

Introduction to Artificial Intelligence

Fall 2018

Prof. Gilles Louppe
g.louppe@uliege.be



Logistics

This course is given by:

- Theory: Prof. Gilles Louppe (g.louppe@uliege.be)
- Exercises: Antoine Wehenkel (antoine.wehenkel@uliege.be)
- Projects: Samy Aittahar (saittahar@uliege.be)

Feel free to contact any of us for help!



Lectures

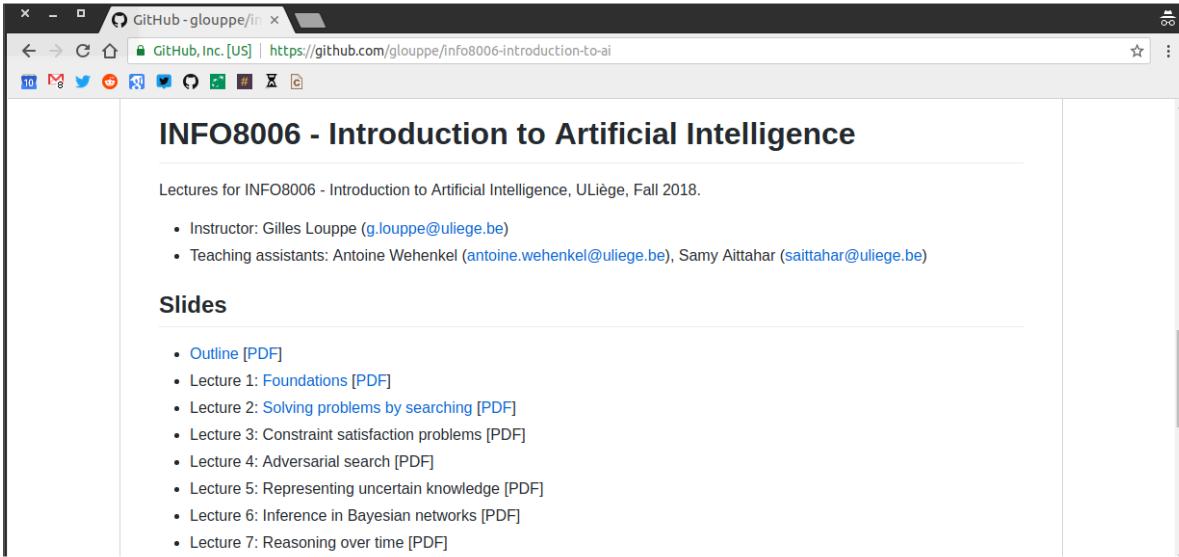
- Theoretical lectures
- Exercise sessions

Materials

Slides are available at github.com/glouuppe/info8006-introduction-to-ai.

- In HTML and in PDFs.
- Posted online the day before the lesson (hopefully).
- Slightly different from previous years.

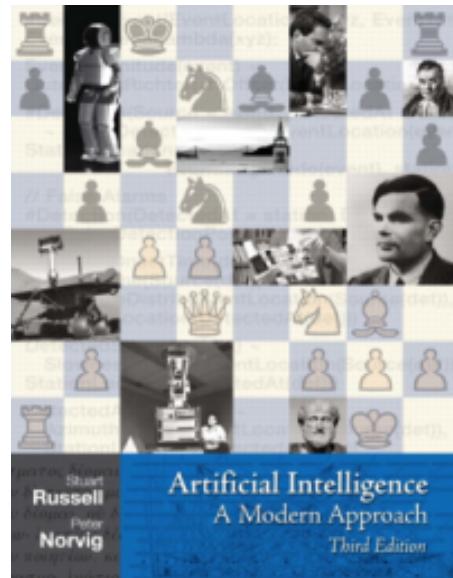
Some lessons are partially adapted from "[Introduction to Artificial Intelligence](#)" (CS188), from UC Berkeley.



The screenshot shows a web browser window with the GitHub URL <https://github.com/glouuppe/info8006-introduction-to-ai>. The page title is "INFO8006 - Introduction to Artificial Intelligence". Below the title, it says "Lectures for INFO8006 - Introduction to Artificial Intelligence, ULiège, Fall 2018." A list of instructors and teaching assistants is provided. Under the heading "Slides", there is a list of lecture topics with links to their corresponding PDF files:

- Outline [PDF]
- Lecture 1: Foundations [PDF]
- Lecture 2: Solving problems by searching [PDF]
- Lecture 3: Constraint satisfaction problems [PDF]
- Lecture 4: Adversarial search [PDF]
- Lecture 5: Representing uncertain knowledge [PDF]
- Lecture 6: Inference in Bayesian networks [PDF]
- Lecture 7: Reasoning over time [PDF]

Textbook



The core content of this course is based on the following textbook:

Stuart Russel, Peter Norvig. "Artificial Intelligence: A Modern Approach", Third Edition, Global Edition.

This textbook is **strongly recommended**, although not required.

Philosophy

Thorough and detailed

- Understand the landscape of artificial intelligence.
- Be able to write from scratch, debug and run (some) AI algorithms.

Well established algorithms and state-of-the-art

- Well-established algorithms for building intelligent agents.
- Introduction to materials new from research (\leq 5 years old).
- Understand some of the open questions and challenges in the field.

Practical

- Fun and challenging course project.

Outline

- Lecture 1: Foundations
- Lecture 2: Solving problems by searching
- Lecture 3: Constraint satisfaction problems
- Lecture 4: Adversarial search
- Lecture 5: Representing uncertain knowledge
- Lecture 6: Inference in Bayesian networks
- Lecture 7: Reasoning over time
- Lecture 8: Making decisions
- Lecture 9: Learning
- Lecture 10: Communication
- Lecture 11: Artificial General Intelligence and beyond

Projects

Reading assignment

Read, summarize and criticize a major scientific paper in Artificial Intelligence.
(Paper to be announced later.)

ARTICLE

Mastering the game of Go with deep neural networks and tree search

David Silver¹, Aja Huang¹, Chris J. Maddison¹, Arthur Guez², Laurent Sifre³, George de Jong on the Driessche⁴, Julian Schrittwieser⁵, Ioannis Antonoglou⁶, Veda Pannhereslavym⁷, Marc Lanctot⁸, Sander Dieleman⁹, Dominikin Grewe¹⁰, John Nham², Nal Kalchbrenner¹¹, Ilya Sutskever², Timothy Lillicrap¹, Madeleine Leach¹, Koray Kavukcuoglu¹, Thore Graepel¹ & Demis Hassabis¹

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to solving Go based on “value function” learning and “policy” networks. The core idea is that deep neural networks are trained via a two-level learning of expert knowledge, learning to play Go at the level of state learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a novel way of training neural networks with variable policy networks. Using this search algorithm, our program AlphaGo achieved a 98.8% winning rate against Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(\cdot)$, which determines the outcome of the game, from every board position or state x , under perfect play by all players. These may be solved using dynamic programming, which finds the value functions v^* by iteratively containing approximately b possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 30$), $d \approx 80$) and checkers ($b \approx 40$), $d \approx 50$), exhaustive search over the entire search space can be reduced by general principles. First, the depth of the search may be reduced by position evaluations, truncating the search tree at state x and replacing the subsequent b^d possible sequences of moves with a single estimate of the outcome from state x . This approach has led to superhuman performance in chess¹, checkers² and, although it was believed to be intractable in general, Go³. Second, the search space may be further reduced by sampling moves from a policy $p(a|x)$ that is a probability distribution over possible moves a in position x . For example, Monte Carlo rollouts⁴ search to maximum depth without branching, instead of exploring the search space, by sampling moves according to a policy p . Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance backgammon⁵ and poker⁶. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance backgammon⁵ and poker⁶.

Scrabble¹, and weak amateur level play in poker.² Monte Carlo tree search (MCTS)^{1,2,3} uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant value becomes more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, the policy converges to the optimal policy and the evaluations converge to the optimal value function^{1,2}. The strength of Go programs has been built on MCTS, enhanced by policies that are trained to predict human expert moves³. These policies are used to narrow the search to a beam of high probability actions, and to sample actions during rollouts. This approach has achieved strong amateur play^{1,3}. However, prior work has been limited to shallow

policies¹³⁻¹⁵ or value functions¹⁶ based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains, for example, playing Go [1]. The layers of neurons, each arranged in overlapping tiles, are highly interconnected, abstractly located, and distributed representations of an image [2]. We employ a similar architecture for the game of Go. We pass in the board position as input, and output a 19 × 19 image where each board location contains the probability of a stone being placed there. We use these probabilities to calculate the effective depth and breadth of the search tree: evaluating positionality using a value network, and sampling actions using a policy network.

Supervised learning of policy network

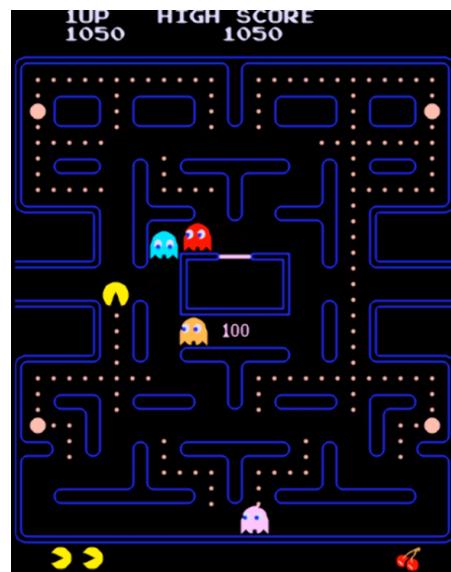
Supervised learning of policy networks
 For the first stage of the training pipeline, we build on prior work on predicting expert moves in the game of Go using supervised learning^{12,13,14}. The SL policy network $p_a(a|s)$ alternates between convolutional layers with weights θ , and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves a . The input to the policy network is a simple representation of the board state (see Extended Data Table 2). The policy network is trained on random

¹Google DeepMind, 5 New Street Square, London EC4A 3TW, UK. ²Google, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA.

Programming project

Implement an intelligent agent for playing **Pacman**. The project will be divided into three parts, with increasing levels of difficulty:

- Eat as many dots as possible
- Eat as many dots as possible, while not getting killed by ghosts (deterministic)
- Eat as many dots as possible, while not getting killed by ghosts (stochastic)



Evaluation

- Exam (60%)
- Reading assignment (10%)
- Programming project (30%)

Projects are **mandatory** for presenting the exam.

Going further

This course is designed as an introduction to the many other courses available at ULiège and related to AI, including:

- ELEN0062: Introduction to Machine Learning
- INFO8004: Advanced Machine Learning
- INFO8010: Deep Learning
- INFO8003: Optimal decision making for complex problems
- INFO0948: Introduction to Intelligent Robotics
- INFO0049: Knowledge representation
- ELEN0016: Computer vision
- DROI8031: Introduction to the law of robots

Introduction to Artificial Intelligence

Lecture 1: Foundations

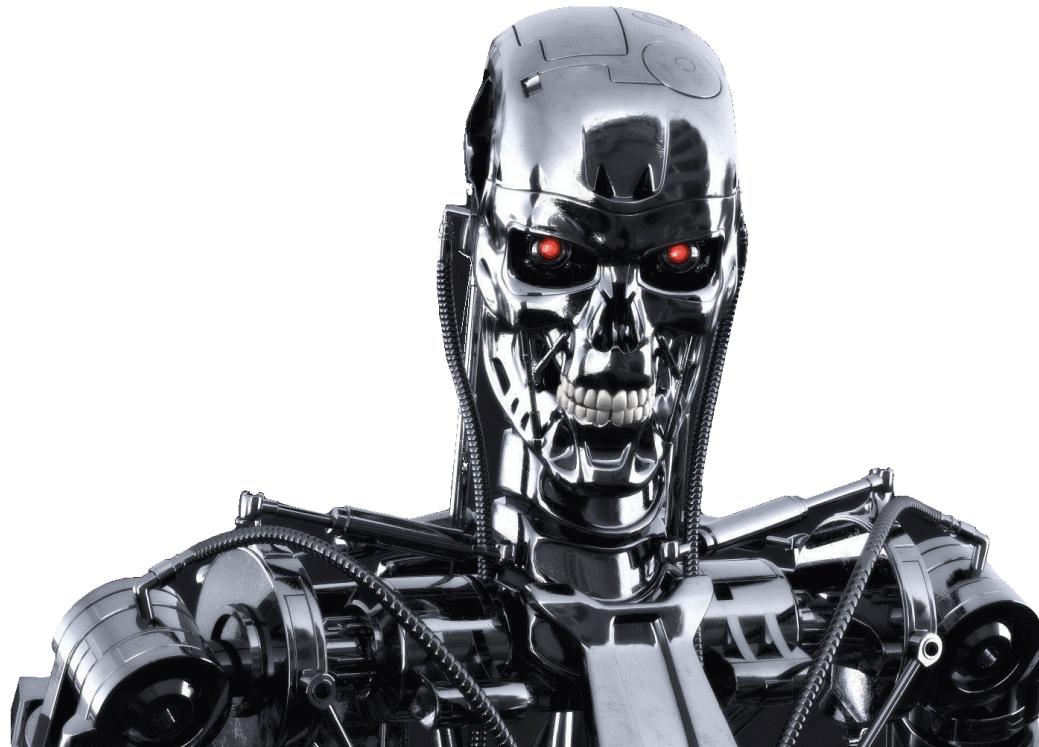
Prof. Gilles Louppe
g.louppe@uliege.be



Today

- Course outline
- Introduction to Artificial Intelligence
- Intelligent agents

Introduction



"With artificial intelligence we are summoning the demon" -- Elon Musk



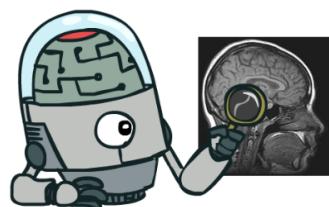
"We're really closer to a smart washing machine than Terminator" -- Fei-Fei Li,
Director of Stanford AI Lab.

What is AI?

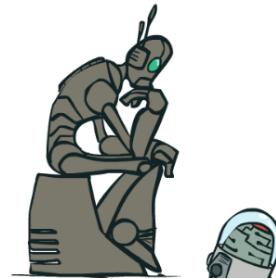
What is AI?

Artificial intelligence is the science of making machines or programs that:

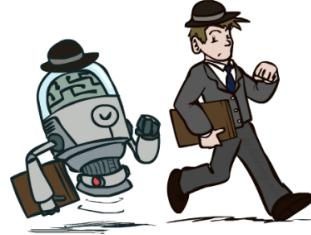
Think like people



Think rationally



Act like people



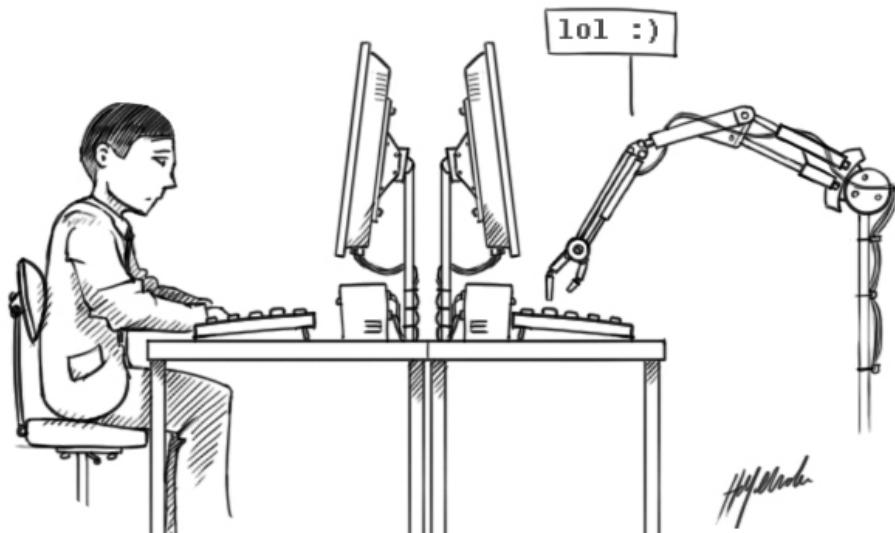
Act rationally



Acting humanly

The Turing test

A computer passes the **Turing test** (aka the Imitation Game) if a human operator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.



*Can machines think?
(Alan Turing, 1950)*

An agent would not pass the Turing test without the following **requirements**:

- natural language processing
- knowledge representation
- automated reasoning
- machine learning
- computer vision (total Turing test)
- robotics (total Turing test)

Despite being proposed almost 70 years ago, the Turing test is **still relevant** today.

Limitations of the Turing test

The Turing test tends to focus on **human-like errors, linguistic tricks**, etc.

However, it seems more important to study the **principles** underlying intelligence than to replicate an exemplar.



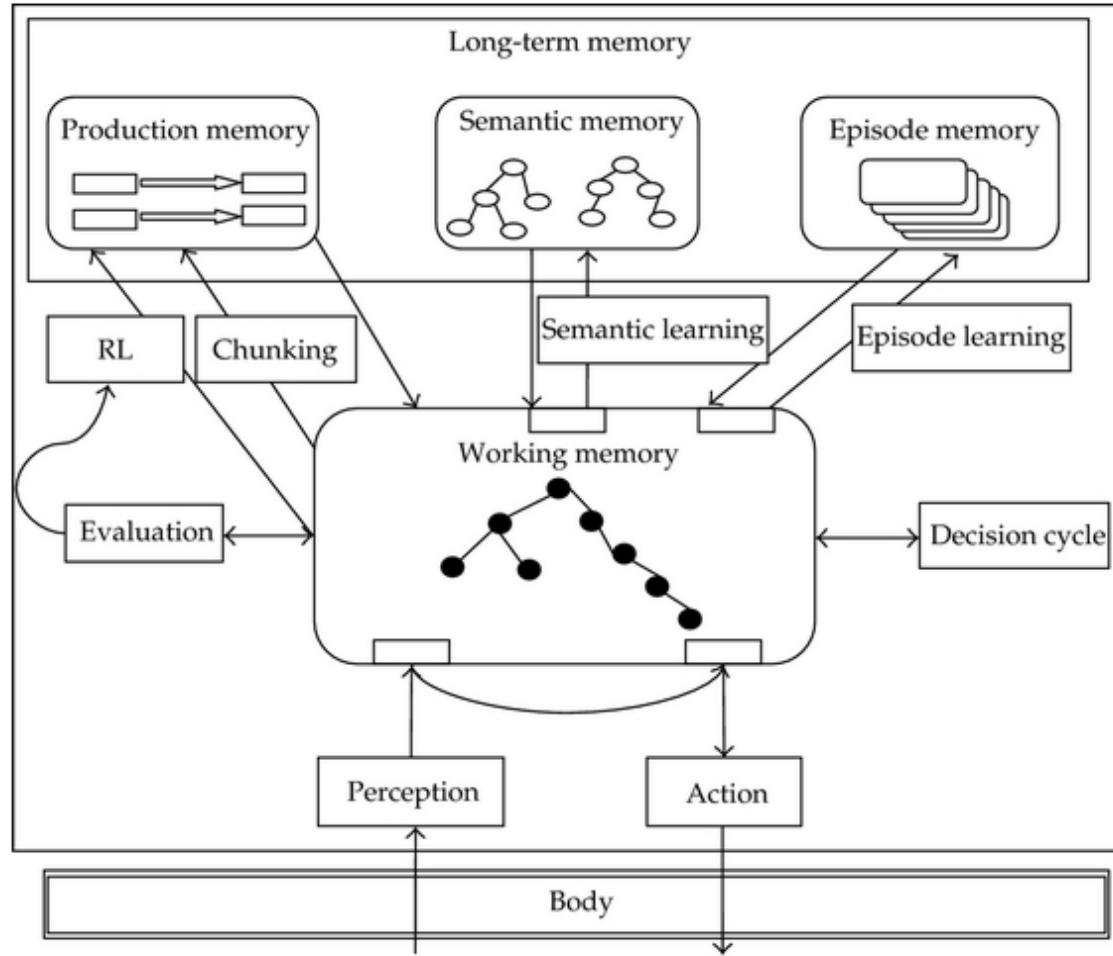
*Aeronautics is not defined as the field of making machines
that fly so exactly like pigeons that they can fool even other pigeons.*

Thinking humanly

Cognitive science

Study of the **human mind** and its processes.

- The goal of cognitive science is to form a theory about the structure of the mind, summarized as a comprehensive **computer model**.
- It includes language, problem-solving, decision-making and perception.
- A **cognitive architecture** usually follows human-like reasoning and can be used to produce testable predictions (time of delays during problem solving, kinds of mistakes, learning rates, etc).

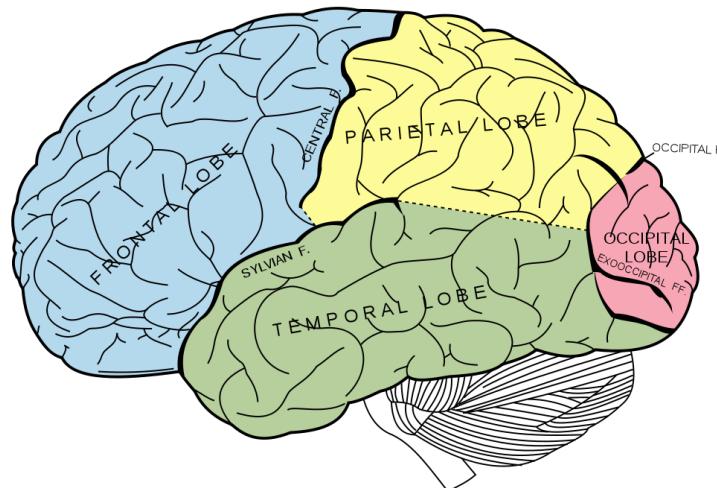


The modern SOAR cognitive architecture.

Neuroscience

Study of the anatomy and physiology of neural tissue.

- Neurobiology is concerned with the the anatomy and physiology of the brain, from major structures down to neurons and molecules.
- Neuroscience adds to that the study of **how the brain works**, mechanistically, functionally, and systematically to produce observable behavior.



Can we build a computer model of the brain?

Limitations of cognition and neuroscience for AI

- In linguistics, the argument of **poverty of the stimulus** states that children do not receive sufficient input to generalize grammatical rules through linguistic input alone.
 - A baby hears too few sentences to deduce the grammar of English before he speaks correctly.
- (Controversial) Therefore, humans must be **biologically pre-wired** with **innate knowledge** for representing language.



*How do we know what we know?
(Noam Chomsky, 1980)*

It may not be possible to implement a fully functioning computer model of the human mind without background knowledge of some sort. This is a huge technical **obstacle**, as accessing this knowledge would require reverse-engineering the brain.

Thinking rationally

The logical approach

- The rational thinking approach is concerned with the study of **irrefutable reasoning processes**. It ensures that all actions performed by a computer are formally **provable** from inputs and prior knowledge.
- The "laws of thought" were supposed to govern the operation of the mind. Their study initiated the field of **logic** and the **logician tradition** of AI (1960-1990).

The Zebra puzzle

- There are five houses.
- The English man lives in the red house.
- The Swede has a dog.
- The Dane drinks tea.
- The green house is immediately to the left of the white house.
- They drink coffee in the green house.
- The man who smokes Pall Mall has birds.
- In the yellow house they smoke Dunhill.
- In the middle house they drink milk.
- The Norwegian lives in the first house.
- The man who smokes Blend lives in the house next to the house with cats.
- In a house next to the house where they have a horse, they smoke Dunhill.
- The man who smokes Blue Master drinks beer.
- The German smokes Prince.
- The Norwegian lives next to the blue house.
- They drink water in a house next to the house where they smoke Blend.

Who owns the zebra?

```

select([A|As],S) :- select(A,S,S1), select(As,S1).
select([],_).

next_to(A,B,C) :- left_of(A,B,C) ; left_of(B,A,C).
left_of(A,B,C) :- append(_, [A,B|_], C).

zebra(Owns, HS) :- % color,nation,pet,drink,smokes
    HS = [ h(_,norwegian,_,_,_), _, h(_,_,_,milk,_), _, _ ],
    select( [ h(red,englishman,_,_,_), h(_,swede,dog,_,_), 
              h(_,dane,_,tea,_), h(_,german,_,_,prince) ], HS ),
    select( [ h(_,_,birds,_,pallmall), h(yellow,_,_,_,dunhill),
              h(_____,beer,bluemaster) ], HS ),
    left_of( h(green,_,_,coffee,_), h(white,_,_,_,_) ), HS,
    next_to( h(_____,_,_,dunhill), h(_,_,horse,_,_) ), HS,
    next_to( h(_____,_,_,blend), h(_,_,cats,_,_) ), HS,
    next_to( h(_____,_,_,blend), h(_,_,_,water,_), HS ),
    next_to( h(_,norwegian,_,_,_), h(blue,_,_,_,_) ), HS,
    member( h(_,Owns,zebra,_,_), HS ).

:- ?- time(( zebra(Who, HS), maplist(writeln,HS), nl, write(Who), nl, nl,
            ; write('No more solutions.')) )).

```

Output =

```
h(yellow,norwegian, cats, water, dunhill)
h(blue, dane,      horse, tea,   blend)
h(red,  englishman,birds, milk,  pallmall)
h(green, german,    zebra, coffee,prince)
h(white, swede,     dog,   beer, bluemaster)
```

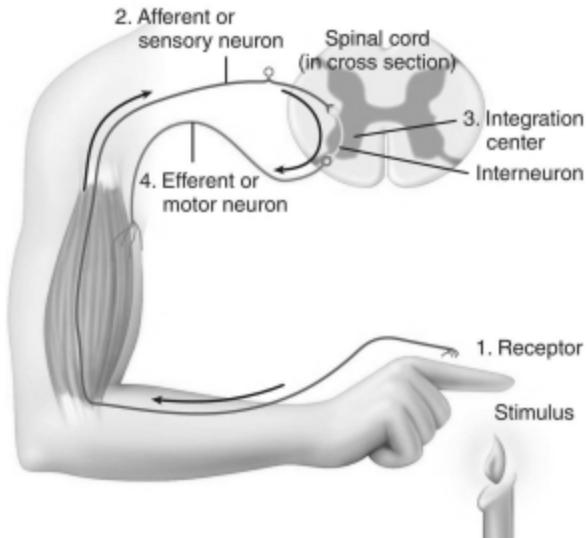
german

No more solutions.

% 5,959 inferences, 0.000 CPU in 0.060 seconds (0% CPU, Infinite Lips)

Limitations of logical inference

- Representation of **informal** knowledge is difficult.
- Hard to define provable **plausible** reasoning.
- **Combinatorial explosion** (in time and space).
- Logical inference is only a part of intelligence. It does not cover everything:
 - e.g., might be no provably correct thing to do, but still something must be done;
 - e.g., reflex actions can be more successful than slower carefully deliberated ones.



Pain withdrawal reflexes do not involve inference.

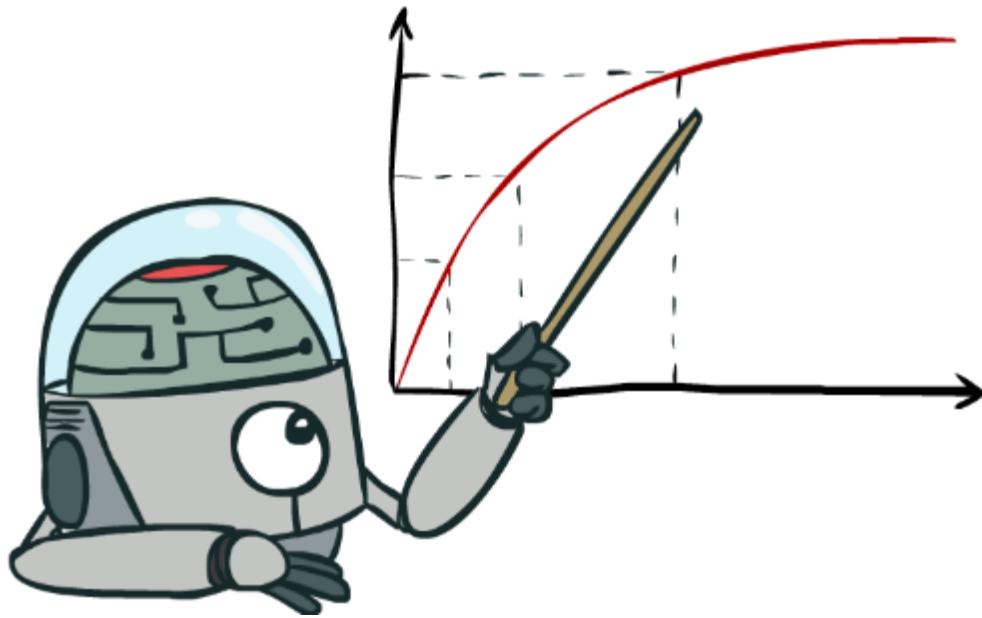
Acting rationally

A **rational agent** acts so as to achieve the best (expected) outcome.

- Correct logical inference is just one of several possible mechanisms for achieving this goal.
- Perfect rationality cannot be achieved due to computational limitations! The amount of reasoning is adjusted according to available resources and importance of the result.
- The brain is good at making rational decisions but not perfect either.

Rationality only concerns **what** decisions are made (not the thought process behind them, human-like or not).

Goals are expressed in terms of the **performance** or **utility** of outcomes. Being rational means maximizing its expected performance. The standard of rationality is general and mathematically well defined.



In this course, Artificial intelligence = **Maximizing expected performance**

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, psychophysics.
- **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.

A short history of AI

1940-1950: Early days

- 1943: McCulloch and Pitts: Boolean circuit model of the brain.
- 1950: Turing's "Computing machinery and intelligence".

1950-1970: Excitement and expectations

- 1950s: Early AI programs, including Samuel's checkers program, Newell and Simon's Logic Theorist and Gelernter's Geometry Engine.
- 1956: Dartmouth meeting: "Artificial Intelligence" adopted.
- 1958: Rosenblatt invents the perceptron.
- 1965: Robinson's complete algorithm for logical reasoning.
- 1966-1974: AI discovers computational complexity.

1956 Dartmouth Conference: The Founding Fathers of AI



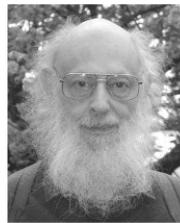
John McCarthy



Marvin Minsky



Claude Shannon



Ray Solomonoff



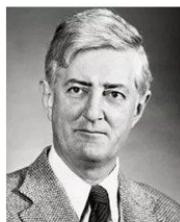
Alan Newell



Herbert Simon



Arthur Samuel



Oliver Selfridge



Nathaniel Rochester



Trenchard More



1970-1990: Knowledge-based approaches

- 1969: Neural network research almost disappears after Minsky and Papert's book.
- 1969-1979: Early development of knowledge-based systems.
- 1980-1988: Expert systems industrial boom.
- 1988-1993: Expert systems industry busts (AI winter).

1990-Present: Statistical approaches

- 1985-1995: The return of neural networks.
- 1988-: Resurgence of probability, focus on uncertainty, general increase in technical depth.
- 1995-2010: New fade of neural networks.
- 1995-: Complete intelligent agents and learning systems.
- 2000-: Availability of very large datasets.
- 2010-: Availability of fast commodity hardware (GPUs).
- 2012-: Resurgence of neural networks with deep learning approaches.

What can AI do at present?

- Translate spoken Chinese to spoken English, live?
- Answer multi choice questions, as good as an 8th grader?
- Converse with a person for an hour?
- Play decently at Chess? Go? Poker? Soccer?
- Buy groceries on the web? in a supermarket?
- Prove mathematical theorems?
- Drive a car safely on a parking lot? in New York?
- Perform a surgery?
- Identify skin cancer better than a dermatologist?
- Write a funny story?
- Paint like Van Gogh? Compose music?
- Show common sense?



Playing Atari games

n



Beat the best human Go players (2016)



Beat teams of human players at real-time strategy games (Dota 2) (2018)



Speech translation and synthesis (2012)



Semantic segmentation (2017)



Generating image descriptions (2015)

n



Detecting skin cancer (2017)



Learning to walk (2017)

R



Folding laundry (2010)



Playing soccer (2018)



Driving a car (2016)



Learning to sort waste (Norman Marlier, ULiège, 2018)



Learning to sort waste (Norman Marlier, ULiège, 2018)

What is missing?

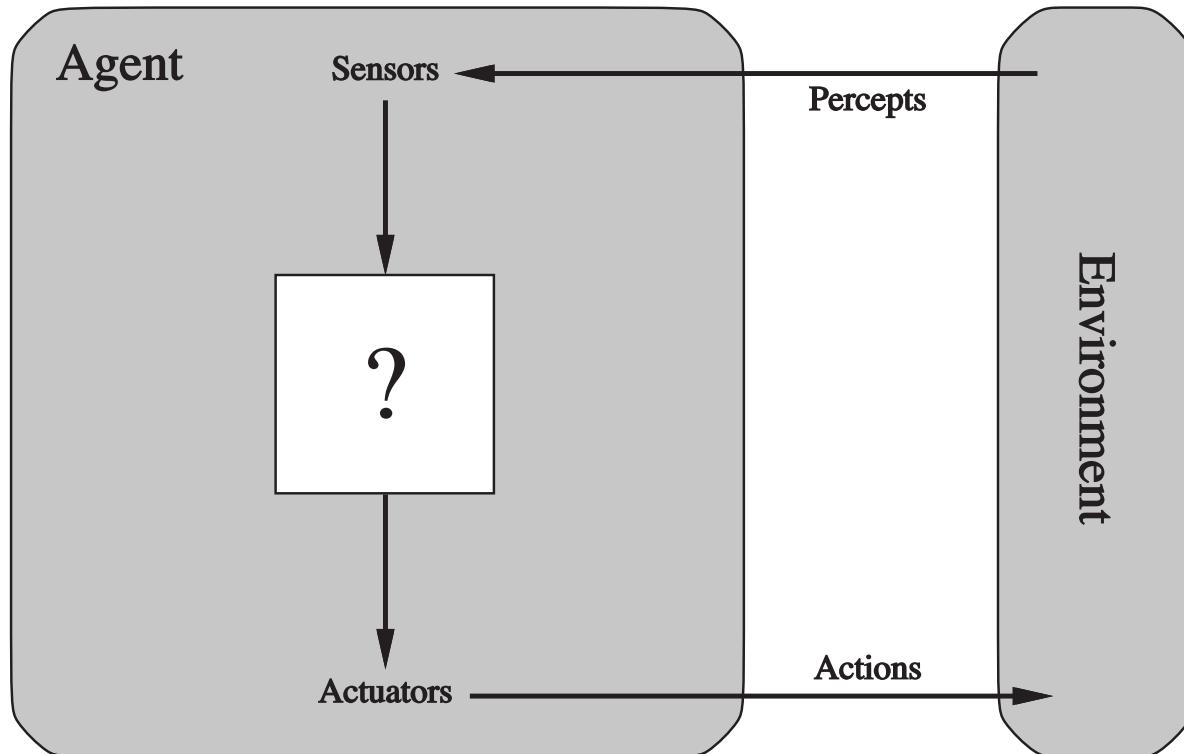
Intelligence is not just about **pattern recognition**, which is something most of these works are based on.

It is about **modeling the world**:

- explaining and understanding what we see;
- imagining things we could see but haven't yet;
- problem solving and planning actions to make these things real;
- building new models as we learn more about the world.

Intelligent agents

Agents and environments



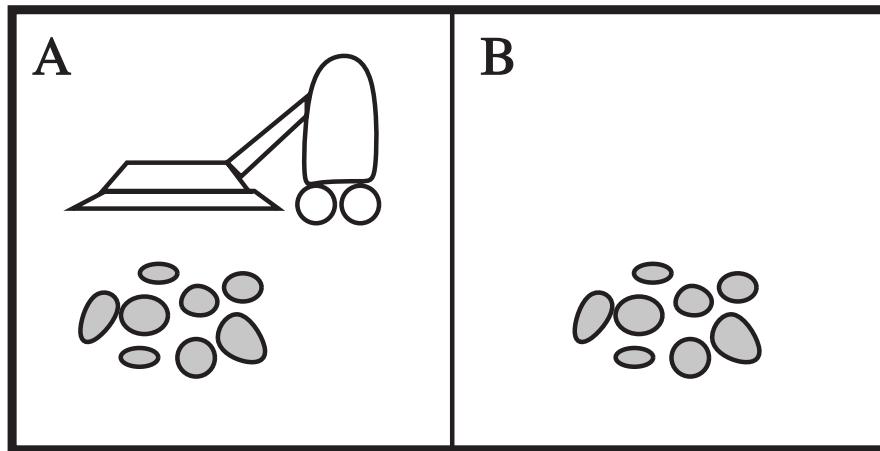
Agents

- An **agent** is an entity that **perceives** its environment through sensors and take **actions** through actuators.
- The agent behavior is described by the **agent function**, or **policy**, that maps percept histories to actions:

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

- The **agent program** runs on the physical architecture to produce f .

Vacuum-cleaner world



- Percepts: location and content, e.g. [A, Dirty]
- Actions: Left, Right, Suck, NoOp

A vacuum-cleaner agent

Partial tabulation of a simple vacuum-cleaner agent function:

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[A, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
(...)	(...)

The optimal vacuum-cleaner?

What is the **right** agent function?

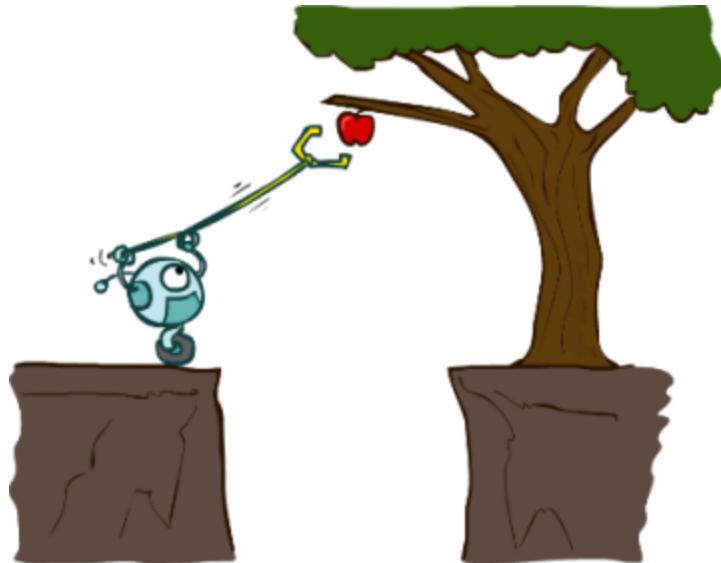
How to formulate the **goal** of the vacuum-cleaner agent?

- 1 point per square cleaned up at time t ?
- 1 point per clean square per time step, minus one per move?
- penalize for $> k$ dirty squares?

Can it be implemented in a **small** agent program?

Rational agents

- Informally, a **rational agent** is an agent that does the "right thing".
- A **performance measure** evaluates a sequence of environment states caused by the agent's behavior.
- A rational agent is an agent that chooses whichever action that **maximizes** the **expected** value of the performance measure, given the percept sequence to date.



- Rationality \neq omniscience
 - percepts may not supply all relevant information.
- Rationality \neq clairvoyance
 - action outcomes may not be as expected.
- Hence, rational \neq successful.
- However, rationality leads to exploration, learning and autonomy.

Performance, environment, actuators, sensors

The characteristics of the performance measure, environment, action space and percepts dictate techniques for selecting rational actions.

These characteristics are summarized as the **task environment**.

Example 1: an autonomous car

- **performance measure**: safety, destination, legality, comfort, ...
- **environment**: streets, highways, traffic, pedestrians, weather, ...
- **actuators**: steering, accelerator, brake, horn, speaker, display, ...
- **sensors**: video, accelerometers, gauges, engine sensors, GPS, ...

Example 2: an Internet shopping agent

- **performance measure**: price, quality, appropriateness, efficiency
- **environment**: current and future WWW sites, vendors, shippers
- **actuators**: display to user, follow URL, fill in form, ...
- **sensors**: web pages (text, graphics, scripts)

Environment types

- Fully observable vs. partially observable
 - Whether the agent sensors give access to the complete state of the environment, at each point in time.
- Deterministic vs. stochastic
 - Whether the next state of the environment is completely determined by the current state and the action executed by the agent.
- Episodic vs. sequential
 - Whether the agent's experience is divided into atomic independent episodes.
- Static vs. dynamic
 - Whether the environment can change, or the performance measure can change with time.
- Discrete vs. continuous
 - Whether the state of the environment, the time, the percepts or the actions are continuous.
- Single agent vs. multi-agent
 - Whether the environment include several agents that may interact with each other.
- Known vs unknown
 - Reflects the agent's state of knowledge of the "law of physics" of the environment.

Are the following task environments fully observable? deterministic? episodic? static? discrete? single agents? Known?

- Crossword puzzle
- Chess, with a clock
- Poker
- Backgammon
- Taxi driving
- Medical diagnosis
- Image analysis
- Part-picking robot
- Refinery controller
- The real world

Agent programs

The job of AI is to design an **agent program** that implements the agent function. This program will run on an **architecture**, that is a computing device with physical sensors and actuators.

$$\text{agent} = \text{program} + \text{architecture}$$

Implementation

Agent programs can be designed and implemented in many ways:

- with tables
- with rules
- with search algorithms
- with learning algorithms

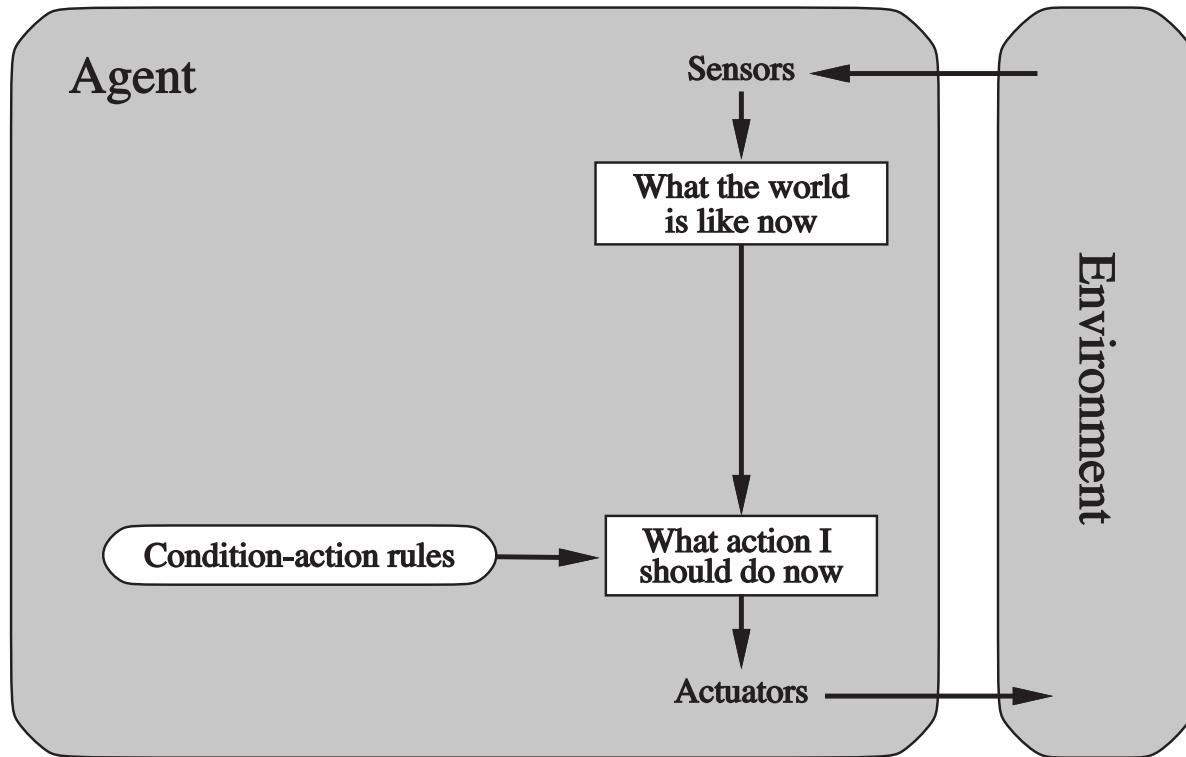
Table-driven agents

A **table-driven agent** determines its next action with a lookup table that contains the appropriate action for every possible percept sequence.

Issues

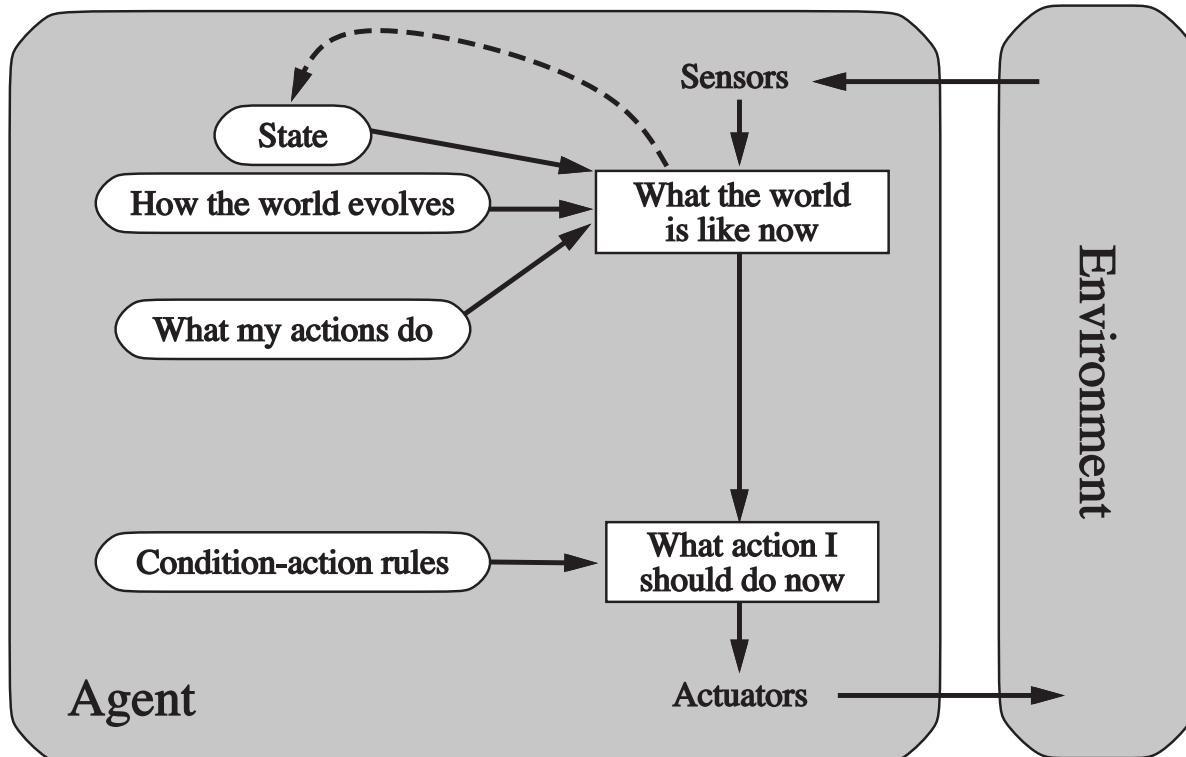
- **Design issue:** one needs to anticipate all sequence of percepts and how the agent should respond.
- **Technical issue:** the lookup table will contain $\sum_{t=1}^T |\mathcal{P}|^t$ entries.
- Example (autonomous car): using a 30fps 640x480 RBG camera as sensor, this results in a table with over $10^{250000000000}$ entries for an hour of driving.

Simple reflex agents



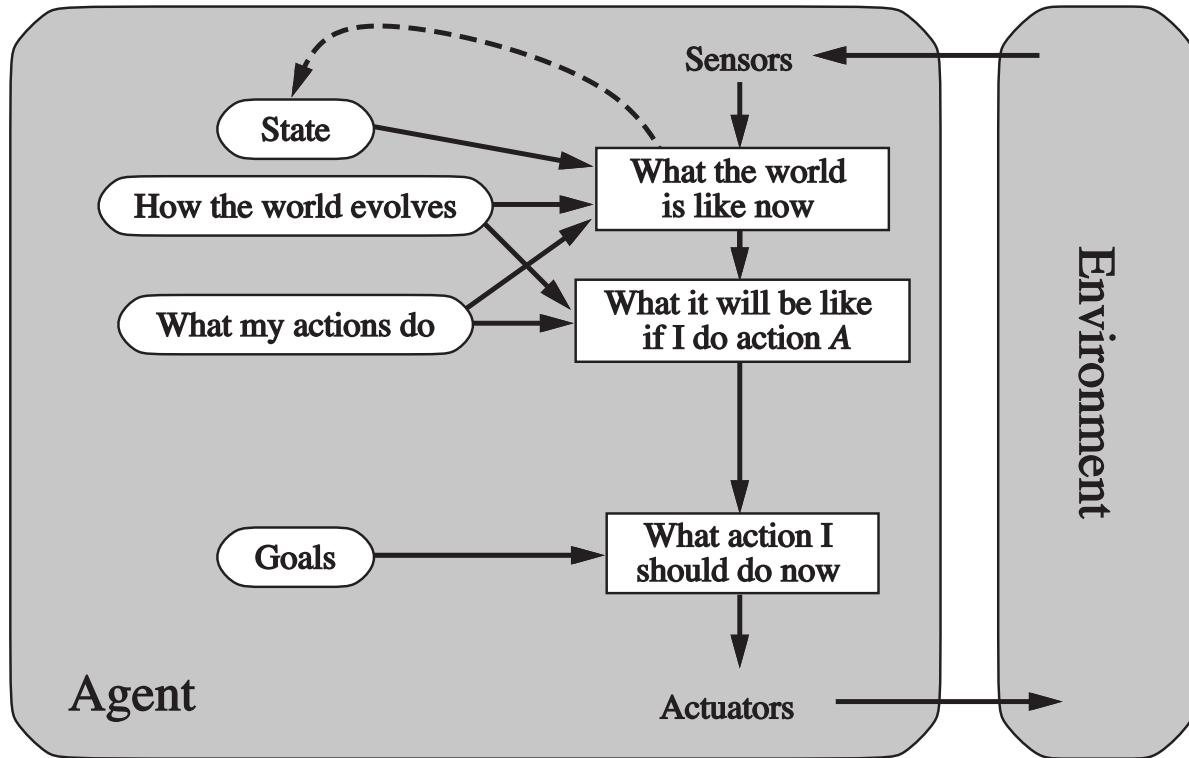
- Simple reflex agents select actions on the basis of the current percept, ignoring the rest of the percept history.
- They implement condition-action rules that match the current percept to an action.
- Rules provide a way to compress the function table.
 - Example (autonomous car): If a car in front of you slow down, you should break. The color and model of the car, the music on the radio or the weather are all irrelevant.
- Simple reflex agents are simple but they turn out to have limited intelligence.
- They can only work in a Markovian environment, that is if the correct decision can be made on the basis of only the current percept. In other words, if the environment is fully observable.

Model-based reflex agents



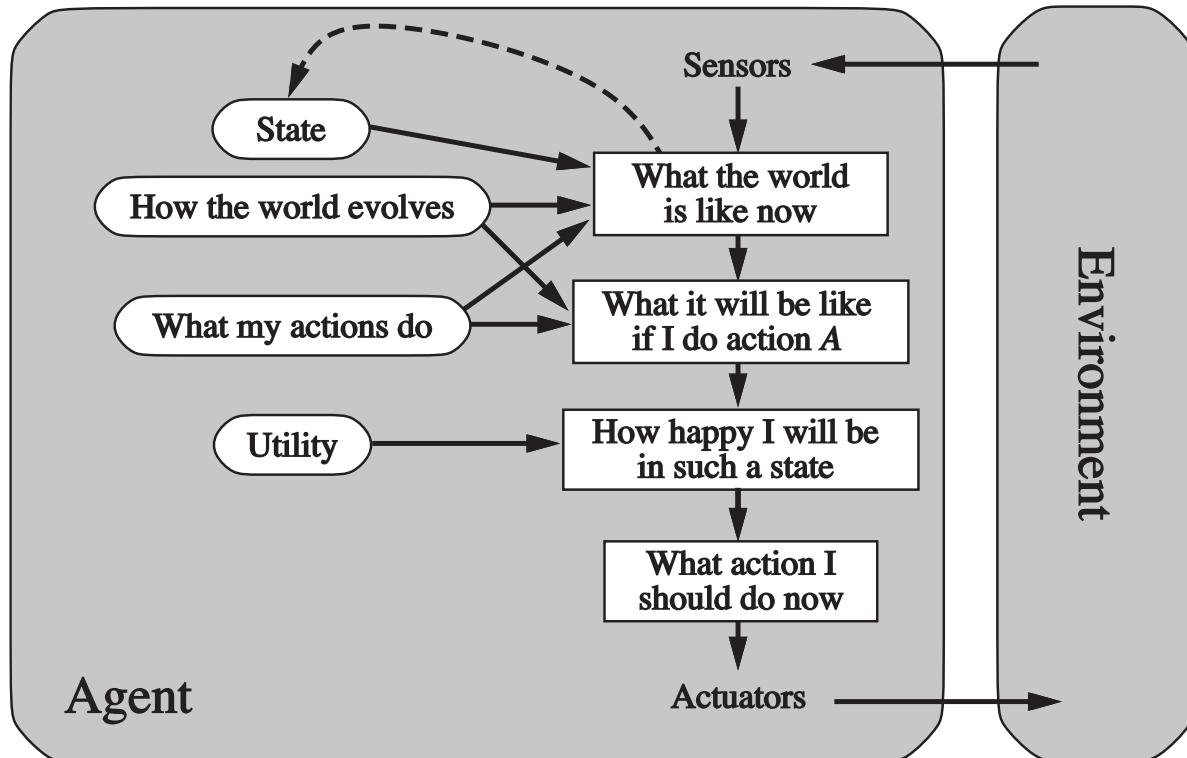
- Model-based agents handle partial observability of the environment by keeping track of the part of the world they cannot see now.
- The internal state of model-based agents is updated on the basis of a model which determines:
 - how the environment evolves independently of the agent;
 - how the agent actions affect the world.

Goal-based agents



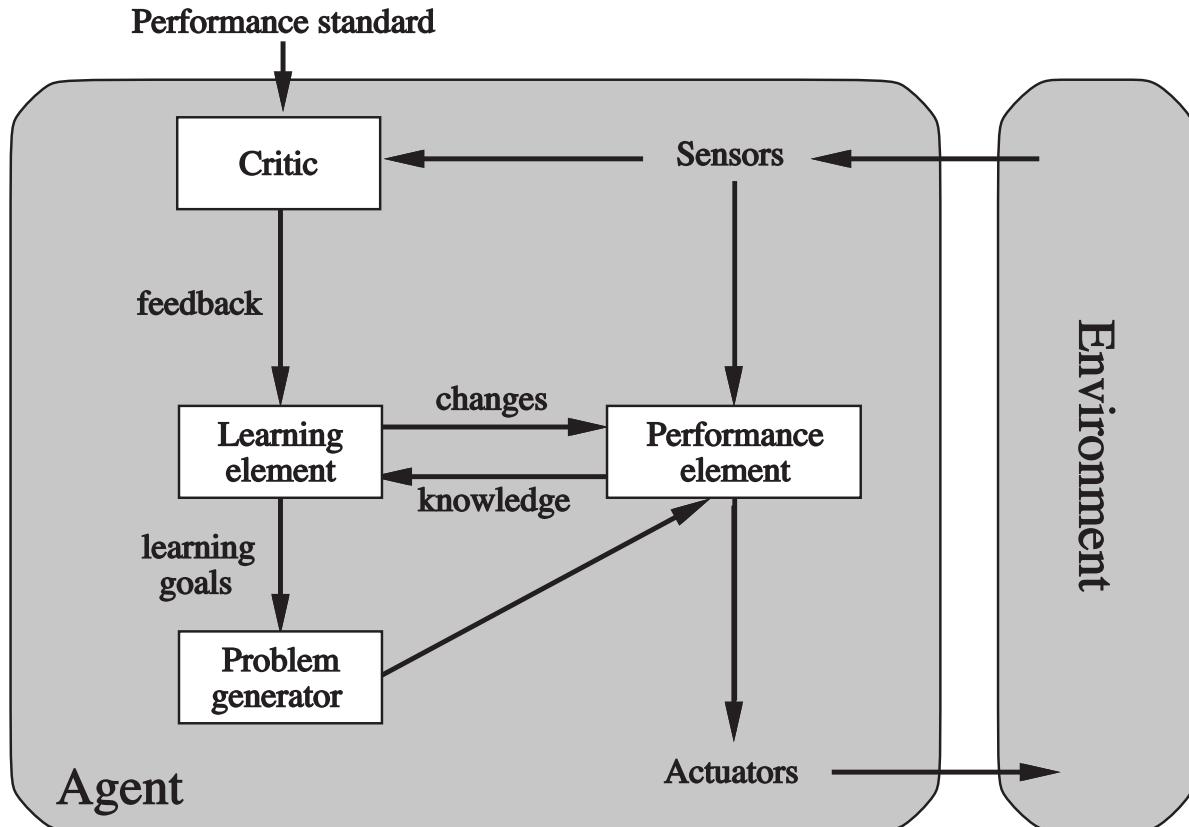
- Principle:
 1. generate possible sequences of actions
 2. predict the resulting states
 3. assess **goals** in each.
- A **goal-based agent** chooses an action that will achieve the goal.
 - More general than rules. Goals are rarely explicit in condition-action rules.
 - Finding action sequences that achieve goals is difficult. **Search** and **planning** are two strategies.
- Example (autonomous car): Has the car arrived to destination?

Utility-based agents



- Goals are often not enough to generate high-quality behavior.
 - Example (autonomous car): There are many ways to arrive to destination, but some are quicker or more reliable.
 - Goals only provide binary assessment of performance.
- A utility function scores any given sequence of environment states.
 - The utility function is an internalization of the performance measure.
- A rational utility-based agent chooses an action that maximizes the expected utility of its outcomes.

Learning agents



- Learning agents are capable of self-improvement. They can become more competent than their initial knowledge alone might allow.
- They can make changes to any of the knowledge components by:
 - learning how the world evolves;
 - learning what are the consequences of actions;
 - learning the utility of actions through rewards.

A learning autonomous car

- Performance element:
 - The current system for selecting actions and driving.
- The **critic** observes the world and passes information to the **learning element**.
 - E.g., the car makes a quick left turn across three lanes of traffic. The critic observes shocking language from the other drivers and informs bad action.
 - The learning element tries to modify the performance element to avoid reproducing this situation in the future.
- The **problem generator** identifies certain areas of behavior in need of improvement and suggest experiments.
 - E.g., trying out the brakes on different surfaces in different weather conditions.

Summary

- An **agent** is an entity that perceives and acts in an environment.
- The **performance measure** evaluates the agent's behavior. **Rational agents** act so as to maximize the expected value of the performance measure.
- **Task environments** includes performance measure, environment, actuators and sensors. They can vary along several significant dimensions.
- The **agent program** effectively implements the agent function. Their designs are dictated by the task environment.
- **Simple reflex agents** respond directly to percepts, whereas **model-based reflex agents** maintain internal state to track the world. **Goal-based agents** act to achieve goals while **utility-based agents** try to maximize their expected performance.
- All agents can improve their performance through **learning**.

References

- Turing, Alan M. "Computing machinery and intelligence." *Mind* 59.236 (1950): 433-460.
- Newell, Allen, and Herbert Simon. "The logic theory machine--A complex information processing system." *IRE Transactions on information theory* 2.3 (1956): 61-79.
- Chomsky, Noam. "Rules and representations." *Behavioral and brain sciences* 3.1 (1980): 1-15.

Introduction to Artificial Intelligence

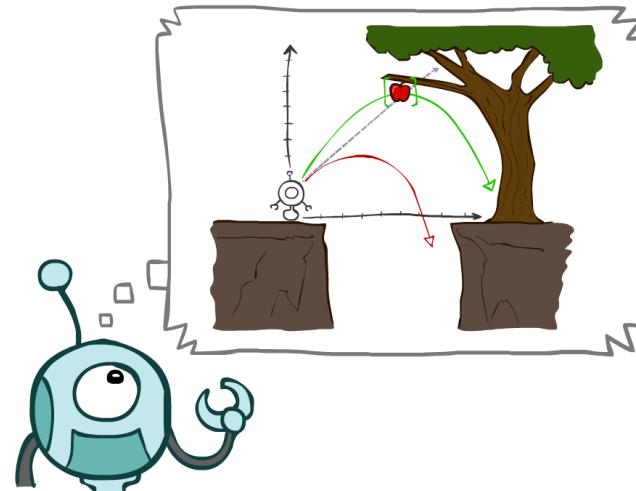
Lecture 2: Solving problems by searching

Prof. Gilles Louppe
g.louuppe@uliege.be



Today

- Planning agents
- Search problems
- Uninformed search methods
 - Depth-first search
 - Breadth-first search
 - Uniform-cost search
- Informed search methods
 - A*
 - Heuristics

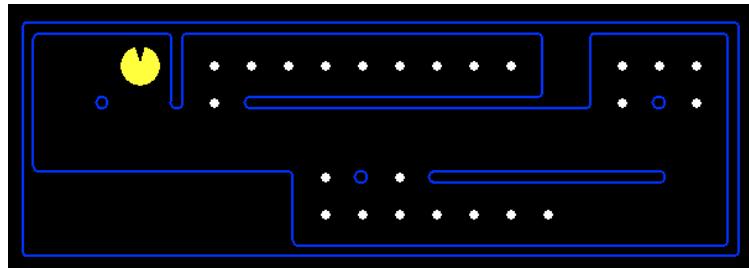
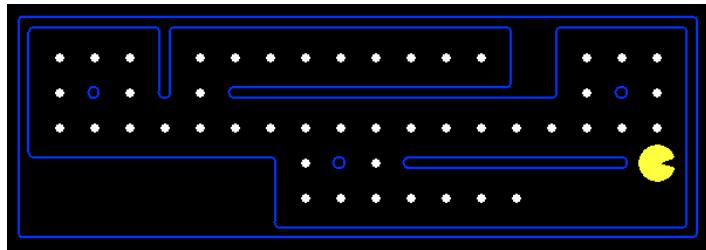


Planning agents

Reflex agents

Reflex agents

- select actions on the basis of the current percept;
- may have a model of the world current state;
- do not consider the future consequences of their actions;
- consider only **how the world is now**.



*For example, a simple reflex agent moves to a dot if there is one in its neighborhood.
No planning is involved to take this decision.*

[Q] Can a reflex agent be rational?

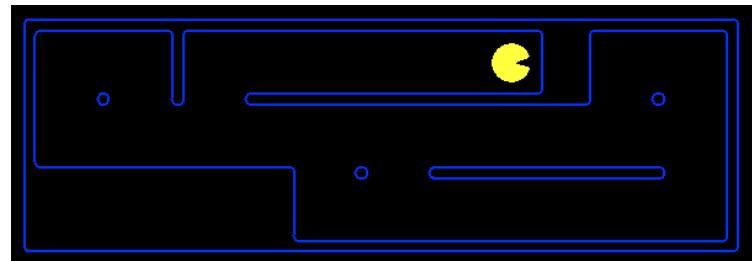
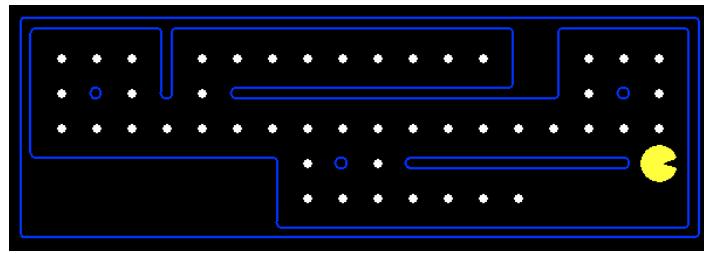
Problem-solving agents

Assumptions:

- Observable, deterministic (and known) environment.

Problem-solving agents

- take decisions based on (hypothesized) consequences of actions;
- must have a model of how the world evolves in response to actions;
- formulate a goal, explicitly;
- consider how the world would be.



A planning agent looks for sequences of actions to eat all the dots.

```

function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
    state, some description of the current world state
    goal, a goal, initially null
    problem, a problem formulation

  state  $\leftarrow$  UPDATE-STATE(state, percept)
  if seq is empty then
    goal  $\leftarrow$  FORMULATE-GOAL(state)
    problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
    seq  $\leftarrow$  SEARCH(problem)
    if seq = failure then return a null action
    action  $\leftarrow$  FIRST(seq)
    seq  $\leftarrow$  REST(seq)
  return action

```

Figure 3.1 A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

Offline vs. Online solving

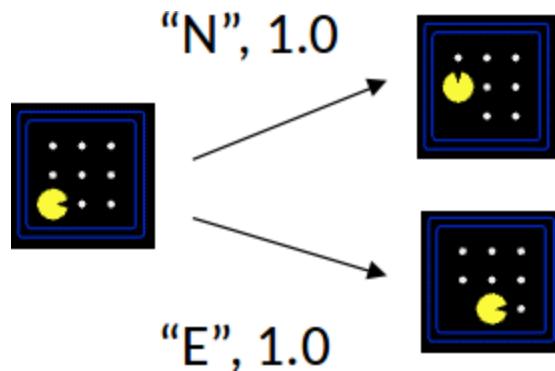
- Problem-solving agents are **offline**. The solution is executed "eyes closed", ignoring the percepts.
- **Online** problem solving involves acting without complete knowledge. In this case, the sequence of actions might be recomputed at each step.

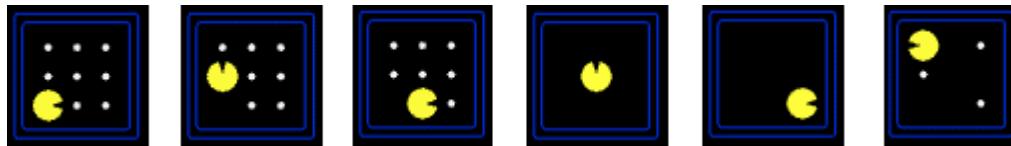
Search problems

Search problems

A **search problem** consists of the following components:

- The **initial state** of the agent.
- A description of the **actions** available to the agent given a state s , denoted $\text{actions}(s)$.
- A **transition model** that returns the state $s' = \text{result}(s, a)$ that results from doing action a in state s .
 - We say that s' is a **successor** of s if there is an acceptable action from s to s' .



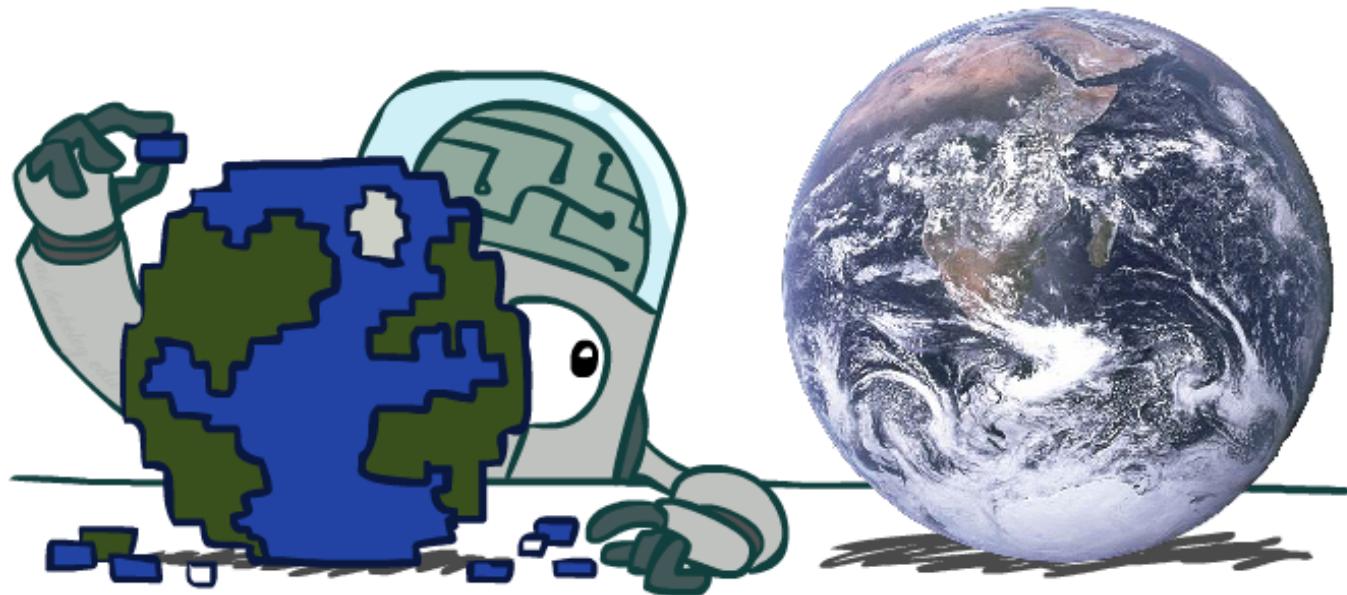


- Together, the initial state, the actions and the transition model define the **state space** of the problem, i.e. the set of all states reachable from the initial state by any sequence of action.
 - The state space forms a directed graph:
 - nodes = states
 - links = actions
 - A **path** is a sequence of states connected by actions.
- A **goal test** which determines whether the solution of the problem is achieved in state s .
- A **path cost** that assigns a numeric value to each path.
 - We may also assume that the path cost corresponds to a sum of positive **step costs** $c(s, a, s')$ associated to the action a in s leading to s' .

A **solution** to a problem is an action sequence that leads from the initial state to a goal state.

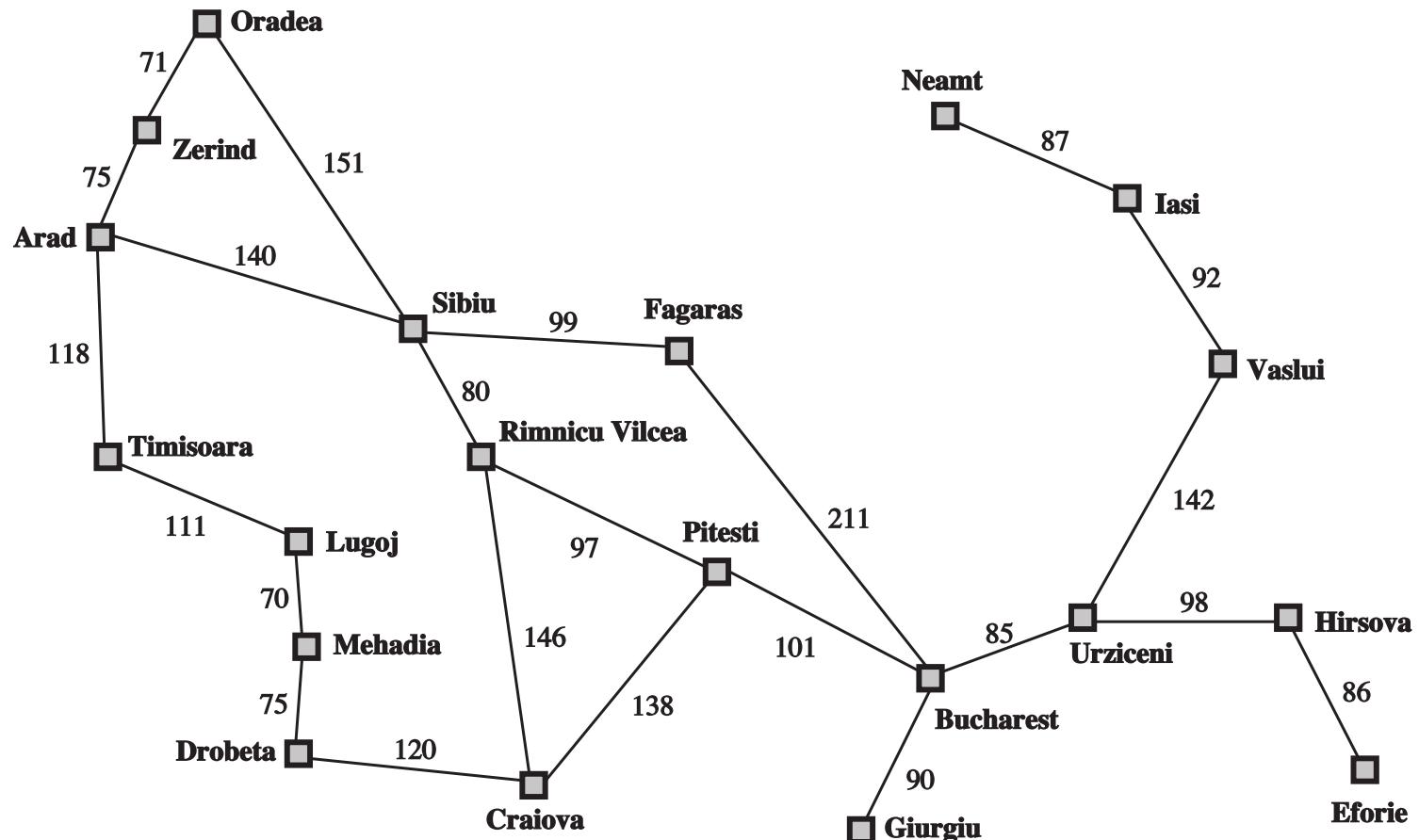
- A solution quality is measured by the path cost function.
- An **optimal solution** has the lowest path cost among all solutions.

[Q] What if the environment is partially observable? non-deterministic?



Search problems are **models**.

Example: Traveling in Romania

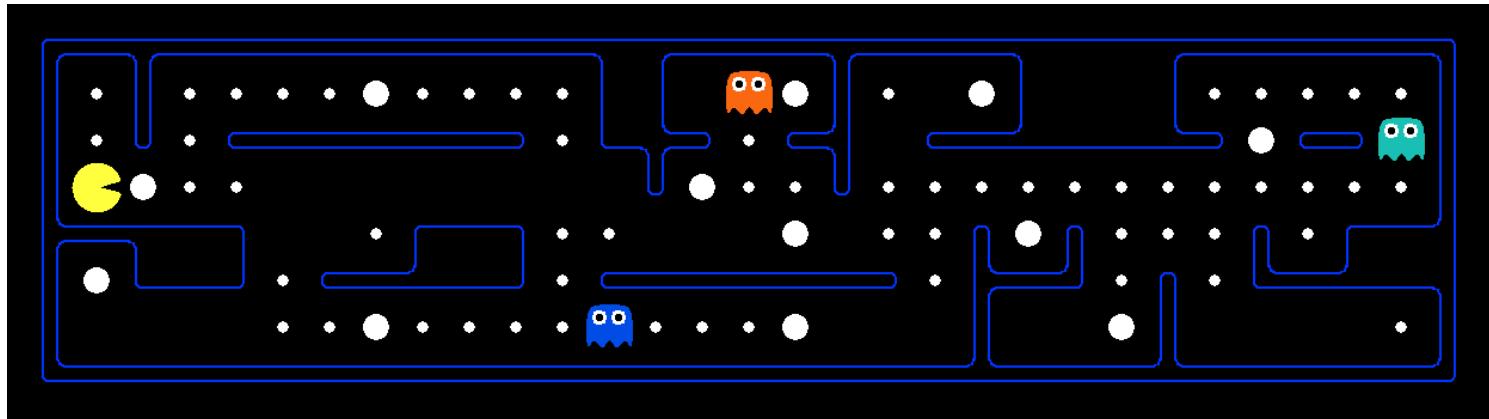


How to go from Arad to Bucharest?

- Initial state = the city we start in.
 - $s_0 = \text{in(Arad)}$
- Actions = Going from the current city to the cities that are directly connected to it.
 - $\text{actions}(s_0) = \{\text{go(Sibiu)}, \text{go(Timisoara)}, \text{go(Zerind)}\}$
- Transition model = The city we arrive in after driving to it.
 - $\text{result}(\text{in}(Arad), \text{go}(Zerind)) = \text{in}(Zerind)$
- Goal test: whether we are in Bucharest.
 - $s \in \{\text{in(Bucharest)}\}$
- Step cost: distances between cities.

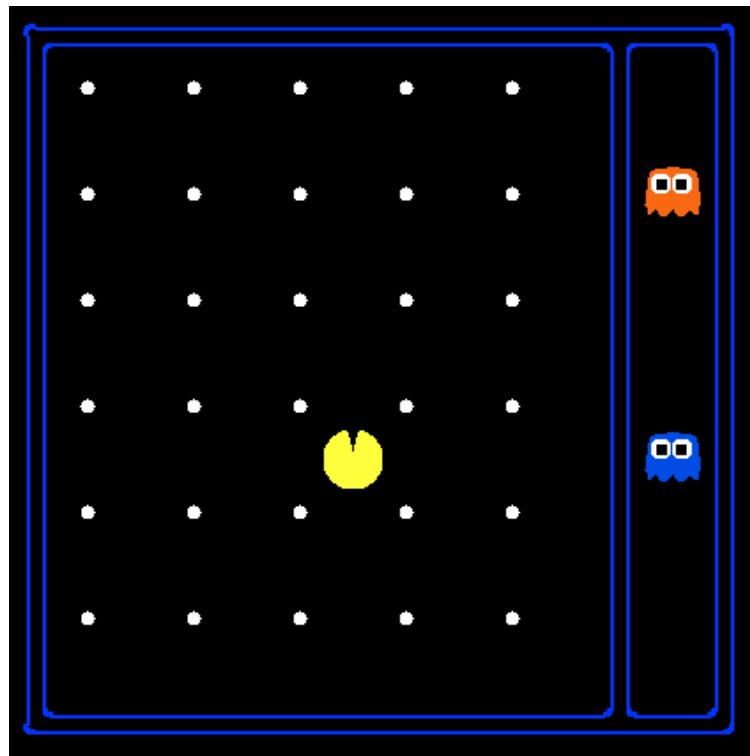
Selecting a state space

- Real world is absurdly **complex**.
 - The **world state** includes every last detail of the environment.
 - State space must be **abstracted** for problem solving.
- A **search state** keeps only the details needed for planning.
 - Example: eat-all-dots
 - States: $\{(x, y), \text{dot booleans}\}$
 - Actions: NSEW
 - Transition: update location and possibly a dot boolean
 - Goal test: dots all false



State space size

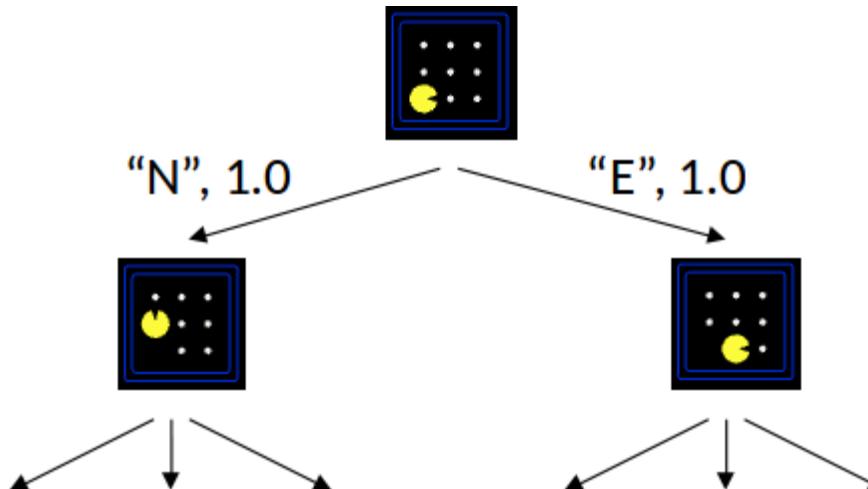
- World state:
 - Agent positions: 120
 - Found count: 30
 - Ghost positions: 12
 - Agent facing: NSEW
- How many?
 - World states?
 - $120 \times 2^{30} \times 12^2 \times 4$
 - States for eat-all-dots?
 - 120×2^{30}



Search trees

The set of possible acceptable sequences starting at the initial state form a **search tree**:

- Nodes correspond to states in the state space, where the initial state is the root node.
- Branches correspond to applicable actions.
 - Child nodes correspond to successors.
- For most problems, we can never actually build the whole tree.



Tree search algorithms

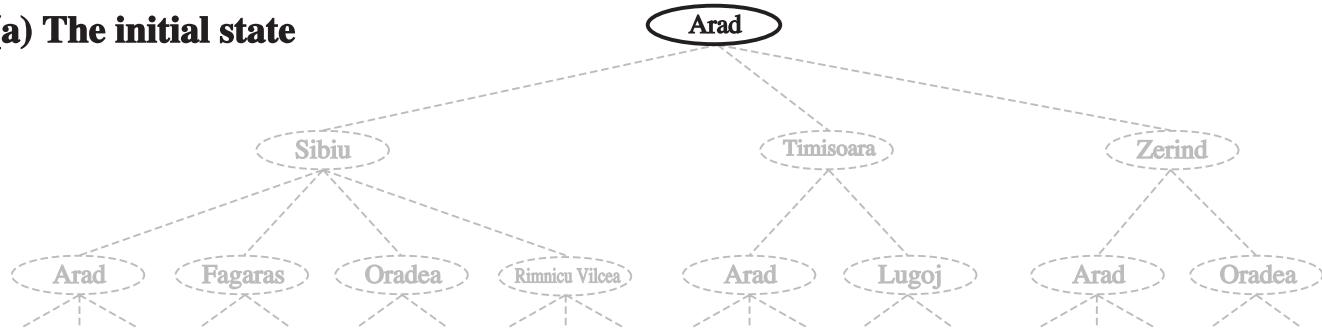
```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

Important ideas:

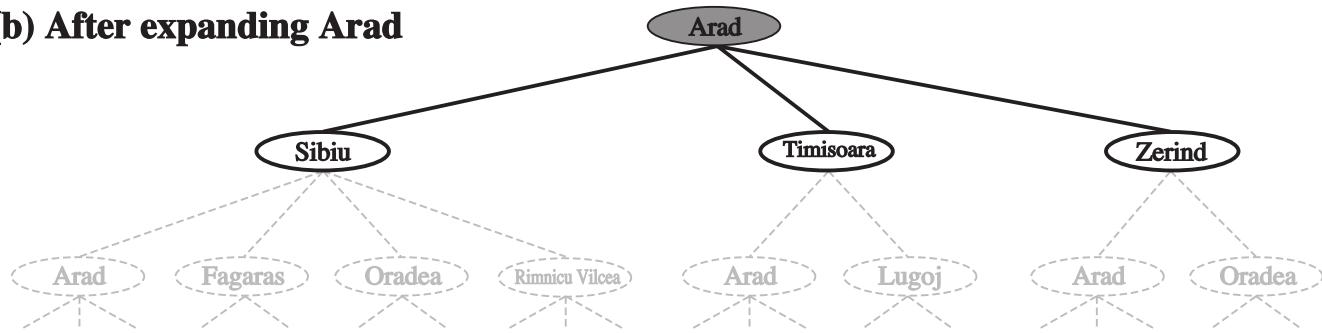
- Fringe (or frontier) of partial plans under consideration
- Expansion
- Exploration

[Q] Which fringe nodes to explore? How to expand as few nodes as possible, while achieving the goal?

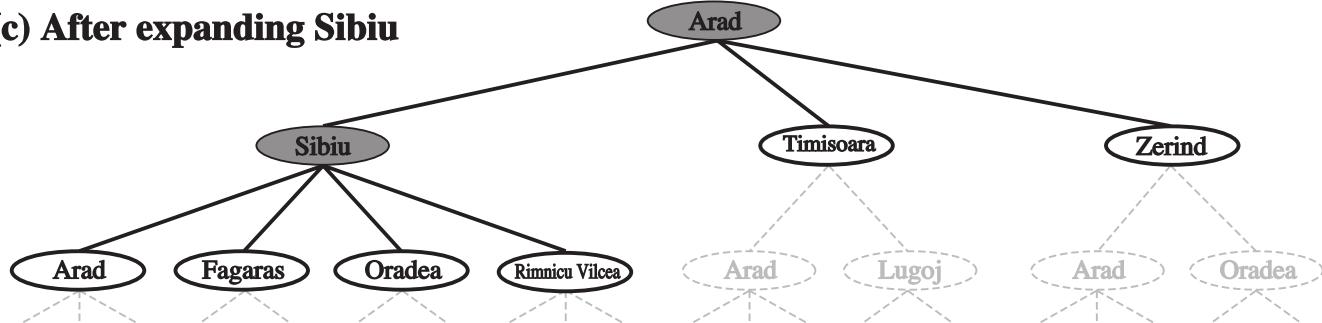
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu



Uninformed search strategies

Uninformed search strategies use only the information available in the problem definition.

- They do not know whether a state looks more promising than some other.

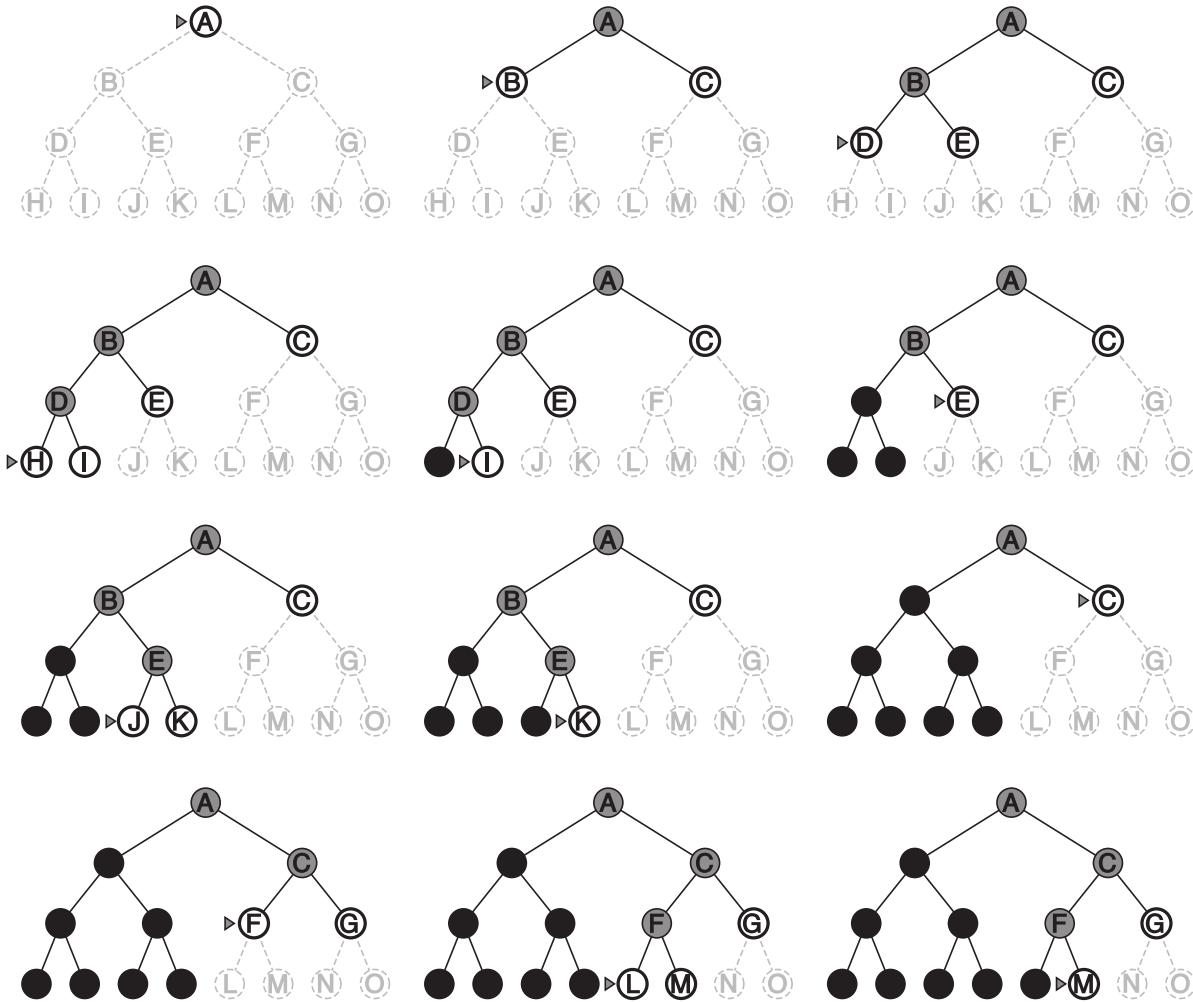
Strategies:

- Depth-first search
- Breadth-first search
- Uniform-cost search
- Iterative deepening

Depth-first search



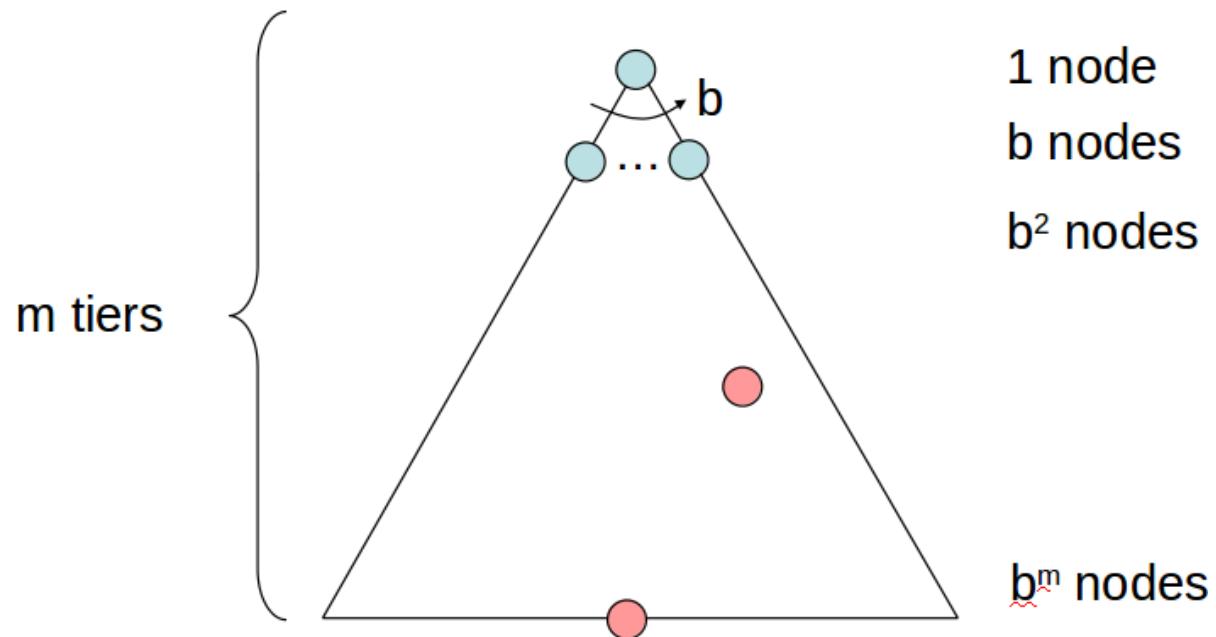
- **Strategy:** expand the deepest node in the fringe.
- **Implementation:** fringe is a **LIFO stack**.



Properties of search strategies

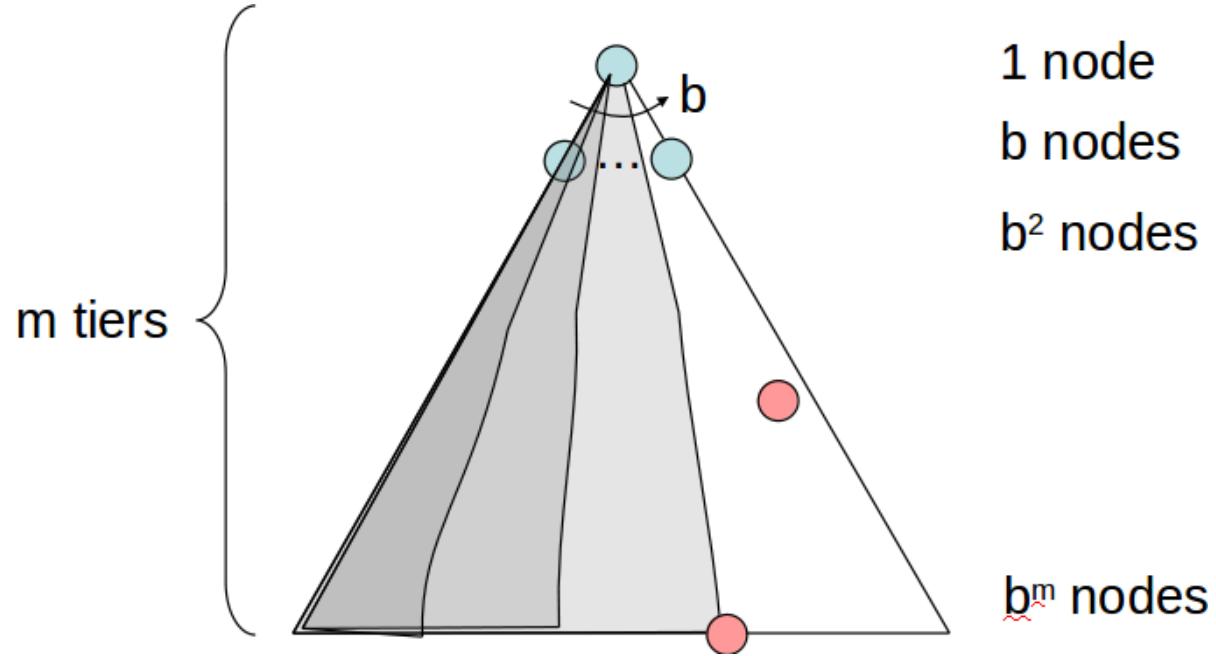
- A strategy is defined by picking the **order of expansion**.
- Strategies are evaluated along the following dimensions:
 - **Completeness**: does it always find a solution if one exists?
 - **Optimality**: does it always find the least-cost solution?
 - **Time complexity**: how long does it take to find a solution?
 - **Space complexity**: how much memory is needed to perform the search?
- Time and complexity are measured in terms of
 - **b** : maximum branching factor of the search tree
 - **d** : depth of the least-cost solution
 - the depth of s is defined as the number of actions from the initial state to s .
 - **m** : maximum length of any path in the state space (may be ∞)

[Q] Number of nodes in a tree?

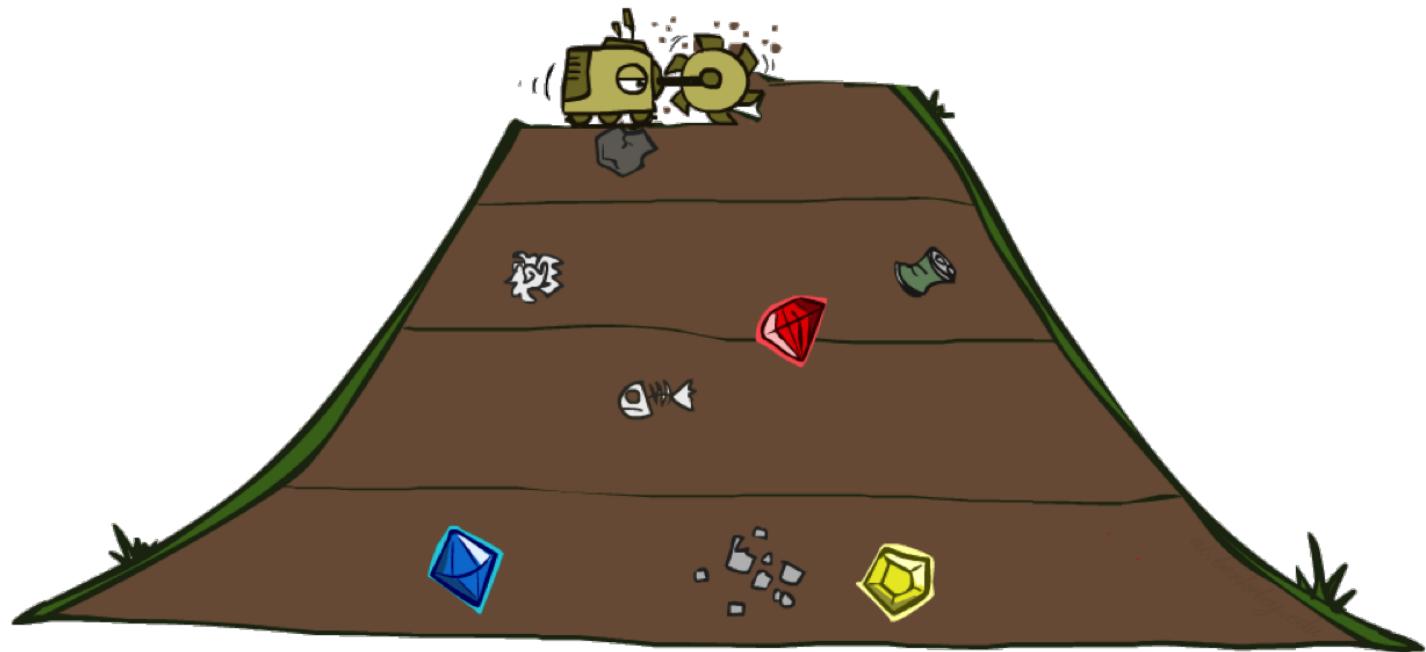


Properties of DFS

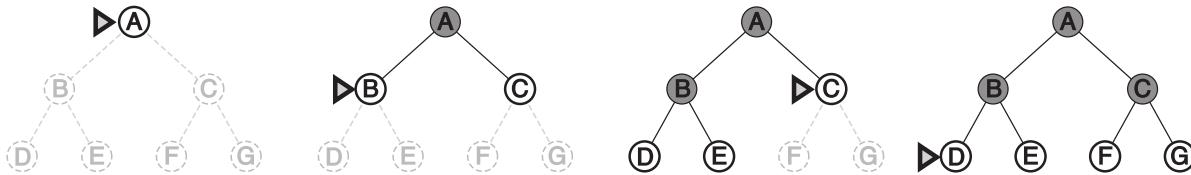
- **Completeness:**
 - m could be infinite, so only if we prevent cycles (more on this later).
- **Optimality:**
 - No, DFS finds the leftmost solution, regardless of depth or cost.
- **Time complexity:**
 - May generate the whole tree (or a good part of it, regardless of d). Therefore $O(b^m)$, which might much greater than the size of the state space!
- **Space complexity:**
 - Only store siblings on path to root, therefore $O(bm)$.
 - When all the descendants of a node have been visited, the node can be removed from memory.



Breadth-first search

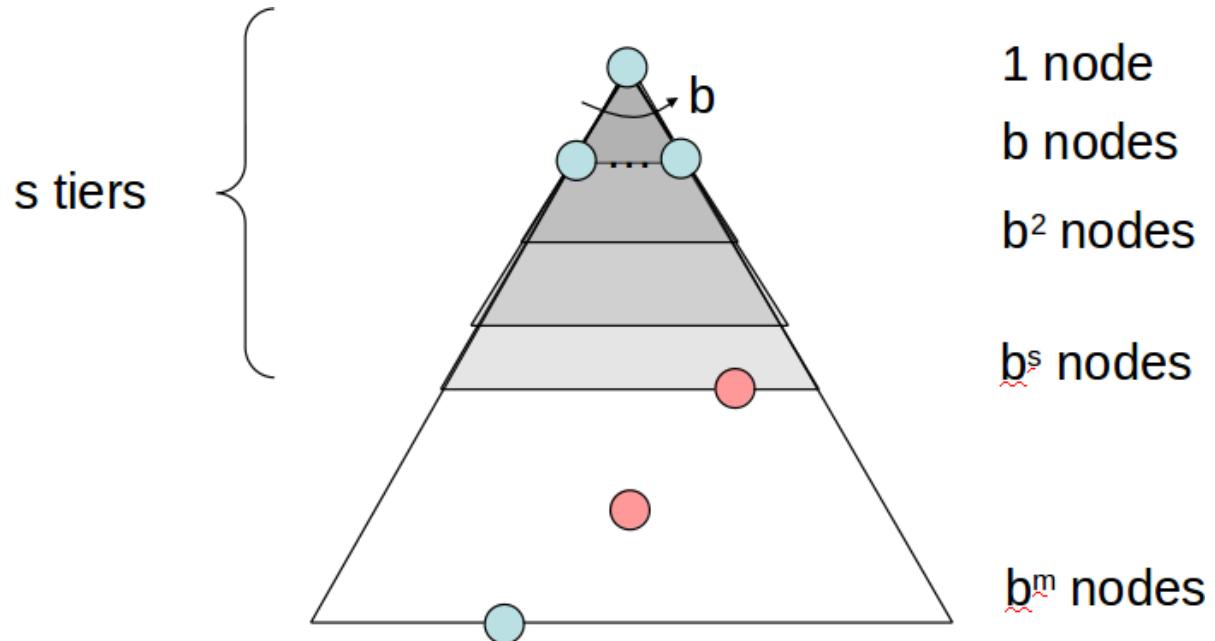


- **Strategy:** expand the shallowest node in the fringe.
- **Implementation:** fringe is a **FIFO queue**.



Properties of BFS

- **Completeness:**
 - If the shallowest goal node is at some finite depth d , BFS will eventually find it after generating all shallower nodes (provided b is finite).
- **Optimality:**
 - The shallowest goal is not necessarily the optimal one.
 - BFS is optimal only if the path cost is a non-decreasing function of the depth of the node.
- **Time complexity:**
 - If the solution is at depth d , then the total number of nodes generated before finding this node is $b + b^2 + b^3 + \dots + b^d = O(b^d)$
- **Space complexity:**
 - The number of nodes to maintain in memory is the size of the fringe, which will be the largest at the last tier. That is $O(b^d)$

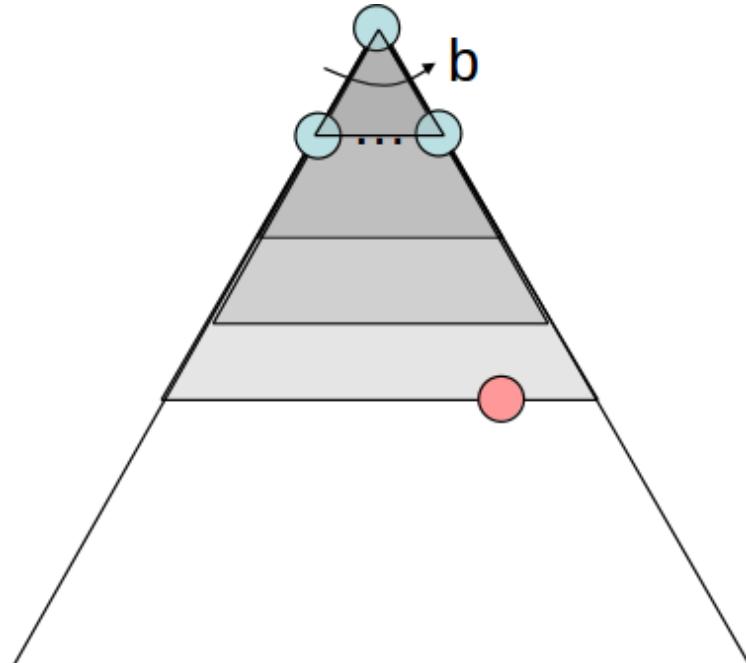


Iterative deepening

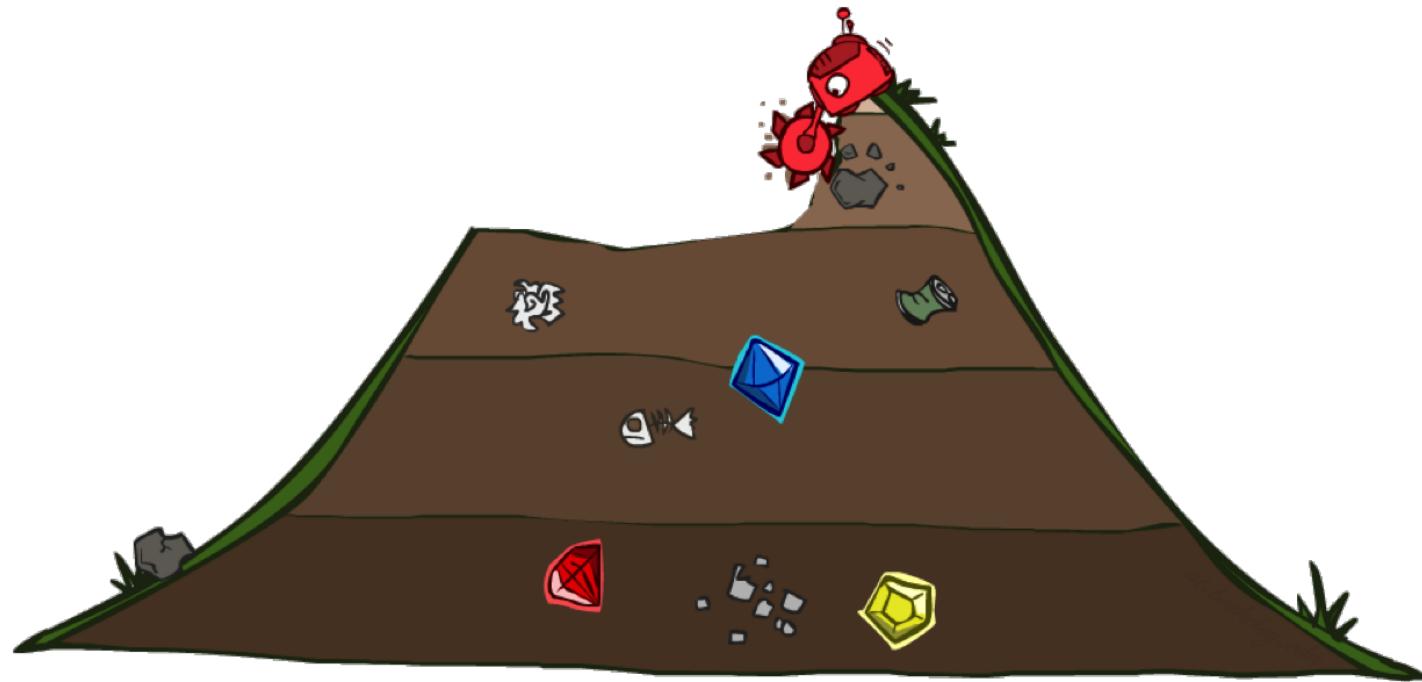
- Idea: get DFS's space advantages with BFS's time/shallow solution advantages.
 - Run DFS with depth limit 1.
 - If no solution, run DFS with depth limit 2.
 - If no solution, run DFS with depth limit 3.
 - ...

[Q] What are the properties of iterative deepening?

[Q] Isn't this process wastefully redundant?



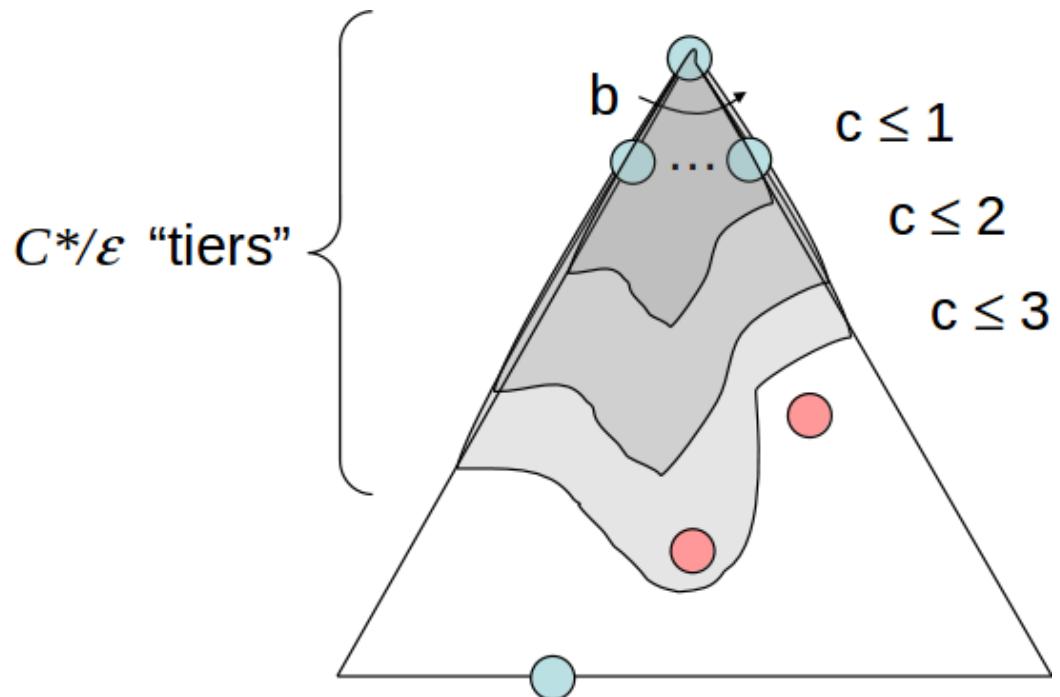
Uniform-cost search



- **Strategy**: expand the cheapest node in the fringe.
- **Implementation**: fringe is a **priority queue**, using the cumulative cost $g(n)$ from the initial state to node n as priority.

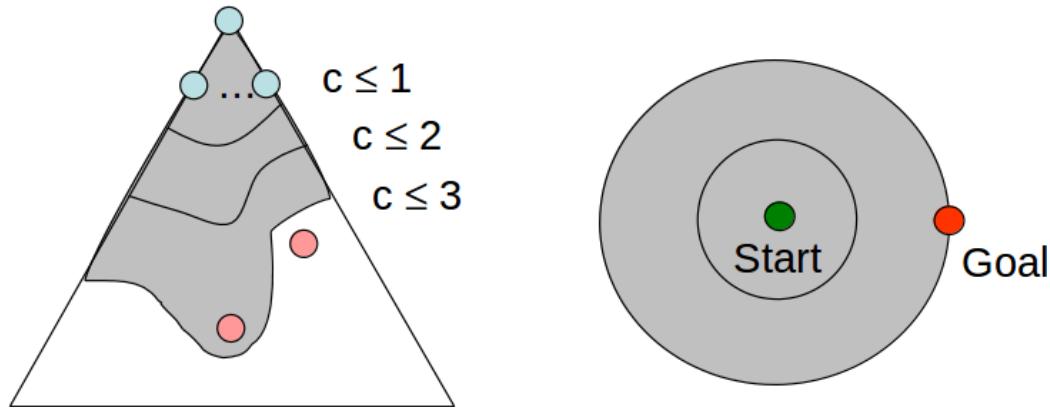
Properties of UCS

- **Completeness:**
 - Yes, if step cost are all such that $c(s, a, s') \geq \epsilon > 0$.
- **Optimality:**
 - Yes, since UCS expands nodes in order of their optimal path cost.
- **Time complexity:**
 - Assume C^* is the cost of the optimal solution and that step costs are all $\geq \epsilon$.
 - The "effective depth" is then roughly C^*/ϵ .
 - The worst-case time complexity is $O(b^{C^*/\epsilon})$.
- **Space complexity:**
 - The number of nodes to maintain is the size of the fringe, so as many as in the last tier $O(b^{C^*/\epsilon})$.



Informed search strategies

One of the **issues of UCS** is that it explores the state space in **every direction**, without exploiting information about the (plausible) location of the goal node.



Informed search strategies aim to solve this problem by expanding nodes in the fringe in decreasing order of **desirability**.

- Greedy search
- A*

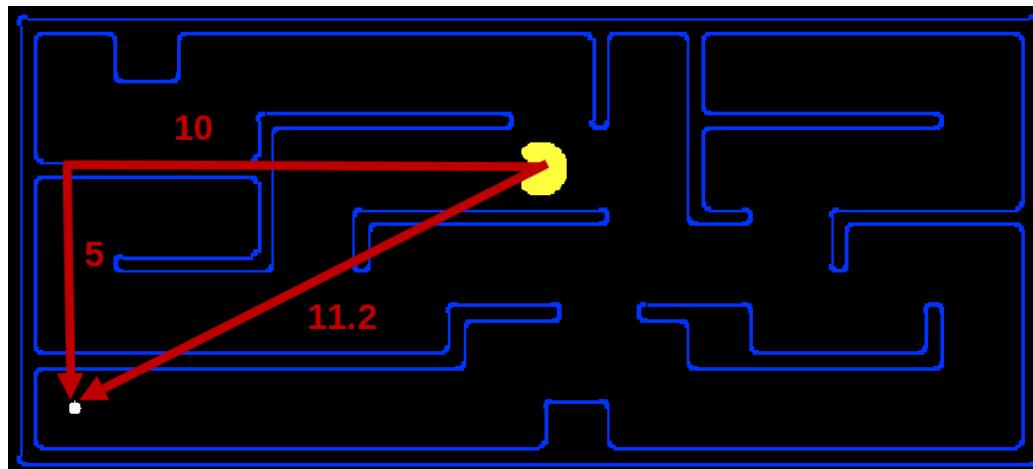
Greedy search



Heuristics

A **heuristic** (or evaluation) function $h(n)$ is:

- a function that **estimates** the cost of the cheapest path from node n to a goal state;
 - $h(n) \geq 0$ for all nodes n
 - $h(n) = 0$ for a goal state.
- is designed for a **particular** search problem.



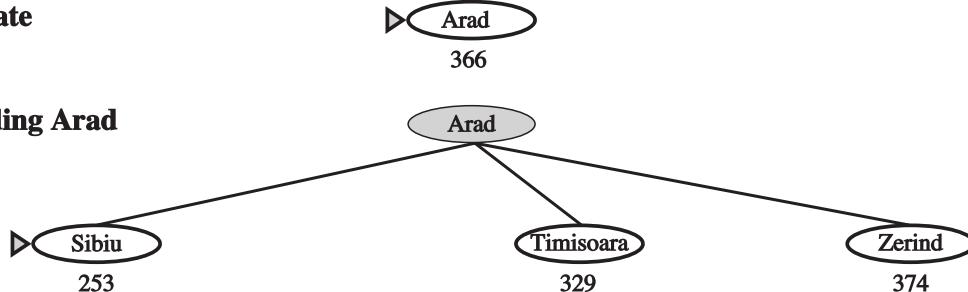
Greedy search

- **Strategy:** expand the node n in the fringe for which $h(n)$ is the lowest.
- **Implementation:** fringe is a **priority queue**, using $h(n)$ as priority.

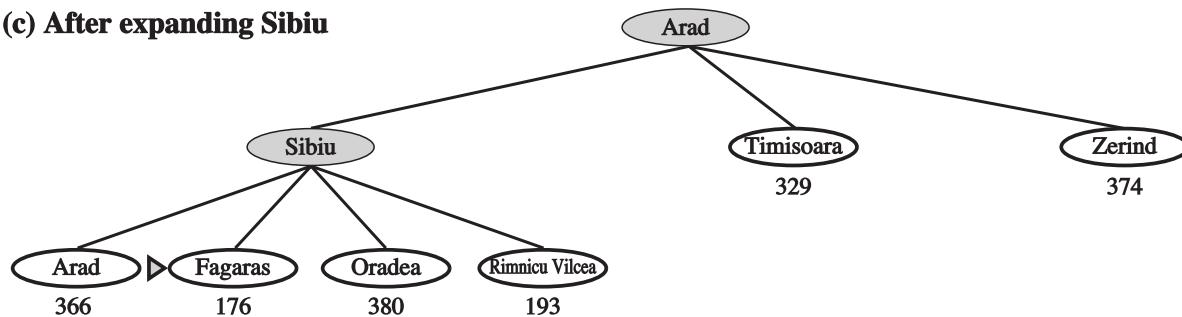
(a) The initial state



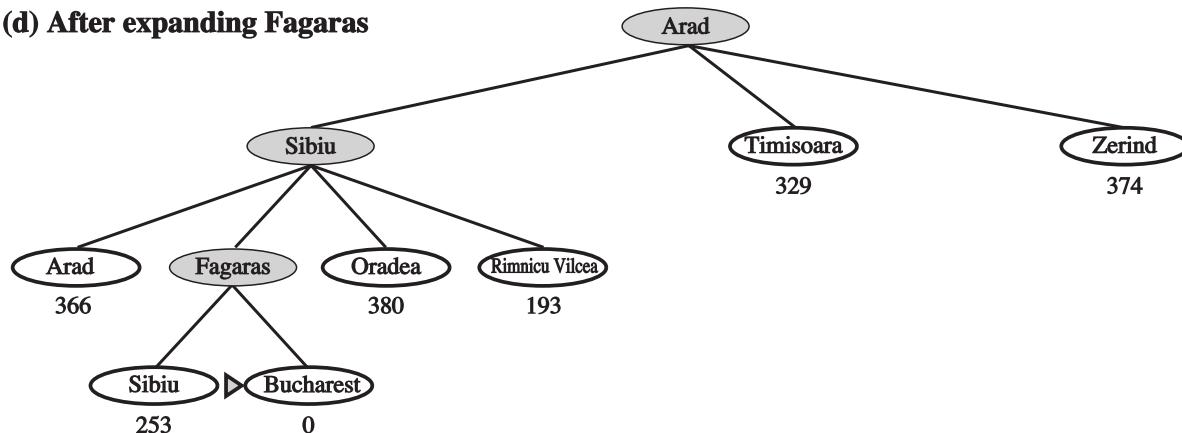
(b) After expanding Arad



(c) After expanding Sibiu



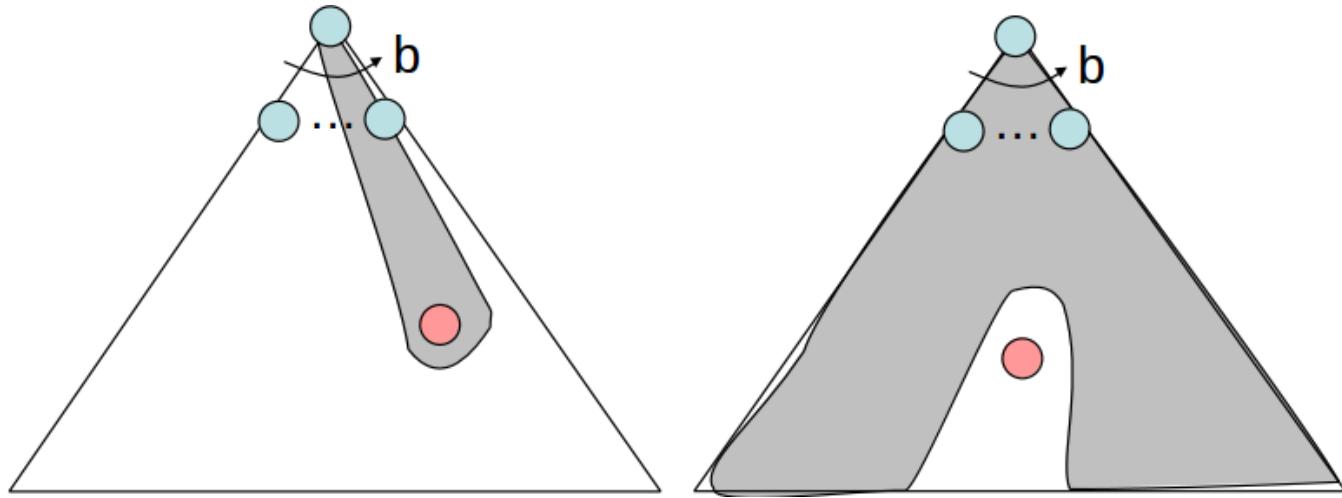
(d) After expanding Fagaras



$h(n)$ = straight line distance to Bucharest.

Properties of greedy search

- Completeness:
 - No, unless we prevent cycles (more on this later).
- Optimality:
 - No, e.g. the path via Sibiu and Fagaras is 32km longer than the path through Rimnicu Vilcea and Pitesti.
- Time complexity:
 - $O(b^m)$, unless we have a good heuristic function.
- Space complexity:
 - $O(b^m)$, unless we have a good heuristic function.



At best, greedy search takes you straight to the goal.

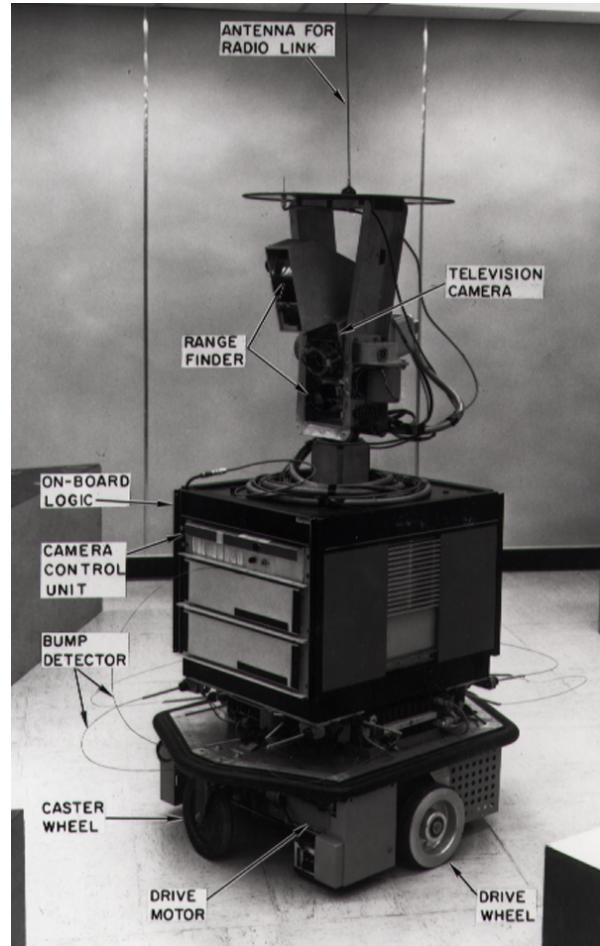
At worst, it is like a badly-guided BFS.

A*



Shakey the Robot

- A* was first proposed in **1968** to improve robot planning.
- Goal was to navigate through a room with obstacles.



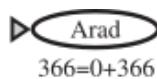
A*

- Uniform-cost orders by path cost, or **backward cost** $g(n)$
- Greedy orders by goal proximity, or **forward cost** $h(n)$
- A* combines the two algorithms and orders by the sum

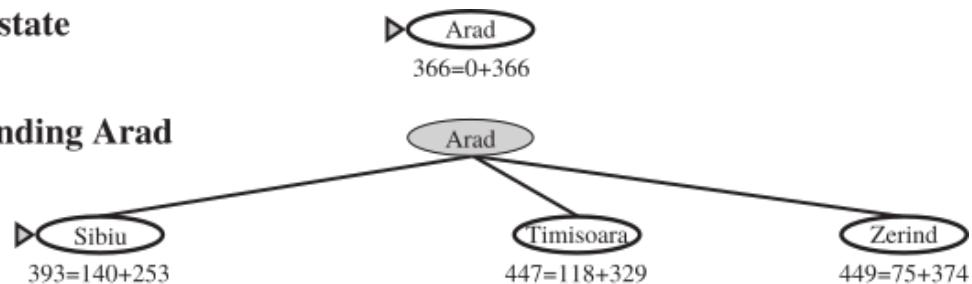
$$f(n) = g(n) + h(n)$$

- $f(n)$ is the estimated cost of cheapest solution through n .

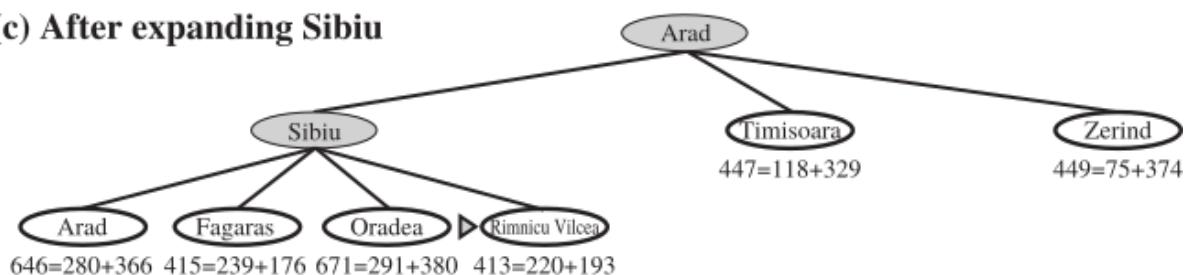
(a) The initial state



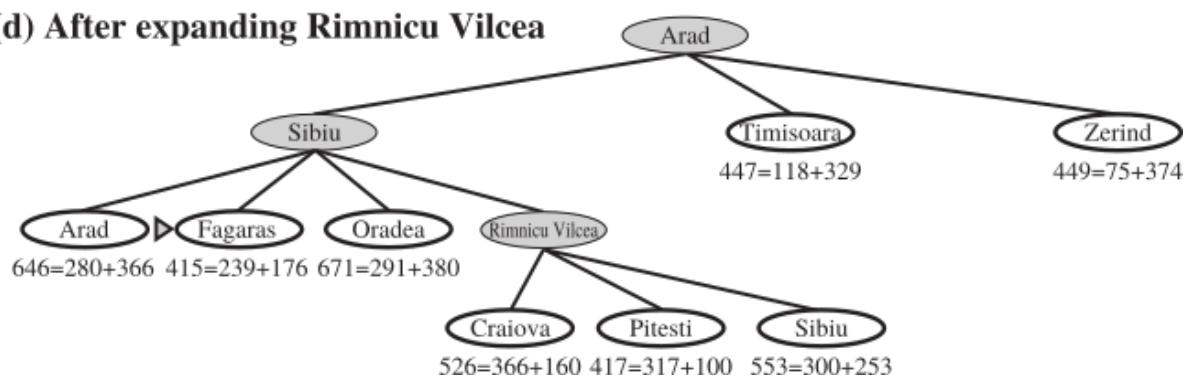
(b) After expanding Arad



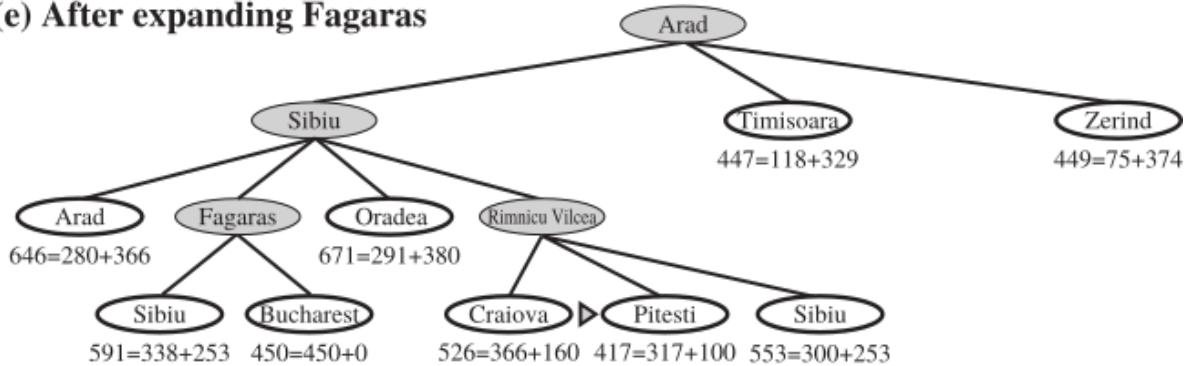
(c) After expanding Sibiu



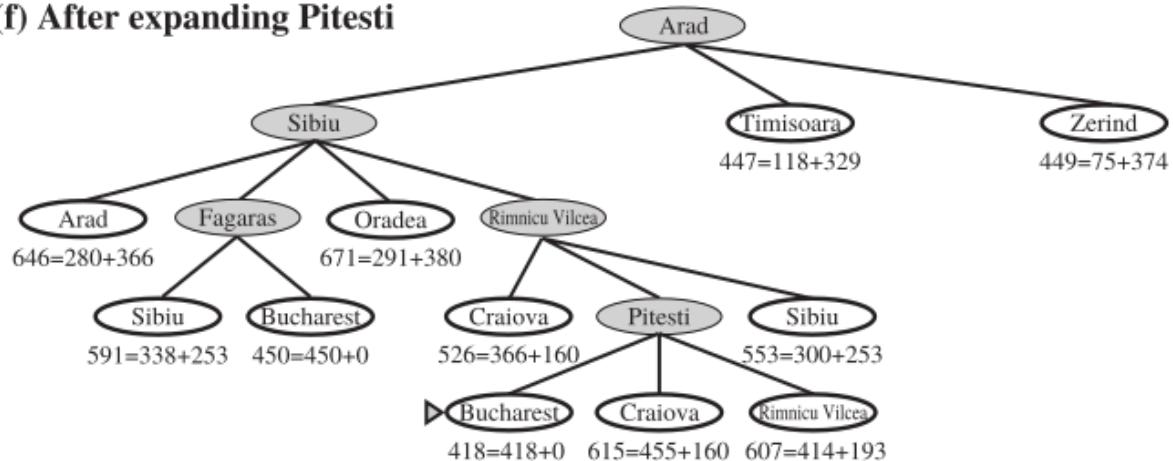
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



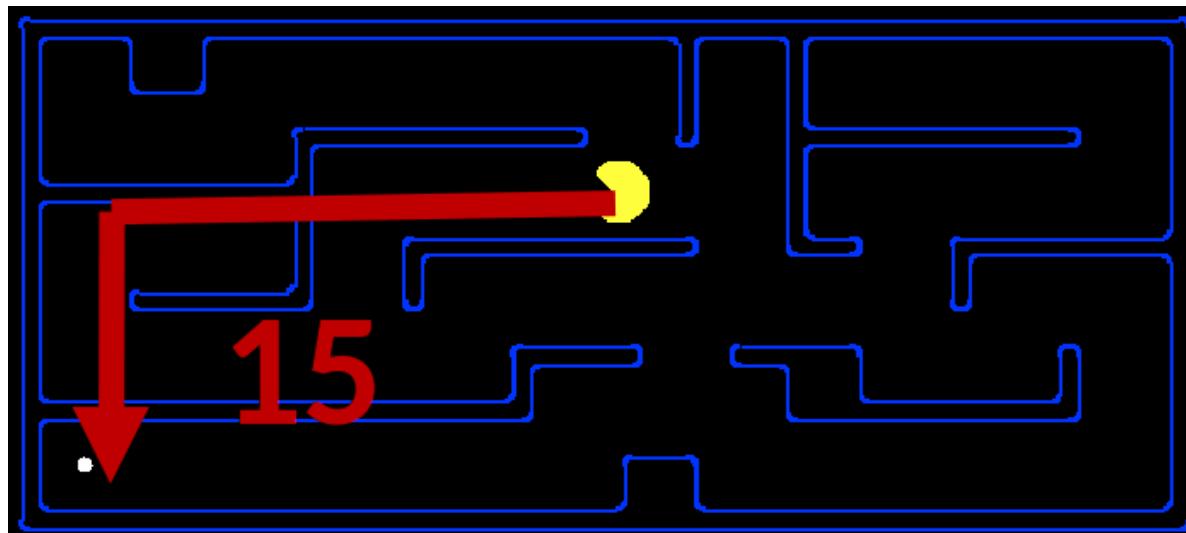
[Q] Why doesn't A* stop at step (e), since Bucharest is in the fringe?

Admissible heuristics

A heuristic h is **admissible** if

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal.



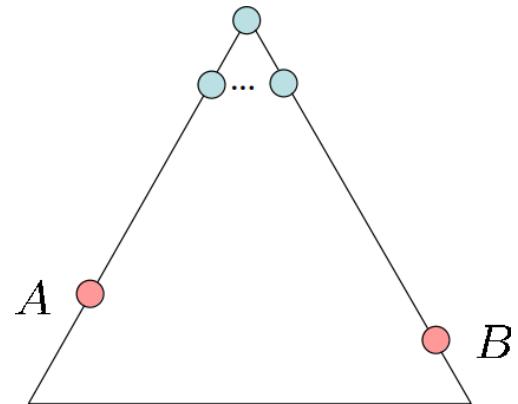
Optimality of A* (tree search)

Assumptions

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim

A will exit the fringe before B .

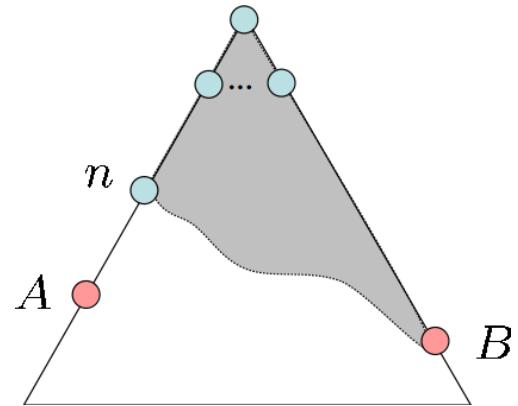


Claim

n will be expanded before B .

Proof

Assume B is on the fringe. Some ancestor n of A is on the fringe too.

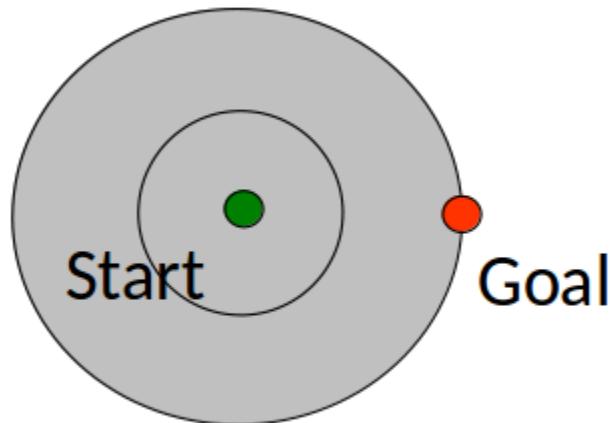


- $f(n) \leq f(A)$
 - $f(n) = g(n) + h(n)$ (by definition)
 - $f(n) \leq g(A)$ (admissibility of h)
 - $f(A) = g(A) + h(A) = g(A)$ ($h = 0$ at a goal)
- $f(A) < f(B)$
 - $g(A) < g(B)$ (B is suboptimal)
 - $f(A) < f(B)$ ($h = 0$ at a goal)
- n expands before B
 - since $f(n) \leq f(A) < f(B)$

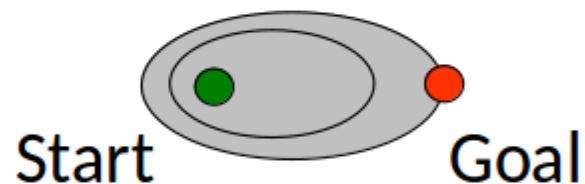
All ancestors of A expand before B , including A . Therefore A^* is optimal.

A* contours

- Assume f -costs are non-decreasing along any path.
- We can define **contour levels t** in the state space, that include all nodes n for which $f(n) \leq t$.



For UCS ($h(n) = 0$ for all n), bands are circular around the start.



For A* with accurate heuristics, bands stretch towards the goal.



Greedy search



UCS



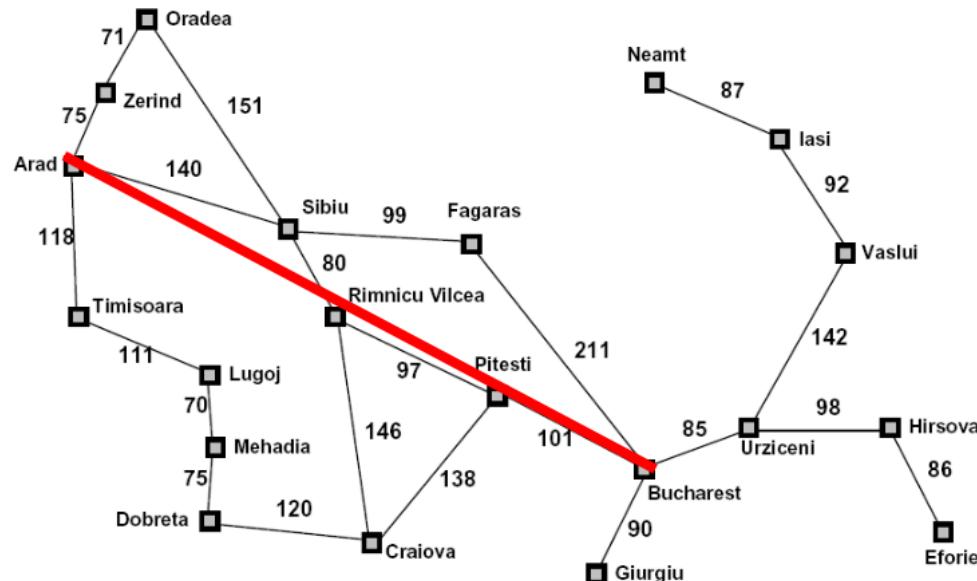
A*

Creating admissible heuristics

Most of the work in solving hard search problems optimally is in finding admissible heuristics.

Admissible heuristics can be derived from the exact solutions to [relaxed problems](#), where new actions are available.

366



Dominance

- If h_1 and h_2 are both admissible and if $h_2(n) \geq h_1(n)$ for all n , then h_2 dominates h_1 and is better for search.
- Given any admissible heuristics h_a and h_b ,

$$h(n) = \max(h_a(n), h_b(n))$$

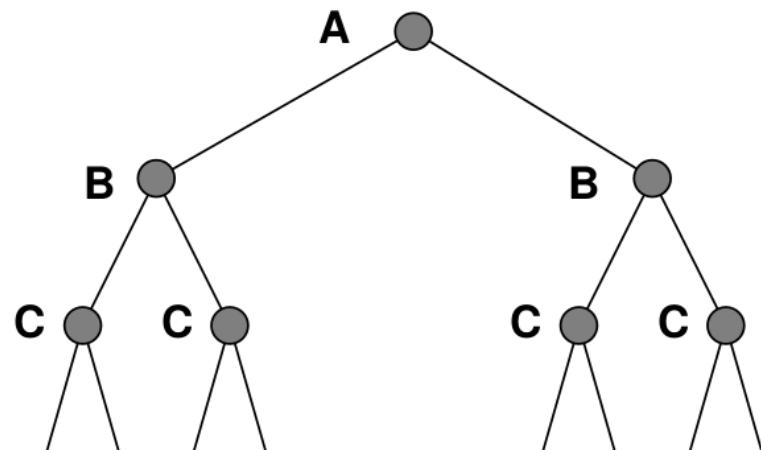
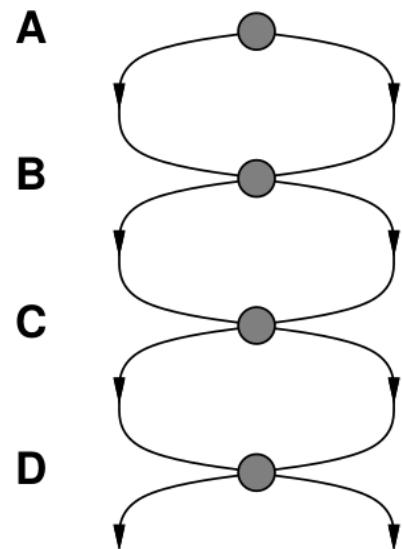
is also admissible and dominates h_a and h_b .

Learning heuristics from experience

- Assuming an **episodic** environment, an agent can **learn** good heuristics by playing the game many times.
- Each optimal solution s^* provides **training examples** from which $h(n)$ can be learned.
- Each example consists of a state n from the solution path and the actual cost $g(s^*)$ of the solution from that point.
- The mapping $n \rightarrow g(s^*)$ can be learned with **supervised learning** algorithms.
 - Linear models, Neural networks, etc.

Graph search

The failure to detect **repeated states** can turn a linear problem into an exponential one!



Redundant paths and cycles can be avoided by **keeping track** of the states that have been **explored**. This amounts to grow a tree directly on the state-space graph.

```
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
```

[Q] What are the properties of DFS/GFS/UCS/GS/A* based on graph search?

Recap example: Super Mario



- Task environment?
 - performance measure, environment, actuators, sensors?
- Type of environment?
- Search problem?
 - initial state, actions, transition model, goal test, path cost?
- Good heuristic?



A* in action

Summary

- Problem formulation usually requires **abstracting away real-world details** to define a state space that can feasibly be explored.
- Variety of uninformed search strategies (**DFS**, **BFS**, **UCS**, **Iterative deepening**)
- **Heuristic functions** estimate costs of shortest paths.
- Good heuristic can dramatically **reduce search cost**.
- **Greedy best-first search** expands lowest h .
 - **incomplete** and **not always optimal**.
- **A*** search expands lower $f = g + h$
 - **complete** and **optimal**
- Admissible heuristics can be derived from exact solutions of relaxed problems or **learned** from training examples.
- **Graph search** can be exponentially more efficient than tree search.

References

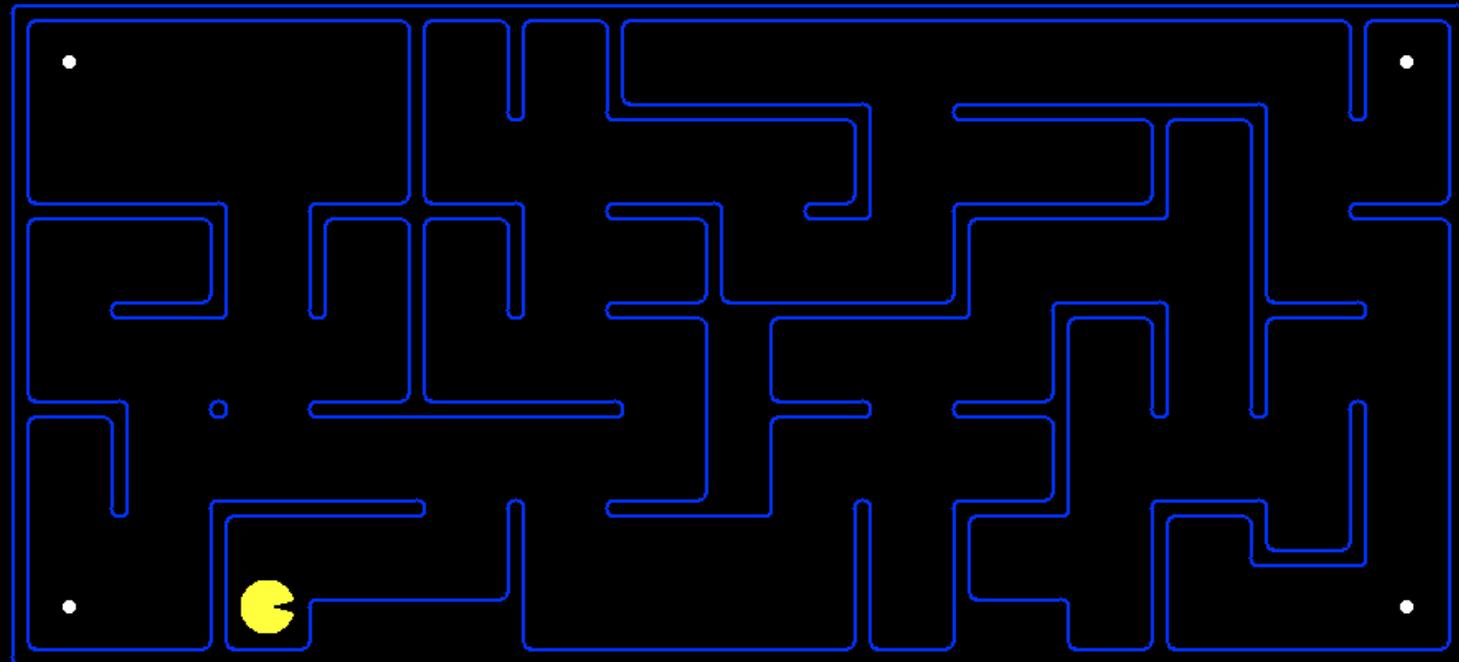
- Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." IEEE transactions on Systems Science and Cybernetics 4.2 (1968): 100-107.

Introduction to Artificial Intelligence

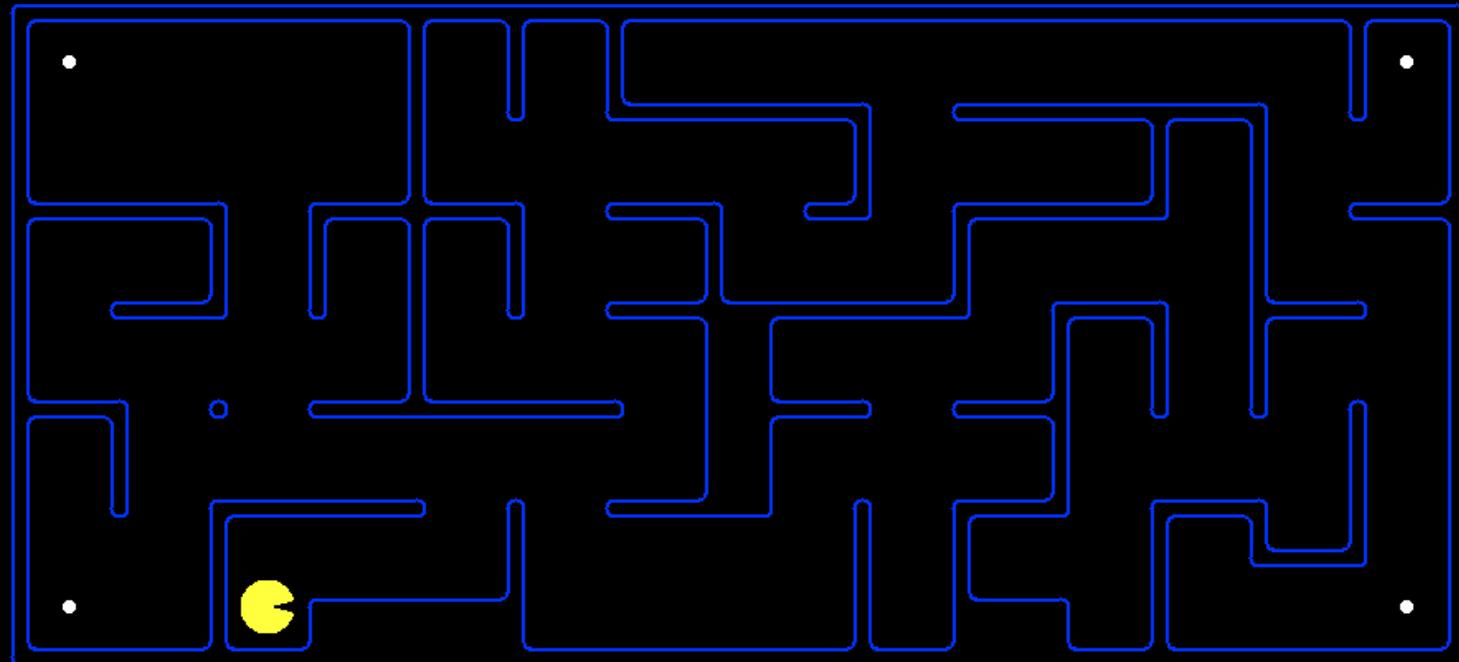
Lecture 3: Constraint satisfaction problems

Prof. Gilles Louppe
g.louppe@uliege.be

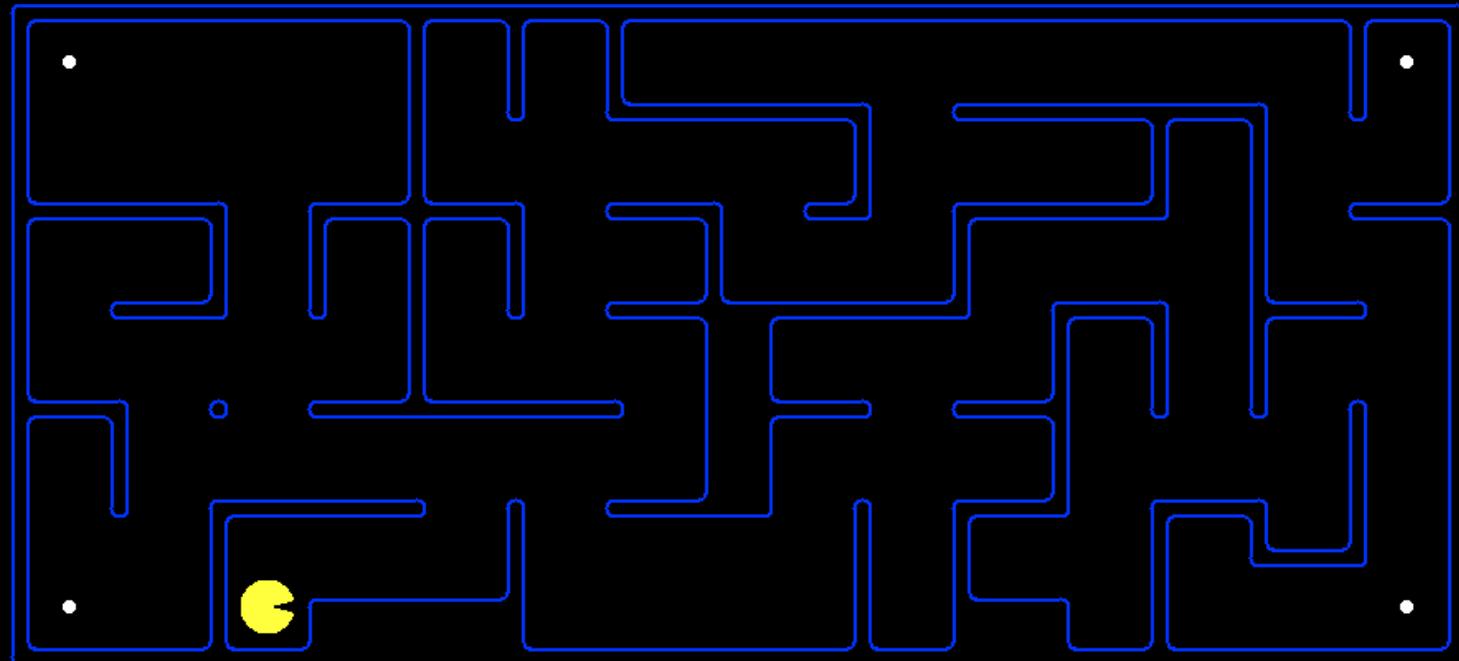




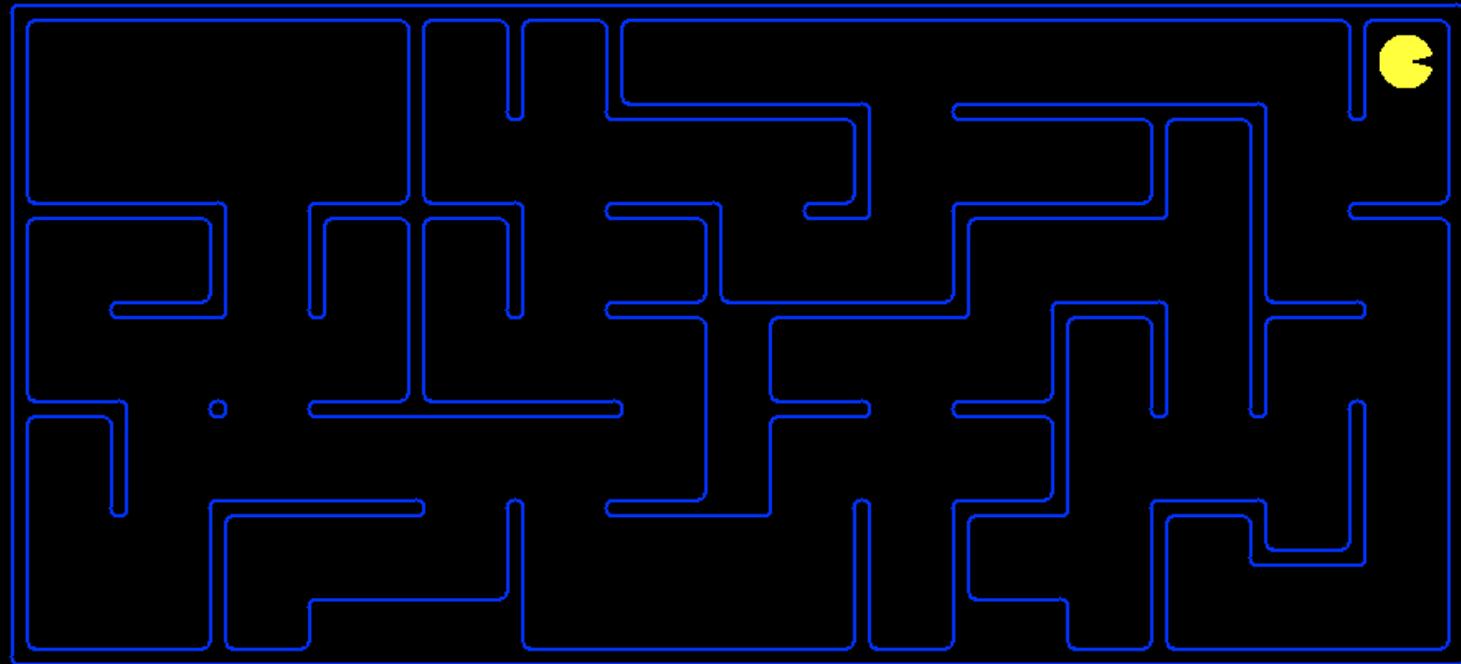
Hmmm, let me think...



(...)



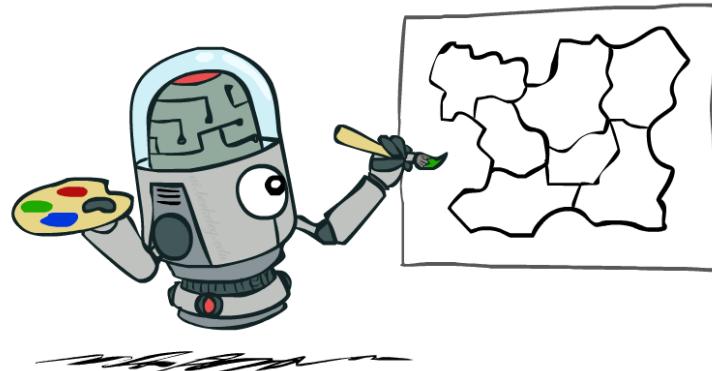
(5 minutes later)



Solution found! [Can we do better?]

Today

- Constraint satisfaction problems:
 - Exploiting the representation of a state to accelerate search.
 - Backtracking.
 - Generic heuristics.
- Logical agents
 - Propositional logic for reasoning about the world.
 - ... and its connection with CSPs.



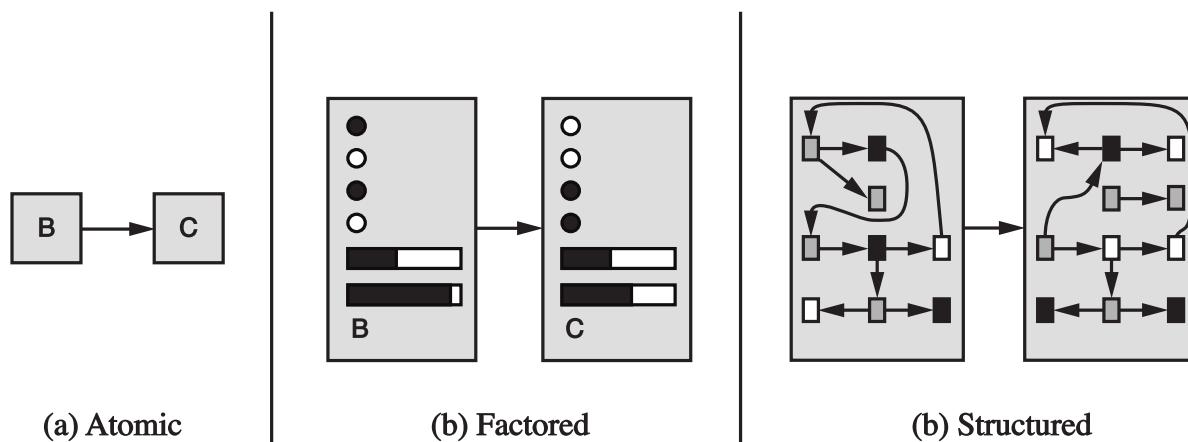
Constraint satisfaction problems

Motivation

In standard search problems:

- States are evaluated by domain-specific heuristics.
- States are tested by a domain-specific function to determine if the goal is achieved.
- From the point of view of the search algorithms however, **states are atomic**.

Instead, if states have **a factored representation**, then the structure of states can be exploited to improve the **efficiency of the search**.



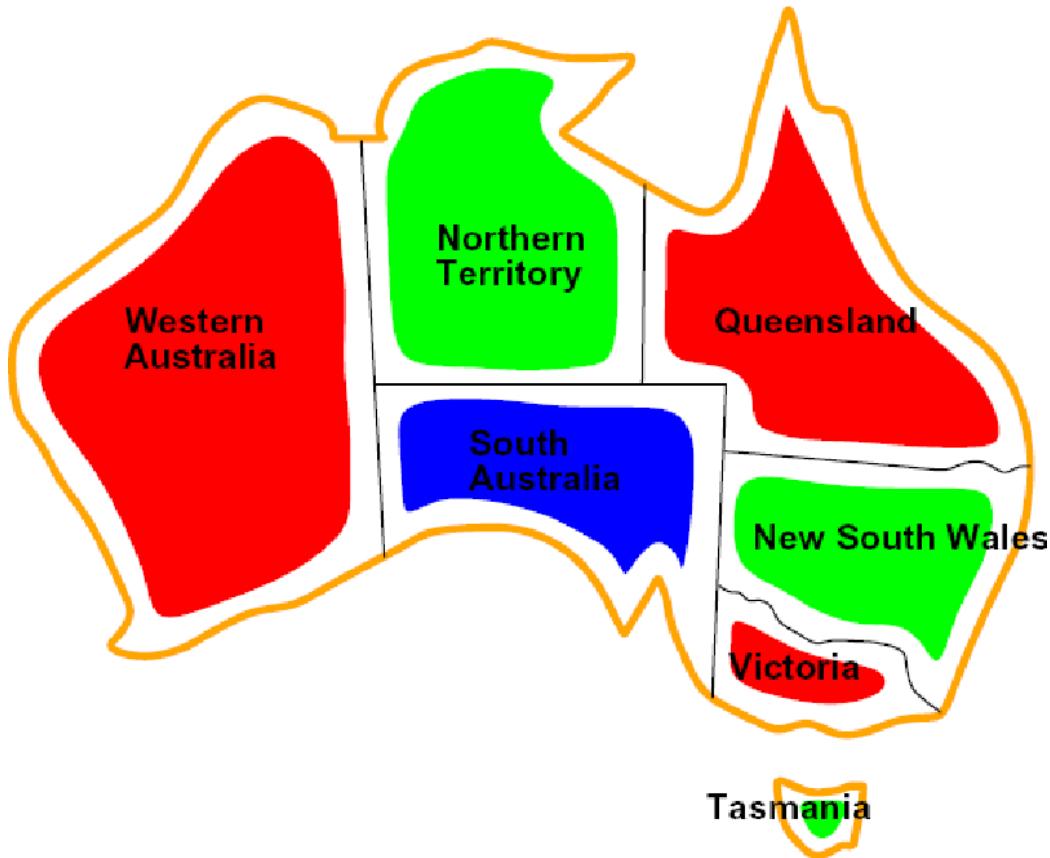
Constraint satisfaction problems

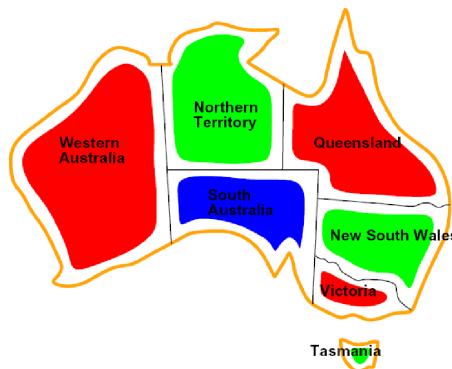
- Constraint satisfaction problem algorithms take advantage of factored state representations and use general-purpose heuristics to solve complex problems.
- CSPs are specialized to a family of search sub-problems.
- Main idea: eliminate large portions of the search space all at once, by identifying combinations of variable/value that violate constraints.

Formally, a **constraint satisfaction problem** (CSP) consists of three components X , D and C :

- X is a set of **variables**, $\{X_1, \dots, X_n\}$,
- D is a set of **domains**, $\{D_1, \dots, D_n\}$, one for each variable,
- C is a set of **constraints** that specify allowable combinations of values.

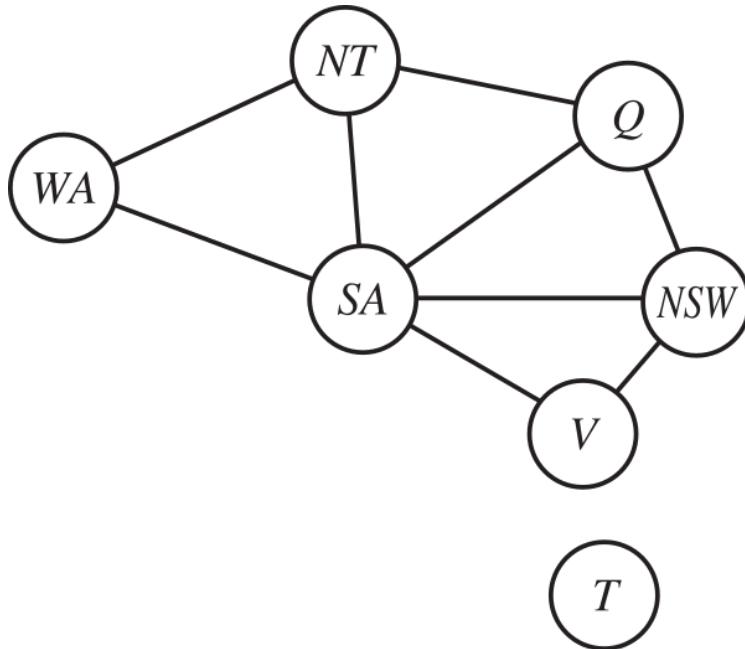
Example: Map coloring





- Variables: $X = \{\text{WA}, \text{NT}, \text{Q}, \text{NSW}, \text{V}, \text{SA}, \text{T}\}$
- Domains: $D_i = \{\text{red}, \text{green}, \text{blue}\}$ for each variable.
- Constraints: $C = \{\text{SA} \neq \text{WA}, \text{SA} \neq \text{NT}, \text{SA} \neq \text{Q}, \dots\}$
 - Implicit: $\text{WA} \neq \text{NT}$
 - Explicit: $(\text{WA}, \text{NT}) \in \{\{\text{red}, \text{green}\}, \{\text{red}, \text{blue}\}, \dots\}$
- Solutions are **assignments** of values to the variables such that constraints are all satisfied.
 - e.g., $\{\text{WA} = \text{red}, \text{NT} = \text{green}, \text{Q} = \text{red}, \text{SA} = \text{blue}, \text{NSW} = \text{green}, \text{V} = \text{red}, \text{T} = \text{green}\}$

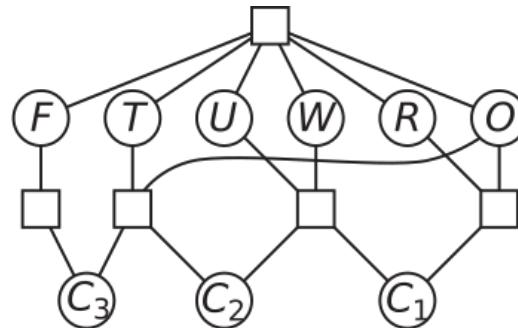
Constraint (hyper)graph



- **Nodes** = variables of the problems
- **Edges** = constraints in the problem involving the variables associated to the end nodes.
- General purpose CSP algorithms **use the graph structure** to speedup search.
 - e.g., Tasmania is an independent subproblem.

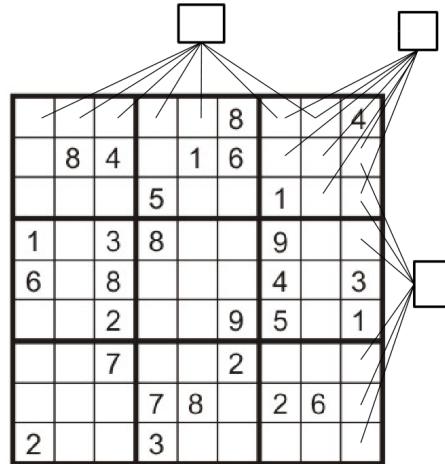
Example: Cryptarithmetic

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



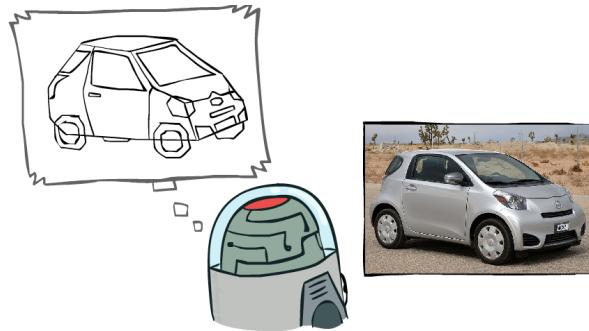
- Variables: $\{T, W, O, F, U, R, C_1, C_2, C_3\}$
- Domains: $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
 - $\text{alldiff}(T, W, O, F, U, R)$
 - $O + O = R + 10 \times C_1$
 - $C_1 + W + W = U + 10 \times C_2$
 - ...

Example: Sudoku



- Variables: each (open) square
- Domains: $D_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
 - 9-way **alldiff** for each column
 - 9-way **alldiff** for each row
 - 9-way **alldiff** for each region

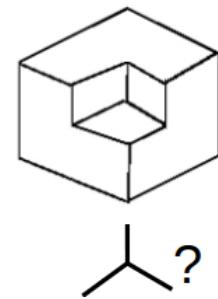
Example: The Waltz algorithm



The Waltz algorithm is a procedure for interpreting 2D line drawings of solid polyhedra as 3D objects. Early example of an AI computation posed as a CSP.

CSP formulation:

- Each intersection is a variable.
- Adjacent intersections impose constraints on each other.
- Solutions are physically realizable 3D objects.



Variations on the CSP formalism

- Discrete variables

- Finite domains
 - Size d means $O(d^n)$ complete assignments.
 - e.g., boolean CSPs, including the SAT boolean satisfiability problem (NP-complete).
- Infinite domains
 - e.g., job scheduling, variables are start/end days for each job.
 - need a constraint language, e.g. $start_1 + 5 \leq start_2$.
 - Solvable for linear constraints, undecidable otherwise.

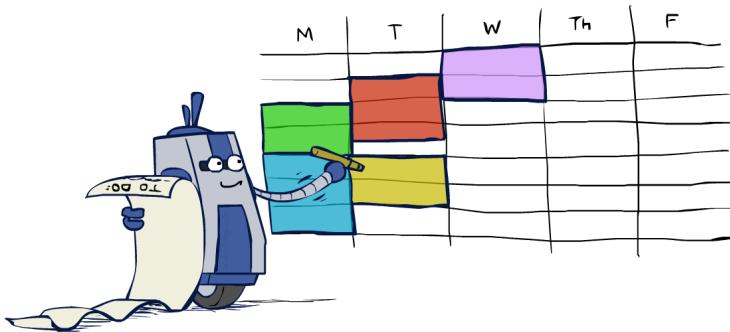
- Continuous variables

- e.g., precise start/end times of experiments.
- Linear constraints solvable in polynomial time by LP methods.

- **Varieties of constraints:**
 - Unary constraint involve a single variable.
 - Equivalent to reducing the domain, e.g. $SA \neq$ green.
 - Binary constraints involve pairs of variables, e.g. $SA \neq WA$.
 - Higher-order constraints involve 3 or more variables.
- **Preferences (soft constraints)**
 - e.g., red is better than green.
 - Often representable by a cost for each variable assignment.
 - Results in constraint optimization problems.
 - (We will ignore those for now.)

Real-world examples

- Assignment problems
 - e.g., who teaches what class
- Timetabling problems
 - e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Circuit layout
- ... and many more

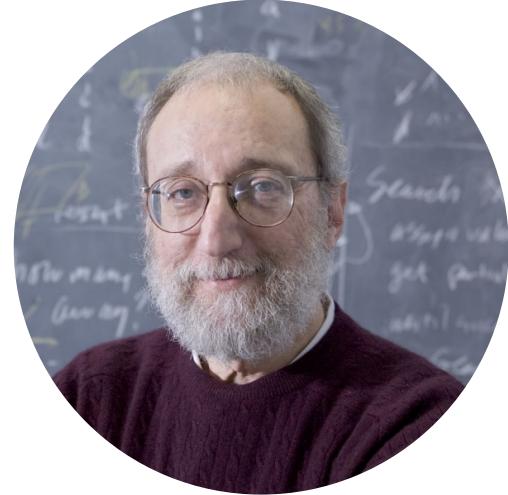


Notice that many real-world problems involve real-valued variables.

Constraint programming

*Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming:
the user states the problem, the computer solves it.*

(Eugene Freuder)



Constraint programming is a programming paradigm in which the user specifies the program as a CSP. The resolution of the problem is left to the computer.

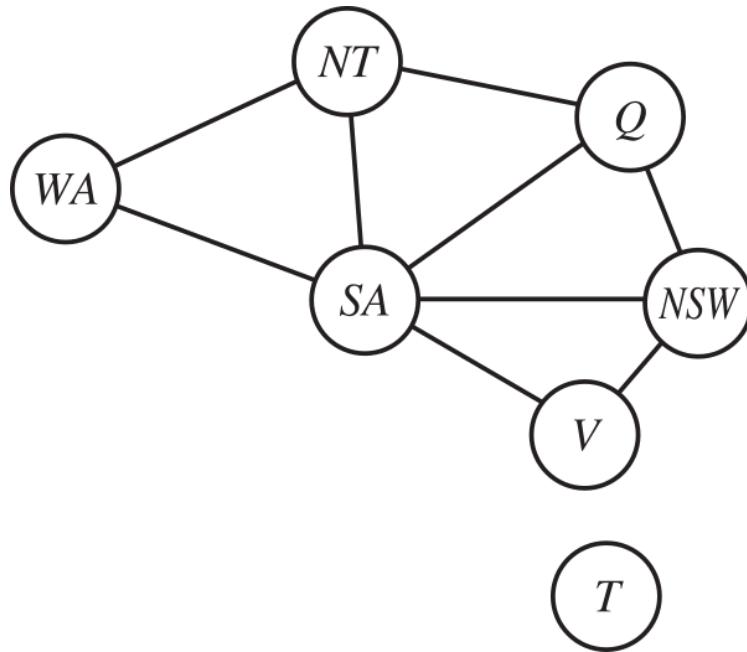
Examples:

- Prolog
- ECLiPSe

Solving CSPs

Standard search formulation

- CSPs can be cast as standard search problems.
 - For which we have solvers, including DFS, BFS or A^{*}.
- States are **partial assignments**:
 - The initial state is the empty assignment $\{\}$.
 - Actions: assign a value to an unassigned variable.
 - Goal test: the current assignment is complete and satisfies all constraints.
- This algorithm is **the same** for all CSPs!



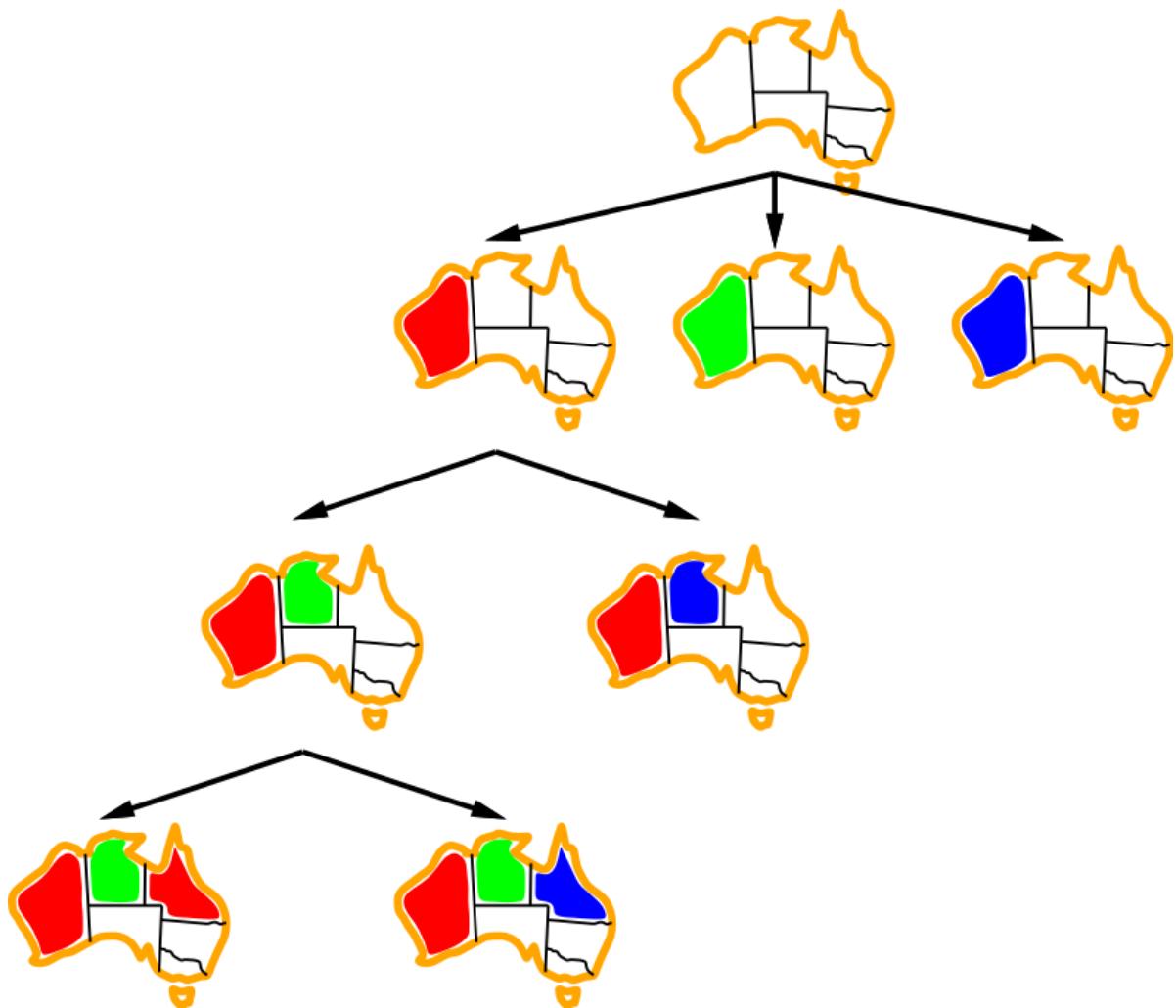
What would BFS or DFS do? What problems does naive search have?

For n variables of domain size d :

- $b = (n - l)d$ at depth l ;
- we generate a tree with $n!d^n$ leaves even if there are only d^n possible assignments!

Backtracking search

- Backtracking search is the basic uninformed algorithm for solving CSPs.
- Idea 1: **One variable at a time:**
 - The naive application of search algorithms ignore a crucial property: variable assignments are **commutative**. Therefore, fix the ordering.
 - $WA = \text{red}$ then $NT = \text{green}$ is the same as $NT = \text{green}$ then $WA = \text{red}$.
 - One only needs to consider assignments to a single variable at each step.
 - $b = d$ and there are d^n leaves.
- Idea 2: **Check constraints as you go:**
 - Consider only values which do not conflict with current partial assignment.
 - Incremental goal test.



```

function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences  $\leftarrow$  INFERENCE(csp, var, value)
      if inferences  $\neq$  failure then
        add inferences to assignment
        result  $\leftarrow$  BACKTRACK(assignment, csp)
        if result  $\neq$  failure then
          return result
        remove {var = value} and inferences from assignment
  return failure

```

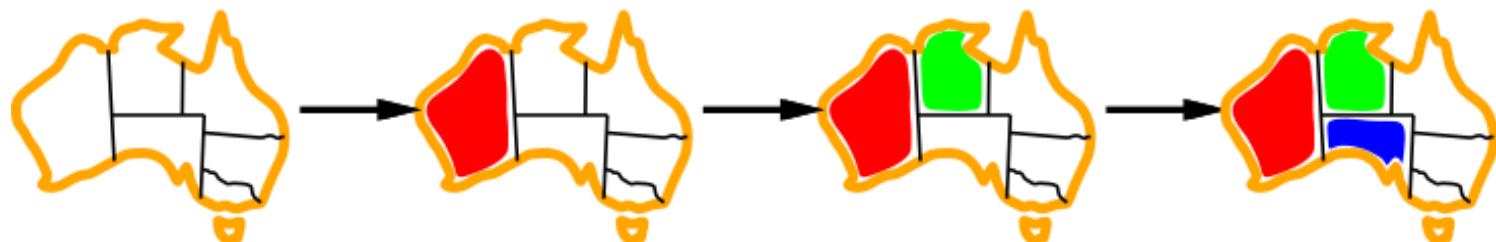
Figure 6.5 A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter 3. By varying the functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES, we can implement the general-purpose heuristics discussed in the text. The function INFERENCE can optionally be used to impose arc-, path-, or *k*-consistency, as desired. If a value choice leads to failure (noticed either by INFERENCE or by BACKTRACK), then value assignments (including those made by INFERENCE) are removed from the current assignment and a new value is tried.

Improving backtracking

- Can we improve backtracking using **general-purpose** ideas, without domain-specific knowledge?
- **Ordering:**
 - Which variable should be assigned next?
 - In what order should its values be tried?
- **Filtering:** can we detect inevitable failure early?
- **Structure:** can we exploit the problem structure?

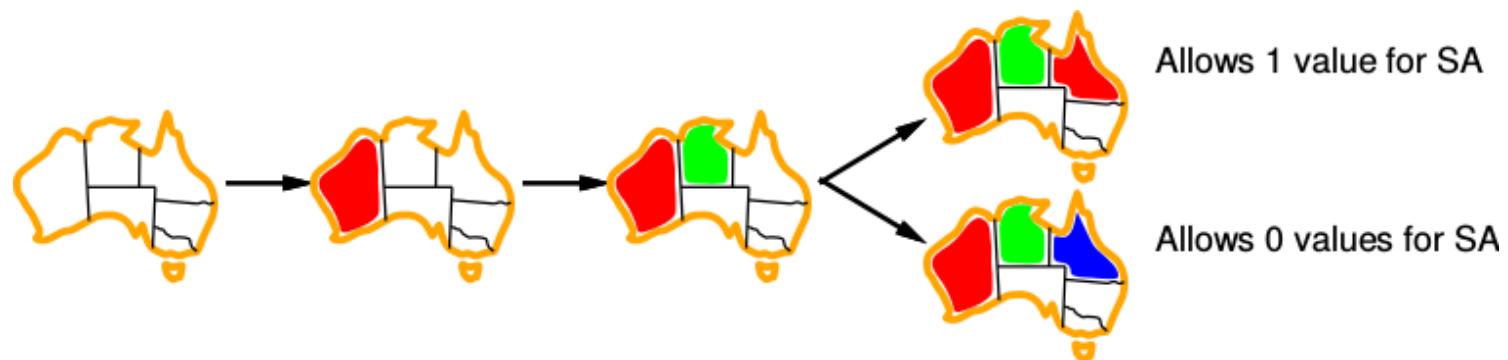
Variable ordering

- Minimum remaining values: Choose the variable with the fewest legal values left in its domain.
- Also known as the fail-first heuristic.
 - Detecting failures quickly is equivalent to pruning large parts of the search tree.



Value ordering

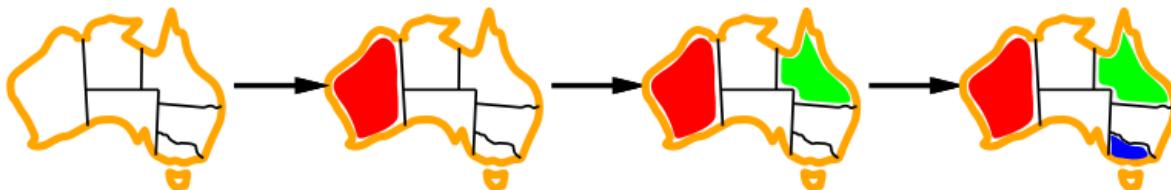
- **Least constraining value:** Given a choice of variable, choose the **least constraining value**.
- i.e., the value that rules out the fewest values in the remaining variables.



[Q] Why should variable selection be fail-first but value selection be fail-last?

Filtering: Forward checking

- Keep track of remaining legal values for unassigned variables.
 - Whenever a variable X is assigned, and for each unassigned variable Y that is connected to X by a constraint, delete from Y 's domain any value that is inconsistent.
- Terminate search when any variable has no legal value left.



WA	NT	Q	NSW	V	SA	T
█ Red	█ Green	█ Blue				
█ Red		█ Green	█ Blue			
█ Red			█ Blue	█ Green		
█ Red				█ Blue		

Filtering: Constraint propagation

Forward checking propagates information assigned to unassigned variables, but does not provide early detection for all failures:

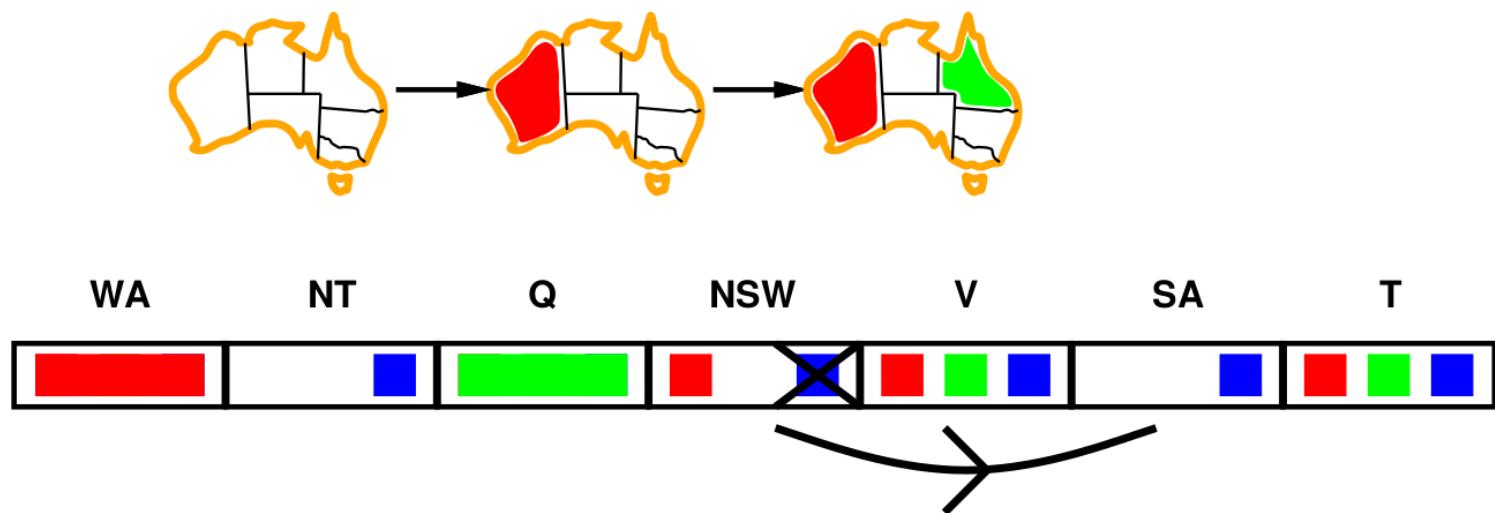


WA	NT	Q	NSW	V	SA	T
Red	Green	Blue	Red	Green	Blue	Red
Red		Green	Blue	Red	Green	Blue
Red		Blue		Red	Green	Blue

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

Arc consistency

- An arc $X \rightarrow Y$ is **consistent** if and only if for every value x in the domain of X there is some value y in the domain of Y that satisfies the associated binary constraint.
- Forward checking \Leftrightarrow enforcing consistency of arcs pointing to each new assignment.
- This principle can be generalized to enforce consistency for **all** arcs.



function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise

inputs: *csp*, a binary CSP with components (*X*, *D*, *C*)

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

while *queue* is not empty **do**

 (*X_i*, *X_j*) \leftarrow REMOVE-FIRST(*queue*)

if REVISE(*csp*, *X_i*, *X_j*) **then**

if size of *D_i* = 0 **then return** false

for each *X_k* **in** *X_i*.NEIGHBORS - {*X_j*} **do**

 add (*X_k*, *X_i*) to *queue*

return true

function REVISE(*csp*, *X_i*, *X_j*) **returns** true iff we revise the domain of *X_i*

revised \leftarrow false

for each *x* **in** *D_i* **do**

if no value *y* in *D_j* allows (*x,y*) to satisfy the constraint between *X_i* and *X_j* **then**

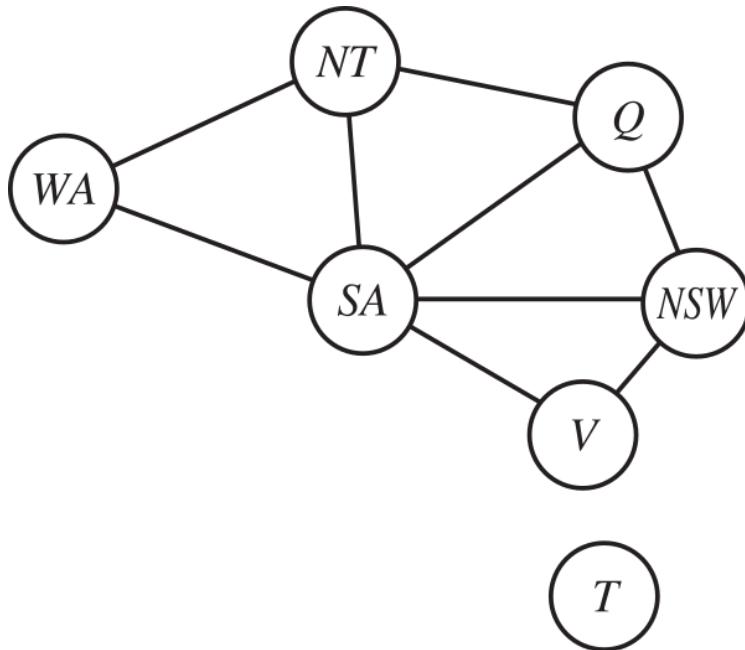
 delete *x* from *D_i*

revised \leftarrow true

return *revised*

[Q] When in backtracking shall this procedure be called?

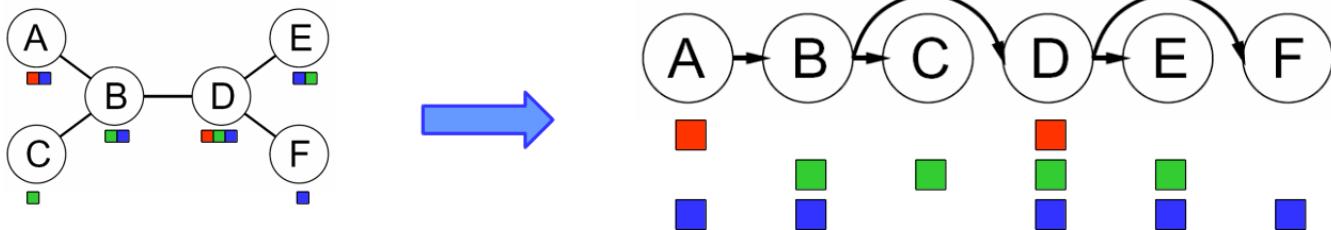
Structure



- Tasmania and mainland are **independent subproblems**.
 - Any solution for the mainland combined with any solution for Tasmania yields a solution for the whole map.
- Independence can be ascertained by finding **connected components** of the constraint graph.

- Time complexity: Assume each subproblem has c variables out of n in total.
Then $O\left(\frac{n}{c}d^c\right)$.
 - E.g., $n = 80, d = 2, c = 20$.
 - $2^{80} = 4$ billion years at 10 million nodes/sec.
 - $4 \times 2^{20} = 0.4$ seconds at 10 million nodes/sec.

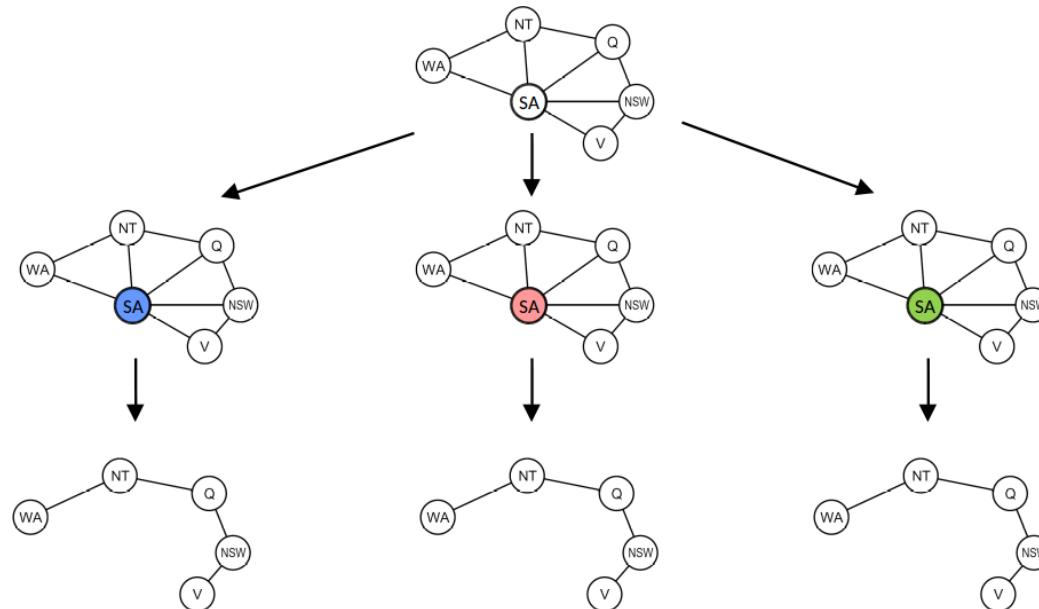
Tree-structured CSPs



- Algorithm for tree-structured CSPs:
 - Order: choose a root variable, order variables so that parents precede children (topological sort).
 - Remove backward:
 - for $i = n$ down to 2, enforce arc consistency of $\text{parent}(X_i) \rightarrow X_i$.
 - Assign forward:
 - for $i = 1$ to n , assign X_i consistently with its $\text{parent}(X_i)$.
- Time complexity: $O(nd^2)$
 - Compare to general CSPs, where worst-case time is $O(d^n)$.

Nearly tree-structured CSPs

- **Conditioning:** instantiate a variable, prune its neighbors' domains.
- **Cutset conditioning:**
 - Assign (in all ways) a set S of variables such that the remaining constraint graph is a tree.
 - Solve the residual CSPs (tree-structured).
 - If the residual CSP has a solution, return it together with the assignment for S .



Logical agents

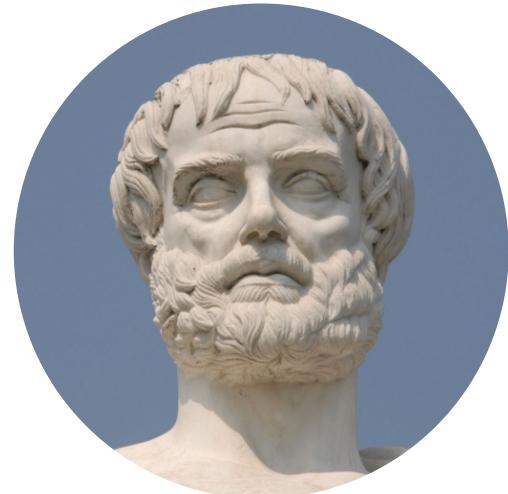
The logicist tradition

- The rational thinking approach to artificial intelligence is concerned with the study of **irrefutable reasoning processes**. It ensures that all actions performed by an agent are formally **provable** from inputs and prior knowledge.
- The Greek philosopher Aristotle was one of the first to attempt to formalize rational thinking. His **syllogisms** provided a pattern for argument structures that always yield correct conclusion when given correct premises.

All men are mortal.

Socrates is a man.

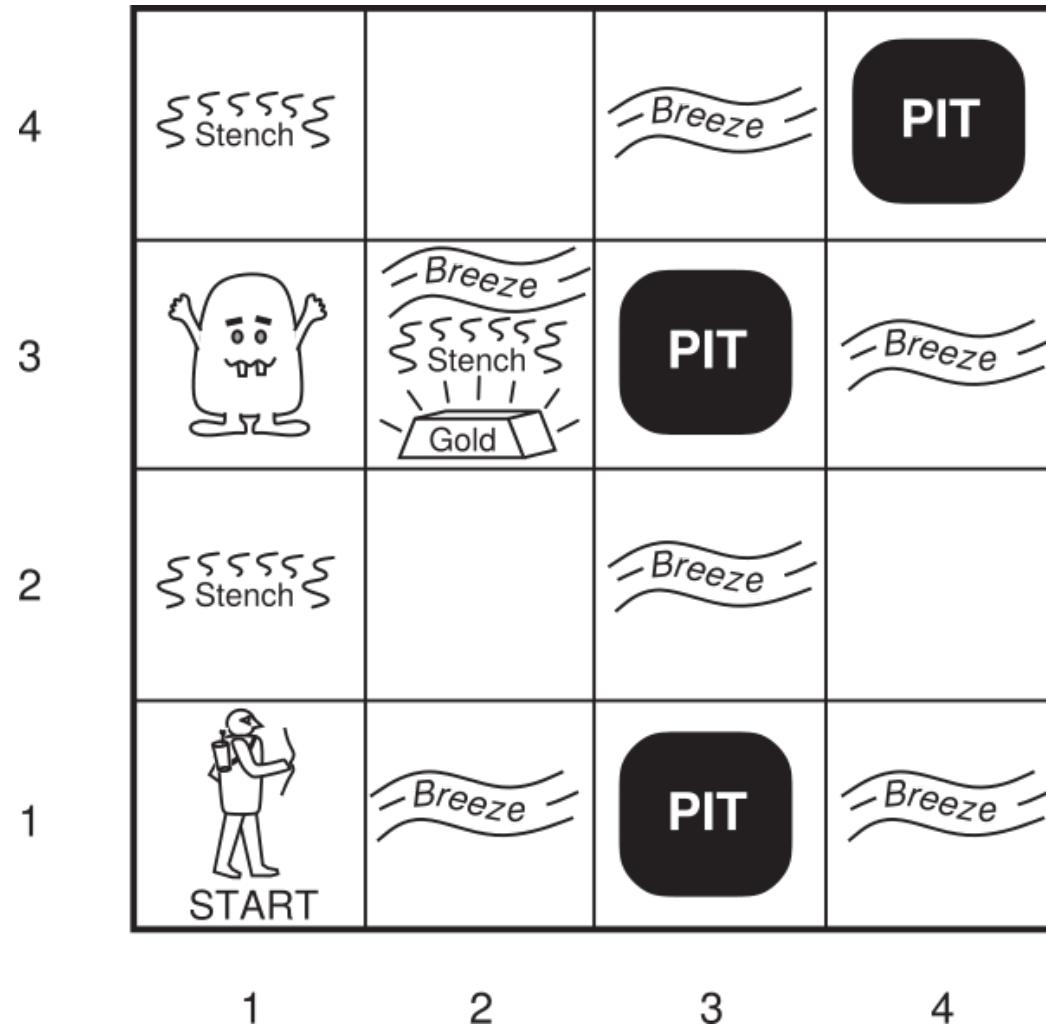
Therefore, Socrates is mortal.



(Aristotle, 384-322 BC)

- Logicians of the 19th century developed a precise notation for statements about all kinds of objects in the world and relationships among them.
- By 1965, programs existed that could, in principle, solve any solvable problem described in logical notation.
- The logicist tradition within AI hopes to build on such programs to create intelligent systems.

The Wumpus world



PEAS description

- **Performance measure:**
 - +1000 for climbing out of the cave with gold;
 - -1000 for falling into a pit or being eaten by the wumpus;
 - -1 per step.
- **Environment:**
 - 4×4 grid of rooms;
 - The agent starts in the lower left square labeled [1, 1], facing right;
 - Locations for gold, the wumpus and pits are chosen randomly from squares other than the start square.
- **Actuators:**
 - Forward, Turn left by 90° or Turn right by 90°
- **Sensors:**
 - Squares adjacent to wumpus are **smelly**;
 - Squares adjacent to pit are **breezy**;
 - **Glitter** if gold is in the same square;
 - Gold is picked up by reflex, and cannot be dropped.
 - You **bump** if you walk into a wall.
 - The agent program receives the percept **[Stench, Breeze, Glitter, Bump]**.

Wumpus world characterization

- **Deterministic**: Yes, outcomes are exactly specified.
- **Static**: Yes, Wumpus and pits do not move.
- **Discrete**: Yes.
- **Single-agent**: Yes, Wumpus is essentially a part of the environment.
- **Fully observable**: No, only **local** perception.
 - This is our first example of partial observability.
- **Episodic**: No, what was observed before is very useful.

The agent need to maintain a model of the world and to update this model upon percepts.

We will use **logical reasoning** to overcome the initial ignorance of the agent.

Exploring the Wumpus world (1)

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A OK	2,1 OK	3,1	4,1

(a)

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1 A V OK	2,1 B OK	3,1 P?	4,1

(b)

(a) Percept = [None, None, None, None]

(b) Percept = [None, Breeze, None, None]

Exploring the Wumpus world (2)

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P! V OK	4,1

(a)

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P! V OK	4,1

(b)

(a) Percept = [Stench, None, None, None]

(b) Percept = [Stench, Breeze, Glitter, None]

Logical agents

- Most useful in non-episodic, partially observable environments.
- Logic (knowledge-based) agents combine:
 - A knowledge base (KB): a list of facts that are known to the agent.
 - Current percepts.
- Hidden aspects of the current state are inferred using rules of inference.
- Logic provides a good formal language for both
 - Facts encoded as axioms.
 - Rules of inference.

```
function KB-AGENT(percept) returns an action
    persistent: KB, a knowledge base
                t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action  $\leftarrow$  ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t  $\leftarrow$  t + 1
    return action
```

Propositional logic

Syntax

The **syntax** of propositional logic defines allowable **sentences**. The syntax of propositional logic is formally defined by the following **grammar**:

```
Sentence → AtomicSentence | ComplexSentence
AtomicSentence → True | False | P | Q | R | ...
ComplexSentence → ( Sentence ) | [ Sentence ]
                     |  $\neg$  Sentence
                     | Sentence  $\wedge$  Sentence
                     | Sentence  $\vee$  Sentence
                     | Sentence  $\Rightarrow$  Sentence
                     | Sentence  $\Leftrightarrow$  Sentence

OPERATOR PRECEDENCE :  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$ ,  $\Leftrightarrow$ 
```

Semantics

- In propositional logic, a **model** is an assignment of truth values for every proposition symbol.
 - E.g., if the sentences of the knowledge base make use of the symbols P_1 , P_2 and P_3 , then one possible model is $m = \{P_1 = \text{False}, P_2 = \text{True}, P_3 = \text{True}\}$.
- The **semantics** for propositional logic specifies how to (recursively) evaluate the **truth value** of any complex sentence, with respect to a model m , as follows:
 - The truth value of a proposition symbol is specified in m .
 - $\neg P$ is true iff P is false;
 - $P \wedge Q$ is true iff P and Q are true;
 - $P \vee Q$ is true iff either P or Q is true;
 - $P \Rightarrow Q$ is true unless P is true and Q is false;
 - $P \Leftrightarrow Q$ is true iff P and Q are both true or both false.

Wumpus world sentences

- Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
- Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

Examples:

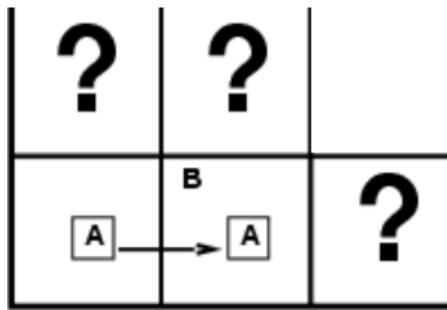
- There is no pit in $[1, 1]$:
 - $R_1 : \neg P_{1,1}$.
- Pits cause breezes in adjacent squares:
 - $R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
 - $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.
 - These are true in all wumpus worlds.
- Breeze percept for the first two squares, for the specific world we consider:
 - $R_4 : \neg B_{1,1}$.
 - $R_5 : B_{2,1}$.

4	~~~~~ Stench ~~~~		~~~~ Breeze ~~~~	PIT
3	Wumpus	~~~~ Breeze ~~~~ ~~~~~ Stench ~~~~ Gold	PIT	~~~~ Breeze ~~~~
2	~~~~~ Stench ~~~~		~~~~ Breeze ~~~~	
1	START	~~~~ Breeze ~~~~	PIT	~~~~ Breeze ~~~~
	1	2	3	4

Entailment

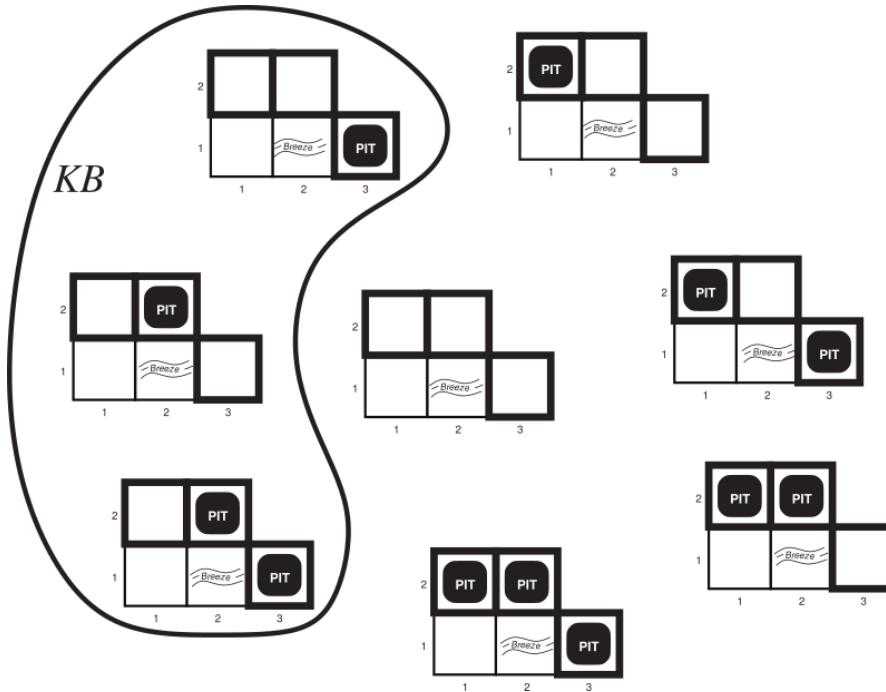
- We say a model m satisfies a sentence α if α is true in m .
 - $M(\alpha)$ is the set of all models that satisfy α .
- $\alpha \models \beta$ iff $M(\alpha) \subseteq M(\beta)$.
 - We say that the sentence α entails the sentence β .
 - β is true in all models where α is true.
 - That is, β follows logically from α .

Wumpus models



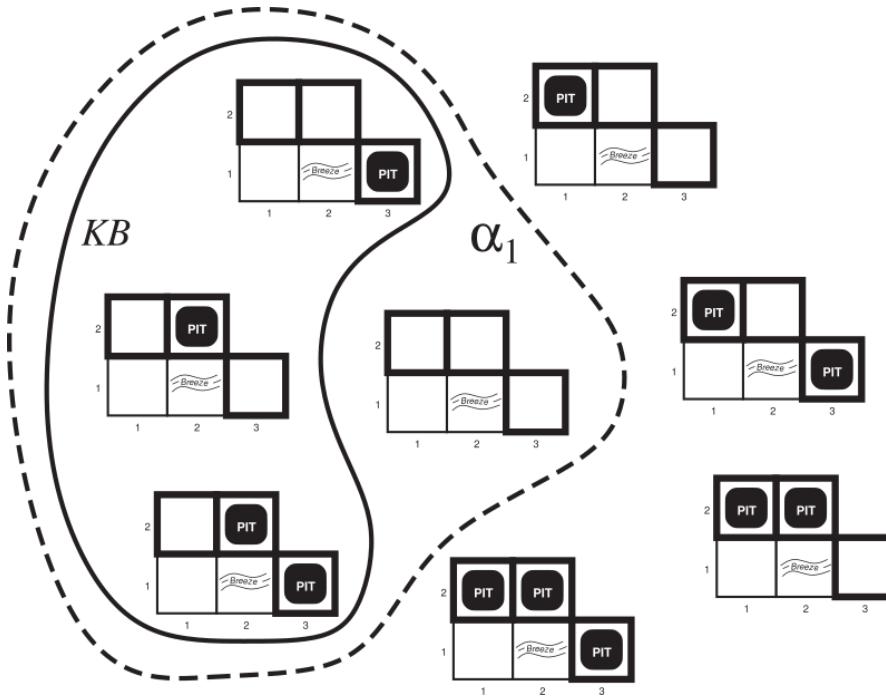
- Let consider possible models for KB assuming only pits and a reduced Wumpus world with only 5 squares and pits.
- We consider the situation after:
 - detecting nothing in [1, 1],
 - moving right, sensing breeze in [2, 1].

[Q] How many models are there?

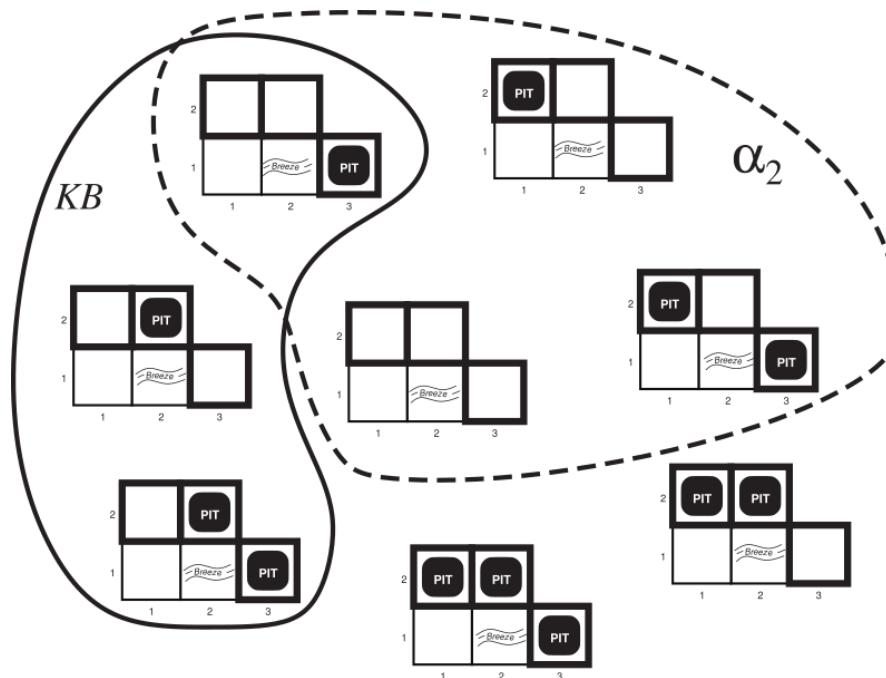


- All 8 possible models in the reduced Wumpus world.
- The knowledge base **KB** contains all possible Wumpus worlds consistent with the observations and the physics of the world.

Entailments



- $\alpha_1 = "[1, 2]"$ is safe". Does KB entails α_1 ?
- $KB \models \alpha_1$ since $M(KB) \subseteq M(\alpha_1)$.
 - This proof is called **model checking** because it **enumerates** all possible models to check whether α_1 is true in all models where KB is true.
- Entailment can be used to carry out **logical inference**.



- $\alpha_2 = "[2, 2]" \text{ is safe}.$ Does KB entail α_2 ?
- $KB \not\models \alpha_2$ since $M(KB) \not\subseteq M(\alpha_2)$.
- We **cannot** conclude whether $[2, 2]$ is safe (it may or may not).

Unsatisfiability theorem

$\alpha \vDash \beta$ iff $(\alpha \wedge \neg\beta)$ is unsatisfiable

- A sentence γ is unsatisfiable iff $M(\gamma) = \{\}$.
 - i.e., there is no assignment of truth values such that γ is true.
- Proving $\alpha \vDash \beta$ by checking the unsatisfiability of $\alpha \wedge \neg\beta$ corresponds to the proof technique of *reductio ad absurdum*.
- Checking the satisfiability of a sentence γ can be cast as CSP!
 - More efficient than enumerating all models, but remains NP-complete.
 - Alternatively, propositional satisfiability (SAT) solvers can be used instead of CSPs. These are tailored for this specific problem. Many of them are variants of backtracking.

Limitations

- Representation of **informal** knowledge is difficult.
- Hard to define provable **plausible** reasoning.
- **Combinatorial explosion** (in time and space).
- Logical inference is only a part of intelligence.

Summary

- Constraint satisfaction problems:
 - States are represented by a set of variable/value pairs.
 - Backtracking, a form of depth-first search, is commonly used for solving CSPs.
 - The complexity of solving a CSP is strongly related to the structure of its constraint graph.
- Logical agents:
 - Intelligent agents need knowledge about the world in order to reach good decisions.
 - Logical inference can be used as tool to reason about the world.
 - The inference problem can be cast as the problem of determining the unsatisfiability of a formula.
 - This in turn can be cast as a CSP.

References

- Newell, A., & Simon, H. (1956). The logic theory machine--A complex information processing system. IRE Transactions on information theory, 2(3), 61-79.
- McCarthy, J. (1960). Programs with common sense (pp. 300-307). RLE and MIT computation center.

Introduction to Artificial Intelligence

Lecture 4: Games and Adversarial search

Prof. Gilles Louppe
g.loupe@uliege.be





Today

- How to act rationally in a **multi-agent** environment?
- How to anticipate and respond to the **arbitrary behavior** of other agents?
- Adversarial search
 - Minimax
 - $\alpha - \beta$ pruning
 - H-Minimax
 - Expectiminimax
 - Monte Carlo Tree Search
- Modeling assumptions
- State-of-the-art agents.

Minimax

Games

- A **game** is a multi-agent environment where agents may have either **conflicting** or **common** interests.
- Opponents may act **arbitrarily**, even if we assume a deterministic fully observable environment.
 - The solution to a game is a **strategy** specifying a move for every possible opponent reply.
 - This is different from search where a solution is a **fixed sequence**.
- Time **limits**.
 - Branching factor is often very large.
 - Unlikely to find goal with standard search algorithms, we need to **approximate**.

Types of games

- Deterministic or stochastic?
- Perfect or imperfect information?
- Two or more players?

Formal definition

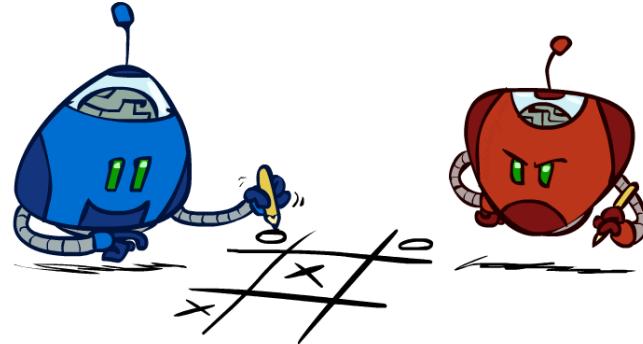
A **game** is formally defined as a kind of search problem with the following components:

- The **initial state** s_0 of the game.
- A function $\text{player}(s)$ that defines which **player** $p \in \{1, \dots, N\}$ has the move in state s .
- A description of the legal **actions** (or **moves**) available to a state s , denoted $\text{actions}(s)$.
- A **transition model** that returns the state $s' = \text{result}(s, a)$ that results from doing action a in state s .
- A **terminal test** which determines whether the game is over.

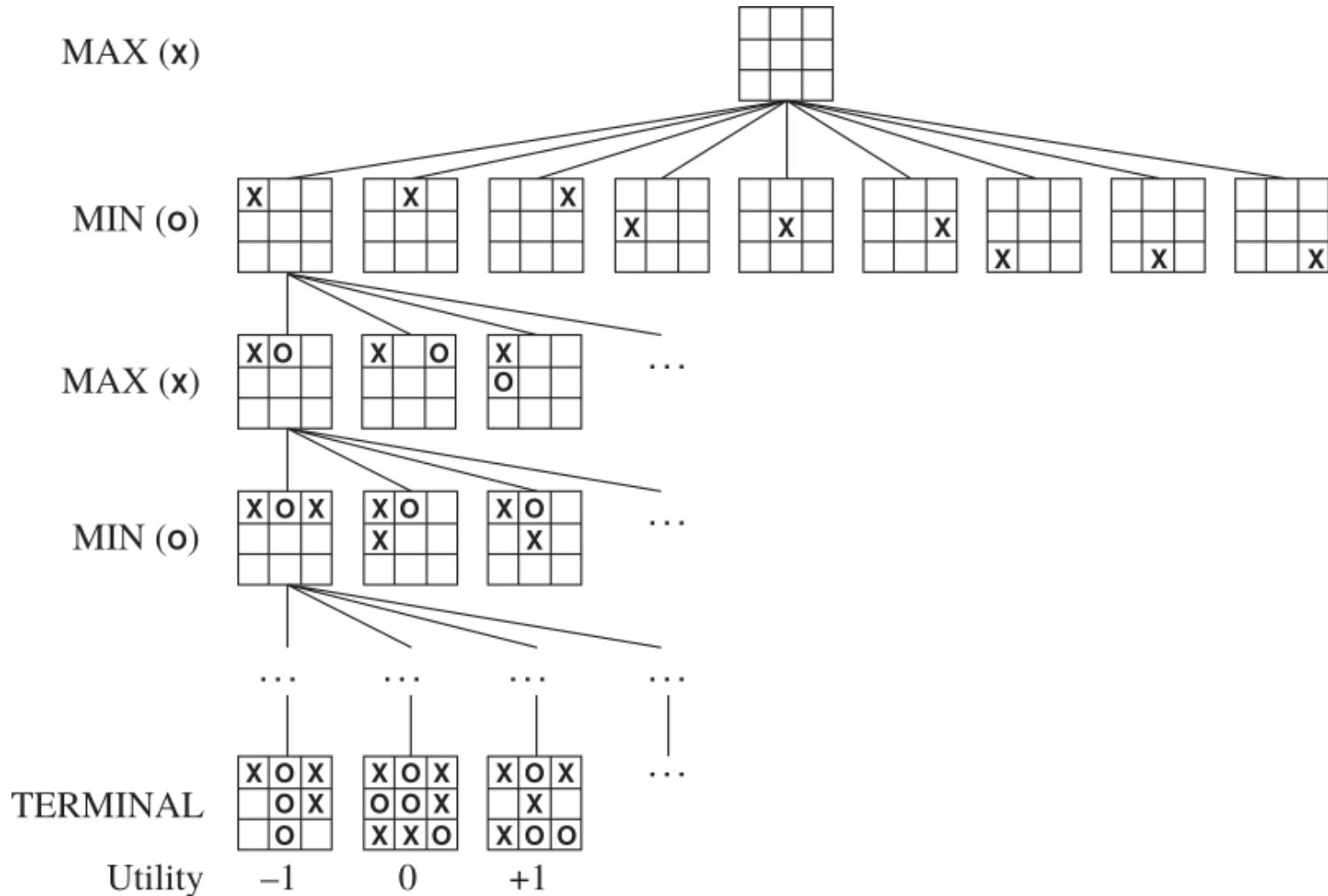
- A utility function $\text{utility}(s, p)$ (or payoff) that defines the final numeric value for a game that ends in s for a player p .
 - E.g., $1, 0$ or $\frac{1}{2}$ if the outcome is win, loss or draw.
- Together, the initial state, the $\text{actions}(s)$ function and the $\text{result}(s, a)$ function define the game tree.
 - Nodes are game states.
 - Edges are actions.

Assumptions

- We assume a **deterministic, turn-taking, two-player zero-sum game** with **perfect information**.
 - e.g., Tic-Tac-Toe, Chess, Checkers, Go, etc.
- We will call our two players **MAX** and **MIN**. **MAX** moves first.



Game tree



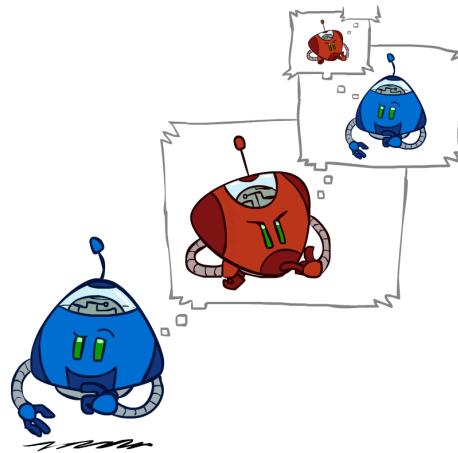
Zero-sum games

- In a **zero-sum** game, the total payoff to all players is **constant** for all games.
 - e.g., in chess: $0 + 1, 1 + 0$ or $\frac{1}{2} + \frac{1}{2}$.
- For two-player games, agents share the **same utility** function, but one wants to **maximize** it while the other wants to **minimize** it.
 - MAX maximizes the game's **utility** function.
 - MIN minimizes the game's **utility** function.
- **Strict competition.**
 - If one wins, the other loses, and vice-versa.



Adversarial search

- In a search problem, the optimal solution is a sequence of actions leading to a goal state.
 - i.e., a terminal state where MAX wins.
- In a game, the opponent (MIN) may react **arbitrarily** to a move.
- Therefore, a player (MAX) must define a contingent **strategy** which specifies
 - its moves in the initial state,
 - its moves in the states resulting from every possible response by MIN,
 - its moves in the states resulting from every possible response by MIN in those states,
 - ...



[Q] What is an optimal strategy (or perfect play)? How do we find it?

Minimax

The **minimax value** $\text{minimax}(s)$ is the largest achievable payoff (for MAX) from state s , assuming an optimal adversary (MIN).

$\text{MINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

The **optimal** next move (for MAX) is to take the action that maximizes the minimax value in the resulting state.

- Assuming that MIN is an optimal adversary that maximizes the **worst-case outcome** for MAX.
- This is equivalent to not making an assumption about the strength of the opponent.

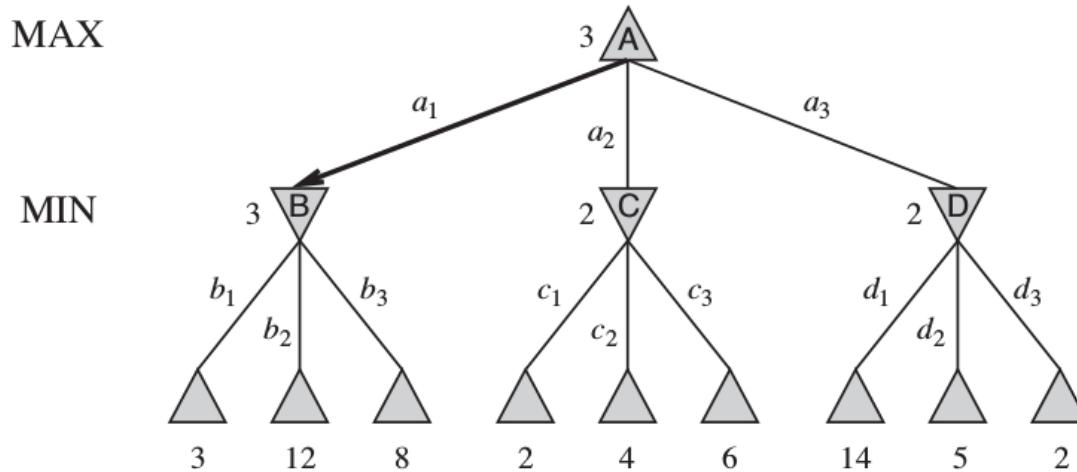


Figure 5.2 A two-ply game tree. The \triangle nodes are “MAX nodes,” in which it is MAX’s turn to move, and the ∇ nodes are “MIN nodes.” The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX’s best move at the root is a_1 , because it leads to the state with the highest minimax value, and MIN’s best reply is b_1 , because it leads to the state with the lowest minimax value.

Properties of Minimax

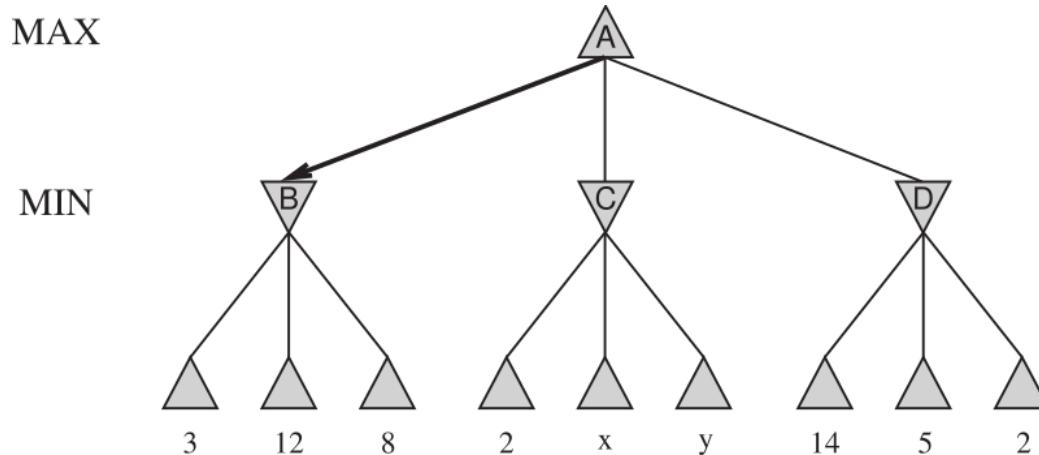
- **Completeness:**
 - Yes, if tree is finite.
- **Optimality:**
 - Yes, if MIN is an optimal opponent.
 - What if MIN is suboptimal?
 - Show that MAX will do even better.
 - What if MIN is suboptimal and predictable?
 - Other strategies might do better than Minimax. However they will do worse on an optimal opponent.

Minimax efficiency

- Assume $\text{minimax}(s)$ is implemented using its recursive definition.
- How efficient is minimax?
 - Time complexity: same as DFS, i.e., $O(b^m)$.
 - Space complexity:
 - $O(bm)$, if all actions are generated at once, or
 - $O(m)$, if actions are generated one at a time.

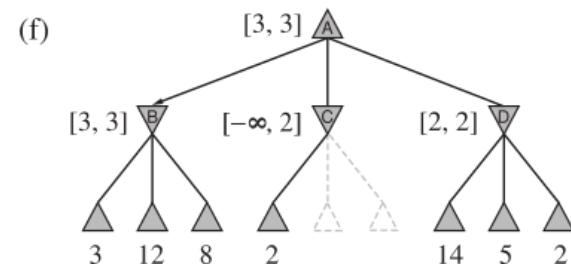
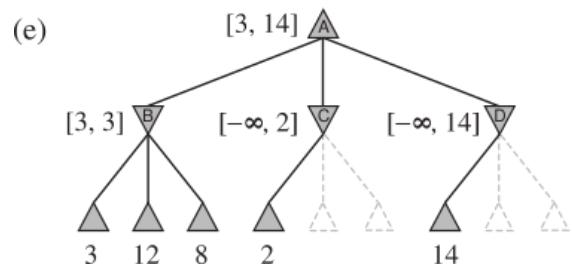
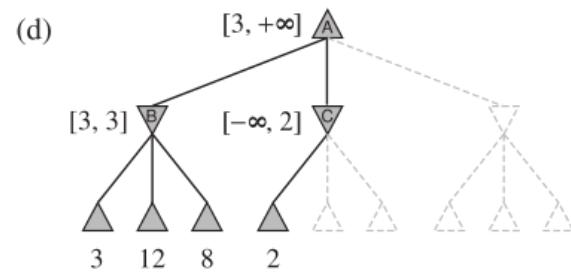
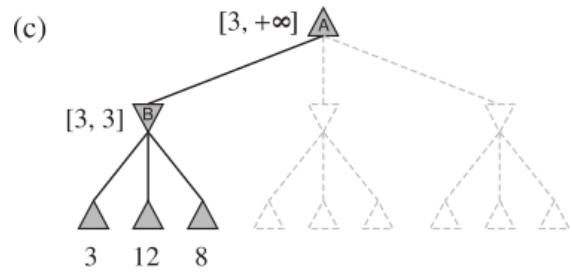
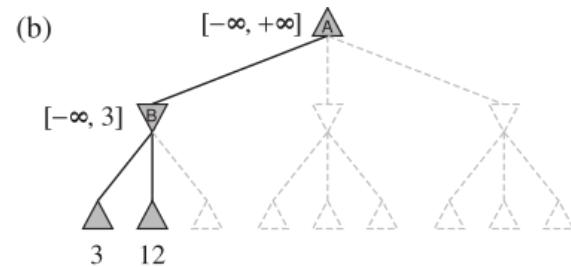
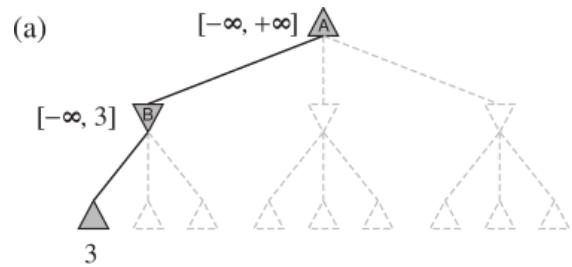
[Q] Do we need to explore the whole game tree?

Pruning



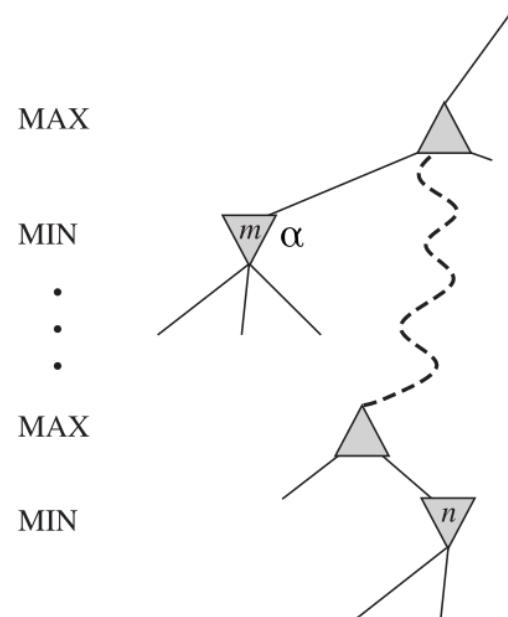
$$\begin{aligned}\text{MINIMAX}(\text{root}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \quad \text{where } z = \min(2, x, y) \leq 2 \\ &= 3.\end{aligned}$$

Therefore, it is possible to compute the **correct** minimax decision **without looking at every node** in the tree.



We want to compute $v = \text{minimax}(n)$, for player(n)=MIN.

- We loop over n 's children.
- The minimax values are being computed one at a time and v is updated iteratively.
- Let α be the best value (i.e., the highest) at any choice point along the path for MAX.
- If v becomes lower than α , then n will never be reached in actual play.
- Therefore, we can stop iterating over the remaining n 's other children.



Similarly, β is defined as the best value (i.e., lowest) at any choice point along the path for MIN. We can halt the expansion of a MAX node as soon as v becomes larger than β .

α - β pruning

- Updates the values of α and β as the path is expanded.
- Prune the remaining branches (i.e., terminate the recursive calls) as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively.

α - β search

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow$   $-\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MAX(v, MIN-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\geq$   $\beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v
```

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow$   $+\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq$   $\alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

Properties of α - β search

- Pruning has **no effect** on the minimax values. Therefore, **completeness** and **optimality** are preserved from Minimax.
- **Time complexity:**
 - The effectiveness depends on the order in which the states are examined.
 - If states could be examined in **perfect order**, then $\alpha - \beta$ search examines only $O(b^{m/2})$ nodes to pick the best move, vs. $O(b^m)$ for minimax.
 - $\alpha - \beta$ can solve a tree twice as deep as minimax can in the same amount of time.
 - Equivalent to an effective branching factor \sqrt{b} .
- **Space complexity:** $O(m)$, as for Minimax.

Game tree size



Chess:

- $b \approx 35$ (approximate average branching factor)
- $d \approx 100$ (depth of a game tree for typical games)
- $b^d \approx 35^{100} \approx 10^{154}$.
- For $\alpha - \beta$ search and perfect ordering, we get $b^{d/2} \approx 35^{50} = 10^{77}$.

Finding the exact solution is completely **infeasible**.

Imperfect real-time decisions

- Under **time constraints**, searching for the exact solution is not feasible in most realistic games.
- Solution: cut the search earlier.
 - Replace the **utility(s)** function with a heuristic **evaluation function eval(s)** that estimates the state utility.
 - Replace the terminal test by a **cutoff test** that decides when to stop expanding a state.

H-MINIMAX(s, d) =

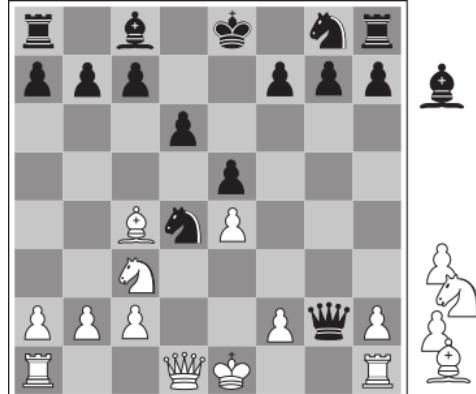
$$\begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s, d) \\ \max_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if } \text{PLAYER}(s) = \text{MIN}. \end{cases}$$

[Q] Can $\alpha - \beta$ search be adapted to implement H-Minimax?

Evaluation functions

- An evaluation function $\text{eval}(s)$ returns an **estimate** of the expected utility of the game from a given position s .
- The computation **must be short** (that is the whole point to search faster).
- Ideally, the evaluation should **order** terminal states in the same way as in Minimax.
 - The evaluation values may be different from the true minimax values, as long as order is preserved.
- In non-terminal states, the evaluation function should be strongly **correlated with the actual chances of winning**.
- Like for heuristics in search, evaluation functions can be **learned** using machine learning algorithms.

Quiescence



(a) White to move



(b) White to move

- These states only differ in the position of the rook at lower right.
- However, Black has advantage in (a), but not in (b).
- If the search stops in (b), Black will not see that White's next move is to capture its Queen, gaining advantage.
- Cutoff should only be applied to positions that are **quiescent**.
 - i.e., states that are unlikely to exhibit wild swings in value in the near future.

The horizon effect

- Evaluations functions are **always imperfect**.
- Often, the deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters.
- If not looked deep enough, **bad moves may appear as good moves** (as estimated by the evaluation function) because their consequences are hidden beyond the search horizon.
 - and vice-versa!



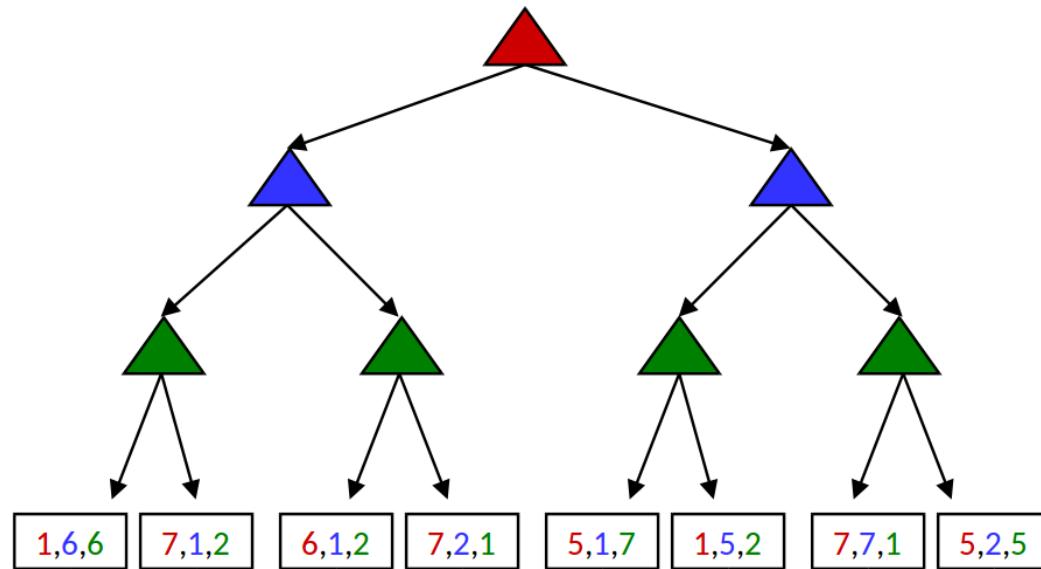
Cutoff at depth 2, evaluation = the closer to the dot, the better.



Cutoff at depth 10, evaluation = the closer to the dot, the better.

Multi-agent utilities

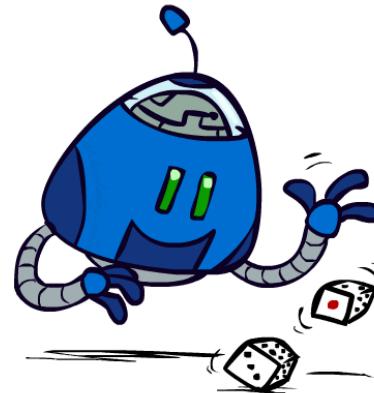
- What if the game is not zero-sum, or has **multiple players**?
- Generalization of Minimax:
 - Terminal states are labeled with utility **tuples** (1 value per player).
 - Intermediate states are also labeled with utility tuples.
 - Each player maximizes its own component.
 - May give rise to cooperation and competition dynamically



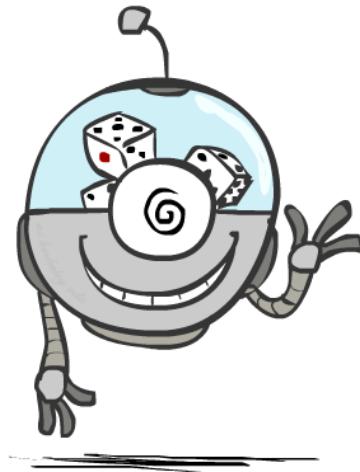
Stochastic games

Stochastic games

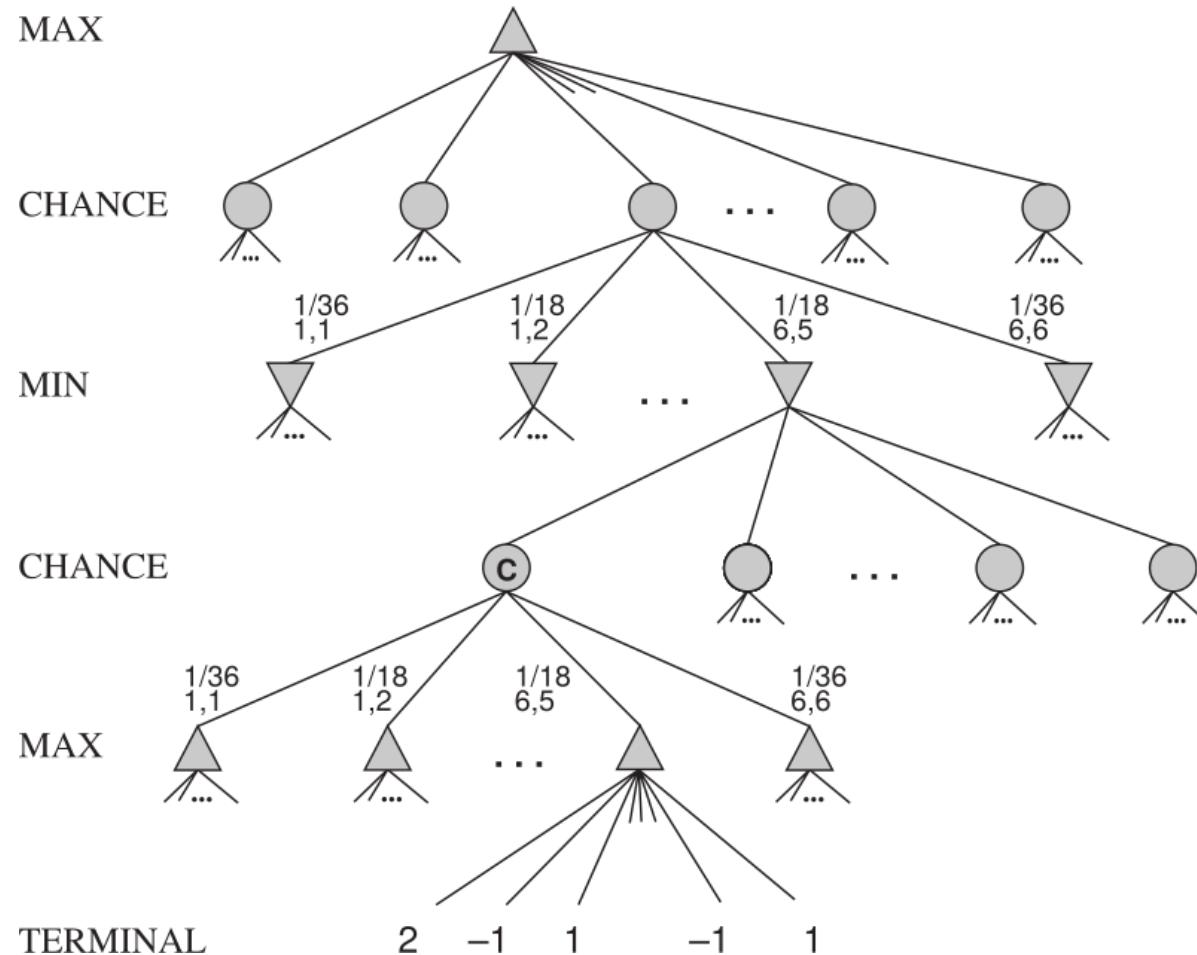
- In real life, many unpredictable external events can put us into unforeseen situations.
- Games that mirror this unpredictability are called **stochastic games**. They include a random element, such as:
 - explicit randomness: rolling a dice;
 - unpredictable opponents: ghosts respond randomly;
 - actions may fail: when moving a robot, wheels might slip.



- In a game tree, this random element can be **modeled** with **chance nodes** that map a state-action pair to the set of possible outcomes, along with their respective **probability**.
- This is equivalent to considering the environment as an extra **random agent** player that moves after each of the other players.



Stochastic game tree



[Q] What is the best move?

Expectiminimax

- Because of the uncertainty in the action outcomes, states no longer have a **definite minimax** value.
- We can only calculate the **expected** value of a state under optimal play by the opponent.
 - i.e., the average over all possible outcomes of the chance nodes.
 - **minimax** values correspond instead to the worst-case outcome.

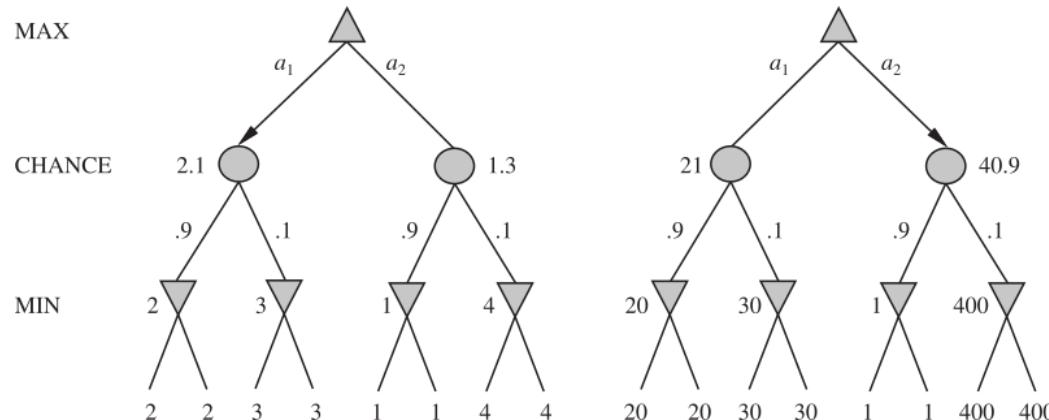
$\text{EXPECTIMINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\ \sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE} \end{cases}$$

[Q] Does taking the rational move mean the agent will be successful?

Evaluation functions

- As for $\text{minimax}(n)$, the value of $\text{expectiminimax}(n)$ may be approximated by stopping the recursion early and using an evaluation function.
- However, to obtain correct move, the evaluation function should be a **positive linear transformation** of the expected utility of the state.
 - It is not enough for the evaluation function to just be order-preserving.
- If we assume bounds on the utility function, $\alpha - \beta$ search can be adapted to stochastic games.



An order-preserving transformation on leaf values changes the best move.

Monte Carlo Tree Search

Random playout evaluation

- To evaluate a state, have the algorithm play **against itself** using **random moves**, thousands of times.
- The sequence of random moves is called a **random playout**.
- Use the proportion of wins as the state evaluation.
- This strategy does **not require domain knowledge!**
 - The game engine is all that is needed.

Monte Carlo Tree Search

The focus of MCTS is the analysis of the most promising moves, as incrementally evaluated with random playouts.

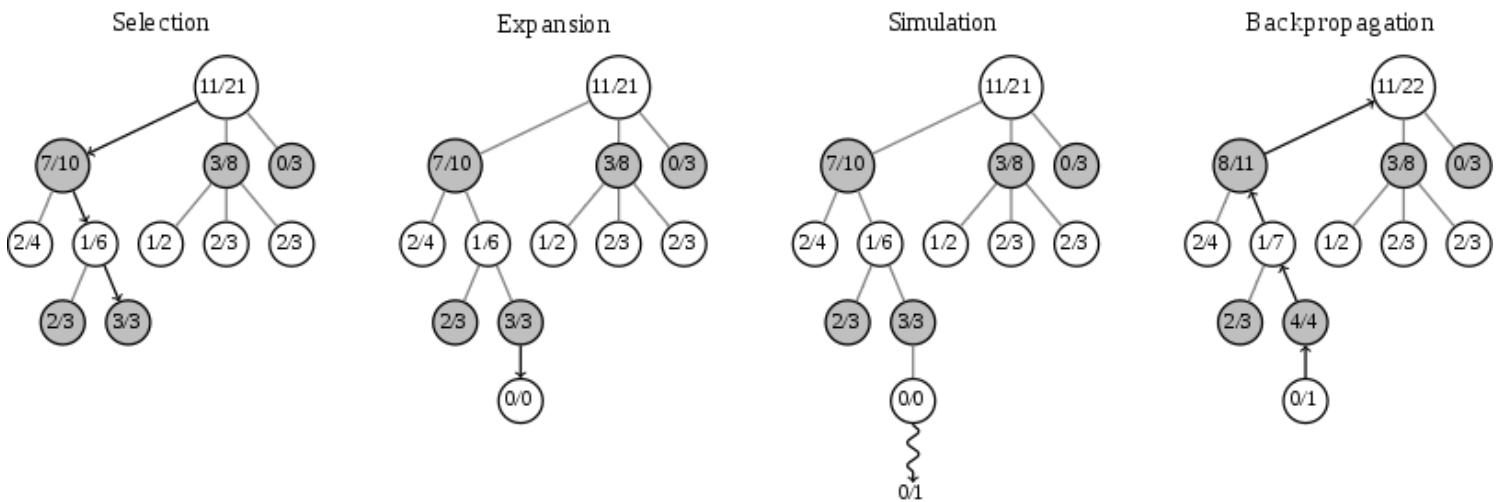
Each node n in the current search tree maintains two values:

- the number of wins $Q(n, p)$ of player p for all playouts that passed through n ;
- the number $N(n)$ of times it has been visited.

The algorithm searches the game tree as follows:

1. **Selection**: start from root, select successive child nodes down to a node n that is not fully expanded.
2. **Expansion**: unless n is a terminal state, create a new child node n' .
3. **Simulation**: play a random playout from n' .
4. **Backpropagation**: use the result of the playout to update information in the nodes on the path from n' to the root.

Repeat 1-4 for as long the time budget allows. Pick the best next direct move.



Black is about to move. Which action should it take?

Exploration and exploitation

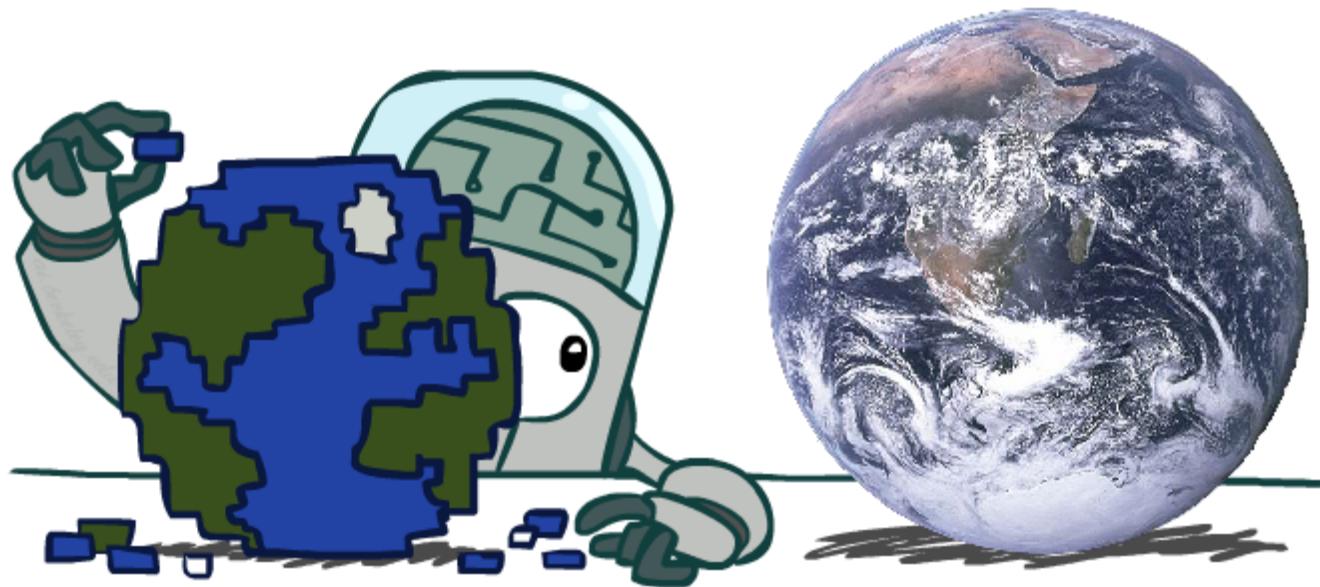
Given a limited budget of random playouts, the efficiency of MCTS critically depends on the choice of the nodes that are selected at step 1.

At a node n during the selection step, the UCB1 policy picks the child node n' of n that maximizes

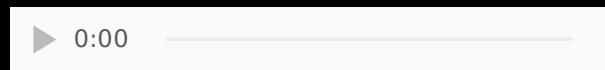
$$\frac{Q(n', p)}{N(n')} + c \sqrt{\frac{2 \log N(n)}{N(n')}}.$$

- The first term encourages the **exploitation** of higher-reward nodes.
- The second term encourages the **exploration** of less-visited nodes.
- The constant $c > 0$ controls the trade-off between exploitation and exploration.

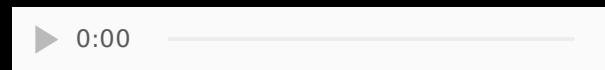
Modeling assumptions



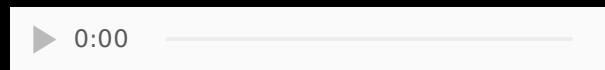
What if our assumptions are incorrect?



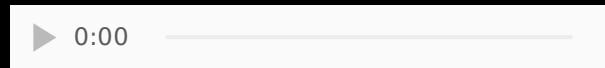
Minimax Pacman vs. Adversarial ghost



Minimax Pacman vs. Random ghost



Expectiminimax Pacman vs. Random ghost



Expectiminimax Pacman vs. Adversarial ghost

State-of-the-art game programs

Checkers

1951

First computer player by Christopher Strachey.

1994

The computer program [Chinook](#) ends the 40-year-reign of human champion Marion Tinsley.

- Library of opening moves from grandmasters;
- A deep search algorithm;
- A good move evaluation function (based on a linear model);
- A database for all positions with eight pieces or fewer.

2007

Checkers is [solved](#). A weak solution is computationally proven.

- The number of involved calculations was 10^{14} , over a period of 18 years.
- The best an optimal player can achieve against Chinook is a draw.

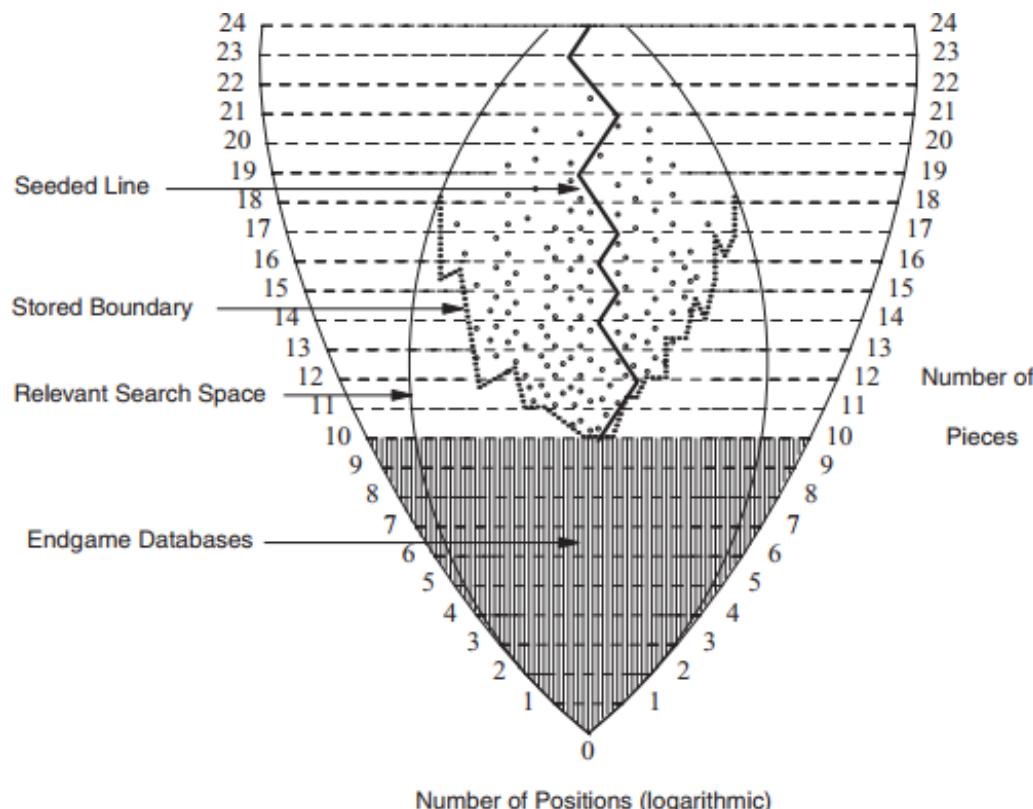


Fig. 2. Forward and backward search. The number of pieces on the board are plotted (vertically) versus the logarithm of the number of positions (Table 1). The shaded area shows the endgame database part of the proof—i.e., all positions with ≤ 10 pieces. The inner oval area shows that only a portion of the search space is relevant to the proof. Positions may be irrelevant because they are unreachable or are not required for the proof. The small open circles indicate positions with more than 10 pieces for which a value has been proven by a solver. The dotted line shows the boundary between the top of the proof tree that the manager sees (and stores on disk) and the parts that are computed by the solvers (and are not saved in order to reduce disk storage needs). The solid seeded line shows a “best” sequence of moves.

Chess

1997

Deep Blue defeats human champion Gary Kasparov.

- 200000000 position evaluations per second.
- Very sophisticated evaluation function.
- Undisclosed methods for extending some lines of search up to 40 plies.

Modern programs (e.g., Stockfish) are better, if less historic. Chess remains [unsolved](#) due to the complexity of the game.



Go

For long, Go was considered as the Holy Grail of AI due to the size of its game tree.

- On a 19x19, the number of legal positions is $\pm 2 \times 10^{170}$.
- This results in $\pm 10^{800}$ games, considering a length of 400 or less.



2010-2014

Using Monte Carlo tree search and machine learning, computer players reach low dan levels.

2015-2017

Google Deepmind invents AlphaGo.

- 2015: AlphaGo beat Fan Hui, the European Go Champion.
- 2016: AlphaGo beat Lee Sedol (4-1), a 9-dan grandmaster.
- 2017: AlphaGo beat Ke Jie, 1st world human player.

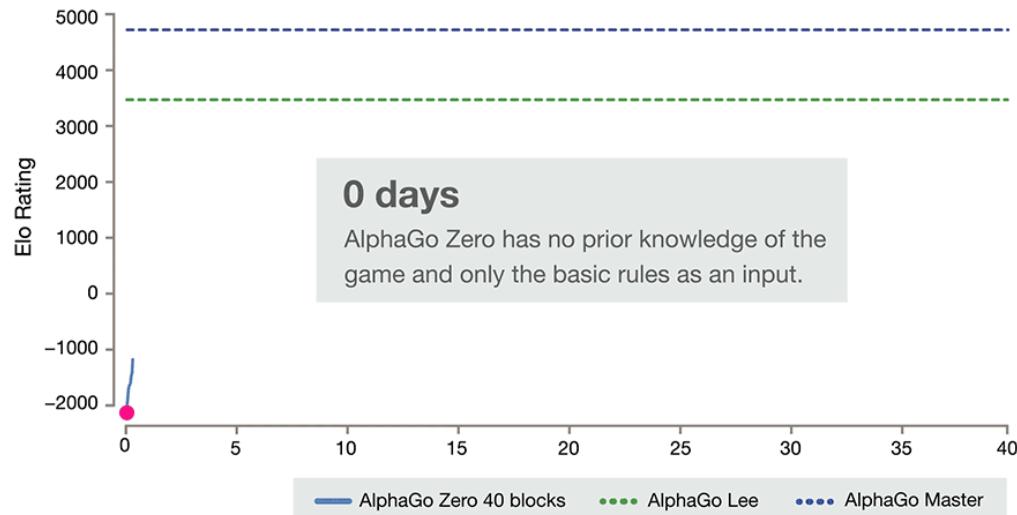
AlphaGo combines Monte Carlo tree search and deep learning with extensive training, both from human and computer play.



Press coverage for the victory of AlphaGo against Lee Sedol.

2017

AlphaGo Zero combines [Monte Carlo tree search](#) and [deep learning](#) with extensive training, with [self-play only](#).



Summary

- Multi-player games are variants of search problems.
- The difficulty rise in the fact that opponents may respond arbitrarily.
 - The optimal solution is a **strategy**, and not a fixed sequence of actions.
- **Minimax** is an optimal algorithm for deterministic, turn-taking, two-player zero-sum game with perfect information.
 - Due to practical time constraints, exploring the whole game tree is often **infeasible**.
 - Approximations can be achieved with heuristics, reducing computing times.
 - Minimax can be adapted to stochastic games.
 - Minimax can be adapted to games with more than 2 players.
- Optimal behavior is **relative** and depends on the assumptions we make about the world.

References

- Browne, Cameron B., et al. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012): 1-43.
- Schaeffer, Jonathan, et al. "Checkers is solved." science 317.5844 (2007): 1518-1522.
- Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

Introduction to Artificial Intelligence

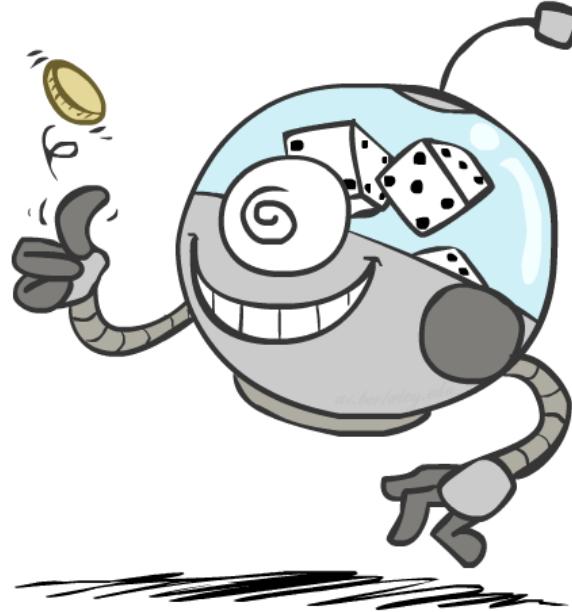
Lecture 5: Representing uncertain knowledge

Prof. Gilles Louppe
g.louuppe@uliege.be



Today

- Probability:
 - Random variables
 - Joint and marginal distributions
 - Conditional distributions
 - Product rule, Chain rule, Bayes' rule
 - Inference
- Bayesian networks:
 - Representing uncertain knowledge
 - Semantics
 - Construction



Do not overlook this lecture!

You will need these concepts a lot. (Now and in future courses)

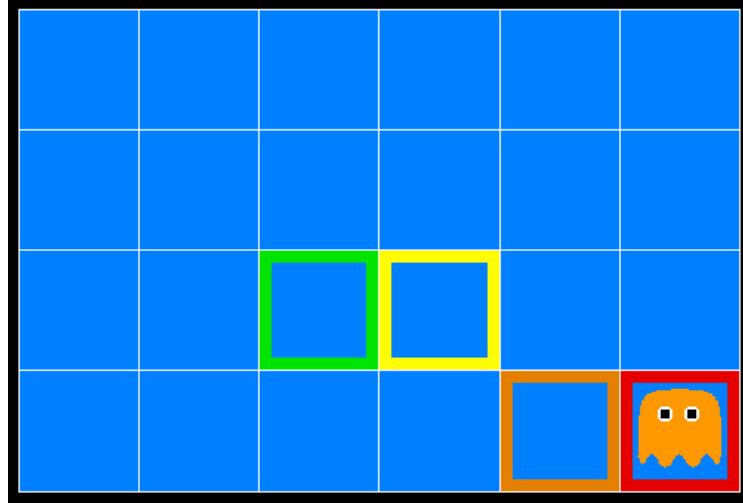
Quantifying uncertainty

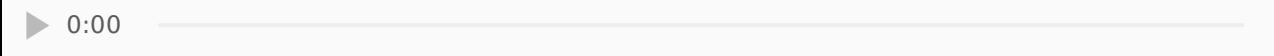
Ghostbusters

A ghost is **hidden** in the grid somewhere. Sensor readings tell how close a square is to the ghost.

- On the ghost: red
- 1 or 2 away: orange
- 3 away: yellow
- 4+ away green

Sensors are **noisy**, but we know the probability $P(\text{color}|\text{distance})$ for all colors and all distances.





[Q] Could we use a logical agent for this game?

Uncertainty

General situation:

- **Observed variables** (evidence): agent knows certain things about the state of the world (e.g., sensor readings).
- **Unobserved variables**: agent needs to reason about other aspects that are **uncertain** (e.g., where the ghost is).
- (Probabilistic) **model**: agent knows or believes something about how the known variables relate to the unknown variables.

How to handle uncertainty?

- A purely logical approach either:
 - risks falsehood (because of ignorance about the world or laziness in the model), or
 - leads to conclusions that are too weak for decision making.
- **Probabilistic reasoning** provides a framework for managing our knowledge and **beliefs**.

Probability

- Probabilistic assertions express the agent's inability to reach a definite decision regarding the truth of a proposition.
- Probability values **summarize** effects of
 - **laziness** (failure to enumerate all world states)
 - **ignorance** (lack of relevant facts, initial conditions, correct model, etc).
- (Bayesian subjective) Probabilities relate propositions to one's own state of knowledge (or lack thereof).
 - e.g., $P(\text{ghost in cell } [3, 2]) = 0.02$
- These are **not** claims of a "probabilistic tendency" in the current situation (but might be learned from past experience of similar situations).

Kolmogorov's probability theory

Begin with a set Ω , the sample space.

$\omega \in \Omega$ is a sample point, possible world or atomic event.

A probability space is a sample space equipped with an assignment $P : \mathcal{P}(\Omega) \rightarrow \mathbb{R}$ such that:

- 1st axiom: $P(\omega) \in \mathbb{R}, 0 \leq P(\omega)$ for all $\omega \in \Omega$.
- 2nd axiom: $P(\Omega) = 1$.
- 3rd axiom: $P(\{\omega_1, \dots, \omega_n\}) = \sum_{i=1}^n P(\omega_i)$ for any set of samples.

where $\mathcal{P}(\Omega)$ the power set of Ω .

Example

- Ω = the 6 possible rolls of a die.
- ω_i (for $i = 1, \dots, 6$) are the sample points, each corresponding than an outcome of the die.
- Assignment P for a fair die:

$$P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = \frac{1}{6}$$

Random variables

- A **random variable** is a function $X : \Omega \rightarrow D_X$ from the sample space to some domain defining its **outcomes**.
 - e.g., $Odd : \Omega \rightarrow \{\text{true, false}\}$ such that $Odd(\omega) = (\omega \% 2 = 1)$.
- P induces a **probability distribution** for any random variable X .
 - $P(X = x_i) = \sum_{\{\omega : X(\omega) = x_i\}} P(\omega)$
 - e.g., $P(Odd = \text{true}) = P(1) + P(3) + P(5) = \frac{1}{2}$.
- In practice, we will use random variables to represent aspects of the world about which we (may) have uncertainty.
 - R : Is it raining?
 - T : Is it hot or cold?
 - L : Where is the ghost?

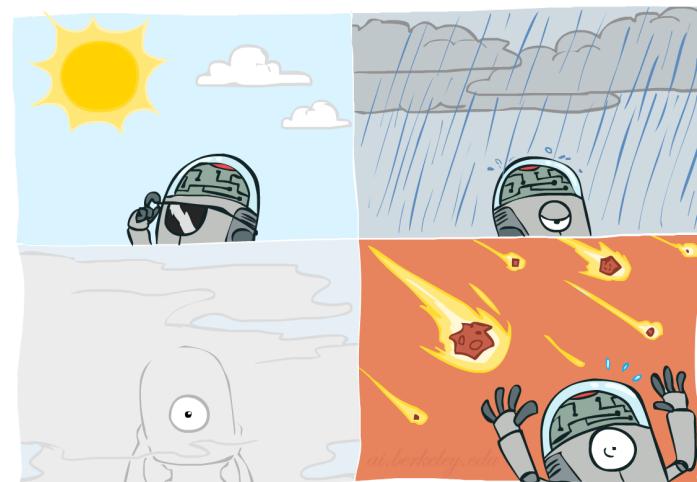
Probability distributions

For discrete variables, the **probability distribution** can be encoded by a discrete list of the probabilities of the outcomes, known as the **probability mass function**.

One can think of the probability distribution as a **table** that associates a probability value to each **outcome** of the variable.

- This table can be infinite!
- By construction, probability values are **normalized** (i.e., sum to 1).

$P(W)$	
W	P
sun	0.6
rain	0.1
fog	0.3
meteor	0.0



Notations

- Random variables are written in upper roman letters: X, Y , etc.
- Realizations of a random variable are written in corresponding lower case letters. E.g., x_1, x_2, \dots, x_n could be of outcomes of the random variable X .
- The probability value of the realization x is written as $P(X = x)$.
- When clear from context, this will be abbreviated as $P(x)$.
- The probability distribution of the random variable X is denoted as $P(X)$.
This corresponds e.g. to a full probability table (and not to a single scalar value!).

Joint distributions

A **joint probability distribution** over a set of random variables X_1, \dots, X_n specifies the probability of each (combined) outcome:

$$P(X_1 = x_1, \dots, X_n = x_n) = \sum_{\{\omega : X_1(\omega) = x_1, \dots, X_n(\omega) = x_n\}} P(\omega)$$

Example

$$P(T, W)$$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

Events

An **event** is a set E of outcomes.

- $P(E) = \sum_{(x_1, \dots, x_n) \in E} P(X_1 = x_1, \dots, X_n = x_n)$

From a joint distribution, the probability of **any event** can be calculated.

- Probability that it is hot and sunny?
- Probability that it is hot?
- Probability that it is hot or sunny?

Interesting events often correspond to **partial assignments**.

- e.g., $P(T = \text{hot})$

Marginal distributions

The **marginal distribution** of a subset of a collection of random variables is the joint probability distribution of the variables contained in the subset.

$$P(T, W)$$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$P(T)$$

T	P
hot	0.5
cold	0.5

$$P(W)$$

W	P
sun	0.6
rain	0.4

$$P(t) = \sum_w P(t, w) \quad P(w) = \sum_t P(t, w)$$

Intuitively, marginal distributions are sub-tables which eliminate variables.

[Q] To what events are marginal probabilities associated?

Conditional probability

Prior or unconditional probabilities of an event correspond to some initial belief, prior to arrival of any evidence.

- e.g., $P(W = \text{sun}) = 0.6$.

Posterior or conditional probabilities correspond to the probability of an event, given some observed evidence. Formally,

$$P(A = a | B = b) = \frac{P(A = a, B = b)}{P(B = b)}.$$

Example

$$P(W = \text{sun} | T = \text{cold}) = \frac{P(W = \text{sun}, T = \text{cold})}{P(T = \text{cold})} = \frac{0.2}{0.2 + 0.3} = 0.4$$

Conditional distributions

Conditional distributions are probability distributions over some variables, given **fixed** values for others.

$$P(T, W)$$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$P(W|T = \text{hot})$$

T	P
sun	0.8
rain	0.2

$$P(W|T = \text{cold})$$

W	P
sun	0.4
rain	0.6

[Q] To what events are conditional probabilities associated?

[Q] Is a conditional distribution defined on the same sample and probability space than the joint?

Normalization trick

$$P(T, W)$$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$\rightarrow P(T = \text{cold}, W)$$

T	W	P
cold	sun	0.2
cold	rain	0.3

$$\rightarrow P(W|T = \text{cold})$$

W	P
sun	0.4
rain	0.6

Select the joint probabilities matching the evidence $T = \text{cold}$.

Normalize the selection (make it sum to 1).

[Q] Why does this work?

Probabilistic inference

Probabilistic **inference** is the problem of computing a desired probability from other known probabilities (e.g., conditional from joint).

- We generally compute conditional probabilities.
 - e.g., $P(\text{on time}|\text{no reported accidents}) = 0.9$
 - These represent the agent's **beliefs** given the evidence.
- Probabilities change with new evidence:
 - e.g., $P(\text{on time}|\text{no reported accidents, 5AM}) = 0.95$
 - e.g., $P(\text{on time}|\text{no reported accidents, rain}) = 0.8$
 - e.g., $P(\text{ghost in [3, 2]}|\text{red in [3, 2]}) = 0.99$
 - Observing new evidence causes **beliefs to be updated**.



General case

- Evidence variables: $E_1, \dots, E_k = e_1, \dots, e_k$
- Query variables: Q
- Hidden variables: H_1, \dots, H_r
- $(Q \cup E_1, \dots, E_k \cup H_1, \dots, H_r) = \text{all variables } X_1, \dots, X_n$

Inference is the problem of computing $P(Q|e_1, \dots, e_k)$.

Inference by enumeration

Start from the joint distribution $P(Q, E_1, \dots, E_k, H_1, \dots, H_r)$.

1. Select the entries consistent with the evidence $E_1, \dots, E_k = e_1, \dots, e_k$.
2. Marginalize out the hidden variables to obtain the joint of the query and the evidence variables:

$$P(Q, e_1, \dots, e_k) = \sum_{h_1, \dots, h_r} P(Q, h_1, \dots, h_r, e_1, \dots, e_k).$$

3. Normalize:

$$\begin{aligned} Z &= \sum_q P(q, e_1, \dots, e_k) \\ P(Q|e_1, \dots, e_k) &= \frac{1}{Z} P(Q, e_1, \dots, e_k) \end{aligned}$$

Example

- $P(W)$?
- $P(W|\text{winter})$?
- $P(W|\text{winter, hot})$?

S	T	W	P
summer	hot	sun	0.3
summer	hot	rain	0.05
summer	cold	sun	0.1
summer	cold	rain	0.05
winter	hot	sun	0.1
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.2

Complexity

- Inference by enumeration can be used to answer probabilistic queries for **discrete variables** (i.e., with a finite number of values).
- However, enumeration **does not scale!**
 - Assume a domain described by n variables taking at most d values.
 - Space complexity: $O(d^n)$
 - Time complexity: $O(d^n)$
- Can we reduce the size of the representation of the joint distribution?

The product rule

$$P(b)P(a|b) = P(a, b)$$

Example

$$P(W)$$

W	P
sun	0.8
rain	0.2

$$P(D|W)$$

D	W	P
wet	sun	0.1
dry	sun	0.9
wet	rain	0.7
dry	rain	0.3

$$P(D, W)$$

D	W	P
wet	sun	?
dry	sun	?
wet	rain	?
dry	rain	?

The chain rule

More generally, any joint distribution can always be written as an incremental product of conditional distributions:

$$P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$$

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i|x_1, \dots, x_{i-1})$$

[Q] Why is this always true?

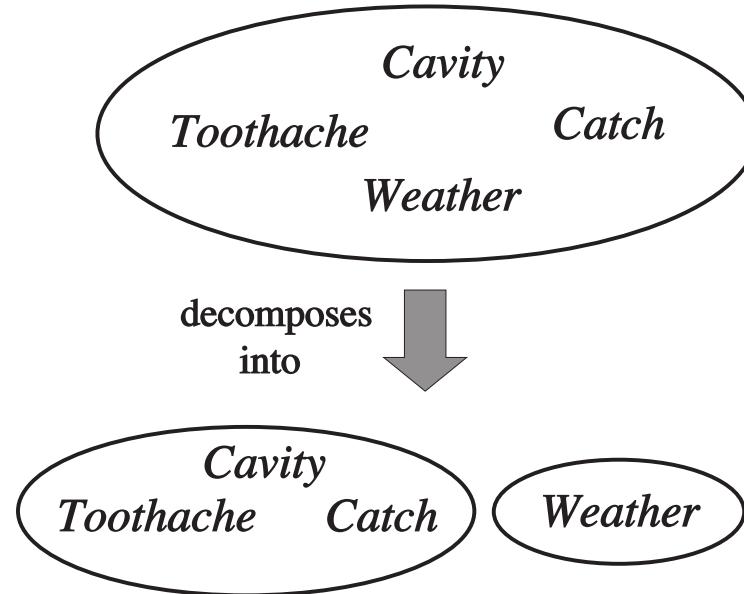
Independence

A and B are **independent** iff, for all $a \in D_A$ and $b \in D_B$,

- $P(a|b) = P(a)$, or
- $P(b|a) = P(b)$, or
- $P(a, b) = P(a)P(b)$

Independence is also written as $A \perp B$.

Example 1



$$\begin{aligned} & P(\text{toothache}, \text{catch}, \text{cavity}, \text{weather}) \\ & = P(\text{toothache}, \text{catch}, \text{cavity})P(\text{weather}) \end{aligned}$$

The original 32-entry table reduces to one 8-entry and one 4-entry table (assuming 4 values for **Weather** and boolean values otherwise).

Example 2

For n independent coin flips, the joint distribution can be fully factored and represented as the product of n 1-entry tables.

- $2^n \rightarrow n$

[Q] In practice, absolute independence is rare. What can we do?

Conditional independence

A and B are **conditionally independent** given C iff, for all $a \in D_A, b \in D_B$ and $c \in D_C$,

- $P(a|b, c) = P(a|c)$, or
- $P(b|a, c) = P(b|c)$, or
- $P(a, b|c) = P(a|c)P(b|c)$

Conditional independence is also written as $A \perp B|C$.

- Using the chain rule, the join distribution can be factored as a product of conditional distributions.
- Each conditional distribution may potentially be **simplified by conditional independence**.
- Conditional independence assertions allow probabilistic models to **scale up**.

Example 1

- Assume three random variables **Toothache**, **Catch** and **Cavity**.
- **Catch** is conditionally independent of **Toothache**, given **Cavity**.

Therefore, we can write:

$$\begin{aligned} & P(\text{toothache, catch, cavity}) \\ &= P(\text{toothache}|\text{catch, cavity})P(\text{catch}|\text{cavity})P(\text{cavity}) \\ &= P(\text{toothache}|\text{cavity})P(\text{catch}|\text{cavity})P(\text{cavity}) \end{aligned}$$

In this case, the representation of the joint distribution reduces to $2 + 2 + 1$ independent numbers (instead of $2^n - 1$).

Example 2 (Naive Bayes)

- More generally, from the product rule, we have:
 - $P(\text{cause}, \text{effect}_1, \dots, \text{effect}_n) = P(\text{effect}_1, \dots, \text{effect}_n | \text{cause})P(\text{cause})$
- Assuming pairwise conditional independence between the effects given the cause, it comes:
 - $P(\text{cause}, \text{effect}_1, \dots, \text{effect}_n) = P(\text{cause}) \prod_i P(\text{effect}_i | \text{cause})$
- This probabilistic model is called a **naive Bayes** model.
 - The complexity of this model is $O(n)$ instead of $O(2^n)$ without the conditional independence assumptions.
 - Naive Bayes can work surprisingly well in practice, even when the assumptions are wrong.

The next slide is **super** important.

The Bayes' rule

The product rule defines two ways to factor the joint distribution of two random variables.

$$P(a, b) = P(a|b)P(b) = P(b|a)P(a)$$

Therefore,

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}.$$



- $P(a)$ is the prior belief on a .
- $P(b)$ is the probability of the evidence b .
- $P(a|b)$ is the posterior belief on a , given the evidence b .
- $P(b|a)$ is the conditional probability of b given a . Depending on the context, this term is called the likelihood.

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

Why is this helpful?

- The Bayes rule let us build one conditional from its reverse.
- Often one conditional is tricky, but the other is simple.

This equation is the **foundation of many AI systems**.

Example: diagnostic probability from causal probability.

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}$$

- $P(\text{effect}|\text{cause})$ quantifies the relationship in the **causal direction**.
- $P(\text{cause}|\text{effect})$ describes the **diagnostic direction**.

Let S =stiff neck and M =meningitis. Given

- $P(s|m) = 0.7$
- $P(m) = 1/50000$
- $P(s) = 0.01,$

it comes

- $P(m|s) = \frac{P(s|m)P(m)}{P(s)} = \frac{0.7 \times 1/50000}{0.01} = 0.0014.$

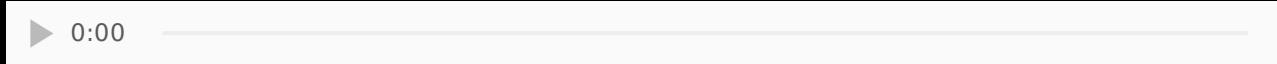
Ghostbusters, revisited

- Let assume a random variable G for the ghost location and a set of random variables $R_{i,j}$ for the individual readings.
- We define a **prior distribution** $P(G)$ over ghost locations.
 - Assume it is uniform.
- We assume a sensor **reading model** $P(R_{i,j}|G)$.
 - That is, we know what the sensors do.
 - $R_{i,j}$ = reading color measured at $[i, j]$
 - e.g., $P(R_{1,1} = \text{yellow}|G = [1, 1]) = 0.1$
 - Two readings are conditionally independent, given the ghost position.

- We can calculate the **posterior distribution** $P(G|R_{i,j})$ using Bayes' rule:

$$P(G|R_{i,j}) = \frac{P(R_{i,j}|G)P(G)}{P(R_{i,j})}.$$

- For the next reading $R_{i',j'}$, this posterior distribution becomes the prior distribution over ghost locations, which we update similarly.



[Q] What if we had chosen a different prior?

Frequentism vs. Bayesianism

What do probability values represent?

- The objectivist **frequentist** view is that probabilities are real aspects of the universe.
 - i.e., propensities of objects to behave in certain ways.
 - e.g., the fact that a fair coin comes up heads with probability **0.5** is a propensity of the coin itself.
- The subjectivist **Bayesian** view is that probabilities are a way of characterizing an agent's beliefs or uncertainty.
 - i.e., probabilities do not have external physical significance.
 - **This is the interpretation of probabilities that we will use!**

Probabilistic reasoning

Representing knowledge

- The joint probability distribution can answer any question about the domain.
- However, its representation can become **intractably large** as the number of variable grows.
- **Independence** and **conditional independence** reduce the number of probabilities that need to be specified in order to define the full joint distribution.
- These relationships can be represented explicitly in the form of a **Bayesian network**.

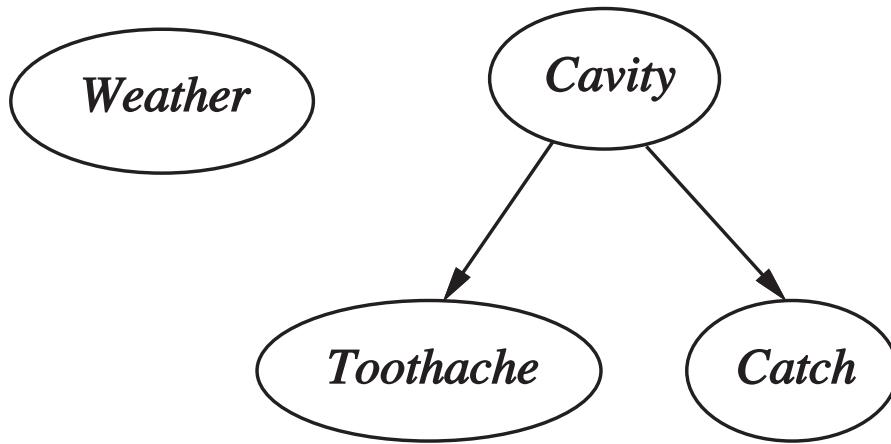
Bayesian networks

A **Bayesian network** is a **directed graph** in which:

- Each **node** corresponds to a **random variable**.
 - Can be observed or unobserved.
 - Can be discrete or continuous.
- Each **edge** indicates "direct influence" between variables.
 - If there is an arrow from node X to node Y , X is said to be a **parent** of Y .
 - The graph has no directed cycle (i.e., the graph is a DAG).
- Each node X_i is annotated with a **conditional probability distribution** $P(X_i|\text{parents}(X_i))$ that quantifies the effect of the parents on the node.
 - In the simplest case, conditional distributions are represented as conditional probability table (CTP).

Example 1

The topology of the network encodes conditional independence assertions:

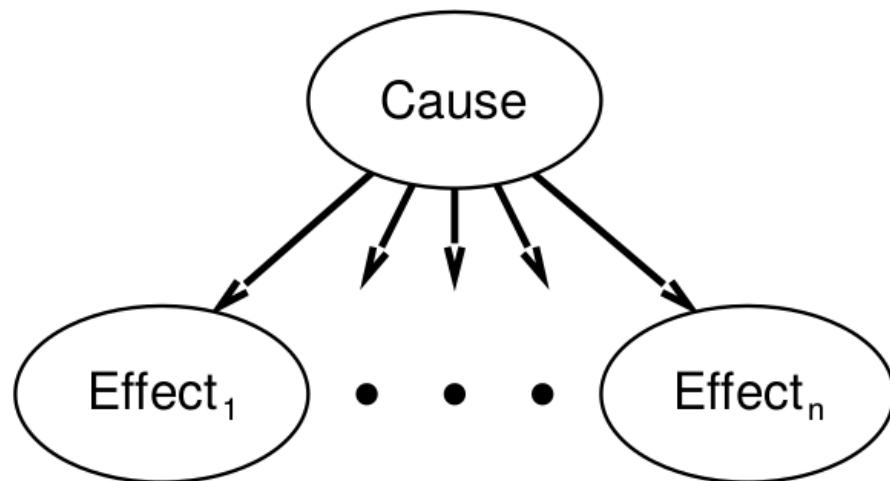


- **Weather** is independent of the other variables.
- **Toothache** and **Catch** are conditionally independent given **Cavity**.

[Q] What is the BN for n independent coin flips?

Example 2 (Naive Bayes)

In a Naive Bayes model, we assume pairwise conditional independence between the effects given the cause:

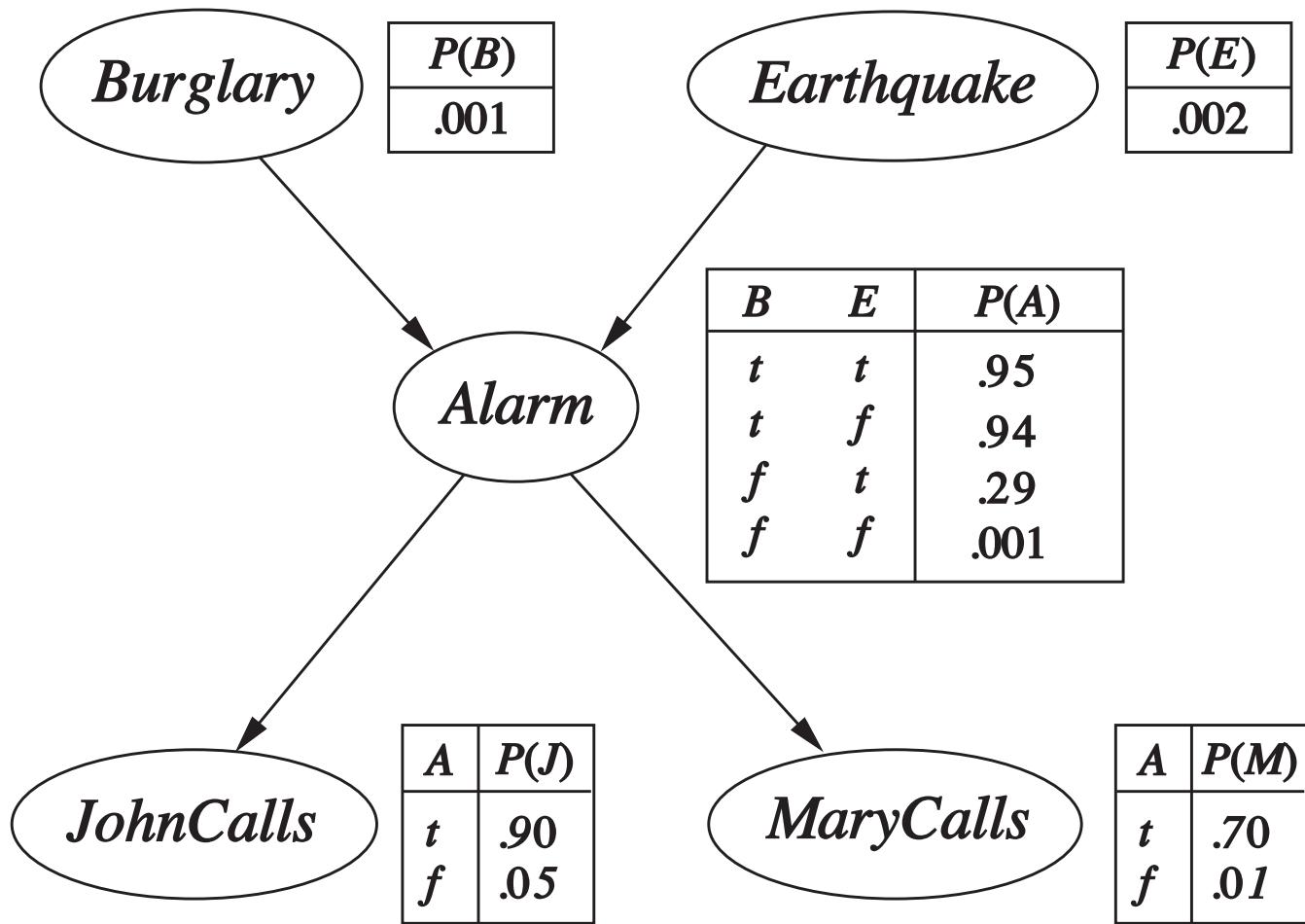


Example 3



I am at work, neighbor John calls to say my alarm is ringing, but neighbor Mary does not call. Sometimes it's set off by minor earthquakes. Is there a burglar?

- Variables: **Burglar**, **Earthquake**, **Alarm**, **JohnCalls**, **MaryCalls**.
- Network topology from "causal" knowledge:
 - A burglar can set the alarm off
 - An earthquake can set the alarm off
 - The alarm can cause Mary to call
 - The alarm can cause John to call



Semantics

A Bayesian network implicitly **encodes** the full joint distribution as the product of the local distributions:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$$

Example

$$\begin{aligned} & P(j, m, a, \neg b, \neg e) \\ &= P(j|a)P(m|a)P(a|\neg b, \neg e)P(\neg b)P(\neg e) \\ &= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \\ &\approx 0.00063 \end{aligned}$$

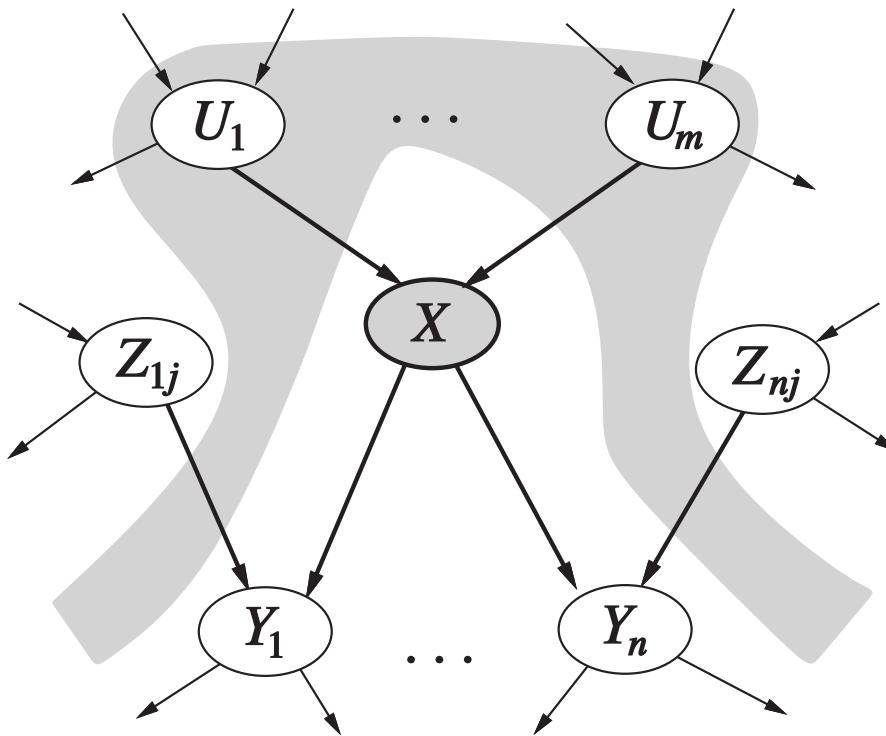
Why does $\prod_{i=1}^n P(x_i | \text{parents}(X_i))$ result in a proper joint distribution?

- By the **chain rule**, $P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | x_1, \dots, x_{i-1})$.
- Assume **conditional independencies** of X_i with its predecessors in the ordering given the parents, and provided $\text{parents}(X_i) \subseteq \{X_1, \dots, X_{i-1}\}$:

$$P(x_i | x_1, \dots, x_{i-1}) = P(x_i | \text{parents}(X_i))$$

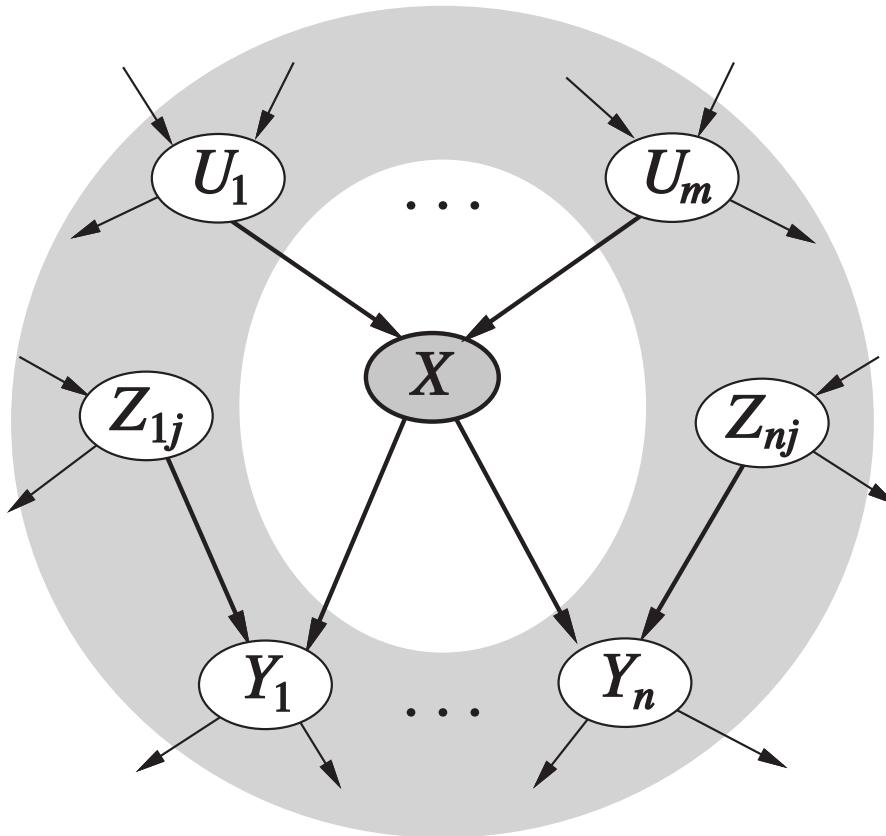
- Therefore $P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$.

Local semantics



A node X is conditionally independent to its non-descendants (the Z_{ij}) given its parents (the U_i).

Global semantics



A node X is conditionally independent of all other nodes in the network given its Markov blanket.

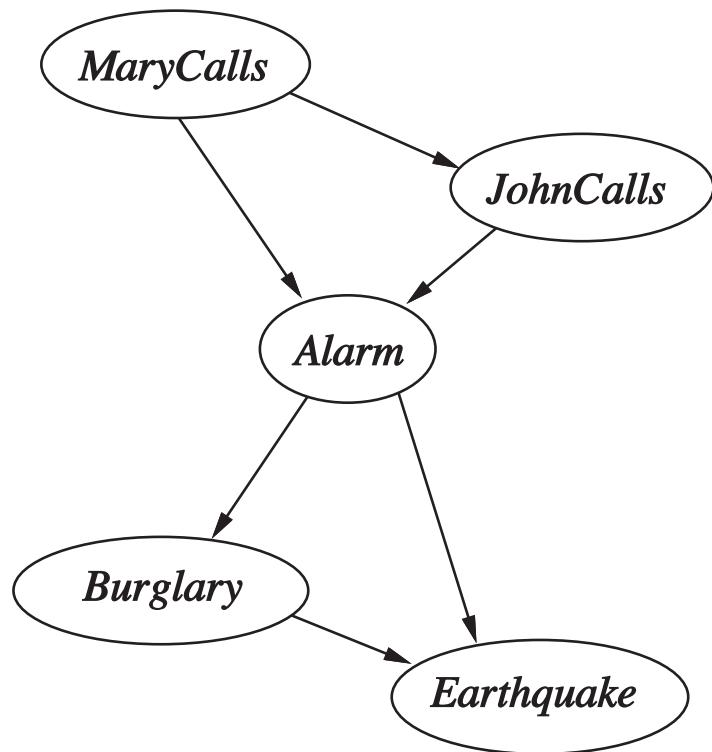
Construction

Bayesian networks are correct representations of the domain only if each node is conditionally independent of its other predecessors in the node ordering, given its parents.

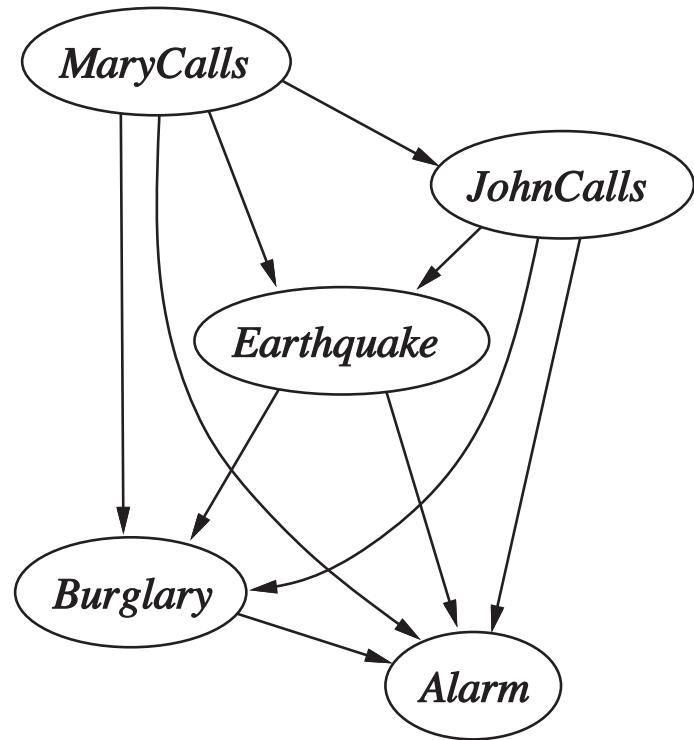
This suggests the following methodology for **building** Bayesian networks:

1. Choose an **ordering** of variables X_1, \dots, X_n .
2. For $i = 1$ to n :
 - Add X_i to the network.
 - Select a minimal set of parents from X_1, \dots, X_{i-1} such that $P(x_i|x_1, \dots, x_{i-1}) = P(x_i|\text{parents}(X_i))$.
 - For each parent, insert a link from the parent to X_i .
 - Write down the CPT.

Examples



(a)



(b)

These networks are alternatives to Example 3.

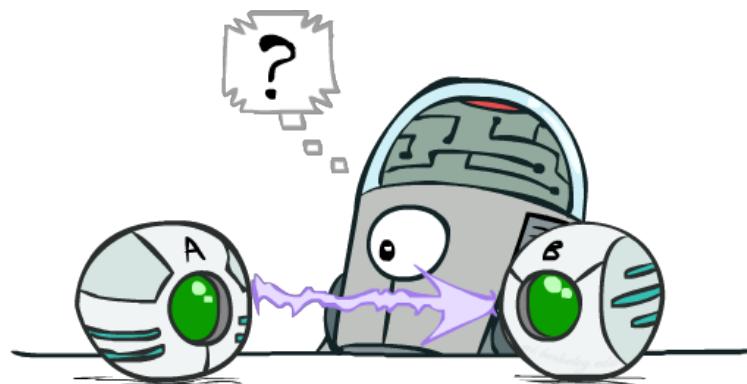
[Q] What do you think of these BNs?

Compactness

- A CPT for boolean X_i with k boolean parents has 2^k rows for the combinations of parent values.
- Each row requires one number p for $X_i = \text{true}$.
 - The number of $X_i = \text{false}$ is just $1 - p$.
- If each variable has no more than k parents, the complete network requires $O(n \times 2^k)$ numbers.
 - i.e., grows linearly with n , vs. $O(2^n)$ for the full joint distribution.
- For the burglary net, we need $1 + 1 + 4 + 2 + 2 = 10$ numbers (vs. $2^5 - 1 = 31$).
- Compactness depends on the **node ordering**.
 - Compare the three networks representing the same joint distribution.

Causality?

- When the network reflects the true causal patterns:
 - Often more compact (nodes have fewer parents).
 - Often easier to think about.
 - Often easier to elicit from experts.
- But, Bayesian networks **need not be causal**.
 - Sometimes no causal network exists over the domain (e.g., if variables are missing).
 - Edges reflect **correlation**, not causation.
- What do the edges really mean then?
 - Topology **may** happen to encode causal structure.
 - **Topology really encodes conditional independence.**





Judeas Pearl

Correlation does not imply causation.

Causes cannot be expressed in the language of probability theory.

Philosophers have tried to define causation in terms of probability: X causes Y if X raises the probability of Y .

However, the inequality

$$P(Y|X) > P(Y)$$

fails to capture the intuition behind "probability raising", which is fundamentally a causal concept connoting a causal influence of X over Y .

- Instead, the expression means that if we observe X , then the probability of Y increases.
- But this increase may come about for other reasons!

The correct formulation should read

$$P(Y|do(X = x)) > P(Y),$$

where $do(X = x)$ stands for an external intervention where X is set to the value x instead of being observed.

Observing vs. intervening

- The reading in barometer is useful to predict rain.

$$P(\text{rain} | \text{Barometer} = \text{high}) > P(\text{rain} | \text{Barometer} = \text{low})$$

- But hacking a barometer will not cause rain!

$$P(\text{rain} | \text{Barometer hacked to high}) = P(\text{rain} | \text{Barometer hacked to low})$$

Summary

- Uncertainty arises because of laziness and ignorance. It is **inescapable** in complex non-deterministic or partially observable environments.
- **Probabilistic reasoning** provides a framework for managing our knowledge and **beliefs**.
- Bayesian networks are DAGs whose nodes correspond to random variables; each node has a conditional distribution for the node, given its parents.
- A Bayesian Network specifies a full joint distribution.
 - They are often **exponentially** smaller than an explicitly enumerated joint distribution.

Introduction to Artificial Intelligence

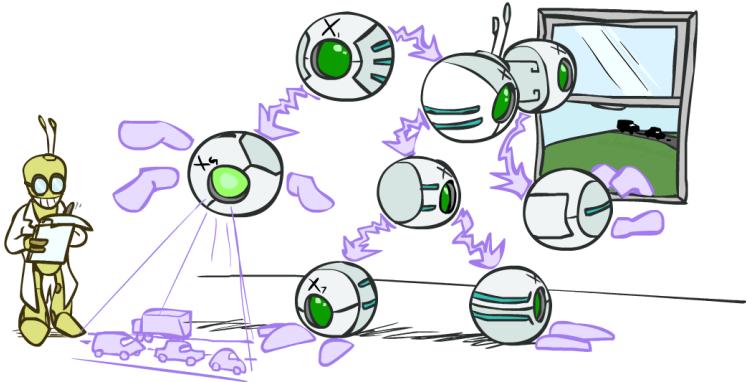
Lecture 6: Inference in Bayesian networks

Prof. Gilles Louppe
g.louppe@uliege.be



Today

- Exact inference
 - Inference by enumeration
 - Inference by variable elimination
- Continuous variables
- Approximate inference
 - Ancestral sampling
 - Rejection sampling
 - Likelihood weighting
 - Gibbs sampling



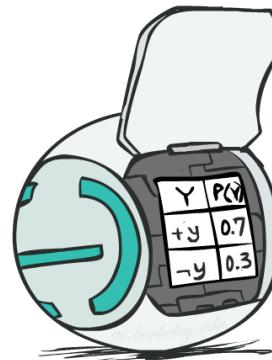
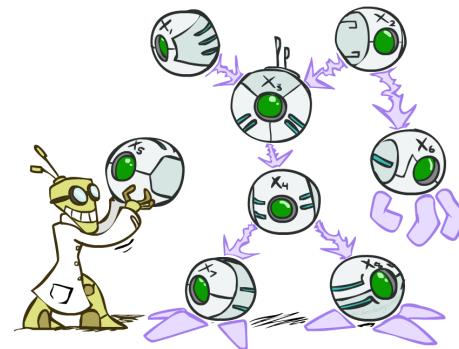
Bayesian networks

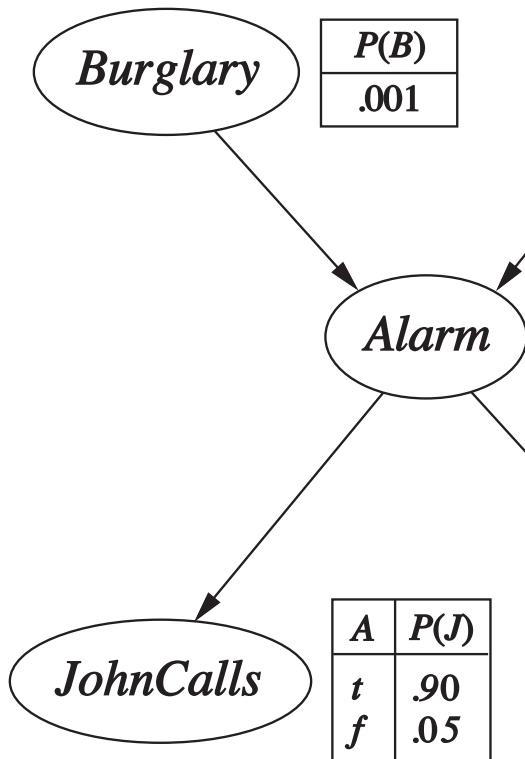
A Bayesian network is a **directed acyclic graph** in which:

- Each node corresponds to a **random variable** X_i .
- Each node X_i is annotated with a **conditional probability distribution** $P(X_i|\text{parents}(X_i))$ that quantifies the effect of the parents on the node.

A Bayesian network implicitly **encodes** the full joint distribution as the product of the local distributions:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i|\text{parents}(X_i))$$

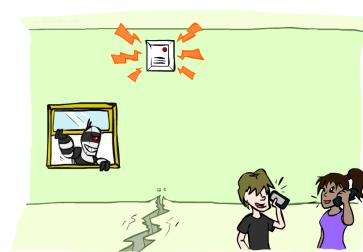




B	E	$P(A)$
t	t	.95
t	f	.94
f	t	.29
f	f	.001

A	$P(J)$
t	.90
f	.05

A	$P(M)$
t	.70
f	.01



$$\begin{aligned}
 P(b, \neg e, a, \neg j, m) &= P(b)P(\neg e)P(a|b, \neg e)P(\neg j|a)P(m, a) \\
 &= 0.001 \times 0.998 \times 0.94 \times 0.1 \times 0.7
 \end{aligned}$$

Exact inference

Inference

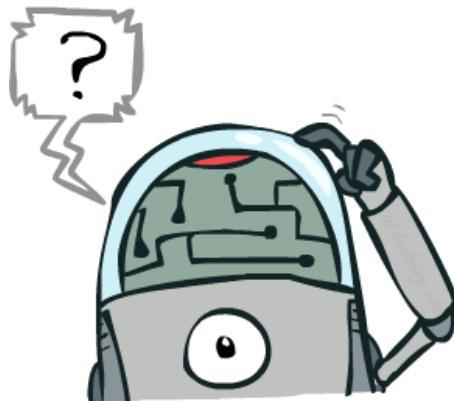
Inference is concerned with the problem **computing a marginal and/or conditional probability** from a joint probability distribution:

Simple queries: $P(X_i|e)$

Conjunctive queries: $P(X_i, X_j|e) = P(X_i|e)P(X_j|X_i, e)$

Most likely explanation: $\arg \max_q P(q|e)$

Optimal decisions: $\arg \max_a \mathbb{E}_{p(s'|s,a)} [V(s')]$



Inference by enumeration

Start from the joint distribution $P(Q, E_1, \dots, E_k, H_1, \dots, H_r)$.

1. Select the entries consistent with the evidence $E_1, \dots, E_k = e_1, \dots, e_k$.
2. Marginalize out the hidden variables to obtain the joint of the query and the evidence variables:

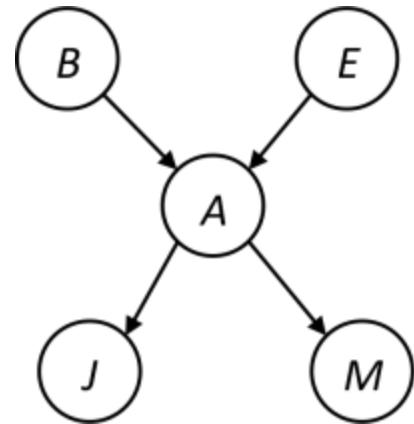
$$P(Q, e_1, \dots, e_k) = \sum_{h_1, \dots, h_r} P(Q, h_1, \dots, h_r, e_1, \dots, e_k).$$

3. Normalize:

$$\begin{aligned} Z &= \sum_q P(q, e_1, \dots, e_k) \\ P(Q|e_1, \dots, e_k) &= \frac{1}{Z} P(Q, e_1, \dots, e_k) \end{aligned}$$

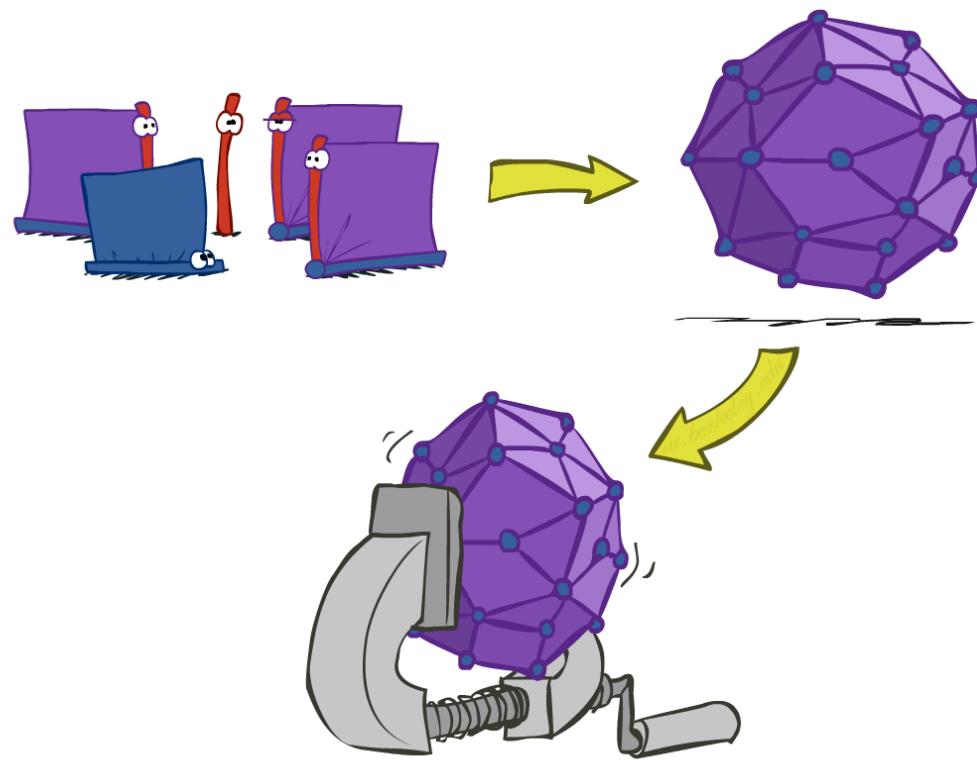
Consider the alarm network and the query $P(B|j, m)$:

$$\begin{aligned} P(B|j, m) &= \frac{1}{Z} \sum_e \sum_a P(B, j, m, e, a) \\ &\propto \sum_e \sum_a P(B, j, m, e, a) \end{aligned}$$



Using the Bayesian network, the full joint entries can be rewritten as the product of CPT entries:

$$P(B|j, m) \propto \sum_e \sum_a P(B)P(e)P(a|B, e)P(j|a)P(m|a)$$



Inference by enumeration is slow because the whole joint distribution is joined up before summing out the hidden variables.

Notice that factors that do not depend on the variables in the summations can be factored out, which means that marginalization does not necessarily have to be done at the end:

$$\begin{aligned} P(B|j, m) &\propto \sum_e \sum_a P(B)P(e)P(a|B, e)P(j|a)P(m|a) \\ &= P(B) \sum_e P(e) \sum_a P(a|B, e)P(j|a)P(m|a) \end{aligned}$$

function ENUMERATION-ASK(X, \mathbf{e}, bn) **returns** a distribution over X

inputs: X , the query variable
 \mathbf{e} , observed values for variables \mathbf{E}
 bn , a Bayes net with variables $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$ /* $\mathbf{Y} = \text{hidden variables}$ */

$\mathbf{Q}(X) \leftarrow$ a distribution over X , initially empty

for each value x_i of X **do**

$\mathbf{Q}(x_i) \leftarrow$ ENUMERATE-ALL($bn.\text{VARS}, \mathbf{e}_{x_i}$)
 where \mathbf{e}_{x_i} is \mathbf{e} extended with $X = x_i$

return NORMALIZE($\mathbf{Q}(X)$)

function ENUMERATE-ALL($vars, \mathbf{e}$) **returns** a real number

if EMPTY?($vars$) **then return** 1.0

$Y \leftarrow \text{FIRST}(vars)$

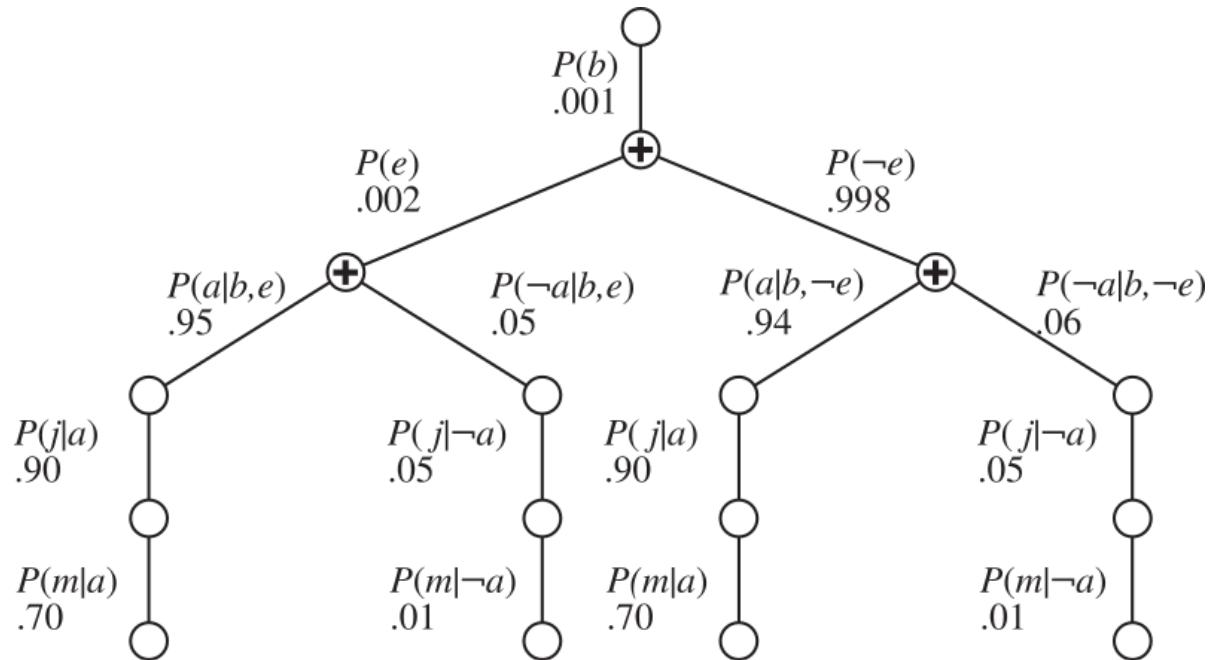
if Y has value y in \mathbf{e}

then return $P(y | parents(Y)) \times$ ENUMERATE-ALL(REST($vars$), \mathbf{e})

else return $\sum_y P(y | parents(Y)) \times$ ENUMERATE-ALL(REST($vars$), \mathbf{e}_y)
 where \mathbf{e}_y is \mathbf{e} extended with $Y = y$

Same complexity as DFS: $O(n)$ in space, $O(d^n)$ in time.

Evaluation tree for $P(b|j, m)$



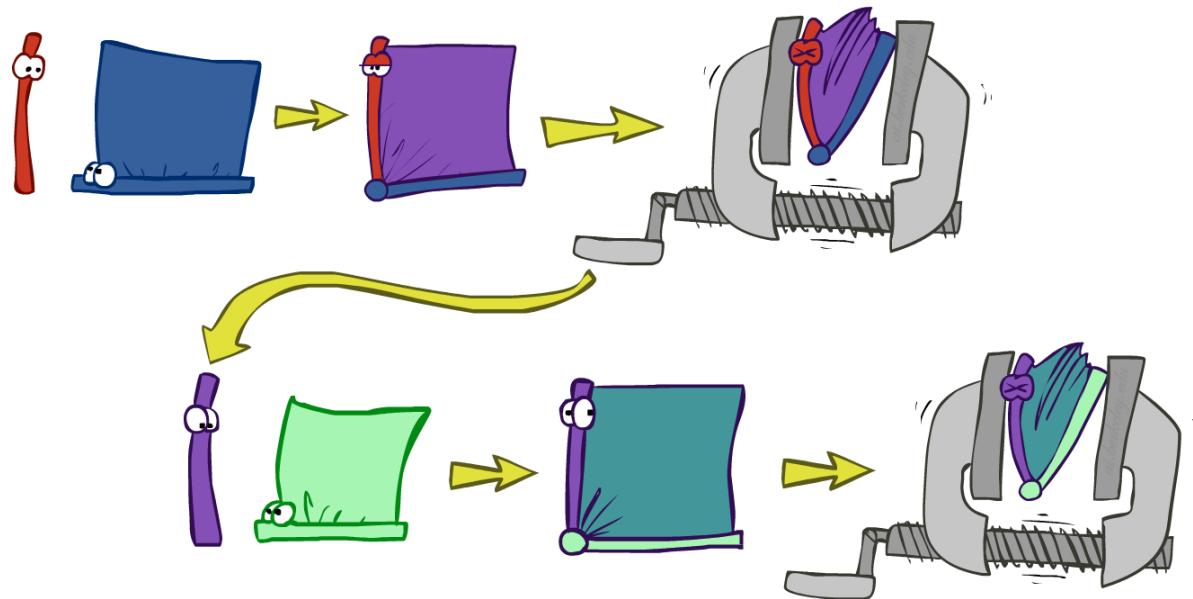
Enumeration is still **inefficient**: there are repeated computations!

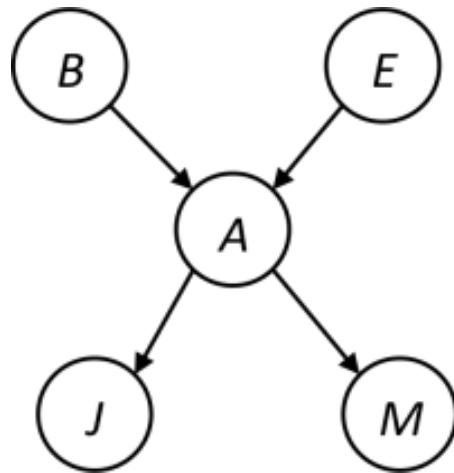
- e.g., $P(j|a)P(m|a)$ is computed twice, once for e and once for $\neg e$.
- These can be avoided by **storing intermediate results**.

Inference by variable elimination

The **variable elimination** (VE) algorithm carries out summations right-to-left and stores intermediate results (called **factors**) to avoid recomputations. The algorithm interleaves:

- Joining sub-tables
- Eliminating hidden variables





Example

$$\begin{aligned}
 P(B|j, m) &\propto P(B, j, m) \\
 &= P(B) \sum_e P(e) \sum_a P(a|B, e) P(j|a) P(m|a) \\
 &= \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(e) \times \sum_a \mathbf{f}_3(a, B, e) \times \mathbf{f}_4(a) \times \mathbf{f}_5(a) \\
 &= \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(e) \times \mathbf{f}_6(B, e) \quad (\text{sum out } A) \\
 &= \mathbf{f}_1(B) \times \mathbf{f}_7(B) \quad (\text{sum out } E)
 \end{aligned}$$

Factors

- Each **factor \mathbf{f}_i** is a multi-dimensional array indexed by the values of its argument variables. E.g.:

$$\mathbf{f}_4 = \mathbf{f}_4(A) = \begin{pmatrix} P(j|a) \\ P(j|\neg a) \end{pmatrix} = \begin{pmatrix} 0.90 \\ 0.05 \end{pmatrix}$$

$$\mathbf{f}_4(a) = 0.90$$

$$\mathbf{f}_4(\neg a) = 0.5$$

- Factors are initialized with the CPTs annotating the nodes of the Bayesian network, conditioned on the evidence.

Join

The pointwise product \times , or **join**, of two factors \mathbf{f}_1 and \mathbf{f}_2 yields a new factor \mathbf{f}_3 .

- Exactly like a **database join**!
- The variables of \mathbf{f}_3 are the **union** of the variables in \mathbf{f}_1 and \mathbf{f}_2 .
- The elements of \mathbf{f}_3 are given by the product of the corresponding elements in \mathbf{f}_1 and \mathbf{f}_2 .

A	B	$\mathbf{f}_1(A, B)$	B	C	$\mathbf{f}_2(B, C)$	A	B	C	$\mathbf{f}_3(A, B, C)$
T	T	.3	T	T	.2	T	T	T	$.3 \times .2 = .06$
T	F	.7	T	F	.8	T	T	F	$.3 \times .8 = .24$
F	T	.9	F	T	.6	T	F	T	$.7 \times .6 = .42$
F	F	.1	F	F	.4	T	F	F	$.7 \times .4 = .28$
						F	T	T	$.9 \times .2 = .18$
						F	T	F	$.9 \times .8 = .72$
						F	F	T	$.1 \times .6 = .06$
						F	F	F	$.1 \times .4 = .04$

Figure 14.10 Illustrating pointwise multiplication: $\mathbf{f}_1(A, B) \times \mathbf{f}_2(B, C) = \mathbf{f}_3(A, B, C)$.

Elimination

Summing out, or eliminating, a variable from a factor is done by adding up the submatrices formed by fixing the variable to each of its values in turn.

For example, to sum out A from $\mathbf{f}_3(A, B, C)$, we write:

$$\begin{aligned}\mathbf{f}(B, C) &= \sum_a \mathbf{f}_3(a, B, C) = \mathbf{f}_3(a, B, C) + \mathbf{f}_3(\neg a, B, C) \\ &= \begin{pmatrix} 0.06 & 0.24 \\ 0.42 & 0.28 \end{pmatrix} + \begin{pmatrix} 0.18 & 0.72 \\ 0.06 & 0.04 \end{pmatrix} = \begin{pmatrix} 0.24 & 0.96 \\ 0.48 & 0.32 \end{pmatrix}\end{aligned}$$

General Variable Elimination algorithm

- Query: $P(Q|e_1, \dots, e_k)$.
- Start with the initial factors.
 - The local CPTs, instantiated by the evidence.
- While there are still hidden variables:
 - Pick a hidden variable H
 - Join all factors mentioning H
 - Eliminate H
- Join all remaining factors
- Normalize

Example

(blackboard example)

Relevance

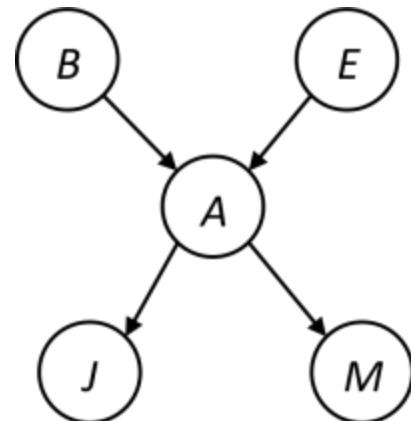
Consider the query $P(J|b)$:

$$P(J|b) \propto P(b) \sum_e P(e) \sum_a P(a|b, e) P(J|a) \sum_m P(m|a)$$

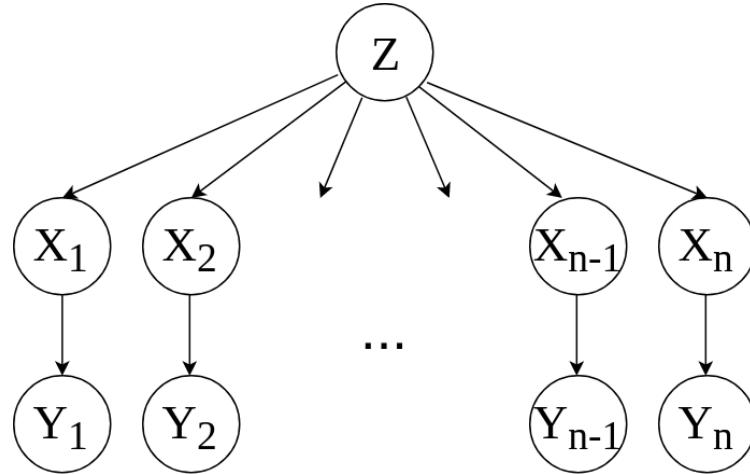
- $\sum_m P(m|a) = 1$, therefore M is **irrelevant** for the query.
- In other words, $P(J|b)$ remains unchanged if we remove M from the network.

Theorem

H is irrelevant for $P(Q|e)$ unless
 $H \in \text{ancestors}(\{Q\} \cup E)$.



Complexity



Consider the query $P(X_n | y_1, \dots, y_n)$. Work through the two elimination orderings:

- Z, X_1, \dots, X_{n-1}
- X_1, \dots, X_{n-1}, Z

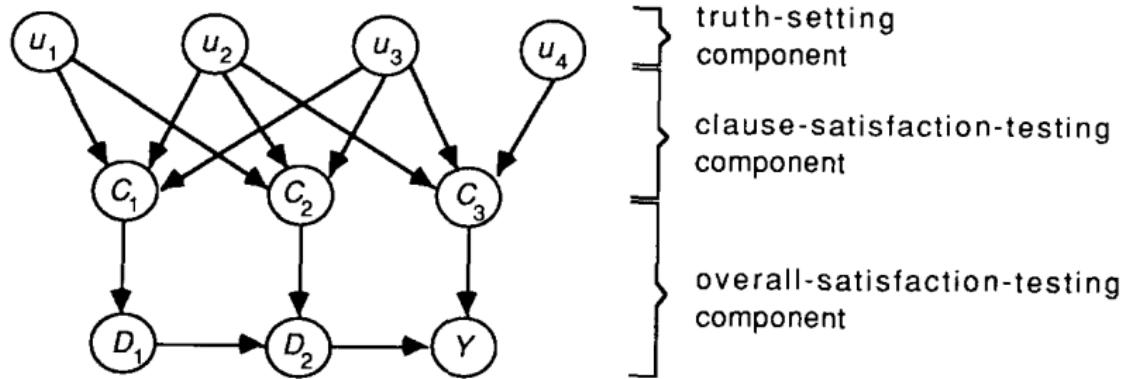
What is the size of the maximum factor generated for each of the orderings?

- Answer: 2^{n+1} vs. 2^2 (assuming boolean values)

The computational and space complexity of variable elimination is determined by the largest factor.

- The elimination ordering can greatly affect the size of the largest factor.
- Does there always exist an ordering that only results in small factors? No!
- Singly connected networks (polytrees):
 - Any two nodes are connected by at most one (undirected path).
 - For these networks, time and space complexity of variable elimination are $O(nd^k)$.

Worst-case complexity?



3SAT is a special case of inference:

- CSP: $(u_1 \vee u_2 \vee u_3) \wedge (\neg u_1 \vee \neg u_2 \vee u_3) \wedge (u_2 \vee \neg u_3 \vee u_4)$
- $P(U_i = 0) = P(U_i = 1) = 0.5$
- $C_1 = U_1 \vee U_2 \vee U_3; C_2 = \neg U_1 \vee \neg U_2 \vee U_3; C_3 = U_2 \vee \neg U_3 \vee U_4$
- $D_1 = C_1; D_2 = D_1 \wedge C_2$
- $Y = D_2 \wedge C_3$

If we can answer whether $P(Y = 1) > 0$, then we answer whether 3SAT has a solution.

- By reduction, inference in Bayesian networks is therefore **NP-complete**.
- There is no known efficient probabilistic inference algorithm in general.

Continuous variables

Random variables

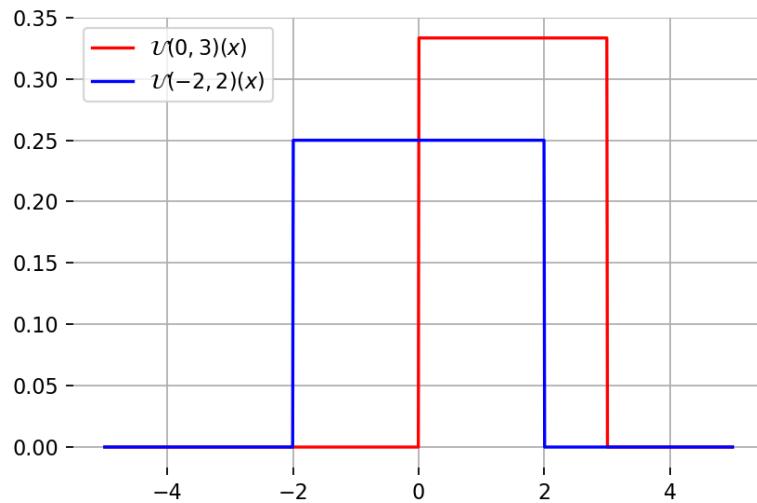
Let $X : \Omega \rightarrow D_X$ be a random variable.

- When D_X is finite or countably infinite, X is called a discrete random variable.
- Its probability distribution is described by a probability mass function that assigns a probability to each value $x \in D_X$.
- When D_X is uncountably infinite (e.g., $D_X = \mathbb{R}$), X is called a continuous random variable.
- If X is absolutely continuous, its probability distribution is described by a density function p that assigns a probability to any interval $[a, b] \subseteq D_X$ such that

$$P(a < X \leq b) = \int_a^b p(x)dx,$$

where p is non-negative piecewise continuous and such that $\int_{D_X} p(x)dx = 1$.

Uniform

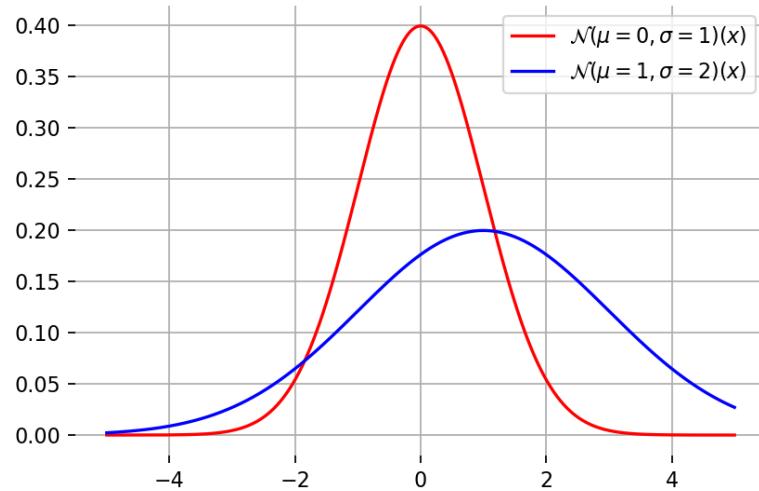


The uniform distribution $\mathcal{U}(a, b)$ is described by the density function

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{otherwise} \end{cases}$$

where $a \in \mathbb{R}$ and $b \in \mathbb{R}$ are the bounds of its support.

Normal

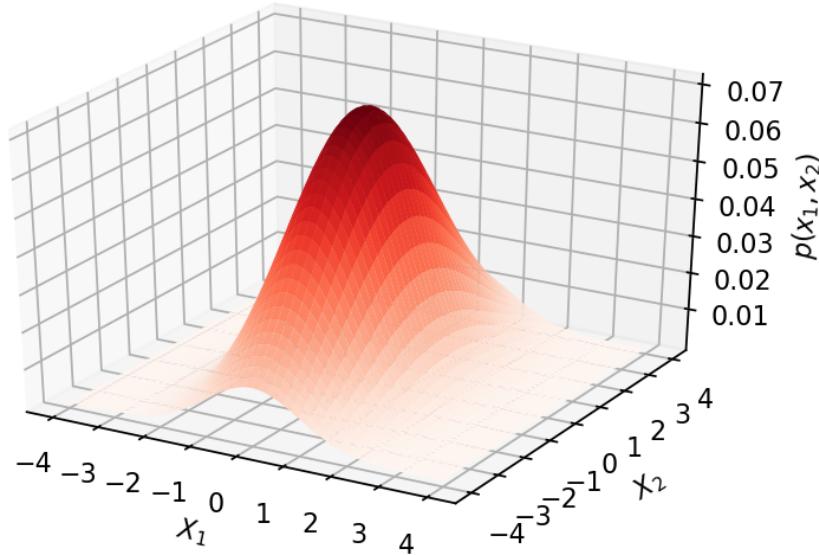


The normal (or Gaussian) distribution $\mathcal{N}(\mu, \sigma)$ is described by the density function

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

where $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$ are its mean and standard deviation parameters.

Multivariate normal



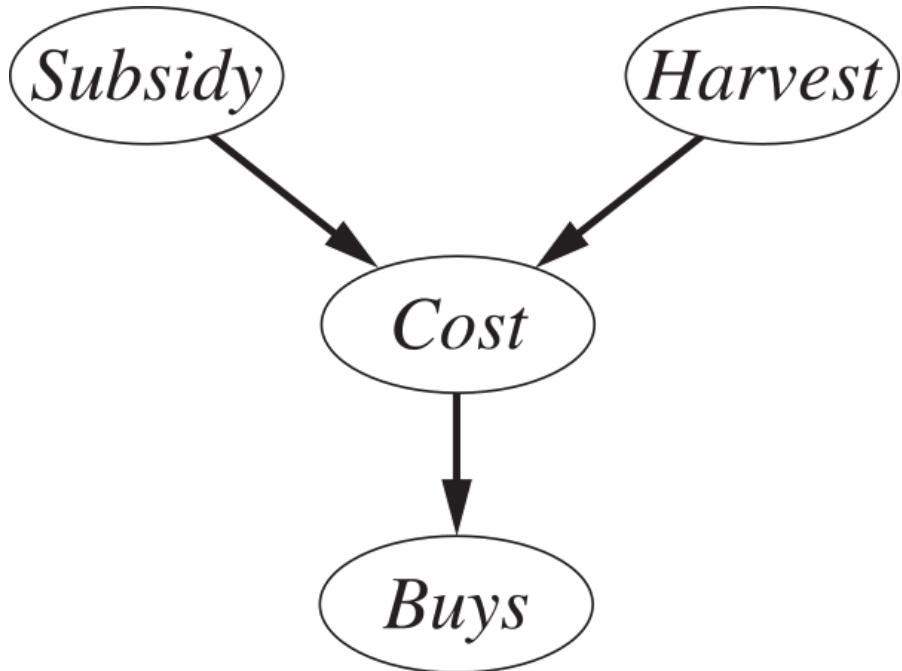
The multivariate normal distribution generalizes to N random variables. Its (joint) density function is defined as

$$p(\mathbf{x} = x_1, \dots, x_n) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right)$$

where $\boldsymbol{\mu} \in \mathbb{R}^n$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ is positive semi-definite.

- The (multivariate) Normal density is the only density for real random variables that is **closed under marginalization and multiplication**.
- Also, a linear (or affine) function of a Normal random variable is Normal; and, a sum of Normal variables is Normal.
- For these reasons, the algorithms we will discuss will be tractable only for finite random variables or Normal random variables.

Hybrid Bayesian networks



What if we have both **discrete** (e.g., **Subsidy** and **Buys**) and **continuous** variables (e.g., **Harvest** and **Cost**) in a same Bayesian network?

Options

- Discretization: transform continuous variables into discrete variables.
 - Issues: possibly large errors due to precision loss, large CPTs.
- Define the conditional distribution with a **finitely parameterized** canonical distribution.
 - e.g., assume it is a Gaussian distribution.
- Use a non-parametric representation.

Continuous child variables

- We need to specify a conditional density function for each continuous child variable given continuous parents, for each possible assignment to discrete parents.
 - e.g., we need to specify both $p(c|h, s)$ and $p(c|h, \neg s)$
- Common choice: the **linear Gaussian model** (LG):
 - $p(c|h, s) = \mathcal{N}(c|a_t h + b_t, \sigma_t^2)$
 - $p(c|h, \neg s) = \mathcal{N}(c|a_f h + b_f, \sigma_f^2)$

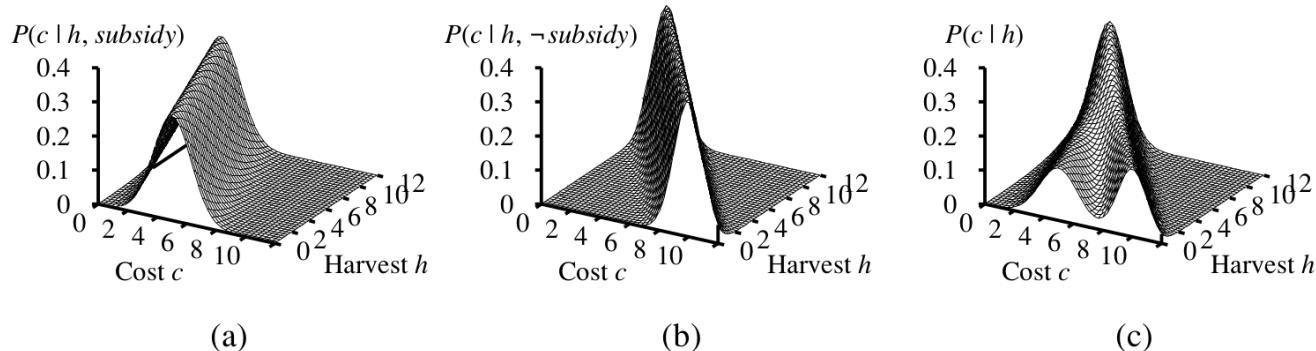


Figure 14.6 The graphs in (a) and (b) show the probability distribution over *Cost* as a function of *Harvest* size, with *Subsidy* true and false, respectively. Graph (c) shows the distribution $P(\text{Cost} | \text{Harvest})$, obtained by summing over the two subsidy cases.

Conditional Gaussian network

- The joint distribution of an all-continuous network with linear Gaussian distributions is a multivariate Gaussian.
- The joint distribution of a network with discrete or linear Gaussian continuous variables is a **conditional Gaussian network**.
 - i.e., a multivariate Gaussian over all continuous variables for each combination of the discrete variable values.

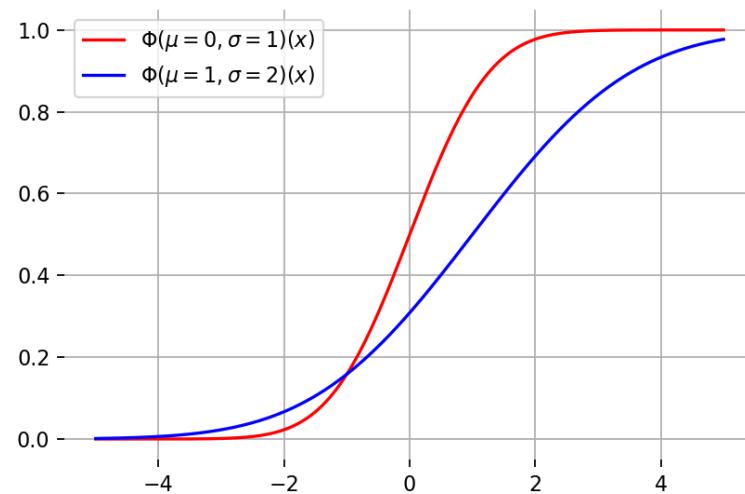
Discrete child variables, with continuous parents

- We need to specify a conditional distribution for each discrete child variable, given continuous parents.
- It is often reasonable to assume that the probability values of the discrete outcomes are almost piece-wise constant but **vary smoothly in intermediate regions**.

For example, if B is binary, $P(b|c)$ could be a "soft" threshold, such as the **probit distribution** for which

$$P(b|c) = \Phi((c - \mu)/\sigma),$$

where Φ is the cumulative distribution function of the (standard) normal distribution.



Variable elimination

Variable elimination in hybrid Bayesian networks can be conducted similarly as in the discrete case, by replacing **summations with integrations**.

- Exact inference remains possible **under some assumptions**.
 - e.g., for linear Gaussian models, queries can all be derived analytically.
- However, this often **does not scale** to arbitrary continuous distributions.
 - e.g., numerical approximations of integrals amount to discretize continuous variables.

Approximate inference

a.k.a. Monte Carlo methods

Exact inference is **intractable** for most probabilistic models of practical interest.
(e.g., involving many variables, continuous and discrete, undirected cycles, etc).

Solution

Abandon exact inference and develop **approximate** but **faster** inference algorithms:

- **Sampling methods**: produce answers by repeatedly generating random numbers from a distribution of interest.
- **Variational methods**: formulate inference as an optimization problem.
- **Belief propagation methods**: formulate inference as a message-passing algorithm.

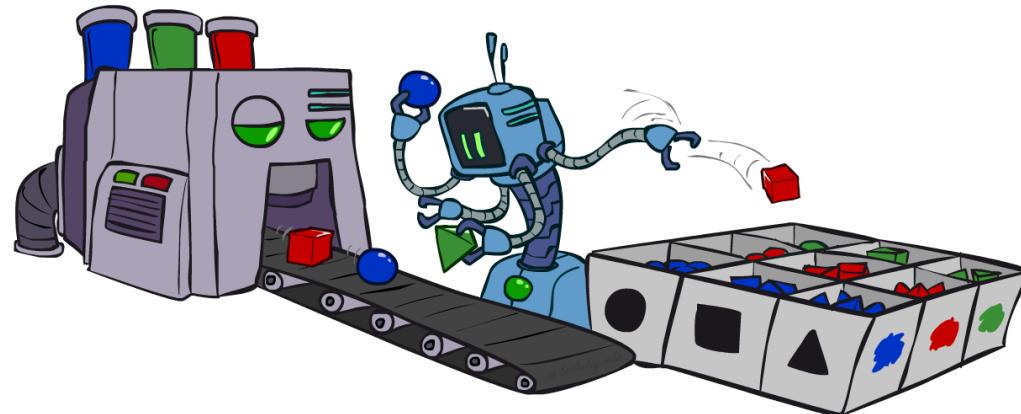
Sampling methods

Basic idea:

- Draw N samples from a sampling distribution S .
- Compute an approximate posterior probability \hat{P} .
- Show this approximate converges to the true probability distribution P .

Why sampling?

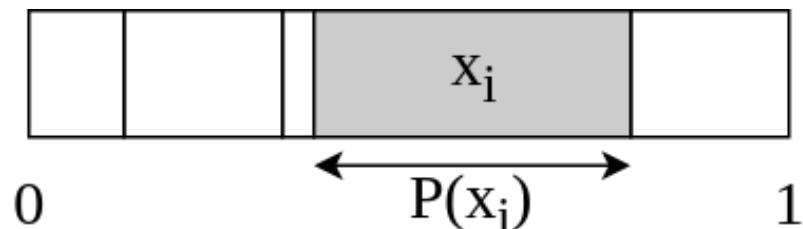
Generating samples is often much faster than computing the right answer (e.g., with variable elimination).



Sampling

How to sample from the distribution of a discrete variable X ?

- Assume k discrete outcomes x_1, \dots, x_k with probability $P(x_i)$.
- Assume sampling from $\mathcal{U}(0, 1)$ is possible.
 - e.g., as enabled by a standard `rand()` function.
- Divide the $[0, 1]$ interval into k regions, with region i having size $P(x_i)$.
- Sample $u \sim \mathcal{U}(0, 1)$ and return the value associated to the region in which u falls.



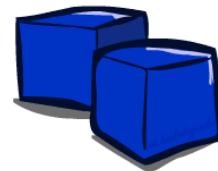
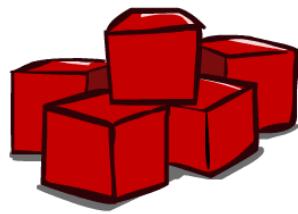
$P(C)$

C	P
red	0.6
green	0.1
blue	0.3

$0 \leq u < 0.6 \rightarrow C = \text{red}$

$0.6 \leq u < 0.7 \rightarrow C = \text{green}$

$0.7 \leq u < 1 \rightarrow C = \text{blue}$



The same algorithm extends to continuous variables, assuming access to the **inverse cumulative distribution function** F^{-1} .

- for $u \in [0, 1]$, $F^{-1}(u) = b$ such that $F(b) = u$, where F is the cumulative distribution function

$$F(b) = P(X < b) = \int_0^b p(x)dx.$$

- F^{-1} is known analytically for most canonical distributions.

[Q] How to extend to arbitrary multivariate distributions?

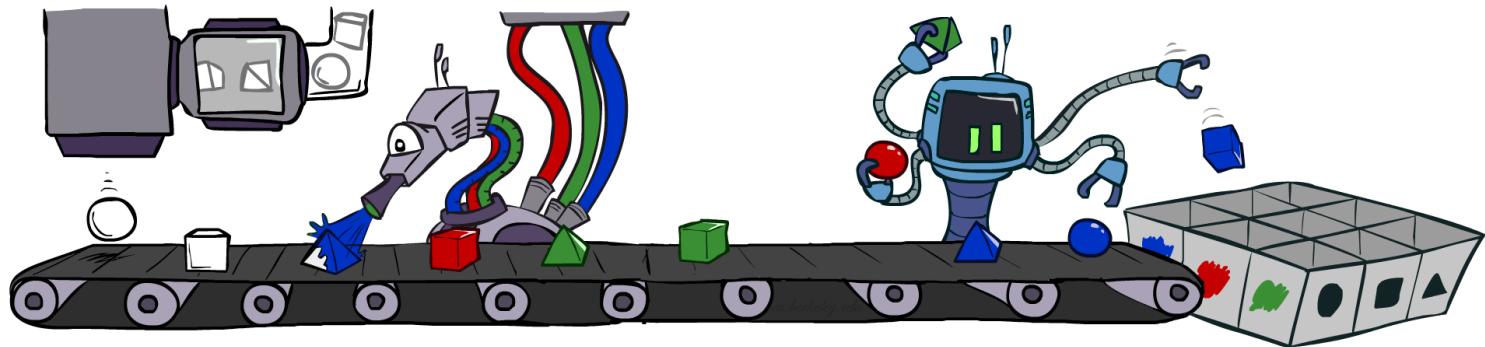
Prior sampling

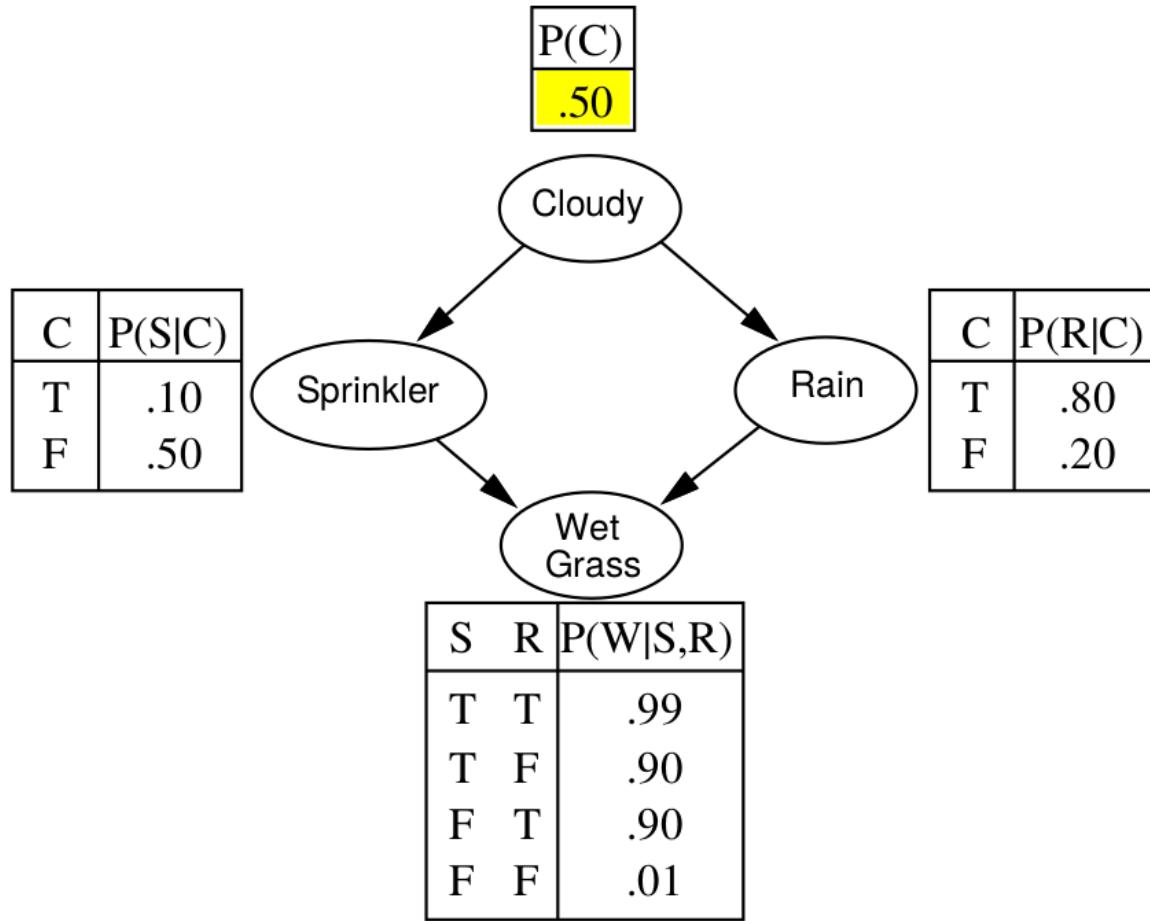
Sampling from a Bayesian network, [without observed evidence](#):

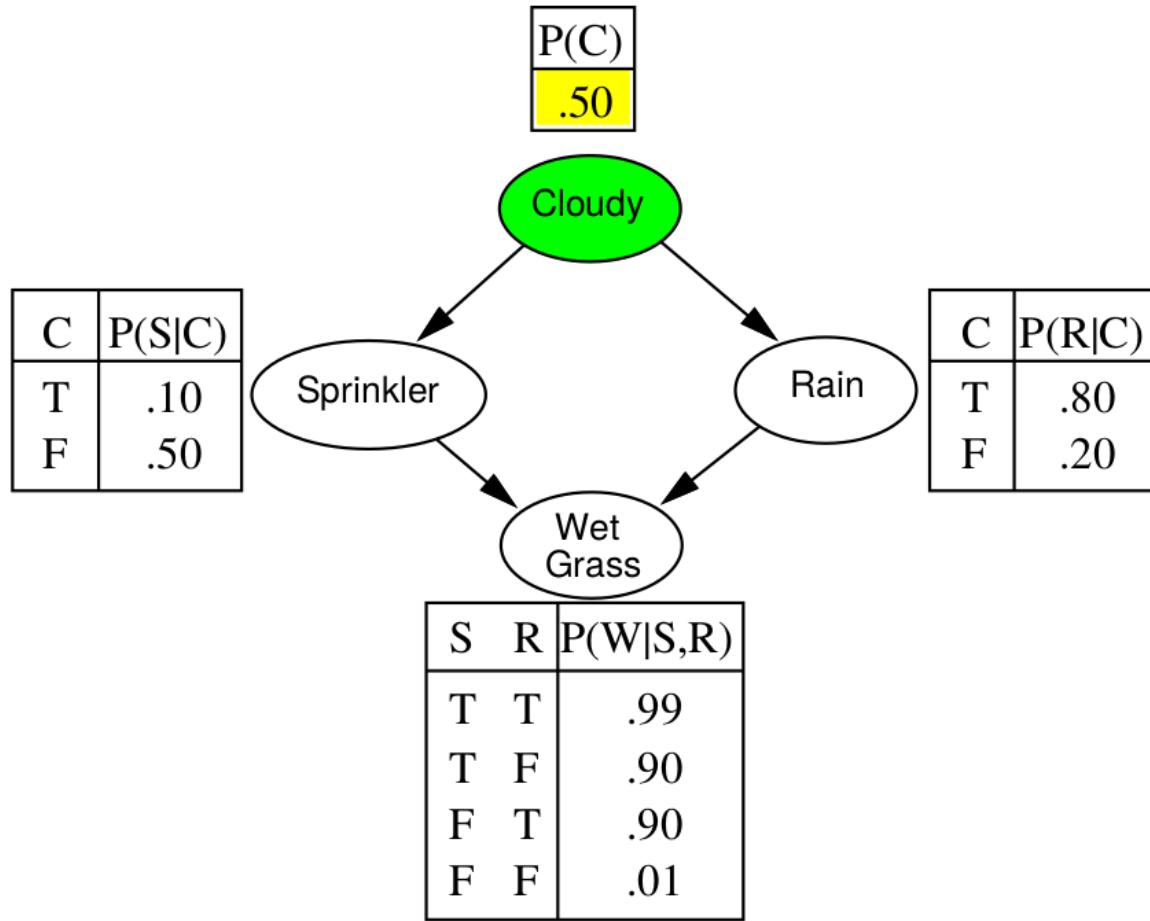
- Sample each variable in turn, [in topological order](#).
- The probability distribution from which the value is sampled is conditioned on the values already assigned to the variable's parents.

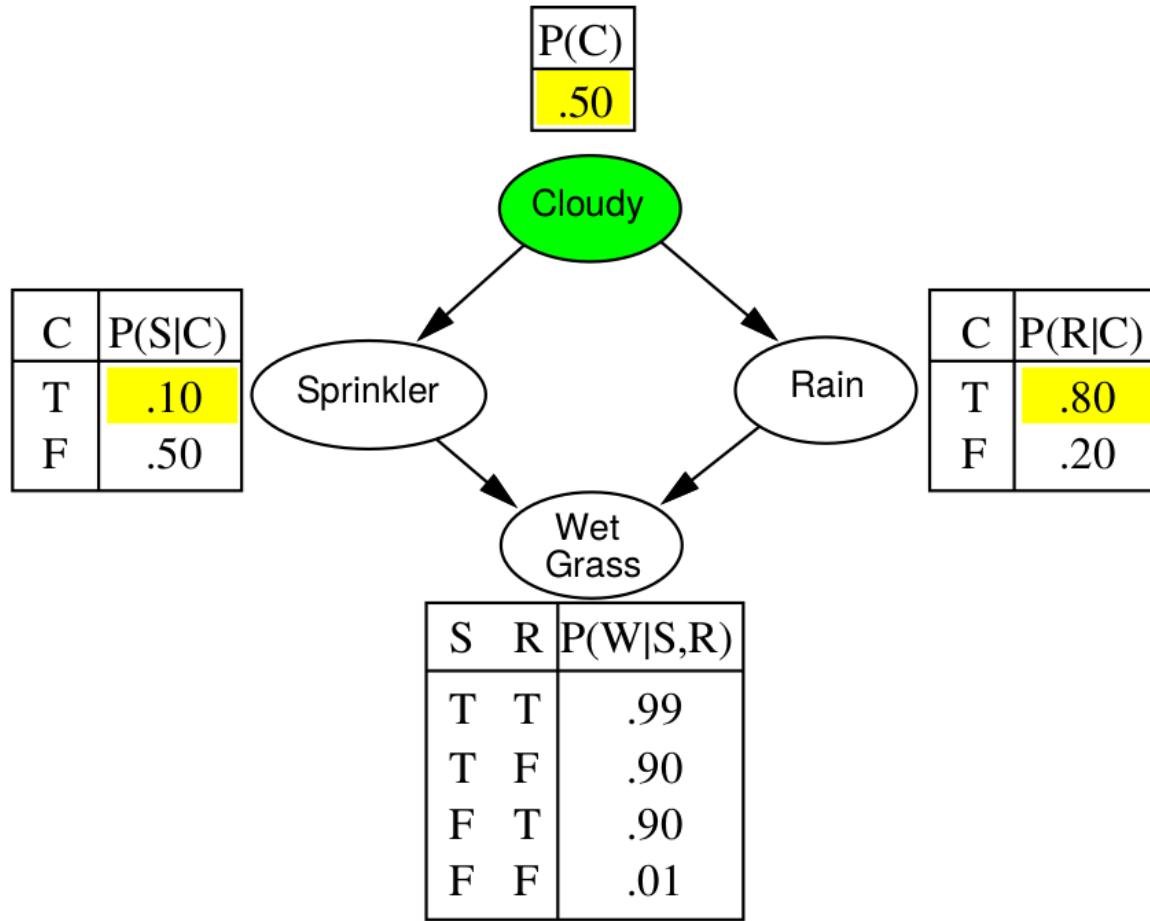
function PRIOR-SAMPLE(bn) **returns** an event sampled from the prior specified by bn
inputs: bn , a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \dots, X_n)$

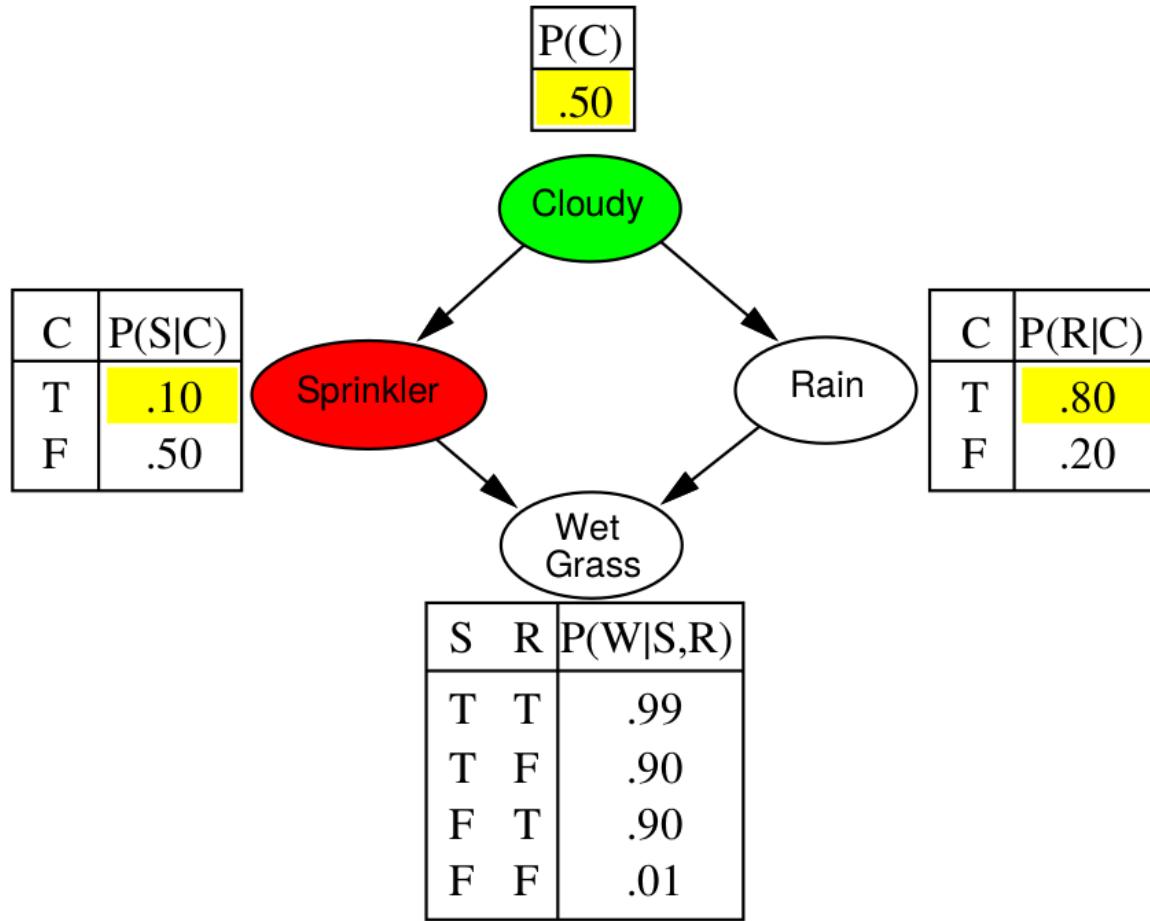
```
x  $\leftarrow$  an event with  $n$  elements
foreach variable  $X_i$  in  $X_1, \dots, X_n$  do
    x[i]  $\leftarrow$  a random sample from  $\mathbf{P}(X_i \mid \text{parents}(X_i))$ 
return x
```

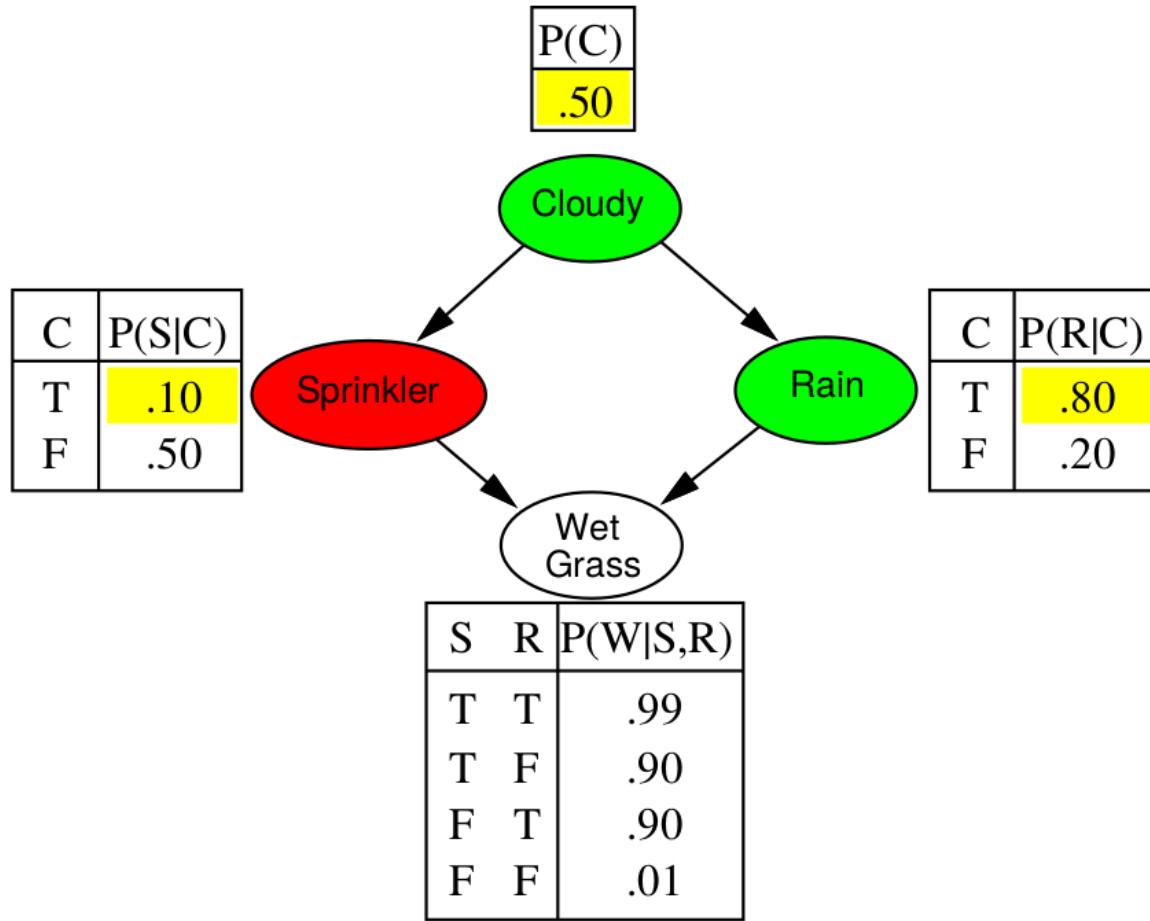


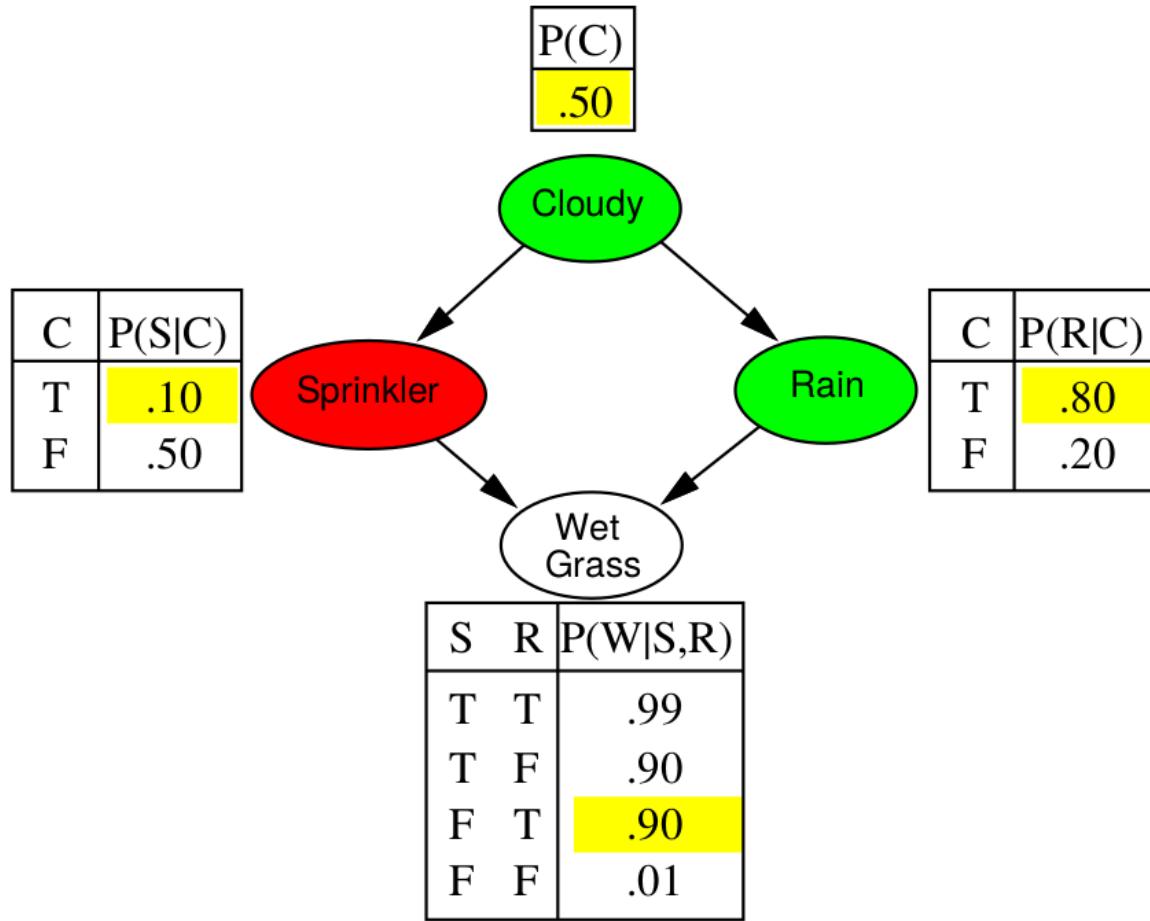


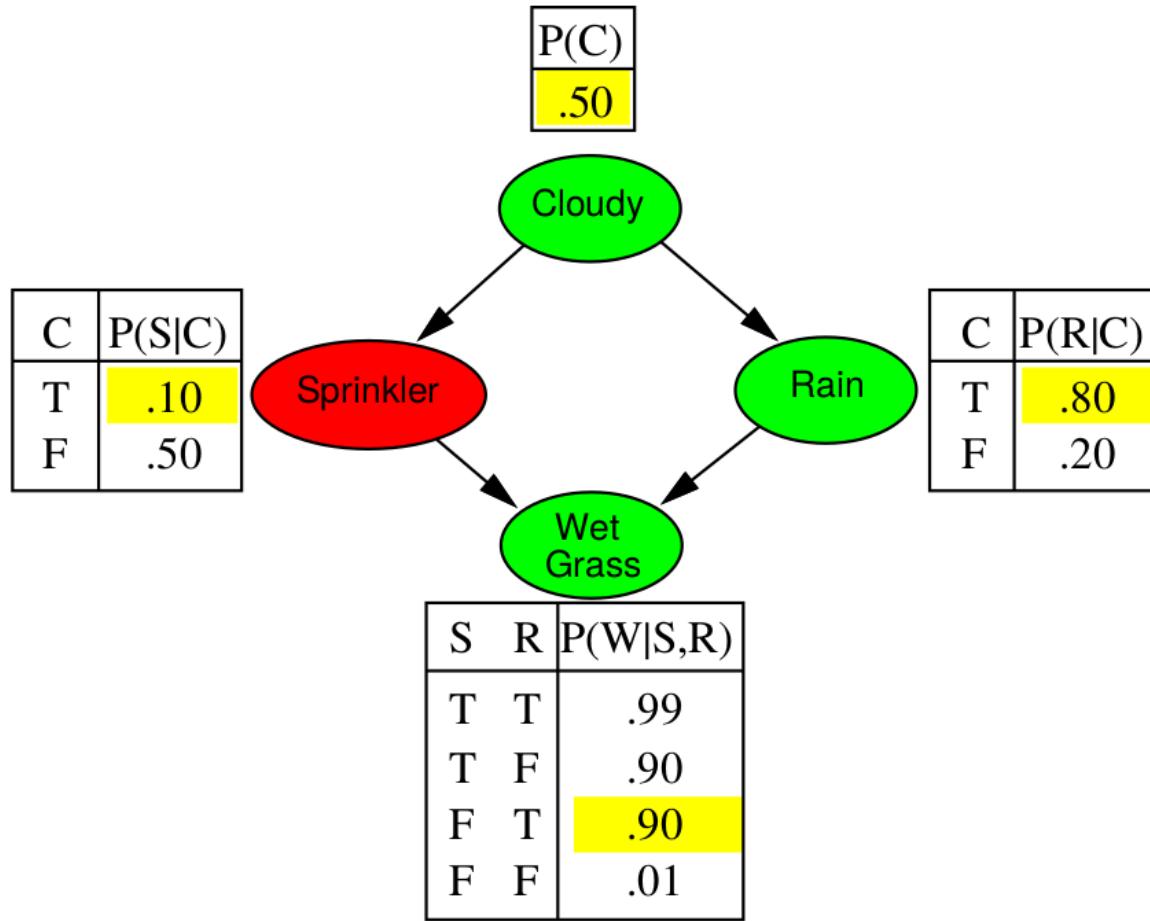












Example

We will collect a bunch of samples from the Bayesian network:

$c, \neg s, r, w$

c, s, r, w

$\neg c, s, r, \neg w$

$c, \neg s, r, w$

$\neg c, \neg s, \neg r, w$

If we want to know $P(W)$:

- We have counts $\langle w : 4, \neg w : 1 \rangle$
- Normalize to obtain $\hat{P}(W) = \langle w : 0.8, \neg w : 0.2 \rangle$
- This will get closer to the true distribution $P(W)$ as we collect more samples.

Analysis

The probability that prior sampling generates a particular event is

$$S_{PS}(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i)) = P(x_1, \dots, x_n)$$

i.e., the Bayesian network's joint probability.

Let $N_{PS}(x_1, \dots, x_n)$ denote the number of samples of an event. We define the probability **estimate**

$$\hat{P}(x_1, \dots, x_n) = N_{PS}(x_1, \dots, x_n)/N.$$

Then,

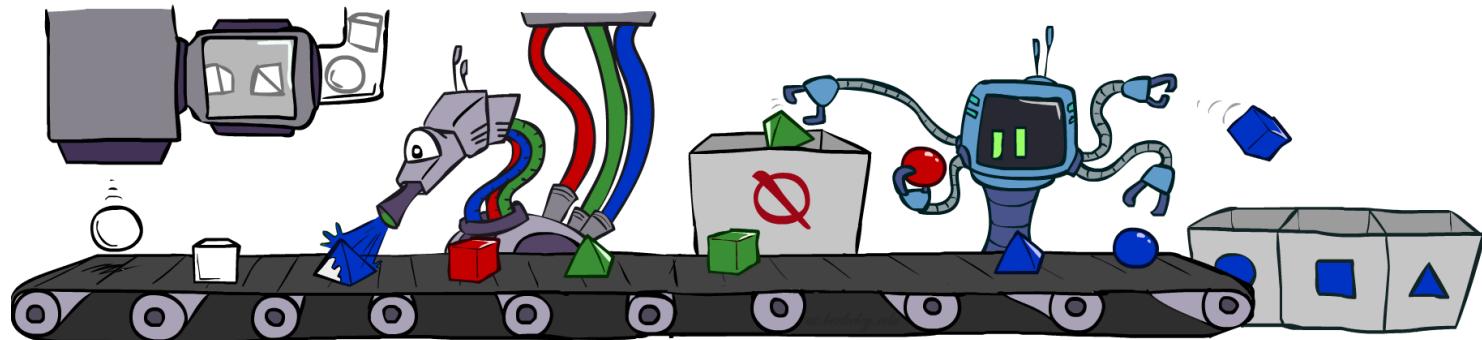
$$\begin{aligned}\lim_{N \rightarrow \infty} \hat{P}(x_1, \dots, x_n) &= \lim_{N \rightarrow \infty} N_{PS}(x_1, \dots, x_n)/N \\ &= S_{PS}(x_1, \dots, x_n) \\ &= P(x_1, \dots, x_n)\end{aligned}$$

Therefore, prior sampling is consistent:

$$P(x_1, \dots, x_n) \approx N_{PS}(x_1, \dots, x_n)/N$$

Rejection sampling

Using prior sampling, an estimate $\hat{P}(x|e)$ can be formed from the proportion of samples x agreeing with the evidence e among all samples agreeing with the evidence.



function REJECTION-SAMPLING(X, \mathbf{e}, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$

inputs: X , the query variable

\mathbf{e} , observed values for variables \mathbf{E}

bn , a Bayesian network

N , the total number of samples to be generated

local variables: \mathbf{N} , a vector of counts for each value of X , initially zero

for $j = 1$ to N **do**

$\mathbf{x} \leftarrow \text{PRIOR-SAMPLE}(bn)$

if \mathbf{x} is consistent with \mathbf{e} **then**

$\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where x is the value of X in \mathbf{x}

return NORMALIZE(\mathbf{N})

Analysis

Let consider the posterior probability estimate $\hat{P}(x|e)$ formed by rejection sampling:

$$\begin{aligned}\hat{P}(x|e) &= N_{PS}(x, e)/N_{PS}(e) \\ &= \frac{N_{PS}(x, e)}{N} / \frac{N_{PS}(e)}{N} \\ &\approx P(x, e)/P(e) \\ &= P(x|e)\end{aligned}$$

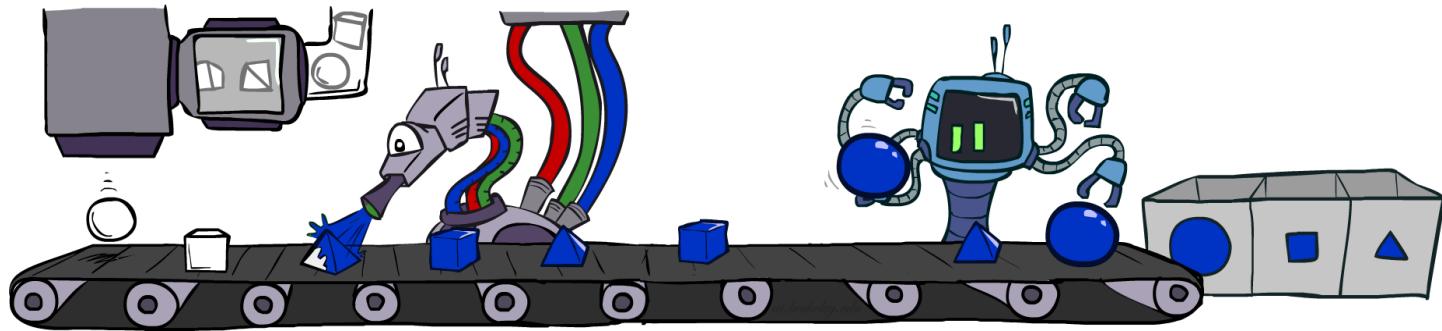
Therefore, rejection sampling returns **consistent** posterior estimates.

- The standard deviation of the error in each probability is $O(1/\sqrt{n})$.
- **Problem:** many samples are rejected!
 - Hopelessly expensive if the evidence is unlikely, i.e. if $P(e)$ is small.
 - Evidence is not exploited when sampling.

Likelihood weighting

Idea: **clamp** the evidence variables, sample the rest.

- Problem: the resulting sampling distribution is not consistent.
- Solution: **weight** by probability of evidence given parents.



function LIKELIHOOD-WEIGHTING(X, \mathbf{e}, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$

inputs: X , the query variable
 \mathbf{e} , observed values for variables \mathbf{E}
 bn , a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \dots, X_n)$
 N , the total number of samples to be generated

local variables: \mathbf{W} , a vector of weighted counts for each value of X , initially zero

for $j = 1$ to N **do**
 $\mathbf{x}, w \leftarrow$ WEIGHTED-SAMPLE(bn, \mathbf{e})
 $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ where x is the value of X in \mathbf{x}

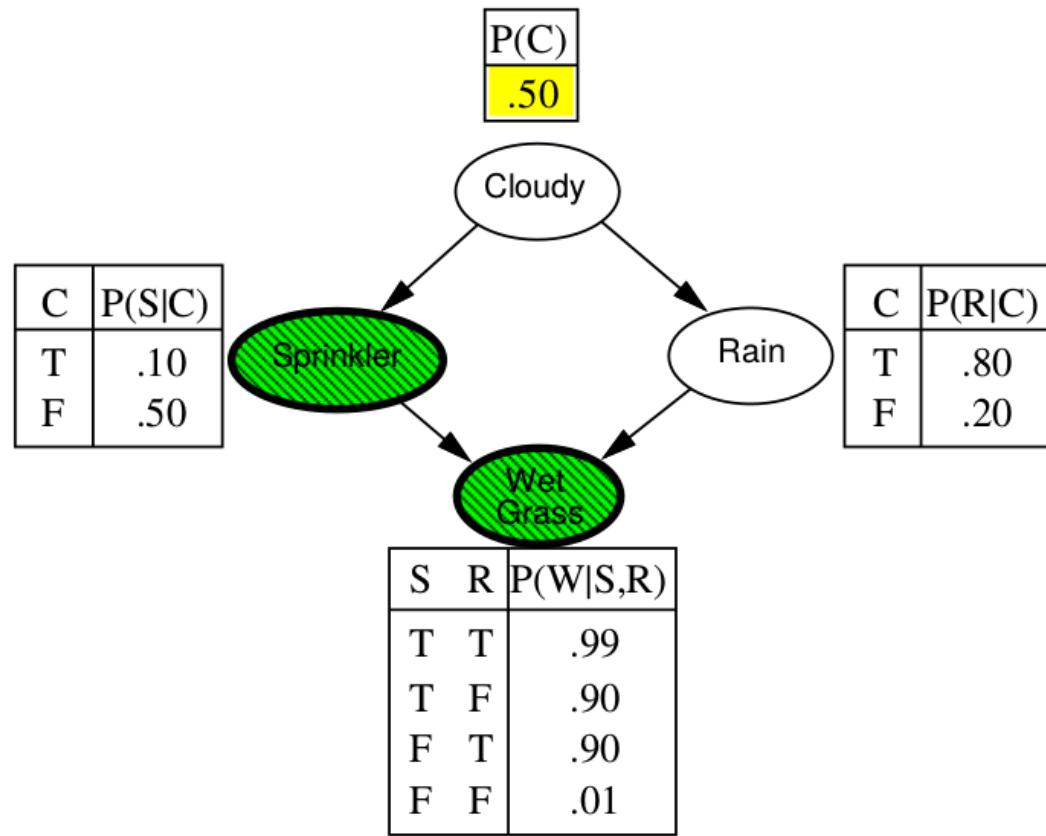
return NORMALIZE(\mathbf{W})

function WEIGHTED-SAMPLE(bn, \mathbf{e}) **returns** an event and a weight

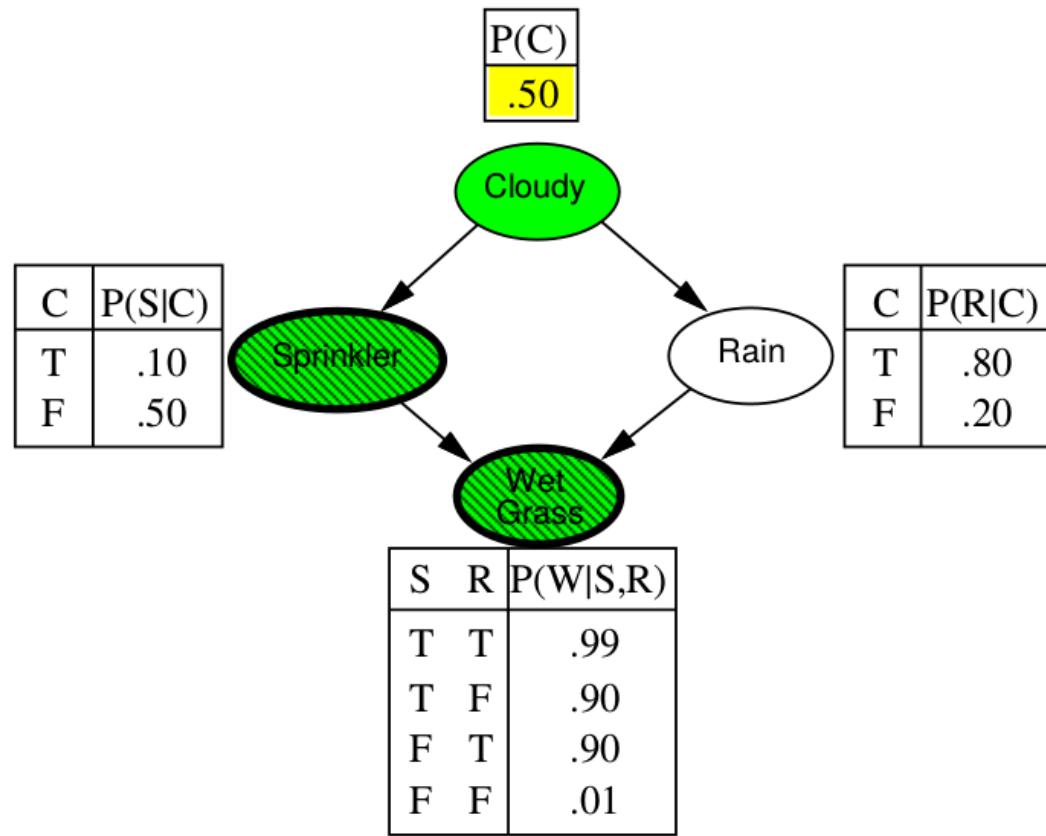
$w \leftarrow 1; \mathbf{x} \leftarrow$ an event with n elements initialized from \mathbf{e}

foreach variable X_i **in** X_1, \dots, X_n **do**
if X_i is an evidence variable with value x_i in \mathbf{e}
then $w \leftarrow w \times P(X_i = x_i | parents(X_i))$
else $\mathbf{x}[i] \leftarrow$ a random sample from $\mathbf{P}(X_i | parents(X_i))$

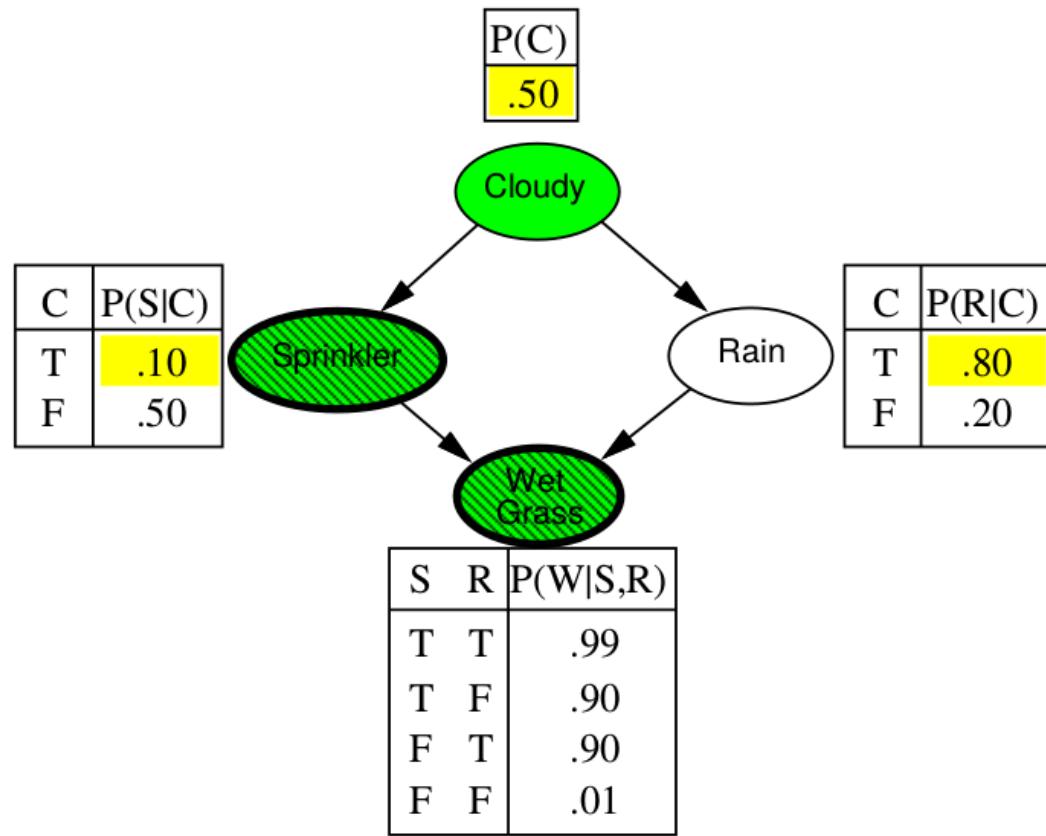
return \mathbf{x}, w



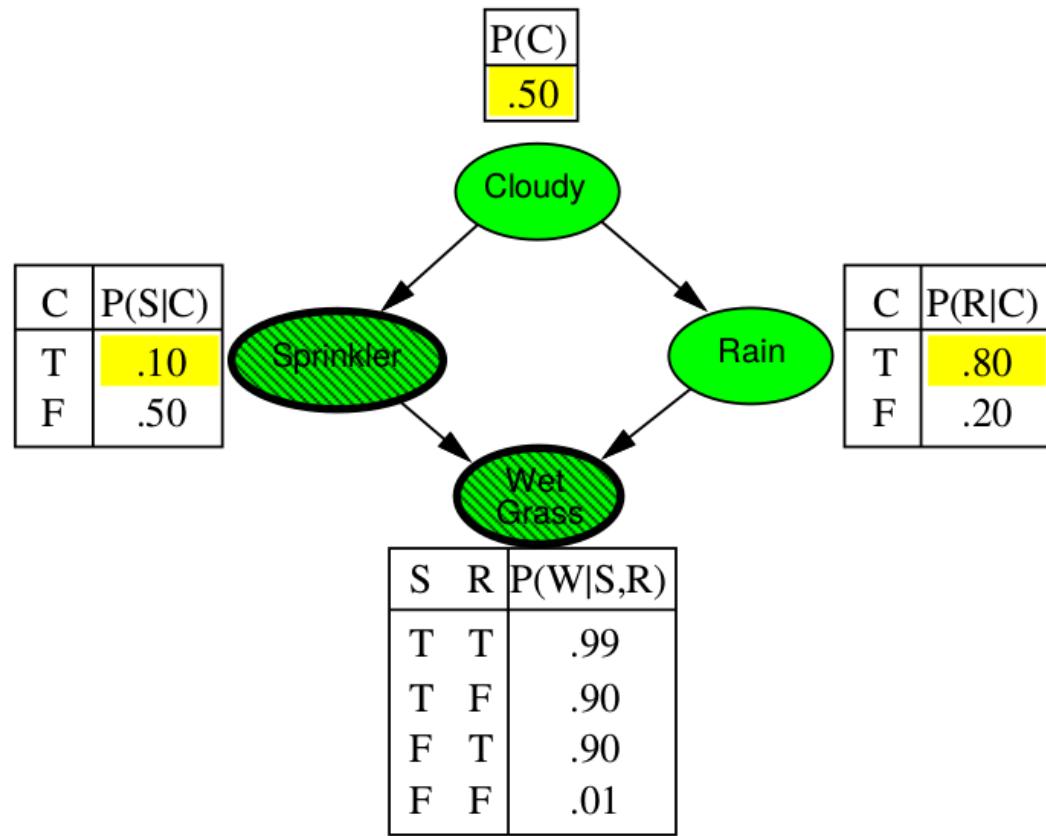
$$w = 1.0$$



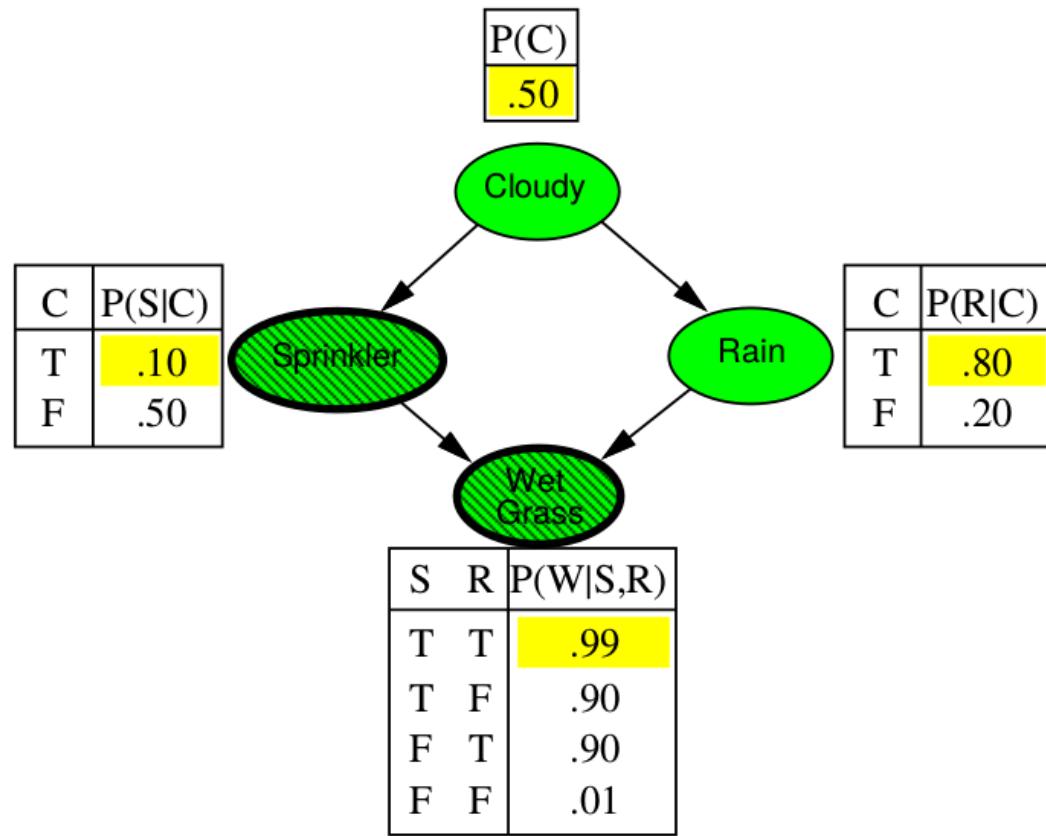
$$w = 1.0$$



$$w = 1.0$$



$$w = 1.0 \times 0.1$$



$$w = 1.0 \times 0.1 \times 0.99 = 0.099$$

Analysis

The sampling probability for an event with likelihood weighting is

$$S_{WS}(x, e) = \prod_{i=1}^l P(x_i | \text{parents}(X_i)),$$

where the product is over the non-evidence variables. The weight for a given sample x, e is

$$w(x, e) = \prod_{i=1}^m P(e_i | \text{parents}(E_i)),$$

where the product is over the evidence variables.

The weighted sampling probability is

$$\begin{aligned} S_{WS}(x, e)w(x, e) &= \prod_{i=1}^l P(x_i | \text{parents}(X_i)) \prod_{i=1}^m P(e_i | \text{parents}(E_i)) \\ &= P(x, e) \end{aligned}$$

The estimated joint probability is computed as follows:

$$\begin{aligned}\hat{P}(x, e) &= N_{WS}(x, e)w(x, e)/N \\ &\approx S_{WS}(x, e)w(x, e) \\ &= P(x, e)\end{aligned}$$

From this, the estimated posterior probability is given by:

$$\begin{aligned}\hat{P}(x|e) &= \hat{P}(x, e)/\hat{P}(e) \\ &\approx P(x, e)/P(e) = P(x|e)\end{aligned}$$

Hence likelihood weighting returns **consistent** estimates.

- Performance **still degrades** with many evidence variables.
- A few samples have nearly all the total weight.

Comments

- Likelihood weighting is **good**:
 - The evidence is taken into account to generate a sample.
 - More of the samples will reflect the state of the world suggested by the evidence.
- Likelihood weighting **does not solve all problems**:
 - The evidence influences the choice of downstream variables, but not upstream ones.
- We would like to consider evidence when we sample **every variable**.

Gibbs sampling

Procedure

- Keep track of a full instance x_1, \dots, x_n .
- Start with an arbitrary instance consistent with the evidence.
- Sample one variable at a time, conditioned on all the rest, but keep the evidence fixed.
- Keep repeating this for a long time.

The sampling process settles into a **dynamic equilibrium** in which the long-run fraction of time spent in each state is exactly proportional to its posterior probability.

```

function GIBBS-ASK( $X, \mathbf{e}, bn, N$ ) returns an estimate of  $\mathbf{P}(X|\mathbf{e})$ 
  local variables:  $\mathbf{N}$ , a vector of counts for each value of  $X$ , initially zero
     $\mathbf{Z}$ , the nonevidence variables in  $bn$ 
     $\mathbf{x}$ , the current state of the network, initially copied from  $\mathbf{e}$ 

  initialize  $\mathbf{x}$  with random values for the variables in  $\mathbf{Z}$ 
  for  $j = 1$  to  $N$  do
    for each  $Z_i$  in  $\mathbf{Z}$  do
      set the value of  $Z_i$  in  $\mathbf{x}$  by sampling from  $\mathbf{P}(Z_i|mb(Z_i))$ 
       $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$  where  $x$  is the value of  $X$  in  $\mathbf{x}$ 
  return NORMALIZE( $\mathbf{N}$ )

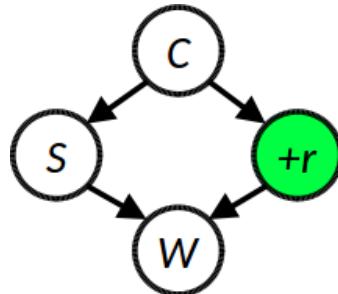
```

Rationale

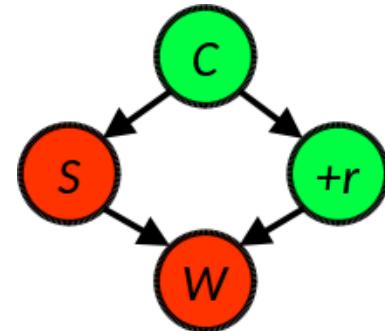
- Both upstream and downstream variables condition on evidence.
- In contrast, likelihood weighting only conditions on upstream evidence, and hence the resulting weights might be very small.

Example

- 1) Fix the evidence.

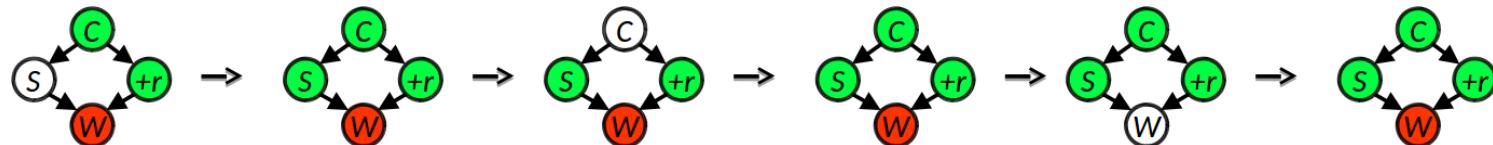


- 2) Randomly initialize the other variables



- 3) Repeat

- Choose a non-evidence variable X .
- Resample X from $P(X|\text{all other variables})$.



Demo

See code/lecture6-gibbs.ipynb.

MCMC

- Gibbs sampling is a special case of a more general set of methods called **Markov chain Monte Carlo** (MCMC) methods.
- Metropolis-Hastings is one of the most famous MCMC methods.
 - Gibbs sampling is a special case of Metropolis-Hastings.

Summary

- Exact inference by variable elimination .
 - NP-complete on general graphs, but polynomial on polytrees.
 - space = time, very sensitive to topology.
- Approximate inference gives reasonable estimates of the true posterior probabilities in a network and can cope with much larger networks than can exact algorithms.
 - LW does poorly when there is lots of evidence.
 - LW and GS generally insensitive to topology.
 - Convergence can be slow with probabilities close to 1 or 0.
 - Can handle arbitrary combinations of discrete and continuous variables.
- Want to know more about sampling methods? Follow [MATH2022](#).

References

- Cooper, Gregory F. "The computational complexity of probabilistic inference using Bayesian belief networks." *Artificial intelligence* 42.2-3 (1990): 393-405.

Introduction to Artificial Intelligence

Lecture 7: Reasoning over time

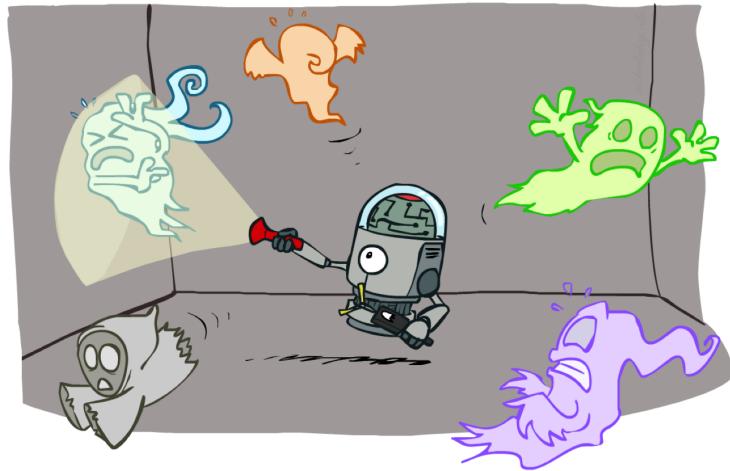
Prof. Gilles Louppe
g.louppe@uliege.be

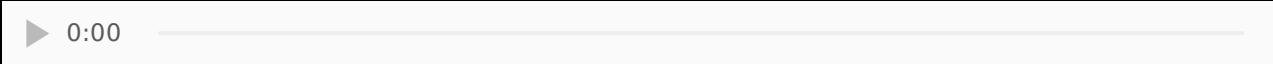


Today

Maintain a **belief state** about the world, and update it as time passes and evidence is collected.

- Markov models
 - Markov processes
 - Inference tasks
 - Prediction
 - Filtering
 - Smoothing
 - Most likely explanation
 - Hidden Markov models
- Filters
 - Kalman filter
 - Particle filter





Pacman revenge: How to make good use of the sonar readings?

Markov models

Reasoning over time

Often, we want to **reason about a sequence** of observations.

- Speech recognition
- Robot localization
- User attention
- Medical monitoring.

For this reason, we need to introduce **time** (or space) in our model.

Modelling time

Consider the world as a **discrete** series of **time slices**, each of which contains a set of random variables:

- \mathbf{X}_t denotes the set of **unobservable** state variables at time t .
- \mathbf{E}_t denotes the set of **observable** evidence variables at time t .

We specify:

- a prior $P(\mathbf{X}_0)$ that defines our initial belief state over hidden state variables.
- a **transition model** $P(\mathbf{X}_t | \mathbf{X}_{0:t-1})$ (for $t > 0$) that defines the probability distribution over the latest state variables, given the previous (unobserved) values.
- a **sensor model** $P(\mathbf{E}_t | \mathbf{X}_{0:t}, \mathbf{E}_{0:t-1})$ (for $t > 0$) that defines the probability distribution over the latest evidence variables, given all previous (observed and unobserved) values.

Markov processes

Markov assumption

- The current state of the world depends only on its immediate previous state(s).
- i.e., \mathbf{X}_t depends on only a bounded subset of $\mathbf{X}_{0:t-1}$.

Random processes that satisfy this assumption are called **Markov processes**.

First-order Markov processes

- Markov processes such that

$$P(\mathbf{X}_t | \mathbf{X}_{0:t-1}) = P(\mathbf{X}_t | \mathbf{X}_{t-1}).$$

- i.e., \mathbf{X}_t and $\mathbf{X}_{0:t-2}$ are conditionally independent given \mathbf{X}_{t-1} .

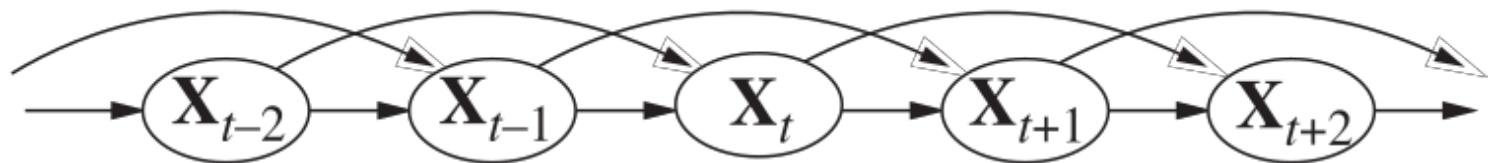


Second-order Markov processes

- Markov processes such that

$$P(\mathbf{X}_t | \mathbf{X}_{0:t-1}) = P(\mathbf{X}_t | \mathbf{X}_{t-2}, \mathbf{X}_{t-1}).$$

- i.e., \mathbf{X}_t and $\mathbf{X}_{0:t-3}$ are conditionally independent given \mathbf{X}_{t-1} and \mathbf{X}_{t-2} .



Sensor Markov assumption

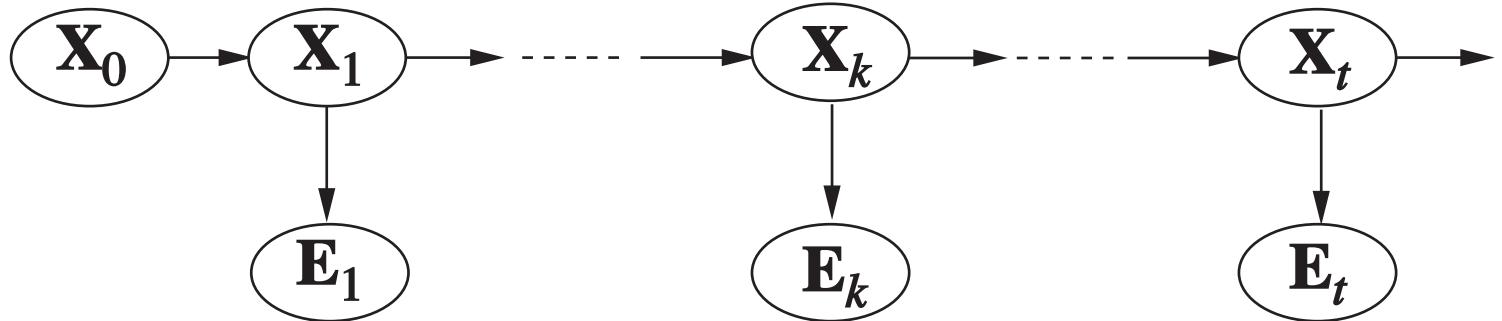
- Additionally, we make a (first-order) **sensor Markov assumption**:

$$P(\mathbf{E}_t | \mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = P(\mathbf{E}_t | \mathbf{X}_t)$$

Stationarity assumption

- The transition and the sensor models are the same for all t (i.e., the laws of physics do not change with time).

Joint distribution

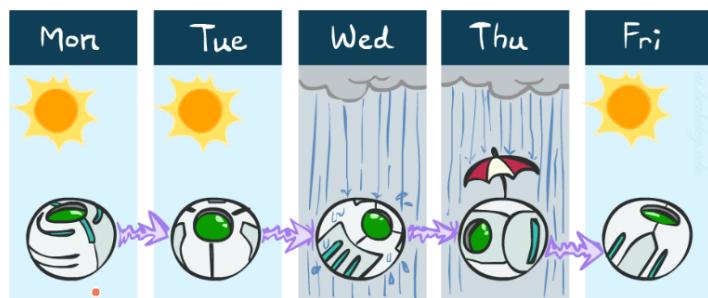
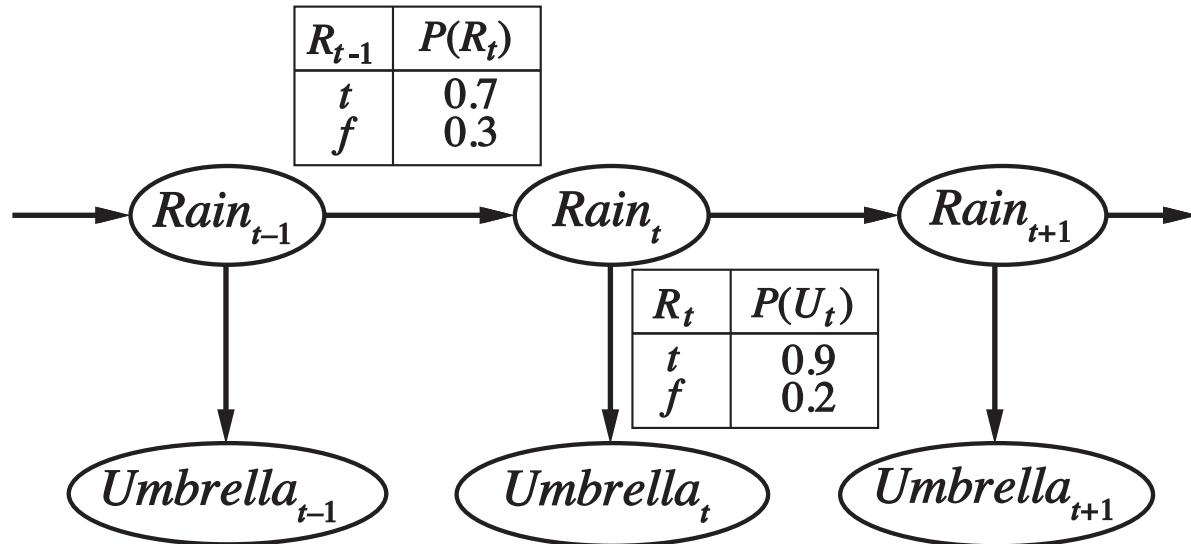


A Markov process can be described as a **growable** Bayesian network, unrolled infinitely through time, with a specified **restricted structure** between time steps.

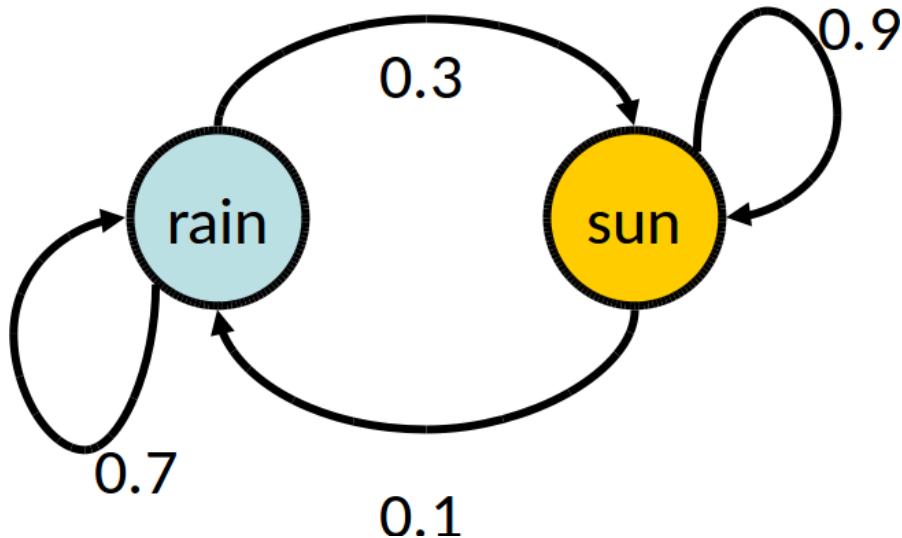
Therefore, the **joint distribution** of all variables up to t in a (first-order) Markov process is

$$P(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = P(\mathbf{X}_0) \prod_{i=1}^t P(\mathbf{X}_i | \mathbf{X}_{i-1}) P(\mathbf{E}_i | \mathbf{X}_i).$$

Example: Will you take your umbrella today?



- $P(Umbrella_t | Rain_t)$?
- $P(Rain_t | Umbrella_{0:t-1})$?
- $P(Rain_{t+2} | Rain_t)$?

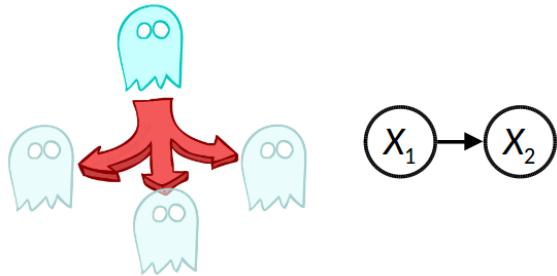


The transition model $P(\text{Rain}_t | \text{Rain}_{t-1})$ can equivalently be represented by a state transition diagram.

Inference tasks

- **Prediction:** $P(\mathbf{X}_{t+k} | \mathbf{e}_{1:t})$ for $k > 0$
 - Computing the posterior distribution over future states.
 - Used for evaluation of possible action sequences.
- **Filtering:** $P(\mathbf{X}_t | \mathbf{e}_{1:t})$
 - Filtering is what a rational agent does to keep track of the current hidden state \mathbf{X}_t , its **belief state**, so that rational decisions can be made.
- **Smoothing:** $P(\mathbf{X}_k | \mathbf{e}_{1:t})$ for $0 \leq k < t$
 - Computing the posterior distribution over past states.
 - Used for building better estimates, since it incorporates more evidence.
 - Essential for learning.
- **Most likely explanation:** $\arg \max_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t} | \mathbf{e}_{1:t})$
 - Decoding with a noisy channel, speech recognition, etc.

Base cases



$$\begin{aligned} P(\mathbf{X}_2) &= \sum_{\mathbf{x}_1} P(\mathbf{X}_2, \mathbf{x}_1) \\ &= \sum_{\mathbf{x}_1} P(\mathbf{x}_1)P(\mathbf{X}_2|\mathbf{x}_1) \end{aligned}$$

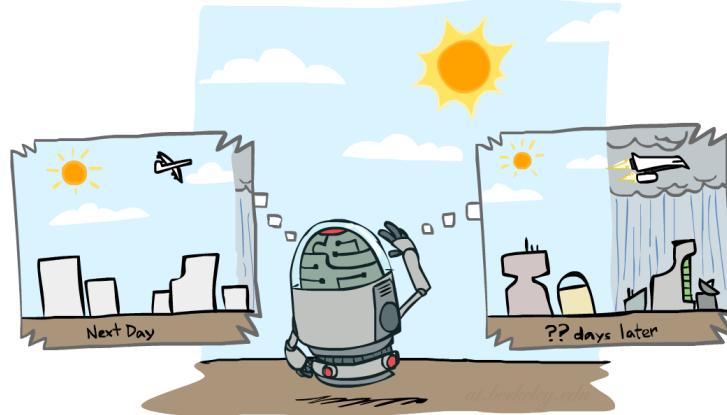
(Predict) Push $P(\mathbf{X}_1)$ forward through the transition model.



$$\begin{aligned} P(\mathbf{X}_1|\mathbf{e}_1) &= \frac{P(\mathbf{X}_1, \mathbf{e}_1)}{P(\mathbf{e}_1)} \\ &\propto P(\mathbf{X}_1, \mathbf{e}_1) \\ &= P(\mathbf{X}_1)P(\mathbf{e}_1|\mathbf{X}_1) \end{aligned}$$

(Update) Update $P(\mathbf{X}_1)$ with the evidence \mathbf{e}_1 , given the sensor model.

Prediction



To predict the future $P(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$:

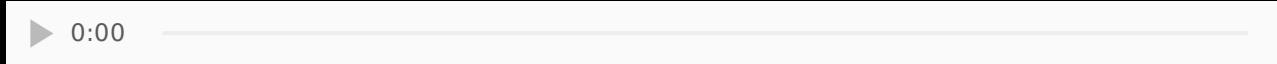
- Push the prior belief state $P(\mathbf{X}_t|\mathbf{e}_{1:t})$ through the transition model:

$$P(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t})$$

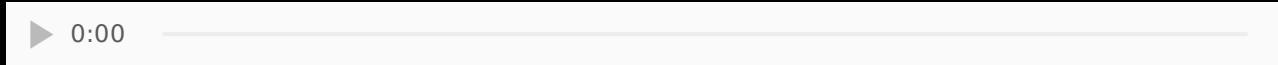
- Repeat up to $t + k$, using $P(\mathbf{X}_{t+k-1}|\mathbf{e}_{1:t})$ to compute $P(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$.



Basic dynamics (Ghostbusters)



Circular dynamics (Ghostbusters)



Whirlpool dynamics (Ghostbusters)

<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
<0.01	<0.01	1.00	<0.01	<0.01	<0.01
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01

$T = 1$

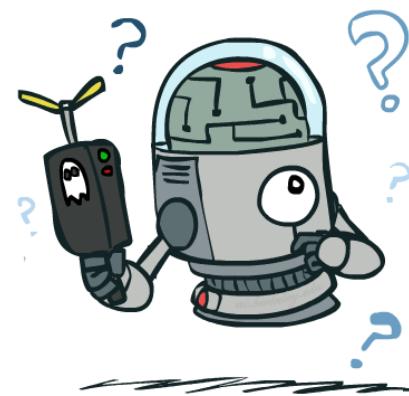
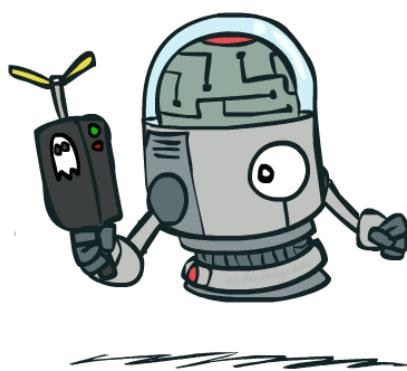
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
<0.01	<0.01	0.06	<0.01	<0.01	<0.01
<0.01	0.76	0.06	0.06	<0.01	<0.01
<0.01	<0.01	0.06	<0.01	<0.01	<0.01

$T = 2$

0.05	0.01	0.05	<0.01	<0.01	<0.01
0.02	0.14	0.11	0.35	<0.01	<0.01
0.07	0.03	0.05	<0.01	0.03	<0.01
0.03	0.03	<0.01	<0.01	<0.01	<0.01

$T = 5$

...



As time passes, uncertainty "accumulates" if we do not accumulate new evidence.

Stationary distributions

What if $t \rightarrow \infty$?

- For most chains, the influence of the initial distribution gets lesser and lesser over time.
- Eventually, the distribution converges to a fixed point, called the **stationary distribution**.
- This distribution is such that

$$P(\mathbf{X}_\infty) = P(\mathbf{X}_{\infty+1}) = \sum_{\mathbf{x}_\infty} P(\mathbf{X}_{\infty+1} | \mathbf{x}_\infty) P(\mathbf{x}_\infty)$$

Example

$$\begin{aligned} P(\mathbf{X}_\infty = \text{sun}) &= P(\mathbf{X}_{\infty+1} = \text{sun}) \\ &= P(\mathbf{X}_{\infty+1} = \text{sun} | \mathbf{X}_\infty = \text{sun})P(\mathbf{X}_\infty = \text{sun}) \\ &\quad + P(\mathbf{X}_{\infty+1} = \text{sun} | \mathbf{X}_\infty = \text{rain})P(\mathbf{X}_\infty = \text{rain}) \\ &= 0.9P(\mathbf{X}_\infty = \text{sun}) + 0.3P(\mathbf{X}_\infty = \text{rain}) \end{aligned}$$

Therefore, $P(\mathbf{X}_\infty = \text{sun}) = 3P(\mathbf{X}_\infty = \text{rain})$.

Which implies that $P(\mathbf{X}_\infty = \text{sun}) = \frac{3}{4}$ and $P(\mathbf{X}_\infty = \text{rain}) = \frac{1}{4}$.

Filtering

0.05	0.01	0.05	<0.01	<0.01	<0.01
0.02	0.14	0.11	0.35	<0.01	<0.01
0.07	0.03	0.05	<0.01	0.03	<0.01
0.03	0.03	<0.01	<0.01	<0.01	<0.01

Before observation

<0.01	<0.01	<0.01	<0.01	0.02	<0.01
<0.01	<0.01	<0.01	0.83	0.02	<0.01
<0.01	<0.01	0.11	<0.01	<0.01	<0.01
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01

After observation

What if we collect new observations? Beliefs get reweighted, and uncertainty "decreases":

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t})$$

Bayes filter

An agent maintains a **belief state** estimate $P(\mathbf{X}_t | \mathbf{e}_{1:t})$ and updates it as new evidences \mathbf{e}_{t+1} are collected.

Recursive Bayesian estimation: $P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, P(\mathbf{X}_t | \mathbf{e}_{1:t}))$

- (Predict step): Project the current belief state forward from t to $t + 1$ through the transition model.
- (Update step): Update this new state using the evidence \mathbf{e}_{t+1} .

$$\begin{aligned}
P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) &= P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \\
&= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) \\
&= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) \\
&= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t | \mathbf{e}_{1:t}) \\
&= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t})
\end{aligned}$$

where

- the normalization constant

$$\alpha = \frac{1}{P(\mathbf{e}_{t+1} | \mathbf{e}_{1:t})} = 1 / \sum_{\mathbf{x}_{t+1}} P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) P(\mathbf{x}_{t+1} | \mathbf{e}_{1:t})$$

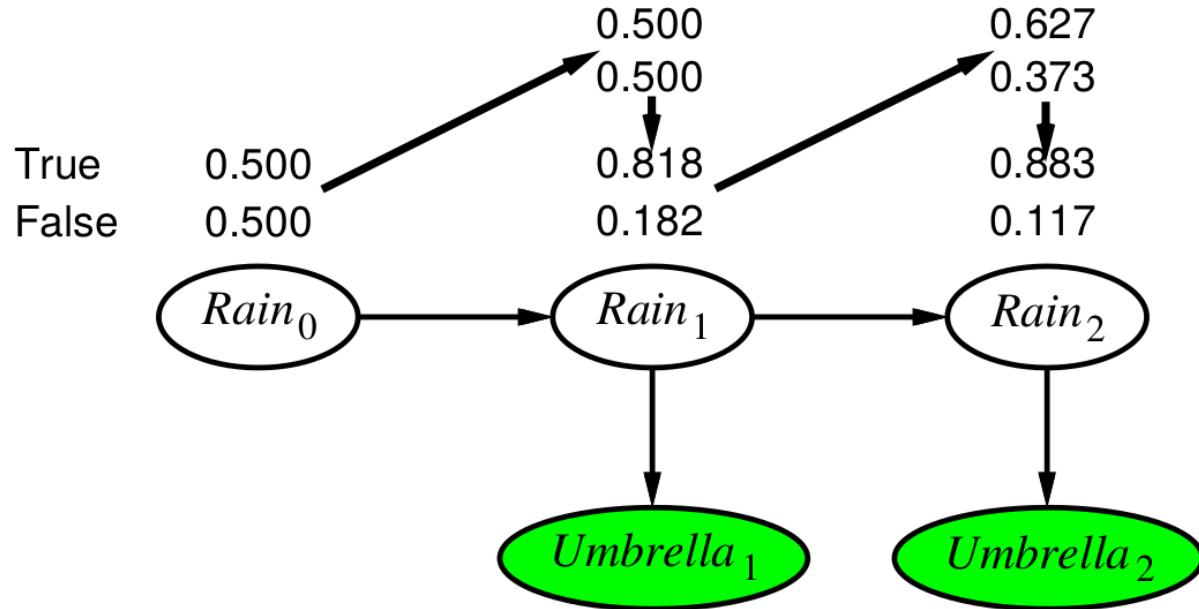
is used to make probabilities sum to 1;

- in the last expression, the first and second terms are given by the model while the third is obtained recursively.

We can think of $P(\mathbf{X}_t | \mathbf{e}_{1:t})$ as a message $\mathbf{f}_{1:t}$ that is propagated forward along the sequence, modified by each transition and updated by each new observation.

- Thus, the process can be implemented as $\mathbf{f}_{1:t+1} = \alpha \text{forward}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1})$.
- The complexity of a forward update is constant (in time and space) with t .

Example

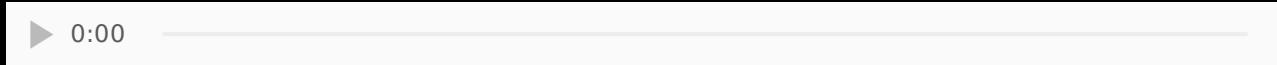


$$R_{t-1} \quad P(R_t)$$

<i>true</i>	0.7
<i>false</i>	0.3

$$R_t \quad P(U_t)$$

<i>true</i>	0.9
<i>false</i>	0.2



Ghostbusters with a Bayes filter

Smoothing

We want to compute $P(\mathbf{X}_k | \mathbf{e}_{1:t})$ for $0 \leq k < t$.

Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$ and $\mathbf{e}_{k+1:t}$. Then,

$$\begin{aligned} P(\mathbf{X}_k | \mathbf{e}_{1:t}) &= P(\mathbf{X}_k | \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\ &= \alpha P(\mathbf{X}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{e}_{1:k}) \\ &= \alpha P(\mathbf{X}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{X}_k). \end{aligned}$$

Let the **backward** message $\mathbf{b}_{k+1:t}$ correspond to $P(\mathbf{e}_{k+1:t} | \mathbf{X}_k)$. Then,

$$P(\mathbf{X}_k | \mathbf{e}_{1:t}) = \alpha \mathbf{f}_{1:k} \times \mathbf{b}_{k+1:t},$$

where \times is a pointwise multiplication of vectors.

This backward message can be computed using backwards recursion:

$$\begin{aligned} P(\mathbf{e}_{k+1:t} | \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \end{aligned}$$

The first and last factors are given by the model. The second factor is obtained recursively. Therefore,

$$\mathbf{b}_{k+1:t} = \text{backward}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1}).$$

Forward-backward algorithm

function FORWARD-BACKWARD(\mathbf{ev} , $prior$) **returns** a vector of probability distributions

inputs: \mathbf{ev} , a vector of evidence values for steps $1, \dots, t$

prior, the prior distribution on the initial state, $\mathbf{P}(\mathbf{X}_0)$

local variables: \mathbf{fv} , a vector of forward messages for steps $0, \dots, t$

\mathbf{b} , a representation of the backward message, initially all 1s

\mathbf{sv} , a vector of smoothed estimates for steps $1, \dots, t$

$\mathbf{fv}[0] \leftarrow prior$

for $i = 1$ **to** t **do**

$\mathbf{fv}[i] \leftarrow \text{FORWARD}(\mathbf{fv}[i - 1], \mathbf{ev}[i])$

for $i = t$ **downto** 1 **do**

$\mathbf{sv}[i] \leftarrow \text{NORMALIZE}(\mathbf{fv}[i] \times \mathbf{b})$

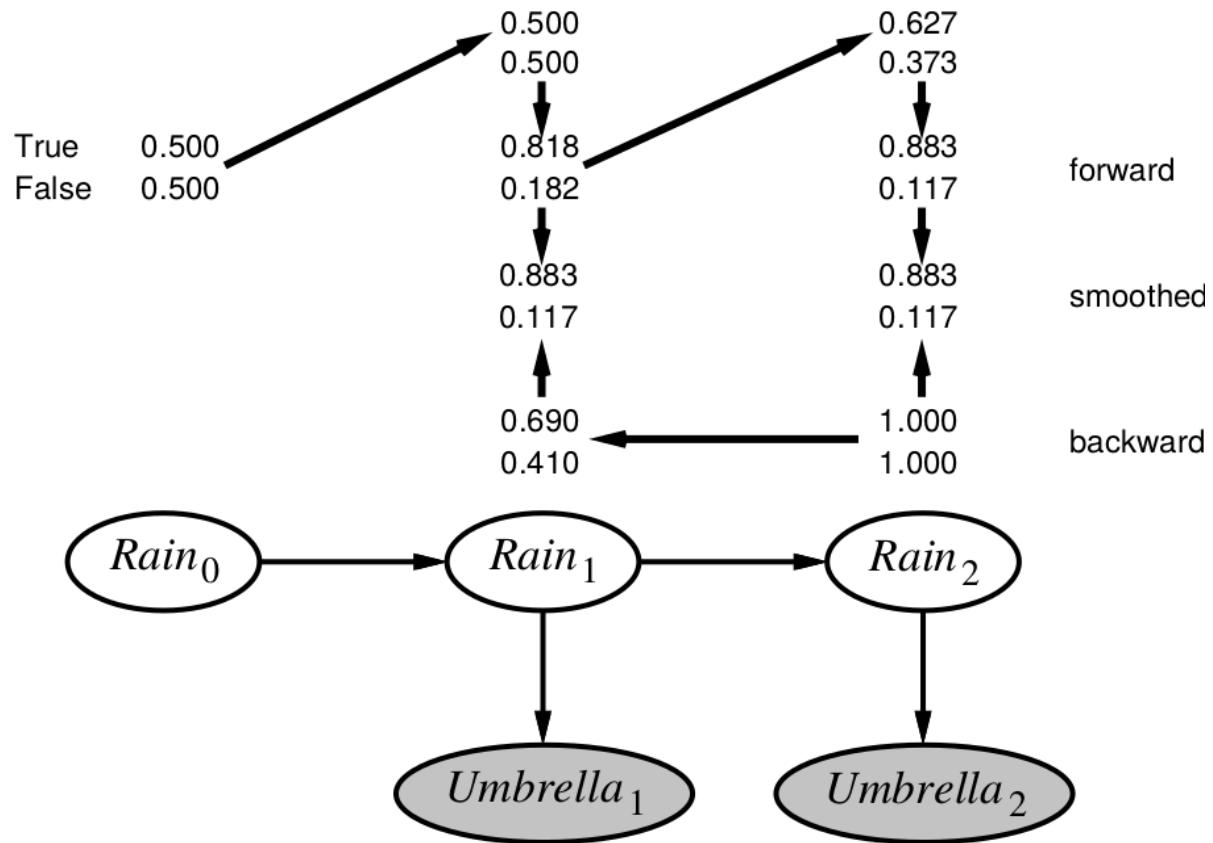
$\mathbf{b} \leftarrow \text{BACKWARD}(\mathbf{b}, \mathbf{ev}[i])$

return \mathbf{sv}

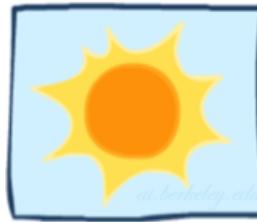
Complexity:

- Smoothing for a particular time step k takes: $O(t)$
- Smoothing a whole sequence (because of caching): $O(t)$

Example



Most likely explanation



Suppose that $[true, true, false, true, true]$ is the umbrella sequence.

What is the weather sequence that is the most likely to explain this?

- Does the absence of umbrella at day 3 means it wasn't raining?
- Or did the director forget to bring it?
- If it didn't rain on day 3, perhaps it didn't rain on day 4 either, but the director brought the umbrella just in case?

Among all 2^5 sequences, is there an (efficient) way to find the most likely one?

- The most likely sequence **is not** the sequence of the most likely states!
- The most likely path to each \mathbf{x}_{t+1} , is the most likely path to **some** \mathbf{x}_t plus one more step. Therefore,

$$\begin{aligned} & \max_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t}, \mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) \\ &= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} (P(\mathbf{X}_{t+1} | \mathbf{x}_t) \max_{\mathbf{x}_{1:t-1}} P(\mathbf{x}_{1:t-1}, \mathbf{x}_t | \mathbf{e}_{1:t})) \end{aligned}$$

- Identical to filtering, except that the forward message $\mathbf{f}_{1:t} = P(\mathbf{X}_t | \mathbf{e}_{1:t})$ is replaced with

$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_{1:t-1}} P(\mathbf{x}_{1:t-1}, \mathbf{X}_t | \mathbf{e}_{1:t}),$$

where $\mathbf{m}_{1:t}(i)$ gives the probability of the most likely path to state i .

- The update has its sum replaced by max, resulting in the **Viterbi algorithm**:

$$\mathbf{m}_{1:t+1} = \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t) \mathbf{m}_{1:t}$$

Example

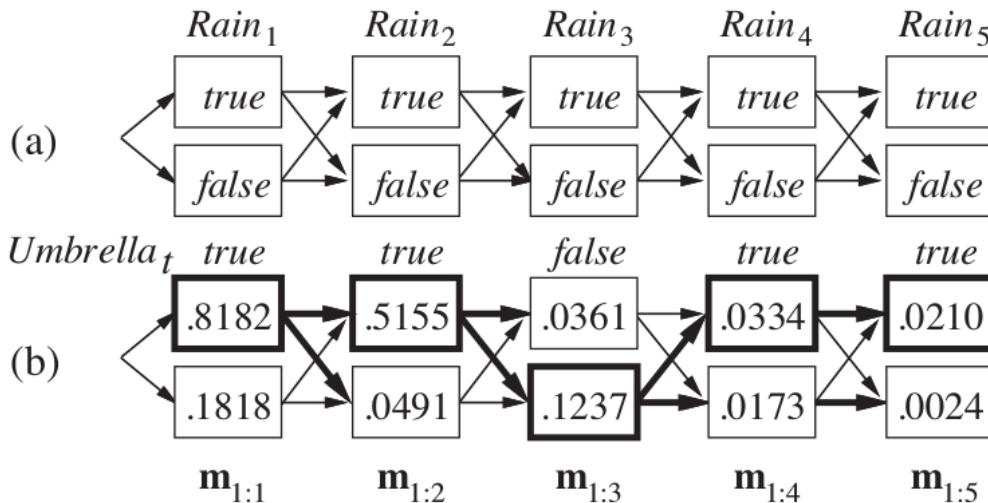


Figure 15.5 (a) Possible state sequences for $Rain_t$ can be viewed as paths through a graph of the possible states at each time step. (States are shown as rectangles to avoid confusion with nodes in a Bayes net.) (b) Operation of the Viterbi algorithm for the umbrella observation sequence [true, true, false, true, true]. For each t , we have shown the values of the message $\mathbf{m}_{1:t}$, which gives the probability of the best sequence reaching each state at time t . Also, for each state, the bold arrow leading into it indicates its best predecessor as measured by the product of the preceding sequence probability and the transition probability. Following the bold arrows back from the most likely state in $\mathbf{m}_{1:5}$ gives the most likely sequence.

Hidden Markov models

So far, we described Markov processes over arbitrary sets of state variables \mathbf{X}_t and evidence variables \mathbf{E}_t .

- A **hidden Markov model** (HMM) is a Markov process in which the state \mathbf{X}_t and the evidence \mathbf{E}_t are both **single discrete** random variables.
 - $\mathbf{X}_t = X_t$, with domain $D_{X_t} = \{1, \dots, S\}$
 - $\mathbf{E}_t = E_t$, with domain $D_{E_t} = \{1, \dots, R\}$
- This restricted structure allows for a reformulation of the forward-backward algorithm in terms of matrix-vector operations.

Note on terminology

Some authors instead divide Markov models into two classes, depending on the observability of the system state:

- Observable system state: Markov chains
- Partially-observable system state: Hidden Markov models.

We follow here the terminology of the textbook.

Simplified matrix algorithms

- The prior $P(X_0)$ becomes a (normalized) column vector $\mathbf{f}_0 \in \mathbb{R}_+^S$.
- The transition model $P(X_t|X_{t-1})$ becomes an $S \times S$ transition matrix \mathbf{T} , such that

$$\mathbf{T}_{ij} = P(X_t = j | X_{t-1} = i).$$

- The sensor model $P(E_t|X_t)$ is defined as an $S \times R$ sensor matrix \mathbf{B} , such that

$$\mathbf{B}_{ij} = P(E_t = j | X_t = i).$$

- Let the observation matrix \mathbf{O}_t be a diagonal matrix whose elements corresponds to the column e_t of the sensor matrix \mathbf{B} .
- If we use column vectors to represent forward and backward messages, then we have

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^T \mathbf{f}_{1:t}$$

$$\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t},$$

where $\mathbf{b}_{t:t}$ is an all-one vector of size S .

- Therefore the forward-backward algorithm needs time $O(S^2 t)$ and space $O(St)$.

Example

Suppose that $[true, true, false, true, true]$ is the umbrella sequence.

$$\mathbf{f}_0 = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

$$\mathbf{T} = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

$$\mathbf{B} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

$$\mathbf{O}_1 = \mathbf{O}_2 = \mathbf{O}_4 = \mathbf{O}_5 = \begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix}$$

$$\mathbf{O}_3 = \begin{pmatrix} 0.1 & 0.0 \\ 0.0 & 0.8 \end{pmatrix}$$

See code/lecture7-forward-backward.ipynb for the execution.

Stationary distribution

The stationary distribution \mathbf{f} of a HMM is a distribution such that

$$\mathbf{f} = \mathbf{T}^T \mathbf{f}.$$

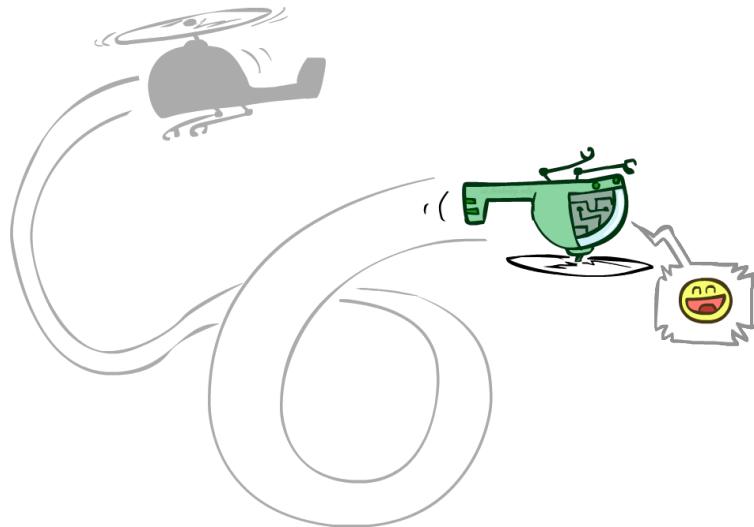
Therefore, the stationary distribution corresponds to the (normalized) eigenvector of the transposed transition matrix with an eigenvalue of 1 .

Applications

HMMs are used in many fields where the goal is to recover a data sequence that is not immediately observable, but other data that depend on the sequence are.

- Speech recognition (see Lecture 10)
- Speech synthesis
- Part-of-speech tagging
- Machine translation
- Robotics
- Computational finance
- Handwriting recognition
- Time series analysis
- Activity recognition

Filters



Suppose we want track the position and velocity of a robot from noisy observations collected over time.

Formally, we want to estimate **continuous** state variables such as

- the position \mathbf{X}_t of the robot at time t ,
- the velocity $\dot{\mathbf{X}}_t$ of the robot at time t .

We assume **discrete** time steps.

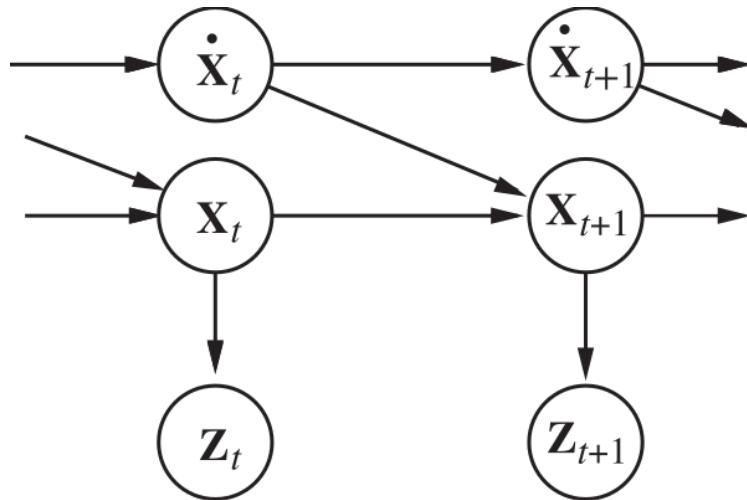
The Bayes filter similarly applies to **continuous** state and evidence variables \mathbf{X}_t and \mathbf{E}_t , in which case summations are replaced with integrals:

$$p(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha p(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) \int p(\mathbf{x}_{t+1} | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{e}_{1:t}) d\mathbf{x}_t$$

where the normalization constant is

$$\alpha = 1 / \int p(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) p(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}) d\mathbf{x}_{t+1}.$$

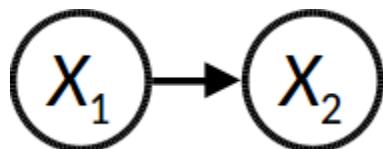
Kalman filter



The **Kalman filter** is a special case of the Bayes filter, which assumes:

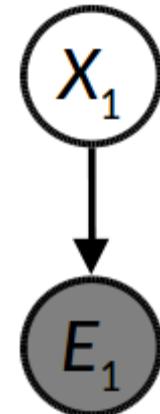
- Gaussian prior
- Linear Gaussian transition model
- Linear Gaussian sensor model

Linear Gaussian models



$$p(\mathbf{x}_{t+1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t+1}|\mathbf{Ax}_t + \mathbf{b}, \Sigma_{\mathbf{x}})$$

Transition model



$$p(\mathbf{e}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{e}_t|\mathbf{Cx}_t + \mathbf{d}, \Sigma_{\mathbf{e}})$$

Sensor model

Cheat sheet for Gaussian models (Bishop, 2006)

Given a marginal Gaussian distribution for \mathbf{x} and a linear Gaussian distribution for \mathbf{y} given \mathbf{x} in the form

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$$
$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{Ax} + \mathbf{b}, \mathbf{L}^{-1})$$

the marginal distribution of \mathbf{y} and the conditional distribution of \mathbf{x} given \mathbf{y} are given by

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^T)$$
$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\Sigma}(\mathbf{A}^T\mathbf{L}(\mathbf{y} - \mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}), \boldsymbol{\Sigma})$$

where

$$\boldsymbol{\Sigma} = (\boldsymbol{\Lambda} + \mathbf{A}^T \mathbf{L} \mathbf{A})^{-1}.$$

Filtering Gaussian distributions

- *Prediction step:*

If the distribution $p(\mathbf{x}_t | \mathbf{e}_{1:t})$ is Gaussian and the transition model $p(\mathbf{x}_{t+1} | \mathbf{x}_t)$ is linear Gaussian, then the one-step predicted distribution given by

$$p(\mathbf{x}_{t+1} | \mathbf{e}_{1:t}) = \int p(\mathbf{x}_{t+1} | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{e}_{1:t}) d\mathbf{x}_t$$

is also a Gaussian distribution.

- *Update step:*

If the prediction $p(\mathbf{x}_{t+1} | \mathbf{e}_{1:t})$ is Gaussian and the sensor model $p(\mathbf{e}_{t+1} | \mathbf{x}_{t+1})$ is linear Gaussian, then after conditioning on new evidence, the updated distribution

$$p(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) \propto p(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) p(\mathbf{x}_{t+1} | \mathbf{e}_{1:t})$$

is also a Gaussian distribution.

Therefore, for the Kalman filter, $p(\mathbf{x}_t | \mathbf{e}_{1:t})$ is a multivariate Gaussian distribution $\mathcal{N}(\mathbf{x}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ for all t .

- Filtering reduces to the computation of the parameters $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$.
- By contrast, for general (nonlinear, non-Gaussian) processes, the description of the posterior grows **unboundedly** as $t \rightarrow \infty$.

1D example

Gaussian random walk:

- Gaussian prior:

$$p(x_0) = \mathcal{N}(x_0 | \mu_0, \sigma_0^2)$$

- The transition model adds random perturbations of constant variance:

$$p(x_{t+1} | x_t) = \mathcal{N}(x_{t+1} | x_t, \sigma_x^2)$$

- The sensor model yields measurements with Gaussian noise of constant variance:

$$p(e_t | x_t) = \mathcal{N}(e_t | x_t, \sigma_e^2)$$

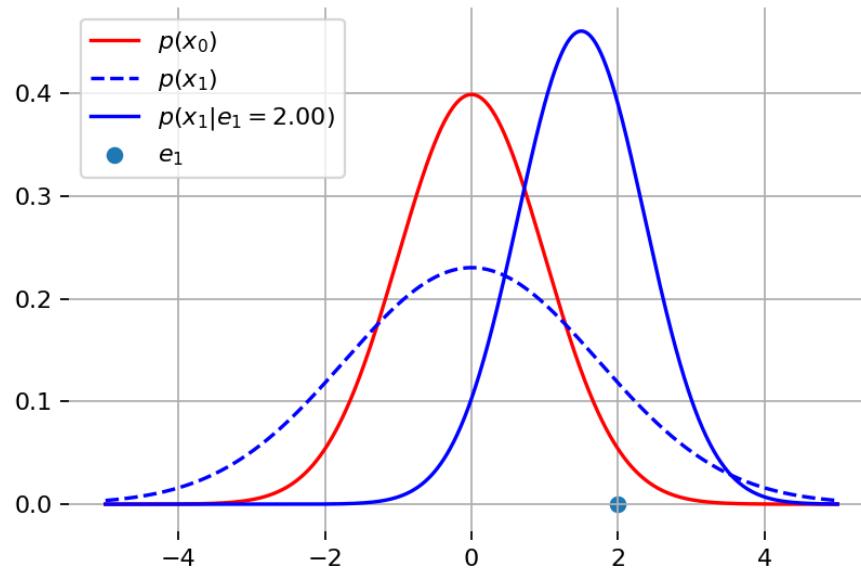
The one-step predicted distribution is given by

$$\begin{aligned} p(x_1) &= \int p(x_1|x_0)p(x_0)dx_0 \\ &= \alpha \int \exp\left(-\frac{1}{2}\frac{(x_1 - x_0)^2}{\sigma_x^2}\right) \exp\left(-\frac{1}{2}\frac{(x_0 - \mu_0)^2}{\sigma_0^2}\right) dx_0 \\ &= \alpha \int \exp\left(-\frac{1}{2}\frac{\sigma_0^2(x_1 - x_0)^2 + \sigma_x^2(x_0 - \mu_0)^2}{\sigma_0^2\sigma_x^2}\right) dx_0 \\ &\dots \text{ (simplify by completing the square)} \\ &= \alpha \exp\left(-\frac{1}{2}\frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2}\right) \\ &= \mathcal{N}(x_1|\mu_0, \sigma_0^2 + \sigma_x^2) \end{aligned}$$

Note that the same result can be obtained by using instead the Gaussian models identities.

For the update step, we need to condition on the observation at the first time step:

$$\begin{aligned}
 p(x_1|e_1) &= \alpha p(e_1|x_1)p(x_1) \\
 &= \alpha \exp\left(-\frac{1}{2} \frac{(e_1 - x_1)^2}{\sigma_e^2}\right) \exp\left(-\frac{1}{2} \frac{(x_1 - \mu_0)^2}{\sigma_0^2 + \sigma_x^2}\right) \\
 &= \alpha \exp\left(-\frac{1}{2} \frac{\left(x_1 - \frac{(\sigma_0^2 + \sigma_x^2)e_1 + \sigma_e^2\mu_0}{\sigma_0^2 + \sigma_x^2 + \sigma_e^2}\right)^2}{\frac{(\sigma_0^2 + \sigma_x^2)\sigma_e^2}{\sigma_0^2 + \sigma_x^2 + \sigma_e^2}}\right) \\
 &= \mathcal{N}\left(x_1 \middle| \frac{(\sigma_0^2 + \sigma_x^2)e_1 + \sigma_e^2\mu_0}{\sigma_0^2 + \sigma_x^2 + \sigma_e^2}, \frac{(\sigma_0^2 + \sigma_x^2)\sigma_e^2}{\sigma_0^2 + \sigma_x^2 + \sigma_e^2}\right)
 \end{aligned}$$



In summary, the update equations given a new evidence e_{t+1} are:

$$\mu_{t+1} = \frac{(\sigma_t^2 + \sigma_x^2)e_{t+1} + \sigma_e^2\mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_e^2}$$

$$\sigma_{t+1}^2 = \frac{(\sigma_t^2 + \sigma_x^2)\sigma_e^2}{\sigma_t^2 + \sigma_x^2 + \sigma_e^2}$$

General Kalman update

The same derivations generalize to multivariate normal distributions.

Assuming the transition and sensor models

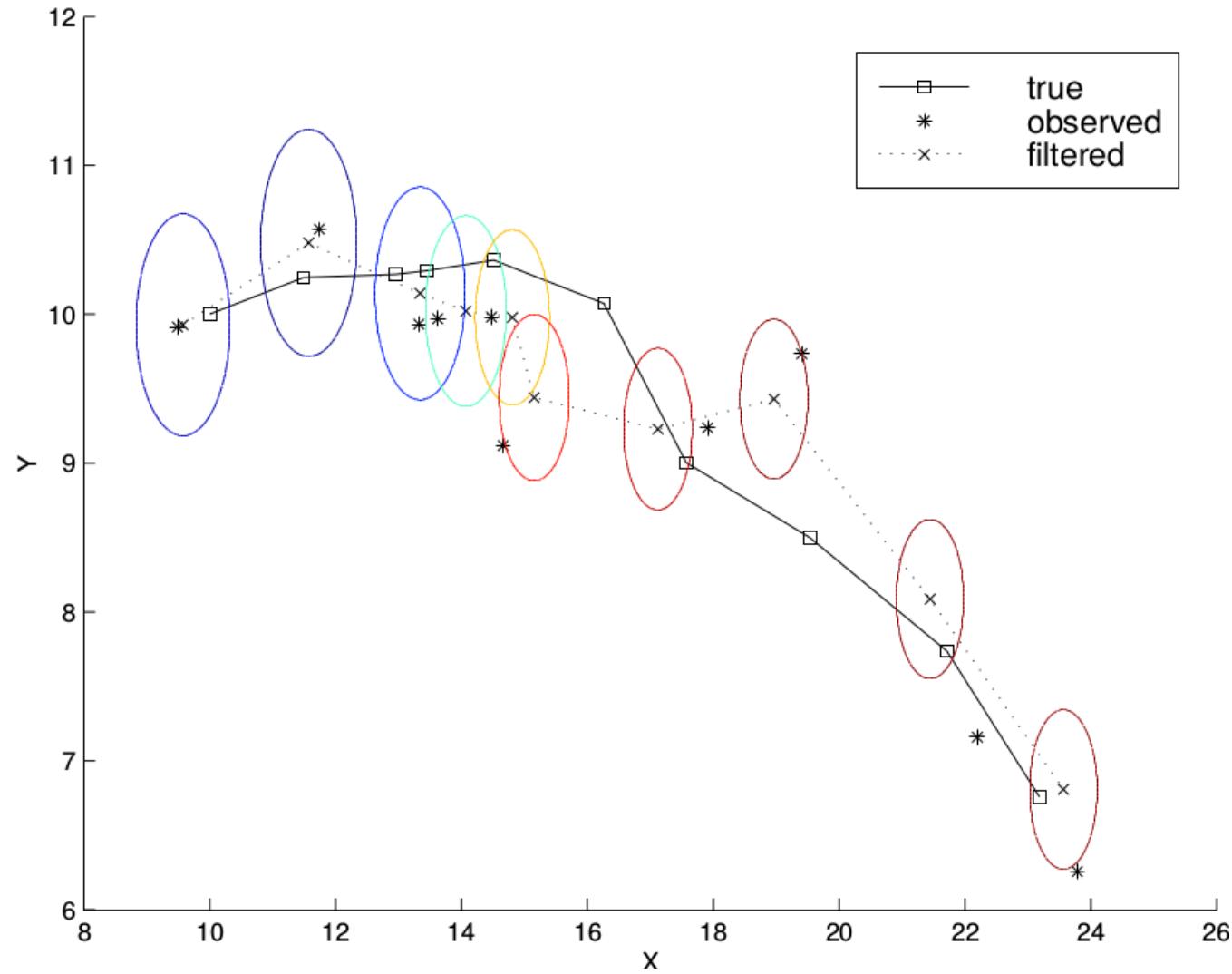
$$p(\mathbf{x}_{t+1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t+1} | \mathbf{F}\mathbf{x}_t, \boldsymbol{\Sigma}_x)$$
$$p(\mathbf{e}_t | \mathbf{x}_t) = \mathcal{N}(\mathbf{e}_t | \mathbf{H}\mathbf{x}_t, \boldsymbol{\Sigma}_e),$$

we arrive at the following general update equations:

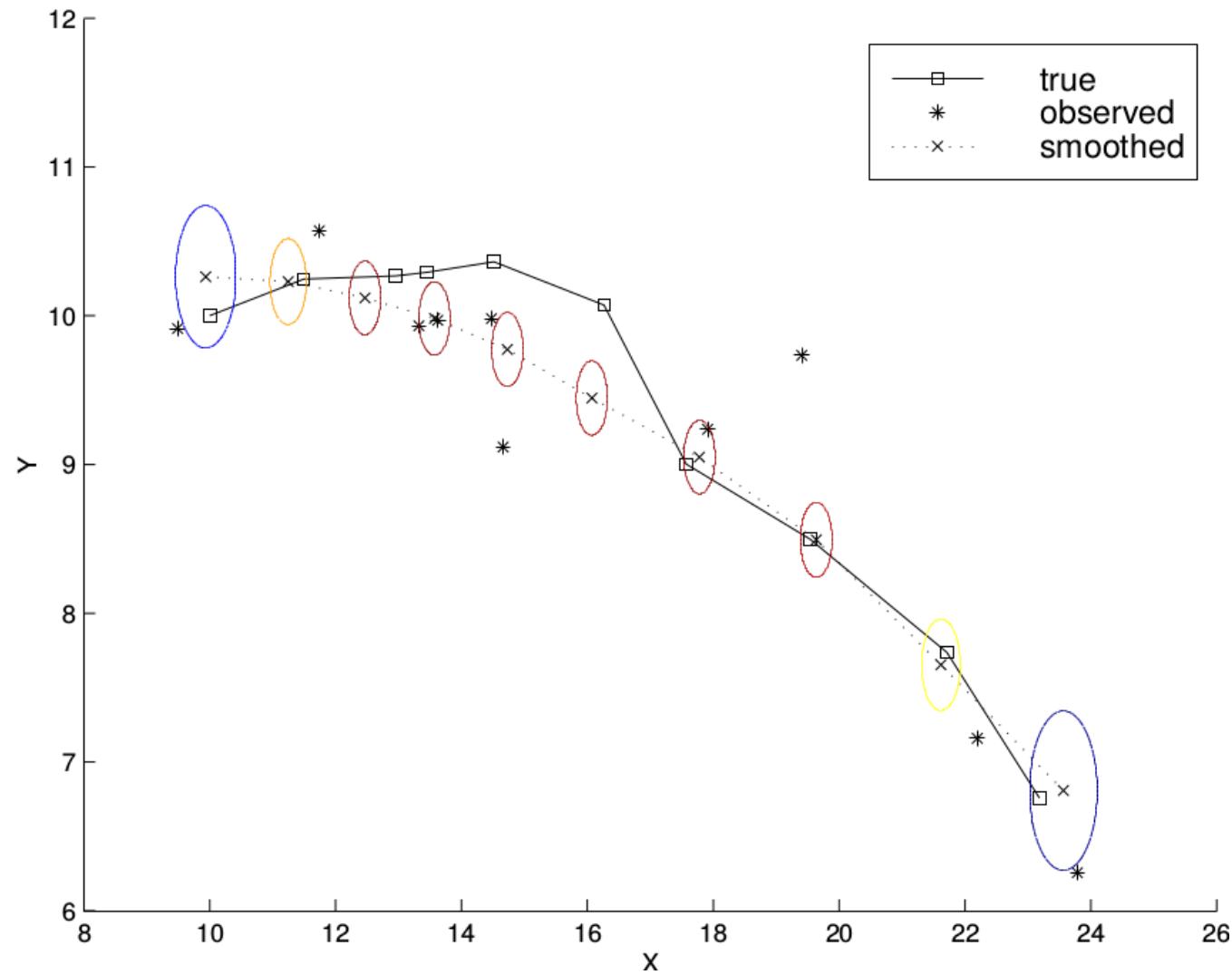
$$\mu_{t+1} = \mathbf{F}\mu_t + \mathbf{K}_{t+1}(\mathbf{e}_{t+1} - \mathbf{H}\mathbf{F}\mu_t)$$
$$\boldsymbol{\Sigma}_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^T + \boldsymbol{\Sigma}_x)$$
$$\mathbf{K}_{t+1} = (\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^T + \boldsymbol{\Sigma}_x)\mathbf{H}^T(\mathbf{H}(\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^T + \boldsymbol{\Sigma}_x)\mathbf{H}^T + \boldsymbol{\Sigma}_e)^{-1}$$

where \mathbf{K}_{t+1} is the Kalman gain matrix.

2D tracking: filtering



2D tracking: smoothing



Apollo guidance computer

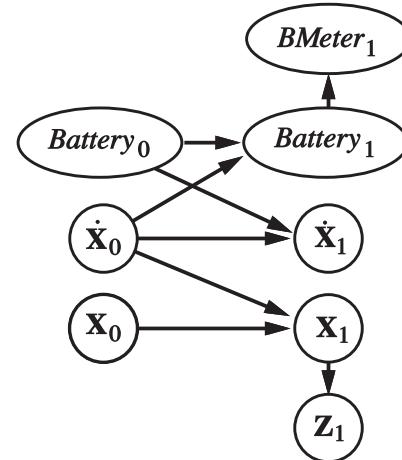
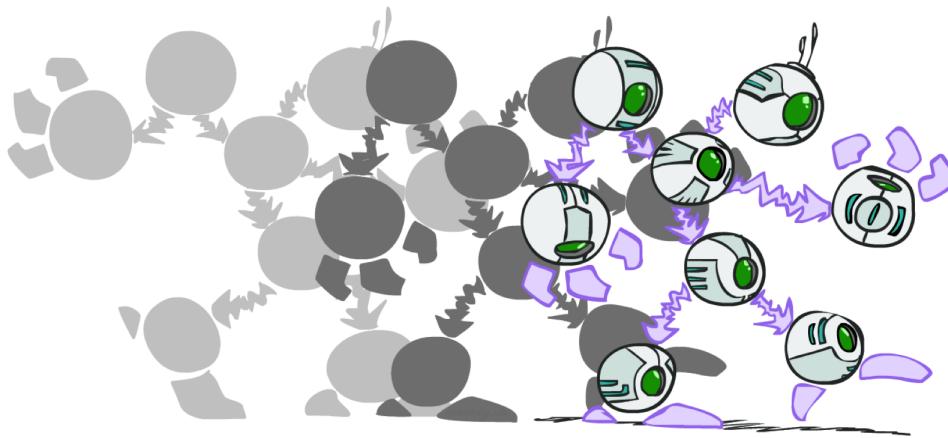
- The Kalman filter put man on the Moon, literally!
- The onboard guidance software of Saturn-V used a Kalman filter to merge new data with past position measurements to produce an optimal position estimate of the spacecraft.



```
# INCORP2 -- INCORPORATES THE COMPUTED STATE VECTOR DEVIATIONS INTO THE
# ESTIMATED STATE VECTOR.  THE STATE VECTOR UPDATED MAY BE FOR EITHER THE
# LEM OR THE CSM.  DETERMINED BY FLAG VEHUPFLG.  (ZERO = LEM) (1 = CSM)
#
# INPUT
#   PERMANENT STATE VECTOR FOR EITHER THE LEM OR CSM
#   VEHUPFLG = UPDATE VEHICLE C=LEM 1=CSM
#   W = ERROR TRANSITION MATRIX
#   DELTAX = COMPUTED STATE VECTOR DEVIATIONS
#   DMENFLG = SIZE OF W MATRIX (ZERO=6X6) (1=9X9)
#   GAMMA = SCALAR FOR INCORPORATION
#   ZI = VECTOR USED IN INCORPORATION
#   OMEGA = WEIGHTING VECTOR
#
# OUTPUT
#   UPDATED PERMANENT STATE VECTOR
```



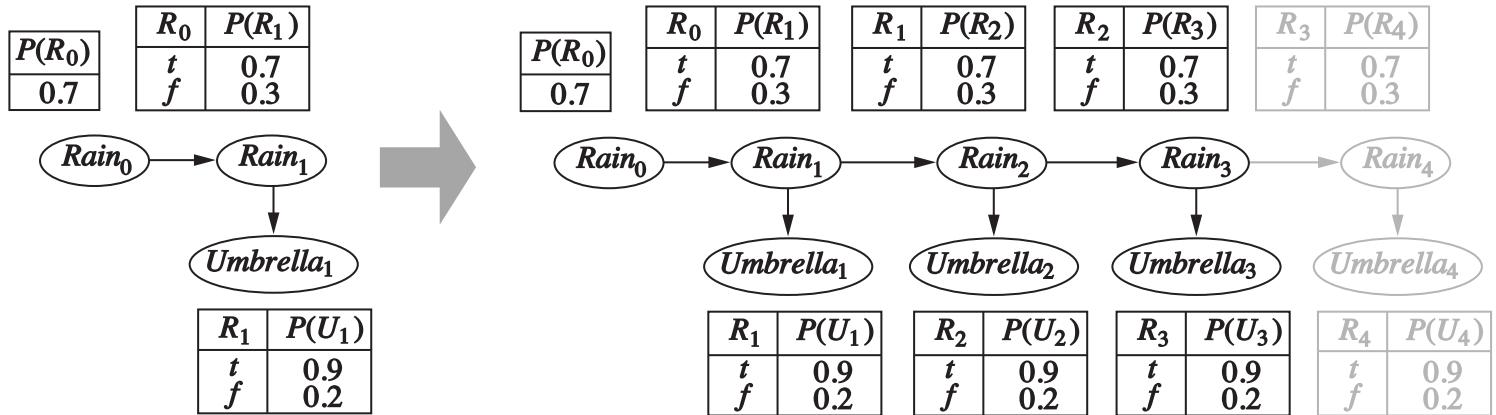
Dynamic Bayesian networks



Dynamics Bayesian networks (DBNs) can be used for tracking multiple variables over time, using multiple sources of evidence.

- Idea: Repeat a fixed Bayes net structure at each time t .
- Variables from time t condition on those from $t - 1$.
- DBNs are a generalization of HMMs and of the Kalman filter.

Exact inference



Unroll the network through time and run any exact inference algorithm (e.g., variable elimination)

- Problem: inference cost for each update grows with t .
- Rollup filtering: add slice $t+1$, sum out slice t using variable elimination.
 - Largest factor is $O(d^{n+k})$ and the total update cost per step is $O(nd^{n+k})$.
 - Better than HMMs, which is $O(d^{2n})$, but still **infeasible** for large numbers of variables.

Likelihood weighting

If exact inference in DBNs intractable, then let's use [approximate inference](#) instead.

- Likelihood weighting? Generated LW samples **pay no attention** to the evidence!
- The fraction of samples that remain close to the actual series of events drops exponentially with t .
- Therefore, the number of required samples for inference grows exponentially with t .

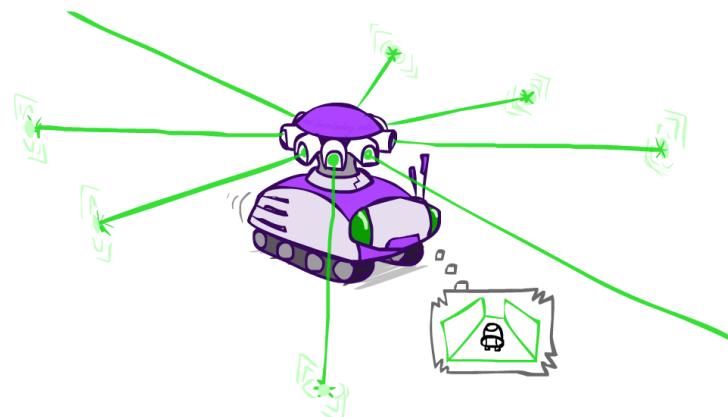
⇒ We need a better solution!

Particle filter

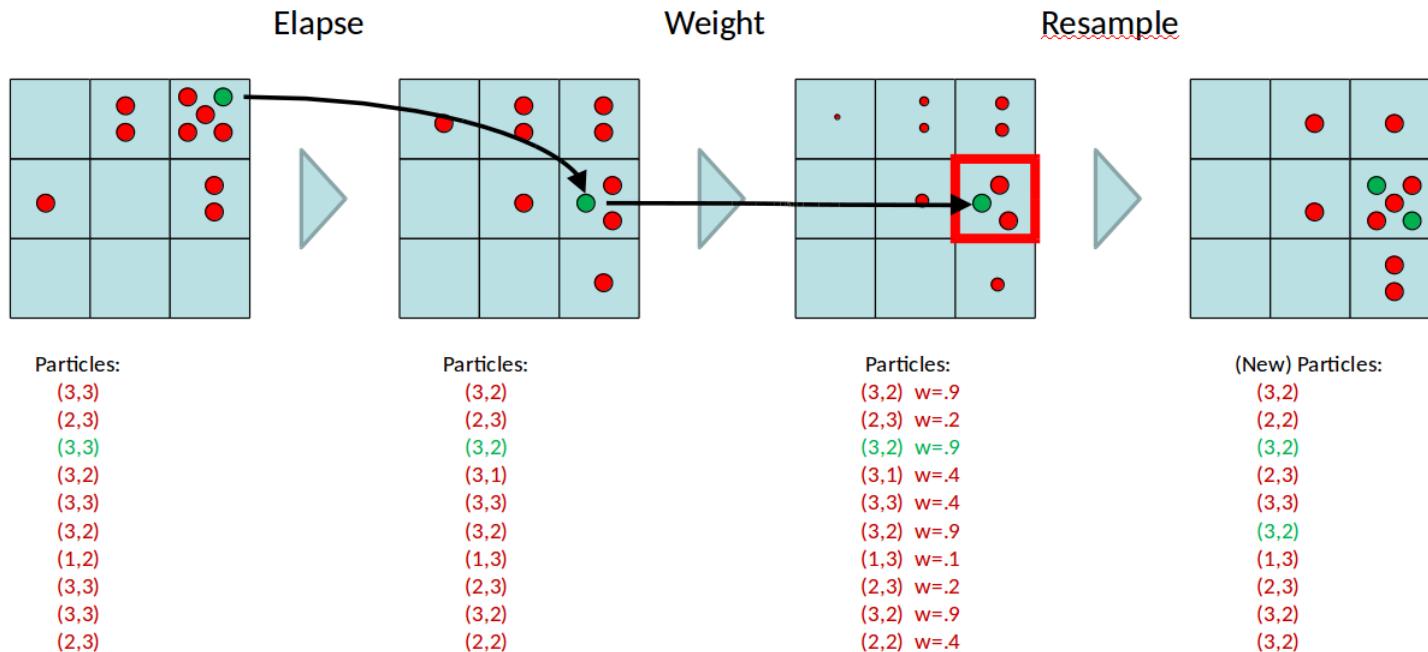
Basic idea:

- Maintain a finite population of samples, called **particles**.
 - The representation of our beliefs is a list of N particles.
- Ensure the particles track the high-likelihood regions of the state space.
- Throw away samples that have very low weight, according to the evidence.
- Replicate those that have high weight.

This scales to high dimensions!



Update cycle



function PARTICLE-FILTERING(\mathbf{e}, N, dbn) **returns** a set of samples for the next time step

inputs: \mathbf{e} , the new incoming evidence
 N , the number of samples to be maintained
 dbn , a DBN with prior $\mathbf{P}(\mathbf{X}_0)$, transition model $\mathbf{P}(\mathbf{X}_1|\mathbf{X}_0)$, sensor model $\mathbf{P}(\mathbf{E}_1|\mathbf{X}_1)$

persistent: S , a vector of samples of size N , initially generated from $\mathbf{P}(\mathbf{X}_0)$
local variables: W , a vector of weights of size N

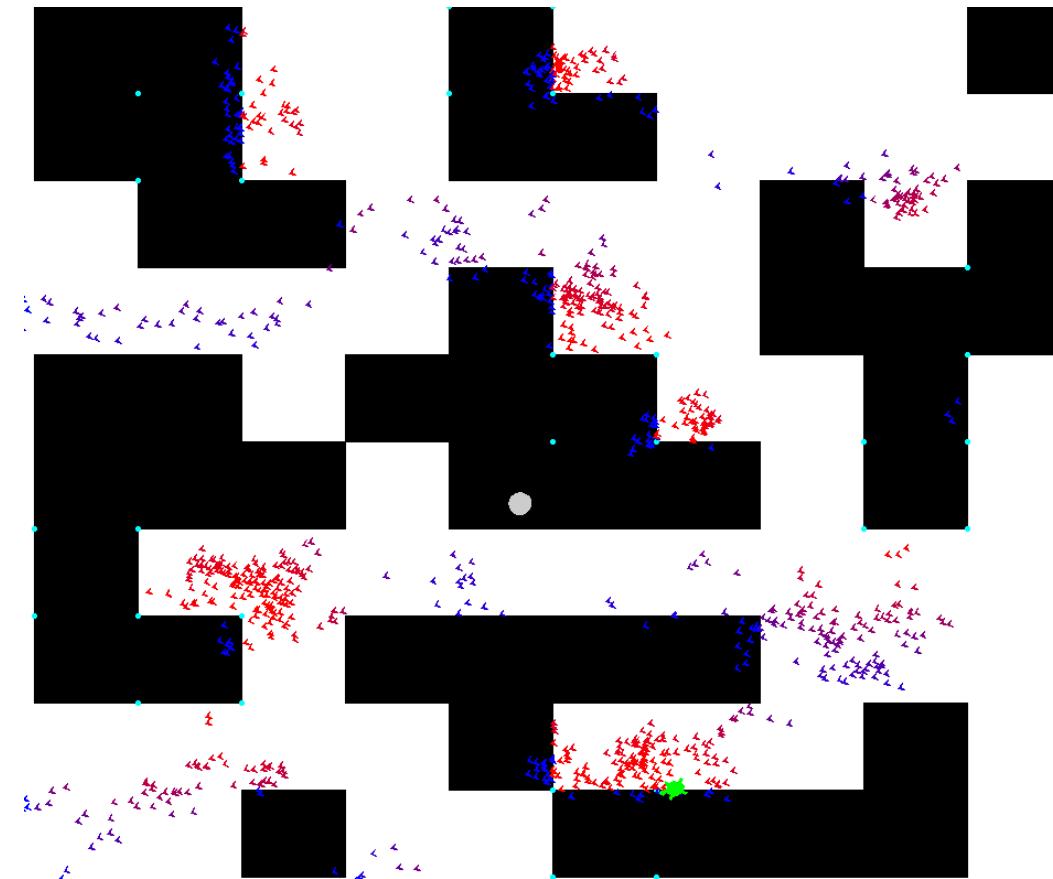
for $i = 1$ to N **do**

- $S[i] \leftarrow$ sample from $\mathbf{P}(\mathbf{X}_1 \mid \mathbf{X}_0 = S[i])$ /* step 1 */
- $W[i] \leftarrow \mathbf{P}(\mathbf{e} \mid \mathbf{X}_1 = S[i])$ /* step 2 */

$S \leftarrow$ WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S, W) /* step 3 */

return S

Robot localization



(See demo)

Summary

- Temporal models use state and sensor variables replicated over time.
 - Their purpose is to maintain a belief state as time passes and as more evidence is collected.
- The Markov and stationarity assumptions imply that we only need to specify
 - a transition model $P(\mathbf{X}_{t+1}|\mathbf{X}_t)$,
 - a sensor model $P(\mathbf{E}_t|\mathbf{X}_t)$.
- Inference tasks include filtering, prediction, smoothing and finding the most likely sequence.
- Filter algorithms are all based on the core of idea of
 - projecting the current belief state through the transition model,
 - updating the prediction according to the new evidence.

References

- Kalman, Rudolph Emil. "A new approach to linear filtering and prediction problems." *Journal of basic Engineering* 82.1 (1960): 35-45.
- Bishop, Christopher "Pattern Recognition and Machine Learning" (2006).

Introduction to Artificial Intelligence

Lecture 8: Making decisions

Prof. Gilles Louppe
g.louppé@uliege.be



Today



Reasoning under uncertainty and **taking decisions**:

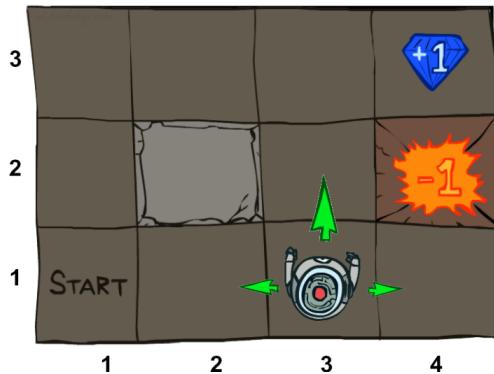
- Markov decision processes
 - MDPs
 - Bellman equation
 - Value iteration
 - Policy iteration
- Partially observable Markov decision processes

Grid world

Assume our agent lives in a 4×3 grid environment.

- Noisy movements: actions do not always go as planned.
 - Each action achieves the intended effect with probability **0.8**.
 - The rest of the time, with probability **0.2**, the action moves the agent at right angles to the intended direction.
 - If there is a wall in the direction the agent would have been taken, the agent stays put.
- The agent receives rewards at each time step.
 - Small 'living' reward each step (can be negative).
 - Big rewards come at the end (good or bad).

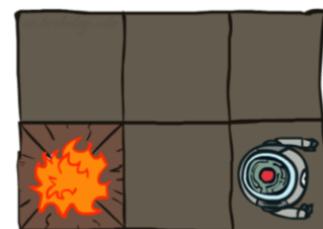
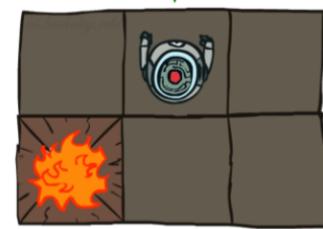
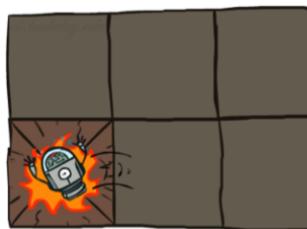
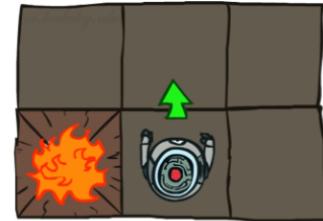
Goal: maximize sum of rewards.



Deterministic
actions



Stochastic actions

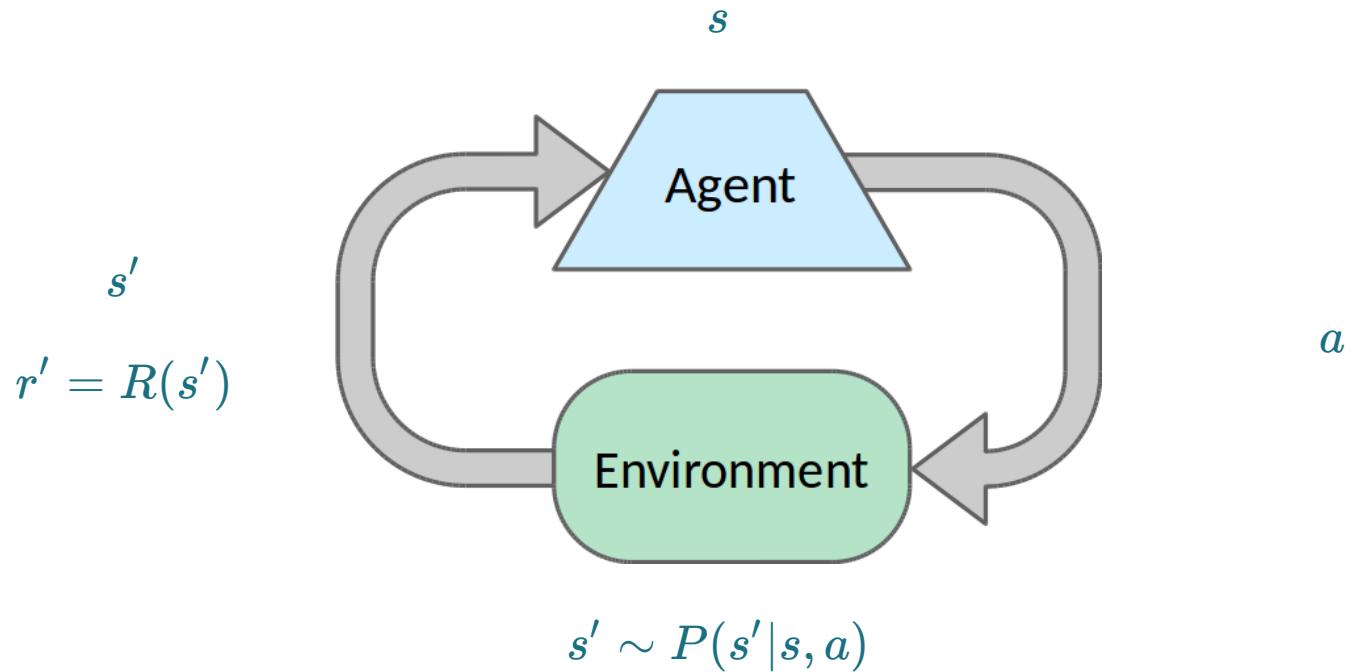


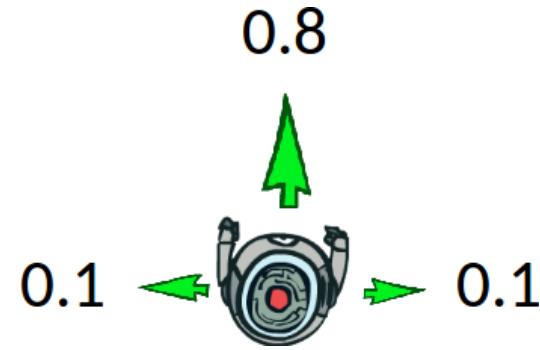
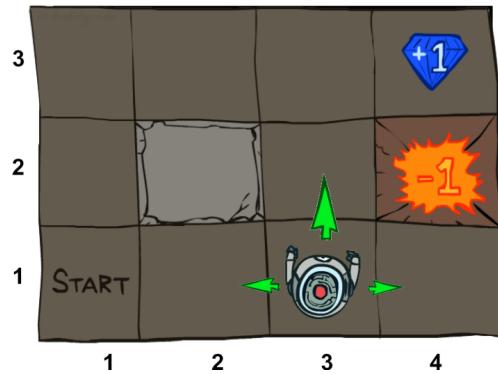
Markov decision processes

Markov decision processes

A **Markov decision process** (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, P, R)$ such that:

- \mathcal{S} is a set of states s ;
- \mathcal{A} is a set of actions a ;
- P is a (stationary) transition model such that $P(s'|s, a)$ denotes the probability of reaching state s' if action a is done in state s ;
- R is a reward function that maps immediate (finite) reward values $R(s)$ obtained in states s .





Example

- \mathcal{S} : locations (i, j) on the grid.
- \mathcal{A} : [Up, Down, Right, Left].
- Transition model: $P(s'|s, a)$
- Reward:

$$R(s) = \begin{cases} -0.3 & \text{for non-terminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

What is Markovian about MDPs?

Given the present state, the future and the past are independent:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0) = P(s_{t+1}|s_t, a_t)$$

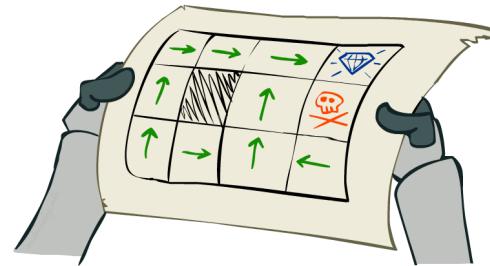
This is similar to search problems, where the successor function could only depend on the current state.



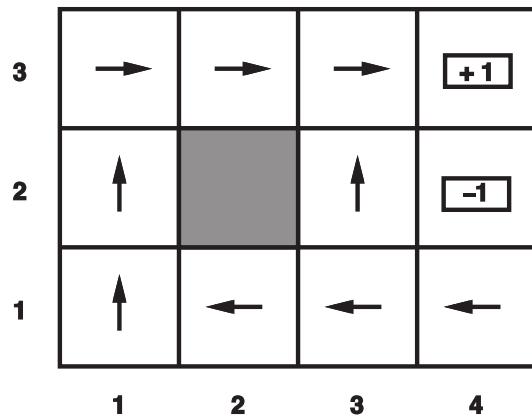
Andrey Markov

Policies

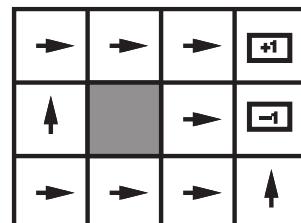
- In deterministic single-agent search problems, our goal was to find an optimal plan, or **sequence** of actions, from start to goal.
- For MDPs, we want to find an optimal **policy** $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$.
 - A policy π maps states to actions.
 - An optimal policy is one that maximizes the expected utility, e.g. the expected sum of rewards.
 - An explicit policy defines a reflex agent.
- Expectiminimax did not compute entire policies, but only some action for a single state.



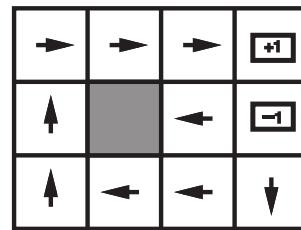
Optimal policy when
 $R(s) = -0.3$ for all non-terminal states s .



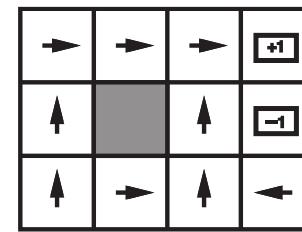
(a)



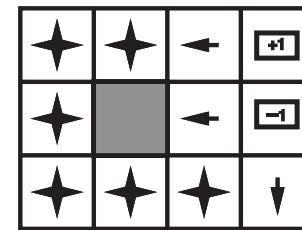
$$R(s) < -1.6284$$



$$-0.0221 < R(s) < 0$$



$$-0.4278 < R(s) < -0.0850$$

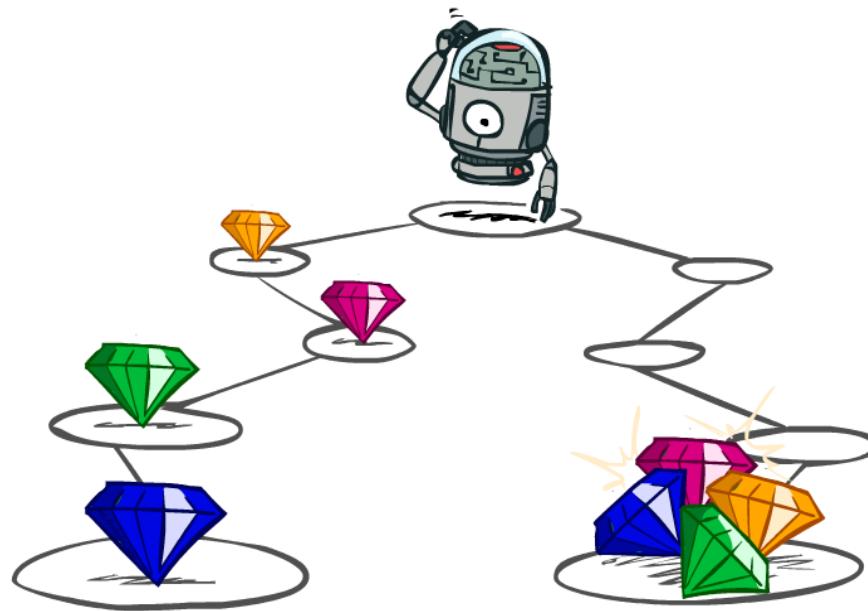


$$R(s) > 0$$

(a) Optimal policy when $R(s) = -0.04$ for all non-terminal states s . (b) Optimal policies for four different ranges of $R(s)$.

Depending on $R(s)$, the balance between risk and reward changes from risk-taking to very conservative.

Utilities over time



What preferences should an agent have over state or reward sequences?

- More or less? $[2, 3, 4]$ or $[1, 2, 2]$?
- Now or later? $[1, 0, 0]$ or $[0, 0, 1]$?

Theorem

If we assume **stationary** preferences over reward sequences, i.e. such that

$$[r_0, r_1, r_2, \dots] \succ [r_0, r'_1, r'_2, \dots] \Leftrightarrow [r_1, r_2, \dots] \succ [r'_1, r'_2, \dots],$$

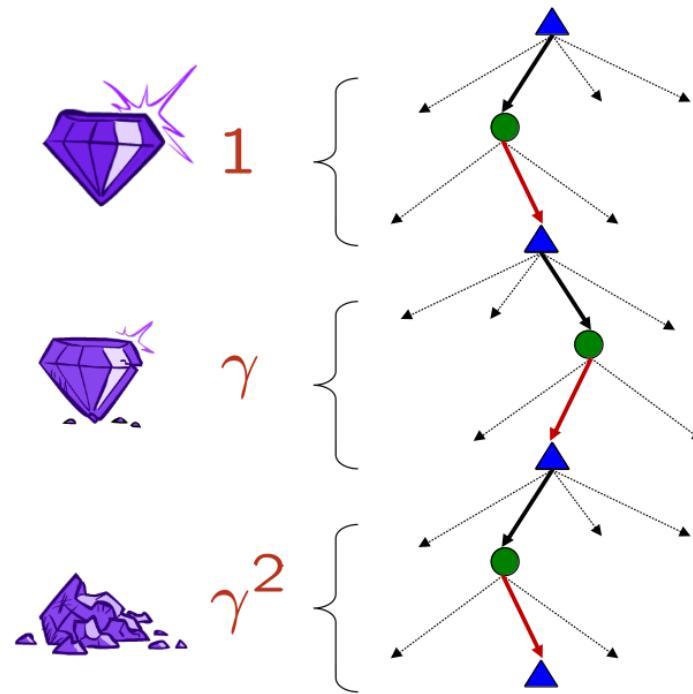
then there are only two coherent ways to assign utilities to sequences:

Additive utility: $V([r_0, r_1, r_2, \dots]) = r_0 + r_1 + r_2 + \dots$

Discounted utility:
 $(0 < \gamma < 1)$ $V([r_0, r_1, r_2, \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$

Discounting

- Each time we transition to the next state, we multiply in the discount once.
- Why discount?
 - Sooner rewards probably do have higher utility than later rewards.
 - Will help our algorithms converge.



Example: discount $\gamma = 0.5$

- $V([1, 2, 3]) = 1 + 0.5 \times 2 + 0.25 \times 3$
- $V([1, 2, 3]) < V([3, 2, 1])$

Infinite sequences

What if the agent lives forever? Do we get infinite rewards? Comparing reward sequences with $+\infty$ utility is problematic.

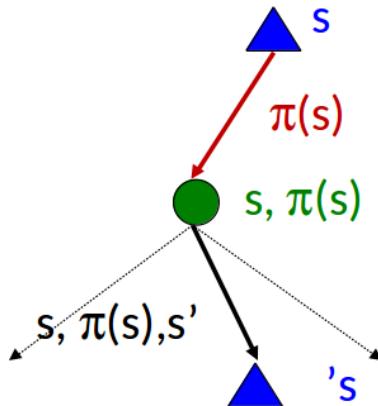
Solutions:

- Finite horizon: (similar to depth-limited search)
 - Terminate episodes after a fixed number of steps $\textcolor{teal}{T}$.
 - Results in non-stationary policies (π depends on time left).
- Discounting (with $0 < \gamma < 1$ and rewards bounded by $\pm R_{\max}$):

$$V([r_0, r_1, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{\max}}{1 - \gamma}$$

Smaller γ results in a shorter horizon.

- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached.



Policy evaluation

The expected utility obtained by executing π starting in s is given by

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right] \Big|_{s_0=s}$$

where the expectation is with respect to the probability distribution over state sequences determined by s and π .

Optimal policies

Among all policies the agent could execute, the **optimal policy** is the policy π_s^* that maximizes the expected utility:

$$\pi_s^* = \arg \max_{\pi} V^\pi(s)$$

Because of discounted utilities, the optimal policy is **independent** of the starting state s . Therefore we simply write π^* .

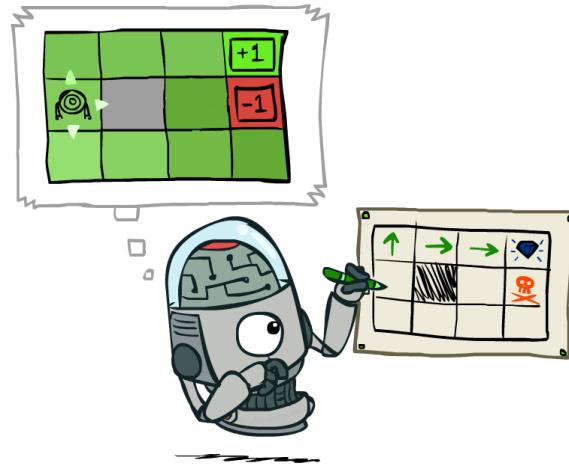
Utilities of states

The utility (or value) $V(s)$ of a state is now simply defined as $V^{\pi^*}(s)$.

- That is, the expected (discounted) reward if the agent executes an optimal policy starting from s .
- Notice that $R(s)$ and $V(s)$ are quite different quantities:
 - $R(s)$ is the short term reward for having reached s .
 - $V(s)$ is the long term total reward from s onward.

	1	2	3	4
3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388

Utilities of the states in Grid World, calculated with $\gamma = 1$ and $R(s) = -0.04$ for non-terminal states.

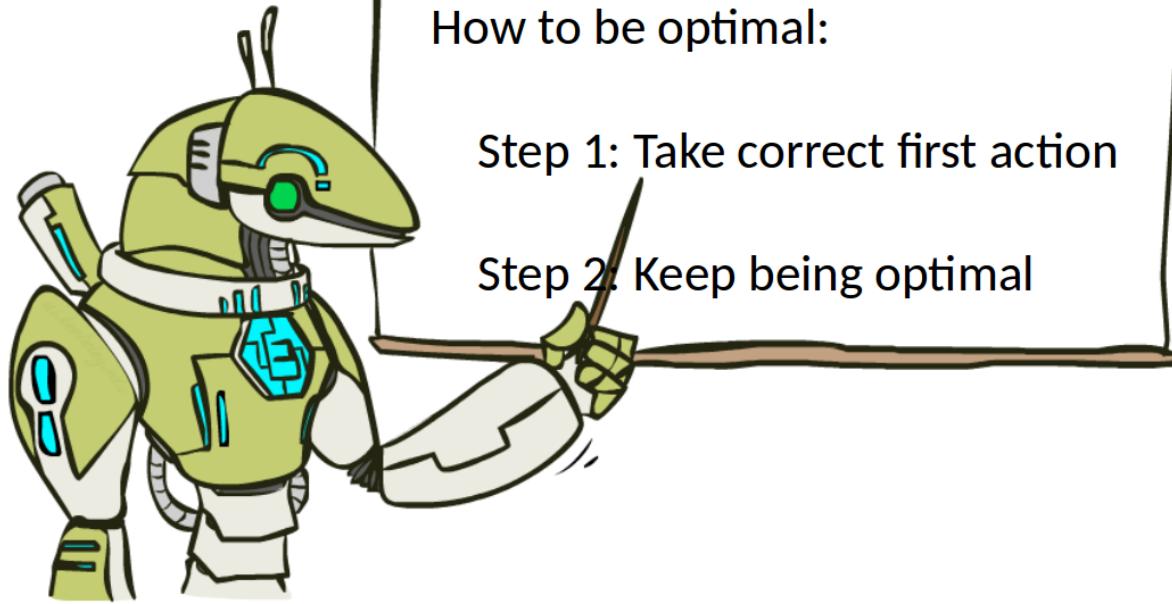


Policy extraction

Using the principle of maximum expected utility (MEU), the optimal action maximizes the expected utility of the subsequent state. That is,

$$\pi^*(s) = \arg \max_a \sum_{s'} P(s'|s, a)V(s').$$

Therefore, we can extract the optimal policy provided we can estimate the utilities of states.



The Bellman equation

There is a direct relationship between the utility of a state and the utility of its neighbors:

The utility of a state is the immediate reward for that state, plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

That is,

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a)V(s').$$

- These equations are called the **Bellman equations**. They form a system of $n = |\mathcal{S}|$ non-linear equations with as many unknowns.
- The utilities of states, defined as the expected utility of subsequent state sequences, are solutions of the set of Bellman equations.

Example

$$\begin{aligned}V(1,1) = -0.04 + \gamma \max & [0.8V(1,2) + 0.1V(2,1) + 0.1V(1,1), \\& 0.9V(1,1) + 0.1V(1,2), \\& 0.9V(1,1) + 0.1V(2,1), \\& 0.8V(2,1) + 0.1V(1,2) + 0.1V(1,1)]\end{aligned}$$

Value iteration

Because of the **max** operator, the Bellman equations are non-linear and solving the system is problematic.

The **value iteration** algorithm provides a fixed-point iteration procedure for computing the state utilities $V(s)$:

- Let $V_i(s)$ be the estimated utility value for s at the i -th iteration step.
- The **Bellman update** consists in updating simultaneously all the estimates to make them **locally consistent** with the Bellman equation:

$$V_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) V_i(s')$$

- Repeat until convergence.

```

function VALUE-ITERATION( $mdp, \epsilon$ ) returns a utility function
  inputs:  $mdp$ , an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ ,
            rewards  $R(s)$ , discount  $\gamma$ 
             $\epsilon$ , the maximum error allowed in the utility of any state
  local variables:  $U, U'$ , vectors of utilities for states in  $S$ , initially zero
             $\delta$ , the maximum change in the utility of any state in an iteration

repeat
   $U \leftarrow U'; \delta \leftarrow 0$ 
  for each state  $s$  in  $S$  do
     $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']$ 
    if  $|U'[s] - U[s]| > \delta$  then  $\delta \leftarrow |U'[s] - U[s]|$ 
  until  $\delta < \epsilon(1 - \gamma)/\gamma$ 
  return  $U$ 

```

Convergence

Let V_i and V_{i+1} be successive approximations to the true utility V .

Theorem. For any two approximations V_i and V'_i ,

$$\|V_{i+1} - V'_{i+1}\|_\infty \leq \gamma \|V_i - V'_i\|_\infty.$$

- That is, the Bellman update is a contraction by a factor γ on the space of utility vector.
- Therefore, any two approximations must get closer to each other, and in particular any approximation must get closer to the true V .

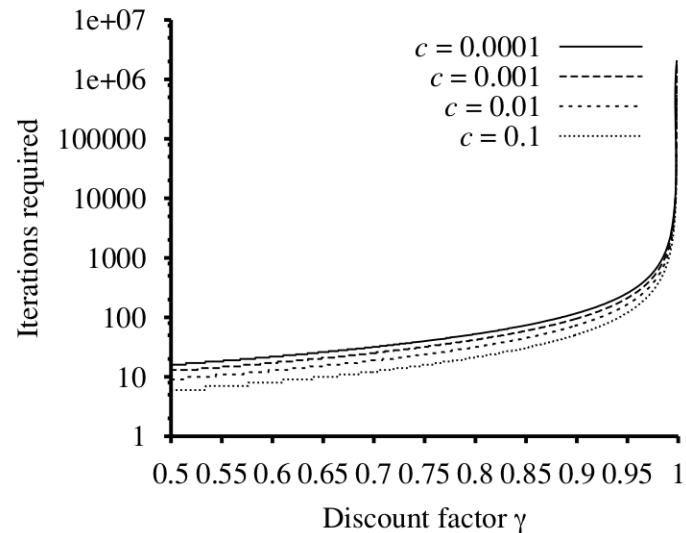
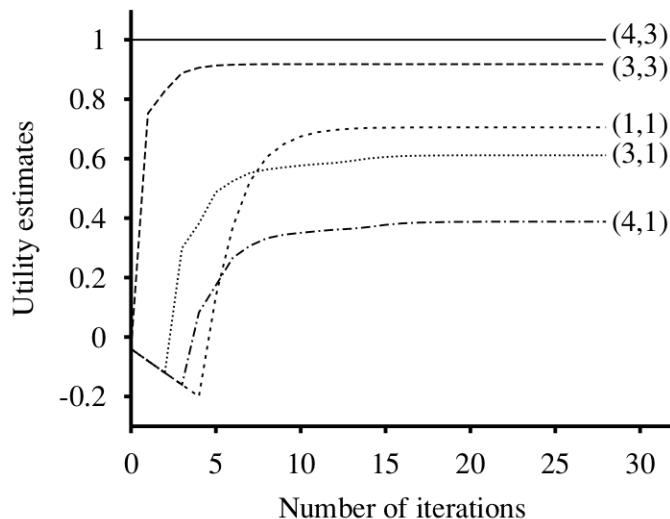
⇒ Value iteration always converges to a unique solution of the Bellman equations whenever $\gamma < 1$.

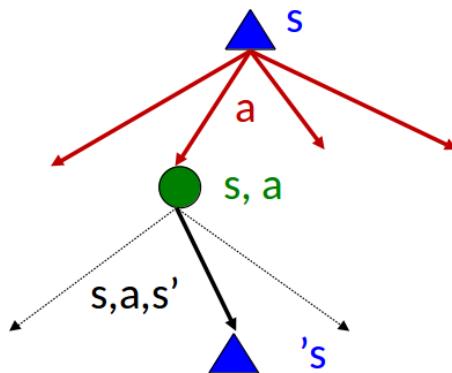
Performance

Since $\|V_{i+1} - V\|_\infty \leq \gamma \|V_i - V\|_\infty$, the error is reduced by a factor of at least γ at each iteration.

Therefore, value iteration converges exponentially fast:

- The maximum initial error is $\|V_0 - V\|_\infty \leq 2R_{\max}/(1 - \gamma)$.
- To reach an error of at most ϵ after N iterations, we require $\gamma^N 2R_{\max}/(1 - \gamma) \leq \epsilon$.





Problems with value iteration

Value iteration repeats the Bellman updates:

$$V_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) V_i(s')$$

- Problem 1: it is slow - $\mathcal{O}(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration.
- Problem 2: the **max** at each state rarely changes.
- Problem 3: the policy π_i extracted from the estimate V_i might be optimal even if V_i is inaccurate!

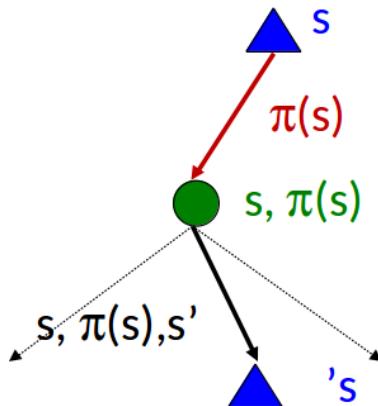
Policy iteration

The **policy iteration** algorithm instead directly computes the policy (instead of state values). It alternates the following two steps:

- Policy evaluation: given π_i , calculate $V_i = V^{\pi_i}$, i.e. the utility of each state if π_i is executed.
- Policy improvement: calculate a new policy π_{i+1} using one-step look-ahead based on V_i :

$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} P(s'|s, a) V_i(s')$$

This algorithm is still optimal, and might converge (much) faster under some conditions.



Policy evaluation

At the i -th iteration we have a simplified version of the Bellman equations that relate the utility of s to the utilities of its neighbors:

$$V_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) V_i(s')$$

These equations are now **linear** because the **max** operator has been removed.

- for n states, we have n equations with n unknowns;
- this can be solved exactly in $O(n^3)$ by standard linear algebra methods.

In some cases $\mathcal{O}(n^3)$ is too prohibitive. Fortunately, it is not necessary to perform exact policy evaluation. An approximate solution is sufficient.

One way is to run k iterations of simplified Bell updates:

$$V_{i+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))V_i(s)$$

This hybrid algorithm is called **modified policy iteration**.

function POLICY-ITERATION(mdp) **returns** a policy

inputs: mdp , an MDP with states S , actions $A(s)$, transition model $P(s' | s, a)$

local variables: U , a vector of utilities for states in S , initially zero
 π , a policy vector indexed by state, initially random

repeat

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp)$

$unchanged? \leftarrow \text{true}$

for each state s **in** S **do**

if $\max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s'] > \sum_{s'} P(s' | s, \pi[s]) U[s']$ **then do**

$\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']$

$unchanged? \leftarrow \text{false}$

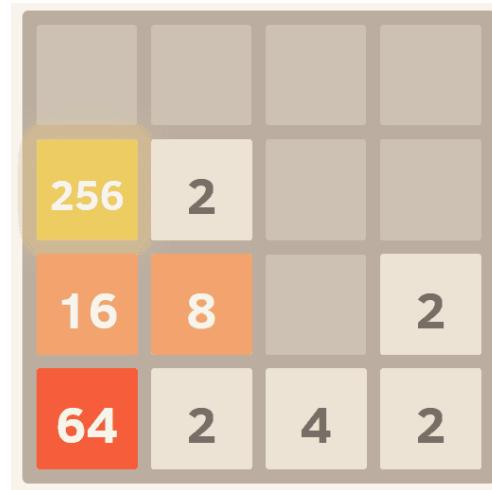
until $unchanged?$

return π

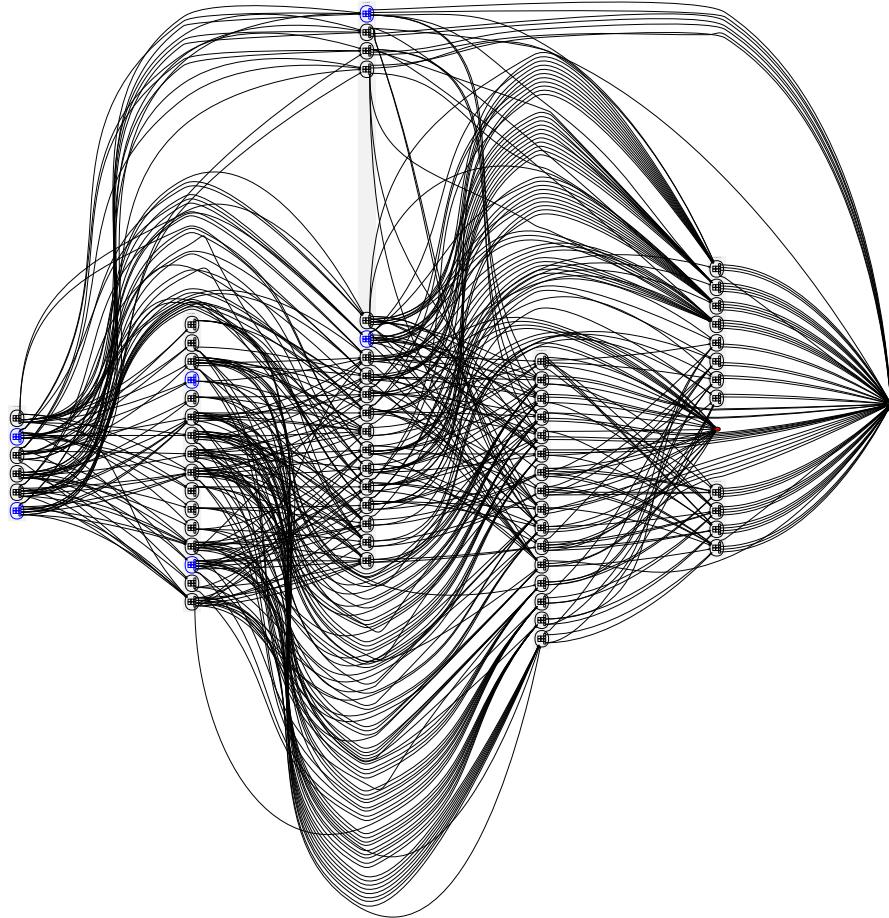
Recap example: 2048

The game 2048 is a Markov decision process!

- \mathcal{S} : all possible configurations of the board (huge!)
- \mathcal{A} : swiping left, right, up or down.
- $P(s'|s, a)$: encodes the game's dynamic
 - collapse matching tiles
 - place a random tile on the board
- $R(s) = 1$ if s is a winning state, and 0 otherwise.



[Q] For $\gamma < 1$, is the optimal solution also the shortest?



The transition model for a 2×2 board and a winning state at 8.

Optimal play for a 3×3 grid and a winning state at 1024.

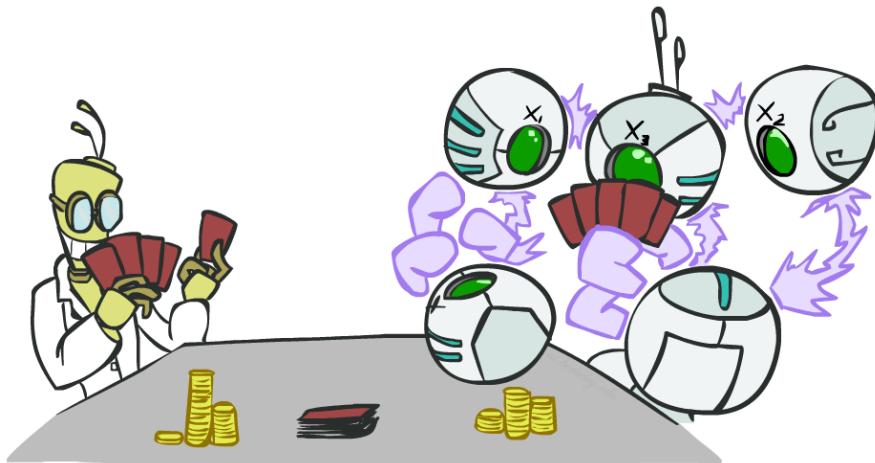
See jdlm.info: The Mathematics of 2048.

Partially observable Markov decision processes

POMPDs

What if the environment is only **partially observable**?

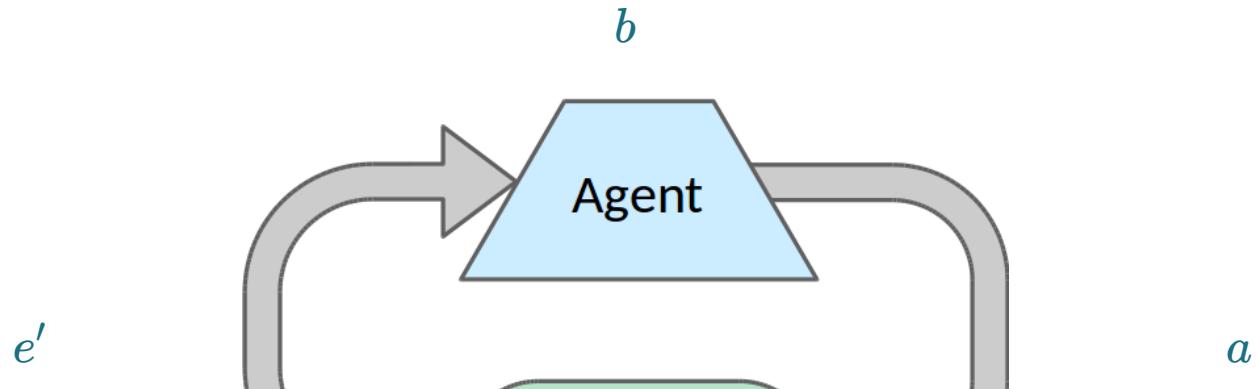
- The agent does not know in which state s it is in.
 - Therefore, it cannot evaluate the reward $R(s)$ associated to the unknown state.
 - Also, it makes no sense to talk about a policy $\pi(s)$.
- Instead, the agent collects percepts e through a sensor model $P(e|s)$, from which it can reason about the unknown state s .



We will assume that the agent maintains a belief state b .

- b represents a probability distribution $P(S)$ of the current agent's beliefs over its state;
- $b(s)$ denotes the probability $P(S = s)$ under the current belief state;
- the belief state b is updated as evidence e are collected.

This is filtering!



b

Agent

a

Environment

e'

$$s' \sim P(s'|s, a)$$

$$e' \sim P(e'|s')$$

Belief MDP

Theorem (Astrom, 1965). The optimal action depends only on the agent's current belief state.

- The optimal policy can be described by a mapping $\pi^*(b)$ from beliefs to actions.
- It does not depend on the actual state the agent is in.

In other words, POMDPs can be reduced to an MDP in belief-state space, provided we can define a transition model $P(b'|b, a)$ and a reward function ρ over belief states.

If b was the previous belief state and the agent does action a and perceives e , then the new belief state over S' is given by

$$b' = \alpha P(e|S') \sum_s P(S'|s, a) b(s) = \alpha \text{forward}(b, a, e).$$

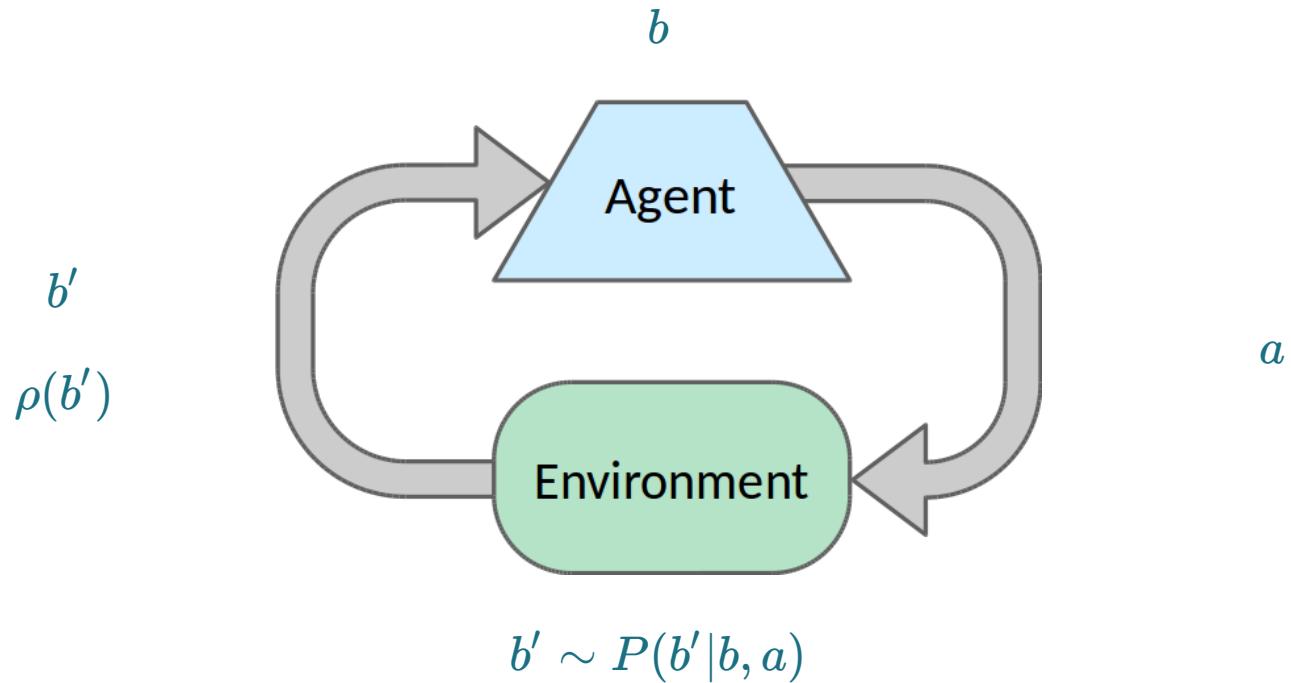
Therefore,

$$\begin{aligned} P(b'|b, a) &= \sum_e P(b', e|b, a) \\ &= \sum_e P(b'|b, a, e) P(e|b, a) \\ &= \sum_e P(b'|b, a, e) \sum_{s'} P(e|b, a, s') P(s'|b, a) \\ &= \sum_e P(b'|b, a, e) \sum_{s'} P(e|s') \sum_s P(s'|s, a) b(s) \end{aligned}$$

where $P(b'|b, a, e) = 1$ if $b' = \text{forward}(b, a, e)$ and 0 otherwise.

We can also define a reward function for belief states as the expected reward for the actual state the agent might be in:

$$\rho(b) = \sum_s b(s)R(s)$$



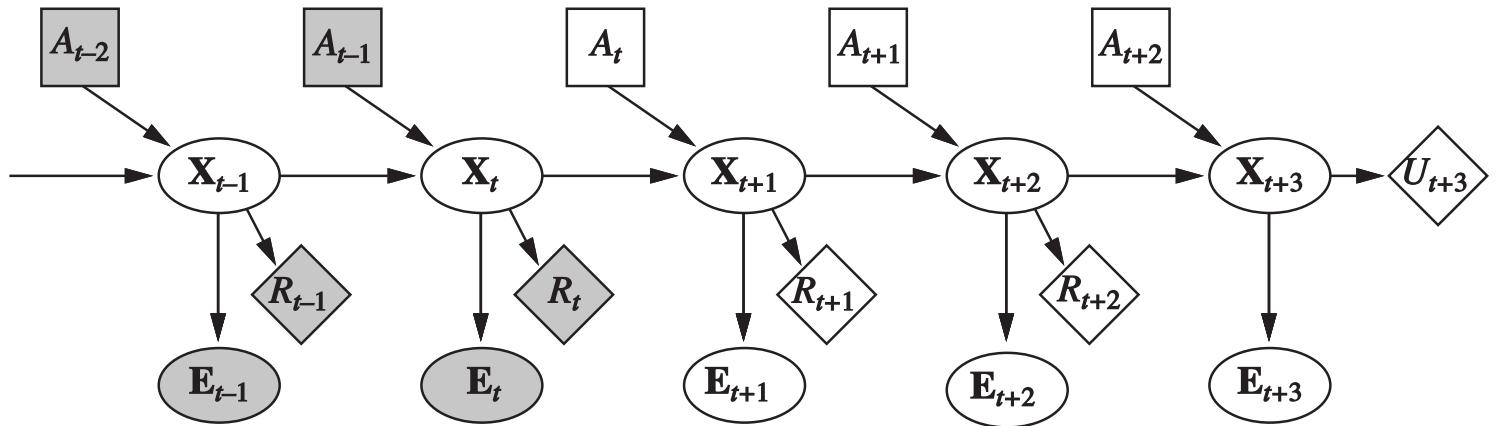
Although we have reduced POMDPs to MPDs, the Belief MDP we obtain has a **continuous** (and usually high-dimensional) state space.

- None of the algorithms described earlier directly apply.
- In fact, solving POMDPs remains a difficult problem for which there is no known efficient exact algorithm.
- Yet, Nature is a POMDP.

Online agents

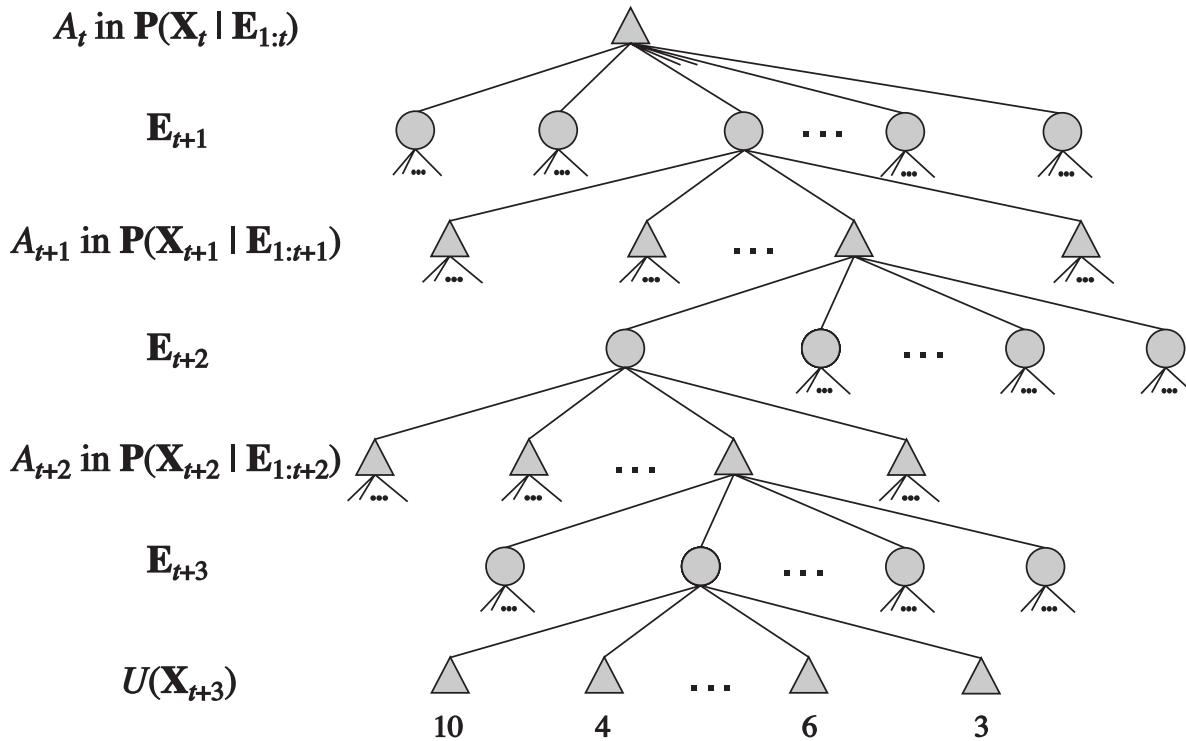
While it is difficult to directly derive π^* , a decision-theoretic agent can be constructed for POMDPs:

- The transition and sensor models are represented by a [dynamic Bayesian network](#);
- The dynamic Bayesian network is extended with decision (A) and utility (R and U) nodes to form a dynamic decision network;
- A [filtering algorithm](#) is used to incorporate each new percept and action and to update the belief state representation;
- Decisions are made by projecting forward possible action sequences and choosing (approximately) the best one, in a manner similar to a truncated [Expectiminimax](#).



At time t , the agent must decide what to do.

- Shaded nodes represent variables with known values.
- The network is unrolled for a finite horizon.
- It includes nodes for the reward of \mathbf{X}_{t+1} and \mathbf{X}_{t+2} , but the (estimated) utility of \mathbf{X}_{t+3} .



Part of the look-ahead solution of the previous decision network.

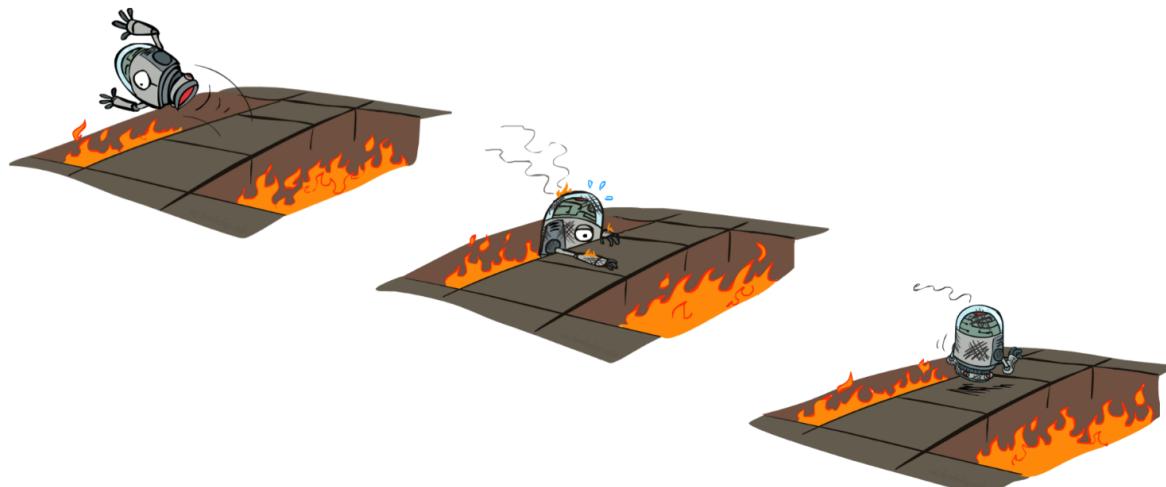
A decision can be extracted from the search tree by backing up the (estimated) utility values from the leaves, taking the average at the chance nodes and taking the maximum at the decision nodes.

Reinforcement learning

A Markov decision process assumes the knowledge of a transition model $P(s'|s, a)$ and of a reward function R . What if the MDP is unknown?

- We do not know which states are good nor what actions do!
- We must observe or interact with the environment in order to jointly **learn** these dynamics and act upon them.

⇒ This is **reinforcement learning**.



Reinforcement learning include two main families of algorithms:

Model-based

- Learn an approximate model for $P(s'|s, a)$ and R based on experiences.
- Solve the resulting approximate MDP with value or policy iteration.

Model-free

- Learn an approximate model for V based on experiences.
- Extract a policy from the approximate state values V .

Learning approximate models from experiences can be done with [supervised learning](#) (see Lecture 9).

e



Playing Pinball

e



Playing Seaquest

e



Playing Enduro

Summary

- Sequential decision problems in uncertain environments, called MDPs, are defined by transition model and a reward function.
- The utility of a state sequence is the sum of all the rewards over the sequence, possibly discounted over time.
 - The solution of an MDP is a policy that associates a decision with every state that the agent might reach.
 - An optimal policy maximizes the utility of the state sequence encountered when it is executed.
- Value iteration and policy can both be used for solving MDPs.
- POMDPs are much more difficult than MDPs. However, a decision-theoretic agent can be constructed for those environments.
 - Research is going on!

References

- Åström, Karl J. "Optimal control of Markov processes with incomplete state information." *Journal of Mathematical Analysis and Applications* 10.1 (1965): 174-205.

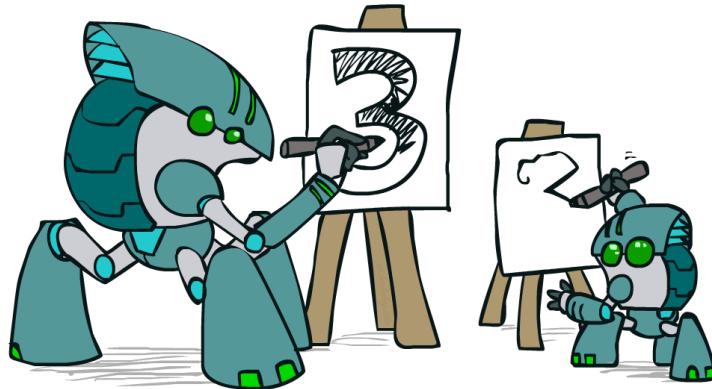
Introduction to Artificial Intelligence

Lecture 9: Learning

Prof. Gilles Louppe
g.louppe@uliege.be



Today



Make our agents capable of self-improvement through a **learning** mechanism.

- Statistical learning
- Supervised learning
 - Linear models
 - Perceptron
 - Neural networks
- Unsupervised learning

Intelligence?

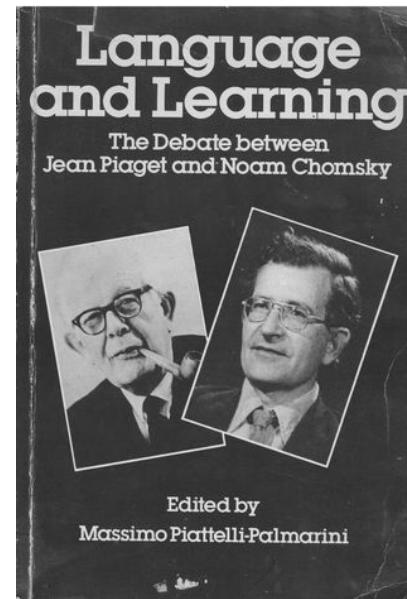
What we covered so far:

- Search algorithms, using a state space specified by domain knowledge.
- Constraint satisfaction problems, by exploiting a known structure of the states.
- Logical inference, using well-specified facts and inference rules.
- Adversarial search, for known and fully observable games.
- Reasoning about uncertain knowledge, as represented using domain-motivated probabilistic models.
- Taking optimal decisions, under uncertainty and possibly under partial observation.

Sufficient to implement complex and rational behaviors, in some situations. But is that **intelligence**? Aren't we missing a critical component?

Chomsky vs. Piaget

- Noam Chomsky's **innatism**:
 - State that humans possess a genetically determined faculty for thought and language.
 - The structures of language and thought are set in motion through interaction with the environment.
- Jean Piaget's **constructivism**:
 - Deny the existence of innate cognitive structure specific for thought and language.
 - Postulate instead all cognitive acquisitions, including language, to be the outcome of a gradual process of construction, i.e., a learning procedure.



[Q] What about AI? Should it be a pre-wired efficient machine? Or a machine that can learn and improve? or maybe a bit of both?

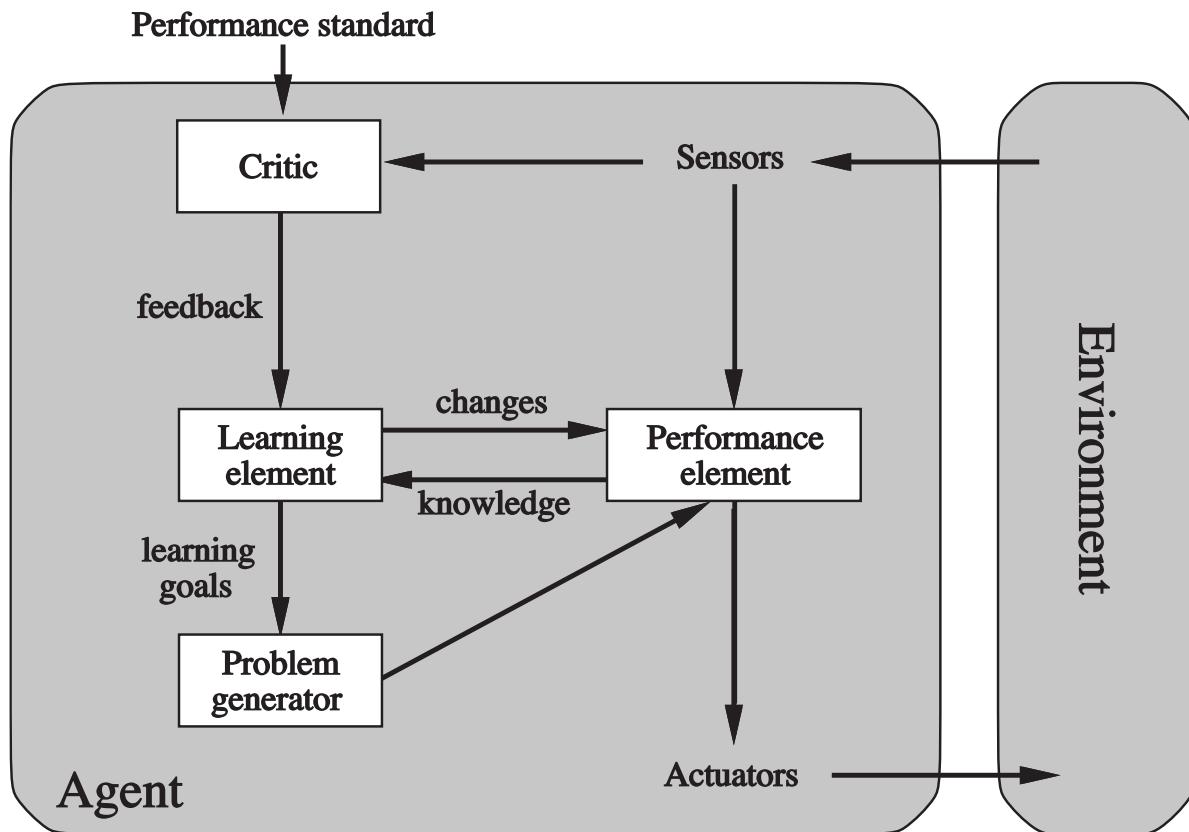


The debate continues...

Learning agents

What if the environment is **unknown**?

- Learning can be used as a system construction method.
- Expose the agent to reality rather trying to hardcode reality into the agent's program.
- Learning provides an automated way to modify the agent's internal decision mechanisms to improve its own performance.



The design of the **learning element** is dictated by:

- What type of performance element is used.
- Which functional component is to be learned.
- How that functional component is represented.
- What kind of feedback is available.

Performance element	Component	Representation	Feedback
Alpha–beta search	Eval. fn.	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 fn	Neural net	Correct action

Statistical learning

Bayesian learning

View **learning** as a Bayesian update of a probability distribution $P(H)$ over a **hypothesis space**, where

- H is the hypothesis variable
- values are h_1, h_2, \dots
- the prior is $P(H)$
- \mathbf{d} is the observed data

Given data, each hypothesis has a posterior probability

$$P(h_i|\mathbf{d}) = \frac{P(\mathbf{d}|h_i)P(h_i)}{P(\mathbf{d})},$$

where $P(\mathbf{d}|h_i)$ is called the **likelihood**.

Predictions use a likelihood-weighted average over the hypotheses:

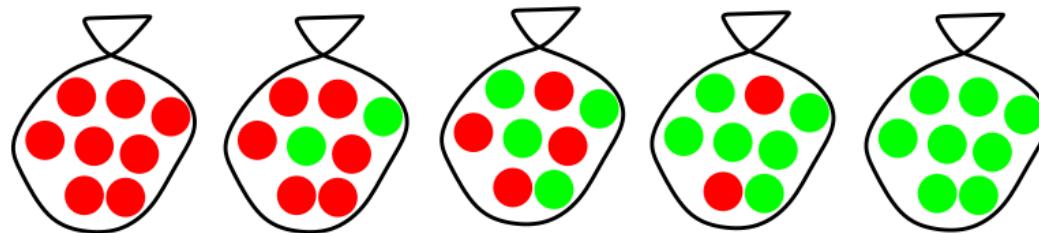
$$P(X|\mathbf{d}) = \sum_i P(X|\mathbf{d}, h_i)P(h_i|\mathbf{d}) = \sum_i P(X|h_i)P(h_i|\mathbf{d})$$

No need to pick one best-guess hypothesis!

Example

Suppose there are five kinds of bags of candies. Assume a prior $P(H)$:

- $P(h_1) = 0.1$, with h_1 : 100% cherry candies
- $P(h_2) = 0.2$, with h_2 : 75% cherry candies + 25% lime candies
- $P(h_3) = 0.4$, with h_3 : 50% cherry candies + 50% lime candies
- $P(h_4) = 0.2$, with h_4 : 25% cherry candies + 75% lime candies
- $P(h_5) = 0.1$, with h_5 : 100% lime candies

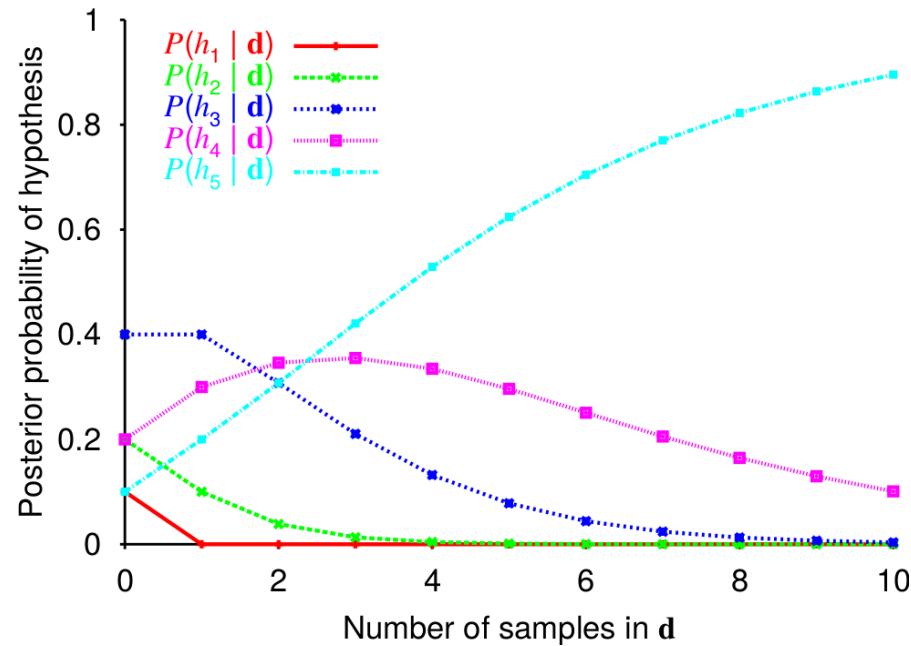


Then we observe candies drawn from some bag:

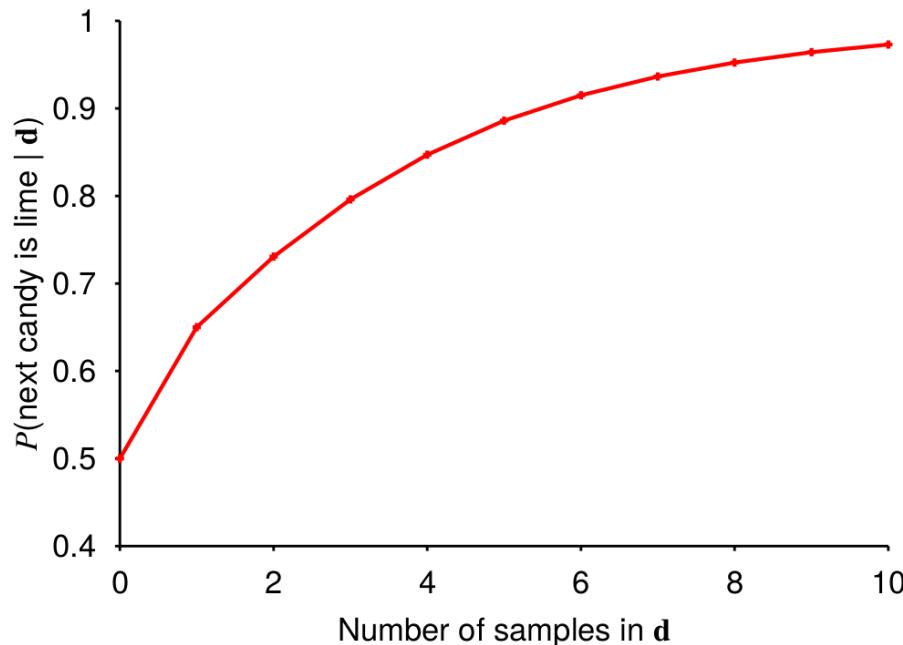


- What kind of bag is it?
- What flavor will the next candy be?

Posterior probability of hypotheses



Prediction probability



- This example illustrates the fact that the Bayesian prediction eventually agrees with the true hypothesis.
- The posterior probability of any false hypothesis eventually vanishes (under weak assumptions).

Maximum a posteriori

Summing over the hypothesis space is often **intractable**.

Instead, **maximum a posteriori estimation** (MAP) consists in using the hypothesis

$$h_{MAP} = \arg \max_{h_i} P(h_i | \mathbf{d}).$$

That is, maximize $P(\mathbf{d}|h_i)P(h_i)$ or $\log P(\mathbf{d}|h_i) + \log P(h_i)$.

- Log terms can be viewed as (negative number of) bits to encode data given hypothesis + bits to encode hypothesis.
 - This is the basic idea of minimum description length learning, i.e., Occam's razor.
- Finding the MAP hypothesis is often much easier than Bayesian learning.
 - It requires solving an optimization problem instead of a large summation problem.

Maximum likelihood

For large data sets, the prior $P(H)$ becomes irrelevant.

In this case, maximum likelihood estimation (MLE) consists in using the hypothesis

$$h_{MLE} = \arg \max_{h_i} P(\mathbf{d}|h_i)$$

That is, simply get the best fit to the data.

- Identical to MAP for uniform prior.
- Maximum likelihood estimation is the standard (non-Bayesian) statistical learning method.

Recipe

- Choose a **parameterized** family of models to describe the data (e.g., a Bayesian network).
 - requires substantial insight and sometimes new models.
- Write down the log-likelihood L of the data as a function of the parameters.
 - may require summing over hidden variables, i.e., inference.
- Write down the derivative of the log likelihood w.r.t. each parameter θ .
- Find the parameter values θ such that the derivatives are zero.
 - may be hard/impossible; modern optimization techniques help.

Parameter learning in Bayesian networks

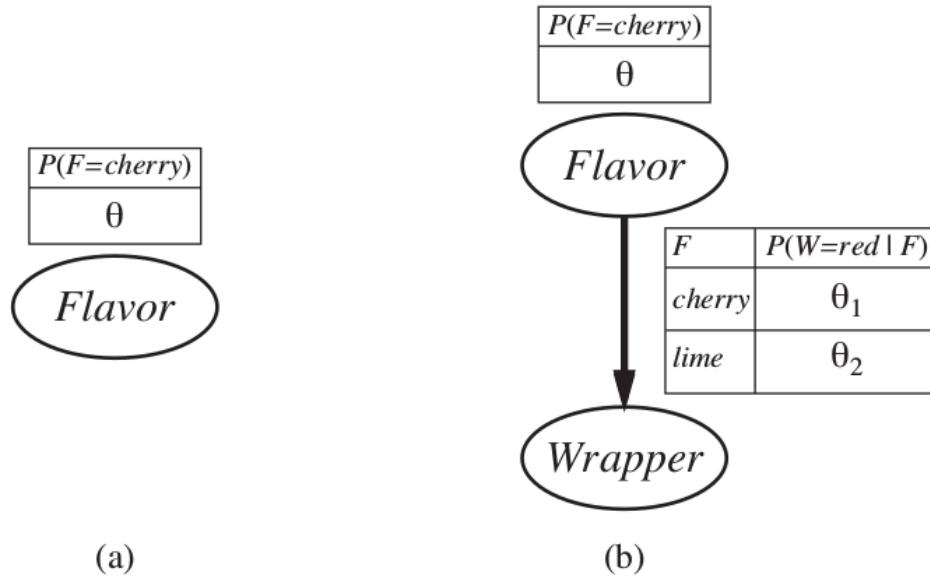


Figure 20.2 (a) Bayesian network model for the case of candies with an unknown proportion of cherries and limes. (b) Model for the case where the wrapper color depends (probabilistically) on the candy flavor.

MLE, case (a)

Bag from a new manufacturer; fraction θ of cherry candies?

- Any $\theta \in [0, 1]$ is possible: continuum of hypotheses h_θ .
- θ is a **parameter** for this simple binomial family of models.

Suppose we unwrap N candies, and get c cherries and $l = N - c$ limes. These are i.i.d. observations, so:

$$P(\mathbf{d}|h_\theta) = \prod_{j=1}^N P(d_j|h_\theta) = \theta^c(1-\theta)^l$$

Maximize this w.r.t. θ , which is easier for the log-likelihood:

$$\begin{aligned} L(\mathbf{d}|h_\theta) &= \log P(\mathbf{d}|h_\theta) = c \log \theta + l \log(1-\theta) \\ \frac{dL(\mathbf{d}|h_\theta)}{d\theta} &= \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \end{aligned}$$

Therefore $\theta = \frac{c}{N}$.

MLE, case (b)

Red/green wrapper depends probabilistically on flavor. E.g., the likelihood for a cherry candy in green wrapper:

$$\begin{aligned} P(\text{cherry, green} | h_{\theta, \theta_1, \theta_2}) \\ = P(\text{cherry} | h_{\theta, \theta_1, \theta_2}) P(\text{green} | \text{cherry}, h_{\theta, \theta_1, \theta_2}) \\ = \theta(1 - \theta_1) \end{aligned}$$

The likelihood for the data, given N candies, r_c red-wrapped cherries, g_c green-wrapped cherries, etc. is:

$$\begin{aligned} P(\mathbf{d} | h_{\theta, \theta_1, \theta_2}) &= \theta^c (1 - \theta)^l \theta_1^{r_c} (1 - \theta_1)^{g_c} \theta_2^{r_l} (1 - \theta_2)^{g_l} \\ L &= c \log \theta + l \log(1 - \theta) + \\ &\quad r_c \log \theta_1 + g_c \log(1 - \theta_1) + \\ &\quad r_l \log \theta_2 + g_l \log(1 - \theta_2) \end{aligned}$$

Derivatives of L contain only the relevant parameter:

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \Rightarrow \theta = \frac{c}{c+l}$$

$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \Rightarrow \theta_1 = \frac{r_c}{r_c + g_c}$$

$$\frac{\partial L}{\partial \theta_2} = \frac{r_l}{\theta_2} - \frac{g_l}{1-\theta_2} = 0 \Rightarrow \theta_2 = \frac{r_l}{r_l + g_l}$$

- This can be extended to any Bayesian network with parameterized CPTs.
- With complete data, maximum likelihood parameter estimation for a Bayesian network decomposes into separate optimization problems, one for each parameter.

Supervised learning

(mostly neural networks)

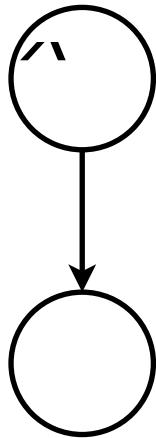
Problem statement

Assume data $\mathbf{d} \sim P(X, Y)$ of N example input-output pairs

$$\mathbf{d} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\},$$

where \mathbf{x}_i are the input data and y_i was generated by an unknown function $y_i = f(\mathbf{x}_i)$.

- From this data, we wish to **learn a function** $h \in \mathcal{H}$ that approximates the true function f .
- \mathcal{H} is huge! How do we **find** a good hypothesis?

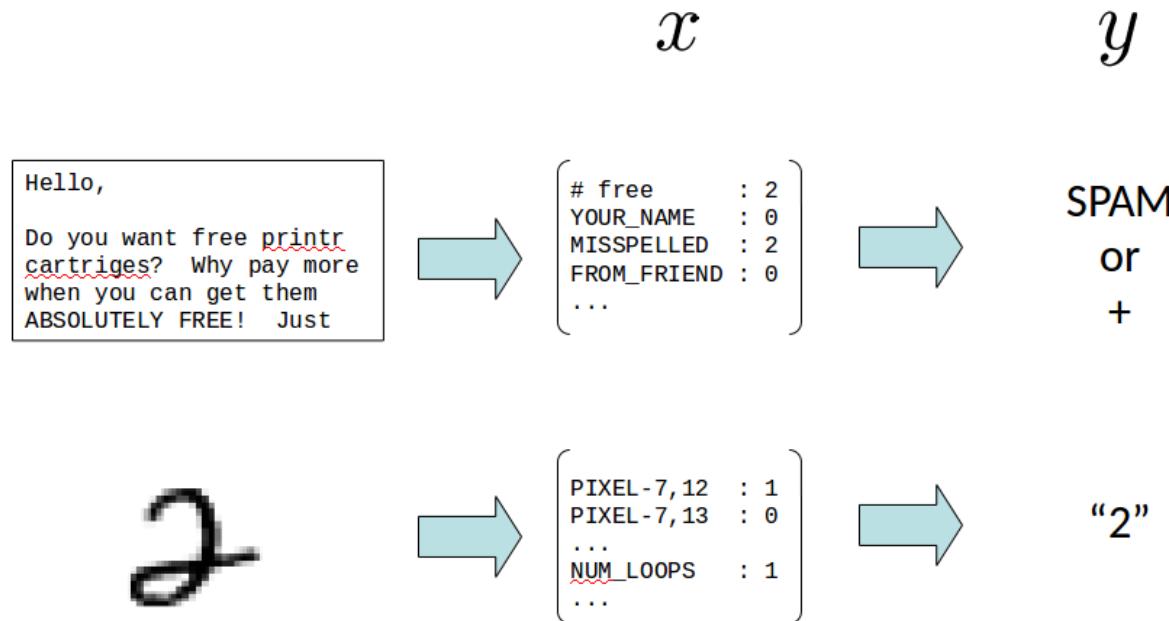


In general, f will be **stochastic**. In this case, y is not strictly a function x , and we wish to learn the conditional $P(Y|X)$.

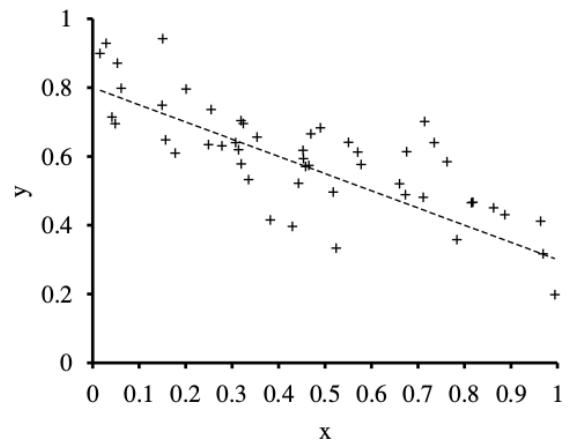
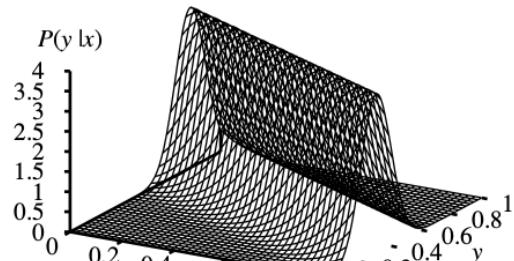
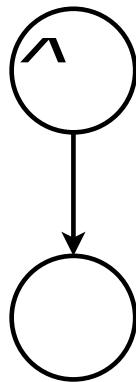
Most of supervised learning is actually (approximate) maximum likelihood estimation on (huge) parametric models.

Feature vectors

- Assume the input samples $\mathbf{x}_i \in \mathbb{R}^p$ are described as real-valued vectors of p attributes or features values.
- If the data is not originally expressed as real-valued vectors, then it needs to be prepared and transformed to this format.



Linear regression



Let us assume a **parameterized** linear Gaussian model

$$y = \mathbf{w}^T \mathbf{x} + b + \epsilon$$

with one continuous parent \mathbf{X} , one continuous child \mathbf{Y} and $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

To learn the conditional distribution $p(y|x)$, we maximize

$$p(y|x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - (\mathbf{w}^T \mathbf{x} + b))^2}{2\sigma^2}\right)$$

w.r.t. \mathbf{w} and b over the data $\mathbf{d} = \{(x_j, y_j)\}$.

By constraining the derivatives of the log-likelihood to 0, we arrive to the problem of minimizing

$$\sum_{j=1}^N (y_j - (\mathbf{w}^T \mathbf{x}_j + b))^2.$$

Therefore, minimizing the sum of squared errors corresponds to the MLE solution for a linear fit, assuming Gaussian noise of fixed variance.

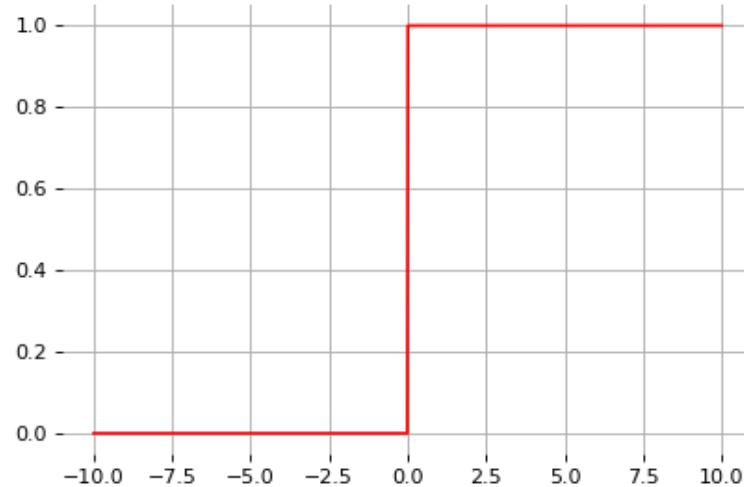
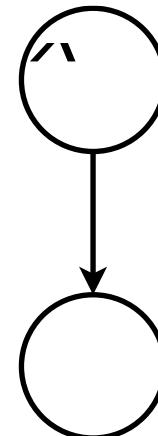
Linear classification

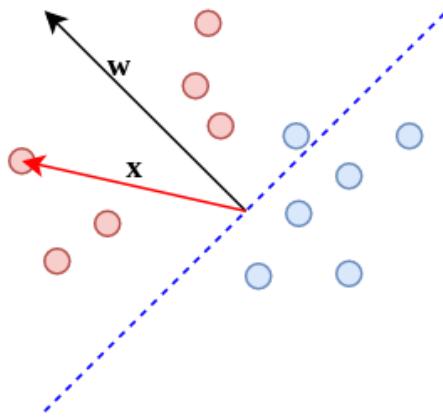
Let us now assume Y takes discrete values in $\{0, 1\}$.

Decision rules

The linear classifier model is a squashed linear function of its inputs:

$$h(\mathbf{x}; \mathbf{w}, b) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

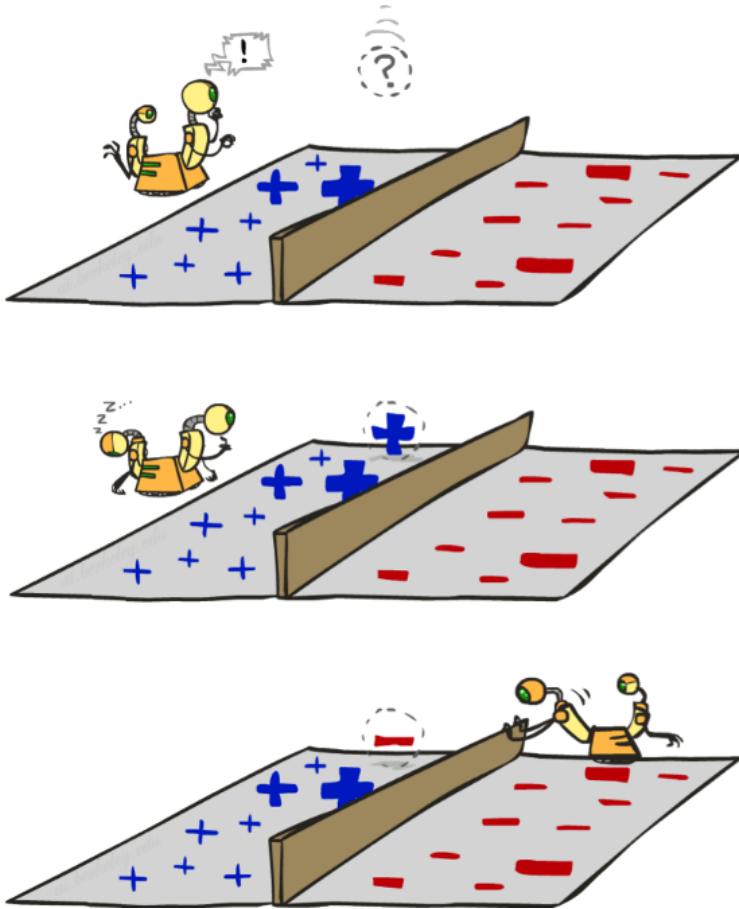
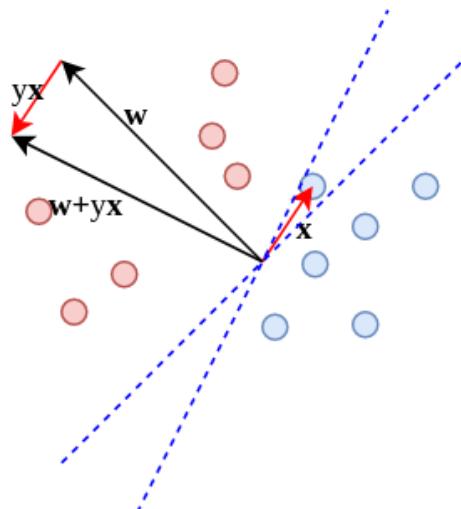




- Without loss of generality, the model can be rewritten without b as $h(\mathbf{x}; \mathbf{w}) = \text{sign}(\mathbf{w}^T \mathbf{x})$, where $\mathbf{w} \in \mathbb{R}^{p+1}$ and \mathbf{x} is extended with a dummy element $x_0 = 1$.
- Predictions are computed by comparing the feature vector \mathbf{x} to the weight vector \mathbf{w} . Geometrically, $\mathbf{w}^T \mathbf{x}$ corresponds to the $\|\mathbf{w}\| \|\mathbf{x}\| \cos(\theta)$.
- The family \mathcal{H} of hypothesis is induced from the set of possible parameters values \mathbf{w} , that is \mathbb{R}^{p+1} . Learning consists in finding a good vector \mathbf{w} in this space.

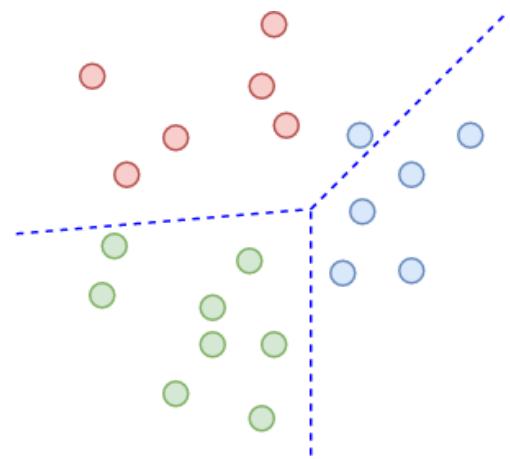
Perceptron

- Start with $\mathbf{w} = 0$.
- For each training example (\mathbf{x}, y) :
 - Classify with current weights:
 $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$
 - If $y = \hat{y}$, do nothing.
 - Otherwise, update parameters:
 $\mathbf{w} = \mathbf{w} + y\mathbf{x} - (1 - y)\mathbf{x}$



Multiclass perceptron

- If we have more than 2 classes, then
 - Define a weight vector \mathbf{w}_c for each class c .
 - The activation for class c is $\mathbf{w}_c^T \mathbf{x}$.
- Learning:
 - Start with $\mathbf{w}_c = \mathbf{0}$ for all c .
 - For each training example (\mathbf{x}, y) :
 - Classify with current weights: $\hat{y} = \arg \max_c \mathbf{w}_c^T \mathbf{x}$
 - If $y = \hat{y}$, do nothing.
 - Otherwise, update parameters:
 - $\mathbf{w}_y = \mathbf{w}_y + \mathbf{x}$ (raise score of right answer)
 - $\mathbf{w}_{\hat{y}} = \mathbf{w}_{\hat{y}} - \mathbf{x}$ (lower score of wrong answer).



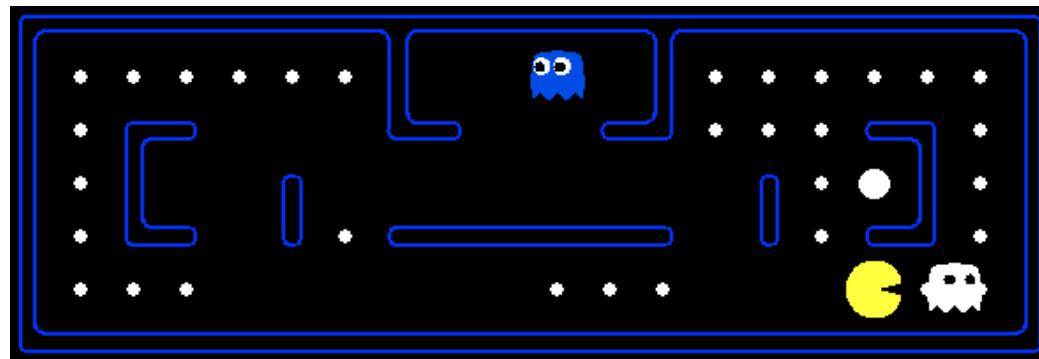
A video player control bar is visible at the bottom of the screen. It features a play button icon (a triangle pointing right) followed by the text "0:00". To the right is a horizontal progress bar consisting of a thin grey line with a small grey square marker indicating the current position.

▶ 0:00

Apprenticeship

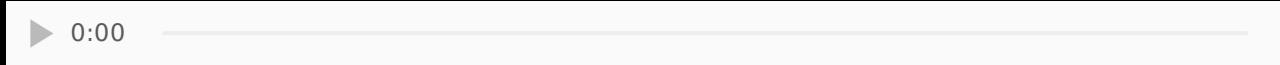
Can we learn to play Pacman from observations?

- Feature vectors $\mathbf{x} = g(\mathbf{s})$ are extracted from the game states \mathbf{s} . Output values y corresponds to actions a .
- State-action pairs (\mathbf{x}, y) are collected by observing an expert playing.
- We want to learn the actions that the expert would take in a given situation. That is, learn the mapping $f : \mathbb{R}^p \rightarrow \mathcal{A}$.
- This is a multiclass classification problem.





The Perceptron agent observes a very good Minimax-based agent for two games and updates its weight vectors as data are collected.





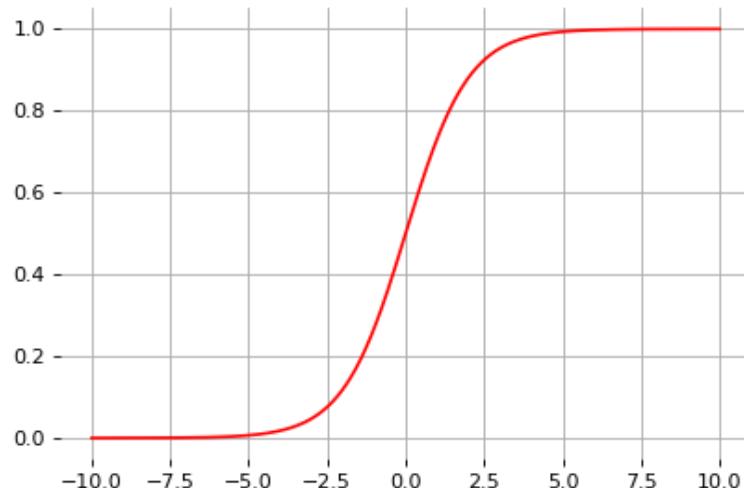
After two training episodes, the Perceptron agents plays.
No more Minimax!

Logistic regression

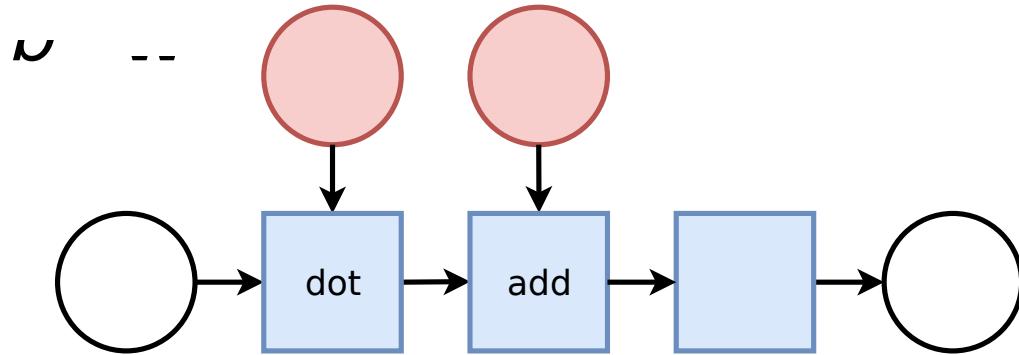
An alternative to the hard threshold model based on the **sign** function is to consider that $P(Y = 1|\mathbf{x})$ varies smoothly with \mathbf{x} . The **logistic regression** model postulates

$$P(Y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b),$$

where the sigmoid activation function $\sigma(x) = \frac{1}{1+\exp(-x)}$ looks like a soft heavyside:



In terms of **matrix operations**, the computational graph of h can be represented as:



where

- white nodes correspond to inputs and outputs;
- red nodes correspond to model parameters;
- blue nodes correspond to intermediate operations, which themselves produce intermediate output values (not represented).

This unit is the **core component** all neural networks!

Following the principle of maximum likelihood estimation, we have

$$\begin{aligned}
 & \arg \max_{\mathbf{w}, b} P(\mathbf{d} | \mathbf{w}, b) \\
 &= \arg \max_{\mathbf{w}, b} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} P(Y = y_i | \mathbf{x}_i, \mathbf{w}, b) \\
 &= \arg \max_{\mathbf{w}, b} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} \sigma(\mathbf{w}^T \mathbf{x}_i + b)^{y_i} (1 - \sigma(\mathbf{w}^T \mathbf{x}_i + b))^{1-y_i} \\
 &= \arg \min_{\mathbf{w}, b} \underbrace{\sum_{\mathbf{x}_i, y_i \in \mathbf{d}} -y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i + b) - (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i + b))}_{\mathcal{L}(\mathbf{w}, b) = \sum_i \ell(y_i, \hat{y}(\mathbf{x}_i; \mathbf{w}, b))}
 \end{aligned}$$

This loss is an instance of the **cross-entropy**

$$H(p, q) = \mathbb{E}_p[-\log q]$$

for $p = Y | \mathbf{x}_i$ and $q = \hat{Y} | \mathbf{x}_i$.

Gradient descent

Let $\mathcal{L}(\theta)$ denote a loss function defined over model parameters θ (e.g., \mathbf{w} and b).

To minimize $\mathcal{L}(\theta)$, **gradient descent** uses local linear information to iteratively move towards a (local) minimum.

For $\theta_0 \in \mathbb{R}^d$, a first-order approximation around θ_0 can be defined as

$$\hat{\mathcal{L}}(\theta_0 + \epsilon) = \mathcal{L}(\theta_0) + \epsilon^T \nabla_{\theta} \mathcal{L}(\theta_0) + \frac{1}{2\gamma} \|\epsilon\|^2.$$

A minimizer of the approximation $\hat{\mathcal{L}}(\theta_0 + \epsilon)$ is given for

$$\begin{aligned}\nabla_\epsilon \hat{\mathcal{L}}(\theta_0 + \epsilon) &= 0 \\ &= \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{\gamma} \epsilon,\end{aligned}$$

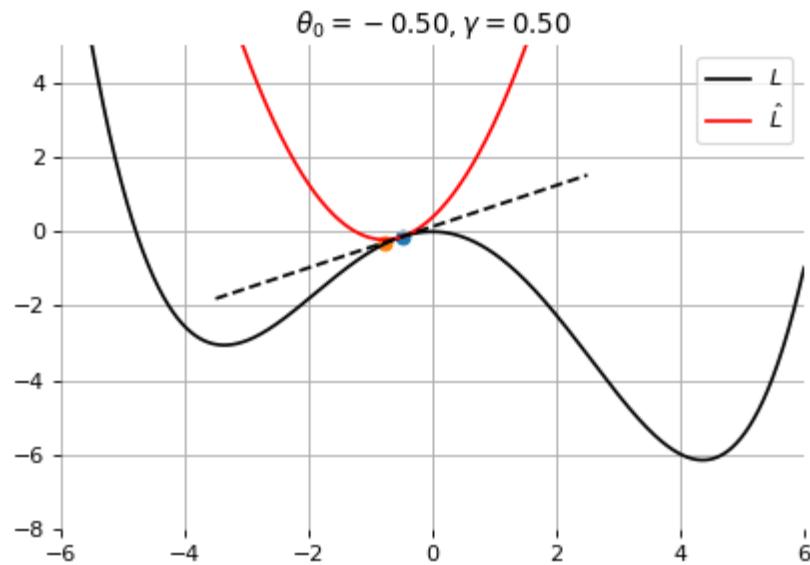
which results in the best improvement for the step $\epsilon = -\gamma \nabla_\theta \mathcal{L}(\theta_0)$.

Therefore, model parameters can be updated iteratively using the update rule:

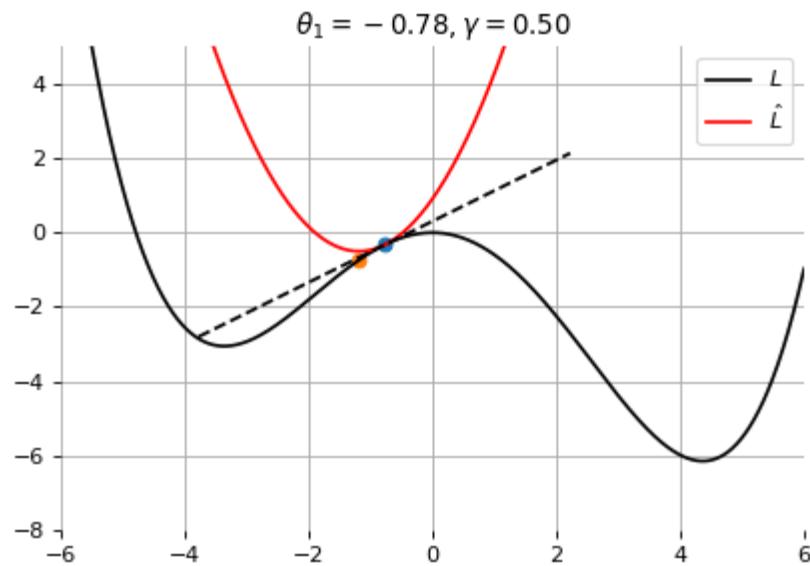
$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

Notes:

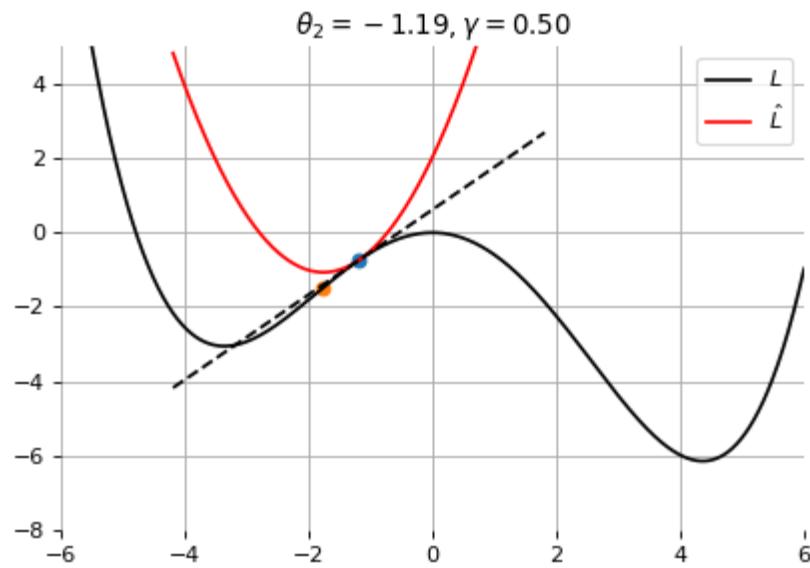
- θ_0 are the initial parameters of the model;
- γ is the **learning rate**;
- both are critical for the convergence of the update rule.



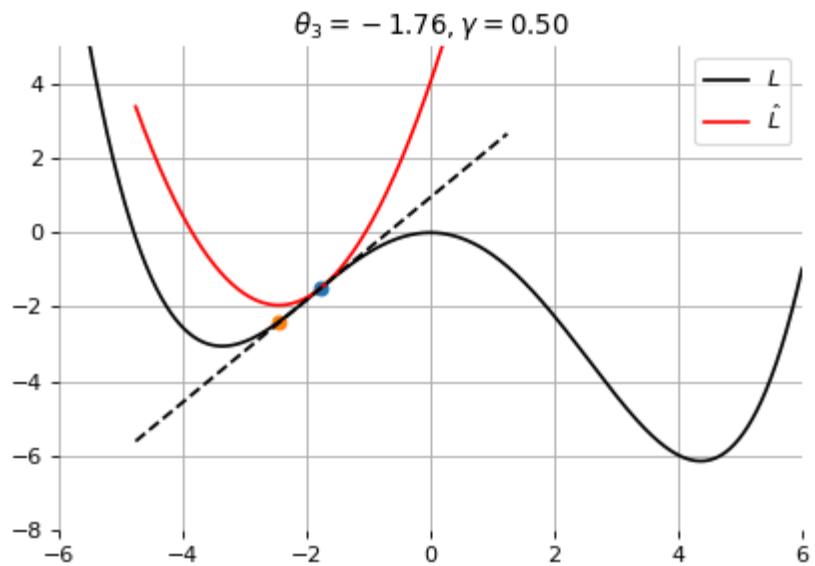
Example 1: Convergence to a local minima



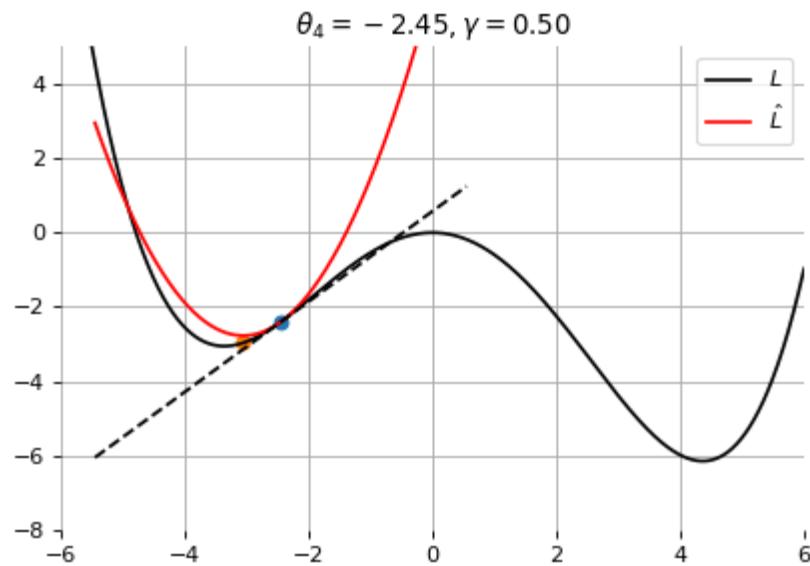
Example 1: Convergence to a local minima



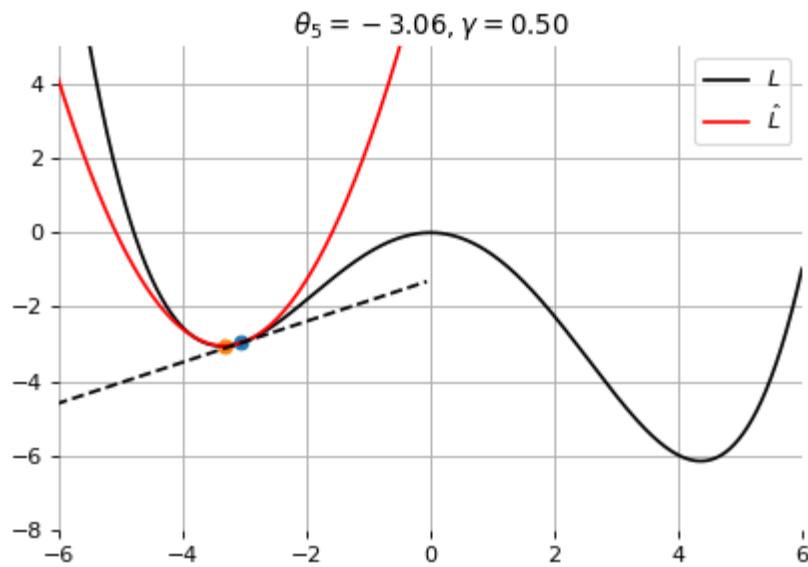
Example 1: Convergence to a local minima



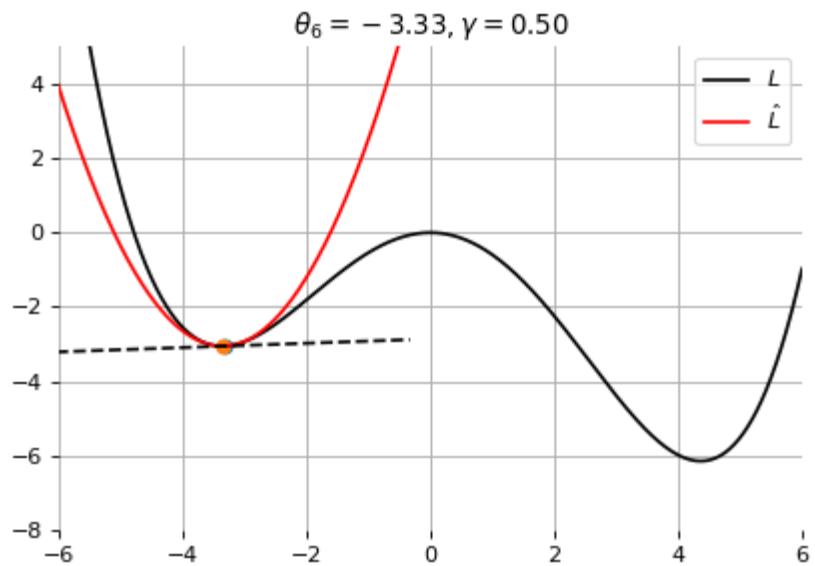
Example 1: Convergence to a local minima



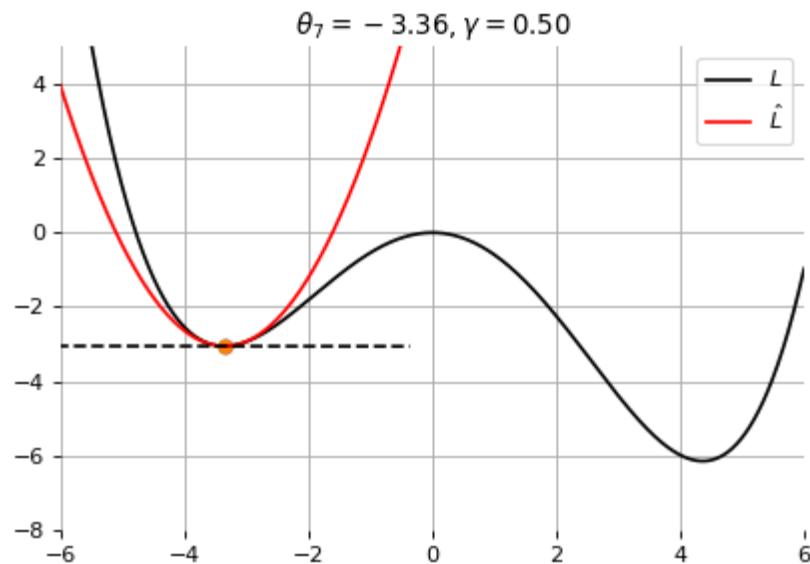
Example 1: Convergence to a local minima



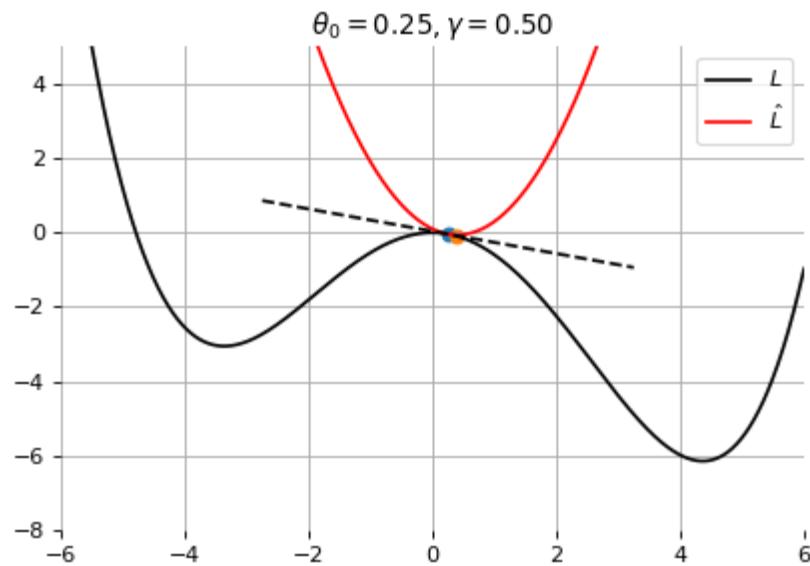
Example 1: Convergence to a local minima



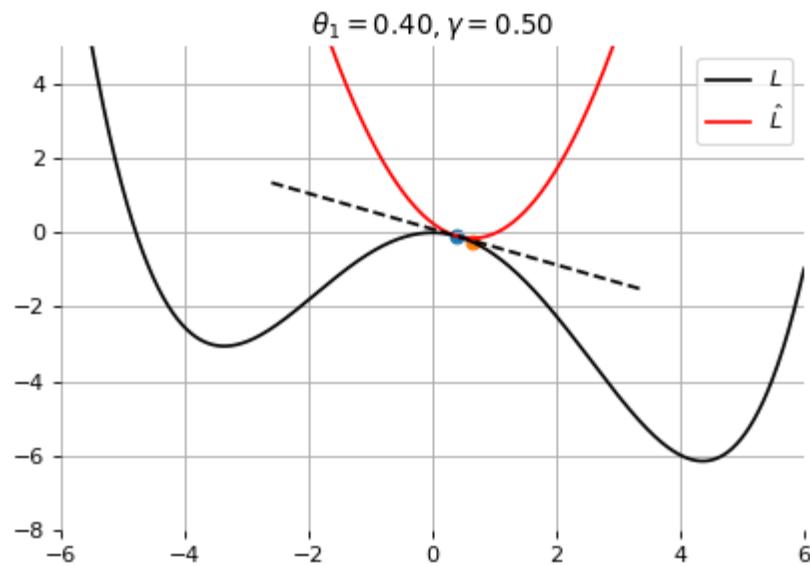
Example 1: Convergence to a local minima



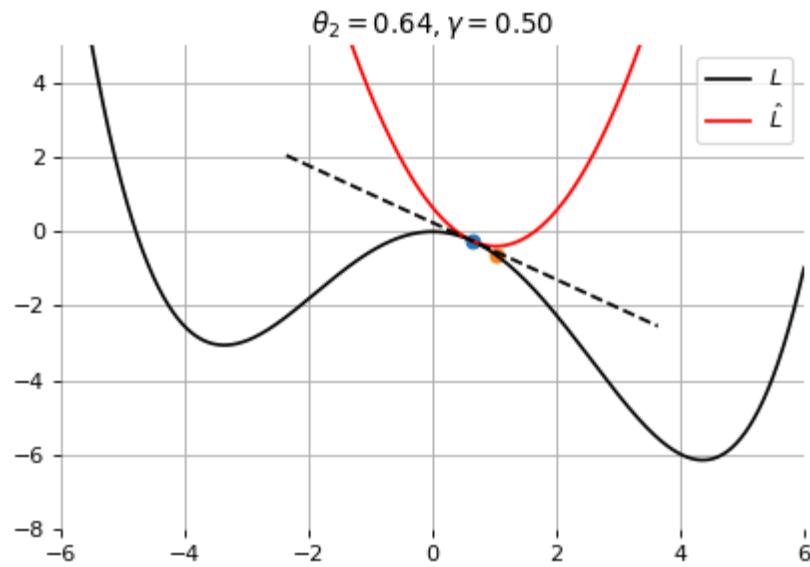
Example 1: Convergence to a local minima



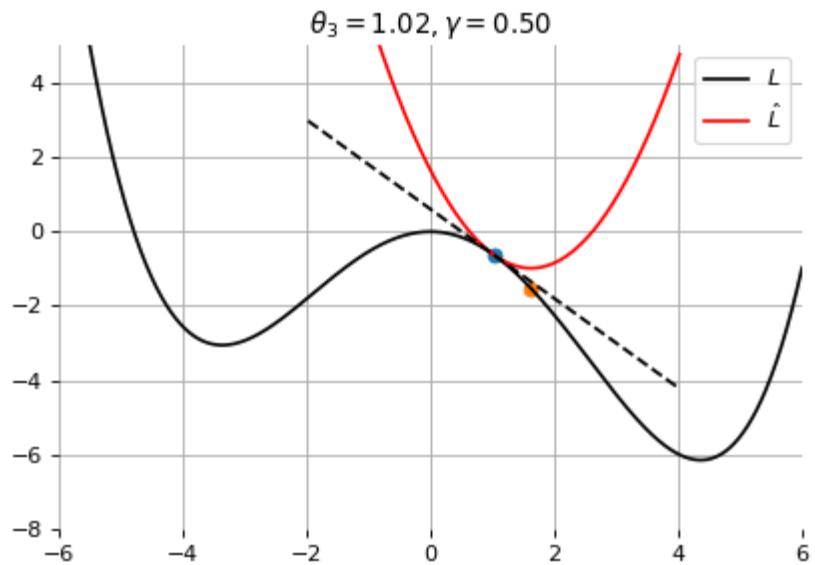
Example 2: Convergence to the global minima



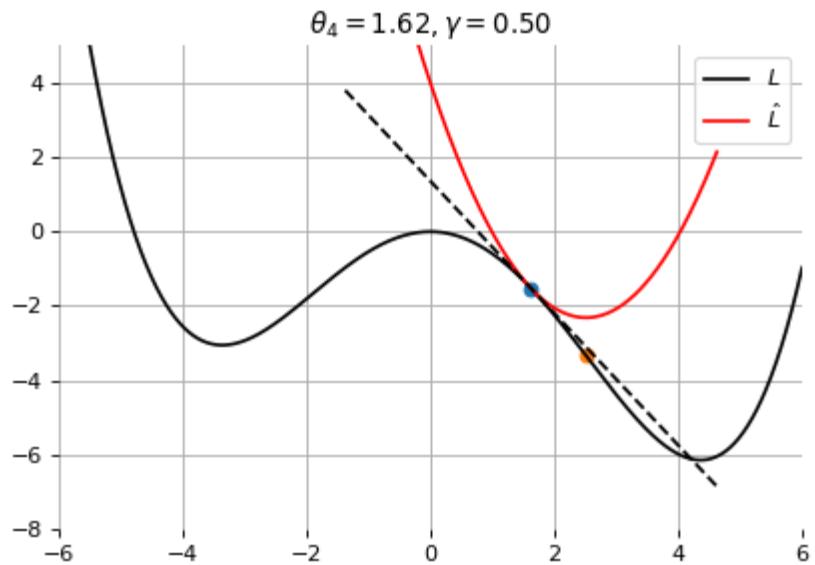
Example 2: Convergence to the global minima



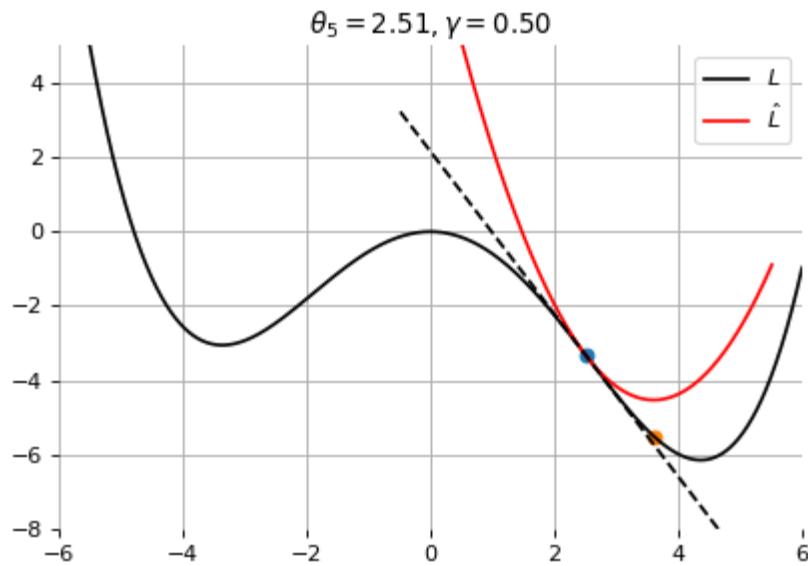
Example 2: Convergence to the global minima



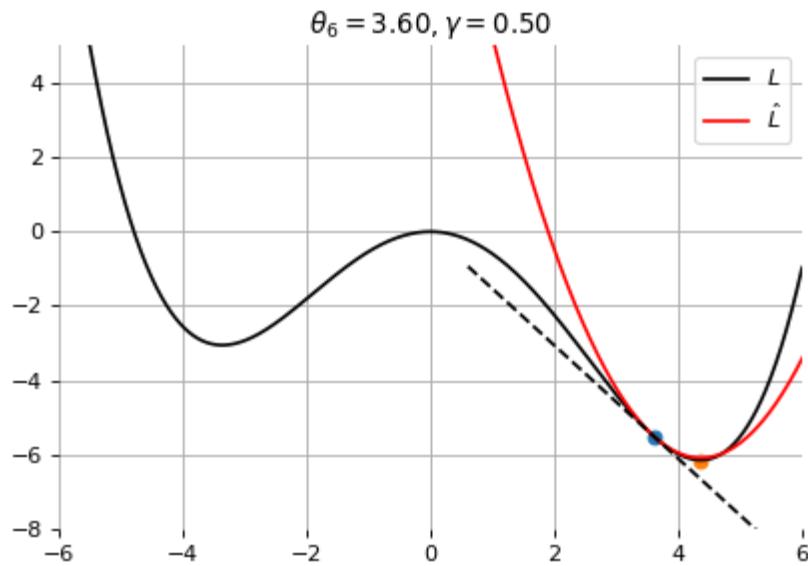
Example 2: Convergence to the global minima



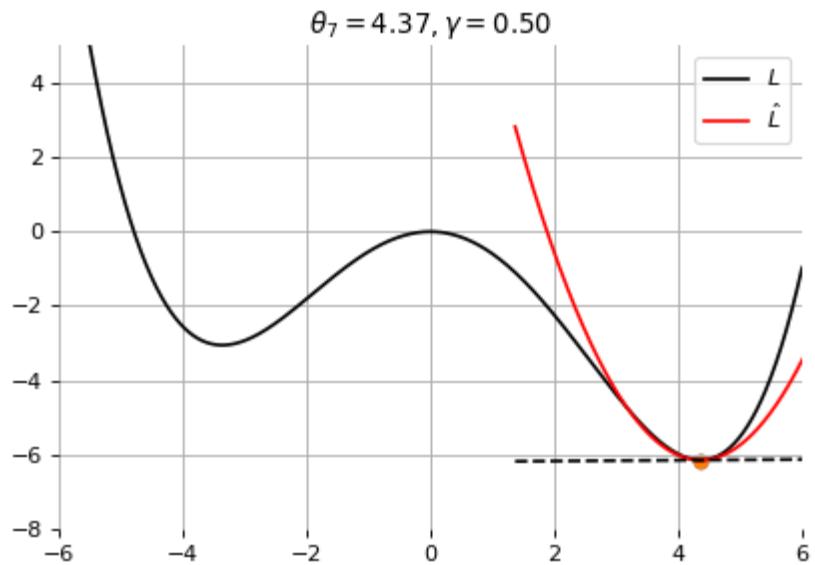
Example 2: Convergence to the global minima



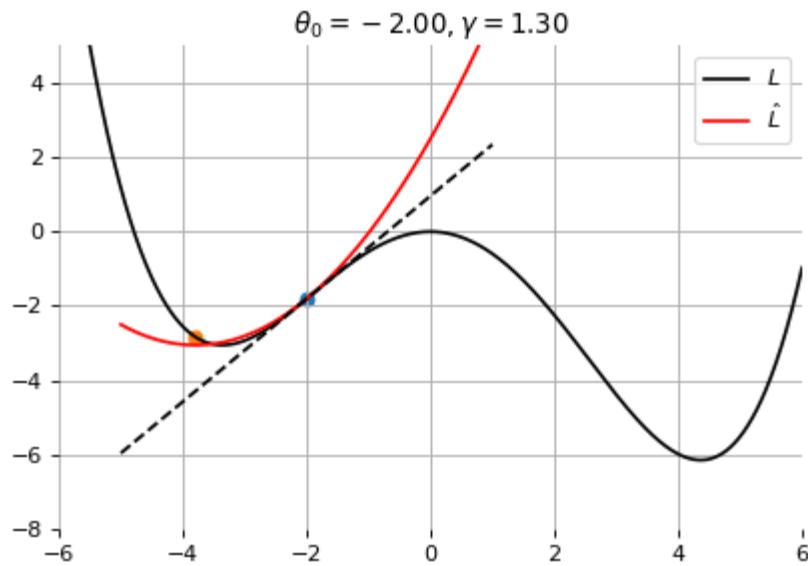
Example 2: Convergence to the global minima



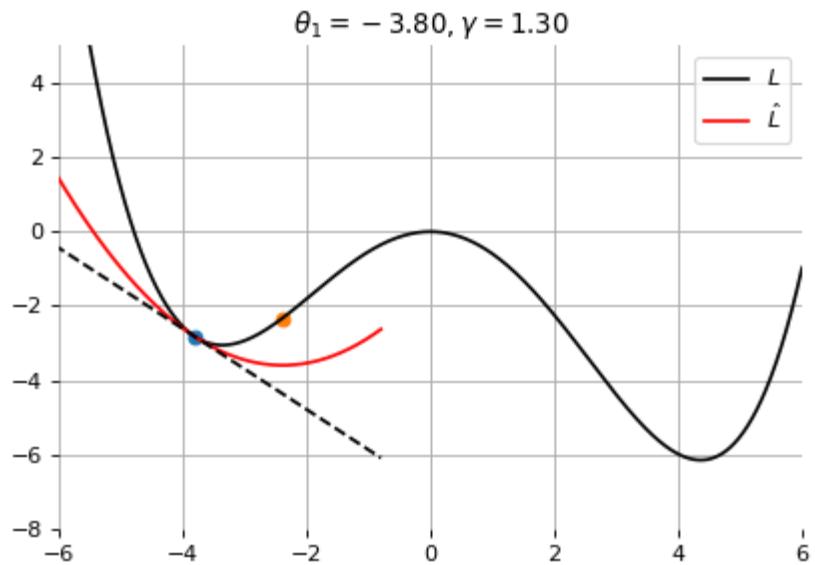
Example 2: Convergence to the global minima



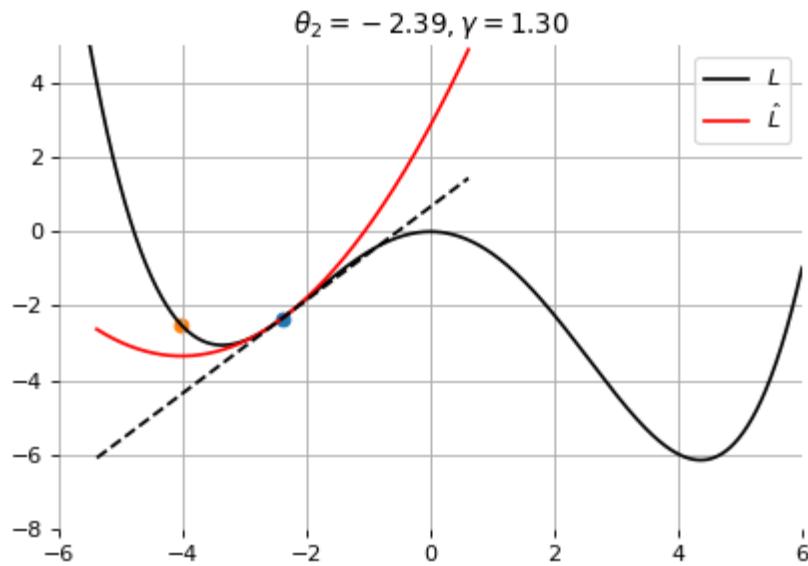
Example 2: Convergence to the global minima



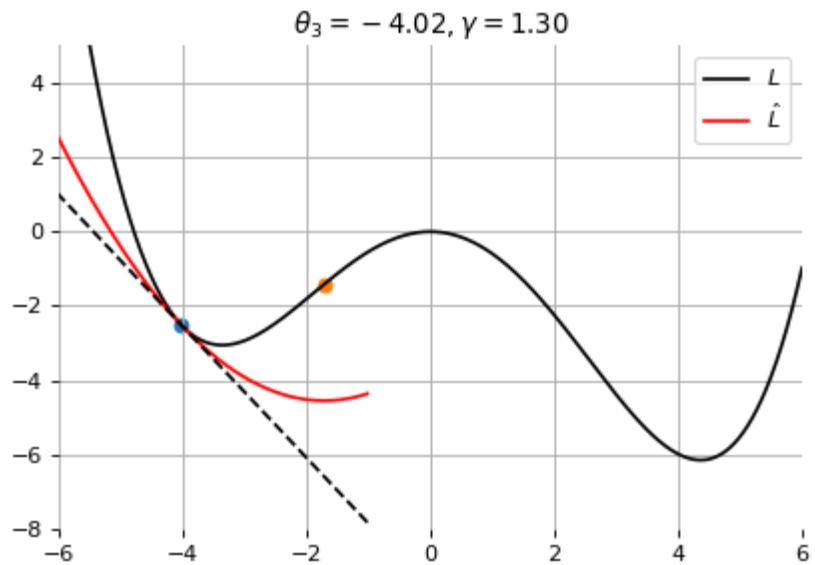
Example 3: Divergence due to a too large learning rate



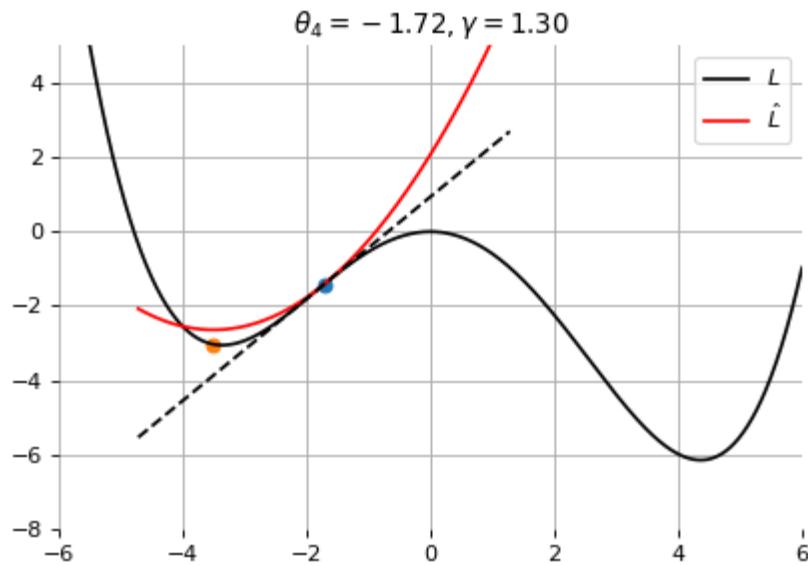
Example 3: Divergence due to a too large learning rate



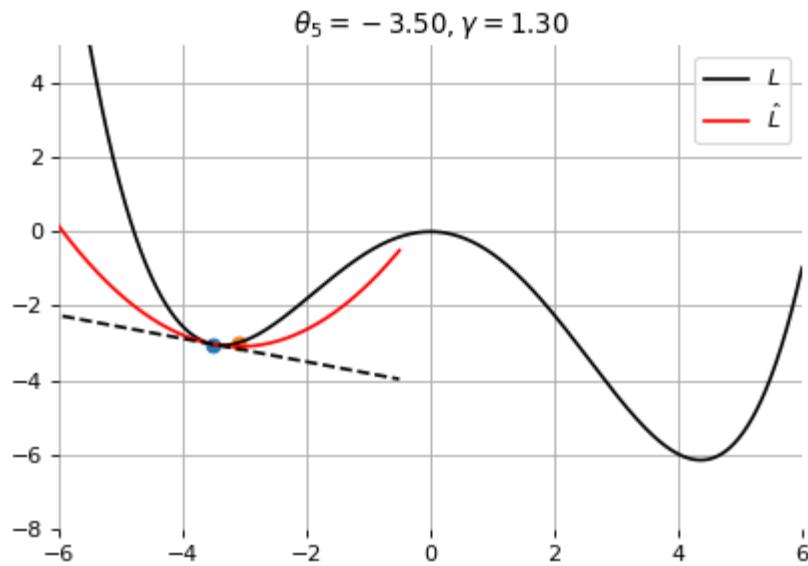
Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate

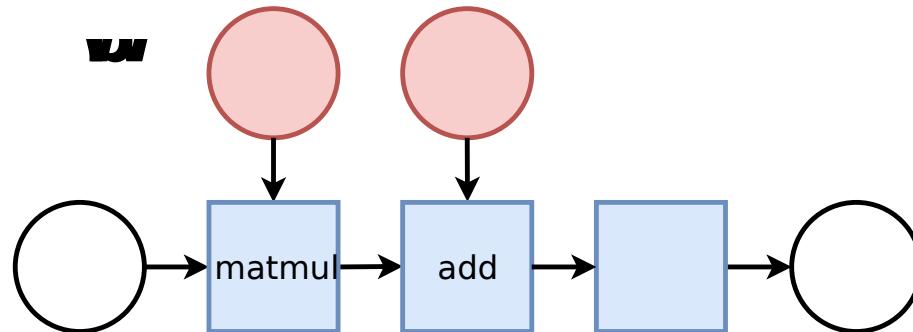
Layers

So far we considered the logistic unit $h = \sigma(\mathbf{w}^T \mathbf{x} + b)$, where $h \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$.

These units can be composed **in parallel** to form a **layer** with q outputs:

$$\mathbf{h} = \sigma(\mathbf{W}^T \mathbf{x} + \mathbf{b})$$

where $\mathbf{h} \in \mathbb{R}^q$, $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{W} \in \mathbb{R}^{p \times q}$, $\mathbf{b} \in \mathbb{R}^q$ and where $\sigma(\cdot)$ is upgraded to the element-wise sigmoid function.



Multi-layer perceptron

Similarly, layers can be composed **in series**, such that:

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1^T \mathbf{h}_0 + \mathbf{b}_1)$$

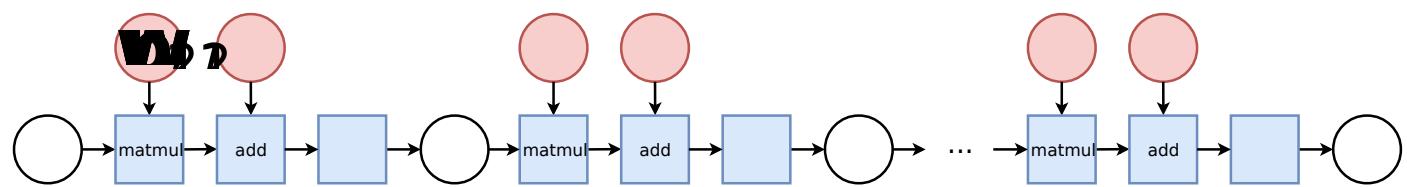
...

$$\mathbf{h}_L = \sigma(\mathbf{W}_L^T \mathbf{h}_{L-1} + \mathbf{b}_L)$$

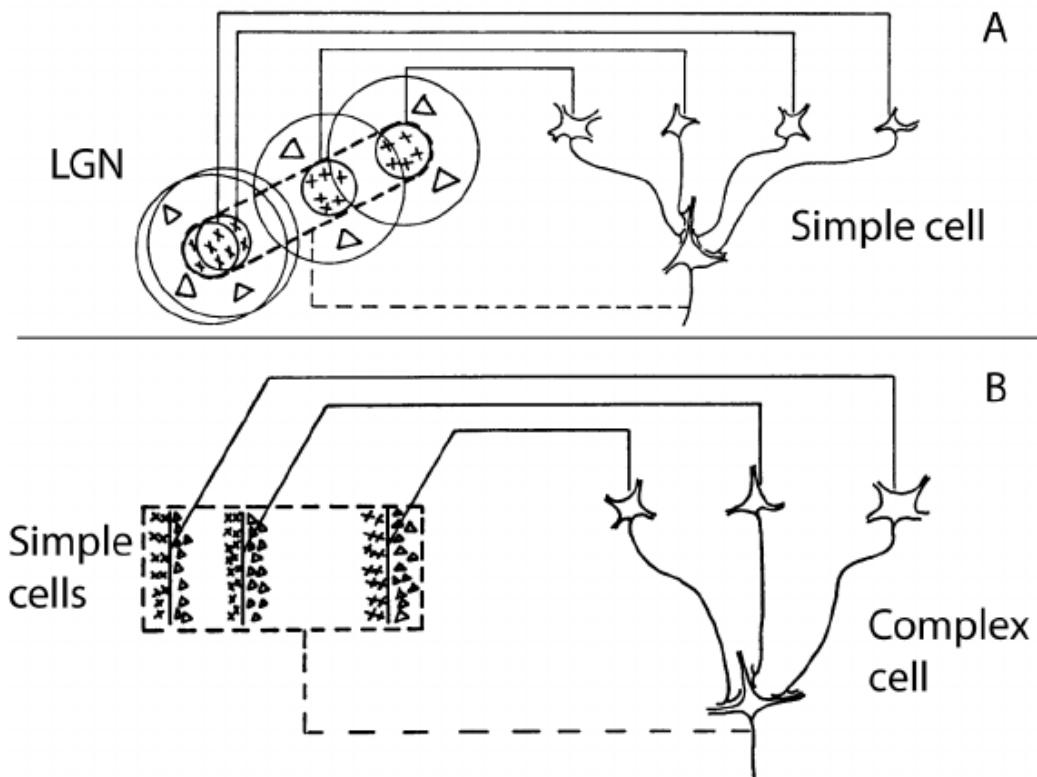
$$f(\mathbf{x}; \theta) = \mathbf{h}_L$$

where θ denotes the model parameters $\{\mathbf{W}_k, \mathbf{b}_k, \dots | k = 1, \dots, L\}$.

- This model is the **multi-layer perceptron**, also known as the fully connected feedforward network.
- Optionally, the last activation σ can be skipped to produce unbounded output values $\hat{y} \in \mathbb{R}$.

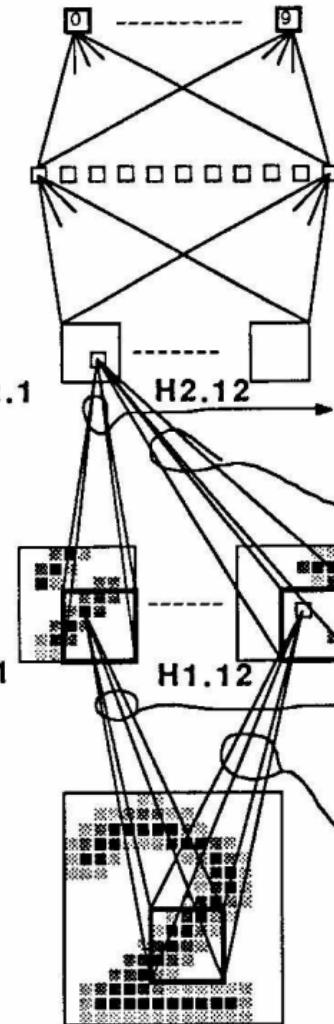


Convolutional networks



Hubel and Wiesel, 1962

10 output units



fully connected
~ 300 links

fully connected
~ 6000 links

~ 40,000 links
from 12 kernels
5 x 5 x 8

~20,000 links
from 12 kernels
5 x 5



Deep neural networks learn a hierarchical composition of features.



LeNet-1, LeCun et al, 1993.

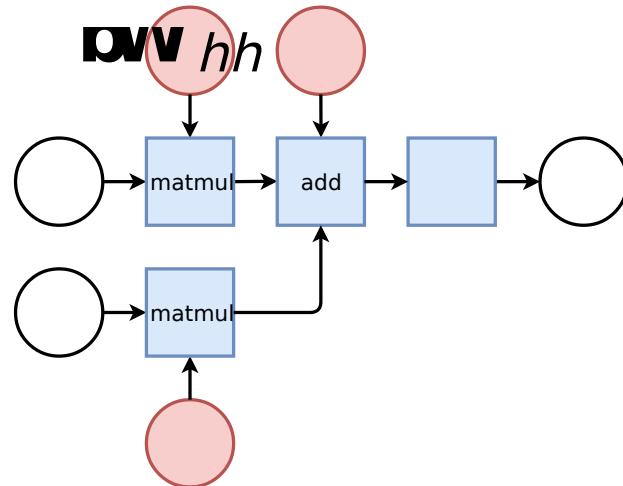
Recurrent networks

When the input is a sequence $\mathbf{x}_{1:T}$, the feedforward network can be made **recurrent** by computing a sequence $\mathbf{h}_{1:T}$ of hidden states where \mathbf{h}_t is a function of both \mathbf{x}_t and the previous hidden states in the sequence.

For example,

$$\mathbf{h}_t = \sigma(\mathbf{W}_{xh}^T \mathbf{x} + \mathbf{W}_{hh}^T \mathbf{h}_{t-1} + \mathbf{b}),$$

where \mathbf{h}_{t-1} is the previous hidden state in the sequence.



Notice how this is similar to filtering and dynamic decision networks:

- \mathbf{h}_t can be viewed as some current belief state;
- $\mathbf{x}_{1:T}$ is a sequence of observations;
- \mathbf{h}_{t+1} is computed from the current belief state \mathbf{h}_t and the latest evidence \mathbf{x}_t through some fixed computation (in this case a neural network, instead of being inferred from the assumed dynamics).
- \mathbf{h}_t can also be used to decide on some action, through another network f such that $a_t = f(\mathbf{h}_t; \theta)$.



A recurrent network playing Mario Kart.

Applications

Neural networks are now at the core of many [state-of-the-art systems](#), including:

- Image recognition
- Speech recognition and synthesis
- Natural language processing
- Scientific studies
- Reinforcement learning
- Autonomous agents

... and many many many others.



Autonomous drone navigation with deep learning



Pose detection and video synthesis



Deep learning and AI at NVIDIA

Unsupervised learning

Unsupervised learning

- Most of the learning performed by animals and humans is **unsupervised**.
 - Without labeled examples nor rewards.
- We learn how the world works by observing it:
 - We learn that the world is 3-dimensional.
 - We learn that objects can move independently of each other.
 - We learn **object permanence**.
 - We learn to predict what the world will look one second or one hour from now.



Common sense

We build a model of the world through [predictive unsupervised learning](#).

- This predictive model gives us **common sense**.
- Unsupervised learning discovers regularities in the world.

If I say: "Bernard picks up his bag and leaves the room".

You can **infer**:

- Bernard stood up, extended his arm to pick the bag, walked towards the door, opened the door, walked out.
- He and his bag are not in the room anymore.
- He probably did not dematerialized or flied out.



How do we do that?

We have no clue! (mostly)

Summary

- Learning is (supposedly) a key element of intelligence.
- Statistical learning aims at learning probabilistic models (their parameters or structures) automatically from data.
- Supervised learning is used to learn functions from a set of training examples.
 - Linear models are simple predictive models, effective on some tasks but usually insufficiently expressive.
 - Neural networks are defined as a composition of squashed linear models.
- Reinforcement learning = learning to behave in an unknown environment from sparse rewards.
- Unsupervised learning = learning a model of the world by observing it.

Going further?

- ELEN0062: Introduction to Machine Learning
- INFO8004: Advanced Machine Learning
- INFO8010: Deep Learning

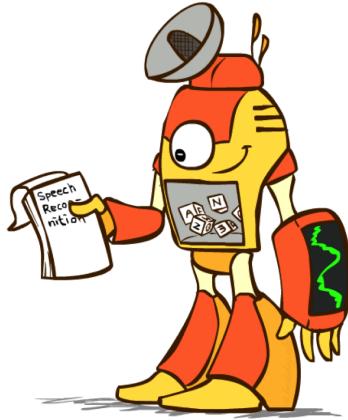
Introduction to Artificial Intelligence

Lecture 10: Communication

Prof. Gilles Louppe
g.louppé@uliege.be



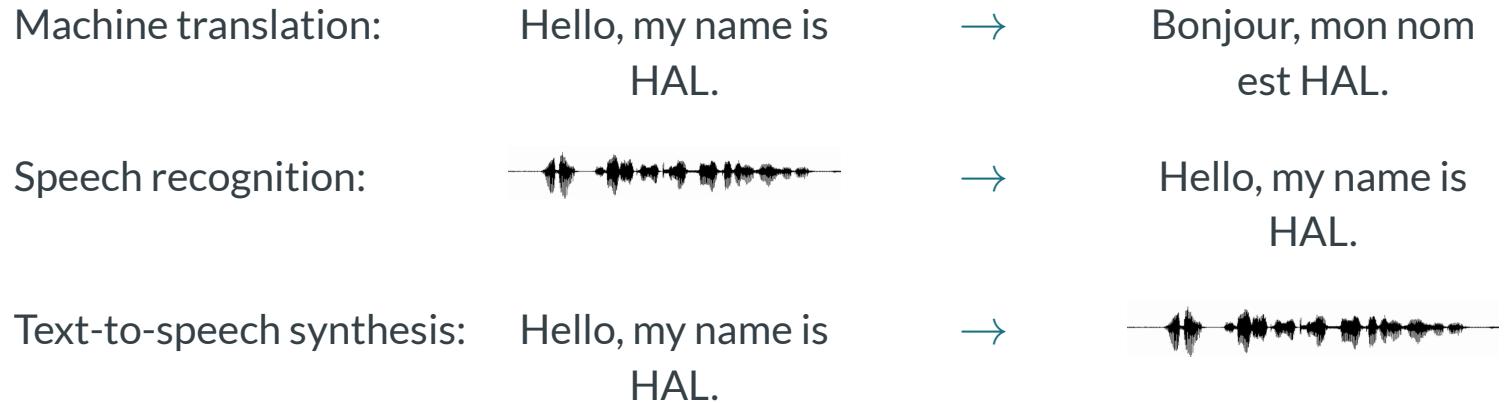
Today



Can you **talk** to an artificial agent? Can it understand what you say?

- Machine translation
- Speech recognition
- Text-to-speech synthesis

Sequence-to-sequence mapping



Machine translation

The screenshot shows a machine translation interface with two panels. The left panel has tabs for ANGLAIS - DÉTECTÉ, FRANÇAIS, ANGLAIS, and ARABE. The right panel has tabs for FRANÇAIS, ANGLAIS, and ARABE. Both panels have dropdown menus and a double-headed arrow icon. The left panel contains the text: "Our intelligence is what makes us human, and AI is an extension of that quality. (Yann Le Cun)" with a note below it: "Essayez avec cette orthographe : Our intelligence is what makes us human, and AI is an extension of that quality. (Yann *Lecun*)". The right panel shows the translated text: "Notre intelligence est ce qui fait de nous un humain, et l'intelligence artificielle est un prolongement de cette qualité. (Yann Le Cun)". There are also icons for microphone, speaker, and edit, along with a star icon and a comment button.

ANGLAIS - DÉTECTÉ FRANÇAIS ANGLAIS ARABE

FRANÇAIS ANGLAIS ARABE

Our intelligence is what makes us human, and AI is an extension of that quality. (Yann Le Cun)

Notre intelligence est ce qui fait de nous un humain, et l'intelligence artificielle est un prolongement de cette qualité. (Yann Le Cun)

Essayez avec cette orthographe : Our intelligence is what makes us human, and AI is an extension of that quality. (Yann *Lecun*)

94/5000

Envoyer des commentaires

Machine translation

Automatic translation of text from one natural language (the source) to another (the target), while preserving the intended meaning.

[Q] How would you engineer a machine translation system?

Issue of dictionary lookups

顶部 /**top**/roof/

顶端 /summit/peak/**top**/apex/

顶头 /coming directly towards one/**top**/end/

盖 /lid/**top**/cover/canopy/build/Gai/

盖帽 /surpass/**top**/

极 /extremely/pole/utmost/**top**/collect/receive/

尖峰 /peak/**top**/

面 /fade/side/surface/aspect/**top**/face/flour/

摘心 /**top**/topping/

Natural languages are not 1:1 mappings of each other!

The screenshot shows a translation interface with two main sections. The left section is for English, and the right section is for French. Both sections have dropdown menus for selecting other languages (English, Spanish, French) and a 'Send feedback' button at the bottom right.

ENGLISH - DETECTED ENGLISH SPANISH FRENCH ↴ ↵ **FRENCH** ENGLISH SPANISH ↴

The soccer ball hit the window. It broke. × Le ballon de football a frappé la fenêtre. Ça s'est cassé. ☆

Microphone icon, speaker icon, 41/5000, edit icon, speaker icon, copy icon, edit icon, link icon.

Send feedback

To obtain a correct translation, one must decide whether "it" refers to the soccer ball or to the window.

Therefore, one must understand physics as well as language.

History



Warren Weaver

When I look at an article in Russian, I say: "This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode."



John Pierce

"Machine Translation" presumably means going by algorithm from machine-readable source text to useful target text... In this context, there has been no machine translation...

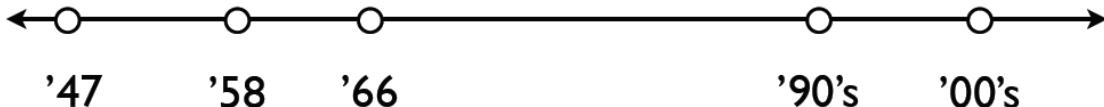
Berkeley's first MT grant

MT is the "first" non-numeral compute task

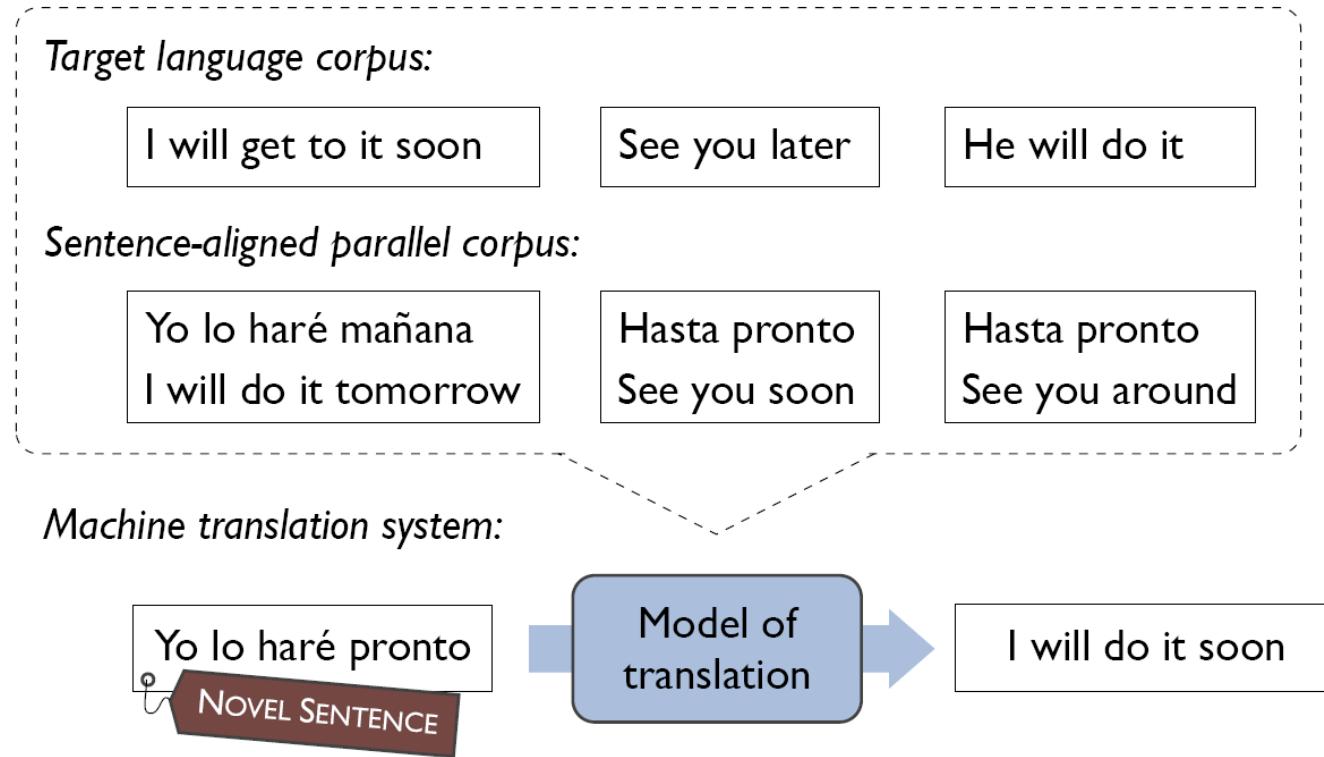
ALPAC report deems MT bad

Statistical MT thrives

Statistical data-driven approach introduced



Data-driven machine translation



Machine translation systems

Translation systems must model the source and target languages, but systems vary in the type of models they use.

- Some systems analyze the source language text all the way into an **interlingua knowledge representation** and then generate sentences in the target language from that representation.
- Other systems are based on a **transfer model**. They keep a database of translation rules and whenever the rule matches, they translate directly. Transfer can occur at the lexical, syntactic or semantic level.

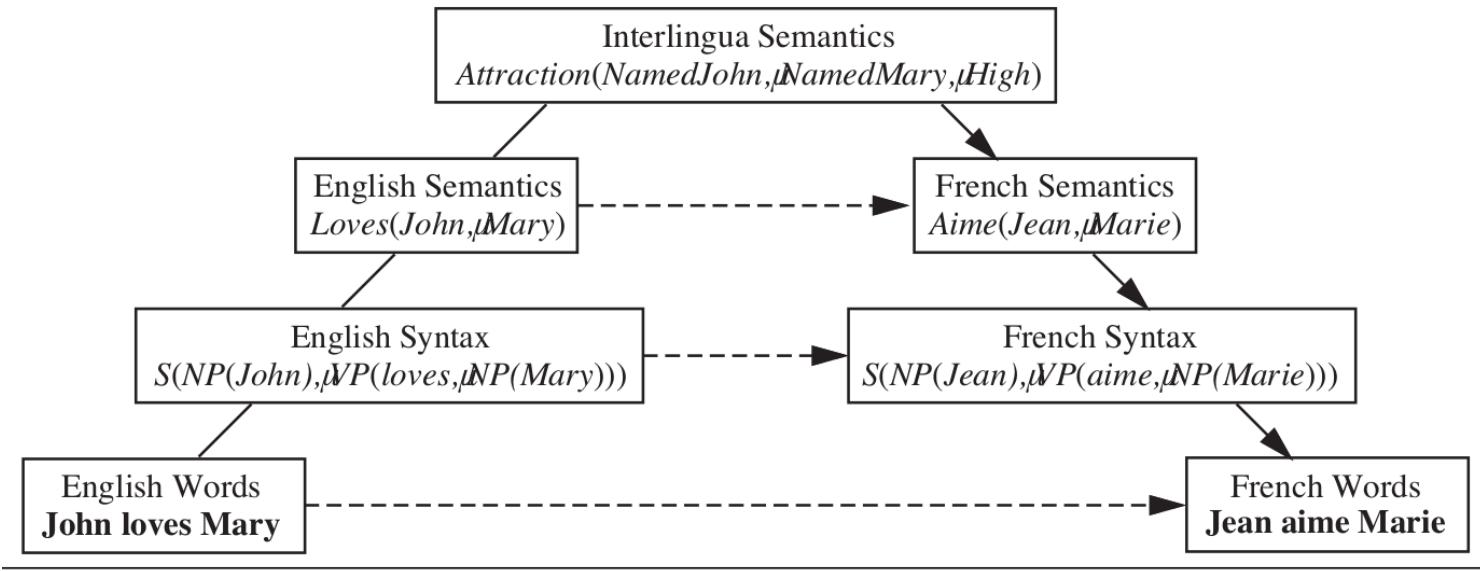


Figure 23.12 The Vauquois triangle: schematic diagram of the choices for a machine translation system (Vauquois, 1968). We start with English text at the top. An interlingua-based system follows the solid lines, parsing English first into a syntactic form, then into a semantic representation and an interlingua representation, and then through generation to a semantic, syntactic, and lexical form in French. A transfer-based system uses the dashed lines as a shortcut. Different systems make the transfer at different points; some make it at multiple points.

Statistical machine translation

To translate an English sentence e into a French sentence f , we seek the strings of words f^* such that

$$f^* = \arg \max_f P(f|e).$$

- The language model $P(f|e)$ is learned from a **bilingual corpus**, i.e. a collection of parallel texts, each an English/French pair.
- Most of the English sentences to be translated will be novel, but will be composed of phrases that have been seen before.
- The corresponding French phrases will be reassembled to form a French sentence that makes sense.

Given an English source sentence e , finding a French translation f is a matter of three steps:

- Break e into phrases e_1, \dots, e_n .
- For each phrase e_i , choose a corresponding French phrase f_i . We use the notation $P(f_i|e_i)$ for the phrasal probability that f_i is a translation of e_i .
- Choose a permutation of the phrases f_1, \dots, f_n . For each f_i , we choose a distortion

$$d_i = \text{start}(f_i) - \text{end}(f_{i-1}) - 1,$$

which is the number of words that phrase f_i has moved with respect to f_{i-1} ; positive for moving to the right, negative for moving the left.

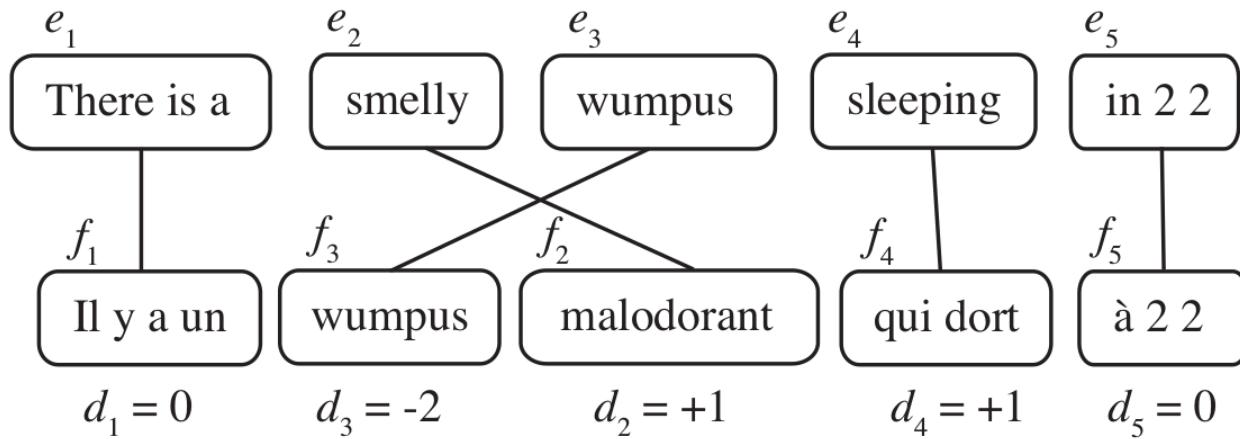


Figure 23.13 Candidate French phrases for each phrase of an English sentence, with distortion (d) values for each French phrase.

We define the probability $P(f, d|e)$ that the sequence of phrases f with distortions d is a translation of the sequence of phrases e .

Assuming that each phrase translation and each distortion is independent of the others, we have

$$P(f, d|e) = \prod_i P(f_i|e_i)P(d_i).$$

- The best f and e cannot be found through enumeration because of the combinatorial explosion.
- Instead, local beam search with a heuristic that estimates probability has proven effective at finding a nearly-most-probable translation.

All that remains is to learn the phrasal and distortion probabilities:

1. Find parallel texts.
2. Segment into sentences.
3. Align sentences.
4. Align phrases.
5. Extract distortions.
6. Improve estimates with expectation-maximization.

Neural machine translation

Modern machine translation systems are all based on **neural networks** of various types, often architectured as compositions of

- recurrent networks for sequence-to-sequence learning,
- convolutional networks for modeling spatial dependencies.

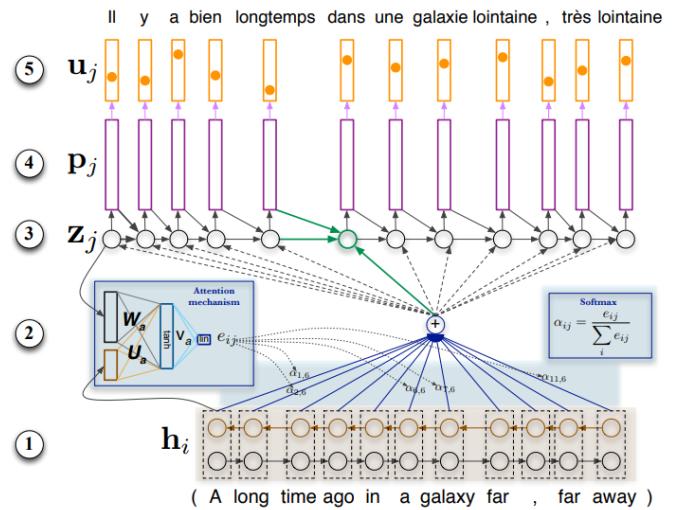


Google's Neural Machine Translation

...reduced translation errors by an average
of 60% when compared to the prior Google
Translate technology

Attention-based recurrent neural network

- Encoder: bidirectional RNN, producing a set of annotation vectors \mathbf{h}_i .
- Decoder: attention-based.
 - Compute attention weights α_{ij} .
 - Compute the weighted sum of the annotation vectors, as a way to align the input words to the output words.
 - Decode using the context vector, the embedding of the previous output word and the hidden state.

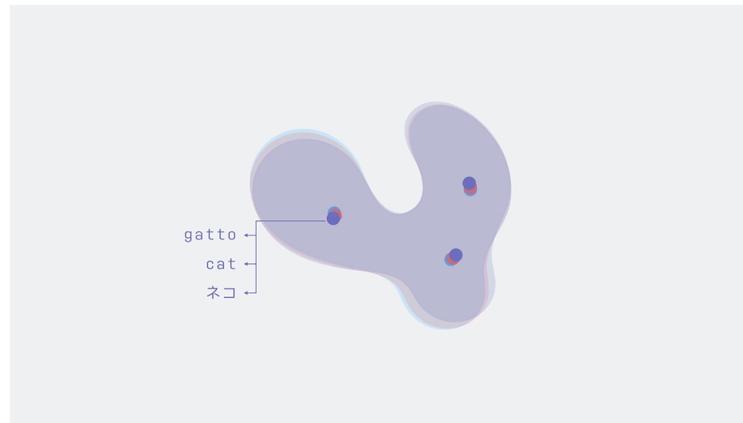


Unsupervised machine translation

- The latest approaches do not even need to have a bilingual corpus!
- Machine translation can be learned in a **fully unsupervised** way with unsupervised alignment.

Word-by-word translation:

- Learn a neural word embedding trained to predict the words around a given word using a context.
- Embedding in different languages share similar neighborhood structure.
- The system learn rotation of the word embedding in one language to match the word embedding in the other language, using adversarial training.
- This can be used to infer a fairly accurate bilingual dictionary without access to any translation!



Translating sentences:

- Bootstrap the translation model with word-by-word initialization.
- The neural translation model must be able to reconstruct a sentence in a given language from a noisy version of it.
- The model also learns to reconstruct any source sentence given a noisy translation of the same sentence in the target domain, and vice-versa.
- The source and target sentence latent representations are constrained to have the same latent distributions through adversarial training.

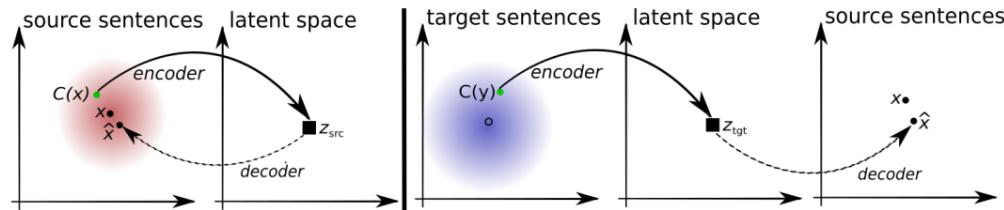
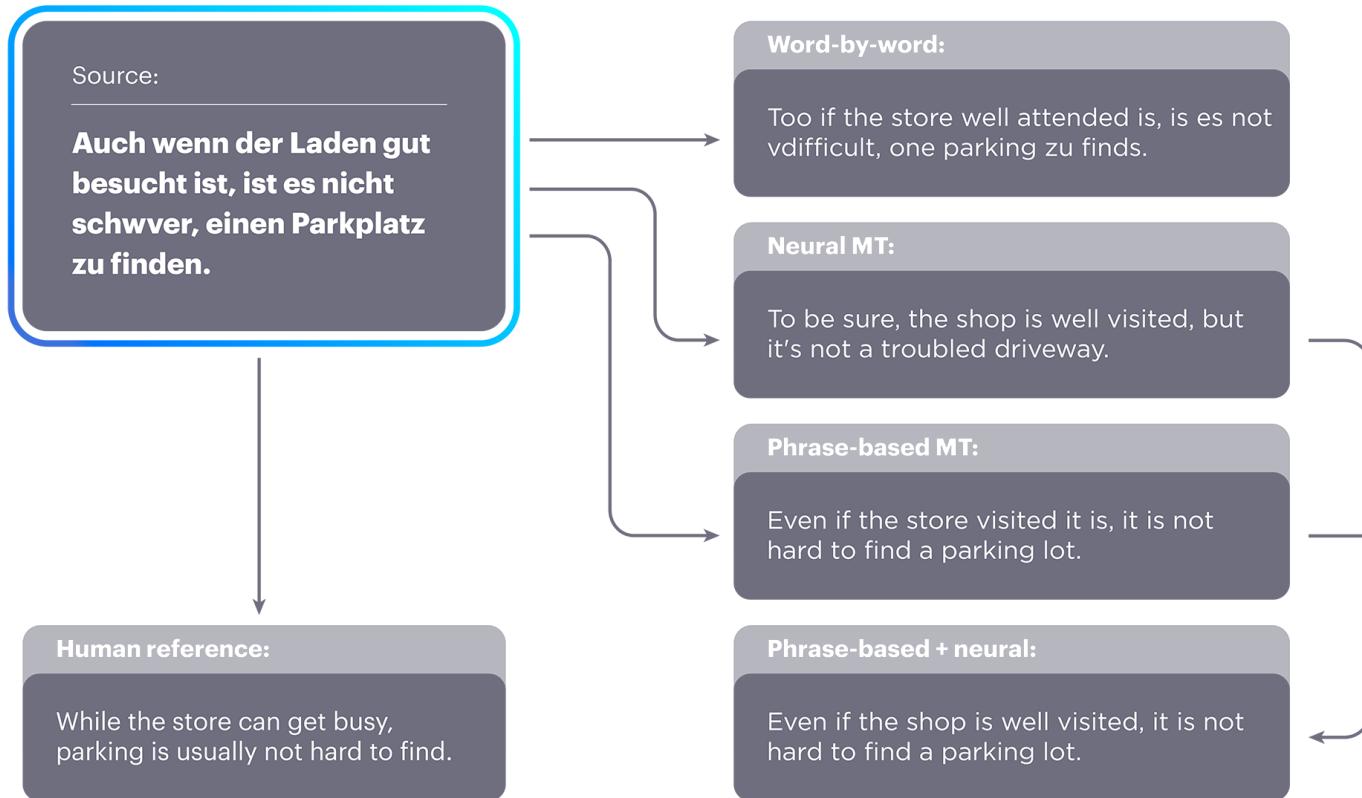


Figure 1: Toy illustration of the principles guiding the design of our objective function. Left (auto-encoding): the model is trained to reconstruct a sentence from a noisy version of it. x is the target, $C(x)$ is the noisy input, \hat{x} is the reconstruction. Right (translation): the model is trained to translate a sentence in the other domain. The input is a noisy translation (in this case, from source-to-target) produced by the model itself, M , at the previous iteration (t), $y = M^{(t)}(x)$. The model is symmetric, and we repeat the same process in the other language. See text for more details.



Speech recognition



Recognition as inference



Speech recognition can be viewed as an instance of the problem of **finding the most likely sequence** of state variables $\mathbf{w}_{1:L}$, given a sequence of observations $\mathbf{y}_{1:T}$.

- In this case, (hidden) state variables are the words and the observations are sounds.
- The input audio waveform from a microphone is converted into a sequence of fixed size acoustic vectors $\mathbf{y}_{1:T}$ in a process called **feature extraction**.
- The decoder attempts to find the sequence of words $\mathbf{w}_{1:L} = w_1, \dots, w_L$ which is the most likely to have generated $\mathbf{y}_{1:T}$:

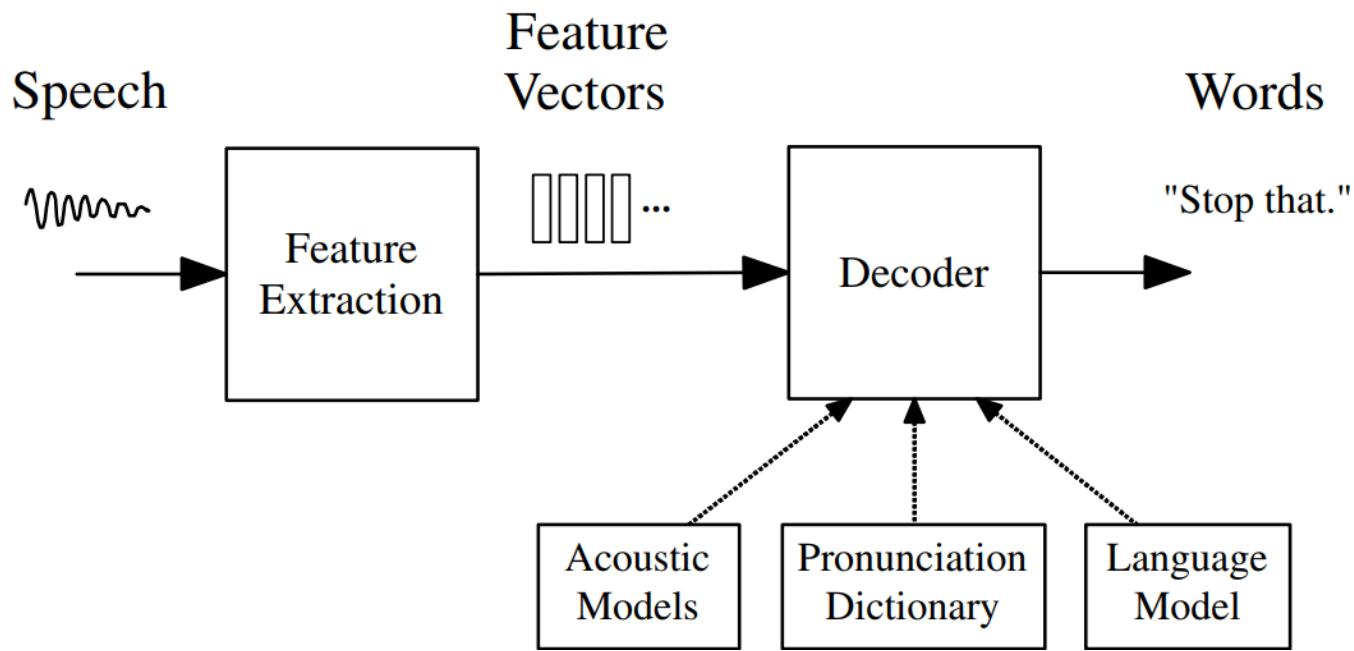
$$\hat{\mathbf{w}}_{1:L} = \arg \max_{\mathbf{w}_{1:L}} P(\mathbf{w}_{1:L} | \mathbf{y}_{1:T})$$

Since $P(\mathbf{w}_{1:L} | \mathbf{y}_{1:T})$ is difficult to model directly, Bayes' rule is used to solve the equivalent problem

$$\hat{\mathbf{w}}_{1:L} = \arg \max_{\mathbf{w}_{1:L}} p(\mathbf{y}_{1:T} | \mathbf{w}_{1:L}) P(\mathbf{w}_{1:L}),$$

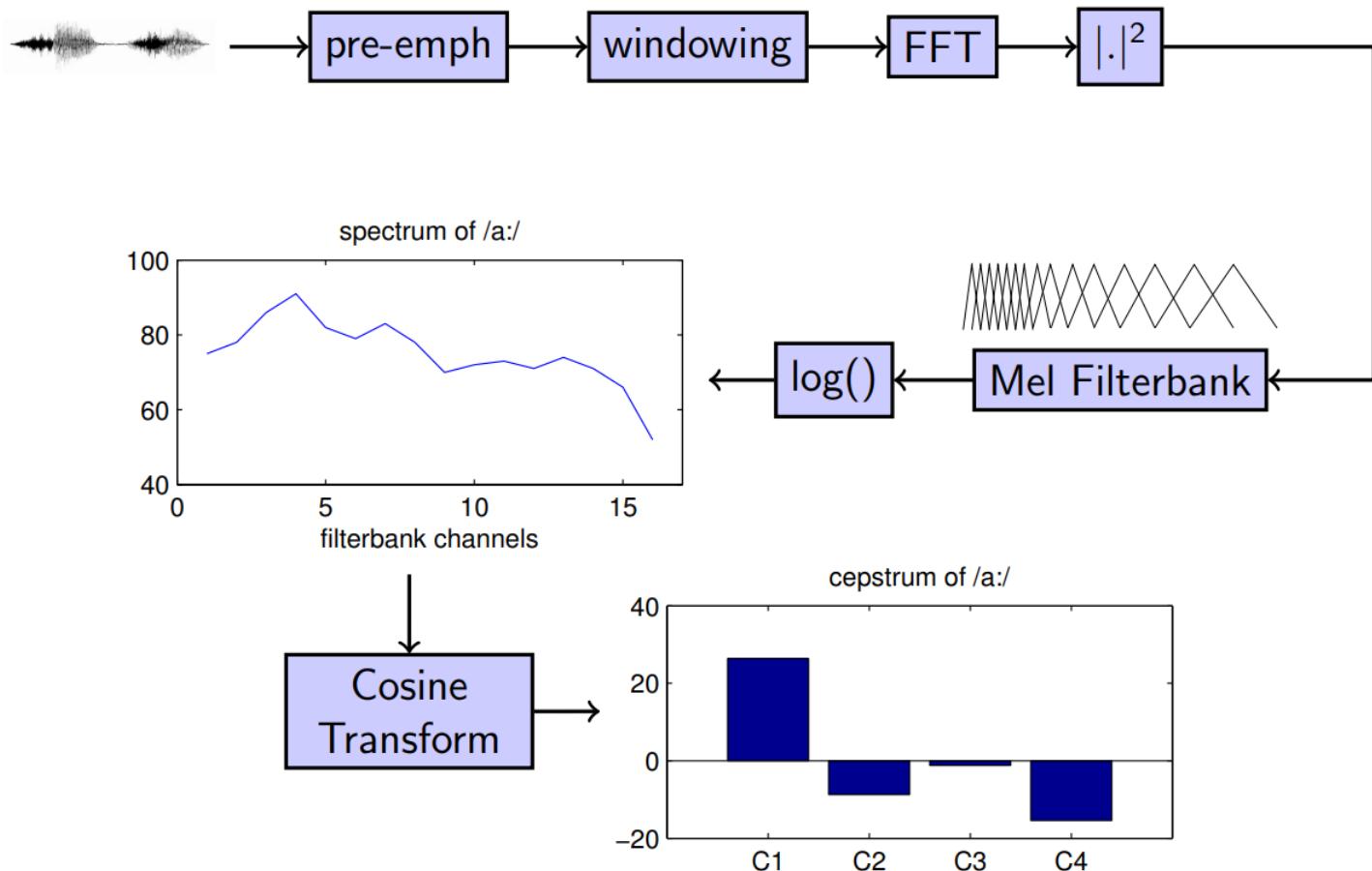
where

- the likelihood $p(\mathbf{y}_{1:T} | \mathbf{w}_{1:L})$ is the **acoustic model**;
- the prior $P(\mathbf{w}_{1:L})$ is the **language model**.

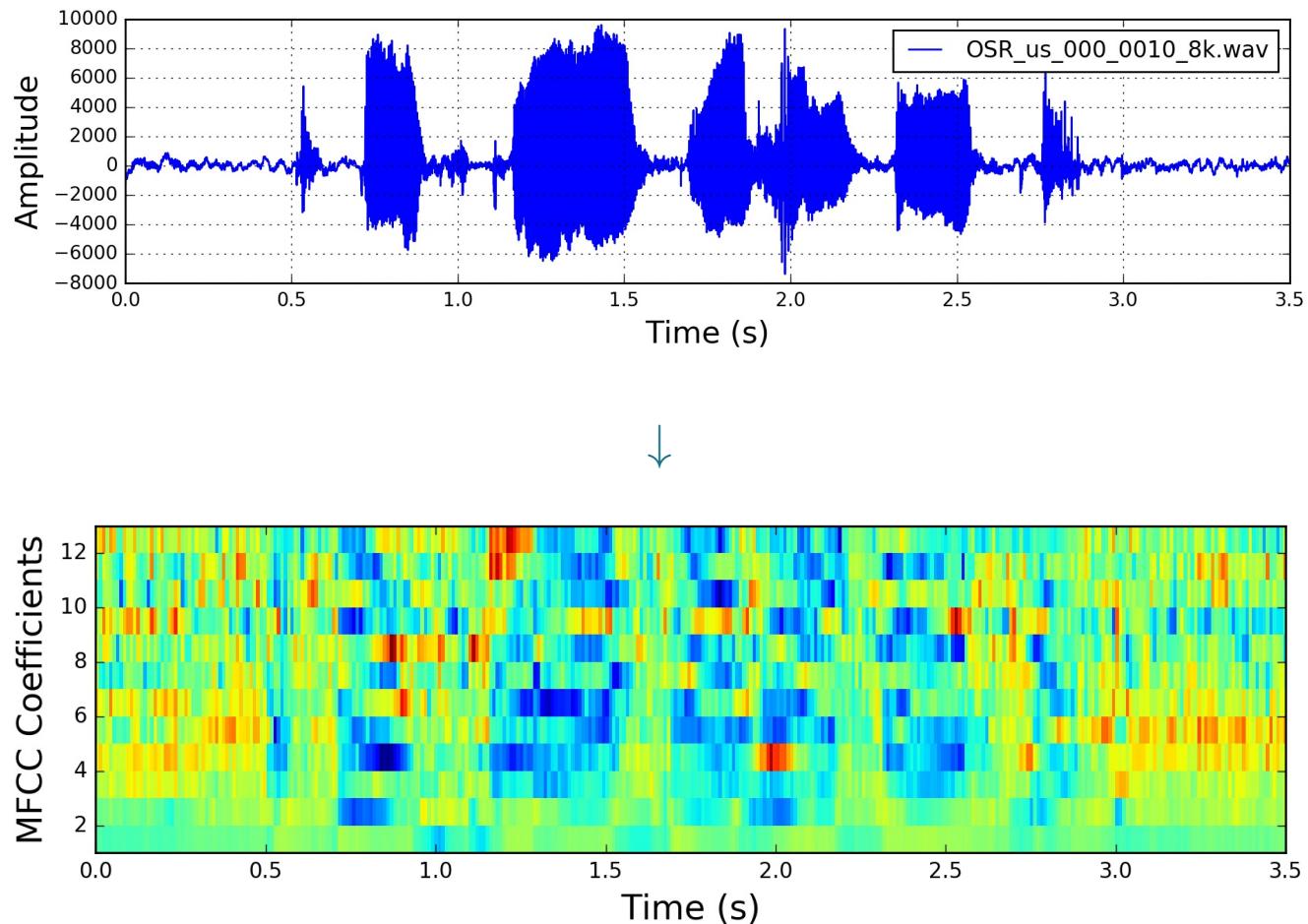


Feature extraction

- The feature extraction seeks to provide a compact representation $\mathbf{y}_{1:T}$ of the speech waveform.
- This form should minimize the loss of information that discriminates between words.
- One of the most widely used encoding schemes is based on **mel-frequency cepstral coefficients** (MFCCs).



MFCCs calculation.



Feature extraction from the signal in the time domain to MFCCs.

Acoustic model

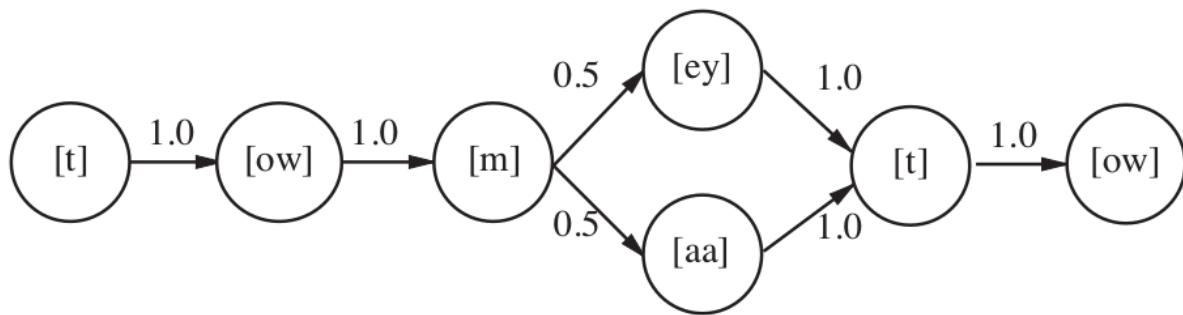
A spoken word w is decomposed into a sequence of K_w basic sounds called **base phones** (such as vowels or consonants).

- This sequence is called its pronunciation $\mathbf{q}_{1:K_w}^w = q_1, \dots, q_{K_w}$.
- Pronunciations are related to words through **pronunciations models** defined for each word.

Vowels		Consonants B–N		Consonants P–Z	
Phone	Example	Phone	Example	Phone	Example
[iy]	<u>b</u>eat	[b]	<u>b</u>et	[p]	<u>p</u>et
[ih]	<u>b</u>it	[ch]	<u>C</u>het	[r]	<u>r</u>at
[eh]	<u>b</u>et	[d]	<u>d</u>ebt	[s]	<u>s</u>et
[æ]	<u>b</u>at	[f]	<u>f</u>at	[sh]	<u>sh</u>oe
[ah]	<u>b</u>ut	[g]	<u>g</u>et	[t]	<u>t</u>en
[ao]	<u>b</u>ought	[hh]	<u>h</u>at	[th]	<u>th</u>ick
[ow]	<u>b</u>oat	[hv]	<u>h</u>igh	[dh]	<u>th</u>at
[uh]	<u>b</u>ook	[jh]	<u>j</u>et	[dx]	<u>b</u>utter
[ey]	<u>b</u>ait	[k]	<u>k</u>ick	[v]	<u>v</u>et
[er]	<u>B</u>ert	[l]	<u>l</u>et	[w]	<u>w</u>et
[ay]	<u>b</u>uy	[el]	<u>b</u>ottle	[wh]	<u>w</u>hich
[oy]	<u>b</u>oy	[m]	<u>m</u>et	[y]	<u>y</u>et
[axr]	<u>d</u>iner	[em]	<u>b</u>ottom	[z]	<u>z</u>oo
[aw]	<u>d</u>own	[n]	<u>n</u>et	[zh]	<u>m</u>easure
[ax]	<u>a</u>bout	[en]	<u>b</u>utton		
[ix]	<u>r</u>oses	[ng]	<u>s</u>ing		
[aa]	<u>c</u>ot	[eng]	<u>w</u>ashing	[-]	<i>silence</i>

Figure 23.14 The ARPA phonetic alphabet, or ARPAbet, listing all the phones used in American English. There are several alternative notations, including an International Phonetic Alphabet (IPA), which contains the phones in all known languages.

(a) Word model with dialect variation:



(b) Word model with coarticulation and dialect variations

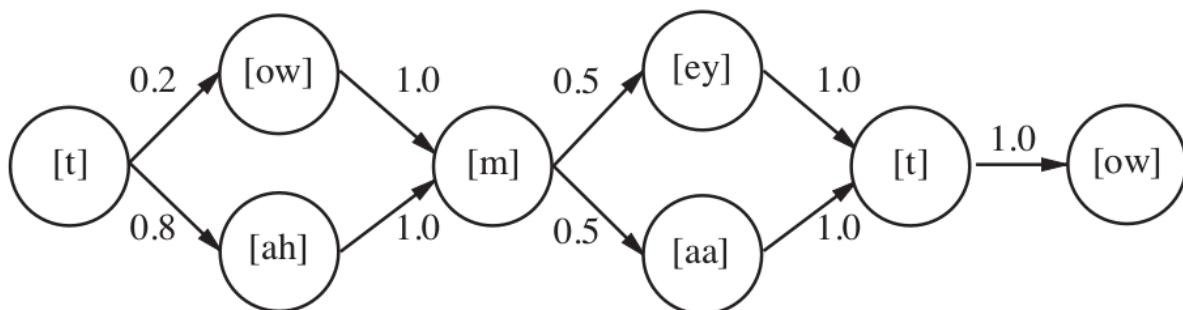
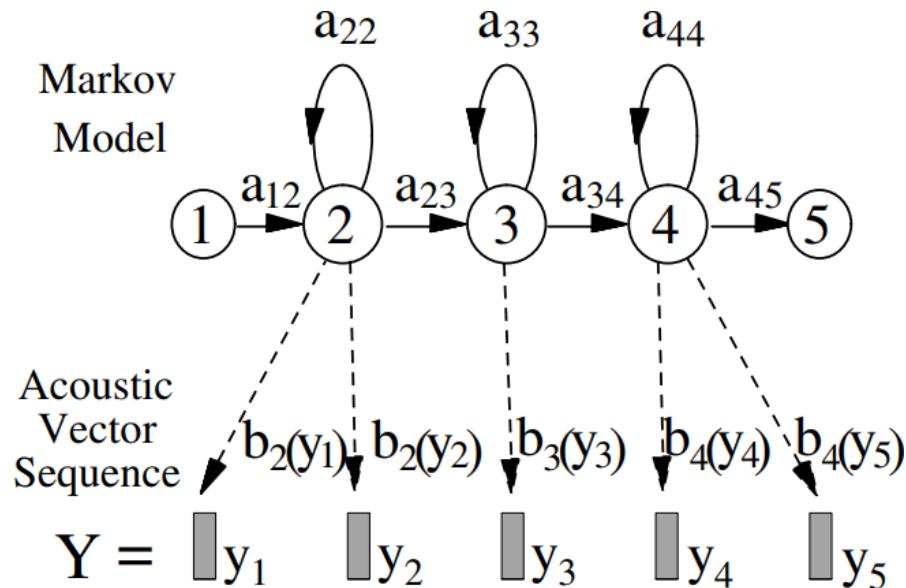


Figure 23.17 Two pronunciation models of the word “tomato.” Each model is shown as a transition diagram with states as circles and arrows showing allowed transitions with their associated probabilities. (a) A model allowing for dialect differences. The 0.5 numbers are estimates based on the two authors’ preferred pronunciations. (b) A model with a coarticulation effect on the first vowel, allowing either the [ow] or the [ah] phone.



Each base phone q is represented by **phone model** defined as a three-state continuous density HMM, where

- the transition probability parameter a_{ij} corresponds to the probability of making the particular transition from state s_i to s_j ;
- the output sensor models are Gaussians $b_j(\mathbf{y}) = \mathcal{N}(\mathbf{y}; \mu^{(j)}, \Sigma^{(j)})$ and relate state variables s_j to MFCCs \mathbf{y} .

The full acoustic model can now be defined as a composition of pronunciation models with individual phone models:

$$p(\mathbf{y}_{1:T} | \mathbf{w}_{1:L}) = \sum_{\mathbf{Q}} P(\mathbf{y}_{1:T} | \mathbf{Q}) P(\mathbf{Q} | \mathbf{w}_{1:L})$$

where the summation is over all valid pronunciation sequences for $\mathbf{w}_{1:L}$, \mathbf{Q} is a particular sequence $\mathbf{q}^{w_1}, \dots, \mathbf{q}^{w_L}$ of pronunciations,

$$P(\mathbf{Q} | \mathbf{w}_{1:L}) = \prod_{l=1}^L P(\mathbf{q}^{w_l} | w_l)$$

as given by the pronunciation model, and where \mathbf{q}^{w_l} is a valid pronunciation for word w_l .

Given the composite HMM formed by concatenating all the constituent pronunciations $\mathbf{q}^{w_1}, \dots, \mathbf{q}^{w_L}$ and their corresponding base phones, the acoustic likelihood is given by

$$p(\mathbf{y}_{1:T} | \mathbf{Q}) = \sum_{\mathbf{s}} p(\mathbf{s}, \mathbf{y}_{1:T} | \mathbf{Q})$$

where $\mathbf{s} = s_0, \dots, s_{T+1}$ is a state sequence through the composite model and

$$p(\mathbf{s}, \mathbf{y}_{1:T} | \mathbf{Q}) = a_{s_0, s_1} \prod_{t=1}^T b_{s_t}(\mathbf{y}_t) a_{s_t s_{t+1}}.$$

From this formulation, all model parameters can be efficiently estimated from a corpus of training utterances with expectation-maximization.

N-gram language model

The prior probability of a word sequence $\mathbf{w} = w_1, \dots, w_L$ is given by

$$P(\mathbf{w}) = \prod_{l=1}^L P(w_l | w_{l-1}, \dots, w_{l-N+1}).$$

The N-gram probabilities are estimated from training texts by counting N-gram occurrences to form maximum likelihood estimates.

Decoding

The composite model corresponds to a HMM, from which the most-likely state sequence **S** can be inferred using (a variant of) **Viterbi**.

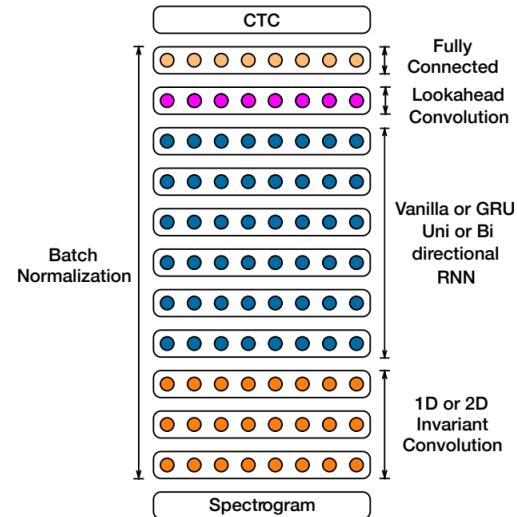
By construction, states **S** relate to phones, phones to pronunciations, and pronunciations to words.

Neural speech recognition

Modern speech recognition systems are now based on [end-to-end](#) deep neural network architectures trained on large corpus of data.

Deep Speech 2

- Recurrent neural network with
 - one or more convolutional input layers,
 - followed by multiple recurrent layers,
 - and one fully connected layer before a softmax layer.
- Total of 35M parameters.
- Same architecture for both English and Mandarin.





Deep Speech 2

Text-to-speech synthesis

$\mathbf{w}_{1:L}$

My name is HAL.



$\mathbf{y}_{1:T}$



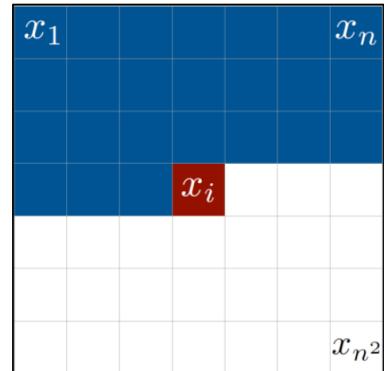
Autoregressive models

By the chain rule, any joint distribution can always be written as an incremental product of conditional distributions:

$$p(x_1, \dots, x_n) = \prod_{k=1}^n p(x_k | x_1, \dots, x_{k-1}).$$

If h_{k-1} is a lossless statistic of the previous observations in the sequence, then we can express

$$p(x_k | x_1, \dots, x_{k-1}) = g_\theta(x_k | h_{k-1}).$$



Autoregressive model
for images.

An autoregressive model can be formulated as a recurrent neural network such that

$$\begin{aligned} p(x_k | x_1, \dots, x_{k-1}) &= g_\theta(x_k | h_{k-1}) \\ h_k &= f_\theta(x_k, h_{k-1}) \end{aligned}$$

where

- g specifies a probability density function for the next x given h .
- f specifies (in a deterministic way) the next state h given x .

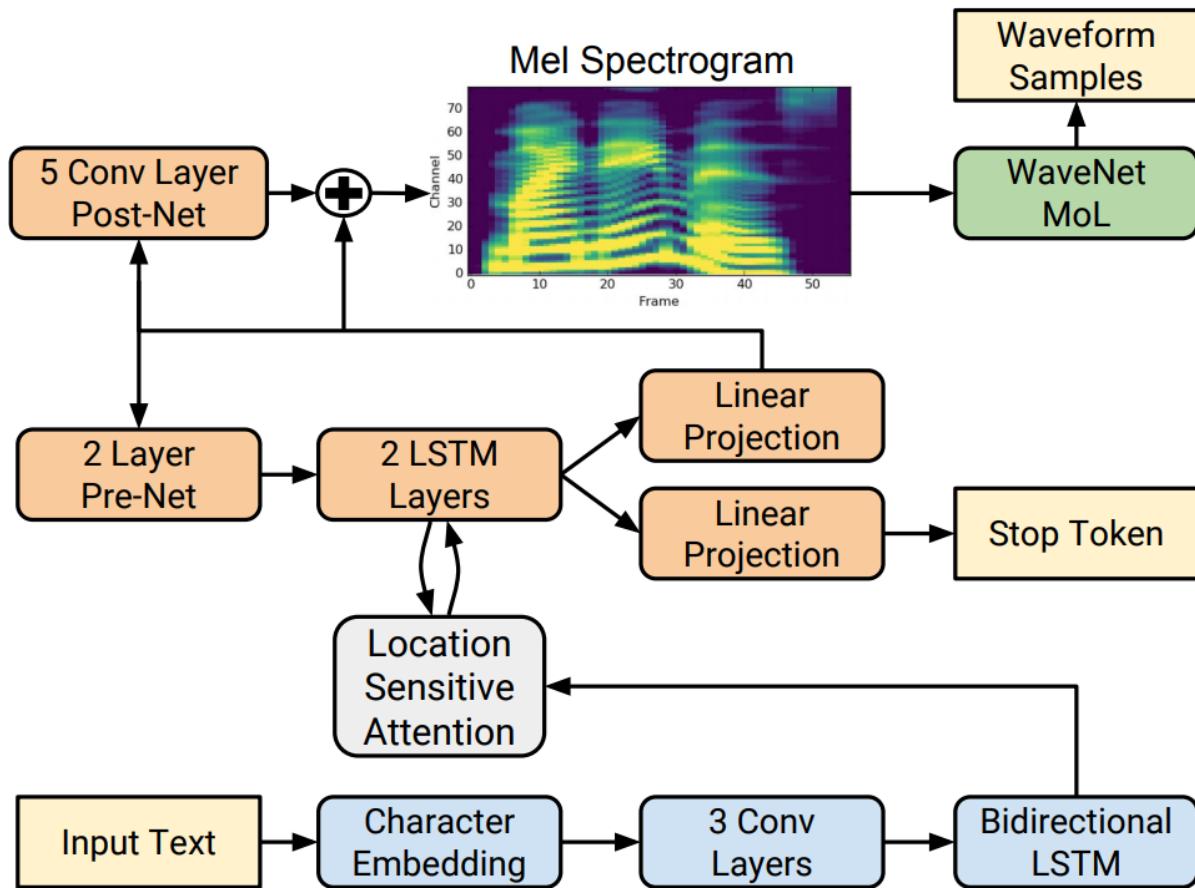
If g and f have enough capacity (e.g., f and g are large neural networks), then the resulting autoregressive model is often capable of modeling complex distributions, such as text, images or speech.

The same architecture can be used for modeling **conditional** distributions over inputs.

Tacotron 2

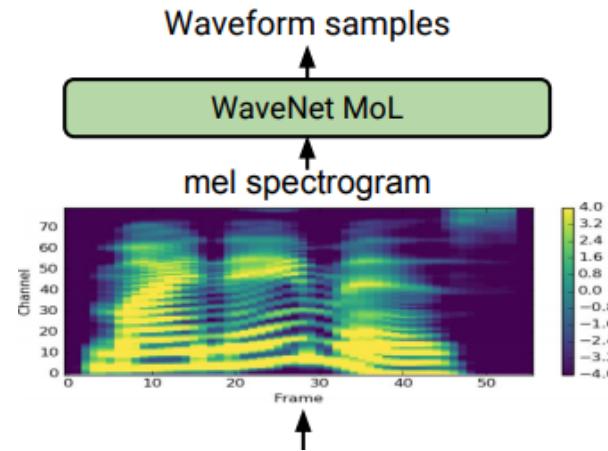
The Tacotron 2 system is a **sequence-to-sequence neural network** architecture for text-to-speech. It consists of two components:

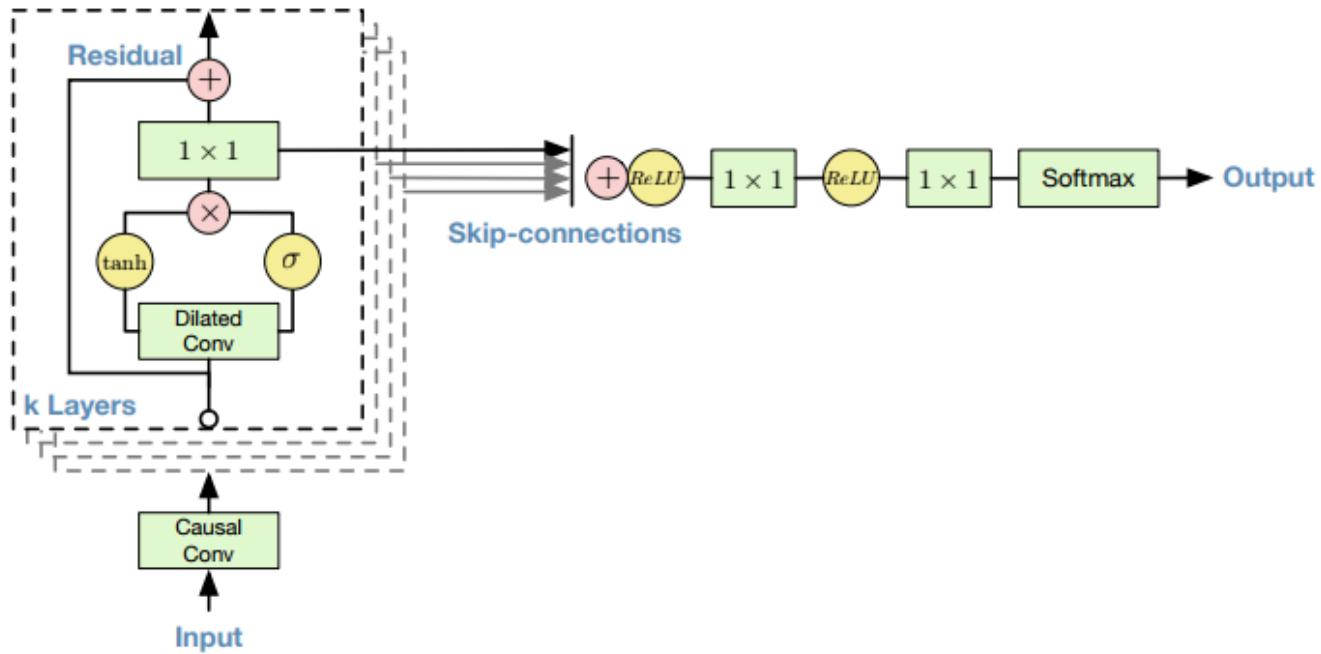
- a recurrent sequence-to-sequence feature prediction network with attention which predicts a sequence of mel spectrogram frames from an input character sequence;
- a **Wavenet vocoder** which generates time-domain waveform samples conditioned on the predicted mel spectrogram frames.



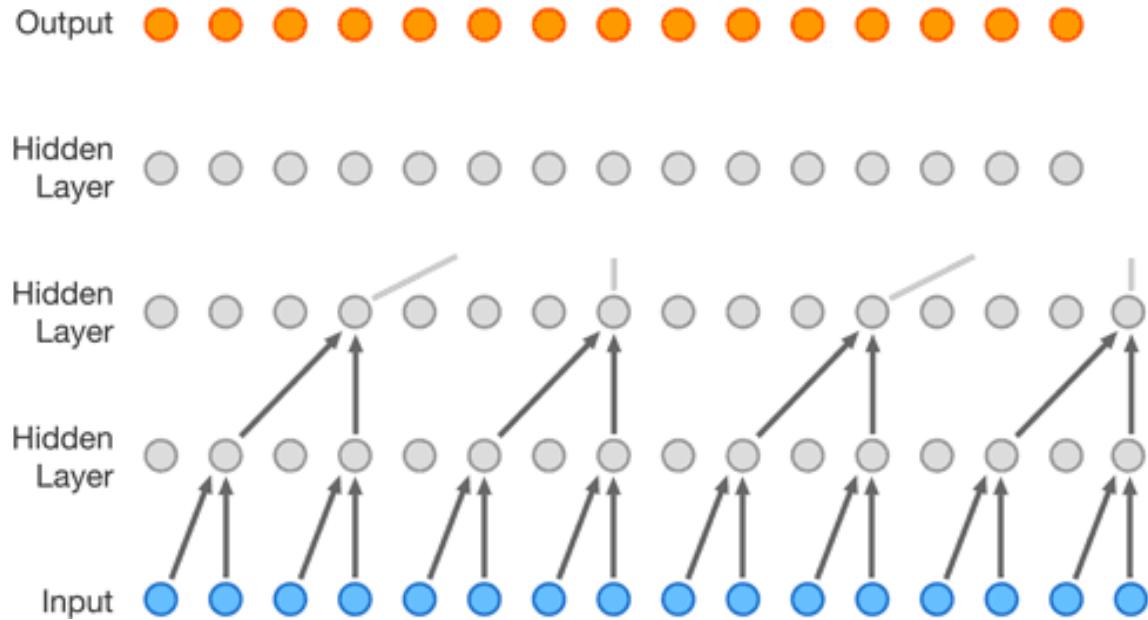
Wavenet

- The Tacotron 2 architecture produces mel spectrograms as outputs, which remain to be synthesized as waveforms.
- This last step can be performed through another autoregressive neural model, such as **Wavenet**, to transform mel-scale spectrograms into high-fidelity waveforms.





The Wavenet architecture.



Dilated convolutions.

Audio samples at

- deepmind.com/blog/wavenet-generative-model-raw-audio
- google.github.io/tacotron



Google Assistant: Soon in your smartphone.

Summary

- Natural language understanding is one of the most important subfields of AI.
- Machine translation, speech recognition and text-to-speech synthesis are instances of sequence-to-sequence problems.
- All problems can be tackled with traditional statistical inference methods but require sophisticated engineering.
- State-of-the-art methods are now based on neural networks.

References

- Gales, M., & Young, S. (2008). The application of hidden Markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3), 195-304.

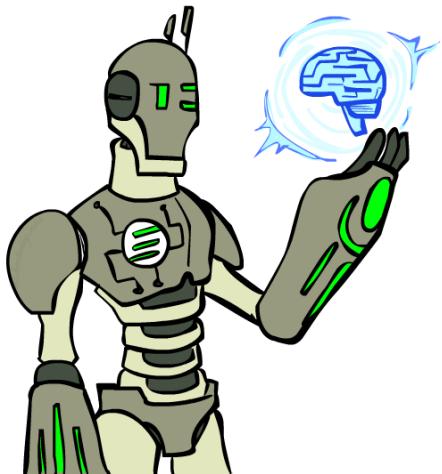
Introduction to Artificial Intelligence

Lecture 11: Artificial General Intelligence

Prof. Gilles Louppe
g.louppé@uliege.be



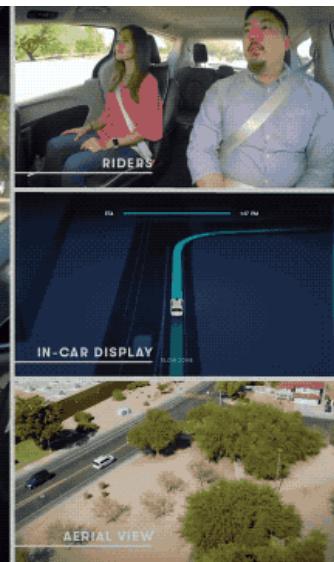
Today*



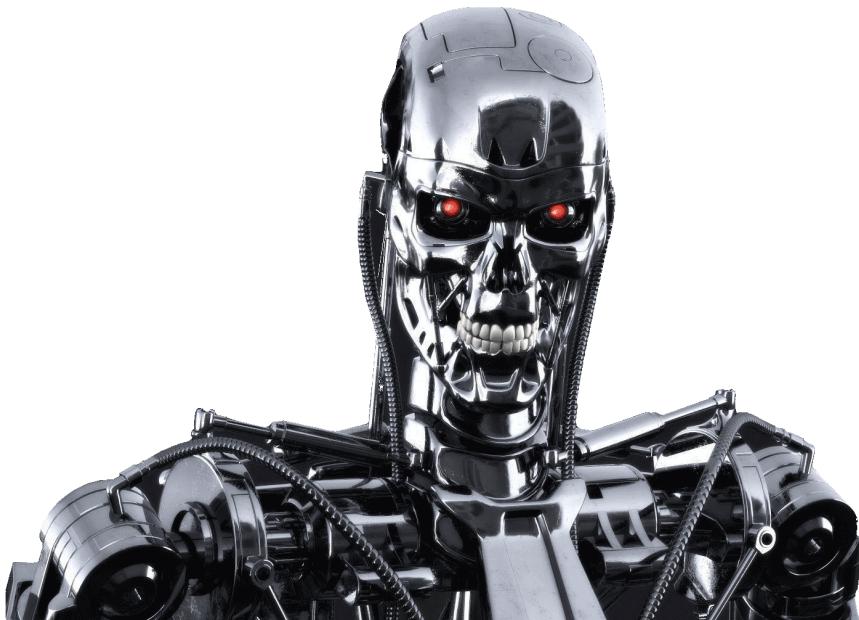
Towards **generally intelligent** agents?

- Artificial general intelligence
- AIXI
- Artificial life

*: Take today's lecture with a grain of salt. Image credits: CS188, UC Berkeley.



From technological breakthroughs...



... to press coverage.

The screenshot shows the SingularityHub homepage with a dark background and a central image of a brain-like network. Below the image is the article title 'Will Artificial Intelligence Become Conscious?' followed by a short summary and a 'SIGN ME UP' button.

The Express website header includes categories like NEWS, WORLD, SHOWBIZ & TV, SPORT, COMMENT, FINANCE, TRAVEL, ENTERTAINMENT, and LIFE & STYLE. A news article is visible with the headline 'Rise of the machines: Super intelligent robots could 'spell the end of the human race'.'

A video player on the left shows a thumbnail of Stephen Hawking. To the right is a sidebar titled 'Latest videos' with several thumbnail images and titles, including 'Stephen Hawking' and 'Elon Musk'sizarre'.

The Independent article features a large image of a robotic hand holding a gun. The headline reads: 'KILLER ROBOTS' WILL START SLAUGHTERING PEOPLE IF THEY'RE NOT BANNED SOON, AI EXPERT WARNS'. Below the headline is a quote: 'These will be weapons of mass destruction'.

The Le Journal website has a search bar at the top. Below it, a video player shows a man speaking, with the caption: 'Si nous ne faisons rien, l'intelligence artificielle nous écrabouillera dans 30 ans'.



Artificial narrow intelligence

Artificial intelligence today remains **narrow**:

- Modern AI systems often reach super-human level performance.
- ... but only at **very specific problems!**
- They **do not generalize** to the real world nor to arbitrary tasks.

The case of AlphaGo

Convenient properties of the game of Go:

- Deterministic (no noise in the game).
- Fully observed (each player has complete information)
- Discrete action space (finite number of actions possible)
- Perfect simulator (the effect of any action is known exactly)
- Short episodes (200 actions per game)
- Clear and fast evaluation (as stated by Go rules)
- Huge dataset available (games)





Can we run AlphaGo on a robot for the Amazon Picking Challenge?

- Deterministic: Yes.
- Fully observed: **Almost.**
- Discrete action space: Yes
- Perfect simulator: **Nope! Not at all.**
- Short episodes: **Not really...**
- Clear and fast evaluation: Not good.
- Huge dataset available: **Nope.**

Artificial general intelligence

Artificial general intelligence (AGI) is the intelligence of a machine that could successfully perform any intellectual task that a human being can.

- No clear and definitive definition.
- Agreement that AGI is required to do the following:
 - reason, use strategy, solve puzzle, plan,
 - make judgments under uncertainty,
 - represent knowledge, including commonsense knowledge,
 - improve and learn new skills,
 - communicate in natural language,
 - integrate all these skills towards common goals.
- This is similar to our definition of thinking rationally, but applied broadly to any set of tasks.



Irving John Good (1965)

Singularity

- Let an **ultraintelligent** machine be defined as a machine that can far surpass all the intellectual activities of any man however clever.
- Since the design of machines is one of these intellectual activities, an ultraintelligent machine could **design even better machines**.
- There would then unquestionably be an **intelligence explosion**, and the intelligence of man would be left far behind.
- Thus the first ultraintelligent machine is the **last invention** that man need ever make, provided that the machine is docile enough to tell us how to keep it under control.



What happens when our computers get smarter than we are? (Nick Bostrom)

Roads towards Artificial General Intelligence

Several working **hypothesis**:

- Supervised learning
- Unsupervised learning
- AIXI
- Artificial life
- Brain simulation

Or maybe (certainly) something else?

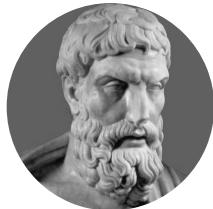
AIXI

In which environment is the agent?

- In general, we do not know!
- Solution:
 - maintain a prior over environments,
 - update it as evidence is collected,
 - follow the Bayes-optimal solution.



Occam: Prefer the simplest consistent hypothesis.



Epicurus: Keep all consistent hypotheses.



$$\text{Bayes: } P(h|d) = \frac{P(d|h)P(h)}{P(d)}$$



Turing: It is possible to invent a single machine which can be used to compute any computable sequence.

Solomonoff induction

- Use computer programs μ as hypotheses/environments.
- Make a weighted prediction based on all consistent programs, with short programs weighted higher.



$$\Upsilon(\pi) := \sum_{\mu \in E} 2^{-K(\mu)} V_\mu^\pi$$

- $\Upsilon(\pi)$ formally defines the **universal intelligence** of an agent π .
- μ is the environment of the agent and E is the set of all computable reward bounded environments.
- $V_\mu^\pi = \mathbb{E}[\sum_{i=1}^{\infty} R_i]$ is the expected sum of future rewards when the agent π interacts with environment μ .
- $K(\cdot)$ is the Kolmogorov complexity, such that $2^{-K(\mu)}$ weights the agent's performance in each environment, inversely proportional to its complexity.
 - Intuitively, $K(\mu)$ measures the complexity of the shortest Universal Turing Machine program that describes the environment μ .

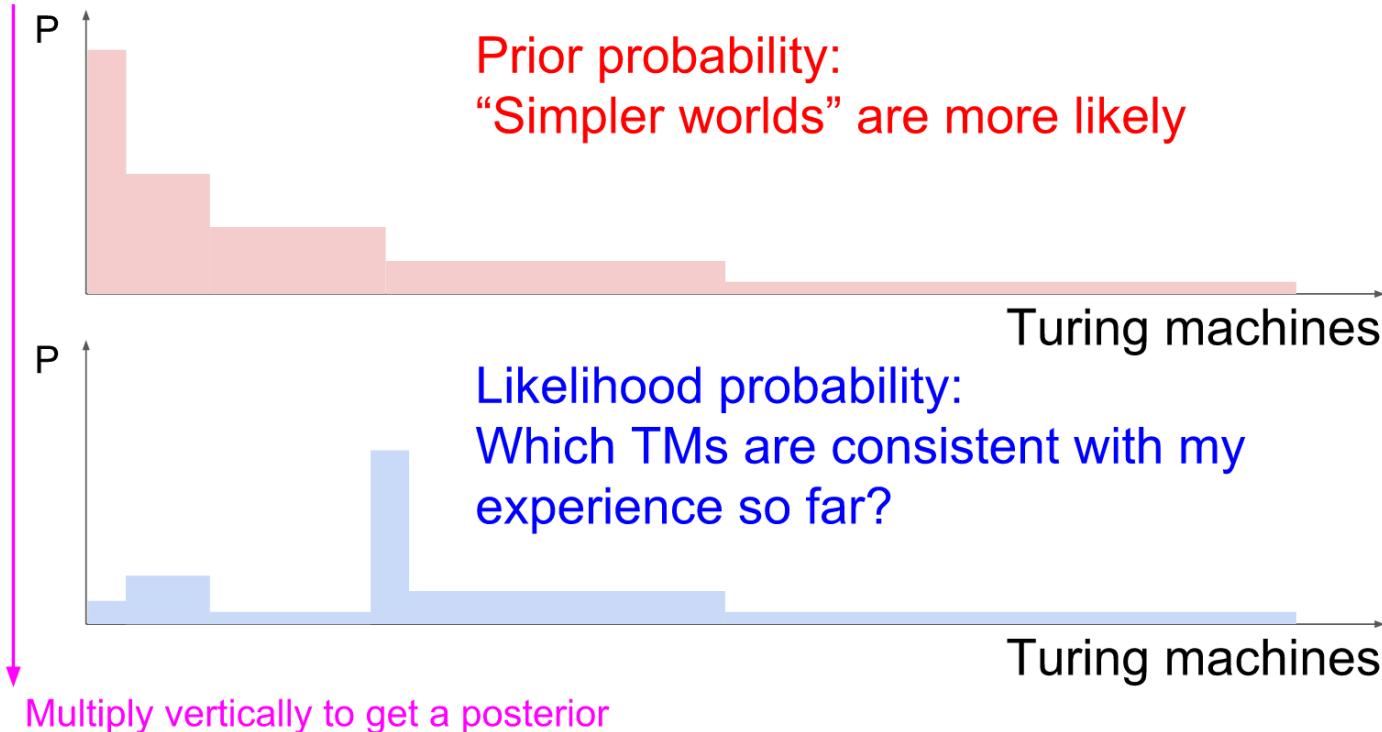
AIXI

$$\bar{\Upsilon} = \max_{\pi} \Upsilon(\pi) = \Upsilon(\pi^{AIXI})$$

π^{AIXI} is a **perfect** theoretical agent.

System identification

- Which Turing machine is the agent in? If it knew, it could plan perfectly.
- Use the [Bayes rule](#) to update the agent beliefs given its experience so far.



Acting optimally

- The agent always picks the action which has the greatest expected reward.
- For every environment $\mu \in E$, the agent must:
 - Take into account how likely it is that it is facing μ given the interaction history so far, and the prior probability of μ .
 - Consider all possible future interactions that might occur, assuming optimal future actions.
 - Evaluate how likely they are.
 - Then select the action that maximizes the expected future reward.

$$a_t^{\pi^\xi} := \arg \max_{a_t} \lim_{m \rightarrow \infty} \sum_{x_t} \max_{a_{t+1}} \sum_{x_{t+1}} \cdots \max_{a_m} \sum_{x_m} [\gamma_t r_t + \cdots + \gamma_m r_m] \xi(ax_{<t} ax_{t:m})$$

$$\xi(ax_{1:n}) := \sum_{\nu \in E} 2^{-K(\nu)} \nu(ax_{1:n})$$

(description length of the TM, number of bits)

Complete history of interactions up to this point

\bullet $ax_{<t}$

time t

all possible future action-state sequences

time m

Weighted average of the total discounted reward, across all possible Turing Machines.

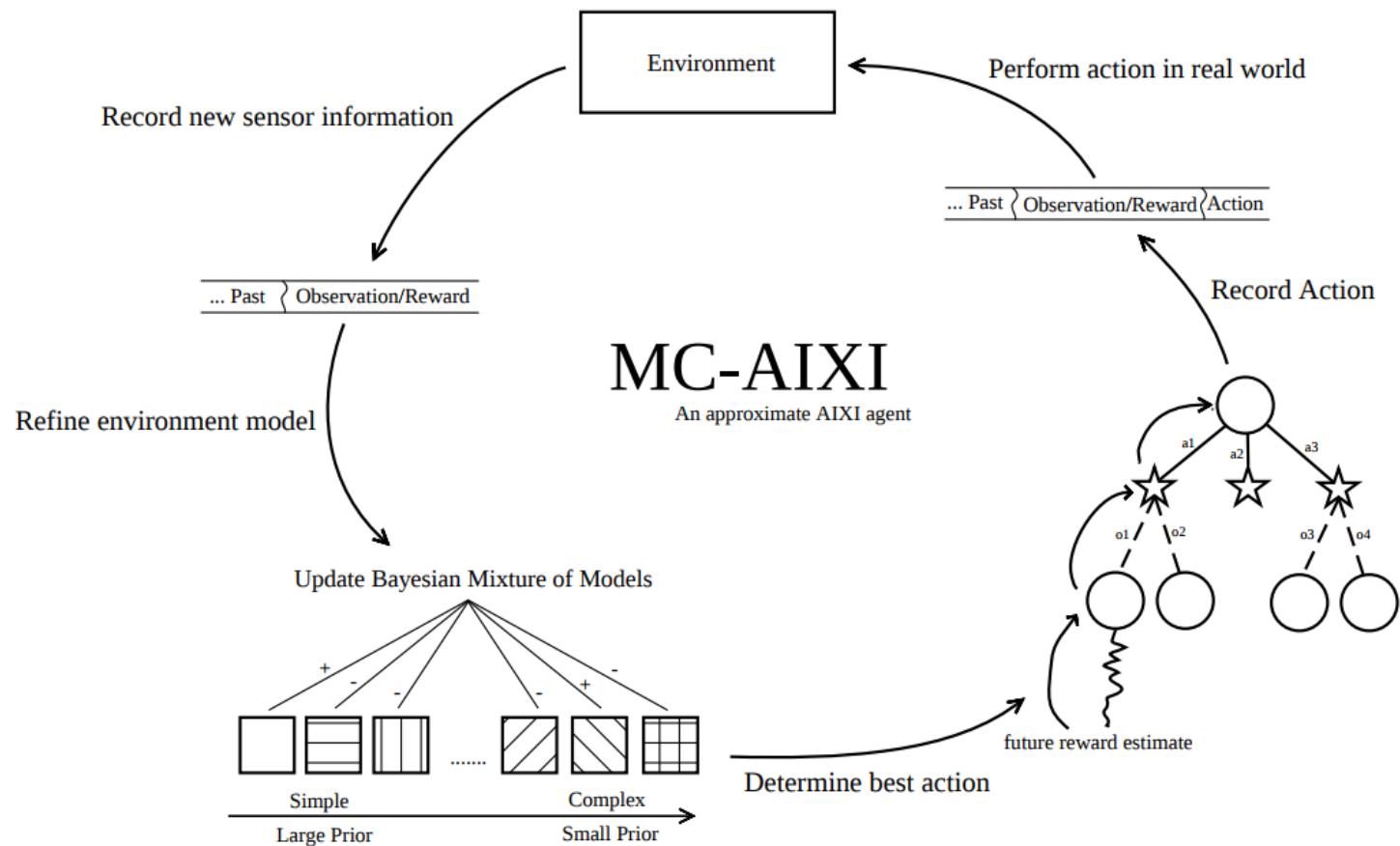
The weights are [prior] x [likelihood] for each Turing machine.

Incomputability

$$a_t^{\pi^\xi} := \arg \max_{a_t} \lim_{m \rightarrow \infty} \left[\sum_{x_t} \max_{a_{t+1}} \sum_{x_{t+1}} \cdots \max_{a_m} \sum_{x_m} [\gamma_t r_t + \dots + \gamma_m r_m] \xi(\underline{ax}_{<t} \underline{ax}_{t:m}) \right]$$

$$\xi(\underline{ax}_{1:n}) := \sum_{\nu \in E} 2^{-K(\nu)} \nu(\underline{ax}_{1:n})$$

Monte Carlo approximation

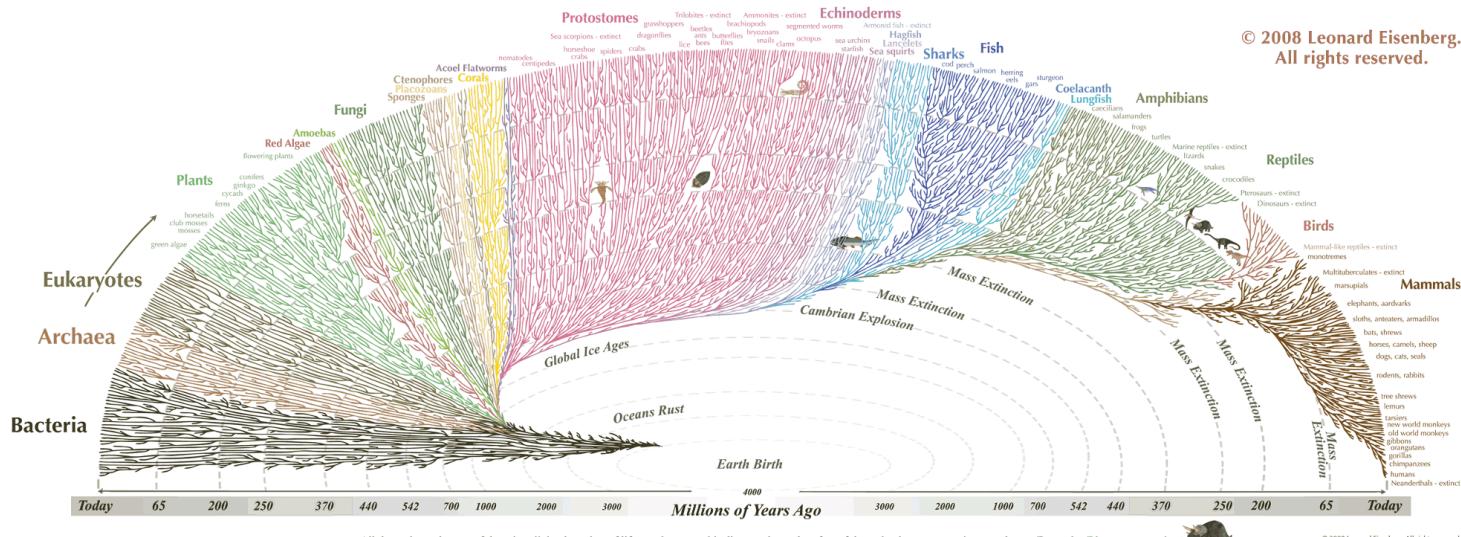


Benefits of a foundational theory of AI

AIXI provides

- high-level **blue-print** or inspiration for design;
- common terminology and goal formulation;
- understand and predict behavior of yet-to-be-built agents;
- appreciation of **fundamental challenges** (e.g., exploration-exploitation);
- **definition/measure** of intelligence.

Artificial life



How did intelligence arise in Nature?

Artificial life

- Artificial life is the study of systems related to natural life, its processes and its evolution, through the use of simulations with computer models, robotics or biochemistry.
- One of its goals is to synthesize life in order to understand its origins, development and organization.
- There are three main kinds of artificial life, named after their approaches:
 - Software approaches (soft)
 - Hardware approaches (hard)
 - Biochemistry approaches (wet)
- Artificial life is related to AI since synthesizing complex life forms would, hypothetically, induce intelligence.
- The field of AI has traditionally used a top down approach. Artificial life generally works from the bottom up.



Wet artificial life: The line between life and not-life (Martin Hanczyc).

Evolution

Evolution may **hypothetically** be interpreted as an (unknown) algorithm.

- This algorithm gave rise to AGI (e.g., it induced humans).
- Can we **simulate** the **evolutionary process** to reproduce life and intelligence?
- Using software simulation, we can work at a high level of abstraction.
 - We don't have to simulate physics or chemistry to simulate evolution.
 - We can also bootstrap the system with agents that are better than random.

Evolutionary algorithms

- Start with a random population of creatures.
- Each creature is tested for their ability to perform a given task.
 - e.g., swim in a simulated environment.
 - e.g., stay alive as long as possible (without starving or being killed).
- The most successful survive.
- Their virtual genes containing coded instructions for their growth are copied, combined and mutated to make offspring for a new population.
- The new creatures are tested again, some of which may be improvements on their parents.
- As this cycle of variation and selection continues, creatures with more and more successful behaviors may emerge.

C



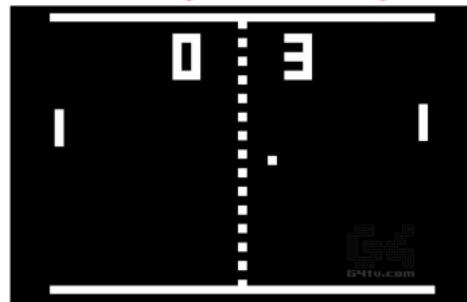


Neurevolution [demo](#).

Environments for AGI?

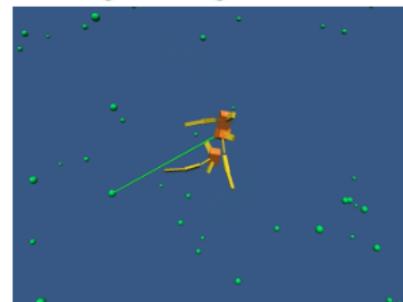
For the emergence of generally intelligent creatures, we presumably need environments that **incentivize** the emergence of a **cognitive toolkit** (attention, memory, knowledge representation, reasoning, emotions, forward simulation, skill acquisition, ...).

Doing it wrong:



Incentives a lookup table of correct moves.

Doing it right:



Incentivises cognitive tools.

Multi-agent environments are certainly better because of:

- Variety: the environment is parameterized by its agent population. The optimal strategy must be derived dynamically.
- Natural curriculum: the difficulty of the environment is determined by the skill of the other agents.

Conclusions

Roads towards AGI

In order of (subjective) promisingness

- Artificial life
- Something not on our radar
- Supervised learning
- Unsupervised learning
- AIXI
- Brain simulation

What do you think?



A note of optimism: Don't fear intelligent machines,
work with them (Garry Kasparov).

Outline

- Lecture 1: Foundations
- Lecture 2: Solving problems by searching
- Lecture 3: Constraint satisfaction problems
- Lecture 4: Adversarial search
- Lecture 5: Representing uncertain knowledge
- Lecture 6: Inference in Bayesian networks
- Lecture 7: Reasoning over time
- Lecture 8: Making decisions
- Lecture 9: Learning
- Lecture 10: Communication
- Lecture 11: Artificial General Intelligence and beyond

Going further

This course is designed as an introduction to the many other courses available at ULiège and related to AI, including:

- ELEN0062: Introduction to Machine Learning
- INFO8004: Advanced Machine Learning
- INFO8010: Deep Learning
- INFO8003: Optimal decision making for complex problems
- INFO0948: Introduction to Intelligent Robotics
- INFO0049: Knowledge representation
- ELEN0016: Computer vision
- DROI8031: Introduction to the law of robots

Research opportunities

Feel free to contact us

- for research Summer internship opportunities (locally or abroad),
- MSc thesis opportunities,
- PhD thesis opportunities.



Thanks for following Introduction to Artificial Intelligence!

References

- Bostrom, Nick. *Superintelligence*. Dunod, 2017.
- Legg, Shane, and Marcus Hutter. "Universal intelligence: A definition of machine intelligence." *Minds and Machines* 17.4 (2007): 391-444.
- Hutter, Marcus. "One decade of universal artificial intelligence." *Theoretical foundations of artificial general intelligence* (2012): 67-88.
- Sims, Karl. "Evolving 3D morphology and behavior by competition." *Artificial life* 1.4 (1994): 353-372.
- Kasparov, Garry. *Deep Thinking: Where Machine Intelligence Ends and Human Creativity Begins*, 2017.