SS24 Introduction to Artificial Intelligence

What is Artificial Intelligence?

Definitions and Goals of AI:
Systems that think/act like humans or think/act rationally

- **Strong AI**: Thinking/acting humanly
- **Weak AI**: Thinking/acting rationally

History of AI:

- Key milestones: Turing Test, early programs (e.g., chess, theorem proving), expert systems, machine learning, neural networks
- 2010s: Rise of Deep Learning, large-scale data processing (IBM Watson, AlphaGo, AlphaFold)

Current State of AI:

- Progress in Deep Learning, language/image processing, generative AI.
- Applications: Chatbots, virtual assistants, autonomous vehicles, medical diagnostics
- Generative AI (e.g., ChatGPT): Creates text, images, code; faces issues like hallucination and inaccuracies

Challenges and Limitations:

- **Lacks general human understanding, common sense**
- Struggles with scalability, fairness, and explainability
- Narrow in scope

Intelligent Agents

Agents and Environments:

- Agents: Entities that perceive (via **sensors**) and act (via **actuators**) on their environment
- **Agent Function**: Maps perception sequences to actions.
- **Agent Program**: Runs on physical architecture to produce actions

Rationality:

- Rational agents **maximize performance based on percept history, prior knowledge, and possible actions**
- **Rational ≠ omniscient, clairvoyant (hellseherisch) or guaranteed success**
- Key aspects;
- **Exploration**: Gather new information
- **Learning**: Adapt from experience
- **Autonomy**: Update initial knowledge

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

Agent Characteristics (PEAS):

- **Performance Measure**: Defines success criteria
- **Environment**: Conditions and factors affecting the agent
- **Actuators**: Means of action (e.g., steering, brakes)
- **Sensors**: Means of perception (e.g., GPS, camera)

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

Environment Types:

- **Fully/Partially Observable**: All or limited information
- **Deterministic/Stochastic**: Predictable vs. probabilistic outcomes
- **Episodic/Sequential**: Independent vs. dependent actions
- **Static/Dynamic**: Unchanging vs. changing environments
- **Discrete/Continuous**: Limited states vs. continuous range
- **Known/Unknown**: Familiar vs. unfamiliar rules
- **Single/Multi-Agent**: One agent vs. cooperative/ competitive agents

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

Agent Types:
Simple Reflex Agents: Rule-based; act on current percepts, no memory, no sequences of percepts

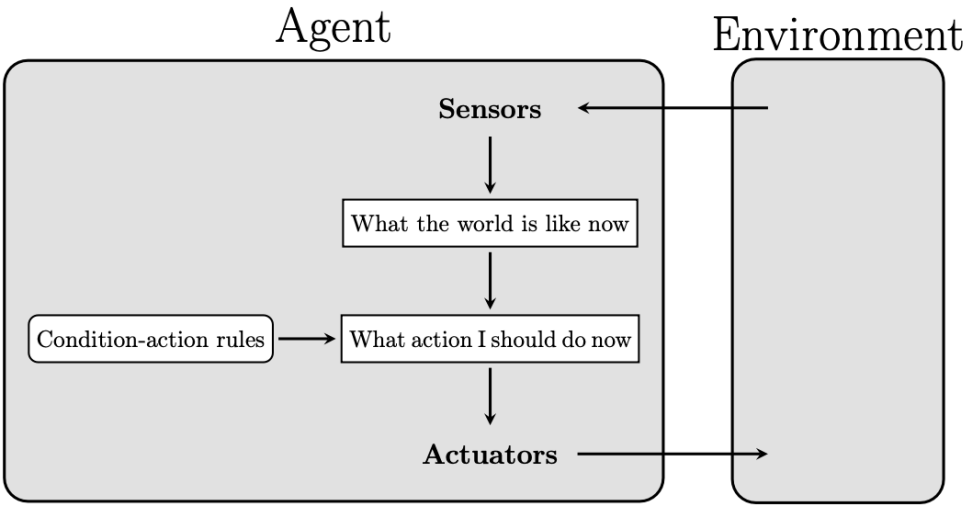


Figure 1: Simple Reflex Agents

Model-based Reflex Agents: Add on: Internal state and model of the world's evolution memory, maintain / update world state• reason about unobservable parts, deal with uncertainty, implicit goals

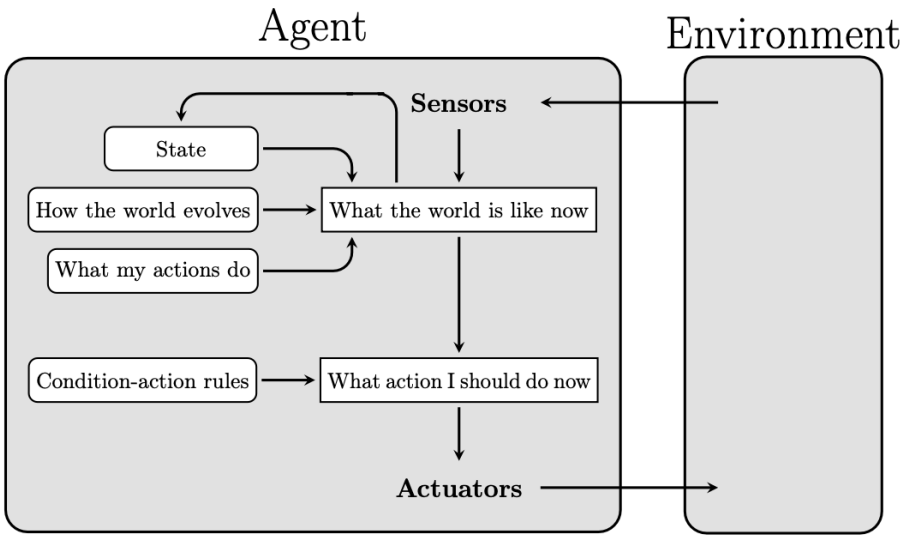


Figure 2: Model-based Reflex Agents

Goal-based Agents: Add on: Use explicit goals
allow for planning and flexibility, model the world, goals, and actions with their effects explicitly, more flexible & better maintainable

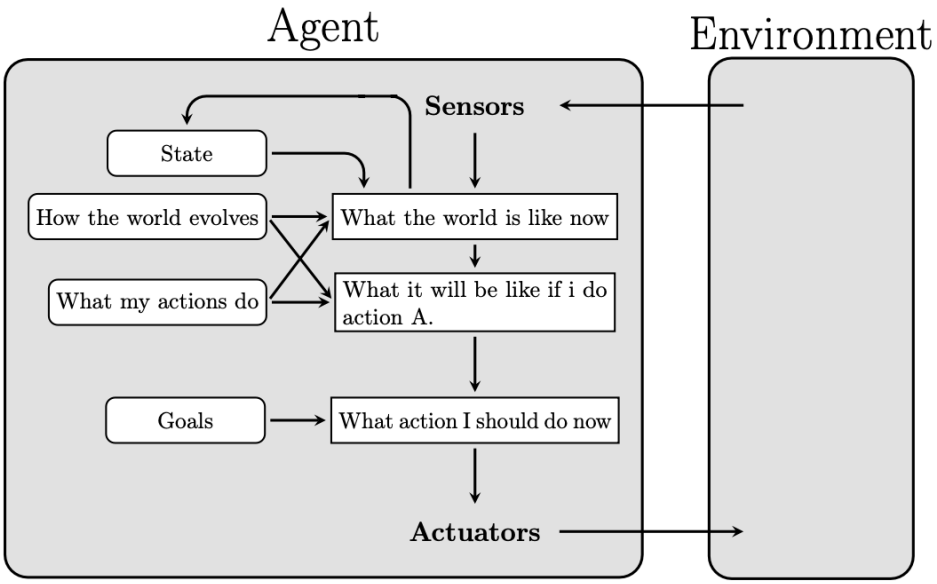


Figure 3: Goal-Based Agents

Utility-based Agents: Add on: take happiness into account
assess goals with a utility function, resolve conflicting goals, use expected utility for decision

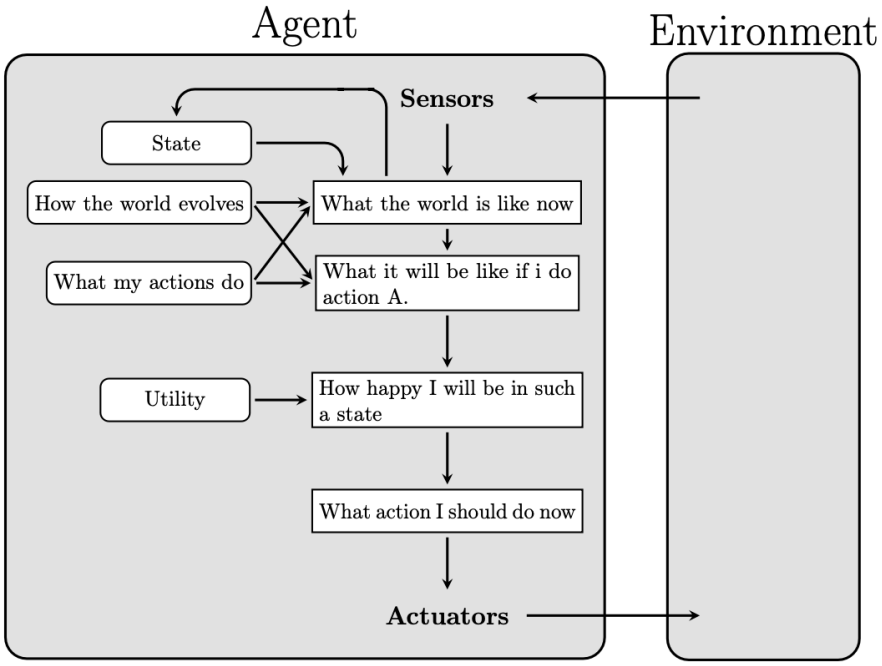


Figure 4: Utility-Based Agents

Problem Solving and Search (I)

Problem Formulation:

- Define a search problem with:
  1. **Initial State**: Starting point (e.g., in Arad)
  2. **Successor Function**: set of action-state pairs
  3. **Goal Test**: Determines if goal is reached (explicit or implicit)
  4. **Path Cost**: Total cost of actions to reach a goal

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

Basic Search Algorithms

- Tree-like Search:
- offline, simulated exploration of state space by generation successors of already-explored states. (expanding states)
- maintain a list of states available for expansion (frontier, open list).
- A **node** is a data structure constituting part of a search tree. It includes parent, children, depth and the path cost g(x)
- A **state** is a representation of a physical configuration. States do not have parents, children, depth or path cost!
- Graph Search:
- Use an explored set to avoid revisiting nodes.
- Efficiently manages repeated states.

Uninformed Search Strategies

Breadth-First Search (BFS):

- Expands shallowest node first.

Uniform-Cost Search (UCS):

- Expands lowest-cost node.

Depth-First Search (DFS):

- Expands deepest node first.

Depth-Limited Search (DLS):

- DFS with depth limit \ell ; avoids infinite loops.

Iterative Deepening Search (IDS): (best)

- Repeated DLS with increasing depth.

Bidirectional Search (BDS):

- Searches from initial and goal states simultaneously.

2.7.3 Uninformed Search Algorithms Summary

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

Criterion	BFS	UCS	DFS	DLS	IDS
Completeness	Yes$^{\alpha}$	Yes$^{\alpha,\beta}$	No	No	Yes$^{\alpha}$
Time	$O(b^d)$	$O(b^{1+\lfloor \frac{C^*}{\epsilon} \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor \frac{C^*}{\epsilon} \rfloor})$	$O(b \cdot m)$	$O(b \cdot l)$	$O(b \cdot d)$
Optimality	Yes$^{\gamma}$	Yes	No	No	Yes$^{\gamma}$

- $^{\alpha}$ if b is finite
- $^{\beta}$ if step cost $\leq \epsilon$ for positive ϵ
- $^{\gamma}$ if step costs are all identical

- **Completeness**: Can it find a solution?
- **Time Complexity**: Number of nodes generated
- **Space Complexity**: Maximum nodes in memory
- **Optimality**: Finds least-cost solution if one exists

Problem Solving and Search (II)

Heuristic Search:

- Uses problem-specific knowledge to evaluate desirability of states
- **Heuristic function (h)**: Estimates minimal cost from state n to goal (e.g., straight-line distance)

Greedy Search

- Evaluation function: f(n) = h(n)
- Expands node that appears closest to the goal based on heuristic
- Does not take already spent costs into account. That means decisions are based on local information

A*-Search

- Combines cost to reach node g(n) and estimated cost to goal h(n): f(n) = g(n) + h(n)
  - g(n): path costs from start to n, i.e., costs so far up to n
  - h(n): estimated cost to goal from n, like in greedy search
  - f(n): estimated total cost of path trough n to goal

  —> solves problems of greedy search
- heuristic must be **admissible**: Must not overestimate actual cost (optimistic)
- Ensures A* finds an optimal path
- **Consistency**: Heuristic is consistent if h(n) ≤c(n,a,n′) + h(n′); If h is consistent, then f is non-decreasing along every path
- Guarantees optimality for graph search

Algorithm	Completeness	Time Complexity	Space Complexity	Optimality	Notes
Greedy Search	***No***, can stuck in loops. ***Yes***, with loop checks.	$O(b^m)$, i.e., exponential in m	$O(b^m)$, i.e., exponential in m, because it keeps every node in memory	***No***	
A^*-Search	***Yes***, unless there are infinitely many nodes n with $f(n) \leq f(G)$	Exponential in $\epsilon \times d$: • $\epsilon = \frac{h(n_0)-h^*(n_0)}{h^*(n_0)}$ ... relative error $h^*(n_0) = C^*$ • d = solution depth path length	Exponential (keeps all nodes in memory)	***Yes***	A^* expands: • **all** nodes with $f(n) < C^*$ • **some** nodes with $f(n) = C^*$ • **no** nodes with $f(n) > C^*$ • fewest nodes safely possible if h is *consistent*

Memory-Bounded Search

- Limits memory usage for A*.
- Iterative Deepening A* (IDA*):
- Repeatedly explores paths within increasing cost limits.
- Retains only necessary nodes.
- Recursive Best-First Search (RBFS):
- Expands paths while retaining only the best alternative in memory.

Admissible (zulässige) Heuristics:

- ***Admissibility*:** $h(n) \leq h^*(n)$ - an admissible heuristic never overestimates the actual costs
- ***Consistency:*** Consistency ensures that the estimated cost $h(n)$ is always less than or equal to the cost of moving to a neighboring node plus the estimate from there: $h(n) \leq c(n,a,n') + h(n')$ holds, where $c(n,a,n')$ are the path costs for a
  If a heuristic is consistent, it is automatically admissible
- ***Dominance:*** For admissible heuristics h1 and h2, h2 dominates h1, if $h2(n) \geq h1(n)$ for every node n —> h2 provides more informative estimates, as it is closer to the true cost while still remaining admissible

- Relaxed Problems: The solution cost of a relaxed problem is a guaranteed admissible heuristic because it will never overestimate the actual cost.

Problem Solving and Search (II)

Local Search:

- Focuses on finding a solution by improving a single current state rather than exploring entire paths from start to goal (iterative improvement)
- State Space: set of "complete" configurations (all possible tours in TSP)
- Objective: Improve the current state by iteratively making it better according to some evaluation function
- Requires constant space (only tracks the current state)
- Works well even in environments that are partially known or dynamic

Hill-climbing (or gradient ascent/descent):

- Simple and Greedy
- Chooses the neighbor that improves the evaluation function the most
- Challenges: Local Maxima, Ridges (path requires downhill first), Plateaus
- Solutions: Sideways moves, random restart, alternative neighbor selection (random

Simulated Annealing:

- Aims to escape local maxima by occasionally allowing "bad" moves
- Over time gradually decrease their size and frequency
- Boltzmann Distribution: Determines the probability of being in a certain state based on its energy and temperature

Local Beam Search:

- Keep k states instead of one - choose top k of all their successors
- Select the k best successors based on their evaluation function
- Repeat until a solution is found or no improvement is possible
- The k states in Local Beam Search share information —> can influence others
- Challenges: Local Maxima, diversity
- Solutions: Random selection among good states, stochastic variant

Genetic Algorithms:

Genetic algorithm (GA) = stochastic local beam search + successors from pairs of states

- Choose individuals for reproduction based on their fitness
- Advantages: Exploration of Large Search Spaces, Diversity Maintenance
- Challenges: Representation Matters, Parameter Tuning, Convergence

Continuous State Spaces:

- problems where the variables can take any value within a range (no discrete steps)
- Constraint Optimization (Mathematical Programming): mathematical techniques like convex optimization or linear programming
- Discretization: dividing continuous space into a grid with fixed spacing
- Gradient-Based Methods

Nondeterministic Actions:

- Search with na requires robust strategies to handle uncertainty
- Tracking belief states, Using AND-OR trees for modeling, Considering worst-case scenarios to ensure goal achievement

Online Search:

- used in environments where agent has limited or no prior knowledge about the state space, environment may be dynamic or partially observable
- interleaving of acting and thinking; search as you go..
- Online Search Methods: DFS with Backtracking, Random Walks, Memory-Based Methods

Learning from Examples (I)

Types of Representations of the world:

- Atomic: States treated as indivisible units (like labels)
- Factored: States represented by attributes with values (e.g., temperature = 20°C)
- Structured: Models dependencies between attributes (e.g., Bayesian networks)

Learning Modes:

- **Supervised Learning**: agent is provided with labeled data (e.g., input-output pairs)
- **Unsupervised Learning**: No explicit labels; agent must identify patterns or group similar data
- **Reinforcement Learning**: Feedback comes as rewards or punishments for actions
- **Semi-Supervised Learning**: A mix of labeled and unlabeled examples

Inductive Learning:

- Simplest form: learn a function from examples (also known as science)
- Given a (finite) training set $(x_1,y_1),\ldots,(x_N,y_N)$ of examples, find a function h that approximates f ($h \approx f$)
- Challenges: Overfitting, Underfitting, Assumptions

Decision Tree Learning:

- Decision trees can express any function of the input attributes
- Advantages: Understandable, flexible
- Perform well when trained on enough data to capture patterns but not overfit
- Learning: (recursively) choose “most significant” attribute as root of (sub)tree
- Split data into subsets that are as “pure” as possible (all positive or all negative)
- Information Gain: Measures how much an attribute reduces uncertainty

Alternative Hypotheses:

- Conjunctive Hypotheses: Hungry ^ -Rain (hungry and no rain)
- Decision Lists: if condition 1, then output A; else if condition 2, then output B

Model Selection:

- Choosing the best model from to balance complexity and performance
- Overfitting remedies: Decision Tree Pruning, Cross-Validation
- Cross-validation ensures reliable evaluation of model performance
- Simpler models are often preferred when validation errors are comparable, following Ockham’s Razor
- Balancing model complexity and performance is crucial for generalization

Learning from Examples (II)

Learning by Regression:

- Predicts a continuous value based on input data (minimize error via loss function)
- Key Idea: Fits a straight line (or plane) to minimize prediction errors
- Learning: Adjusts weights step-by-step using gradient descent to reduce error

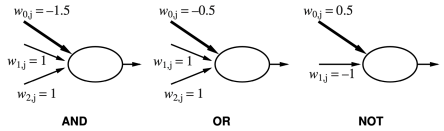
- Challenges: Overfitting with too many variables or noisy data; solved by regularization

Perceptrons Learning:

- Input: The perceptron receives several input signals, computes the weighted sum of inputs
- Threshold Check: If the sum is greater than or equal to a threshold, the perceptron outputs 1, otherwise 0 (threshold could also be activation function ; not as harsh)
- Perceptrons learn by adjusting weights to reduce the error (predictions/outputs)

Perceptron Learning Rule

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



Neural Networks:

- Input Layer, Hidden Layer, Output Layer
- Neurons in one layer are connected to neurons in the next layer via weights
- Each neuron calculates a weighted sum of its inputs and applies an activation function to decide the output
- **Feedforward** Neural Networks: information flows in one direction (input to output)
- **Single-Layer Perceptrons**: can only solve simple, linearly separable problems
- **Multi-Layer Perceptrons:** include one or more hidden layers, making them capable of approximating any function given enough neurons and layers (Universal Approximation Theorem)

Neural Network Learning:

- Learn by adjusting the weights of connections between neurons to minimize the error between their predictions and the actual outputs

Forward Propagation: input data moves forward through the network

- Each neuron computes a weighted sum of its inputs, applies an activation function, and passes the result to the next layer —> network produces a final output
- Error Calculation: Common loss function: Loss = 1/2 (y-´y)^2 (´y prediction)

Backward Propagation (Backpropagation): error is sent backward through the network

- Each neuron adjusts its weights slightly to reduce its contribution to the error

Applications:

- Image Recognition, Speech Recognition, Natural Language Processing (NLP), Finance, Healthcare,
- Advantages: Adaptability, Accuracy, Automation
- Challenges: Need for Large Data, Interpretability, Computational Requirements

Learning from Examples (III)

Computation Graphs:

- A way to view NNs as data flow diagrams, showing operations like addition, multiplication, and activation
- Input Layer: Receives & encodes input data
- Hidden Layers: Transform input data into abstract representations through weighted connections and activation functions
- Output Layer: Produces the network's predictions
- Learning: Uses backpropagation to adjust weights via gradient descent

Convolutional Neural Networks (CNNs):

- Specialized neural networks designed to process image data effectively (not fully connected)
- Perform convolution operations by sliding small filters/kernels across the input

- Output: Feature Maps, representing the detected features
- Pooling Layers: Reduce the size of Feature Maps to decrease computation and increase robustness

Recurrent Neural Networks (RNNs):

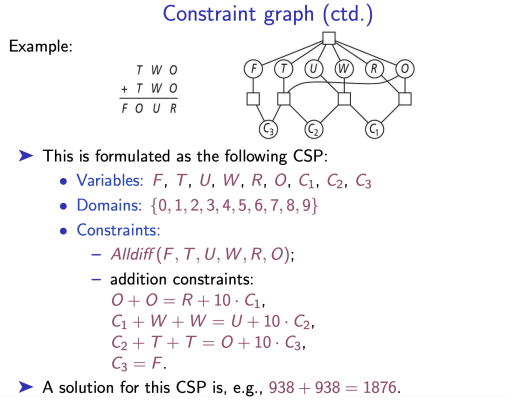
- specialized for sequential data, capable of retaining information through directed cycles (short term memory)
- Challenges: Vanishing gradients, instability, and computational inefficiency
- Solutions: LSTMs and GRUs mitigate long-term dependency issues, while bidirectional RNNs enhance context understanding

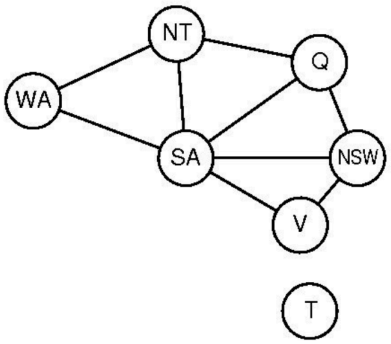
Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSPs):

A CSP is a specialized type of search problem characterized by:

- **Variables**: Entities that must be assigned a value
- **Domains**: Each variable has a set of possible values
- **Constraints**: Rules that determine which combinations of values are valid
- Example: Map-Coloring Problem
- Binary CSPs (where constraints involve pairs of variables), problems can be visualized as a constraint graph: Nodes: represent variables, Edges: constraints





Algorithms for Solving CSPs:

- **Backtracking Search:** depth-first search that assigns values to one variable at a time, checks consistency after each assignment
- Heuristics to Improve Backtracking: **Minimum Remaining Values (MRV), Degree Heuristic, Least Constraining Value (LCV):**
- **Forward Checking:** After assigning a variable, eliminate inconsistent values from the domains of its neighbors
- The simplest form of constraint propagation is arc consistency; ensures that for every value of X , there is a valid value for Y (and vice versa) in a constraint X -> Y

Knowledge Representation

- Structures knowledge for reasoning and decision-making
- Components: Logic (rules), ontology (concepts), and data structures
- Approaches: Declarative (explicit rules) or procedural (embedded in actions)
- Applications: Expert systems, robotics, natural language understanding
- Goal: Make implicit knowledge computable

Knowledge-Based Agents

- Use a knowledge base (KB) to store and reason about facts
- Operate using Tell (add knowledge) and Ask (query knowledge)
- Perform three steps: perceive, infer, act

- Combine domain-specific knowledge with general reasoning
- Example: Diagnosing systems or decision-making robots

Elements of Propositional and First-Order Logic:

- Syntax: Rules that define valid sentences (e.g., $x + 2 \geq y$ is valid; $x^2 + y >$ is not)
- Semantics: Assigns meaning to sentences, determining their truth in a given world
- Entailment: KB \models α : Sentence α is true in all worlds where the knowledge base KB is true
- Equivalence: α ≡ β : Sentences α and β are true in the same models
- Satisfiability: A sentence has at least one model where it is true
- Validity: A sentence is true in all possible models
- Inference: the process of deriving new sentences (or conclusions) from a given knowledge base (KB) using rules.

1. Propositional Logic Example:
   A ⇒ B : "If it rains, then the ground is wet."
2. First-Order Logic Example:
   ∀x(Dog(x) ⇒ Mammal(x)) : "All dogs are mammals."

Reasoning:

- **Entailment** (logische Folgerung) vs. **Inference** (Ableitung)
- Entailment is based on models -> expresses a semantical relation
- Inference is based on derivations -> relation between syntactical elements
- **Soundness** (Korrektheit): Alles, was abgeleitet wird, ist logisch richtig
- **Completeness** (Vollständigkeit): Es gibt nichts, was „übersehen" wird

Inference Rules (Schlussfolgerungsregeln):
Modus Ponens:
Wenn „ A -> B " (Wenn A, dann B) und A wahr ist, folgt B ist wahr.
Beispiel: „Wenn es regnet, ist die Straße nass" + „Es regnet" → „Die Straße ist nass".
^-Elimination:
Wenn „ A ^ B " (A und B) wahr ist, dann kannst du einzeln sagen, dass A wahr ist oder B wahr ist.
Beispiel: „Die Katze ist schwarz und schläft" → „Die Katze ist schwarz".
v-Introduction:
Wenn A wahr ist, kannst du sagen, dass „ A v B " (A oder B) wahr ist.
Beispiel: „Es regnet" → „Es regnet oder die Sonne scheint".

Cut Rule:
Wenn du \alpha beweisen kannst und daraus \beta folgern kannst, dann kannst du \beta direkt aus KB ableiten.
Beispiel: KB = {A, A => B\} .
Du folgerst erst A → dann B .
Deduction Theorem:
Wenn du mit \alpha im Hintergrund \beta beweisen kannst, dann kannst du \alpha =>\beta (Wenn \alpha, dann \beta) ableiten.
Proof by Contradiction (Beweis durch Widerspruch):
Wenn du annimmst, \alpha ist wahr, und das führt zu einem Widerspruch, dann muss \neg \alpha (Nicht-\alpha) wahr sein.
Beispiel: „Nehmen wir an, es gibt keine Lösung. Aber ich finde eine Lösung → Widerspruch! Also muss es eine Lösung geben."

Resolution: spezielle Methode, um logische Aussagen effizient zu überprüfen:

1. Schreibe alles in eine spezielle Form (CNF: Konjunktive Normalform).
   - (A \vee \neg B) \wedge (\neg A \vee C) → Kombination von „oder" und „und".
2. Suche Widersprüche (z.B. A und \neg A).
3. Wenn du einen Widerspruch findest, ist die Annahme falsch.

Ontological Engineering:

- A formal framework for describing concepts and their relationships
- Upper Ontology: general framework for fundamental concepts such as time, space, and objects
- Categories: Help classify objects and make predictions (e.g., Taxonomies)
- Composition: Represents relationships between parts and wholes (e.g., PartOf)
- Substances vs. Objects: Differentiates between countable things and uncountable materials.
- Properties: Divided into intrinsic (inherent) and extrinsic (context-dependent)

Planning

General Considerations:

- Planning is finding a sequence of actions to achieve a goal
- States: Represent the condition of the world at a given time
- Transition: Applying an action transforms one state into another

Challenges in Planning

- **Frame Problem**: Representing what remains unchanged after an action
- **Ramification Problem**: Dealing with indirect effects
- **Qualification Problem**: Listing all conditions needed for an action to succeed
- **Search vs. Planning**: limit irrelevant actions (search explores all)

STRIPS Planning Language

Action(Fly(p,from,to)),
Precond : At(p,from) ∧Plane(p) ∧Airport(from) ∧Airport(to)
Effect : ¬At (p,from) ∧At (p,to)

ADL (Action Description Language): More expressive: Allows negative literals, quantified variables, and conditional effects.

Fly(Plane, From, To) with the condition From ≠ To.

Search Strategies in Planning:

- Forward State-Space Search (Progression): Start from the initial state and find actions leading to the goal
- Backward State-Space Search (Regression): Start from the goal and determine required actions
- Partial-Order Planning: Doesn't commit to a fixed sequence of actions upfront (POP-Algorithm)

Making Simple Decisions

Decision Theory:

- Decision-Theoretic Agents: Combine probability and utility to make rational decisions, evaluate the quality of outcomes continuously
- Utility Function: Assigns a numerical value to each state or outcome, reflecting its desirability
- Expected Utility (EU): Average utility of outcomes, weighted by probabilities
- Lottery: A lottery L, with possible outcomes S1,...,Sn that occur with probabilities p1,...,pn is written as L= [p1,S1; p2,S2;...; pn,Sn]

Axioms of Utility Theory (rational preferences follow these rules)

1. Orderability: You can always decide which option you prefer or if you're indifferent
2. Transitivity: If A > B and B > C, then A > C
3. Continuity: Preferences can be expressed in terms of probabilities (e.g., a lottery between two outcomes can balance with certainty)
4. Substitutability: Indifference between outcomes extends to lotteries involving them
5. Monotonicity: Higher probability for a preferred outcome makes the lottery more desirable

6. Decomposability: Compound lotteries can be reduced to simpler ones

Violating these axioms leads to irrational behavior (e.g., losing money through inconsistent decisions)

Utility Scales:

- Normalize utility: Assign 0 to the worst possible outcome and 1 to the best
- For example, micromorts and QALYs measure the value of life and health in medical contexts
- Money's utility isn't linear; the first million dollars is worth more than the next million.
- People are often risk-averse (prefer certain gains over risky, higher-expected-value outcomes).

Paradoxes in Decision Making:

- Allais Paradox: People prefer certainty (e.g., $3000 guaranteed) over risky higher payoffs, but behave inconsistently in similar scenarios.
- Certainty effect: Strong attraction to guaranteed outcomes.
- Ellsberg Paradox: People avoid ambiguous probabilities even when they offer similar or better chances (ambiguity aversion).
- Example: Prefer known 1/3 chance over unknown odds (0–2/3).

Decision Networks:

- Chance Nodes (ovals): Represent random variables with uncertainty.
- Decision Nodes (rectangles): Represent choices available to the agent.
- Utility Nodes (diamonds): Represent the utility of outcomes.
- Evaluating Decision Networks: Set evidence for the current state, For each action, calculate: probabilities of outcomes & resulting utility-pick action with highest utility

The Value of Information (VoI)

- Iimprovement in expected utility from acquiring new information before deciding
- Example: In an oil survey, knowing where the oil is worth as much as the price of the block.
- Key Properties:
- Non-negativity: Information is always valuable or neutral.
- Order Independence: Order of information doesn't affect its value.