CS 188 – Final Review Topics

- Definitions of Artificial Intelligence have two primary bases:
  1. intelligent thought processes and reasoning
  2. human-like behavior
- **Rationality** – Property of an agent: "does the right thing" given its knowledge.
- **Turing Test** – artificial intelligence test proposed by Alan Turing requiring indistinguishably of human and computer responses to questions posed.
- **Total Turing Test** – Addition of visual elements of response and perception.
- Approaches to Artificial Intelligence:
  1. Cognitive Modeling – AI modeled on theory of human thought processes.
  2. Laws of Thought – AI built on the logical laws as a basis for rational thought processes.
     - Inference is only one of several mechanisms for rationality.
  3. Rational Agent – AI that acts to achieve the best expected outcome given the observations perceived.
     - A clearly defined and completely general standard for rationality.

Intelligent Agents

Agent – anything viewed as perceiving its *environment* through *sensors* and acting upon the environment through *actuators*.

- general assumption: an agent can perceive their actions but necessarily their effect
- **percept** – an agent's percetual inputs at a given instance.
- **percept sequence** – the history of all that an agent has perceived.
- *In general, an agent's choice of action at any given time can depend on the entire percept sequence thus far observed.*
- **agent function** - a map from the precept sequence to an action

Performance Measure

- **performance measure** – a *subjective* criterion to measure the success of an agent's behavior typically stipulated by the designer of the agent.
- *Generally, it is better to design a performance measure according to what one wants accomplished rather than how one thinks the agent should behave.*

Rationality

- **Rationality** – behavior depending on 4 factors:
  1. The performance measure defining "success"
  2. The agent's prior knowledge about the environment
  3. The actions the agent can perform
  4. The agent's percept sequence to date
- **Rationality vs. Perfection** – rationality maximizes expected performance while perfection maximizes actual performance.
- **rational agent** – For each possible percept sequence, a rational agent should select an action from its set of allowable actions that is expected to maximize its performance measure given the evidence of the percept sequence and the agent's prior knowledge about the environment.
  - **Information Gathering (Exploration)** – actions taken specifically to modify future percepts by increasing the agent's knowledge.
  - **learning** – the process of modifying prior knowledge based on experience.
  - **autonomy** – the ability to compensate for partial or incorrect prior knowledge thereby eventually becoming independent of prior knowledge.

Task Environments

task environment – the problem the agent is solving as characterized by
1) Performance Measure 2) Environment 3) Actuators 4) Sensors – PEAS.

Dimensions of task environments

1) *Fully Observable vs. Partially Observable*
   - **fully observable** – sensor's detect all relevant aspects for choosing an action
2) *Deterministic vs. Stochastic*
   - **deterministic** –completely determined by current state and agent's action
   - **stochastic** – as though environment is random dependent on state and action.
   - **strategic** – environment deterministic except for other agent's actions.
3) *Episodic vs. Sequential*
   - **episodic** – experience divided into atomic independent episodes.
   - **sequential** – current decision could affect all future decisions.
4) *Static vs. Dynamic*
   - **static** – environment doesn't change until agent makes a decision.
   - **dynamic** – environment doesn't wait for agent.
   - **semi-dynamic** – environment is static, but performance measure may change during course of the decision.
5) *Discrete vs. Continuous*
6) *Single agent vs. Multiagent*

Hardest: *partially observable, stochastic, sequential, dynamic, continuous, multiagent.*

Agent Structure

- Agent = Architecture + Agent Program
- **Architecture** – the machinery that an agent executes on.
- **Agent Program** - a concrete implementation of an agent function.
    - **Simple Reflex Agents** – an agent that chooses actions only based on the current percept.
        - **condition-action rule** –maps a state (condition) to an action.
        - *Rational only if the correct decision can be made solely on the basis of the current precept… the environment is fully observable.*
    - **Model-Based Reflex Agents** – uses a model of the world to choose actions.
        - **internal state** – a representation of unobserved aspects of current state dependent on percept history.
        - **model** – knowledge about "how the world works".
    - **Goal-Based Agents** – an agent that chooses actions to achieve goals.
        - **goal** – description of situations that are desirable.
    - **Utility-Based Agents** – agent chooses actions based on a preference (*utility*) for each state.
        - **utility function** – a mapping of a (sequence of) state(s) to a real number describing the "degree of happiness" of that state.

Learning – the process of modification of each component of an agent to make the components agree closer with the available feedback thereby improving the agent's performance.

- **learning element** – responsible for making improvements
- **performance element** – responsible for selecting external actions… the agent being modified.
- **critic** – provides feedback on the agent's performance and suggests improvements.
    - **performance standard** – a *fixed* measure of agent's performance.
        - distinguishes the *reward* in the percept by providing direct feedback on quality of agent's performance.
- **problem generator** – suggests actions that will lead to exploration.

Problem Solving

- **problem**
    - initial state – the initial configuration given as input to the agent.
    - actions – the set of actions (currently) available to the agent.
        - successor function *succ(s)* – returns the set of $\langle action, state \rangle$ tuples that defines the new *state* achieved by taking *action* from state *s*.
    - goal test – determines whether a state is a goal state.
        - goal (state) – a state that is desirable to the agent.
    - path cost – the numeric cost of a path reflecting a performance measure. Typically defined as the sum of costs of each step on the path.
        - **step cost** $c(x,a,y)$ - cost of action *a* going from state *x* to state *y*.
- **state space** – the set of all possible states an agent could be in.
- **path** – a sequence of states connected by a sequence of actions.
- **solution** – a path from the initial state to a goal state.
    - **optimal solution** – the solution with minimal path cost.
- **problem solving agent** – an agent that attempts to discover a *solution* to a problem so that following the actions in that *solution* will lead to a goal.
    - ***formulation*** – building a representation of the agent's world.
        - **abstraction** – the process of removing details about the world.
        - **goal formulation** – defining goals to limit the agent's objectives.
        - **problem formulation** – defining the set of actions and states relevant to achieving a goal.
    - ***search*** – takes a *problem* as input and returns a *solution* to the problem.
    - ***execute*** – carrying out the actions recommended in the *solution*.
- **incremental formulation** – a state is modified by operators that augment the state (add new components without altering old ones).
- **complete-state formulation** – every state contains all objects and are modified by operators that alter the arrangement of those objects.

Search

Terminology

- **search tree (graph)** – the path the search algorithm follows in exploring the state space via an initial state and a successor function
  - **search node** – a state from the state space which has a successor function. A node is comprised of the following:
    1. <u>state</u> – the state the node represents
    2. <u>parent</u> – the predecessor of the node.
    3. <u>action</u> – the action applied to the parent to reach the node.
    4. **path-cost *g(n)*** – the cost of the path from the initial state.
    5. <u>depth</u> – the number of search steps along the path.
  - *expanding a node* – generating a new set of states via the node's successor function. <u>A node is not checked to be terminal until it is expanded.</u>
  - Note that several nodes in the search tree may contain the same states, generated by different paths. Hence, <u>*the search becomes a graph in state space.*</u>
- **search strategy** – the methodology for choosing the next node to expand.
- **fringe** – the collection of nodes generated but not yet expanded.
  - this collection typically imposes an ordering on which nodes in the collection will be expanded next based on a preference → **queue**.

Assessing Algorithms

- Performance Measures for our algorithms:
  - **completeness** – Is algorithm guaranteed to find an existing solution?
  - **optimality** – Does the algorithm find the optimal solution first?
  - **time complexity** – How long does it take to find a solution
  - **space complexity** – How much memory is needed to find a solution?
- Relevant quantities:
  - **branching factor** *b* – maximum number of successors of a node.
  - *d* – depth of the shallowest goal node.
  - *m* – maximum length of any path in the state space.
- **path cost** – a function used to define a numeric cost to each path.

Uninformed (Blind) search – search solely on the basis of being to expand the successors of a state and being able to distinguish a goal-state.

Criterion	**BFS**	**Uniform**	**DFS**	**DLS**	**Iterative**	**Bidirect.**
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^{d+1})$	$O\left(b^{\lceil C^*/\varepsilon \rceil}\right)$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O\left(b^{\lceil C^*/\varepsilon \rceil}\right)$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

1. complete if *b* is finite.
2. complete if step cost is at least $\varepsilon > 0$.
3. optimal if step costs are all identical.
4. if both directions use BFS.

- **Breadth-first Search** – all nodes at a given depth in the search tree are expanded before any of the nodes at larger depths → implemented with FIFO queue
- **Uniform-cost Search** – expands the next unexpanded node with the *lowest path cost* → implemented by a priority queue. When costs are equal, becomes BFS.
- **Depth-first Search** – always expands the *deepest* node in the current fringe of the search tree (search backs-up when path unsuccessful) → implemented by Stack.
- **Depth-limited Search** – depth-first search with a predetermined depth limit *l*. Becomes DFS when $l = \infty$.
- **Iterative Deepening Depth-first Search** – iteratively repeated depth-limited search where *l* is increased by 1 on each iteration from an initial value of 0. This combines the benefits of BFS and DFS.
- **Bidirectional Search** – simultaneous searches from the initial state forward and from the goal state backwards that stop when the 2 searches meet. Encouraged by the fact that $b^{d/2} + b^{d/2} \ll b^d$
- **Avoiding Repeated States** (Graph Search)– avoiding repeated visits to states that have already been visited can result in substantial savings in space and time. *Algorithms that forget their history are doomed to repeat it.*

Searching with Partial Information

- ***Sensorless (Conformant) Problems*** – agent has no sensors.
  - **belief state** – a set of states representing the agent's belief of what states it might be in. In general, environment of *S* states has 2^S belief states.
  - **coercion** – executing actions that cause the agent's belief state to collapse to a certain set of states.
    - *solution* – coercing the belief state to a set of all goal states.
- ***Contingency Problems*** – environment is partially observable or the outcome of an agent's actions is uncertain.
  - **adversarial** – uncertainty is caused by actions of other agents.
  - **contingency plan** - trees of decisions made based on the current set of percepts made after the last action.
  - *Agent can act before finding a guaranteed plan*
    - idea of acting and seeing what contingences actually arise.

Informed Search – use problem-specific knowledge to improve search.

- **Best-First Search** – general Tree (Graph) Search where node's are selected based on an evaluation function *f(n)* – cost of cheapest path to goal through node n.
- **Greedy Best-First Search** – Assumes *f(n) = h(n)*; a heuristic function.
  - susceptible to false starts
  - not optimal; incomplete.
  - Worst case time and space: O(b^m).

A* Search – *f(n) = g(n) + h(n)* where *g(n)* is the cost to reach the node *n* and *h(n)* is a heuristic function estimating the cheapest cost to a goal through *n*.

- A* is *optimal* if *h(n)* is an *admissible* {*consistent* for Graph-Search} heuristic.
- A* is *complete*.
- If h(n) is *consistent*, the values of f(n) along any path are nondecreasing!
- A* is **optimally efficient** for any given heuristic since any algorithm that doesn't expand a node n with $f(n) < C^*$ might miss the optimal solution.

Heuristic Functions

- **Heuristic Function** h(n) – estimated cost of cheapest path to goal through node n.
- **Admissible Heuristic** – h(n) never overestimates the cost to reach a goal.
- **Consistent (Monotonic) Heuristic** – h(n) is not more than the cost through n to n' plus h(n'). Thus, a general triangle inequality:
$$h(n) \le c(n,a,n') + h(n')$$
- **Dominance** – a heuristic h_1 is said to **dominate** another heuristic h_2 if, for any node n, $h_1(n) \ge h_2(n)$.
- **Relaxed Problem** – a problem with fewer restrictions on the actions allowable in the problem domain. *The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem!*
- <u>MultiHeuristic</u>: if we have a set of heuristics $\{h_i\}$ we can combine them into a single heuristic: $\tilde{h}(n) = \max_i \{h_i(n)\}$ if $\{h_i\}$ is admissible, $\tilde{h}$ is admissible.

Local Search

- **Local Search Algorithms** – When the path to reach goal is irrelevant, local search are methods for only maintaining *current state* and (generally) only moving to its neighbors. Often used in optimizations where the goal is to minimize a objective function.
  - **state space landscape** – the space of possible states defined by a "*location*" corresponding to state and a "*elevation*" corresponding to an evaluation, cost, or objective function.
  - **complete local search** – always finds a goal (if any exist)
  - **optimal local search** – always finds the global min/max.
  - **greedy local search** – moves to "good" neighbor without considering future.

- **Hill-Climbing Search** – Always moves in "uphill" direction to maximize objective only searching amongst immediate neighbors of current state and terminating when no improvements can be made ➔ *greedy*.
  - Problems: 1) Local Max/Min 2) Ridges 3) Plateaux
  - <u>sideways moves</u> – moves along "flat" objective to get off plateau.

- **Simulated Annealing** – Hill-Climbing with random walk.
  - Candidate move β is randomly selected. If candidate is uphill, it is always accepted. Otherwise, it is accepted with a probability exponentially decreasing with "badness" ΔE and decreasing as temperature T is lowered → *Boltzmann Distrubution*.
$$P(n_{t+1} = \beta) = \min\left(1, \exp\left\{\Delta E(\beta)/T\right\}\right)$$
  - If the *schedule* for T cools "slowly enough", simulated annealing finds global optimum with probability approaching 1.

- **Local Beam Search** – maintains the *k* "best" successor states; an approach more powerful than *k* independent searches since information effectively passes between the "search threads."

- **Genetic Algorithms** – A variant of stochastic beam search in which successors are generated through combinations of 2 current states.
  - **population** – the *k* states maintained by the algorithm
  - **individual** – an instance in the state space.
  - **fitness function** – an evaluation function that returns higher values for better states.
  - *Essence of Algorithm*
    - Parents are randomly selected with probabilities based on fitness.
    - **Crossover** points are selected randomly in accordance with the rules of the state.
    - **Random Mutation** occurs in each successor with ε probability.
- *Continuous Spaces*
  - **Gradient Descent (Ascent)** – moves the current solution in the direction of the gradient in the state-space landscape.

Gradient - $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$

Update - $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

adversarial search (games) – competitive multiagent environments (agent's have conflicting goals). In particular, adversarial search is mixture of *search* and *game theory*. The typical game is a *deterministic, turn-taking, two-player* **zero-sum** game of **perfect information**. These games are a sequence of decisions that reach a *terminal state*.

- **game tree** – a representation that represents all legal sequences of decisions.
  - **root –** the *initial state* of the game (with a starting player).
  - **(internal) nodes –** represents decision made by one of the players. The node is labeled by the player making the decision (*Max/Min*).
  - **edges –** legal choices for a given decision in the tree. These are specified by a *successor function* that lists legal (*move*, *state*) pairs.
  - **terminal node –** an ending of the game giving a *utility* to each player.
    - **utility function –** maps a terminal state to a value.
- **optimal strategy** – a contingent strategy that leads to an outcome at least as good as any other strategy by assuming the opponent is infallible.
  - <u>minimax algorithm</u> – finds an optimal strategy by depth-first exhaustive search which annotates each node of the tree with a **minimax-value**:

$$\text{minimax-value}(n)=\begin{cases}\text{utility}(n) & n\in Terminal\\ \max\limits_{s\in child(n)} \text{minimax-value}(s) & n\in MAX\\ \min\limits_{s\in child(n)} \text{minimax-value}(s) & n\in MIN\end{cases}$$

- **alpha-beta pruning** – a modified minimax search that prunes branches that cannot influence the final result.
  - α – the maximum value so far at any choice point along the path for MAX
  - β - the minimum value so far at any choice point along the path for MIN

<u>Stopping search prematurely</u> – time limits prevent full exploration of the game tree.

- **evaluation function** – a heuristic for accessing the utility of a nonterminal game state; that is, it returns an estimate of the expected value of a state.
- **cutting-off search** – determine a reasonable time to stop search (e.g. *iterative deepening* explores deeper until time elapses).

<u>Games of Chance</u>

- **chance nodes** – nodes (denoted by circles) indicating an element of chance is introduced and arcs from this node are probabilistic transitions
  - The minimax algorithm is identical & chance nodes are *expected values:*

$$\text{expectiminimax}(n)=\sum_{s\in child(n)} P(s)\cdot \text{expectiminimax}(s) \qquad n\in Chance$$

    - In games of chance, the evaluation function *must be a* ***positive linear transform*** *of the probability of winning from a position.*

<u>Games of Chance with *imperfect information*</u>

- **averaging over clairvoyancy –** the strategy of computing optimal moves by averaging over possibilities for the unseen variables.
  - This strategy is flawed as it assumes all future uncertainty will have disappeared by the time the future is reached.
- **belief states** – games states are replaced by *possible* states along with their corresponding probabilities.

Propositional Logic

Knowledge-Based Agents

- logical agents are always definite – each proposition is either true/false or unknown (agnostic).
- **knowledge representation language (KRL) –** expresses world knowledge.
  - **declarative approach –** language is designed to be able to easily express knowledge for the world the language is being implemented for.
  - **procedural approach –** encodes desired behaviors directly in code.
- **sentence –** a statement expressing a truth about the world in the KRL.
- **knowledge base (KB) –** a set of KRL sentences describing the world.
  - **background knowledge –** initial knowledge in the KB
  - **knowledge level –** we only need to specify what the agent knows and what its goals are in order to specify its behavior
  - **Tell(P) –** function that adds knowledge P to the KB.
  - **Ask(P) –** function that queries the agent about the truth of P.
- **inference –** the process of deriving new sentences from the knowledge base.
  - *When the agent draws a conclusion from available information, it is guaranteed to be correct if the available information is correct.*

Logic

- **syntax –** description of a KRL in terms of well-formed sentences of the language.
- **semantics –** defines the truth of statements in the KRL w.r.t. each possible world.
- **model –** the "possible world" that is described by a KB.
  - **model checking –** enumeration of all possible models to ensure that α is true in all models in which KB is true.
- **logical inference –** the process of using entailment to derive conclusions
- **logical entailment –** the concept of 1 sentence following from another sentence:

  $\alpha \models \beta$ if α is true, then β must also be true.

  Note: while similar to the notion of implication, entailment is a meta-statement, not a part of the language itself. That is, statements using entailment are used to describe other logical statements.
  - <u>Monotonicity</u> – a set of entailed sentences can only *increase* in information as information is added to the knowledge base.

$$KB \models \alpha \quad \Rightarrow \quad KB \wedge \beta \models \alpha$$

- **derivation –** if an inference procedure *i* can derive α from KB,

$$KB \vdash_i \alpha$$

- **sound (truth-preserving) inference –** an inference procedure that derives only entailed sentences.
  - *if KB is true in the real world, the any sentence* α *derived from KB by a sound inference procedure is also true in the real world.*
- **complete inference –** an inference procedure that can derive all entailed sentence.
- **grounding –** the connection, if any, between the logical reasoning processes and the real environment.

Propositional Logic

- **atomic sentence –** indivisible syntactic elements consisting of a single **propositional symbol**. *True* and *False* have fixed meaning.
- **complex sentence –** sentence constructed from other sentences joined by logical connectives:
  - **logical connectives:**
    - **not** $\neg$ **–** negation, **and** $\wedge$ **–** conjunction, **or** $\vee$ **–** disjunction
    - **implies** $\Rightarrow$ – implication $((\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta))$ *Note: if α is false, $\alpha \Rightarrow \beta$ says nothing about β.*
    - **if and only if** $\Leftrightarrow$ **–** biconditional
  - **order of operations (high->low):** $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- *Every known inference algorithm for propositional logic has a worst-case complexity exponential in the size of the input.*
- **logical equivalence –** two sentences α and β are logically equivalent if they are true in the same set of models.

  $$\alpha \equiv \beta \quad \Leftrightarrow \quad \alpha \models \beta \wedge \beta \models \alpha$$
- **validity –** a sentence is valid if it is true in all models.
  - **tautology –** sentences that are necessarily true.
- **Deduction Theorem –** *For any sentences α and β, $\alpha \models \beta$ if and only if the sentence $\alpha \Rightarrow \beta$ is valid.*
- **Satisfiablility –** a sentence is satisfiable if it is true in some model.
  - *Determining satisfiablity in propositional logic is NP-complete.*
  - **Proof by contradiction (refutation):** $\alpha \models \beta$ *if and only if the sentence* $\neg(\alpha \Rightarrow \beta)$ or rather $(\alpha \wedge \neg\beta)$ is unsatisfiable.
- **inferentially equivalent** – two sentences α and β are inferentially equivalent if the satisfiablity of α implies the satisfiablity of β and vice versa.

Reasoning Patterns in Propositional Logic

Common Patterns

	Modus Pones	And Eliminate	Bidirectional	Resolution
Premises	$\alpha \Rightarrow \beta, \quad \alpha$	$\alpha \wedge \beta$	$\alpha \Leftrightarrow \beta$	$\ell_1 \vee \ldots \vee \ell_k, \quad \neg\ell_i$
Conclusion	β	α	$(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$	$\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k$

Full Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \quad m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{l+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where ℓ_i and m_j are complementary literals

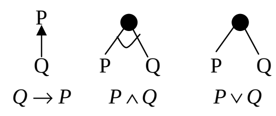
- **conjunctive normal form (CNF)** – every sentence of propositional logic is *logically equivalent* to a conjunction of disjunctions of literals.

  $$\left(l_{1,1} \vee \ldots \vee l_{1,n_1}\right) \wedge \ldots \wedge \left(l_{m,1} \vee \ldots \vee l_{m,n_m}\right)$$

  1. Eliminate biconditionals: $\alpha \Leftrightarrow \beta \quad \equiv \quad (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
  2. Eliminate implications $\alpha \Rightarrow \beta \quad \equiv \quad \neg\alpha \vee \beta$
  3. Move $\neg$ inwards
  4. Distribute $\wedge$ over $\vee$.
- A *complex sentence* can always be represented in CNF.
  1. literal – atomic sentence (positive) or negated atomic sentence (negative).
  2. clause – a disjunction of literals
  3. sentence – a conjunction of clauses.
- *Definite Clauses* – disjunction of literals of which exactly one is positive.

  $$\neg n_1 \vee \ldots \vee \neg n_m \vee p \quad \equiv \quad \underbrace{n_1 \wedge \ldots \wedge n_m}_{body} \Rightarrow \underbrace{p}_{head}$$
  - **head** – the positive literal.
  - **body** – the negative literals; the premises.
  - **fact** – a definite clause with no negative literals.
  - **Horn clause** (*integrity constraint*) – a disjunction of literals at most one of which is positive. Horn clauses have the following advantages:
    - Inference can be done with forward/backward chaining.
    - Deciding entailment is *linear* in the size of the KB.
- **resolution** – a *sound* inference algorithm based on the resolution rule.
  - *By applying the only the resolution rule, any complete search algorithm can derive any conclusion entailed by any KB in propositional logic.*
  - **refutation completeness** – resolution can be used to confirm or refute any sentence, but it cannot enumerate all true sentences.
  - **resolution algorithm**
    - to show $KB \models \alpha$ we will show that $KB \wedge \neg\alpha$ is unsatisfiable.
    - $KB \wedge \neg\alpha$ is converted into CNF… a sequence of clauses
    - The resolution rule is applied to resulting clauses… each pair with complementary literals is resolved into a new clause.
      - if no new clauses can be added, α is not entailed.
      - if the *empty clause* {} is derived, α is entailed.

- **forward chaining** – a *sound* and *complete* inference algorithm (for Horn clauses) that is essentially Modus Pones. This algorithm is *data-driven reasoning*; reasoning which starts from the known data.
    - **AND-OR graph** – represents the derivation by a graph of literals. Disjunctions are represented by converging links and conjunctions are represented by multiple links joined by an arc.



- **backward chaining** – a *sound* and *complete* inference algorithm (for Horn clauses) based on Modus Pones. This algorithm is *goal-directed reasoning*; reasoning that works backward from the goal.

Satisfiability

- **Davis-Putnam algorithm** – an algorithm for checking satisfiability based on the fact that satisfiability is commutative. Essentially, it is a DFS method of *model checking*.
    - Heuristics
        - **early termination** – short-circuit logical evaluations. A clause is true if *any* literal is true. A sentence is false if *any* clause is false.
        - **pure symbol heuristic** – a symbol that appears with the same sign in all clauses of a sentence (all positive literals or negative ones).
            - Making these literals *true* can never make a clause *false*. Hence, pure symbols are fixed respectively.
        - **unit clause heuristic** – assignment of true to unit clauses.
            - **unit clause** – a clause in which all literals but one have been assigned false → 1 way to make clause true.
            - unit propagation – assigning one unit clause creates another causing a cascade of forced assignments.
- **WalkSAT** – a local search algorithm based on the idea of a random walk that randomly alters the current assignment based on a *min-conflicts* heuristic.
    - If a satisfying assignment exists, it will be found, eventually.
    - WalkSAT can not guarantee a sentence is unsatisfiable.
- *Hard Satisfiablility*
    - Let *m* be the number of clauses and *n* be the number of symbols.
    - The ratio *m/n* is indicative of the difficulty of the problem.
        - **underconstrained** – relatively small *m/n* thus making the expected number of satisfying assignments high.
        - **overconstrained** – relatively high *m/n* thus making the expected number of satisfying assignments low.
        - **critical point** – value of *m/n* such that the problem is nearly satisfiable and nearly unsatisfiable. Thus, the most difficult cases for satifiablity algorithms

Propositional Logic Agents

- **inference-based agent** – an agent that maintains a knowledge base of propositions and uses the inference procedures described above for reasoning.
    - It is beyond the power of propositional logic to efficiently express statements that are true for sets of objects – FOL.
    - A proliferation of clauses occurs due to the fact that a different set of clauses is needed for each step in time.
- **circuit-based agent –** a reflex agent in which percepts are inputs to a sequential circuit – a network of gates (logical connectives) and registers (store truth value of a single proposition)
    - **dataflow** – at each time step, the inputs are set for that time step and signals propagate through the circuit.
    - **delay line** – implements internal state by feeding output of a register back into the register as input at the next time step. The delay is represented as a triangular gate.

First-Order Logic

Differences in Logics

- **Ontological Commitment** – What the logic assumes about the state of reality.
- **Epistemological Commitment –** the possible state of knowledge a logic allows with respect to each fact.

Language	Ontological	Epistemological
Propositional Logic	Facts	T/F/unknown
1st-order Logic	Facts, objects, relations	T/F/unknown
Temporal Logic	Facts, objects, relations, time	T/F/unknown
Probability Theory	Facts	Degree of belief
Fuzzy Logic	Facts with degree of truth	Known interval value

Syntax of First Order Logic

- **Objects**: the <u>domain</u> of the model is the set of objects in it ➔ *constant* symbols.
- **Relations**: a set of <u>tuples</u> of objects that are related ➔ *predicate* symbols.
- **Function**: an object can be related to exactly 1 object. Functions in FOL must be <u>total functions</u> that must be defined over the whole domain ➔ *function* symbols.
    - The *arity* of a relation or function is the number of arguments it takes.
- **intended interpretation** – the interpretation specify what each symbol actually represents in the real world.
- **term** – a logical expression that refers to an object.
    - **ground term** – a term with no variables.
- **atomic sentence** – a predicate symbol followed by a parenthesized list of terms; the arguments. An atomic sentence is true (under the given interpretation) if the predicate holds for the objects given as arguments.
- **complex sentence** – atomic sentences joined by logical connectives.
- **quantifiers** – allow us to express properties of groups of objects.
    - **Universal** $\forall x \quad P(x)$ - for all objects x, P(x) holds. P(x) usually formed with connective ==>.
    - **Existential** $\exists x \quad P(x)$ - there exists an object x, such that P(x) holds. Typically used with the connective AND.
- **Equality –** Asserts that two statements are equivalent.

Semantics of First-Order Logic

- **assertion** – sentences that assert what is known about the world.
- **queries (goals)** – questions asked to the KB. Answered in a list of substitutions that satisfy the query.
- **substitution (binding list)** $\{x/X\}$ - set of variable/term pairs that represent substituting the term *X* for the variable *x*.
    - $SUBST(\theta,\alpha)$ applies the substitution θ to sentence α.
- **axiom** – basic factual information from which conclusions are derived. Each axiom cannot be entailed by the KB without explicitly including it.
- **theorems** – facts entailed by the axioms.
- **perception –** the reasoning process by which percepts are entailed from the KB.

Types of rules in FOL

- **synchronic** – sentences relating properties of a world state to other properties in the same world state.
    - **Diagnostic Rules –** lead from observed effects to hidden causes.
    - **Casual Rules –** hidden properties cause observed precepts.
    - **Model-Based Reasoning –** Systems that use casual rules to model the world.
- **diachronic** - sentences relating properties of a world state to other properties in another world state; thus allowing reasoning across time.
- *If the axioms correctly and completely describe the way the world works and the way that percepts are produced, then any complete logical inference procedure will infer the strongest possible description of the world state given the available precepts.*

Knowledge Engineering – process of constructing a knowledge base.

1. Identify the task
2. Assemble relevant knowledge; **knowledge acquisition**
3. Decide on vocabulary of predicates, functions, and constants = **ontology**.
4. Encode general knowledge about the domain = **axioms**.
5. Encode a description of the specific problem instance = knowledge base.
6. Pose queries KB and get answers.
7. Debug the knowledge base.

Inference in First-Order Logic

The question of entailment for first-order logic is ***semidecidable*** *– algorithms exist that are able to correctly identify every entailed sentence, but no algorithm can correctly identify every nonentailed sentence.*

Fundamentals

- Converting quantified statements
  - Universal Instantiation – substitute variable of a universally quantified statement with a ground term. Can be applied repeatedly.
    $$\frac{\forall v \quad \alpha}{\text{SUBST}(\{v/g\},\alpha)}$$
  - Existential Instantiation – substituting a ground term, Skolem constant, for the variable in an existential statement. Can only be applied once!
    $$\frac{\exists v \quad \alpha}{\text{SUBST}(\{v/k\},\alpha)}$$
    If the Existential quantifier is embedded in a universally quantified statement, you must use a skolem function of the universally quantified variables.
- **Propositionalization** – the process of converting a first-order sentences into propositional logic.
  - a propositionalized KB is *inferentially equivalent* to the original KB but not *logically equivalent*.
- **Conjunctive Normal Form** – a conjunction of clauses where each clause is a disjunction of literals.
  - *Every sentence in FOL can be converted into an inferentially equivalent CNF sentence.*
  - Conversion to CNF:
    1. Eliminate implications using: $p \Rightarrow q \quad \equiv \quad \neg p \vee q$
    2. Move $\neg$ inwards in quantified statements.
    3. **Standardize Variables**: eliminate repeated names.
    4. **Skolemization**: removal of existential quantifiers with Skolem functions (for every enclosing universal quantifier variable) or Skolem constants.
    5. Drop universal quantifiers.
    6. Distribute $\wedge$ over $\vee$.

- **Unification** – the process of finding substitutions that make different logical expressions look identical. Given two sentences, *p* and *q*, UNIFY returns the most general unifier θ, if a unifier exists.
  $$\text{UNIFY}(p,q)=\theta \quad | \quad \text{SUBST}(\theta,p)=\text{SUBST}(\theta,q)$$
  - standardizing apart – renaming variables to avoid name clashes.
  - most general unifier – for every unifiable pair of expressions there is a unique unifier that is more general than any other.
    - A unifier is *more general* than another unifier if the first places fewer restrictions on the values of the variables.
  - occur check – when matching a variable against a complex term, one must check whether the variable itself occurs in the term, in which case the unification fails.
- **Generalized Modus Ponens** – A lifted version of Modus Pones for FOL. For atomic sentences p_i, p_i', and q, where $\exists\theta\left(\forall i \quad \text{SUBST}(\theta,p_i')=\text{SUBST}(\theta,p_i)\right)$
  $$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{SUBST(\theta,q)}$$

Forward Chaining

Forward Chaining – Uses Modus Pones to infer new atomic sentences in the KB until no more inferences are possible. This approach is *sound* and is *complete* for definite clause KBs. FOL Inference with definite clauses is *semidecidable* due to functions.

- *Datalog KB* – set of first-order definite clauses with no function symbols.
- *fixed point* – a state of the KB for which no further sentences can be inferred.

Backward Chaining

Backward Chaining – works backwards from the goal, matching the effects of rules to support the desired proof based on Modus Pones. It is a DFS search ➔ linear space complexity; repeated states and incompleteness.

- Algorithm outline:
  - A list of unsatisfied goals is kept as a stack; 1 for each branch of the proof.
  - If all goals of the stack are satisfied by the initial state, the proof succeeds.
  - Goals are popped off the stack and if unsatisfied:
    - Every clause whose *head* unifies with the goal makes a separate branch of the proof.
    - The *body* of each such clause are added to the stack: *a new branch*.
- Problems with Backward Chaining
  1. *Repeated States* – inference can be exponential in number of ground facts.
     - **memoization** – caching solutions to subgoals for reuse.
  2. *Infinite Paths* –makes backward chaining incomplete.

Logic Programming – logic is expressed as formal declarative language and used to solve problems via inference.

- *Prolog* – logic PL with depth-first backward chaining.
  - Rules expressed as follows:
    $$p \coloneqq n_1, \ldots, n_m. \quad \equiv \quad n_1 \wedge \ldots \wedge n_m \Rightarrow p$$
  - Built in arithmetic functions and some predicates have side-effects
  - **Negation as failure** – if a goal P can not be proved, $\neg P$ is considered true.
  - *Occur Check is omitted* ➔ *unsound.*
  - Prolog can be compiled or interpreted.
    - compiled – a miniature theorem prover is created.
    - interpreted – intermediate language executed by Warren Abstract Machine.

Generalized Resolution – an extension of propositional resolution to FOL.

- **Proof by contradiction** – *The goal is negated and added to the KB. If the empty clause {} is derived, the goal has been proved.*
  - Proofs derived in this way are *non-constructive*: they only indicate whether the query is true or false, not what variables make it so.
    1. Restrict query variables to a single binding and backtrack.
    2. Add an **answer literal** as a disjunction with the negated goal. The resulting non-constructive proof will have a disjunction of possible answers instead of an empty clause → multiple answers.
- Algorithm outline
  - Begin with propositionalization and conversion to CNF.
  - Binary resolution can be applied to clauses with complementary literals.
  - The Resolution Inference Rule
    - *First-order literals are* complementary *if one unifies with the negation of the other.*
    - **binary resolution rule**:
      $$\frac{l_1 \vee \ldots \vee l_k \quad m_1 \vee \ldots \vee m_n}{\text{SUBST}\left(\theta, l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n\right)}$$
      where $UNIFY\left(l_i, \neg m_j\right) = \theta$
    - **factoring** – reduces two literals to one if they are unifiable.
- Together, the *binary resolution rule* and *factoring* are complete.
- **Completeness of Resolution** – If S is an unsatisfiable set of clauses, then the application of a finite number of resolution steps to S will yield a contradiction.

Acting Under Uncertainty

- Agents almost never have access to all of their environment's information.
  - This qualification problem is due to 3 things
    1. Laziness – too much work to explore all possiblities
    2. Incomplete Theory
    3. Practical Limitiations – not all tests can be performed
  - Must be able to derive a plan that will work most of the time.
  - In uncertain environments, the rational decision depends on
    1. The relative importance of various goals
    2. The likelihood and degree of achievement of each goal
- Agent's knowledge provides a degree of belief ➔ Probability Theory
  - Probability Theory provides the degree to which an agent believes a statement given all plausible alternative situations that are indistinguishable from the current situation.
  - Probability Theory has the same ontological commitment as logic
    - *Facts either hold or do not hold in the world*
  - Probabilities depend on the evidence (precepts) the agent has aquired.
- Agents must have preferences towards the outcomes of different plans.
  - **Utility Theory** – Theory of preference based on an object's utility.
    - Preference indicates that different agents might have different goals so while goals might seem misguided, they are not necessarily irrational.
  - **Decision Theory** – Combination of Probability and Utility Theory.
    - **Principle of Maximum Expected Utility** - An agent is rational iff it chooses the action that yields the highest expected utility (averaged over all outcomes).
- **Belief State** – a representation of probabilities of all possible actual states of the world.

Probability Theory

- Language of Probability – ascribes degrees of belief to propositions.
  - **Random Variable** – refers to part of the world with initial unknown status
  - Domain – the values a random variable can take on.
    - Boolean – true/false
    - Discrete – countable, mutually exclusive, and exhaustive.
    - Continuous – uncountable
  - **Atomic Event** – a specific complete specification of the world about which the agent is uncertain.
    - The set of atomic events is mutually exclusive and exhaustive → forms a partition
    - Any atomic event entails true/false for every proposition.
    - Any proposition is equivalent to the disjunction of all atomic events that entail the proposition as true.
- **Prior Probability P(x)** – the degree of belief of proposition x in the absence of any evidence about other propositions.
  - **Probability Distribution**, P(X) – the vector of probabilities ascribed to each of the possible states of proposition X.
  - **Joint Probability Distribution**, $P(X_1,X_2,\dots,X_n)$ – the probabilities of all combinations of values of variables X_1, X_2, …, X_n.
  - **Full Joint Probability Distribution** – includes the complete set of variables for the environment.
  - **Probability Density Function**, p(x)dx – The probability of a continuous variable on the interval [x,x+dx] in the limit as dx → 0.
- **Conditional Probability P(x|y)** – the degree of belief in proposition x given the evidence y.
- **Product Rule**: $P(X,Y)=P(X\mid Y)P(Y)$
- **Kolmogrov's Axioms**
  1. All probabilities are between 0 and 1:
     $$0\le P(a)\le 1$$
  2. True propositions have probability 1; false propositions have 0:
     $$P(true)=1 \qquad P(false)=0$$
  3. Probability of a disjunction is given by
     $$P(a\vee b)=P(a)+P(b)-P(a\wedge b)$$
- Using these Axioms we can derive other important rules:
  - Let a discrete variable X have a domain $\{x_1,x_2,\dots,x_n\}$ or χ:
    $$\sum_{i=1}^{n}P(X=x_i)=1 \qquad \text{or} \qquad \int_{\chi}P(x)\,dx=1$$
  - The probability of a proposition is equal to the sum of the probabilities in which it holds; the set **e**(a) for proposition a.
    $$P(a)=\sum_{e_i\in\mathbf{e}(a)}P(e_i)$$

- In probability, statements do not refer directly to the world, but rather to an agent's belief about the world, so why can't agent's beliefs violate the probability axioms?
  - **de Finetti Theorem** – If Agent 1 expresses a set of degrees of belief that violate Kolmogorov's Axioms, then there is a combination of bets Agent 2 can place that guarantees that Agent 1 will lose money every time.
- Probability Philosophy
  - **Frequentist** – probabilities are results of repeated experiments.
  - **Objectivist** – probabilities come from a propensity of objects to act in a certain way.
  - **Subjectivist** – probabilities are a way of characterizing beliefs.

Probabilistic Inference – computation of posterior probability of query proposition from observed evidence.

- **Marginalization (Conditioning)** – variables other than the query variable are summed out in order to obtain the probability of the query variable:
  $$\text{joint:} \qquad P(Y)=\sum_z P(Y,z)$$
  $$\text{conditional:} \quad P(Y)=\sum_z P(Y\mid z)P(z)$$
- **Normalization –** introduction of constant α that normalizes the distribution to 1.

(Marginal) Independence – variables independent of each other can be factored:
$$P(X,Y)=P(X)P(Y) \qquad P(X\mid Y)=P(X)$$

- If a complete set of variables can be divided into independent subsets, then the full joint distribution can be factored into separated joint distributions on those subsets.

Bayes' Rule – application of product rule that allows diagnostic beliefs to be derived from casual beliefs:
$$P(Y\mid X)=\frac{P(X\mid Y)P(Y)}{P(X)} \qquad P(Y\mid X,e)=\frac{P(X\mid Y,e)P(Y\mid e)}{P(X\mid e)}$$

- Diagnostic knowledge is often more fragile than casual knowledge → direct casual (model-based) knowledge provides robustness needed for probabilistic systems to function in the real world.
- Conditional Independence – implies that two variables X,Y are independent given variable Z:
  $$P(X,Y\mid Z)=P(X\mid Z)P(Y\mid Z) \qquad P(X\mid Y,Z)=P(X\mid Z)$$

Naïve Bayes Model – a single cause Y directly influences a number of events X_i that are all conditionally independent given the cause:
$$P(Y,X_1,X_2,\dots,X_n)=P(Y)\prod_i P(X_i\mid Y)$$

Probabilistic Reasoning

- **Bayesian Network** (Belief Network, Probabilistic Network, Casual Network, or Knowledge Map)
  - A *directed acyclic graph* (DAG) representing the dependency structure amongst random variables thus providing a concise specification of any full joint probability distribution.
    - **Nodes** – the random variables of the problem (observed variables are shaded).
    - Each node X_i has a conditional probability distribution representing $P(X_i \mid parents(X_i))$. In the discrete variable case, this can be represented by a Conditional Probability Table (CPT).
    - **Directed Arcs** – represent the dependency of one random variable on another. In the Undirected case, represents interdependency between two variables.
  - The Bayesian Network captures conditional independence relationships in its edges.
  - Probabilities summarize a potentially infinite set of circumstances that are not explicit in the model but rather appear implicitly in the probability.
  - If each variable is influenced by at most k others and we have n random variables, we only need to specify $n*2^k$ probabilities instead of 2^n.
  - More general case of Bayesian Network is "Graphical Model".
- **Conditional Probability Table (CPT) -** describes the conditional probability of each value of the node for each *conditioning case*, or possible combination of the values of the parent nodes.
  - In general the size of the table is

    $$\#(X_i)\cdot\prod_{X_j\in parents(X_i)}\#(X_j)$$

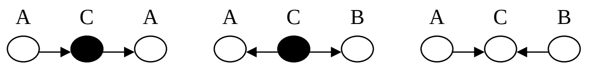
    where #() specifies the size of the domain of a variable.
  - The size of the table can be reduced since the total probability for each conditioning case must be 1.
- **Chain Rule:**
  - General:

    $$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_1, X_2, \ldots, X_{i-1})$$

  - In the case of a BN, if we number the nodes in topological order, the conditional terms in $P(X_i \mid X_1, X_2, \ldots, X_{i-1})$ are all predecessors of X_i and thus, conditional independence reduces this to $P(X_i \mid parents(X_i))$.
  - Chain Rule in BN:

    $$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid parents(X_i))$$

- Constructing a Bayesian Network
  - The parents of node X_i should be all nodes that directly influence X_i from the set of nodes $X_1, \ldots, X_{i-1}$.
  - The correct order in which to add nodes is to add the "root causes" first, then the variables they influence, and so on until leaves are reached.
  - Slight dependences may not be worth adding due to increased complexity.
  - Constructing Networks for a "Causal Model" will result in specifying fewer numbers, which are often easier to come up with.
- Independence in a Bayesian Network
  - A node is conditionally independent of all non-descendants given its parents.
  - A node is conditionally independent of all other nodes in network given its *Markov Blanket* (parents, children, and children's parents).
  - **d-separation** – two nodes X and Y in a BN are d-separated if every path between X and Y is blocked.
    - A path between X and Y is blocked if it has any of the following 3 cases for any 3 nodes along the path.
      - head-to-tail with intermediary observed: $A \perp B \mid C$
      - tail-to-tail with intermediary observed: $A \perp B \mid C$
      - head to head with neither the intermediary nor any of its descendants observed: $A \perp B \mid \varnothing$



- Canonical distributions: fit a standard pattern with an easily filled in CPT.
  - **deterministic node** – value is specified exactly by value of parents with no uncertainty.
  - **noisy-OR** – uncertain ability of each parent to cause the child to be true since the causal relationships may be inhibited.
    - Assumes all possible causes are listed (others can be grouped in a *leak node* for miscellaneous causes).
    - Assumes the inhibition of each parent is independent of the others.
    - Hence, we need only specify O(k) parameters instead of O(2^k) for k causes: the probability of inhibition of each of the causes.
  - other noisy-operators (MAX, AND, etc).

- Continuous Random Variables:
    - **discretization** – dividing variable's possible values into intervals.
    - **parameterization** – describing the variable's distribution by a finite set of parameters.
    - **hybrid BN** – a BN containing both discrete and continuous variables.
    - conditional distributions for continuous variables:
        - discrete parents' values are enumerated.
        - continuous parents' must be summarized in a distribution, for instance, the linear Gaussian distribution where mean varies linearly with parents' value and std dev is fixed: $\mu = ax + b$.
        - linear Gaussian has joint distribution is multivariate Gaussian over all variables. These are combined with discrete variables in conditional Gaussians.
    - conditional distributions for discrete variables with continuous parents.

Approximate Inference in Bayesian Networks

- Monte Carlo algorithms – algorithms that approximate a desired quantity through random sampling.
- Direct Sampling
- Rejection Sampling
- Likelihood Weighting
- Markov chain Monte Carlo (MCMC) – a sampling technique that settles into a *dynamic equilibrium* such that the long-term fraction of time spent in each state is exactly its posterior probability given certain conditions.
    - **Markov chain** – a structure that defines the probability of transitioning from the "current" state to the "next" state.
        - *transition probability* $q(\mathbf{x} \to \mathbf{x}')$ - the probability that the process transitions from state **x** to state **x'**.
        - ***ergodic*** – essentially every state much be reachable from every other and there can be no strictly periodic cycles.
        - *state distribution* $\pi_t(\mathbf{x})$ - the probability of being in state **x** at the *t*-th step of the Markov chain.
    - ***stationary distribution*** – a state distribution such that $\pi_t = \pi_{t+1}$

$$\forall \mathbf{x}' \qquad \pi(\mathbf{x}') = \sum_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \to \mathbf{x}')$$

        - This distribution is unique if the chain is *ergodic*.
        - A distribution is stationary if it satisfies the *detailed balance equation:*

$$\forall \mathbf{x}, \mathbf{x}' \qquad \pi(\mathbf{x}) q(\mathbf{x} \to \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \to \mathbf{x})$$

Probabilistic Reasoning Over Time

Modeling Uncertainty over Time

- Setting
    - X_t - a set of unobserved state variables at time *t*.
    - E_t - a set of observable evidence variables for time *t*.
    - *a:b* – denotes an interval from *a* to *b*.
- **Stationary Process** – process of change that is governed by laws that do not change over time.
- **Markov Assumption** – current state depends only on a *finite* history of previous states. Processes satisfying this assumption are *Markov Processes (Chains)*.
    - **transition model** – law describing how state changes over time.

$$P(X_t \mid X_{0:t-1}) = P(X_t \mid X_\alpha) \text{ where } \alpha \subseteq \{1 \dots t-1\}$$

    - **first-order Markov Process** – current state is solely dependent on the previous state
        - transition model: $P(X_t \mid X_{t-1})$
- We assume the evidence variables at time *t* depend only on the current state.
    - **sensor model** – law describing how the evidence depends on the state.

$$P(E_t \mid X_{0:t}, E_{0:t-1}) = P(E_t \mid X_t)$$

- prior probability for the initial state: $P(X_0)$
- complete joint

$$P(X_{0:T}, E_{1:T}) = P(X_0) \prod_{t=1}^{T} P(X_t \mid X_{t-1}) P(E_t \mid X_t)$$

Filter (monitoring) – the task of computing the *belief state* – the posterior distribution of the current state given all evidence; $P(X_T \mid e_{1:T})$.

- Recursive estimation – forward chaining.

$$P(X_t \mid e_{1:t}) \propto P(e_t \mid X_t) \sum_{X_{t-1}} P(X_t \mid X_{t-1}) \underbrace{P(X_{t-1} \mid e_{1:t-1})}_{\text{recursive estimate}}$$

$$f_{1:t} \propto FORWARD(f_{1:t-1}, e_t)$$

- When the state variables are discrete, this update is constant in space and time.
- *Likelihood* $P(e_{1:T})$ can be calculated by a likelihood message: $l_{1:t} = P(X_t, e_{1:t})$:

Prediction

Prediction – task of computing the posterior distribution over a *future* state, given all evidence; $P\left(X_{T+k} \mid e_{1:T}\right)$ where *k > 0*.

- This is equivalent to filtering without new evidence. Hence, we can easily derive the following update:
$$P\left(X_{T+k} \mid e_{1:T}\right) = \sum_{X_{t+k}} P\left(X_{T+k} \mid X_{T+k-1}\right) \underbrace{P\left(X_{T+k-1} \mid e_{1:T}\right)}_{\text{recursive estimate}}$$
- **stationary distribution** – The fixed point of the Markov process that is approached upon successive applications of the transition model.
  - **mixing time** – the amount of time required to reach stationarity.
  - Prediction is doomed to failure for future times more than a small fraction of the mixing time.

Smoothing (hindsight)

Smoothing (hindsight) – task of computing posterior distribution for a *past* state, given all evidence; $P\left(X_k \mid e_{1:T}\right)$ where $0 \le k < T$.

- Accounting for hindsight is done with an additional backwards message:
$$P\left(X_k \mid e_{1:T}\right) \propto \underbrace{P\left(X_k \mid e_{1:k}\right)}_{f_{1:k}} \underbrace{P\left(e_{k+1:T} \mid X_k\right)}_{b_{k+1:T}}$$
$$b_{k+1:T} = \sum_{X_{k+1}} P\left(e_{k+1} \mid X_{k+1}\right) P\left(X_{k+1} \mid X_k\right) b_{k+2:T}$$
- The time and space needed for each backward message are constant.
- Thus, the process of smoothing with respect to $e_{1:T}$ is *O(t)*.
- Thus, to smooth the whole sequence naively, requires $O(t^2)$.
- using dynamic programming the cost is only *O(t)* by recording results of forward filtering over the entire sequence while running the backward algorithm from *T* to 1 and use the smoothed message at each time step ➔ **forward-backward algo**.
  - space is now $O\left(|f|t\right)$

Most Likely Explanation

Most Likely Explanation – task of finding the sequence of states most likely to have generated a sequence of observations; $\arg\max_{x_{1:t}} P\left(x_{1:t} \mid e_{1:t}\right)$.

- most likely sequence must consider joint probabilities over all time steps.
- *there is a recursive relationship between most likely paths to each state X_{t+1} and the most likely paths to each state X_t.*
- Recursive formulation:
  - messages: $m_{1:t} = \max_{X_{1:t-1}} P\left(X_{1:t} \mid e_{1:t}\right)$
  - summation over X_t replaced by a maximization.
- Pointers are used to retrieve the most-likely explanation
- Viterbi algorithm has a space and time requirement of *O(t)*.

Hidden Markov Models (HMM)

Hidden Markov Models (HMM) – a temporal probabilistic model in which the state of the process is described by a *single discrete* random variable and transitions obey the Markov assumption.

- transition model: $T_{ij} = P\left(X_t = j \mid X_{t-1} = i\right)$
- observation model: $\left(\mathbf{O_t}\right)_{i,i} = P\left(e_t \mid X_t = i\right)$
  - *forward* message - $\mathbf{f}_{1:t+1} \propto \mathbf{O}_{t+1}\mathbf{T}^T\mathbf{f}_{1:t}$
  - *backward* message - $\mathbf{b}_{k+1:t} \propto \mathbf{T}\mathbf{O}_{k+1}\mathbf{b}_{k+2:t}$

Kalman Filters

Kalman Filters – a temporal probabilistic model for continuous state spaces under the Markov assumption and using linear Gaussian distributions to model the states.

- a *multivariate Gaussian* distribution can be specified completely by its mean $\boldsymbol{\mu}$ and its covariance matrix $\boldsymbol{\Sigma}$.
- In general, filtering with continuous or hybrid spaces generate state distributions whose representations grow without bound, but the Gaussian distribution is “well-behaved” since it has the following properties:
  1. If the current distribution $P\left(\mathbf{X}_t \mid \mathbf{e}_{1:t}\right)$ is Gaussian and the transition model $P\left(\mathbf{X}_{t+1} \mid \mathbf{x}_t\right)$ is linear Gaussian, then the predicted distribution is:
  $$P\left(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}\right) = \int_{\mathbf{x}_t} P\left(\mathbf{X}_{t+1} \mid \mathbf{x}_t\right) P\left(\mathbf{x}_t \mid \mathbf{e}_{1:t}\right) d\mathbf{x}_t$$
  2. If the predicted distribution is Gaussian and the observation (sensor) model is linear Gaussian, then conditioning on new evidence yields:
  $$P\left(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}\right) \propto P\left(\mathbf{e}_{1:t+1} \mid \mathbf{X}_{t+1}\right) P\left(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}\right)$$
- General formulation:
$$P\left(\mathbf{x}_{t+1} \mid \mathbf{x}_t\right) = N\left(\mathbf{F}\mathbf{x}_t, \boldsymbol{\Sigma}_x\right)\left(\mathbf{x}_{t+1}\right)$$
  - $\mathbf{F}$ and $\boldsymbol{\Sigma}_x$ describe the linear transition model & noise.
$$P\left(\mathbf{z}_t \mid \mathbf{x}_t\right) = N\left(\mathbf{H}\mathbf{x}_t, \boldsymbol{\Sigma}_z\right)\left(\mathbf{z}_t\right)$$
  - $\mathbf{H}$ and $\boldsymbol{\Sigma}_z$ describe the linear sensor model & noise.
- **Extended Kalman Filter (EKF)** – allows for limited nonlinearity in the model by modeling the system *locally* as linear in $\mathbf{x}_t$ in the region of $\mathbf{x}_t = \boldsymbol{\mu}_t$.
- **Switching Kalman Filter** – multiple Kalman Filters run in parallel each using a different model of the system. The resulting prediction is a weighted sum depending on how well each filter fits the current data.

Dynamic Bayesian Networks (DBN) – a Bayesian network that represents a temporal probability model by having state variables $\mathbf{X_t}$ replicated over time slices with the same conditional independences. We also have evidence at each time slice $\mathbf{E_t}$.

- *Constructing DBNs*
  - We need 3 broad types of information:
    1. a <u>prior distribution</u> on the initial variables: $\mathbf{P}(\mathbf{X_0})$
    2. a <u>transition model</u>: $\mathbf{P}(\mathbf{X_{t+1}} \mid \mathbf{X_t})$
    3. a <u>sensor model</u>: $\mathbf{P}(\mathbf{E_t} \mid \mathbf{X_t})$
  - In addition, we must specify a local and temporal topology of the nodes.
  - Issues we need to deal with:
    - *<u>Noise</u>*: we assume that our measurements are noisy, which we model with a **Gaussian error model**.
    - *<u>Failure</u>*: in the real-world, sensors fail…
      - ***transient failure model*** – allocates a probability that the sensor will return some nonsense value.
      - ***persistent failure model*** – describes how a sensor behaves under normal and failure conditions. There is a small probability of failure, but it models the fact that sensors tend to remain broken.
- *Exact Inference* – given a sequence of *n* observations, we simply construct the necessary DBN of *n* time slices – a process known as ***unrolling***.
  - An efficient process uses ***variable elimination*** before proceeding to the next time slice – this is equivalent to starting at $\mathbf{X_t}$ with a new initial distribution determined by our variable elimination.
  - *We cannot efficiently and exactly reason about the complex temporal processes represented by general DBNs.*
- *Approximate Inference*
  - Overcoming these blocks relies on 2 observations.
    - *We use the samples as approximate representations of the current state distribution.*
    - Generating the samples with naïve likelihood weighting will have ~0 probability of matching the evidence.
      - *We want to focus the set of samples on the high-probability regions of the sate space.*
  - **<u>particle filtering</u>** – leverages the above observations to make an efficient sampling algorithm that is ***consistent***. We begin with *N* samples from the prior distribution at time 0: $\mathbf{P}(\mathbf{X_0})$. Then we use an <u>update cycle</u>:
    - Each sample is propagated to next time slice by sampling the next state value $\mathbf{x_{t+1}}$ given $\mathbf{x_t}$ using the transition model $\mathbf{P}(\mathbf{X_{t+1}} \mid \mathbf{X_t})$.
    - Each sample is weighted by the likelihood it assigns to the new evidence: $\mathbf{P}(\mathbf{e_{t+1}} \mid \mathbf{x_{t+1}})$ from the sensor model.
    - A new population of *N* samples is *resampled*: each new sample is selected proportional to its likelihood weight.

Making Simple Decisions

decision-theoretic agent – an agent capable of making decisions in the face of uncertainty and conflicting goals. Unlike a goal-based agent (only views each state as 'good' or 'bad') a decision-theoretic agent makes a continuous measure of state quality.

Combining Belief and Desire under Uncertainty

- **utility function** – a function that assigns a single number to each state expressing its desirability. These are combined with the probability of each action's actual outcome to give expected utility of the action.
  - **expected utility**:
    $$EU[A \mid E] = \sum_i P(\text{Result}_i(A) \mid Do(A), E)\, U(\text{Result}_i(A))$$
    - *A* is the action, *E* is the evidence
  - **principle of maximum expected utility (MEU)** – a rational agent should choose the action that maximizes it's expected utility.

Utility Theory

1. $A \succ B$ — *A* is preferred to *B*.
2. $A \sim B$ — agent is indifferent between *A* and *B*.
3. $A \succsim B$ — agent prefers *A* to *B* or is indifferent.

- **Lottery** – a set of outcomes C_i with a probability p_i: $L = [p_1, C_1; p_2, C_2; \ldots; p_n, C_n]$
- **Axioms of Utility Theory**
  1. **Orderability** – for any two states, an agent must prefer one to the other or else be indifferent between them.
     $(A \succ B) \vee (A \prec B) \vee (A \sim B)$
  2. **Transitivity** –A preferred to B, & B preferred to C, then A preferred to C.
     $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
  3. **Continuity** – If B is between A and C in preference, there exists probability *p* for which the agent is indifferent between getting B for sure and a lottery that yields A with probability *p* and C with probability *1-p*.
     $A \succ B \succ C \quad \Rightarrow \quad \exists p \quad [p, A; 1-p, C] \sim B$
  4. **Substitutability** – an agent indifferent to A and B is indifferent to 2 more complex lotteries, 1 with each A and B.
     $A \sim B \quad \Rightarrow \quad [p, A; 1-p, C] \Rightarrow [p, B; 1-p, C]$
  5. **Monotonicity** – If 2 lotteries have the same outcomes, A and B, and agent prefers A to B, then it also prefers the lottery with higher probability of A.
     $A \succ B \quad \Rightarrow \quad p \geq q \Leftrightarrow [p, A; 1-p, B] \succsim [q, A; 1-p, B]$
  6. **Decomposability** – Compound lotteries can be decomposed:
     $[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C]$

- Utility
  1. **Utility Principle**
     $$U(A) > U(B) \quad \Leftrightarrow \quad A \succ B$$
     $$U(A) = U(B) \quad \Leftrightarrow \quad A \sim B$$
  2. **Maximum Expected Utility Principle**
     $$U([p_1, S_1; \ldots; p_n, S_n]) = \sum_i p_i U(S_i)$$

Multiattribute Utility Functions

- **multiattribute utility theory** – utility theory for outcomes involving two or more attributes: $\mathbf{X} = X_1, \ldots, X_n$.
- **strict dominance** – option 1 has higher value on all attributes than another option 2. Clearly the 1st option is chosen.
- **stochastic dominance** – if two actions A_1 and A_2 lead to probability distributions $p_1(x)$ and $p_2(x)$ on attribute X, then A_1 stochastically dominates A_2 on X if,
  $$\forall x \quad \int_{-\infty}^{x} p_1(y)\,dy \le \int_{-\infty}^{x} p_2(y)\,dy$$
  - *If A_1 stochastically dominates A_2, then for any monotonically nondecreasing utility function $U(x)$, the expected utility of A_1 is at least as high as the expected utility of A_2.*
  - **qualitative probabilistic networks** – algorithms for making rational decisions based on stochastic dominance alone.
- **representation theorems** – theorems that identify regularities in preference behavior; $U(x_1, \ldots, x_n) = f\left[f_1(x_1), \ldots, f_n(x_n)\right]$
- **preference independence** – attributes X_1 and X_2 are preferentially independent of X_3 if the preference between outcomes $\langle x_1, x_2, x_3 \rangle$ and $\langle x_1', x_2', x_3 \rangle$ doesn't depend on the value x_3.
- **mutual preferential independence (MPI)** – no attributes affect the way in which one trades off to the other attributes against each other
  - If attributes $X_1, \ldots, X_n$ are mutually preferentially independent, then the agent's preference behavior can be described as maximizing the function
    $$V(x_1, \ldots, x_n) = \sum_i V_i(x_i)$$
    where each V_i is a value function referring only to the attribute X_i.
  - **additive value function** – a multiattribute value function that is the sum of value functions for individual attributes.
- **utility-independence** – an extension of preference independence to lotteries. A set of attributes **X** is utility-independent of a set of attributes **Y** if preferences between lotteries on the attributes in **X** are independent of the particular values of the attributes in **Y**.
- **mutually utility-independent (MUI)** – each subset of a set of attributes is utility-independent of the remaining attributes.
  - **multiplicative utility function** – a function that can express the behavior of any agent exhibiting MUI in only *n* single-attribute utilities and *n* constants for *n* attributes.

Decision Networks

- **decision network** – a Bayesian network with additional node types for actions and utilities. Contains information about the agent's current state, its possible actions, the state resulting from the agent's action, and the utility of the state.
  - Structure
    - Chance nodes (ovals) – represent random variables each with a conditional distribution indexed by parent states. Parents can be other chance nodes or decision nodes.
    - Decision nodes (rectangles) – represent points where agent has a choice to make.
    - Utility nodes (diamonds) – represent the agent's utility function. Its parents are all variables directly affecting utility.
  - **action-utility tables** – A simplified form in which the action is connected directly to the utility thus making the utility node represent the expected utility… a compiled version.

- **algorithm to evaluate decision nets**
  1. Set the evidence variables of the current state
  2. For each possible value of the decision node
     a. set the decision node to that action
     b. calculate the posterior probabilities for the parents of the utility
     c. calculate the resulting utility
  3. Choose the action with highest resulting utility.

The Value of Information

- **information value theory** – theory describing what information is best to acquire in order to make a decision… *one of the most important parts of decision making is know what questions to ask.*
  - Tests/Questions can be expensive/hazardous but the information they yield may overweigh those risks.
  - sensing actions – actions preformed in order to acquire information
  - **value of information** – the value of a piece of information is the difference between the expected utility between the best possible actions before and after information is acquired.
    - *Information has value to the extent that it is likely to cause a change of plan and to the extent that the new plan will be significantly better than the old one.*
- **value of perfect information (VPI)** – value of information assuming exact evidence E_j of some random variable is obtained:

$$VPI_E\left(E_j\right)=\left(\sum_k P\left(E_j=e_{jk}\right)EU\left(\alpha_{e_{jk}} \mid E, E_j=e_{jk}\right)\right)-EU\left(\alpha \mid E\right)$$

  - Properties
    - VPI is non-negative: $\forall j, E \quad VPI_E\left(E_j\right)\geq 0$
    - VPI is not additive (in general): $VPI_E\left(E_j,E_k\right)\neq VPI_E\left(E_j\right)+VPI_E\left(E_k\right)$
    - VPI is order-independent: $VPI_E\left(E_j,E_k\right)=VPI_E\left(E_j\right)+VPI_{E,E_j}\left(E_k\right)=VPI_{E,E_k}\left(E_j\right)+VPI_E\left(E_k\right)$

- **Information-Gathering Agent**
  - agent is **myopic** since the VPI formulation only accounts for the effect of evidence E_j given that only that E_j is observed without including the possibility that future evidence may make the observation of E_j more valueable.
  - To consider all possible sequences of information requests would require *conditional planning*.

Making Complex Decisions

Sequential Decision Problems – an agent's utility depends upon a sequence of decisions.

- **transition model *T(s,a,s')*** – the probability of transitioning from state *s* to *s'* due to action *a*.
  - **Markovian** – the probability of reaching *s'* from *s* depends only on state *s* and not on the entire history of earlier states.
- **environment history** – the sequence of states on which utility depends.
  - In state *s*, the agent receives a **reward** of *R(s)*.
  - environment history is simply a sum of rewards received.
- **Markov Decision Process** – a fully observable environment with a Markovian transition model and additive rewards.
  - Components
    1. Initial state S_0
    2. Transition model *T(s,a,s')*
    3. Reward function *R(s)*
- **policy π** - a plan of what action to take in a given state: $a_t=\pi\left(s_t\right)$
  - The quality of the policy is measured as the expected utility of all possible environment histories generated by a policy.
  - **optimal policy π^*** - a policy that yields the highest expected utility.
- Optimality for a sequential decision process
  - Is the task episodic or continual?
    - **finite horizon** – the decision process goes on for a fixed time *N*.
      - *With a finite horizon, the optimal action in a given state could change over time* → optimal policy is **nonstationary**.
    - **infinite horizon** – the process continues indefinitely.
      - optimal policy is **stationary**.
  - How to calculate the utility of state sequences?
    - **multiattribute utility theory** – each state s_i is viewed as an attribute of the state sequence $\left[s_0,s_1,\ldots\right]$.
    - **stationary preference assumption** – if two state sequences, $\left[s_0,s_1,s_2,\ldots\right]$ and $\left[s_0',s_1',s_2',\ldots\right]$, begin with the same state, $s_0=s_0'$, then the preference order of the two sequences should be the as sequences $\left[s_1,s_2,\ldots\right]$ and $\left[s_1',s_2',\ldots\right]$ are ordered.

- Under stationarity, there are only two possible utilities:
  - **Additive Rewards**

    $$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{T} R(s_t)$$

  - **Discounted Rewards**

    $$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{T} \gamma^t R(s_t)$$

    - γ is discount factor between 0 and 1 indicative of preference between current and future rewards
      - $\gamma \approx 0$: future rewards are insignificant
      - $\gamma \approx 1$: equivalent to additive rewards
      - equivalent to an interest rate of $(1/\gamma)-1$.
- How to calculate utility when history is infinite.
  1. For discounted rewards with a maximum reward $R_{\max}$ and $\gamma < 1$, utility is still finite:
     i. $U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \le R_{\max}/(1-\gamma)$
  2. **Proper policy** – the agent is guaranteed to get to a terminal state eventually, so infinite sequences can be ignored.
  3. Compare infinite sequences in terms of average reward per time step.

- How to choose between policies?
  - In general, a policy π generates a whole range of possible state sequences, each with a certain probability determined by the transition model.
  - Value of policy is the expected sum of discounted rewards.
  - optimal policy:

    $$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right]$$

Value Iteration – an algorithm to calculate the optimal policy by calculating the utility of each state and using state utilities to select an optimal action in each state.

- *Utility* of a state *s* by following policy π:

  $$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right]$$

- *True Utility* of state *s*: $U(s) = U^{\pi}(s)$
- From *Maximum Expected Utility (MEU)* principle, we have an optimal policy:

  $$\pi^* = \arg\max_{a} \sum_{s'} T(s,a,s') U(s')$$

- **Bellman Equation**

  $$U(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s,a,s') U(s')$$

  - *The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.*
  - If there are *n* possible states, there will be *n* Bellman equations in *n* unknowns. Unfortunately they are nonlinear.
- **Iterative Approach** – calculates the utility of each state on the basis of the utility of their neighbors → propagates information through the state space via local updates.
  - **Bellman Update**: $U_{i+1}(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s,a,s') U_i(s')$
  - Converges to a unique solution whose corresponding policy is optimal.
    - **contraction** – a unary function that, when applied to two different values in turn, causes their corresponding output values to be "closer together", by at least some constant amount, than the original argument were.
      - has a single fixed point
      - When applied to any argument, the output must be closer to the fixed point than the original argument was.
    - The Bellman update can be viewed as an operator *B* applied to the set of utilities: $U_{i+1} = BU_i$
    - **max norm**: $\|U\| = \max_{s} |U(s)|$
      - the distance between 2 vectors is the maximum distance between any two corresponding elements.
    - The Bellman update is a contraction by a factor γ on the space of utility vectors. That is, let U_i and U_j be two utility vectors, then

      $$\|BU_i - BU_j\| \le \gamma \|U_i - U_j\|$$

    - if $\|U_i - U\|$ is the *error* in estimate U_i.
      - value iteration converges exponentially in the number of iterations. Unfortunately, this is degraded by $\gamma \approx 1$
      - If R_{max} is the bound on the rewards, then the number of iterations required to reach an error of at most ε is,

        $$N = \left\lceil \frac{\log(2R_{\max}) - \log(\varepsilon(1-\gamma))}{-\log(\gamma)} \right\rceil$$

    - If the update is small, then the corresponding error is small

      $$\|U_{i+1} - U_i\| < \varepsilon(1-\gamma)/\gamma \quad \Rightarrow \quad \|U_{i+1} - U\| < \varepsilon$$

Policy Iteration – an alternative way to find optimal policies by alternating between 2 steps: policy evaluation and policy iteration.

- **Policy Evaluation** – given a policy π_i, calculate $U_i = U^{\pi_i}$.
  - since policy is chosen, Bellman equations become linear:
    $$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U(s')$$
  - Thus, given *n* states, this can be solved using linear algebra in $O(n^3)$.
- **Policy Iteration** – calculate a new MEU policy π_{i+1} based on maximizing U_i.
- **Modified Policy Iteration**
  - Use simplified Bellman updates repeated *k* times for the evaluation step:
    $$U_{i+1}(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$
  - Often more efficient than either value iteration or policy iteration
- **Asynchronous Policy Iteration** – pick any subset of states and apply either policy evaluation or policy iteration to that subset.
  - Under certain conditions on the initial policy and utility function, will still converge to optimal policy
  - Allows freedom to choose what states to work on.

Partially Observable MDPs (POMDP) – an MDP agent operating in a partially observable environment where the optimal action is state *s* also depends on how much the agent knows in state *s*. Defined in terms of a *transition model* $T(s,a,s')$, a *reward function* $R(s)$, and an *observation model* $O(s,o)$ that specifies the probability of perceiving observation *o* in state *s*.

- **belief-state *b*** – the set of actual states the agent might be in, represented by a probability distribution over all states.
  - If *b(s)* was the previous belief state when the agent executes action *a* and observes observation *o*, the new belief state is
    $$b'(s') \propto O(s',o) \sum_{s} T(s,a,s') b(s)$$
- *The optimal action depends only on the agent's current belief state* → a mapping $\pi^*(b)$ from belief states to actions.
- *Solving a POMDP on a physical state space can be reduced to solving an MDP on the corresponding belief state space with transition model* τ *and rewards* ρ.
  - The probability of an observation *o* given action *a* in belief state *b* is,
    $$P(o \mid a,b) = \sum_{s'} O(s',o) \sum_{s} T(s,a,s') b(s)$$
  - The *probability of transitioning* from belief state *b* to belief state *b'* via action *a* is,
    $$\tau(b,a,b') = \sum_{o} P(b' \mid o,a,b) \sum_{s'} O(s,o) \sum_{s} T(s,a,s') b(s)$$
  - The *reward function* for belief states is, $\rho(b) = \sum_{s} b(s) R(s)$
  - Finding even approximately optimal POMDPs is difficult – PSPACE-hard

Learning from Observations

Forms of Learning

- Recall, in designing a learning agent, there is a <u>performance element</u> (decides on actions) and a <u>learning element</u> (modifies performance element to make better decisions).
- Issues in design:
  1. Which components are to be learned
  2. What type of feedback is available for learning
     - (a) **<u>supervised learning</u>** – learning a function using examples of inputs and corresponding outputs.
     - (b) **<u>unsupervised learning</u>** – learning patterns in input without any corresponding output values.
     - (c) **<u>reinforcement learning</u>** – learning by means of reinforcement (positive/negative rewards).
  3. What representation is used for learned components
  4. How to incorporate available prior knowledge

Inductive Learning

- **<u>pure inductive inference (induction)</u>** – given a collection of examples of *f*, return a **hypothesis** *h* that approximates *f*.
  - **generalization** – a good hypothesis will predict unseen examples correctly.
  - **hypothesis space H** – the set of hypotheses to consider.
  - **consistent hypothesis** – a hypothesis that agrees with all observed data.
  - **realizable** – a learning problem in which the true function is contained within the hypothesis space.
    - prior knowledge can be used to define the hypothesis space to make the problem realizable.
- **Complexity** (of hypothesis) vs. **Expressiveness** (of hypothesis space
  - How to choose between multiple consistent hypotheses?
    - **Ockham's razor** – prefer the simplest hypothesis consistent with data.
    - *For nondeterministic functions, there is a tradeoff between complexity of the hypothesis and the degree of fit to the data.*
  - An expressive hypothesis space makes it possible to find a simple hypothesis to fit the data. Restricting the expressiveness forces consistent theories to be complex.
  - *There is a tradeoff between the expressiveness of a hypothesis space and the complexity of finding simple, consistent hypotheses within it.*

Learning Decision Trees

decision tree – a tree of rules for classifying a set of attributes (inputs) describing the object, in which the tree represents the series of tests used to make a classification. Decision tree has nice property that its learned function is human-readable.

- types of learning
  - **classification** – learning a discrete valued function
  - **regression** – learning a continuously valued function
- expressiveness of a decision tree
  - some functions require tree exponentially large in # of inputs.
    - Parity function – returns 1 iff an even number of inputs are 1.
    - Majority function – returns 1 iff more than half of inputs are 1.
  - Problem is fundamental
    - For n-attributes, the truth table contains 2^n rows. Hence, some functions will always require at least that many bits to represent.
    - Moreover, for 2^n bits, there are 2^{2^n} possible functions!
- smallest possible decision tree
  - We want to find smallest possible consistent decision tree (intractable). Instead, we use greedy heuristic always selecting 'most important attribute' as next in tree.
- *decision tree learning algorithm*
  - Given a current leaf in the tree:
    - If there both positive and negative examples, choose the 'best' attribute to split them
      - If no attributes remain, noise is present. A simple way to decide is take a majority vote of remaining examples.
    - If there are only 1 type of example remaining, decide based on this remaining label.
    - If there are no examples remaining, no observation of this scenario has been observed → use majority vote of parent node.
- *choosing best attributes* – an attribute is chosen at each leaf of the tree to divide up all training examples relevant to that leaf thus branching the tree downward. The attribute is chosen according to which maximizes information gain.
  - *expected amount of information* – a concept from information theory in which information content of an attribute is measured in **bits**.

$$I\left(P(v_1),\ldots,P(v_n)\right)=\sum_{i=1}^{n}-P(v_i)\log_2 P(v_i)$$

  - *number of bits required* to classify after testing attribute A:

$$\text{Remainder}(A)=\sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I\left(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i}\right)$$

    where there are *p* positive examples and *n* negative examples in the remaining training set and the *i*-th value of attribute A has p_i positive and n_i negative examples at that level of the tree.
  - **information gain**

$$Gain(A_t)=Gain(A_{t-1})-\text{Remainder}(A_t) \qquad Gain(A_0)=I\left(\frac{p}{p+n},\frac{n}{p+n}\right)$$

    where *t* is the depth of the current branch.
- *assessing performance of learning*
  - **Algorithm**
    - Collect set of examples
    - Divide set into disjoint sets: **training** and **test** randomly
    - Apply learning to **training** set – hypothesis *h*
    - Measure percentage of examples in **test** correctly classified by *h*.
    - Continue process
  - **learning curve** – the performance of the algorithm plotted against the number of examples used to train it. If there exists a pattern in the data being learned, the curve should approach 1 as the number of examples grows.
  - **peeking** – often test data is used to tune the algorithm, and reusing this tainted test set is invalid since hypothesis was selected on basis of test-set performance!

Statistical Learning Methods

Statistical Learning

- **evidence** – data; instantiations of random variables describing the domain.
- **hypothesis** – a probabilistic theory of how the domain works.
- **Bayesian Learning** – makes predictions on the basis of calculating the probability of each hypothesis given the data. Prediction uses all the hypotheses weighted by their probabilities.
    - Probability of a hypothesis given data **d**:
      $$P(h_i \mid \mathbf{d}) \propto P(\mathbf{d} \mid h_i) P(h_i)$$
    - Probability of a data X given previous data **d** (hypotheses marginalized)
      $$P(X \mid \mathbf{d}) = \sum_i P(X \mid h_i) P(h_i \mid \mathbf{d})$$
        - hypothesis prior: $P(h_i)$
        - data likelihood: $P(\mathbf{d} \mid h_i)$
    - Observations assumed independently and identically distributed (IID):
      $$P(\mathbf{d} \mid h_i) = \prod_j P(d_j \mid h_i)$$
    - The true hypothesis eventually dominates the Bayesian prediction! However, it is often difficult to achieve optimality.
- **Maximum a Posteriori (MAP) hypothesis**: an approximation of Bayesian optimality through the most probable hypothesis;
  $$h_{MAP} = \arg\max_{h_i} P(h_i \mid \mathbf{d})$$
  $$P(X \mid \mathbf{d}) \approx P(X \mid h_{MAP})$$
    - **overfitting** – a consequence of an over-expressive hypothesis space. Hence, Bayesian and MAP learning use the prior to penalize complexity; more complex hypotheses have lower priors.
    - For deterministic hypotheses, $P(\mathbf{d} \mid h_i)$ is 1 if h_i is consistent with the data and 0 otherwise. *Hence, h_{MAP} is the simplest logical theory that is consistent with the data… an embodiment of Ockham's razor.*
    - h_{MAP} is equivalent to minimizing $-\log_2 P(\mathbf{d} \mid h_i) - \log_2 P(h_i)$. From information theory, $-\log_2 P(\mathbf{d} \mid h_i)$ is the number of bits required to encode the data given the hypothesis and $-\log_2 P(h_i)$ is the number of bits required for the hypothesis ➔ minimize data compression!
- **Minimum Description Length (MDL)** – attempts to minimize the size of hypothesis and data encodings directly rather than work with probabilities.
- **Maximum-Liklihood (ML) hypothesis** – a simplification of MAP learning in which a uniform prior is given over all hypotheses: h_{ML}.

Learning with Complete Data

- **parameteric learning** – finding numeric parameters for a probability model with fixed structure.
- **complete data** – each data point contains values for every variable of the model.
- **ML discrete parameteric learning** – object is to find the parameters θ that maximize the *likelihood*, or often, the *log likelihood* by the following method;
    1. Write down an expression for the likelihood of the data as a function of the parameters
    2. Write down the derivative of the log likelihood w.r.t. each parameter.
    3. Find the parameter values that make the derivative 0.
- *When the data set is small, some events will not have been observed so the h_{MAP} gives them a probability of 0.*
    - In practice, often these events are given a count of 1 so they aren't excluded from the hypothesis.
- *With complete data, the ML-parametric learning problem for a Bayesian network decomposes into separate learning problems; one per parameter.*
- **Naïve Bayes Models** – model in which 'class' variable *C* is the root of 'attribute' variables X_i. Thus, the probability of each class is given as,
  $$P(C \mid x_1, \ldots, x_n) \propto P(C) \prod_i P(x_i \mid C)$$
    - Naïve Bayes scales well: for *n* Boolean attributes, there are *2n+1* parameters.
    - No search for h_{MAP} is required.
    - No difficulty with noisy data and can give probabilistic predictions.

Reinforcement Learning

Reinforcement Learning (RL) – the task of using observed *rewards* to learn a (approximately) optimal policy for an environment by choosing an action that will maximize the *expected reward* given the current observed state of the agent.

- **Reward (Reinforcement)** – feedback that differentiates between *good* and *bad* outcomes; thus allowing the agent to make choices.
- RL builds on the studies of animal psychologists in differentiating between *reward* and other sensory inputs.
- Unlike MDPs, RL agents assume no prior knowledge of either the environment or the reward function.
- *In a sense, the RL task encompasses all of AI*: an agent is placed in an environment where it must behave successfully.
- Three types of agent designs:
    - ***utility-based agent*** – learn a utility function for states, which the agent will use to select actions in order to maximize expected utility.
        - requires an environment model to map actions to successor states.
    - ***Q-learning agent*** – learns a utility function on the state-action pairs; a so-called Q-function.
        - able to compare actions without knowing their outcomes.
        - without knowing action outcome, look ahead is not possible.
    - ***reflex agent*** – learns a policy that maps states to actions.

Passive Reinforcement Learning – the agent's has a fixed policy π: perform action $\pi(s)$ in state *s*. This is similar to *policy iteration*, but we lack the *transition model* $T(s,a,s')$ and the *reward function* $R(s)$. Thus, the agent performs a set of **trials** and uses the observed rewards to estimate the expected utility of each state $U^{\pi}(s)$. Starting in state *s* we want to estimate the (discounted) expected reward from future states:

$$U^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t) \mid \pi, s_0 = s\right]$$

- **Direct Utility Estimation** – the utility of a state is the expected reward starting from that state, so each **trial** is a sample for each state visited.
    - In this setting, the problem becomes a supervised learning problem of mapping state to value ➔ an inductive learning problem.
    - This *Monte-Carlo* approach assumes independence of the utility function between states. This ignores the fact that utilities are coupled in the Bellman equations! Thus, this approach does not **bootstrap**!
        - Without bootstrapping, invaluable information for learning is lost and thus the technique converges very slowly.
- **Adaptive Dynamic Programming (ADP)** – as the agent moves through the environment, the transition model is estimated and the MDP for the corresponding model estimate is solved incrementally using dynamic programming.
    - Learning the environment:
        - The transition model $T(s,a,s')$ is estimated from the frequency from state *s* to state *s'* via action *a*.
    - The MDP is solved using <u>policy iteration</u> or <u>modified policy iteration</u>.
    - ADP is intractable for large state spaces.
    - approximate ADP – bounds the number of adjustments per transition.
        - *prioritizing sweep heuristic* – prefers to adjust states whose successors have recently had a large utility adjustment.
- **Temporal Difference (TD) Learning** – a mixture of sampling and constraint bootstrapping in which the values of the observed states are modified to reflect the constraints between states given by the MDP.
    - TD equation: given a learning rate α we update the expected utilities:
      $$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha\left(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)\right)$$
      The TD equations converges to the MDP equilibrium even though only visited states are considered – the frequency of visits to a state are a substitute for the explicit transition model.
    - TD is an efficient approximation of ADP:
        - the utility function is updated by local adjustments.
        - TD only adjusts w.r.t. the observed transition and only makes a single update per transition.

Active Reinforcement Learning – policy is no longer fixed; active agents must decide on actions to take.

- **Exploration**
    - ***greedy agent*** – follows the current "optimal policy" according to the current estimates of the utility of each state.
        - unlikely to converge to the "optimal policy" since neglected states have poor estimates of their utility functions.
    - Trade-off between exploration and exploitation
        - ***exploitation*** – utilizing current knowledge to perform actions that maximize rewards.
        - ***exploration*** – trying suboptimal actions with the hope of improving our current estimates for the utility function.
        - <u>*n*-armed bandit</u> – a slot machine with *n*-levers – gambler must choose to exploit the lever with highest payoff or explore other levers to better estimate their payoff.
            - *Gittins index* – a measure of this tradeoff in independent situations (doesn't extend to sequential decisions).

- **Greedy in the limit of infinite exploration (GLIE**) – exploration schemes that are eventually optimal.
    - simple GLIE scheme – try a random action with probability $1/t$; otherwise, perform the optimal action.
    - <u>optimistic utility estimates</u> that favor unexplored states:
      $$U^+(s) = R(s) + \gamma \max_a f\left(\sum_{s'} T(s,a,s')U^+(s'), N(a,s)\right)$$
        - U^+ is the optimistic utility function
        - $N(a,s)$ is the # of times action *a* is done in state *s*.
        - *exploration function* $f(u,n)$ - trade-off between greed and curiosity that must increase in *u* and decrease in *n*. e.g.
          $$f(u,n) = \begin{cases} R^+ & n < N_e \\ u & otherwise \end{cases}$$
        - policy converges quickly while utility estimates converge slowly, but all we need is correct policy!

- **Action-Value Function**
    - *TD-learning* can be adapted to the active setting simply by choosing an action based on the current *U* estimate via 1-step look-ahead. However, we still have to learn the environment model to select actions.
    - **Q-learning** – learns an action-value representation instead of utilities.
        - Q-values: $U(s) = \max_a Q(a,s)$
        - ***model-free*** – does not require an environment model for learning or action selection.
        - *Bellman equations for Q-values*:
          $$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s')$$
        - *TD Q-learning*: (model-free)
          $$Q(a,s) \leftarrow Q(a,s) + \alpha\left(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)\right)$$
            - TD doesn't enforce consistency between values by using the model so it learns slower!
    - **knowledge-based approach** – method of representing the agent function by building a model of some aspects of the agent's environment.
        - Has definite advantages over model-free learning agents as the environment becomes more complex.

Generalization in Reinforcement Learning – we now consider methods for scaling RL to worlds with enormous state spaces. Standard tabular RL is impractical since the table has one entry per state and since most states would be visited rarely.

- **function approximation** - representing the value function in (approximate) non-tabular forms, e.g., a linear combination of ***features*** of the state:
  $$\hat{U}_\theta(s) = \theta_1 f_1(s) + \ldots + \theta_n f_n(s)$$
    - Thus we want to learn the parameters $\theta_1, \ldots, \theta_n$ to best approximate the value function. *Note*: features can be non-linear in the state variables.
    - *Function approximation allows the agent to broadly generalize between many states via states' common attributes.*
    - Unfortunately, the best utility function may be a poor estimate!
    - Online learning updates (**Widrow-Hoff** or **Delta Rule**): uses derivatives of squared error to update parameters.
      $$\theta_i \leftarrow \theta_i + \alpha\left(u_j(s) - \hat{U}_\theta(s)\right)\frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$
        - TD update: $\theta_i \leftarrow \theta_i + \alpha\left(R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)\right)\frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$
        - Q: $\theta_i \leftarrow \theta_i + \alpha\left(R(s) + \gamma \max_{a'} \hat{Q}_\theta(a',s') - \hat{Q}_\theta(a,s)\right)\frac{\partial \hat{Q}_\theta(a,s)}{\partial \theta_i}$
    - These updates converge to the optimal estimate for linear functions, but can wildly diverge for non-linear ones.
- Function approximation can also be used to estimate the environment model:
    - in *observable models*, this is a supervised task.
    - in *partially-observable* models, DBNs with latent variables can be used.

Robotics

Robots – physical agents that perform tasks manipulating the physical world using *effectors* based on observations from their *sensors*. Robots must deal with environments that are partially observable, stochastic, dynamic, and continuous.

- ***Sensors*** – instruments for perceiving the environment.
    - **passive** – capture signals generated from other sources in the environment.
    - **active** – send energy into the environment and observe the response; e.g. sonar. This technique runs the risk of interference from other sensors.
    - **range finders** – measure distance to nearby objects.
    - **imaging sensors** – cameras that provide images of the environment. Vision techniques can then build models of the environment.
    - **proprioceptive sensors** – inform the robot about its internal state.
        - e.g. shaft decoders for *odometry*; force and torque sensors.
- ***Effectors*** – means by which robots move and change body shape or otherwise manipulate their environment.
    - **kinematic state (pose)** – 3 location coordinates (x,y,z) + 3 angular coordinates (*yaw*, *roll*, *pitch*).
    - **dynamic state** – 1 additional dimension for the change in each dimension of the kinematic state.
    - **degrees of freedom (DOF)** – the number of independent directions in which an effector (robot) can move. Hand has 6 degrees of freedom.
        - **controllable DOF** – the DOF that the robot can directly change.
        - **effective DOF** – the DOF that the robot indirectly has (over time).
        - **holonomic** – the controllable DOF is the same as effective DOF.
        - **nonholonomic** – the effective DOF is greater than controllable.

Perception

- Kalman filters, HMMs, DBNs can be used for representing transition and sensor models in a partially observable environment through belief states.
    - *filtering* – task of updating the belief state. In a continuous environment, the recursive update equations are modified by replacing summations with integrations.
    - transition (motion) model - $P(X_{t+1} \mid X_t, a_t)$
    - sensor (observation) model - $P(z_{t+1} \mid X_{t+1})$

Localization

Localization – problem of determining where objects are in the robot's world.

- **tracking problem** – initial pose of object is known and we wish to track its changing pose over time.
- **global localization** – initial pose of object is unknown and we wish to figure out where the robot is.
- **kidnapping problem** –the object to be localized is "kidnapped".

- Motion Model: A crude approximation is to make each action into a instantaneous specification of translational velocity v_t and rotational velocity ω_t over a small interval of time Δt:

$$\hat{X}_{t+1} = f\left(\hat{X}_t, \underbrace{v_t, \omega_t}_{a_t}\right) = \hat{X}_t + \begin{pmatrix} v_t \Delta t \cos\theta_t \\ v_t \Delta t \sin\theta_t \\ \omega_t \Delta t \end{pmatrix}$$

    - Uncertainty is added by Gaussian noise with a covariance Σ_x:

$$P(X_{t+1} \mid X_t, a_t) = N(\hat{X}_{t+1}, \Sigma_x)$$

- A sensor model.
    - **landmarks** – stable recognizable features of the environment.
    - Suppose the robot is at location $(x_t^R, y_t^R, \theta_t^R)$ and an observed landmark has know location (x_i, y_i) then the *range* and *bearing* are:

$$\hat{\mathbf{z}}_{i,t} = h(x_t^R, y_t^R, \theta_t^R) = \begin{pmatrix} \sqrt{(x_t^R - x_i)^2 + (y_t^R - y_i)^2} \\ \arctan\left(\frac{y_t^R - y_i}{x_t^R - x_i}\right) - \theta_t^R \end{pmatrix}$$

    - Assume there is Gaussian noise with covariance Σ_z:

$$P(\mathbf{z}_{i,t} \mid X_t) = N(\hat{\mathbf{z}}_{i,t}, \Sigma_z)$$

    - Now assuming the errors for different beam directions are IID and we have *M* beam directions to a landmark:

$$P(\mathbf{z}_t \mid X_t) \propto \prod_{j=1}^{M} \exp\left\{-(z_j - \hat{z}_{j,t}) / 2\sigma^2\right\}$$

- **Monte Carlo Localization** – localization via a particle filter represent each belief state as a single particle corresponding to possible states.
- **Kalman Filter Localization** – localization where the posterior (belief) $P(X_t \mid z_{1:t}, a_{1:t-1})$ is represented by a Gaussian.
    - Gaussian beliefs are *closed* only under assumption that our motion model *f* and measurement model *h*, are linear.
    - **Extended Kalman Filter (EKF)** – a Kalman filter than linearizes *f* and *h* by a first-degree approximation from the Taylor expansion.
- **data association** – problem of identifying which landmark is which.

Mapping – robot builds a map of its environment

- **simultaneous localization and mapping (SLAM)** – robot constructs map without knowing where it is.

$$P\left(X_{t+1}, M \mid z_{1:t+1}, a_{1:t}\right) \propto$$
$$P\left(z_{t+1} \mid X_{t+1}, M\right) \int P\left(X_{t+1} \mid X_t, a_t\right) P\left(X_t, M \mid z_{1:t}, a_{1:t-1}\right) dX_t$$

  - This is essentially the same flavor as localization with the caveat that the space of poses and mappings is much larger.
- EKF approach uses Gaussian posterior with higher dimensional mean.
  - As the robot explores, it gradually loses certainty of where it is; this can be represented by *error ellipses*.
  - When landmarks are observed, uncertainty in current and previous locations is drastically reduced.
    - There is still a problem with uncertainty in landmark estimation and identification.
  - Since map *M*'s size and the number of landmarks is not known in advance, the parameter space may change adding new elements to the posterior.

Planning to Move

- **point-to-point motion** – problem of delivering the robot or its effector to a desired target location.
- **compliant motion** – motion constrained to being always against an obstacle.
- ***workspace representation*** – representing the robot in terms of the location of its movable parts in the coordinate system of the external world.
  - **linkage constraints** – constraints on the space of attainable workspace.
  - The robot's task is often formulated in workspace coordinates of the object's it seeks to manipulate → *inverse kinematics*.
- ***configuration space*** – space of robot states defined by its degrees of freedom and those of its various effectors. Each configuration of the robot is a point.
  - **kinematics** – transforming configuration coordinates into workspace coordinates via a series of coordinate transformations.
  - **inverse kinematics** – transforming workspace coordinates into configuration space. This is generally hard and ambiguous.
  - **free space** – space of all configurations that a robot can obtain.
  - **occupied space** – space of unattainable configurations.
- **path planning problem** – finding a path between configurations of the robot.
  - **cell decomposition** – free space is decomposed into a finite number of contiguous regions, cells, typically a regularly spaced grid.
    - Problems:
      - *mixed cells* – cells that overlap free and occupied space.
        - if included, path planner may be *unsound*
        - if excluded, path planner may be *incomplete*.
      - the # of cells is exponential in the dimension, *d*, of space.
      - the path's may have sharp, unattainable changes.
  - **potential field** – function that is higher further from boundaries.
    - Trade-off: maximize clearance while minimizing path length.
  - **Skeletonization** – reduces free space to a *skeleton* simplifying paths.
    - **skeleton** - a 1D representation based on a Voronoi graph, or the path that maximizes the distance between 2 or more obstacles.
  - **Probabilistic Roadmap** – skeletonization by randomly sampling configurations and keeping those that are attainable and connecting points that are easy to transition between.
    - Technically incomplete since a bad random sampling is possible.
    - Scales to high dimensions and tends to produce safer paths.

Planning Uncertain Moves – problem is that uncertainty arises from partial observablity and stochasticity of robot's actions.

- Common practice is to use *most likely state*, however, in many cases, uncertainty is too large and robot's true position is not necessarily the ML estimate.
- For fully observable with uncertainty in state transitions, an *MDP* is good.
  - **navigation function** – a policy of the *MDP* obtained by computing the gradient of the value function.
- In partially observable situations, a *POMDP* seems appropriate.
  - Belief states represent what agent knows and what it doesn't
    - *information gathering* – actions used to reduce uncertainty.
  - *Unfortunately it is not known how to apply POMDPs to continuous space.*
  - Hence, we attempt to minimize pose uncertainty.
    - **costal navigation** – robot stays close to known landmarks to decrease uncertainty and as new landmarks are observed, it is able to explore new territory.
- **robust methods** – assume that each aspect of the problem has a bounded amount of uncertainty but does not assign probabilities. Thus, the idea is to build plans that will work irregardless of actual values.
  - In the extreme, this becomes *conformant planning*.
  - **fine-motion planning** – moving a robot (arm) in close proximity to an object. Solutions are plans guaranteed to work in all situations.
    - **guarded motion** – a motion command along with a termination condition that will preempt the motion if it becomes true.
    - **compliant motion** – allow robot to slide if motion would cause collision.