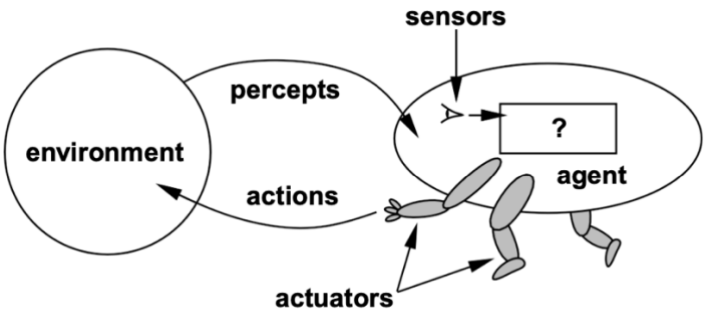
Artificial Intelligence Fundamentals

2 – Agents



Agents include humans, robots, softbots, thermostats, etc. An agent can be anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
The agent function mapsfrom percept histories to actions: f: P*→ A.
The agent program runs on the physical architecture to produce f.
A rational agent is one that does the right thing, we evaluate an agent's behavior by its consequences.
A rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date.

PEAS

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

- Performance measure
- Environment
- Actuators
- Sensors

Environment types

- Observable
- Deterministic

- Episodic
- Static
- Discrete
- Single-agent

The hardest case is partially observable, multiagent, nondeterministic, sequential, dynamic, continuous. The real world falls into this category (of course).

Agent structure

The job of AI is to design an agent program that implements the agent function - the mapping from percepts to actions.
Agent architecture agent = architecture + program

Table driven agent: does do what we want, assuming the table is filled in correctly: it implements the desired agent function.

Four basic types in order of increasing generality:

Simple reflex agents

These agents select actions based on the current percept, ignoring the rest of the percept history. It only works if decisions can be made on just the current percept and the environment is fully observable.

Model-based agents

A model-based agent can handle partially observable environments by the use of a model about the world. The agent has to keep track of the internal state which is adjusted by each percept and that depends on the percept history. The most effective way to handle partial observability is for the agent to keep track of the part of the world «it can't see now».

Goal-based agents

A goal-based agent takes it a step further by using a goal in the future to help make decisions about how best to reach that outcome. Search and planning are the subfields of AI devoted to finding action sequences that achieve the agent's goals.

Utility-based agents

A performance measure assigns a score to any given sequence of environment states, so it can easily distinguish between more and less desirable ways of getting to the taxi's destination. An agent's utility function is essentially an internalization of the performance measure.

Learning agents

A learning agent in AI is the type of agent that can learn from its past experiences, or it has learning capabilities. It starts to act with basic knowledge and then is able to act and adapt automatically through learning. The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting external actions.

4 – Solving by Search

Problem solving agents

When the correct action to take is not immediately obvious, an agent may need to plan ahead: to consider a sequence of actions that form a path to a goal state. Such an agent is called a problem-solving agent, and the computational process it undertakes is called search.

We distinguish between informed algorithms, in which the agent can estimate how far it is from the goal, and uninformed algorithms, where no such estimate is available.

A state space is the set of all possible configurations of a system.

Problem types

- Deterministic, fully observable: single-state problem, agent knows exactly which state it will be in; solution is a sequence
- Non-observable: conformant problem, agent may have no idea where it is; solution (if any) is a sequence
- Nondeterministic and/or partially observable: contingency problem, percepts provide new information about current state. Solution is a contingent plan or a policy; often interleave search, execution
- Unknown state space: exploration problem (“online”)

Single-state problem formulation

A problem is defined by four items: initial state, successor function (actions), goal and path cost. A solution is a sequence of actionsleading from the initial state to a goal state.

Real world is absurdly complex → state space must be abstracted for problem solving.

Tree search algorithms

Basic idea: offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. expanding states).

A strategy is defined by picking the order of node expansion. Strategies are evaluated along the following dimensions:

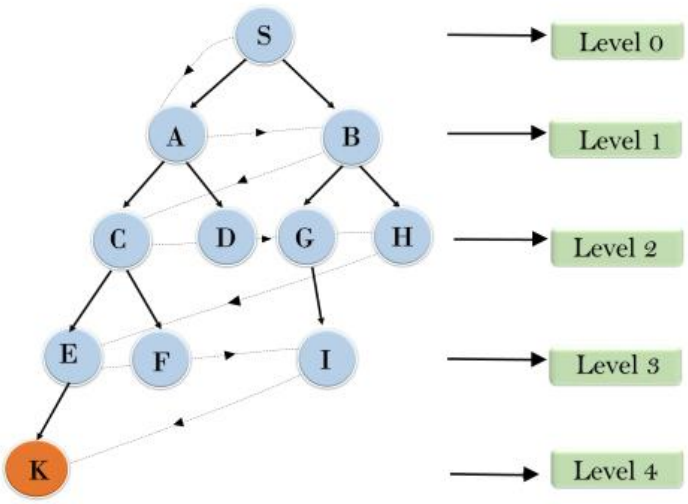
- completeness—does it always find a solution if one exists?
- time complexity—number of nodes generated/expanded
- space complexity—maximum number of nodes in memory
- optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of :

- b—maximum branching factor of the search tree
- d—depth of the least-cost solution
- m—maximum depth of the state space

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

Breadth-first search



Complete? Yes (if b is finite)

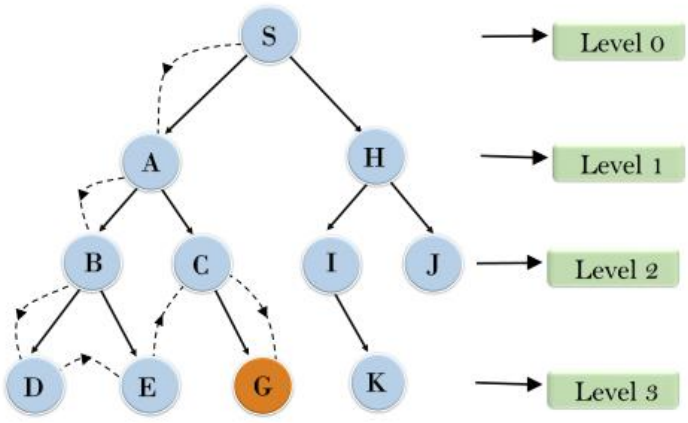
Time? $O(b^{})$

Space? $O(b^{d+1})$ (keeps every node in memory)

Optimal? Yes (if cost = 1 per step); not optimal in general

Space is the big problem

Depth-first search



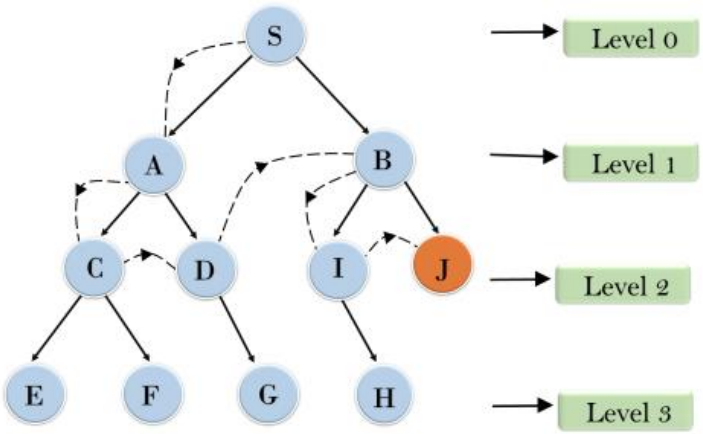
Complete? No: fails in infinite-depth spaces, spaces with loops Modify to avoid repeated states along path. Complete in finite spaces

Time? $O(b^m)$: terrible if m is much larger than d but if solutions are dense, may be much faster than breadth-first

Space? $O(bm)$, i.e., linear space!

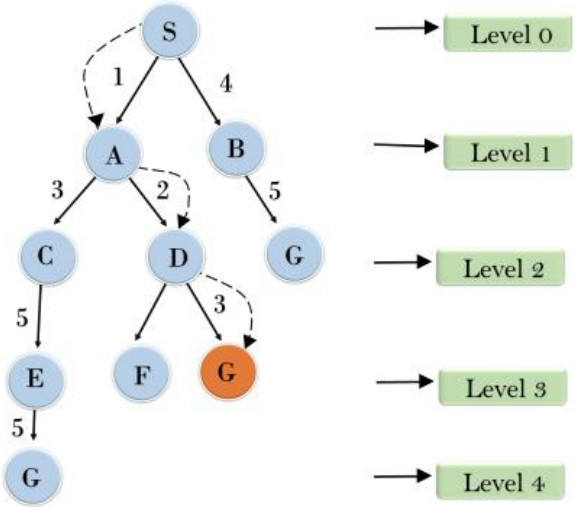
Optimal?? No

Depth-limited search



Depth-first search with depth limit *l*. Choosing the right l is the key.

Uniform-cost search



Expand least-cost unexpanded node. called **Dijkstra's algorithm** by the theoretical computer science community, and uniform-cost search Uniform-cost search by the AI community

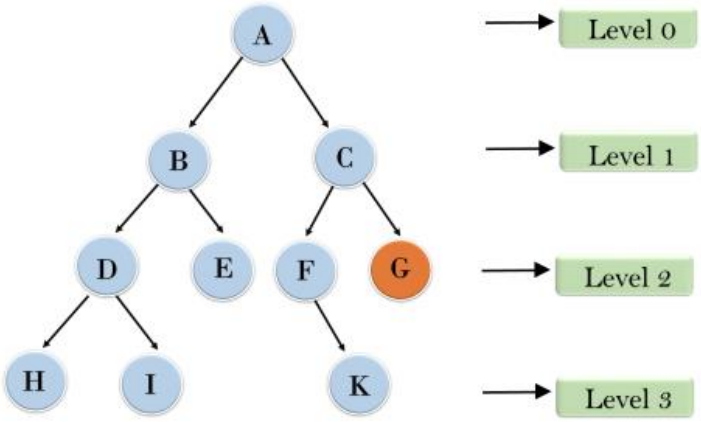
Complete? Yes, if step cost ≥ c(minstepcost,>0)

Time? number of nodes with $g \leq$ cost of optimal solution, $O(b^{\text{fC*/cl}})$ where C^* is the cost of the optimal solution

Space? number of nodes with $g \leq$ cost of optimal solution, $O(b^{\text{fC*/cl}})$

Optimal? Yes—nodes expanded in increasing order of $g(n)$

Iterative deepening search



1st Iteration → A (level 0)
2nd Iteration → A, B, C (level 1)
3rd Iteration → A, B, D, E, C, F, G (level 2)
4th Iteration →A, B, D, H, I, E, C, F, K, G (level 3)

Complete? Yes
Time? $O(b^d)$
Space? $O(bd)$
Optimal? Yes, if step cost = 1

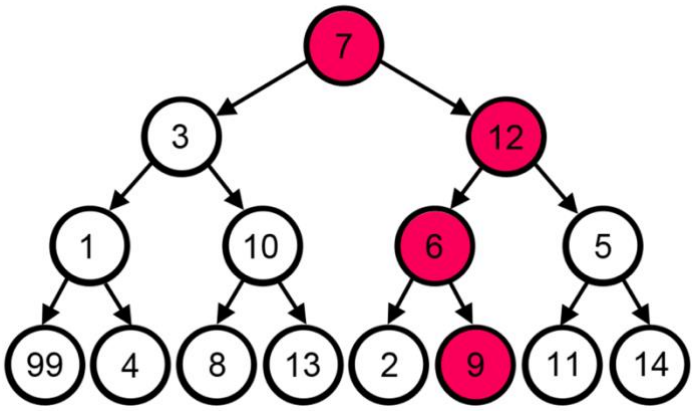
Comparison

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Informed (Heuristic) Search Strategies

Use an evaluation function for each node to estimate of "desirability" → Expand most desirable unexpanded node

Greedy search



Complete? No–can get stuck in loops, complete in finite space with repeated-state checking

Time? $O(b^m)$, but a good heuristic can give dramatic improvement

Space? $O(b^m)$—keeps all nodes in memory

Optimal? No

A* search

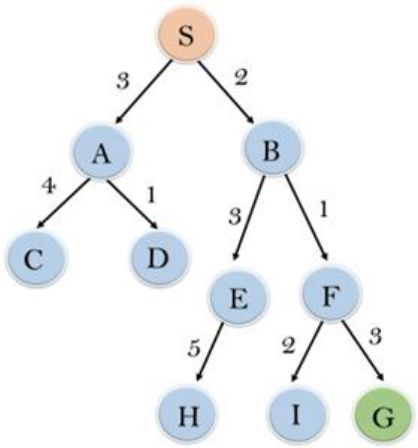
Evaluation function $f(n) = g(n) + h(n)$

$g(n)$ = cost so far to reach n

$h(n)$ = estimated cost to goal from n

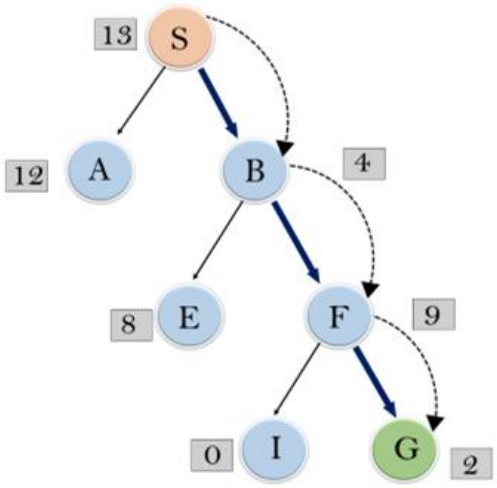
$f(n)$ = estimated total cost of path through n to goal

A* expands in the best direction (f increasing/decreasing).



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Solution



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

Time? Exponential in [relative error in h × length of soln.]

Space? Keeps all nodes in memory

Optimal? Yes – cannot expand f_{i+1} until f_i is finished

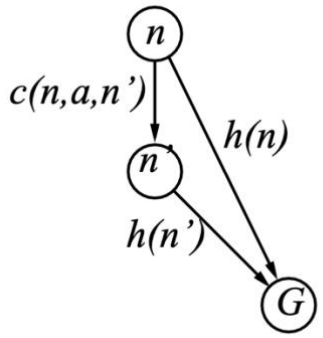
A*expands all nodes with $f(n) < C^*$

A* expands some nodes with $f(n) = C^*$

A* expands no nodes with $f(n) > C^*$

Consistency of heuristics

A heuristic is consistent if $h(n) \leq c(n, a, n') + h(n')$



If $h_2(n) \geq h_1(n)$ for all n (both admissible), then h_2 dominates h_1 and is better for search.

5 – Search in Complex Environment

Local Search and Optimization Problems

Local search algorithms operate by searching from a start state to neighboring states, without keeping track of the paths, nor the set of states that have been reached. They are not systematic—they might never explore a portion of the search space where a solution actually resides.

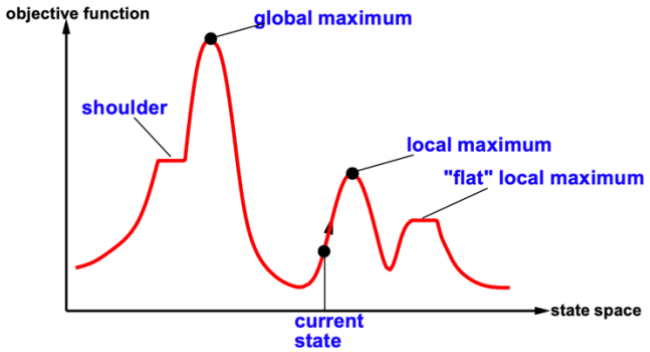
However, they have two key advantages:

- they use very little memory
- they can often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable

Hill-climbing (or gradient ascent/descent)

"Like climbing Everest in thick fog with amnesia"

It keeps track of one current state and on each iteration moves to the neighboring state with highest value—that is, it heads in the direction that provides the steepest ascent (better solution).



Simulated annealing: escape local maxima by allowing some "bad" moves but gradually decrease their size and frequency.

It seems reasonable to try to combine hill climbing with a random walk in a way that yields both efficiency and completeness.

Local beam search

Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations. Keep k states instead of 1; choose top k of all their successors.
Problem: quite often, all k states end up on same local hill. Choose k successors randomly, biased towards good ones

Genetic algorithms

Evolutionary algorithms can be seen as variants of stochastic beam search that are explicitly motivated by the metaphor of natural selection in biology: there is a population of individuals (states), in which the fittest (highest value) individuals produce offspring (successor states) that populate the next generation, a process called *recombination*.
Variations occur in reproduction and will be preserved in successive generations *approximately in proportion* to their effect on reproductive fitness.

Genetic algorithms = stochastic local beam search + generate successors from pairs of state

Continuous state spaces

Discretization methods turn continuous space into discrete space (map grid), e.g., empirical gradient considers $\pm\delta$ change in each coordinate

Search with Nondeterministic Actions

If the agent doesn't know the state its transitioned to after the action, the environment is nondeterministic. Rather, it will know the possible states it will be in, which is called "belief state"

Many problems in the real, physical world are contingency problems, **because exact prediction of the future is impossible**.

Search in Partially Observable Environments

Where the agent's percepts are not enough to pin down the exact state.

Searching with no observation: agent's percepts provide no information at all → sensorless problem (or a conformant problem).

Searching in partially observable environments: requires a function that monitors or estimates the environment to maintain the belief state.

6 - Constraint satisfaction problems

Defining Constraint Satisfaction Problems (CSP)

Each state: a set of variables, each of which has a value.
A problem is solved when each variable has a value that satisfies all the constraints on the variable. A problem described this way is called a constraint satisfaction problem, or CSP.

Constraint satisfaction problem consists of three components, X, D, and C:

- X is a set of variables, $\{X_1 \ldots\ldots X_n\}$.
- D is a set of domains, $\{D_1, \ldots\ldots, D_n\}$, one for each variable
- C is a set of constraints that specify allowable combination of values (*domain knowledge at play here!*)

CSPs deal with assignments of values to variables.

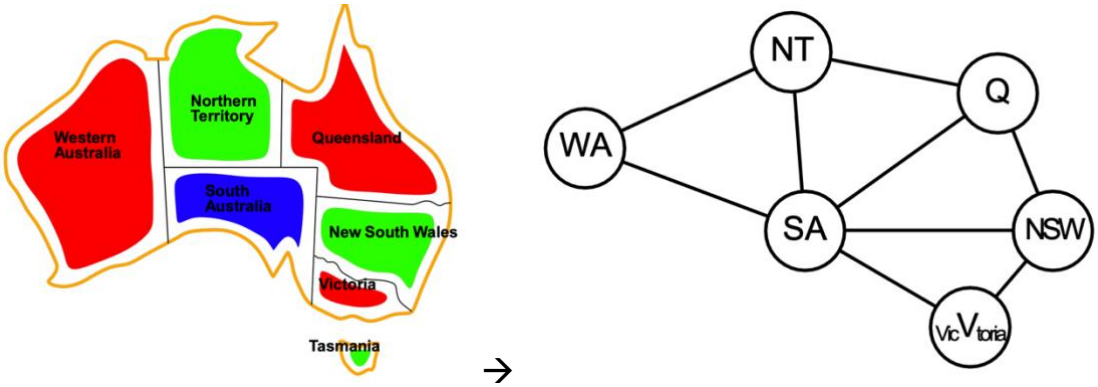
- A complete assignment is one in which every variable is assigned a value, and a solution to a CSP is a consistent, complete assignment.
- A partial assignment is one that leaves some variables unassigned.
- A partial solution is a partial assignment that is consistent

State is defined by variables X_i with values from domain D_i goal test is a set of constraints specifying allowable combinations of values for subsets of variables. Many real-world problems involve real-valued variables.

Constraint graph

Binary CSP: each constraint relates at most two variables.

Constraint graph: nodes are variables, arcs show constraints.



Varieties of constraints

Unary: constraints involve a single variable (*SA /= green)*

Binary: constraints involve pairs of variables (*SA /= WA)*

Higher-order: constraints involve 3 or more variables (cryptarithmetic)

Preferences: soft constraints (*red* is better than *green)* often representable by a cost for each variable assignment → constrained optimization problems

Standard search formulation (incremental)

Initial state: the empty assignment, { }

Successor function: assign a value to an unassigned variable that does not conflict with current assignment → fail if no legal assignments (not fixable!)

Goal test: the current assignment is complete

1) This is the same for all CSPs!

2) Every solution appears at depth n with n variables → use depth-first search

3) Path is irrelevant, so can also use complete-state formulation

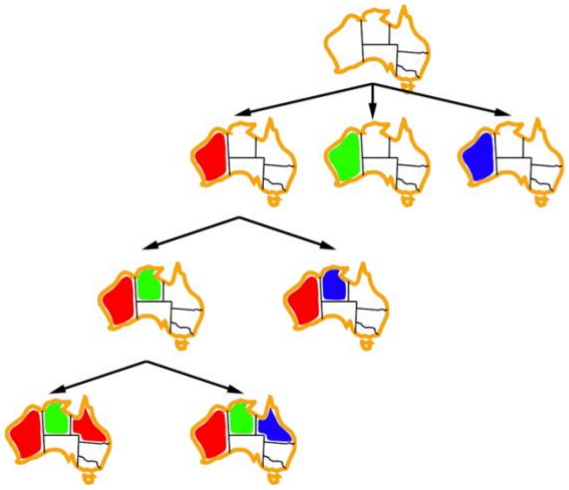
4) Problem: $b = (n - 1)d$ at depth n, hence $n!d^n$ leaves!

Backtracking search

Variable assignments are commutative

Only need to consider assignments to a single variable at each node

Depth-first search for CSPs with single-variable assignmentsis called is backtracking search



Improving backtracking efficiency

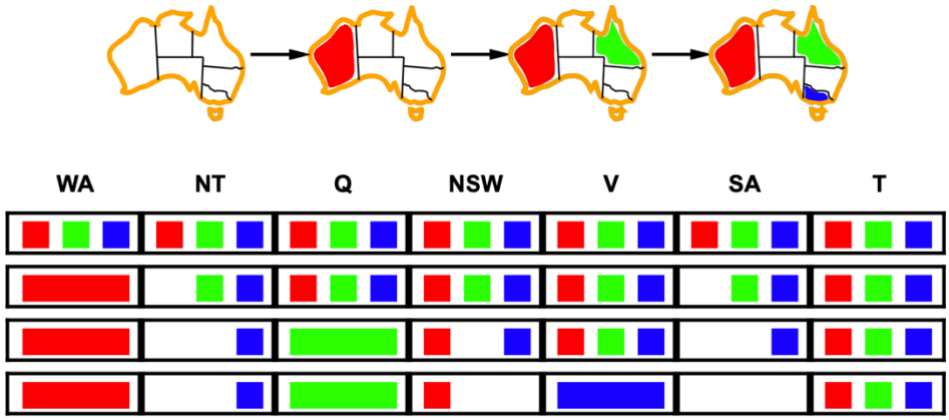
General-purpose methods can give huge gains in speed

- Minimum remaining values (MRV): choose the variable with the fewest legal values
- Degree heuristic: choose the variable with the most constraints on remaining variables (break tie MRV)
- Given a variable, choose the least constraining value:the one that rules out the fewest values in the remaining variables

Variable selection is fail-first, but value selection is fail-last.

Forward checking

Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values.



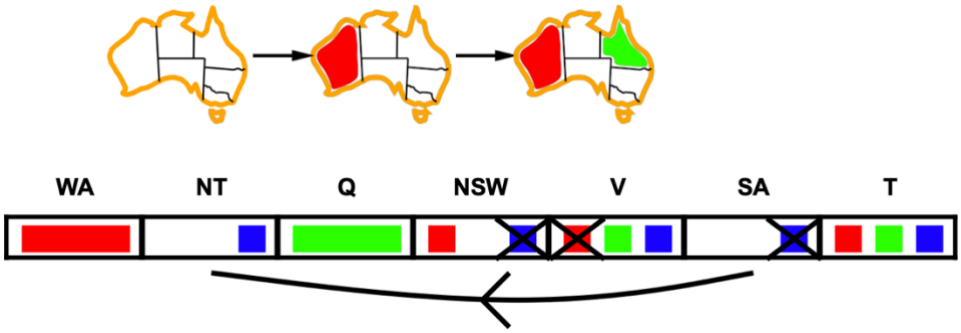
Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures.

Constraint propagation

Repeatedly enforces constraints locally

Arc consistency

Simplest form of propagation makes each arc consistent $X \rightarrow Y$ is consistent if and only if for every value x of X there is some allowed y. If X loses a value, neighbors of X need to be rechecked



Local Search for CSPs

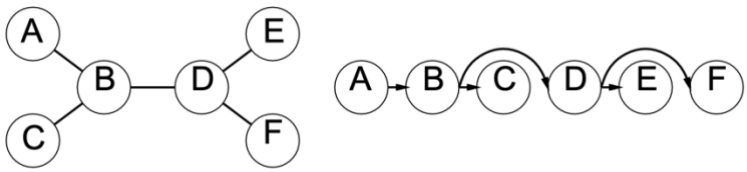
Local search algorithms can be very effective in solving many CSPs.

Local search algorithms use a complete-state formulation where each state assigns a value to every variable, and the search changes the value of one variable at a time.

Min-conflicts heuristic: value that results in the minimum number of conflicts with other variables that brings us closer to a solution. • Usually has a series of plateaus
Plateau search: allowing sideways moves to another state with the same score. Can help local search find its way off the plateau.
Constraint weighting aims to concentrate the search on the important constraints. Each constraint is given a numeric weight, initially all 1. Weights adjusted by incrementing when it is violated by the current assignmen.

Problem structure

Constraint graph is a tree when any two variables are connected by only one path. Tree-structured CSPs can be solved in linear time if there's no loops.



The CSP representation allows analysis of problem structure.

Iterative algorithms for CSPs

Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned.

To apply to CSPs: allow states with unsatisfied constraints operators reassign variable values.
Variable selection: randomly select any conflicted variable.
Value selection by min-conflicts heuristic: choose value that violates the fewest constraints; i.e., hillclimb with h(n) = total number of violated constraints

7 – Games

Games Theory

In this lecture we cover competitive environments, in which two or more agents have conflicting goals, giving rise to adversarial search problems.

For simplicity we consider:

- Two players: max-min, taking turns, fully observable
- Moves: action
- Position: state
- Zero sum: good for one player, bad for another, no win-win outcome.

Games vs. search problems

"Unpredictable" opponent → solution is a strategy specifying a move for every possible opponent reply
Time limits → unlikely to find goal, must design approximate, plan of attack

Types of games

- Deterministic/chance (deterministic games are dominated by computers).
- Perfect/imperfect information.

Minimax

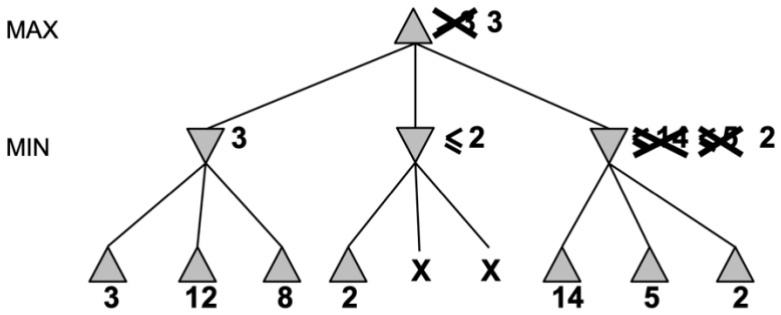
Perfect play for deterministic, perfect-information games.

Choose moves to position with highest minimax value = best achievable payoff against best play = minimizing the possible loss for a worst case (maximum loss) scenario.
Min tries to minimize the utility function while max tries to maximize

Complete? Yes, if tree is finite (chess has specific rules for this)
Optimal? Yes, against an optimal opponent. Otherwise??
Time complexity? O(bm)
Space complexity? O(bm) (depth-first exploration)

α–β pruning

We don't need to explore every path



α is the best value (to max) found so far off the current path If V is worse than α, max will avoid it → prune that branch. Define β similarly for min
Pruning does not affect final result but improves times complexity.

Monte Carlo Tree Search

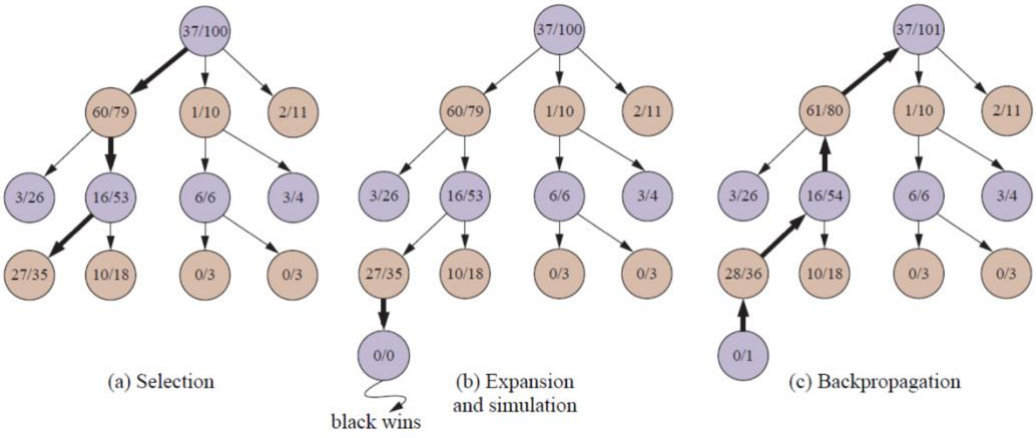
The basic Monte Carlo Tree Search (MCTS) strategy does not use a heuristic evaluation function. Value of a state is estimated as the average utility over number of simulations.

- Playout: simulation that chooses moves until terminal position reached.
- Selection: Start of root, choose move repeated moving down tree
- Expansion: Search tree grows by generating a new child of selected node
- Simulation: playout from generated child node
- Back-propagation: use the result of the simulation to update all the search tree nodes going up to the root

UCT: Effective selection policy is called “upper confidence bounds applied to trees”.
UCT ranks each possible move based on an upper confidence bound formula UCT called UCB1.

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$$

, where U(n) is the total utility of all playouts that went through node n, N(n) is the number of playouts through node n, C a constant, and PARENT(n) is the parent node of n in the tree.
U(n)/N(n) is the exploitation term, and the rest is the exploration term.



Resource limits

Use cutoff-test instead of terminal-test e.g., depth limit (perhaps add quiescence search).
To do that we have to use an evaluation function that estimates desirability of position instead of the utility.

Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling (random). Expectiminimax gives perfect play, just like Minimax, except we must also handle chance nodes

In practice: as depth increases, probability of reaching a given node shrinks → value of lookahead is diminished.

Games of imperfect information

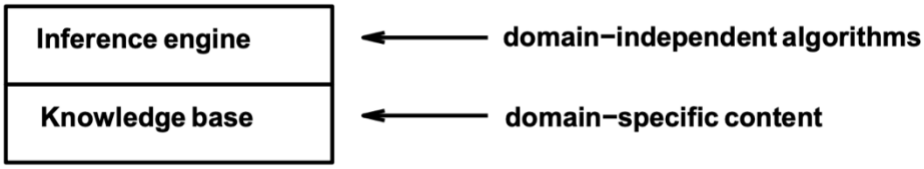
Games such as card games, where opponent's initial cards are unknown. Compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals.

Limitations of Game Search Algorithms

- Alpha–beta search vulnerable to errors in the heuristic function.
- Waste of computational time for deciding best move where it is obvious (meta-reasoning).
- Reasoning done on individual moves. Humans reason on abstract levels.
- Possibility to incorporate Machine Learning into game search process.

8 – Logical Agents

Knowledge bases



Knowledge bases is the set of sentences in a formal language.
Declarative approach to building an agent (or other system): tell it what it needs to know, then it can ask itself what to do, answers should follow from the Knowledge Base (KB). *KB* = rules + observations.

Agents can be viewed at the knowledge level: what they know, regardless of how implemented.
Or at the implementation level: data structures in KB and algorithms that manipulate them.

Simple KB Agent

The agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world. Deduce hidden properties of the world. Deduce appropriate actions

Logic in general

Logics are formal languages for representing information such that conclusions can be drawn.

Syntax defines the sentences in the language.

Semantics define the "*meaning*" of sentences; i.e., define truth of a sentence in a world

Entailment means that one thing follows from another: $KB \models \alpha$. KB entails sentence α if and only if α is true in all worlds where KB is true.
Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

Models are formally structured worlds with respect to which truth can be evaluated. We say m is a model of a sentence α if α is true in m.
$M(\alpha)$ is the set of all models of α.
Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

Inference means $KB \vdash \alpha$ a sentence α can be derived from KB by procedure i

Consequencesof KB are a haystack; α is a needle.
Entailment = needle in haystack; *inference* = finding it

Propositional logic

Syntax

If S is a sentence, $\neg S$ is a sentence (negation)

If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)

If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)

If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Semantics

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Inference by enumeration: depth-first enumeration of all models is sound and complete.

Logical equivalence: two sentences are logically equivalent iff true in same models: $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$.

Validity and satisfiability: a sentence is valid if it is true in all models. Validity is connected to inference: $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid.

A sentence is satisfiable if it is true in some model, a sentence is unsatisfiable if it is true in no models. Satisfiability is connected to inference: $KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable.

Proof methods

Proof is a *"chain of conclusions that leads to the desired goal"*. The methods are divided into (roughly) two kinds:

1) Application of inference rules

- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
- Typically require translation of sentences into a normal form

2) Model checking

- truth table enumeration (always exponential in n)
- improved backtracking
- heuristic search in model space (sound but incomplete)

Resolution

Resolution by inference rule, resolution is sound and complete for propositional logic.

Forward and backward chaining

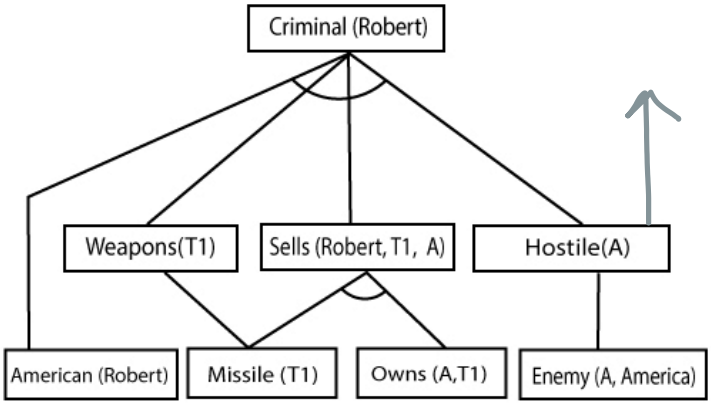
Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches.

KB = conjunction of Horn clauses

$$\frac{a_1, \ldots, a_n, \qquad a_1 \wedge \cdots \wedge a_n \Rightarrow \beta}{\beta}$$

Forward chaining

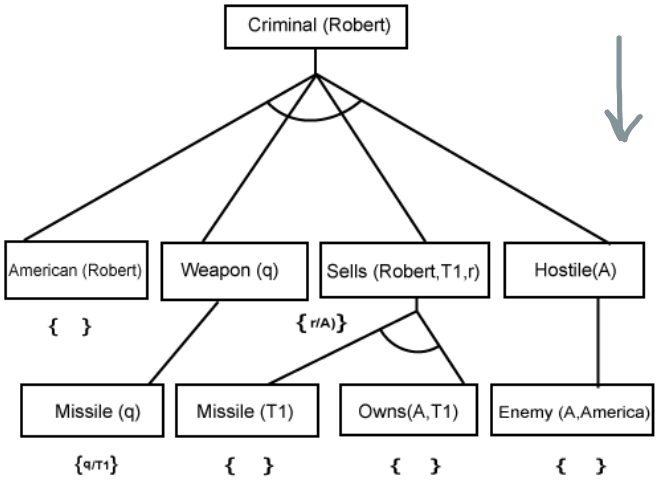
Fire any rule whose premises are satisfied in the *KB*, add its conclusion to the *KB*, until query is found.



Backward chaining

Work backwards from the query *q*: to prove *q* by BC, check if *q* is known already, or prove by BC all premises of some rule concluding *q*

Avoid loops and repeated works.



Forward vs. backward chaining

Are both linear-time, complete for Horn clauses

FC is data-driven, cf. automatic, unconscious processing, e.g., object recognition, routine decisions. May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving

09 – First Order Logic

Pros and cons of propositional logic

+ Propositional logic is declarative: pieces of syntax correspond to facts.
+ Propositional logic allows partial/disjunctive/negated information (unlike most data structures and databases).
+ Propositional logic is compositional: meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$.
+ Meaning in propositional logic is context-independent (unlike natural language, where meaning depends on context).

- Propositional logic has very limited expressive power (unlike natural language). E.g., cannot say "pits cause breezes in adjacent squares", except by writing one sentence for each square

First-order logic

Whereas propositional logic assumes world contains facts, first-order logic (like natural language) assumes the world contains: objects, functions, relations, etc.

Models for FOL

Sentences are true with respect to a model and an interpretation
Model contains≥ 1 objects(domain elements) and relations among them
An atomic sentence $predicate(term_1,...,term_n)$ is true iff the objects referred to by $term_1, ..., term_n$ are in the relation referred to by *predicate*.
There're a lot of models for FOL so computing entailment by enumerating FOL models is not easy!

Universal quantification

∀ {*variables*} *sentence*s

Typically, ⇒ is the main connective with ∀ (not ∧).

Existential quantification

∃ {*variables*} *sentence*s

∃*x P* is true in a model *m* iff *P* is true with *x* being some possible object in the model

Typically, ∧ is the main connective with ∃ (not ⇒)

Equality

$term_1 = term_2$ is true under a given interpretation

if and only if $term_1$ and $term_2$ refer to the same object

Interacting with FOL KBs

Given a sentence *S* and a substitution *σ*, *Sσ* denotes the result of plugging *σ* into *S*. *Ask*(*KB, S*) returns some/all *σ* such that *K B* |= *Sσ*

Deducing hidden properties

Diagnostic rule—infer cause from effect

Causal rule—infer effect from cause

Neither of these is complete

Keeping track of change

Facts hold in situations, rather than eternally. Situation calculus is one way to represent change in FOL: adds a situation argument to each non-eternal predicate. Situations are connected by the *Result* function *Result*(*a, s*) is the situation that results from doing *a* in *s*

Describing actions

"Effect" axiom: describe changes due to action
"Frame" axiom: describe non-changes due to action

Frame problem: find an elegant way to handle non-change

- representation—avoid frame axioms
- inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats (communication)
Ramification problem: real actions have many secondary consequences

Successor-state axioms solve the representational frame problem Each axiom is "about" a predicate (not an action per se):
P true afterwards ⇔
[an action made P true ∨ P true already and no action made P false]

Making plans

Represent plans as action sequences $[a_1, \ldots, a_n]$

PlanResult(*p, s*) is the result of executing *p* in *s*.
Then the query *Ask*(*KB*,∃*p Holding*(*Gold*,*PlanResult*(*p*,S_0))) has the solution {*p*/[*Forward*,*Grab*]}

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Knowledge Engineering in FOL

Knowledge engineering: the general process of knowledge-base construction.

Developing a KB in FOL requires a careful process of analyzing the domain, choosing a vocabulary, and encoding the axioms required to support the desired inferences

The steps used in the knowledge engineering process:

1. Identify the questions.
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the problem instance
6. Pose queries to the inference procedure and get answers
7. Debug and evaluate the knowledge base

10 – Inference First Order Logic

Instantiation

Universal instantiation (UI)

Every instantiation of a universally quantified sentence is entailed by it
$\forall v\, \alpha \,/\, Subst(\{v/g\},\alpha)$

Existential instantiation (EI)

For any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base:
$\exists v\, \alpha \,/\, Subst(\{v/k\},\alpha)$

UI can be applied several times to add new sentences; the new K B is logically equivalent to the old.
EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old,but is satisfiable iff the old K B was satisfiable

Reduction to propositional inference

Instantiate the universal KB sentence in all possible ways. The new KB propositionalized has only proposition symbols resulted.

Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences. We can get the inference immediately if we can find a substitution θ.
$Unify(\alpha,\beta) = \theta$ if $\alpha\theta = \beta\theta$

Generalized Modus Ponens (GMP)

$$\frac{p_1', \; p_2', \; \ldots, \; p_n', \; (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

where $p_i'\theta = p_i\theta$ for all i

It is sound and complete.

Forward chaining

Properties

Sound and complete for first-order definite clauses

Datalog = first-order definite clauses + no functions (e.g., crime KB) *k*.
FC terminates for Datalog in poly iterations: at most $p \cdot n$ literals. May not terminate in general if α is not entailed.
This is unavoidable: entailment with definite clauses is semidecidable

Efficiency

Simple observation: no need to match a rule on iteration *k* if a premise wasn't added on iteration $k - 1$ → match each rule whose premise contains a newly added literal.
Forward chaining is widely used in deductive databases

Backward chaining properties

Depth-first recursive proof search: space is linear in size of proof Incomplete due to infinite loops
→ fix by checking current goal against every goal on stack Inefficient due to repeated subgoals (both success and failure)
→ fix using caching of previous results (extra space!) Widely used (without improvements!) for logic programming

Logic programming

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Prolog systems

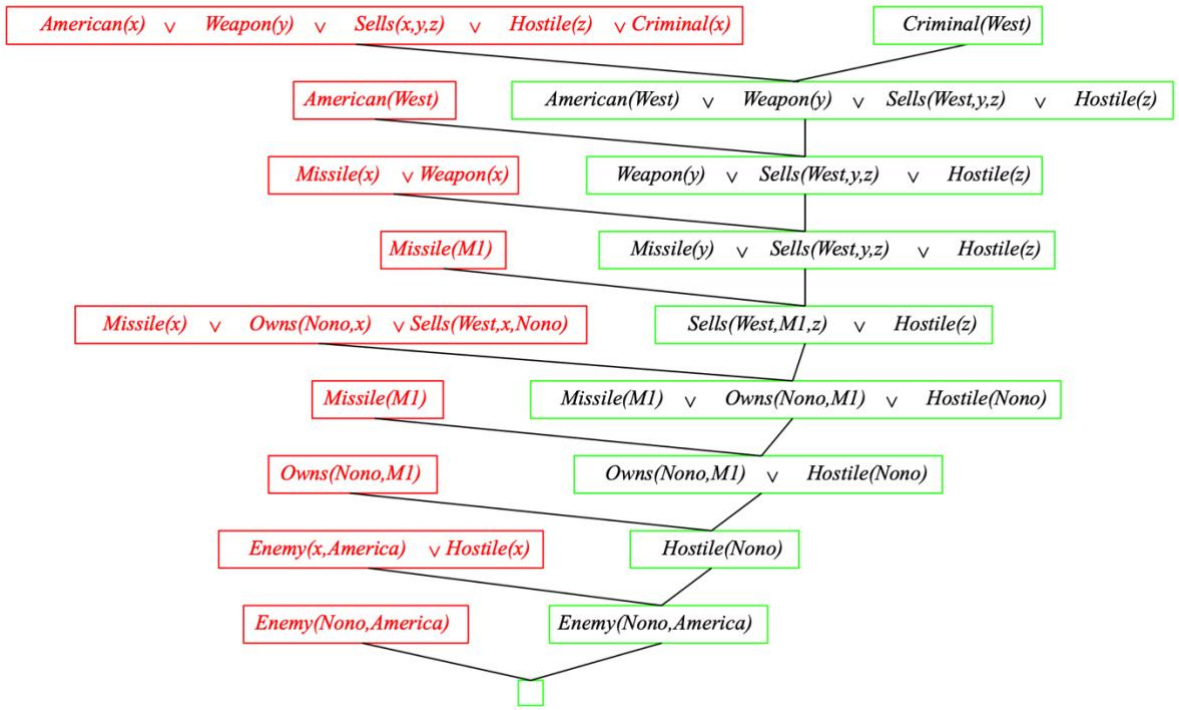
Program = set of clauses = head :- literal$_1$, ... literal$_n$.

Prolog, unlike first-order logic, uses a closed world with the unique names assumption and negation as failure.

Efficient unification by open coding.
Efficient retrieval of matching clauses by direct linking Depth-first, left-to-right backward chaining.
Built-in predicates for arithmetic etc., e.g., X i s Y*Z+3.
Closed-world assumption ("negation as failure") e.g.,given alive(X) :- not dead(X). alive(joe) succeeds if dead(joe) fails.

Resolution

The generalized resolution inference rule provides a complete proof system for first order logic, using knowledge bases in conjunctive normal form.



Gödel's Incompleteness Theorem

- There are true arithmetic sentences that cannot be proved
- For any set of true sentences of number theory, and in particular any set of basic axioms, there are other true sentences that cannot be proved from those axioms.
- We can never prove all the theorems of mathematics within any given system of axioms.

Resolution strategies

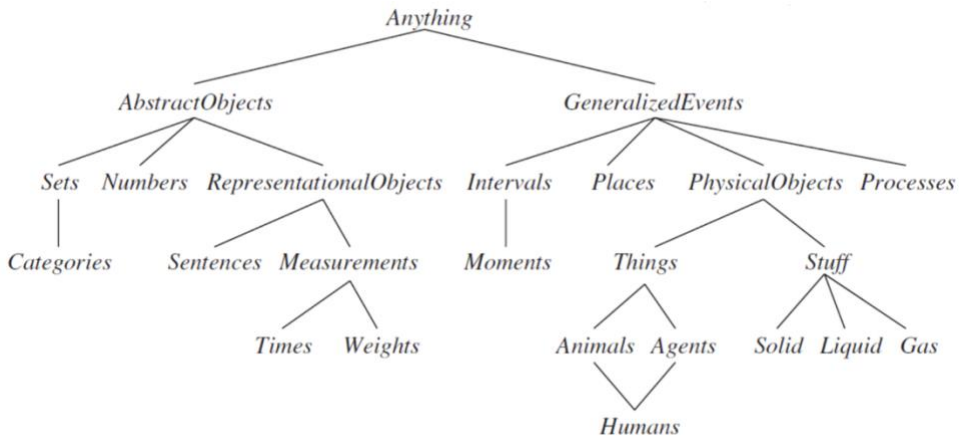
Unit preference: prefers to do resolutions where one of the sentences is a single literal (unit clause).
Set of support: every resolution step involve at least one element of a special set of clauses.
Input resolution: every resolution combines one of the KB input sentences with other sentences.
Subsumption: eliminates all sentences that are subsumed by KB sentences.
Learning: learning from experience (machine learning).

11 – Knowledge

Ontological Engineering

It is the general, flexible and hierarchical representations for complex domains.

Upper ontology: The general framework of concepts



Concentrate on general concepts—such as Events, Time, Physical Objects, and Beliefs— that occur in many different domains.

Categories and Objects

The organization of objects into categories is a vital part of knowledge representation. Categories for FOL can be represented by predicates and objects.
Physical composition: one object can be part of another is a familiar one.
Measurements: values that we assign for properties of objects.
First-order logic makes it easy to state facts about the categories, either by relating objects to categories or by quantifying over their members.

Stuff: a significant portion of reality that seems to defy any obvious individuation—division into distinct objects

This is major distinction between stuff and things. If we cut an animal in half, we do not get two animals (unfortunately).

Properties of objects

- Intrinsic: they belong to the very substance of the object, rather than to the object as a whole (density, boiling point, flavor).
- Extrinsic: not retained under subdivision (weight, length, shape).
- Substance: a category of objects that includes in its definition only intrinsic properties (mass noun), e.g. animals, holes, and theorems.
- Count noun: class that includes any extrinsic properties, e.g. butter, water, and energy.

Events calculus

In real world there is a huge range of actions or events to deal with.

Consider a continuous action, such as *filling a bathtub*. Successor-state axiom can say that the tub is empty before the action and full when the action is done, but it can't talk about what happens during the action. It also can't easily describe two actions happening at the same time— such as brushing one's teeth while waiting for the tub to fill.
To handle such cases we introduce an approach known as event calculus: events, fluents, and time points.

$T(f,t_1,t_2)$	Fluent f is true for all times between t_1 and t_2
$Happens(e,t_1,t_2)$	Event e starts at time t_1 and ends at t_2
$Initiates(e,f,t)$	Event e causes fluent f to become true at time t
$Terminates(e,f,t)$	Event e causes fluent f to cease to be true at time t
$Initiated(f,t_1,t_2)$	Fluent f become true at some point between t_1 and t_2
$Terminated(f,t_1,t_2)$	Fluent f cease to be true at some point between t_1 and t_2
$t_1 < t_2$	Time point t_1 occurs before time t_2

We can describe the effects of a flying event:

$$E = Flyings(a,here,there) \wedge Happens(E,t_1,t_2) \Rightarrow$$
$$Terminates(E,At(a,here),t_1) \wedge Initiates(E,At(a,there),t_2)$$

Mental Objects and Modal Logic

Mental objects are knowledge in someone's head (or KB).
Propositional attitudes are attitudes that an agent can have toward mental objects *(Believes*, *Knows*, *Wants*, and *Informs)*.

Modal logic addresses this, with special modal operators that take sentences (rather than terms) as arguments.
"*A knows P*" is represented with the notation $\mathbf{K}_A P$, where **K** is the modal operator for knowledge. It takes two arguments, an agent (written as the subscript) and a sentence. The syntax of modal logic is the same as first-order logic, except that sentences can also be formed with modal operators

Agents are able to draw conclusions. If an agent knows *P* and knows that *P* implies *Q*, then the agent knows *Q*.
Logical agents (but not all people) are able to introspect on their own knowledge. If they know something, then they know that they know it.

Semantics

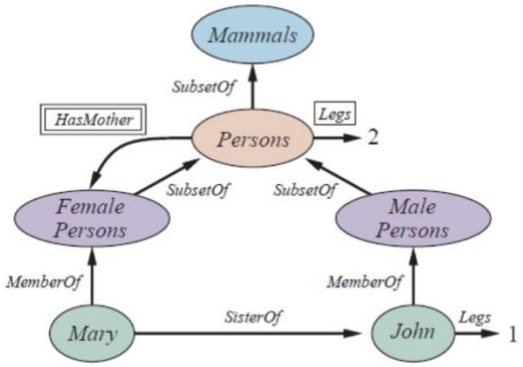
In modal logic we want to be able to consider both the possibility that Superman's secret identity is Clark and the possibility that it isn't.
Therefore, we will need a more complicated model, one that consists of a collection of possible worlds rather than just one true world.
The worlds are connected in a graph by accessibility relations, one relation for each modal operator.
A knowledge atom K_a (P) is true in world w if and only if P is true *in every world accessible from w*. The truth of more complex sentences is derived by recursive application of this rule and the normal rules of first-order logic.

We can define similar axioms for belief (often denoted by **B**) and other modalities. However, one problem with the modal logic approach is that it assumes logical omniscience on the part of agents.

Reasoning Systems for Categories

Semantic networks

Convenient to perform inheritance reasoning



Description logics

Notations that are designed to make it easier to describe definitions and properties of categories (not just talk about objects as in FOL).
Evolved from semantic networks in response to pressure to formalize what the networks mean while retaining the emphasis on taxonomic structure as an organizing principle. Principal inference tasks:

- Subsumption: checking if one category is a subset of another by comparing their definitions
- Classification: checking whether an object belongs to a category

The CLASSIC language is a typical description logic (e.g: bachelors are unmarried adult males $Bachelor = And(Unmarried, Adult, Male)$.
Notice that the description logic has an algebra of operations on predicates, which of course we can't do in first-order logic.

$Bachelor = And(Unmarried,Adult,Male)$ **vs** $Bachelor(x) \Leftrightarrow Unmarried(x) \wedge Adult(x) \wedge Male(x)$

Perhaps the most important aspect of description logics is their emphasis on tractability of inference. In standard first-order logic systems, predicting the

solution time is often impossible. The thrust in description logics, on the other hand, is to ensure that subsumption-testing can be solved in time polynomial in the size of the descriptions.

Reasoning with Default Information

Circumscription and default logic

Circumscription: can be seen as a more powerful and precise version of the *closed-world assumption*. It is an example of a *model preference logic:* a sentence is entailed (with default status) if it is true in all preferred models of the KB, as opposed to the requirement of truth in all models in classical logic. The conclusions are reached by default, in the absence of any reason to doubt it.

Default logic is a formalism in which default rules can be written to generate contingent nonmonotonic conclusions

Truth maintenance systems

Belief revision: inferred facts will turn out to be wrong and will have to be retracted in.

Truth maintenance systems, or TMSs, are designed to handle complications of any additional sentences that inferred from a wrong sentence.

Justification-based truth maintenance system (JTMS):

- Each sentence in the knowledge base is annotated with a justification consisting of the set of sentences from which it was inferred.
- *Justifications* make retraction efficient
- Assumes that sentences that are considered once will probably be considered again.

The CYC project

Cyc is a long-term AI project that aims to assemble a comprehensive ontology and knowledge base that spans the basic concepts and rules about how the world works.
CYC is a titanic effort trying to model common sense knowledge overcoming the limitations of knowledge engineering in Logic systems.

12 – Prolog

Prolog is a logic programming language that supports symbolic, non-numeric computation. It is very well suited for solving problems that involve objects and relations among those objects.

Facts and rules

Prolog mainly works with facts:

```
parent(tom,bob).
parent(bob,ann).
```

And rules:

```
grandparent(X,Y) :-
    parent(X,Z),
    parent(Z,Y).
```

Facts and rules constitute a KB.

Queries

KB can be queried loading them into a Prolog execution environment.:

```
?- grandparent(X,Y).
X = tom,
Y = ann .
```

To answer a query Prolog tries to satisfy all goals via backtracking.

Declarative vs procedural

Prolog has a:

- declarative meaning, which concerns what will be the output of a program

- procedural meaning, which concerns how such output is computed.

Knowledge representation

We can represent states as lists of stacks:

```
% [Stack1, Stack2, Stack3]
[[c,a,b], [], []] % Start state
```

A goal state is for instance one among:

```
[[a,b,c], [], []]
[[], [a,b,c], []]
[[], [], [a,b,c]]
```

We can check that we reached a goal state via the rule:

```
goal_state(State) :- member([a,b,c], State).
```

13 – Planning

Definition of Classical Planning

Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.

We have seen two main methods to plan ahead already:

- Problem-solving agent (with search algorithms)
- Propositional Logical agent (with logic inference)

Main limits:

- They both require ad hoc heuristics for each new domain: a heuristic evaluation function for search, and hand-written code for the logic agent.
- They both need to explicitly represent an exponentially large state space.

Planning Domain Definition Language (Ghallab et al., 1998). PDDL is a factored representation.

- Basic PDDL can handle classical planning domains, and extensions can handle non-classical domains that are continuous, partially observable, concurrent, and multi-agent.
- Allows us to express all 4T * n^2 (*T*=time steps, *n* locations) actions with a single *action schema*, and does not need domain-specific knowledge
- State: represented as a conjunction of *ground atomic fluents* (a single predicate with no variables)
- Uses database semantics: the *closed-world assumption* means that any fluents that are not mentioned are false + *unique names assumption*

An action schema represents a family of ground actions.The schema consists of the action name, a list of all the variables used in the schema, a precondition and an effect.

A set of action schemas serves as a definition of a planning domain. A specific problem within the domain is defined with the addition of an initial state and a goal state.

The initial state is a conjunction of ground fluents. The goal is just like a precondition: a conjunction of literals (positive or negative) that may contain variables

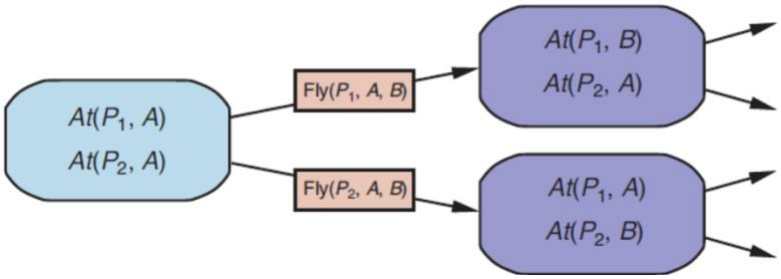
$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: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

Algorithms for Classical Planning

Forward state-space search for planning

- Start at initial state
- To determine the applicable actions we unify the current state against the preconditions of each action schema
- For each unification that successfully results in a substitution, we apply the substitution to the action schema to yield a ground action with no variables.

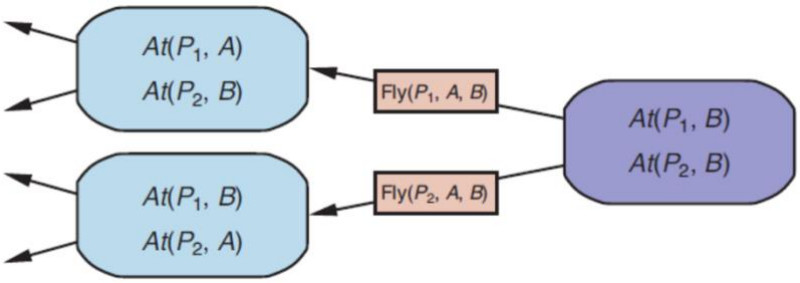
Strong domain-independent heuristics with PDDL can be derived automatically; that is what makes forward search feasible



Backward search for planning

- Start at the goal and apply the actions backward until we find a sequence of steps that reaches the initial state
- Consider relevant actions at each step.
- Reduces branching factor

A relevant action is one with an effect that unifies with one of the goal literals, but with no effect that negates any part of the goal.



Other classical planning approaches

An approach called Graph plan uses a specialized data structure, a planning graph

Situation calculus is a method of describing planning problems in first-order logic → no big impact in practical applications, perhaps since FOL provers are not as well developed (as propositional satisfiability programs).

An alternative called Partial-Order Planning (POP) represents a plan as a graph rather than a linear sequence.

Heuristics for Planning

Ignore preconditions heuristic: drops all preconditions from actions → Every action becomes applicable.
Ignore-delete-lists heuristic: removing the *delete lists* from all actions (i.e., removing all negative literals from effects).

Domain-independent pruning

- symmetry reduction: prune out consideration all symmetric branches of the search tree except for one
- forward pruning: might prune away an optimal solution
- relaxed plan: solution to a relaxed problem
- preferred action: step of the relaxed plan, or it achieves some precondition of the relaxed plan

State abstraction in planning

- state abstraction: a many-to-one mapping from states in the ground representation of the problem to the abstract representation
- relaxations that decrease the number of states
- decomposition: dividing a problem into parts, solving each part independently, and then combining the parts
- Subgoal independence assumption: the cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving each subgoal independently
- The subgoal independence assumption can be optimistic or pessimistic

Hierarchical Planning

Higher levels of abstraction with hierarchical decomposition to manage complexity. Hierarchical structure reduces computational task to a small number of activities at the next lower level.
Computational overhead to find correct way to arrange those activities for the current problem is small.

High Level Actions (HLA):

- Assume full observability & determinism & primitive actions standard precondition-effect schemas
- HLA offers one or more possible *refinements* into a sequence of actions.
- Implementation of HLA: An HLA refinement that contains only primitive actions.

High level Plan (HLP)

- is the concatenation of implementations of each HLA in the sequence.
- a high-level plan achieves the goal from a given state if *at least one of its implementations* achieves the goal from that state

Planning and Acting in Nondeterministic Domains

Sensorless planning (no observations)

Open-world assumption in which states contain both positive and negative fluents, and if a fluent does not appear, its value is unknown.
Given belief state b, the agent can consider any action whose preconditions are satisfied by b.

Conditional effect: an actions effect is dependant on a state (eg: robot's location)
"when condition: effect," where condition is a logical formula to be compared against the current state, and effect is a formula describing the resulting state.

In general, conditional effects can induce arbitrary dependencies among the fluents in a belief state, leading to belief states of exponential size in the worst case. All conditional effects whose conditions are satisfied have their effects applied to generate the resulting belief state; if none are satisfied, then the resulting state is unchanged.

Contingent planning (with observations)

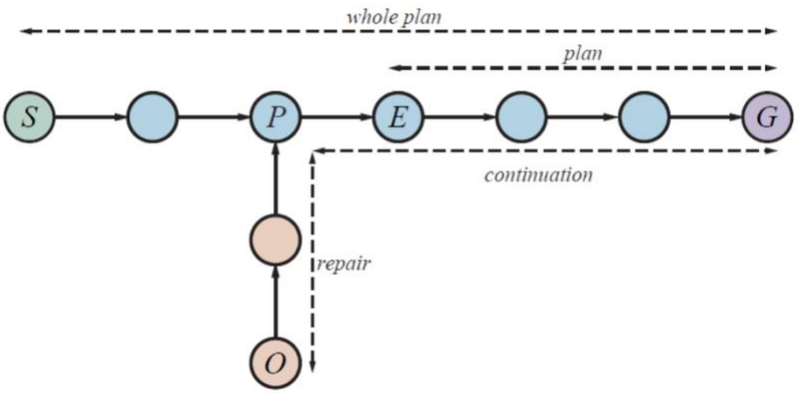
The generation of plans with conditional branching based on percepts— is appropriate for environments with partial observability, nondeterminism, or both.When executing this plan, a contingent-planning agent can maintain its belief state as a logical formula. Evaluate each branch condition by determining if the belief state entails the condition formula or its negation.

Online planning

- The need for a new plan: execution monitoring
- When there are too many contingencies to prepare for
- When a contingency is not prepared, replanning is required
- Replanning is needed if the agent's model of the world is incorrect (missing precondition, effect or fluent

For online planning an agent monitor based on three approaches:

- Action monitoring: before executing an action, the agent verifies that all the preconditions still hold.
- Plan monitoring: before executing an action, the agent verifies that the remaining plan will still succeed.
- Goal monitoring: before executing an action, the agent checks to see if there is a better set of goals it could be trying to achieve.



Time, Schedules, and Resources

Classical planning talks about what to do, in what order, but does not talk about time: how long an action takes and when it occurs.
Many actions consume resources, such as money, gas, or raw materials → *planning + scheduling*

Representing temporal and resource constraints

Actions can have a duration and constraints.
Constraints: type of resource, number of resources, resource is consumable or reusable
Aggregation: representation of resources as numerical quantities

The approach we take is "plan first, schedule later": divide the overall problem into:

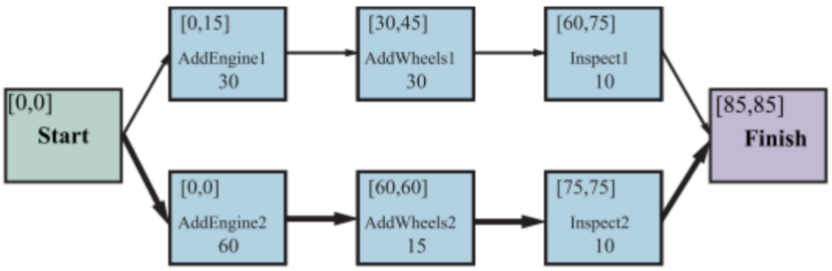
1. Planning phase: in which actions are selected, with some ordering constraints, to meet the goals of the problem;
2. Scheduling phase: in which temporal information is added to the plan to ensure that it meets resource and deadline constraints.

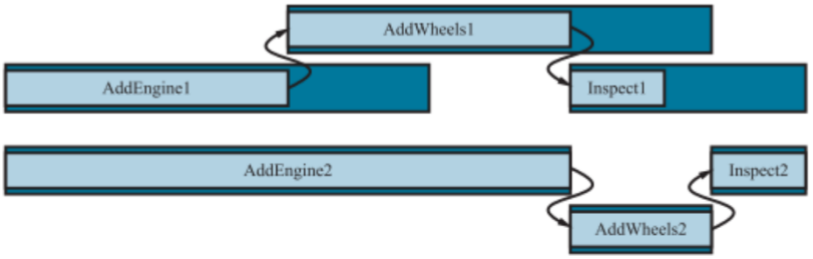
Solving scheduling problems

We need to model it as a directed graph (multiple actions in parallel). Uses critical path method (CPM) to determine the possible start and end times of each action

A path through a graph representing a partial-order plan is a linearly ordered sequence of actions beginning with “Start” and ending with “Finish”.

The critical path is that path whose total duration is longest; the path is “critical” because it determines the duration of the entire plan. Actions that are off the critical path have a window of time in which they can be executed. This window of time is known as Slack. The window has a earliest possible start time *ES* and latest possible start time *LS*. Together the *ES* and *LS* times for all the actions constitute a schedule for the problem.





Analysis of Planning Approaches

Planning combines the two major areas of AI we have covered so far: search and logic. Planning helps in controlling combinatorial explosion through the identification of independent subproblems.

14 – Uncertainty

Acting Under Uncertainty

An agent strives to choose the right thing to do—the rational decision—depends on both the relative importance of various goals and the likelihood that they will be achieved.
Large domains such as *medical diagnosis* fail to three main reasons:

- laziness: it is too much work to list the complete set of antecedents or consequents needed to ensure an exceptionless rule
- theoretical ignorance: medical science has no complete theory for the domain
- practical ignorance: even if we know all the rules, we might be uncertain about a particular patient because not all the necessary tests have been or can be run.

Probability theory: to deal with degrees of belief of relevant sentences.
Summarizes the uncertainty that comes from our laziness and ignorance.

Uncertainty and rational decisions

AI agents require preference among different possible outcomes of various plans.

- utility theory: the quality of the outcome being useful, every state has a degree of usefulness/utility, higher utility is preferred.
- decision theory: preferences (utility theory) combined with probabilities agent is rational if and only if it chooses the action that yields the highest expected utility, averaged over all the possible outcomes of the action. Principle of maximum expected utility (MEU)

Basic Probability Notation

For our agent to represent and use probabilistic information, we need a formal language.

The basic axioms:

$$0 \leq P(\omega) \leq 1 \text{ for every } \omega \text{ and } \sum_{\omega \in \Omega} P(\omega) = 1 .$$

Unconditional or prior probability: degrees of belief in propositions in the absence of any other information.

Conditional or posterior probability: given evidence that has happened, degree of belief of new event

Probability of a given b:

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

Factored representation: possible world is represented by a set of variable/value pairs.

$P(Weather = sun) = 0.6$
$P(Weather = rain) = 0.1$
$P(Weather = cloud) = 0.29$
$P(Weather = snow) = 0.01$,

→ probability distribution: $\mathbf{P}(Weather) = \langle 0.6, 0.1, 0.29, 0.01 \rangle$,

Inference Using Full Joint Distribution

Start with the joint distribution:

	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬cavity	**.016**	**.064**	**.144**	**.576**

Can also compute conditional probabilities:

$$P(\neg cavity|toothache) = \frac{P(\neg cavity \wedge toothache)}{P(toothache)} = \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

Let X be all the variables. Typically, we want the posterior joint distribution of the query variables Y given specific values e for the evidence variables E.
Let the hidden variables be H = X –Y –E
Then the required summation of joint entries is done by summing out the hidden variables.

Independence

$P(a \mid b) = P(a)$ or $P(b \mid a) = P(b)$ or $P(a \wedge b) = P(a)P(b)$

Bayes' Rule and Its Use

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

Often, we perceive as evidence the effect of some unknown cause and we would like to determine that cause. In that case, Bayes' rule becomes:

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

The conditional probability $P(effect|cause)$ quantifies the relationship in the causal direction, whereas $P(cause|effect)$ describes the diagnostic direction.

Naïve Bayes Models

The full joint distribution can be written as:

$\boldsymbol{P}(Cause, Effect_1, \ldots, Effect_n) = \boldsymbol{P}(Cause) \prod_i \boldsymbol{P}(Effect_i | Cause)$

Such a probability distribution is called a naive Bayes model—"naive" because it is often used (as a simplifying assumption) in cases where the "effect" variables are not strictly independent given the cause variable (*still useful in practice*).

Call the observed effects **E**=**e**, while the remaining effect variables **Y** are unobserved

$$\begin{aligned}\mathbf{P}(Cause|\mathbf{e}) &= \alpha\sum_{\mathbf{y}}\mathbf{P}(Cause)\mathbf{P}(\mathbf{y}|Cause)\left(\prod_j \mathbf{P}(e_j|Cause)\right)\\ &= \alpha\mathbf{P}(Cause)\left(\prod_j \mathbf{P}(e_j|Cause)\right)\sum_{\mathbf{y}}\mathbf{P}(\mathbf{y}|Cause)\\ &= \alpha\mathbf{P}(Cause)\prod_j \mathbf{P}(e_j|Cause)\end{aligned}$$

Conditional Independence

When observation is irrelevant or redundant when evaluating the certainty of a hypothesis. If A is the hypothesis, and B and C are observations, conditional independence can be stated as an equality:
P(A|B, C)=P(A|C).
Conditional independence brought about by direct causal relationships in the domain allows the full joint distribution to be factored into smaller, conditional distributions.

15 – Probabilistic Reasoning

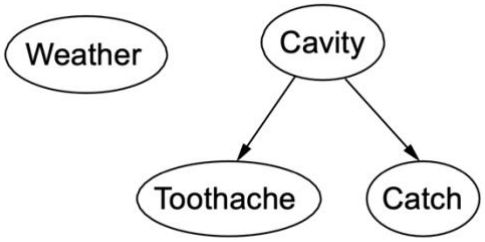
Representing Knowledge in an Uncertain Domain

Bayesian networks: represents dependencies among variables.
A simple, directed graph in which each node is annotated with quantitative probability information.

Syntax:

- a set of nodes, one per variable
- a directed acyclic graph(link ≈ "directly influences")
- a conditional distribution for each node given its parents:
  $\mathrm{P}(X_i|Parents(X_i))$

In the simplest case, conditional distribution represented as a conditional probability table (CPT) giving the distributionoverX_i for each combination of parent values.



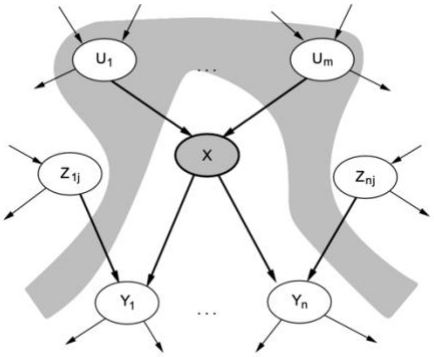
Compactness

A CPT for Boolean X_i with k Boolean parents has 2^k rows for the combinations of parent values.Each row requires one number p for X_i =*true* (the number for X_i =*false* is just 1– p). If each variable has no more than k parents,the complete network requires $O(n \cdot 2^k)$ numbers

The Semantics of Bayesian Network

Global semantics defines the full joint distribution as the product of the local conditional distributions: $P(x_1,\ldots,x_n) = \Pi_{i=1} P(x_i|parents(X_i))$

Local semantics: each node is conditionally independent of its nondescendants given its parents



Theorem: Local semantics ⇔ global semantics

Constructing Bayesian networks

Need a method such that a series of locally testable assertions of conditional independence guarantees the required global semantics

1. Choose an ordering of variables $X_1,\ldots,X_n$

2. For $i = 1$ *to* n

add X_i *to the network selectparentsfrom*$X_1,\ldots,X_{i-1}$ *such that*

$P(X_i|Parents(X_i)) = P(X_i|X_1,\ldots,X_{i-1})$

Compact conditional distributions

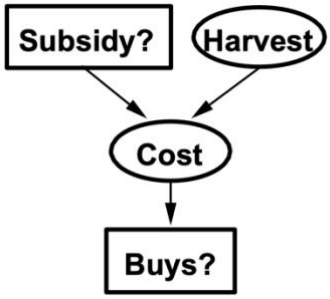
CPT grows exponentially with number of parents. CPT becomes infinite with continuous-valued parent or child.

Solution: canonical distributions that are defined compactly.

Deterministic nodes are the simplest case: $X = f(Parents(X))$ for some function f

Noisy-OR: each parent is capable to execute its influence on the node independently of other parents, whereby the individual effects are then summarized with the Boolean function OR

Hybrid (discrete+continuous) networks



- Continuous variable, discrete+continuous parents (e.g., *Cost*)
- Discrete variable, continuous parents (e.g., *Buys?*)

Option 1: discretization—possibly large errors, large CPTs

Option 2: finitely parameterized canonical families.

Exact Inference in Bayesian Networks

Simple queries: compute posterior marginal $P(X_i|E = e)$.

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

Optimal decisions: decision networks include utility information; probabilistic inference required for P (*outcome*|*action*, *evidence*).

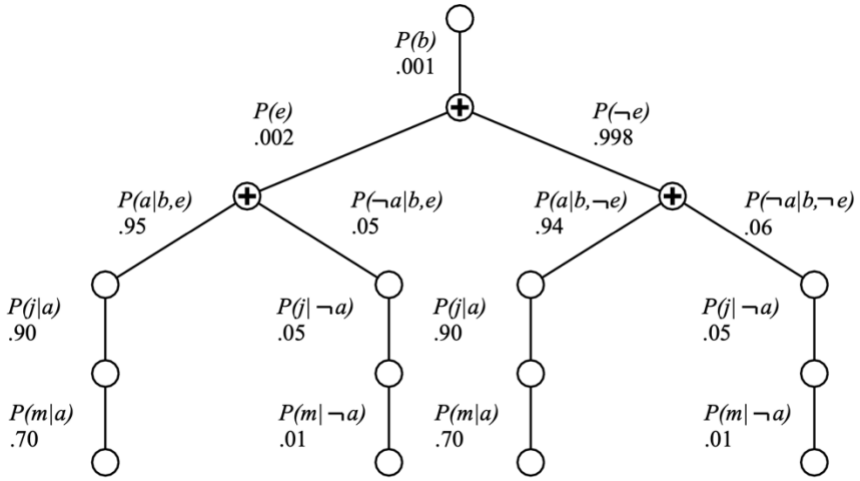
Value of information: which evidence to seek next?

Sensitivity analysis: which probability values are most critical?

Explanation: why do I need a new starter motor?

Inference by enumeration

Slightly intelligent way to sum out variables from the joint without actually constructing its explicit representation.



Enumeration is inefficient: repeated computation

Inference by variable elimination

Variable elimination: carry out summations right-to-left, storing intermediate results (factors) to avoid recomputation

Operation

Summing out a variable from a product of factors: move any constant factors outside the summation add up submatrices in pointwise product of remaining factors:

$$\Sigma_x f_1 \times \cdots \times f_k = f_1 \times \cdots \times f_i \Sigma_x f_{i+1} \times \cdots \times f_k = f_1 \times \cdots \times f_i \times f_{\bar{X}}$$

Pointwise product of factors f_1 and f_2:

$$f_1(x_1, \ldots, x_j, y_1, \ldots, y_k) \times f_2(y_1, \ldots, y_k, z_1, \ldots, z_l) = f(x_1, \ldots, x_j, y_1, \ldots, y_k, z_1, \ldots, z_l)$$

Complexity of exact inference

Multiply connected networks:

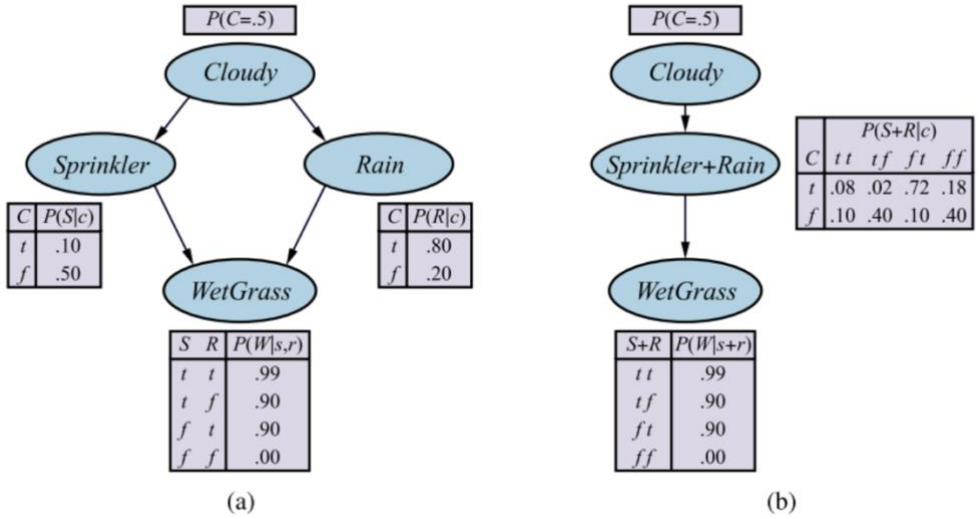
- exponential time and space complexity in the worst case, even when the number of parents per node is bounded

- inference in Bayes nets is NP-hard

Singly connected networks (or polytrees):

- any two nodes are connected by at most one (undirected) path
- time can be reduced to O(n)

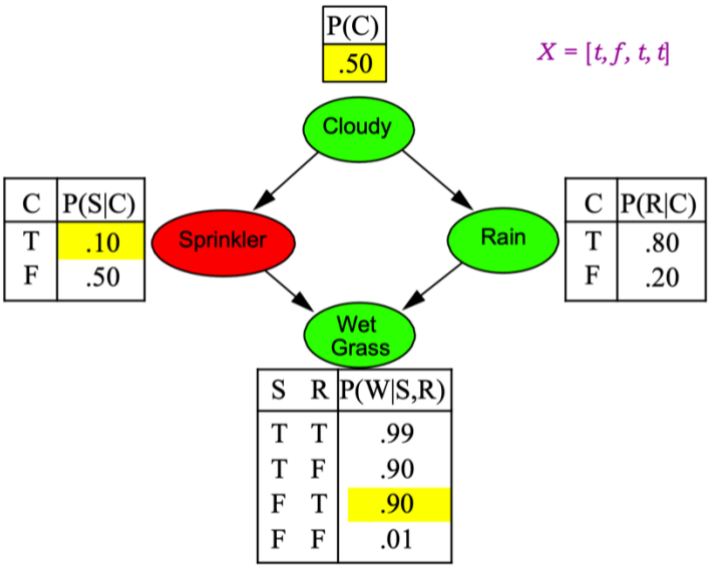
Clustering algorithms (also known as join tree algorithms) are widely used in commercial Bayes net tools.
The basic idea of clustering is to join individual nodes of the network to form cluster nodes in such a way that the resulting network is a polytree.



Approximate Inference for Bayesian Network

They work by generating random events based on the probabilities in the Bayes net and counting up the different answers found in those random events.

With enough samples, we can get arbitrarily close to recovering the true probability distribution—provided the Bayes net has no deterministic conditional distributions.



Random sampling techniques can give reasonable estimates of the true posterior probabilities in a network and can cope with much larger networks than can exact algorithms.

Rejection sampling: use evidence. Similar to basic real-world empirical estimation procedure.

Casual Networks

A restricted class of Bayesian networks that forbids all but causally compatible orderings.

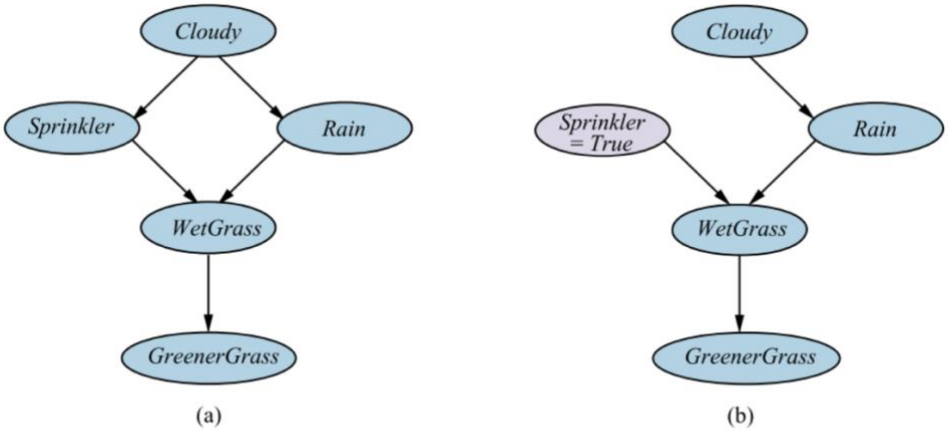


Figure 13.23 (a) A causal Bayesian network representing cause–effect relations among five variables. (b) The network after performing the action "turn *Sprinkler* on."

16 – Probabilistic Reasoning Over Time

Dynamic aspects of the problem are essential, they can change rapidly over time. To assess the current state from the history of evidence and to predict the outcomes of actions, we must model these changes.

Time and Uncertainty

States and observations

Discrete-time models, in which the world is viewed as a series of snapshots or time slices**.** The time interval Δ between slices is assumed to be the same for every interval.

X_t: denotes the set of state variables at time t, which are assumed to be unobservable

E_t : denotes the set of observable evidence variables. The observation at time t is $E_t = e_t$

Transition and sensor models

The transition model specifies the probability distribution over the latest state variables, given the previous values: $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{0:t-1})$.

Problem: the set $\mathbf{X}_{0:t-1}$ is unbounded in size as t increases.

Solution: Markov assumption**,** the current state depends on only a finite fixed number of previous states.

First order, $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$; Second order, $\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-2}, \mathbf{X}_{t-1})$;

$\mathbf{P}(\mathbf{E}_t \mid \mathbf{X}_t)$ is our sensor model, sensor Markov assumption:

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

The prior probability distribution at time 0, $\mathbf{P}(\mathbf{X}_0)$.

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

The first-order Markov assumption says that the state variables contain all the information needed to characterize the probability distribution for the next time slice.

Ways to improve the accuracy of the approximation:

- Increasing the order of the Markov process mode
- Increasing the set of state variables

Stationary vs Non-stationary Environment

We assume a stationary process, i.e., the conditional probability distribution is the same for all t.

$\mathbf{P}(\mathbf{X}_t \mid \mathbf{X}_{t-1})$ is the same for all t $\mathbf{P}(R_t \mid r_{t-1})$ = <0.7, 0.3>

Change is governed by laws that do not themselves change over time.

Inference in Temporal Models

Basic inference tasks that must be solved:

- Filtering (or state estimation) is the task of computing the belief state $P(X_t \mid e_{1:t})$
- Prediction: This is the task of computing the posterior distribution over a future state, given all evidence to date.
- Smoothing: This is the task of computing the posterior distribution over a past state, given all evidence up to the present
- Most likely explanation: Given a sequence of observations, we might wish to find the sequence of states that is most likely to have generated those observations

Besides inference tasks:

- Learning: The transition and sensor models, if not yet known, can be learned from observations

Filtering

A good filtering algorithm maintains a current state estimate and updates it, rather than going back over the entire history of percepts for each time.
The filtering function f takes into account the state estimation computed up to the present and the new evidence.

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

This process is called recursive estimation and is made of two parts:

1. Prediction: the current state distribution is projected forward from t to t+1: $\mathbf{P}(\boldsymbol{X}_{t+1}|\boldsymbol{e}_{1:t})$

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

2. Update: then it is updated using the new evidence e_{t+1}: $\mathbf{P}\ (\boldsymbol{e}_{t+1}|\boldsymbol{X}_{t+1})$

Filtering:

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

----update--- ---one-step prediction----

We can think of $\mathbf{P}\ (\boldsymbol{X}_t\ |\boldsymbol{e}_{1:t})$ as a message $f_{1:t}$ that is propagated forward in the sequence.

Prediction

Task of prediction can be seen simply as filtering without the contribution of new evidence. The filtering process already incorporates a one-step prediction.
In general, looking ahead k steps, at time t+k+1, given evidence up to t:

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

This computation involves only the transition model and not the sensor model. We can show that the predicted distribution converges to a fixed point, after which it remains constant for all time (the stationary distribution of the Markov process). The mixing time is the time to reach the fixed point.

Smoothing

It is the process of computing the posterior distribution of the state at some past time k given a complete sequence of observations up to the present t.

$$\mathbf{P}(\boldsymbol{X}_k|\boldsymbol{e}_{1:t}) \qquad \text{for } 0 \leq k < t$$

The additional evidence is expected to provide more information and more accurate predictions on the past

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

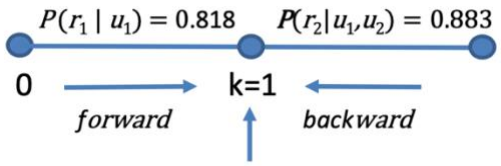
The terms in $f_{1:k}$ and $b_{k+1:t}$ can be implemented by two recursive calls:

one running forward from 1 to k and using the *filtering equation*

the other running backward from t to k + 1 for computing $\mathbf{P}(\boldsymbol{e}_{k+1:t}|\boldsymbol{X}_k)$

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

$b_{k+1:t} = P\,(\boldsymbol{e}_{k+2:t} \mid x_{k+1})$ is defined by the previous equation computing backward.



The complexity for smoothing at a specific time k with respect to evidence $e_{1:t}$ is O(t). For smoothing a sequence $O(t^2)$. A better linear approach exists. The forward–backward algorithm for smoothing; computing posterior probabilities of a sequence of states given a sequence of observations.

Finding the most likely sequence

We want to discover which one is the path maximizing the likelihood, in linear time.

Likelihood of a path: product of the transition probabilities along the path and the probabilities of the given observations at each state.

There is a recursive relationship between the most likely path to each state x_{t+1} and most likely paths to each previous state x_t.

$$max_{x_1 \dots x_t} P(x_1, \dots, x_t, X_{t+1} \mid e_{1:t+1}) =$$
$$= \alpha\, P(e_{t+1} \mid X_{t+1})\, max_{x_t} (P(X_{t+1} \mid x_t)\, max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, x_t \mid e_{1:t}))$$

Hidden Markov Models

Hidden Markov model, or HMM is a temporal probabilistic model in which the state of the process is described by a single, discrete random variable**.** Are a "simple" family but extensively used in practice to solve real-world problems.

Simplified matrix algorithms

Transition model $\mathbf{P}(X_t \mid X_{t-1})$ becomes an $S \times S$ matrix $\mathbf{T}$ where:

$\boldsymbol{T}_{ij} = P(X_t = j \mid X_{t-1} = i)$, $\boldsymbol{T}_{ij}$ is the probability of a transition from state i to state j.

The evidences are placed in a $S \times S$ diagonal matrix where the i-th diagonal entry is $\mathbf{O}_{ii} = \mathbf{P}(e_t \mid X_t = i)$ and the other entries are zeros.

The forward operation becomes:

$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t}$ and the backward $\mathbf{b}_{k+1:t} = \mathbf{T}\mathbf{O}_{k+1}\mathbf{b}_{k+2:t}$.

In HMMs, the matrix formulation provides an elegant description of the filtering and smoothing and reveals opportunities for improved algorithms

17 – Multi Agent Decisions

So far, we have largely assumed that only one agent has been doing the sensing, planning, and acting. But this represents a huge simplifying assumption, which fails to capture many real-world AI settings

Multi Agent Environments properties

One Decision Maker

- Multiple (logical or physical) actors, it contains only one decision maker.
- Benevolent agent assumption: agents will simply do what they are told.
- Multi-effector planning manages each effector while handling positive and negative interactions among the effectors.
- Multi-body planning: effectors are physically decoupled into detached units.

Multiple Decision Maker

- Each have preferences and choose and execute their own plan.
- There are two possibilities:
  - common goal for the actors, coordination problem (same direction)
  - own personal preference and can be diametrically opposed

Multiagent planning

Issue of concurrency: plans of each agent may be executed simultaneously. Agents must take into account the way in which their own actions interact with the actions of other agents.

Interleaved execution approach:

- certain the order of actions in the respective plans will be preserved
- assume that actions are atomic

- must be correct with respect to all possible interleavings of the plans
- does not model the case where two actions actually happen at the same time.
- the number of interleaved sequences is exponential with the number of agents and actions rising.

True concurrency approach:

- do not attempt to create a full serialized ordering of the actions,
- partially ordered

Perfect synchronization approach:

- global clock,
- same time, same duration, actions are always simultaneous
- simple semantics

Concurrent action constraint:

- stating which actions must or must not be executed concurrently

Cooperation and Coordination

Now let us consider a true multiagent setting in which each agent makes its own plan. Even if they have shared KB and goals, how can they coordinate to make sure they agree on the plan?
Adopt convention before engaging in joint activity, constraint on the selection of joint plans. When conventions are widespread, they are called social laws. Agents can use communication to achieve common knowledge of a feasible joint plan.

Plan recognition: single action (or short sequence of actions) by one agent is enough for the other to determine a joint plan unambiguously.

Non-Cooperative Game Theory

Games with a Single Move: "Normal Form Games"

All players take action simultaneously; no player has knowledge of the other players' choices.
They are defined by 3 components: players, actions and a utility to each player for each combination of actions by all the players (payoff function).
Decision making in multiagent settings is quite different in character to decision making in single-agent settings, because the players need to take each other's reasoning into account.

A pure strategy is a deterministic policy; for a single-move game, a pure strategy is just a single action.
Mixed strategy: a randomized policy that selects actions according to a probability distribution.
A strategy profile is an assignment of a strategy to each player.

s for player *p* strongly dominates strategy *s'* if the outcome for *s* is better for *p* than the outcome for *s'*, for every choice of strategies by the other player(s).
Strategy *s* weakly dominates *s'* if *s* is better than *s'* on at least one strategy profile and no worse on any other. A dominant strategy is a strategy that dominates all others.

Rational player will always choose a *dominant strategy* and avoid adominated strategy. Where all players choose a *dominant strategy*, then the outcome that results is said to be a dominant strategy equilibrium

Nash Equilibrium

In a Nash equilibrium, every player is simultaneously playing a best response to the choices of their counterparts. A Nash equilibrium represents a stable point in a game: stable in the sense that there is no rational incentive for any player to deviate. However, Nash equilibria are local stable points: a game may contain multiple Nash equilibria.

Social Welfare

Want to choose the best overall outcome, the outcome that would be best "for society as a whole".
Main idea: avoid outcomes that waste utility

- Pareto optimality: there is no other outcome that would make one player better off without making someone else worse off
- Utilitarian social welfare is a measure of how good an outcome is in the aggregate
- Egalitarian social welfare: maximize expected utility of the worst-off member of society
- Gini coefficient, which summarizes how evenly utility is spread among the players.

Computing equilibria

Exhaustive search can be used to find equilibria but it's not efficient.

Myopic best response (or "iterated best response")

1. Start by choosing a strategy profile at random; then,
2. If some player is not playing their optimal choice given the choices of others, flip their choice to an optimal one, and repeat the process
3. The process will converge if it leads to a strategy profile in which every player is making an optimal choice, given the choices of the others—a Nash equilibrium

Minimax algorithm, once the first player has revealed a strategy, the second player might as well choose a pure strategy

Repeated games

Simplest kind of multiple move game. Players repeatedly play rounds of a single-move game, called the stage game.

Need a finite model of strategies for games that will be played an infinite number of rounds → we use Finite State Machines (FSMs).
Tic-Tac, Hawk and Dove, GRIM strategies.

Cooperative Game Theory

Cooperative games capture decision making scenarios in which agents can form binding agreements with one another to cooperate. They can then benefit from receiving extra value compared to what they would get by acting alone.

Cooperative games with transferable utility

The idea of the model is that when a group of agents cooperate, the group as a whole obtains some utility value, which can then be split among the group members.
A cooperative game is G = (N, v), where G is defined as N = {1, ..., n} is a set of players and v a characteristic function which computes the value every subset of players could obtain if they cooperate.

Coalition: subset of players C.

- the set of all players N is known as the grand coalition.
- choice of joining one coalition creates partitions.
- N is a set of coalitions $\{C_1, \ldots, C_k\}$

The payoff must satisfy the constraint that each coalition C splits up all of its value $v(C)$ among its members.

Superadditivity: some cooperative games have two coalitions merge together: $v(C \cup D) \geq v(C) + v(D)$ for all $C, D \subseteq N$.
Players will opt to form coalitions when the value of the payoff is equal to or greater than if they were to work alone.

Shapley value: “fair” distributions scheme. Divide the $v(N)$ value among the players, given that the grand coalition N formed, according to contribution creating the value v(N).

Making Collective Decisions

Mechanism design—the problem of designing the right game for a collection of agents to play. Formally, a mechanism consists of:

- a language for describing the set of allowable strategies that agents may adopt
- a distinguished agent, called the *center*, that collects reports of strategy choices from the agents in the game
- an outcome rule, known to all agents, that the center uses to determine the payoffs to each agent, given their strategy choices. This section discusses some of the most important mechanisms.

Allocating scarce resources with auctions

- Ascending-bid auction: classic auction
- Sealed-bid auction: each bidder makes a single bid and communicates it to the auctioneer, without the other bidders seeing it.
- Sealed-bid second-price (aka “Vickrey auction”): winner pays second highest bid rather than own.

Voting

Using a social welfare function, to come up with a social preference order: a ranking of the candidates, from most preferred down to least preferred.

4 properties of good social welfare function to satisfy:

- The pareto condition: simply says that if every voter ranks A above B , then A is a better choice than B.

- The condorcet winner condition: winner is a candidate that would beat every other candidate in a pairwise election.
- Independence of irrelevant alternatives: is a condition that states that the relative likelihood of choosing A or B won't change if a third choice is placed into the mix. Candidate A wins the election. Candidate C is later disqualified and removed; candidate A should still win the election.
- No dictatorships

Arrow's theorem says impossible to satisfy all four conditions

19 - Probabilistic Programming

Bayesian are “factored representations”: the set of random variables is fixed and finite, and each has a fixed range of possible values. This fact limits their applicability.

There are two routes to introducing expressive power into probability theory:

1. via logic: to devise a language that defines probabilities over first-order possible worlds, rather than the propositional possible worlds of Bayes nets.
2. via traditional programming languages: we introduce stochastic elements— random choices, for example—into such languages, and view programs as defining probability distributions over their own execution traces.

Relational Probability Models

A probability model defines a set **Ω** of possible worlds with a probability **P(ω)** for each world **ω**.
For a first-order probability model, we need the possible worlds to be those of first-order logic. The model also needs to define a probability for each such possible world.
We can obtain the probability of any first-order logical sentence φ as a sum over the possible worlds where it is true. Conditional probabilities **P(φ|e)** can be obtained similarly, so we can, in principle, ask any question we want of our model.

Problem. Possible infinite first order models.
Solution. Database Semantics:

- Unique names assumption— here, we adopt it for the constant symbols.

- Domain closure—there are no more objects beyond those that are named.

We call these models Relational Probability Models (RPM) which have no closed-world assumptions: we can't just assume that every unknown fact is false.

Syntax and semantics

Constant, function, and predicate symbols. We also add type signature for each function: specification of the type of each argument and the function's value.
Basic random variables of the RPM are obtained by instantiating each function with each possible combination of objects

Inference in relational probability models

RPM Inference approach: construct the equivalent Bayesian network, given the known constant symbols belonging to each type (grounding/unrolling)

Open-Universe Probability Models

Database semantics is appropriate when we know exactly the set of relevant objects that exist and can identify them unambiguously. In many real-world settings, however, these assumptions are untenable.

Open Universe Probability Model (OUPM): allows generative steps compared to RPM: add objects to the possible world under construction, where the number and type of objects may depend on the objects that are already in that world and their properties and relations.

Inference in open-universe probability models

Because of the potentially huge and sometimes unbounded size of the implicit Bayes net that corresponds to a typical OUPM, unrolling it fully and

performing exact inference is quite impractical.
Instead, we must consider approximate inference: a logical query can be evaluated incrementally in each world visited, usually in constant time per world, rather than being recomputing from scratch.

Programs as Probability Models

The possible worlds are execution traces and the probability of any such trace is the probability of the random choices required for that trace to happen.
PPLs created in this way inherit all of the expressive power of the underlying language, including complex data structures, recursion, and, in some cases, higher-order functions.
Many PPLs are in fact computationally universal: they can represent any probability distribution

20 – Philosophy, Ethics, Safety and Future of AI

The limits of AI (Philosophy)

Weak AI: the idea that machines could act as if they were intelligent.
Strong AI: the assertion that machines that do so are *actually consciously* thinking.
The argument from informality says that human behavior is far too complex to be captured by any formal set of rules
The argument from disability a machine can never be kind, resourceful, beautiful, friendly, etc.
Overall, programs exceed human performance in some tasks and lag behind on others. The one thing that it is clear they can't do is be exactly human.

Can Machines Really Think?

Some philosophers claim that a machine that acts intelligently would not be actually thinking, but would be only a simulation of thinking. Turing argues the polite convention that everyone and machine think.

The Ethics of AI

Given that AI is a powerful technology, we have a moral obligation to use it well, to promote the positive aspects and avoid or mitigate the negative ones.
AI is a dual use technology: AI technologies that have peaceful applications can easily be applied to military purposes.

Surveillance, security, and privacy

As more of our institutions operate online, more vulnerable to cybercrime and cyberterrorism. More data on us is being collected by governments and corporation.

Fairness and bias

Machine learning models (especially) can perpetuate societal bias. Designers of machine learning systems have a moral responsibility to ensure that their systems are fair.
Sample size disparity can lead to biased results. In most data sets there will be fewer training examples of minority class. Machine learning algorithms give better accuracy with more training data, so that means that members of minority classes will experience lower accuracy

Set of best practices:

- Make sure that the software engineers talk with social scientists and domain experts to understand the issues and perspectives, and consider fairness from the start.
- Create an environment that fosters the development of a diverse pool of software engineers that are representative of society.

Trust and transparency

People need to be able to trust the systems they use, consumers want to know what is going on inside a system (transparency).

The future of work

An immediate reduction in employment when an employer finds a mechanical method to perform work previously done by a person. More automation with physical robots.

Robot Rights

If robots can feel pain, if they can dread death, if they are considered "persons," then the argument can be made that they have rights and deserve to have their rights recognize.

AI Safety

Design a robot to have low impact, instead of just maximizing utility, maximize the utility minus a weighted summary of all changes to the state of the world. Need to be very careful in specifying what we want, because with utility maximizers we get what we actually asked for. The value alignment problem

AI Components

- Sensors and actuators
- Representing the state of the world
- Selecting Actions
- Deciding What we Want
- Learning
- Resources

AI Architectures

AI has long had a split between symbolic systems (based on logical and probabilistic inference) and connectionist systems (based on loss minimization over a large number of uninterpreted parameters).

Future

AI is different from previous revolutionary technologies. Improving printing, plumbing, air travel, and telephony to their logical limits would not produce anything to threaten human supremacy in the world. Improving AI to its logical limit certainly could.
Automation is already changing the way people work. As a society, we will have to deal with these changes.