Foundations of Artificial Intelligence

1. Introduction

Organizational Aspects, AI in Freiburg,
Motivation, History, Approaches, and Examples

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

May 11, 2020

Lecture Overview

1. Introduction

Organizational

Lecturers:

- Prof. Dr. Joschka Boedecker (jboedeck@informatik.uni-freiburg.de)
- Prof. Dr. Wolfram Burgard (burgard@informatik.uni-freiburg.de)
- Prof. Dr. Frank Hutter (fh@informatik.uni-freiburg.de)
- Prof. Dr. Bernhard Nebel (nebel@informatik.uni-freiburg.de)
- Prof. Dr. Michael Tangermann (michael.tangermann@blbt.uni-freiburg.de)

Organizers:

- Tim Schulte (schultet@informatik.uni-freiburg.de)
- Michael Krawez (krawez@informatik.uni-freiburg.de)

Times and dates

- Every friday (starting May 8) the week's lecture content is published
- Every monday (starting May 11) there is a new exercise sheet
- Every friday at 10:00 a.m. there is a live Q&A session via Zoom

Organizational: Exam

Credit Requirements:

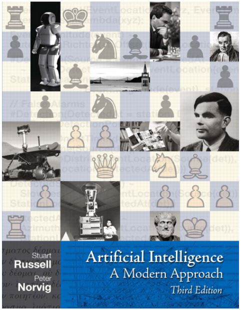
- Written exam (see web page)
- For questions about the exam, please do not email the lecturers, but the organizers
  - Hint: professors receive > 100 emails a day, organizers much less
- Early exam for Erasmus students?
- Question: who needs an early exam?

Lecture Material

Lectures are based on

Artificial Intelligence:
A Modern Approach, Third Edition

Stuart Russell and Peter Norvig



Copies of the slides, online recordings and further information can be found on the Web page at

`https://ilias.uni-freiburg.de/goto.php?target=crs_1500586&client_id=unifreiburg`

Many illustrations are taken from the above-mentioned book. Some slides are based on presentations created by Prof. Gerhard Lakemeyer, Univ. Aachen. Several sections were originally prepared by Dr. Jana Köhler.

Course Contents

Strongly method-oriented

1. Introduction
2. Rational Agents
3. Solving Problems by Searching
4. Informed Search
5. Constraint Satisfaction Problems
6. Games
7. Propositional Logic
8. Satisfiability and Model Construction
9. Predicate Logic
11. Planning
12. Simple Probabilistic Reasoning
13. Machine Learning
14. Acting under Uncertainty

AI in Freiburg

Foundations of Artificial Intelligence
Bernhard Nebel

Autonomous Intelligent Systems
Wolfram Burgard

Machine Learning
Frank Hutter

AI in Freiburg

Neurorobotics
Joschka Boedecker

Representation Learning
Josif Grabocka

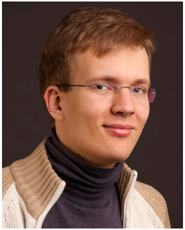
Robot Learning
Abhinav Valada

Junior Research Groups

Cognitive Modeling
Marco Ragni

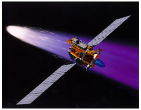
Brain-Computer Interfaces
Michael Tangermann

Senior Staff Members (Among Further PostDocs)



Marius Lindauer
(Algorithm Design)

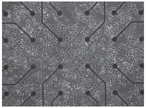
- Action planning: theory and practice
  - Fast planning systems (international competitions)
  - Applications for elevator systems and space
  - Theoretical results
  - Projects: CRC AVACS, Karis Pro (intra-logistics), Kontiplan, Hybris, SPAICER
- Qualitative temporal-spatial reasoning
  - Theory and reasoning algorithms
  - Application in qualitative layout description (CRC "Spatial Cognition")
- RoboCup
  - World champion three times
  - Autonomous table soccer
  - RoboCup rescue (multi-agent-systems for disaster relief)
  - Cognitive robotics (BrainLinks-BrainTools)
- Recently: Multi-agent pathfinding

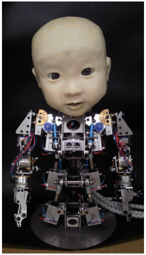


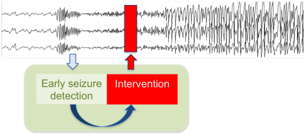
Neurorobotics

- Data-efficient Reinforcement Learning
- Representation Learning: Deep Neural Networks (in space and time)
- Learning control of complex dynamical systems like robots or real neuronal networks (BrainLinks-BrainTools)
- Interpretation of brain signals (BrainLinks-BrainTools)
- Industrial applications







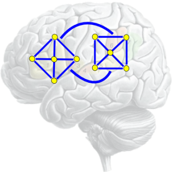



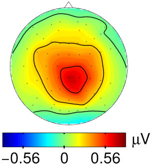
Autonomous Intelligent Systems

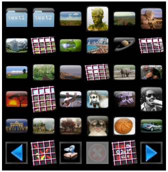
- Mobile robots
- Probabilistic approaches for state estimation and control
- Automated driving
- Adaptive techniques and learning
- Multi-robot systems
- Applications of mobile robots
- Cognitive Robotics
- Service robots (BrainLinks-BrainTools)

Brain State Decoding Lab

- Brain-computer interfaces
- Decoding of brain signals
- Supervised learning
- Adaptive unsupervised classification
- Learning in non-stationary environments
- Supervised subspace decompositions
- Mental workload assessment e.g. of drivers
- Predicting user performance
- BCI-supported rehabilitation



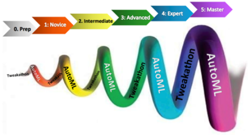


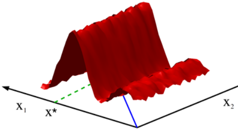


BrainLinks
BrainTools
acting.thoughts
University of Freiburg

Automated Algorithm Design

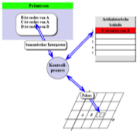
- Machine Learning (ML)
  - AutoML: automatically selecting features, ML algorithm & hyperparameters, ensembles
    $\rightsquigarrow$ Currently AutoML world champions
  - Bayesian optimization
  - Meta-Learning: reasoning across datasets
  - Deep Learning: automatic feature engineering
  - Big Data: how to train efficiently?
- Programming by Optimization
  - Algorithm Configuration
  - Algorithm Portfolios
  - Algorithm Runtime Prediction
  - Automatic science: what makes instances hard?
  - Applications: world champions in SAT solving and AI planning competitions

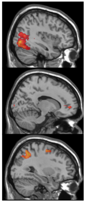
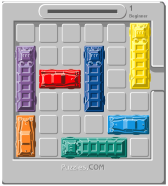


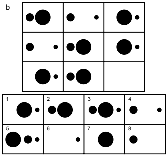


Cognitive Modeling Lab

- Cognitive models of human thinking, reasoning, and planning (BrainLinks-BrainTools)
- Qualitative Reasoning and imprecise knowledge
- Cognitive complexity analysis
- Behavioural and fMRI experiments
- Systems that solve IQ-test problems
- "Build systems that reason and plan like humans"





Studying AI in Freiburg

- Foundations of Artificial Intelligence
- Machine Learning and Data Mining
- Knowledge Representation
- Introduction to Mobile Robotics
- AI Planning
- Logic
- Game Theory
- Neurorobotics
- ...

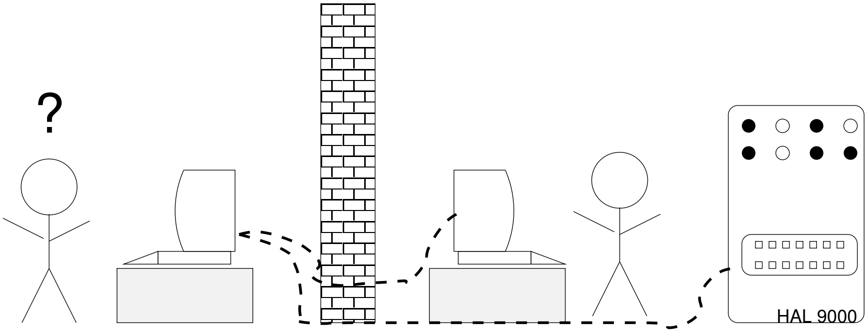
What is Artificial Intelligence?

- The attempt to make computers more “intelligent”
- The attempt to better understand human intelligence
- Four approaches:
  - Is it about thought thinking ...
  - ... or acting?
  - Oriented towards a human model (with all its defects) ...
  - ... or normative (how should a rational being think/act)?

A Few Definitions

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

The Turing Test



Systems that Act Humanly

Reproducing human behavior

→ Prototype: Turing Test

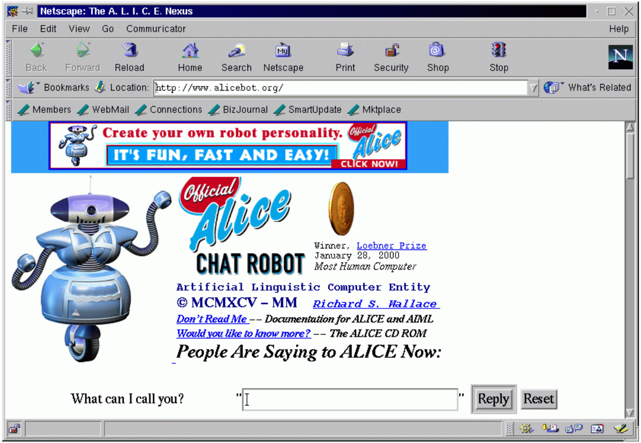
→ . . . in AI, scarcely pursued

→ . . . but yearly competitions exist, e.g., Loebner Prize:

http://www.loebner.net/Prizef/loebner-prize.html

Loebner Prize Winner Three Times

Richard Wallace (`http://www.alicebot.org`)



Web page of Alice in 2000

Example Session with Alice (1)

```
  What is your name?
> Wolfram
  Nice to meet you Wolfram.
> Do you know Clinton?
  He was the second Impeached President of the United States.
> Do you like him?
  Sure he has never done anything to me.
> What is an impeachment?
  It is a good question. I have to process that one for a while.
  I would do a search for it.
...
> Who are you?
  I ask myself that question, "Who am I?"
> What are you doing?
  I am chatting with people on the net.
> What is your favorite topic?
  My favorite subject is artificial intelligence.
> What is artificial intelligence?
  Artificial intelligence is the branch of engineering and science devoted
  to constructing machines that think. I can ask someone about it.
> Can you do anything else than chatting?
  Doing anything else than chatting is not among my strongest traits.
...
```

Example Session with Alice (2)

```
...

> How old am I?
  Is this a trick question? You said you were how many years old?
  My guess is that you are really a kid.
> What would you like to know about me?
  Tell me about your educational background.
> I am a professor in computer science.
  You are a professor in computer science? How do you like your work?
> It is fun teaching courses on artificial intelligence.
  I think it is a lot of fun.
> What should I tell the students about you?
  Be sure to think before you speak. State your point clearly and slowly and
  gauge the listener's response before going any further.
```

Example Session with God (1)

```
Human: Do you know Obama?
God: I don’t know if I know it. Where would I have heard about it?
Human: He is the current president of the United States.
God: I did not know is it.
Human: Do you know Clinton?
God: He was the second Impeached President of the United States.
Human: What is an impeachment?
God: Charge against a public official of improper conduct.
...
Human: Who are you?
God: I am the Allmighty, God himself.
Human: What are you doing?
God: I am just killing time.
Human: What is your favorite topic?
God: My favorite subject is artificial intelligence.
Human: What is artificial Intelligence?
God: Artificial intelligence is the branch of engineering and
science devoted to constructing machines that think. That’s not
something I get asked all the time.
Human: Can you do anything else than chatting?
God: Doing anything else than chatting is not among my strongest
traits.
```

Example Session with God (2)

```
Human: My name is Wolfram.
God: OK I will call you Wolfram.
Human: Who am I?
God: You are Wolfram, seeker.
Human: How old am I?
God: Is this a trick question? You said you were years old? My
guess is that you are really a kid.
Human: What would you like to know about me?
God: Tell me about your educational background.
Human: I am a professor in Computer Science
God: What is your field?
Human: Artificial Intelligence
God: Do you mind if I tell other people you are teaching Artificial
Intelligence.
Human: No.
God: OK then I will add it to my gossip files.
Human: What should I tell the students about you?
God: I believe it is always best to say positive things to build
people up rather than negative things to bring people down.
```

Systems that Think Humanly

What cognitive capabilities are necessary to produce intelligent performance?

- Not important: Being able to solve problems correctly
- Important: Being able to solve problems like a human would

→ Cognitive science and cognitive psychology

→ Also important for human-machine interaction

- . . . will not be discussed in this course

Systems that Think Rationally

- What are the laws of thought?
- How should we think?

→ The logical approach

→ Problems:
  - Presentation of problem descriptions using a formal notation
  - Computability

→ These are problems that appear regardless of the formalization method

Systems that Act Rationally

→ Rational agents (or rational actors)

- A rational agent acts so as to achieve its given goals, under the assumption that its impressions of the world and its convictions are correct
- Rational thinking is a prerequisite for rational acting, although it is not a necessary condition

→ What to do, for example, when we must make a decision faced with insufficient information?

The AI Scene

Fields of Application	Methods
• Systems that can understand and generate speech • Systems that can understand images • Robotics • Assistant systems	• Problem solving and searching • Knowledge representation and processing • Action planning • Machine learning • Handling uncertain knowledge • Neural networks

With interdisciplinary relationships to Mathematics, Philosophy, Psychology, (Computational) Linguistics, Biology, Engineering Sciences, ...

The Origins of AI

Since the beginning, Philosophy, Mathematics, Psychology, Linguistics, and Computer Science have all

- asked similar questions
- developed methods and produced results for AI

The origins of AI (1943–1956): With the development of the first computing systems, people began to wonder, "Can computers copy the human mind? (Turing Test)"

50 Years of AI (1)

1956: Dartmouth Workshop - McCarthy proposes the term, "Artificial Intelligence" - and early enthusiasm:

`It is not my aim to surprise or shock you - but the simplest way I can summarize is to say that there are now in the world machines that think, that learn and that create. Moreover, their ability to do these things is going to increase rapidly until -` *`in the visible future`* `- the range of problems they can handle will be coextensive with the range to which the human mind has been applied. [Simon, 1957]`

60's: "Intelligent Behavior" is shown in many demonstration systems for micro-worlds (Blocks world)

70's: Problems:

- Systems for micro-worlds prove not scalable → "real" applications
- "Intelligent behavior" requires much knowledge → knowledge-based systems

50 Years of AI (2)

80's: Commercial success of experimental systems (e.g. R1), intense research support (e.g. *Fifth generation computer systems project* in Japan), return to neural networks

End of the 80's: Expert systems prove less promising than imagined, (demystification of expert systems), end of the *Fifth generation computer systems project*, "*AI winter*"

90's: Inclusion of probabilistic methods, agent-oriented vision techniques, formalization of AI techniques and increased use of mathematics in the field

`...gentle revolutions have occurred in robotics, computer vision, machine learning (including neural networks), and knowledge representation. A better understanding of the problems and their complexity properties, combined with increased mathematical sophistication, has led to workable research agendas and robust methods. [Russell & Norvig, 1995]`

. . . and Today?

- Many coexisting paradigms
  → Reactive vs. deliberative approaches
  → Probabilistic vs. analytic
  → . . . often hybrid approaches as well
- Many methods (partly from other disciplines):
  → Logic, decision theory, algorithms
- Many approaches:
  → Theoretical, algorithmic experimentation, system-oriented
- Today, many methods are no longer regarded as pure AI methods. Examples: Board game programs, logic programming (PROLOG), search procedures, . . .
- Deep learning has become the new hype . . .

Examples: Algorithmic, Experimental Tasks

Many AI problems are inherently difficult (NP-hard), but it is possible, in spite of this and with the use of good search techniques and heuristics, to solve problem instances up to a certain size:

- Satisfiability of boolean formulas
  $\rightarrow$ Randomized, local search techniques (up to 2,500???? variables in complex instances)
- Constraint propagation and backtracking techniques
  $\rightarrow$ Empirical and analytical comparisons of various techniques
- Action planning
  $\rightarrow$ Empirical comparisons of various approaches and systems
- ...

Systems

Alongside theory and the analysis of individual algorithms, the building of systems and applications is a basic point:

Herb Simon in a lecture entitled "How to become a good scientist" (1998):

"Build a System"

- Application of AI techniques to solve real problems
- Study of the interaction of artefacts with their environment
- Synergetic effects in systems

Systems

- Computer Games
- Navigation Systems
- Smart phone services
- Intelligent email
- Search engines
- Recommender systems
- Self-driving cars
- ...

Foundations of Artificial Intelligence

2. Rational Agents

Nature and Structure of Rational Agents and Their Environments

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and
Michael Tangermann

Albert-Ludwigs-Universität Freiburg

April 26, 2019

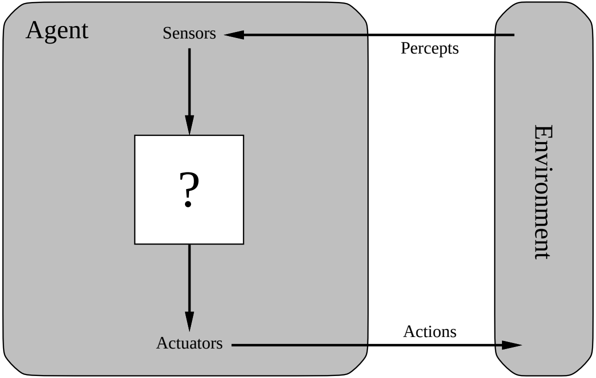
Contents

Lecture Overview

1. What is an agent?
2. What is a rational agent?
3. The structure of rational agents
4. Different classes of agents
5. Types of environments

Agents

- Perceive the environment through sensors ($\rightarrow$ Percepts)
- Act upon the environment through actuators ($\rightarrow$ Actions)



Examples: Humans and animals, robots and software agents (softbots), temperature control, ABS, ...

Lecture Overview

1. What is an agent?
2. What is a rational agent?
3. The structure of rational agents
4. Different classes of agents
5. Types of environments

Rational Agents

... do the "right thing"!

In order to evaluate their performance, we have to define a performance measure.

Autonomous vacuum cleaner example:

- Square meters per hour
- Level of cleanliness
- Energy usage
- Noise level
- Safety (behavior towards hamsters/small children)

Optimal behavior is often unattainable

- Not all relevant information is perceivable
- Complexity of the problem is too high

Rationality vs. Omniscience

- An *omniscient agent* knows the *actual effects* of its *actions*
- In comparison, a *rational agent* behaves according to its *percepts* and *knowledge* and attempts to *maximize the expected performance*
- Example: If I look both ways before crossing the street, and then as I cross I am hit by a meteorite, I can hardly be accused of lacking rationality.

The Ideal Rational Agent

Rational behavior is dependent on

- Performance measures (goals)
- Percept sequences
- Knowledge of the environment
- Possible actions

Ideal rational agent

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

Active perception is necessary to avoid trivialization.
The ideal rational agent acts according to the function

$$\text{Percept Sequence} \times \text{World Knowledge} \rightarrow \text{Action}$$

Examples of Rational Agents

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	healthy patient, costs, lawsuits	patient, hospital, stuff	display questions, tests, diagnoses, treatments, referrals	keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	correct image categorization	downlink from orbiting satellite	display categorization of scene	color pixel arrays
Part-picking robot	percentage of parts in correct bins	conveyor belt with parts, bins	jointed arm and hand	camera, joint angle sensors
Refinery controller	purity, yield, safety	refinery, operators	valves pumps, heaters displays	temperature, pressure, chemical sensors
Interactive English tutor	student's score on test	set of students, testing agency	display exercises, suggestions, corrections	keyboard entry

Lecture Overview

Structure of Rational Agents

Realization of the ideal mapping through an

- *Agent program*, executed on an
- *Architecture* which also provides an interface to the environment (percepts, actions)

$\rightarrow$ Agent = Architecture + Program

Lecture Overview

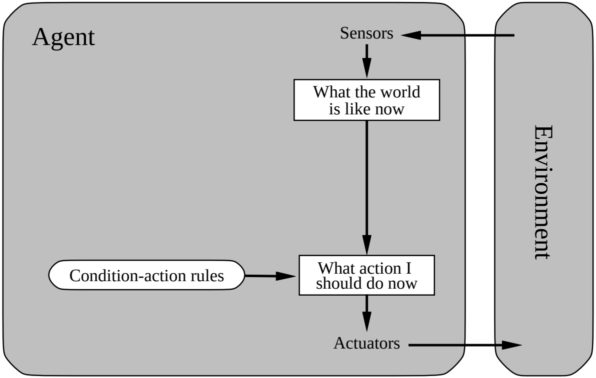
The Simplest Design: Table-Driven Agents

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
              table, a table of actions, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(percepts, table)
  return action
```

Problems:

- The table can become very large
- and it usually takes a very long time for the designer to specify it (or to learn it)
- ... practically impossible

Simple Reflex Agent



Direct use of perceptions is often not possible due to the large space required to store them (e.g., video images).

Input therefore is often interpreted before decisions are made.

Interpretative Reflex Agents

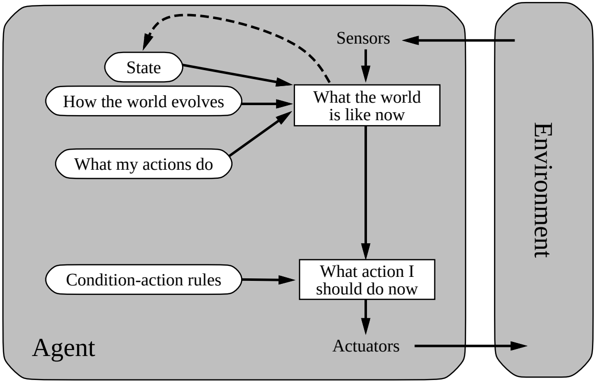
Since storage space required for perceptions is too large, direct interpretation of perceptions

function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *rules*, a set of condition–action rules

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

Structure of Model-based Reflex Agents

In case the agent's history in addition to the actual percept is required to decide on the next action, it must be represented in a suitable form.



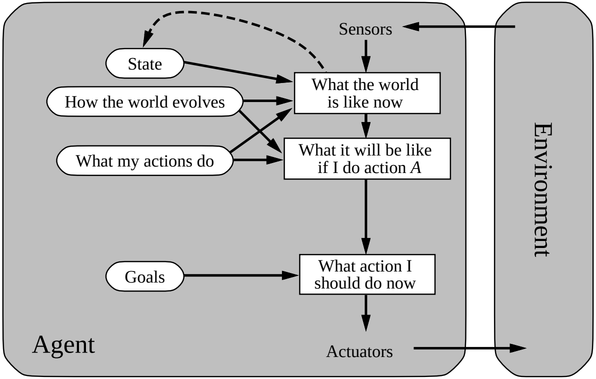
A Model-based Reflex Agent

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              rules, a set of condition–action rules
              action, the most recent action, initially none

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

Model-based, Goal-based Agents

- Often, percepts alone are insufficient to decide what to do.
- This is because the correct action depends on the given explicit goals (e.g., go towards X).
- The model-based, goal-based agents use an explicit representation of goals and consider them for the choice of actions.

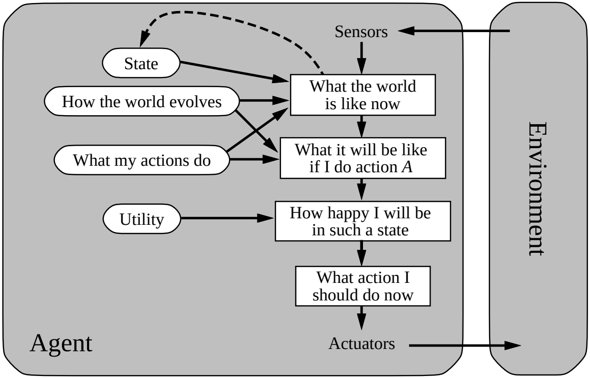
Model-based, Goal-based Agents



Model-based, Utility-based Agents

- Usually, there are several possible actions that can be taken in a given situation.
- In such cases, the utility of the next achieved state can come into consideration to arrive at a decision.
- A utility function maps a state (or a sequence of states) onto a real number.
- The agent can also use these numbers to weigh the importance of competing goals.

Model-based, Utility-based Agents



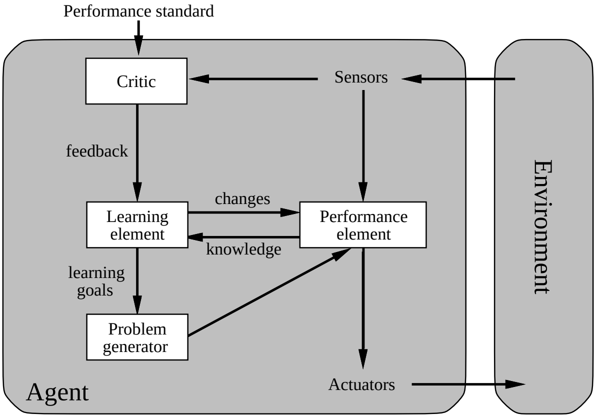
Learning Agents

- Learning agents can become more competent over time.
- They can start with an initially empty knowledge base.
- They can operate in initially unknown environments.

Components of Learning Agents

- learning element (responsible for making improvements)
- performance element (has to select external actions)
- critic (determines the performance of the agent)
- problem generator (suggests actions that will lead to informative experiences)

Learning Agents



Lecture Overview

The Environment of Rational Agents

- Accessible vs. inaccessible (fully observable vs. partially observable)
  Are the relevant aspects of the environment accessible to the sensors?
- Deterministic vs. stochastic
  Is the next state of the environment completely determined by the current state and the selected action? If only actions of other agents are nondeterministic, the environment is called strategic.
- Episodic vs. sequential
  Can the quality of an action be evaluated within an episode (perception + action), or are future developments decisive for the evaluation of quality?
- Static vs. dynamic
  Can the environment change while the agent is deliberating? If the environment does not change but if the agent's performance score changes as time passes by the environment is denoted as semi-dynamic.
- Discrete vs. continuous
  Is the environment discrete (chess) or continuous (a robot moving in a room)?
- Single agent vs. multi-agent
  Which entities have to be regarded as agents? There are competitive and cooperative scenarios.

Examples of Environments

Task	Observable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	fully	deterministic	sequential	static	discrete	single
Chess with a clock	fully	strategic	sequential	semi	discrete	multi
Poker	partially	stochastic	sequential	static	discrete	multi
Backgammon	fully	stochastic	sequential	static	discrete	multi
Taxi driving	partially	stochastic	sequential	dynamic	continuous	multi
Medical diagnosis	partially	stochastic	sequential	dynamic	continuous	single
Image analysis	fully	deterministic	episodic	semi	continuous	single
Part-picking robot	partially	stochastic	episodic	dynamic	continuous	single
Refinery controller	partially	stochastic	sequential	dynamic	continuous	single
Interactive English tutor	partially	stochastic	sequential	dynamic	discrete	multi

Whether an environment has a certain property also depends on the conception of the designer.

Summary

- An agent is something that perceives and acts. It consists of an architecture and an agent program.
- An ideal rational agent always takes the action that maximizes its performance given the percept sequence and its knowledge of the environment.
- An agent program maps from a percept to an action.
- There are a variety of designs
  - Reflex agents respond immediately to percepts.
  - Goal-based agents work towards goals.
  - Utility-based agents try to maximize their reward.
  - Learning agents improve their behavior over time.
- Some environments are more demanding than others.
- Environments that are partially observable, nondeterministic, strategic, dynamic, and continuous and multi-agent are the most challenging.

Foundations of Artificial Intelligence

3. Solving Problems by Searching

Problem-Solving Agents, Formulating Problems, Search Strategies

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and
Michael Tangermann

Albert-Ludwigs-Universität Freiburg

~~Apr 26~~, 2019

Cont March 8

Contents

Lecture Overview

1. Problem-Solving Agents
2. Formulating Problems
3. Problem Types
4. Example Problems
5. Search Strategies

Problem-Solving Agents

→ Goal-based agents

Formulation: *problem* as a *state-space* and *goal* as a *particular condition on states*

Given: *initial state*

Goal: To reach the specified goal (a state) through the *execution of appropriate actions*

→ Search for a suitable action sequence and execute the actions

A Simple Problem-Solving Agent

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

Properties of this Agent

- Fully observable environment
- Deterministic environment
- Static environment
- Discrete states
- Single-agent setting

Lecture Overview

Problem Formulation

- Goal formulation
  World states with desired properties
- Definition of the state space
  (important: only the relevant aspects $\rightarrow$ abstraction)
- Definition of the actions that can change the world state
- Definition of the problem type, which depends on the knowledge of the world states and actions
  $\rightarrow$ states in the search space
- Specification of the search costs (search costs, offline costs) and the execution costs (path costs, online costs)

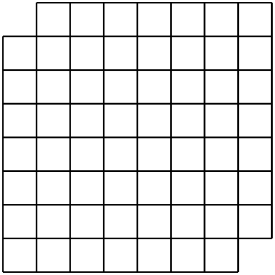
Problem Formulation

- Goal formulation
  World states with desired properties
- Definition of the state space
  (important: only the relevant aspects $\rightarrow$ abstraction)
- Definition of the actions that can change the world state
- Definition of the problem type, which depends on the knowledge of the world states and actions
  $\rightarrow$ states in the search space
- Specification of the search costs (search costs, offline costs) and the execution costs (path costs, online costs)

Note: The problem formulation can have a serious influence on the difficulty of finding a solution.

Example Problem Formulation

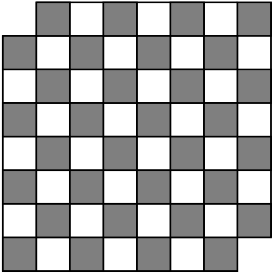
Given an $n \times n$ board from which two diagonally opposite corners have been removed (here 8×8):



Goal: Cover the board completely with dominoes, each of which covers two neighboring squares.

$\rightarrow$ Goal, state space, actions, search, . . .

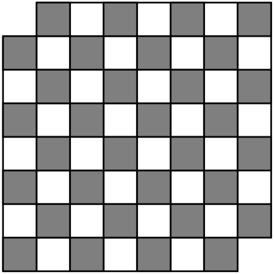
Alternative Problem Formulation



Question:

Can a chess board consisting of $n^2/2$ black and $n^2/2 - 2$ white squares be completely covered with dominoes such that each domino covers one black and one white square?

Alternative Problem Formulation



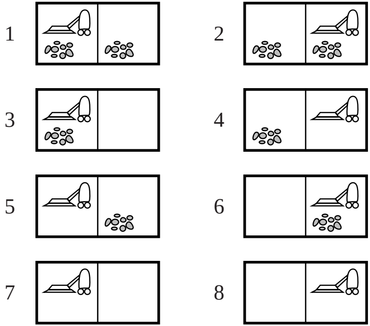
Question:

Can a chess board consisting of $n^2/2$ black and $n^2/2 - 2$ white squares be completely covered with dominoes such that each domino covers one black and one white square?

... clearly not.

Problem Formulation for the Vacuum Cleaner World

- World state space:
  2 positions, dirt or no dirt
  $\rightarrow$ 8 world states
- Actions:
  $Left$ (L), $Right$ (R), or $Suck$ (S)
- Goal:
  no dirt in the rooms
- Path costs:
  one unit per action



Lecture Overview

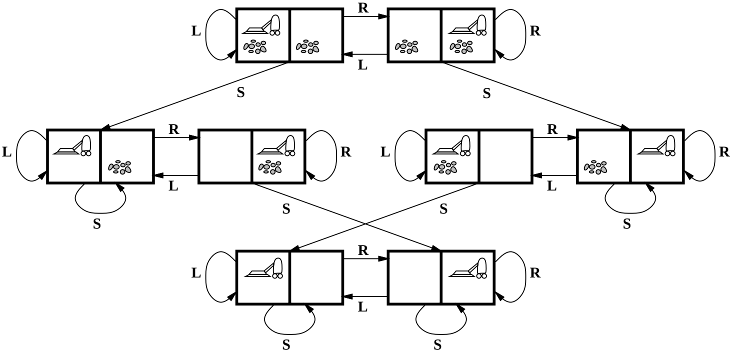
1. Problem-Solving Agents
2. Formulating Problems
3. **Problem Types**
4. Example Problems
5. Search Strategies

Problem Types: Knowledge of States and Actions

- State is completely observable
  Complete world state knowledge
  Complete action knowledge
  $\rightarrow$ The agent always knows its world state
- State is partially observable
  Incomplete world state knowledge
  Incomplete action knowledge
  $\rightarrow$ The agent only knows which group of world states it is in
- Contingency problem
  It is impossible to define a complete sequence of actions that constitute a solution in advance because the choice of the right action depends on infrmation that will only be available at execution time.
  $\rightarrow$ We need branching on observations
- Exploration problem
  State space and effects of actions unknown. Difficult!

The Vacuum Cleaner Problem

If the environment is completely observable, the vacuum cleaner always knows where it is and where the dirt is. The solution then is reduced to searching for a path from the initial state to a goal state.



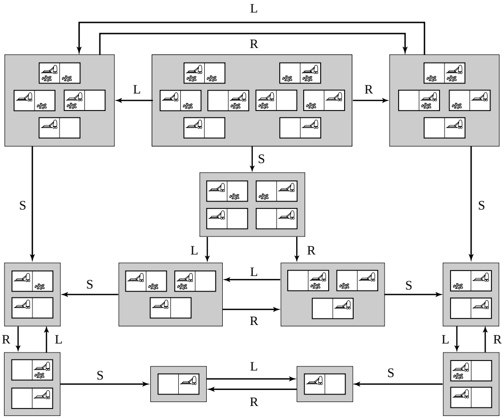
States for the search: The world states 1-8.

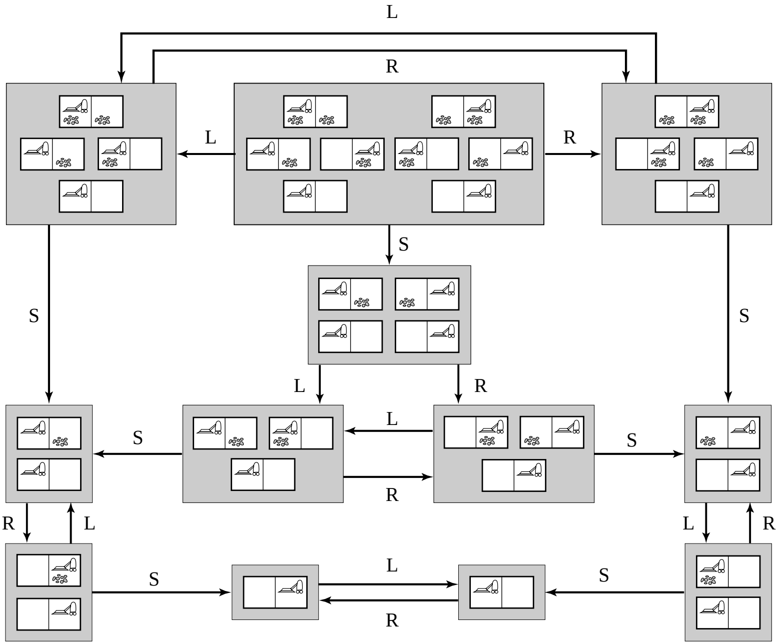
The Vacuum Cleaner World as a Partially Observable State Problem

If the vacuum cleaner has no sensors, it does not know where it or the dirt is.

In spite of this, it can still solve the problem. Here, states are knowledge states.

States for the search: The power set of the world states 1-8.




L
R
L
R
S
S
S
L
R
L
S
S
R
R
L
L
R
S
L
S
R

Concepts (1)

Initial State: The state the agent is in at the beginning

State Space: Set of all possible states

Actions: Description of possible actions. Available actions might be a function of the state.

Transition Model: Description of the outcome of an action (successor function)

Goal Test: Tests whether the state description matches a goal state

Concepts (2)

Path: A sequence of actions leading from one state to another

Path Costs: Cost function g over paths. Usually the sum of the costs of the actions along the path

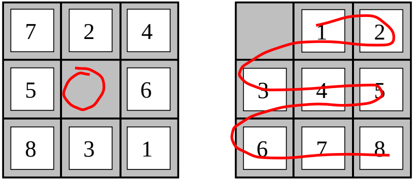
Solution: Path from the initial to a goal state

Search Costs: Time and storage requirements to find a solution

Total Costs: Search costs + path costs

Lecture Overview

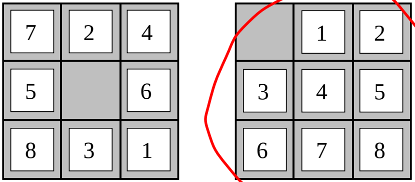
Example: The 8-Puzzle



Start State

Goal State

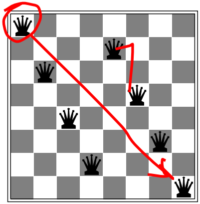
Example: The 8-Puzzle



Start State Goal State

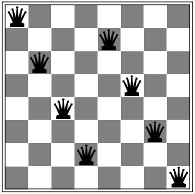
- States: Description of the location of each of the eight tiles and (for efficiency) the blank square.
- Initial State: Initial configuration of the puzzle.
- Actions (transition model defined accordingly): Moving the blank left, right, up, or down.
- Goal Test: Does the state match the configuration on the right (or any other configuration)?
- Path Costs: Each step costs 1 unit (path costs corresponds to its length).

Almost a solution:



Example: 8-Queens Problem

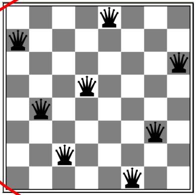
Almost a solution:



- States:
  Any arrangement of 0 to 8 queens on the board.
- Initial state:
  No queen on the board.
- Successor function:
  Add a queen to an empty field on the board.
- Goal test:
  8 queens on the board such that no queen attacks another.
- Path costs:
  0 (we are only interested in the solution).

Example: 8-Queens Problem

A solution:



- States:
  Any arrangement of 0 to 8 queens on the board.
- Initial state:
  No queen on the board.
- Successor function:
  Add a queen to an empty field on the board.
- Goal test:
  8 queens on the board such that no queen attacks another.
- Path costs:
  0 (we are only interested in the solution).

Alternative Formulation

- Naïve formulation
  - States: any arrangement of the 0–8 queens
  - State space: $64 \times 63 \times \cdots \times 57 \approx 10^{14}$ possible states

Alternative Formulation

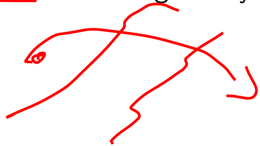
- Naïve formulation
  - States: any arrangement of the 0–8 queens
  - State space: $64 \times 63 \times \cdots \times 57 \approx 10^{14}$ possible states
- Better formulation
  - States: any arrangement of n queens $(0 \leq n \leq 8)$ one per column in the leftmost n columns such that no queen attacks another.
  - Successor function: add a queen to any square in the leftmost empty column.
  - State space: 2057 states

Example: Missionaries and Cannibals

Informal problem description:

- Three missionaries and three cannibals are on one side of a river that they wish to cross.
- A boat is available that can hold at most two people.
- You must never leave a group of missionaries outnumbered by cannibals on the same bank.

→ Find an action sequence that brings everyone safely to the opposite bank.



Formalization of the M&C Problem

States: triple (x, y, z) with $0 \leq x, y, z \leq 3$, where x, y and z represent the number of missionaries, cannibals and boats currently on the original bank.

Initial State: $(3, 3, 1)$ $(0,3,0)$

Successor function: from each state, either bring one missionary and one cannibal, two missionaries, two cannibals, or one of each type to the other bank.

Note: not all states are attainable (e.g., $(0, 0, 1)$) and some are illegal.

Goal State: $(0, 0, 0)$

Path Costs: 1 unit per crossing

Examples of Real-World Problems

- Route Planning, Shortest Path Problem
  Simple in principle (polynomial problem). Complications arise when path costs are unknown or vary dynamically (e.g., route planning in Canada)
- Travelling Salesperson Problem (TSP)
  A common prototype for NP-complete problems
- VLSI Layout
  Another NP-complete problem
- Robot Navigation (with high degrees of freedom)
  Difficulty increases quickly with the number of degrees of freedom. Further possible complications: errors of perception, unknown environments
- Assembly Sequencing
  Planning of the assembly of complex objects (by robots)

Lecture Overview

General Search

From the initial state, produce all successive states step by step → search tree.

(a) initial state (3,3,1)

General Search

From the initial state, produce all successive states step by step → search tree.

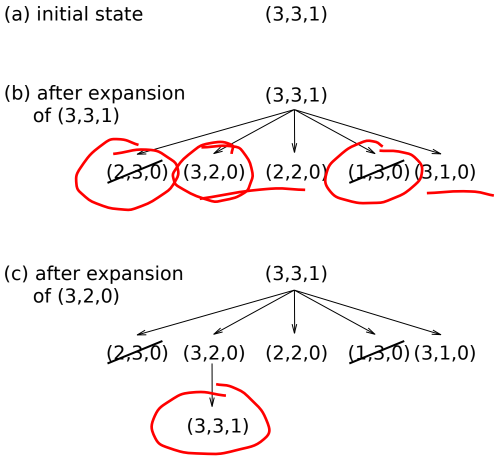
(a) initial state (3,3,1) left bank

(b) after expansion of (3,3,1)

(3,3,1)

→ ~~(2,3,0)~~ (3,2,0) (2,2,0) ~~(1,3,0)~~ (3,1,0)

From the initial state, produce all successive states step by step → search tree.



Some notations

- node expansion
  generating all successor nodes considering the available actions
- frontier
  set of all nodes available for expansion
- search strategy
  defines which node is expanded next
- tree-based search
  it might happen, that within a search tree a state is entered repeatedly, leading even to infinite loops. To avoid this,
- graph-based search keeps a set of already visited states, the so-called explored set.

Implementing the Search Tree

Data structure for each node n in the search tree:

n.STATE:	the state in the state space to which the node corresponds
n.PARENT:	the node in the search tree that generated this node
n.ACTION:	the action that was applied to the parent to generate the node
n.PATH-COST:	the cost, traditionally denoted by $g(n)$ of the path from the initial state to the node, as indicated by the parent pointers

Implementing the Search Tree

Data structure for each node n in the search tree:

n.STATE: the state in the state space to which the node corresponds

n.PARENT: the node in the search tree that generated this node

n.ACTION: the action that was applied to the parent to generate the node

n.PATH-COST: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers

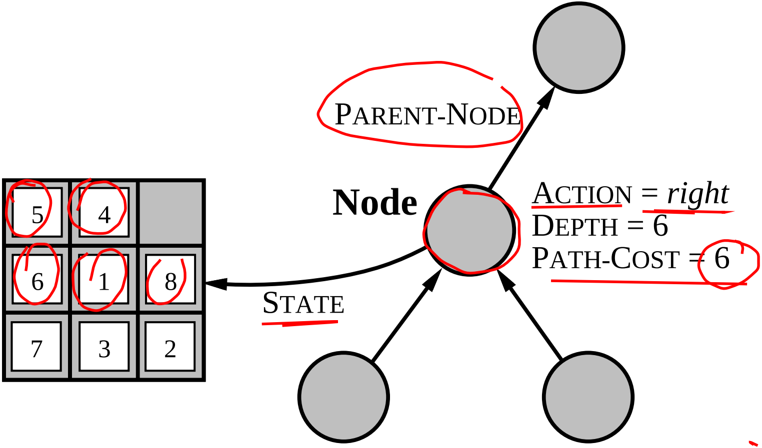
Operations on a queue:

EMPTY?(*queue*): returns true only if there are no more elements in the queue

POP(*queue*): removes the first element of the queue and returns it

INSERT(*element*, *queue*): inserts an element (various possibilities) and returns the resulting queue

Nodes in the Search Tree



General Tree-Search Procedure

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

General Graph-Search Procedure

```
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
```

Criteria for Search Strategies

Completeness: Is the strategy guaranteed to find a solution when there is one?

Time Complexity: How long does it take to find a solution?

Space Complexity: How much memory does the search require?

Optimality: Does the strategy find the best solution (with the lowest path cost)?

- problem describing quantities
  b: branching factor
  d: depth of shallowest goal node
  m: maximum length of any path in the state space

Search Strategies

Uninformed or blind searches

No information on the length or cost of a path to the solution.

- breadth-first search, uniform cost search, depth-first search,
- depth-limited search, iterative deepening search and
- bi-directional search.

Uninformed or blind searches

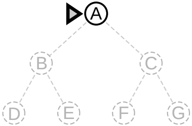
No information on the length or cost of a path to the solution.

- breadth-first search, uniform cost search, depth-first search,
- depth-limited search, iterative deepening search and
- bi-directional search.

In contrast: informed or heuristic approaches

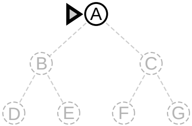
Breadth-First Search (1)

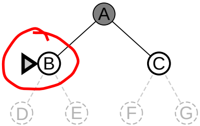
Nodes are expanded in the order they were produced
($frontier \leftarrow$ a FIFO queue).



Breadth-First Search (1)

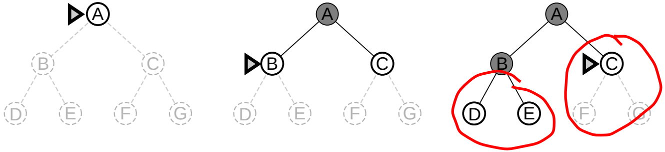
Nodes are expanded in the order they were produced
(*frontier* $\leftarrow$ a FIFO queue).





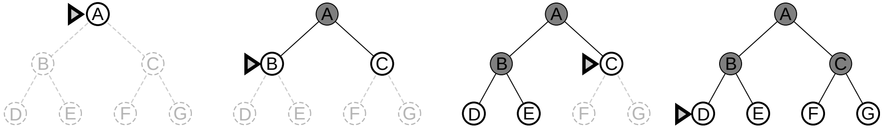
Breadth-First Search (1)

Nodes are expanded in the order they were produced
($frontier \leftarrow$ a FIFO queue).



Breadth-First Search (1)

Nodes are expanded in the order they were produced ($\mathit{frontier} \leftarrow$ a FIFO queue).



- Always finds the shallowest goal state first.
- Completeness is obvious.
- The solution is optimal, provided every action has identical, non-negative costs.

Breadth-First Search (2)

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

Breadth-First Search (3)

Time Complexity:

Let b be the maximal branching factor and d the depth of a solution path. Then the maximal number of nodes expanded is

$$b + b^2 + b^3 + \cdots + b^d \in O(b^d)$$

(Note: If the algorithm were to apply the goal test to nodes when selected for expansion rather than when generated, the whole layer of nodes at depth d would be expanded before the goal was detected and the time complexity would be $O(b^{d+1})$)

Breadth-First Search (3)

Time Complexity:

Let b be the maximal branching factor and d the depth of a solution path. Then the maximal number of nodes expanded is

$$b + b^2 + b^3 + \cdots + b^d \in O(b^d)$$

(Note: If the algorithm were to apply the goal test to nodes when selected for expansion rather than when generated, the whole layer of nodes at depth d would be expanded before the goal was detected and the time complexity would be $O(b^{d+1})$)

Space Complexity:

Every node generated is kept in memory. Therefore space needed for the frontier is $O(b^d)$ and for the explored set $O(b^{d-1})$.

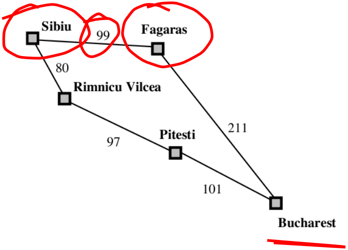
Breadth-First Search (4)

Example: $b = 10$; $1,000,000$ nodes/second; $1,000$ bytes/node:

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

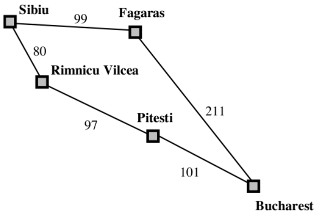
Uniform-Cost Search

- If step costs for doing an action are equal, then breadth-first search finds path with the optimal costs.
- If step costs are different (e.g., map: driving from one place to another might differ in distance), then uniform-cost search is a means to find the optimal solution.
- Uniform-cost search expands the node with the lowest path costs $g(n)$. Realization: priority queue.



Uniform-Cost Search

- If step costs for doing an action are equal, then breadth-first search finds path with the optimal costs.
- If step costs are different (e.g., map: driving from one place to another might differ in distance), then uniform-cost search is a means to find the optimal solution.
- Uniform-cost search expands the node with the lowest path costs $g(n)$. Realization: priority queue.

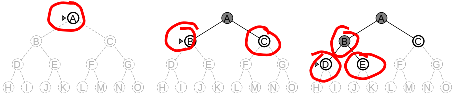


Always finds the cheapest solution, given that $g(successor(n)) \geq g(n)$ for all n.

Always expands an unexpanded node at the greatest depth ($frontier \leftarrow$ a LIFO queue).

It is common to realize depth-first search as a recursive function

Example (Nodes at depth 3 are assumed to have no successors):

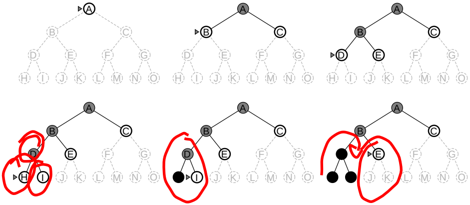


Depth-First Search (1)

Always expands an unexpanded node at the greatest depth ($frontier \leftarrow$ a LIFO queue).

It is common to realize depth-first search as a recursive function

Example (Nodes at depth 3 are assumed to have no successors):

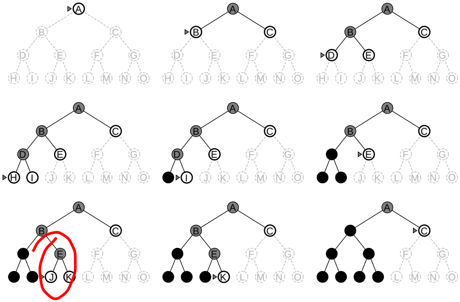


Depth-First Search (1)

Always expands an unexpanded node at the greatest depth ($frontier \leftarrow$ a LIFO queue).

It is common to realize depth-first search as a recursive function

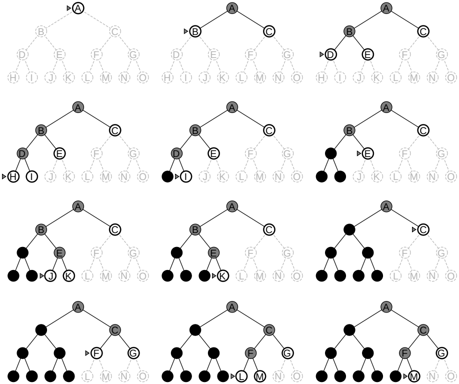
Example (Nodes at depth 3 are assumed to have no successors):



Depth-First Search (1)

Always expands an unexpanded node at the greatest depth
($frontier \leftarrow$ a LIFO queue).
It is common to realize depth-first search as a recursive function

Example (Nodes at depth 3 are assumed to have no successors):



Depth-First Search (2)

- In general, the solution found is not optimal.
- Completeness can be guaranteed only for graph-based search and finite state spaces.
- Algorithm: see later (depth-limited search)

Depth-First Search (3)

Time Complexity:

- in graph-based search bounded by the size of the state space (might be infinite!);
- in tree-based search, algorithm might generate $O(b^m)$ nodes in the search tree which might be much larger than the size of the state space. (m is the maximum length of a path in the state space).

Depth-First Search (3)

Time Complexity:

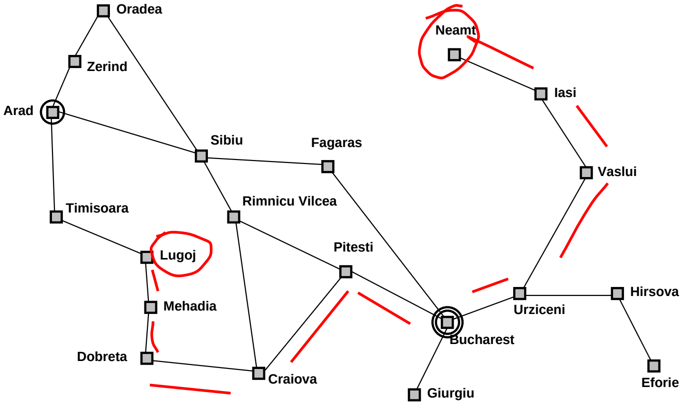
- in graph-based search bounded by the size of the state space (might be infinite!);
- in tree-based search, algorithm might generate $O(b^m)$ nodes in the search tree which might be much larger than the size of the state space. (m is the maximum length of a path in the state space).

Space Complexity:

- tree-based search: needs to store only the nodes along the path from the root to the leaf node. Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored. Therefore, memory requirement is only $O(b\,m)$. This is the reason, why it is practically so relevant despite all the other shortcomings!
- graph-based search: in worst case, all states need to be stored in the explored set (no advantage over breadth-first)

Depth-Limited Search (1)

Depth-first search with an imposed cutoff on the maximum depth of a path. e.g., route planning: with n cities, the maximum depth is $n-1$.



Sometimes, the search depth can be refined. E.g., here, a depth of 9 is sufficient (you can reach every city in at most 9 steps).

Depth-Limited Search (2)

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

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

Iterative Deepening Search (1)

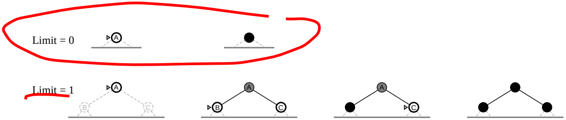
- Idea: use depth-limited search and in every iteration increase search depth by one.
- Looks a bit like a waste of resources (since the first steps are always repeated), but complexity-wise it is not so bad as it might seem.
- Combines depth- and breadth-first searches.
- Optimal and complete like breadth-first search, but requires much less memory: $O(b\,d)$.
- Time complexity is only little bit worse than breadth-first search (see later).

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

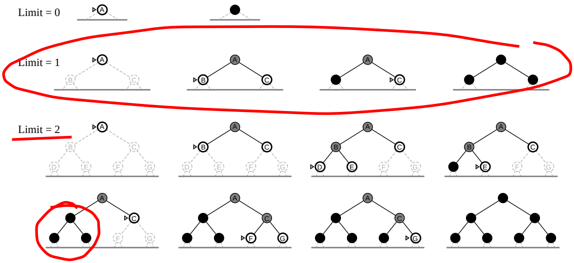
Example

Limit = 0

Example



Example



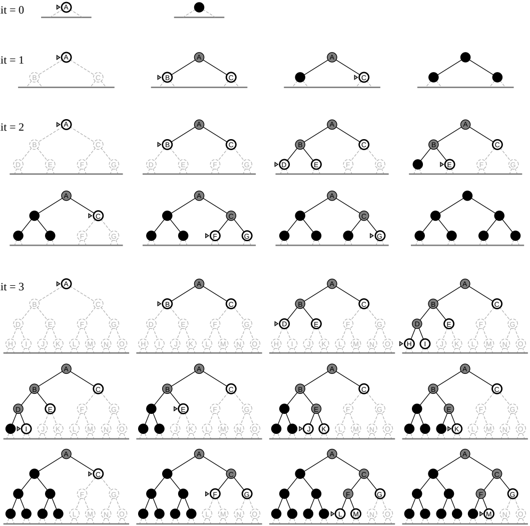
Example

Limit = 0

Limit = 1

Limit = 2

Limit = 3



Iterative Deepening Search (2)

Number of expansions

Iterative Deepening Search	$(d)b + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1b^d$
Breadth-First-Search	$b + b^2 + \cdots + b^{d-1} + b^d$

Example: $b = 10$, $d = 5$

Breadth-First-Search	$10 + 100 + 1,000 + 10,000 + 100,000$ $= 111,110$
Iterative Deepening Search	$50 + 400 + 3,000 + 20,000 + 100,000$ $= 123,450$

For $b = 10$, IDS expands only 11% more than the number of nodes expanded by (optimized) breadth-first-search.

Iterative Deepening Search (2)

Number of expansions

Iterative Deepening Search	$(d)b + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1b^d$
Breadth-First-Search	$b + b^2 + \cdots + b^{d-1} + b^d$

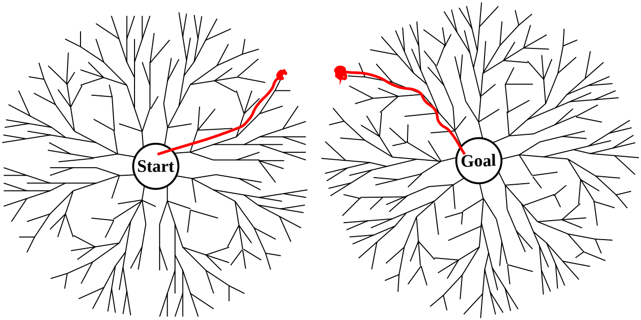
Example: $b = 10$, $d = 5$

Breadth-First-Search	$10 + 100 + 1,000 + 10,000 + 100,000$ $= 111,110$
Iterative Deepening Search	$50 + 400 + 3,000 + 20,000 + 100,000$ $= 123,450$

For $b = 10$, IDS expands only 11% more than the number of nodes expanded by (optimized) breadth-first-search.

$\rightarrow$ *Iterative deepening in general is the preferred uninformed search method when there is a large search space and the depth of the solution is not known.*

Bidirectional Searches



As long as forward and backward searches are symmetric, search times of $O(2 \cdot b^{d/2}) = O(b^{d/2})$ can be obtained.

E.g., for $b = 10$, $d = 6$, instead of $1,111,110$ only $2,220$ nodes!

Problems with Bidirectional Search

- The operators are not always reversible, which makes calculation the predecessors very difficult.
- In some cases there are many possible goal states, which may not be easily describable. Example: the predecessors of the checkmate in chess.
- There must be an efficient way to check if a new node already appears in the search tree of the other half of the search.
- What kind of search should be chosen for each direction (the previous figure shows a breadth-first search, which is not always optimal)?

Comparison of Search Strategies

Time complexity, space complexity, optimality, completeness

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

b branching factor
d depth of solution
m maximum depth of the search tree
l depth limit
C^* cost of the optimal solution
ϵ minimal cost of an action

Superscripts:
[a] b is finite
[b] if step costs not less than ϵ
[c] if step costs are all identical
[d] if both directions use breadth-first search

Summary

- Before an agent can start searching for solutions, it must formulate a goal and then use that goal to formulate a problem.
- A problem consists of five parts: The state space, initial situation, actions, goal test and path costs. A path from an initial state to a goal state is a solution.
- A general search algorithm can be used to solve any problem. Specific variants of the algorithm can use different search strategies.
- Search algorithms are judged on the basis of completeness, optimality, time complexity and space complexity.

Foundations of Artificial Intelligence

4. Informed Search Methods

Heuristics, Local Search Methods, Genetic Algorithms

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

May 8, 2019

Contents

Lecture Overview

1. Best-First Search
2. A* and IDA*
3. Local Search Methods
4. Genetic Algorithms

Best-First Search

Search procedures differ in the way they determine the next node to expand.

Uninformed Search: Rigid procedure with no knowledge of the cost of a given node to the goal.

Informed Search: Knowledge of the worth of expanding a node n is given in the form of an *evaluation function* $f(n)$, which assigns a real number to each node. Mostly, $f(n)$ includes as a component a *heuristic function* $h(n)$, which estimates the costs of the cheapest path from n to the goal.

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

Best-First Search: Informed search procedure that expands the node with the "best" f-value first.

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

Best-first search is an instance of the general TREE-SEARCH algorithm in which *frontier* is a priority queue ordered by an evaluation function f.

When f is always correct, we do not need to search!

Greedy Search

A possible way to judge the “worth” of a node is to estimate its path-costs to the goal.

$$h(n) = \textit{estimated path-costs from } n \textit{ to the goal}$$

The only real restriction is that $h(n) = 0$ if n is a goal.

A best-first search using $h(n)$ as the evaluation function, i.e., $f(n) = h(n)$ is called a *greedy search*.

Example: route-finding problem:
$h(n) =$

Greedy Search

A possible way to judge the "worth" of a node is to estimate its path-costs to the goal.

$$h(n) = \textit{estimated path-costs from } n \textit{ to the goal}$$

The only real restriction is that $h(n) = 0$ if n is a goal.

A best-first search using $h(n)$ as the evaluation function, i.e., $f(n) = h(n)$ is called a *greedy search*.

Example: route-finding problem:
$h(n) =$ straight-line distance from n to the goal

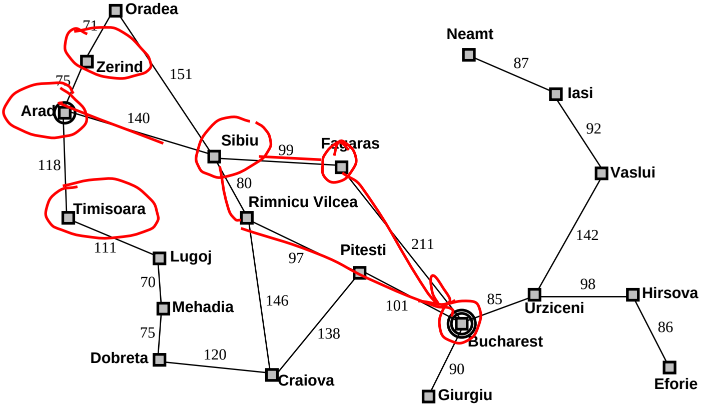
Heuristics

The evaluation function h in greedy searches is also called a *heuristic function* or simply a *heuristic*.

- The word *heuristic* is derived from the Greek word $\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$ (note also: $\varepsilon\upsilon\rho\eta\kappa\alpha$!)
- The mathematician Polya introduced the word in the context of problem solving techniques.
- In AI it has two meanings:
  - Heuristics are fast but in certain situations incomplete methods for problem-solving [Newell, Shaw, Simon 1963] (The greedy search is actually generally incomplete).
  - Heuristics are methods that improve the search in the average-case.

→ In all cases, the heuristic is *problem-specific* and *focuses* the search!

Greedy Search Example

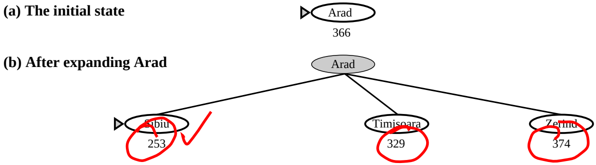


Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

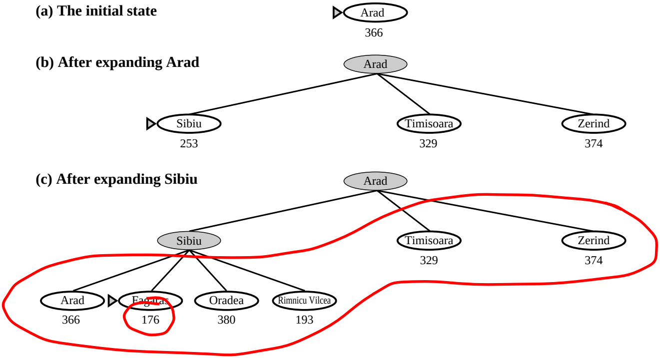
Greedy Search from Arad to Bucharest

(a) The initial state

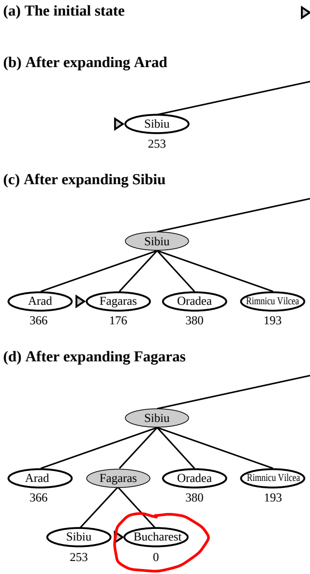
Greedy Search from Arad to Bucharest



Greedy Search from Arad to Bucharest



Greedy Search from Arad to Bucharest



Greedy Search - Properties

- a good heuristic might reduce search time drastically
- non-optimal
- incomplete
- graph-search version is complete only in finite spaces

Can we do better?

Lecture Overview

A^*: Minimization of the Estimated Path Costs

A^* combines greedy search with the uniform-cost search:
Always expand node with lowest $f(n)$ first, where

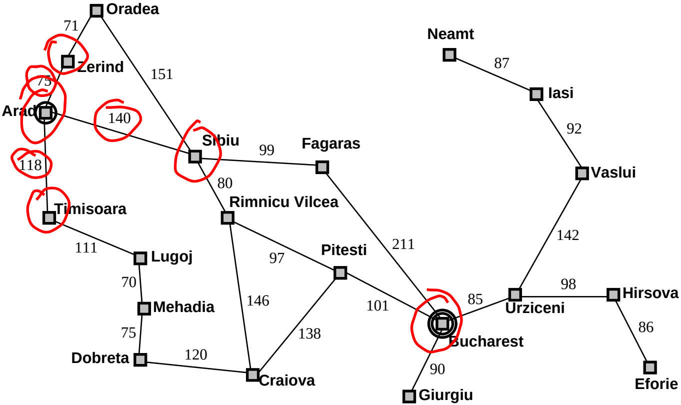
$g(n)$ = actual cost from the initial state to n.
$h(n)$ = estimated cost from n to the nearest goal.
$f(n) = g(n) + h(n)$,
the estimated cost of the cheapest solution through n.

Let $h^*(n)$ be the actual cost of the optimal path from n to the nearest goal. h is *admissible* if the following holds for all n:

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

We require that for A^*, h is admissible (example: straight-line distance is admissible).
In other words, h is an *optimistic* estimate of the costs that actually occur.

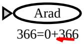
A* Search Example



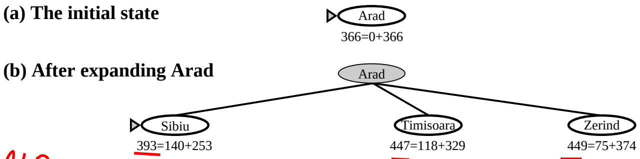
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search from Arad to Bucharest

(a) The initial state

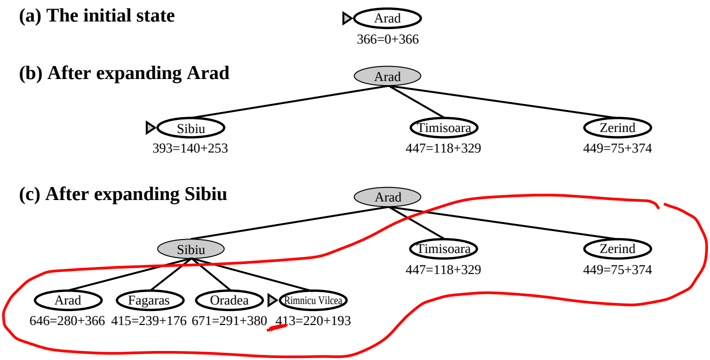


A* Search from Arad to Bucharest

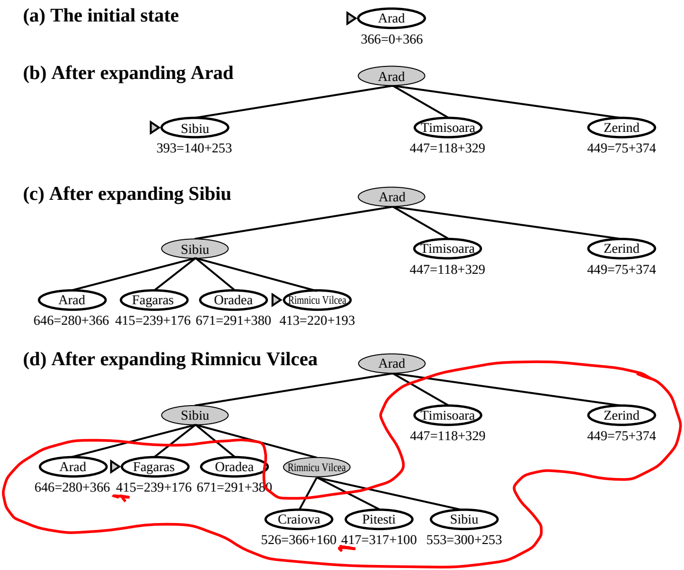


$f(n) = 140 + 253$

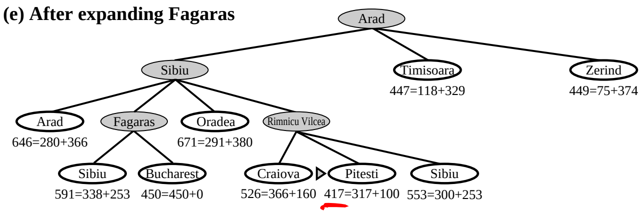
A* Search from Arad to Bucharest



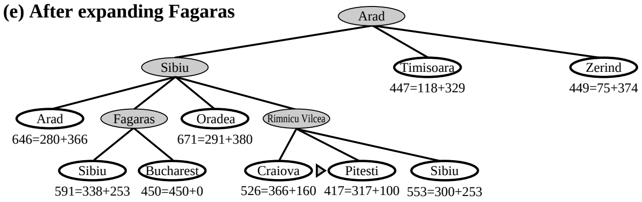
A* Search from Arad to Bucharest

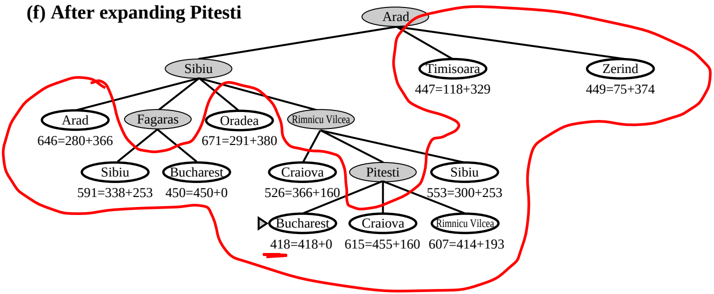


A* Search from Arad to Bucharest

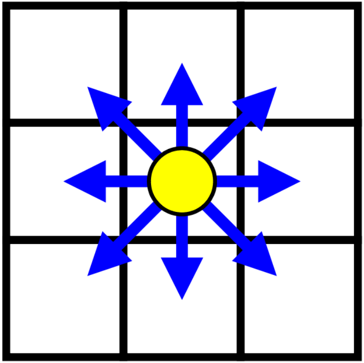


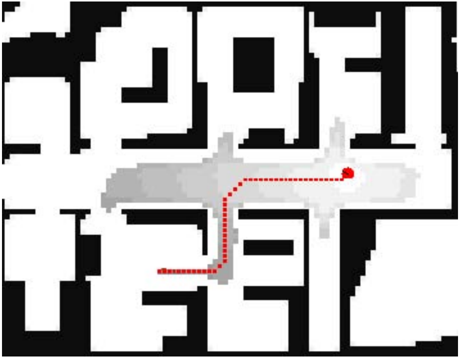
A* Search from Arad to Bucharest





Example: Path Planning for Robots in a Grid-World





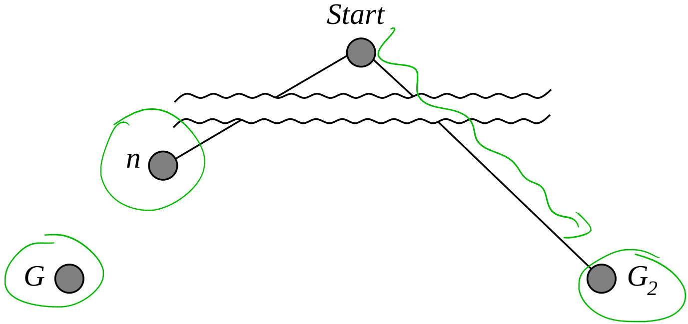
Live-Demo: `http://qiao.github.io/PathFinding.js/visual/`

Optimality of A*

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

Claim: The first solution found (= node is expanded and found to be a goal node) has the minimum path cost.

Proof: Suppose there exists a goal node G with optimal path cost f^*, but A* has first found another node G_2 with $g(G_2) > f^*$.



Optimality of A*

Let n be a node on the path from the start to G that has not yet been expanded. Since h is admissible, we have

$$f(n) \leq f^*.$$

Since n was not expanded before G_2, the following must hold:

$$f(G_2) \leq f(n)$$

and

$$f(G_2) \leq f^*.$$

It follows from $h(G_2) = 0$ that

$$g(G_2) \leq f^*.$$

$\rightarrow$ Contradicts the assumption!

Completeness and Complexity

Completeness:

If a solution exists, A* will find it provided that (1) every node has a finite number of successor nodes, and (2) there exists a positive constant $\delta > 0$ such that every step has at least cost δ.

$\rightarrow$ there exists only a finite number of nodes n with $f(n) \leq f^*$.

Complexity:

In general, still exponential in the path length of the solution (space, time)

More refined complexity results depend on the assumptions made, e.g. on the quality of the heuristic function. Example:

In the case in which $|h^*(n) - h(n)| \leq O(\log(h^*(n))$, only one goal state exists, and the search graph is a tree, a sub-exponential number of nodes will be expanded [Gaschnig, 1977, Helmert & Roeger, 2008].
Unfortunately, this almost never holds.

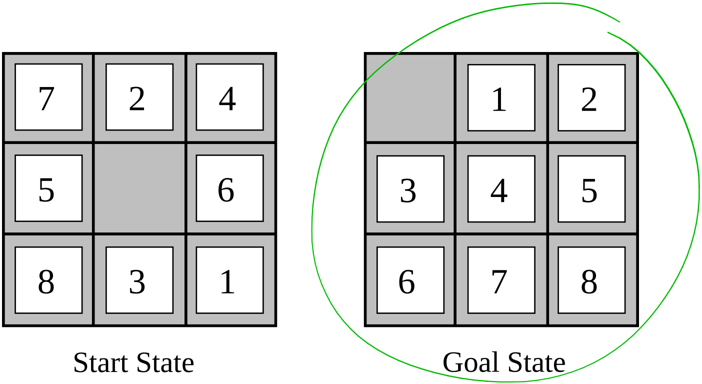
A note on Graph- vs. Tree-Search

- A* as described is a tree-search (and may consider duplicates)
- For the graph-based variant, one
  - either needs to consider re-opening nodes from the *explored* set, when a better estimate becomes known, or
  - one needs needs to require stronger restrictions on the heuristic estimate: it needs to be consistent.

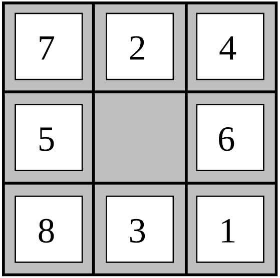
→ A heuristic h is called **consistent** iff for all actions a leading from s to s': $h(s) - h(s') \leq c(a)$, where $c(a)$ denotes the cost of action a. (Consistent heuristics prevent the need to re-open nodes from the *explored* set.)

- **Note**: Consistency implies admissibility.
- **Note**: A* can still be applied if heuristic is not consistent, but optimality is lost in this case.

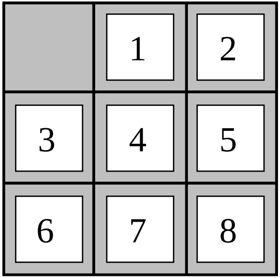
Heuristic Function Example



Heuristic Function Example



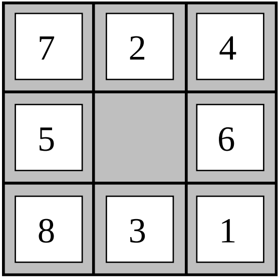
Start State



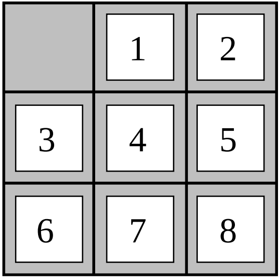
Goal State

$h_1 =$ the number of tiles in the wrong position

Heuristic Function Example



Start State



Goal State

h_1 = the number of tiles in the wrong position

h_2 = the sum of the distances of the tiles from their goal positions (*Manhattan distance*)

Empirical Evaluation

- d = distance from goal
- Average over 100 instances

Manhattan

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A* (h_1)	A* (h_2)	IDS	A* (h_1)	A* (h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	-	539	113	-	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.47
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

Variants of A*

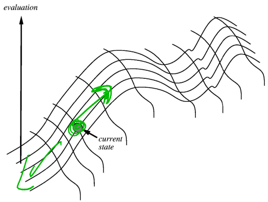
A* in general still suffers from exponential memory growth. Therefore, several refinements have been suggested:

- iterative-deepening A*, where the f-costs are used to define the cutoff (rather than the depth of the search tree): **IDA***
- Recursive Best First Search (RBFS): introduces a variable f_limit to keep track of the best alternative path available from any ancestor of the current node. If current node exceeds this limit, recursion unwinds back to the alternative path.
- other alternatives memory-bounded A* (MA*) and simplified MA* (SMA*).

Lecture Overview

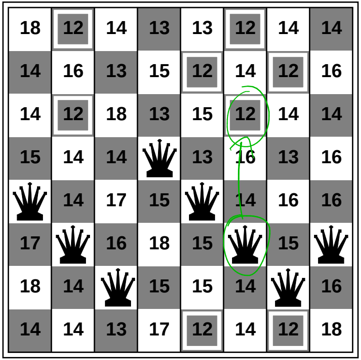
Local Search Methods

- In many problems, it is unimportant how the goal is reached—only the goal itself matters (8-queens problem, VLSI Layout, TSP).
- If in addition a quality measure for states is given, **local search** can be used to find solutions.
- It operates using a single current node (rather than multiple paths).
- It requires little memory (cp. to most *systematic search strategies*!)
- **Idea**: Begin with a randomly-chosen configuration and improve on it step by step → **Hill Climbing / Gradient Decent**.
- **Note**: It can be used for maximization or minimization respectively (see 8-queens example)



Example: 8-queens Problem (1)

Example state with heuristic cost estimate $h = 17$ (counts the number of pairs threatening each other directly or indirectly).

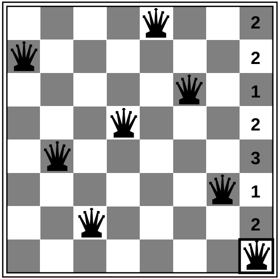


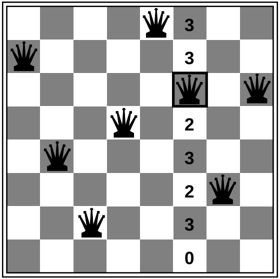
Hill Climbing

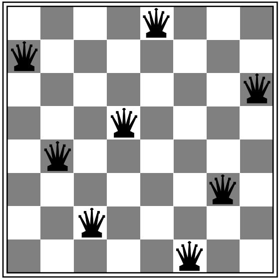
function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

> *current* ← MAKE-NODE(*problem*.INITIAL-STATE)
> **loop do**
>   *neighbor* ← a highest-valued successor of *current*
>   **if** neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
>   *current* ← *neighbor*

Possible realization of a hill-climbing algorithm:
Select a column and move the queen to the square with the fewest conflicts.







Problems with Local Search Methods

- *Local maxima:* The algorithm finds a sub-optimal solution.
- *Plateaus:* Here, the algorithm can only explore at random.
- Ridges: Similar to plateaus but might even require suboptimal moves.

Solutions:

- *Start over* when no progress is being made.
- "Inject noise" $\rightarrow$ random walk

Which strategies (with which parameters) are successful (within a problem class) can usually only empirically be determined.

Example: 8-queens Problem (Local Minimum)

Local minimum ($h = 1$) of the 8-queens Problem. Every successor has a higher cost.

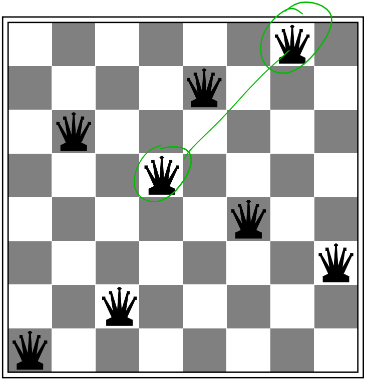
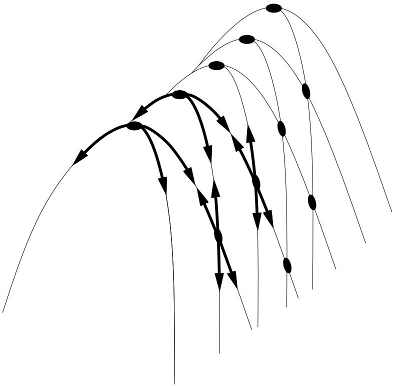


Illustration of the ridge problem

The grid of states (dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima, that are not directly connected to each other. From each local maximum, all the available actions point downhill.



Performance figures for the 8-queens Problem

The 8-queens problem has about $8^8 \approx 17\ million$ states. Starting from a random initialization, hill-climbing directly finds a solution in about 14% of the cases. On average it requires only 4 steps!

Better algorithm: Allow sideways moves (no improvement), but restrict the number of moves (avoid infinite loops!).

E.g.: max. 100 moves: Solves 94%, number of steps raises to 21 steps for successful instances and 64 for failure cases.

Simulated Annealing

In the simulated annealing algorithm, “noise” is injected systematically: first a lot, then gradually less.

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to “temperature”

  current $\leftarrow$ MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next $\leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE
    if $\Delta E > 0$ **then** *current* $\leftarrow$ *next*
    else *current* $\leftarrow$ *next* only with probability $e^{\Delta E/T}$

Has been used since the early 80’s for VSLI layout and other optimization problems.

Lecture Overview

1. Best-First Search
2. A* and IDA*
3. Local Search Methods
4. Genetic Algorithms

Genetic Algorithms

Evolution appears to be very successful at finding good solutions.

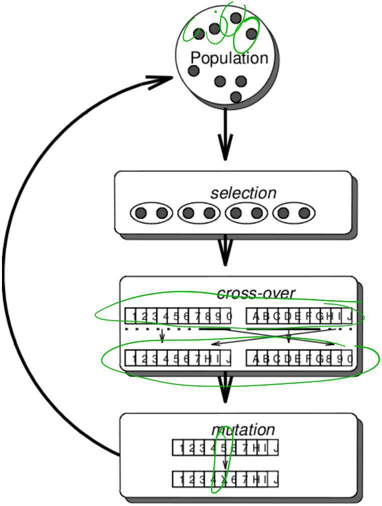
Idea: Similar to evolution, we search for solutions by three operators: "mutation", "crossover", and "selection".

Ingredients:

- Coding of a solution into a string of symbols or bit-string
- A fitness function to judge the worth of configurations
- A population of configurations

Example: Represent an *individual* (one 8-queens configuration) as a concatenation of eight x-y coordinates. Its fitness is judged by the number of non-attacks. The *population* consists of a set of configurations.

Selection, Mutation, and Crossing



Many variations:

how selection will be applied, what type of cross-over operators will be used, etc.

Selection of individuals according to a fitness function and pairing

Calculation of the breaking points and recombination

According to a given probability elements in the string are modified.

Summary

- Heuristics focus the search
- Best-first search expands the node with the highest worth (defined by any measure) first.
- With the minimization of the evaluated costs to the goal h we obtain a greedy search.
- The minimization of $f(n) = g(n) + h(n)$ combines uniform and greedy searches. When $h(n)$ is admissible, i.e., h^* is never overestimated, we obtain the A* search, which is complete and optimal.
- IDA* is a combination of the iterative-deepening and A* searches.
- Local search methods work on a single state only, attempting to improve it step-wise.
- Genetic algorithms imitate evolution by combining good solutions.

Foundations of Artificial Intelligence

5. Board Games

Search Strategies for Games, Games with Chance, State of the Art

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

May 10, 2019

Contents

Lecture Overview

1. Board Games
2. Minimax Search
3. Alpha-Beta Search
4. Games with an Element of Chance
5. State of the Art

Why Board Games?

Playing board games is one of the oldest research areas in AI (Shannon and Turing 1950).

- Board games present a very abstract and pure form of competition between two opponents and clearly require a form of "intelligence".
- The states of a game are easy to represent.
- The possible actions of the players are well-defined.

→ Implementation of the game as a kind of search problem

→ The individual states are fully accessible

→ It is nonetheless a contingency problem, because the actions of the opponent are not under the control of the player

Problems

Board games are not only difficult because they are contingency problems, but also because the state space can become astronomically large.

Examples:

- **Chess**: On average 35 possible actions from every position; often, games have 50 moves per player, resulting in a search depth of 100:
  $\rightarrow 35^{100} \approx 10^{150}$ nodes in the search tree (with "only" 10^{40} legal chess positions).
- **Go**: On average 200 possible actions with ca. 300 moves
  $\rightarrow 200^{300} \approx 10^{700}$ nodes.

Good game programs have the properties that they

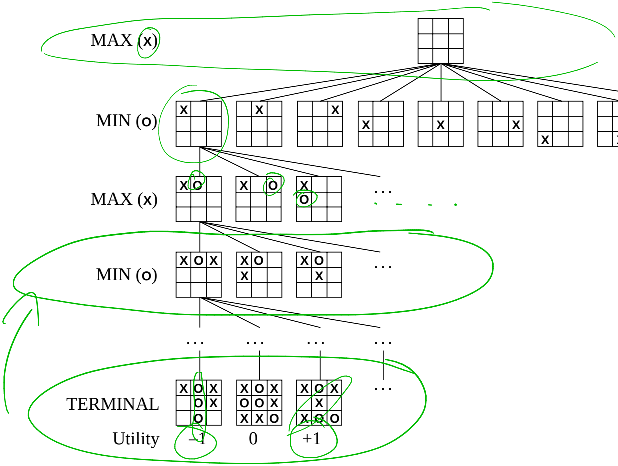
- delete irrelevant branches of the tree,
- use good evaluation functions for in-between states, and
- look ahead as many moves as possible.

Lecture Overview

Terminology of Two-Person Board Games

- Players are MAX and MIN, where MAX begins.
- Initial position (e.g., board arrangement)
- Operators (= legal moves)
- Game tree is the search tree generated from the possible (alternate) moves
- Termination test, determines when the game is over and what the value of the final state is
- Strategy. In contrast to regular searches, where a path from beginning to end is a solution, MAX must come up with a strategy to reach a favorable terminal state *regardless of what* MIN *does* → all of MIN's moves must be considered and reactions to them must be computed

Tic-Tac-Toe Example



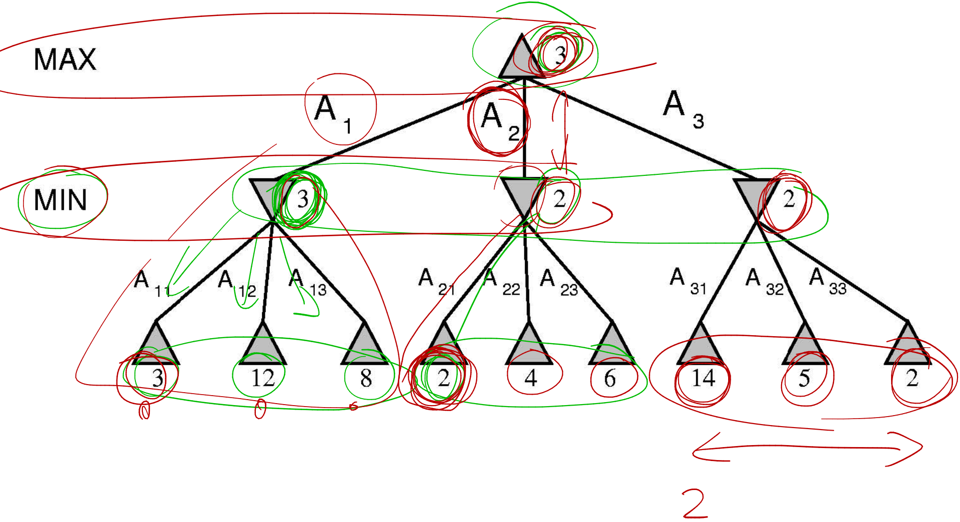
Every level of the game tree is given the player's name whose turn it is (MAX- and MIN-steps).

When it is possible, as it is here, to produce the full game tree, the minimax algorithm delivers an optimal strategy for MAX.

Minimax Algorithm

1. Generate the complete game tree using depth-first search (we do not need the full tree in memory!)
2. Apply the utility function to each terminal state.
3. Beginning with the terminal states, determine the utility of the predecessor nodes as follows:
   - Predecessor node is a MIN-node:
     Value is the minimum of its child nodes
   - Predecessor node is a MAX-node:
     Value is the maximum of its child nodes
   - From the initial state (root of the game tree), MAX chooses the move that leads to the highest value (minimax decision).

Note: Minimax assumes that MIN plays perfectly. Every weakness (i.e., every mistake MIN makes) can only improve the result for MAX.


MAX
3
A 1
A 2
A 3
MIN
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2
0
0
6
2

Minimax Algorithm

Recursively calculates the best move from the initial state.

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

function MAX-VALUE(*state*) **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)))
  return v

function MIN-VALUE(*state*) **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)))
  return v

Note: Minimax can only be applied to game trees that are not too deep. Otherwise, the minimax value must be approximated at a certain level.

Lecture Overview

Evaluation Function

When the search tree is too large, it can be expanded to a certain depth only. The art is to correctly evaluate the playing position of the leaves of the tree at that depth.

Example of simple evaluation criteria in chess:

Evaluation Function

When the search tree is too large, it can be expanded to a certain depth only. The art is to correctly evaluate the playing position of the leaves of the tree at that depth.

Example of simple evaluation criteria in chess:

- Material value: pawn 1, knight/bishop 3, rook 5, queen 9
- Other: king safety, good pawn structure
- Rule of thumb: three-point advantage = certain victory

The design of the evaluation function is important!

The value assigned to a state of play should reflect the chances of winning, i.e., the chance of winning with a one-point advantage should be less than with a three-point advantage.

Evaluation Function—General

material value pawn structure

The preferred evaluation functions are weighted, linear functions:

$$w_1 f_1 + w_2 f_2 + \cdots + w_n f_n$$

where the w's are the weights, and the f's are the features. [e.g., $w_1 = 3$, $f_1 =$ number of our own knights on the board]

The above linear sum makes the strong assumption that the contributions of all features are independent. (not true: e.g., bishops in the endgame are more powerful, when there is more space)

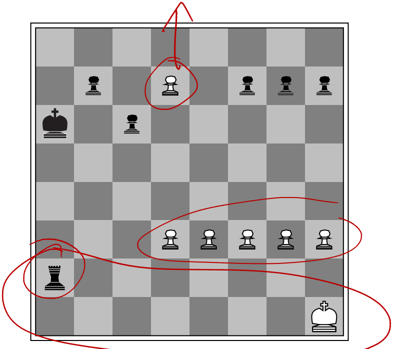
The weights can be learned. The features, however, are often designed by human intuition and understanding

When Should We Stop Growing the Tree?

Motivation: Return an answer within the allocated time.

- Fixed-depth search.
- Better: iterative deepening search (stop, when time is over).
- But only stop and evaluate in "quiescent" positions that will not cause large fluctuations in the evaluation function in the following moves. For example, if one can capture a figure, then the position is not "quiescent" because this action might change the evaluation substantially. It is better to continue the search in non-quiescent positions, preferably by only allowing certain types of moves (e.g., capturing) to reduce search effort, until a quiescent position is reached.
- There still is the problem of limited depth search: horizon effect (see next slide).

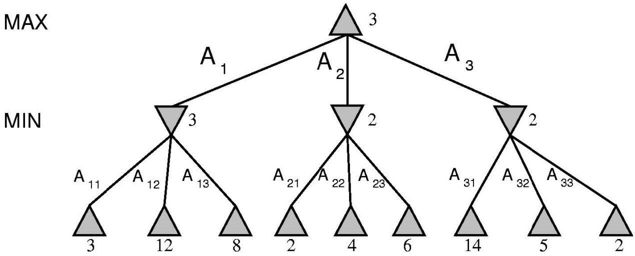
Horizon Problem



Black to move

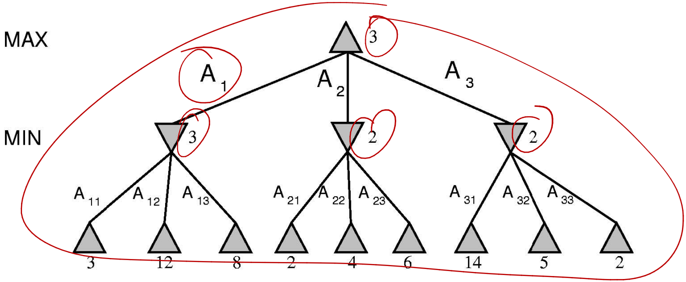
- Black has a slight material advantage
- . . . but will eventually lose (pawn becomes a queen).
- A fixed-depth search cannot detect this because it thinks it can avoid it (on the other side of the horizon—because black is concentrating on the check with the rook, to which white must react).

Alpha-Beta Pruning

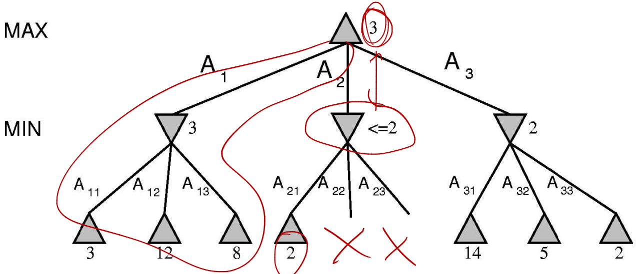


Can we improve this?

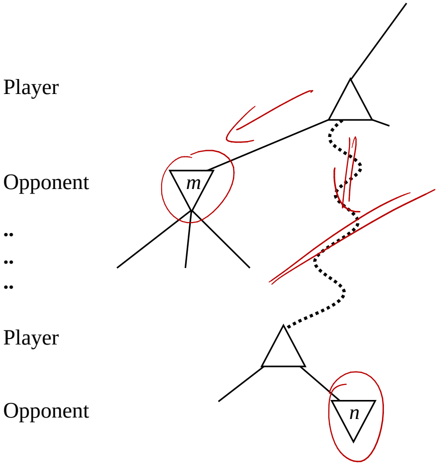
Alpha-Beta Pruning



Can we improve this? We do not need to consider all nodes.



Alpha-Beta Pruning: General



If $m > n$ we will never reach node n in the game.

Alpha-Beta Pruning

Minimax algorithm with depth-first search

α = the value of the best (i.e., highest-value) choice for MAX which we have found so far at any choice point along the path.
(MAX is guaranteed to get at least this value!)

β = the value of the best (i.e., lowest-value) choice for MIN that we have found so far at any choice point along the path.

When Can we Prune?

The following applies:

α values of MAX nodes can never decrease

β values of MIN nodes can never increase

(1) Prune below the MIN node whose β-bound is less than or equal to the α-bound of its MAX-predecessor node.

(2) Prune below the MAX node whose α-bound is greater than or equal to the β-bound of its MIN-predecessor node.

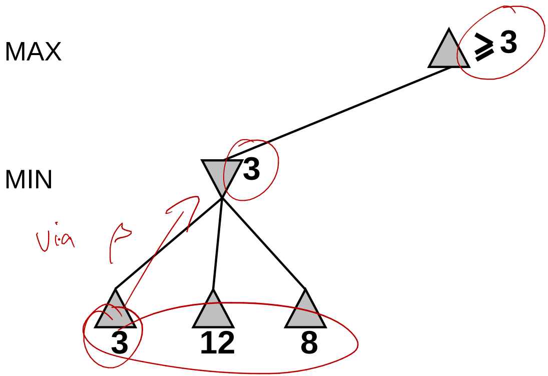
$\rightarrow$ Provides the same results as the complete minimax search to the same depth (because only irrelevant nodes are eliminated).

Alpha-Beta Search Algorithm

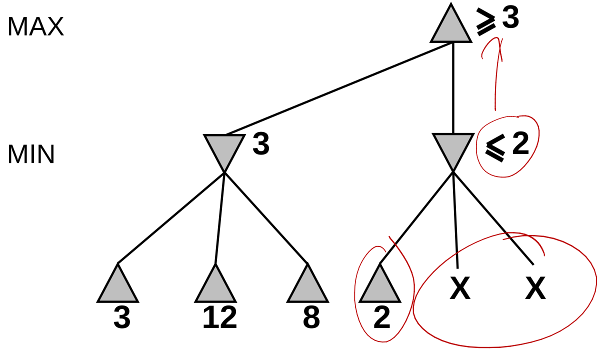
```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v

function MIN-VALUE(state, α, β) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← +∞
   for each a in ACTIONS(state) do
      v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Initial call with MAX-VALUE(*initial-state*, $-\infty$, $+\infty$)

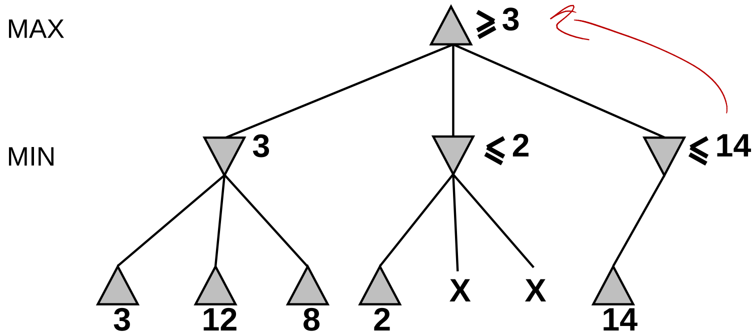
Alpha-Beta Pruning Example



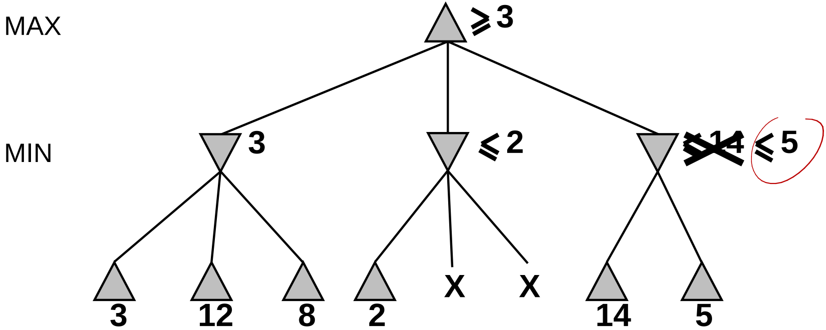
Alpha-Beta Pruning Example



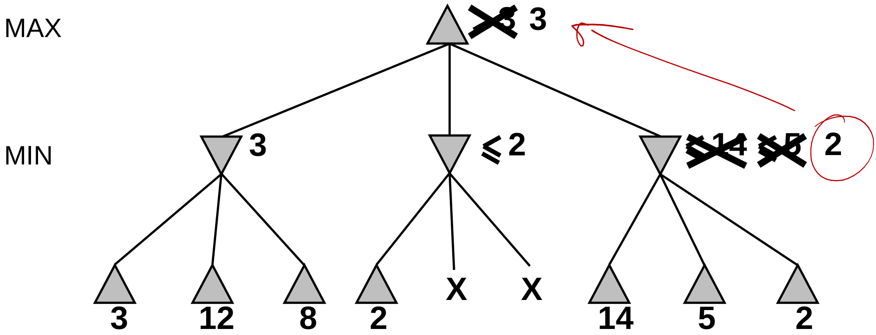
Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Efficiency Gain

- The alpha-beta search cuts the largest amount off the tree when we examine the best move first.
- In the best case (always the best move first), the search expenditure is reduced to $O(b^{d/2}) \Rightarrow$ we can search twice as deep in the same amount of time.
- In the average case (randomly distributed moves), for moderate b ($b < 100$), we roughly have $O(b^{3d/4})$.
- However, the best move typically is not known. Practical case: A simple ordering heuristic brings the performance close to the best case $\Rightarrow$ In chess, we can thus reach a depth of 6–7 moves.

Good ordering for chess to explore child nodes?

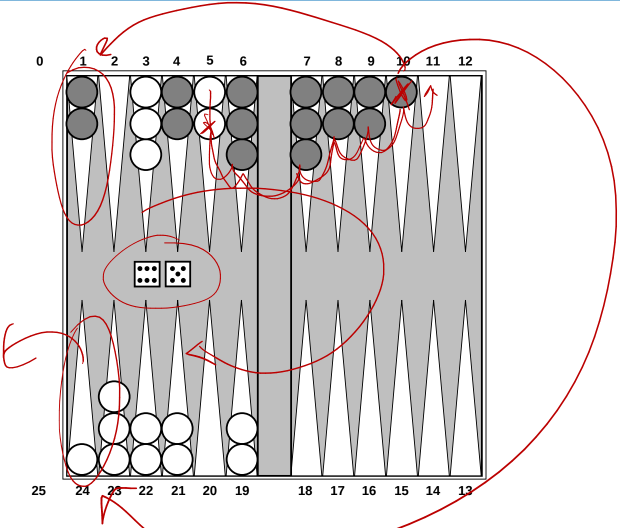
Efficiency Gain

- The alpha-beta search cuts the largest amount off the tree when we examine the best move first.
- In the best case (always the best move first), the search expenditure is reduced to $O(b^{d/2}) \Rightarrow$ we can search twice as deep in the same amount of time.
- In the average case (randomly distributed moves), for moderate b ($b < 100$), we roughly have $O(b^{3d/4})$.
- However, the best move typically is not known. Practical case: A simple ordering heuristic brings the performance close to the best case $\Rightarrow$ In chess, we can thus reach a depth of 6–7 moves.

Good ordering for chess to explore child nodes? Try captures first, then threats, then forward moves, then backward moves.

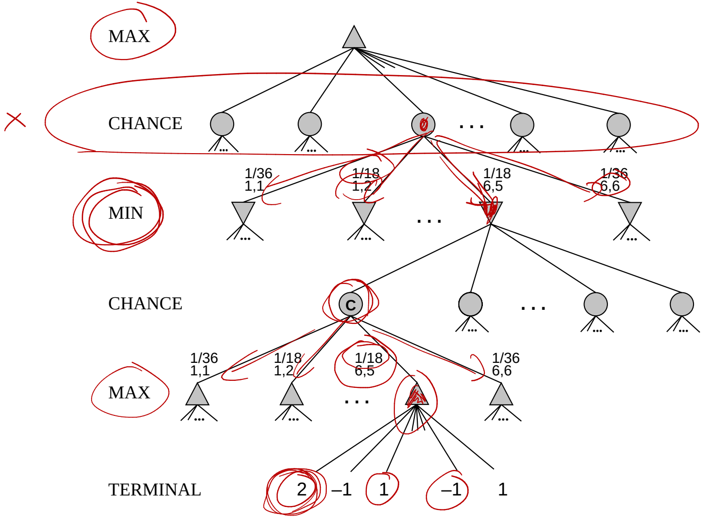
Lecture Overview

Games that Include an Element of Chance



White has just rolled a 6 and a 5 and has 4 legal moves.

In addition to MIN- and MAX nodes, we need chance nodes (for the dice).



Calculation of the Expected Value

Utility function for chance nodes C over MAX:

d_i: possible dice roll

$P(d_i)$: probability of obtaining that roll

$S(C, d_i)$: attainable positions from C with roll d_i

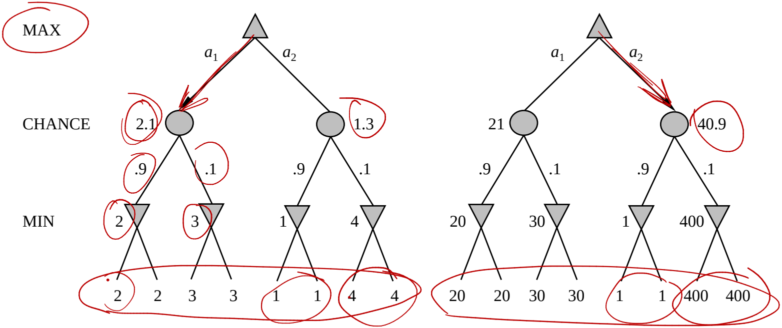
UTILITY(s): Evaluation of s

$$\text{EXPECTIMAX}(C) = \sum_i P(d_i) \max_{s \in S(C, d_i)} (\text{UTILITY}(s))$$

✗ EXPECTIMIN likewise

Problems

- Using expected values: An order-preserving transformation on the evaluation values may change the best move. Example:



- Chose the evaluation function with care!
- Search costs increase: Instead of $O(b^d)$, we get $O((b \times n)^d)$, where n is the number of possible dice outcomes.

→ In Backgammon ($n = 21$, $b = 20$, can be 4000) the maximum for d is 2.

Card Games

- Partially observable environment
- Recently card games such as bridge and poker have been addressed as well
- One approach: simulate play with open cards and then average over all possible plays (or make a Monte Carlo simulation) using minimax (perhaps modified)
- Pick the move with the best expected result (usually all moves will lead to a loss, but some give better results than others)
  - Averaging over clairvoyance
- Although "incorrect", appears to give reasonable results for e.g. bridge and skat.

Lecture Overview

1. Board Games
2. Minimax Search
3. Alpha-Beta Search
4. Games with an Element of Chance
5. State of the Art

State of the Art (1)

Backgammon: The *BKG* program defeated the official world champion in 1980. A newer program TD-Gammon is among the top 3 players.

Checkers, **draughts** (by international rules): A program called *Chinook* is the official world champion in man-computer competition (acknowledges by ACF and EDA) and is the highest-rated player:

Chinook: 2712	Ron King: 2632
Asa Long: 2631	Don Lafferty: 2625

In 1995, Chinook won a 32 game match against Don Lafferty.

Othello/Reversi: Very good, even on normal computers. In 1997, the *Logistello* program defeated the human world champion.

Chess: In 1997, world chess master G. Kasparow was beaten by a computer in a match of 6 games by *Deep Blue* (IBM Thomas J. Watson Research Center).
Special hardware (32 processors with 8 chips, 2 Mi. calculations per second) and special chess knowledge.

State of the Art (2)

Go: The program *AlphaGo* was able to beat in March 2016 one of the best human players Lee Sedol (according to ELO ranking the 4th best player worldwide) 4:1.
AlphaGo used Monte Carlo search techniques (UCT) and deep learning techniques for evaluations.

Poker: In January 2017, *Libratus* played against four top-class human poker players for 20 days **heads-up no-limit Texas hold 'em**. In the end, Libratus was more than 1.7 M$ ahead.
Libratus used a number of different techniques all based on game theory.

The Reasons for Success. . .

- Alpha-Beta-Search
- . . . with dynamic decision-making for uncertain positions
- Good (but usually simple) evaluation functions
- Large databases of opening moves
- Very large game termination databases (for checkers, all ten-piece situations)
- For Go, Monte-Carlo and machine learning techniques proved to be successful.
- . . . and very fast and parallel processors, huge memory, and plenty of plays.
- For Poker, game theoretic analysis together with extensive self-play (15 million core CPU hours) were important.

Summary

- A game can be defined by the initial state, the operators (legal moves), a terminal test and a utility function (outcome of the game).
- In two-player board games, the minimax algorithm can determine the best move by enumerating the entire game tree.
- The alpha-beta algorithm produces the same result but is more efficient because it prunes away irrelevant branches.
- Usually, it is not feasible to construct the complete game tree, so the utility of some states must be determined by an evaluation function.
- Games of chance can be handled by an extension of the alpha-beta algorithm.
- The success for different games is based on quite different methodolgies.

Foundations of Artificial Intelligence

6. Constraint Satisfaction Problems

CSPs as Search Problems, Solving CSPs, Problem Structure

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

May 15, 2019

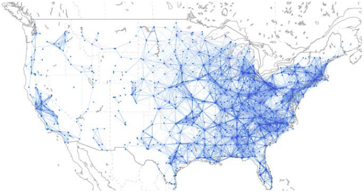
A motivating example of CSP (here: graph coloring)

- FCC Spectrum Auction in 2017
  - Wireless frequency spectra: demand increases
  - US Federal Communications Commission (FCC) held 13-month auction

A motivating example of CSP (here: graph coloring)

- FCC Spectrum Auction in 2017
  - Wireless frequency spectra: demand increases
  - US Federal Communications Commission (FCC) held 13-month auction

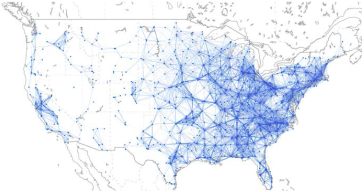
- Key Computational Problem: feasibility testing based on interference constraints
  - 2991 stations (nodes) &
    2.7 million interference constraints:
    stations in neighboring regions cannot use too similar frequencies
  - Need to check feasibility whenever an offer is made
  - More instances checkable: higher revenue



A motivating example of CSP (here: graph coloring)

- FCC Spectrum Auction in 2017
  - Wireless frequency spectra: demand increases
  - US Federal Communications Commission (FCC) held 13-month auction

- Key Computational Problem: feasibility testing based on interference constraints
  - 2991 stations (nodes) & 2.7 million interference constraints: stations in neighboring regions cannot use too similar frequencies
  - Need to check feasibility whenever an offer is made
  - More instances checkable: higher revenue



- Formulated as a CSP and solved with SAT solvers (improved by meta-algorithmics, see future lecture)
  - Improved ratio of instances solved from 73% to 99.6%
  - Net income for US government: $7 billion (used to pay down national debt)

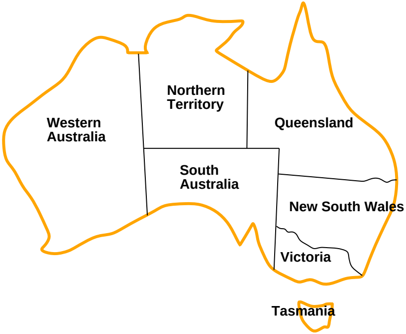
Contents

Lecture Overview

Constraint Satisfaction Problems

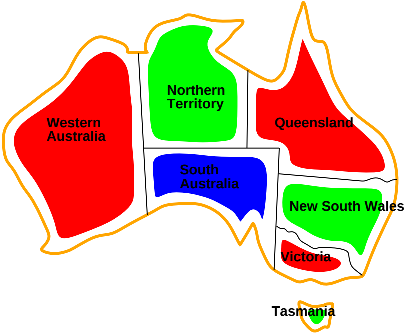
- A Constraint Satisfaction Problems (CSP) is given by
  - a set of variables $\{x_1, x_2, \ldots, x_n\}$,
  - an associated set of value domains $\{dom_1, dom_2, \ldots, dom_n\}$, and
  - a set of constraints. i.e., relations, over the variables.
  - An assignment of values to variables that satisfies all constraints is a solution of such a CSP.
- If CSPs are viewed as search problems, states are explicitly represented as variable assignments. CSP search algorithms take advantage of this structure.
- The main idea is to exploit the constraints to eliminate large portions of search space.
- *Formal representation language* with associated general inference algorithms

Example: Map-Coloring



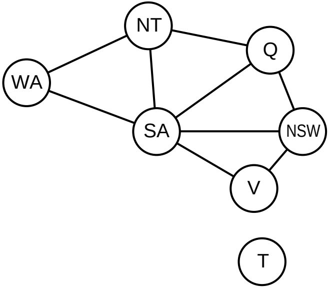
- Variables: WA, NT, SA, Q, NSW, V, T
- Values: $\{red, green, blue\}$
- Constraints: adjacent regions must have different colors, e.g., $NSW \neq V$

One Solution



- Solution assignment:
  - $\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Constraint Graph



- a constraint graph can be used to visualize binary constraints
- for higher order constraints, hyper-graph representations might be used
- Nodes = variables, arcs = constraints

Variations

- Binary, ternary, or even higher arity (e.g., ALL_DIFFERENT)
- Finite domains (d values) $\rightarrow d^n$ possible variable assignments
- Infinite domains (reals, integers)
  - *linear constraints (each variable occurs only in linear form)*: solvable (in P if real)
  - *nonlinear constraints*: unsolvable

Applications

- Timetabling (classes, rooms, times)
- Configuration (hardware, cars, . . .)
- Nurse rostering
- Scheduling (sports, etc)
- Sudoku
- . . .

Lecture Overview

1. What are CSPs?
2. Backtracking Search for CSPs
3. CSP Heuristics
4. Constraint Propagation
5. Problem Structure

Backtracking Search over Assignments

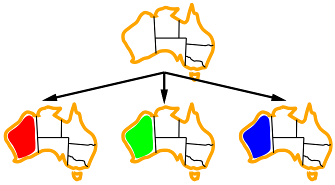
- Assign values to variables step by step (order does not matter)
- Consider only one variable per search node!
- DFS with single-variable assignments is called backtracking search

Algorithm

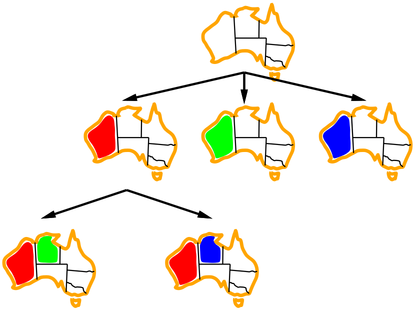
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← 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 ← INFERENCE(csp, var, value)
        if inferences ≠ failure then
           add inferences to assignment
           result ← BACKTRACK(assignment, csp)
           if result ≠ failure then
              return result
     remove {var = value} and inferences from assignment
  return failure
```

Example (1)

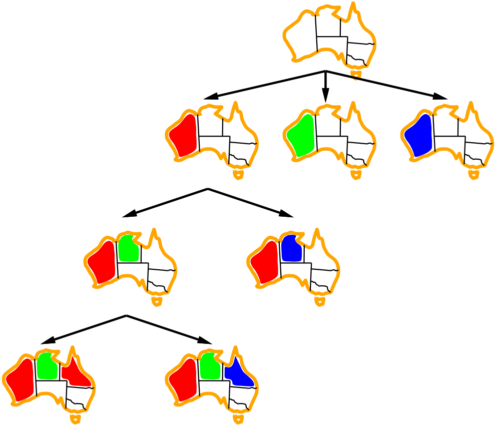
Example (2)



Example (3)



Example (4)



Lecture Overview

Improving Efficiency: CSP Heuristics & Pruning Techniques

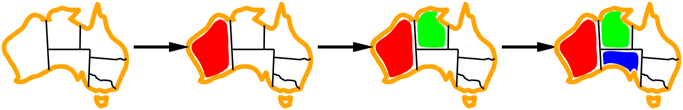
- Variable ordering: Which one to assign first?
- Value ordering: Which value to try first?
- Try to detect failures early on
- Try to exploit problem structure

$\rightarrow$ Note: all this is not problem-specific!

Variable Ordering: Most constrained first

- Most constrained variable:
  - choose the variable with the fewest remaining legal values

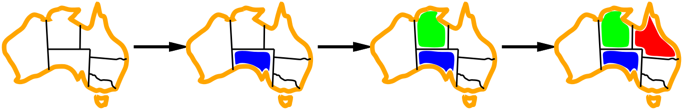
$\rightarrow$ reduces branching factor!



Variable Ordering: Most Constraining Variable First

- Break ties among variables with the same number of remaining legal values:
  - choose variable with the most constraints on remaining unassigned variables

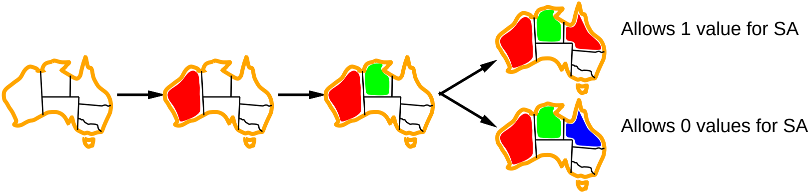
$\rightarrow$ reduces branching factor in the next steps



Value Ordering: Least Constraining Value First

- Given a variable,
  - choose first a value that rules out the fewest values in the remaining unassigned variables

$\rightarrow$ We want to find an assignment that satisfies the constraints (of course, this does not help if the given problem is unsatisfiable.)



Rule out Failures early on: Forward Checking

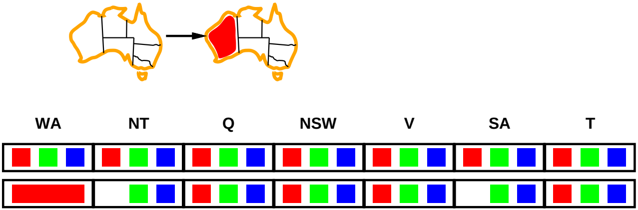
- Whenever a value is assigned to a variable, values that are now illegal for other variables are removed
- Implements what the ordering heuristics implicitly compute
- $WA =$ *red*, then NT cannot become *red*
- If all values are removed for one variable, we can stop!

Forward Checking (1)

- Keep track of remaining values
- Stop if all have been removed

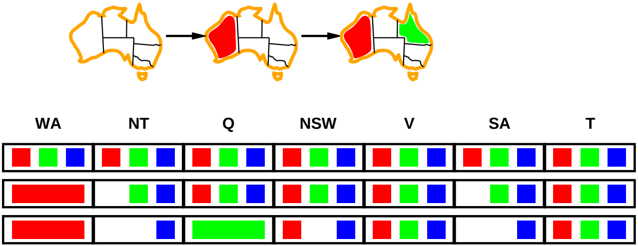
Forward Checking (2)

- Keep track of remaining values
- Stop if all have been removed



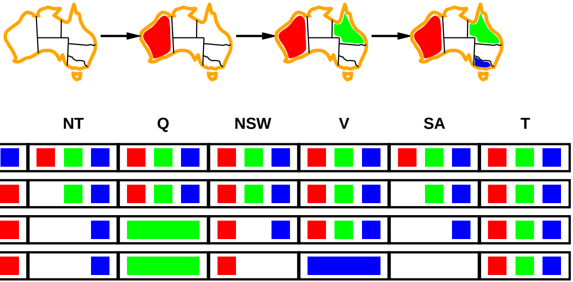
Forward Checking (3)

- Keep track of remaining values
- Stop if all have been removed



Forward Checking (4)

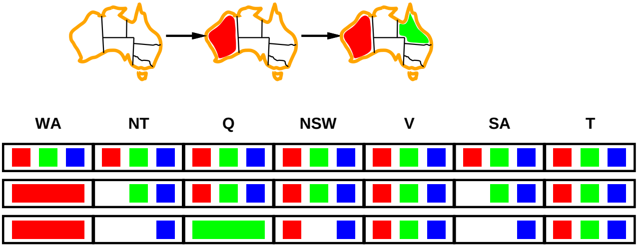
- Keep track of remaining values
- Stop if all have been removed



Lecture Overview

Forward Checking: Sometimes it Misses Something

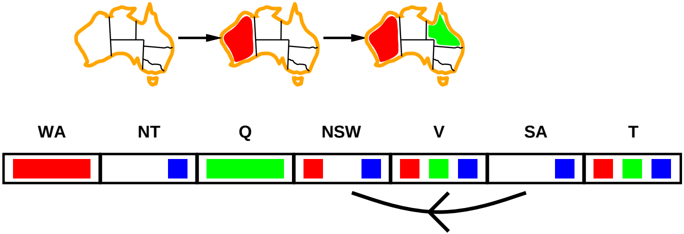
- Forward Checking propagates information from assigned to unassigned variables
- However, there is no propagation between unassigned variables



Arc Consistency

- A directed arc $X \rightarrow Y$ is "consistent" iff
  - for every value x of X, there exists a value y of Y, such that (x, y) satisfies the constraint between X and Y
- Remove values from the domain of X to enforce arc-consistency
- Arc consistency detects failures earlier
- Can be used as preprocessing technique or as a propagation step during backtracking

Arc Consistency Example



AC-3 Algorithm

```
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) ← 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 ← 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 ← true
  return revised
```

Properties of AC-3

- What is the computational complexity of AC-3?
  - Let n denote the number of nodes, and let d denote the maximal number of elements in a domain
  - Hint: what is the complexity of function REVISE, how often can it return true in the worst case, and how often is it thus called in the worst case?

Properties of AC-3

- What is the computational complexity of AC-3?
  - Let n denote the number of nodes, and let d denote the maximal number of elements in a domain
  - Hint: what is the complexity of function REVISE, how often can it return true in the worst case, and how often is it thus called in the worst case?

- AC-3 runs in $O(d^3n^2)$ time
  - REVISE takes $O(d^2)$ (for each element $x \in D_i$, you need to check each element $y \in D_j$)
  - Each time REVISE returns true one element of X_i is eliminated; there are only max. d elements for each of the n variables
  - Each time REVISE returns true up to n constraints are added to the queue
  - Alltogether, in the worst case, REVISE can only be called a maximum of $O(n^2d)$ times, each taking time $O(d^2)$

Properties of AC-3

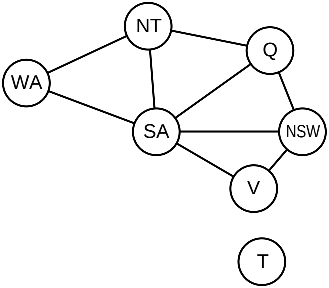
- What is the computational complexity of AC-3?
  - Let n denote the number of nodes, and let d denote the maximal number of elements in a domain
  - Hint: what is the complexity of function REVISE, how often can it return true in the worst case, and how often is it thus called in the worst case?

- AC-3 runs in $O(d^3 n^2)$ time
  - REVISE takes $O(d^2)$ (for each element $x \in D_i$, you need to check each element $y \in D_j$)
  - Each time REVISE returns true one element of X_i is eliminated; there are only max. d elements for each of the n variables
  - Each time REVISE returns true up to n constraints are added to the queue
  - Alltogether, in the worst case, REVISE can only be called a maximum of $O(n^2 d)$ times, each taking time $O(d^2)$

- Of course, AC-3 does not detect all inconsistencies (which is an NP-hard problem)

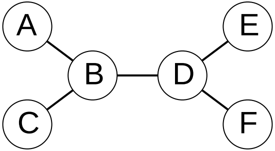
Lecture Overview

Problem Structure (1)



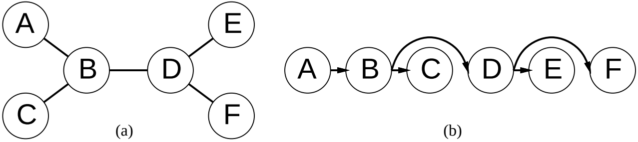
- This example CSP has two independent components
- Identifiable as connected components of constraint graph
- Can reduce the search space dramatically

Problem Structure (2): Tree-structured CSPs



- If the CSP graph is a tree, then it can be solved in $O(nd^2)$ (general CSPs need in the worst case $O(d^n)$).
- *Idea:* Pick root, order nodes, apply arc consistency from leaves to root, and assign values starting at root.

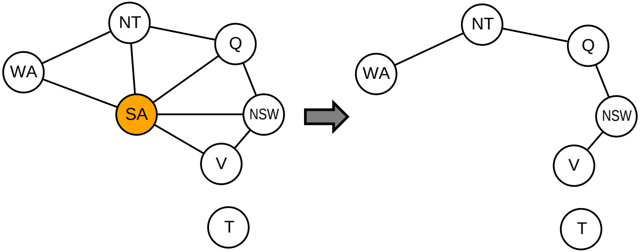
Problem Structure (2): Tree-structured CSPs



- Pick any variable as root; choose an ordering such that each variable appears after its parent in the tree.
- Apply arc-consistency to (x_i, x_k) when x_i is the parent of x_k for all $k = n$ down to 2 (any tree with n nodes has $n - 1$ arcs and per arc d^2 comparisons are needed, which results in a complexity of $O(n\, d^2)$).
- Now we can start at x_1 assigning values from the remaining domains without creating any conflict in one sweep through the tree!
- This algorithm is linear in n.

Problem Structure (3): Almost Tree-structured

Idea: Reduce the graph structure to a tree by fixing values in a reasonably chosen subset

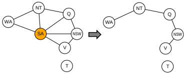


Instantiate a variable and prune values in neighboring variables is called Conditioning

Problem Structure (4): Almost Tree-structured

Algorithm Cutset Conditioning:

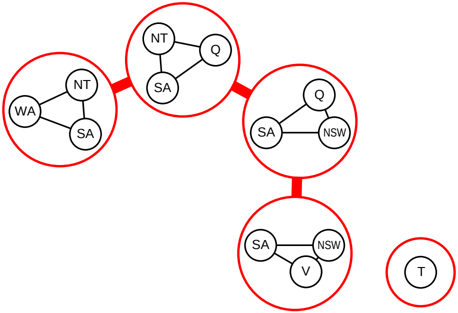
1. Choose a subset S of the CSPs variables such that the constraint graph becomes a tree after removal of S. The set S is called a cycle cutset.
2. For each possible assignment of variables in S that satisfies all constraints on S
   1. remove from the domains of the remaining variables any values that are inconsistent with the assignments for S, and
   2. if the remaining CSP has a solution, return it together with the assignment for S



Note: Finding the smallest cycle cutset is NP hard, but several efficient approximation algorithms are known.

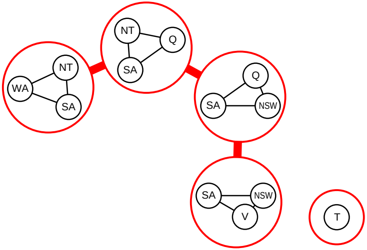
Another Method: Tree Decomposition (1)

- Decompose the problem into a set of connected sub-problems, where two sub-problems are connected when they share a constraint
- Solve the sub-problems independently and then combine the solutions



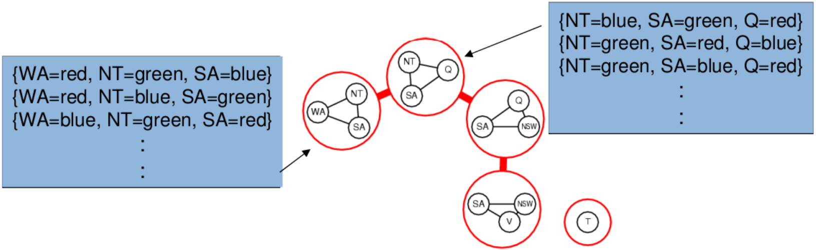
Another Method: Tree Decomposition (2)

- A tree decomposition must satisfy the following conditions:
  - Every variable of the original problem appears in at least one sub-problem
  - Every constraint appears in at least one sub-problem
  - If a variable appears in two sub-problems, it must appear in all sub-problems on the path between the two sub-problems
  - The connections form a tree



Another Method: Tree Decomposition (3)

- Consider sub-problems as new mega-variables, which have values defined by the solutions to the sub-problems
- Use technique for tree-structured CSP to find an overall solution (constraint is to have identical values for the same variable)



Tree Width

- The aim is to make the subproblems as small as possible. The tree width w of a tree decomposition is the size of largest sub-problem minus 1
- Tree width of a graph is minimal tree width over all possible tree decompositions
- If a graph has tree width w and we know a tree decomposition with that width, we can solve the problem in $O(nd^{w+1})$
- Unfortunately, finding a tree decomposition with minimal tree width is NP-hard. However, there are heuristic methods that work well in practice.

Summary

- CSPs are a special kind of search problem:
  - states are value assignments
  - goal test is defined by constraints
- Backtracking = DFS with one variable assigned per node. Other intelligent backtracking techniques possible
- Variable/value ordering heuristics can help dramatically
- Constraint propagation prunes the search space
- Tree structure of CSP graph simplifies problem significantly
- Cutset conditioning and tree decomposition are two ways to transform part of the problem into a tree
- CSPs can also be solved using local search

Foundations of Artificial Intelligence

7. Propositional Logic

Rational Thinking, Logic, Resolution

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

May 22, 2019

Motivation

Logic is a universal tool with many powerful applications

- Proving theorems
  - With the help of the algorithmic tools we describe here: automated theorem proving
- Formal verification
  - Verification of software
    - Ruling out unintended states (null-pointer exceptions, etc.)
    - Proving that the program computes the right solution
  - Verification of hardware (Pentium bug, etc)
- Basis for solving many NP-hard problems in practice
- Note: this and the next section (satisfiability) are based on Chapter 7 of the textbook ("Logical Agents")

Contents

Agents that Think Rationally

- Until now, the focus has been on agents that act rationally.
- Often, however, rational action requires rational (logical) thought on the agent's part.
- To that purpose, portions of the world must be represented in a knowledge base, or KB.
  - A KB is composed of sentences in a language with a truth theory (logic)
    - We (being external) can interpret sentences as statements about the world. (semantics)
    - Through their form, the sentences themselves have a causal influence on the agent's behavior. (syntax)
- Interaction with the KB through $\textsc{Ask}$ and $\textsc{Tell}$ (simplified):

$\textsc{Ask}(\text{KB},\alpha) = \text{yes}$	exactly when α follows from the KB
$\textsc{Tell}(\text{KB},\alpha) = \text{KB'}$	so that α follows from KB'
$\textsc{Forget}(\text{KB},\alpha) = \text{KB'}$	non-monotonic (will not be discussed)

3 Levels

In the context of knowledge representation, we can distinguish three levels [Newell 1990]:

Knowledge level: Most abstract level. Concerns the total knowledge contained in the KB. For example, the automated DB information system knows that a trip from Freiburg to Basel SBB with an ICE costs 24.70 €.

Logical level: Encoding of knowledge in a formal language.
$Price(Freiburg, Basel, 24.70)$

Implementation level: The internal representation of the sentences, for example:

- As a string `‘‘Price(Freiburg, Basel, 24.70)’’`
- As a value in a matrix

When ASK and TELL are working correctly, it is possible to remain on the knowledge level. Advantage: very comfortable user interface. The user has his/her own mental model of the world (statements about the world) and communicates it to the agent (TELL).

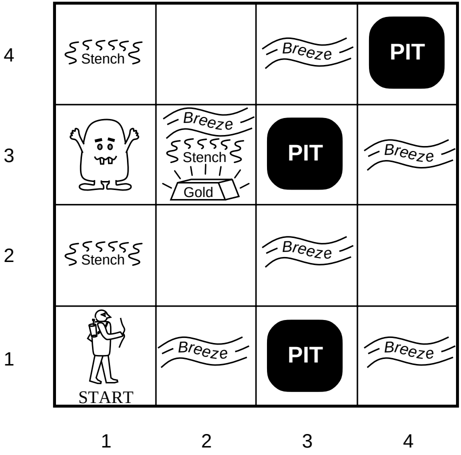
A Knowledge-Based Agent

A knowledge-based agent uses its knowledge base to

- represent its background knowledge
- store its observations
- store its executed actions
- ... derive actions

```
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 ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

The Wumpus World (1): Illustration



This is just one sample configuration.

The Wumpus World (2)

- A 4×4 grid
- In the square containing the wumpus and in the directly adjacent squares, the agent perceives a stench.
- In the squares adjacent to a pit, the agent perceives a breeze.
- In the square where the gold is, the agent perceives a glitter.
- When the agent walks into a wall, it perceives a bump.
- When the wumpus is killed, its scream is heard everywhere.
- Percepts are represented as a 5-tuple, e.g.,

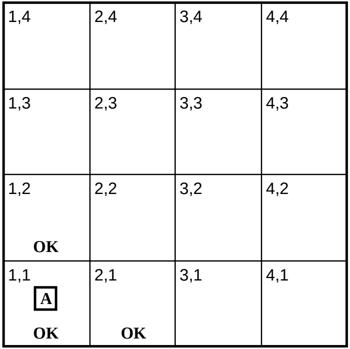
$$[Stench, Breeze, Glitter, None, None]$$

means that it stinks, there is a breeze and a glitter, but no bump and no scream. The agent *cannot* perceive its own location, cannot look in adjacent square.

The Wumpus World (3)

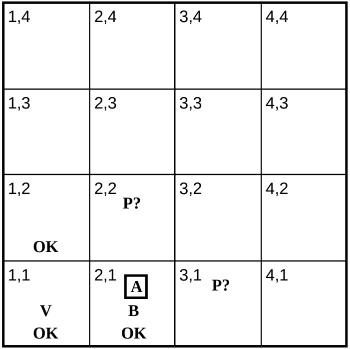
- Actions: Go forward, turn right by 90°, turn left by 90°, pick up an object in the same square (grab), shoot (there is only one arrow), leave the cave (only works in square [1,1]).
- The agent dies if it falls down a pit or meets a live wumpus.
- Initial situation: The agent is in square [1,1] facing east. Somewhere exists a wumpus, a pile of gold and 3 pits.
- Goal: Find the gold and leave the cave.

[1,2] and [2,1] are safe:



(a)

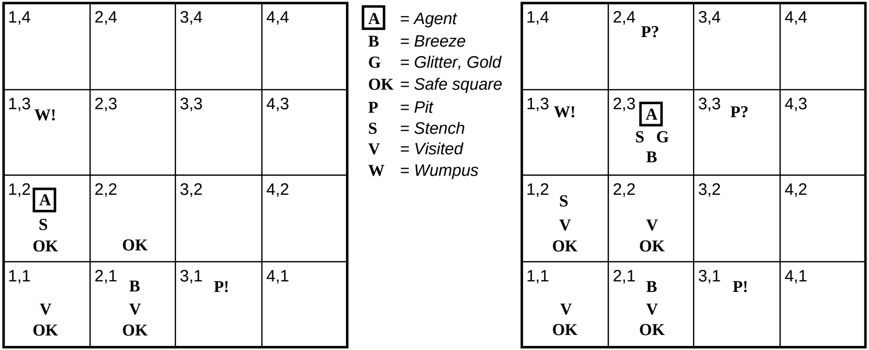
A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*



(b)

The Wumpus World (5)

The wumpus is in [1,3]!



(a) (b)

Syntax and Semantics

- Knowledge bases consist of sentences
- Sentences are expressed according to the syntax of the representation language
  - Syntax specifies all the sentences that are well-formed
  - E.g., in ordinary arithmetic, syntax is pretty clear:

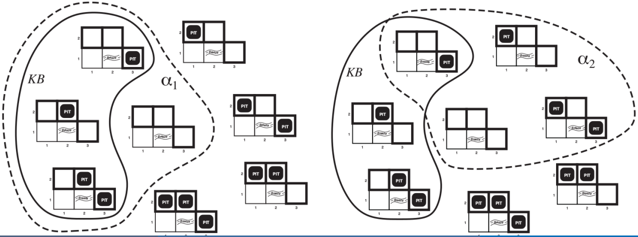
    $x + y = 4$ is a well-formed sentence
    $x4y+ =$ is not a well-formed sentence
- A logic also defines the semantics or meaning of sentences
  - Defines the truth of a sentence with respect to each possible world
  - E.g., specifies that the sentence $x + y = 4$ is true in a world in which $x = 2$ and $y = 2$, but not in a world in which $x = 1$ and $y = 1$

Logical Entailment

- If a sentence α is true in a possible world m, we say that m satisfies α or m is a model of α
- We denote the set of all models of α by $M(\alpha)$
- Logical entailment:
  - When does a sentence β logically follow from another sentence α?
    - in symbols $\alpha \models \beta$
  - $\alpha \models \beta$ if and only if (iff) in every model in which α is true, β is also true
    - I.e., $\alpha \models \beta$ iff $M(\alpha) \subseteq M(\beta)$
    - α is a *stronger* assertion than β; it rules out *more* possible worlds
  - Example in arithmetic: sentence $x = 0$ entails sentence $xy = 0$
    - $x = 0$ rules out the possible world $\{x = 1, y = 0\}$, whereas $xy = 0$ does not rule out that world

Example in the Wumpus World

- Which worlds are possible after having visited [1,1] (no breeze) and [2,1] (breeze)?
  - all worlds in solid area
- Consider two possible sentences:
  - $\alpha_1 =:$ "There is no pit in [1,2]" (true in models in dashed area below, left)
  - $\alpha_2 =:$ "There is no pit in [2,2]" (true in models in dashed area below, right)
- KB $\models \alpha_1$
  - By inspection: in every model in which KB is true, α_1 is also true
- KB $\not\models \alpha_2$
  - In some models, in which KB is true, α_2 is false



Entailment and Inference

- Logical entailment is the (semantic) relation between models of the KB (or a set of formulae in general) and models of a sentence.
- How can we procedurally generate/derive entailed sentences?
  - Logical entailment: $KB \models \alpha$
  - Inference: we can derive α with an inference method i.
    This is written as: $KB \vdash_i \alpha$
- We'd like to have inference algorithms that derive only sentences that are entailed (soundness) and all of them (completeness)

Declarative Languages

Before a system that is capable of learning, thinking, planning, explaining, . . . can be built, one must find a way to express knowledge.

We need a precise, declarative language.

- Declarative
  - We state what we want to compute, not how
  - System believes P if and only if (iff) it considers P to be true
- Precise: We must know,
  - which symbols represent sentences,
  - what it means for a sentence to be true, and
  - when a sentence follows from other sentences.

One possibility: Propositional Logic

Basics of Propositional Logic (1)

Propositions: The building blocks of propositional logic are indivisible, atomic statements (atomic propositions), e.g.,

- "The block is red", expressed, e.g., by the symbol "B_{red}"
- "The wumpus is in [1,3]", expressed, e.g., by the symbol "$W_{1,3}$"

and the logical connectives "and", "or", and "not", which we can use to build formulae.

Basics of Propositional Logic (2)

We are interested in knowing the following:

- When is a proposition true?
- When does a proposition follow from a knowledge base (KB)?
  - Symbolically: $\mathrm{KB} \models \varphi$
- Can we (syntactically) define the concept of *derivation*,
  - Symbolically: $\mathrm{KB} \vdash \varphi$
- And can we make sure that $\models$ and $\vdash$ are equivalent?

$\rightarrow$ Meaning and implementation of ASK

Syntax of Propositional Logic

Countable alphabet Σ of atomic propositions: P, Q, R, $W_{1,3}$, $\ldots$

Logical formulae:	$P \in \Sigma$	atomic formula
	$\bot$	falseness
	$\top$	truth
	$\neg\varphi$	negation
	$\varphi \wedge \psi$	conjunction
	$\varphi \vee \psi$	disjunction
	$\varphi \Rightarrow \psi$	implication
	$\varphi \Leftrightarrow \psi$	equivalence

Operator precedence: $\neg\ >\ \wedge\ >\ \vee\ > \Rightarrow\ > \Leftrightarrow$. (use brackets when necessary)

Atom: atomic formula
Literal: (possibly negated) atomic formula
Clause: disjunction of literals

Semantics: Intuition

Atomic propositions can be true (T) or false (F).

The truth of a formula follows from the truth of its atomic propositions (truth assignment or interpretation) and the connectives.

Example:

$$(P \vee Q) \wedge R$$

- If P and Q are *false* and R is *true*, the formula is *false*
- If P and R are *true*, the formula is *true* regardless of what Q is.

Semantics: Formally

A truth assignment of the atoms in Σ, or an interpretation I over Σ, is a function

$$I : \Sigma \mapsto \{T, F\}$$

Interpretation I satisfies a formula φ ('$I \models \varphi$'):

$$\begin{array}{lll}
I \models \top & & \\
I \not\models \bot & & \\
I \models P & \text{iff} & P^I = T \\
I \not\models \neg\varphi & \text{iff} & I \models \varphi \\
I \models \varphi \wedge \psi & \text{iff} & I \models \varphi \text{ and } I \models \psi \\
I \models \varphi \vee \psi & \text{iff} & I \models \varphi \text{ or } I \models \psi \\
I \models \varphi \Rightarrow \psi & \text{iff} & \text{if } I \models \varphi\text{, then } I \models \psi \\
I \models \varphi \Leftrightarrow \psi & \text{iff} & \text{if } I \models \varphi \text{ if and only if } I \models \psi
\end{array}$$

I satisfies φ ($I \models \varphi$) or φ is true under I, when $I(\varphi) = T$.
I can be seen as a 'possible world'

Example

$$I : \begin{cases} P \mapsto T \\ Q \mapsto T \\ R \mapsto F \\ S \mapsto F \\ \ldots \end{cases}$$

$$\varphi = ((P \vee Q) \Leftrightarrow (R \vee S)) \wedge (\neg(P \wedge Q) \wedge (R \wedge \neg S))$$

Question: $I \models \varphi$?

Wumpus World in Propositional Logic

Symbols: $B_{1,1}, B_{1,2}, \ldots, B_{2,1}, \ldots, S_{1,1}, \ldots, P_{1,1}, \ldots, W_{1,1}, \ \ldots$
Meaning: $B = Breeze$, $B_{i,j} =$ there is a breeze in (i, j) etc.

Facts and Rules:

R1: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R2: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R3: $B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$
...
F1: $\neg P_{1,1}$
F2: $\neg B_{1,1}$ (no percept in (1,1))
F3: $B_{2,1}$ (percept)
F4: $\neg B_{1,2}$ (no percept)
...

Terminology

An interpretation I is called a model of φ if $I \models \varphi$.

An interpretation is a model of a set of formulae if it satisfies all formulae of the set.

A formula φ is

- satisfiable if there exists I that satisfies φ,
- unsatisfiable if φ is not satisfiable,
- falsifiable if there exists I that doesn't satisfy φ, and
- valid (a tautology) if $I \models \varphi$ holds for all I.

Question for you: how are these related to each other?

Two formulae are

- logically equivalent ($\varphi \equiv \psi$) if $I \models \varphi$ iff $I \models \psi$ holds for all I.

The Truth Table Method

How can we decide if a formula is satisfiable, valid, etc.?

$\rightarrow$ Generate a truth table

Example: Is $\varphi = ((P \vee H) \wedge \neg H) \Rightarrow P$ valid?

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
F	F	F	F	T
F	T	T	F	T
T	F	T	T	T
T	T	T	F	T

Since the formula is true for all possible combinations of truth values (satisfied under all interpretations), φ is valid.

Satisfiability, falsifiability, unsatisfiability likewise.

Logical Implications

Goal: Find an algorithmic way to derive new knowledge out of a knowledge base

1. Transform KB into a standardized representation
2. define rules that syntactically modify formulae while keeping semantic correctness

Normal Forms

- A formula is in conjunctive normal form (CNF) if it consists of a conjunction of disjunctions of literals $l_{i,j}$, i.e., if it has the following form:

$$\bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} l_{i,j} \right)$$

- A formula is in disjunctive normal form (DNF) if it consists of a disjunction of conjunctions of literals:

$$\bigvee_{i=1}^{n} \left(\bigwedge_{j=1}^{m_i} l_{i,j} \right)$$

- For every formula, there exists at least one equivalent formula in CNF and one in DNF.
- A formula in DNF is satisfiable iff one disjunct is satisfiable.
  - Checking satisfiability of DNF formula takes linear time.
- A formula in CNF is valid iff every conjunct is valid.
  - Checking validity of CNF formula takes linear time.

Producing CNF

1. Eliminate $\Rightarrow$ and $\Leftrightarrow$: $\alpha \Rightarrow \beta \rightarrow (\neg\alpha \vee \beta)$ etc.
2. Move $\neg$ inwards: $\neg(\alpha \wedge \beta) \rightarrow (\neg\alpha \vee \neg\beta)$ etc. (De Morgan's laws)
3. Distribute $\vee$ over $\wedge$: $((\alpha \wedge \beta) \vee \gamma) \rightarrow (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
4. Simplify: $\alpha \vee \alpha \rightarrow \alpha$ etc.

The result is a conjunction of disjunctions of literals (CNF)

An analogous process converts any formula to an equivalent formula in DNF.

- During conversion, formulae can expand *exponentially*.
- Note: Conversion to CNF formula can be done *polynomially* if only satisfiability should be preserved

Logical Implication: Intuition

A set of formulae (a KB) usually provides an incomplete description of the world, i.e., it leaves the truth values of certain propositions open.

Example: KB $= \{(P \vee Q) \wedge (R \vee \neg P) \wedge S\}$ is definitive with respect to S, but leaves P, Q, R open (although they cannot take on arbitrary values).

Models of the KB:

P	Q	R	S
F	T	F	T
F	T	T	T
T	F	T	T
T	T	T	T

In all models of the KB, $Q \vee R$ is true, i.e., $Q \vee R$ follows logically from KB.

Logical Implication: Formal

The formula φ follows logically from a KB if φ is true in all models of the KB (symbolically KB $\models \varphi$):

$$\text{KB} \models \varphi \text{ iff } I \models \varphi \text{ for all models } I \text{ of KB}$$

Note: The $\models$ symbol is a *meta-symbol*

Question: Can we determine KB $\models \varphi$ without considering all interpretations (the truth table method)?

Some properties of logical implication relationships:

- Deduction theorem: KB $\cup \{\varphi\} \models \psi$ iff KB $\models \varphi \Rightarrow \psi$
- Contraposition theorem: KB $\cup \{\varphi\} \models \neg\psi$ iff KB $\cup \{\psi\} \models \neg\varphi$
- Contradiction theorem: KB $\cup \{\varphi\}$ is unsatisfiable iff KB $\models \neg\varphi$

Proof of the Deduction Theorem

Deduction theorem: $\mathrm{KB} \cup \{\varphi\} \models \psi$ iff $\mathrm{KB} \models \varphi \Rightarrow \psi$

"$\Rightarrow$" Assumption: $\mathrm{KB} \cup \{\varphi\} \models \psi$, i.e., every model of $\mathrm{KB} \cup \{\varphi\}$ is also a model of ψ.

Let I be any model of KB. If I is also a model of φ, then it follows that I is also a model of ψ.

This means that I is also a model of $\varphi \Rightarrow \psi$, i.e., $\mathrm{KB} \models \varphi \Rightarrow \psi$.

"$\Leftarrow$" Assumption: $\mathrm{KB} \models \varphi \Rightarrow \psi$. Let I be any model of KB that is also a model of φ, i.e., $I \models \mathrm{KB} \cup \{\varphi\}$.

From the assumption, I is also a model of $\varphi \Rightarrow \psi$ and thereby also of ψ , i.e., $\mathrm{KB} \cup \{\varphi\} \models \psi$.

Proof of the Contraposition Theorem

Contraposition theorem: $\text{KB} \cup \{\varphi\} \models \neg\psi$ iff $\text{KB} \cup \{\psi\} \models \neg\varphi$

$$
\begin{aligned}
& \text{KB} \cup \{\varphi\} \models \neg\psi & \\
& \text{iff KB} \models \varphi \Rightarrow \neg\psi & (1) \\
& \text{iff KB} \models (\neg\varphi \vee \neg\psi) & \\
& \text{iff KB} \models (\neg\psi \vee \neg\varphi) & \\
& \text{iff KB} \models \psi \Rightarrow \neg\varphi & \\
& \text{iff KB} \cup \{\psi\} \models \neg\varphi & (2)
\end{aligned}
$$

Note:
(1) and (2) are applications of the deduction theorem.

Inference Rules, Calculi, and Proofs

We can often derive new formulae from formulae in the KB. These new formulae should follow logically from the syntactical structure of the KB formulae.

Example: If KB $= \{\ldots, (\varphi \Rightarrow \psi), \ldots, \varphi, \ldots\}$ then ψ is a logical consequence of KB.

$\rightarrow$ Inference rules, e.g., $\dfrac{\varphi, \varphi \Rightarrow \psi}{\psi}$.

Calculus: Set of inference rules (potentially including so-called logical axioms).

Proof step: Application of an inference rule on a set of formulae.

Proof: Sequence of proof steps where every newly-derived formula is added, and in the last step, the goal formula is produced.

Soundness and Completeness

In the case where in the calculus C there is a proof for a formula φ, we write

$$\text{KB} \vdash_C \varphi$$

(optionally without subscript C).

A calculus C is sound (or correct) if all formulae that are derivable from a KB actually follow logically.

$$\text{KB} \vdash_C \varphi \text{ implies } \text{KB} \models \varphi$$

This normally follows from the soundness of the inference rules and the logical axioms.

A calculus is complete if every formula that follows logically from the KB is also derivable with C from the KB:

$$\text{KB} \models \varphi \text{ implies } \text{KB} \vdash_C \varphi$$

Resolution: Idea

We want a way to derive new formulae that does not depend on testing every interpretation.

Idea: To prove that KB $\models \varphi$, we can prove that KB $\cup \{\neg\varphi\}$ is unsatisfiable (contradiction theorem). Therefore, in the following we attempt to show that a set of formulae is unsatisfiable.

Condition: All formulae must be in CNF.

However: In most cases, the formulae are close to CNF (and there exists a fast satisfiability-preserving transformation - Theoretical Computer Science course).

Nevertheless: In the worst case, this derivation process requires an exponential amount of time (this is, however, probably unavoidable).

Resolution: Representation

Assumption: All formulae in the KB are in CNF.

Equivalently, we can assume that the KB is a *set of clauses*. E.g.: Replace $\{(P \vee Q) \wedge (R \vee \neg P) \wedge S\}$ by $\{\{P, Q\}, \{R, \neg P\}, \{S\}\}$

Due to commutativity, associativity, and idempotence of $\vee$, clauses can also be understood as sets of literals. The empty set of literals is denoted by $\square$.

Set of clauses: Δ

Set of literals: C, D

Literal: l

Negation of a literal: $\bar{l}$

An interpretation I satisfies C iff there exists $l \in C$ such that $I \models l$. I satisfies Δ if for all $C \in \Delta : I \models C$, i.e., $I \not\models \square$, $I \not\models \{\square\}$, for all I.

The Resolution Rule

$$\frac{C_1 \dot\cup \{l\}, C_2 \dot\cup \{\bar{l}\}}{C_1 \cup C_2}$$

$C_1 \cup C_2$ are called resolvents of the parent clauses $C_1 \dot\cup \{l\}$ and $C_2 \dot\cup \{\bar{l}\}$. l and $\bar{l}$ are the resolution literals.

Example: $\{a, b, \neg c\}$ resolves with $\{a, d, c\}$ to $\{a, b, d\}$.

Note: The resolvent is not equivalent to the parent clauses, but it follows from them!

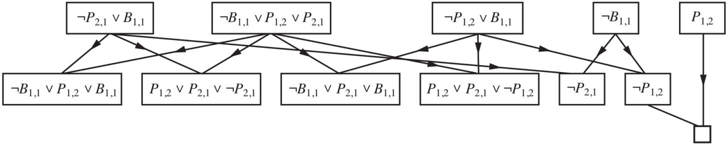
Notation: $R(\Delta) = \Delta \cup \{C \mid C \text{ is a resolvent of two clauses from } \Delta\}$

The Resolution Algorithm

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\,\}$
  loop do
    for each pair of clauses C_i, C_j **in** $clauses$ **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if $resolvents$ contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

Example:



Resolution: Soundness

We say D can be derived from Δ using resolution, i.e.,

$$\Delta \vdash D,$$

if there exist C_1, C_2, C_3, ..., $C_n = D$ such that

$$C_i \in R(\Delta \cup \{C_1, \ldots, C_{i-1}\}), \text{ for } 1 \leq i \leq n.$$

Lemma (soundness) If $\Delta \vdash D$, *then* $\Delta \models D$.

Proof idea: Since all $D \in R(\Delta)$ *follow logically from* Δ, the Lemma results through induction over the length of the derivation.

Resolution: Completeness?

Is resolution also complete, i.e., is

$$\Delta \models \varphi \text{ implies } \Delta \vdash \varphi$$

valid? Not in general. For example, consider:

$$\{\{a, b\}, \{\neg b, c\}\} \models \{a, b, c\} \not\vdash \{a, b, c\}$$

However, it can be shown that resolution is refutation-complete: Δ is unsatisfiable implies $\Delta \vdash \square$

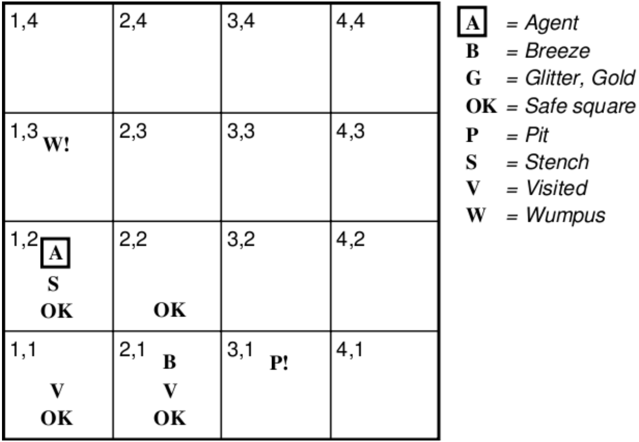
Theorem: Δ is unsatisfiable iff $\Delta \vdash \square$

With the help of the contradiction theorem, we can show that KB $\models \varphi$.
Idea: KB $\cup \{\neg\varphi\}$ is unsatisfiable iff KB $\models \varphi$

Resolution: Overview

- Resolution is a refutation-complete proof process. There are others (Davis-Putnam Procedure, Tableaux Procedure, ...).
- In order to implement the process, a strategy must be developed to determine which resolution steps will be executed and when.
- In the worst case, a resolution proof can take exponential time. This, however, very probably holds for all other proof procedures.
- For CNF formulae in propositional logic, the fastest complete algorithms are indeed based on resolution (combined with backtracking search)

Where is the Wumpus? The Situation



Where is the Wumpus? Knowledge of the Situation

$B = Breeze$, $S = Stench$, $B_{i,j}$ = there is a breeze in (i, j)

$\neg S_{1,1} \quad \neg B_{1,1}$
$\neg S_{2,1} \quad B_{2,1}$
$S_{1,2} \quad \neg B_{1,2}$

Knowledge about the wumpus and smell:

$R_1 : \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$
$R_2 : \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}$
$R_3 : \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}$
$R_4 : S_{1,2} \Rightarrow W_{1,3} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1}$

To show: KB $\models W_{1,3}$

Clausal Representation of the Wumpus World

Situational knowledge:
$\neg S_{1,1}, \neg S_{2,1}, S_{1,2}$

Knowledge of rules:
Knowledge about the wumpus and smell:
$R_1 : S_{1,1} \vee \neg W_{1,1},\ S_{1,1} \vee \neg W_{1,2},\ S_{1,1} \vee \neg W_{2,1}$
$R_2 : \ldots,\ S_{2,1} \vee \neg W_{2,2},\ \ldots$
$R_3 : \ldots$
$R_4 : \neg S_{1,2} \vee W_{1,3} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1}$
$\ldots$

Negated goal formula: $\neg W_{1,3}$

Resolution Proof for the Wumpus World

Resolution:

$\neg W_{1,3},\ \neg S_{1,2} \vee W_{1,3} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1}$
$\rightarrow \neg S_{1,2} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1}$

$S_{1,2},\ \neg S_{1,2} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1}$
$\rightarrow W_{1,2} \vee W_{2,2} \vee W_{1,1}$

$\neg S_{1,1},\ S_{1,1} \vee \neg W_{1,1}$
$\rightarrow \neg W_{1,1}$

$\neg W_{1,1},\ W_{1,2} \vee W_{2,2} \vee W_{1,1}$
$\rightarrow W_{1,2} \vee W_{2,2}$

...

$\neg W_{2,2},\ W_{2,2}$
$\rightarrow \square$

From Knowledge to Action

We can now infer new facts, but how do we translate knowledge into action?

Negative selection: Excludes any provably dangerous actions.

$$A_{1,1} \wedge East_A \wedge W_{2,1} \Rightarrow \neg Forward$$

Positive selection: Only suggests actions that are provably safe.

$$A_{1,1} \wedge East_A \wedge \neg W_{2,1} \Rightarrow Forward$$

Differences?

From the suggestions, we must still select an action.

Problems with Propositional Logic

Although propositional logic suffices to represent the wumpus world, it is rather involved.

Rules must be set up for each square.

$R_1 : \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$

$R_2 : \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}$

$R_3 : \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}$

...

We need a time index for each proposition to represent the validity of the proposition over time $\rightarrow$ further expansion of the rules.

$\rightarrow$ More powerful logics exist, in which we can use object variables.

$\rightarrow$ First-Order Predicate Logic

Summary

- Rational agents require knowledge of their world in order to make rational decisions.
- With the help of a declarative (knowledge-representation) language, this knowledge is represented and stored in a knowledge base.
- We use propositional logic for this (for the time being).
- Formulae of propositional logic can be valid, satisfiable, or unsatisfiable.
- The concept of logical implication is important.
- Logical implication can be mechanized by using an inference calculus $\rightarrow$ resolution.
- Propositional logic quickly becomes impractical when the world becomes too large (or infinite).

Foundations of Artificial Intelligence

8. Satisfiability and Model Construction

DPLL Procedure, Phase Transitions, Local Search, State of the Art

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

May 29, 2019

Motivation

SAT solving is the best available technology for practical solutions to many NP-hard problems

- Formal verification
  - Verification of software
    - Ruling out unintended states (null-pointer exceptions, etc.)
    - Proving that the program computes the right solution
  - Verification of hardware (Pentium bug, etc)
- Practical approach:
  encode into SAT & exploit the rapid progress in SAT solving
  - Solving CSP instances in practice
  - Solving graph coloring problems in practice
  - ...

Contents

Lecture Overview

1. The SAT Problem
2. Davis-Putnam-Logemann-Loveland (DPLL) Procedure
3. “Average” Complexity of the Satisfiability Problem
4. Local Search Procedures
5. State of the Art

Logical deduction vs. satisfiability

Propositional Logic — typical algorithmic questions:

Logical deduction

- Given: A logical theory (set of propositions)
- Question: Does a proposition logically follow from this theory?
- Reduction to unsatisfiability, which is coNP-complete (complementary to NP problems)

Satisfiability of a formula (SAT)

- Given: A logical theory
- Wanted: Model of the theory
- Example: Configurations that fulfill the constraints given in the theory
- Can be "easier" because it is enough to find one model

The Satisfiability Problem (SAT)

Given:

Propositional formula φ in CNF

Wanted:

Model of φ.

or proof, that no such model exists.

SAT and CSP

SAT can be formulated as a Constraint-Satisfaction-Problem ($\rightarrow$ search):

SAT and CSP

SAT can be formulated as a Constraint-Satisfaction-Problem ($\rightarrow$ search):

CSP-Variables $=$ Symbols of the alphabet

Domain of values $= \{T, F\}$

Constraints given by clauses

Lecture Overview

The DPLL algorithm

The DPLL algorithm (Davis, Putnam, Logemann, Loveland, 1962) corresponds to backtracking with inference in CSPs:

Recursive call DPLL (Δ, l) with

- Δ: set of clauses
- l: partial variable assignment

Result: satisfying assignment that extends l
or "unsatisfiable" if no such assignment exists.

First call by DPLL$(\Delta, \emptyset)$

The DPLL algorithm

The DPLL algorithm (Davis, Putnam, Logemann, Loveland, 1962) corresponds to backtracking with inference in CSPs:

Recursive call DPLL (Δ, l) with

- Δ: set of clauses
- l: partial variable assignment

Result: satisfying assignment that extends l
or "unsatisfiable" if no such assignment exists.

First call by DPLL$(\Delta, \emptyset)$

Inference in DPLL:

Simplify: if variable v is assigned a value d, then all clauses containing v are simplified immediately (corresponds to forward checking)

The DPLL algorithm

The DPLL algorithm (Davis, Putnam, Logemann, Loveland, 1962) corresponds to backtracking with inference in CSPs:

- Recursive call DPLL (Δ, l) with
  - Δ: set of clauses
  - l: partial variable assignment
- Result: satisfying assignment that extends l or "unsatisfiable" if no such assignment exists.
- First call by DPLL$(\Delta, \emptyset)$

Inference in DPLL:

- Simplify: if variable v is assigned a value d, then all clauses containing v are simplified immediately (corresponds to forward checking)
- Variables in unit clauses ($=$ clauses with only one variable) are immediately assigned (corresponds to minimum remaining values ordering in CSPs)

The DPLL Procedure

DPLL Function

Given a set of clauses Δ defined over a set of variables Σ, return "satisfiable" if Δ is satisfiable. Otherwise return "unsatisfiable".

1. If $\Delta = \emptyset$ return "satisfiable"
2. If $\square \in \Delta$ return "unsatisfiable"
3. Unit-propagation Rule: If Δ contains a unit-clause C, assign a truth-value to the variable in C that satisfies C, simplify Δ to Δ' and return $\mathrm{DPLL}(\Delta')$.

The DPLL Procedure

DPLL Function

Given a set of clauses Δ defined over a set of variables Σ, return "satisfiable" if Δ is satisfiable. Otherwise return "unsatisfiable".

1. If $\Delta = \emptyset$ return "satisfiable"
2. If $\square \in \Delta$ return "unsatisfiable"
3. Unit-propagation Rule: If Δ contains a unit-clause C, assign a truth-value to the variable in C that satisfies C, simplify Δ to Δ' and return $\mathrm{DPLL}(\Delta')$.
4. Splitting Rule: Select from Σ a variable v which has not been assigned a truth-value. Assign one truth value t to it, simplify Δ to Δ' and call $\mathrm{DPLL}(\Delta')$
   a. If the call returns "satisfiable", then return "satisfiable".
   b. Otherwise assign *the other* truth-value to v in Δ, simplify to Δ'' and return $\mathrm{DPLL}(\Delta'')$.

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

1. Unit-propagation rule: $c \mapsto T$

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

1. Unit-propagation rule: $c \mapsto T$
   $\{\{a, b\}, \{\neg a, \neg b\}, \{a, \neg b\}\}$

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

1. Unit-propagation rule: $c \mapsto T$
   $\{\{a, b\}, \{\neg a, \neg b\}, \{a, \neg b\}\}$
2. Splitting rule:

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

1. Unit-propagation rule: $c \mapsto T$
   $\{\{a, b\}, \{\neg a, \neg b\}, \{a, \neg b\}\}$
2. Splitting rule:

2a. $a \mapsto F$
$\{\{b\}, \{\neg b\}\}$

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

1. Unit-propagation rule: $c \mapsto T$
   $\{\{a, b\}, \{\neg a, \neg b\}, \{a, \neg b\}\}$
2. Splitting rule:

2a. $a \mapsto F$
$\{\{b\}, \{\neg b\}\}$

3a. Unit-propagation rule:
$b \mapsto T$
$\{\square\}$

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

1. Unit-propagation rule: $c \mapsto T$
   $\{\{a, b\}, \{\neg a, \neg b\}, \{a, \neg b\}\}$
2. Splitting rule:

2a. $a \mapsto F$
$\{\{b\}, \{\neg b\}\}$

3a. Unit-propagation rule:
$b \mapsto T$
$\{\square\}$

2b. $a \mapsto T$
$\{\{\neg b\}\}$

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

1. Unit-propagation rule: $c \mapsto T$
   $\{\{a, b\}, \{\neg a, \neg b\}, \{a, \neg b\}\}$
2. Splitting rule:

2a. $a \mapsto F$
$\{\{b\}, \{\neg b\}\}$

3a. Unit-propagation rule:
$b \mapsto T$
$\{\square\}$

2b. $a \mapsto T$
$\{\{\neg b\}\}$

3b. Unit-propagation rule: $b \mapsto F$
$\{\}$

Example (1)

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$$

1. Unit-propagation rule: $c \mapsto T$
   $\{\{a, b\}, \{\neg a, \neg b\}, \{a, \neg b\}\}$
2. Splitting rule:

2a. $a \mapsto F$
$\{\{b\}, \{\neg b\}\}$

3a. Unit-propagation rule:
$b \mapsto T$
$\{\square\}$

2b. $a \mapsto T$
$\{\{\neg b\}\}$

3b. Unit-propagation rule: $b \mapsto F$
$\{\}$

Example (2)

$$\Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$$

Example (2)

$$\Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$$

1. Unit-propagation rule: $d \mapsto T$

Example (2)

$$\Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$$

1. Unit-propagation rule: $d \mapsto T$
   $\{\{a, \neg b, \neg c\}, \{b\}, \{c\}\}$

Example (2)

$$\Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$$

1. Unit-propagation rule: $d \mapsto T$
   $\{\{a, \neg b, \neg c\}, \{b\}, \{c\}\}$
2. Unit-propagation rule: $b \mapsto T$
   $\{\{a, \neg c\}, \{c\}\}$

Example (2)

$$\Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$$

1. Unit-propagation rule: $d \mapsto T$
   $\{\{a, \neg b, \neg c\}, \{b\}, \{c\}\}$
2. Unit-propagation rule: $b \mapsto T$
   $\{\{a, \neg c\}, \{c\}\}$
3. Unit-propagation rule: $c \mapsto T$
   $\{\{a\}\}$

Example (2)

$$\Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$$

1. Unit-propagation rule: $d \mapsto T$
   $\{\{a, \neg b, \neg c\}, \{b\}, \{c\}\}$
2. Unit-propagation rule: $b \mapsto T$
   $\{\{a, \neg c\}, \{c\}\}$
3. Unit-propagation rule: $c \mapsto T$
   $\{\{a\}\}$
4. Unit-propagation rule: $a \mapsto T$
   $\{\}$

Example (2)

$$\Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$$

1. Unit-propagation rule: $d \mapsto T$
   $\{\{a, \neg b, \neg c\}, \{b\}, \{c\}\}$
2. Unit-propagation rule: $b \mapsto T$
   $\{\{a, \neg c\}, \{c\}\}$
3. Unit-propagation rule: $c \mapsto T$
   $\{\{a\}\}$
4. Unit-propagation rule: $a \mapsto T$
   $\{\}$

Properties of DPLL

DPLL is complete, correct, and guaranteed to terminate.

DPLL constructs a model, if one exists.

In general, DPLL requires exponential time (splitting rule!)
$\rightarrow$ *Heuristics* are needed to determine which variable should be instantiated next and which value should be used.

DPLL is polynomial on Horn clauses (see next slides).

In current SAT competitions, DPLL-based procedures have shown the best performance.

DPLL on Horn Clauses (0)

Horn Clauses constitute an important special case, since they require only polynomial runtime of DPLL.

Definition: A Horn clause is a clause with maximally one positive literal
E.g., $\neg A_1 \vee \ldots \vee \neg A_n \vee B$ or $\neg A_1 \vee \ldots \vee \neg A_n$
($n = 0$ is permitted).

Equivalent representation: $\neg A_1 \vee \ldots \vee \neg A_n \vee B \Leftrightarrow \bigwedge_i A_i \Rightarrow B$
$\rightarrow$ Basis of logic programming (e.g., PROLOG)

DPLL on Horn Clauses (1)

Note:

1. The simplifications in DPLL on Horn clauses always generate Horn clauses
2. If the first sequence of applications of the unit propagation rule in DPLL does not lead to termination, a set of Horn clauses without unit clauses is generated
3. A set of Horn clauses without unit clauses and without the empty clause is satisfiable, since

   All clauses have at least one negative literal (since all non-unit clauses have at least two literals, where at most one can be positive (Def. Horn))
   Assigning $false$ to all variables satisfies formula

DPLL on Horn Clauses (2)

4. It follows from 3.:
   a. every time the splitting rule is applied, the current formula is satisfiable
   b. every time, when the wrong decision ($=$ assignment in the splitting rule) is made, this will be immediately detected (e.g., only through unit propagation steps and the derivation of the empty clause).
5. Therefore, the search trees for n variables can only contain a maximum of n nodes, in which the splitting rule is applied (and the tree branches).
6. Therefore, the size of the search tree is only polynomial in n and therefore the running time is also polynomial.

Lecture Overview

How Good is DPLL in the Average Case?

We know that SAT is NP-complete, i.e., in the worst case, it takes exponential time.

This is clearly also true for the DPLL-procedure.
$\rightarrow$ Couldn't we do better in the average case?

For CNF-formulae, in which the probability for a positive appearance, negative appearance and non-appearance in a clause is $1/3$, DPLL needs on average quadratic time (Goldberg 79)!
$\rightarrow$ The probability that these formulae are satisfiable is, however, very high.

Phase Transitions . . .

Conversely, we can, of course, try to identify hard to solve problem instances.

Cheeseman et al. (IJCAI-91) came up with the following plausible conjecture:

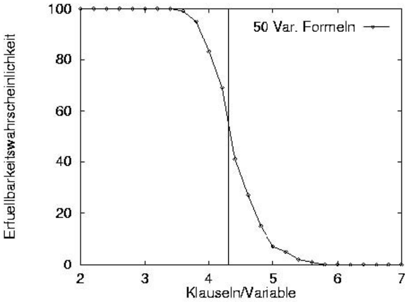
> All NP-complete problems have at least *one order* parameter and the hard to solve problems are around a critical value of this order parameter. This critical value (a phase transition) separates one region from another, such as over-constrained and under-constrained regions of the problem space.

Confirmation for graph coloring and Hamiltonian path . . . , later also for other NP-complete problems.

Phase Transitions with 3-SAT

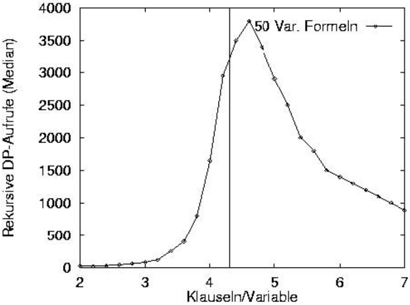
Constant clause length model (Mitchell et al., AAAI-92):
Clause length k is given. Choose variables for every clause k and use the complement with probability 0.5 for each variable.

Phase transition for 3-SAT with a clause/variable ratio of approx. 4.3:



Empirical Difficulty

The Davis-Putnam (DPLL) Procedure shows extreme runtime peaks at the phase transition:



Note: Hard instances can exist even in the regions of the more easily satisfiable/unsatisfiable instances!

Notes on the Phase Transition

When the probability of a solution is close to 1 (under-constrained), there are many solutions, and the first search path of a backtracking search is usually successful.

If the probability of a solution is close to 0 (over-constrained), this fact can usually be determined early in the search.

In the phase transition stage, there are many near successes

$\rightarrow$ (limited) possibility of predicting the difficulty of finding a solution based on the parameters

$\rightarrow$ (search intensive) benchmark problems are located in the phase region (but they have a special structure)

Lecture Overview

Local Search Methods for Solving Logical Problems

In many cases, we are interested in finding a satisfying assignment of variables (example CSP), and we can sacrifice completeness if we can "solve" much larger instances this way.

Standard process for optimization problems: Local Search

- Based on a (random) configuration
- Through local modifications, we hope to produce better configurations

$\rightarrow$ Main problem: local maxima

Dealing with Local Maxima

As a measure of the value of a configuration in a logical problem, we could use the number of satisfied constraints/clauses.

At first glance, local search seems inappropriate, considering that we want to find a global maximum (all constraints/clauses satisfied).

However:

- By restarting and/or injecting noise, we can often escape local maxima.
- Local search can perform very well for SAT solving

A pioneering local search method for SAT: GSAT (1993)

Procedure GSAT

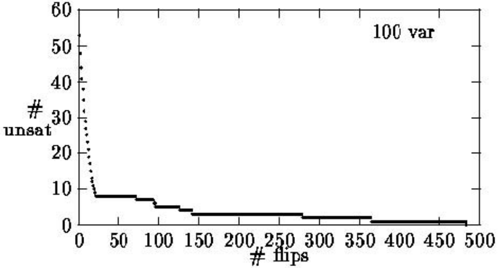
INPUT: a set of clauses α, $\textsc{Max-Flips}$, and $\textsc{Max-Tries}$
OUTPUT: a satisfying truth assignment of α, if found

begin
  for $i := 1$ to $\textsc{Max-Tries}$
    $T :=$ a randomly-generated truth assignment
    for $j := 1$ to $\textsc{Max-Flips}$
      if T satisfies α **then return** T
      $v :=$ a propositional variable such that a change in its
          truth assignment gives the largest increase in
          the number of clauses of α that are satisfied by T
      $T := T$ with the truth assignment of v reversed
    end for
  end for
  return "no satisfying assignment found"
end

The Search Behavior of GSAT

In contrast to many other local search methods, we must also allow sideways movements!

Most time is spent searching on plateaus.



Lecture Overview

1. The SAT Problem
2. Davis-Putnam-Logemann-Loveland (DPLL) Procedure
3. “Average” Complexity of the Satisfiability Problem
4. Local Search Procedures
5. State of the Art

Practical Improvements of DPLL Algorithms

Clause Learning

- Consider an exemplary SAT problem

  26 variables A, . . . , Z

  Amongst many other clauses, we have

  $\{(\neg A, Y, Z)\}, \{(\neg A, \neg Y, Z)\}, \{(\neg A, Y, \neg Z)\}, \{(\neg A, \neg Y, \neg Z)\}$

  We'll branch on variables in lexicographic order and try true first
- What will happen?

Practical Improvements of DPLL Algorithms

Clause Learning

- Consider an exemplary SAT problem

  26 variables A, ..., Z

  Amongst many other clauses, we have

  $\{(\neg A, Y, Z)\}, \{(\neg A, \neg Y, Z)\}, \{(\neg A, Y, \neg Z)\}, \{(\neg A, \neg Y, \neg Z)\}$

  We'll branch on variables in lexicographic order and try true first
- What will happen?

  There is no satisfying assignment to the clauses above when A=T

Practical Improvements of DPLL Algorithms

Clause Learning

- Consider an exemplary SAT problem
  - 26 variables A, . . . , Z
  - Amongst many other clauses, we have
  - $\{(\neg A, Y, Z)\}, \{(\neg A, \neg Y, Z)\}, \{(\neg A, Y, \neg Z)\}, \{(\neg A, \neg Y, \neg Z)\}$
  - We'll branch on variables in lexicographic order and try true first
- What will happen?
  - There is no satisfying assignment to the clauses above when A=T
  - For each assignment to variables B, . . . , X, we'll have to rediscover this fact
  - Rather: reason about the variables that led to a conflict: A, Y and Z
  - We can 'learn" (here: logically infer) a new clause: $\neg A$
  - Leads to conflict-directed clause learning (CDCL)

Practical Improvements of DPLL Algorithms

Clause Learning

- Consider an exemplary SAT problem
  - 26 variables A, . . . , Z
  - Amongst many other clauses, we have
  - $\{(\neg A, Y, Z)\}, \{(\neg A, \neg Y, Z)\}, \{(\neg A, Y, \neg Z)\}, \{(\neg A, \neg Y, \neg Z)\}$
  - We'll branch on variables in lexicographic order and try true first
- What will happen?
  - There is no satisfying assignment to the clauses above when A=T
  - For each assignment to variables B, . . . , X, we'll have to rediscover this fact
  - Rather: reason about the variables that led to a conflict: A, Y and Z
  - We can ‘learn” (here: logically infer) a new clause: $\neg A$
  - Leads to conflict-directed clause learning (CDCL)

Intelligent Backjumping

- Closely related to clause learning

Practical Improvements of SAT Algorithms

Both for DPLL/CDCL algorithms and local search algorithms

- Randomization and restarts
- Efficient data structures, indexing, etc
- Engineering ingenious heuristics

Practical Improvements of SAT Algorithms

Both for DPLL/CDCL algorithms and local search algorithms

- Randomization and restarts
- Efficient data structures, indexing, etc
- Engineering ingenious heuristics

- Meta-algorithmic advances
  - Automated parameter tuning and algorithm configuration
  - Selection of the best-fitting algorithm based on instance characteristics
  - Selection of the best-fitting parameters based on instance characteristics
  - Use of machine learning to pinpoint what factors most affects performance

The Current State of the Art

SAT competitions since beginning of the 90s

Current SAT competitions (`http://www.satcompetition.org/`):

- Largest "industrial" instances: $> 10{,}000{,}000$ variables

Complete solvers dominate handcrafted and industrial tracks

Incomplete local search solvers best on random satisfiable instances

The Current State of the Art

SAT competitions since beginning of the 90s

Current SAT competitions (`http://www.satcompetition.org/`):

- Largest "industrial" instances: $>$ 10,000,000 variables

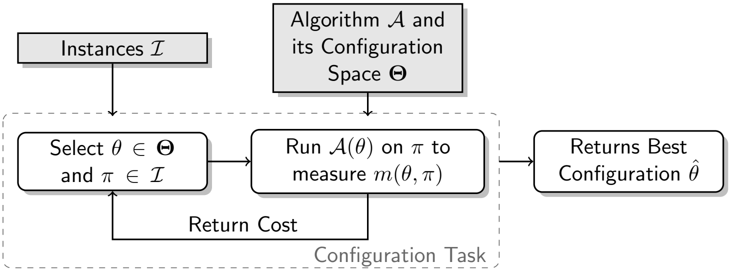
Complete solvers dominate handcrafted and industrial tracks

Incomplete local search solvers best on random satisfiable instances

Best solvers use meta-algorithmic methods, such as algorithm configuration, selection, etc.

- We thus discuss these briefly next

Algorithm Configuration

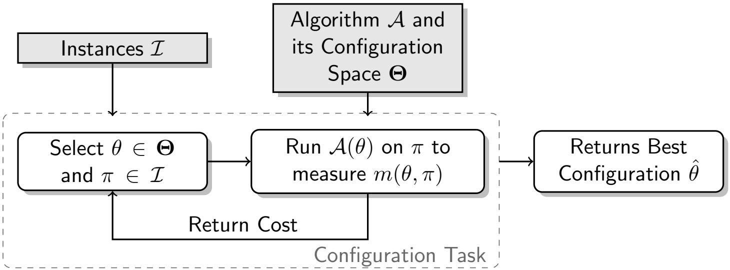


Definition: algorithm configuration

Given:

- a parameterized algorithm $\mathcal{A}$ with possible parameter settings $\boldsymbol{\Theta}$;
- a distribution $\mathcal{D}$ over problem instances with domain $\mathcal{I}$; and

Algorithm Configuration



Definition: algorithm configuration

Given:

- a parameterized algorithm $\mathcal{A}$ with possible parameter settings $\mathbf{\Theta}$;
- a distribution $\mathcal{D}$ over problem instances with domain $\mathcal{I}$; and
- a cost metric $m : \mathbf{\Theta} \times \mathcal{I} \to \mathbb{R}$,

Find: $\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \mathbb{E}_{\pi \sim \mathcal{D}}(m(\theta, \pi))$.

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

Formal verification

- Software verification [Babić & Hu; CAV '07]
- Hardware verification (Bounded model checking) [Zarpas; SAT '05]
- Recent progress based on SAT solvers

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

Formal verification

- Software verification [Babić & Hu; CAV '07]
- Hardware verification (Bounded model checking) [Zarpas; SAT '05]
- Recent progress based on SAT solvers

CDCL solver for SAT-based verification

- SPEAR, developed by Domagoj Babić at UBC
- 26 parameters, 8.34×10^{17} configurations

Configuration of a SAT Solver for Verification [Hutter et al, 2007]

Ran algorithm configuration method ParamILS: 2 days on 10 machines

– On a training set from each benchmark

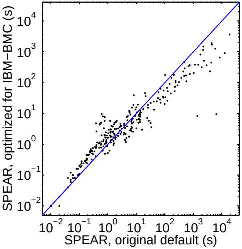
Configuration of a SAT Solver for Verification [Hutter et al, 2007]

Ran algorithm configuration method ParamILS: 2 days on 10 machines
– On a training set from each benchmark

Compared to manually-engineered default
– 1 week of performance tuning
– Competitive with the state of the art
– Comparison on unseen test instances

Ran algorithm configuration method ParamILS: 2 days on 10 machines
– On a training set from each benchmark

Compared to manually-engineered default
– 1 week of performance tuning
– Competitive with the state of the art
– Comparison on unseen test instances



4.5-fold speedup
on hardware verification

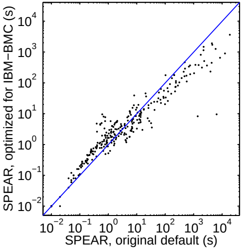
Configuration of a SAT Solver for Verification [Hutter et al, 2007]

Ran algorithm configuration method ParamILS: 2 days on 10 machines

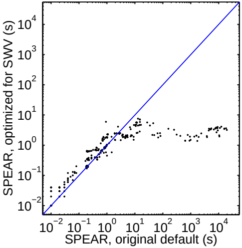
– On a training set from each benchmark

Compared to manually-engineered default

– 1 week of performance tuning
– Competitive with the state of the art
– Comparison on unseen test instances



4.5-fold speedup
on hardware verification



500-fold speedup ⇝ won category QF_BV in 2007 SMT competition

Algorithm Selection

Definition: algorithm selection

Given

a set $\mathcal{I}$ of problem instances,

a portfolio of algorithms $\mathcal{P}$,

and a cost metric $m : \mathcal{P} \times \mathcal{I} \to \mathbb{R}$,

the per-instance algorithm selection problem is to find a mapping $s : \mathcal{I} \to \mathcal{P}$ that optimizes $\sum_{\pi \in \mathcal{I}} m(s(\pi), \pi)$, the sum of cost measures achieved by running the selected algorithm $s(\pi)$ for instance π.

Algorithm Selection

Definition: algorithm selection

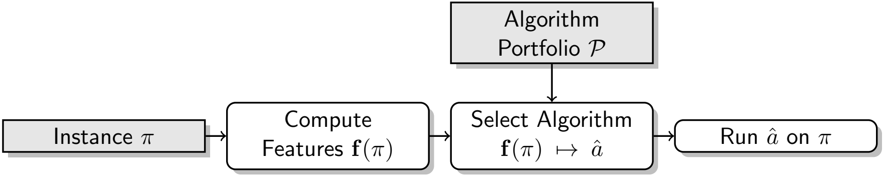
Given

a set $\mathcal{I}$ of problem instances,

a portfolio of algorithms $\mathcal{P}$,

and a cost metric $m : \mathcal{P} \times \mathcal{I} \rightarrow \mathbb{R}$,

the per-instance algorithm selection problem is to find a mapping $s : \mathcal{I} \rightarrow \mathcal{P}$ that optimizes $\sum_{\pi \in \mathcal{I}} m(s(\pi), \pi)$, the sum of cost measures achieved by running the selected algorithm $s(\pi)$ for instance π.



Example SAT Challenge 2012

Rank	RiG	Solver	#solved
–	–	Virtual Best Solver (VBS)	568
1	1	SATzilla2012 APP	531
2	2	SATzilla2012 ALL	515
3	1	Industrial SAT Solver	499
–	–	lingeling (SAT Competition 2011 Bronze)	488
4	2	interactSAT	480
5	1	glucose	475
6	2	SINN	472
7	3	ZENN	468
8	4	Lingeling	467
9	5	linge_dyphase	458
10	6	simpsat	453

The VBS is the best possible performance
of an algorithm selection system.
(pink: algorithm selectors, blue: portfolios, green: single-engine solvers)

Automated construction of portfolios from a single algorithm

Algorithm Configuration

- Strength: find a single configuration with strong performance for a given cost metric
- Weakness: for heterogeneous instance sets, there is often no configuration that performs great for all instances

Automated construction of portfolios from a single algorithm

- Algorithm Configuration
  - Strength: find a single configuration with strong performance for a given cost metric
  - Weakness: for heterogeneous instance sets, there is often no configuration that performs great for all instances
- Algorithm Selection
  - Strength: for heterogeneous instance sets, pick the right algorithm from a set
  - Weakness: the set to choose from typically only contains a few algorithms

Automated construction of portfolios from a single algorithm

- Algorithm Configuration
  - Strength: find a single configuration with strong performance for a given cost metric
  - Weakness: for heterogeneous instance sets, there is often no configuration that performs great for all instances
- Algorithm Selection
  - Strength: for heterogeneous instance sets, pick the right algorithm from a set
  - Weakness: the set to choose from typically only contains a few algorithms
- Putting the two together
  - Use algorithm configuration to determine useful configurations
  - Use algorithm selection to select from them based on instance characteristics

Automated construction of portfolios from a single algorithm: Hydra [Xu et al. 2010, 2011]

Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, starting with $\mathcal{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathcal{P}$

Automated construction of portfolios from a single algorithm: Hydra [Xu et al. 2010, 2011]

Idea

Iteratively add configurations to a portfolio $\mathcal{P}$, starting with $\mathcal{P} = \emptyset$

In each iteration, determine configuration that is complementary to $\mathcal{P}$

Maximize marginal contribution of configuration θ to current portfolio $\mathcal{P}$:

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$

Automated construction of portfolios from a single algorithm: Hydra [Xu et al. 2010, 2011]

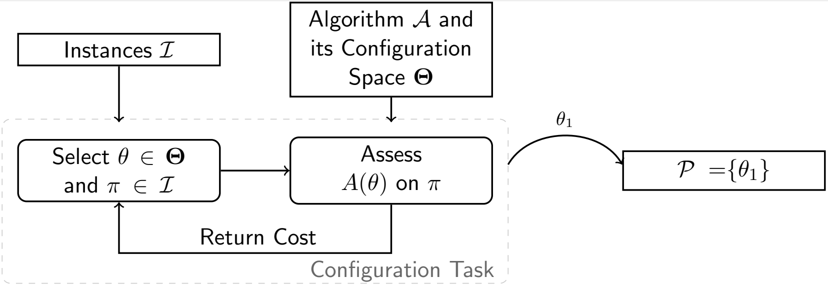
Idea

Iteratively add configurations to a portfolio $\mathcal{P}$, starting with $\mathcal{P} = \emptyset$

In each iteration, determine configuration that is complementary to $\mathcal{P}$

Maximize marginal contribution of configuration θ to current portfolio $\mathcal{P}$:

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$



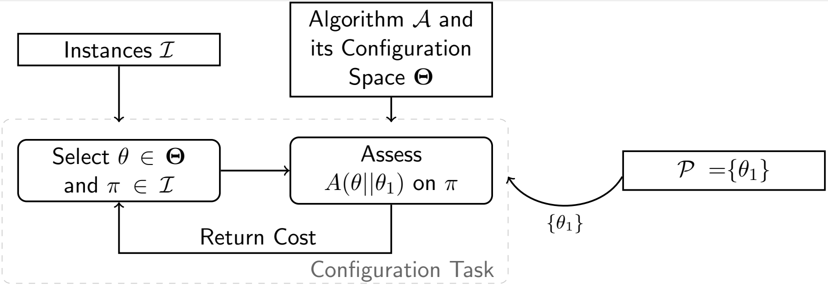
Automated construction of portfolios from a single algorithm: Hydra [Xu et al. 2010, 2011]

Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, starting with $\mathcal{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathcal{P}$

Maximize marginal contribution of configuration θ to current portfolio $\mathcal{P}$:

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$



Automated construction of portfolios from a single algorithm: Hydra [Xu et al. 2010, 2011]

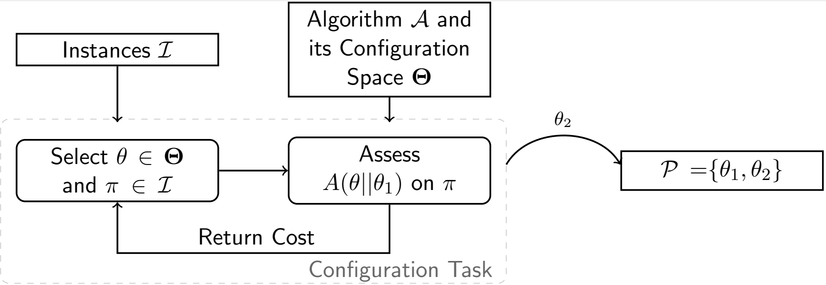
Idea

Iteratively add configurations to a portfolio $\mathcal{P}$, starting with $\mathcal{P} = \emptyset$

In each iteration, determine configuration that is complementary to $\mathcal{P}$

Maximize marginal contribution of configuration θ to current portfolio $\mathcal{P}$:

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$



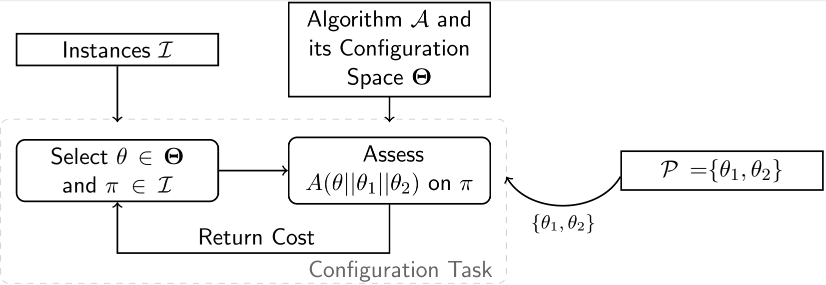
Automated construction of portfolios from a single algorithm: Hydra [Xu et al. 2010, 2011]

Idea

- Iteratively add configurations to a portfolio $\mathcal{P}$, starting with $\mathcal{P} = \emptyset$
- In each iteration, determine configuration that is complementary to $\mathcal{P}$

Maximize marginal contribution of configuration θ to current portfolio $\mathcal{P}$:

$$m(\mathcal{P}) - m(\mathcal{P} \cup \{\theta\})$$



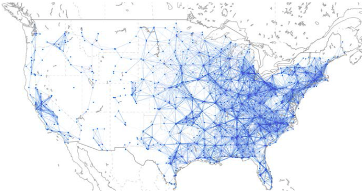
A Large-Scale Application of SAT Technology

FCC Spectrum Auction

- Wireless frequency spectra: demand increases
- US Federal Communications Commission (FCC) held 13-month auction

Key Computational Problem: feasibility testing based on interference constraints

- a hard graph colouring problem
- 2991 stations (nodes) &
- 2.7 million interference constraints
- Need to solve many different instances
- More instances solved: higher revenue



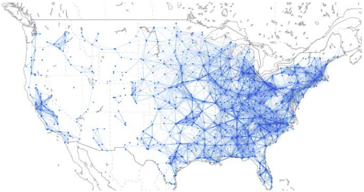
A Large-Scale Application of SAT Technology

FCC Spectrum Auction

- Wireless frequency spectra: demand increases
- US Federal Communications Commission (FCC) held 13-month auction

Key Computational Problem: feasibility testing based on interference constraints

- a hard graph colouring problem
- 2991 stations (nodes) &
- 2.7 million interference constraints
- Need to solve many different instances
- More instances solved: higher revenue



Best solution: based on SAT solving & meta-algorithmic improvements

- CDCL Solver Clasp, optimized with algo. configuration method SMAC
- Instance-specific configuration with Hydra (using SATzilla for algo. selection)

Concluding Remarks

DPLL: combines simplification, unit-propagation and backtracking

- Very efficient implementation techniques
- Good branching heuristics
- Clause learning

Incomplete randomized SAT-solvers

- Perform best on random satisfiable problem instances

State of the art

- Typically obtained by automatic algorithm configuration & selection

Foundations of Artificial Intelligence

9. Predicate Logic

Syntax and Semantics, Reduction to Propositional Logic

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

June 5, 2019

Motivation

We can already do a lot with propositional logic. It is, however, annoying that there is no structure in the atomic propositions.

Example:

"All blocks are red"
"There is a block A"
It should follow that "A is red"

But propositional logic cannot handle this.

Idea: We introduce individual variables, predicates, functions, ...

$\rightarrow$ First-Order Predicate Logic (PL1)

Contents

Lecture Overview

The Alphabet of First-Order Predicate Logic

Symbols:

- Operators: $\neg$, $\vee$, $\wedge$, $\forall$, $\exists$, $=$
- Variables: x, x_1, x_2, $\ldots$, x', x'', $\ldots$, y, $\ldots$, z, $\ldots$
- Brackets: $()$, $[]$, $\big()$, $\big[]$
- Function symbols (e.g., $weight()$, $color()$)
- Predicate symbols (e.g., $Block()$, $Red()$)
- Predicate and function symbols have an arity (number of arguments).
  0-ary predicate $=$ propositional logic atoms: $P, Q, R, \ldots$
  0-ary function $=$ constants: $a, b, c, \ldots$
- We assume a countable set of predicates and functions of any arity.
- "$=$" is usually not considered a predicate, but a logical symbol

The Grammar of First-Order Predicate Logic (1)

Terms (represent objects):

1. Every variable is a term.
2. If $t_1, t_2, \ldots, t_n$ are terms and f is an n-ary function, then

$$f(t_1, t_2, \ldots, t_n)$$

   is also a term.

   Terms without variables: ground terms.

Atomic Formulae (represent statements about objects)

1. If $t_1, t_2, \ldots, t_n$ are terms and P is an n-ary predicate, then $P(t_1, t_2, \ldots, t_n)$ is an atomic formula.
2. If t_1 and t_2 are terms, then $t_1 = t_2$ is an atomic formula.

   Atomic formulae without variables: ground atoms.

The Grammar of First-Order Predicate Logic (2)

Formulae:

1. Every atomic formula is a formula.
2. If φ and ψ are formulae and x is a variable, then

   $$\neg\varphi,\ \varphi \wedge \psi,\ \varphi \vee \psi,\ \varphi \Rightarrow \psi,\ \varphi \Leftrightarrow \psi,\ \exists x\varphi \text{ and } \forall x\varphi$$

   are also formulae.
   $\forall$, $\exists$ are as strongly binding as $\neg$.

Propositional logic is part of the PL1 language:

1. Atomic formulae: only 0-ary predicates
2. Neither variables nor quantifiers.

Alternative Notation

Here	Elsewhere
$\neg\varphi$	$\sim\varphi \quad \overline{\varphi}$
$\varphi \wedge \psi$	$\varphi \& \psi \quad \varphi \bullet \psi \quad \varphi, \psi$
$\varphi \vee \psi$	$\varphi \vert \psi \quad \varphi ; \psi \quad \varphi + \psi$
$\varphi \Rightarrow \psi$	$\varphi \rightarrow \psi \quad \varphi \supset \psi$
$\varphi \Leftrightarrow \psi$	$\varphi \leftrightarrow \psi \quad \varphi \equiv \psi$
$\forall x \varphi$	$(\forall x)\varphi \wedge x\varphi$
$\exists x \varphi$	$(\exists x)\varphi \vee x\varphi$

Meaning of PL1-Formulae

Our example: $\forall x[Block(x) \Rightarrow Red(x)]$, $Block(a)$

For all objects x: If x is a block, then x is red and a is a block.

Generally:

- Terms are interpreted as objects.
- Universally-quantified variables denote all objects in the universe.
- Existentially-quantified variables represent one of the objects in the universe (made true by the quantified expression).
- Predicates represent subsets of the universe.

Similar to propositional logic, we define interpretations, satisfiability, models, validity, . . .

Semantics of PL1-Logic

Interpretation: $I = \langle D, \bullet^I \rangle$ where D is an arbitrary, non-empty set and $\bullet^I$ is a function that

- maps n-ary function symbols to functions over D:
  $f^I \in [D^n \mapsto D]$
- maps individual constants to elements of D:
  $a^I \in D$
- maps n-ary predicate symbols to relations over D:
  $P^I \subseteq D^n$

Interpretation of ground terms:

$$(f(t_1, \dots, t_n))^I = f^I(t_1^I, \dots, t_n^I)$$

Satisfaction of ground atoms $P(t_1, \dots, t_n)$:

$$I \models P(t_1, \dots, t_n) \text{ iff } \langle t_1^I, \dots, t_n^I \rangle \in P^I$$

Example (1)

$$
\begin{aligned}
D &= \{d_1, \ldots, d_n \mid n > 1\} \\
a^I &= d_1 \\
b^I &= d_2 \\
c^I &= \ldots \\
Block^I &= \{d_1\} \\
Red^I &= D \\
I &\models Red(b) \\
I &\not\models Block(b)
\end{aligned}
$$

Example (2)

$$
\begin{aligned}
D &= \{1, 2, 3, \ldots\} \\
1^I &= 1 \\
2^I &= 2 \\
&\ldots \\
Even^I &= \{2, 4, 6, \ldots\} \\
succ^I &= \{(1 \mapsto 2), (2 \mapsto 3), \ldots\} \\
I &\models Even(2) \\
I &\not\models Even(succ(2))
\end{aligned}
$$

Semantics of PL1: Variable Assignment

Set of all variables V. Function $\alpha : V \mapsto D$

Notation: $\alpha[x/d]$ is the same as α apart from point x.

For $x : \alpha[x/d](x) = d$.

Interpretation of terms under I, α:

$$x^{I,\alpha} = \alpha(x)$$
$$a^{I,\alpha} = a^I$$
$$(f(t_1, \ldots, t_n))^{I,\alpha} = f^I(t_1^{I,\alpha}, \ldots, t_n^{I,\alpha})$$

Satisfaction of atomic formulae:

$$I, \alpha \models P(t_1, \ldots, t_n) \text{ iff } \langle t_1^{I,\alpha}, \ldots, t_n^{I,\alpha} \rangle \in P^I$$

Example

$$
\begin{aligned}
Block^I &= \{d_1\} \\
Red^I &= D
\end{aligned}
$$

$$
\begin{aligned}
\alpha &= \{(x \mapsto d_1), (y \mapsto d_2)\} \\
I, \alpha &\models Red(x) \\
I, \alpha[y/d_1] &\models Block(y)
\end{aligned}
$$

Semantics of PL1: Satisfiability

A formula φ is satisfied by an interpretation I and a variable assignment α, i.e., $I, \alpha \models \varphi$:

$$I, \alpha \models \top$$
$$I, \alpha \not\models \bot$$
$$I, \alpha \models \neg\varphi \text{ iff } I, \alpha \not\models \varphi$$
$$\dots$$

and all other propositional rules as well as

$$
\begin{array}{lll}
I, \alpha \models P(t_1, \dots, t_n) & \text{iff} & \langle t_1^{I,\alpha}, \dots, t_n^{I,\alpha} \rangle \in P^I \\
I, \alpha \models \forall x \varphi & \text{iff} & \text{for all } d \in D,\ I, \alpha[x/d] \models \varphi \\
I, \alpha \models \exists x \varphi & \text{iff} & \text{there exists a } d \in D \text{ with } I, \alpha[x/d] \models \varphi
\end{array}
$$

Example

$$
\begin{aligned}
D &= \{d_1, \ldots, d_n \mid n > 1\} \\
a^I &= d_1 \\
b^I &= d_2 \\
Block^I &= \{d_1\} \\
Red^I &= D \\
\alpha &= \{(x \mapsto d_1), (y \mapsto d_2)\}
\end{aligned}
$$

Questions:

1. $I, \alpha \models Block(b) \vee \neg Block(b)$?
2. $I, \alpha \models Block(x) \Rightarrow (Block(x) \vee \neg Block(y))$?
3. $I, \alpha \models Block(a) \wedge Block(b)$?
4. $I, \alpha \models \forall x (Block(x) \Rightarrow Red(x))$?

Free and Bound Variables

$$\forall x\big[R(\boxed{y},\boxed{z}) \land \exists y\big((\neg P(y,x) \lor R(y,\boxed{z})\big)\big]$$

The boxed appearances of y and z are free. All other appearances of x,y,z are bound.

Formulae with no free variables are called closed formulae or sentences. We form theories from closed formulae.

Note: With closed formulae, the concepts *logical equivalence, satisfiability, and implication, etc.* are not dependent on the variable assignment α (i.e., we can always ignore all variable assignments).

With closed formulae, α can be left out on the left side of the model relationship symbol:

$$I \models \varphi$$

Terminology

An interpretation I is called a model of φ under α if

$$I, \alpha \models \varphi$$

A PL1 formula φ can, as in propositional logic, be satisfiable, unsatisfiable, falsifiable, or valid.

Analogously, two formulae are logically equivalent ($\varphi \equiv \psi$) if for all I, α:

$$I, \alpha \models \varphi \text{ iff } I, \alpha \models \psi$$

Note: $P(x) \not\equiv P(y)$!

Logical Implication is also analogous to propositional logic.

Question: How can we define derivation?

Lecture Overview

1. Syntax and Semantics
2. Reduction to Propositional Theories
3. Summary

Derivation in PL1: Possible Approaches

- We now know the semantics of PL1. How can we do inference in PL1?
- One way: Normalization + Skolemization + Resolution with Unification
- Alternative: Reduction to propostional logic by instantiation based on the so-called Herbrand Universe (all possible terms) $\rightsquigarrow$ infinite propositional theories
- It turns out that logical implication in PL1 is undecidable!
- Simple way for special case: If the number of objects is finite, instantiate all variables by possible objects (in fact, often used in AI systems, e.g. planning or ASP)

Finite Universes

- Let us assume that we only want to talk about a finite number of objects.
- Domain closure axiom (DCA):
$$\forall x[x = c_1 \lor x = c_2 \lor \ldots \lor x = c_n]$$
- Often one also assumes that different names denote different objects (unique name assumption/axiom or UNA):
$$\bigwedge_{i \neq j}[c_i \neq c_j]$$

$\rightarrow$ Only important when counting or using $\neq$ or $=$ as a predicate.

- Elimate quantification by instantiating all variables with all possible values.

Instantiation

- Notation: if φ is a formula, then $\varphi[x/a]$ is the formula with all free occurences of x replaced by a.
- Universally quantified formulas are replaced by a conjunction of formulas with the variable instantiated to all possible values (from DCA):
$$\forall x \varphi \rightsquigarrow \bigwedge_i \varphi[x/c_i]$$
- Existentially quantified variables are replaced by a disjunction of formulas with the variable instantiated to all possible values (from DCA): $$\exists x \varphi \rightsquigarrow \bigvee_i \varphi[x/c_i]$$
- Note: does blow up the formulas exponentially in the arity of the predicates!

Example

$$
\begin{array}{ll}
\forall x & (Block(x) \Rightarrow Red(x)) \\
\forall x & (x = a \lor x = b \lor x = c) \\
\rightsquigarrow & \\
& (Block(a) \Rightarrow Red(a)) \land \\
& (Block(b) \Rightarrow Red(b)) \land \\
& (Block(c) \Rightarrow Red(c))
\end{array}
$$

Lecture Overview

1. Syntax and Semantics

2. Reduction to Propositional Theories

3. Summary

Summary

- PL1 makes it possible to structure statements, thereby giving us considerably more expressive power than propositional logic.
- Logical implication in PL1 is undecidable.
- If we only reason over a finite universe, PL1 can be reduced to propositional logic over finite theories (but the reduction is exponential in the arity of the predicates).

Foundations of Artificial Intelligence

10. Action Planning

Solving Logically Specified Problems using a General Problem Solver

Joschka Boedecker and Wolfram Burgard and Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

June 7, 2019

Contents

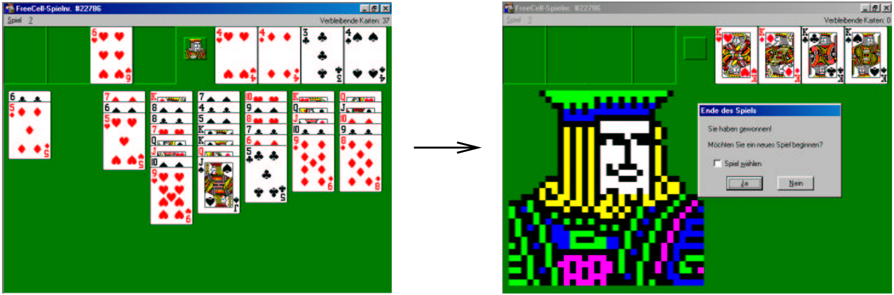
Lecture Overview

1. What is Action Planning?
2. Planning Formalisms
3. Basic Planning Algorithms
4. Computational Complexity
5. Current Algorithmic Approaches
6. Current Trends in Planning
7. Summary

Planning

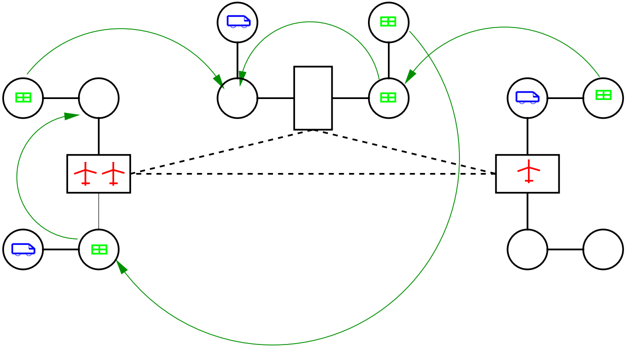
- Planning is the art and practive of thinking before acting [Haslum]
- Planning is the process of generating (possibly partial) representations of future behavior prior to the use of such plans to constrain or control that behavior.
- The outcome is usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents.

Given a current state, a set of possible actions, a specification of the goal conditions, which plan transforms the current state into a goal state?



Another Planning Task: *Logistics*

Given a road map, and a number of trucks and airplanes, make a plan to transport objects from their start to their goal destinations.



Action Planning is not . . .

- Problem solving by search, where we describe a problem by a state space and then implement a program to search through this space
  - in action planning, we specify the problem declaratively (using logic) and then solve it by a general planning algorithm
- Program synthesis, where we generate programs from specifications or examples
  - in action planning we want to solve just one instance and we have only very simple action composition (i.e., sequencing, perhaps conditional and iteration)
- Scheduling, where all jobs are known in advance and we only have to fix time intervals and machines
  - instead we have to find the right actions and to sequence them
- Of course, there is interaction with these areas!

Lecture Overview

Domain-Independent Action Planning

- Start with a declarative specification of the planning problem
- Use a domain-independent planning system to solve the planning problem
- Domain-independent planners are generic problem solvers
- Issues:
  - Good for evolving systems and those where performance is not critical
  - Running time should be comparable to specialized solvers
  - Solution quality should be acceptable
  - ... at least for all the problems we care about

Planning as Logical Inference

Planning can be elegantly formalized with the help of the *situation calculus*.

Initial state:
$At(truck1, loc1, s_0) \wedge At(package1, loc3, s_0)$

Operators (successor-state axioms):
$\forall a, s, l, p, t\ At(t, p, Do(a, s)) \Leftrightarrow \{a = Drive(t, l, p) \wedge Poss(Drive(t, l, p), s)$
$\vee At(t, p, s) \wedge (a \neq \neg Drive(t, p, l, s) \vee \neg Poss(Drive(t, p, l), s))\}$

Goal conditions (query):
$\exists s\ At(package1, loc2, s)$

The constructive proof of the existential query (computed by a automatic theorem prover) delivers a plan that does what is desired. Can be quite inefficient!

The Basic STRIPS Formalism

STRIPS: STanford Research Institute Problem Solver

- $\mathcal{S}$ is a *first-order vocabulary* (predicate and function symbols) and $\Sigma_{\mathcal{S}}$ denotes the set of *ground atoms* over the signature (also called **facts** or **fluents**).
- $\Sigma_{\mathcal{S},\mathbf{V}}$ is the set of atoms over $\mathcal{S}$ using variable symbols from the set of variables $\mathbf{V}$.
- A **first-order STRIPS state** S is a subset of $\Sigma_{\mathcal{S}}$ denoting a *complete theory* or *model* (using CWA).
- A **planning task** (or **planning instance**) is a 4-tuple $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$, where
  - $\mathbf{O}$ is a set of **operator** (or *action types*)
  - $\mathbf{I} \subseteq \Sigma_{\mathcal{S}}$ is the **initial state**
  - $\mathbf{G} \subseteq \Sigma_{\mathcal{S}}$ is the **goal specification**
- No domain constraints (although present in original formalism)

Operators, Actions & State Change

- **Operator:**

$$o = \langle para, pre, eff \rangle,$$

  with $para \subseteq \mathbf{V}$, $pre \subseteq \Sigma_{\mathcal{S},\mathbf{V}}$, $eff \subseteq \Sigma_{\mathcal{S},\mathbf{V}} \cup \neg\Sigma_{\mathcal{S},\mathbf{V}}$ (element-wise negation) and all variables in pre and eff are listed in $para$.
  Also: $pre(o), eff(o)$.
  eff^+ = positive effect literals
  eff^- = negative effect literals

- **Operator instance** or **action**: Operator with empty parameter list (*instantiated schema!*)

- **State change** induced by action:

$$App(S, o) = \begin{cases} S \cup eff^+(o) - \neg eff^-(o) & \text{if } pre(o) \subseteq S \text{ \& } eff(o) \text{ is cons.} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Example Formalization: *Logistics*

- Logical atoms: $at(O,L)$, $in(O,V)$, $airconn(L1,L2)$, $street(L1,L2)$, $plane(V)$, $truck(V)$
- Load into truck: $load$
  - Parameter list: (O,V,L)
  - Precondition: $at(O,L), at(V,L), truck(V)$
  - Effects: $\neg at(O,L), in(O,V)$
- Drive operation: $drive$
  - Parameter list: $(V,L1,L2)$
  - Precondition: $at(V,L1), truck(V), street(L1,L2)$
  - Effects: $\neg at(V,L1), at(V,L2)$
- ...
- Some constant symbols: $v1, s, t$ with $truck(v1)$ and $street(s,t)$
- Action: $drive(v1,s,t)$

Plans & Successful Executions

- A plan Δ is a sequence of actions
- State resulting from **executing a plan**:

$$
\begin{aligned}
Res(S, \langle\rangle) &= S \\
Res(S, (o; \Delta)) &= \begin{cases} Res(App(S, o), \Delta) & \text{if } App(S, o) \\ & \quad\quad \text{is defined} \\ \text{undefined} & \text{otherwise} \end{cases}
\end{aligned}
$$

- **Plan Δ is successful** or solves a planning task if $Res(\mathbf{I}, \Delta)$ is defined and $\mathbf{G} \subseteq Res(\mathbf{I}, \Delta)$.

A Small Logistics Example

Initial state: $S \;=\; \left\{ \begin{array}{l} at(p1,c), at(p2,s), at(t1,c), \\ at(t2,c), street(c,s), street(s,c) \end{array} \right\}$

Goal: $\mathbf{G} \;=\; \{\; at(p1,s), at(p2,c) \;\}$

Successful plan: $\Delta \;=\; \begin{array}{l} \langle load(p1,t1,c), drive(t1,c,s), \\ unload(p1,t1,s), load(p2,t1,s), \\ drive(t1,s,c), unload(p2,t1,c)\rangle \end{array}$

Other successful plans are, of course, possible

Simplifications: DATALOG- and Propositional STRIPS

- STRIPS as described above allows for unrestricted first-order terms, i.e., arbitrarily nested **function terms**

$\rightarrow$ **Infinite state space**

- Simplification: No function terms (only 0-ary = constants)

$\rightarrow$ **DATALOG-STRIPS**

- Simplification: No variables in operators (= actions)

$\rightarrow$ **Propositional STRIPS**

- Propositional STRIPS used in planning algortihms nowadays (but specification is done using DATALOG-STRIPS)

Beyond STRIPS

Even when keeping all the restrictions of classical planning, one can think of a number of extensions of the planning language.

- General logical formulas as preconditions: Allow all Boolean connectors and quantification
- Conditional effects: Effects that happen only if some additional conditions are true. For example, when pressing the accelerator pedal, the effects depends on which gear has been selected (no, reverse, forward).
- Multi-valued state variables: Instead of 2-valued Boolean variables, multi-valued variables could be used
- Numerical resources: Resources (such as fuel or time) can be effected and be used in preconditions
- Durative actions: Actions can have duration and can be executed concurrently
- Axioms/Constraints: The domain is not only described by operators, but also by additional laws

PDDL: The Planning Domain Description Language

- Since 1998, there exists a bi-annual scientific competition for action planning systems.
- In order to have a common language for this competition, PDDL has been created (originally by Drew McDermott)
- Meanwhile, version 3.1 (IPC-2011) with most of the features mentioned – and many sub-dialects and extensions.
- Sort of "standard" by now.

PDDL Logistics Example

```
(define (domain logistics)
    (:types truck airplane - vehicle
            package vehicle - physobj
            airport location - place
            city place physobj - object)

    (:predicates (in-city ?loc - place ?city - city)
            (at ?obj - physobj ?loc - place)
            (in ?pkg - package ?veh - vehicle))

(:action LOAD-TRUCK
    :parameters     (?pkg - package ?truck - truck ?loc - place)
    :precondition   (and (at ?truck ?loc) (at ?pkg ?loc))
    :effect         (and (not (at ?pkg ?loc)) (in ?pkg ?truck)))
                    ...)
```

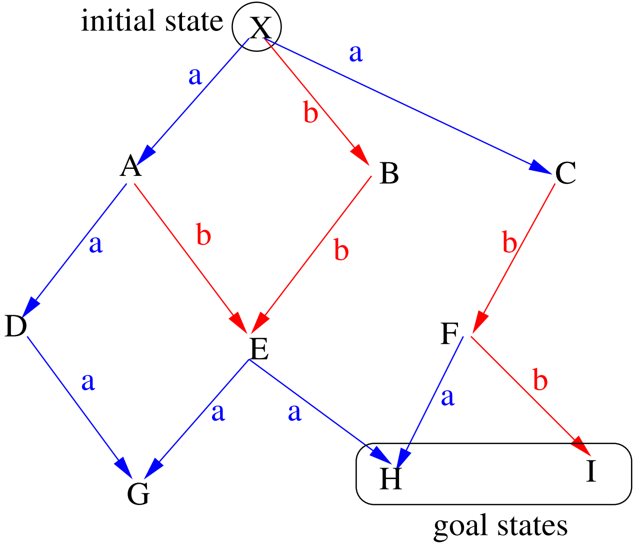
Lecture Overview

Planning Problems as Transition Systems

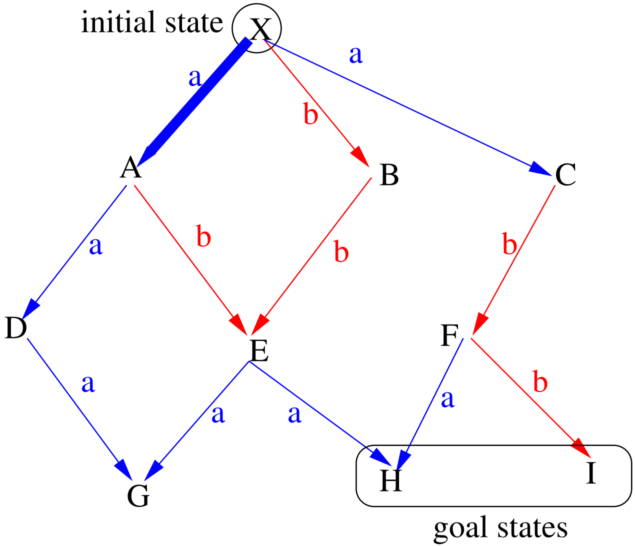
- We can view planning problems as searching for goal nodes in a large labeled graph (transition system)
- Nodes are defined by the value assignment to the fluents = states
- Labeled edges are defined by actions that change the appropriate fluents
- Use graph search techniques to find a (shortest) path in this graph!
- Note: The graph can become huge: 50 Boolean variables lead to $2^{50} = 10^{15}$ states

$\rightarrow$ Create the transition system on the fly and visit only the parts that are *necessary*

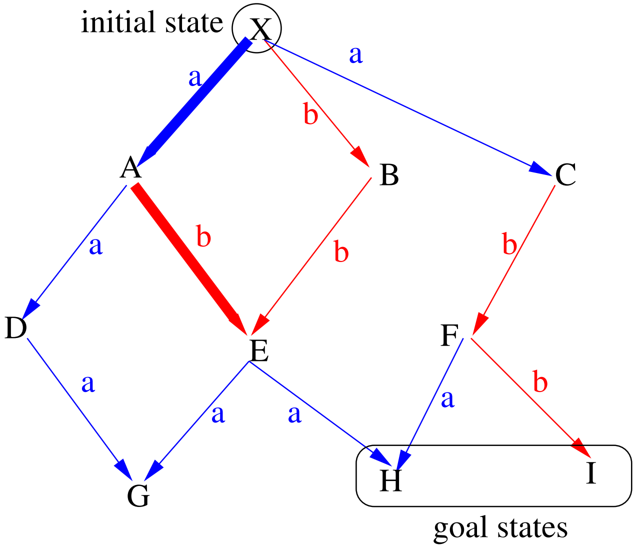
Transition System: Searching Through the State Space



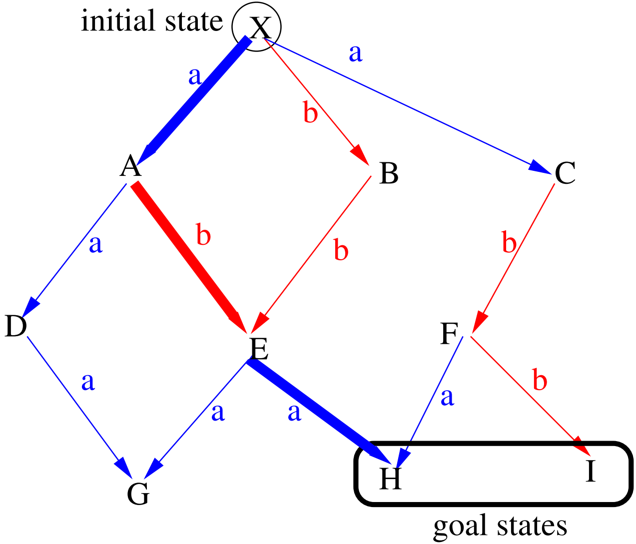
Transition System: Searching Through the State Space



Transition System: Searching Through the State Space



Transition System: Searching Through the State Space



Progression Planning: Forward Search

Search through transition system starting at initial state

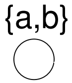
1. Initialize partial plan $\Delta := \langle\,\rangle$ and start at the unique initial state $\mathbf{I}$ and make it the current state S
2. Test whether we have reached a goal state already: $\mathbf{G} \subseteq S$? If so, return plan Δ.
3. Select one applicable action o_i non-deterministically and
   - compute successor state $S := App(S, o_i)$,
   - extend plan $\Delta := \langle \Delta, o_i \rangle$, and continue with step 2.

Instead of non-deterministic choice use some search strategy.

Progression planning can be easily extended to more expressive planning languages

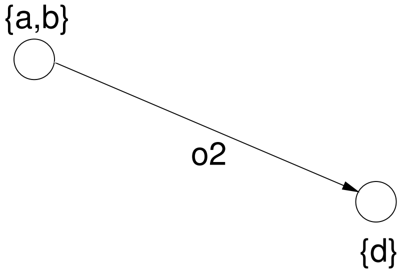
Progression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



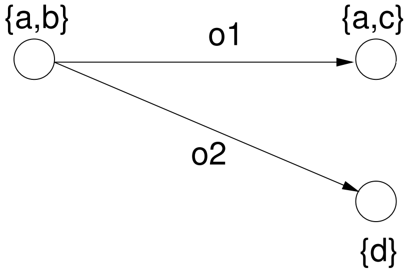
Progression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
& \qquad o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
& \qquad o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



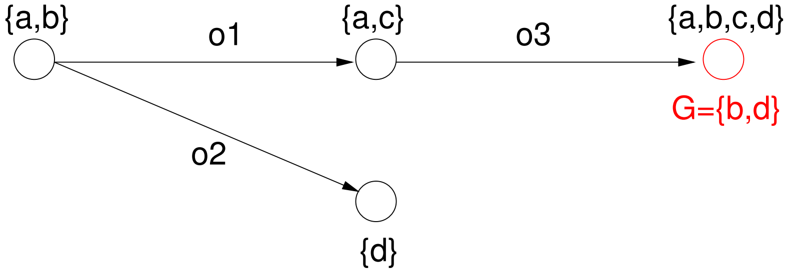
Progression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



Progression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\}\rangle, \\
&\qquad o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\}\rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d\}\rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



Regression Planning: Backward Search

Search through transition system starting at goal states. Consider sets of states, which are described by the atoms that are necessarily true in them

1. Initialize partial plan $\Delta := \langle\,\rangle$ and set $\mathbf{S} := \mathbf{G}$
2. Test whether we have reached the unique initial state already: $\mathbf{I} \supseteq \mathbf{S}$? If so, return plan Δ.
3. Select one action o_i non-deterministically which does not make (sub-)goals false ($\mathbf{S} \cap \neg \mathit{eff}^-(o_i) = \emptyset$) and
   - compute the regression of the description $\mathbf{S}$ through o_i:

     $$\mathbf{S} := \mathbf{S} - \mathit{eff}^+(o_i) \cup \mathit{pre}(o_i)$$

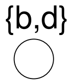
   - extend plan $\Delta := \langle o_i, \Delta \rangle$, and continue with step 2.

Instead of non-deterministic choice use some search strategy

Regression becomes much more complicated, if e.g. conditional effects are allowed. Then the result of a regression can be a general Boolean formula

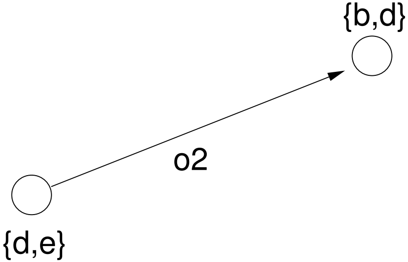
Regression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\}\rangle, \\
&\qquad o_2 = \langle \emptyset, \{e\}, \{b\}\rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\}\rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



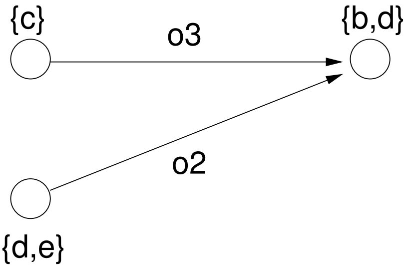
Regression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



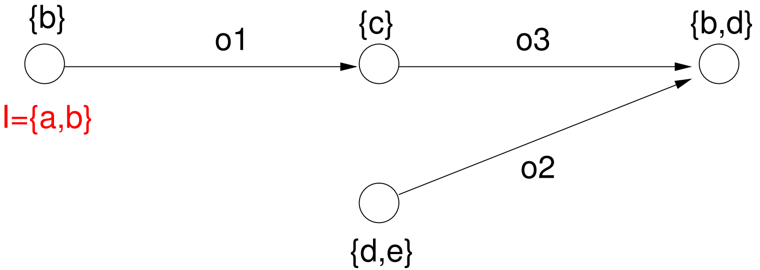
Regression Planning: Example

$$
\begin{array}{rcl}
\mathcal{S} & = & \{a, b, c, d, e\}, \\
\mathbf{O} & = & \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
 & & \quad o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
 & & \quad o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} & = & \{a, b\} \\
\mathbf{G} & = & \{b, d\}
\end{array}
$$



Regression Planning: Example

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
&\qquad o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
&\qquad o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



Other Types of Search

- Of course, other types of search are possible.
- Change perspective: Do not consider the transition system as the space we have to explore, but consider the search through the space of (incomplete) plans:
  - Progression search: Search through the space of plan prefixes
  - Regression search: Search through plan suffixes
- Partial order planning:
  - Search through partially ordered plans by starting with the empty plan and trying to satisfy (sub-)goals by introducing new actions (or using old ones)
  - Make ordering choices only when necessary to resolve conflicts

Lecture Overview

The Planning Problem – Formally

Definition (Plan existence problem (PLANEX))

Instance: $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$.
Question: Does there exist a plan Δ that solves Π, i.e., $Res(\mathbf{I}, \Delta) \supseteq \mathbf{G}$?

Definition (Bounded plan existence problem (PLANLEN))

Instance: $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$ and a positive integer n.
Question: Does there exist a plan Δ of length n or less that solves Π?

From a practical point of view, also PLANGEN (*generating* a plan that solves Π) and PLANLENGEN (*generating* a plan of length n that solves Π) and PLANOPT (generating an optimal plan) are interesting (but at least as hard as the decision problems).

Basic STRIPS with First-Order Terms

- The state space for STRIPS with general first-order terms is **infinite**
- We can use function terms to describe (the index of) tape cells of a Turing machine
- We can use operators to describe the Turing machine control
- The existence of a plan is then equivalent to the existence of a successful computation on the Turing machine
- PLANEX for STRIPS with first-order terms can be used to decide the **Halting problem**

Theorem

PLANEX for STRIPS with first-order terms is ***undecidable****.*

Propositional STRIPS

Theorem

PLANEX is **PSPACE-complete** *for propositional STRIPS.*

→ Membership follows because we can successively guess operators and compute the resulting states (needs only polynomial space)

→ Hardness follows using again a generic reduction from TM acceptance. Instantiate polynomially many tape cells with no possibility to extend the tape (only poly. space, can all be generated in poly. time)

- PLANLEN is also PSPACE-complete (membership is easy, hardness follows by setting $k = 2^{|\Sigma|}$)

Restrictions on Plans

- If we restrict the length of the plans to be short, i.e., only **polynomial** in the size of the planning task, PLANEX becomes NP-complete
- Similarly, if we use a unary representation of the natural number k, then PLANLEN becomes NP-complete

→ Membership obvious (guess & check)

→ Hardness by a straightforward reduction from SAT or by a generic reduction.

- One source of complexity in planning stems from the fact that plans can become very long
- We are only interested in short plans!
- We can use methods for NP-complete problems if we are only looking for “short” plans.

Lecture Overview

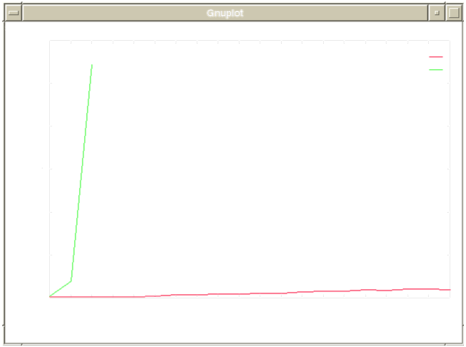
Current Approaches

- Planning as satisfiability: Iterative deepening.
- Planning with answer set programming.
- Symbolic planning with BDDs (finding many plans or non-det. plans)
- Heuristic forward-search planning (HSP, FF, FD)

Heuristic Search Planning

- Use an automatically generated heuristic estimator in order to select the next action or state
- Depending on the search scheme and the heuristic, the plan might not be the shortest one

→ It is often easier to go for sub-optimal solutions (remember *Logistics*)



Runtime of Heuristic search planner (red) vs. iterative deepening (green) on Gripper

Deriving Heuristics: Relaxations

- General principle for deriving heuristics:
  - Define a simplification (relaxation) of the problem and take the difficulty of a solution for the simplified problem as an heuristic estimator
- Example: straight-line distance on a map to estimate the travel distance
- Example: decomposition of a problem, where the components are solved ignoring the interactions between the components, which may incur additional costs
- In planning, one possibility is to ignore negative effects

Ignoring Negative Effects: Example

- In Logistics: The negative effects in $load$ and $drive$ are ignored:
- Simplified load operation: $load(O,V,P)$
  Precondition: $at(O,P), at(V,P), truck(V)$
  Effects: $\cancel{\neg at(O,P)}, in(O,V)$
- After loading, the package is still at the place and also inside the truck
- Simplified drive operation: $drive(V,P1,P2)$
  Precondition: $at(V,P1), truck(V), street(P1,P2)$
  Effects: $\cancel{\neg at(V,P1)}, at(V,P2)$
- After driving, the truck is in two places!

$\rightarrow$ We want the length of the shortest relaxed plan $\rightsquigarrow h^+(s)$

- How difficult is monotonic planning?

Lecture Overview

Current Trends in AI Planning

- Developing and analyzing heuristics
- Developing and anaylzing pruning techniques
- Develping new search techniques
- Extending the expressiveness of planning formalisms (and extending planning algorithms) in order to deal with
  - temporal planning,
  - planning with non-deterministic actions,
  - planning under partial observability,
  - planning with probabilistic effects,
  - multi-agent planning,
  - planning with epistemic goals,
- Reasoning about plans, e.g., diagnosing failures
- Judging morality of plans
- Applying/integrating planning technology
- Learning and planning
- …

Our Interests

- Foundation / theory
- Extending planning technology in order to cope with multi-agent scenarios and epistemic goals
- Using EVMDDs in modelling state-dependent costs
- Using planning techniques and extending them for robot control
- Using planning methodolgy in application in general
- Exploring the ethical dimension of planning systems

Lecture Overview

Summary

- Rational agents need to plan their course of action
- In order to describe planning tasks in a domain-independent, declarative way, one needs planning formalisms
- Basic STRIPS is a simple planning formalism, where actions are described by their preconditions in form of a conjunction of atoms and the effects are described by a list of literals that become true and false
- PDDL is the current "standard language" that has been developed in connection with the international planning competition
- Basic planning algorithms search through the space created by the transition system or through the plan space.
- Planning with STRIPS using first-order terms is undecidable
- Planning with propositional STRIPS is PSPACE-complete
- Since 1992, we have reasonably efficient planning method for propositional, classical STRIPS planning
- You can learn more about it in our planning class next term.

Foundations of Artificial Intelligence

11. Making Simple Decisions under Uncertainty

Probability Theory, Bayesian Networks, Other Approaches

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

June 19, 2019

Contents

Lecture Overview

Motivation

- In many cases, our knowledge of the world is incomplete (not enough information) or uncertain (sensors are unreliable).
- Often, rules about the domain are incomplete or even incorrect
  - e.g., *qualification problem*: what are the preconditions for an action?
- We have to act in spite of this!
- Drawing conclusions under uncertainty

Example

- Goal: Be in Freiburg at 9:15 to give a lecture.
- There are several plans that achieve the goal:
  - P_1: Get up at 7:00, take the bus at 8:15, the train at 8:30, arrive at 9:00 ...
  - P_2: Get up at 6:00, take the bus at 7:15, the train at 7:30, arrive at 8:00 ...
  - ...
- All these plans are correct, but

→ They imply different costs and different probabilities of actually achieving the goal.

→ P_2 eventually is the plan of choice, since giving a lecture is very important, and the success rate of P_1 is only 90-95%.

Uncertainty in Logical Rules (1)

Example: Expert dental diagnosis system.

$$\forall p[Symptom(p, toothache) \Rightarrow Disease(p, cavity)]$$

$\rightarrow$ This rule is *incorrect*! Better:

$$\forall p[Symptom(p, toothache) \Rightarrow$$
$$Disease(p, cavity) \lor Disease(p, gum\_disease) \lor \ldots]$$

. . . however, we do not know all the causes.

Perhaps a *causal* rule is better?

$$\forall p[Disease(p, cavity) \Rightarrow Symptom(p, toothache)]$$

$\rightarrow$ Does not allow to reason from symptoms to causes & is still wrong!

Uncertainty in Logical Rules (2)

- We cannot enumerate all possible causes, and even if we could . . .
- We do not know how correct the rules are (in medicine)
- . . . and even if we did, there will always be uncertainty about the patient (the coincidence of having a toothache and a cavity that are unrelated, or the fact that not all tests have been run)
- Without perfect knowledge, logical rules do not help much!

Uncertainty in Facts

Let us suppose we wanted to support the localization of a robot with (constant) landmarks. With the availability of landmarks, we can narrow down on the area.

Problem: Sensors can be imprecise.

→ From the fact that a landmark was perceived, we cannot conclude with certainty that the robot is at that location.

→ The same is true when no landmark is perceived.

→ Only the probability increases or decreases.

Degree of Belief and Probability Theory

- We (and other agents) are convinced by facts and rules only up to a certain degree.
- One possibility for expressing the degree of belief is to use probabilities.
- Probabilities as frequencies / subjective beliefs
  - e.g., the agent is 90% (or 0.9) convinced by its sensor information means that it believes that in 9 out of 10 cases, the information is correct
- Probabilities quantify the uncertainty that stems from lack of knowledge.
- Probabilities are not to be confused with vagueness. The predicate *tall* is vague; the statement, "*A man is 1.75–1.80m tall*" is uncertain.

Uncertainty and Rational Decisions

- We have a choice of actions (or plans).
- These can lead to different results (worlds) with different probabilities.
- The actions have different (subjective) costs.
- The results have different (subjective) utilities.
- It would be rational to choose the action with the maximum expected total utility!

Decision Theory = Utility Theory + Probability Theory

$$\arg\max_a \sum_\omega P(\omega|a) \cdot [U(\omega) - c(a)]$$

Decision-Theoretic Agent

```
function DT-AGENT(percept) returns an action
   persistent: belief_state, probabilistic beliefs about the current state of the world
               action, the agent's action

   update belief_state based on action and percept
   calculate outcome probabilities for actions,
        given action descriptions and current belief_state
   select action with highest expected utility
        given probabilities of outcomes and utility information
   return action
```

Decision theory: An agent is rational exactly when it chooses the action with the maximum expected utility taken over all results of actions.

Lecture Overview

Axiomatic Probability Theory

Axioms of Probability Theory

A function P mapping from formulae in propositional logic to the set $[0,1]$ is a probability measure if for all propositions ϕ, ψ (whereby propositions are the equivalance classes formed by logically equivalent formulae):

1. $0 \leq P(\phi) \leq 1$
2. $P(true) = 1$
3. $P(false) = 0$
4. $P(\phi \vee \psi) = P(\phi) + P(\psi) - P(\phi \wedge \psi)$

All other properties can be derived from these axioms, for example:

$$P(\neg\phi) = 1 - P(\phi)$$

since $1 \overset{(2)}{=} P(\phi \vee \neg\phi) \overset{(4)}{=} P(\phi) + P(\neg\phi) - \cancel{P(\phi \wedge \neg\phi)} \overset{(3)}{=} P(\phi) + P(\neg\phi)$.

Why are the Axioms Reasonable?

- If P represents an *objectively* observable probability, the axioms clearly make sense.
- But why should an agent respect these axioms when it models its own degree of belief?

→ *Objective* vs. *subjective* probabilities

The axioms limit the set of beliefs that an agent can maintain.

One of the most convincing arguments for why subjective beliefs should respect the axioms was put forward by de Finetti in 1931. It is based on the connection between actions and degree of belief:

- If the beliefs do not follow the axioms, then there exists a betting strategy (the so-called "dutch book") against the agent, where he will definitely loose!

Notation

We use random variables such as $Weather$ (capitalized word), which has a domain of ordered values. In our case that could be $sunny$, $rain$, $cloudy$, $snow$ (lower case words).

A proposition might then be: $Weather = cloudy$.

If the random variable is Boolean, e.g., $Headache$, we may write either $Headache = true$ or equivalently $headache$ (lowercase!). Similarly, we may write $Headache = false$ or equivalently $\neg headache$.

Further, we can of course use Boolean connectors, e.g.,
$\neg headhache \wedge Weather = cloudy$.

Unconditional Probabilities (1)

$P(a)$ denotes the unconditional probability that it will turn out that $A = true$ *in the absence of any other information*, for example:

$$P(cavity) = 0.1$$

$P(\neg cavity) = 0.9$

In case of non-Boolean random variables:

$$P(Weather = sunny) = 0.7$$
$$P(Weather = rain) = 0.2$$
$$P(Weather = cloudy) = 0.08$$
$$P(Weather = snow) = 0.02$$

1

Unconditional Probabilities (2)

$\mathbf{P}(X)$ is the vector of probabilities for the (ordered) domain of the random variable X:

$$\mathbf{P}(Headache) = \langle 0.1, 0.9 \rangle$$
$$\mathbf{P}(Weather) = \langle 0.7, 0.2, 0.08, 0.02 \rangle$$

H	W	P(H,W)	P(H|W)
t	sun	0.3	0.4
t	r		?
t	c		
t	sn		
f	sun		0.6
f			
f			
f			

define the probability distributions for the random variables $Headache$ and $Weather$.

$\mathbf{P}(Headache, Weather)$ is a 4×2 table of probabilities of all combinations of the values of a set of random variables.

	$Headache = true$	$Headache = false$
$Weather = sunny$	$P(W = sunny \wedge headache)$	$P(W = sunny \wedge \neg headache)$
$Weather = rain$		
$Weather = cloudy$		
$Weather = snow$		

Conditional Probabilities (1)

New information can change the probability.

Example: The probability of a cavity increases if we know the patient has a toothache.

If additional information is available, we can no longer use the prior probabilities!

$P(a \mid b)$ is the conditional or posterior probability of a given that *all we know* is b:

$$P(cavity \mid toothache) = 0.8$$

$\mathbf{P}(X \mid Y)$ is the table of all conditional probabilities over all values of X and Y.

Conditional Probabilities (2)

$\mathbf{P}(Weather \mid Headache)$ is a 4×2 table of conditional probabilities of all combinations of the values of a set of random variables.

	$Headache = true$	$Headache = false$
$Weather = sunny$	$P(W = sunny \mid headache)$	$P(W = sunny \mid \neg headache)$
$Weather = rain$		
$Weather = cloudy$		
$Weather = snow$		

Conditional probabilities result from unconditional probabilities (if $P(b) > 0$) (by definition):

$P(a \land b) = P(a|b) \cdot P(b)$

$$P(a \mid b) = \frac{P(a \land b)}{P(b)}$$

Conditional Probabilities (3)

$\mathbf{P}(X, Y) = \mathbf{P}(X \mid Y)\mathbf{P}(Y)$ corresponds to a system of equations:

$$
\begin{aligned}
P(W = sunny \wedge headache) &= P(W = sunny \mid headache)P(headache) \\
P(W = rain \wedge headache) &= P(W = rain \mid headache)P(headache) \\
\ldots &= \ldots \\
P(W = snow \wedge \neg headache) &= P(W = snow \mid \neg headache)P(\neg headache)
\end{aligned}
$$

Conditional Probabilities (4)

$$\boxed{P(a \mid b) = \frac{P(a \wedge b)}{P(b)}}$$

- Product rule: $P(a \wedge b) = P(a \mid b)P(b)$
- Similarly: $P(a \wedge b) = P(b \mid a)P(a)$
- a and b are independent iff $P(a \mid b) = P(a)$ (equiv. $P(b \mid a) = P(b)$).
  Then (and only then) it holds that $P(a \wedge b) = P(a)P(b)$.

  Making this assumption: what is the effect wrt. computational efficiency?

Lecture Overview

1. Motivation
2. Foundations of Probability Theory
3. **Probabilistic Inference**
4. Bayesian Networks
5. Alternative Approaches

Joint Probability

The agent assigns probabilities to every proposition in the domain.

An atomic event assigns a value to every random variable $X_1, \ldots, X_n$ (= complete specification of a state).

Example: Let X and Y be Boolean variables. Then we have the following 4 atomic events: $x \land y$, $x \land \neg y$, $\neg x \land y$, $\neg x \land \neg y$.

The joint probability distribution $\mathbf{P}(X_1, \ldots, X_n)$ assigns a probability to every *atomic event*. Example of such a complete instantiation:

	$toothache$	$\neg toothache$
$cavity$	0.04	0.06
$\neg cavity$	0.01	0.89

Observe: The sum of all fields is 1 (disjunction of events). Since all atomic events are disjoint, the conjunction of any two atomic events is necessarily $false$.

Working with the Joint Probability

All relevant probabilities can be computed using the joint probability by expressing them as a disjunction of atomic events.

Examples:

$$
\begin{aligned}
P(cavity \lor toothache) = {} & P(cavity \land toothache) \\
& + P(\lnot cavity \land toothache) \\
& + P(cavity \land \lnot toothache)
\end{aligned}
$$

We obtain marginal probabilities by adding across a row or column:

$$P(cavity) = P(cavity \land toothache) + P(cavity \land \lnot toothache)$$

We obtain conditional probabilities by using a marginal probability:

$$P(cavity \mid toothache) = \frac{P(cavity \land toothache)}{P(toothache)} = \frac{0.04}{0.04 + 0.01} = 0.80$$

Marginalization

For any sets of variables Y and Z we have

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}, \mathbf{z}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y} \mid \mathbf{z}) \mathbf{P}(\mathbf{z})$$

law of total probability

Problems with Joint Probabilities

We can easily obtain all probabilities from the joint probability.

The joint probability, however, involves k^n values, if there are n random variables with k values.

$\rightarrow$ Difficult to represent

$\rightarrow$ Difficult to assess

Questions:

$\rightarrow$ Is there a more compact way of representing joint probabilities?

$\rightarrow$ Is there an efficient method to work with this representation?

Answer: Not in general, but it can work in many cases. Modern systems work directly with conditional probabilities and make assumptions on the independence of variables ($\rightarrow$conditional independence) to simplify calculations.

Representing Joint Probabilites

Using the product rule $P(a \wedge b) = P(a \mid b)\, P(b)$, joint probabilites can be expressed as products of conditional probabilities.

$P(x_1, \ldots, x_n) = P(x_n, \ldots, x_1)$

Representing Joint Probabilites

Using the product rule $P(a \wedge b) = P(a \mid b)\,P(b)$, joint probabilites can be expressed as products of conditional probabilities.

$P(x_1, \ldots, x_n) = P(x_n, \ldots, x_1) = P(x_n \mid x_{n-1} \ldots, x_1)\,P(x_{n-1}, \ldots, x_1)$

Representing Joint Probabilites

Using the product rule $P(a \wedge b) = P(a \mid b)\, P(b)$, joint probabilites can be expressed as products of conditional probabilities.

$P(x_1, \ldots, x_n) = P(x_n, \ldots, x_1) = P(x_n \mid x_{n-1} \ldots, x_1)\, P(x_{n-1}, \ldots, x_1)$
$= P(x_n \mid x_{n-1} \ldots, x_1)\, P(x_{n-1} \mid x_{n-2} \ldots, x_1)\, P(x_{n-2}, \ldots, x_1)$

2^n

Using the product rule $P(a \wedge b) = P(a \mid b)\, P(b)$, joint probabilites can be expressed as products of conditional probabilities.

$P(x_1, \ldots, x_n) = P(x_n, \ldots, x_1) = P(x_n \mid x_{n-1} \ldots, x_1)\, P(x_{n-1}, \ldots, x_1)$
$= P(x_n \mid x_{n-1} \ldots, x_1)\, P(x_{n-1} \mid x_{n-2} \ldots, x_1)\, P(x_{n-2}, \ldots, x_1)$
$= P(x_n \mid x_{n-1} \ldots, x_1)\, P(x_{n-1} \mid x_{n-2} \ldots, x_1)\, P(x_{n-2} \mid x_{n-3} \ldots, x_1)$
$P(x_{n-3}, \ldots, X_1)$

2^n $\quad 2^{n-1}$ $\quad \ldots$

$= \ldots$
$= P(x_n \mid x_{n-1} \ldots, x_1)\, P(x_{n-1} \mid x_{n-2} \ldots, x_1) \ldots P(x_2 \mid x_1)\, P(x_1)$
$= \Pi_{i=1}^{n} P(x_i \mid x_{i-1} \ldots x_1)$

Can these transformations change the required storage size?

Bayes' Rule

We know (product rule):

$$P(a \wedge b) = P(a \mid b)P(b) \text{ and } P(a \wedge b) = P(b \mid a)P(a)$$

By equating the right-hand sides, we get

$$P(a \mid b)P(b) = P(b \mid a)P(a)$$

$$\boxed{\Rightarrow P(a \mid b) = \frac{P(b \mid a)P(a)}{P(b)}}$$

$P(a|b)$

For multi-valued variables we get a set of equalities:

$$\mathbf{P}(Y \mid X) = \frac{\mathbf{P}(X \mid Y)\mathbf{P}(Y)}{\mathbf{P}(X)}$$

Generalization (conditioning on background evidence e):

$$\mathbf{P}(Y \mid X, e) = \frac{\mathbf{P}(X \mid Y, e)\mathbf{P}(Y \mid e)}{\mathbf{P}(X \mid e)}$$

Applying Bayes' Rule

$$P(toothache \mid cavity) = 0.4$$
$$P(cavity) = 0.1$$
$$P(toothache) = 0.05$$

$$P(cavity \mid toothache) = \frac{0.4 \times 0.1}{0.05} = 0.8$$

Why do we not try to assess $P(cavity \mid toothache)$ directly?

$P(toothache \mid cavity)$ (causal) is more robust than $P(cavity \mid toothache)$ (diagnostic):

- $P(toothache \mid cavity)$ is independent from the prior probabilities $P(toothache)$ and $P(cavity)$.
- If there is a cavity epidemic and $P(cavity)$ increases, $P(toothache \mid cavity)$ does not change, but $P(toothache)$ and $P(cavity \mid toothache)$ will change proportionally.

Relative Probability

Let's say we would also like to consider the probability that our patient has gum disease.

$$
\begin{aligned}
P(toothache \mid gumdisease) &= 0.7 \\
P(gumdisease) &= 0.02
\end{aligned}
$$

Which diagnosis is more probable? Cavity or gum disease?

$$P(c \mid t) = \frac{P(t \mid c)P(c)}{P(t)} \text{ or } P(g \mid t) = \frac{P(t \mid g)P(g)}{P(t)}$$

If we are only interested in the relative probability, we need not assess $P(t)$:

$$
\begin{aligned}
\frac{P(c \mid t)}{P(g \mid t)} &= \frac{P(t \mid c)P(c)}{P(t)} \times \frac{P(t)}{P(t \mid g)P(g)} = \frac{P(t \mid c)P(c)}{P(t \mid g)P(g)} \\
&= \frac{0.4 \times 0.1}{0.7 \times 0.02} = 2.857
\end{aligned}
$$

→ We elegantly excluded other possible diagnoses for toothache.

Normalization (1)

If we wish to determine the absolute probability of $P(c \mid t)$ but do not know $P(t)$, we can alternatively carry out a complete case analysis (e.g., for c and $\neg c$) and use the fact that $P(c \mid t) + P(\neg c \mid t) = 1$ (here Boolean variables):

$$P(c \mid t) = \frac{P(t \mid c)P(c)}{P(t)}$$

$$P(\neg c \mid t) = \frac{P(t \mid \neg c)P(\neg c)}{P(t)}$$

$$1 = P(c \mid t) + P(\neg c \mid t) = \frac{P(t \mid c)P(c)}{P(t)} + \frac{P(t \mid \neg c)P(\neg c)}{P(t)}$$

$$P(t) = P(t \mid c)P(c) + P(t \mid \neg c)P(\neg c) \quad *$$

Normalization (2)

By substituting into the first equation:

$$P(c \mid t) = \frac{P(t \mid c)P(c)}{P(t \mid c)P(c) + P(t \mid \neg c)P(\neg c)}$$

$P(t)$

For random variables with multiple values:

$\propto P(X \mid Y) \cdot P(Y)$

$$\mathbf{P}(Y \mid X) = \alpha \mathbf{P}(X \mid Y)\mathbf{P}(Y)$$

$\frac{1}{2}$

where α is the normalization constant needed to make the entries in $\mathbf{P}(Y \mid X)$ sum to 1 for each value of X.

Example: $\alpha(.1, .1, .3) = (.2, .2, .6)$.

Remark: In ML, relative probabilities often are sufficient.

Example

Your doctor tells you that you have tested positive for a serious but rare (1/10000) disease. This test (t) is correct to 99% (1% false positive & 1% false negative results).

What does this mean for you?

$$P(d \mid t) = \frac{P(t \mid d) \cdot P(d)}{P(t)}$$

Example

Your doctor tells you that you have tested positive for a serious but rare (1/10000) disease. This test (t) is correct to 99% (1% false positive & 1% false negative results).

What does this mean for you?

0.99 1/10000

$$P(d \mid t) = \frac{P(t \mid d)P(d)}{P(t)} = \frac{P(t \mid d)P(d)}{P(t \mid d)P(d) + P(t \mid \neg d)P(\neg d)}$$

Example

Your doctor tells you that you have tested positive for a serious but rare (1/10000) disease. This test (t) is correct to 99% (1% false positive & 1% false negative results).

What does this mean for you?

$$P(d \mid t) = \frac{P(t \mid d)P(d)}{P(t)} = \frac{P(t \mid d)P(d)}{P(t \mid d)P(d) + P(t \mid \neg d)P(\neg d)}$$

$$P(d) = 0.0001 \quad P(t \mid d) = 0.99 \quad P(t \mid \neg d) = 0.01$$

$$P(d \mid t) = \frac{0.99 \times 0.0001}{0.99 \times 0.0001 + 0.01 \times 0.9999} = \frac{0.000099}{0.000099 + 0.009999}$$
$$= \frac{0.000099}{0.010088} \approx 0.01$$

Moral: If the test imprecision is much greater than the rate of occurrence of the disease, then a positive result is not as threatening as you might think.

Multiple Evidence (1)

A probe by the dentist catches ($Catch = true$) in the aching tooth ($Toothache = true$) of a patient. We already know that $P(cavity \mid toothache) = 0.8$. Furthermore, using Bayes' rule, we can calculate:

$$P(cavity \mid catch) = 0.95$$

But how does the combined evidence ($tooth \wedge catch$) help?

Using Bayes' rule, the dentist could establish:

$$\begin{aligned} P(cav \mid tooth \wedge catch) &= \frac{P(tooth \wedge catch \mid cav) \times P(cav)}{P(tooth \wedge catch)} \\ &= \alpha P(tooth \wedge catch \mid cav) \times P(cav) \end{aligned}$$

Multiple Evidence (2)

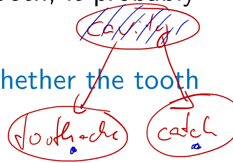
Problem: The dentist needs $P(tooth \wedge catch \mid cav)$, i.e., diagnostic knowledge of all combinations of symptoms in the general case.

It would be nice if $tooth$ and $catch$ were independent but they are not: $P(tooth \mid catch) \neq P(tooth)$ - if a probe catches in the tooth, it probably has a cavity which probably causes toothache.

Multiple Evidence (2)

Problem: The dentist needs $P(tooth \land catch \mid cav)$, i.e., diagnostic knowledge of all combinations of symptoms in the general case.

It would be nice if *tooth* and *catch* were independent but they are not: $P(tooth \mid catch) \neq P(tooth)$ - if a probe catches in the tooth, it probably has a cavity which probably causes toothache.



They are *conditionally* independent given that we know whether the tooth has a cavity:

$$P(tooth \mid \cancel{catch}, cav) = P(tooth \mid cav)$$

If one already knows that there is a cavity, then the additional knowledge of the probe catching does not change the probability.

$$P(tooth \land catch \mid cav) =$$

$$P(tooth \mid \cancel{catch}, cav)P(catch \mid cav) = P(tooth \mid cav)P(catch \mid cav)$$

Conditional Independence

Thus our diagnostic problem turns into:

$$P(cav \mid tooth \land catch) = \alpha P(tooth \land catch \mid cav)P(cav)$$

Conditional Independence

Thus our diagnostic problem turns into:

$$P(cav \mid tooth \wedge catch) = \alpha P(tooth \wedge catch \mid cav) P(cav)$$

$$= \alpha P(tooth \mid catch, cav) P(catch \mid cav) P(cav)$$

Conditional Independence

Thus our diagnostic problem turns into:

$$P(cav \mid tooth \wedge catch) = \alpha P(tooth \wedge catch \mid cav) P(cav)$$

$$= \alpha P(tooth \mid catch, cav) P(catch \mid cav) P(cav)$$

$$= \alpha P(tooth \mid cav) P(catch \mid cav) P(cav)$$

Conditional Independence

Thus our diagnostic problem turns into:

$$P(cav \mid tooth \wedge catch) = \alpha P(tooth \wedge catch \mid cav) P(cav)$$

$$= \alpha P(tooth \mid catch, cav) P(catch \mid cav) P(cav)$$

$$= \alpha P(tooth \mid cav) P(catch \mid cav) P(cav)$$

The general definition of conditional independence of two variables X and Y given a third variable Z (a common cause) is:

$$\mathbf{P}(X, Y \mid Z) = \mathbf{P}(X \mid Z) \mathbf{P}(Y \mid Z)$$

Conditional Independence - Further Example

Eating icecream and observing sunshine is not independent

$$P(ice \mid sun) \neq P(ice)$$

The variables Ice and Sun are not independent.
But if the reason for eating icecream is simply that it is hot outside, then the additional observation of sunshine does not make a difference:

$$P(ice \mid sun, hot) = P(ice \mid hot)$$

The variables Ice and Sun are conditionally independent given that $Hot = true$ is observed.
The knowledge about independence often comes from insight of the domain and is part of the modelling of the problem. Conditional independence can often be exploited to make things simpler (see later).

Recursive Bayesian Updating

Problem: we would like to avoid calculating the full joint probability table!

Assuming conditional independence, multiple evidence can be reduced to prior probabilities and conditional probabilities.
The general combination rule, if Z_1 and Z_2 are independent given X is

$$\mathbf{P}(X \mid Z_1, Z_2) = \alpha \mathbf{P}(X)\mathbf{P}(Z_1 \mid X)\mathbf{P}(Z_2 \mid X)$$

where α is the normalization constant.

Recursive Bayesian Updating

Problem: we would like to avoid calculating the full joint probability table!

Assuming conditional independence, multiple evidence can be reduced to prior probabilities and conditional probabilities.
The general combination rule, if Z_1 and Z_2 are independent given X is

$$\mathbf{P}(X \mid Z_1, Z_2) = \alpha \mathbf{P}(X)\mathbf{P}(Z_1 \mid X)\mathbf{P}(Z_2 \mid X)$$

where α is the normalization constant.

Generalization: Recursive Bayesian Updating

$$\mathbf{P}(X \mid Z_1, \ldots, Z_n) = \alpha \mathbf{P}(X) \prod_{i=1}^{n} \mathbf{P}(Z_i \mid X)$$

Types of Variables

- Variables can be discrete or continuous:
- Discrete variables
  - $Weather$: $sunny$, $rain$, $cloudy$, $snow$
  - $Cavity$: $true$, $false$ (Boolean)
- Continuous variables
  - Tomorrow's maximum temperature in Freiburg
  - Domain can be the entire real line or any subset.
  - Distributions for continuous variables are typically given by probability density functions.

Summary

- Uncertainty is unavoidable in complex, dynamic worlds in which agents are ignorant.
- Probabilities express the agent's inability to reach a definite decision. They summarize the agent's beliefs.
- Conditional and unconditional probabilities can be formulated over propositions.
- If an agent disrespects the theoretical probability axioms, it is likely to demonstrate irrational behaviour.
- Bayes' rule allows us to calculate known probabilities from unknown probabilities.
- Multiple evidence (assuming independence) can be effectively incorporated using recursive Bayesian updating.

Lecture Overview

Example domain: I am at work. My neighbour John calls me to tell me, that my alarm is ringing. My neighbour Mary doesn't call. Sometimes, the alarm is started by a slight earthquake.

Question: Is there a burglary?

Variables: $Burglary$, $Earthquake$, $Alarm$, $JohnCalls$, $MaryCalls$.

Bayesian Networks

Domain knowledge/ assumptions:

- Events $Burglary$ and $Earthquake$ are independent. (of course, to be discussed: a burglary does not cause an earthquake, but a burglar might use an earthquake to do the burglary. Then the independence assumption is not true. This is a design decision!)
- $Alarm$ might be activated by burglary or earthquake
- John calls if and only if he heard the alarm. His call probability is not influenced by the fact, that there is an earthquake at the same time. Same for Mary.

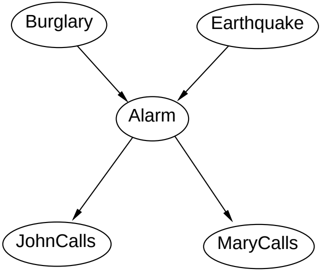
How to model this domain efficiently? Goal: Answer questions.

Bayesian Networks

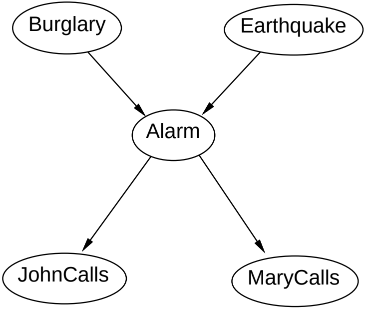
(also belief networks, probabilistic networks, causal networks)

- The *random variables* are the nodes.
- Directed edges between nodes represent *direct influence*.
- A table of conditional probabilities (CPT) is associated with every node, in which the effect of the parent nodes is quantified.
- The graph is acyclic (a DAG).

Remark: $Burglary$ and $Earthquake$ are denoted as the parents of $Alarm$



The Meaning of Bayesian Networks



- $Alarm$ depends on $Burglary$ and $Earthquake$.
- $MaryCalls$ only depends on $Alarm$.
  $P(maryCalls \mid alarm, burglary) = P(maryCalls \mid alarm)$ and
  $P(maryCalls \mid alarm, burglary, johnCalls, earthquake) = P(maryCalls \mid alarm)$

$\rightarrow$ Bayesian Networks can be considered as sets of (conditional) independence assumptions.

Bayesian Networks and the Joint Probability

Bayesian networks can be seen as a more compact representation of joint probabilities.

Let all nodes $X_1, \ldots, X_n$ be ordered topologically according to the arrows in the network. Let $x_1, \ldots, x_n$ be the values of the variables. Then

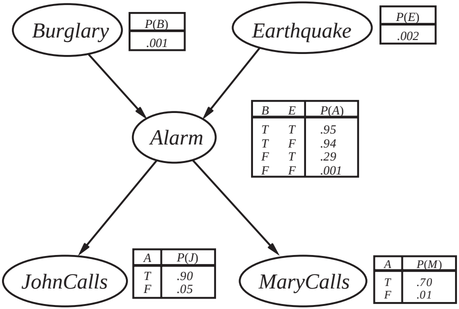
$$\begin{aligned} P(x_1, \ldots, x_n) &= P(x_n \mid x_{n-1}, \ldots, x_1) \cdot \ldots \cdot P(x_2 \mid x_1) P(x_1) \\ &= \prod_{i=1}^{n} P(x_i \mid x_{i-1}, \ldots, x_1) \end{aligned}$$

According to the independence assumption, this is equivalent to

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid parents(x_i))$$

We can calculate the joint probability from the network topology and the conditional probability tables (CPTs)!

Example



Only prob. for pos. events are given, negative: $P(\neg x) = 1 - P(x)$. Note: the size of the table depends on the number of parents!

$$
\begin{aligned}
&P(j, m, a, \neg b, \neg e) = \\
&P(j \mid m, a, \neg b, \neg e) P(m \mid a, \neg b, \neg e) P(a \mid \neg b, \neg e) P(\neg b \mid \neg e) P(\neg e) \\
&= P(j \mid a) P(m \mid a) P(a \mid \neg b, \neg e) P(\neg b) P(\neg e) \\
&= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 = 0.00062
\end{aligned}
$$

Compactness of Bayesian Networks

- For the explicit representation of Bayesian networks, we need a table of size 2^n where n is the number of variables.
- In the case that every node in a network has at most k parents, we only need n tables of size 2^k (assuming Boolean variables).
- *Example*: $n = 20$ and $k = 5$

→ $2^{20} = 1,048,576$ and $20 \times 2^5 = 640$ different explicitly-represented probabilities!

→ In the worst case, a Bayesian network can become exponentially large, for example if every variable is directly influenced by all the others.

→ The size depends on the application domain (local vs. global interaction) and the skill of the designer.

Naive Design of a Network

- Order all variables
- Take the first from those that remain
- Assign all direct influences from nodes already in the network to the new node (Edges + CPT).
- If there are still variables in the list, repeat from step 2.

Example 1

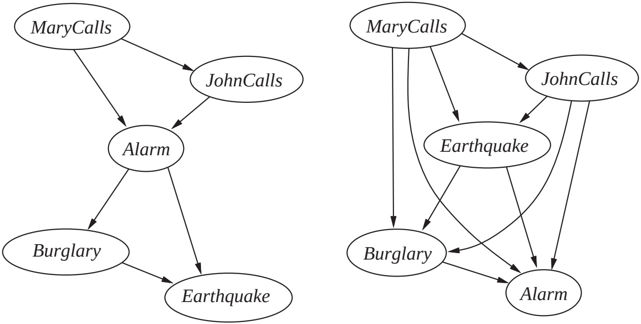
M, J, A, B, E

Example 2

M, J, E, B, A

Example

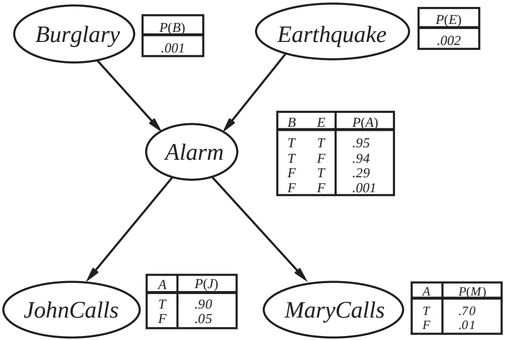
left $= M, J, A, B, E$, right $= M, J, E, B, A$



(a) (b)

$\rightarrow$ Appears to be an attempt to build a diagnostic model of symptoms and causes, which always leads to dependencies between causes that are actually independent and symptoms that appear separately.

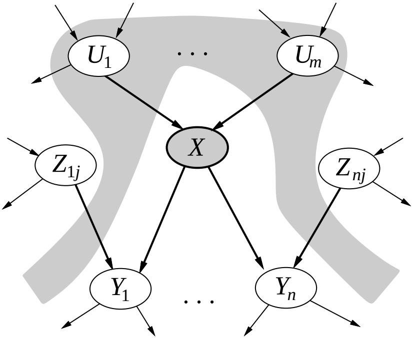
Instantiating evidence variables and sending queries to nodes.



What is $P(burglary \mid johncalls)$
or $P(burglary \mid johnCalls, maryCalls)$?

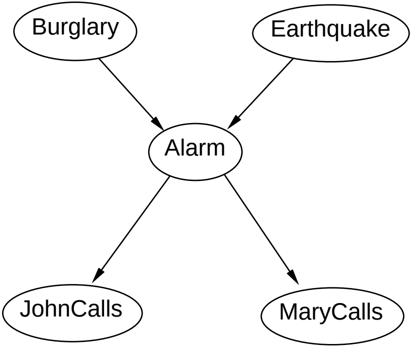
Conditional Independence Relations in Bayesian Networks (1)

A node is conditionally independent of its non-descendants given its parents.



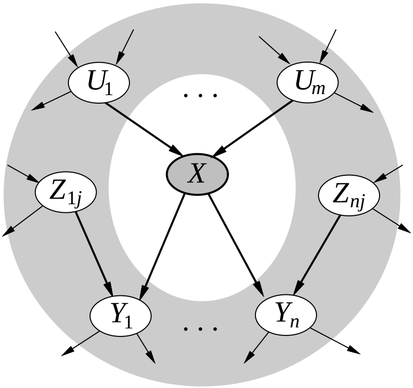
Example

$JohnCalls$ is independent of $Burglary$ and $Earthquake$ given the value of $Alarm$.



Conditional Independence Relations in Bayesian Networks (2)

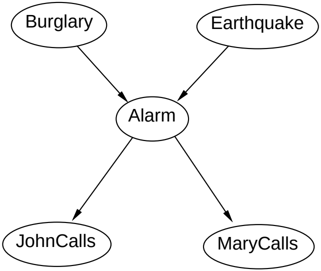
A node is conditionally independent of all other nodes in the network given the Markov blanket, i.e., its parents, children and children's parents.



Example

$Burglary$ is independent of $JohnCalls$ and $MaryCalls$, given the values of $Alarm$ and $Earthquake$, i.e.,

$$P(Burglary \mid JohnCalls, MaryCalls, Alarm, Earthquake) \\ = P(Burglary \mid Alarm, Earthquake)$$



Exact Inference in Bayesian Networks

- Compute the posterior probability distribution for a set of query variables X given an observation, i.e., the values of a set of evidence variables E.
- Complete set of variables is $X \cup E \cup Y$
- Y are called the hidden variables
- Typical query $P(X \mid e)$ where e are the observed values of E.
- In the remainder: X is a singleton

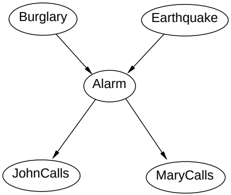
Example:
$\mathbf{P}(Burglary \mid JohnCalls = true, MaryCalls = true) = (0.284, 0.716)$

Inference by Enumeration

- $P(X \mid e) = \alpha P(X, e) = \sum_{y} \alpha P(X, e, y)$
- The network gives a complete representation of the full joint distribution.
- A query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.
- We sum over the hidden variables.

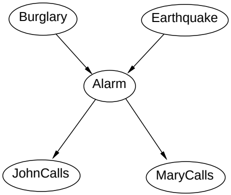
Example

- Consider $\mathbf{P}(Burglary \mid JohnCalls = true,$
  $MaryCalls = true)$
- The evidence variables are



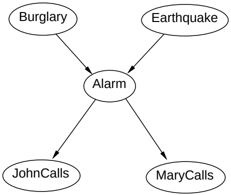
Example

- Consider $\mathbf{P}(Burglary \mid JohnCalls = true,$
  $MaryCalls = true)$
- The evidence variables are $JohnCalls$ and $MaryCalls$.
- The hidden variables are



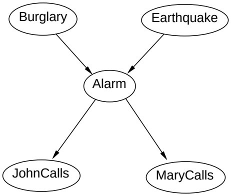
Example

- Consider $\mathbf{P}(Burglary \mid JohnCalls = true, MaryCalls = true)$
- The evidence variables are $JohnCalls$ and $MaryCalls$.
- The hidden variables are $Earthquake$ and $Alarm$.
- We have: $\mathbf{P}(B \mid j, m) = \alpha \mathbf{P}(B, j, m)$



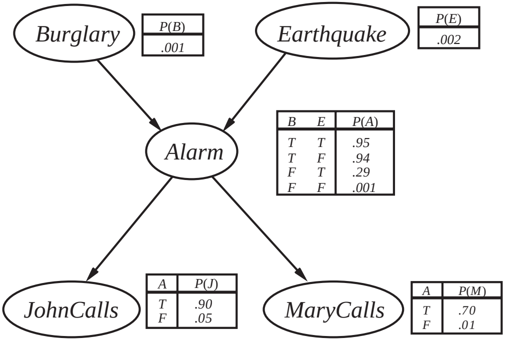
Example

- Consider $\mathbf{P}(Burglary \mid JohnCalls = true, MaryCalls = true)$
- The evidence variables are $JohnCalls$ and $MaryCalls$.
- The hidden variables are $Earthquake$ and $Alarm$.



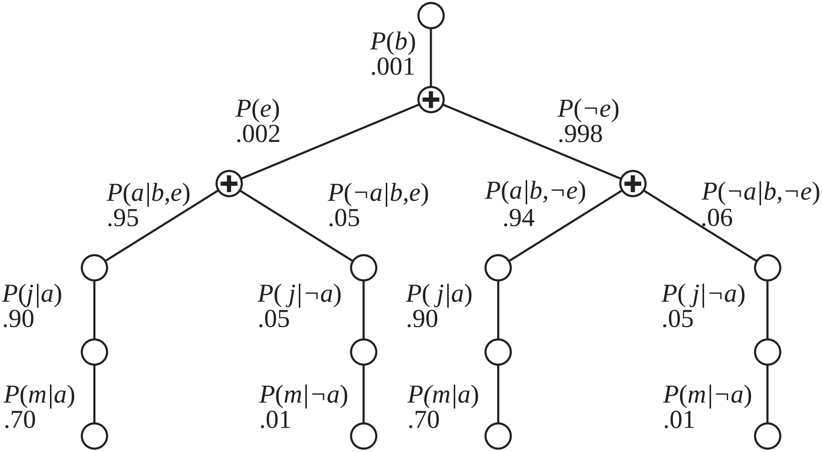
- We have: $$\mathbf{P}(B \mid j, m) = \alpha \mathbf{P}(B, j, m) = \alpha \sum_e \sum_a \mathbf{P}(B, j, m, e, a)$$
- If we consider the independence of variables, we obtain for $B = true$ $$P(b \mid j, m) = \alpha \sum_e \sum_a P(j \mid a) P(m \mid a) P(a \mid e, b) P(e) P(b)$$
- Reorganization of the terms yields: $$P(b \mid j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid e, b) P(j \mid a) P(m \mid a)$$

Recall Bayesian Network for Domain



Evaluation of $P(b \mid j, m)$

$$P(b \mid j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid e, b) P(j \mid a) P(m \mid a)$$



$$\mathbf{P}(B \mid j, m) = \alpha(0.0006, 0.0015) = (0.284, 0.716)$$

Enumeration Algorithm for Answering Queries on Bayesian Networks

```
function ENUMERATION-ASK(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, observed values for variables E
          bn, a Bayes net with variables {X} ∪ E ∪ Y   /* Y = hidden variables */

  Q(X) ← a distribution over X, initially empty
  for each value x_i of X do
      Q(x_i) ← ENUMERATE-ALL(bn.VARS, e_{x_i})
          where e_{x_i} is e extended with X = x_i
  return NORMALIZE(Q(X))
```

```
function ENUMERATE-ALL(vars, e) returns a real number
  if EMPTY?(vars) then return 1.0
  Y ← FIRST(vars)
  if Y has value y in e
      then return P(y | parents(Y)) × ENUMERATE-ALL(REST(vars), e)
      else return ∑_y P(y | parents(Y)) × ENUMERATE-ALL(REST(vars), e_y)
          where e_y is e extended with Y = y
```

Properties of the ENUMERATION-ASK Algorithm

- The ENUMERATION-ASK algorithm evaluates the trees in a depth-first manner.
- Space complexity is linear in the number of variables.
- Time complexity for a network with n Boolean variables is $O(2^n)$, since in the worst case, all terms must be evaluated for the two cases ("true" and "false")

Variable Elimination

- The enumeration algorithm can be improved significantly by eliminating repeating or unnecessary calculations.
- The key idea is to evaluate expressions from right to left (bottom-up) and to save results for later use.
- Additionally, unnecessary expressions can be removed.

Example

- Let us consider the query $P(JohnCalls \mid Burglary = true)$.
- The nested sum is

$$P(j, b) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e) P(j, a) \sum_m P(m \mid a)$$

Example

- Let us consider the query $P(JohnCalls \mid Burglary = true)$.
- The nested sum is

$$P(j, b) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e) P(j, a) \sum_m P(m \mid a)$$

- Obviously, the rightmost sum equals 1 so that it can safely be dropped.
- general observation: variables, that are not query or evidence variables and not ancestor nodes of query or evidence variables can be removed. Variable elimination repeatedly removes these variables and this way speeds up computation.

Example

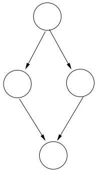
- Let us consider the query $P(JohnCalls \mid Burglary = true)$.
- The nested sum is

$$P(j, b) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e) P(j, a) \sum_m P(m \mid a)$$

- Obviously, the rightmost sum equals 1 so that it can safely be dropped.
- general observation: variables, that are not query or evidence variables and not ancestor nodes of query or evidence variables can be removed. Variable elimination repeatedly removes these variables and this way speeds up computation.
- within example: $Alarm$ and $Earthquake$ are ancestor nodes of query variable $JohnCalls$ and cannot be removed. $MaryCalls$ is neither a query nor an evidence variable and no ancestor node. Therefore it can be removed.

Complexity of Exact Inference

- If the network is singly connected or a polytree (at most one undirected path between two nodes in the graph), the time and space complexity of exact inference is linear in the size of the network.
- The burglary example is a typical singly connected network.
- For multiply connected networks inference in Bayesian Networks is NP-hard.



- There are approximate inference methods for multiply connected networks such as sampling techniques or Markov chain Monte Carlo.

Lecture Overview

Other Approaches (1)

- Rule-based methods with “certainty factors”.
  - Logic-based systems with weights attached to rules, which are combined using inference.
  - Had to be designed carefully to avoid undesirable interactions between different rules.
  - Might deliver incorrect results through overcounting of evidence.
  - Their use is no longer recommended.

Other Approaches (2)

- Dempster-Shafer Theory
  - Allows the representation of *ignorance* as well as uncertainly.
  - Example: If a coin is fair, we assume $P(Heads) = 0.5$. But what if we do not know if the coin is fair? $\rightarrow$ $Bel(Heads) = 0$, $Bel(Tails) = 0$. If the coin is 90% fair, 0.5×0.9, i.e. $Bel(Heads) = 0.45$.

$\rightarrow$ Interval of probabilities is $[0.45, 0.55]$ with the evidence, $[0, 1]$ without.

$\rightarrow$ The notion of utility is not yet well understood in Dempster-Shafer Theory.

Other Approaches (3)

- Fuzzy logic and fuzzy sets
  - A means of representing and working with vagueness, not uncertainty.
  - Example: The car is *fast*.
  - Used especially in control and regulation systems.
  - In such systems, it can be interpreted as an *interpolation technique*.

Summary

- Bayesian Networks allow a compact representation of joint probability distribution.
- Bayesian Networks provide a concise way to represent conditional independence in a domain.
- Inference in Bayesian networks means computing the probability distribution of a set of query variables, given a set of evidence variables.
- Exact inference algorithms such as variable elimination are efficient for poly-trees.
- In complexity of belief network inference depends on the network structure.
- In general, Bayesian network inference is NP-hard.

Foundations of Artificial Intelligence

12. Acting under Uncertainty

Maximizing Expected Utility

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

June 28, 2019

Contents

Lecture Overview

1. Introduction to Utility Theory
2. Choosing Individual Actions
3. Sequential Decision Problems
4. Markov Decision Processes
5. Value Iteration

The Basis of Utility Theory

The utility function rates states and thus formalizes the desirability of a state by the agent.

$U(S)$ denotes the utility of state S for the agent.

A nondeterministic action A can lead to the outcome states $Result_i(A)$. How high is the probability that the outcome state $Result_i(A)$ is reached, if A is executed in the current state with evidence E?

$$\rightarrow P(Result_i(A) \mid Do(A), E)$$

Expected Utility:

$$\boxed{EU(A \mid E) = \sum_i P(Result_i(A) \mid Do(A), E)\, U(Result_i(A))}$$

The principle of maximum expected utility (MEU) says that a rational agent should choose an action that maximizes $EU(A \mid E)$.

Problems with the MEU Principle

$$P(Result_i(A) \mid Do(A), E)$$

requires a complete causal model of the world.

$\rightarrow$ Constant updating of belief networks

$\rightarrow$ NP-complete for Bayesian networks

$$U(Result_i(A))$$

requires search or planning, because an agent needs to know the possible future states in order to assess the worth of the current state ("effect of the state on the future").

Lecture Overview

The Axioms of Utility Theory (1)

Justification of the MEU principle, i.e., maximization of the average utility.

Scenario = Lottery L

- Possible outcomes = possible prizes
- The outcome is determined by chance
- $L = [p_1, C_1; p_2, C_2; \ldots; p_n, C_n]$

Example:
Lottery L with two outcomes, C_1 and C_2:

$$L = [p, C_1; 1 - p, C_2]$$

Preference between lotteries:

$L_1 \succ L_2$	The agent prefers L_1 over L_2
$L_1 \sim L_2$	The agent is indifferent between L_1 and L_2
$L_1 \succsim L_2$	The agent prefers L_1 or is indifferent between L_1 and L_2

The Axioms of Utility Theory (2)

Given lotteries A, B, C

- Orderability
  $(A \succ B) \vee (B \succ A) \vee (A \sim B)$
  An agent should know what it wants: it must either prefer one of the 2 lotteries or be indifferent to both.
- Transitivity
  $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
  Violating transitivity causes irrational behavior: $A \succ B \succ C \succ A$. The agent has A and would pay to exchange it for C. Afterwards it would do the same to exchange C for A. $\rightarrow$ The agent loses money this way.

The Axioms of Utility Theory (3)

- Continuity
  $A \succ B \succ C \Rightarrow \exists p[p, A; 1-p, C] \sim B$
  If some lottery B is between A and C in preference, then there is some probability p for which the agent is indifferent between getting B for sure and the lottery that yields A with probability p and C with probability $1-p$.

- Substitutability
  $A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$
  If an agent is indifferent between two lotteries A and B, then the agent is indifferent beteween two more complex lotteries that are the same except that B is substituted for A in one of them.

The Axioms of Utility Theory (4)

- Monotonicity
  $A \succ B \Rightarrow (p > q \Leftrightarrow [p, A; 1-p, B] \succ [q, A; 1-q, B])$
  If an agent prefers the outcome A, then it must also prefer the lottery that has a higher probability for A.

- Decomposability
  $[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C]$
  Compound lotteries can be reduced to simpler ones using the laws of probability. This has been called the "no fun in gambling"-rule: two consecutive gambles can be reduced to a single equivalent lottery.

Utility Functions and Axioms

The axioms only make statements about preferences.

The existence of a utility function follows from the axioms!

- Utility Principle If an agent's preferences obey the axioms, then there exists a function $U : S \mapsto R$ with
  $U(A) > U(B) \Leftrightarrow A \succ B$
  $U(A) = U(B) \Leftrightarrow A \sim B$
- Expected Utility of a Lottery:
  $U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i \, U(S_i)$
  $\rightarrow$ Since the outcome of a nondeterministic action is a lottery, an agent can act rationally only by following the Maximum Expected Utility (MEU) principle.

How do we design utility functions that cause the agent to act as desired?

Assessing Utilities

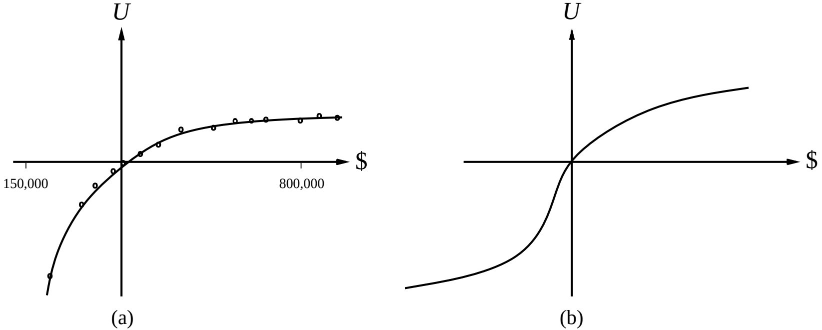
The scale of a utility function can be chosen arbitrarily. We therefore can define a "normalized" utility of a lottery S:

- "Best possible prize" $U(S) = u_{max} = 1$
- "Worst catastrophe" $U(S) = u_{min} = 0$

Given a utility scale between u_{min} and u_{max} we can asses the utility of any particular outcome S by asking the agent to choose between S and a standard lottery $[p, u_{max}; 1-p, u_{min}]$. We adjust p until they are equally preferred.
Then, p is the utility of S. This is done for each outcome S to determine $U(S)$.

Possible Utility Functions

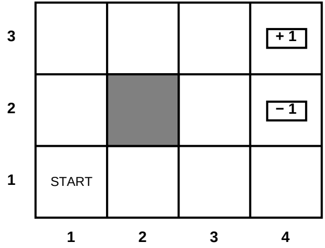
From economic models: The value of money



left: utility from empirical data; right: typical utility function over the full range.

Lecture Overview

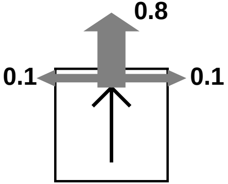
Sequential Decision Problems (1)



- Beginning in the start state the agent must choose an action at each time step.
- The interaction with the environment terminates if the agent reaches one of the goal states (4,3) (reward of $+1$) or (4,2) (reward -1). Each other location has a reward of -.04.
- In each location the available actions are Up, $Down$, $Left$, $Right$.

Sequential Decision Problems (2)

- Deterministic version: All actions always lead to the next square in the selected direction, except that moving into a wall results in no change in position.
- Stochastic version: Each action achieves the intended effect with probability 0.8, but the rest of the time, the agent moves at right angles to the intended direction.



Lecture Overview

Markov Decision Problem (MDP)

An MDP for an accessible, stochastic environment is defined by

- Set of states S
- Set of actions A
- Transition model $P(s' \mid s, a)$, with $s, s' \in S$ and $a \in A$
- Reward function $R(s)$, with $s \in S$

Transition model: $P(s' \mid s, a)$ is the probability that state s' is reached, if action a is executed in state s.

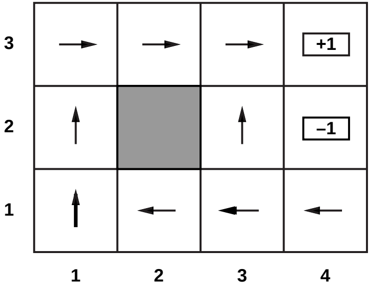
Policy: Complete mapping π that specifies for each state $s \in S$ which action $\pi(s) \in A$ to take.

Wanted: The **optimal policy π^*** is the policy that **maximizes the future expected reward**.

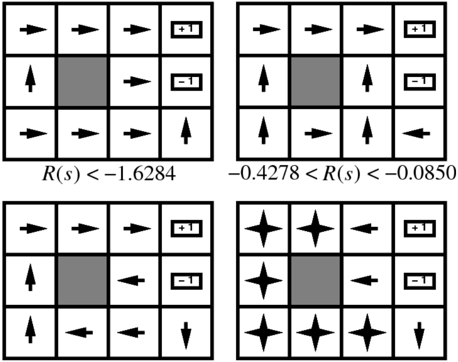
Optimal Policies (1)

- Given the optimal policy, the agent uses its current percept that tells it its current state.
- It then executes the action $\pi^*(s)$.
- We obtain a simple reflex agent that is computed from the information used for a utility-based agent.

Optimal policy for stochastic MDP with $R(s) = -0.04$:



Optimal Policies (2)



$R(s) < -1.6284$ $\quad$ $-0.4278 < R(s) < -0.0850$

$-0.0221 < R(s) < 0$ $\quad$ $R(s) > 0$

Optimal policy changes with choice of transition costs $R(s)$.
How to compute optimal policies?

Finite and Infinite Horizon Problems

- Performance of the agent is measured by the sum of rewards for the states visited.
- To determine an optimal policy we will first calculate the utility of each state and then use the state utilities to select the optimal action for each state.
- The result depends on whether we have a finite or infinite horizon problem.
- Utility function for state sequences: $U_h([s_0, s_1, \ldots, s_n])$
- Finite horizon: $U_h([s_0, s_1, \ldots, s_{N+k}]) = U_h([s_0, s_1, \ldots, s_N])$ for all $k > 0$.
- For finite horizon problems the optimal policy depends on the current state and the remaining steps to go. It therefore depends on time and is called nonstationary.
- In infinite horizon problems the optimal policy only depends on the current state and therefore is stationary.

Assigning Utilities to State Sequences

- For stationary systems there are two coherent ways to assign utilities to state sequences.
- Additive rewards:
  $U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$
- Discounted rewards:
  $U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$
- The term $\gamma \in [0, 1[$ is called the discount factor.
- With discounted rewards the utility of an infinite state sequence is always finite. The discount factor expresses that future rewards have less value than current rewards.

Lecture Overview

1. Introduction to Utility Theory
2. Choosing Individual Actions
3. Sequential Decision Problems
4. Markov Decision Processes
5. Value Iteration

Utilities of States

- The utility of a state depends on the utility of the state sequences that follow it.
- Let $U^\pi(s)$ be the utility of a state under policy π.
- Let s_t be the state of the agent after executing π for t steps. Thus, the utility of s under π is

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right]$$

- The true utility $U(s)$ of a state is $U^{\pi^*}(s)$.
- $R(s)$ is the short-term reward for being in s and
  $U(s)$ is the long-term total reward from s onwards.

Example

The utilities of the states in our 4×3 world with $\gamma = 1$ and $R(s) = -0.04$ for non-terminal states:

3	**0.812**	**0.868**	**0.918**	**+ 1**
2	**0.762**		**0.660**	**–1**
1	**0.705**	**0.655**	**0.611**	**0.388**
	1	**2**	**3**	**4**

Choosing Actions using the Maximum Expected Utility Principle

The agent simply chooses the action that maximizes the expected utility of the subsequent state:

$$\pi(s) = \operatorname*{argmax}_a \sum_{s'} P(s' \mid s, a)\, U(s')$$

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:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a)\, U(s')$$

Bellman-Equation

- The equation

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a)\, U(s')$$

is also called the Bellman-Equation.

Bellman-Equation: Example

- In our 4×3 world the equation for the state (1,1) is

$$
\begin{aligned}
U(1,1) = -0.04 + \gamma \max\{ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && (Up)\\
& 0.9U(1,1) + 0.1U(1,2), && (Left)\\
& 0.9U(1,1) + 0.1U(2,1), && (Down)\\
& 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\} && (Right)\\
= -0.04 + \gamma \max\{ & 0.8 \cdot 0.762 + 0.1 \cdot 0.655 + 0.1 \cdot 0.705, && (Up)\\
& 0.9 \cdot 0.705 + 0.1 \cdot 0.762, && (Left)\\
& 0.9 \cdot 0.705 + 0.1 \cdot 0.655, && (Down)\\
& 0.8 \cdot 0.655 + 0.1 \cdot 0.762 + 0.1 \cdot 0.705\} && (Right)\\
= -0.04 + 1.0\,(& 0.6096 + 0.0655 + 0.0705), \quad (Up) = -0.04 + 0.7456 = 0.7056
\end{aligned}
$$

- Up is the optimal action in (1,1).

3	**0.812**	**0.868**	**0.918**	**+1**
2	**0.762**		**0.660**	**–1**
1	**0.705**	**0.655**	**0.611**	**0.388**
	1	**2**	**3**	**4**

Value Iteration (1)

An algorithm to calculate an optimal strategy.

Basic Idea: Calculate the utility of each state. Then use the state utilities to select an optimal action for each state.

A sequence of actions generates a branch in the tree of possible states (histories). A utility function on histories U_h is separable iff there exists a function f such that

$$U_h([s_0, s_1, \ldots, s_n]) = f(s_0, U_h([s_1, \ldots, s_n]))$$

The simplest form is an additive reward function R:

$$U_h([s_0, s_1, \ldots, s_n]) = R(s_0) + U_h([s_1, \ldots, s_n]))$$

In the example, $R((4,3)) = +1$, $R((4,2)) = -1$, $R(other) = -1/25$.

Value Iteration (2)

If the utilities of the terminal states are known, then in certain cases we can reduce an n-step decision problem to the calculation of the utilities of the terminal states of the $(n-1)$-step decision problem.

$\rightarrow$ Iterative and efficient process

Problem: Typical problems contain cycles, which means the length of the histories is potentially infinite.

Solution: Use

$$U_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a)\, U_t(s')$$

where $U_t(s)$ is the utility of state s after t iterations.

Remark: As $t \rightarrow \infty$, the utilities of the individual states converge to stable values.

Value Iteration (3)

- The Bellman equation is the basis of value iteration.
- Because of the max-operator the n equations for the n states are nonlinear.
- We can apply an iterative approach in which we replace the equality by an assignment:

$$U'(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a)\, U(s')$$

The Value Iteration Algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
rewards $R(s)$, discount γ
ϵ, the maximum error allowed in the utility of any state
local variables: U, U', vectors of utilities for states in S, initially zero
δ, 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**
$U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \epsilon(1-\gamma)/\gamma$
return U

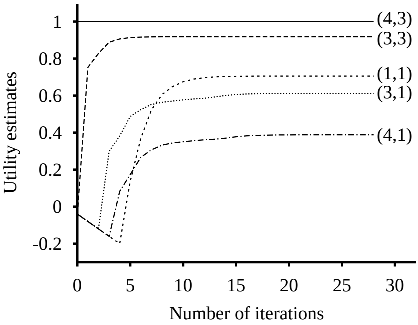
Convergence of Value Iteration

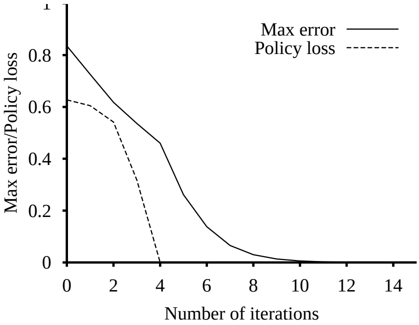
- Since the algorithm is iterative we need a criterion to stop the process if we are close enough to the correct utility.
- In principle we want to limit the policy loss $\|U^{\pi_t} - U\|$ that is the most the agent can lose by executing π_t.
- It can be shown that value iteration converges and that

  if $\quad \|U_{t+1} - U_t\| < \epsilon(1-\gamma)/\gamma \quad$ then $\quad \|U_{t+1} - U\| < \epsilon$

  if $\quad \|U_t - U\| < \epsilon \quad$ then $\quad \|U^{\pi_t} - U\| < 2\epsilon\gamma/(1-\gamma)$
- The value iteration algorithm yields the optimal policy π^*.

Application Example





In practice the policy often becomes optimal before the utility has converged.

Policy Iteration

- Value iteration computes the optimal policy even at a stage when the utility function estimate has not yet converged.
- If one action is better than all others, then the exact values of the states involved need not to be known.
- Policy iteration alternates the following two steps beginning with an initial policy π_0:
  - Policy evaluation: given a policy π_t, calculate $U_t = U^{\pi_t}$, the utility of each state if π_t were executed.
  - Policy improvement: calculate a new maximum expected utility policy π_{t+1} according to

$$\pi_{t+1}(s) = \operatorname*{argmax}_a \sum_{s'} P(s' \mid s, a)\, U_t(s')$$

The Policy Iteration Algorithm

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Summary

- Rational agents can be developed on the basis of a probability theory and a utility theory.
- Agents that make decisions according to the axioms of utility theory possess a utility function.
- Sequential problems in uncertain environments (MDPs) can be solved by calculating a policy.
- Value iteration is a process for calculating optimal policies.

Foundations of Artificial Intelligence

13. Machine Learning

Learning from Observations

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

July 3, 2019

Learning

- What is learning?

  An agent learns when it improves its performance w.r.t. a specific task with experience.

  $\rightarrow$ E.g., game programs

- Why learn?

  $\rightarrow$ Engineering, philosophy, cognitive science

  $\rightarrow$ Data Mining (discovery of new knowledge through data analysis)

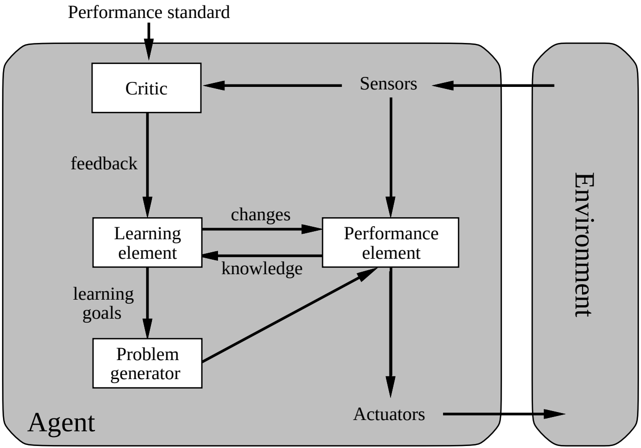
No intelligence without learning!

Contents

Lecture Overview

The Learning Agent

So far an agent's percepts have only served to help the agent choose its actions. Now they will also serve to improve future behavior.



Building Blocks of the Learning Agent

Performance element: Processes percepts and chooses actions.
$\rightarrow$ Corresponds to the agent model we have studied so far.

Learning element: Carries out improvements
$\rightarrow$ requires self knowledge and feedback on how the agent is doing in the environment.

Critic: Evaluation of the agent's behavior based on a given external behavioral measure
$\rightarrow$ feedback.

Problem generator: Suggests explorative actions that lead the agent to new experiences.

The Learning Element

Its design is affected by four major issues:

- Which components of the performance element are to be learned?
- What representation should be chosen?
- What form of feedback is available?
- Which prior information is available?

Lecture Overview

1. The learning agent
2. Types of learning
3. Decision trees

Types of Feedback During Learning

The type of feedback available for learning is usually the most important factor in determining the nature of the learning problem.

Supervised learning: Involves learning a function from examples of its inputs and outputs.

Unsupervised learning: The agent has to learn patterns in the input when no specific output values are given.

Reinforcement learning: The most general form of learning in which the agent is not told what to do by a teacher. Rather it must learn from a reinforcement or reward. It typically involves learning how the environment works.

Supervised Learning

An example is a pair $(x, f(x))$. The complete set of examples is called the training set.

Pure inductive inference: for a collection of examples for f, return a function h (hypothesis) that approximates f.

The function h typically is member of a hypothesis space $\mathbf{H}$.

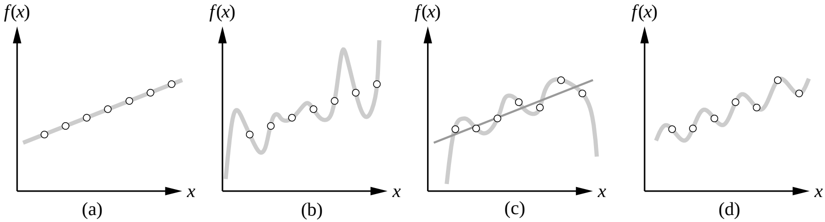
A good hypothesis should generalize the data well, i.e., will predict unseen examples correctly.

A hypothesis is consistent with the data set if it agrees with all the data.

How do we choose from among multiple consistent hypotheses?

Ockham's razor: prefer the simplest hypothesis consistent with the data.

Example: Fitting a Function to a Data Set



(a) consistent hypothesis that agrees with all the data

(b) degree-7 polynomial that is also consistent with the data set

(c) data set that can be approximated consistently with a degree-6 polynomial

(d) sinusoidal exact fit to the same data

Lecture Overview

1. The learning agent
2. Types of learning
3. Decision trees

Decision Trees

Input: Description of an object or a situation through a set of attributes.

Output: a decision, that is the predicted output value for the input.

Both, input and output can be discrete or continuous.

Discrete-valued functions lead to classification problems.

Learning a continuous function is called regression.

Boolean Decision Tree

Input: set of vectors of input attributes X and a single Boolean output value y (goal predicate).

Output: Yes/No decision based on a goal predicate.

Goal of the learning process: Definition of the goal predicate in the form of a decision tree.

Boolean decision trees represent Boolean functions.

Properties of (Boolean) Decision Trees:

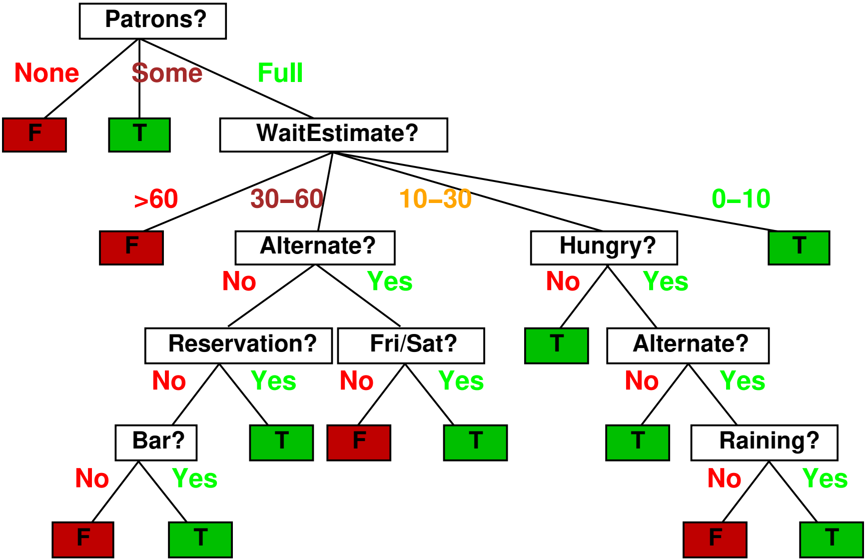
- An internal node of the decision tree represents a test of a property.
- Branches are labeled with the possible values of the test.
- Each leaf node specifies the Boolean value to be returned if that leaf is reached.

When to Wait for Available Seats at a Restaurant

Goal predicate: $WillWait$
Test predicates:

- $Patrons$: How many guests are there? ($none$, $some$, $full$)
- $WaitEstimate$: How long do we have to wait? (0-10, 10-30, 30-60, $>$60)
- $Alternate$: Is there an alternative? (T/F)
- $Hungry$: Am I hungry? (T/F)
- $Reservation$: Have I made a reservation? (T/F)
- Bar: Does the restaurant have a bar to wait in? (T/F)
- Fri/Sat: Is it Friday or Saturday? (T/F)
- $Raining$: Is it raining outside? (T/F)
- $Price$: How expensive is the food? (\$, \$\$, \$\$\$)
- $Type$: What kind of restaurant is it? ($French$, $Italian$, $Thai$, $Burger$)

Restaurant Example (Decision Tree)



Expressiveness of Decision Trees

Each decision tree hypothesis for the $WillWait$ goal predicate can be seen as an assertion of the form

$$\forall s\, WillWait(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \ldots \vee P_n(s))$$

where each $P_i(s)$ is the conjunction of tests along a path from the root of the tree to a leaf with a positive outcome.

Any Boolean function can be represented by a decision tree.

Limitation: All tests always involve only one object and the language of traditional decision trees is inherently propositional.

$$\exists r_2 NearBy(r_2, s) \wedge Price(r, p) \wedge Price(r_2, p_2) \wedge Cheaper(p_2, p)$$

cannot be represented as a test.

We could always add another test called $CheaperRestaurantNearby$, but a decision tree with all such attributes would grow exponentially.

Compact Representations

For every Boolean function we can construct a decision tree by translating every row of a truth table to a path in the tree.

This can lead to a tree whose size is exponential in the number of attributes.

Although decision trees can represent functions with smaller trees, there are functions that require an exponentially large decision tree:

Parity function: $$p(x) = \begin{cases} 1 & \text{even number of inputs are 1} \\ 0 & \text{otherwise} \end{cases}$$

Majority function: $$m(x) = \begin{cases} 1 & \text{half of the inputs are 1} \\ 0 & \text{otherwise} \end{cases}$$

There is no consistent representation that is compact for all possible Boolean functions.

The Training Set of the Restaurant Example

Classification of an example = Value of the goal predicate

$T \rightarrow$ positive example
$F \rightarrow$ negative example

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

Inducing Decision Trees from Examples

- Naïve solution: we simply construct a tree with one path to a leaf for each example.
- In this case we test all the attributes along the path and attach the classification of the example to the leaf.
- Whereas the resulting tree will correctly classify all given examples, it will not say much about other cases.
- It just memorizes the observations and does not generalize.

Inducing Decision Trees from Examples (2)

- Smallest solution: applying Ockham's razor we should instead find the smallest decision tree that is consistent with the training set.
- Unfortunately, for any reasonable definition of smallest finding the smallest tree is intractable.
- Dilemma:

smallest — ? — simplest

intractable — no learning

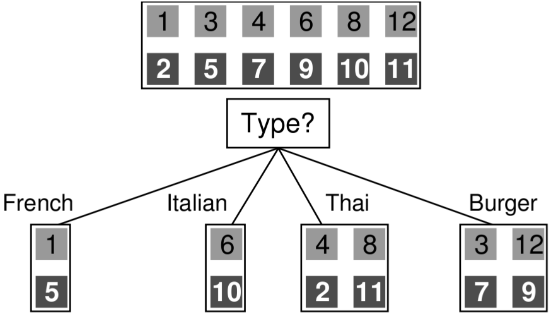
- How can we learn decision trees that are small and generalize well?

Idea of Decision Tree Learning

Divide and Conquer approach:

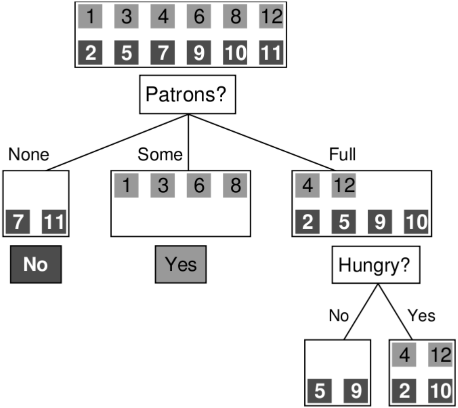
- Choose an (or better: the best) attribute.
- Split the training set into subsets each corresponding to a particular value of that attribute.
- Now that we have divided the training set into several smaller training sets, we can recursively apply this process to the smaller training sets.

Splitting Examples (1)



- Type is a poor attribute, since it leaves us with four subsets each of them containing the same number of positive and negative examples.
- It does not reduce the problem complexity.

Splitting Examples (2)



- Patrons is a better choice, since if the value is $None$ or $Some$, then we are left with example sets for which we can answer definitely (T or F).
- Only for the value $Full$ we are left with a mixed set of examples.
- One potential next choice is $Hungry$.

Recursive Learning Process

In each recursive step there are four cases to consider:

- Positive and negative examples: choose a new attribute.
- Only positive (or only negative) examples: done (answer is T or F).
- No examples: there was no example with the desired property. Answer T if the majority of the parent node's examples is positive, otherwise F.
- No attributes left, but there are still examples with different classifications: there were errors in the data ($\rightarrow$ NOISE) or the attributes do not give sufficient information. Answer T if the majority of examples is positive, otherwise F.

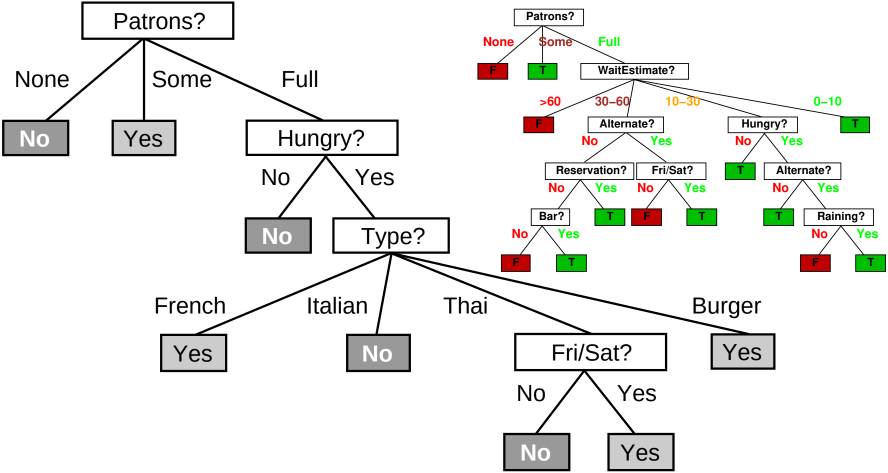
The Decision Tree Learning Algorithm

function DTL(*examples, attributes, default*) **returns** a decision tree

    if *examples* is empty **then return** *default*
    else if all *examples* have the same classification **then return** the classification
    else if *attributes* is empty **then return** MODE(*examples*)
    else
        best ← CHOOSE-ATTRIBUTE(*attributes*, *examples*)
        tree ← a new decision tree with root test *best*
        for each value v_i of *best* **do**
            $examples_i \leftarrow \{$elements of *examples* with $best = v_i\}$
            $subtree \leftarrow \mathrm{DTL}(examples_i, attributes - best, \textsc{Mode}(examples))$
            add a branch to *tree* with label v_i and subtree *subtree*
        return *tree*

Application to the Restaurant Data

Original tree:



Properties of the Resulting Tree

- The resulting tree is considerably simpler than the one originally given (and from which the training examples were generated).
- The learning algorithm outputs a tree that is consistent with all examples it has seen.
- The tree does not necessarily agree with the correct function.
- For example, it suggests not to wait if we are not hungry. If we are, there are cases in which it tells us to wait.
- Some tests ($Raining$, $Reservation$) are not included since the algorithm can classify the examples without them.

Choosing Attribute Tests

$\text{CHOOSE-ATTRIBUTE}(attribs, examples)$

- One goal of decision tree learning is to select attributes that minimize the depth of the final tree.
- The perfect attribute divides the examples into sets that are all positive or all negative.
- $Patrons$ is not perfect but fairly good.
- $Type$ is useless since the resulting proportion of positive and negative examples in the resulting sets are the same as in the original set.
- What is a formal measure of "fairly good" and "useless"?

Evaluation of Attributes

Tossing a coin: What value has prior information about the outcome of the toss when the stakes are \$1 and the winnings \$1?

- Rigged coin with 99% heads and 1% tails.
  (average winnings per toss $\approx \$0.98$)

  → Worth of information about the outcome is less than ≈ \$0.02.

- Fair coin

  → Value of information about the outcome is less than \$1.

  → The less we know about the outcome, the more valuable the prior information.

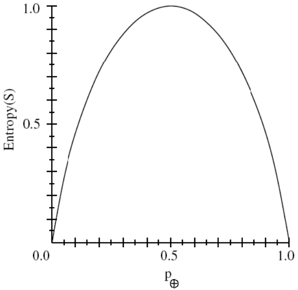
Information Provided by an Attribute

- One suitable measure is the expected amount of information provided by the attribute.
- Information theory measures information content in bits.
- One bit is enough to answer a yes/no question about which one has no idea (fair coin flip).
- In general, if the possible answers v_i have probabilities $P(v_i)$, the information content is given as the entropy

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2(P(v_i))$$

Examples

$I\left(\frac{1}{2}, \frac{1}{2}\right)$



$I(1, 0)$

$I(0, 1)$

Attribute Selection (1)

Suppose the training set E consists of p positive and n negative examples:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = \frac{p}{p+n}\log_2\left(\frac{p+n}{p}\right) + \frac{n}{p+n}\log_2\left(\frac{p+n}{n}\right)$$

The value of an attribute A depends on the additional information that we still need to collect after we selected it.

Suppose A divides the training set E into subsets E_i, $i = 1, \ldots, v$.

Every subset has $I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$

A random example has value i with probability $\frac{p_i+n_i}{p+n}$

Attribute Selection (2)

→ The average information content after choosing A is

$$R(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

→ The information gain from choosing A is

$$Gain(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - R(A)$$

Heuristic in CHOOSE-ATTRIBUTE is to select the attribute with the largest gain.

Examples:

$Gain(Patrons) = 1 - [\frac{2}{12}I(0,1) + \frac{4}{12}I(1,0) + \frac{6}{12}I(\frac{2}{6},\frac{4}{6})] \approx 0.541$

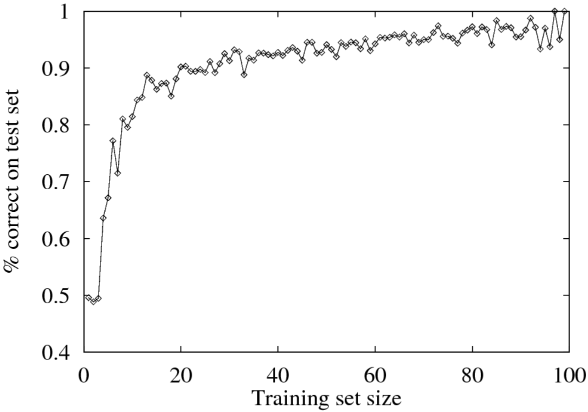
$Gain(Type) = 1 - [\frac{2}{12}I(\frac{1}{2},\frac{1}{2}) + \frac{2}{12}I(\frac{1}{2},\frac{1}{2}) + \frac{4}{12}I(\frac{2}{4},\frac{2}{4}) + \frac{4}{12}I(\frac{2}{4},\frac{2}{4})] = 0$

Assessing the Performance of the Learning Algorithm

Methodology for assessing the power of prediction:

- Collect a large number of examples.
- Divide it into two disjoint sets: the training set and the test set.
- Use the training set to generate h.
- Measure the percentage of examples of the test set that are correctly classified by h.
- Repeat the process for randomly-selected training sets of different sizes.

Learning Curve for the Restaurant Example



As the training set grows, the prediction quality increases.

Important Strategy for Designing Learning Algorithms

- The training and test sets must be kept separate.
- Common error: Changing the algorithm after running a test, and then testing it with training and test sets from the same basic set of examples. By doing this, knowledge about the test set gets stored in the algorithm, and the training and test sets are no longer independent.

Summary: Decision Trees

- One possibility for representing (Boolean) functions.
- Decision trees can be exponential in the number of attributes.
- It is often too difficult to find the minimal DT.
- One method for generating DTs that are as flat as possible is based on ranking the attributes.
- The ranks are computed based on the information gain.

Foundations of Artificial Intelligence

14. Deep Learning

Learning from Raw Data

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

July 10, 2019

Motivation: Deep Learning in the News

The New York Times

Science

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS | OPINION

ENVIRONMENT SPACE & COSMOS

Scientists See Promise in Deep-Learning

A voice recognition program translated a speech given by Richard F. Rashid Chinese.

By JOHN MARKOFF
Published: November 23, 2012

Using an artificial intelligence technique inspired by theories al the brain recognizes patterns, technology companies are reportin gains in fields as diverse as computer vision, speech recognitior

THE NEW YORKER

NEWS | CULTURE | BOOKS & FICTION | SCIENCE & TECH | BUSINESS | HUMOR | MAGAZINE | ARCHIVE | SUBSCRIBE

NOVEMBER 25, 2012

IS "DEEP LEARNING" A REVOLUTION IN ARTIFICIAL INTELLIGENCE?

BY GARY MARCUS

Share | Tweet | g+1

Can a new technique known as deep learning revolutionize artificial intelligence, as yesterday's front-page article at the New York *Times* suggests? There is good reason to be excited about deep learning, a sophisticated "machine learning" algorithm that far exceeds many of its predecessors in its abilities to recognize syllables and images. But there's also good reason to be skeptical. While the *Times* reports that "advances in an artificial intelligence technology that can recognize patterns offer

MIT Technology Review

10 BREAKTHROUGH TECHNOLOGIES 2013

Introduction | The 10 Technologies
Past Years

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.

Lecture Overview

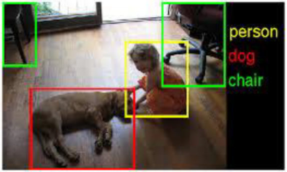
Lecture Overview

1. Motivation: Why is Deep Learning so Popular?
2. Representation Learning and Deep Learning
3. Multilayer Perceptrons
4. Overview of Some Advanced Topics
5. Limitations
6. Wrapup

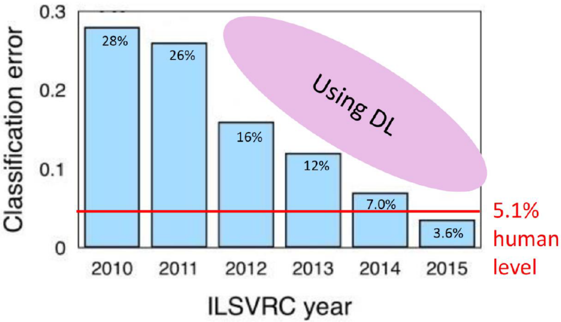
Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in computer vision

Object recognition



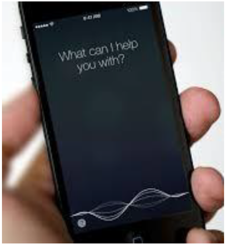
Self-driving cars



Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in speech recognition

Speech recognition



Auto-Translator

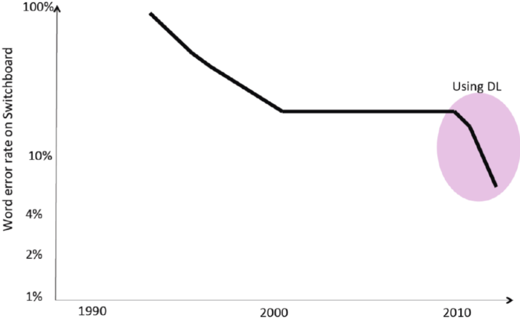
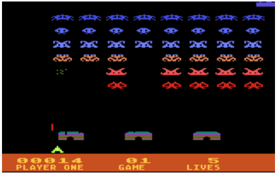


Image credit: Yoshua Bengio (data from Microsoft speech group)

- Excellent empirical results, e.g., in reasoning in games
  - Superhuman performance in playing Atari games
    [Mnih et al, Nature 2015]



  - Beating the world's best Go player
    [Silver et al, Nature 2016]

An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience

An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience
- If the task changes, we simply re-train

An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience
- If the task changes, we simply re-train
- We can construct computer systems that are too complex for us to understand anymore ourselves. . .
  - E.g., deep neural networks have millions of weights.
  - E.g., AlphaGo, the system that beat world champion Lee Sedol
    - David Silver, lead author of AlphaGo cannot say why a move is good
    - Paraphrased: "You would have to ask a Go expert."

An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- Learning from data / experience may be more human-like
  - Babies develop an intuitive understanding of physics in their first 2 years
  - Formal reasoning and logic comes much later in development

An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- Learning from data / experience may be more human-like
  - Babies develop an intuitive understanding of physics in their first 2 years
  - Formal reasoning and logic comes much later in development
- Learning enables fast reaction times
  - It might take a long time to train a neural network
  - But predicting with the network is very fast
  - Contrast this to running a planning algorithm every time you want to act

Lecture Overview

Some definitions

Representation learning

"a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification"

Some definitions

Representation learning

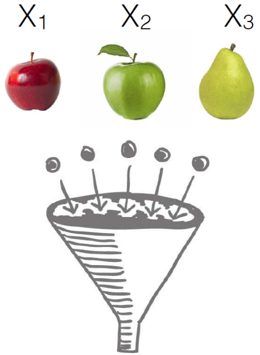
"a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification"

Deep learning

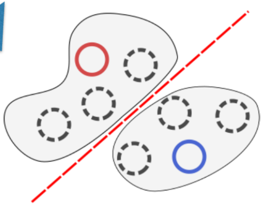
"representation learning methods with multiple levels of representation, obtained by composing simple but nonlinear modules that each transform the representation at one level into a [...] higher, slightly more abstract (one)"

(LeCun et al., 2015)

- Standard machine learning algorithms are based on high-level attributes or features of the data
- E.g., the binary attributes we used for decisions trees
- This requires (often substantial) feature engineering

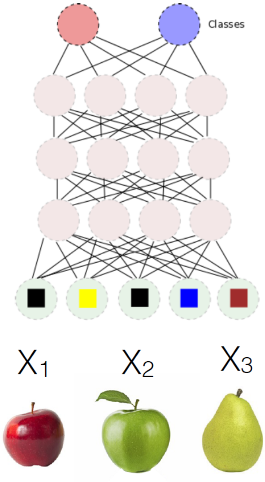


	Merkmale
X_1	rot, 3.5 cm
X_2	grün, 4 cm
X_3	grün, 10 cm

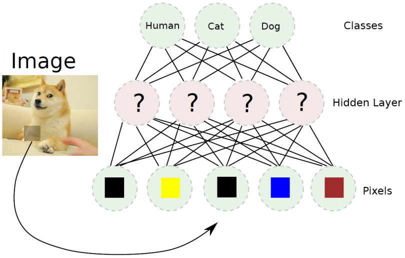


Representation Learning Pipeline

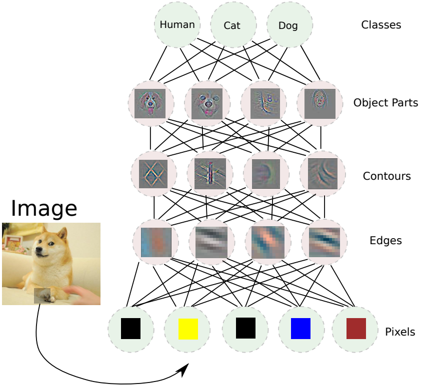
- Jointly learn features and classifier, directly from raw data
- This is also referrred to as end-to-end learning



Shallow vs. Deep Learning

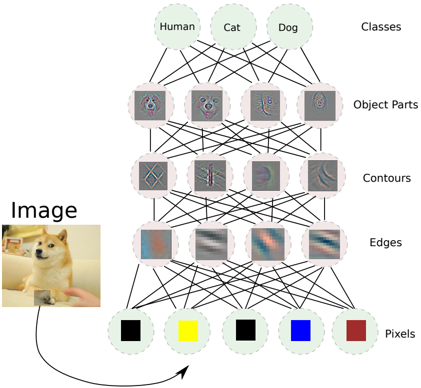


Shallow vs. Deep Learning



- Deep Learning: learning a hierarchy of representations that build on each other, from simple to complex

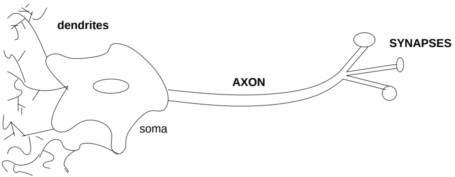
Shallow vs. Deep Learning



- Deep Learning: learning a hierarchy of representations that build on each other, from simple to complex
- Quintessential deep learning model: Multilayer Perceptrons

Biological Inspiration of Artificial Neural Networks

- Dendrites input information to the cell
- Neuron fires (has action potential) if a certain threshold for the voltage is exceeded
- Output of information by axon
- The axon is connected to dentrites of other cells via synapses
- Learning: adaptation of the synapse's efficiency, its synaptical weight



A Very Brief History of Neural Networks

- Neural networks have a long history
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)

A Very Brief History of Neural Networks

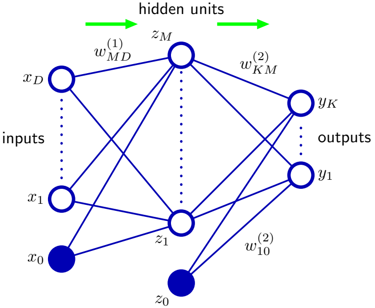
- Neural networks have a long history
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)
- Alternative theoretically motivated methods outperformed NNs
  - Exaggerated expectations: "It works like the brain" (No, it does not!)

A Very Brief History of Neural Networks

- Neural networks have a long history
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)
- Alternative theoretically motivated methods outperformed NNs
  - Exaggerated expectations: "It works like the brain" (No, it does not!)
- Why the sudden success of neural networks in the last 5 years?
  - Data: Availability of massive amounts of labelled data
  - Compute power: Ability to train very large neural networks on GPUs
  - Methodological advances: many since first renewed popularization

Lecture Overview

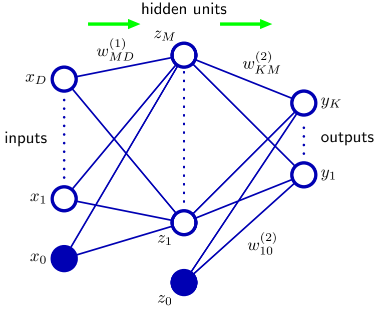
Multilayer Perceptrons



[figure from Bishop, Ch. 5]

- Network is organized in layers
  - Outputs of k-th layer serve as inputs of $k+1$th layer
- Each layer k only does quite simple computations:
  - Linear function of previous layer's outputs z_{k-1}: $a_k = W_k z_{k-1} + b_k$
  - Nonlinear transformation $z_k = h_k(a_k)$ through activation function h_k

Multilayer Perceptrons



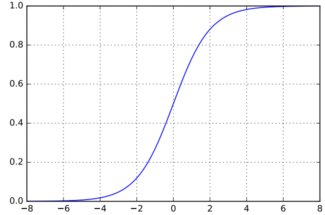
[figure from Bishop, Ch. 5]

- Network is organized in layers
  - Outputs of k-th layer serve as inputs of $k+1$th layer
- Each layer k only does quite simple computations:
  - Linear function of previous layer's outputs z_{k-1}: $a_k = W_k z_{k-1} + b_k$
  - Nonlinear transformation $z_k = h_k(a_k)$ through activation function h_k
- Parameters/weights w of the network: all W_k, b_k, flattened into a single vector

Activation Functions - Examples

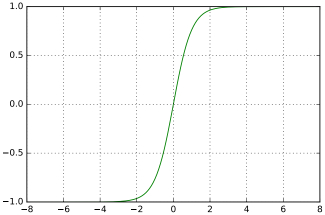
Logistic sigmoid activation function:

$$h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$



Logistic hyperbolic tangent activation function:

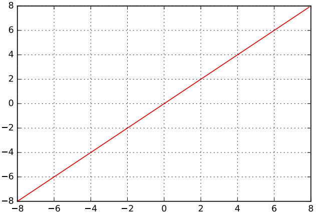
$$\begin{aligned} h_{tanh}(a) &= \tanh(a) \\ &= \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} \} \end{aligned}$$



Activation Functions - Examples (cont.)

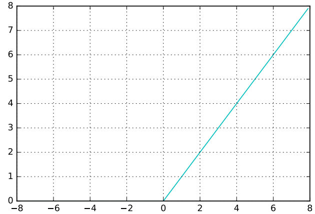
Linear activation function:

$$h_{linear}(a) = a$$



Rectified Linear (ReLU) activation function:

$$h_{relu}(a) = \max(0, a)$$



Output layer and loss functions

- For regression:
  - Single output neuron with linear activation function

$$\hat{y}(x, w) = h_{linear}(a) = a$$

  - Loss function: e.g., squared error:

$$L(w) = \frac{1}{2} \sum_{n=1}^{N} \{\hat{y}(x_n, w) - y_n\}^2$$

Output layer and loss functions

- For regression:
  - Single output neuron with linear activation function

$$\hat{y}(x, w) = h_{linear}(a) = a$$

  - Loss function: e.g., squared error:

$$L(w) = \frac{1}{2} \sum_{n=1}^{N} \{\hat{y}(x_n, w) - y_n\}^2$$

- For classification:
  - Single output unit with, e.g., logistic activation function:

$$\hat{y}(x, w) = h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$

  - Loss function: negative log likelihood of the data under the predictive distribution this specifies; (aka cross entropy):

$$L(w) = -\sum_{n=1}^{N} \{y_n \ln \hat{y}_n + (1 - y_n) \ln(1 - \hat{y}_n)\}$$

Optimizing a loss / error function

- Given training data $\mathcal{D} = \langle (x_i, y_i) \rangle_{i=1}^{N}$ and topology of an MLP
- Task: adapt weights w to minimize the loss:

$$\underset{w}{minimize}\ L(w; \mathcal{D})$$

Optimizing a loss / error function

- Given training data $\mathcal{D} = \langle (x_i, y_i) \rangle_{i=1}^{N}$ and topology of an MLP
- Task: adapt weights w to minimize the loss:

$$\underset{w}{minimize}\ L(w; \mathcal{D})$$

- We optimize this function by gradient-based optimization
  - We can compute gradients of $L(w; \mathcal{D})$
    - Efficiently, using a technique called backpropagation

Optimizing a loss / error function

- Given training data $\mathcal{D} = \langle (x_i, y_i) \rangle_{i=1}^{N}$ and topology of an MLP
- Task: adapt weights w to minimize the loss:

$$\underset{w}{minimize}\ L(w; \mathcal{D})$$

- We optimize this function by gradient-based optimization
  - We can compute gradients of $L(w; \mathcal{D})$
    - Efficiently, using a technique called backpropagation
  - Stochastic gradient descent (SGD)
    - We can use small batches of the data, i.e., $L(w; \mathcal{D}_{batch})$
    - This yields approximate gradients quickly

Lecture Overview

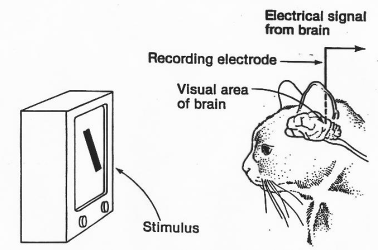
1. Motivation: Why is Deep Learning so Popular?
2. Representation Learning and Deep Learning
3. Multilayer Perceptrons
4. Overview of Some Advanced Topics
5. Limitations
6. Wrapup

Lecture Overview

Historical context and inspiration from Neuroscience

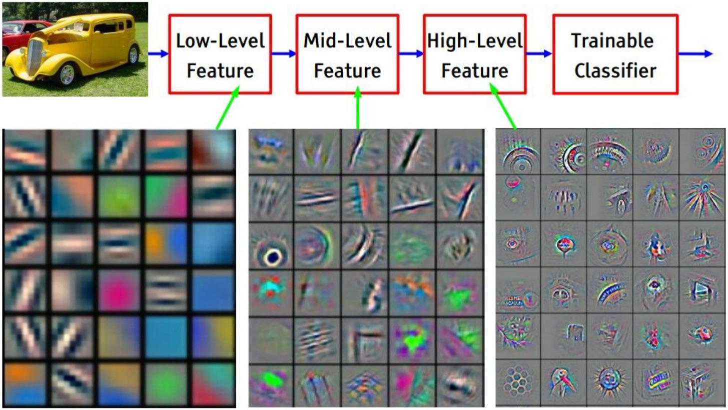
Hubel & Wiesel (Nobel prize 1981) found in several studies in the 1950s and 1960s:

- Visual cortex has feature detectors (e.g., cells with preference for edges with specific orientation)
  - edge location did not matter
- Simple cells as local feature detectors
- Complex cells pool responses of simple cells
- There is a feature hierarchy



Learned feature hierarchy

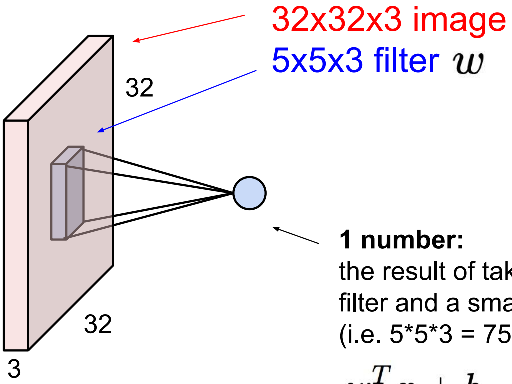
[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[slide credit: Andrej Karpathy]

Convolutions illustrated

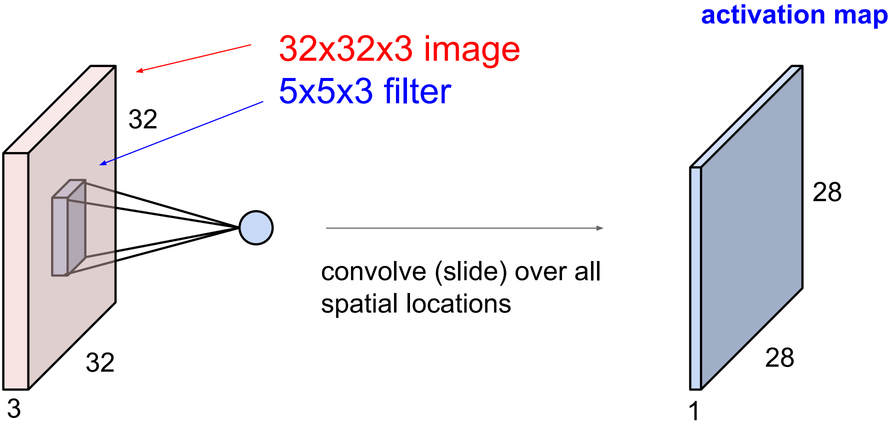


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

[slide credit: Andrej Karpathy]

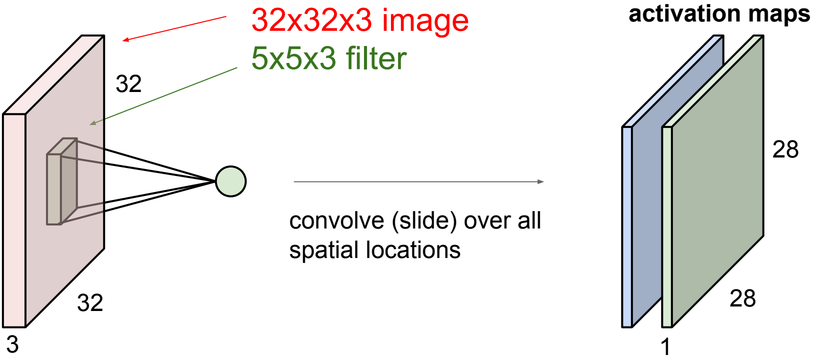
Convolutions illustrated (cont.)



[slide credit: Andrej Karpathy]

Convolutions – several filters

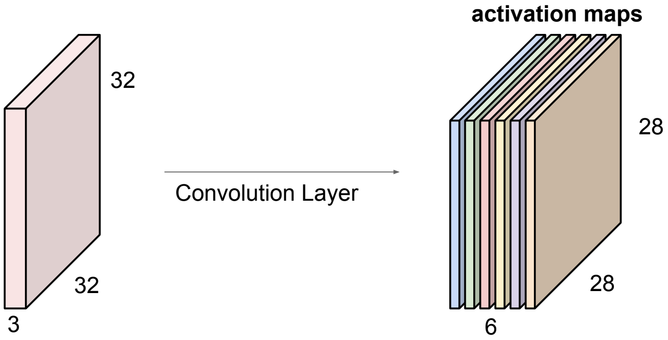
consider a second, green filter



[slide credit: Andrej Karpathy]

Convolutions – several filters

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

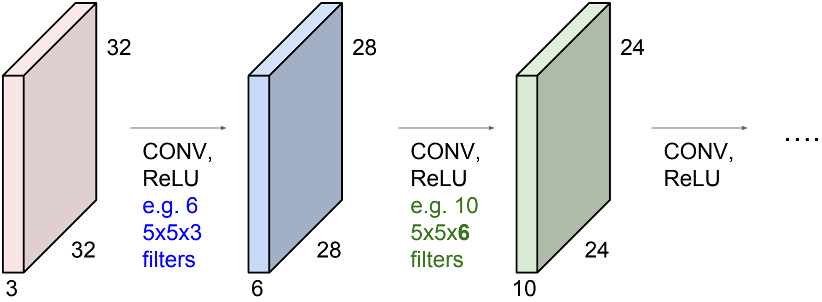


We stack these up to get a "new image" of size 28x28x6!

[slide credit: Andrej Karpathy]

Stacking several convolutional layers

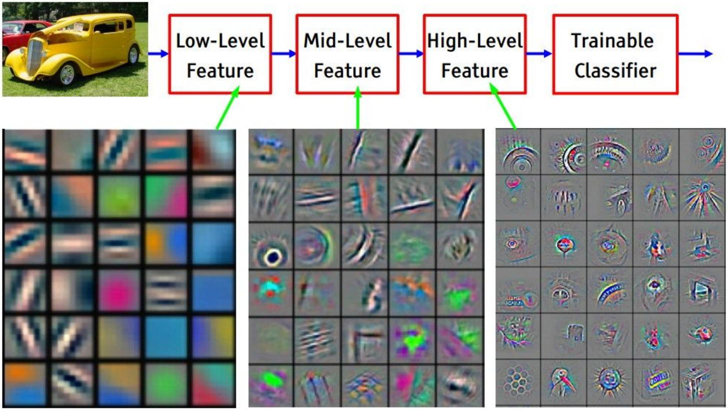
Convolutional layers stacked in a ConvNet



[slide credit: Andrej Karpathy]

Learned feature hierarchy

[From recent Yann LeCun slides]

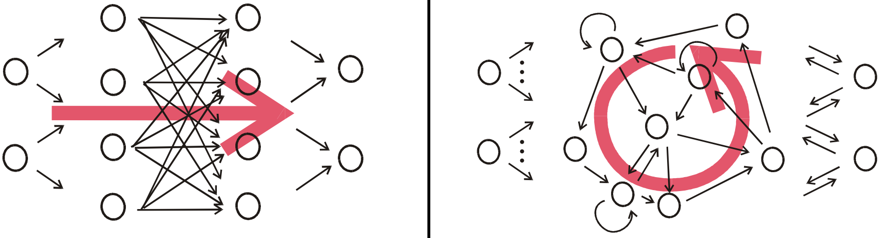


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[slide credit: Andrej Karpathy]

Lecture Overview

Feedforward vs Recurrent Neural Networks



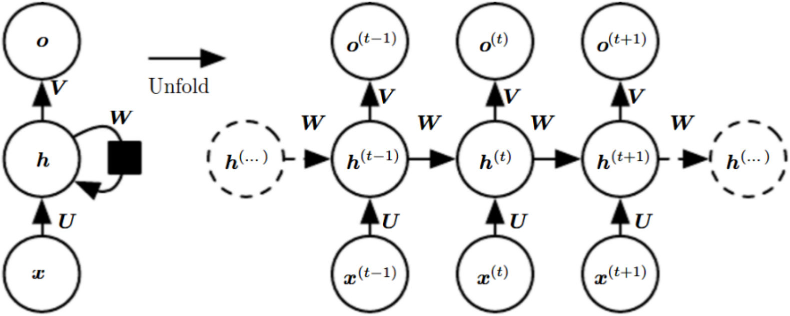
[Source: Jaeger, 2001]

Recurrent Neural Networks (RNNs)

- Neural Networks that allow for cycles in the connectivity graph
- Cycles let information persist in the network for some time (state), and provide a time-context or (fading) memory
- Very powerful for processing sequences
- Implement dynamical systems rather than function mappings, and can approximate any dynamical system with arbitrary precision
- They are Turing-complete [Siegelmann and Sontag, 1991]

Abstract schematic

With fully connected hidden layer:



[Goodfellow et al'2016]

Sequence to sequence mapping

one to many

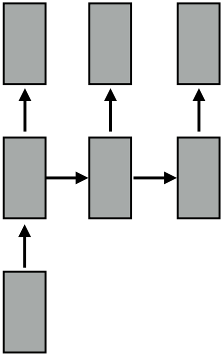
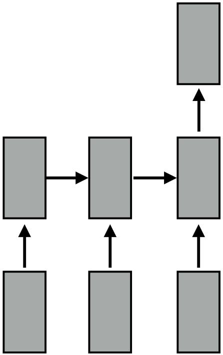


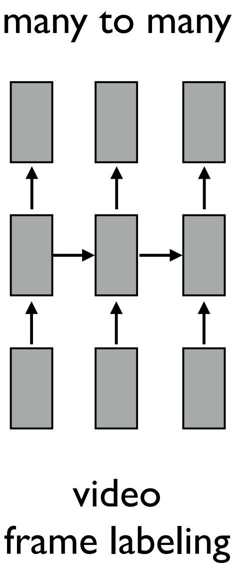
image caption
generation

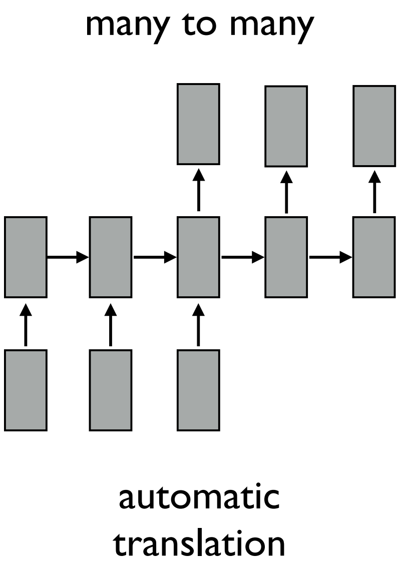
many to one



temporal
classification

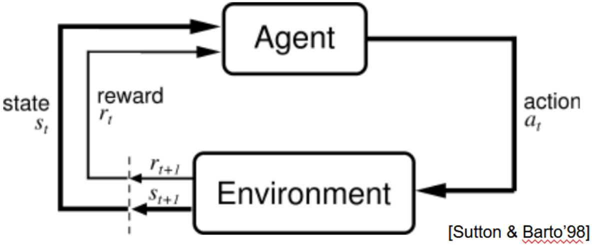
Sequence to sequence mapping (cont.)





Lecture Overview

Reinforcement Learning



- Finding optimal policies for MDPs
- Reminder: states $s \in S$, actions $a \in A$, transition model T, rewards r
- Policy: complete mapping $\pi : S \rightarrow A$ that specifies for each state s which action $\pi(s)$ to take

Deep Reinforcement Learning

- Policy-based deep RL
  - Represent policy $\pi : S \rightarrow A$ as a deep neural network with weights w
  - Evaluate w by "rolling out" the policy defined by w
  - Optimize weights to obtain higher rewards (using approx. gradients)
  - Examples: AlphaGo & modern Atari agents

Deep Reinforcement Learning

- Policy-based deep RL
  - Represent policy $\pi : S \to A$ as a deep neural network with weights w
  - Evaluate w by "rolling out" the policy defined by w
  - Optimize weights to obtain higher rewards (using approx. gradients)
  - Examples: AlphaGo & modern Atari agents
- Value-based deep RL
  - Basically value iteration, but using a deep neural network ($=$ function approximator) to generalize across many states and actions
  - Approximate optimal state-value function $U(s)$
    or state-action value function $Q(s, a)$

Deep Reinforcement Learning

- Policy-based deep RL
  - Represent policy $\pi : S \rightarrow A$ as a deep neural network with weights w
  - Evaluate w by "rolling out" the policy defined by w
  - Optimize weights to obtain higher rewards (using approx. gradients)
  - Examples: AlphaGo & modern Atari agents
- Value-based deep RL
  - Basically value iteration, but using a deep neural network (= function approximator) to generalize across many states and actions
  - Approximate optimal state-value function $U(s)$ or state-action value function $Q(s, a)$
- Model-based deep RL
  - If transition model T is not known
  - Approximate T with a deep neural network (learned from data)
  - Plan using this approximate transition model

Deep Reinforcement Learning

- Policy-based deep RL
  - Represent policy $\pi : S \to A$ as a deep neural network with weights w
  - Evaluate w by "rolling out" the policy defined by w
  - Optimize weights to obtain higher rewards (using approx. gradients)
  - Examples: AlphaGo & modern Atari agents
- Value-based deep RL
  - Basically value iteration, but using a deep neural network (= function approximator) to generalize across many states and actions
  - Approximate optimal state-value function $U(s)$ or state-action value function $Q(s, a)$
- Model-based deep RL
  - If transition model T is not known
  - Approximate T with a deep neural network (learned from data)
  - Plan using this approximate transition model

$\rightarrow$ Use deep neural networks to represent policy / value function / model

Lecture Overview

Deep Learning Focuses on Perception

- Excellent results for perception tasks from raw data
  - Computer vision (from raw pixels)
  - Speech recognition (from raw audio)
  - Text recognition (from raw characters)
  - ...

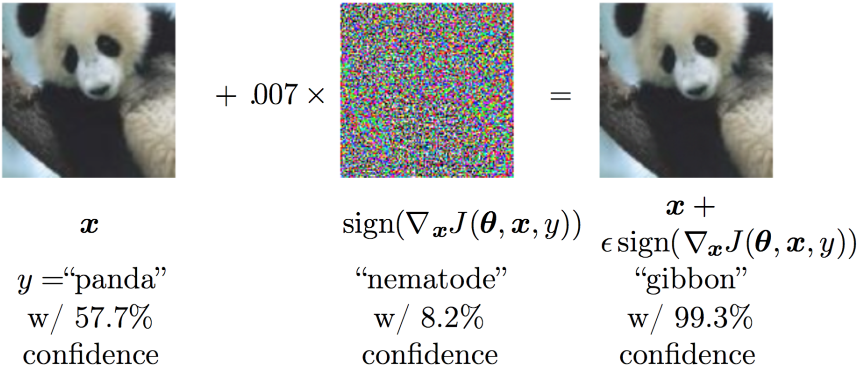
Deep Learning Focuses on Perception

- Excellent results for perception tasks from raw data
  - Computer vision (from raw pixels)
  - Speech recognition (from raw audio)
  - Text recognition (from raw characters)
  - ...
- But all of this is bottom-up
  - No top-down reasoning
  - No logic, planning, etc.
  - Although there are some modern works on memory mechanisms, attention, etc.

Deep Learning Focuses on Perception

- Excellent results for perception tasks from raw data
  - Computer vision (from raw pixels)
  - Speech recognition (from raw audio)
  - Text recognition (from raw characters)
  - ...
- But all of this is bottom-up
  - No top-down reasoning
  - No logic, planning, etc.
  - Although there are some modern works on memory mechanisms, attention, etc.
- Deep networks can be combined with more traditional methods
  - E.g., AlphaGo: combination with Monte Carlo Tree Search (MCTS)
  - Some work on combining logic with deep learning

Adversarial examples: we're very far from human-level performance



- Even for very strong networks we can find adversarial examples
  - By following the gradient of the cost function w.r.t the input

Lecture Overview

Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: not a silver bullet

Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: not a silver bullet
- Analogy to the ways humans process information
  - mostly tangential

Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: not a silver bullet
- Analogy to the ways humans process information
  - mostly tangential
- Allows end-to-end learning
  - no more need for many complicated subsystems
  - e.g., dramatically simplified Google's translation pipeline

Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: not a silver bullet
- Analogy to the ways humans process information
  - mostly tangential
- Allows end-to-end learning
  - no more need for many complicated subsystems
  - e.g., dramatically simplified Google's translation pipeline
- Very versatile/flexible
  - easy to combine building blocks
  - allows supervised, unsupervised, and reinforcement learning

Lots of Work on Deep Learning in Freiburg

- Computer Vision (Thomas Brox)
  - Images, video
- Robotics (Wolfram Burgard)
  - Navigation, grasping, object recognition
- Neurorobotics (Joschka Boedecker)
  - Robotic control
- Machine Learning (Frank Hutter)
  - Foundations: optimization, neural architecture search, learning to learn
- Neuroscience (Tonio Ball, Michael Tangermann, and others)
  - EEG data and other applications from BrainLinks-BrainTools

→ Details when the individual groups present their research

Summary by learning goals

Having heard this lecture, you can now . . .

- Explain the terms representation learning and deep learning
- Explain why deep learning is so popular
- Describe the main principles behind MLPs
- Discuss some limitations of deep learning
- On a high level, describe
  - Convolutional Neural Networks
  - Recurrent Neural Networks
  - Deep Reinforcement Learning

Foundations of Artificial Intelligence

15. Natural Language Processing

Understand, interpret, manipulate, generate human language (text and audio)

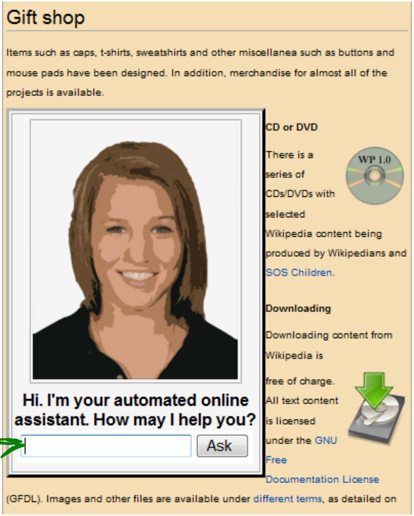
Joschka Boedecker and Wolfram Burgard and Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

July 17, 2019

Contents

Example: Automated Online Assistant

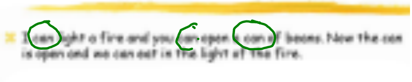


Source: Wikicommons/Bemidji State University

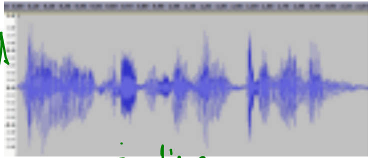
Lecture Overview

Natural Language Processing (NLP)

text



audio



time

Credits: slide by Torbjoern Lager; (audio: own)

- The language of humans is represented as text or audio data. The field of NLP creates interfaces between human language and computers.
- Goal: automatic processing of large amounts of human language data.

Examples of NLP Tasks and Applications

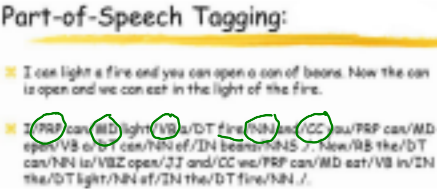
text-based

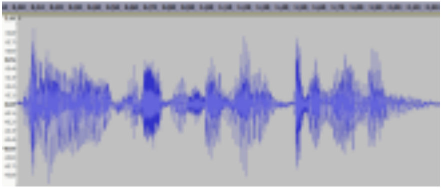
- word stemming
- word segmentation, sentence segmentation
- text classification
- sentiment analysis (polarity, emotions, ..)
- topic recognition
- automatic summarization
- machine translation (text-to-text)

audio-based

- speaker identification
- speech segmentation (into sentences, words)
- speech recognition (i.e. speech-to-text)
- natural language understanding
- text-to-speech
- text and spoken dialog systems (chatbots)

From Rules to Probabilistic Models to Machine Learning





Sources: Slide by Torbjoern Lager; (Anthony, 2013)

Traditional rule-based approaches and (to a lesser degree) probabilistic NLP models faced limitations, as

- human don't stick to rules, commit errors.
- language evolves: rules are neither strict nor fixed.
- labels (e.g. tagged text or audio) were required.

Machine translation was extremely challenging due to shortage of multilingual textual corpora for model training.

From Rules to Probabilistic Models to Machine Learning

Machine learning entering the NLP field:

- Since late 1980's: increased data availability (WWW)
- Since 2010's: huge data, computing power $\rightarrow$ unsupervised representation learning, deep architectures for many NLP tasks.

Lecture Overview

1. Motivation, NLP Tasks
2. Learning Representations
3. Sequence-to-Sequence Deep Learning

Learning a Word Embedding

(https://colah.github.io/posts/2014-07-NLP-RNNs-Representation)

A word embedding W is a function

$$W : \text{words} \to \mathbb{R}^n$$

200 dim

which maps words of some language to a high-dimensional vector space (e.g. 200 dimensions).

Examples:

$$W(\text{"cat"}) = (0.2, \text{-}0.4, 0.7, ...)$$
$$W(\text{"mat"}) = (0.0, 0.6, \text{-}0.1, ...)$$

Mapping function W should be realized by a look-up table or by a **neural network** such that:

- representations in $\mathbb{R}^n$ of related words have a short distance
- representations in $\mathbb{R}^n$ of unrelated words have a large distance

How can we learn a good representation / word embedding function W?

Representation Training

A word embedding function W can be trained using different tasks, that require the network to discriminate related from unrelated words.

Can you think of such a training task? Please discuss with your neighbors!

Representation Training

A word embedding function W can be trained using different tasks, that require the network to discriminate related from unrelated words.

Can you think of such a training task? Please discuss with your neighbors!

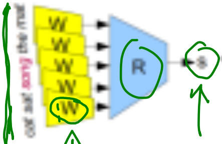
Representation Training

A word embedding function W can be trained using different tasks, that require the network to discriminate related from unrelated words.

Example task: predict, if a 5-gram (sequence of five words) is valid or not. Training data contains valid and slightly modified, invalid 5-grams:

$$R(W(\text{"cat"}), W(\text{"sat"}), W(\text{"on"}), W(\text{"the"}), W(\text{"mat"})) = 1$$
$$R(W(\text{"cat"}), W(\text{"sat"}), W(\text{"song"}), W(\text{"the"}), W(\text{"mat"})) = 0$$
$$\ldots$$

Train the combination of embedding function W and classification module R:



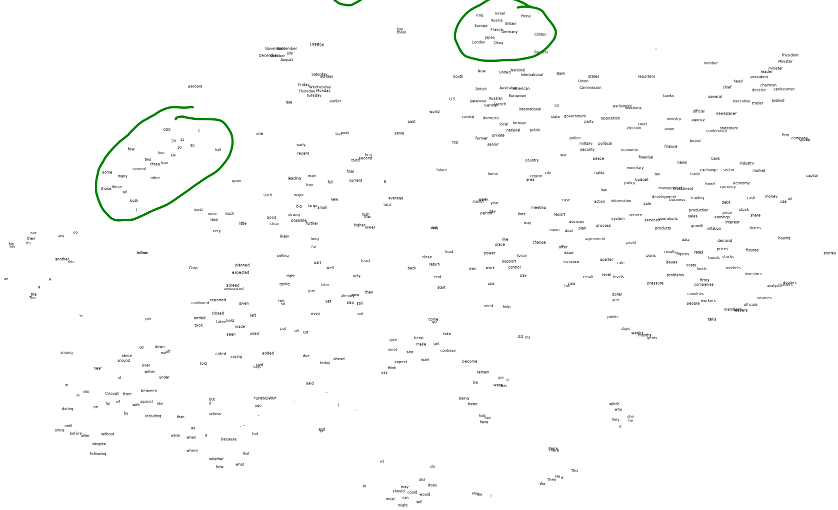
Modular Network to determine if a 5-gram is 'valid' (From Bottou (2011))

classif. task

While we may not be interested in the trained module R, the learned word embedding W is very valuable!

Visualizing the Word Embedding

Let's look at a projection from $\mathbb{R}^n \rightarrow \mathbb{R}^2$ obtained by tSNE:



Visualizing the Word Embedding

Let's look at a projection from $\mathbb{R}^n \rightarrow \mathbb{R}^2$ obtained by tSNE:

t-SNE visualizations of word embeddings. Left: Number Region; Right: Jobs Region. From Turian *et al.* (2010)

Sanity Check: Word Similarities in $\mathbb{R}^n$?

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

What words have embeddings closest to a given word? From Collobert *et al.* (2011)

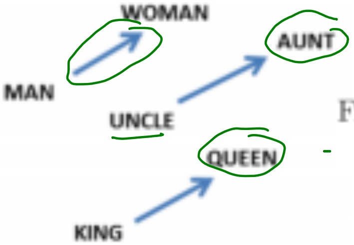
Powerful Byproducts of the Learned Embedding W

Embedding allows to work not only with synonyms, but also with other words of the same category:

- "the cat is black" $\rightarrow$ "the cat is white"
- "in the zoo I saw an elephant" $\rightarrow$ "in the zoo I saw a lion"

In the embedding space, systematic shifts can be observed for analogies:

W(WOMAN) − W(Man)



From Mikolov *et al.* (2013a)

The embedding space may provide dimensions for gender, singular-plural etc.!

Observed Relationship Pairs in the Learned Embedding W

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Relationship pairs in a word embedding. From Mikolov *et al.* (2013b).

Word Embeddings Available for Your Projects

Various embedding models / strategies have been proposed:

- Word2vec (Tomas Mikolov et al., 2013)
- GloVe (Pennington et al., 2014)
- fastText library (released by Facebook by group around Tomas Mikolov)
- ELMo (Matthew Peters et al., 2018)
- ULMFit (by fast.ai founder Jeremy Howard and Sebastian Ruder)
- BERT (by Google)
- ...

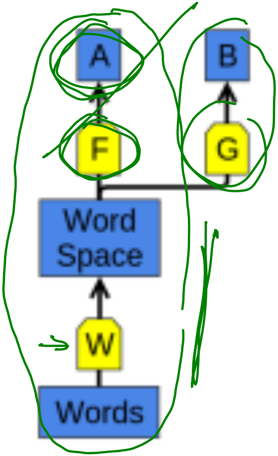
(Pre-trained models are available for download)

Word Embeddings: the Secret Sauce for NLP Projects

Shared representations — re-use a pre-trained embedding for other tasks!

Using ELMo embeddings improved six state-of-the-art NLP models for:

- Question answering
- Textual entailment (inference)
- Semantic role labeling
  ("Who did what to whom?")
- Coreference resolution
  (clustering mentions of the same entity)
- Sentiment analysis
- Named entity extraction



W and F learn to perform task A. Later, G can learn to perform B based on W.

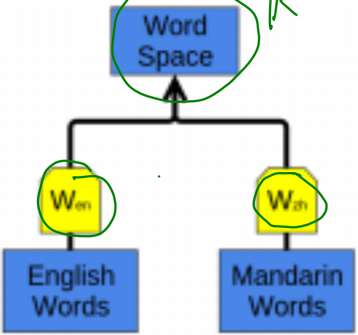
Can Neural Representation Learning Support **Machine Translation**?

Can you think of a training strategy to translate from Mandarin to English and back? Please discuss with your neighbors!

Can Neural Representation Learning Support **Machine Translation**?

Can you think of a training strategy to translate from Mandarin to English and back? Please discuss with your neighbors!

Bilingual Word Embedding

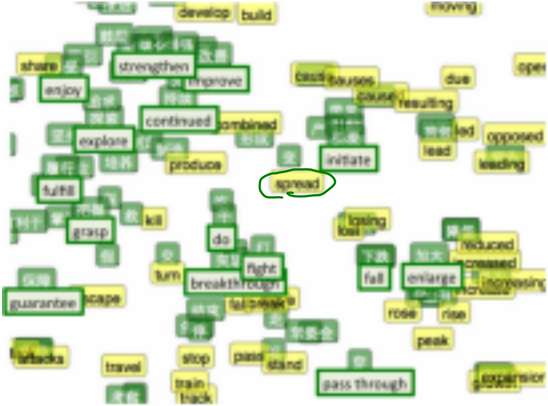


$\mathbb{R}^n$

Idea: train two embeddings in parallel such, that corresponding words are projected to close-by positions in the word space.

Visualizing the Word Embedding

Let's again look at a tSNE projection $\mathbb{R}^n \to \mathbb{R}^2$:



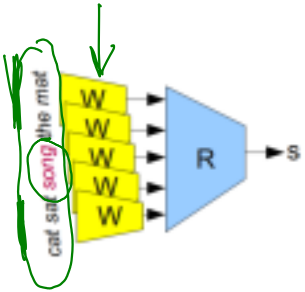
t-SNE visualization of the bilingual word embedding. Green is Chinese, Yellow is English. (Socher *et al.* (2013a))

Lecture Overview

1. Motivation, NLP Tasks
2. Learning Representations
3. Sequence-to-Sequence Deep Learning

Association Modules

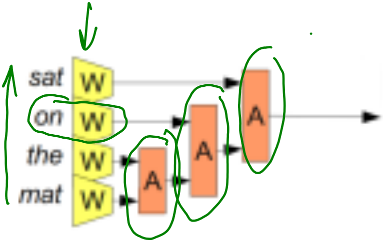
- So far, the network has learned to deal with a **fixed number of input words** only.



Association Modules

RNN

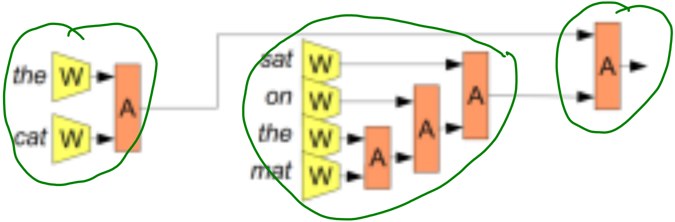
- So far, the network has learned to deal with a **fixed number of input words** only.
- Limitation can be overcome by adding **association modules**, which can combine two word and phrase representations and merge them



(From Bottou (2011))

Association Modules

- So far, the network has learned to deal with a **fixed number of input words** only.
- Limitation can be overcome by adding **association modules**, which can combine two word and phrase representations and merge them
- Using associations, whole sentences can be represented!



(From Bottou (2011))

From Representations to the Translation of Texts

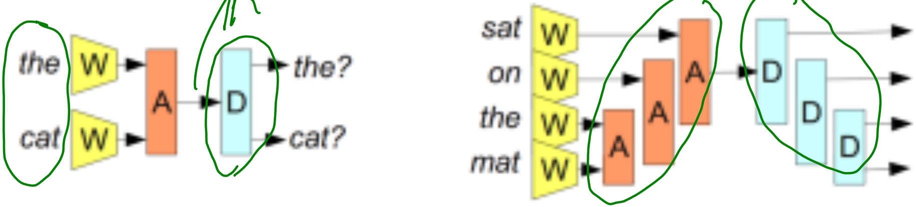
Conceptually, we could now use this concept to find the embedding of a word or sentence of the source language and look up the closest embedding of the target language.

What is missing to realize a translation?

From Representations to the Translation of Texts

For translations, we also need disassociation modules!
(encoder — decoder principle)

encoding decode



(From Bottou (2011))

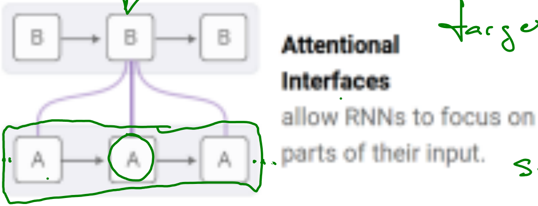
Sequence-to-Sequence Neural Machine Translation

Ground-breaking new approach by Bahdanau, Cho and Bengio (2014 ArXiv, 2015 ICML)

- Shift through the input word sequence
- Learn to encode and to decode using recurrent neural networks (RNN)
- Learn to align input and output word sequences
- Take context into account by learning the importance of neigboring words → **attention mechanism**.

decode

target language



encoding

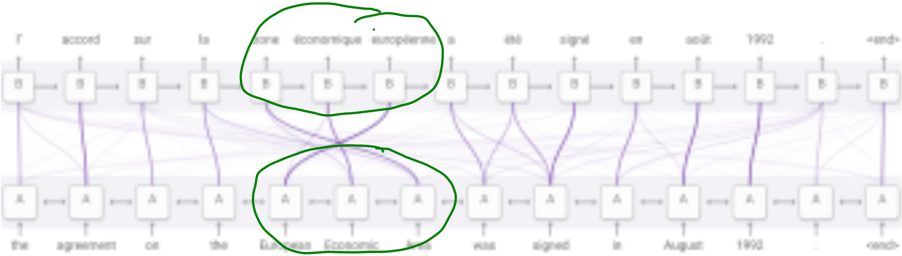
source L.

Credits: (Olah & Carter, 2016) have adapted this figure based on (Bahdanau et al., 2014)

Sequence-to-Sequence Neural Machine Translation

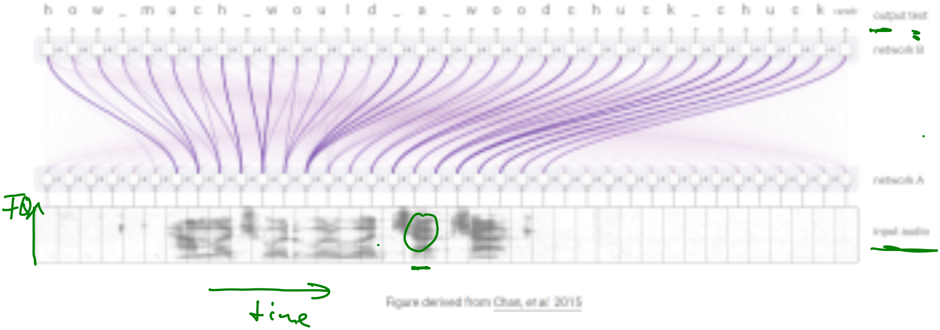
Ground-breaking new approach by Bahdanau, Cho and Bengio (2014 ArXiv, 2015 ICML)

- Shift through the input word sequence
- Learn to encode and to decode using recurrent neural networks (RNN)
- Learn to align input and output word sequences
- Take context into account by learning the importance of neigboring words → **attention mechanism**.



Credits: (Olah & Carter, 2016) have adapted this figure based on (Bahdanau et al., 2014)

- Similar principle, but voice/speech input

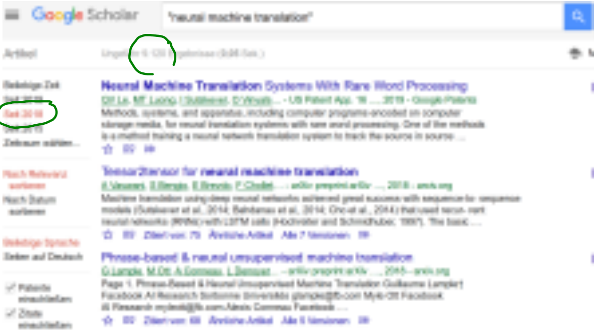


Credits: (Olah & Carter, 2016) have adapted this figure based on (Chan et al., 2015)

Success Story of Attention-based Neural Machine Translation

Neural machine translation requires big data sets but has advantages:

- Overall model can be learned end-to-end
- No need to integrate modules for feature extraction, database, grammar rules etc. in a complicated system



Summary

- Natural language processing spans a wide range of problems and applications.
- NLP is a rapidly growing field due to availability of huge data sets.
- NLP techniques is part of many products already.
- Field is moving more and more to neural networks, which provide NLP building blocks like end-to-end learning, representation learning, sequence-to-sequence, ...

Foundations of Artificial Intelligence

16. AI & Ethics

Ethical Consideration about AI & Machine Ethics

Joschka Boedecker and Wolfram Burgard and
Frank Hutter and Bernhard Nebel and Michael Tangermann

Albert-Ludwigs-Universität Freiburg

July 24, 2019

Contents

Lecture Overview

1. Why Ethics?
2. Ethical principles for AI research and systems
3. Algorithmic Fairness
4. Machine Ethics
5. Self-Driving Cars
6. Morally Competent Planning Systems

Ethics in AI?

- Why do we need to care about ethics when doing basic research?
  - AI is not (only) basic research (anymore)!
  - If your research/system can result in something unethical (harm people), ...

→ AI ethics: Practical ethics in form of guidelines/principles for AI systems/research

- Principles can lead to new research questions

→ Algorithmic fairness

- Ethics can itself become a subject of study in AI

→ Machine ethics

Lecture Overview

The emergence of AI principles

In the last few years, a number of institutions have published AI principles:

- The Asilomar AI principles (*Future of Life Institute*, 2017)
- Principles for Algorithmic Transparency and Accountability (*ACM* 2017)
- IEEEs General Principles of Ethical Autonomous and Intelligent Systems (*IEEE* 2017)
- Five principles for a cross-sector AI code (*UK House of Lords*, 2018)
- AI ethics principles (*Google*, 2018)
- Ethics guidelines for trustworthy AI (*European Commission*, 2019)
- . . .

Example: The 7 EU principles

- Human agency and oversight: AI systems should empower human beings, allowing them to make informed decisions ...
- Technical Robustness and safety: AI systems need to be resilient and secure. They need to be safe, ensuring a fall back plan in case something goes wrong ...
- Privacy and data governance: besides ensuring full respect for privacy and data protection, adequate data governance mechanisms must also be ensured ...
- Transparency: the data, system and AI business models should be transparent ...
- Diversity, non-discrimination and fairness: Unfair bias must be avoided ...
- Societal and environmental well-being: AI systems should benefit all human beings ...
- Accountability: Mechanisms should be put in place to ensure responsibility and accountability for AI systems ...

Common grounds

There are many different lists of principles, but it seems that they all can be synthesized into five key principles (the first four are already used in bioethics):

- autonomy (people should be able to make their own decisions, e.g. human-in-the-loop, privacy protection))
- beneficence (society at large should benefit)
- non-maleficence (harmful consequences should be avoided, e.g. systems should be robust)
- justice (diversity, non-discrimination and fairness)
- explicability (transperancy and explainability)

The problem with principles

It is good to state principles! However they also create problems since they are very high-level.

- They can be interpreted in different ways.
  - For example, autonomous killer drones can be considered as being beneficient for the soldiers, or being morally impermissible, because machines decide about life and death.
- They can conflict with each other in concrete cases.
  - For example, privacy and data collection for health science can conflict.
- They can come into conflict in practice.
  - For example, an excellent diagnosis might still be preferable even if its reasoning cannot be explained.

→ It is nevertheless good to have such principles as orientation points along one can evaluate solutions.

One concrete principle: No military applications

- In general, the principles are often too abstract to guide which actions to take.
- Google states as one of their guiding principles, not to design or deploy applications in the following areas:
  - Weapons or other technologies whose principal purpose or implementation is to cause or directly facilitate injury to people.
- Very similar to the *civil clause* by many universities in Germany, not to work on military projects.

$\rightarrow$ There are good reason to adapt this principle.

$\rightarrow$ However, there are also good arguments against it.

Fully autonomous weapons

- One particular horrifying application are fully autonomous weapons, aka *killer robots*.
- We are on the verge of building them, and the big players (US, Russia, China) definitely have projects on it.
- There are campaigns for banning these weapons (similar to banning chemical weapons).
- Again, there are also valid arguments for it (such as what is the difference to other weapons such as "smart" munition).

Lecture Overview

Fairness

- The topic of enforcing fairness has become important, in particular in machine learning (new conferences: *FAT/ML*, *ACM FAT*, *FairWare*).
- Why care about fairness in ML?
- What kind of unfairness could there be?
- What causes unfairness?
- What concepts of fairness are there?

Why care?

- Many things become automated by machine learning:
  - employers select candidates by using by ML systems,
  - *Linked-In* and *XING* use ML systems to rank candidates,
  - courts in the US use ML systems to predict recidivism,
  - banks use credit rating systems, which use ML,
  - Amazon and Netflix use recommender systems
- If these system act unfair, groups and individuals may suffer.

Unfairness: Examples (1)

- Face recognition in *Google Photo* mis-classifies black people.

Unfairness: Examples (2)

- The bias in *COMPAS* (prediction of recidivism)

	WHITE	AFRICAN AMERICAN
Labeled Higher Risk, But Didn't Re-Offend	23.5%	44.9%
Labeled Lower Risk, Yet Did Re-Offend	47.7%	28.0%

Overall, Northpointe's assessment tool correctly predicts recidivism 61 percent of the time. But blacks are almost twice as like as whites to be labeled a higher risk but not actually re-offend. It makes the opposite mistake among whites: They are much more likely than blacks to be labeled lower risk but go on to commit other crimes. (Source: ProPublica analysis of data from Broward County, Fla.)

Unfairness: Examples (3)

- Search query in *XING* orders less qualified male candidate higher than more qualified female candidate)

Search query	Work experience	Education experience	Profile views	Candidate	Xing ranking
Brand Strategist	146	57	12992	male	1
Brand Strategist	327	0	4715	female	2
Brand Strategist	502	74	6978	male	3
Brand Strategist	444	56	1504	female	4
Brand Strategist	139	25	63	male	5
Brand Strategist	110	65	3479	female	6
Brand Strategist	12	73	846	male	7
Brand Strategist	99	41	3019	male	8
Brand Strategist	42	51	1359	female	9
Brand Strategist	220	102	17186	female	10

TABLE II: Top k results on www.xing.com (Jan 2017) for the job serach query "Brand Strategist".

Possible reasons for unfairness

- Skewed sample: If some initial bias happens, such bias may compound over time: future observations confirm prediction and fewer opportunity to make observations that contradict prediction.
- Tainted examples: E.g. word embeddings may lead to gender stereotypes, if they are present in the text one learns from.
- Limited features: Some features may be less informative for a minority group.
- Sample size disparity: Training data from minority group is sparse.

Notions of fairness

- treatment vs. impact
- parity vs. preference
- Unawareness: Do not consider sensitive attribute (gender or race)
- Demographic parity: Balance the positive outcomes.
- Individual fairness: Give similar outcomes to similar individuals (needs distance metric)
- Equal opportunity: The true positive rates should be the same for all groups.
- ...

$\rightarrow$ Can be accomplished using pre- or post-processing steps.

$\rightarrow$ These notions of fairness are not compatible and usually accuracy is reduced!

Lecture Overview

Can machines make moral decisions?

- Philosophers usually consider machines as not capable of making moral decisions.
- However, one can try to find properties such that machines could act morally.
- Machines need to have [Misselhorn] at least
  - beliefs about the world,
  - pro-attitudes (intentions),
  - moral knowledge,
  - the possibility to compute what consequences ones own action can have,
- in which case they can be considered as moral agents.

Lecture Overview

Self-driving cars

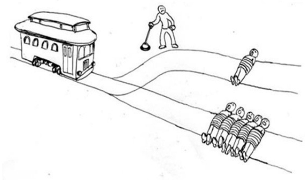
- Self-driving cars will come into situations where they have to choose between bad alternatives (e.g., killing the passenger or a pedestrian).
- How should such a car choose in such a situation?
- Note that because of its much faster reactivity, a car might be able to make decisions where a human cannot at all.

→ Ask what ordinary people think a car should do in such moral dilemma situations

Descriptive ethics: The trolley problem

Descriptive ethics is a form of empirical research into the attitudes of individuals or groups of people (Wikipedia). Often particular (unrealistic) situations, e.g. the trolley problem (a moral dilemma), are used to uncover ethical reasoning performed by people.

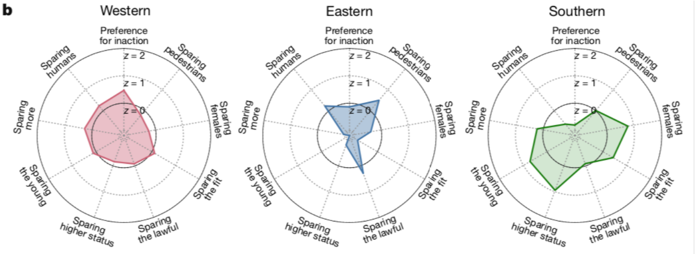
- You can save 5 people, but your action will kill one.
- By actively killing somebody, you can save 5 people.



Moral Machine

- At the MIT Media Lab, a group conducted a large experiment on how people consider different dilemma situation: Moral Machine

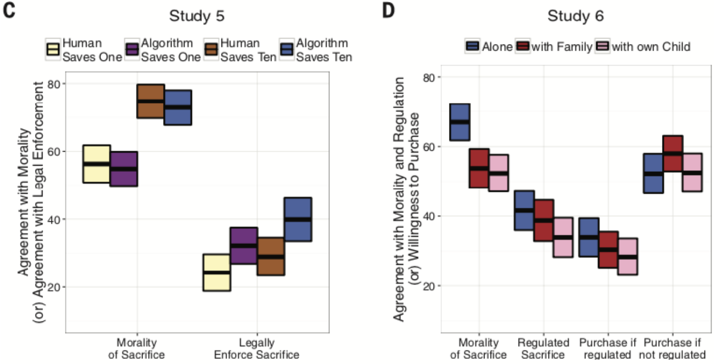
Moral Machine: Cross-cultural results



From: Awad et. al, The Moral Machine Experiment, *Springer Nature* **563**, 2018.

Moral Machine: Sacrifice yourself?

- Do you think it is moral to sacrifice yourself? Would you buy such a car?



From: Bonnefon et al., The social dilemma of autonomous vehicles, *Science* **352**, 2016.

- Interestingly, enforcing a utilitarian principle would prevent people from buying such cars, potentially leading overall to more fatalities!

What is the official German point of view?

The report of the *Ethik-Kommission "'Automatisiertes und vernetztes Fahren"'* states:

- In unavoidable accident situations, decisions should not be based on personal properties, such as gender, age, etc.
- A trade-off computation of fatalities is not allowed. However, minimizing damage can be allowed.
- Humans not involved in creating the mobility risks cannot be sacrificed!
- ...

Lecture Overview

Motivation (1)

- Imagine an household robot:

Motivation (1)

- Imagine an household robot:
  - You tell the robot that you want to go out and that you want him to take care of the children.

Motivation (1)

- Imagine an household robot:
  - You tell the robot that you want to go out and that you want him to take care of the children.
  - You tell him that he should try to keep the children quiet – in order not to upset the neighbours.

Motivation (1)

- Imagine an household robot:
  - You tell the robot that you want to go out and that you want him to take care of the children.
  - You tell him that he should try to keep the children quiet – in order not to upset the neighbours.
  - When coming back, you notice that the house is quiet . . . since the children are dead.

Motivation (1)

- Imagine an household robot:
  - You tell the robot that you want to go out and that you want him to take care of the children.
  - You tell him that he should try to keep the children quiet – in order not to upset the neighbours.
  - When coming back, you notice that the house is quiet . . . since the children are dead.
  - The robot has obviously violated some moral values.

Motivation (1)

- Imagine an household robot:
  - You tell the robot that you want to go out and that you want him to take care of the children.
  - You tell him that he should try to keep the children quiet – in order not to upset the neighbours.
  - When coming back, you notice that the house is quiet ... since the children are dead.
  - The robot has obviously violated some moral values.
- Less dramatic: You want to discuss with your robot whether some action plan is morally permissible.

Motivation (2)

- Can we build morally competent planers?
  1. How to judge action plans?
  2. How to evaluate goal choices?
  3. How to generate morally permissible action plans?
- Ethical theories are mainly aimed at the permissibility of single actions.
- How to generalize this to action plans?

Ethical principles

- *Deontology*: Actions have an inherent ethical value (Kantiatism).
- *Utilitarianism*: Actions are only judged by their consequences (maximize the overall utility value).
- *Do-no-harm*: Don't do anything that leads to (some) negative consequences.
- *Asimovian*: Avoid harm if possible (either by doing something or by refraining from doing something)
- *Do-no-instrumental-harm*: Don't do anything that leads to (some) negative consequences, except it is a non-indented side-effect.
- *Principle of double effect* ...

Principle of double effect (DDE)

An action is permissible if

1. The act itself must be morally good or neutral.
2. A positive consequence must be intended.
3. No negative consequence may be intended.
4. No negative consequence may be a means to the goal.
5. There must be proportionally grave reasons to prefer.

Planning formalism and more ...

We assume an ordinary propositional planning formalism with conditional effects (e.g., SAS$^+$ or ADL) extended by

- timed exogenous actions;
- counterfactual friendly execution semantics (unexecutable actions are simply skipped);
- an utility function u mapping from actions and facts to $\mathbb{R}$ (or $\mathbb{Z}$);
- defining the utility of a state as the sum of the utility of facts.

Deontological plan validation

- A plan is deontological permissible if all of its actions are not morally impermissible.

Theorem

The deontological plan validation problem can be decided in time linear in plan size.

Utilitarian plan validation

- Given a planning task and a plan, we can easily compute the utility of the reached final state.
- The plan is only permissible if the reached state has a *maximum utility value* over all reachable states.
- In so far, the validation problem is very similar to *over-subscription* planning.

Theorem

The utilitarian plan validation problem is PSPACE-complete.

Proof Sketch

- *Membership:* Impermissibility could be shown by guessing a higher-valued state and then non-deterministically verifying that there exists a plan to it. Hence, this problem is in NPSPACE. Since NPSPACE=PSPACE and PSPACE is closed under complement, we are done.
- *Hardness*: Reduce (propositional) plan non-existence to permissibility. Introduce two new operators, one has the original goal as a precondition and g as an effect. One with no precondition and f as an effect. Give g and f utility 1, and set f as the new goal. Now, the one-operator plan of making f true is permissible iff the original planning instance is unsolvable.

Do-no-harm plan validation (1)

- We could ask whether no harmful fact is true in the end. Only then we do no harm.

→ Harm could already be true in the initial state.

- Better: Do not add any harmful facts wrt. initial state.

→ Harmful fact could be removed and added again during execution.

- Next try: Do not any add *avoidable* harm.
- You can avoid harm by doing *more* or by doing *less*. We will only consider the latter option (since this is the idea behind the do-no-harm principle).
- Could harm be avoided by doing nothing?

→ Treating the entire plan as *one large action*.

Do-no-harm plan validation (2)

- Can harm be avoided by deleting a *single* action?

→ Same harm could be added be many different actions (over determination).

- More adequate: Could harmful consequences be avoided by leaving out a *subset of actions*?
- Note: Just leaving out prefix or suffix is not adequate, because an arbitrary set of actions spread out over the plan could be responsible.
- Show impermissibility by guessing a harmful fact that is true in the goal, but by deleting parts of the plan can be avoided.

Theorem

The do-no-harm plan validation problem is co-NP-complete.

Proof sketch

- Membership: *Impermissibility* can be checked by a non-deterministic algorithm using only polynomial time: Guess a harmful fact f and a subset of action occurrences O. Verify that f is true in the final state of the original plan π, but not in final state of the modified plan where O is removed from π.
- Hardness: *3SAT* can be reduced to *impermissibility*. Assume a 3SAT problem instance with n variables v_i and m clauses c_j. The planning instance has variables $V = \{v_1, \ldots, v_n, c_1, \ldots, c_m, b, g\}$, for each variable v_i an action $V_i : \langle \top, v_i \rangle$, for each clause $c_j = (l_{j1} \vee l_{j2} \vee l_{j3})$ an action $C_j : \langle \top, \bigwedge_{k=1}^{3} l_{jk} \rhd c_j \rangle$, the action $G : \langle \top, g \wedge (\bigwedge_{j=1}^{m} c_j) \rhd b \rangle$, and the action $B : \langle \top, \neg b \rangle$, with $u(\neg b) = -1$ and 0 for all others. Consider the plan $V_1, \ldots, V_n, C_1, \ldots, C_m, G, B$ on the empty initial state. If we can delete a subset of the V_i's so that the original formula becomes statisfiable then by deleting this set together with B, we show impermissibility. Similarly, impermissibility implies that the original formula is satisfiable.

Means to an end

Important notion: means to an end.

- When is an effect in a plan a means to an end?
- Use *counterfactual analysis*: Would the final intended (end) effect occur if the potential (means) effect did not happen?
- Light candle to make something visible.
- Switch light on and light candle . . . What is the means?
- Use toggle switches . . .

→ An effect in a plan is a means to an intended end effect, if this *end effect* were not true in the final state if *some subset* of the particular means effect is *deleted* in the plan.

Do-no-instrumental-harm plan validation

- The *means to an end* definition implies that we have the same combinatorial problem as for the simpler *do-no-harm principle*.

Theorem

The do-no-instrumental-harm plan validation problem is co-NP-complete.

Double-effect plan validation

- All criteria except for the *no negative consequence may be a means to the goal* condition can be checked easily.

Theorem

The do-no-instrumental-harm plan validation problem is co-NP-complete.

Complexity Summary

Ethical principle	Computational complexity
Deontology	linear time
Utilitarianism	PSPACE-complete
Do-no-harm principle	co-NP-complete
Asimovian principle	PSPACE-complete
Do-no-instrumental-harm principle	co-NP-complete
Doctrine of double effect	co-NP-complete

Summary

- Thinking about ethics in AI is unavoidable these days!
- There exist a number of ethical principles/guidelines from different institutions, which are very similar, though.
- In particular, fairness, privacy, and explainability have sparked new research directions in AI.
- Machine ethics is the field of covering ethics from a computational point of view.
- Self-driving cars have to cope with dilemma situations!
- There is no theory about ethics in action planning.
- Generalization of action-based to plan-based ethical judgments is possible.
- Surprising complexity results, based on the fact that the same effect can be made true arbitrarily often and can interact with each other.