INTELLIGENT AGENTS

Intelligent agent: perceives its environment through sensors and acts upon the environment through actuators.
Percept sequence: Complete history of its perception (sensor readings) **Agent function**: Maps percept sequence to an action **Rational agent**: For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the prior percept sequence and its built-in knowledge **Omniscient agent** Agent knows the actual outcome of its actions (impossible in reality) **Learning** Learn from perception, i.e. agent improves their knowledge of the environment over time **Autonomy** Agent is considered more autonomous if it is less dependent on prior knowledge and uses newly learned abilities **Task environment**: Defined by PEAS (performance measure, environment, actuators, sensors). **Properties**: **Fully vs. partially observable** (agent can/cannot detect complete state of env.) **Single vs. multi agent** (single/several agent(s) in env.) **Deterministic vs. stochastic** (next state is (not) fully determined by its current state and the action of the agent) **Episodic vs. sequential** (action in one episode does (not) affect later episodes) **Discrete vs. continuous** (applies to both state & time) **Static vs. dynamic** (env. only changes based on actions of the agent (not)) **Known vs. unknown** (agent (not) knows the outcome of its action)
Agent types: **Simple reflex agent**: Selects action only based on current percept **Model-based reflex agent**: Has internal state that depends on the percept history → can handle partial observability **Goal-based agent**: Additional has explicit goal → reflected in actions performed by agent; Reaching goal is achieved by search and planning [ex: "reach final destination"] **Utility-based ag.** Additionally tries to maximize utility [ex: "reach final dest. in 25 min."]
All can be turned into **Learning agents**.

UNINFORMED SEARCH

Tree search: actions not reversible ∧ finite
acyclic transition graph ⇒ finite search tree
Properties: No additional information besides problem statement **Well defined problems**: Initial state, actions, transition model, goal test, path cost **Tree search**: Current node + frontier list → can get stuck in loops **Graph search**: Additionally has "explored set", so a node cannot be visited twice **Notation** branching factor b (max. number of successors of any node), depth d of shallowest goal node, Max. length m of any path in the state space, cost C^* of optimal solution, ε minimum step cost

Criterion	BFS	Dijkstra	DFS	DLS	IDS
Complete?	Yes[1]	Yes[1,2]	No*	Yes, if $l \geq d$	Yes[1]
Optimal?	Yes[3]	Yes	No	No, if $l > d$	Yes[3]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$ → rec.	$O(bl)$	$O(bd)$

1: complete if b is finite (and d) 2: Complete if step costs $\geq \varepsilon$ for pos. ε 3. optimal if all step costs are identical

FIFO
Breadth-first search (BFS): Expands shallowest nodes first. Goal test performed for children of current node **Uniform-cost search (Dijkstra)** Order frontier as priorit queue, ordered by g → Expands node with lowest path cost $g(n)$. Goal test performed when node selected for expansion (≠ BFS) **Depth-first search (DFS)** Expands deepest unexpanded node first. *Not complete if recursively implemented; Complete if repeated states are avoided and state space is finite. Does not terminate in infinite state spaces.
LIFO **Depth-limited search** l: depth limit **Iterative deepening search (IDS)** Calls DFS with increasing l. **Bidirectional search** Run two searches: One from initial state and one from goal. Motivation: $b^{d/2} + b^{d/2} < b^d$. Difficult when goal state is unknown or there are multiple goals or the actions are not reversible.

```
function Breadth-First/Depth-First (problem) returns a solution, or failure
node ← a node with State = problem.Initial-State
if problem.Goal-Test(node.State) then return Solution(node)
frontier ← a FIFO/LIFO queue with node as the only element
explored ← an empty set                                   Graph search w/ Tree search modifications
loop do
    if Empty(frontier) then return failure
    node ← Pop(frontier) /*chooses the shallowest/deepest 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 neither in explored nor frontier then
            if problem.Goal-Test(child.State) then return Solution(child)
            frontier ← Insert(child, frontier)
```

```
function Depth-Limited-S. (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)                    Tree search
else if limit = 0 then return cutoff                                           limit = 2:
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
```

INFORMED SEARCH

Properties Know whether a state is "better" than another to reach a goal by using an evaluation function: $f(n) = g(n) + h(n)$
g: sum of path cost, h: estimated cost current node → goal, $h(n) \geq 0$, $h(\text{goal}) = 0$; all informed search alg. are based on Dijkstra's alg.

Greedy best-first search Expands node closest to goal by using heuristic func $f(n) = h(n)$. Completeness: Yes, with graph search, else not Not optimal. Time $O(b^m)$ (heuristic misleading, solution found last). Space $O(b^m)$ **A* search** $f(n) = g(n) + h(n)$ → never overestimate the cost. **Admissible**: $h(n) \leq g(n, \text{goal})$ → needed for optimality of tree search. **Consistent** $h(n) \leq g(n, n') + h(n')$ → needed for optimality of graph search (stronger). Completeness: Yes, if costs > 0. Optimal if $h(n)$ is valid. Time $O(b^{\varepsilon \cdot d})$. Space $O(b^{\varepsilon \cdot d})$. $\varepsilon = \frac{h^* - h}{h^*}$, h^*/h: actual/estimated cost from root to goal. **Heuristic function** **Effective branching factor** b^* avg. number of children in search tree → Lower b^*, better heuristic **Domination** h_2 dominates h_1 if $h_2(n) \geq h_1(n)$ for all nodes $n \to A^*$ using h_2 will never expand more nodes than with h_1
Obtain heuristics e.g. from relaxed problems or pattern databases

```
function Uniform-Cost/Greedy-Best-First/A*(problem) returns a solution, or failure
node ← a node with State = problem.Initial-State
frontier ← a priority queue by Evaluation-Function with node as the only element
explored ← an empty set                                   Graph search w/ Tree search modifications
loop do
    if Empty(frontier) then return failure
    node ← Pop(frontier) /*chooses the node in frontier with lowest Evaluation-Function*/
    if problem.Goal-Test(node.State) then return Solution(node)
    add node.State to explored
    for each action in problem.Actions(node.State) do
        child ← Child-Node(problem, node, action)
        if child.State is neither in explored nor frontier then
            frontier ← Insert(child, frontier)
        else if child.State is in frontier with higher Evaluation-Function then
            replace that frontier node with child
```

CONSTRAINT SATISFACTION PROBLEMS

CSP (X, D, C) $X = \{X_1, \dots, X_n\}$ variables, $D = \{D_1, \dots, D_n\}$ domains of vars, $C = \{C_1, \dots, C_m\}$ constraints, $D_i = \{v_1, \dots, v_k\}$ allowable values **Constraint types** Unary (1 var involved), Binary (2 vars), Higher-Order (>2 vars) **Convert n-ary constraint to binary constraint** 1. Replace ℓ-ary constraint with var Z with domain of ℓ-tuple and relation $R(X_1, X_2, \dots, X_\ell) : dom(Z) = \{(z_1, \dots, z_\ell) \mid R(z_1, \dots, z_\ell), z_i \in dom(X_i), \dots, z_\ell \in dom(X_\ell)\}$ 2. Introduce new binary constraints to match values of Z with values of neighboring vars: $C' = \{\langle Z, X_i \rangle, fst(Z) = X_i\}, \dots, \langle Z, X_\ell \rangle, \ell^{th}(Z) = X_\ell\}$ 3. New CSP: $X' = X \cup Z$, $D' = D \cup dom(Z)$, C' [Hidden transformation] **Backtracking heuristics** **Minimum Remaining Value (MRV)** (Variable Selection) Assign Var with fewest possible values first **Degree heuristic** (Variable Selection) Select variable involved in the largest number of constraints on other unassigned vars.
Least Constraining Value (Value Selection) Choose value that rules out fewest choices for neighboring values in the constraint graph **Backtracking Search** DFS for CSPs with single variable assignment in each step 1. Select Var to assign 2. Set value of domain 3. Check consistency (inference) → add new assignment or backtrack and try new assignment if you find inconsistency. **Inference algorithms** **Forward Checking** (After each assignment): inconsistent values of neighboring vars are removed. **Arc consistency algorithm (AC-3)** (also as pre-processing): Inconsistent values of all Vars are removed.

```
function AC-3 (csp, queue) returns failure or the reduced csp otherwise
inputs: csp: a binary CSP, queue: a queue of arcs (X_i, X_j)
while queue is not empty do                                   Constraint graph:
    (X_i, X_j) ← Remove-First(queue)                          
    if Remove-Inconsistent-Values(X_i, X_j) then              A = B + C
        if size of Domain(X_i) = 0 then return failure
        for each X_k in Neighbors[X_i] \ {X_j} do             including assigned neighbors
            add (X_k, X_i) to queue
return csp
function Remove-Inconsistent-Values (X_i, X_j) returns true iff succeeds
removed ← false
for each x in Domain[X_i] do
    if no value y in Domain[X_j] allows (x,y) to satisfy the constraint of (X_i, X_j)
        then delete x from Domain[X_i]; removed ← true
return removed
```

Initialization of variable "queue" in AC-3: 1) After each assignment: After assigning Var X_i, add the arcs (X_i, X_j) to queue (X_j: all unassigned neighbors of X_i). 2) As pre-processing: add all arcs of the CSP to queue **Arc consistency** 1) Variable: X_i is arc-consistent with X_j, if for every value in D_i there exists a value in D_j satisfying the binary constraint of the arc (X_i, X_j). 2) CSP: A constraint graph is arc-con. if every Var is arc-con. with every other Var. **Independent subproblems** Completely disconnected in constraint graph. **Tree-structured CSPs** (if graph has no loops): Can be solved in $O(n \cdot d^2)$ time (vs. $O(d^n)$ in general) (n nodes, d domain size) → Use Conditioning and Tree decomposition for transformation into tree.

LOGICAL AGENTS / PROPOSITIONAL LOGIC

Satisfaction If sentence α is true in model m → m satisfies α. KB satisfiable ⇔ $M(KB) \neq \emptyset$ [$M(\alpha)$: set of all models satisfying α] **Entailment** Truth of one sentence $\overset{\alpha}{}$ requires the truth of the other $\overset{\beta}{}$ sentence: $\alpha \models \beta$ if α entails β. Or: $\alpha \models \beta$ iff $M(\alpha) \subseteq M(\beta)$. $(x=0) \models (xy=0)$. **Inference by enumeration** Enumerate all models. $KB \models \alpha \Leftrightarrow \alpha$ is true in all models in which KB is true. **Theorem proving** Apply inference rules **Logical equivalences** • $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$. • $(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ **Validity/Tautology** Sentence is valid iff true for all models. $M(\alpha \vee \neg\alpha) = M(\text{true}) = \text{All Possible Models}$. α valid ⇔ $\neg\alpha$ unsatisfiable **Satisfiability** Sentence is satisfiable if true for some models. Unsatisfiable. $M(\alpha \wedge \neg\alpha) = M(\text{false}) = \emptyset$ **Inference rules** **Modus Ponens** $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$ **And-Elimination** $\frac{\alpha \wedge \beta}{\alpha}$ **Conjunctive normal form (CNF)** $(A \vee B \vee C) \wedge (\neg A \vee \neg C)$ **Resolution algorithm** To prove $KB \models \alpha$, we show $KB \wedge \neg\alpha$ is unsatisfiable. Apply resolution rule $\frac{A \vee B, \neg B \vee C}{A \vee C}$ to CNF until: 1) no new clause produced ⇒ $KB \not\models \alpha$ (proof by contradiction) 2) Clauses resolve to yield empty clause $\emptyset$ ⇒ $KB \models \alpha$ **Horn clauses** $(p_1 \wedge \dots \wedge p_k) \Rightarrow q$, q and p_i: propositional symbols. KB of Horn clauses only requires Modus Ponens for inference → Complete for horn clauses
AND-OR graph $A \wedge B \Rightarrow C$, $C \Rightarrow D$  (Complete) **Forward + Backward Chaining** Linear time. Data/goal driven. May do irrelevant work.

FIRST-ORDER LOGIC (FOL)

Quantifiers ⇒ goes with ∀ and ∧ with ∃. **Logic translation** 1) $\forall x\ P(x) \Rightarrow Q(x)$: All Ps are Qs. 2) $\forall x\ P(x) \Rightarrow \neg Q(x)$: No Ps are Qs. 3) $\exists x\ P(x) \wedge Q(x)$: Some Ps are Qs. 4) $\exists x\ P(x) \wedge \neg Q(x)$: Some Ps aren't Qs.
Syntax Atomic Sentence ::= Predicate | Predicate(Term,...) | Term = Term; Term ::= Function(Term,...) | Constant | Variable **De Morgan rules** quantifiers $\forall x\ \neg P \equiv \neg\exists x\ P$, $\neg\forall x\ P \equiv \exists x \neg P$, $\forall x\ P \equiv \neg\exists x \neg P$, $\exists x \equiv \neg\forall x \neg P$
predicates represent relations

INFERENCE IN FOL

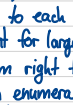
Removing Quantifiers ∀: Infer any sentence from substituting a ground term for the variable $\frac{\forall x\ \alpha}{\text{Subst}\{x/g\}, \alpha}$ (several times applicable) ∃: Replace Var by a single new Skolem constant $\frac{\exists x\ \alpha}{\text{Subst}\{x/k\}, \alpha}$ (once applicable) **Proportionalization** Instantiate universal sentence in all possible ways **Generalized Modus Ponens** For atomic sentences p_i, p_i' and q, where $\forall i$ there exists a subst. θ such that $\text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i)$: $\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{Subst}(\theta, q)}$ (Complete for horn clauses) **Unification** $\text{Unify}(p, q) = \theta$ s.t. $\text{Subst}(\theta, p) = \text{Subst}(\theta, q)$, e.g. $\theta = \{x/J, y/E\}$ **FOL Horn Clauses** Like in PL + universally quantified vars ("∀" is typically omitted) **Backward Chaining algorithm** Start from goal, search rules to find known facts that support the proof **Needs** horn clauses (then complete).

goals: {goal, ...} θ applied. Initialize with sentence to prove.
θ ← {...} Current substitution. Initialize with empty set ϕ.
$q' \leftarrow \text{Subst}(\theta, \text{goal})$ → q': ...
Using rule i: sentence$_1 \wedge \dots \wedge$ sentence$_n \Rightarrow$ goal **Standardize apart** $\theta' \leftarrow \{x_n/\text{object}\}$ Current unification
new_goals ← {sentence$_1$, ..., sentence$_n$, old_goal$_1$, ...}
if goals empty → successful proof

Conversion to CNF 1) **Eliminate** implications ($A \Rightarrow B \equiv \neg A \vee B$) 2) **Move** ¬ inwards 3) **Standardize** vars ($x \to x_n$) 4) **Skolemization** ($\forall c\ \exists p\ \text{Parent}(p,c) \to \forall c\ \text{Parent}(F(c), c)$) 5) Drop universal quantifiers ∀ 6) Distribute ∨ over ∧
Resolution rule $\frac{\ell_1 \vee \dots \vee \ell_k \vee p,\ \neg q \vee m_1 \vee \dots \vee m_n}{\text{Subst}(\theta, \ell_1 \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_n)}$ with $\theta = \text{Unify}(p, \neg q)$
Resolution algorithm Like in PL but important to standardize apart. In FOL, resolution can always prove that a sentence is unsatisfiable. Attempting to prove a satisfiable first-order formula as unsatisfiable may result in a non-terminating computation (unlike in PL).

BAYESIAN NETWORKS

static environment, without actions
Bayesian network Directed acyclic graph, where • each node corresponds to a random variable, • arrows between nodes start at parents, • each node N_i has a conditional probability distribution $P(X_i \mid \text{Parents}(X_i))$ **Bayes rule** $P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)}$
Independence $P(X,Y) = P(X) \cdot P(Y) \Leftrightarrow$ Variables X and Y share no common ancestry. **Conditional independence** X conditionally independent of Y given E: $P(X,Y|E) = P(X|E) \cdot P(Y|E) \Leftrightarrow P(X|Y,E) = P(X|E) \Leftrightarrow X$ and Y are independent given E iff every path U from X to Y is blocked, i.e. for every path U after all manipulations there is no undirected path between X and Y.

Normalization $P(X|Y) = \alpha \cdot P(X,Y)$, $\alpha \cdot P(\neg x, Y) + \alpha \cdot P(x, Y) \doteq 1$
Joint probability $P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents}(X_i))$ **Markov blanket** A node in the Bayesian network is independent of all other nodes given its parents, children and children's parents.
Inference by Enumeration $P(a|d,e) = \alpha \cdot P(a,d,e) = \alpha \cdot \sum_B \sum_C P(a,B,C,d,e)$
$= \alpha \cdot P(a) \cdot \sum_C P(d|C) \cdot P(e|C) \cdot \sum_B P(B) \cdot P(C|a,B)$
Construct evaluation tree and multiply down to each leaf and sum over all paths. → very inefficient for large networks. 
Inference by Elimination Sum from right to left, storing intermediate results (often faster than enumeration)
$P(A|d,e) = \alpha \cdot P(A) \cdot \sum_C P(d|C) \cdot P(e|C) \cdot \sum_B P(B) \cdot P(C|A,B)$
$= \alpha \cdot f_A(A) \times \sum_C f_{\bar{B}}(C) \times f_{\bar{C}}(C) \times \sum_B f_B(B) \times f_{\bar{C}}(C,A,B)$ (x: point-wise multiplication)
(1×2, 2, 2, 4×8×2, 2×2×2)
$f_{\bar{B}}(C,A,B) = \left\{ \begin{bmatrix} P(c|a,b) & P(c|a,\neg b) \\ P(c|\neg a,b) & P(c|\neg a,\neg b) \end{bmatrix}, \begin{bmatrix} P(\neg c|a,\neg b) & P(\neg c|a,\neg b) \\ P(\neg c|\neg a,\neg b) & P(\neg c|\neg a,\neg b) \end{bmatrix} \right\}$
Variable Ordering Heuristics: Eliminate whichever variable minimizes the size of the next factor to be constructed. **Monte Carlo Simulation** Computation for large Bayesian Networks is too expensive → MC Simulation. Basic procedure: 1) Create samples according to given probabilities. 2) Deterministic simulations 3) Aggregation of individual simulations to obtain expected values of probability distributions **Direct Sampling** S_{PS}: Prob. that a specific event is generated by sampling, N_{PS}: # generated samples for a specific event. $S_{PS}(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents}(X_i)) = P(x_1, \dots, x_n) \approx N_{PS}(x_1, \dots, x_n) / N$ **Rejection sampling** Used to produce samples from a hard-to-sample distribution given an easy-to-sample distribution (e.g. full joint prob). Determine $P(X|e)$ by rejecting all samples not matching evidence e. $P(X|e) \approx \frac{N_{PS}(X,e)}{N_{PS}(e)}$. Expensive for small $P(e)$. **Likelihood Weighting** More efficient 1) Fix values of evidence $E \to$ only sample non-evidence vars 2) Not all events are equally likely, weight each event with likelihood of occurance.

HIDDEN MARKOV MODELS

dynamic environment, without actions

X_{k-1}	$P(X_k \mid X_{k-1})$	X_k	$P(E_k \mid X_k)$
+	...	+	...
f	...	f	...

Stochastic process Sequence of RVs $X_1, X_2, X_3, \ldots$ **Markov Process** Stochastic process that has the **Markov property** **Markov property** $P(X_n \mid X_{n-1}, X_{n-2}, \ldots, X_0) = P(X_n \mid X_{n-1})$ **Stationary process** Stochastic process whose joint prob. distribution does not change when shifted in time. For Markov processes: $\forall t: P(X_n \mid X_{n-1}) = P(X_{n+t} \mid X_{n-1+t})$

Stationary Markov Chain Discrete stat. process with Markov prop. $P(X_n = x_i) = \sum_{j=1}^{S} P(X_n = x_i \mid X_{n-1} = x_j) \cdot P(X_{n-1} = x_j)$. Matrix notation: $p_n = T \cdot p_{n-1}$, $(p_i)_n = P(X_n = x_i)$, $T_{i,j} = P(X_n = x_i \mid X_{n-1} = x_j)$ **Convert m^{th}-order MCs to first-order:** Introduce new states corresponding to the m-th previous states: $P(X_n = x_n \mid X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}) = P(X_n = \hat{x}_n \mid X_{n-1} = \hat{x}_{n-1})$ with $\hat{x}_n = (x_n, x_{n-1})$, $\hat{x}_{n-1} = (x_{n-1}, x_{n-2})$ **HMM** = Markov Chain + sensor model. State cannot be measured directly → infered from sensor values (which only depend on current state: $P(E_t \mid X_t)$)

Basic formulas $(p_t)_k = P(X_k = x_t)$, $(\hat{p}_t)_k = P(E_t = e_t)$

$\dim(T) = |S| \times |S|^m$ $T_{i,j} = P(X_k = x_i \mid X_{k-1} = x_j) = \begin{bmatrix} P(x_k \mid x_{k-1}) & P(x_k \mid \neg x_{k-1}) \\ P(\neg x_k \mid x_{k-1}) & P(\neg x_k \mid \neg x_{k-1}) \end{bmatrix}$

$p_k = T \cdot p_{k-1}$, $\hat{p}_k = H \cdot p_k$, $H_{i,j} = P(E_k = e_i \mid X_k = x_j)$

Joint prob. distr.: $P(X_{0:t}, E_{1:t}) = \left(\prod_{i=1}^{t} P(E_i \mid X_i) \cdot P(X_i \mid X_{i-1})\right) \cdot P(X_0)$

Filtering Recursive state estimation algorithm. Time: $O(t)$. Space: $O(1)$

$$f_{1:t} = P(X_t \mid e_{1:t}) = \alpha \cdot P(e_t \mid X_t) \cdot \sum_{x_{t-1}} P(X_t \mid X_{t-1}) \cdot P(X_{t-1} \mid e_{1:t-1}) = \alpha \cdot O_t \cdot T^{\top} \cdot f_{1:t-1}$$

Begin: $t = 1$

$(f_i)_{1:t} = P(X_t = x_i \mid e_{1:t})$, Initialization: $f_{1:0} = (f_i)_{1:0} = P(X_0 = x_i)$

$(O_{i,j})_t = \begin{cases} P(e_t \mid X_t = x_i), \text{ if } i = j \\ 0, \text{ else} \end{cases} = \begin{bmatrix} P(e_t \mid x_t) & 0 \\ 0 & P(e_t \mid \neg x_t) \end{bmatrix}$

Prediction Like filtering without adding evidence (inaccurate for high k)

$$P(X_{t+k+1} \mid e_{1:t}) = \sum_{x_{t+k}} P(X_{t+k+1} \mid X_{t+k}) \cdot P(X_{t+k} \mid e_{1:t}) = T^{k+1} \cdot f_{1:t}$$

Smoothing Computing the distribution over past states given evidence up to the present. $P(X_k \mid e_{1:t})$ for $0 \le k < t$.

$$P(X_k \mid e_{1:t}) = \alpha \cdot P(X_k \mid e_{1:k}) \times P(e_{k+1:t} \mid X_k) = \alpha \cdot f_{1:k} \times b_{k+1:t}$$

Backward recursion: Initialization with $b_{t+1:t} = (1, \ldots, 1)^{\top}$

$$b_{k+1:t} = P(e_{k+1:t} \mid X_k) = \sum_{x_{k+1}} P(e_{k+1} \mid X_{k+1}) \cdot P(e_{k+2:t} \mid X_{k+1}) \cdot P(X_{k+1} \mid X_k) = T^{\top} \cdot O_{k+1} \cdot b_{k+2:t}$$

$(b_i)_{k+1:t} = P(e_{k+1:t} \mid X_k = x_i)$ Begin: $k = t-1$

For whole sequence time complexity $O(t^2)$, can make it $O(t)$ using dynamic programming (→ forward-backward algorithm)

Viterbi algorithm Determine the most likely sequence of states. Time: $O(t)$. Space: $O(t)$.

Initialize: $\mu_1(X_t) = f_{1:1} = \mu_1(X_t)$

$$\mu_{t+1}(X_{t+1}) = \max_{x_1 \ldots x_t} P(x_1, \ldots, x_t, X_{t+1} \mid e_{1:t+1}) = \alpha \cdot P(e_{t+1} \mid X_{t+1}) \cdot \max_{x_t} \left\{ P(X_{t+1} \mid x_t) \cdot \max_{x_1 \ldots x_{t-1}} P(x_1, \ldots, x_t \mid e_{1:t}) \right\}$$

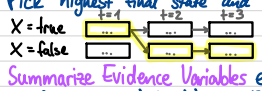
Matrix notation:

$$\mu_{t+1}(X_{t+1}) = \alpha \, O_{t+1} \begin{bmatrix} \max_{X_t} \{(T_{11} \cdot \mu_t(X_t)_1), \ldots, (T_{1n} \cdot \mu_t(X_t)_n)\} \\ \vdots \\ \max_{X_t} \{(T_{n1} \cdot \mu_t(X_t)_1), \ldots, (T_{nn} \cdot \mu_t(X_t)_n)\} \end{bmatrix}$$

(first col / last col)

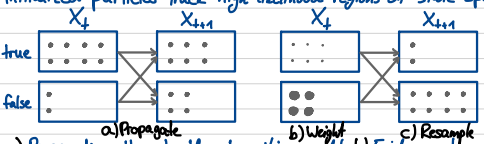
$$\mu_{t+1}(X_{t+1}) = \alpha \, O_{t+1} \begin{bmatrix} \max_{X_t} \{(T_{11} \cdot \mu_t(x_t)), (T_{12} \cdot \mu_t(\neg x_t))\} \\ \max_{X_t} \{(T_{21} \cdot \mu_t(x_t)), (T_{22} \cdot \mu_t(\neg x_t))\} \end{bmatrix}$$

Pick highest final state and backtrack! (α can be omitted)



Summarize Evidence Variables e.g. $A, B \in \{t, f\}$ to $E \in \{(a,b), (a,\neg b), (\neg a,b), (\neg a,\neg b)\}$ (for $k \to \infty$) **Steady state** Condition: $f_{1:k} = f_{1:k+1} = f_{ss}$ $f_{ss} = \alpha \cdot O_k \cdot T \cdot f_{ss}$ (Filtering formula) → solution: Eigenvector $v \to f_{ss} = \vec{\alpha} \cdot v$

Particle Filtering (Monte Carlo for HMM): Population of randomly initialized particles track high-likelihood regions of state-space.



a) Propagation through the transition model b) Evidence observed, weight each sample by its likelihood for the observation c) Generate new set of samples by weighted random selection from current set

RATIONAL / SIMPLE DECISIONS

actions in static envs

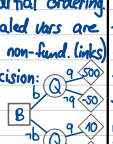
$P(\text{Result}(a) = s' \mid e) = P(s' \mid a, e) = \sum_s P(s) \cdot P(s' \mid s, a, e)$, evidence e, action a

$EU(a \mid e) = \sum_{s'} P(\text{Result}(a) = s' \mid a, e) \cdot U(s')$, $MEU(a \mid e) = \max_a EU(a \mid e)$

$EU(x) = \sum_y EU(x,y) = \sum_y P(x,y) \cdot U(x,y)$ ← decision $P(d_i) = 1$

$EU(x \mid y) = P(x \mid y) \cdot U(x,y) = \sum_z P(x,z \mid y) \cdot U(x,y,z)$

Decision networks Decision, RV, U Bayesian network with decision and utility nodes **Partial ordering** 1) Identify the first decision D_1 and all vars X_0 to make that decision (Identify 2) next decision D_2 and the vars X_i that are revealed between D_1 and D_2 3) Place unrevealed vars X_n at the end of the ordering → $X_0 < D_1 < X_1 < D_2 < \ldots < X_n$, X_i may be a set $\{\ldots\}$ of vars **Fundamental links** Removal would not change partial ordering **No forgetting assumption:** All past decisions and revealed vars are available at the current decision (→ only need to draw non-fund. links) **Decision trees** Nodes as decision networks. EU of a decision: weighted summation of all branches from decision to all reachable leaves from the decision. → exponential complexity with increasing num. of decisions

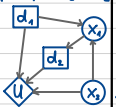


Value of information Guides an agent to choose what information to acquire $VOI_e(E_j) = [\sum_k P(E_j = e_{j,k} \mid e) \cdot MEU(\alpha_{e_{j,k}} \mid e, E_j = e_{j,k})] - MEU(\alpha \mid e)$ = expected value of best action given the information at no charge − expected value of best action without information **Optimal policies** $\pi^*(D_i \mid x_{1:i-1}, d_{1:i-1}) = \arg\max_{D_i} EU(D_i \mid x_{1:i-1}, d_{1:i-1})$ x_i: chance nodes d_i: decision nodes **Joint probability** $P(x_{1:n}, d_{1:n}) = P(x_{1:n} \mid d_{1:n}) = \prod_{i=1}^{n} P(x_i \mid x_{1:i-1}, d_{1:i-1})$ **Decision maximizing the EU for each decision var.**

$$MEU(d_{1:n}) = \max_{d_1} \sum_{x_1} \ldots \max_{d_n} \sum_{x_n} \prod_{i=1}^{n} P(x_i \mid x_{1:i-1}, d_{1:i}) \cdot U(x_{1:n}, d_{1:n})$$

$d_1 < x_1 < d_2 < x_2$

$$EU(d_1) = \max_{d_2} \sum_{x_1} \sum_{x_2} P(x_1, x_2 \mid d_1, d_2) \cdot U(x_1, x_2, d_1, d_2) = \sum_{x_1} P(x_1 \mid d_1) \cdot \max_{d_2} \sum_{x_2} P(x_2 \mid x_1, d_1, d_2) \cdot U(x_1, x_2, d_1, d_2) = \sum_{x_1} P(x_1 \mid d_1) \cdot \max_{d_2} EU(d_2 \mid x_1, d_1)$$



1) Calc $EU(d_i \mid x_{1:i-1}, d_{1:i-1})$ for all possible instantiations of vars $x_{1:i-1}, d_{1:i-1}$ (begin with last decision d_n) 2) Derive policy $\pi^*(D_i \mid x_{1:i-1}, d_{1:i-1})$ for all possible instantiations of vars $x_{1:i-1}, d_{1:i-1}$ 3) Continue with step 1) with d_{i-1} and calc EUs by using the obtained policies (→ replace D_i with $\pi^*(D_i \mid x_{1:i-1}, d_{1:i-1})$ or directly use the max. EUs.

RATIONAL DECISIONS OVER TIME (Complex Decisions) actions in dyn. env.

Markov Decision Process (MDP) Sequential decision problem for a fully observable, stochastic env. with Markovian transition model and additive rewards (MDP = markov chain + actions + rewards) **Definition** A MDP is a 5-tuple (S, A, P, R, γ), where S is a finite set of states, A is a finite set of actions (or $A(s)$ is the finite set of actions available from state s), **transition model** $P_a(s, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$, **Reward** $R(s', a, s)$, **discount factor** $\gamma \in [0,1]$ (represents the difference in importance between future and present rewards). **Aim** Find **optimal policy**, i.e. the best action for every possible state s → the optimal policy maximizes the expected utility (EU)

Utility / value of a state: $U(s)$ = expected (discounted) sum of rewards (until termination) if you start in state s assuming optimal actions. **Q-function** $Q(s,a)$ = (Expected) Utility when starting in a given state with a given action. **Bellmann Principle of Optimality** An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. **Bellman equation** Expected sum of rewards = current reward + γ × expected sum of rewards after taking best action

$$U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U(s') \quad \text{(simplified)}$$

$$= \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot [R(s, a, s') + \gamma \cdot U(s')]$$

Value Iteration Algorithm 1) Start with arbitrary utility values (except for terminal states, where $U(s) = R(s)$) → initial utilities normally $U^0(s) = 0$ 2) Repeat for every s simultaneously (except terminal states) until "no change" (Bellman Update):

$$U^{i+1}(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U^i(s') \quad \text{(simplified)}$$

→ calculate this separately for all actions $a \in A(s)$

$$= \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot [R(s, a, s') + \gamma \cdot U^i(s')]$$

Optimal policy (calculate this for all $a \in A(s)$)

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U(s') \quad \text{(simplified)}$$

$$= \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot [R(s, a, s') + \gamma \cdot U(s')]$$

Policy iteration Possible to get an optimal policy even when the utily function estimate is inaccurate (→ normally less computation needed)

1) Start with an arbitrary initial policy $\pi_0(s)$ for every state s.

2) Policy evaluation: Solve the linear system to compute $U_i(s)$ for all s

$$U_i(s) = R(s) + \gamma \cdot \sum_{s'} P(s' \mid s, \pi_i(s)) \cdot U_i(s') \quad \text{(simplified)}$$

$$= \sum_{s'} P(s' \mid s, \pi_i(s)) \cdot [R(s, \pi_i(s), s') + \gamma \cdot U_i(s')]$$

3) Policy improvement for each state s: (calculate this for all $a \in A(s)$)

$$\pi_{i+1}(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U_i(s') \quad \text{(simplified)}$$

$$= \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot [R(s, a, s') + \gamma \cdot U_i(s')]$$

The algorithm terminates when the policy improvement yields no change in the utilities.

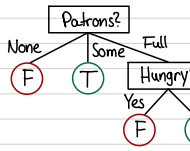
Logical equivalences

$\alpha \wedge \beta$	$\equiv$	$\beta \wedge \alpha$	$\forall x \forall y\ A(x) \Rightarrow (B(y) \Rightarrow C(x,y)) \equiv \forall x \forall y\ (A(x) \wedge B(y)) \Rightarrow C(x,y)$
$\alpha \vee \beta$	$\equiv$	$\beta \vee \alpha$	
$(\alpha \wedge \beta) \wedge \gamma$	$\equiv$	$\alpha \wedge (\beta \wedge \gamma)$	conjunction $\wedge$
$(\alpha \vee \beta) \vee \gamma$	$\equiv$	$\alpha \vee (\beta \vee \gamma)$	disjunction $\vee$
$\neg(\neg\alpha)$	$\equiv$	α	
$\alpha \Rightarrow \beta$	$\equiv$	$\neg\beta \Rightarrow \neg\alpha$	Better use quantifiers without negation or only negate first quantifier!
$\alpha \Rightarrow \beta$	$\equiv$	$\neg\alpha \vee \beta$	
$\alpha \Leftrightarrow \beta$	$\equiv$	$(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$	
$\neg(\alpha \wedge \beta)$	$\equiv$	$\neg\alpha \vee \neg\beta$	
$\neg(\alpha \vee \beta)$	$\equiv$	$\neg\alpha \wedge \neg\beta$	
$\alpha \wedge (\beta \vee \gamma)$	$\equiv$	$(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$	
$\alpha \vee (\beta \wedge \gamma)$	$\equiv$	$(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$	

Performance measure vs. Utility function Performance measure is a specification from the designer or programmer (external to the agent) to specify what the agent should do. If an agent always acts to maximize or achieve the Performance measure, this is a rational agent. / Not all agents do have a Utility function, e.g. reflex agents do not. The Utility func. is used internally by the robot itself to evaluate the best course of action to optimally achieve/maximize the Performance measure(s), given its perceived state.

LEARNING

An agent is learning if it improves its performance on future tasks after making observations about the world. **Types of Learning** **Supervised** Agent observes example input-output pairs and learns a function that maps from input to output. **Unsupervised** Agent learns patterns in data without feedback (labels). **Reinforcement** Agent learn from a series of events and receives either reward or punishment for its actions **Ockham's razor** for supervised learning: Choose the simplest consistent hypothesis **Decision trees** Input $\in \mathbb{R}^n$ → output $\in \{\text{true}, \text{false}\}$ **DT structure** An internal node represents a test of a property. Edges are annotated with the possible test values. Each leaf node has the Boolean value which should be returned **Building DTs** For n boolean attributes: 2^{2^n} possible DTs → use greedy olivide & conquer heuristic to test most important attribute first. **Quantifying importance of attributes** **Entropy** of boolean RV X that is true with prob. q: $H(X) = B(q) = -q \cdot \log_2(q) - (1-q) \cdot \log_2(1-q)$ **Information gain** of an attribute A_d with d values and binary classification $Gain(A) = B(\frac{p}{p+n}) - \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} \cdot B(\frac{p_k}{p_k + n_k})$



p and n: number of pos./neg. samples
p_k and n_k: number of pos./neg. samples of k^{th} value
$B(\frac{3}{5}) = 0.9183$, $B(\frac{4}{11}) = 0.8113$
$B(1) = B(0) = 0$, $B(\frac{1}{2}) = 1$
2^n rows in truth table for n attributes

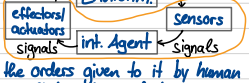
```
function DecisionTreeLearning(examples, attributes, parent_examples) returns tree
  if examples is empty then return Plurality-Value(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return Plurality-Value(examples)
  else
      A ← arg max_{a ∈ attributes} Importance(a, examples)
      tree ← a new decision tree with root test A
      for each value v_k of A do
          exs ← {e | e ∈ examples and e.A = v_k}
          subtree ← DecisionTreeLearning(exs, attributes \ A, examples)
          add a branch to tree with label (A = v_k) and subtree subtree
      return tree
```

A: input attribute
v_k: possible value of input attribute A
Plurality-Value: selects the most common output value among examples.

Reinforcement learning **Agent** Entity perceiving its env and acting upon it **State** Agent observes its state from the environm. **Action** Agent acts on the env. through possible actions. **Reward** Indicates how beneficial the last action was w.r.t. solving the task **Policy** Function which selects the actions for the agent depending on the current state **Exploration** Randomly choosing the next action to create new experience for improving the policy **Exploitation** Using the existing policy to decide for the next action.

ROBOTICS

Robots Physical agents that perform tasks by manipulating the physical world.



3 Laws of Robotics 1) A robot may not injure a human being or, through inaction, allow a human being to come to harm. 2) A robot must obey the orders given to it by human beings, except where such orders would conflict with the first law. 3) A robot must protect its own existence as long as such protection does not conflict with the first or second law **Primary categories** **Manipulators** Robot arms, physically anchored to their workspace **Mobile robots** move about their environ. using wheels, legs or similar mechanisms. **Mobile manipulators** Combination of previous categories, e.g. humanoid robots. **Sensor Categories** **Passive sensors** Detection of effects generated by other sources in the env. (e.g. video/infrared cameras, GPS, light sensor) **Active sensors** Energy is used to send signals into the env. that are reflected and detected (typically more inform provided, but increased power consumption & danger of interference (Sonar, Lidar, Radar) **Proprioceptive sensors** inform robot of its own state (angle, force/strain, acceleration sensor) **Degree of freedom** Number of independent parameters that define the configuration of the robot (body in $\mathbb{R}^3$ has 6 DOFs: 3 translational and 3 rotational, in $\mathbb{R}^2$ 3 DOFs. 2 translational x,y and 1 rotational θ **Nonholonomic** robot has more effective DOFs than controllable DOFs (e.g. car) **Holonomic** effective DOFs = controllable DOFs **Special wheels** Omni wheels (3 subpixels) and Mecanum wheels (2 × 6 wheels) are used to build holonomic mobile platforms **Legged mobile platforms** Good in rough terrain. Bad on flat surfaces compared to wheeled platforms **Snake robots** Versatile (climb, crawl, swim, robust, hard to detect, huge size variations). Very slow. **Sensing** The detection of a physical presence and its conversion into a signal that can be read by an observer or an instrument. **Perception** The organization, identification and interpretation of sensory information in order to represent and understand the environment. / Difficult due to noisy sensors and partially observable, unpredictable and often dynamic env. **Localization** Problem of finding out where things are, including the robot itself. (Particular instance of filtering / state estimation) **Deterministic / stochastic transition model** Model state transitions with(out) considering uncertainty → uncertainty because robots are to some degree unpredictable **Landmark** Stable, recognizable feature of the environment used for navigation. **SLAM** Simultaneous localization and mapping if no map of the env. is available