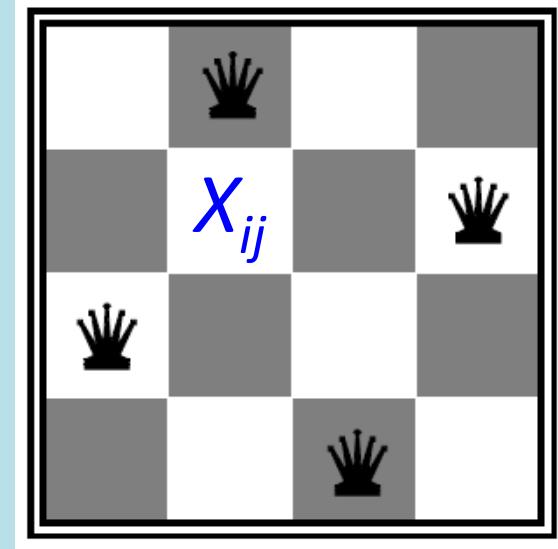
$$\sum_{i,j} X_{ij} = N$$

$$(X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\}$$

$$(X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\}$$

$$(X_{ij}, X_{i+k, j+k}) \in \{(0, 0), (0, 1), (1, 0)\}$$

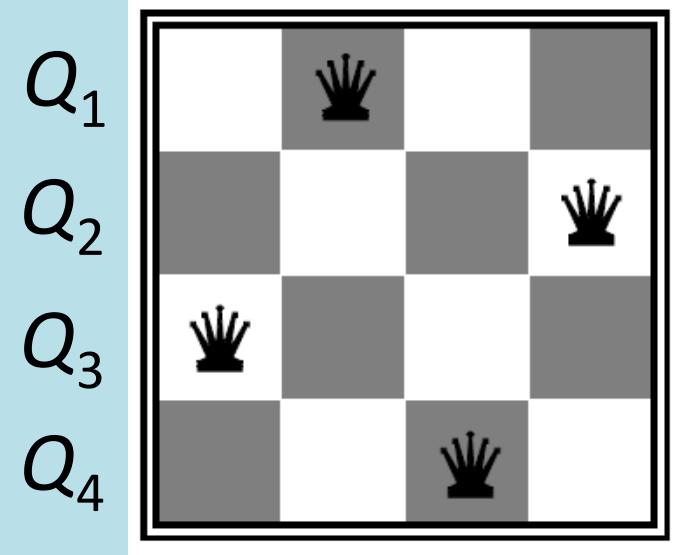
$$(X_{ij}, X_{i+k, j-k}) \in \{(0, 0), (0, 1), (1, 0)\}$$



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

2.5 N-Queens: Alternative formulation

- **Variables:** Q_i
- **Domains:** $\{1, \dots, N\}$
- **Constraints:**
 $\forall i, j \text{ non-threatening } (Q_i, Q_j)$



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

2.5 Example: Cryptarithmetic

- **Variables:** T, W, O, F, U, R

X_1, X_2

- **Domains:** {0, 1, 2, ..., 9}

- **Constraints:**

Alldiff(T, W, O, F, U, R)

$$O + O = R + 10 * X_1$$

$$W + W + X_1 = U + 10 * X_2$$

$$T + T + X_2 = O + 10 * F$$

$$T \neq 0, F \neq 0$$

$$\begin{array}{r} X_2 \ X_1 \\ T \ W \ O \\ + \ T \ W \ O \\ \hline F \ O \ U \ R \end{array}$$

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

2.5 Example: Sudoku

- **Variables:** X_{ij}
- **Domains:** {1, 2, ..., 9}
- **Constraints:**
Alldiff(X_{ij} in the same unit)

A 9x9 Sudoku grid. Some cells contain numbers: Row 1: 8, 4, blank, 1, 6, blank, 8, blank, 4. Row 2: blank, 5, blank, blank, 1. Row 3: 1, 3, 8, blank, 9. Row 4: 6, 8, blank, blank, 4, 3. Row 5: blank, 2, blank, 9, 5, 1. Row 6: 7, blank, blank, 2. Row 7: 2, 7, 8, blank, 2, 6. Row 8: 2, blank, 3, blank, blank. Row 9: blank, blank, blank, blank, blank. A blue 'X' is placed over the cell at row 7, column 1, labeled X_{ij} .

						8		4
	8	4		1	6			
			5			1		
1		3	8			9		
6		8			4	3		
		2		9	5	1		
	7			2				
2		7	8		2	6		
2		3						

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

- Assignment problems
 - e.g., who teaches what class
- Timetable problems
 - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling

- More examples of CSPs: <http://www.csplib.org/>

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

2.5 Standard search formulation (incremental)

- **States:**
 - Values assigned so far
- **Initial state:**
 - The empty assignment { }
- **Successor function:**
 - Choose any unassigned variable and assign to it a value that does not violate any constraints
 - Fail if no legal assignments
- **Goal test:**
 - The current assignment is complete and satisfies all constraints

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

2.5 Standard search formulation (incremental)

- What is the depth of any solution?
 - n (with n variables assigned)
 - This is the good news (why?)
- Given that there are m possible values for any variable, how many paths are there in the search tree?
 - $n! \cdot m^n$
 - This is the bad news
- How can we reduce the branching factor?

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

2.5 Backtracking search

- In CSP's, variable assignments are **commutative**
 - For example, $[WA = \text{red} \text{ then } NT = \text{green}]$ is the same as $[NT = \text{green} \text{ then } WA = \text{red}]$
- We only need to consider assignments to a single variable at each level (i.e., we fix the order of assignments)
 - Then there are only m^n leaves
- Depth-first search for CSPs with single-variable assignments is called **backtracking search**

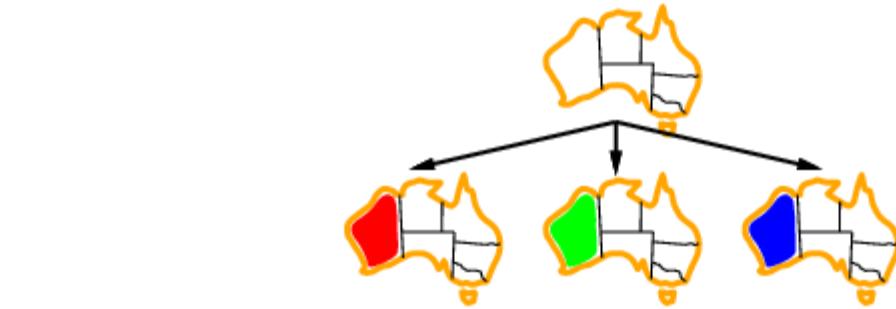
Adapted from Russell, S., & Norvig, P. (2016);

2.5 Example



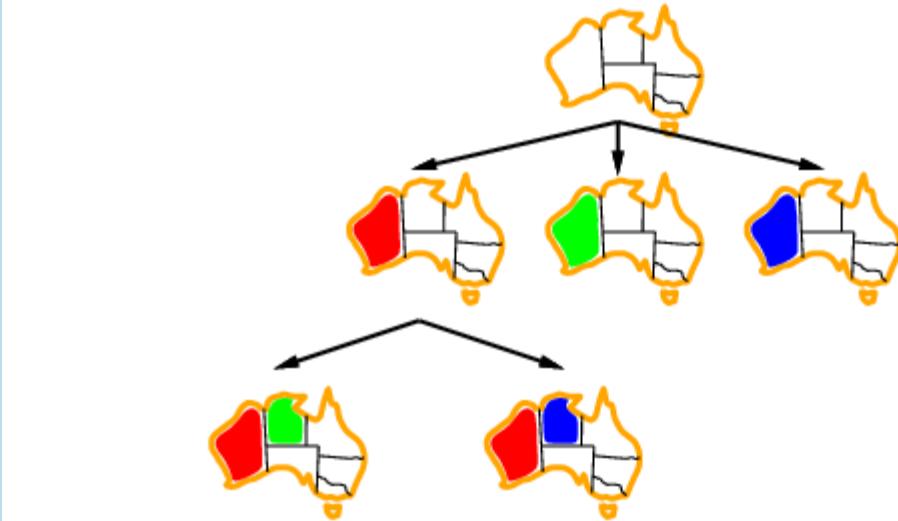
Adapted from Russell, S., & Norvig, P. (2016);

2.5 Example



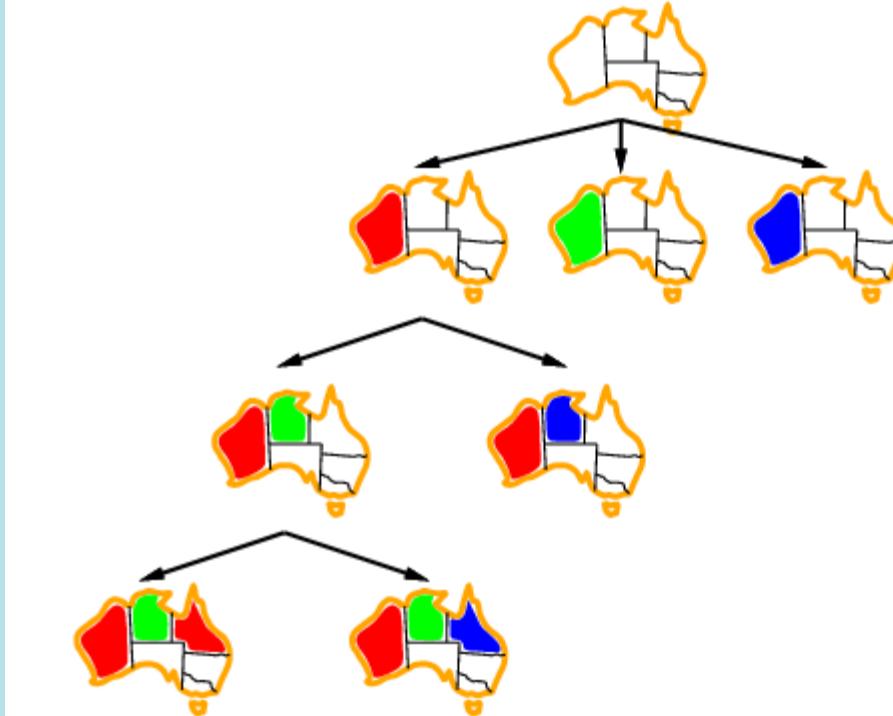
Adapted from Russell, S., & Norvig, P. (2016);

2.5 Example



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

2.5 Example



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

2.5 Backtracking search algorithm

```
function RECURSIVE-BACKTRACKING(assignment, csp)
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp)
        if value is consistent with assignment given CONSTRAINTS[csp]
            add {var = value} to assignment
            result  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
            if result  $\neq$  failure then return result
            remove {var = value} from assignment
    return failure
```

- Improving backtracking efficiency:
 - Which variable should be assigned next?
 - In what order should its values be tried?
 - Can we detect inevitable failure early?

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

2.5 Which variable should be assigned next?

- **Most constrained variable:**

- Choose the variable with the fewest legal values
- A.k.a. **minimum remaining values (MRV)** heuristic

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

2.5 Which variable should be assigned next?

- **Most constrained variable:**

- Choose the variable with the fewest legal values
- A.k.a. **minimum remaining values (MRV)** heuristic



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

- **Most constraining variable:**

- Choose the variable that imposes the most constraints on the remaining variables
- Tie-breaker among most constrained variables

2.5 Which variable should be assigned next?

■ Most constraining variable:

- Choose the variable that imposes the most constraints on the remaining variables
- Tie-breaker among most constrained variables



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

2.5 Given a variable, in which order should its values be tried?

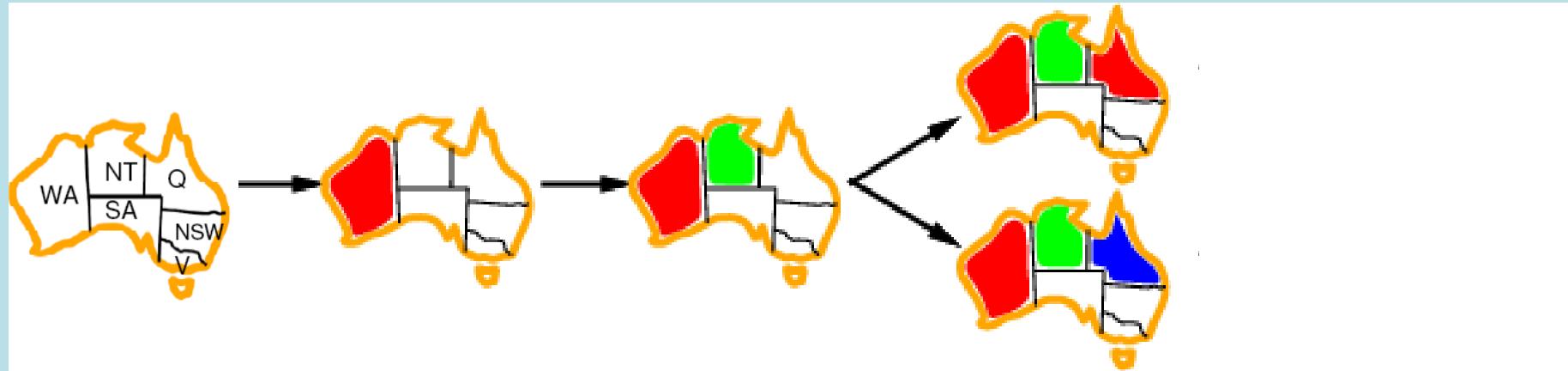
- Choose the **least constraining value**:
 - The value that rules out the fewest values in the remaining variables

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

2.5 Given a variable, in which order should its values be tried?

- Choose the **least constraining value**:
 - The value that rules out the fewest values in the remaining variables

Which assignment
for Q should we
choose?



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

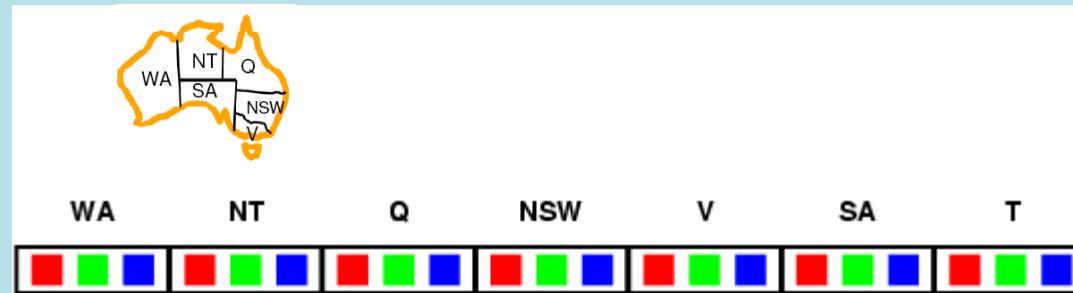
2.5 Early detection of failure: Forward checking

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values

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

2.5 Early detection of failure: Forward checking

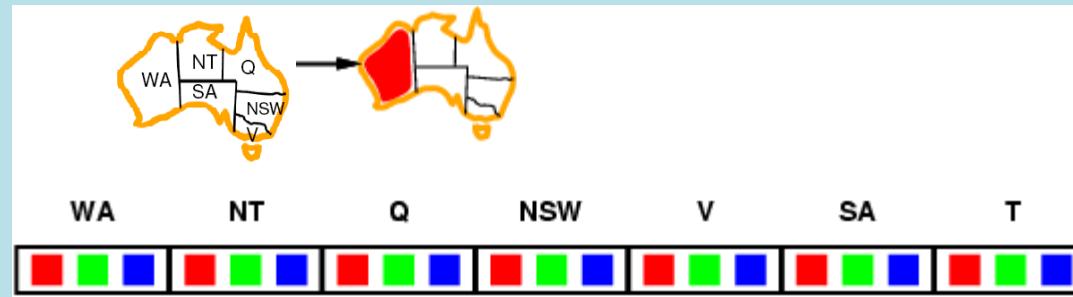
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



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

2.5 Early detection of failure: Forward checking

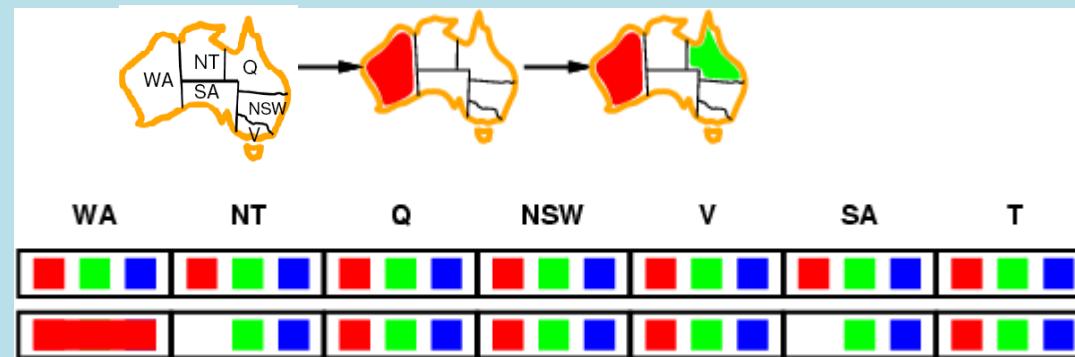
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



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

2.5 Early detection of failure: Forward checking

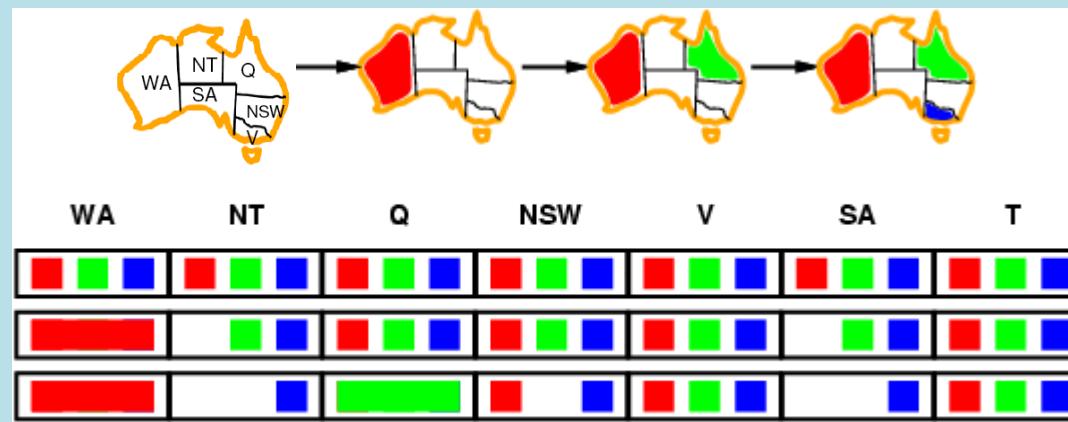
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



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

2.5 Early detection of failure: Forward checking

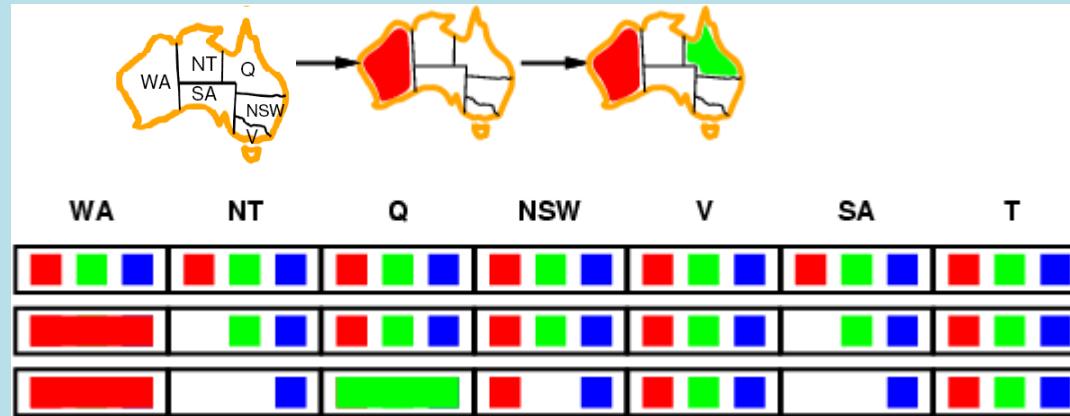
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



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

2.5 Constraint propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures

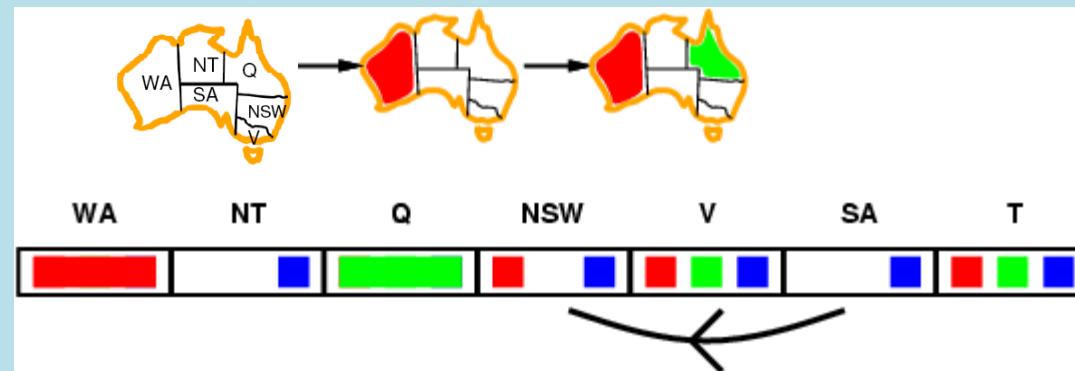


- NT and SA cannot both be blue!
- **Constraint propagation** repeatedly enforces constraints locally

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

2.5 Arc consistency

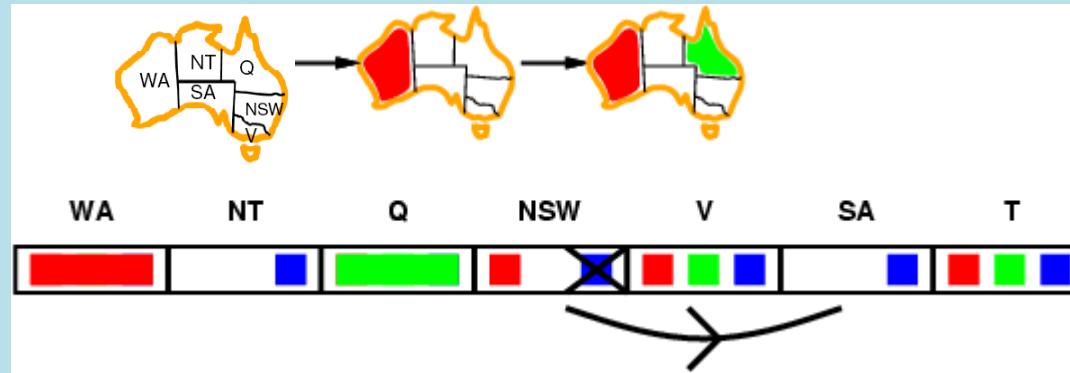
- Simplest form of propagation makes each pair of variables **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y



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

2.5 Arc consistency

- Simplest form of propagation makes each pair of variables **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y

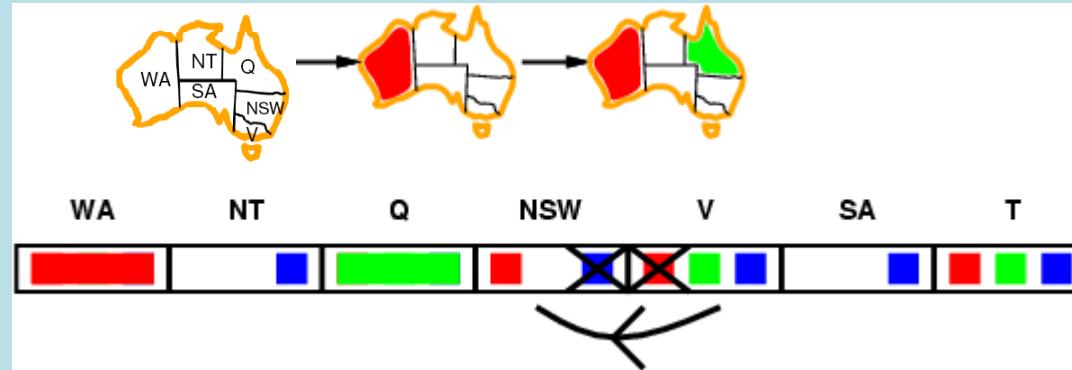


- If X loses a value, all pairs $Z \rightarrow X$ need to be rechecked

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

2.5 Arc consistency

- Simplest form of propagation makes each pair of variables **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y

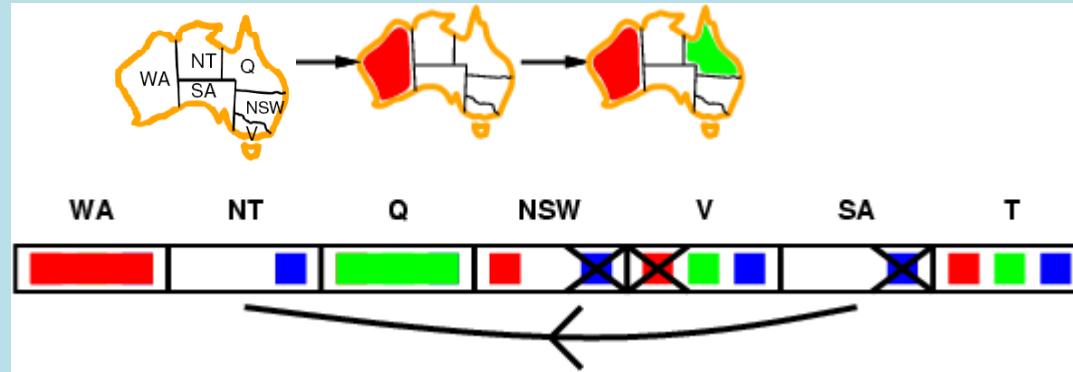


- If X loses a value, all pairs $Z \rightarrow X$ need to be rechecked

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

2.5 Arc consistency

- Simplest form of propagation makes each pair of variables **consistent**:
 - $X \rightarrow Y$ is consistent iff for **every** value of X there is **some** allowed value of Y



- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

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

2.5 Arc consistency algorithm AC-3

function $\text{AC-3}(csp)$ **returns** the CSP, possibly with reduced domains

inputs: csp , a binary CSP with variables $\{X_1, X_2, \dots, X_n\}$

local variables: $queue$, a queue of arcs, initially all the arcs in csp

while $queue$ is not empty

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(queue)$

if $\text{REMOVE-INCONSISTENT-VALUES}(X_i, X_j)$ **then**

for each X_k **in** $\text{NEIGHBORS}[X_i]$ **do**

 add (X_k, X_i) to $queue$

function $\text{REMOVE-INCONSISTENT-VALUES}(X_i, X_j)$ **returns** true iff succeeds

$removed \leftarrow false$

for each x **in** $\text{DOMAIN}[X_i]$

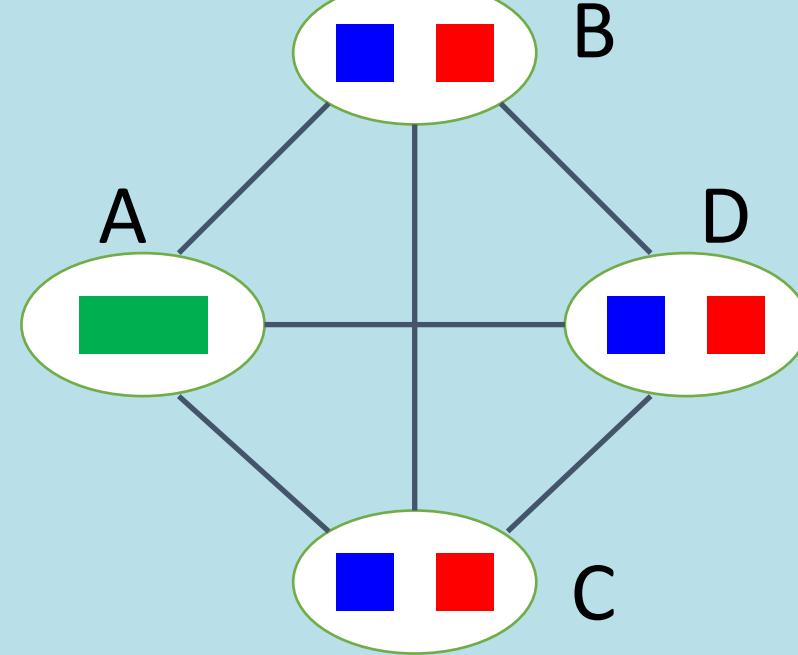
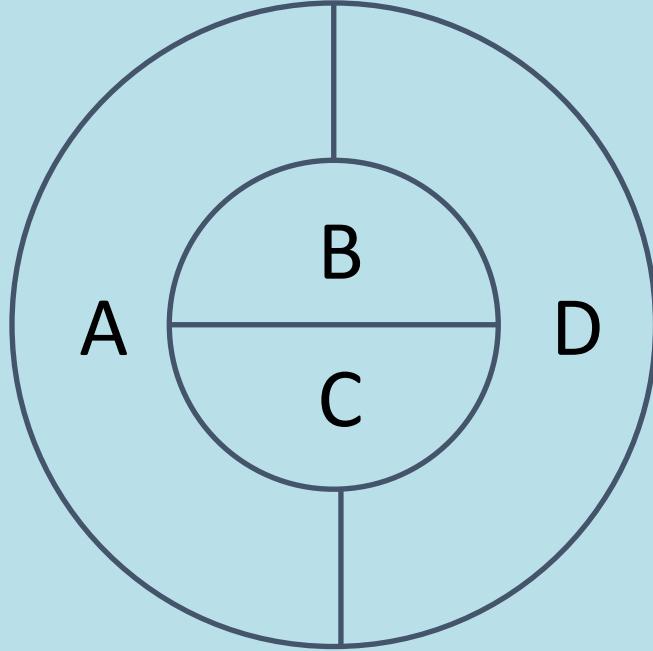
if no value y in $\text{DOMAIN}[X_j]$ allows (x, y) to satisfy the constraint $X_i \leftrightarrow X_j$

then delete x from $\text{DOMAIN}[X_i]$; $removed \leftarrow true$

return $removed$

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

2.5 Does arc consistency always detect the lack of a solution?



- There exist stronger notions of consistency (path consistency, k-consistency), but we won't worry about them

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

2.5 Backtracking search with inference

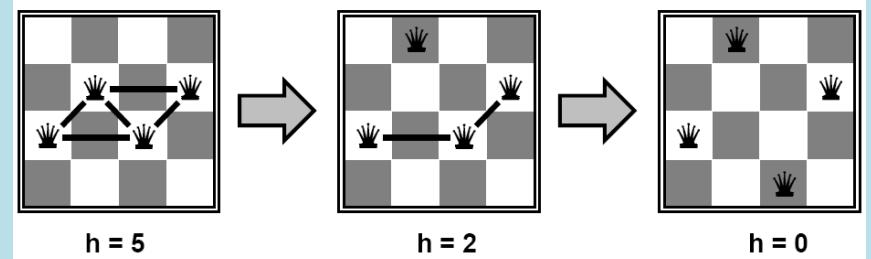
```
function RECURSIVE-BACKTRACKING(assignment, csp)
    if assignment is complete then return assignment
    var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp)
        if value is consistent with assignment given CONSTRAINTS[csp]
            add  $\{ \text{var} = \text{value} \}$  to assignment
            result  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
            if result  $\neq$  failure then return result
            remove  $\{ \text{var} = \text{value} \}$  from assignment
    return failure
```



- Do inference (forward checking or constraint propagation) here

2.5 Local search for CSPs

- Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned
- To apply to CSPs:
 - Allow states with unsatisfied constraints
 - Attempt to improve states by **reassigning** variable values
- Variable selection:
 - Randomly select any conflicted variable
- Value selection by **min-conflicts** heuristic:
 - Choose value that violates the fewest constraints
 - I.e., hill-climb with $h(n)$ = total number of violated constraints



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

2.5 Summary

- CSPs are a special kind of search problem:
 - States defined by values of a fixed set of variables
 - Goal test defined by constraints on variable values
- **Backtracking** = depth-first search where successor states are generated by considering assignments to a single variable
 - **Variable ordering** and **value selection** heuristics can help significantly
 - **Forward checking** prevents assignments that guarantee later failure
 - **Constraint propagation** (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Local search can be done by iterative min-conflicts

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

Your turn!

Task

- What is the difference between a consistent assignment and a complete assignment in the context of CSPs, and why is consistency important for solving CSPs efficiently?
- How does backtracking search work in CSPs, and what is one technique used to improve its efficiency (e.g., forward checking or arc consistency)?

Workbook Exercises

- Please read the chapters 2-4 from Russell, 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. Russell, 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.ntv.de/mediathek/videos/wirtschaft/Skate-Shop-verkauft-Fotos-statt-Schuhe-article21229466.html>

Images

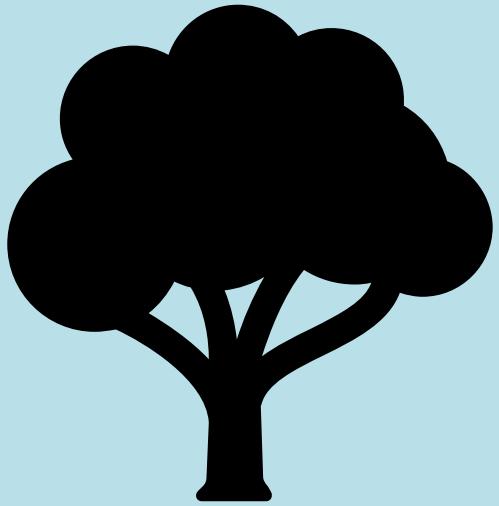
All images that were not marked other ways are made by myself, or licensed ↗[CC0](#) from ↗[Pixabay](#).

Further reading

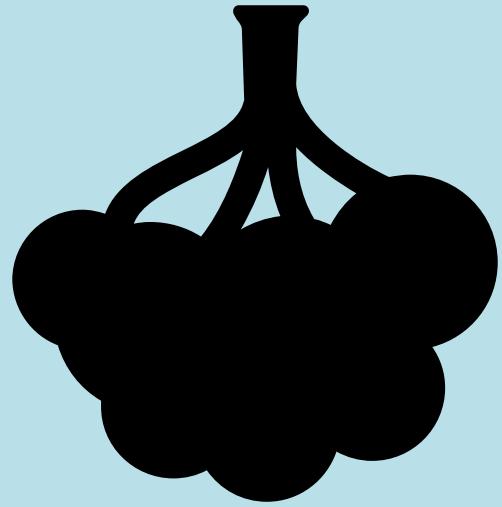
- If you are also interested in visiting all US national parks, I can recommend to take a look at the blog article from *University of Penn data scientist Dr. Randal Olson* ([↗http://www.randalolson.com](http://www.randalolson.com)). He created a roadtrip of the optimal way to visit all 47* U.S. National Parks in the mainland United States. Take a look at his Python projects ([↗Github](#)).
- I can you also recommend the following Golem articel ([↗https://www.golem.de/](https://www.golem.de/)) to read more about the application of the A* algorithm in pathfinding in games.

*before Gateway Arch in 2018 became a national park

Agent	<i>An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators (Russell & Norvig, 2016)</i>
Agent program	<i>Implements the agents function (Russell & Norvig, 2016, p.59)</i>
Performance measure	<i>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 (Russell & Norvig, 2016, p.59)</i>
Rationality/Rational agent	<i>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 (Russell & Norvig, 2016, p.37)</i>
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	<i>External environment of an agent including the performance measure, the external environment, the actuators, and the sensors (Russell & Norvig, 2016, p.59)</i>



How normal
people see trees



How computer
scientists see trees