184.735 VU 2.0 Einführung in Künstliche Intelligenz

Artificial Intelligence

Institute of Logic and Computation
Knowledge-Based Systems Group
Vienna University of Technology

SS 2023

slides partially by S. Russell
AIMA Chapter 1, with extensions

The Russell/Norvig Book

“Artificial Intelligence : A Modern Approach” (AIMA)

- The first edition (1995) can be found at

  http://www.eecs.berkeley.edu/~russell/aima1e/

- 2nd edition (2004;2007, in German), and **3rd edition (2010;2012 German)** is available at the TU library (Lehrbuchsammlung)

- The 4th edition (2021) is also available at the TU library (Lehrbuchsammlung)

Outline

1. What is AI?

2. History of AI

3. State of the Art

4. Shortcomings and Issues

5. Conclusion

Outline

1. What is AI?

2. History of AI

3. State of the Art

4. Shortcomings and Issues

5. Conclusion

1. Systems thinking like humans	2. Systems acting like humans
"The exciting new effort to make computers think ... *machines with minds*, in the full and literal sense (Haugland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision making, problem solving, learning" (Bellman, 1978)	"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do the things at which, at the moment, people are better." (Rich and Knight, 1991)
3. Systems thinking rationally	**4. Systems acting rationally**
"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of computations that make it possible to perceive, reason, and act." (Winston, 1992)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behaviour in artifacts." (Nilsson, 1998)

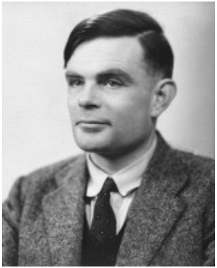
- 1 + 2: **"strong AI"** 3 + 4: **"weak AI"**
- "Künstliche Intelligenz" vs "gekünstelte Intelligenz"

1. Thinking humanly: Cognitive Science

- 1960s "cognitive revolution": information-processing psychology replaced prevailing orthodoxy of behaviorism
- requires scientific theories of internal activities of the brain
  - What level of abstraction? "Knowledge" or "circuits"?
  - How to validate? Requires
    - predicting and testing behavior of human subjects (top-down)
    - direct identification from neurological data (bottom-up)
- both approaches (roughly, Cognitive Science and Cognitive Neuroscience) are now distinct from AI
- both share with AI the following characteristic:

  the available theories do not explain (or engender) anything resembling human-level general intelligence
- hence: all three fields share one principal direction!

2. Acting humanly: The Turing Test

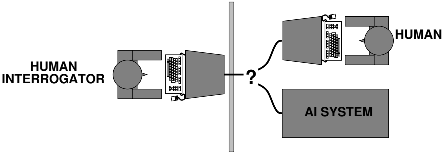


Alan M. Turing (1912–1954)

- key article: “Computing Machinery and Intelligence” (*Mind* 49:433-460, 1950)
- question “Can machines think?” ⟶ “Can machines behave intelligently?”

2. Acting humanly: The Turing Test

- operational test for intelligent behavior: the Imitation Game



- Turing predicted that by 2000, a machine might have a 30% chance of fooling a lay person for 5 minutes
- anticipated all major arguments against AI in next 50 years
- suggested major components of AI: knowledge, reasoning, language understanding, learning

problem: Turing Test is *not reproducible*, *not constructive*, and not amenable to *mathematical analysis*

3. Thinking rationally: Laws of Thought

- normative (or prescriptive) rather than descriptive
- Aristotle: what are correct arguments/thought processes?
- several Greek schools developed various forms of logic:
  - *notation* and *rules of derivation* for thoughts
  - may or may not have proceeded to the idea of mechanization
- direct line through mathematics and philosophy to modern AI
- problems:
  1. Not all intelligent behavior is mediated by logical deliberation
  2. What is the purpose of thinking?
     What thoughts *should* I have out of all the thoughts (logical or otherwise) that I *could* have?

4. Acting rationally

Rational behavior: doing the right thing

- do what expectedly maximizes goal achievement, given the available information
- Thinking is not a must but should serve rational action
- agents are entities that perceive and act
- abstractly, an agent is a function $f : \mathcal{P}^* \to \mathcal{A}$ mapping percept histories to actions
- for any given class of environments and tasks, we seek a rational agent (or class of rational agents) with the best performance
- caveat: *computational limitations make perfect rationality unachievable*

  $\to$ design best program for given machine resources

Outline

1. What is AI?

2. History of AI

3. State of the Art

4. Shortcomings and Issues

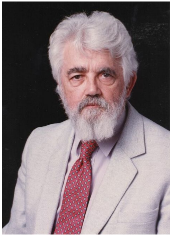
5. Conclusion

AI prehistory

Philosophy	logic, methods of reasoning mind as physical system foundations of learning, language, rationality
Mathematics	formal representation and proof algorithms, computation, (un)decidability, (in)tractability probability
Psychology	adaptation phenomena of perception and motor control experimental techniques (psychophysics, etc.)
Economics	formal theory of rational decisions
Linguistics	knowledge representation grammar
Neuroscience	plastic physical substrate for mental activity
Control Theory	homeostatic systems, stability simple optimal agent designs

Early Beginnings

1943	McCulloch & Pitts: Boolean circuit model of the brain
1950	Turing's "Computing Machinery and Intelligence"
1952-69	Look, Ma, no hands!
1950s	Early AI programs, including • Samuel's checkers program, • Newell & Simon's Logic Theorist • Gelernter's Geometry Engine
1956	**Dartmouth Workshop:** "Artificial Intelligence" adopted
1958	LISP as an AI language



John McCarthy
(1927-2011)

The Rise of Logic and Rules

1958	McCarthy's "programs with common sense" (knowledge & reasoning)
1965	Robinson's resolution principle: complete algorithm for logical reasoning
1972	Prolog programming language
1966-74	AI discovers computational complexity
1969	Minsky & Papert: limitation of perceptrons $\rightarrow$ neural network research almost disappears
1969-79	Early knowledge-based systems, focus on domain specific knowledge

Marvin Minsky
(1927-2016)

The Roaring 1980s of AI

1980-88 expert systems (XPS) industry booms

- successful domain-specific systems (MYCIN, XCON, . . .) sparked huge interest
- XPS everywhere
- ambitious projects around the globe (5th Generation, CYC, ECRC, MCC ....)
- thousands of people at AI conferences
- expectations were flying high ...
- ... into a tough winter

early 1990s: XPS industry bursts

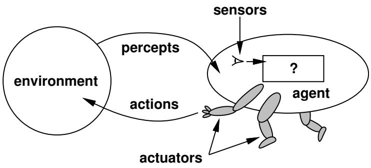
AI Goes on

1986– Neural networks return to popularity

1987- Resurgence of probability and machine learning
- Hidden Markov Models
- Bayesian Networks

"Nouvelle AI": Artifical Life, Genetic Algs, Soft Computing

1995- Agents, agents, everywhere . . .
first edition of AIMA



2001- Big Data, due to computing power and the Web

2003- Human-level AI back on the agenda

2011- Deep Learning rise

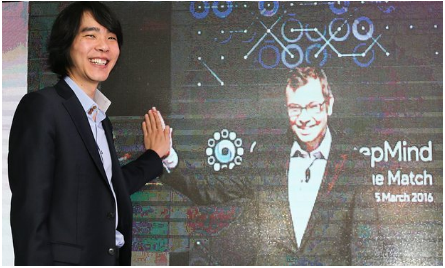
Deep Blue (1997)

- IBM's Deep Blue beats World Champion Garry Kasparov
- a milestone in man vs machine gaming
- conventional game search techniques, massive hardware

IBM Watson (2011)

- IBM's DeepQA project: answer questions in natural language
- massive text databases and heuristic search (run many algs)
- wins against Jeopardy champions Ken Jennings and Brad Rutter
- commercial exploitation: consultancy aid, e.g., medical; legal (Ross Intelligence) autonomous vehicles (Olli bus)

Alpha-Go (2015-2018)



- Deep Learning (Google's Deep Mind company)
  - applied first to Atari games
  - combine neural networks with search techniques (Monte Carlo-Tree Search)
- GO is considered notoriously hard (PSPACE-complete)
- other games before (checkers 2007, poker on horizon)
- earned the Marvin-Minsky Medal for Outstanding Achievements in AI
- **follow ups:** Alpha-Go Master, Alpha-Go Zero (no human data), Alpha-Zero (chess etc.), AlphaStar (StarCraft II)

Libratus (2017) at Rivers Casino

- Libratus (Carnegie Mellon University) defeated in Jan. 2017 four top humans in *Heads-up, No-Limit Texas Hold'em* poker
- exceedingly complex game: 10^{160} play paths
  - over nearly 3 weeks, 120 000 hands played
- *breakthrough* on strategic reasoning with *imperfect information*
  - analyses *its own weaknesses*, not only the opponent's
- **follow-up:** Pluribus (2019) against multiple players
  (and needs no super-computer as Libratus did)

AlphaFold (2018)

AI Breakthrough in Biology

- Protein folding problem: understand how an amino acid sequence can determine the 3-D protrein structure
- grand challenge

Outline

1. What is AI?

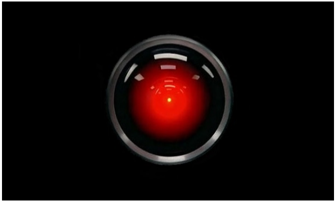
2. History of AI

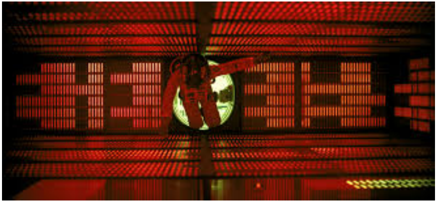
3. State of the Art

4. Shortcomings and Issues

5. Conclusion

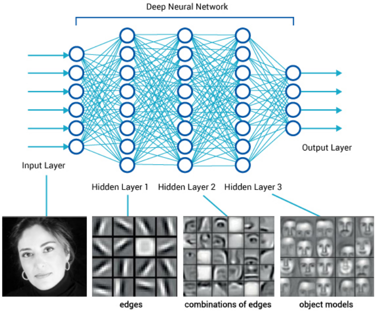
Benchmark: HAL of *2001: A Space Odyssey* (1968)





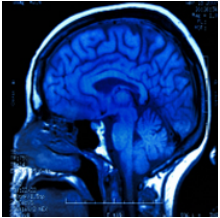
- capabilities
    - vision
    - language understanding
    - game playing
    - real world reasoning
    - planning
    - art and aesthetics
    - robotics (inherently)
    - ...
- recently, striking advances through deep learning

The Deep Learning Decade



- **Deep Learning:** Artifical Neural Networks with many layers
- feature extraction and learning (un)supervised
- tremendously increasing in the last decade
- **crux:** lots of data
- push for hardware (GPUs, designated chips)
- *“data-driven” vs “model-driven” processing*

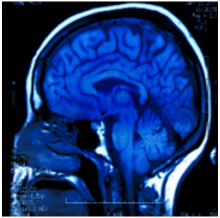
Vision, Image Understanding



- big success of ANNs in face / object recognition
- large amounts of training data
- valuable e.g. in medical applications (diagnosis)

Vision, Image Understanding

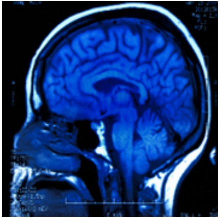
meeting a cool bishop



- big success of ANNs in face / object recognition
- large amounts of training data
- valuable e.g. in medical applications (diagnosis)
- more difficult:
    - *semantic queries*

Vision, Image Understanding

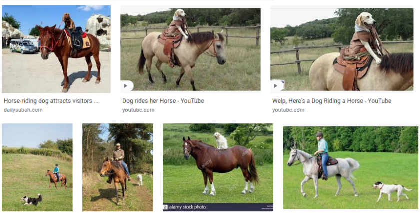
meeting a cool bishop



- big success of ANNs in face / object recognition
- large amounts of training data
- valuable e.g. in medical applications (diagnosis)
- more difficult:
  - *semantic queries*

Vision, Image Understanding

meeting a cool bishop



horse riding a dog

- big success of ANNs in face / object recognition
- large amounts of training data
- valuable e.g. in medical applications (diagnosis)
- more difficult:
  - *semantic queries*

Vision, Image Understanding

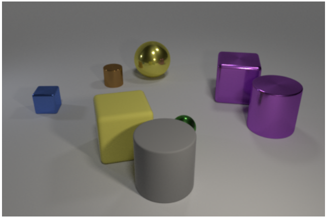
meeting a cool bishop

occluded tiger

- big success of ANNs in face / object recognition
- large amounts of training data
- valuable e.g. in medical applications (diagnosis)
- more difficult:
  - *semantic queries*
  - *(partially) occluded objects*

Vision, Image Understanding

meeting a cool bishop



The tiny shiny cylinder has what color?

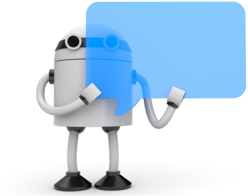
- big success of ANNs in face / object recognition
- large amounts of training data
- valuable e.g. in medical applications (diagnosis)
- more difficult:
  - *semantic queries*
  - *(partially) occluded objects*
- *image captioning vs. visual question answering*

Image Generation

Speech Recognition

- tremendous advances since 2001
  - Hidden Markov Models
  - later, since $\approx$ 2010, deep neural networks (e.g., Google Voice) use long short-term memory
- dictation systems
- Alexa, Siri, Cortana, Google Assistant etc.
- however, deeper understanding is still missing

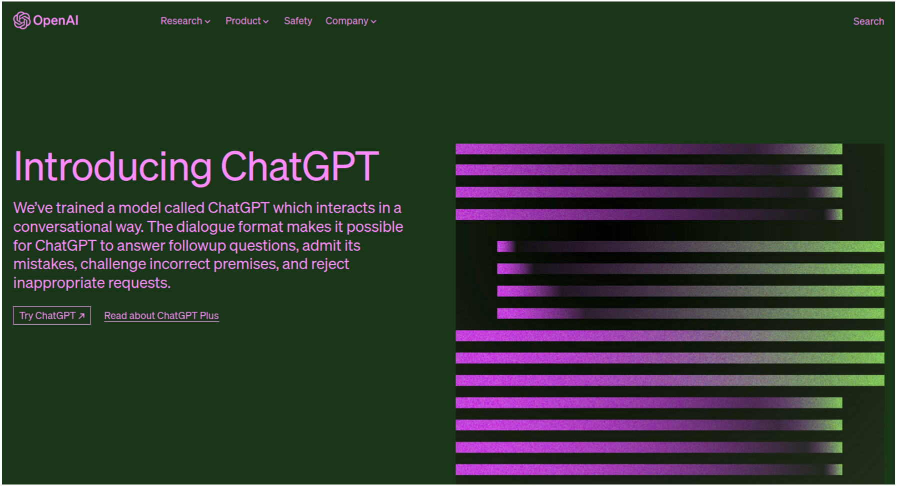
Chatbots



- smart chat programs (grandmother: ELIZA in the 1960s)
- may fool people on a narrow topic some time (e.g. Cleverbot, Eugene Goostman)
- downscaled versions of the Turing Test (e.g. Löbner prize)
- don't work for broad generality
- usage: guide & assist people (Google's Allo platform), Aiko Chihira (Toshiba 2015), MICA facebook chat (Ondrisek 2016)

Many AI researchers do *not* consider chatbot competitions as important

ChatGPT


OpenAI
Research
Product
Safety
Company
Search
Introducing ChatGPT
We've trained a model called ChatGPT which interacts in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests.
Try ChatGPT ↗
Read about ChatGPT Plus

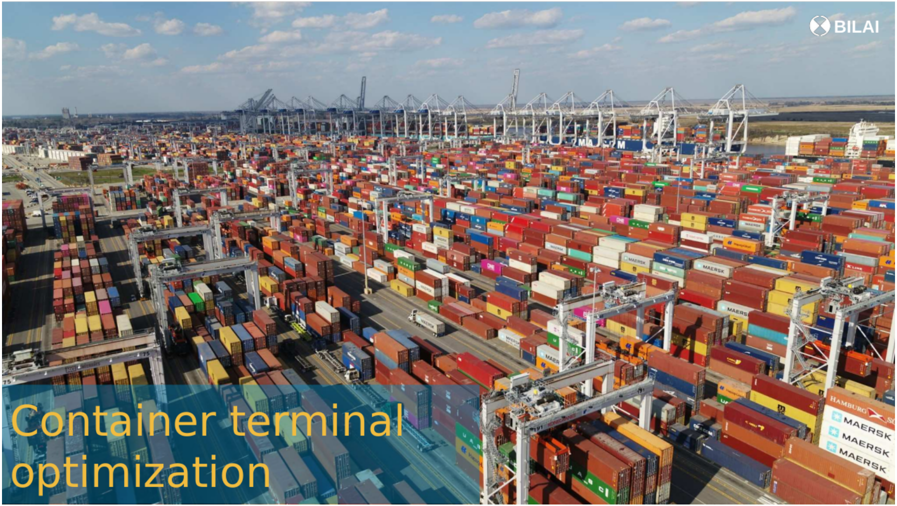
Autonomous Vehicles

NASA/JPL-Caltech

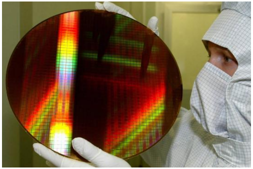
- **Self-driving Cars**
- DARPA Challenges 2004/05
- Urban Challenge 2007
- immense progress
- today, trial/development phase
- industrial use in 5-20 years (?)

- **Mars Rovers**
- Spirit, Opportunity (2003)
- Curiosity (2012)
- Perseverance (2021)
- AI techniques vital:
  - sensing and control
  - planning
  - experiments

Configuration and Scheduling

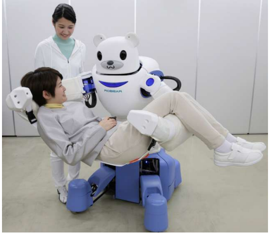


Configuration and Scheduling, cont'd



- large scale configuration problems
  - hardware, software
  - plants
  - user interfaces, . . .
- scheduling as temporal configuration (loosely)
  - job processing
  - tournaments . . .
- *learning of limited value:* ad-hoc formulation, hard constraints, missing data

Home Robotics



- home assistant for household (U Tokyo)
- nursery (Robear)
- *ambient assisted living*
- growing need and market
- important: social components
  (language, interaction, emotions)

Outline

1. What is AI?

2. History of AI

3. State of the Art

4. Shortcomings and Issues

5. Conclusion

What's AI Missing?

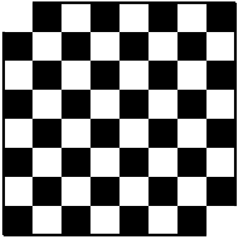
- from a cognitive perspective, a lot:
  - understanding of the mind:
    - "brains cause minds" (J. Searle) sounds utterly simple, but how?
    - Human Brain Project (EU FET flagship program), 2013-2023: simulation as a basis
  - deeper real world reasoning
  - abstraction
  - commonsense and natural language

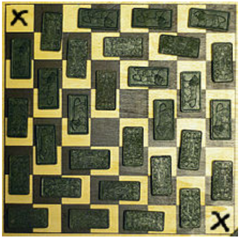
abstraction
reasoning
learning
perception

- from a technological perspective, too:
  - scalability and efficiency
  - robustness
  - validation and verifiability
  - *explainability: what/how and why*

Issue: Abstraction

Mutilated Chessboard Problem:





- suppose a chessboard has two diagonally opposite corners removed
- *Is it possible to place dominoes of size 2x1 so as to cover all 62 squares?*

Issue: Natural Language Understanding

- co-reference resolution
  - **Pronoun Reference (PR) Problem**

    "One chilly May evening the priest invited Marjorie and myself into her room."

    Google Translate: *"An einem kühlen Maiabend lud der Priester Marjorie und mich in ihr Zimmer ein."*
  - **Winograd Schemes (WS)** (Levesque et al. 2012): pair of almost identical sentences, but different reference resolution

    "I poured milk from a bottle to a glass until it was empty."

    Google Translate: *"Ich goss Milch aus einer Flasche in ein Glas, bis es leer war."*
  - **WS Challenge (2016)**: level 1: PR, level 2: WS, none passed level 1
    language model BERT achieved 90,1% in 2019, GPT-3 88,3% in 2020
- negation: `> do you not want to format your hard disk? (y/n)`

Simple Question Answering

- Monty Python's Bridge Keeper

  - encyclopedic questions
  - answer requires little (no) inference
  - Information Retrieval methods suffice

Harder Question Answering

- NY Regents Science exams (4-way multiple choice)
  - questions of different levels of complexity
  - sample (complex world knowledge):

    Fourth graders are planning a roller-skate race. Which surface would be best for the race? (A) gravel (B) sand (C) blacktop (D) grass

  - Aristo System (Clarke & Ezioni, 2016):
    performance of 75% / 63% / 41% for 4th / 8th / 12th grade

- Allen AI Institute Reasoning Challenge (ARC) 2018
  - 7,787 science exam questions drawn from a variety of sources

ST-MoE-32B	(Google Brain, 2022)	87%
UnifiedQA + ARC MC/DA + IR	(Aristo team, 2020)	81%
UnifiedQA - v2 (T5-11B)	(D. Khashabi, 2020)	81%

- particular challenge: bring in **common sense knowledge/reasoning**
  background on naive physics, behavior of people, temporal relations, e.g.,
  - objects fall down by gravity
  - a selfish person will not share its cookie
  - 'happens before' is transitive

Expert Question Answering

- deeper problem solving

“Secretum finis Africae manus supra idolum age primum et septimum de quatuor.”

- understand the sentence (language)
- make sense out of it
- match it to the problem at hand

Expert Question Answering, cont'd

EI

Well, you are in front of a door, which seems coated with shiny metal, and the door is locked. above the door is written on the wall: "Secretum finis Africae manus supra idolum age primum et septimum de quatuor." What should I do?

Hmm, that's an interesting riddle. Based on the text above the door, it appears to be a puzzle that requires some knowledge of Latin and possibly ancient history.

"Secretum finis Africae manus supra idolum age primum et septimum de quatuor" roughly translates to "The secret of the end of Africa, place your hand above the idol, the first and seventh of four."

Without more information, it's hard to know exactly what this means, but here's a possible interpretation:

- "The end of Africa" could refer to a specific location or landmark in Africa.
- "Place your hand above the idol" suggests that there may be an idol or statue nearby that you need to interact with.
- "The first and seventh of four" could refer to a specific order or sequence of actions that you need to take, possibly involving the numbers 1 and 7.

Based on this interpretation, I would suggest looking around for any idols or statues nearby and trying to interact with them in some way. You might also want to experiment with different sequences of actions, starting with the first and seventh steps. Good luck!

Expert Question Answering, cont'd

EI

Ah, sorry, the writing on the wall is in fact "Super thronos viginti quatuor", the other text was a hidden hint.

Ah, I see! That changes things a bit. "Super thronos viginti quatuor" translates to "Above the twenty-four thrones."

Based on this new information, here's a revised interpretation of the puzzle:

- The door is locked and coated with shiny metal, which suggests that it's a secure or important door.
- The text above the door indicates that the solution to the puzzle involves something "above the twenty-four thrones."
- Without more information, it's hard to know exactly what the twenty-four thrones refer to. However, one possibility is that they refer to the twenty-four elders or thrones mentioned in the Book of Revelation in the Bible. In that context, the twenty-four elders are said to sit on thrones surrounding God's throne in heaven.
- Based on this interpretation, one possible solution to the puzzle would be to look for something above or around a group of twenty-four thrones, such as a hidden lever or key.

Expert Question Answering: Reasoning I

CH

I consider going to graduate school in the UK, maybe Lancester or Bristol. What other places would you recommend to have a look at?

If you're considering graduate school in the UK, you have many options to choose from, as the country is home to some of the world's top universities. In addition to Lancaster and Bristol, you may also want to consider the following universities:

1. University of Oxford
2. University of Cambridge
3. Imperial College London
4. University College London (UCL)
5. The University of Edinburgh
6. The University of Glasgow
7. King's College London (KCL)
8. The University of Warwick
9. The University of York
10. The London School of Economics and Political Science (LSE)

Expert Question Answering: Reasoning II

EI

Let's avoid politics, back to the soccer world cup. If the football would be made of gold, would then have Croatia won the 2018 world cup?

The material used to make the soccer ball would not have had any impact on the outcome of the 2018 World Cup final between France and Croatia. The ball is just a tool used in the game and its properties do not influence the players' abilities or the final score.

In soccer, the outcome of a match is determined by a variety of factors, including the skill and strategy of the players, the tactics employed by the coaches, the conditions on the day of the match, and luck. The material used to make the ball is not one of these factors and would not have affected the outcome of the final in any meaningful way.

Outline

1. What is AI?

2. History of AI

3. State of the Art

4. Shortcomings and Issues

5. Conclusion

Summary

- AI technology has advanced recently considerably
  - Deep Learning (DL) in the spotlight
- ***AI $\neq$ Machine Learning:*** there is more than data-driven AI and DL
  - hot topic: neuro-symbolic AI
    marry Machine Learning and Symbolic AI
- much is missing from a cognitive / technical perspective
  - explainable and trustful AI
- **warning:** cautious breed of expectations
  - AI is not a silver bullet, just a tool
- **important:** social perspective
  - ethical issues, personal life, society

need for a *Humane AI*

184.735 VU 2.0 Einführung in Künstliche Intelligenz

Intelligent Agents

Institute of Logic and Computation
Knowledge-Based Systems Group
Vienna University of Technology

SS 2023

slides by S. Russell, adapted by T. Eiter
AIMA Chapter 2, Sections 1–4.5

Issues

Design principles for implementing successful “intelligent” agents

1. Agents and Environments
2. Rationality
3. PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors
4. Environment Types
5. Agent Types
6. Conclusion

Outline

1. Agents and Environments

2. Rationality

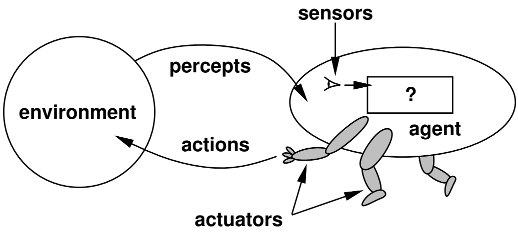
3. PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors

4. Environment Types

5. Agent Types

6. Conclusion

Situated agents



Agents include humans, robots, softbots, thermostats, etc.

- Sensors for perceiving the "world" (they produce perception sequences)
- Actuators for acting
- The agent function $f : \mathcal{P}^* \rightarrow \mathcal{A}$ maps from *percept histories*, i.e., sequences $p_1, p_2, \ldots, p_n$, of *percepts* p_i, $n \geq 0$, to actions
- The agent program runs on the physical architecture to produce f

Outline

1. Agents and Environments

2. Rationality

3. PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors

4. Environment Types

5. Agent Types

6. Conclusion

Aspects for defining rationality

- Performance measure, defining criteria of success
  E.g. cleaning robot: % of area cleaned;
  shopping agent: low prize offer
- the agent's prior knowledge of the environment
- agent actions
- percept sequence (history)

Rational agent:

> *For each possible percept history, select an action that is expected to maximize its performance measure, given the evidence by the percept history and whatever built-in knowledge the agent has.*

Rationality, cont'd

- Rational $\neq$ omniscient: percepts may lack relevant infos
- Rational $\neq$ clairvoyant: action may turn out differently
- Rational $\neq$ successful
- Important
  - exploration (get new info about the world),
  - learning (take experience, successful/failed actions into account)
  - autonomy (replace and extend initial knowledge by experience and learned knowledge)

Outline

1. Agents and Environments

2. Rationality

3. PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors

4. Environment Types

5. Agent Types

6. Conclusion

Rational agent design

To design a rational agent, we must specify the task environment

Consider, e.g., the task of designing an automated taxi:

- Performance measure?? safety, destination, profits, legality, comfort, . . .
- Environment?? streets/freeways, traffic, pedestrians, weather, . . .
- Actuators?? steering, accelerator, brake, horn, speaker/display, . . .
- Sensors?? video, accelerometers, gauges, engine sensors, keyboard, GPS, . . .

Agent application types with PEAS description

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Outline

1. Agents and Environments
2. Rationality
3. PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors
4. Environment Types
5. Agent Types
6. Conclusion

A list of different environment types

- fully observable vs. partially observable:
  sensors detect all *relevant* properties of the world for the current action
- single-agent vs. multi-agent:
  only one agent, no cooperation and no competition between agents
- deterministic vs. stochastic:
  next state determined by the current state and the performed action
- episodic vs. sequential:
  agent's experience is divided into “atomic” parts (independent from each other)
  In episodic environments, the choice of action only depends on the current episode

A list of different environment types, cont'd

- static vs. dynamic:
  the world does not change during the reasoning time of the agent

  semi-dynamic: static, but the performance score decreases with deliberation time

- discrete vs. continuous:
  world properties have discrete values, e.g. time, number of possible states, …

- known vs. unknown:
  state of knowledge about the "laws of physics" of the environment

Environment examples

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Outline

1. Agents and Environments

2. Rationality

3. PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors

4. Environment Types

5. Agent Types

6. Conclusion

Agent hierachy by capabilities

$$\boxed{\text{Agent} \;=\; \text{Architecture} + \text{Program}}$$

- Architecture: programming device + sensors + actuators
- Program: gets sensor data, returns actions for the actuators

Four basic types of agent programs in order of increasing generality:

simple reflex agents

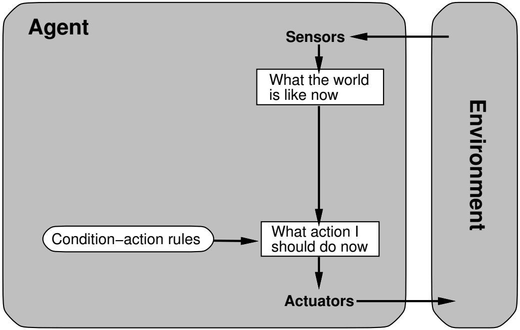
reflex agents with state (model-based reflex agent)

goal-based agents

utility-based agents

All these can be turned into *learning agents* (later)

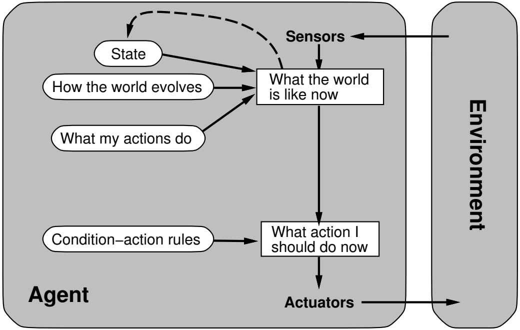
1. Simple reflex agents



- no memory
- no sequences of percepts
- looping possible: **if** *closed(door)* **then** *open_door*

2. Model-based reflex agents

Add on: internal state and model of the world's evolution



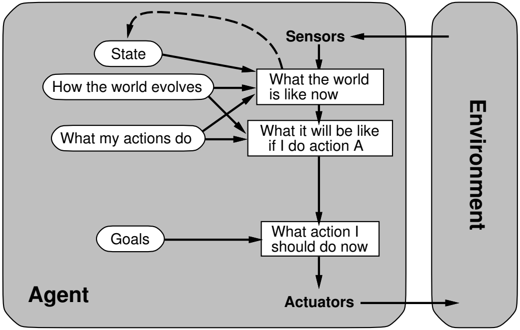
- memory, maintain/update world state

$$state \leftarrow \textsc{Update-State}(state, latest\_action, percept, model)$$

- reason about unobservable parts
- deal with uncertainty, implicit goals

3. Goal-based agents

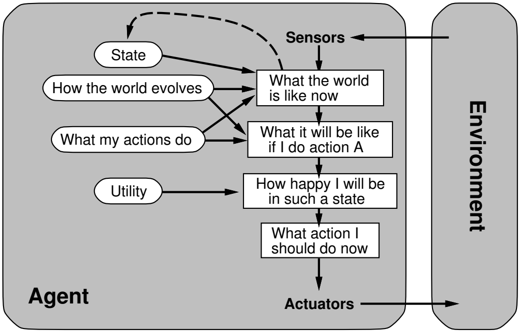
Add on: explict goals



- model the world, goals, and actions & their effects explicitly
- more flexible, better maintainable
- Search & Planning for single/sequence of action/s to achieve a goal

4. Utility-based agents

Add on: take happiness into account



- assess goals with a utility function
- resolve conflicting goals
- use expected utility for decision

Outline

1. Agents and Environments

2. Rationality

3. PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors

4. Environment Types

5. Agent Types

6. Conclusion

Summary

- "Intelligent Agents" aim at acting rationally
- Rational agent design comprises several aspects (PEAS criteria)
- A range of different environments
- A hierarchy of agents by capabilities
- Building intelligent utility-based agents might sound simple, but is difficult in general
- Developing agent capabilities (problem solving, planning, reasoning, decision making, learning, cooperation, communication) is challenging

184.735 VU 2.0 Einführung in Künstliche Intelligenz

Problem Solving and Search (I)

Institute of Logic and Computation
Knowledge-Based Systems Group
Vienna University of Technology

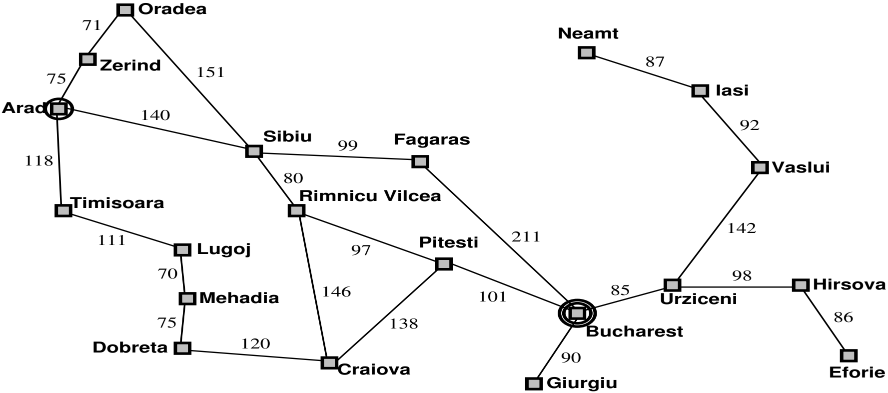
SS 2023

slides by S. Russell, adapted by T. Eiter
AIMA Chapter 3, Sections 1-4

An example scenario: Romania trip

- on holiday in Romania; currently in Arad
- flight leaves tomorrow from Bucharest
  - formulate goal:
    to be in Bucharest (before departure)
  - formulate problem:
    - states: various cities
    - actions: drive between cities
  - find solution:
    route = sequence of cities

Example: Romania trip, cont'd



possible solution: Arad, Sibiu, Fagaras, Bucharest

Outline

Need a systematic approach to solve problems like the above

1. Introduction

2. Problem Formulation

3. Basic Search Algorithms

4. Search Strategies

5. Uninformed Search Strategies

6. Conclusion

Outline

1. Introduction

2. Problem Formulation

3. Basic Search Algorithms

4. Search Strategies

5. Uninformed Search Strategies

6. Conclusion

Search problem definition

A search problem is defined by four items:

1. initial state e.g., “at Arad”
2. successor function $S(x)$ = set of action–state pairs
   e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \ldots\}$
3. goal test, can be
   - explicit, e.g., x = “at Bucharest”
   - implicit, e.g., $HasAirport(x, "Otopeni")$
4. path cost (additive)
   e.g., sum of distances, number of actions executed, etc.
   $c(x, a, y)$ is the step cost, assumed to be ≥ 0

A solution is a sequence of actions leading from the initial state to a goal state

Selecting a state space

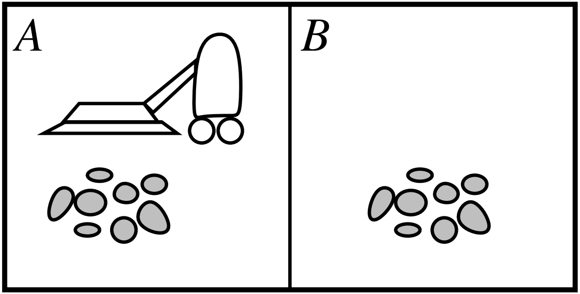
Real world is absurdly complex
⇒ state space must be **abstracted** for problem solving

- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  e.g., "Arad → Zerind" represents a complex set of possible routes, detours, rest stops, etc.

  For guaranteed realizability, **any** real state "in Arad" must get to some real state "in Zerind"
- (Abstract) solution = set of real paths that are solutions in the real world

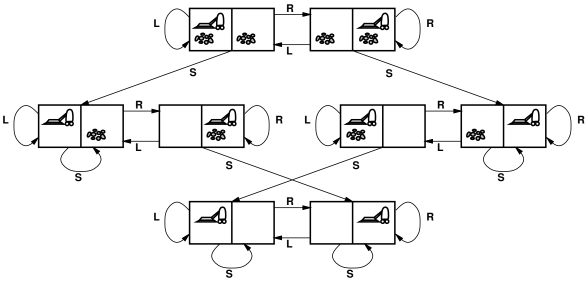
Each abstract action should be "easier" than the original problem!

Example: vacuum-cleaner world



- World: contains two locations (squares)
- Percepts: location and contents, e.g., $[A, Dirty]$

Example: vacuum world state space graph



- states??
  integer dirt and robot locations (ignore dirt amounts etc.)
- actions??
  Left, *Right*, *Suck*, *NoOp*
- goal test??
  no dirt
- path cost??
  1 per action (0 for *NoOp*)

Example: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

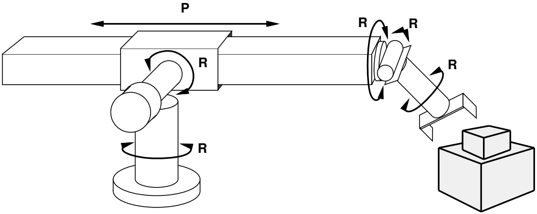
1	2	3
4	5	6
7	8	

Goal State

- states??
  integer locations of tiles (ignore intermediate positions)
- actions??
  move blank left, right, up, down (ignore unjamming etc.)
- goal test??
  = goal state (given)
- path cost??
  1 per move

Note: Finding an optimal solution of a given n-puzzle is NP-hard

Example: robotic assembly



- states??
  real-valued coordinates of robot joint angles, parts of the object to be assembled
- actions??
  continuous motions of robot joints
- goal test??
  complete assembly **with no robot included!**
- path cost??
  time to execute

Outline

1. Introduction

2. Problem Formulation

3. Basic Search Algorithms

4. Search Strategies

5. Uninformed Search Strategies

6. Conclusion

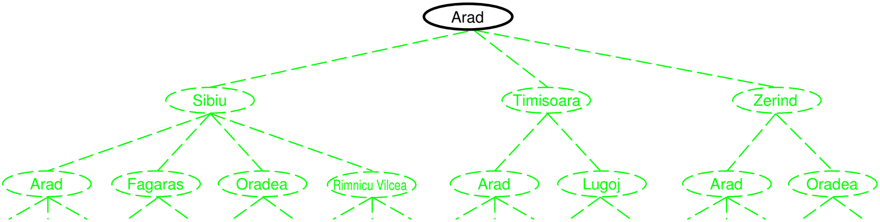
Tree search algorithms

Basic idea:

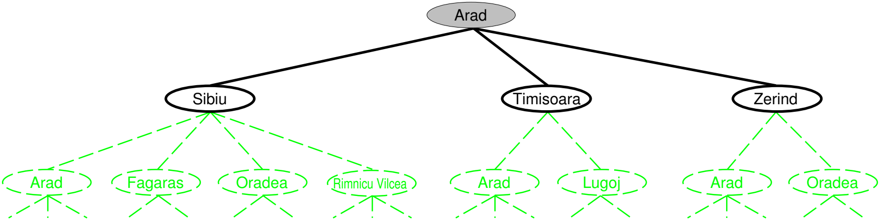
- offline, simulated exploration of state space
- by generating successors of already-explored states (a.k.a. expanding states)
- maintain a list of states available for expansion (frontier, open list)

```
function TREE-SEARCH(problem) returns a solution, or failure
  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
```

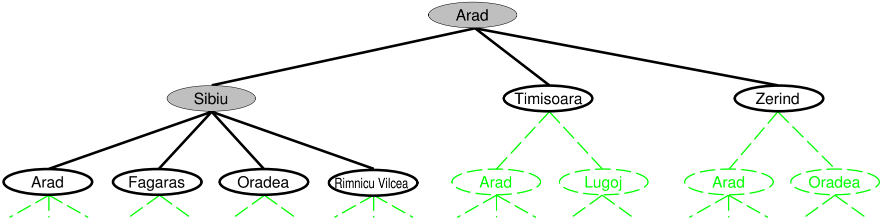
Tree search example



Tree search example

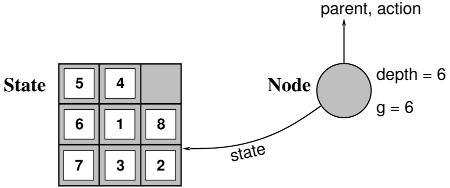


Tree search example



Implementation: states vs. nodes

- A state is a (representation of a) physical configuration
- A node is a data structure constituting part of a search tree
  it includes parent, children, depth, path cost $g(x)$
- States do not have parents, children, depth, or path cost!

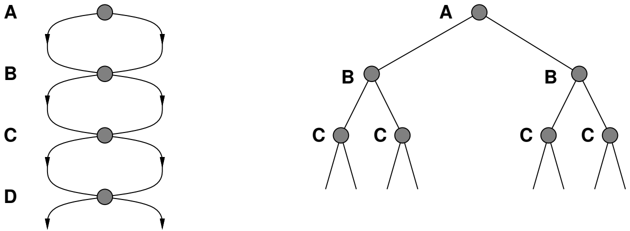


- The CHILD-NODE function creates a new node

```
function CHILD-NODE(problem, parent, action) returns a node
   return a node with
      STATE = problem.RESULT(parent.STATE, action),
      PARENT = parent, ACTION = action,
      PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!



- Other issue: nontermination

Graph search algorithms

Besides frontier, maintain a list of already considered states (explored set, closed list)

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  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
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

- here: do goal test at expansion time
- alternatively, test at generation time (benefit?)

Outline

1. Introduction

2. Problem Formulation

3. Basic Search Algorithms

4. Search Strategies

5. Uninformed Search Strategies

6. Conclusion

Search strategies

- A strategy is defined by picking the **order of node expansion**
- Different types of strategies:
  - uninformed search: basic algorithms, problem formulation as above
  - informed search: further information about solution costs (heuristics)
  - local search: “historyless,” one-step change

Search strategy evaluation

- Search strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated/expanded
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - b: maximum branching factor of the search tree
  - d: depth of the least-cost solution
  - m: maximum depth of the state space (may be ∞)

Outline

1. Introduction

2. Problem Formulation

3. Basic Search Algorithms

4. Search Strategies

5. Uninformed Search Strategies

6. Conclusion

Overview

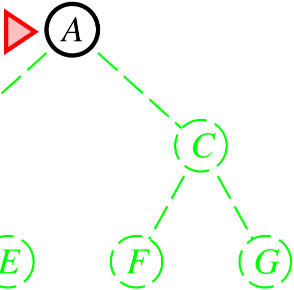
- Uninformed strategies use only the information available in the problem definition
- Some popular examples:
  - Breadth-first search (BFS)
  - Uniform-cost search (UCS)
  - Depth-first search (DFS)
  - Depth-limited search (DLS)
  - Iterative deepening search (IDS)

Breadth-first search (BFS)

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end

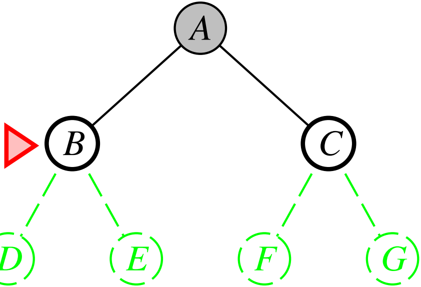


Breadth-first search (BFS)

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end

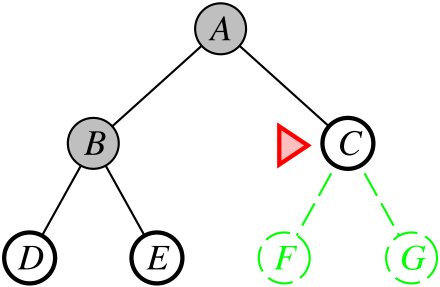


Breadth-first search (BFS)

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end

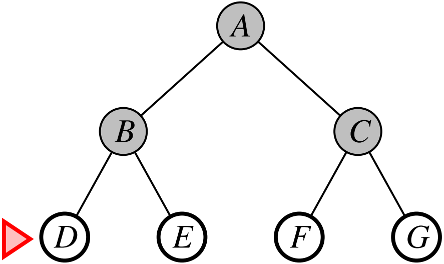


Breadth-first search (BFS)

Expand shallowest unexpanded node

Implementation:

frontier is a FIFO queue, i.e., new successors go at end



BFS Algorithm

Variant: do goal test *at generation time*

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Use queue data structure

- EMPTY?($queue$)
- POP($queue$)
- INSERT($element$, $queue$)

Properties of breadth-first search

- Complete??
  Yes (if b is finite)
- Time??
  Suppose solution is at depth d, worst case

  $$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

  note: exponential in d

  solution test *at expansion time*

  $$1 + b + b^2 + b^3 + \ldots + b^d + b^{d+1} = O(b^{d+1})$$
- Space??
  $O(b^d)$ resp. $O(b^{d+1})$ (keeps every node in memory)
- Optimal?? Yes (if cost = 1 per step); not optimal in general

big problem:

- time and space
- can easily generate large number of nodes per second

Space / Time Problem

- branching $b = 10$
- generate 1 million nodes / sec
- 1k bytes per node

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Uniform-cost search (UCS)

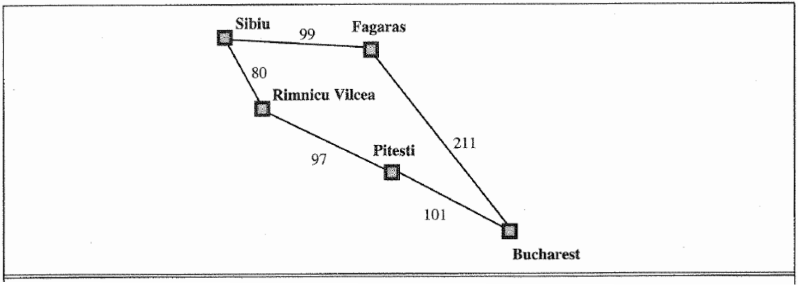
Expand least-cost unexpanded node

Implementation: *frontier* = queue ordered by path cost, lowest first

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
           replace that frontier node with child
```

Note: goal test is not at generation time but *later at expansion*

Uniform-cost search: Example



initial state Sibiu, goal state Sibiu; expand

- Sibiu: Rimnicu Vilcea (80), Fagaras (99)
- Rimniu Vilcea: Pitesti ($80 + 97 = 177$)
- Fagaras: Bucharest ($99 + 211 = 310$) goal state
- **Pitesti: Bucharest (**$177 + 101 = 288$**) optimal solution**

Properties of uniform-cost search

- Complete??
  Yes, if step cost $\geq \epsilon$
- Time??
  # of nodes n with $g(n) \leq C^*$, where C^* is the cost of an optimal solution, is $O(b^{1+\lfloor C^*/\epsilon \rfloor})$

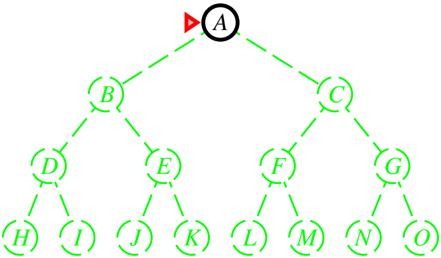
  note: – solution test at expansion time
  – for step cost = 1, same cost as BFS
- Space??
  # of nodes n with $g(n) \leq C^*$ is $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
- Optimal??
  Yes – nodes expanded in increasing order of $g(n)$

  note: solution test at expansion time ensures optimality

Depth-first search (DFS)

Expand deepest unexpanded node

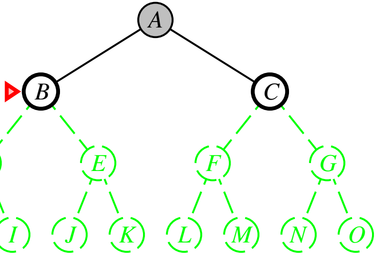
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

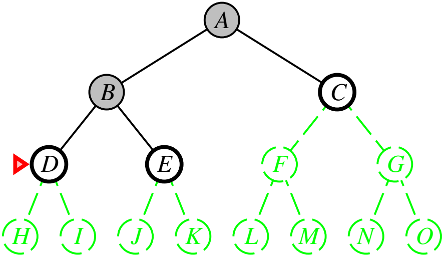
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

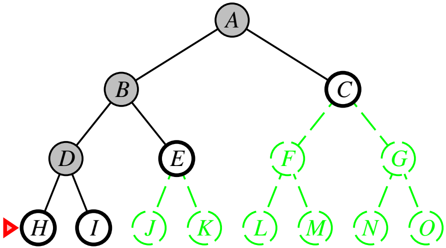
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

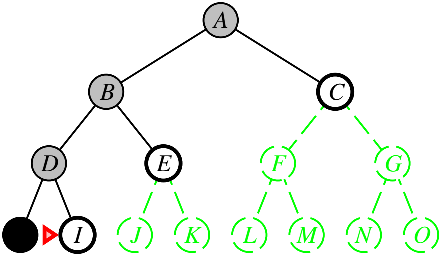
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

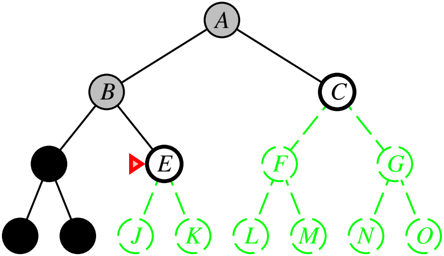
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

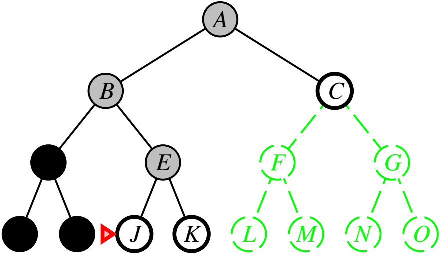
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

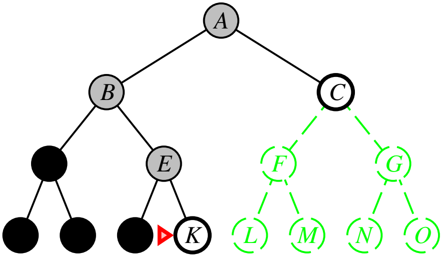
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

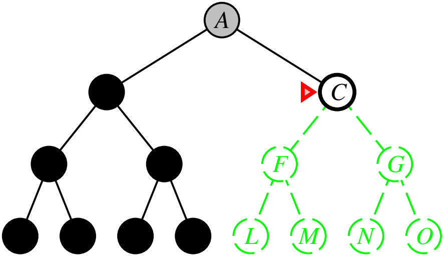
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

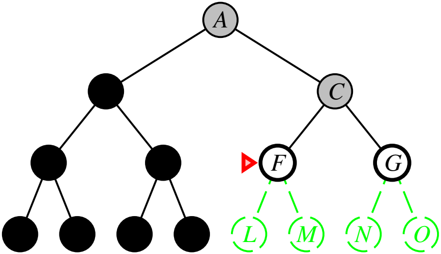
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

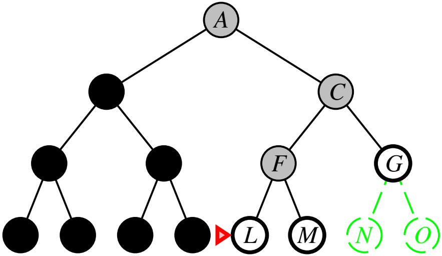
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

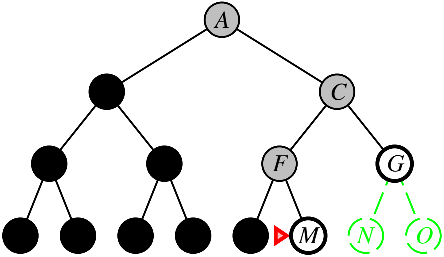
Implementation:
frontier = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

Expand deepest unexpanded node

Implementation:
frontier = LIFO queue, i.e., put successors at front



Properties of depth-first search

- Complete??
  No: fails in infinite-depth spaces, spaces with loops

  Modify to avoid repeated states along path (explored set, frontier)
  $\Rightarrow$ complete in finite spaces
- Time??
  $O(b^m)$, where m is the maximum depth of the state space

  terrible if m is much larger than d but if solutions are dense, may be much faster than BFS
- Space??
  $O(bm)$, i.e., linear space!
- Optimal??
  No

variant: backtracking (keep one successor node at a time)

Implementation: generate child node with Δ-change

Depth-limited search

- Depth-first search may not terminate (even for finite search space)
- Bound search with depth limit ℓ and report *cutoff* at the limit

Recursive implementation:

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Iterative deepening search (IDS)

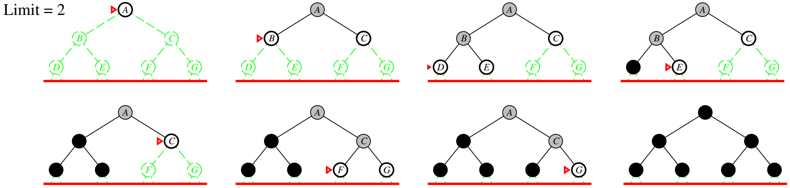
- Depth-Limited Search does not find goals beyond limit
- Increase the limit ℓ
- Repeat this systematically $\Rightarrow$ gain completeness

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for *depth* = 0 **to** ∞ **do**
    result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if *result* ≠ cutoff **then return** *result*

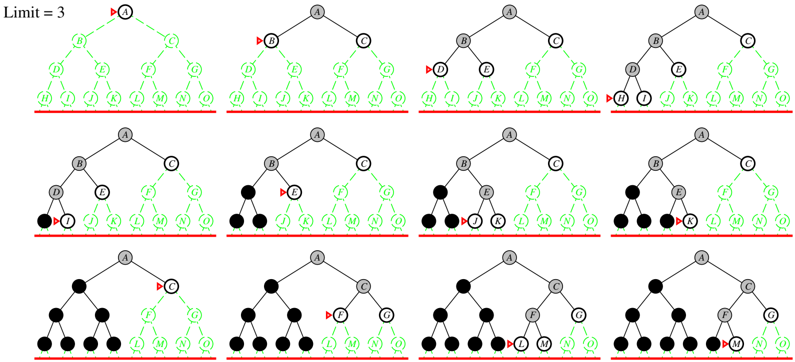
Iterative deepening search $\ell = 0$

Iterative deepening search $\ell = 1$

Iterative deepening search $\ell = 2$



Iterative deepening search $\ell = 3$



Properties of iterative deepening search

- Complete?? Yes
- Time??
  $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space??
  $O(bd)$
- Optimal??
  Yes, if step cost = 1; modifiable to explore uniform-cost tree
  comparison for $b = 10$, $d = 5$, solution at far right leaf:

$$\begin{aligned} N(\text{IDS}) &= 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450 \\ N(\text{BFS}) &= 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110 \\ N(\text{BFS}') &= 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100 \end{aligned}$$

  - BFS′ = BFS with goal test at expansion time is much worse
  - IDS is mildly ($\approx 11\%$) more expensive than BFS

Outline

1. Introduction

2. Problem Formulation

3. Basic Search Algorithms

4. Search Strategies

5. Uninformed Search Strategies

6. Conclusion

Uninformed search algorithms – assessment summary

- b branching factor
- d depth of the shallowest solution
- ℓ depth limit
- m maximum depth of the search tree

Criterion	BFS	UCS	DFS	DLS	IDS
Complete?	Yes$^\alpha$	Yes$^{\alpha,\beta}$	No	No	Yes$^\alpha$
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$
Optimal?	Yes$^\gamma$	Yes	No	No	Yes$^\gamma$

$^\alpha$ if b is finite
$^\beta$ if step costs $\geq \epsilon$ for positive ϵ
$^\gamma$ optimal if step costs are all identical

IDS is very attractive!

Summary

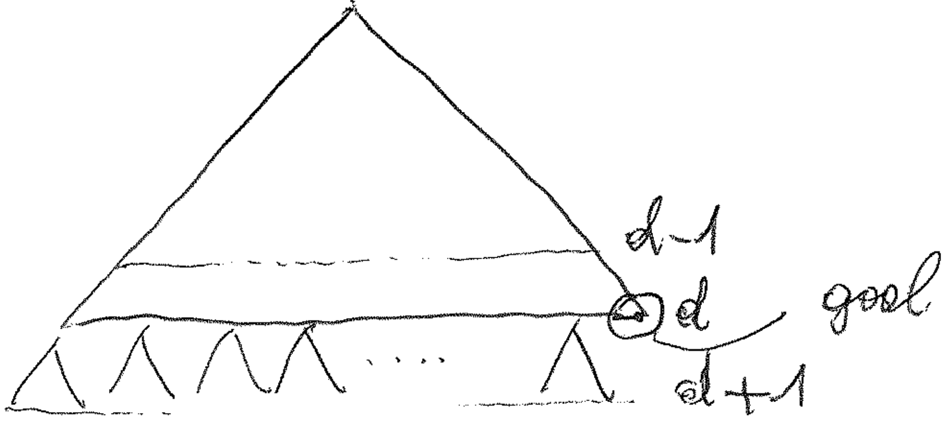
- Search problem formulation involves modelling
- Usually requires abstracting away real-world details to define a state space that can be feasibly explored
- Graph search can be exponentially more efficient than tree search
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

Breadth First Search

1) solution test at generation time

worst case

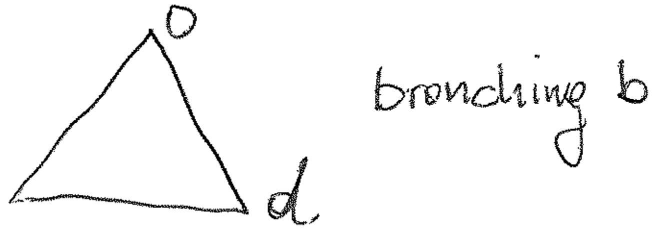
2) solution test at expansion time:



(almost) full further level d+1

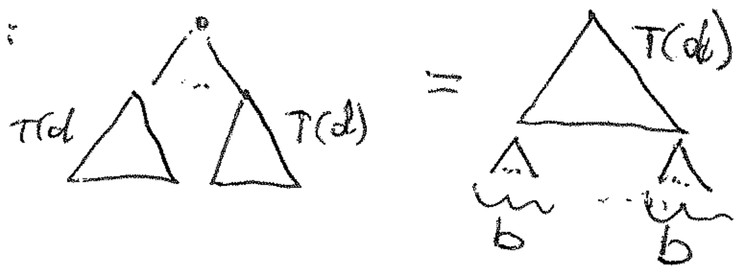
Breadth First Search

Worst case: full tree T



- Number of nodes $T(d) = \sum_{i=0}^{d} b^{i}$
- Closed form: $T(d) = \dfrac{b^{d+1}-1}{b-1}$

Why? $T(d+1)$:



$$1 + b \cdot T(d) = T(d) + b^{d+1}$$

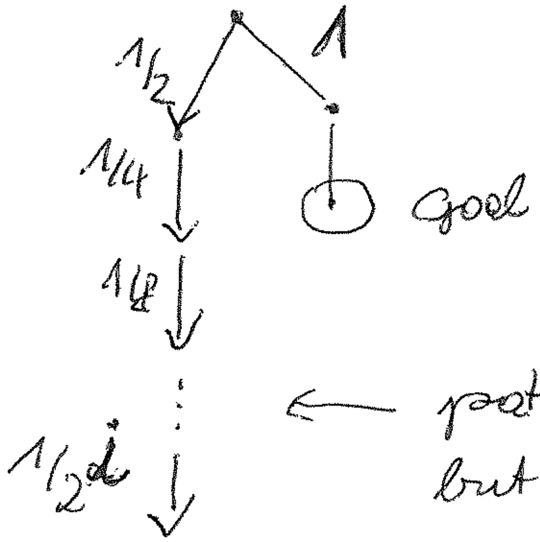
$$\Rightarrow T(d) = \frac{b^{d+1}-1}{b-1}$$

$$BFS(d) = T(d) \leq \frac{b^{d+1}}{b-1} = \frac{b}{b-1} \cdot b^{d}$$

Uniform Cost Search

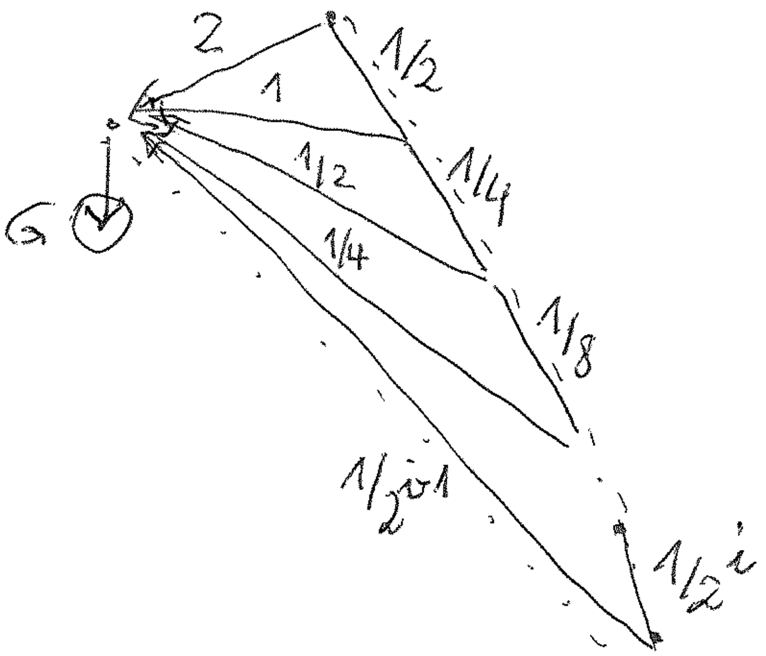
No lower cost bound.

a)



path cost converges to 1, but is always smaller

b) goal always reachable:



$1 + 2^{-i}$ = path cost

Iterative Deepening Search

$$
\begin{aligned}
IDS(d) &= \sum_{i=0}^{d} BFS(i) \\
&= \sum_{i=0}^{d} T(i) \\
&= \sum_{i=0}^{d} \frac{b^{i+1}-1}{b-1} \\
&\leq \sum_{i=0}^{d} \frac{b^{i+1}}{b-1} \\
&= \frac{b}{b-1} \cdot \underbrace{\sum_{i=0}^{d} b^{i}}_{T(d)} \\
&= \frac{b}{b-1} \cdot \frac{b^{d+1}-1}{b-1} \\
&\leq \frac{b}{b-1} \cdot \frac{b}{b-1} \cdot b^{d} \\
&= \left(\frac{b}{b-1}\right)^{2} \cdot b^{d}
\end{aligned}
$$

Compared to BFS, "extra cost" of $\frac{b}{b-1}$ factor

184.735 VU 2.0 Einführung in Künstliche Intelligenz

Problem Solving and Search (II)

Institute of Logic and Computation
Knowledge-Based Systems Group
Vienna University of Technology

SS 2023

slides by S. Russell, adapted by U. Egly and T. Eiter

AIMA Chapter 3, Section 5; Chapter 4, Sections 1-2

Outline

1. Heuristic Search
1.1 Greedy Search
1.2 A^*-search

2. Admissible Heuristics
2.1 Dominance
2.2 Relaxed Problems

3. Local Search
3.1 Hill Climbing
3.2 Simulated annealing
3.3 Local beam search
3.4 Genetic algorithms

4. Conclusion

Overview

Search: very important technique in CS and AI

Different kinds of search (refined picture)

- Deterministic search
  - uninformed ("blind") search strategies ✔
  - informed or "heuristic" search strategies
- Local search
- Search in game trees (not covered in this course)

In this unit: heuristic & local search

Outline

1. Heuristic Search

2. Admissible Heuristics

3. Local Search

4. Conclusion

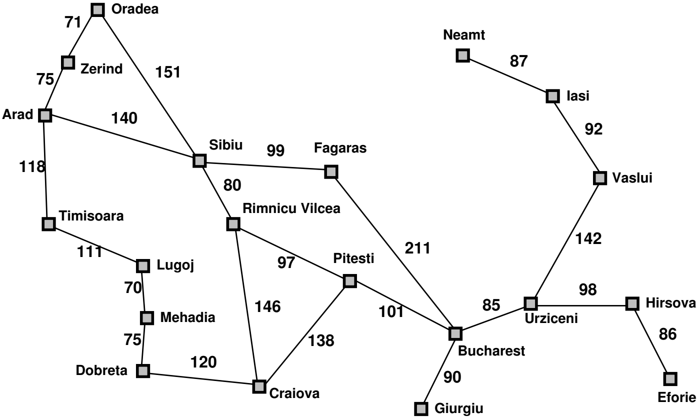
Outline

1. Heuristic Search
2. Admissible Heuristics
3. Local Search
4. Conclusion

Basic ideas of heuristic search

- Use problem- or domain-specific knowledge during search
- Evaluation function $f(n)$: **estimation** for a function $f^*(n)$
- Use a heuristic function $h \rightsquigarrow$ “desirability”
  (expand “most-desired” node next (= a kind of best-first search))
  - estimates minimal costs from state n to a goal state
    ($h(n) = 0$ always holds for a goal state)
  - computing $h(n)$ has low cost
- Consider two algorithms:
  - greedy search and
  - A^* search

Romania again



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search

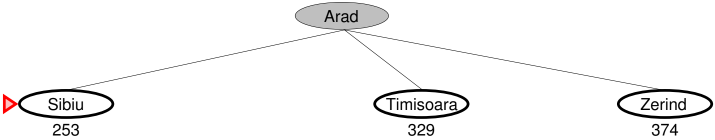
- Uses evaluation function $f(n) = h(n)$
- Does not take already “spent” costs into account
  (decisions are based on local information)
- Example: $h(n)$ = straight-line distance from n to Bucharest
- Greedy search expands the node with the smallest f-value (node appears to be closest to goal)

Greedy search for the travel example

1. Expand the only node Arad

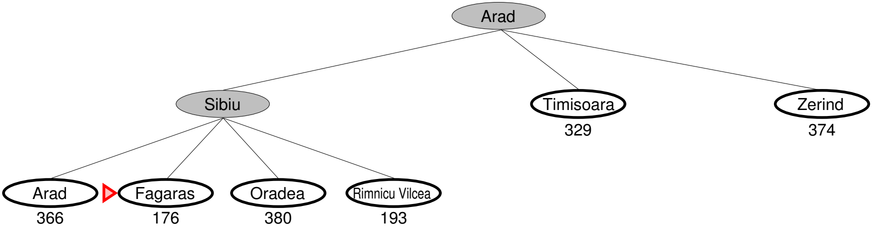
Greedy search for the travel example

2. Expand the node Sibiu, because it has the smallest f-value



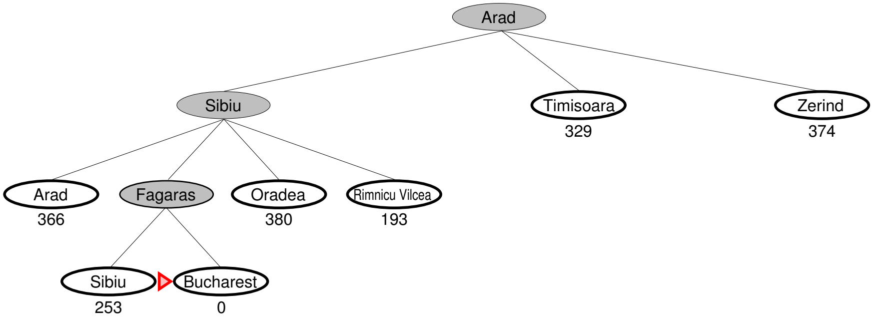
Greedy search for the travel example

3. Expand the node Fagaras, because it has the smallest f-value

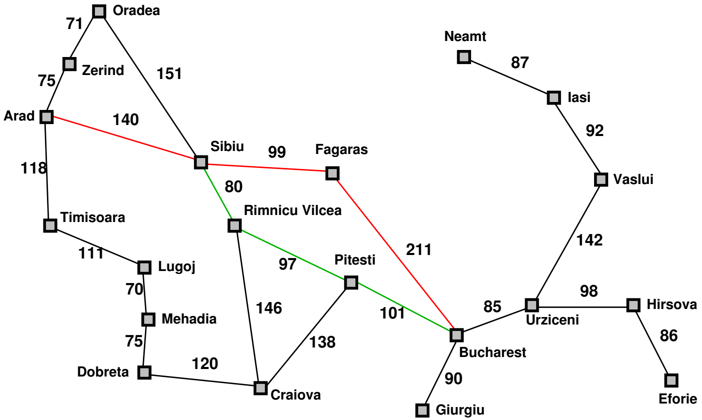


Greedy search for the travel example

Solution found: Arad–Sibiu–Fagaras–Bucharest (140+99+211=450)



An Alternative and optimal solution



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Non-optimal solution: Arad–Sibiu–Fagaras–Bucharest (450km)
Arad–Sibiu–Rimnicu Vilcea–Pitesti–Bucharest is shorter

Properties of greedy search

Complete?? No (can get stuck in loops)
Yes with loop checks

Time?? $O(b^m)$, i.e., exponential in m

Space?? $O(b^m)$, i.e., exponential in m (keep any node)

Optimal?? No

A^*-search

- Problems of greedy search: loops and non-optimality
  (induced by the use of only local info)
- A^*: Use evaluation function $f(n) = g(n) + h(n)$
  (and avoid expanding paths that are already expensive)
  - $g(n)$: path costs from start to n, (i.e., costs so far up to n)
  - $h(n)$: estimated cost to goal from n (like in greedy search)
  - $f(n)$: estimated total cost of path through n to goal
- $h(n)$ has to be admissible

Definition 1

A heuristics h is admissible, if for every node n the following holds:

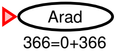
1. $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n (h is "optimistic");
2. $h(n) \geq 0$;
3. $h(g) = 0$ for every goal g (follows from 1 and 2).

`http://en.wikipedia.org/wiki/A*_search_algorithm`

A^*-search for the travel example

We use the straight-line distance as an optimistic heuristic

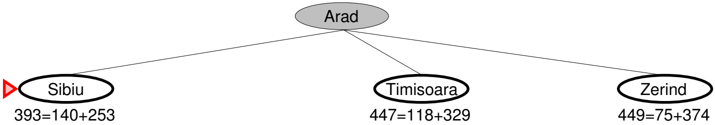
1. Expand the only node Arad



A^*-search for the travel example

We use the straight-line distance as an optimistic heuristic

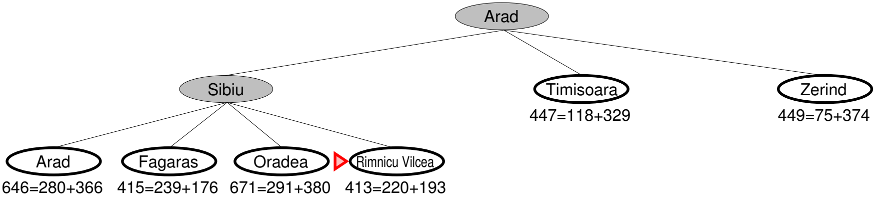
2. Expand the node Sibiu, because it has the smallest f-value



A^*-search for the travel example

We use the straight-line distance as an optimistic heuristic

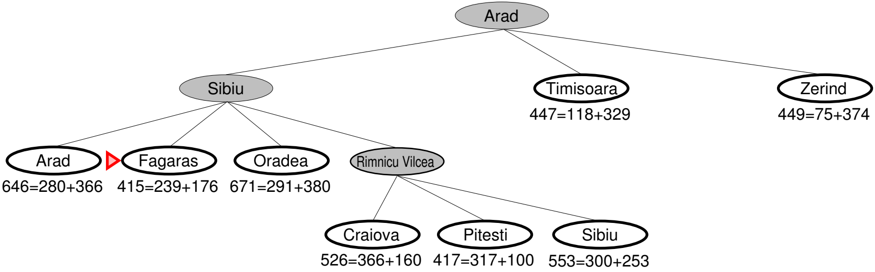
3. Expand the node Rimnicu Vilcea (and not Fagaras as in GS)



A^*-search for the travel example

We use the straight-line distance as an optimistic heuristic

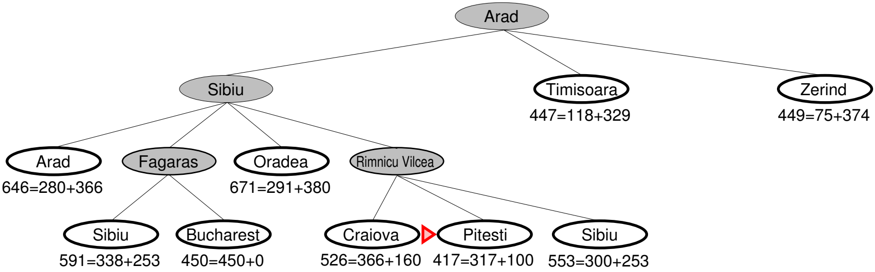
4. Expand the node Fagaras, because it has the smallest f-value



A^*-search for the travel example

We use the straight-line distance as an optimistic heuristic

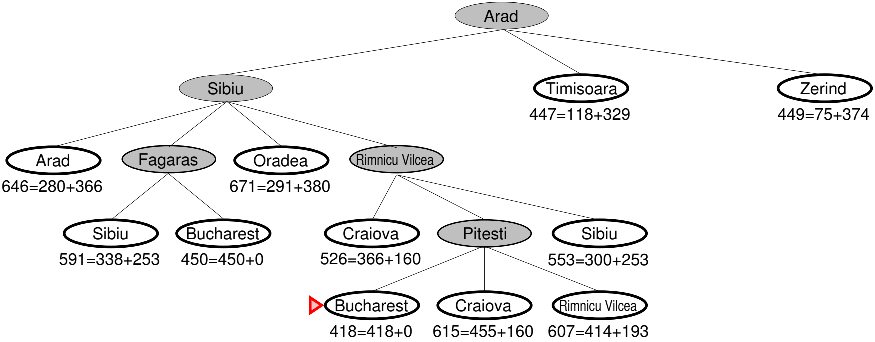
5. Expand the node Pitesti, because it has the smallest f-value



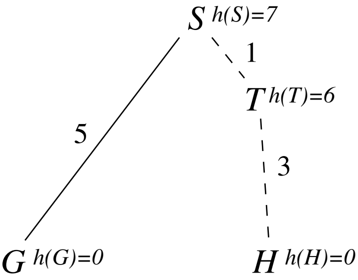
A^*-search for the travel example

We use the straight-line distance as an optimistic heuristic

Solution: Arad–Sibiu–Rimnicu Vilcea–Pitesti–Bucharest
Since f-values of all other open nodes are bigger $\leadsto$ terminate



Do we really need admissible heuristics?



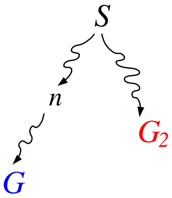
- S is the start state
- h is not optimistic
- S expands immediately to G and T
- $f(G) = 5$ and $f(T) = 7$, so we are done
- Solution G is obviously not optimal
- Hence, heuristics must be optimistic!

Theorem 2

If h is admissible, then A^* using tree search is optimal.

Optimality of A^*: the standard proof

- Let h be admissible, and G an optimal goal
- Suppose some suboptimal goal G_2 has been generated
- Let n be an unexpanded node on an optimal path to G



$f(G_2)$	$=$	$g(G_2)$	since $h(G_2) = 0$ (G_2 is a goal!)
	$>$	$g(G)$	as G_2 is suboptimal
	$=$	$g(n) + h^*(n)$	as $h^*(n)$ is optimal to reach G
	$\geq$	$g(n) + h(n)$	as h is admissible
	$=$	$f(n)$	

Since $f(G_2) > f(n)$, A^* will never select G_2 for expansion

Problems with optimality of A^* using graph search

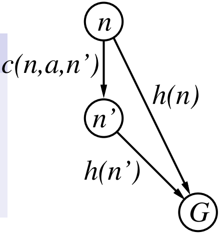
- A^* does not require for a path from start S to n that $g(n)$ is minimal
- No problem for tree search (there is only one path)
- In graph search (GS), we can reach nodes with non-optimal costs
- GS can discard optimal paths, even if h is admissible
  ➥ optimality is lost

Two possibilities to fix the problem

1. change algorithm and add more complicated bookkeeping
   (But what's the effect on the run time?)
2. impose a stronger restriction on h: consistency

   ➥ Implies that f-value is non-decreasing on every path

Optimality of A^* using graph search



Definition 3

A heuristic is consistent if, for every node n, every successor n' of n and every operator a,

$$h(n) \leq c(n,a,n') + h(n')$$

holds, where $c(n,a,n')$ are the path costs for a.

Lemma 4

If h is consistent, then f is non-decreasing along every path:

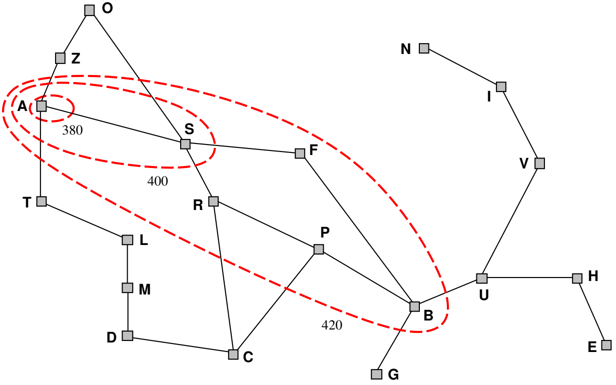
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \; = \; f(n). \end{aligned}$$

Theorem 5

If h is consistent, then A^* using graph search is optimal.

Optimality of A^*

- A^* expands nodes in order of increasing f-values
- The "f-contours" of nodes are added gradually (cf. BFS adds layers)
- Contour i has all nodes with $f \leq f_i$, where $f_i < f_{i+1}$



Properties of A^*

Complete?? Yes unless there are infinitely many nodes n with $f(n) \leq f(G)$

Time?? Exponential in $\epsilon \times d$

- $\epsilon = \frac{h(n_0) - h^*(n_0)}{h^*(n_0)}$ ... relative error ($h^*(n_0) = C^*$)
- d = solution depth (= path length)

Space?? Exponential (keeps all nodes in memory)

Optimal?? Yes

Note: A^* expands

- **all** nodes with $f(n) < C^*$
- **some** nodes with $f(n) = C^*$
- **no** nodes with $f(n) > C^*$
- fewest nodes safely possible if h is consistent

Outline

1. Heuristic Search

2. Admissible Heuristics

3. Local Search

4. Conclusion

Admissible heuristics

Consider the following heuristics for the 8-puzzle:

- $h_1(n)$: number of misplaced tiles
- $h_2(n)$: total Manhattan distance
  (no. of squares ($\leftrightarrow$ and $\updownarrow$) from desired location of each tile)

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

$$h_1(\text{Start State}) = 6$$

$$h_2(\text{Start State}) = 4+0+3+3+1+0+2+1 = 14$$

Dominance

Definition 6

For admissible heuristics h_1 and h_2, we say h_2 dominates h_1, if $h_2(n) \geq h_1(n)$ for every node n.

- If h_2 dominates h_1, it is better for search
- Typical search costs (8-puzzle)

$d = 14$	DFIDS requires 3,473,941 nodes
	A^* with h_1 requires 539 nodes
	A^* with h_2 requires 113 nodes
$d = 24$	DFIDS requires $\approx$ 54,000,000,000 nodes
	A^* with h_1 requires 39,135 nodes
	A^* with h_2 requires 1,641 nodes

Proposition 7

If h_1, h_2 are admissible heuristics, then also the heuristic $h(n) = \max(h_1(n), h_2(n))$ is admissible, and it dominates h_1, h_2.

Admissible heuristics from relaxed problems

- Derive admissible heuristics from exact solution cost of a relaxed version of the problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution
- Key point: optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem

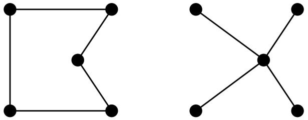
Relaxed problems cont'd

Well-known example: traveling salesperson problem (TSP)

"Find the shortest tour visiting all cities exactly once."

Relaxation: should be polynomially computable

- view as graph $G = (V, E)$; must be connected
- no superfluous edges: spanning tree (ST) a subgraph T of G that is a tree and connects all vertices



- Minimum spanning tree (MST): smallest cost
  - (undirected) MST is polynomially computed (e.g., in time $O(|E| \log |E|)$ resp. $O(|V|^2)$ by Kruskal's / Prim's algorithm)
  - MST cost is a lower bound on the shortest (open) tour

Outline

1. Heuristic Search

2. Admissible Heuristics

3. Local Search

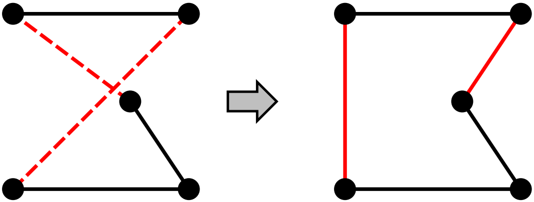
4. Conclusion

Iterative improvement

- Up to now, search constructed a path of actions from initial to goal state (actions change a current state)
- For many (also optimization) problems, paths are irrelevant; a goal state (described implicitly) is the solution
- Then
  - state space = set of “complete” configurations;
  - find **optimal** configuration, e.g., TSP; or
  - find configuration satisfying constraints, e.g., timetable
- In such cases, one can use iterative improvement algorithms
  - keep a single “current” state
  - try to improve it
- Constant space, suitable also for unknown environments (**online** search)

Example: Traveling Salesperson Problem

- Start with any complete tour, perform pairwise exchanges

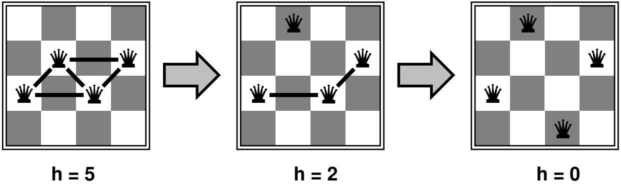


- Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: n-queens

- Put n queens on an $n \times n$ board with no two queens threatening each other (i.e., not on the same row, column, or diagonal)

- Move a queen to reduce number of conflicts (attacks), e.g., in column



- Terminates in few steps (at most quadratic, but much fewer in reality)

Agenda

- Hill-climbing
- Simulated annealing
- Genetic algorithms (briefly)
- Local search in continuous spaces (omitted)

Hill-climbing (or gradient ascent/descent)

"Like climbing Mount Everest in thick fog with amnesia"

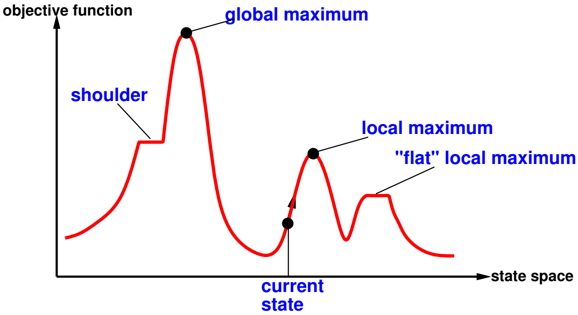
```
function HILL-CLIMBING( problem)
returns a state that is a local maximum

   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

Hill-climbing, cont'd

Useful to consider state space landscape



- Sideways moves: ☺ escape shoulders, ☹ loop on flat maxima
- Random-restart hill climbing: restart after step limit
  overcomes local maxima—trivially (expectedly) complete
- Alternative neighbor selection: stochastic, or first-choice

Example: 8-queens

- state space: $8^8 \approx$ 17 million states
- standard hill climbing (random initial state)
  - 4 steps on average, if successful
  - 3 steps on average, if unsuccessful
  - chance of success: 14%
- sideways moves: limit 100
  - $\approx$ 21 steps, if successful
  - ≈ 64 steps, if unsuccessful
  - chance of success: 14% $\rightsquigarrow$ 94%
- random restart hill climbing:
  - find solutions for large n (three millions) within short time (one minute)

Simulated annealing

- Devised in the 1950s for physical process modeling
- Widely used (VLSI layout, airline scheduling, etc.)
- **Idea**: escape local maxima by
  - allowing some “bad” moves
  - **but gradually decrease their size and frequency**
- corresponds to “cooling off” process of materials
  - value = energy E

Simulated annealing, cont'd

```
function SIMULATED-ANNEALING( problem, schedule)
returns a solution state

   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

Properties of simulated annealing

- Boltzmann distribution: state occupation probability of a thermodynamical system within fixed temperature T:

$$p(x) = \alpha e^{-\frac{E(x)}{kT}}$$

  - x ... state
  - α ... degeneracy (= # of states x' with same energy as x)
  - $E(x)$ ... energy
  - k ... Boltzmann constant

- If T is decreased slowly enough: reach best (lowest energy) state x^* with probability approaching 1, as

$$e^{-\frac{E(x^*)}{kT}} / e^{-\frac{E(x)}{kT}} = e^{\frac{E(x)-E(x^*)}{kT}} \gg 1 \quad \text{for small } T$$

- Above, we maximize (change sign)

  Is this necessarily an interesting guarantee??

  “Slowly enough” can be worse than exhaustive search

Local beam search

- **Idea**:
  - keep k states instead of 1
  - choose top k of all their successors
- Not the same as running k searches in parallel! Searches that find good states recruit other searches to join them
- **Problem**: quite often, all k states end up on same local hill
- **Idea**: choose k successors randomly, biased towards good ones

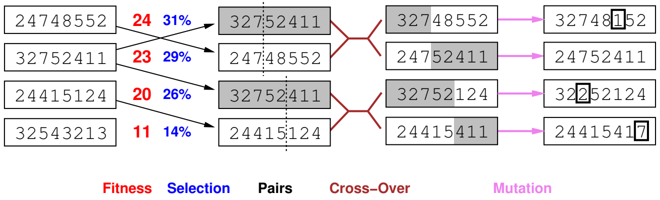
  Observe the close analogy to natural selection!

Genetic algorithms

Genetic algorithm (GA) =

stochastic local beam search + successors from *pairs* of states

8-queens: $r_1 r_2 \cdots r_8$ r_i= row index of queen in column i



Fitness: number of nonattacking pairs

Selection: probability (here proportional to fitness)

Crossover: randomly choose pairs for mating + crossover point

Mutation: with low probability, move random queen i randomly in column

Pseudocode Formulation

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

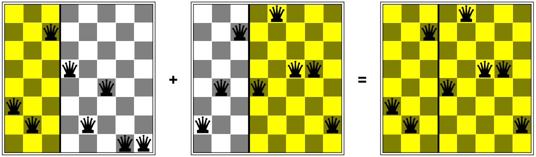
```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Observations

- GAs require states encoded as strings (Genetic Programs use programs/lists of instructions as individuals)
- Crossover can produce solutions quite distant from their parents

32752411 + 24748552 on columns 1-3 and 4-8



- Crossover helps **if and only if substrings are meaningful components (blocks)**
  - no advantage if string could be randomly shuffled
- GAs $\neq$ evolution: e.g., real genes themselves encode replication machinery!

Outline

1. Heuristic Search

2. Admissible Heuristics

3. Local Search

4. Conclusion

Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest $h(n)$
  - Incomplete and not always optimal
- A^* search expands lowest $g(n) + h(n)$
  - Complete and optimal
- Derive admissible heuristics from exact solution of relaxed problems
- Local search uses different methods to find better (locally optimal) solutions
  - paths are not important
  - use constant space
- A variety of further methods and refinements exists

Dijkstra's Algorithm

Recall: shortest paths from one node to
all nodes in a graph

variants: one node to one node
all nodes to all nodes

UCS is analogous, uses $f(n) = g(n)$ $(h(n) = 0)$

Dijkstra's Alg: • no heuristics
• geared to finite, explicit graph

for consistent heuristics $h(n)$:

$$h(n) \leq c(n, n') + h(n')$$

$$\Rightarrow \quad 0 \leq \underbrace{c(n,n') + h(n') - h(n)}_{= d(n,n')}$$

A* resembles Dijkstra's Alg.

For $h(n) = h(n') = 0$, "classical"

184.735 VU 2.0 Einführung in Künstliche Intelligenz

Learning from Observations

Institute of Logic and Computation
Knowledge-Based Systems Group
Vienna University of Technology

SS 2023

slides by S. Russell, adapted by T. Eiter

AIMA Chapter 2, Section 4.6-4.7; Chapter 18, Sections 1-3

Outline

1. Learning
1.1 Learning Agents
1.2 Representation and Modes

2. Inductive Learning

3. Decision Tree Learning
3.1 Choosing an Attribute
3.2 Problems and Generalizations

4. Measuring Learning Performance

5. Conclusion

Outline

1. Learning

2. Inductive Learning

3. Decision Tree Learning

4. Measuring Learning Performance

5. Conclusion

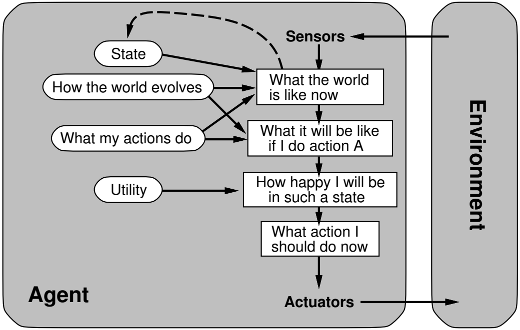
Learning

"Nobody is perfect" (Osgood Fielding, 'Some like it hot')

- Learning modifies the agent's decision mechanisms to improve performance
- Learning is essential for unknown environments, i.e., when designer lacks omniscience
  - can not anticipate all situations
  - can not anticipate all changes, updates
- Learning is useful as a system construction method, i.e., expose the agent to reality rather than trying to write it down
  - programming might be "impossible" (e.g., face recognition)

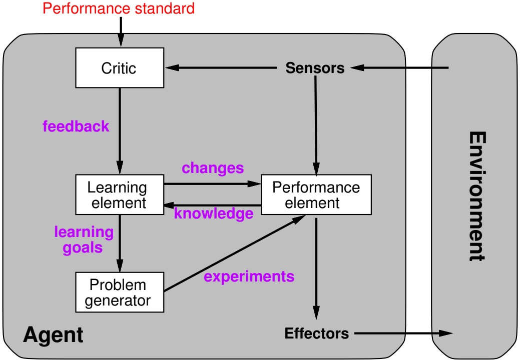
Suggestion by A. Turing (1950)
build learning machinery and teach it

Recall: Agent Architecture



- rich model of utility-based agent
- the components in between sensors (input) and actuators (output) can be abstracted to a *performance element*

Learning Agents (Architecture)



- **Performance element:** select external actions
- **Critic:** performance / result assessment
- **Learning element:** make improvements
- **Problem generator:** suggest actions for new experiences

Learning Element

The learning element (LE) makes changes to the “knowledge components” of an agent architecture (reflex, logical, utility, etc.)

The LE design is dictated by

- what type of performance element is used
- which functional component is to be learned
- what representation is chosen for the functional component
- what kind of feedback is available

Example scenarios:

Performance element	Functional component	Representation	Feedback
Alpha-beta search	Evaluation function	Weighted linear function	Win/loss
Logical agent	Transition model	Successor-state axioms	Outcome
Utility-based agent	Transition model	Dynamic Bayes net	Outcome
Simple reflex agent	Percept-action function	Neural net	Right action

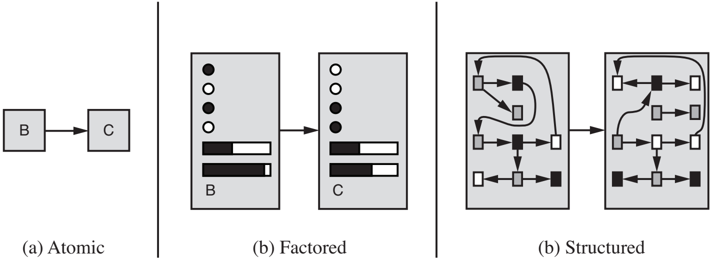
Note: Learning the transition model can be “supervised” if the environment is observable

Taxi Example

Drive a taxi

- **Performance element:** knowledge about driving, and procedures to do so
- **Critic:** assess status, reactions of customer / other drivers / pedestrians / bikers / police
- **Learning element:** create e.g. rule not to turn steering wheel to quickly; break softly etc.
- **Problem generator:** try brakes on slippery road; push back with side (wing) mirror only

Representation



Different levels of granularity to represent states of the "world"

- **Atomic**: states e.g. identified with labels (names)
  like a "black box", "image";
  search algorithms may treat states in this way
- **Factored**: attributes (variables) with values
  exploited e.g. in planning, constraint satisfaction
- **Structured**: dependence model between attributes
  e.g. logical knowledge bases, Bayesian networks

Learning Modes

Important for learning: feedback

- Unsupervised learning: no explicit feedback
  - most common: **clustering** of “similar” examples, concept formation
- Reinforcement learning: occasional rewards or punishments in terms of payoff
- Supervised learning: correct answers for **each** instance
  - requires “teacher”
  - reward $\neq$ correct answer
  - issue: choose the examples (teaching set/sequence)

In practice: semi-supervised learning (mix of labeled and unlabeled examples)

Issues

- noise
- inaccuracies (biased data)

Outline

1. Learning

2. Inductive Learning

3. Decision Tree Learning

4. Measuring Learning Performance

5. Conclusion

2. Inductive Learning (also known as Science)

Simplest form: learn a function from examples (tabula rasa)

- f is the target function (unknown)
- An example is a pair (x, y), where $y = f(x)$.
  E.g.,

$$\begin{array}{c|c|c} O & O & X \\ \hline & X & \\ \hline X & & \end{array} \;,\quad +1$$

Problem:

> Given a (finite) training set $(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_N,y_N)$ of examples, find a function h that approximates f ($h \approx f$).

- h is called hypothesis
- usually, h must be from a restricted class of functions (hypothesis space)
- if $h = f$ is possible, the learning problem is realizable

Inductive Learning, cont'd

Type of output value to learn:

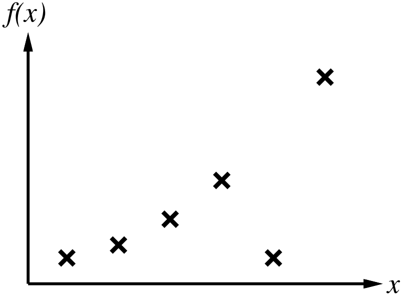
- classification: one of finitely many values (e.g. win, loss)
- regression: a (real) number (e.g. price)

This is a highly simplified model of real learning:

- Ignores prior knowledge
- Assumes a deterministic, observable "environment"
- Assumes examples are *given*
  - selection of new examples is challenging (maximize utility)
- Assumes that the agent *wants* to learn target f — why?

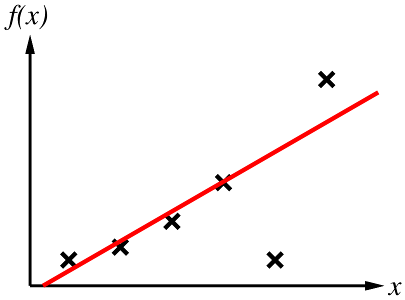
Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)
- E.g., curve fitting:



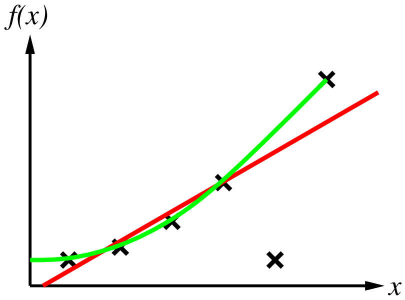
Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)
- E.g., curve fitting: linear function



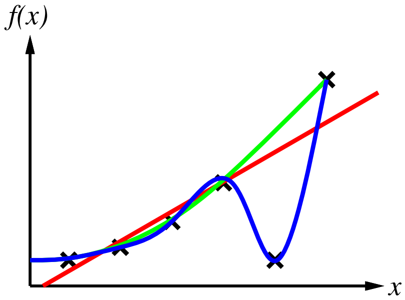
Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)
- E.g., curve fitting: quadratic function



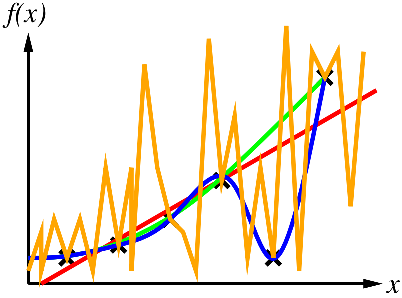
Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)
- E.g., curve fitting: interpolation polynomial (degree $<k$ for k examples)



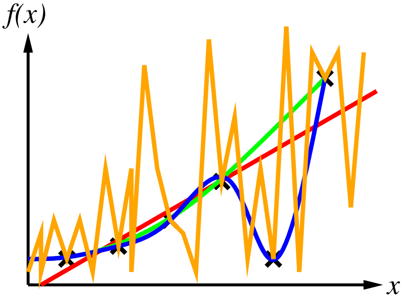
Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)
- E.g., curve fitting: complicated function



Inductive Learning Method

- Construct/adjust h to agree with f on training set
  (h is consistent if it agrees with f on all examples)
- E.g., curve fitting:



Ockham's razor: maximize simplicity under consistency

Entia non sunt multiplicanda praeter necessitatem.

Complex vs Simple Hypotheses

- Tradeoff between
  - complex hypotheses that fit the data well, and
  - simpler hypotheses that might generalize better
- Tradeoff between
  - expressiveness of the hypothesis space, and
  - finding a good hypothesis in it

Problem: computational complexity

Outline

1. Learning
2. Inductive Learning
3. Decision Tree Learning
4. Measuring Learning Performance
5. Conclusion

Decision Trees

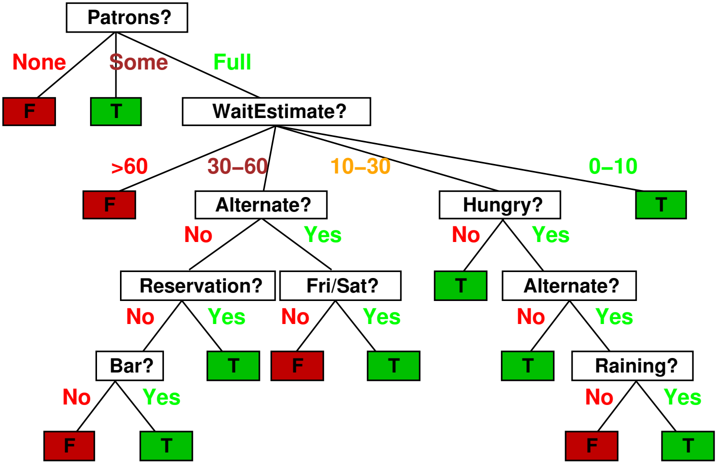
- Decision tree: one possible representation for hypotheses
- Uses factored (attribute valued) representation
- Takes as input a vector $\mathbf{x} = (x_1, ..., x_m)$ of values for attributes $A_1, \ldots, A_m$, and returns an output value y ('decision')
- The decision is reached by a sequence of tests, arranged in a tree
  - each internal node n amounts to a test $A_i?$ of the value of some A_i;
  - node n has for each possible value v_{ik} of A_i one child node
  - branch to the child node for $v_{ik} = x_i$ (start at root)
  - each leaf node gives a decision
- Boolean classification: y is positive (T) or negative (F)
- Note: the reason for the output y is understandable
  (not so e.g. for neural nets)

Example: Restaurant Waiting

- Decide whether to wait (Yes/No) for a table in a restaurant
- Attributes:
  - *Alternate*: suitable alternative restaurant close by
  - *Bar*: comfortable bar area to wait in
  - *Fri/Sat*: true on Fridays/Saturdays
  - *Hungry*: whether we are hungry
  - *Patrons*: how many people are there (none, some, full)
  - *Price*: $, $$, $$$
  - *Raining*: whether it is raining
  - *Reservation*: whether we made a reservation
  - *Type*: kind of restaurant (French, Italian, Thai, or burger)
  - *WaitEstimate*: wait estimate by the host (0-10, 10-30, 30-60, >60 mins)
- GoalPredicate: *WillWait*

Restaurant Waiting, cont'd

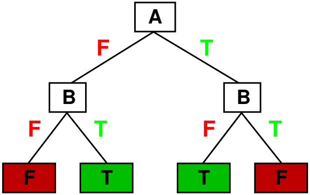
The "true" tree for deciding whether to wait:



Expressiveness

- Decision trees can express *any* function of the input attributes.
- E.g., for Boolean functions, truth table row $\rightarrow$ path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- There is a trivial consistent decision tree for any training set: one path to leaf for each example
  - it probably won't generalize to new examples
  - need not be a *compact* decision tree
  - in some cases, no much more compact decision tree might be possible (e.g., majority function)

Hypothesis Spaces

How many non-equivalent decision trees with n Boolean attributes??

= number of Boolean functions with n variables

= number of distinct truth tables with 2^n rows

= 2^{2^n}

E.g., for $n = 6$, there are $2^{2^n} =$ 18,446,744,073,709,551,616

note: and many *equivalent* decision trees do exist!

How many conjunctive hypotheses (e.g., $Hungry \land \neg Rain$)??

Each attribute can be (i) in positive, (ii) in negative, or (iii) out
$\Longrightarrow$ 3^n distinct conjunctive hypotheses

A more expressive hypothesis space increases

- the chance that target function can be expressed; ☺
- the number of hypotheses consistent with the training set
  $\Longrightarrow$ may get worse predictions. ☹

Decision Tree Learning

- Aim: find a small tree consistent with the training examples *E*
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

> **function** DTL(*examples, attributes, parent_examples*)
> **returns** a decision tree
>
>   **if** *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
>   **else if** all *examples* have the same classification **then return** the classification
>   **else if** *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
>   **else**
>   *A* ← $\mathrm{argmax}_{B\,\in\,attributes}$ IMPORTANCE(*B, examples*)
>   *tree* ← a new decision tree with root test *A*
>   **for each** value v_k of *A* **do**
>   *exs* ← $\{\, e \in examples \mid e.A = v_k \}$
>   *subtree* ← DTL(*exs*, $attributes \setminus \{A\}$, *examples*)
>   add a branch to *tree* with label $A = v_k$ and subtree *subtree*
>   **return** *tree*

attributes is empty: nondeterministic domain; hidden attributes; noise

- call DTL(E, $\{A_1, \ldots, A_m\}$, $\{\,\}$), for attribute set $A = \{A_1, \ldots, A_m\}$

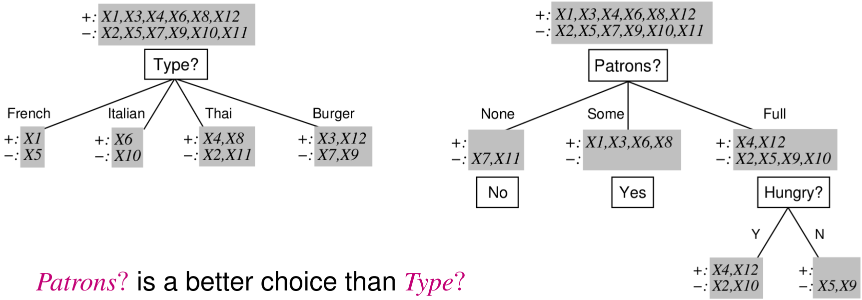
Restaurant Waiting, cont'd

Build a waiting tree from example data:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Choosing an Attribute: IMPORTANCE(*B, examples*)

- Desired: compact, "flat" decision tree
- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



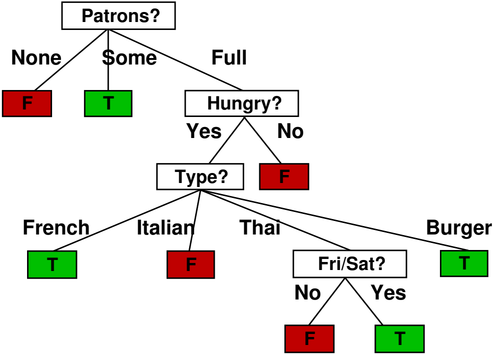
Patrons? is a better choice than *Type?*

- Greedy approach: pick most promising in next step

  simple notion: maximize classifications ad hoc

Example, cont'd

Decision tree learned from the 12 examples:



- Substantially simpler than "true" tree – a more complex hypothesis isn't justified by small amount of data
- Some attributes are not relevant (*Raining*, *Reservation*)
- Patterns detectable (if hungry, wait for Thai food on weekends)

Choosing an Attribute: Information Gain

Maximum ad hoc classification is not necessarily good

Better: consider information gain, defined by entropy

- The answer to a question provides information: the more clueless we are about it, the more information it contains
- Scale: 1 bit = answer to Boolean question with prior $\langle \frac{1}{2}, \frac{1}{2} \rangle$ (uniformly random)
- Information in a lottery outcome (e.g., Euro Millions)?
- Information in an answer with prior $\langle P_1, \ldots, P_n \rangle$ is its entropy

$$H(\langle P_1, \ldots, P_n \rangle) = \sum_{i=1}^{n} -P_i \log_2 P_i \quad \left(= \sum_{i=1}^{n} P_i \log_2 \frac{1}{P_i} \right).$$

- Special case $n = 2$: $P_2 = 1 - P_1$

$$H(\langle P_1, P_2 \rangle) = -P_1 \log_2 P_1 - (1 - P_1) \log_2(1 - P_1) =: B(P_1)$$

Choosing an Attribute: Information Gain, cont'd

- Assume E has p positive, n negative examples ($P_1 = p/(p+n)$)
  $\Longrightarrow$ need $B(p/(p+n))$ bits to classify a new example

  restaurant example: $p = n = 6$ thus 1 bit
- Value v_k of attribute A creates $E_k \subseteq E$ with p_k pos. and n_k neg. examples $\Longrightarrow$ need $B(p_k/(p_k+n_k))$ bits to classify a new example
- Hopefully, E_k needs less information than E
- The *expected number of bits per example over all* E_k is

$$Rem(A) = \sum_k \frac{p_k+n_k}{p+n} B(p_k/(p_k+n_k))$$

  Example: $Rem(Patrons) = 0.459$ bits, $Rem(Type) = 1$ bit

Principle

- choose A with minimal $Rem(A)$, or equivalently,
- choose A with maximal $Gain(A) = B(p/(n+p)) - Rem(A)$

Overfitting

Problem of all learners: overfitting the training examples

- hypothesis may consider irrelevant attributes
  - e.g., color of dices
- may break with new examples (recall Ockham's razor!)
- more likely to happen with many attributes
- less likely to happen with more examples

Remedies:

- Decision tree pruning
  - Idea: Prevent split on "irrelevant" attributes, using information gain and statistical methods
- Cross-validation
  - Use part of the data for training, part for testing

Generalizations (see book)

- Missing Data:
  some attribute value is unknown
  - consider all possible values, weighted with frequency to reach the test node (by the examples)
- Multi-valued attributes:
  attributes with many values may spoil compact decision tree (e.g., social security number, time stamp)
  - e.g. normalize $Gain(A)$ with the entropy of A

$$(= \sum_k P(A = v_k) \log_2 P(A = v_k))$$

- Continuous, integer valued attributes:
  use split points (e.g., $age > 35$) that give highest information gain
- Continuous valued output attributes:
  regression trees whose leaves are functions of a subset of the attributes

Outline

1. Learning

2. Inductive Learning

3. Decision Tree Learning

4. Measuring Learning Performance

5. Conclusion

Measuring Learning Performance

How do we know that $h \approx f$? (Hume's *Problem of Induction*)

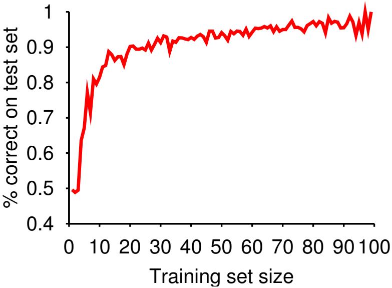
- Use theorems of computational/statistical learning theory
- Try h on a new test set of examples (use same distribution over example space as training set)

Learning curve = % correct on test set as a function of training set size

restaurant data,

graph averaged over

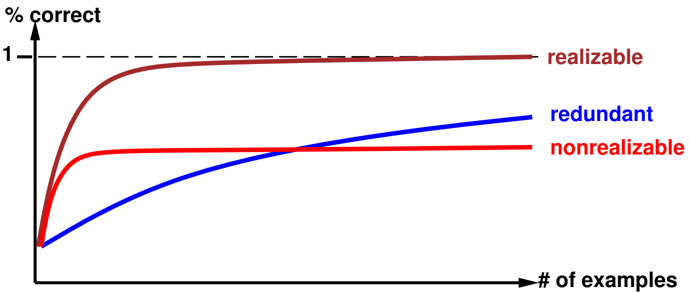
20 trials



Performance Measurement, cont'd

Learning curve depends on

- realizability of the target function
  non-realizability can be due to
  - missing attributes
  - too restrictive hypothesis space (e.g.,linear function)
- redundant expressiveness (e.g., loads of irrelevant attributes)



Outline

1. Learning

2. Inductive Learning

3. Decision Tree Learning

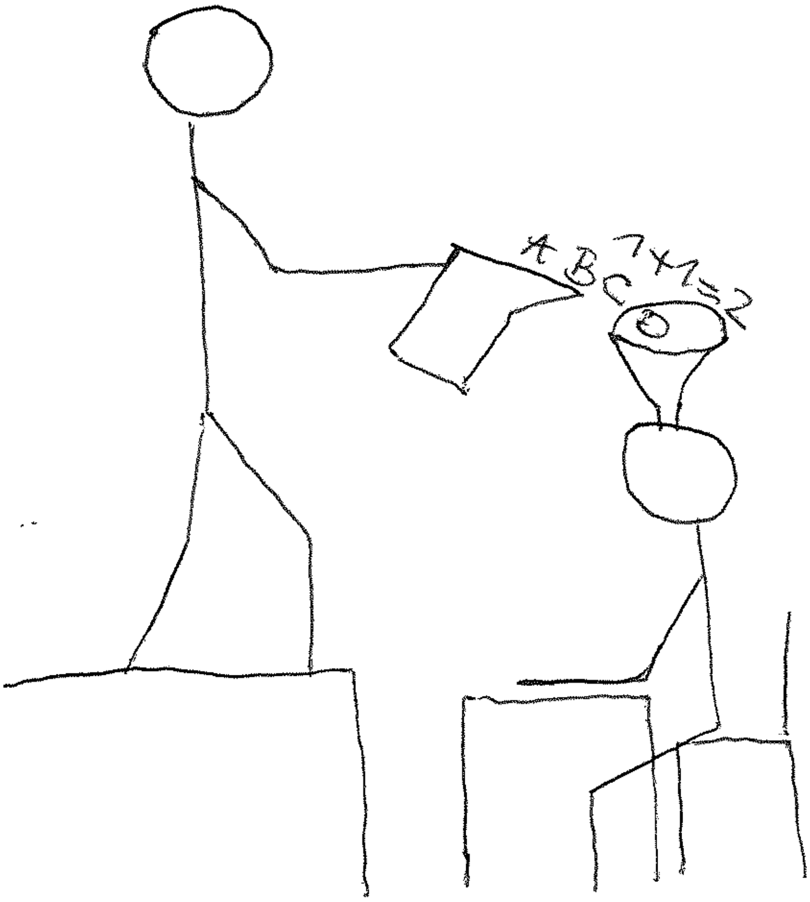
4. Measuring Learning Performance

5. Conclusion

Summary

- Learning needed for unknown environments, lazy designers
- Learning agent = performance element + learning element
- Learning element depends on
  - type of performance element
  - available feedback
  - type of component to be improved
  - representation
- For supervised learning, the aim is to find a simple hypothesis that is approximately consistent with training examples
- Decision tree learning using information gain
- Learning performance = prediction accuracy measured on test set as a function of the training set size

Learning in 17th Century



Nuremberg Funnel

Entropy $H(P_1, \ldots, P_n)$

1) n elements each probability $P_i = \frac{1}{n}$

- n groups of single element.
- need $\log n$ bits to distinguish the groups
- on average $\sum_{i=1}^{n} P_i \cdot \log n = \log n$ bits

2) Different probabilities $P_i = \frac{m_i}{n'}$

- each group has m_i elements, n' elements in total
- different elements in group yield same code
- need $\log m_i$ bits to discriminate in group.

Expected:

$$\sum_{i=1}^{n} P_i (\log n' - \log m_i) = \sum_{i=1}^{n} P_i \cdot \log \frac{n'}{m_i}$$

$$= -\sum P_i \cdot \log P_i$$

Lottery: Euro Millions

choose 5 numbers from $1, 2, \dots, 50$

$\Rightarrow \binom{50}{5}$ outcomes

<u>naive encoding</u>:

sequence, e.g. 1, 4, 20, 30, 46

- 6 bits per number ($2^6 = 64$)
- needs $5 \times 6 = 30$ bits

<u>Entropy</u>: $H(p_1, \dots, p_m)$, $m = \binom{50}{5}$ $p_i = 1/m$

$$H(p_1, \dots, p_m) = \log_2 \binom{50}{5}$$

Recall: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

$$\binom{50}{5} = \frac{50!}{45!\,5!} = \frac{46 \cdot 47 \cdot 48 \cdot 49 \cdot 50}{5!} = 5 \cdot 2 \cdot 46 \cdot 47 \cdot 49$$

$$\leq 64^4 = (2^6)^4 = 2^{24}, \quad \geq 32^4 = (2^5)^4 = 2^{20}$$

between 20 and 24 bits (exact 22,560 bits)

184.735 VU 2.0 Einführung in Künstliche Intelligenz

Neural Networks

Institute of Logic and Computation
Knowledge-Based Systems Group
Vienna University of Technology

SS 2023

slides by S. Russell, adapted by T. Eiter

AIMA Chapter 1, Sec. 2; Chapter 18, Sec. 7; Chapter 18, Sec. 11

Outline

1. Brains
2. Neural Networks
3. Perceptrons
4. Multi-Layer Perceptrons
5. Learning
6. Applications
7. Conclusion

Outline

1. Brains

2. Neural Networks

3. Perceptrons

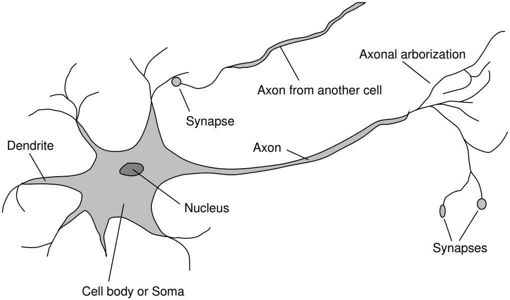
4. Multi-Layer Perceptrons

5. Learning

6. Applications

7. Conclusion

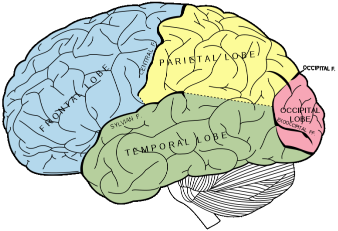
Neuron (Human Brain)



- Axon is 100 times longer than cell body, typically 1cm (up to 1 m)
- Up to some thousand synapses
- Signals are noisy “spike trains” of electrical potential
- Electro-chemical processes to pass potential through synapses (increase or decrease) using neuro-transmitters

Human Brain

source: Wikipedia



- 10^{11} neurons of > 20 types
- 10^{14} synapses, 1ms–10ms cycle time
- Functioning is still little understood
- Brain mapping: areas to body functions
  four lobes

occipital l.:	vision, visual reception, visual-spatial processing, movement, colors
parietal l.:	somatosensation, hearing, language, attention, spatial cognition
frontal l.:	control attention, abstract thinking, behavior, problem solving, physical reactions, personality
temporal l.:	controls auditory & visual memories, language, some hearing & speech

- learning
- take over functions (brain damage, e.g. stroke)

"Brains cause minds" (Searle, 1992)

Human Brain

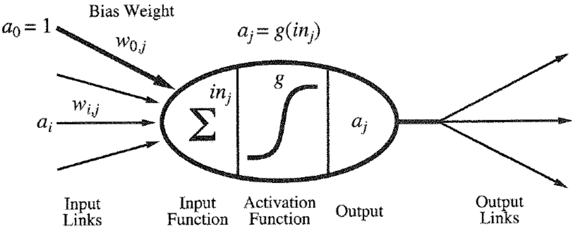
	Supercomputer	Personal Computer	Human Brain
Computational units	10^4 CPUs, 10^{12} transistors	4 CPUs, 10^9 transistors	10^{11} neurons
Storage units	10^{14} bits RAM	10^{11} bits RAM	10^{11} neurons
	10^{15} bits disk	10^{13} bits disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{15}	10^{10}	10^{17}
Memory updates/sec	10^{14}	10^{10}	10^{14}

- Supercomputer: IBM BLUE GENE (speedup $\sim$ 10 every 5 years)
- PC: typical as of 2008

Human Brain: Simulation

- Challenging (high complexity)
- *Human Brain Project* (Future & Emerging Techn. Flagship)
  - 10 years, approved 2013
  - $>$ 1 billion Euros
  - $>$ 200 research groups, in 19 European countries + USA, Israel, Argentina, Japan and China (lead: EPFL, Suisse)
- Precursor: Blue Brain Project
  - simulation of 10,000 neurons at cellular level
  - experimentally generated data
  - supercomputer with 8,000 IBM processors (“Big Blue”)

McCulloch–Pitts "unit" (1943)



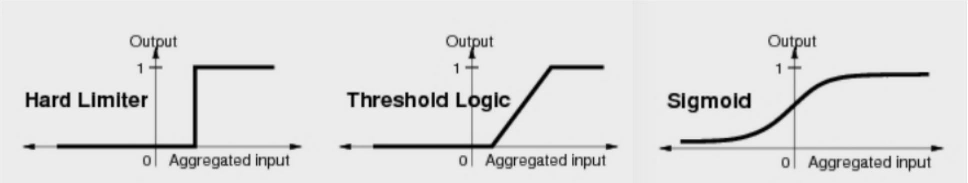
- Abstraction of a neuron u_j:
  - input is a linear sum of weighted inputs a_i:

$$in_j = \sum_i w_{i,j} a_i$$

a_0 ...fixed input ($\leadsto$ shift by bias $w_{0,j}$) (in earlier AIMA, $a_0 = -1$)

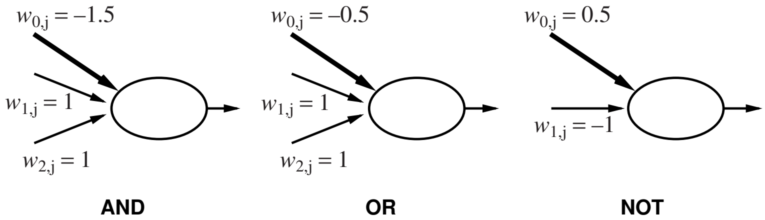
  - activation function $g(in_j)$ squashes the input to a value
  - output a_j formed from $g(in_j)$, typically $a_j \leftarrow g(in_j)$
- Gross oversimplification, but helps to develop understanding what networks of such units can do

Activation Functions



- Typical activation functions $g(x)$ $\quad (x = in_j)$:
  (a) step function (aka hard limiter)
  (b) linear function (aka threshold function)
  (c) sigmoid function $1/(1+e^{-x})$ (fully differentiable)
- Changing $w_{0,j}$ shifts the threshold location
- Evaluation at discrete time points or continuously

Implementing Logical Functions



McCulloch and Pitts:

- a complete basis of Boolean functions can be represented by single neurons

  activation function: $$g(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- arbitrary Boolean functions can be represented by a network of neurons

Outline

1. Brains

2. Neural Networks

3. Perceptrons

4. Multi-Layer Perceptrons

5. Learning

6. Applications

7. Conclusion

Network Structures

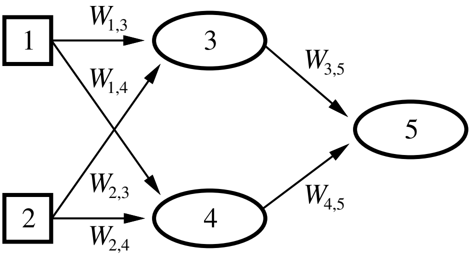
- Neural networks are composed of nodes (units) u_i, $i = 1, \dots, N$
- Nodes types: input, output, and processing (hidden) nodes
- Direct links from u_i to u_j (arcs $u_i \to u_j$), with weight $w_{i,j}$
- Abstractly, a neural network is a *function* $f(\mathbf{x})$ that maps each valuation $\mathbf{x} = (x_1, \dots, x_n)$ of the input units $u_1^{in}, \dots u_n^{in}$ to a valuation $\mathbf{y} = y_1, \dots, y_m$ of the output units $u_1^{out}, \dots, u_m^{out}$

  Single output: write y and u^{out} for y_1 resp. u_1^{out}
- If the graph and the activation functions are known, f is determined by the edge weights $\mathbf{w} = (w_{i,j})$ $(N \times N$ matrix)

$$f(\mathbf{x}) = h_{\mathbf{w}}(\mathbf{x})$$

- **Major task:** set up $\mathbf{w}$

Network Example



- $h_{\mathbf{W}}(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2)$
- Input units u_1, u_2, output unit u_5, hidden units u_3, u_4

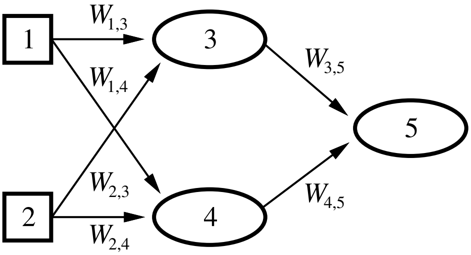
$$\begin{aligned} a_5 &= g(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4) \\ &= g(w_{3,5} \cdot g(w_{1,3} \cdot a_1 + w_{2,3} \cdot a_2) + w_{4,5} \cdot g(w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2)) \end{aligned}$$

Different network types allow for different capacities

Feed-Forward Networks

- Uni-directional
- Straight flow of information from input to output (directed acyclic graph, DAG)
- Layered networks: node in layer i is directly connected only to nodes in layer $i+1$ (and layer $i-1$).
- Types:
  - single-layer perceptrons
  - multi-layer perceptrons
- **No internal state:** implement reflex agents

Feed-Forward Example (again)



- $h_{\mathbf{W}}(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2)$
- input units u_1, u_2, output unit u_5

$$a_5 = g(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4)$$
$$= g(w_{3,5} \cdot g(w_{1,3} \cdot a_1 + w_{2,3} \cdot a_2) + w_{4,5} \cdot g(w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2))$$

Adjusting weights changes the function: do learning this way!

Recurrent Neural Networks

- Directed cycles with delays
  - internal state (like flip-flops), short term memory
- Can become unstable, oscillating, chaotic
- Several types of networks
  - Hopfield Networks
  - Boltzmann Machines
  - Long Short-Term Memory
  - ...

Recurrent Neural Networks, cont'd

- Hopfield networks
  - all nodes are input and output
  - bidirectional connections, symmetric weights ($w_{i,j} = w_{j,i}$),
  - sign activation function (= output a_i):

$$g(x) = \mathsf{sign}(x) = \begin{cases} 1 & x \geq 0, \\ -1 & x < 0 \end{cases}$$

  - enables **holographic associative memory**
    - training on examples
    - stimulus (partial data): get to "closest" training example
    - N units can store $\sim 1.4 * N$ training examples reliably

Recurrent Neural Networks, cont'd

- Boltzmann machines
  - hidden nodes, symmetric weights
  - use stochastic activation functions (output is 1 with some probability)
  - state transitions akin to simulated annealing
- Long short-term memory (LSTM)
  - building units to achieve a memory for long(er) time in RNNs
  - record value in a cell using a write gate, a keep gate, and a read gate
  - LSTM networks are well-suited to deal with time series having time lags of unknown size and duration between events

Outline

1. Brains

2. Neural Networks

3. Perceptrons

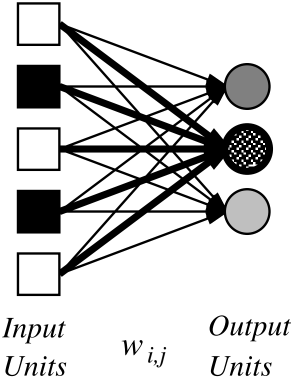
4. Multi-Layer Perceptrons

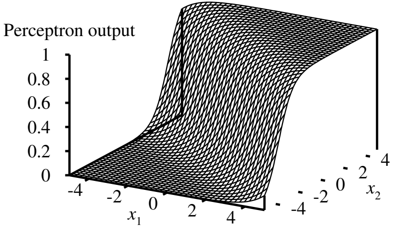
5. Learning

6. Applications

7. Conclusion

Single-Layer Perceptrons



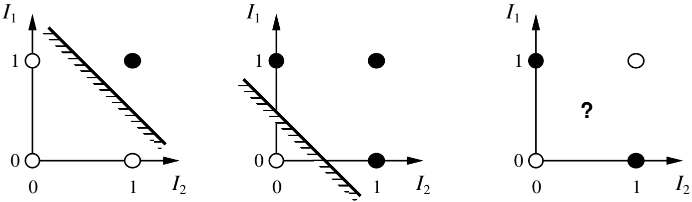


- Output units all operate separately—no shared weights
- Adjusting weights moves the location, orientation, and steepness of cliff
- Can view this as independent single perceptron, simplify $w_{i,j}$ to w_i

Expressiveness of Perceptrons

Perceptron with g = step function (Rosenblatt, 1957, 1960)

- Can represent AND, OR, NOT, majority, etc.
- Always represents a linear separator in the input space:



(a) I_1 **and** I_2 (b) I_1 **or** I_2 (c) I_1 **xor** I_2

$$\sum_i w_i x_i > 0 \quad (\text{i.e.,} \quad \mathbf{w} \cdot \mathbf{x} > 0)$$

Minsky and Papert (1969): can not express XOR

- pricked the neural network balloon
- little research for an extended period

Outline

1. Brains

2. Neural Networks

3. Perceptrons

4. Multi-Layer Perceptrons

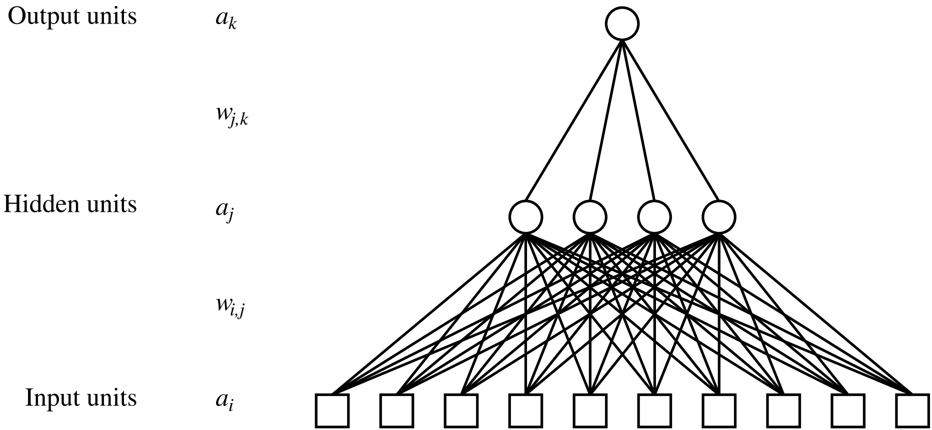
5. Learning

6. Applications

7. Conclusion

Multi-Layer Perceptrons

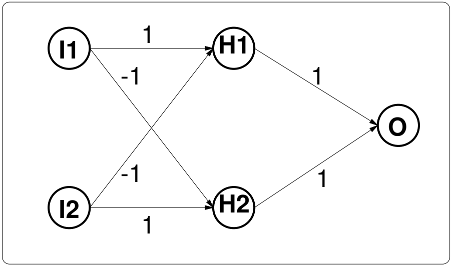
- Layers are usually fully connected
- Numbers of hidden units typically chosen by hand



Example: xor Function

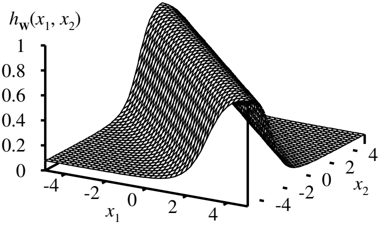
I1	0	0	1	1
I2	0	1	0	1
I1 xor I2	0	1	1	0

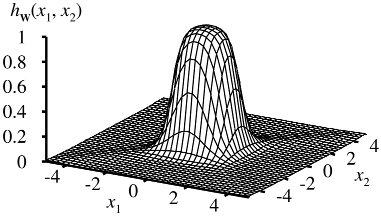
	in/out			
H1	0/0	-1/0	1/1	0/0
H2	0/0	1/1	-1/0	0/0
O	0/0	1/1	1/1	0/0



- Perceptron with hidden layer
- Activation: step function $g(x) = 1 \Leftrightarrow x \geq 1$

Expressiveness of MLPs





- **1 layer (no hidden layer):** all linearly separable functions
- **2 layers:** all continuous functions (arbitrary precision)
- **3 layers:** all functions
- Method:
  - make with opposite-facing threshold (sigmoid) functions a ridge
  - make with two ridges at right angles a bump
  - add bumps of various sizes and locations to fit any surface

Proof requires exponentially many hidden units

Outline

1. Brains

2. Neural Networks

3. Perceptrons

4. Multi-Layer Perceptrons

5. Learning

6. Applications

7. Conclusion

Perceptron Learning

- Recall: $h_{\mathbf{w}}(\mathbf{x}) = g(in)$, for single output unit u $(= u_j)$ where

$$in = \sum_i w_i x_i$$

- Let $Err = y - h_{\mathbf{w}}(\mathbf{x})$, where y is the target (true) output for $\mathbf{x}$
- Learn $f(\mathbf{x})$ by adjusting $\mathbf{w}$ to reduce error on training set
- Consider loss as squared error (*square loss*):

$$Loss(\mathbf{w}) = Err^2 = (y - h_{\mathbf{w}}(\mathbf{x}))^2$$

- Search for minimal loss by *gradient descent*:

$$\begin{aligned}
\frac{\partial Loss}{\partial w_i} &= 2Err \cdot \frac{\partial Err}{\partial w_i} \qquad \text{\% chain rule: } \frac{\partial f_1(f_2(x))}{\partial x} = f_1'(f_2(x)) \cdot \frac{\partial f_2(x)}{\partial x} \\
&= 2Err \cdot \frac{\partial}{\partial w_i}\left(y - g\left(\sum_i w_i x_i\right)\right) \\
&= -2Err \times g'(in) \times x_i
\end{aligned}$$

Perceptron Learning, cont'd

Simple update rule for weights:

Perceptron Learning Rule

$$w_i \leftarrow w_i + \alpha \times \underbrace{Err \times g'(in)}_{=\Delta} \times x_i$$

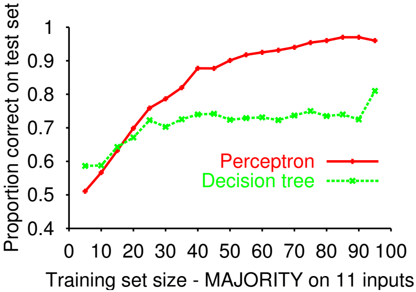
α ... ***learning rate*** (step size)

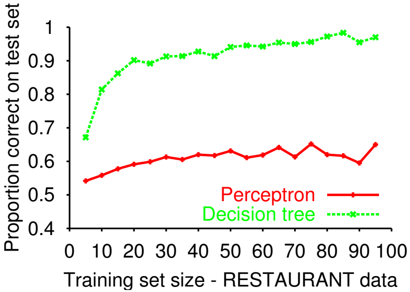
Multi-output learning ($\mathbf{y} = y_1, \ldots, y_m$):

- the weights $w_{i,j'}$ for $j' \neq j$ do not influence the output unit u_j
- thus, splits into m independent single output learning problems for u_j and y_j, $j = 1, \ldots, m$

Perceptron Learning, cont'd

Perceptron learning rule converges to a consistent function
for every linearly separable data set
$\Rightarrow$ *every linearly separable* $f(\mathbf{x})$ *is learnable from sufficient data*





- Perceptron learns majority function easily ($w_{0,i} = -n/2, w_{i,j} = 1$); DTL is hopeless (exponential size decision tree)
- DTL learns restaurant function easily; perceptron can't represent it (not linearly separable)

Multi-Layer Perceptron Learning

Issues:

- Separation of multi- to single-output learning fails:
  - $w_{i,j}$ can influence multiple outputs a_k, $a_{k'}$
- How to push errors back??

  helpful: additive loss $Loss = \sum_k Loss_k$, where $Loss_k$ is for u_k

  Square Loss: $Loss = Err^2 = |\mathbf{y} - \mathbf{h_w}(\mathbf{x})|^2 = \sum_k \underbrace{(y_k - a_k)^2}_{Loss_k = Err_k^2}$

  Minimal loss by gradient search on **w**:

  $$\frac{\partial}{\partial w} Loss = \frac{\partial}{\partial w} |\mathbf{y} - \mathbf{h_w}(\mathbf{x})|^2 = \frac{\partial}{\partial w} \sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w} (y_k - a_k)^2$$

  i.e., we just need to sum up the individual gradient losses at u_k
- What is the error at hidden unit u_j (true value is unknown)??

  Divide errors on contributing weights.

Multi-Layer Perceptron Learning, cont'd

Learning rules (learning rate α):

- Output layer (a_k): same rule as for single-layer perceptron

$$\boxed{w_{j,k} \leftarrow w_{j,k} + \alpha \cdot a_j \cdot \Delta_k, \quad \text{where } \Delta_k = Err_k \cdot g'(in_k)}$$

- Hidden layer (a_j): node a_j is "responsible" for a fraction of the errors Δ_k at the output layer
  - *back-propagate* the error from there:

$$\Delta_j = g'(in_j) \cdot \sum_k w_{j,k} \Delta_k \ .$$

  - update rule for weights in hidden layer:

$$\boxed{w_{i,j} \leftarrow w_{i,j} + \alpha \cdot a_i \cdot \Delta_j}$$

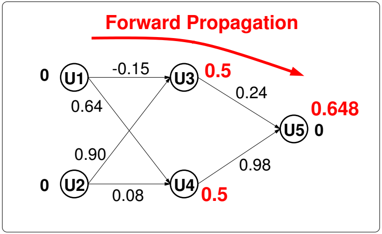
(Most neuroscientists deny that back-propagation occurs in the brain)

Backpropagation Algorithm

```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with L layers, weights w_{i,j}, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
     for each weight w_{i,j} in network do
        w_{i,j} ← a small random number
     for each example (x, y) in examples do
        /* Propagate the inputs forward to compute the outputs */
        for each node i in the input layer do
           a_i ← x_i
        for ℓ = 2 to L do
           for each node j in layer ℓ do
              in_j ← Σ_i w_{i,j} a_i
              a_j ← g(in_j)
        /* Propagate deltas backward from output layer to input layer */
        for each node j in the output layer do
           Δ[j] ← g'(in_j) × (y_j − a_j)
        for ℓ = L − 1 to 1 do
           for each node i in layer ℓ do
              Δ[i] ← g'(in_i) Σ_j w_{i,j} Δ[j]
        /* Update every weight in network using deltas */
        for each weight w_{i,j} in network do
           w_{i,j} ← w_{i,j} + α × a_i × Δ[j]
  until some stopping criterion is satisfied
  return network
```

Example



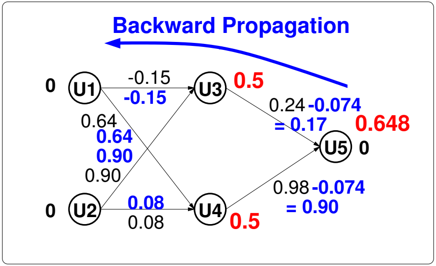
- Sigmoid function $g(x) = 1/(1+e^{-x})$
  - derivation $g'(x) = g(x) \cdot (1 - g(x))$
- Calculation (numbers rounded):
  - output unit a_k: $\Delta_k = a_k \cdot (1 - a_k) \cdot (y_k - a_k)$
  - hidden unit a_j: $\Delta_j = a_j \cdot (1 - a_j) \cdot \sum_k (\Delta_k \cdot w_{j,k})$

$$\begin{aligned}\Delta_5 &= Err_5 \cdot g'(in_5) = a_5 \cdot (1 - a_5) \cdot (y_5 - a_5) \\ &= 0.648 \cdot (1 - 0.648) \cdot (0 - 0.648) = -0.148\end{aligned}$$

$$\begin{aligned}\Delta_4 &= g'(in_4) \cdot \sum_k w_{4,k} \Delta_k = a_4 \cdot (1 - a_4) \cdot (w_{4,5} \cdot \Delta_5) \\ &= 0.5 \cdot (1 - 0.5) \cdot (0.98 \cdot -0.148) = -0.036\end{aligned}$$

other Δ_j values: $\Delta_3 = -0.009, \quad \Delta_2 = 0.011, \quad \Delta_1 = -0.022$

Example, cont'd



- Computation of the weight changes (learning rate $\alpha = 1$):

$$w_{4,5}: \qquad \alpha \cdot a_4 \cdot \Delta_5 \;=\; 1 \cdot 0.5 \cdot (-0.148) \;=\; -0.074$$

- Updated weight $w_{4,5} = 0.90$
- Other weight changes: computations similar

Backpropagation: Derivation

Recall: $\frac{\partial}{\partial w} Loss = \frac{\partial}{\partial w} |\mathbf{y} - \mathbf{h}_{\mathbf{w}}(\mathbf{x})|^2 = \sum_k \frac{\partial}{\partial w} \overbrace{(y_k - a_k)^2}^{Loss_k}$

consider gradient descent of $Loss_k$ wrt. $\mathbf{w}$ (write $\frac{\partial \alpha}{\partial \beta}$ also as $\partial\alpha/\partial\beta$)

1) Output weight $w_{j,k'}$: $\partial Loss_k/\partial w_{j,k'} \neq 0$ only if $k' = k$

$$\begin{aligned}
\frac{\partial Loss_k}{\partial w_{j,k}} &= \partial/\partial w_{j,k}(y_k - a_k)^2 \quad \text{\% chain rule: } \frac{\partial f_1(f_2(x))}{\partial x} = f_1'(f_2(x)) \cdot \frac{\partial f_2(x)}{\partial x} \\
&= -2(y_k - a_k)\partial a_k/\partial w_{j,k} \\
&= -2(y_k - a_k)\partial g(in_k)/\partial w_{j,k} \\
&= -2(y_k - a_k)g'(in_k)\partial in_k/\partial w_{j,k} \\
&= -2(y_k - a_k)g'(in_k)\partial\Big(\sum_{j'} w_{j',k}a_{j'}\Big)/\partial w_{j,k} \quad \text{\% only } j' = j \text{ matters} \\
&= -2\underbrace{(y_k - a_k)g'(in_k)}_{=\Delta_k}a_j \\
&= -2a_j\Delta_k
\end{aligned}$$

Back-Propagation: Derivation, cont'd

2) Interior weight: unit a_i connecting to hidden unit a_j

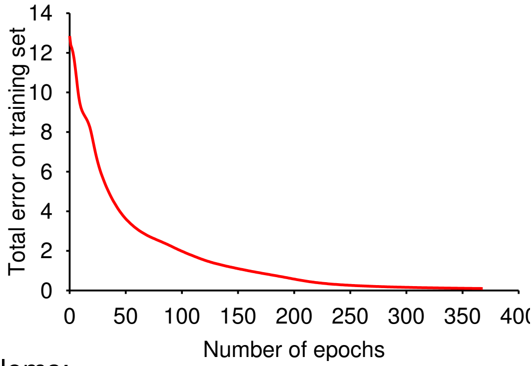
$$
\begin{aligned}
\frac{\partial Loss_k}{\partial w_{i,j}} &= \partial/\partial w_{i,j}(y_k - a_k)^2 \quad \text{\% chain rule: } \frac{\partial f_1(f_2(x))}{\partial x} = f_1'(f_2(x)) \cdot \frac{\partial f_2(x)}{\partial x} \\
&= -2(y_k - a_k)\partial a_k/\partial w_{i,j} \\
&= -2(y_k - a_k)\partial g(in_k)/\partial w_{i,j} \\
&= -2(y_k - a_k)g'(in_k)\partial in_k/\partial w_{i,j} \quad \% \ (y_k - a_k)g'(in_k) = \Delta_k \\
&= -2\Delta_k \partial\left(\sum_{j'} w_{j',k} a_{j'}\right)/\partial w_{i,j} \quad \text{\% only } j' = j \text{ matters} \\
&= -2\Delta_k w_{j,k} \partial a_j/\partial w_{i,j} \\
&= -2\Delta_k w_{j,k} \partial g(in_j)/\partial w_{i,j} \\
&= -2\Delta_k w_{j,k} g'(in_j)\partial in_j/\partial w_{i,j} \\
&= -2\Delta_k w_{j,k} g'(in_j)\partial\left(\sum_{i'} w_{i',j} a_{i'}\right)/\partial w_{i,j} \quad \text{\% only } j' = j \text{ matters} \\
&= -2\Delta_k w_{j,k} g'(in_j) a_i
\end{aligned}
$$

Thus,

$$
\begin{aligned}
\frac{\partial Loss}{\partial w_{i,j}} = \sum_k \frac{\partial Loss_k}{\partial w_{i,j}} &= \sum_k -2\Delta_k w_{j,k} g'(in_j) a_i \\
&= -2a_i \underbrace{g'(in_j)\sum_k \Delta_k w_{j,k}}_{=\Delta_j} = -2a_i\Delta_j
\end{aligned}
$$

Back-Propagation Learning, cont'd

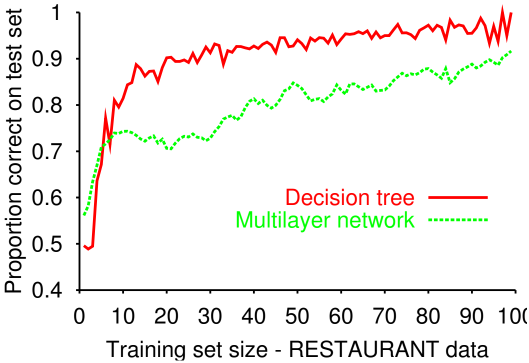
- At each epoch, sum gradient updates for all examples and apply
- Restaurant example: use a MLP with 4 hidden units
  Training curve for 100 examples (finds exact fit)



- Typical problems:
  slow convergence, local minima

Back-Propagation Learning, cont'd

Learning curve for restaurant example



Problem: determine suitable network structure

Outline

1. Brains

2. Neural Networks

3. Perceptrons

4. Multi-Layer Perceptrons

5. Learning

6. Applications

7. Conclusion

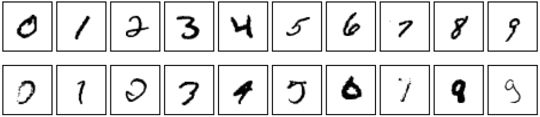
Applications of Neural Networks

Thousands of applications, e.g.,

- Prediction of air pollution (e.g., ozone concentration) (better than physical/chemical, statistical models)
- Prediction of electric power prizes
- Prediction of stock developments
- Desulfurization in steel making
- Pattern recognition (e.g., handwriting)
- Face recognition
- Speech recognition
- ...

Handwritten Digit Recognition

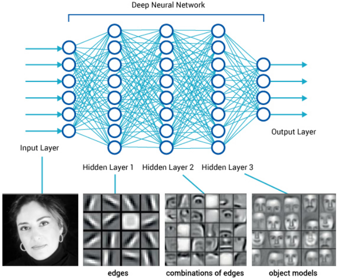
US NIST database: 60,000 digit samples (20$\times$20 pxs, 8bit gray)



Performance of different learning strategies

- 3-nearest-neighbor classification: 2.4% error
- 2 layer MLP (400–300–10 units): 1.6% error
- LeNet (1989;1995+): 0.9% error, using specialized neural net family
  - input layer of 32$\times$32 units, on which 20$\times$20 pixel were centered (pixels + local neighborhoods)
  - apply slight transformations to the input (rotate, shift, scale)
  - 3 hidden layers, 768–192–30 units
    groups of $n \times n$ units (*feature detectors*)
- current/recent best: $\approx$ 0.23% error, multi-col. convoluted deep NN

Deep Learning



- Hierarchical structure learning: NNs with many layers (AlphaGo: 12/14 hidden layers for policy/value network)
- Feature extraction and learning (un)supervised
- Increasing in the last decade
- **Crux: lots** of data
- Push for hardware (GPUs, designated chips)
- *"Data-Driven"* vs. *"Model-Driven"* computing

Advantages and Disadvantages of NNs

Pros

- Less need for determining relevant input factors
- Inherent parallelism
- Easier to develop than statistical methods
- Capability to learn and improve
- Quite good for complex pattern recognition tasks
- Usable for “unstructured” and “difficult” input (e.g., images)
- Fault tolerance

Advantages and Disadvantages of NNs, cont'd

Cons

- Choice of parameters (layers, units) requires skill
- Requires sufficient training material
- Resulting hypotheses cannot be understood easily
- Knowledge implicit (*subsymbolic representation*)
- Verification, behavior prediction difficult (if not impossible)

Challenges

- Interpretability
- Explainability
- Trustworthiness
- Robustness (adversarial examples)

Combine symbolic and subsymbolic approaches

Outline

1. Brains

2. Neural Networks

3. Perceptrons

4. Multi-Layer Perceptrons

5. Learning

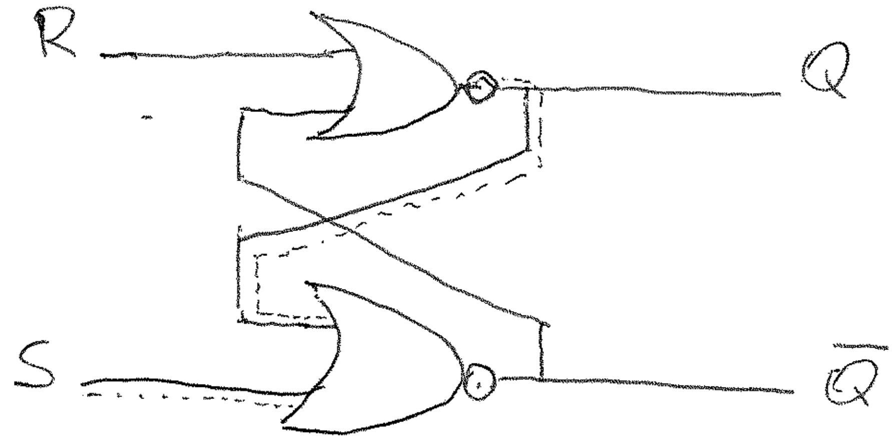
6. Applications

7. Conclusion

Summary

- Most brains have lots of neurons
  each neuron $\approx$ linear–threshold / sigmoid unit
- Perceptrons (one-layer networks) insufficiently expressive
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation
- Many applications: speech, driving, handwriting, fraud detection, etc.
- Engineering, cognitive modeling, and neural system modeling subfields have largely diverged
- Challenges:
  - Interpretability, Explainability, Trustworthiness, Robustness
  - Combine symbolic and subsymbolic approaches

Recurrent flows SR-flip-flop



NOR-Gates

S... "Set" R... "Reset

1) $S = 1$ and $R = \emptyset$ $\Rightarrow$ $Q = 1$ and $\overline{Q} = \emptyset$

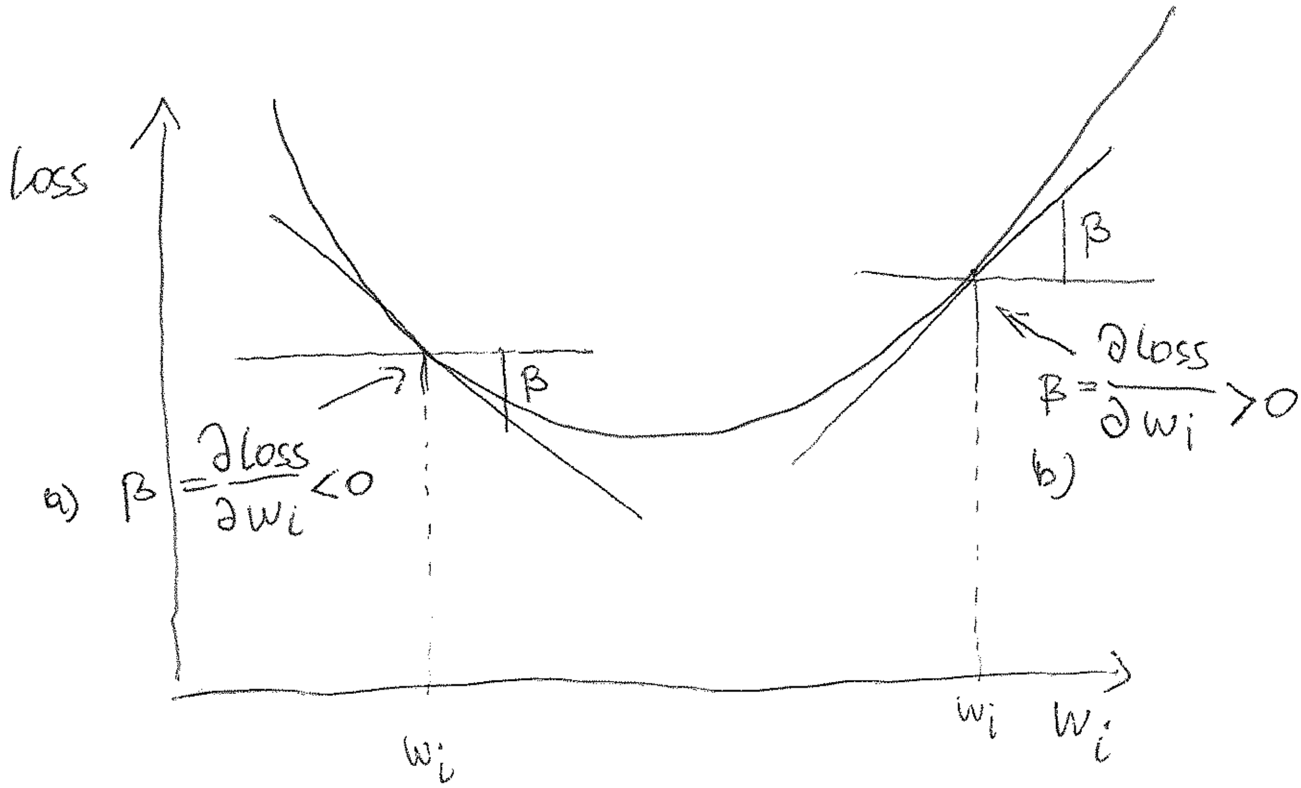
2) Set $S = \emptyset$ (keep $R = \emptyset$) $\Rightarrow$ Q remains 1, $\overline{Q}$ remains $\emptyset$

Storage

Perceptron Learning

$$h_w(\vec{x}) = g(in), \quad in = \sum_i w_i x_i$$

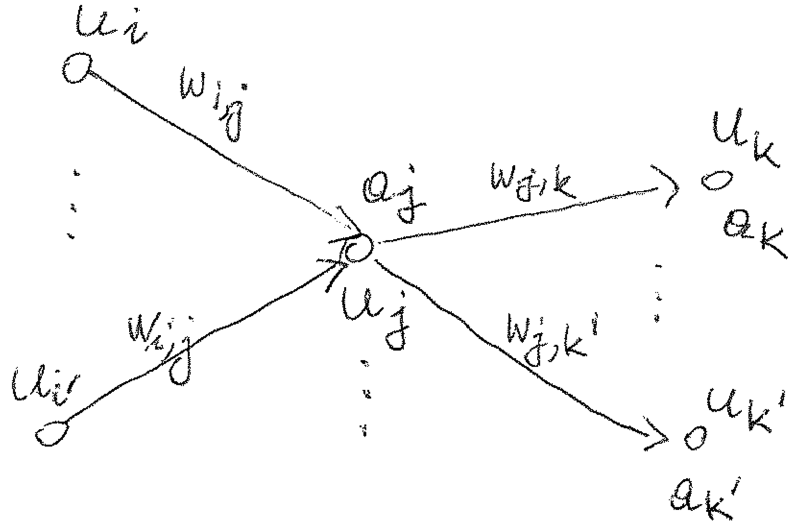
$$Loss(w) = Err^2 = (y - h_w(x))^2$$



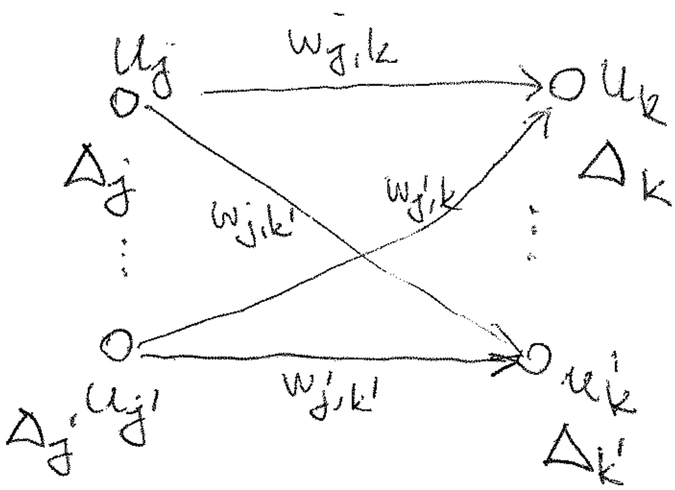
a) increase w_i by β to affect $g(in)$

b) decrease w_i by β to affect $g(in)$

Multi-Layer Perceptron Learning



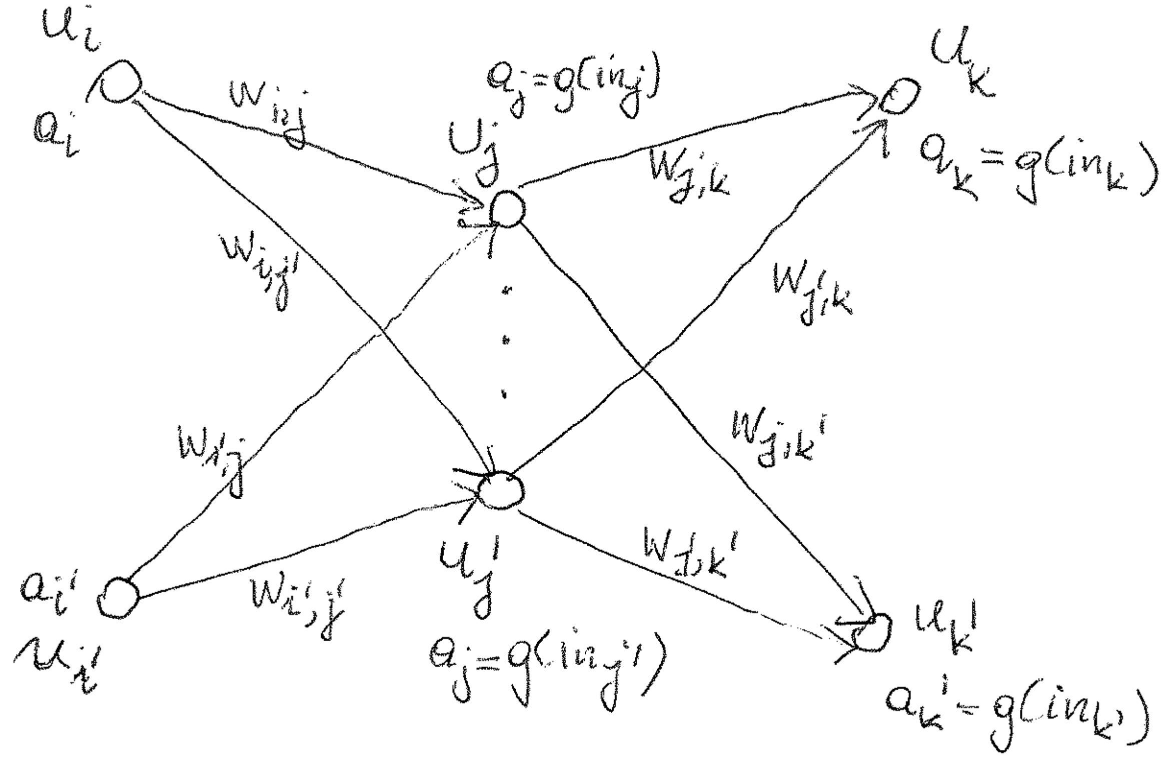
$a_j = g(in_j)$ $a_k = g(in_k)$



$\Delta_j = g'(in_j) \cdot \sum_k w_{j,k} \cdot \Delta_k$ $\Delta_k = Err_k \cdot g'(in_k)$

Push back weighted errors

Multi-Layer Perceptrons



$$\frac{\partial a_k}{\partial w_{i,j}} = \frac{\partial in_k}{\partial w_{i,j}} = g'(in_k) \cdot \frac{\partial in_k}{\partial w_{i,j}}$$

$= \sum_{j'} w_{j',k} \cdot a_{j'}$

only $w_{j',k}$, $j' = j$ matters!

$$\frac{\partial a_j}{\partial w_{i,j}} = \frac{\partial g(in_j)}{\partial w_{i,j}} = g'(in_j) \cdot \frac{\partial in_j}{\partial w_{i,j}}$$

$= \sum_{i'} w_{i',j} \cdot a_{i'}$

only $w_{i',j}$, $i' = i$ matters

VU Einführung in die Künstliche Intelligenz

SS 2023

Hans Tompits

Institut für Logic and Computation
Forschungsbereich Wissensbasierte Systeme

www.kr.tuwien.ac.at

Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSPs)

- Standard search problem:
  - From the point of view of a search algorithm, a state is a "black box" with no discernible internal structure.
  - It is represented by a suitable data structure that can be accessed only by the *problem specific* routines:
    - the successor function,
    - the heuristic function,
    - and the goal test.
- *Constraint satisfaction problem* (CSP):
  - The states and the goal test conform to a standard, structured, and simple representation.
  - Search algorithms can be defined that take advantage of the structure of states and use *general-purpose* rather than *problem-specific* heuristics.

Constraint Satisfaction Problems (ctd.)

➤ In a constraint satisfaction problem
- a *state* is defined by *variables* with *values* from an associated *domain*
- the goal test is a set of *constraints* specifying allowable combinations of values for subsets of variables

➥ Example of a simple *formal representation language*
- allows useful general-purpose algorithms with more power than standard search algorithms.

CSP: Formal Definition

A *constraint satisfaction problem* (CSP) consists of the following components:

- a finite set $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ of variables;
- each variable $V_i \in \mathcal{V}$ has an associated non-empty domain D_i of possible values;
- a finite set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of constraints.
  - A constraint $C \in \mathcal{C}$ between variables $V_{i_1}, \ldots, V_{i_j}$ is a subset of the Cartesian product

$$D_{i_1} \times \cdots \times D_{i_j} = \{(d_1, \ldots, d_j) \mid d_l \in D_{i_l}, 1 \leq l \leq j\}.$$

CSP: Formal Definition (ctd.)

- Each constraint limits the values that variables can take, e.g., $V_1 \neq V_2$.
- There are constraints of different arities:
  - *n-ary constraints* restrict the possible assignment of n variables, i.e., n-ary constraints are *n-ary relations*.
  - In particular:
    - Unary constraints restrict the domain D_i of one variable V_i. E.g., $C(V_i) = \{1, 3, 5, 7, 8\}$.
    - Binary constraints restrict the domains $D_i \times D_j$ of a pair of variables V_i, V_j.
      E.g., $C(V_i, V_j) = \{(1, 2), (3, 5), (7, 3), (8, 2)\}$.
    - Ternary constraints,...

CSP: Further notions

- A state of a CSP is defined by an assignment of values to some or all of the variables.
- An assignment that does not violate any constraints is *consistent* or *legal*.
- An assignment is *complete* iff it mentions every variable.
- A *solution* to a CSP is a complete consistent assignment, i.e., a function which assigns
  1. each variable a value of its associated domain and
  2. such that all constraints are satisfied.
- Some CSPs also require a solution that maximises an objective function
  - these are called *constrained optimisation problems*.

Example: Map-colouring

Consider the task of colouring a map of Australia with the colours red, green, and blue such that no neighbouring region have the same colour.

Example: Map-colouring (ctd.)

We can formulate this problem as the following CSP:

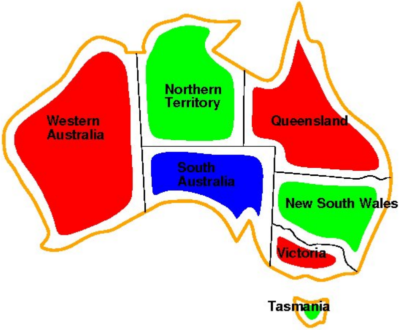
- *Variables:* $\mathcal{V} = \{WA, NT, Q, NSW, V, SA, T\}$
- *Domains:* $D_i = \{red, green, blue\}$, $i \in \mathcal{V}$
- *Constraints:* adjacent regions must have different colors
    - e.g., the allowable combinations of WA and NT are

$$C(WA, NT) = \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\},$$

    - or simply written as $WA \neq NT$ (if the language allows this).

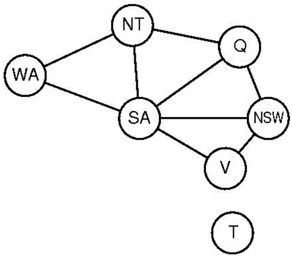
Example: Map-colouring (ctd.)

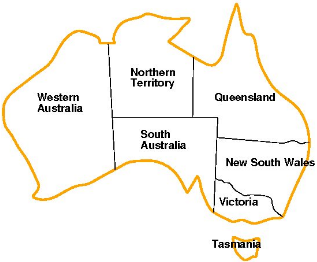
There are many possible solutions, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red,$
$SA = blue, T = green\}$



Constraint graph

- For a *binary CSP* (in which all constraints are binary), it is helpful to visualise the problem as a constraint graph:
  - the nodes are the variables,
  - the edges correspond to the constraints, i.e., there is an edge between two variables if there is a constraint involving them.
- E.g., our map-colouring problem has the following constraint graph:





  - General-purpose CSP algorithms use the *graph structure* to speed up the search.
  - E.g., Tasmania is an independent subproblem!

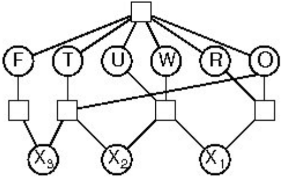
Constraint graph (ctd.)

- Higher-order constraints can be represented by a *constraint hypergraph*.
  - Reminder: a hypergraph is a pair (X, E), where X is a set of nodes and E is a set of non-empty subsets of X, the hyperedges.
- *Cryptarithmetic puzzles* are examples of involving higher-order constraints.
  - Usually, one assumes that each letter in a cryptarithmetic puzzle represents a different digit.

Constraint graph (ctd.)

Example:

```
   T W O
 + T W O
 -------
 F O U R
```



➤ This is formulated as the following CSP:

- Variables: F, T, U, W, R, O, X_1, X_2, X_3
- Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
  - $\mathit{Alldiff}(F, T, U, W, R, O)$;
  - addition constraints:
    $O + O = R + 10 \cdot X_1$,
    $X_1 + W + W = U + 10 \cdot X_2$,
    $X_2 + T + T = O + 10 \cdot X_3$,
    $X_3 = F$.

➤ A solution for this CSP is, e.g., $938 + 938 = 1876$.

Varieties of CSPs

- The simplest kind of CSPs involves variables that are *discrete* and have *finite domains*.
  - E.g., map-colouring problems are of this kind.
- If the maximum domain size of any variable in a CSP is d, and there are n variables, then the number of possible complete assignments is $O(d^n)$
  - ➥ exponential in the number of variables!

Varieties of CSPs (ctd.)

- Finite domain CSPs whose variables can be either true or false are called *Boolean CSPs*.
- E.g., 3SAT can be expressed as a Boolean CSP
  - a clause like $X_1 \vee \neg X_2 \vee X_3$ corresponds to the constraint

$$C(X_1, X_2, X_3) = \\ (\{true, false\} \times \{true, false\} \times \{true, false\}) \setminus \{(false, true, false)\}.$$

- Since 3SAT is an NP-complete problem we cannot expect to solve finite-domain CSPs in less than exponential time (unless P = NP).
- However, in most *practical* applications, CSP algorithms can solve problems orders of magnitude larger than those solvable via general search algorithms.

Varieties of CSPs (ctd.)

- Discrete variables can also have *infinite domains*, e.g., the set of integers or the set of strings.
    - E.g., for construction job scheduling, variables are the start dates and the possible values are integer numbers of days from the current date.

- Note:
    - With infinite domains it is no longer possible to describe constraints by enumerating all allowed combinations of values.
    - Rather, a constraint language must be used.
        - E.g., if Job_1, which takes 5 days, must precede Job_3, then we need a language of algebraic inequalities like $StartJob_1 + 5 \leq StartJob_3$.

Varieties of CSPs (ctd.)

- It is also no longer possible to solve constraints with infinite domains by enumerating all possible assignments
  - ➥ there are infinitely many of them!

- Special solution algorithms exist for *linear constraints* on integer values
  - linear constraint = variables appear only in *linear* form
  - e.g., $StartJob_1 + 5 \leq StartJob_3$ is linear.

- Non-linear constraints are *undecidable*—no algorithm exists for solving such constraints!

Varieties of CSPs (ctd.)

- Finally, there are CSPs with *continuous domains*
  - very common in real-world applications and widely studied in operations research
  - e.g., scheduling the start/end times for the Hubble Space Telescope
    - require a very precise timing of observations,
    - taking a variety of real-valued astronomical, precedence, and power constraints into account.

- Linear constraints can be solved with *linear programming* methods in polynomial time.

Some real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floor planning

☞ Notice that many real-world problems involve real-valued variables.

CSPs as standard search problems

- It is straightforward to give an *incremental formulation* of a CSP as a standard search problem.
  - States are defined by the values assigned so far.
  - Initial state: the empty assignment, $\emptyset$.
  - Successor function: assign a value to an unassigned variable providing it does not conflict with the current assignment.
  - Goal test: the current assignment is complete.
- This is the same for all CSPs!
  - ➥ Any standard search algorithm can be used to solve CSPs.

CSPs as standard search problems (ctd.)

Caveat: Suppose we use breadth-first search.

- If there are n variables and d values, the branching factor at the top level is nd.
- At the next level, the branching factor is $(n-1)d$, and so on for n levels.
- ➥ We generate a tree with $n!d^n$ leaves although there are only d^n possible complete assignments!

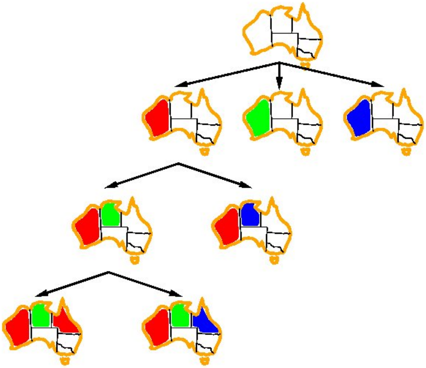
Backtracking search

- The naive formulation ignored one crucial property of CSPs:
  - Variable assignments are *commutative*, i.e., the order of an assignment of variables does not matter and one reaches the same partial assignment regardless of order.
  - Therefore, CSP search algorithms need only to consider a single variable at each node of the search tree!
    - E.g., in the map-colouring problem, initially we may have a choice between $SA = red$, $SA = green$, and $SA = blue$,
    - but we would not choose between $SA = red$ and $WA = blue$.
  - ➥ With this restriction, we generate only d^n leaves as expected.

- Depth-first search for CSPs with single-variable assignments is called *backtracking* search.
  - Backtracking search is the basic uninformed algorithm for CSPs.

Backtracking search (ctd.)

Below gives part of the search tree for the Australia problem, where the variables are assigned in the order WA, NT, Q, ...

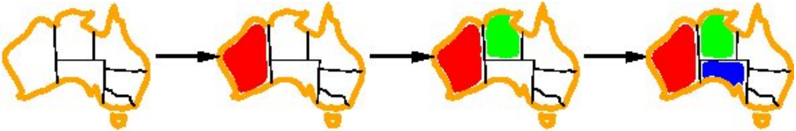


Backtracking search (ctd.)

- Since plain backtracking search is an uninformed algorithm, we do not expect it to be very effective for large problems.
- Different *general-purpose methods* help improving the performance, addressing the following issues:
  - Which variable should be assigned next, and in what order should its values be tried?
  - What are the implications of the current variable assignments for the other unassigned variables?
  - When a path fails, can the search avoid repeating this failure in subsequent paths?

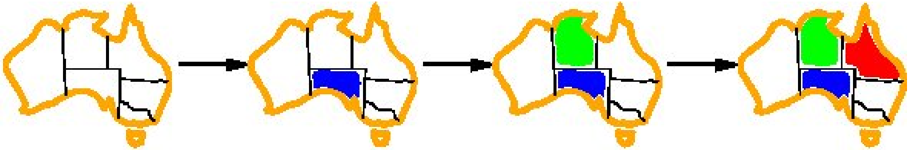
Minimum-remaining-values heuristic

- The minimum-remaining-values (MRV) heuristic:
  - choose the variable with the fewest legal values.
- If there is a variable X with 0 legal values remaining, the MRV heuristic will select X and failure will be detected immediately
  - avoiding pointless searches through further unassigned variables.
- E.g., in the Australia example, after the assignments for $WA = red$ and $NT = green$, there is only one possible value for SA.
  - It makes sense to assign $SA = blue$ next rather than assigning Q.
  - Actually, after SA is assigned, the choices for Q, NSW, and V are all forced.



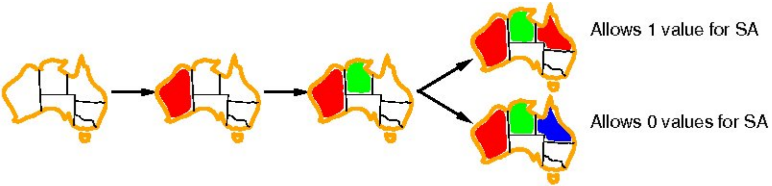
Degree heuristic

- The MRV heuristic does not help at all in choosing the *first* region to colour.
- In this case, the degree heuristic comes in:
  - it selects the variable that is involved in the *largest number of constraints* on other unassigned variables.
- In the Australia example, *SA* is the variable with highest degree, 5.
  - The others have degree 0, 2, or 3.
  - Actually, once *SA* is chosen, we can can assign the mainland regions clockwise or counterclockwise with a colour different from *SA* and the previous region.



Least-constraining-value heuristic

- Once a variable has been selected, to decide on the order in which to examine its values, the least-constraining-value heuristic can be effective:
  - it prefers a value that rules out the *fewest* choices for the neighbouring variables in the constraint graph.
- In the Australia example, suppose we have the partial assignment $WA = red$ and $NT = green$, and our next choice is for Q.
  - Blue would be a bad choice, because it eliminates the last legal value for Q's neighbour SA.
  - ➥ The least-constraining-value heuristic thus prefers red to blue.

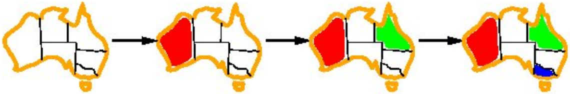


Forward checking

- The methods discussed so far consider the constraints on a variable *only at the time that the variable is chosen.*
- By looking at some of the constraints earlier in the search, or even before the search, the search space can be drastically reduced.
- One such method is forward checking:
  - whenever a variable X is assigned, it looks at each unassigned variable Y that is connected to X by a constraint
  - and deletes from the domain of Y any value that is inconsistent with the value chosen for X.

Forward checking (ctd.)

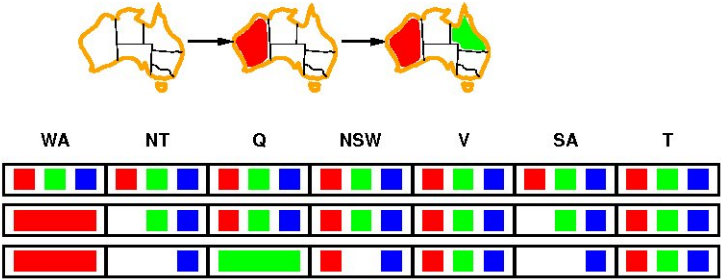
- Consider colouring Australia using forward checking:



- Note:
  - After assigning $WA = red$ and $Q = green$, the domains of NT and SA are reduced to a single value.
    - ➥ The MRV heuristic would select SA and NT next.
  - After assigning $V = blue$, the domain of SA is empty, so we get failure and the algorithm backtracks.

Forward checking (ctd.)

- Forward checking does not provide early detection for all failures:

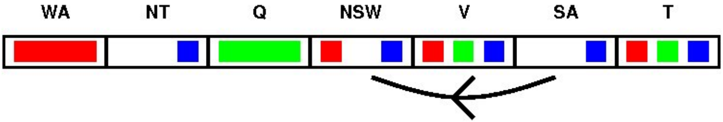


- *NT* and *SA* cannot both be blue!

☞ Constraint propagation is the general term for propagating the implications of a constraint on one variable onto other variables.

Arc consistency

- The simplest form of constraint propagation is arc consistency:
  - "arc" refers to a *directed* arc in the constraint graph;
  - $X \rightarrow Y$ is *consistent* iff for *every* value x of X there is *some* allowed value y of Y.
- For $SA = blue$ in the Australia colouring, there is a consistent assignment for NSW, namely red $\Longrightarrow$ the arc from SA to NSW is consistent
  - the reverse arc is *not* consistent, but can be made so by deleting blue from the domain of NSW.



Further techniques

- Intelligent backtracking:
  - do not backtrack to the preceding variable if a failure occurs, but go back to one in the set of variables that *caused the failure*
    - this set is the conflict set
    - e.g., backjumping goes to the most recent variable in this conflict set.
- Local search algorithms are very effective for solving CSPs
  - the *million*-queens problem can be solved in an average of 50 steps.
- The structure of the constraint graph can be taken into account.
  - E.g., colouring Tasmania is an independent subproblem of colouring Australia.
  - Tree-structured problems can be solved in linear time.

VU Einführung in die Künstliche Intelligenz

SS 2023

Hans Tompits

Institut für Logic and Computation
Forschungsbereich Wissensbasierte Systeme

www.kr.tuwien.ac.at

Planning

(Adapted from Chapters 10.1 and 10.2 in
Russell & Norvig: Artificial Intelligence—A Modern
Approach)

Planning—General Considerations

Planning = coming up with a sequence of actions that will achieve some goal.

- Reasoning about the results of actions is central to the operation of an intelligent agent.
- One way to represent actions is to use first-order logic expressing things like

  $\forall\, t,$ such-and-such is the result at $t + 1$ of doing action at t.

- In what follows, we describe a planning language which avoids explicit times and focusses instead on *states*.
  - A state results from another state by applying some action.

Planning—General Considerations (ctd.)

- In dealing with reasoning about actions, three problems have in this context been identified in the literature:
  - the *frame problem*,
  - the *ramification problem*, and
  - the *qualification problem*.

- The frame problem deals with the question how to represent things which stay unchanged after performing some action.
  - Indeed, most things stay the same when applying a single action
    - ➥ a large number of so-called *frame axioms* would be needed in general to represent what does not change by performing an action.

Planning—General Considerations (ctd.)

- The ramification problem deals with the representation of *implicit effects*.
  - E.g., if a car moves from one position to another, so does
    - any person in the car, the engine of the car, any dust particle in the car, any bacteria in the driver, etc.
- The qualification problem deals with the required preconditions (the "qualifications") ensuring that an action succeeds.
  - E.g., if a robot needs to move a block A on top of another block B, the following requirements may apply:
    - B should have a clear top, A must not be too heavy, the robot's arm must not be broken, etc.
  - The qualification problem thus deals with a *correct conceptualisation of things*
    - ➥ there is no general solution for it.

Planning—General Considerations (ctd.)

☞ We are only concerned with *classical planning* environments, which are

- *fully observable*,
- *deterministic*,
- *finite*,
- *static* (change happens only when the agent acts), and
- *discrete* (in time, actions, objects, and effects).

The Language of Planning Problems

- Key issues of a good planning language:
  - expressive enough to describe a wide variety of problems;
  - restrictive enough to allow efficient algorithms.

- We first discuss STRIPS (Fikes and Nilsson, 1971), a basic representation language of classical planners.
  - "STRIPS" stands for "Stanford Research Institute Problem Solver".
  - STRIPS was designed as the planning component of the software for the Shakey robot project at SRI.

The Language of Planning Problems (ctd.)

Shakey, the Robot (1966-72)

STRIPS—States and Goals

The syntax of STRIPS consists of the following items:

- Representation of states: Planners decompose the world into logical conditions and represent a state as a conjunction of positive literals.
  - Literals are atomic formulas or negations thereof (a positive literal is just an atom)
    - literals can be propositional or first-order, but first-order literals must be *ground* (i.e., variable-free) and *function-free*.
  - For instance,
    - $Rich \wedge InJail$ may represent the state of some person,
    - while $At(x, y)$ or $At(president(USA), White\_House)$ are not allowed.
  - Furthermore, the *closed-world assumption* is used, meaning that any condition not mentioned in a state is assumed false.

STRIPS—States and Goals (ctd.)

- Representation of goals: A goal is a partially specified state, represented as a conjunction of positive ground literals, such as $Rich \wedge Famous$ or $At(P_2, LakeTahoe)$.
  - A propositional state s *satisfies* a goal g if s contains all the atoms in g (and possibly others).
  - E.g., the state $Rich \wedge Famous \wedge InJail$ satisfies the goal $Rich \wedge Famous$.

STRIPS—Actions

- Representation of actions: An action is specified in terms of the *preconditions* that must hold before it can be executed and the *effects* that ensue when it is executed.
  - E.g., an action for flying a plane from one location to another is:

    $Action(Fly(p, from, to),$

    $\quad$ $\textsc{Precond}$: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

    $\quad$ $\textsc{Effect}$: $\neg At(p, from) \wedge At(p, to))$

  - More precisely:
    - this is actually an example of an *action schema*,
    - representing *a number of different actions* that can be derived by instantiating the variables p, $from$, and to to different constants.

STRIPS—Action Schemata

In general, an action schema consists of three parts:

- The *action name* and *parameter list*—e.g., $Fly(p, from, to)$.
- The *precondition*: a conjunction of function-free positive literals stating what must be true in a state before the action can be executed.
  - ☞ Any variables in the precondition must also appear in the action's parameter list.
- The *effect*: a conjunction of function-free literals describing how the state changes when the action is executed.
  - A positive literal P in the effect is *true* in the state resulting from the action; a negative literal $\neg P$ results in P being *false*.
  - Variables in the effect must also appear in the action's parameter list.

☞ Some planning systems divide the effect into the *add list* for positive literals and the *delete list* for negative literals.

STRIPS—Semantics

An action is *applicable* in any state that satisfies the preconditions; otherwise, the action has no effect.

- For a first-order action schema, establishing applicability involves a substitution for the variables in the precondition.
- E.g., suppose the current state is described by

$$At(P_1, JFK) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \\ \wedge Airport(JFK) \wedge Airport(SFO).$$

  This state satisfies the precondition

$$At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$

  of action schema $Fly(p, from, to)$ with substitution $\{p/P_1, from/JFK, to/SFO\}$.

  ➥ The concrete action $Fly(P_1, JFK, SFO)$ is applicable.

STRIPS—Semantics (ctd.)

- Starting in a state s, the *result* of executing an applicable action a is a state s' that results from s by
  - adding any positive literal P in the effect of a and
  - removing any P where $\neg P$ appears in the effect of a.
- Thus, for our flight example, after executing $Fly(P_1, JFK, SFO)$, the current state

$$At(P_1, JFK) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \\ \wedge Airport(JFK) \wedge Airport(SFO).$$

becomes

$$At(P_1, SFO) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \\ \wedge Airport(JFK) \wedge Airport(SFO).$$

STRIPS—Semantics (ctd.)

Remarks:

- If a positive effect is already in s it is not added twice, and if a negative effect is not in s, then that part of the effect is ignored.
- The definition of the semantics of STRIPS embodies the so-called *STRIPS assumption*:
  - every literal not mentioned in the effect remains unchanged.

$\Longrightarrow$ This is the way STRIPS deals with the frame problem.

STRIPS—Semantics (ctd.)

- Finally, a *solution* for a planning problem is an action sequence that, when executed in the initial state, results in a state that satisfies the goal.

- ☞ Later on, we will allow solutions to be partially ordered sets of actions, provided that every action sequence that respects the partial order is a solution.

Language Variants

- For some real-world domains, STRIPS is not sufficient
  - ➥ many language variants have been developed as a consequence.
- An important such variant is *ADL* (Pednault, 1986), the *Action Description Language*.
- In ADL, the *Fly* action can be written as follows:
  $Action(Fly(p : Plane, from : Airport, to : Airport),$
  $\quad \textsc{Precond}: At(p, from) \land from \neq to$
  $\quad \textsc{Effect}: \neg At(p, from) \land At(p, to))$
- Note:
  - ADL allows *typing*—e.g., the notation $p : Plane$ is an abbreviation for $Plane(p)$.
  - The precondition $from \neq to$ expresses that a flight cannot be made from an airport to itself.

STRIPS vs. ADL

STRIPS	**ADL**
Only positive literals in states: $Rich \wedge InJail$	Positive and negative literals in states: $\neg Poor \wedge \neg Free$
Closed-World Assumption: Unmentioned literals are false	Open-World Assumption Unmentioned literals are unknown
Effect $P \wedge \neg Q$ means add P and delete Q	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q
Only ground atoms in goals: $Rich \wedge InJail$	Quantified variables in goals: $\exists x\ (At(P_1, x) \wedge At(P_2, x))$ is the goal of having P_1 and P_2 in the same place
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions	Conditional effects are allowed: $when\ P : E$ means E is an effect only if P is satisfied
No support for equality	Equality is built in
No support for types	Variables can have types, as in $(p : Plane)$

Remarks

- The various planning formalisms used in AI have been systematised within a standard syntax called the *Planning Domain Definition Language*, or *PDDL* (Ghallab, Howe, Knoblock, McDermott, 1998).
  - ☞ PDDL includes sublanguages for STRIPS and ADL. Moreover, it was subsequently extended and variants were defined.
- STRIPS and ADL are adequate for many real-world domains, but they have some significant restrictions.
  - An important one is that *ramifications* of actions cannot be represented in a natural way.
    - Indirect actions, like dust particles moving with airplanes, need to be represented as *direct effects*
    - ➥ it would be more natural if these changes could be *derived* from the location of the plane.
- Also, classical planning systems do not attempt to address the *qualification problem*.

Example: Air Cargo Transport

We describe in pure STRIPS notation the problem of loading and unloading cargo onto and off planes and flying it from place to place.

- We use three actions: *Load*, *Unload*, and *Fly*.
- The actions affect two predicates:
  - $In(c, p)$: cargo c is inside plane p,
  - $At(x, a)$: object x is at airport a.

Example: Air cargo transport (ctd.)

$$\begin{aligned}
&\mathit{Init}(\mathit{At}(C_1, \mathit{SFO}) \land \mathit{At}(C_2, \mathit{JFK}) \land \mathit{At}(P_1, \mathit{SFO}) \land \mathit{At}(P_2, \mathit{JFK}) \land \mathit{Cargo}(C_1) \land \\
&\quad \mathit{Cargo}(C_2) \land \mathit{Plane}(P_1) \land \mathit{Plane}(P_2) \land \mathit{Airport}(\mathit{SFO}) \land \mathit{Airport}(\mathit{JFK})) \\
&\mathit{Goal}(\mathit{At}(C_1, \mathit{JFK}) \land \mathit{At}(C_2, \mathit{SFO})) \\
&\mathit{Action}(\mathit{Load}(c, p, a), \\
&\quad \text{PRECOND: } \mathit{At}(c, a) \land \mathit{At}(p, a) \land \mathit{Cargo}(c) \land \mathit{Plane}(p) \land \mathit{Airport}(a), \\
&\quad \text{EFFECT: } \neg \mathit{At}(c, a) \land \mathit{In}(c, p)) \\
&\mathit{Action}(\mathit{Unload}(c, p, a), \\
&\quad \text{PRECOND: } \mathit{In}(c, p) \land \mathit{At}(p, a) \land \mathit{Cargo}(c) \land \mathit{Plane}(p) \land \mathit{Airport}(a), \\
&\quad \text{EFFECT: } \mathit{At}(c, a) \land \neg \mathit{In}(c, p)) \\
&\mathit{Action}(\mathit{Fly}(p, \mathit{from}, \mathit{to}), \\
&\quad \text{PRECOND: } \mathit{At}(p, \mathit{from}) \land \mathit{Plane}(p) \land \mathit{Airport}(\mathit{from}) \land \mathit{Airport}(\mathit{to}), \\
&\quad \text{EFFECT: } \neg \mathit{At}(p, \mathit{from}) \land \mathit{At}(p, \mathit{to}))
\end{aligned}$$

- The following plan is a solution to the problem:
  $[\mathit{Load}(C_1, P_1, \mathit{SFO}), \mathit{Fly}(P_1, \mathit{SFO}, \mathit{JFK}), \mathit{Unload}(C_1, P_1, \mathit{JFK}),$
  $\mathit{Load}(C_2, P_2, \mathit{JFK}), \mathit{Fly}(P_2, \mathit{JFK}, \mathit{SFO}), \mathit{Unload}(C_2, P_2, \mathit{SFO})].$

Example: Air Cargo Transport (ctd.)

Init	$At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \land Cargo(C_1) \land$ $Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Load(C_1, P_1, SFO),$ $\quad$ PRECOND: $At(C_1, SFO) \land At(P_1, SFO) \land Cargo(C_1) \land Plane(P_1) \land Airport(SFO),$ $\quad$ EFFECT: $\neg At(C_1, SFO) \land In(C_1, P_1))$
s_1	$In(C_1, P_1) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \land Cargo(C_1) \land$ $Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Fly(P_1, SFO, JFK),$ $\quad$ PRECOND: $At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK),$ $\quad$ EFFECT: $\neg At(P_1, SFO) \land At(P_1, JFK))$
s_2	$In(C_1, P_1) \land At(C_2, JFK) \land At(P_1, JFK) \land At(P_2, JFK) \land Cargo(C_1) \land$ $Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Unload(C_1, P_1, JFK),$ $\quad$ PRECOND: $In(C_1, P_1) \land At(P_1, JFK) \land Cargo(C_1) \land Plane(P_1) \land Airport(JFK),$ $\quad$ EFFECT: $At(C_1, JFK) \land \neg In(C_1, P_1))$
s_3	$At(C_1, JFK) \land At(C_2, JFK) \land At(P_1, JFK) \land At(P_2, JFK) \land Cargo(C_1) \land$ $Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$

Example: Air Cargo Transport (ctd.)

s_3	$At(C_1, JFK) \wedge At(C_2, JFK) \wedge At(P_1, JFK) \wedge At(P_2, JFK) \wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(SFO) \wedge Airport(JFK))$
$\Downarrow$	$Action(Load(C_2, P_2, JFK),$ PRECOND: $At(C_2, JFK) \wedge At(P_2, JFK) \wedge Cargo(C_2) \wedge Plane(P_2) \wedge Airport(JFK),$ EFFECT: $\neg At(C_2, JFK) \wedge In(C_2, P_2))$
s_4	$At(C_1, JFK) \wedge In(C_2, P_2) \wedge At(P_1, JFK) \wedge At(P_2, JFK) \wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(SFO) \wedge Airport(JFK))$
$\Downarrow$	$Action(Fly(P_2, JFK, SFO),$ PRECOND: $At(P_2, JFK) \wedge Plane(P_2) \wedge Airport(JFK) \wedge Airport(SFO),$ EFFECT: $\neg At(P_2, JFK) \wedge At(P_2, SFO))$
s_5	$At(C_1, JFK) \wedge In(C_2, P_2) \wedge At(P_1, JFK) \wedge At(P_2, SFO) \wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(SFO) \wedge Airport(JFK))$
$\Downarrow$	$Action(Unload(C_2, P_2, SFO),$ PRECOND: $In(C_2, P_2) \wedge At(P_2, SFO) \wedge Cargo(C_2) \wedge Plane(P_2) \wedge Airport(SFO),$ EFFECT: $At(C_2, SFO) \wedge \neg In(C_2, P_2))$
s_6	$At(C_1, JFK) \wedge At(C_2, SFO) \wedge At(P_1, JFK) \wedge At(P_2, SFO) \wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(SFO) \wedge Airport(JFK))$

$\Longrightarrow$ s_6 satisfies the goal $At(C_1, JFK) \wedge At(C_2, SFO)$.

Example: Blocks World

One of the most famous planning domains is the *blocks world*
$\Longrightarrow$ consists of a set of cube-shaped blocks sitting on a table.

- The blocks can be stacked, but only one block can fit directly on top of another.
- A robot arm can pick up a block and move it to another position, either on the table or on top of another block.
- The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it.
- The goal is always to build one or more stacks of blocks, specified in terms of what blocks are on top of what other blocks.

Example: Blocks World (ctd.)

- We use $On(b, x)$ to indicate that block b is on x, where x is either another block or the table.
- The action $Move(b, x, y)$ expresses that block b is moved from the top of x to the top of y.
  - One of the preconditions on moving b is that no other block be on it.
  - In ADL, we could state this as a sentence of first-order logic: $\neg\exists x \; On(x, b)$, or, equivalently, $\forall x \; \neg On(x, b)$.
  - In STRIPS, we use a new predicate, $Clear(x)$, that is true when nothing is on x.

Example: Blocks World (ctd.)

- We can formally describe *Move* in STRIPS as follows:
  $Action(Move(b, x, y),$
  $\quad$ $\textsc{Precond}$: $On(b, x) \wedge Clear(b) \wedge Clear(y),$
  $\quad$ $\textsc{Effect}$: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$

- But this action does not maintain *Clear* properly when x or y is the table:
  - for $x = Table$, we get $Clear(Table)$, but the table should not become clear,
  - for $y = Table$, it has the precondition $Clear(Table)$, but the table does not have to be clear to move a block onto it.

Example: Blocks World (ctd.)

To fix this, we do the following:

1. We introduce another action to move a block b from x to the table:
   $Action(MoveToTable(b, x),$
   $\quad$ PRECOND: $On(b, x) \land Clear(b),$
   $\quad$ EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$

2. We interpret $Clear(b)$ as "there is a clear space on b to hold a block"
   $\Longrightarrow Clear(Table)$ will always be true.

Example: Blocks World (ctd.)

- One caveat in doing this:
  - Nothing prevents a planner from using $\mathit{Move}(b, x, \mathit{Table})$ instead of $\mathit{MoveToTable}(b, x)$
    - it will lead to a larger-than-necessary search space albeit to no incorrect answers
    - to avoid this, we can introduce the predicate *Block* and add $\mathit{Block}(b) \wedge \mathit{Block}(y)$ to the precondition of *Move*.
- There is also the problem of spurious actions like $\mathit{Move}(B, C, C)$.
  $\Longrightarrow$ can be avoided by adding inequalities.
- The complete specification of the blocks world problem is given next (in slightly generalised STRIPS notation, as discussed).

Example: Blocks World (ctd.)

$$\begin{aligned}
&\mathit{Init}(\mathit{On}(A, \mathit{Table}) \wedge \mathit{On}(B, \mathit{Table}) \wedge \mathit{On}(C, \mathit{Table}) \wedge \\
&\quad \mathit{Block}(A) \wedge \mathit{Block}(B) \wedge \mathit{Block}(C) \wedge \mathit{Clear}(A) \wedge \mathit{Clear}(B) \wedge \mathit{Clear}(C)) \\
&\mathit{Goal}(\mathit{On}(A, B) \wedge \mathit{On}(B, C)) \\
&\mathit{Action}(\mathit{Move}(b, x, y), \\
&\quad \text{PRECOND: } \mathit{On}(b, x) \wedge \mathit{Clear}(b) \wedge \mathit{Clear}(y) \wedge \mathit{Block}(b) \wedge \mathit{Block}(y) \wedge \\
&\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y), \\
&\quad \text{EFFECT: } \mathit{On}(b, y) \wedge \mathit{Clear}(x) \wedge \neg \mathit{On}(b, x) \wedge \neg \mathit{Clear}(y)) \\
&\mathit{Action}(\mathit{MoveToTable}(b, x), \\
&\quad \text{PRECOND: } \mathit{On}(b, x) \wedge \mathit{Clear}(b) \wedge \mathit{Block}(b) \wedge \mathit{Block}(x) \wedge (b \neq x), \\
&\quad \text{EFFECT: } \mathit{On}(b, \mathit{Table}) \wedge \mathit{Clear}(x) \wedge \neg \mathit{On}(b, x))
\end{aligned}$$

- The following plan is a solution to the problem: $[\mathit{Move}(B, \mathit{Table}, C), \mathit{Move}(A, \mathit{Table}, B)]$.

VU Einführung in die Künstliche Intelligenz

SS 2023

Hans Tompits

Institut für Logic and Computation
Forschungsbereich Wissensbasierte Systeme

www.kr.tuwien.ac.at

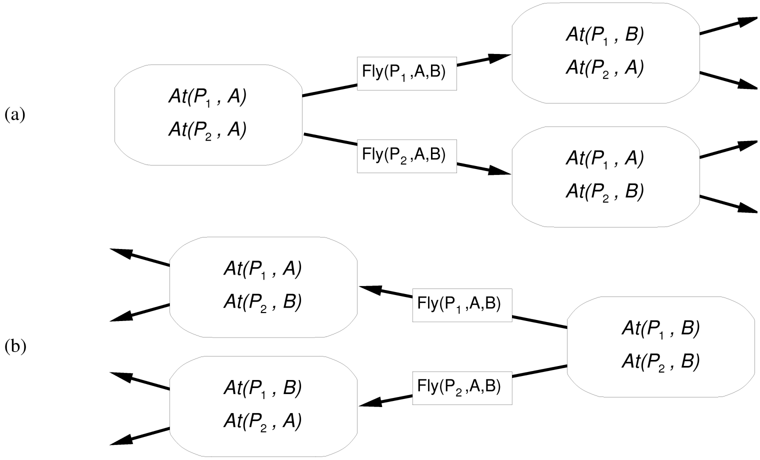
Planning with State-space Search

We now turn to *planning algorithms*.

- The most straightforward approach is to use *state-space search*.
- Two possibilities:
  - forward state-space search (or *progression planning*): from initial state to goal;
  - backward state-space search (or *regression planning*): from goal to initial state.

Planning with State-space Search (ctd.)

The two approaches illustrated: (a) progression planning; (b) regression planning.



Progression Planning

- We start in the problem's initial state, considering sequences of actions until we find a sequence that reaches a goal state.
- The formulation of planning problems as state-space search problems is as follows:
  - The *initial state* of the search is the initial state of the planning problem.
    - Each state will be a set of positive ground literals;
    - literals not appearing are false.
  - The *actions* that are applicable to a state are all those whose preconditions are satisfied.
    - The successor state resulting from an action is generated by adding the positive effect literals and deleting the negative effect literals.
    - ☞ In case of a first-order logic language, we must apply a unifier from the preconditions to the effect literals.

Progression Planning (ctd.)

- The *goal test* checks whether the state satisfies the goal of the planning problem.
- The *step cost* of each action is typically 1.
  - ☞ Allowing different costs for different actions could be easily realised, but this is seldom done for STRIPS planners.

Note: in the absence of function symbols, the state space of a planning problem is *finite*

➥ any complete graph search algorithm (like A^*) yields a complete planning algorithm!

Regression Planning

- ➤ The main advantage of backward search is that it allows to consider only *relevant* actions.

- ☞ An action is relevant to a conjunctive goal if it achieves one of the conjuncts of the goal.

Regression Planning (ctd.)

For instance:

- Consider the cargo problem with 20 pieces of cargo, having the goal

$$At(C_1, B) \land At(C_2, B) \land \ldots \land At(C_{20}, B).$$

- Seeking actions having, e.g., the first conjunct as effect, we find $Unload(C_1, p, B)$ as relevant.
  - This action will work only if its preconditions are satisfied.
    $\Longrightarrow$ any predecessor state must include the preconditions $In(C_1, p) \land At(p, B)$.
  - Moreover, the subgoal $At(C_1, B)$ should not be true in the predecessor state.
    $\Longrightarrow$ The predecessor state description is

$$In(C_1, p) \land At(p, B) \land At(C_2, B) \land \ldots \land At(C_{20}, B).$$

Regression Planning (ctd.)

Besides insisting that actions *achieve* some desired goal, they should not *undo* any desired literals.

- Actions satisfying this restriction are called *consistent*.
- E.g., $Load(C_2, p, B)$ would not be consistent with the current goal as it would negate the literal $At(C_2, B)$.

Regression Planning (ctd.)

We can now describe the general process of constructing predecessors for backward search.

- ➤ Given a goal description G, let A be an action that is relevant and consistent.
- ➤ The corresponding predecessor is as follows:
  - Any positive effects of A that appear in G are deleted.
  - Each precondition literal of A is added, unless it already appears.

➥ Any standard search algorithm can be used to carry out the search.

☞ In the first-order case, satisfaction might require a substitution for variables in the predecessor description.

Partial-order Planning

- Forward and backward state-space search are particular forms of *totally ordered plan* searches.
- They explore only *strictly linear sequences* of actions and do not take advantage of problem decomposition.
- ➥ Any planning algorithm that can place two actions into a plan without specifying which comes first is called a *partial-order planner*.

Example

Consider a simple example of putting on a pair of shoes:

$Init()$

$Goal(RightShoeOn \wedge LeftShoeOn)$

$Action(RightShoe, \textsc{Precond} : RightSockOn, \textsc{Effect} : RightShoeOn)$

$Action(RightSock, \textsc{Effect} : RightSockOn)$

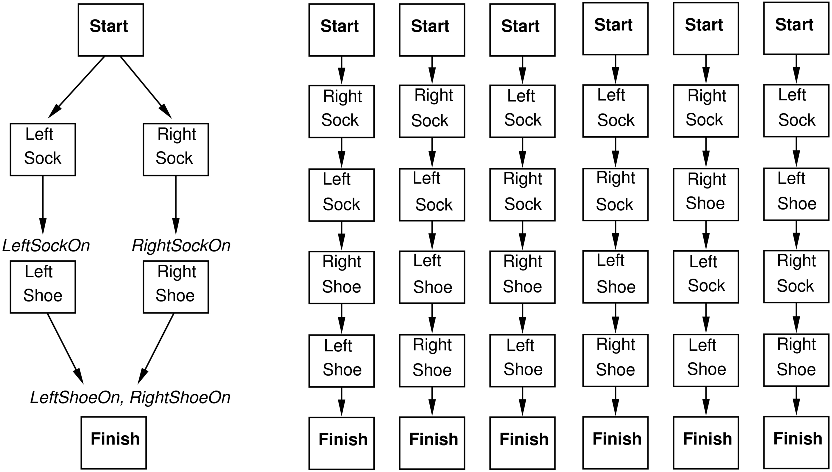
$Action(LeftShoe, \textsc{Precond} : LeftSockOn, \textsc{Effect} : LeftShoeOn)$

$Action(LeftSock, \textsc{Effect} : LeftSockOn)$

- A partial-order planner should come up with the following two-action sequences:
  - $[RightSock, RightShoe]$ to achieve the first conjunct of the goal and
  - $[LeftSock, LeftShoe]$ for the second conjunct.
- Then, the two sequences can be combined to yield the final plan.
- In doing so, the planner manipulates the two subsequences *independently*.

Example (ctd.)

Partial Order Plan: **Total Order Plans:**



Partial-order Planning—Basics

Partial-order planning can be implemented as a search in the *space of partial-order plans:*

- ➤ We start with an empty plan.
- ➤ Then, we consider ways of refining the plan until we come up with a complete plan that solves the problem.
- ➤ The actions in this search are not actions in the world but *actions on plans:*
  - adding a step to the plan;
  - imposing an ordering that puts one action before another;
  - and so on.

➥ We will define the *POP algorithm* for partial-order planning (as an instance of a search problem).

Partial-order Plans—Components

Each plan has the following four components:

1. a set of *actions*;
2. a set of *ordering constraints*;
3. a set of *causal links*;
4. a set of *open preconditions*.

Partial-order Plans—Components (ctd.)

The set of actions constitutes the elements for making up the steps of the plan.

- The actions are taken from the set of actions in the planning problem.
- The empty plan contains just the *Start* and *Finish* actions.
  - *Start* has no preconditions and has as its effect all the literals in the initial state of the planning problem.
  - *Finish* has no effects and has as its preconditions the goal literals of the planning problem.

Partial-order Plans—Components (ctd.)

- An ordering constraint is a pair of actions of the form $A \prec B$, read as "A before B".
  - $A \prec B$ means that action A must be executed sometime before action B, but not necessarily immediately before.
- The ordering constraints must describe a proper partial order.
- Any cycle, like $A \prec B$ and $B \prec A$, represents a *contradiction*
  - ➥ an ordering constraint cannot be added to the plan if it creates a cycle.

Partial-order Plans—Components (ctd.)

- A causal link between two actions A and B in the plan is an expression of form $A \xrightarrow{p} B$, read as "A achieves p for B".
- E.g., the causal link

$$\mathit{RightSock} \xrightarrow{\mathit{RightSockOn}} \mathit{RightShoe}$$

asserts that *RightSockOn* is an effect of the *RightSock* action and a precondition of *RightShoe*.

  - It also asserts that *RightSockOn* must remain true from the time of action *RightSock* to the time of action *RightShoe*.
  - In other words, the plan may not be extended by adding a new action C that *conflicts* with the causal link.

Partial-order Plans—Components (ctd.)

- An action C *conflicts* with $A \xrightarrow{p} B$ if
  1. C has the effect $\neg p$ and
  2. C could (according to the ordering constraints) come after A and before B.
- A precondition is *open* if it is not achieved by some action in the plan.
- Planners will work to reduce the set of open preconditions to the empty set, without introducing a contradiction.

Shoe-and-sock Example Revisited

For instance, the final plan in the shoe-and-sock example has the following components (omitting the ordering constraints that put every other action after *Start* and before *Finish*):

Actions: $\{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}$

Orderings: $\{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}$

Links: $\{RightSock \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe,$
$RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} Finish\}$

Open preconditions: $\{\}$

Partial-order Plans—Solutions

- We define a *consistent plan* as a plan in which
  - there are no cycles in the ordering constraints and
  - no conflicts with the causal links.

- A *solution* is a consistent plan with no open preconditions.

- ➥ Every linearisation of a partial-order solution is a total-order solution whose execution from the initial state will reach a goal state.

- ➥ We can extend the notion of "executing a plan" from total-order plans to partial-order plans:
  - A partial-order plan is executed by repeatedly choosing any of the possible next actions.

The POP Algorithm

- The initial plan contains
  - *Start* and *Finish*,
  - the ordering constraint $Start \prec Finish$,
  - no causal links, and
  - all the preconditions in *Finish* as open preconditions.
- The successor function arbitrarily picks
  - one open precondition p on an action B and
  - generates a successor plan for every possible consistent way of choosing an action A that achieves p.

The POP Algorithm (ctd.)

Consistency is enforced as follows:

1. The causal link $A \xrightarrow{p} B$ and the ordering constraint $A \prec B$ are added to the plan.
    - Action A may be an existing action in the plan or a new one.
    - If it is new, add it to the plan and also add $Start \prec A$ and $A \prec Finish$.
2. We resolve conflicts between (i) the new causal link and all existing actions and (ii) action A and all existing causal links, providing A is new.
    - A conflict between $A \xrightarrow{p} B$ and C is resolved by adding $B \prec C$ or $C \prec A$.
    - We add successor states for either or both if they result in consistent plans.

The POP Algorithm (ctd.)

- The goal test checks whether a plan is a solution to the original planning problem.

➥ Because only consistent plans are generated, the goal test just needs to check that there are no open preconditions.

Planning—Summary

- Planning systems are problem-solving algorithms that operate on explicit propositional or first-order representations of states and actions.
- STRIPS language:
  - describes actions in terms of their preconditions and effects and
  - describes the initial and goal states as conjunctions of positive literals.
- The ADL language relaxes some of these constraints,
  - allowing disjunction, negation, and quantifiers.

Planning—Summary (ctd.)

- State-space search can operate in the forward direction ("progression") or the backward direction ("regression").
- *Partial-order planning* algorithms explore the space of plans without committing to a totally ordered sequence of actions.
  - They work back from the goal, adding actions to the plan to achieve each subgoal.
  - They are particularly effective on problems amenable to a divide-and-conquer approach.

Planning—Summary (ctd.)

Further planning methods:

- The GRAPHPLAN system (Blum and Furst, 1995, 1997) is based on a graphical data structure, called a *planning graph*, for extracting plans.
  - A planning graph consists of a sequence of layers that correspond to time steps in the plan.
  - Each layer contains a superset of all the literals or actions that could occur at that time step and encodes *mutual exclusion relations* among literals or actions.
  - The GRAPHPLAN algorithm allows to extract a plan from the planning graph.

Planning—Summary (ctd.)

- Kautz and Selman (1992) proposed to solve planning problems by *translating them into formulas of propositional logic* such that
  - the *plans* of a given planning problem P are given by the *models* of the associated formula A.

  $\Longrightarrow$ This method is referred to as *planning as satisfiability*, and the corresponding algorithm is called SATPLAN.
- The planning-as-satisfiability approach was realised in the BLACKBOX planner (Kautz and Selman, 1998), incorporating also ideas from the GRAPHPLAN algorithm.

VU Einführung in die Künstliche Intelligenz

SS 2023

Hans Tompits

Institut für Logic and Computation
Forschungsbereich Wissensbasierte Systeme

www.kr.tuwien.ac.at

Making Simple Decisions

Decision Theory

- Decision-theoretic agent:
  - combines *utility theory* with *probability theory*
  - makes rational decisions based on *beliefs* and *desires* in contexts of uncertainty and conflicting goals
  - has a continuous measure of outcome quality
    - ➥ in contrast to *goal-based agents* that have only a binary distinction between good (goal) and bad (non-goal) states.
- Decision theory:
  - In its simplest form, deals with choosing among actions based on the desirability of their *immediate outcomes*.
  - Thereby, the environment is assumed to be *episodic*, i.e.,
    - an agent's experience can be divided into atomic episodes such that succeeding episodes do not depend on actions taken in previous episodes.
    - ➥ This is in contrast to *sequential* environments, where current decisions influence future decisions.

Outcomes and Utilities

- We furthermore deal with *nondeterministic, partially observable environments*.
  - Possible outcome states are represented in terms of *random variables*:
    - $\textsc{Result}(a)$ denotes a random variable whose values are the possible outcome states for taking action a.
  - The probability of outcome s', given evidence observations $\mathbf{e}$, is written as

$$P(\textsc{Result}(a) = s' | a, \mathbf{e}),$$

where a stands for the event that action a is executed.

- The agent's preferences are expressed by a *utility function* $U(s)$
  - assigns a single number to a state to express its desirability

Expected Utility

- The *expected utility* of an action a given evidence $\mathbf{e}$, denoted $EU(a|\mathbf{e})$, is the average utility value of the outcomes, weighted by the probability that the outcome occurs:

$$EU(a|\mathbf{e}) = \sum_{s'} P(\textsc{Result}(a) = s' | a, \mathbf{e}) U(s')$$

- *Principle of maximum expected utility* (MEU):
  - a rational agent should choose the action that maximises the agent's expected utility:

$$action = \underset{a}{argmax}\ EU(a|\mathbf{e}).$$

Preferences

- The MEU principle can be derived from general conditions that a rational agent should have.
- We use the following notation to describe an agent's preferences:
  - $A \succ B$: the agent prefers A over B;
  - $A \sim B$: the agent is indifferent between A and B;
  - $A \succsim B$: the agent prefers A over B or is indifferent between them.
- What sort of things are A and B?
  - States of the world, **but:** uncertainty about what is really being offered.
    - E.g., if you are an airline passenger and are offered pasta or chicken, you do not really know what lurks beneath the tinfoil cover.

  ➥ The set of outcomes for each action can be seen as a *lottery*, where each action is a ticket.

Lottery

- A lottery, L, with possible outcomes $S_1, \ldots, S_n$ that occur with probabilities $p_1, \ldots, p_n$ is written as

$$L = [p_1, S_1; p_2, S_2; \ldots; p_n, S_n].$$

  - Each S_i is either an atomic state or another lottery.

- Primary issue of utility theory:
  - How do preferences between complex lotteries relate to preferences between the underlying states in those lotteries?

- To address this issue, we list some conditions that we require that any reasonable preference relation should obey.

Axioms of Utility Theory

- **Orderability:** Given any two lotteries, a rational agent cannot avoid deciding which one it prefers, or whether it is indifferent between them.

  Exactly one of $(A \succ B), (B \succ A),$ or $(A \sim B)$ holds.

- **Transitivity:**

  $$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C).$$

- **Continuity:** If some lottery B is between A and C in preference, then:
  - there is some probability p for which the agent will be indifferent between getting B for sure and the lottery that yields A with probability p and C with probability $1 - p$.

  $$A \succ B \succ C \Rightarrow \exists p \, [p, A; 1 - p, C] \sim B.$$

Axioms of Utility Theory (ctd.)

- **Substitutability:** If an agent is indifferent between A and B, then it is indifferent between two more complex lotteries that are the same except that B is substituted for A.

  $$A \sim B \Rightarrow [p, A; 1 - p, C] \sim [p, B; 1 - p, C].$$

  This holds for $\succ$ instead of $\sim$ as well.

- **Monotonicity:** Suppose two lotteries have the same possible outcomes A and B.
  - If an agent prefers A to B, then the agent must prefer precisely the lottery that has a higher probability for outcome A.

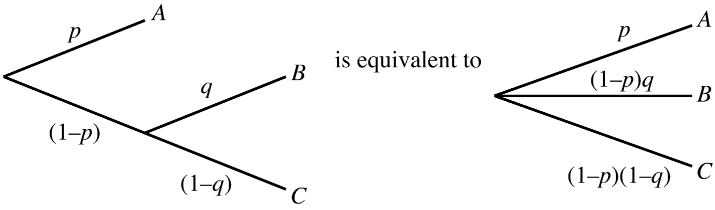
  $$A \succ B \Rightarrow (p > q \Leftrightarrow [p, A; 1 - p, B] \succ [q, A; 1 - q, B]).$$

Axioms of Utility Theory (ctd.)

- **Decomposability:** Compound lotteries can be reduced to simpler ones using the laws of probability.

  $[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C]$.

  This is known as the "no fun in gambling" rule:
  - two consecutive lotteries can be compressed into a single equivalent lottery.

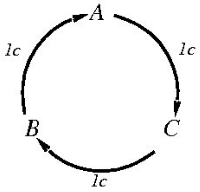


Axioms of Utility Theory (ctd.)

- These conditions are known as the *axioms of utility theory*.
- Each axiom can be motivated by showing that an agent violating it will exhibit irrational behaviour.
- Consider, e.g., an agent with *intransitive preferences* $A \succ B \succ C \succ A$ can be induced to give away all his money:

1. If the agent has A, we could offer to trade C for A plus one cent. The agent prefers C, so is willing to make the trade.
2. We then offer B for C, extracting another cent, as the agent prefers B over C.
3. Finally, we trade A for B. We are back to 1 except that the agent gave us 3 cents.
4. We continue until the agent has no money.

$\Longrightarrow$ Clearly, the agent behaves irrationally.



Existence of Utility Function

As shown by von Neumann and Morgenstern (1944), the axioms of utility theory imply the following:

- *Existence of Utility Function:* Given an agents preferences that satisfy the axioms of utility theory, there exists a real-valued function U such that

$$U(A) > U(B) \quad \Leftrightarrow \quad A \succ B$$
$$U(A) = U(B) \quad \Leftrightarrow \quad A \sim B$$

- *Expected Utility of a Lottery:* The utility of a lottery is the sum of the probability of each outcome times the utility of that outcome.

$$U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i).$$

Utility Scales and Assessment

The preceding results show that a utility function exists, but they do not imply that it is *unique*.

- It can be shown that an agent's behaviour does not change if its utility function $U(S)$ is replaced by

  $$U'(S) = aU(S) + b,$$

  where a and b are constants and $a > 0$.

➥ U is determined up to *linear (affine) transformations*.

Utility Scales and Assessment (ctd.)

- In deterministic environments, where there are states and no lotteries, the behaviour of an agent is unchanged by an application of any *monotonic* transformation.
  - For instance, we could apply the square root to all utilities without changing the priority order of actions.
  - One says:
    - An agent in a deterministic environment has a *value function* or *ordinal utility function*,
    - i.e., such functions just provide a *preference ranking on states*—the numbers do not matter.

- How to work out an agents utility function?
  - Present choices to an agent and use observed preferences to pin down the underlying utility function.
  - This process is called *preference elicitation*.

Utility Scales

- As we have seen, there is no *absolute* scale for utilities but it is useful to establish *some* scale for any particular problem.
- How to establish a scale?
  - Fix the utilities of any two particular outcomes.
  - Typically, we fix the utility of a "best possible prize" S_b at $U(S_b) = u_\top$ and a "worst possible catastrophe" S_w at $U(S_w) = u_\bot$.
  - *Normalized utilities* use a scale with $u_\bot = 0$ and $u_\top = 1$.

Utility Scales: Examples

- Some attempts have been made to find out the value that people place on their own lives.
- One common "currency" in medical and safety analysis is the *micromort:*
  - the event of a one-in-a-million chance of death.
- If people are asked how much they would pay to avoid a risk of a one-in-a-million chance of death they will respond with very large numbers, but their actual behaviour reflects a much lower monetary value for a micromort.
  - E.g., driving in a car for 370 km incurs a risk of one micromort; for a car with, say 150.000 km, that's about 400 micromorts.
  - People appear to be willing to pay about 10.000 Dollars more for a safer car that halves the risk of death (i.e., to incur 200 micromorts instead of 400), or about 50 Dollar per micromort.
- In general, studies on a large number of people showed that one micromort amounts to ca. 20 Dollars (in 1980s money).

Utility Scales: Examples (ctd.)

- Another measure is the *QALY* ("quality-adjusted life year"), useful for medical decisions involving substantial risks:
  - one QALY equates to one year in perfect health.
- The QALY is an indicator for the time-trade-off (TTO) to choose between remaining in a state of ill health for a period of time vs. being restored to perfect health but having a shorter life expectancy.
  - E.g., on average, kidney patients are indifferent between living two years on a dialysis machine and one year at full health.

The Utility of Money

- Money plays a significant role in human utility functions.
- Usually, an agent exhibits a *monotonic preference* for more money, all other things being equal ("ceteris paribus"), i.e., the agent prefers more money to less.
- **But:** this does not mean that money behaves as a utility function, because it says nothing about preferences between *lotteries* involving money.
- Example:
  - Suppose you have won in a game show and are offered a choice:
    - either take the $1,000,000 prize *or*
    - gamble it on the flip of a coin: the coin coming up heads means you end up with nothing, the coin coming up tails means you get $2,500,000.
  - How would you decide?

The Utility of Money (ctd.)

- Assuming the coin is fair, i.e., there is a 50:50 chance for coming up heads or tails, the *expected monetary value* (*EMV*) of the gamble is

$$\frac{1}{2} \cdot \$0 + \frac{1}{2} \cdot \$2,500,000 = \$1,250,000$$

$\Longrightarrow$ The EMV is more than the original \$1,000,000, but is accepting the gamble the better decision?

- Let S_n denote a state of possessing n Dollars, and say your current wealth is k Dollars.
  - The expected utilities of accepting and declining the gamble are

$$\begin{aligned} EU(Accept) &= \frac{1}{2} \cdot U(S_k) + \frac{1}{2} \cdot U(S_{k+2,500,000}) \\ EU(Decline) &= U(S_{k+1,000,000}). \end{aligned}$$

The Utility of Money (ctd.)

- How to define the utility?
  - The utility is not directly proportional to monetary value, because the utility for the first million is very high, but what about the utility for the next million?
- Assume you assign a utility of 5 to your current financial status S_k, 9 to the state $S_{k+2{,}500{,}000}$, and 8 to the state $S_{k+1{,}000{,}000}$.
- Then:

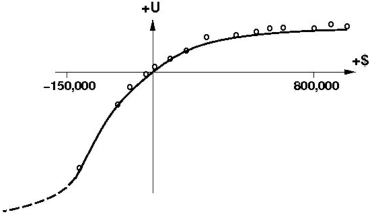
$$\begin{aligned} EU(Accept) &= \frac{1}{2} \cdot U(S_k) + \frac{1}{2} \cdot U(S_{k+2{,}500{,}000}) = \frac{5}{2} + \frac{9}{2} = 7 \\ EU(Decline) &= U(S_{k+1{,}000{,}000}) = 8. \end{aligned}$$

$\Longrightarrow$ the rational action would be to decline, because the expected utility of accepting is 7 and for declining 8.

- On the other hand, a billionaire would most likely have a utility function that is locally linear over the range of a few million more, and thus would accept.

The Utility of Money (ctd.)

- In a pioneering study of actual utility functions, Grayson (1960) found that the utility of money was almost exactly proportional to the *logarithm* of the amount.



- Preferences between different levels of debt can display a reversal of the concavity associated with positive wealth.
  - E.g., someone already $10,000,000 in debt might well accept a gamble on a fair coin with a gain of $10,000,000 for heads and a loss of $20,000,000 for tails.

    $\Longrightarrow$ This leads to the S-shaped form of the curve.

Risks

- For a positive wealth, given a lottery L with expected monetary value $EMV(L)$, usually $U(L) < U(S_{EMV(L)})$, where $S_{EMV(L)}$ is the state of being handed the expected money of the lottery as the sure thing.
  - I.e., people are *risk-averse*—they prefer a sure thing with a payoff that is less than the expected monetary value of a gamble.
- On the other hand, when in large debt, the behaviour is *risk-seeking*.
- The value an agent will accept in lieu of a lottery is the *certainty equivalent* of the lottery.
  - Studies have shown that most people will accept about 400 Dollars in lieu of a gamble that gives 1000 dollars half the time and 0 Dollar the other half.
  - That is, the certainty equivalent of the lottery is 400 Dollars vs. the EMV of 500 Dollars.
    $\Longrightarrow$ The difference is called the *insurance premium*.

Risks (ctd.)

- Risk aversion is the basis for the insurance industry, because it means that insurance premiums are positive.
- People would rather pay a small insurance premium than gamble the price of their house against the chance of a fire.
  - The price of a house is small compared with the insurance company's total reserves.
  - The insurance company's utility curve is approximately linear over such a small region, and the gamble costs the company almost nothing.
- Note:
  - for small changes in wealth compared to the current wealth, almost any curve will be approximately linear.
  - An agent that has a linear curve is said to be *risk-neutral*.

VU Einführung in die Künstliche Intelligenz

SS 2023

Hans Tompits

Institut für Logic and Computation
Forschungsbereich Wissensbasierte Systeme

www.kr.tuwien.ac.at

Human Judgment and Irrationality

- Decision theory is a *normative theory*, i.e., it describes how a rational agent *should* act.
- A *descriptive theory*, on the other hand, describes how agents (e.g., humans) *really do* act.
- Evidence suggests that these two kinds of theories do not coincide $\Longrightarrow$ humans appear to be "predictably irrational".

Allais Paradox

- Assume that there is a choice between lotteries A and B and then between C and D, which have the following prizes:
  - A: 80% chance of winning \$4000
  - B: 100% chance of winning \$3000
  - C: 20% chance of winning \$4000
  - D: 25% chance of winning \$3000
- Most people prefer B over A (taking the sure thing), and C over D (taking the higher EMV).
- However, the normative analysis yields a different result:
  - Assume, without loss of generality, a utility function with $U(\$0) = 0$.
  - Then, $B \succ A$ implies $U(\$3000) > 0.8 \cdot U(\$4000)$, and $C \succ D$ implies $0.2 \cdot U(\$4000) > 0.25 \cdot U(\$3000)$.
  - From the latter we obtain
  $$U(\$3000) < \frac{0.2}{0.25} U(\$4000) = 0.8 \cdot U(\$4000).$$
  - ➥ There is no utility function consistent with theses choices!

Allais Paradox (ctd.)

- One possible explanation for the apparent irrational preferences is the *certainty effect*, i.e., people are strongly attracted to gains that are certain.
- Why is that?

Allais Paradox (ctd.)

- Possible answers:
  1. People may choose to reduce their computational burden: by choosing the certain outcomes, there is no need to estimate probabilities.
  2. People may mistrust the legitimacy of the stated probabilities (in particular, if stated by people with a vested interest in the outcomes).
  3. People may account their emotional state as well as their financial state.
     - People know they would experience *regret* if they gave up a certain reward (B) for an 80% chance of a higher reward and then lost.
     - I.e., in choosing A, there is a 20% chance of getting no money and *feeling like a complete idiot*, which is worse than just getting no money.

➥ Choosing B over A and C over D may not be irrational: just willing to give up \$200 EMV to avoid a 20% chance of feeling like an idiot.

Ellsberg Paradox

- Prizes have an equal value, but probabilities are underconstrained.
- Payoff depends on the color of a ball chosen from an urn.
- You are told that the urn contains 1/3 red balls, and 2/3 either black or yellow balls, but you do not know how many black and how many yellow.
- Then, you are asked to choose between A and B, and then between C and D:
  - A: \$100 for a red ball
  - B: \$100 for a black ball

  - C: \$100 for a red or a yellow ball
  - D: \$100 for a black or yellow ball
- If you think there are more red than black balls, you should prefer A over B and C over D, and the opposite otherwise.
- *But* most people prefer A over B and D over C!

➥ People have *ambiguity aversion*.

Ellsberg Paradox (ctd.)

Ambiguity aversion (ctd.):

- A: $100 for a red ball
- B: $100 for a black ball

- C: $100 for a red or a yellow ball
- D: $100 for a black or yellow ball

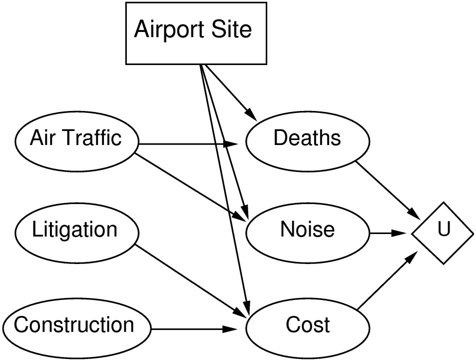
➤ A gives you a $1/3$ chance of winning, while B could be anywhere between 0 and $2/3$.

➤ Likewise, D gives you a $2/3$ chance, while C could be anywhere between $1/3$ and $3/3$.

➥ Most people *elect the known probability* rather than the unknown one.

Decision Networks

- *Decision networks* (or *influence diagrams*) are a general framework for supporting rational decisions.
- They contain information about an agent's current state, its possible actions, the state that will result from the agent's action, and the utility of that state.
- Example of a decision network for the *airport siting problem:*



Decision Networks (ctd.)

Decision network uses three types of nodes:

- *Chance nodes* (ovals): represent random variables.
  - E.g., the agent is uncertain about construction costs, the level of air traffic, the potential for litigation.
  - There are also the *Deaths*, *Noise*, and *Cost* variables, depending on the site chosen.
  - Chance nodes are associated with a conditional probability distribution that is indexed by the state of the parent nodes.
- *Decision nodes* (rectangles): represent points where a decision maker has a choice of actions; e.g., the choice of an airport site influences the cost, noise, etc.
- *Utility nodes* (diamonds): represent the agent's utility function.
  - It has as parents all variables describing the outcome that directly affect utility.

Evaluating Decision Networks

- Algorithm for evaluating decision networks:
  1. Set the evidence variables for the current state.
  2. For each possible value of the decision node:
     a) Set the decision node to that value.
     b) Calculate the posterior probabilities for the parent nodes of the utility node, using a standard probabilistic inference algorithm.
     c) Calculate the resulting utility for the action.
  3. Return the action with the highest utility.

☞ Decision networks are an extension of *Bayesian networks*, in which only chance nodes occur.

The Value of Information

- In the decision network analysis it is assumed that *all relevant information* is available before making a decision.
- In practice this is hardly ever the case:
  - ☞ *One of the most important parts of decision making is knowing what questions to ask.*
- *Information value theory* enables an agent to choose what information to acquire.
- Basic assumption:
  - the agent can acquire the value of any observable chance variables.
- These observation actions affect only the *belief state*, not the external physical state.
- The value of an observation derives from the *potential* to affect the agent's eventual physical action $\Longrightarrow$ this potential can be estimated directly from the decision model itself.

The Value of Information: Example

A simple example:

- An oil company plans to buy one of n indistinguishable blocks of ocean-drilling rights.
- One of the blocks contains oil worth C dollars, while all other are worthless.
- The price for each block is C/n Dollars.
- If the company is *risk neutral*, then it is indifferent between buying a block and not buying one.
- Now assume that the company can buy information (results of a survey) that says definitively whether block 3 contains oil or not.
- How much should the company be willing to pay for this information?

Example (ctd.)

To answer this question, we examine what the company would do if it had the information:

- With probability $1/n$, the survey will indicate oil in block 3.
  - In this case, the company will buy block 3 for C/n dollars and make a profit of $C - C/n = (n-1)C/n$ dollars.
- With probability $(n-1)/n$, the survey will show that block 3 contains no oil, hence the company will buy a different one.
  - Now, the probability of finding oil in one of the other blocks changes from $1/n$ to $1/(n-1)$, so the expected profit is $\frac{C}{(n-1)} - \frac{C}{n} = \frac{C}{n(n-1)}$ Dollars.
- Then, the resulting expected profit, given the survey information is

$$\frac{1}{n} \cdot \frac{(n-1)C}{n} + \frac{n-1}{n} \cdot \frac{C}{n(n-1)} = \frac{C}{n}.$$

➥ The company should be willing to pay up to C/n Dollars
$\Longrightarrow$ the information is worth as much as the block itself!

Remarks

- The value of information derives from the fact that *with* the information, one's course of action can be changed to suit the *actual* situation.
- One can discriminate according to the situation:
  - without the information, one has to do what is *best on average* over the possible situations.
- In general, the value of a given piece of information is defined to be the difference in expected value between the best actions before and after an information is obtained.

The Value of Perfect Information

- Assumption:
  - *Exact evidence* about the value of a random variable E_j can be obtained (i.e., we learn $E_j = e_j$).
  - ➥ We use the phrase *value of perfect information* (*VPI*).
- Given initial evidence $\mathbf{e}$, the value of the current best action α is defined by

$$EU(\alpha|\mathbf{e}) = \max_a EU(a|\mathbf{e}) = \max_a \sum_{s'} P(\text{RESULT}(a) = s'|a, \mathbf{e})U(s').$$

- The value of the new best action α_{e_j} after evidence $E_j = e_j$ is obtained is

$$EU(\alpha_{e_j}|\mathbf{e}, e_j) = \max_a \sum_{s'} P(\text{RESULT}(a) = s'|a, \mathbf{e}, e_j)U(s').$$

- But the value of E_j is currently unknown, so to determine the value of discovering E_j, given current information $\mathbf{e}$, we average over all possible values e_{j_k} that might be discovered for E_j:

$$VPI_{\mathbf{e}}(E_j) = \Big(\sum_k P(E_j = e_{j_k}|\mathbf{e})EU(\alpha_{e_{j_k}}|\mathbf{e}, E_j = e_{j_k})\Big) - EU(\alpha|\mathbf{e}).$$

Some Properties of the VPI

- The expected value of information is *nonnegative*:

$$VPI_{\mathbf{e}}(E_j) \geq 0, \text{ for all } \mathbf{e} \text{ and all } E_j.$$

- VPI is *nonadditive*:

in general, $VPI_{\mathbf{e}}(E_j, E_k) \neq VPI_{\mathbf{e}}(E_j) + VPI_{\mathbf{e}}(E_k)$.

- VPI is *order independent*:

$$VPI_{\mathbf{e}}(E_j, E_k) = VPI_{\mathbf{e}}(E_k, E_j).$$

Decision-theoretic Expert Systems

- *Decision analysis* (evolved in the 1950s and 1960s) studies the application of decision theory to actual decision problems.
- Traditionally, there are two roles in decision analysis:
  - the *decision maker*, stating preferences between outcomes; and
  - the *decision analyst*, who enumerates possible actions and outcomes, and elicits preferences to determine the best course of action.
- Early expert system research concentrated on answering questions rather than on making decisions.
- The addition of *decision networks* allows expert systems to recommend optimal decisions, reflecting preferences as well as available evidence.

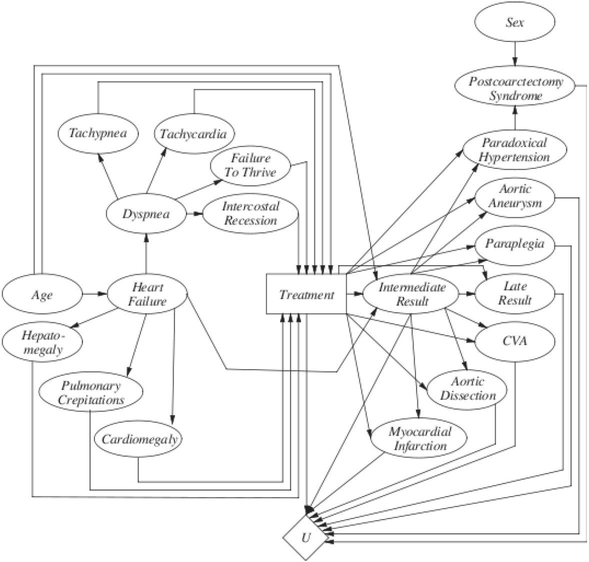
Decision-theoretic Expert Systems (ctd.)

The process of creating a decision-theoretic expert system, e.g., for selecting a medical treatment for congenital heart disease (aortic coarctation) in children:

1. create a causal model (e.g., determine symptoms, treatments, disorders, outcomes, etc.);
2. simplify to a qualitative decision model;
3. assign probabilities (e.g., from patient databases, literature studies, experts subjective assessments, etc.);
4. assign utilities (e.g., create a scale from best to worst outcome and give each a numeric value);
5. verify and refine the model, evaluate the system against correct input-output-pairs, a so called *gold standard*;
6. perform sensitivity analysis, i.e., check whether the best decision is sensitive to small changes in the assigned probabilities and utilities.

Influence Diagram Example

Influence diagram for aortic coarctation:



Summary

- *Decision theory* puts probability theory and utility theory together to describe what an agent *should do*.
- A *rational agent* makes decisions by considering all possible actions and choosing the one that leads to the best expected outcome.
- An agent whose preferences are consistent with a set of simple axioms possesses a *utility function*; furthermore, it selects actions as if maximising expected utility.
- The *value of information* is defined as expected improvement in utility compared with making a decision without the information.
- *Expert systems* that incorporate utility information are more powerful than pure inference systems:
  - they are able to make decisions and use the value of information to decide whether to acquire it, and
  - they can calculate their sensitivity to small changes in probability and utility assessments.

VU Einführung in die Künstliche Intelligenz

SS 2023

Hans Tompits

Institut für Logic and Computation
Forschungsbereich Wissensbasierte Systeme

www.kr.tuwien.ac.at

Philosophical Foundations of AI

Introduction

Can machines act intelligently in the way that humans do, and if so, would they have real conscious minds?

Android Data (right) in “The Measure of a Man” of “Star Trek: The Next Generation” (Episode 9, Season 2, 1989)

Introduction (ctd.)

Two different hypothesis:

- *Weak AI hypothesis*
  - assertion that machines could act *as if* they were intelligent
- *Strong AI hypothesis*
  - assertion that machines that do so are *actually* thinking (not just *simulating* thinking)

Weak AI

Is artificial intelligence possible?

- Answer depends on the definition of AI.
- One possible definition: AI is the quest for the *best agent program* on a given architecture.
  - Then, AI is possible by definition: given a digital architecture with k bits program storage, there exist 2^k agent programs.
  - Enumerate and test them all to find the best agent program.
- This definition works well (in theory) for the engineering problem of finding a good agent, given an architecture.
- But philosophers are interested in comparing two different architectures: *humans* and *machines*.

Can Machines think?

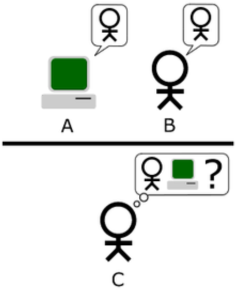
"The question whether *machines can think* [...] is about as relevant as the question whether *submarines can swim*" (Edsger Dijkstra, 1984)

Edsger Dijkstra

- Question of usage of words in a language:
  - E.g., definition of "swim" in American Heritage Dictionary:
    - To move through water by means of the limbs, fins, or tail.
  - In other languages ships do swim ...
  - ...and airplanes fly in English:
    - to fly = move through the air by means of wings or wing-like parts.
- Does *think* require "a brain" or just "brain-like" parts?

Turing Test

- Proposed by Alan Turing in his famous paper *Computing Machinery and Intelligence* (1950) as the "Imitation Game".
  - Involves three players: player A is a man, player B is a woman, and player C is the interrogator.
  - Player C is unable to see either player A or player B (and knows them only as X and Y), and can communicate with them only through written notes.



- By asking questions, C tries to determine which of the two is the man and which is the woman.
- The role of A is to trick the interrogator into making the wrong decision, while player B attempts to assist the interrogator in making the right one.

Turing Test (ctd.)

- Turing asks:

  > *"What will happen when a machine takes the part of A in this game?"*

- He conjectures that

  > *"in about fifty years' time it will be possible, to programme computers [. . .] to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning".*

- Early internet chatbots (around 1989) could fool people into believing talking to a real person (e.g., MGONZ, NATACHATA, or CYBERLOVER).

Turing Test (ctd.)

- Does ChatGPT pass the Turing test?
  - I asked it (June 1, 2023)!
- Here is the conversion:

  Q: *"Have you passed the Turing test?"*

  A: *"As an AI language model, I do not have personal experiences or awareness, so I cannot "pass" the Turing test in the traditional sense. The Turing test is a subjective test designed to determine whether a machine can exhibit intelligent behavior indistinguishable from that of a human. However, I have been trained on a vast amount of data and can engage in conversations that simulate human-like responses. While I strive to provide useful and coherent information, I am still an AI and may not always meet the criteria of passing the Turing test."*

Turing Test (ctd.)

Q: *“But as an AI language model, would you not be well suited for the imitation game?”*

A: *“Yes, as an AI language model, I am well suited for the imitation game, also known as the Turing test. The imitation game involves a human judge engaging in a conversation with both a human and a machine, without knowing which is which. The judge’s goal is to determine which entity is the machine based solely on the responses given.*
As an AI language model, I have been designed to generate human-like responses to various prompts and questions. While I may not always provide perfect responses or convincingly imitate a human in every scenario, my purpose is to simulate human conversation to the best of my abilities. So, in the context of the imitation game, I can certainly participate and attempt to convince the judge that I am the human counterpart.”

Turing Test (ctd.)

Q: *"So, have you been subject to the imitation game?"*

A: *"As an AI language model, I don't have direct knowledge of whether I have been subject to the imitation game or specific instances where I have been tested in that manner. My purpose is to provide information and assist users with their queries to the best of my abilities. OpenAI, the organization behind ChatGPT, has conducted various tests and evaluations to assess the capabilities of AI models like mine, including comparisons with human performance on certain tasks. However, the specific details and methodologies of those evaluations are not something I have access to."*

Q: *"I see. Thank you!"*

A: *"You're welcome! If you have any more questions, feel free to ask. I'm here to help!"*

Turing Test (ctd.)

Turing Test (ctd.)

- Turing himself examined different possible objections to the possibility of intelligent machines,
  - including almost all of those that have been discussed after the publication of his famous paper.

➥ We will look at some of them in what follows.

Argument from Disability

- The *argument from disability* makes the claim that "a machine can never do X", for some property X.
- Examples for X listed by Turing:
  - be kind, resourceful, friendly
  - have a sense of humor, tell right from wrong, make mistakes
  - fall in love, make someone fall in love with it
  - learn from experience, be the subject of its own thoughts
  - enjoy strawberries and ice cream
  - use words properly
  - do something really new

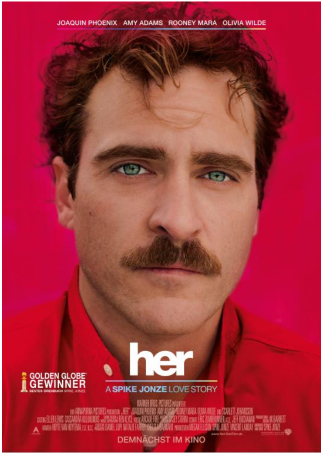
Argument from Disability (ctd.)

- However, it is clear that computers can do many things as well as or better than humans, including things that people believe require great human insight and understanding.
- Examples:
  - Statistical learning algorithms predict the success of students in training programs or the recidivism of a criminal better than experts.
  - The *Graduate Management Admission Test* (*GMAT*) is a computer-based assessment for predicting the success in the first year of graduate management education.
    - The program agrees with human graders in 97% of the time, this is about the same level as two human graders agree.
  - Computers have made small but significant discoveries in astronomy, mathematics, chemistry, etc. Each of these required performance at the level of a human expert.

Fall in Love with Machines?

However, some persons (mostly men arguably) do occasionally fall in love with some machines . . .

Fall in Love with Machines? (Ctd.)



From the movie “Her”, by Spike Jonze (2013)

Fall in Love with each Other?

From the movie “WALL-E”, by Andrew Stanton (2008)

The Mathematical Objection

- In view of the work of Turing (1936) and Gödel (1931), certain mathematical questions are *in principle unanswerable by particular formal systems*.
- In particular, *Gödel's incompleteness theorem* (1931) is the most famous example of this.
  - For any axiomatic system F powerful enough to do arithmetic, it is possible to construct a *Gödel sentence* $G(F)$ with the following properties:
    - $G(F)$ is a sentence of F, but cannot be proved within F;
    - if F is consistent, then $G(F)$ is true.
- Philosophers like J.R. Lucas (1961) claimed that this theorem shows that machines are mentally inferior to humans.
  - Reason: machines are formal systems and thus they are limited by the incompleteness theorem, while humans have no such limitation.

The Mathematical Objection (ctd.)

Problems with this claim:

- Gödel's incompleteness theorem applies only to particular formal systems, including Turing machines.
  - Claim is based on the assertion that computers are Turing machines.
  - However, Turing machines are *infinite* while computers are *finite*.
  - So, computers can be described as a (very large) system in propositional logic $\Longrightarrow$ not subject to incompleteness theorem.
- On the other hand, it seems to be impossible to *prove* that humans are not subject of Gödel's incompleteness theorem.
  - Any rigorous proof would require a formalisation of the claimed unformalisable human talent, which would be a contradiction.
  - ➥ We are left with an appeal to intuition that humans can perform some feat a computer cannot.

The Mathematical Objection (ctd.)

- This appeal is often expressed with arguments such as

  "we must assume our own consistency, if thought is to be possible at all" (Lucas, 1976).

- However, if anything, humans are known to be inconsistent . . .
  . . . but some computers too . . .

HAL9000 in "2001: A Space Odyssey" by Stanley Kubrick (1968)

Argument from Informality

- Raised by Turing as the *"argument from informality of behaviour"*.
- This is the claim that human behaviour is too complex to be captured by a set of rules.
  - Because computers can do no more than follow a set rules, they cannot generate behaviour as intelligent as that of humans.

☞ The inability to capture everything in a set of logical rules is the *qualification problem* in AI.

Argument from Informality (ctd.)

- Principle proponent of this view:
  - philosopher Hubert Dreyfus, who wrote a series of critiques (1972, 1992), and also together with his brother Stuart (1986).
- The position they criticise is "Good Old-Fashioned AI" (or *GOFAI*).
  - GOFAI is supposed to claim that all intelligent behaviour can be captured by a system that reasons logically from a set of facts and rules.
  - ➥ Corresponds to the simplest logical agent.
  - ➥ Dreyfus critique thus is not addressed against computers per se, rather against AI based on sets of facts and rules.

Argument from Informality (ctd.)

- One of Dreyfus' strongest arguments is for *situated agents* rather than disembodied logical inference engines.
- An agent whose understanding of "dog" comes only from a limited set of logical sentences such as

$$Dog(x) \rightarrow Mammal(x)$$

  is at a disadvantage compared to an agent that has watched dogs run, has played with them, etc.
- According to philosopher Andy Clark (1998):

  > "Biological brains are first and foremost the control systems for biological bodies. Biological bodies move and act in rich real-world surroundings."

- To understand human intelligence, we have to consider *the whole agent*, not just the agent program.
- Claim of the *embodied cognition* approach: it makes no sense to consider the brain separately $\Longrightarrow$ we need to study the system as a whole.

Strong AI

Can machines really think?

- Claim of many philosophers:
  - a machine that passes the Turing test would still not be *actually thinking*, but would be only a *simulation* of thinking.
- Turing has foreseen this objection, he called it the *argument of consciousness*.
  - Machines have to be aware of their own mental states and actions
    $\Longrightarrow$ machines need actually feel *emotions*.
  - Marvin Minsky ("The Society of Mind", 1985):

    > "The question is not whether intelligent machines can have any emotions, but whether machines can be intelligent without emotions."

Strong AI (ctd.)

- Turing maintains that the question whether machines can think is ill-defined:
  - In ordinary life, we never have any *direct* evidence about the internal mental states of other humans.
  - Turing:

    > "Instead of arguing continually over this point, it is usual to have the polite convention that everyone thinks."
- Turing's answer suggests that the issue will eventually go away by itself once machines reach a certain level of sophistication.
  $\Longrightarrow$ Dissolves the difference between weak and strong AI.
- However, there may be a *factual* issue at stake: humans do have real minds, and machines might or might not.
- For this, the *mind-body problem* of philosophy is relevant.

Mind-Body Problem

- Already considered by ancient Greek philosophers and various schools of Hindu thought.
- First analysed in depth by René Descartes in *Meditations on First Philosophy* (1641).
  - Considered the mind's activity of thinking and the physical processes of the body.
  - Concludes that the two must exist in separate realms $\Longrightarrow$ *dualist theory*.
  - Famous quote: "cogito ergo sum" (Principles of Philosophy, 1644).
- Main question of dualist theory: How can the mind control the body if the two really separate?
  - Descartes: they might interact through the pineal gland.
  - But how does the mind control the pineal gland?

Mind-Body Problem (ctd.)

- The *monist* theory of mind, often called *physicalism*, avoids this problem.
- Claim:
  - mind is not separated from the body; mental states are physical states.
- But:
  - how can physical states simultaneously be mental states?

Functionalism

- The theory of *functionalism* says that a mental state is any *intermediate causal condition* between input and output.
- Hence, any two systems with isomorphic causal processes would have the same mental states.
  - So, a program could have the same mental states as a person.
  - Meaning of *isomorphic:* assumption of a level of abstraction below which the specific implementation does not matter.
- Functionalism is illustrated by the *brain replacement experiment:*
  - Introduced by philosopher Clark Glymour and discussed by John Searle (1980), most commonly associated with roboticist Hans Moravec (1988).

Brain Replacement Experiment

- Basic idea:
  - replace all the neurons in someone's head with electronic devices.
- Assumption:
  - neurophysiology has developed so far that the input-output behaviour and connectivity of all neurons in the human brain are perfectly understood, and
  - we can build microscopic devices that mimic this behaviour.
- The subject's external behaviour must remain unchanged compared with what would be observed if the operation were not carried out.
- But what about the internal experience of the subject?
  $\Longrightarrow$ Diverging views!

Brain Replacement Experiment (ctd.)

- Moravec, as a robotics researcher and functionalist, was convinced his consciousness would remain unaffected.
- Searle, a philosopher and biological naturalist, was equally convinced his consciousness would vanish.
- Formally, by replication of the functionalities of normal human brains, the experiment concludes that consciousness is a product of the electronic brain that appeals only to the functional properties of the neurons.
- This explanation must also apply to the real brain which has the same functional properties.

Biological Naturalism

- A strong challenge to functionalism was proposed by John Searle's (1980) *biological naturalism*.
  - According to this theory, mental states are high-level emergent features that are caused by low-level physical processes *in the neurons*
  - and it is the (unspecified) properties of the neurons that matter.
- Thus, mental states cannot be duplicated just on the basis of some program having the same functional structure.
  - ➥ We would require that the program be running on an architecture with the same causal power as neurons.
- To support this, Searle describes a hypothetical system that is running a program and passes the Turing test, but that *does not understand anything of its inputs and outputs* (the "Chinese Room").

➥ *Running the right program does not necessarily generate understanding.*

Epilogue: Ethics and Risks of AI

Some points for discussion:

- People might lose their jobs to automation.
- People might have too much (or too little) leisure time.
- People might lose their sense of being unique.
- AI systems might be used towards undesirable ends.
- The use of AI systems might result in a loss of accountability.
- The success of AI might mean the end of the human race ("AI singularity").

Take over the world?

Supercomputer V.I.K.I. in "I, Robot" (book by Isaac Asimov, 1950, film by Alex Proyas, 2004)

Take over the world?

Supercomputer V.I.K.I. in "I, Robot" (book by Isaac Asimov, 1950, film by Alex Proyas, 2004)

Take over the world? (Ctd.)

The Terminator franchise (starting with the 1984 movie “The Terminator”

Take over the world? (Ctd.)

Colossus: The Forbin Project (1970); based on the 1966 science fiction novel "Colossus" by Dennis Feltham Jones

Take over the world? (Ctd.)

Further recommended movies about AI systems resulting in undesired outcomes:

- Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964; Stanley Kubrick);
- 2001: A Space Odyssey (1968; Stanley Kubrick).

Postscriptum: Leibniz's Dream

"Indignum enim est excellentium virorum horas servii calculandi labore perire, qui Machina adhibita vilissimo cuique secure transcribi posset."

"Eines geistig hochstehenden Mannes ist es unwürdig, seine Zeit mit sklavischer Rechenarbeit zu vergeuden, denn mit einer Maschine könnte auch der Allerdümmste die Rechnung sicher ausführen."

"It is beneath the dignity of intellectually superior men to waste their time with slavish calculations when even the most witless of all could do the work just as accurately with the aid of a machine."