Mid-term Review
Chapters 2-5, 7, 13, 14

- Review Agents (2.1-2.3)
- Review State Space Search
  - Problem Formulation (3.1, 3.3)
  - Blind (Uninformed) Search (3.4)
  - Heuristic Search (3.5)
  - Local Search (4.1, 4.2)
- Review Adversarial (Game) Search (5.1-5.4)
- Review Propositional Logic (7.1-7.5)
- Review Probability & Bayesian Networks (13, 14.1-14.5)
- Please review your quizzes and old CS-171 tests
  - At least one question from a prior quiz or old CS-171 test will appear on the mid-term (and all other tests)
Review Agents
Chapter 2.1-2.3

• Agent definition (2.1)
• Rational Agent definition (2.2)
  – Performance measure
• Task environment definition (2.3)
  – PEAS acronym
• Properties of Task Environments
  – Fully vs. partially observable; single vs. multi agent;
    deterministic vs. stochastic; episodic vs. sequential; static
    vs. dynamic; discrete vs. continuous; known vs. unknown
• Basic Definitions
  – Percept, percept sequence, agent function, agent program
Agents

• An agent is anything that can be viewed as *perceiving* its environment through *sensors* and *acting* upon that environment through *actuators*.

• **Human agent:**
  eyes, ears, and other organs for sensors;
  hands, legs, mouth, and other body parts for actuators.

• **Robotic agent:**
  cameras and infrared range finders for sensors; various motors for actuators.
Agents and environments

- **Percept:** agent’s perceptual inputs at an instant

- The **agent function** maps from percept sequences to actions:

  \[ f: \mathcal{P}^* \rightarrow \mathcal{A} \]

- The **agent program** runs on the physical **architecture** to produce \( f \)

- **agent = architecture + program**
Rational agents

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

• **Performance measure**: An objective criterion for success of an agent's behavior

• *E.g.*, performance measure of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.
Task Environment

• Before we design an intelligent agent, we must specify its “task environment”:

PEAS:

Performance measure
Environment
Actuators
Sensors
Environment types

- **Fully observable** (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.

- **Deterministic** (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is strategic)

- **Episodic** (vs. sequential): An agent’s action is divided into atomic episodes. Decisions do not depend on previous decisions/actions.

- **Known** (vs. unknown): An environment is considered to be "known" if the agent understands the laws that govern the environment's behavior.
Environment types

- **Static** (vs. **dynamic**): The environment is unchanged while an agent is deliberating. (The environment is **semidynamic** if the environment itself does not change with the passage of time but the agent's performance score does)

- **Discrete** (vs. **continuous**): A limited number of distinct, clearly defined percepts and actions.

  How do we represent or abstract or model the world?

- **Single agent** (vs. **multi-agent**): An agent operating by itself in an environment. Does the other agent interfere with my performance measure?
Review State Space Search
Chapters 3-4

• Problem Formulation (3.1, 3.3)
• Blind (Uninformed) Search (3.4)
  • Depth-First, Breadth-First, Iterative Deepening
  • Uniform-Cost, Bidirectional (if applicable)
  • Time? Space? Complete? Optimal?
• Heuristic Search (3.5)
  • A*, Greedy-Best-First
• Local Search (4.1, 4.2)
  • Hill-climbing, Simulated Annealing, Genetic Algorithms
  • Gradient descent
Problem Formulation

A problem is defined by five items:

- **initial state** e.g., "at Arad"
- **actions**
  - Actions(X) = set of actions available in State X
- **transition model**
  - Result(S,A) = state resulting from doing action A in state S
- **goal test**, e.g., x = "at Bucharest", Checkmate(x)
- **path cost** (additive, i.e., the sum of the step costs)
  - \( c(x,a,y) \) = step cost of action \( a \) in state \( x \) to reach state \( y \)
    - assumed to be \( \geq 0 \)

A solution is a sequence of actions leading from the initial state to a goal state
Tree search vs. Graph search
Review Fig. 3.7, p. 77

• Failure to detect repeated states can turn a linear problem into an exponential one!
• Test is often implemented as a hash table.
Solutions to Repeated States

• Graph search
  – never generate a state generated before
    • must keep track of all possible states (uses a lot of memory)
    • e.g., 8-puzzle problem, we have 9! = 362,880 states
    • approximation for DFS/DLS: only avoid states in its (limited) memory: avoid infinite loops by checking path back to root.
  – “visited?” test usually implemented as a hash table

State Space

Example of a Search Tree

faster, but memory inefficient
Implementation: states vs. nodes

• A **state** is a (representation of) a physical configuration

• A **node** is a data structure constituting part of a search tree

• A node contains info such as:
  – state, parent node, action, path cost $g(x)$, depth, etc.

• The Expand function creates new nodes, filling in the various fields using the $\text{Actions}(S)$ and $\text{Result}(S, A)$ functions associated with the problem.
General tree search

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure

fringe ← INSERT(MAKE-NODE(Initial-State[problem]), fringe)
loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND(node, problem) returns a set of nodes

successors ← the empty set
for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
    Depth[s] ← Depth[node] + 1
    add s to successors
return successors
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure

closed ← an empty set
fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)

loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    if STATE[node] is not in closed then
        add STATE[node] to closed
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
Breadth-first graph search

function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0 if
    problem.GOAL-TEST(node.STATE) then return SOLUTION(node) frontier ←
    a FIFO queue with node as the only element
    explored ← an empty set

loop do

    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /* chooses the shallowest node in frontier */
    add node.STATE to explored

for each action in problem.ACTIONS(node.STATE) do

    child ← CHILD-NODE(problem, node, action)
    if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)

    frontier ← INSERT(child, frontier)

Figure 3.11  Breadth-first search on a graph.
Uniform cost search: sort by \( g \)
A* is identical but uses \( f=g+h \)
Greedy best-first is identical but uses \( h \)

**Figure 3.14** Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for `frontier` needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.
Depth-limited search & IDS

function Depth-Limited-Search(problem, limit) returns soln/fail/cutoff
    Recursive-DLS(Make-Node(Initial-State[problem]), problem, limit)

function Recursive-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if Goal-Test[problem](State[node]) then return Solution(node)
    else if Depth[node] = limit then return cutoff
    else for each successor in Expand(node, problem) do
        result ← Recursive-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
        if cutoff-occurred? then return cutoff else return failure

function Iterative-Deepening-Search(problem) returns a solution, or failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← Depth-Limited-Search(problem, depth)
        if result ≠ cutoff then return result
When to do Goal-Test? Summary

- For DFS, BFS, DLS, and IDS, the goal test is done when the child node is generated.
  - These are not optimal searches in the general case.
  - BFS and IDS are optimal if cost is a function of depth only; then, optimal goals are also shallowest goals and so will be found first.

- For GBFS the behavior is the same whether the goal test is done when the node is generated or when it is removed.
  - $h(\text{goal})=0$ so any goal will be at the front of the queue anyway.

- For UCS and A* the goal test is done when the node is removed from the queue.
  - This precaution avoids finding a short expensive path before a long cheap path.
Blind Search Strategies (3.4)

- Depth-first: Add successors to front of queue
- Breadth-first: Add successors to back of queue
- Uniform-cost: Sort queue by path cost $g(n)$
- Depth-limited: Depth-first, cut off at limit
- Iterated-deepening: Depth-limited, increasing
- Bidirectional: Breadth-first from goal, too.

- Review example Uniform-cost search
  - Slides 25-34, Lecture on “Uninformed Search”
Search strategy evaluation

• A search strategy is defined by the order of node expansion

• Strategies are evaluated along the following dimensions:
  – completeness: does it always find a solution if one exists?
  – time complexity: number of nodes generated
  – space complexity: maximum number of nodes in memory
  – optimality: does it always find a least-cost solution?

• Time and space complexity are measured in terms of
  – $b$: maximum branching factor of the search tree
  – $d$: depth of the least-cost solution
  – $m$: maximum depth of the state space (may be $\infty$)
  – (for UCS: $C^*$: true cost to optimal goal; $\varepsilon > 0$: minimum step cost)
# Summary of algorithms

**Fig. 3.21, p. 91**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening DLS</th>
<th>Bidirectional (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes[a]</td>
<td>Yes[a,b]</td>
<td>No</td>
<td>No</td>
<td>Yes[a]</td>
<td>Yes[a,d]</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+C*/\varepsilon})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+C*/\varepsilon})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes[c]</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes[c]</td>
<td>Yes[c,d]</td>
</tr>
</tbody>
</table>

There are a number of footnotes, caveats, and assumptions. See Fig. 3.21, p. 91.

[a] complete if $b$ is finite  
[b] complete if step costs $\geq \varepsilon > 0$  
[c] optimal if step costs are all identical  
(also if path cost non-decreasing function of depth only)  
[d] if both directions use breadth-first search  
(also if both directions use uniform-cost search with step costs $\geq \varepsilon > 0$)

Generally the preferred uninformed search strategy
Heuristic function (3.5)

- **Heuristic:**
  - Definition: a commonsense rule (or set of rules) intended to increase the probability of solving some problem
  - “using rules of thumb to find answers”

- **Heuristic function h(n)**
  - Estimate of (optimal) cost from n to goal
  - Defined using only the *state* of node n
  - $h(n) = 0$ if n is a goal node
  - Example: straight line distance from n to Bucharest
    - Note that this is not the true state-space distance
    - It is an estimate – actual state-space distance can be higher

- Provides problem-specific knowledge to the search algorithm
Greedy best-first search

• \( h(n) = \) estimate of cost from \( n \) to goal
  – e.g., \( h(n) = \) straight-line distance from \( n \) to Bucharest

• Greedy best-first search expands the node that \textcolor{red}{appears} to be closest to goal.
  – Sort queue by \( h(n) \)

• Not an optimal search strategy
  – May perform well in practice
A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach $n$
  - $h(n)$ = estimated cost from $n$ to goal
  - $f(n)$ = estimated total cost of path through $n$ to goal
- A* search sorts queue by $f(n)$
- Greedy Best First search sorts queue by $h(n)$
- Uniform Cost search sorts queue by $g(n)$
Admissible heuristics

• A heuristic \( h(n) \) is admissible if for every node \( n \), \( h(n) \leq h^*(n) \), where \( h^*(n) \) is the true cost to reach the goal state from \( n \).
• An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
• Example: \( h_{SLD}(n) \) (never overestimates the actual road distance)
• Theorem: If \( h(n) \) is admissible, \( A^* \) using TREE-SEARCH is optimal
Consistent heuristics

(consistent => admissible)

- A heuristic is consistent if for every node \( n \), every successor \( n' \) of \( n \) generated by any action \( a \),

\[
h(n) \leq c(n,a,n') + h(n')
\]

- If \( h \) is consistent, we have

\[
f(n') = g(n') + h(n') \quad \text{(by def.)}
= g(n) + c(n,a,n') + h(n') \quad \text{(} g(n')=g(n)+c(n,a,n')\text{)}
\geq g(n) + h(n) = f(n) \quad \text{(consistency)}
\]

\[
f(n') \geq f(n)
\]

- i.e., \( f(n) \) is non-decreasing along any path.

- Theorem:
  If \( h(n) \) is consistent, \( \text{A* using } \text{GRAPH-SEARCH} \) is optimal

  keeps all checked nodes in memory to avoid repeated states

It’s the triangle inequality!
Local search algorithms (4.1, 4.2)

• In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

• State space = set of "complete" configurations
• Find configuration satisfying constraints, e.g., n-queens
• In such cases, we can use local search algorithms
• keep a single "current" state, try to improve it.
• Very memory efficient (only remember current state)
Random Restart Wrapper

• These are stochastic local search methods
  – Different solution for each trial and initial state

• Almost every trial hits difficulties (see below)
  – Most trials will not yield a good result (sadly)

• Many random restarts improve your chances
  – Many “shots at goal” may, finally, get a good one

• Restart a random initial state; **many times**
  – Report the best result found; **across many trials**
Random Restart Wrapper

BestResultFoundSoFar <- infinitely bad;
UNTIL ( you are tired of doing it ) DO {
    Result <- ( Local search from random initial state );
    IF ( Result is better than BestResultFoundSoFar ) THEN ( Set BestResultFoundSoFar to Result );
}
RETURN BestResultFoundSoFar;

Typically, “you are tired of doing it” means that some resource limit is exceeded, e.g., number of iterations, wall clock time, CPU time, etc. It may also mean that Result improvements are small and infrequent, e.g., less than 0.1% Result improvement in the last week of run time.
Local Search Difficulties

These difficulties apply to ALL local search algorithms, and become MUCH more difficult as the dimensionality of the search space increases to high dimensions.

- **Problems:** depending on state, can get stuck in local maxima
  - Many other problems also endanger your success!!
Hill-climbing search

• "Like climbing Everest in thick fog with amnesia"

```plaintext
function HILL-CLIMBING(problem) returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
              neighbor, a node

current ← MAKE-NODE(INITIAL-STATE[problem])
loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
```
Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but **gradually decrease** their frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
             schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] − VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^{Δ E/T}
```

 Improvement: Track the BestResultFoundSoFar. Here, this slide follows Fig. 4.5 of the textbook, which is simplified.
\[ P(\text{accepting a worse successor}) \]

Decreases as Temperature \( T \) decreases

Increases as \( |\Delta E| \) decreases

(Sometimes step size also decreases with \( T \))

\[
e^{\left( \frac{\Delta E}{T} \right)}
\]

| \( |\Delta E| \) | \( \text{Temperature } T \) |
|---|---|
| High | High | Medium | Low |
| Low  | High | Medium | Low |

\( next \leftarrow \text{a randomly selected successor of } \text{current} \)

\( \Delta E \leftarrow \text{VALUE}[next] - \text{VALUE}[current] \)

if \( \Delta E > 0 \) then \( \text{current} \leftarrow \text{next} \)
else \( \text{current} \leftarrow \text{next} \) only with probability \( e^{\Delta E / T} \)
Genetic algorithms (Darwin!!)

- A state = a string over a finite alphabet (an individual)
- Start with $k$ randomly generated states (a population)
- **Fitness** function (= our heuristic objective function).
  - Higher fitness values for better states.
- **Select** individuals for next generation based on fitness
  - $P(\text{individual in next gen.}) = \text{individual fitness}/\Sigma \text{population fitness}$
- **Crossover** fit parents to yield next generation (off-spring)
- **Mutate** the offspring randomly with some low probability
Fitness function: #non-attacking queen pairs
- min = 0, max = 8 × 7/2 = 28
- \( \sum_i \text{fitness}_i = 24 + 23 + 20 + 11 = 78 \)
- \( P(\text{child}_1 \text{ in next gen.}) = \frac{\text{fitness}_1}{\sum_i \text{fitness}_i} = 24/78 = 31\% \)
- \( P(\text{child}_2 \text{ in next gen.}) = \frac{\text{fitness}_2}{\sum_i \text{fitness}_i} = 23/78 = 29\% \); etc

How to convert a fitness value into a probability of being in the next generation.
Review Adversarial (Game) Search
Chapter 5.1-5.4

• Minimax Search with Perfect Decisions (5.2)
  – Impractical in most cases, but theoretical basis for analysis
• Minimax Search with Cut-off (5.4)
  – Replace terminal leaf utility by heuristic evaluation function
• Alpha-Beta Pruning (5.3)
  – The fact of the adversary leads to an advantage in search!
• Practical Considerations (5.4)
  – Redundant path elimination, look-up tables, etc.
Games as Search

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
  - Winner gets reward, loser gets penalty.
  - "Zero sum" means the sum of the reward and the penalty is a constant.

- Formal definition as a search problem:
  - Initial state: Set-up specified by the rules, e.g., initial board configuration of chess.
  - Player(s): Defines which player has the move in a state.
  - Actions(s): Returns the set of legal moves in a state.
  - Result(s,a): Transition model defines the result of a move.
  - (2nd ed.: Successor function: list of (move,state) pairs specifying legal moves.)
  - Terminal-Test(s): Is the game finished? True if finished, false otherwise.
  - Utility function(s,p): Gives numerical value of terminal state s for player p.
    - E.g., win (+1), lose (-1), and draw (0) in tic-tac-toe.
    - E.g., win (+1), lose (0), and draw (1/2) in chess.

- MAX uses search tree to determine "best" next move.
An optimal procedure: The Min-Max method

Will find the optimal strategy and best next move for Max:

• 1. Generate the whole game tree, down to the leaves.

• 2. Apply utility (payoff) function to each leaf.

• 3. Back-up values from leaves through branch nodes:
   – a Max node computes the Max of its child values
   – a Min node computes the Min of its child values

• 4. At root: Choose move leading to the child of highest value.
Two-Ply Game Tree
Two-Ply Game Tree

Minimax maximizes the utility of the worst-case outcome for Max

The minimax decision
function MINIMAX-DECISION(state) returns an action
inputs: state, current state in game
return \( \text{arg max}_{a \in \text{ACTIONS}(state)} \text{MIN-VALUE}(\text{Result}(state,a)) \)

function MAX-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
\( \nu \leftarrow -\infty \)
for \( a \) in ACTIONS(state) do
\( \nu \leftarrow \text{MAX}(\nu, \text{MIN-VALUE}(\text{Result}(state,a))) \)
return \( \nu \)

function MIN-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
\( \nu \leftarrow +\infty \)
for \( a \) in ACTIONS(state) do
\( \nu \leftarrow \text{MIN}(\nu, \text{MAX-VALUE}(\text{Result}(state,a))) \)
return \( \nu \)
Properties of minimax

• **Complete?**
  - Yes (if tree is finite).

• **Optimal?**
  - Yes (against an optimal opponent).
  - Can it be beaten by an opponent playing sub-optimally?
    - No. (Why not?)

• **Time complexity?**
  - $O(b^m)$

• **Space complexity?**
  - $O(bm)$ (depth-first search, generate all actions at once)
  - $O(m)$ (backtracking search, generate actions one at a time)
Cutting off search

MinimaxCutoff is identical to MinimaxValue except
1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

Does it work in practice?

\[ b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4 \]

4-ply lookahead is a hopeless chess player!

4-ply \( \approx \) human novice
8-ply \( \approx \) typical PC, human master
12-ply \( \approx \) Deep Blue, Kasparov
Static (Heuristic) Evaluation Functions

• An Evaluation Function:
  – Estimates how good the current board configuration is for a player.
  – Typically, evaluate how good it is for the player, how good it is for the opponent, then subtract the opponent’s score from the player’s.
  – Othello: Number of white pieces - Number of black pieces
  – Chess: Value of all white pieces - Value of all black pieces

• Typical values from -infinity (loss) to +infinity (win) or [-1, +1].

• If the board evaluation is X for a player, it’s -X for the opponent
  – “Zero-sum game”
Evaluation functions

Black to move
White slightly better

White to move
Black winning

For chess, typically \textit{linear} weighted sum of features

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

\text{e.g., } w_1 = 9 \text{ with }

\[ f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.} \]
Alpha-beta Algorithm

• Depth first search
  – only considers nodes along a single path from root at any time

\[ \alpha = \text{highest-value choice found at any choice point of path for MAX} \]
  (initially, \( \alpha = -\text{infinity} \))

\[ \beta = \text{lowest-value choice found at any choice point of path for MIN} \]
  (initially, \( \beta = +\text{infinity} \))

• Pass current values of \( \alpha \) and \( \beta \) down to child nodes during search.
• Update values of \( \alpha \) and \( \beta \) during search:
  – MAX updates \( \alpha \) at MAX nodes
  – MIN updates \( \beta \) at MIN nodes
• Prune remaining branches at a node when \( \alpha \geq \beta \)
Pseudocode for Alpha-Beta Algorithm

```plaintext
function ALPHA-BETA-SEARCH(state) returns an action

inputs: state, current state in game

v ← MAX-VALUE(state, -∞, +∞)

return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

v ← -∞

for a in ACTIONS(state) do

    v ← MAX(v, MIN-VALUE(Result(s,a), α, β))

    if v ≥ β then return v

    α ← MAX(α, v)

return v

(MIN-VALUE is defined analogously)
```
When to Prune?

• **Prune whenever** $\alpha \geq \beta$.

  – Prune below a Max node whose alpha value becomes greater than or equal to the beta value of its ancestors.
    • **Max nodes update alpha** based on children’s returned values.

  – Prune below a Min node whose beta value becomes less than or equal to the alpha value of its ancestors.
    • **Min nodes update beta** based on children’s returned values.
Alpha-Beta Example Revisited

Do DF-search until first leaf

\[ \alpha = -\infty \]
\[ \beta = +\infty \]

\( \alpha, \beta, \) initial values

\( \alpha = -\infty \)
\( \beta = +\infty \)

\( \alpha, \beta, \) passed to kids

Review Detailed Example of Alpha-Beta Pruning in lecture slides.
MIN updates $\beta$, based on kids
MIN updates $\beta$, based on kids.
No change.
Alpha-Beta Example (continued)

MAX updates $\alpha$, based on kids.

$\alpha = 3$

$\beta = +\infty$

3 is returned as node value.
Alpha-Beta Example (continued)

\[ \alpha = 3 \]
\[ \beta = +\infty \]

\(\alpha, \beta,\) passed to kids
\[ \alpha = 3 \]
\[ \beta = +\infty \]
Alpha-Beta Example (continued)

\[ \alpha = 3 \]
\[ \beta = +\infty \]

MIN updates \( \beta \), based on kids.

\[ \alpha = 3 \]
\[ \beta = 2 \]
Alpha-Beta Example (continued)

\[ \alpha = 3, \beta = +\infty \]

\[ \alpha \geq \beta, \text{ so prune.} \]
Alpha-Beta Example (continued)

MAX updates $\alpha$, based on kids.
No change.

$\alpha = 3$
$\beta = +\infty$

2 is returned as node value.

Review Detailed Example of Alpha-Beta Pruning in lecture slides.
Review Propositional Logic
Chapter 7.1-7.5

• Definitions:
  – Syntax, Semantics, Sentences, Propositions, Entails, Follows, Derives, Inference, Sound, Complete, Model, Satisfiable, Valid (or Tautology)

• Syntactic Transformations:
  – E.g., \((A \Rightarrow B) \iff (\neg A \lor B)\)

• Semantic Transformations:
  – E.g., \((KB \models \alpha) \equiv (\models (KB \Rightarrow \alpha))\)

• Truth Tables:
  – Negation, Conjunction, Disjunction, Implication, Equivalence (Biconditional)

• Inference:
  – By Model Enumeration (truth tables)
  – By Resolution
Recap propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas

- The proposition symbols $P_1$, $P_2$ etc are sentences
  - If $S$ is a sentence, $\neg S$ is a sentence (negation)
  - If $S_1$ and $S_2$ are sentences, $S_1 \land S_2$ is a sentence (conjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \lor S_2$ is a sentence (disjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)
Recap propositional logic:

**Semantics**

Each model/world specifies true or false for each proposition symbol

E.g. $P_{1,2}$ $P_{2,2}$ $P_{3,1}$

false true false

With these symbols, 8 possible models can be enumerated automatically.

Rules for evaluating truth with respect to a model $m$:

$\neg S$ is true iff $S$ is false

$S_1 \land S_2$ is true iff $S_1$ is true and $S_2$ is true

$S_1 \lor S_2$ is true iff $S_1$ is true or $S_2$ is true

$S_1 \Rightarrow S_2$ is true iff $S_1$ is false or $S_2$ is true

(i.e., is false iff $S_1$ is true and $S_2$ is false)

$S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true

Simple recursive process evaluates an arbitrary sentence, e.g.,

$\neg P_{1,2} \land (P_{2,2} \lor P_{3,1}) = true \land (true \lor false) = true \land true = true$
Recap propositional logic:

Truth tables for connectives

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>¬P</td>
<td>P ∧ Q</td>
<td>P ∨ Q</td>
<td>P ⇒ Q</td>
<td>P ⇔ Q</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

**OR:** P or Q is true or both are true.

**XOR:** P or Q is true but not both.

**Implication is always true when the premises are False!**
Recap propositional logic:

**Logical equivalence and rewrite rules**

- To manipulate logical sentences we need some rewrite rules.
- Two sentences are **logically equivalent** iff they are true in same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) & \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) & \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) & \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) & \text{associativity of } \lor \\
\neg(\neg \alpha) & \equiv \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv (\neg \beta \Rightarrow \neg \alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv (\neg \alpha \lor \beta) & \text{implication elimination} \\
(\alpha \leftrightarrow \beta) & \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg \alpha \lor \neg \beta) & \text{de Morgan} \\
\neg(\alpha \lor \beta) & \equiv (\neg \alpha \land \neg \beta) & \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) & \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]
Recap propositional logic: Entailment

- **Entailment** means that one thing follows from another:

\[ \text{KB} \models \alpha \]

- Knowledge base \( KB \) entails sentence \( \alpha \) if and only if \( \alpha \) is true in all worlds where \( KB \) is true

  - E.g., the KB containing “the Giants won and the Reds won” entails “The Giants won”.
  - E.g., \( x+y = 4 \) entails \( 4 = x+y \)
  - E.g., “Mary is Sue’s sister and Amy is Sue’s daughter” entails “Mary is Amy’s aunt.”
Review: Models (and in FOL, Interpretations)

- Models are formal worlds in which truth can be evaluated

- We say $m$ is a model of a sentence $\alpha$ if $\alpha$ is true in $m$

- $M(\alpha)$ is the set of all models of $\alpha$

- Then $KB \models \alpha$ iff $M(KB) \subseteq M(\alpha)$
  - E.g. $KB$, = “Mary is Sue’s sister and Amy is Sue’s daughter.”
  - $\alpha$ = “Mary is Amy’s aunt.”

- Think of $KB$ and $\alpha$ as constraints, and of models $m$ as possible states.
- $M(KB)$ are the solutions to $KB$ and $M(\alpha)$ the solutions to $\alpha$.
- Then, $KB \models \alpha$, i.e., $\models (KB \Rightarrow \alpha)$, when all solutions to $KB$ are also solutions to $\alpha$. 
• \( KB = \) all possible wumpus-worlds consistent with the observations and the “physics” of the Wumpus world.
Review: Wumpus models

$\alpha_1 = "[1,2] is safe", \ KB \models \alpha_1$, proved by model checking.

Every model that makes $KB$ true also makes $\alpha_1$ true.
Wumpus models

$\alpha_2 = "[2,2] is safe", \ KB \nvDash\ alpha_2$
If KB is true in the real world, then any sentence $\alpha$ entailed by KB and any sentence $\alpha$ derived from KB by a sound inference procedure is also true in the real world.
Schematic Example: Follows, Entails, and Derives

Inference

“Mary is Sue’s sister and Amy is Sue’s daughter.”

“An aunt is a sister of a parent.”

Derives

Is it provable?

“Mary is Amy’s aunt.”

Entails

Is it true?

“Mary is Amy’s aunt.”

Follows

Is it the case?

“Mary is Amy’s aunt.”

World

Mary \(\xrightarrow{\text{Sister}}\) Sue

Mary \(\xrightarrow{\text{Daughter}}\) Amy

Mary \(\xrightarrow{\text{Aunt}}\) Amy
Recap propositional logic: **Validity and satisfiability**

A sentence is **valid** if it is true in all models,

- e.g., *True*,  \( A \lor \neg A \),  \( A \Rightarrow A \),  \( (A \land (A \Rightarrow B)) \Rightarrow B \)

Validity is connected to inference via the **Deduction Theorem**:

\( KB \models \alpha \) if and only if \( (KB \Rightarrow \alpha) \) is valid

A sentence is **satisfiable** if it is true in some model

- e.g., \( A \lor B \),  \( C \)

A sentence is **unsatisfiable** if it is false in all models

- e.g., \( A \land \neg A \)

Satisfiability is connected to inference via the following:

\( KB \not\models A \) if and only if \( (KB \land \neg A) \) is unsatisfiable

(there is no model for which \( KB \) is true and \( A \) is false)
Inference Procedures

- \( KB \vdash_i A \) means that sentence A can be derived from KB by procedure i.

- **Soundness**: \( i \) is sound if whenever \( KB \vdash_i \alpha \), it is also true that \( KB \models \alpha \)
  - (no wrong inferences, but maybe not all inferences)

- **Completeness**: \( i \) is complete if whenever \( KB \models \alpha \), it is also true that \( KB \vdash_i \alpha \)
  - (all inferences can be made, but maybe some wrong extra ones as well)

- Entailment can be used for inference (Model checking)
  - enumerate all possible models and check whether \( \alpha \) is true.
  - For \( n \) symbols, time complexity is \( O(2^n) \).

- Inference can be done directly on the sentences
  - Forward chaining, backward chaining, resolution (see FOPC, later)
We’d like to prove: $KB \models \alpha$

$\text{equivalent to: } KB \land \neg \alpha \text{ unsatisfiable}$

We first rewrite $KB \land \neg \alpha$ into conjunctive normal form (CNF).

A “conjunction of disjunctions”

$(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

Clause  Clause

• Any KB can be converted into CNF.
• In fact, any KB can be converted into CNF-3 using clauses with at most 3 literals.
Example: Conversion to CNF

Example: \( B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \)

1. Eliminate \( \iff \) by replacing \( \alpha \iff \beta \) with \((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)\).
   \[
   = (B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})
   \]

2. Eliminate \( \Rightarrow \) by replacing \( \alpha \Rightarrow \beta \) with \( \neg \alpha \lor \beta \) and simplify.
   \[
   = (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})
   \]

3. Move \( \neg \) inwards using de Morgan's rules and simplify.
   \[
   \neg(\alpha \lor \beta) = \neg \alpha \land \neg \beta
   \]
   \[
   = (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})
   \]

4. Apply distributive law \( (\land \text{ over } \lor) \) and simplify.
   \[
   = (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})
   \]
Example: Conversion to CNF

Example: \( B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1}) \)

From the previous slide we had:

\[
= (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})
\]

5. KB is the conjunction of all of its sentences (all are true), so write each clause (disjunct) as a sentence in KB:

\[
\text{KB} = \quad \ldots
\]

\[
(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \quad \rightarrow \quad (\neg B_{1,1} \quad P_{1,2} \quad P_{2,1})
\]

\[
(\neg P_{1,2} \lor B_{1,1}) \quad \rightarrow \quad (\neg P_{1,2} \quad B_{1,1})
\]

\[
(\neg P_{2,1} \lor B_{1,1}) \quad \rightarrow \quad (\neg P_{2,1} \quad B_{1,1})
\]

\[
(\text{same})
\]

Often, Won’t Write “\( \lor \)” or “\( \land \)” (we know they are there)
Inference by Resolution

• KB is represented in CNF
  – KB = AND of all the sentences in KB
  – KB sentence = clause = OR of literals
  – Literal = propositional symbol or its negation
  – Add the negated goal sentence to KB

• Find two clauses in KB, one of which contains a literal and the other its negation

• Cancel the literal and its negation

• Bundle everything else into a new clause

• Add the new clause to KB and keep going

• Stop at the empty clause: ( ) = FALSE, you proved it!
  – Or stop when no more new inferences are possible
Resolution = Efficient Implication

Recall that \((A \Rightarrow B) = (\neg A \lor B)\)
and so:

\[(Y \lor X) = (\neg X \Rightarrow Y)\]
\[(\neg Y \lor Z) = (Y \Rightarrow Z)\]

which yields:

\[(Y \lor X) \land (\neg Y \lor Z) = (\neg X \Rightarrow Z) = (X \lor Z)\]

\[
\begin{align*}
(\text{OR } & A B C D) \quad \Rightarrow \text{Same} \quad \Rightarrow \text{Same} \\
(\text{OR } & \neg A E F G) \\
\text{---------------------------} \\
(\text{OR } & B C D E F G) \\
\end{align*}
\]

Recall: All clauses in KB are conjoined by an implicit AND (= CNF representation).
Resolution Examples

• Resolution: inference rule for CNF: sound and complete! *

\[(A \lor B \lor C)\]
\[\neg A\]
\[
\therefore (B \lor C)
\]

“If A or B or C is true, but not A, then B or C must be true.”

\[(A \lor B \lor C)\]
\[\neg (A \lor D \lor E)\]
\[
\therefore (B \lor C \lor D \lor E)
\]

“If A is false then B or C must be true, or if A is true then D or E must be true, hence since A is either true or false, B or C or D or E must be true.”

\[(A \lor B)\]
\[\neg (A \lor B)\]
\[
\therefore (B \lor B) \equiv B
\]

Simplification is done always.

* Resolution is “refutation complete” in that it can prove the truth of any entailed sentence by refutation.

* You can start two resolution proofs in parallel, one for the sentence and one for its negation, and see which branch returns a correct proof.
Only Resolve **ONE** Literal Pair!
If more than one pair, result always = TRUE.

**Useless!!** Always simplifies to TRUE!!

---

No!
(OR A B C D)
(OR ¬A ¬B F G)

-------------------------------

(OR C D F G)

No!

---

Yes! (but = TRUE)
(OR A B C D)
(OR ¬A ¬B F G)

-------------------------------

(OR B ¬B C D F G)

Yes! (but = TRUE)
Resolution example

- $KB = (B_{1,1} \leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$

$KB \land \neg \alpha$

True!

False in all worlds
Detailed Resolution Proof Example

- **In words:** If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

  _Prove that the unicorn is both magical and horned._

  \[
  \begin{align*}
  (\neg Y \neg R) & \land (M Y) & \land (R Y) & \land (H \neg M) \\
  (H R) & \land (\neg H G) & \land (\neg G \neg H) \\
  \end{align*}
  \]

- **Fourth, produce a resolution proof ending in ( ):**
  - Resolve (\neg H \neg G) and (\neg H G) to give (\neg H)
  - Resolve (\neg Y \neg R) and (Y M) to give (\neg R M)
  - Resolve (\neg R M) and (R H) to give (M H)
  - Resolve (M H) and (\neg M H) to give (H)
  - Resolve (\neg H) and (H) to give ( )

- Of course, there are many other proofs, which are OK iff correct.
Propositional Logic --- Summary

• Logical agents apply inference to a knowledge base to derive new information and make decisions

• Basic concepts of logic:
  – syntax: formal structure of sentences
  – semantics: truth of sentences wrt models
  – entailment: necessary truth of one sentence given another
  – inference: deriving sentences from other sentences
  – soundness: derivations produce only entailed sentences
  – completeness: derivations can produce all entailed sentences
  – valid: sentence is true in every model (a tautology)

• Logical equivalences allow syntactic manipulations

• Propositional logic lacks expressive power
  – Can only state specific facts about the world.
  – Cannot express general rules about the world
    (use First Order Predicate Logic instead)
Review Probability
Chapter 13

- Basic probability notation/definitions
- Probability model, unconditional/prior and conditional/posterior probabilities, factored representation (= variable/value pairs), random variable, (joint) probability distribution, probability density function (pdf), marginal probability, (conditional) independence, normalization, etc.
- Probability axioms, basic probability formulae
- Product rule, summation rule, Bayes’ rule, factoring.
Syntax

• Basic element: random variable
• Similar to propositional logic: possible worlds defined by assignment of
  values to random variables.

• Boolean random variables
e.g., \textit{Cavity} (= do I have a cavity?)

• Discrete random variables
e.g., \textit{Weather is one of}
  \textit{<sunny,rainy,cloudy,snow>}
• Domain values must be exhaustive and mutually exclusive

• Elementary proposition is an assignment of a value to a random variable:
e.g., \textit{Weather} = sunny; \textit{Cavity} = false(abbreviated as \textit{\neg}cavity)

• Complex propositions formed from elementary propositions and standard
  logical connectives :
e.g., \textit{Weather} = sunny \lor \textit{Cavity} = false
Probability

• P(a) is the probability of proposition “a”
  – E.g., P(it will rain in London tomorrow)
  – The proposition a is actually true or false in the real-world
  – P(a) = “prior” or marginal or unconditional probability
  – Assumes no other information is available

• Axioms:
  – 0 <= P(a) <= 1
  – P(NOT(a)) = 1 – P(a)
  – P(true) = 1
  – P(false) = 0
  – P(A OR B) = P(A) + P(B) – P(A AND B)

• An agent that holds degrees of beliefs that contradict these axioms will act sub-optimally in some cases
  – e.g., de Finetti proved that there will be some combination of bets that forces such an unhappy agent to lose money every time.
  – No rational agent can have axioms that violate probability theory.
**Conditional Probability**

- $P(a|b)$ is the conditional probability of proposition $a$, conditioned on knowing that $b$ is true,
  - E.g., $P(\text{rain in London tomorrow} \mid \text{raining in London today})$
  - $P(a|b)$ is a “posterior” or conditional probability
  - The updated probability that $a$ is true, now that we know $b$
  - $P(a|b) = \frac{P(a \text{ AND } b)}{P(b)}$
  - Syntax: $P(a \mid b)$ is the probability of $a$ given that $b$ is true
    - $a$ and $b$ can be any propositional sentences
    - E.g., $p(\text{John wins OR Mary wins} \mid \text{Bob wins AND Jack loses})$

- $P(a|b)$ obeys the same rules as probabilities,
  - E.g., $P(a \mid b) + P(\text{NOT}(a) \mid b) = 1$
  - All probabilities in effect are conditional probabilities
    - E.g., $P(a) = P(a \mid \text{our background knowledge})$
Random Variables

• A is a random variable taking values \(a_1, a_2, \ldots, a_m\)
  – Events are \(A = a_1, A = a_2, \ldots\)
  – We will focus on discrete random variables

• Mutual exclusion
  \[ P(A = a_i \text{ AND } A = a_j) = 0 \]

• Exhaustive
  \[ \sum P(a_i) = 1 \]

MEE (Mutually Exclusive and Exhaustive) assumption is often useful
(but not always appropriate, e.g., disease-state for a patient)

For finite \(m\), can represent \(P(A)\) as a table of \(m\) probabilities

For infinite \(m\) (e.g., number of tosses before “heads”) we can represent \(P(A)\) by a function (e.g., geometric)
Joint Distributions

• Consider 2 random variables: A, B
  – P(a, b) is shorthand for P(A = a AND B=b)
  – $\Sigma_a \Sigma_b P(a, b) = 1$
  – Can represent P(A, B) as a table of $m^2$ numbers

• Generalize to more than 2 random variables
  – E.g., A, B, C, ... Z
  – $\Sigma_a \Sigma_b... \Sigma_z P(a, b, ..., z) = 1$
  – P(A, B, .... Z) is a table of $m^K$ numbers, $K = # variables$
    • This is a potential problem in practice, e.g., m=2, K = 20
Linking Joint and Conditional Probabilities

• Basic fact:
  \[ P(a, b) = P(a \mid b) P(b) \]

  – Why? Probability of \( a \) and \( b \) occurring is the same as probability of \( a \) occurring given \( b \) is true, times the probability of \( b \) occurring

• Bayes rule:
  \[ P(a, b) = P(a \mid b) P(b) = P(b \mid a) P(a) \text{ by definition} \]

  \[ \Rightarrow P(b \mid a) = \frac{P(a \mid b) P(b)}{P(a)} \quad \text{[Bayes rule]} \]

Why is this useful?

Often much more natural to express knowledge in a particular “direction”, e.g., in the causal direction

E.g., \( b = \text{disease}, a = \text{symptoms} \)
More natural to encode knowledge as \( P(a \mid b) \) than as \( P(b \mid a) \)
Summary of Probability Rules

- **Product Rule:**
  - \( P(a, b) = P(a|b) P(b) = P(b|a) P(a) \)
  - Probability of “a” and “b” occurring is the same as probability of “a” occurring given “b” is true, times the probability of “b” occurring.
    - e.g., \( P(\text{rain, cloudy}) = P(\text{rain | cloudy}) \times P(\text{cloudy}) \)

- **Sum Rule:** (AKA Law of Total Probability)
  - \( P(a) = \sum_b P(a, b) = \sum_b P(a|b) P(b) \), where B is any random variable
  - Probability of “a” occurring is the same as the sum of all joint probabilities including the event, provided the joint probabilities represent all possible events.
  - Can be used to “marginalize” out other variables from probabilities, resulting in prior probabilities also being called marginal probabilities.
    - e.g., \( P(\text{rain}) = \sum_{\text{Windspeed}} P(\text{rain, Windspeed}) \)
      where Windspeed = \{0-10mph, 10-20mph, 20-30mph, etc.\}

- **Bayes’ Rule:**
  - \( P(b|a) = \frac{P(a|b) P(b)}{P(a)} \)
  - Acquired from rearranging the product rule.
  - Allows conversion between conditionals, from \( P(a|b) \) to \( P(b|a) \).
    - e.g., \( b = \text{disease, } a = \text{symptoms} \)
      More natural to encode knowledge as \( P(a|b) \) than as \( P(b|a) \).
Computing with Probabilities: Law of Total Probability

Law of Total Probability (aka “summing out” or marginalization)
\[ P(a) = \sum_b P(a, b) \]
\[ = \sum_b P(a \mid b) P(b) \]
where B is any random variable

Why is this useful?
Given a joint distribution (e.g., \( P(a,b,c,d) \)) we can obtain any “marginal” probability (e.g., \( P(b) \)) by summing out the other variables, e.g.,

\[ P(b) = \sum_a \sum_c \sum_d P(a, b, c, d) \]

We can compute any conditional probability given a joint distribution, e.g.,

\[ P(c \mid b) = \sum_a \sum_d P(a, c, d \mid b) \]
\[ = \sum_a \sum_d P(a, c, d, b) / P(b) \]
where \( P(b) \) can be computed as above
Independence

• 2 random variables A and B are independent iff
  \[ P(a, b) = P(a) \cdot P(b) \quad \text{for all values } a, b \]

• More intuitive (equivalent) conditional formulation
  – A and B are independent iff
    \[ P(a \mid b) = P(a) \quad \text{OR} \quad P(b \mid a) \cdot P(b), \quad \text{for all values } a, b \]

  – Intuitive interpretation:
    \[ P(a \mid b) = P(a) \] tells us that knowing b provides no change in our probability for a, i.e., b contains no information about a

• Can generalize to more than 2 random variables

• In practice true independence is very rare
  – “butterfly in China” effect
  – Weather and dental example in the text
  – Conditional independence is much more common and useful

• Note: independence is an assumption we impose on our model of the world - it does not follow from basic axioms
Conditional Independence

• 2 random variables A and B are conditionally independent given C iff

\[
P(a, b | c) = P(a | c) \cdot P(b | c) \quad \text{for all values } a, b, c
\]

• More intuitive (equivalent) conditional formulation
  – A and B are conditionally independent given C iff

\[
P(a | b, c) = P(a | c) \quad \text{OR} \quad P(b | a, c) \cdot P(b | c), \quad \text{for all values } a, b, c
\]

  – Intuitive interpretation:

\[
P(a | b, c) = P(a | c) \text{ tells us that learning about } b, \text{ given that we already know } c, \text{ provides no change in our probability for } a,
\]

  i.e., b contains no information about a beyond what c provides

• Can generalize to more than 2 random variables
  – E.g., K different symptom variables X1, X2, ... XK, and C = disease

\[
P(X1, X2, ..., XK | C) = \prod P(Xi | C)
\]

  – Also known as the naïve Bayes assumption
Review Bayesian Networks
Chapter 14.1-14.5

• Basic concepts and vocabulary of Bayesian networks.
  – Nodes represent random variables.
  – Directed arcs represent (informally) direct influences.
  – Conditional probability tables, P( Xi | Parents(Xi) ).

• Given a Bayesian network:
  – Write down the full joint distribution it represents.
  – Inference by Variable Elimination

• Given a full joint distribution in factored form:
  – Draw the Bayesian network that represents it.

• Given a variable ordering and some background assertions of conditional independence among the variables:
  – Write down the factored form of the full joint distribution, as simplified by the conditional independence assertions.
Bayesian Networks

- Represent dependence/independence via a directed graph
  - Nodes = random variables
  - Edges = direct dependence
- Structure of the graph $\iff$ Conditional independence relations
- Recall the chain rule of repeated conditioning:

\[
P(X_1, X_2, X_3, \ldots, X_N) = P(X_1|X_2, X_3, \ldots, X_N)P(X_2|X_3, \ldots, X_N) \cdots P(X_N)
\]

\[
P(X_1, X_2, X_3, \ldots, X_N) = \prod_{i=1}^{n} P(X_i|\text{parents}(X_i))
\]

The full joint distribution $\quad$ The graph-structured approximation

- Requires that graph is acyclic (no directed cycles)
- 2 components to a Bayesian network
  - The graph structure (conditional independence assumptions)
  - The numerical probabilities (for each variable given its parents)
A Bayesian network specifies a joint distribution in a structured form:

- Dependence/independence represented via a directed graph:
  - Node = random variable
  - Directed Edge = conditional dependence
  - Absence of Edge = conditional independence

- Allows concise view of joint distribution relationships:
  - Graph nodes and edges show conditional relationships between variables.
  - Tables provide probability data.

Full factorization:

\[ p(A, B, C) = p(C|A, B)p(A|B)p(B) = p(C|A, B)p(A)p(B) \]

After applying conditional independence from the graph:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>P(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>0.2</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>0.4</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>0.3</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Examples of 3-way Bayesian Networks

Marginal Independence:
\[ p(A, B, C) = p(A) \cdot p(B) \cdot p(C) \]
Examples of 3-way Bayesian Networks

Conditionally independent effects:
\[ p(A,B,C) = p(B|A)p(C|A)p(A) \]

B and C are conditionally independent
Given A

e.g., A is a disease, and we model
B and C as conditionally independent
symptoms given A

e.g. A is culprit, B is murder weapon
and C is fingerprints on door to the
guest’s room
Examples of 3-way Bayesian Networks

Independent Causes:
\[ p(A,B,C) = p(C|A,B)p(A)p(B) \]

“Explaining away” effect:
Given C, observing A makes B less likely
e.g., earthquake/burglary/alarm example

A and B are (marginally) independent
but become dependent once C is known
Examples of 3-way Bayesian Networks

Markov chain dependence:
\[ p(A,B,C) = p(C|B) p(B|A)p(A) \]

e.g. If Prof. Lathrop goes to party, then I might go to party. If I go to party, then my wife might go to party.
Bigger Example

• Consider the following 5 binary variables:
  – B = a burglary occurs at your house
  – E = an earthquake occurs at your house
  – A = the alarm goes off
  – J = John calls to report the alarm
  – M = Mary calls to report the alarm

• Sample Query: What is P(B|M, J) ?

• Using full joint distribution to answer this question requires
  – $2^5 - 1 = 31$ parameters

• Can we use prior domain knowledge to come up with a Bayesian network that requires fewer probabilities?
Constructing a Bayesian Network: Step 1

• Order the variables in terms of influence (may be a partial order)
  
e.g., \{E, B\} -> \{A\} -> \{J, M\}

• \[ P(J, M, A, E, B) = P(J, M | A, E, B) \cdot P(A | E, B) \cdot P(E, B) \]
  
  \[ \approx P(J, M | A) \cdot P(A | E, B) \cdot P(E) \cdot P(B) \]

  \[ \approx P(J | A) \cdot P(M | A) \cdot P(A | E, B) \cdot P(E) \cdot P(B) \]

These conditional independence assumptions are reflected in the graph structure of the Bayesian network
Constructing this Bayesian Network: Step 2

- \( P(J, M, A, E, B) = \)
  - \( P(J \mid A) \) \( P(M \mid A) \) \( P(A \mid E, B) \) \( P(E) \) \( P(B) \)

- There are 3 conditional probability tables (CPDs) to be determined:
  - \( P(J \mid A) \), \( P(M \mid A) \), \( P(A \mid E, B) \)
  - Requiring \( 2 + 2 + 4 = 8 \) probabilities

- And 2 marginal probabilities \( P(E) \), \( P(B) \) \( \rightarrow \) 2 more probabilities

- Where do these probabilities come from?
  - Expert knowledge
  - From data (relative frequency estimates)
  - Or a combination of both - see discussion in Section 20.1 and 20.2 (optional)
The Resulting Bayesian Network

- **Burglary**
  - $P(B) = 0.001$

- **Earthquake**
  - $P(E) = 0.002$

- **Alarm**
  - Values:
    - $P(A)$ for $B = t$, $E = t$: 0.95
    - $P(A)$ for $B = t$, $E = f$: 0.94
    - $P(A)$ for $B = f$, $E = t$: 0.29
    - $P(A)$ for $B = f$, $E = f$: 0.001

- **JohnCalls**
  - $P(J)$ for $A = t$: 0.90
  - $P(J)$ for $A = f$: 0.05

- **MaryCalls**
  - $P(M)$ for $A = t$: 0.70
  - $P(M)$ for $A = f$: 0.01
The Bayesian Network from a different Variable Ordering

(b)
Inference by Variable Elimination

- Say that query is $P(B \mid j,m)$
  - $P(B \mid j,m) = P(B,j,m) / P(j,m) = \alpha P(B,j,m)$
- Apply evidence to expression for joint distribution
  - $P(j,m,A,E,B) = P(j \mid A)P(m \mid A)P(A \mid E,B)P(E)P(B)$
- Marginalize out $A$ and $E$

\[
P(B \mid j, m) = \alpha \sum_a \sum_e p(j \mid a)p(m \mid a)p(a \mid e, B)P(e)P(B)
\]
\[
= \alpha P(B) \sum_e P(e) \sum_a p(j \mid a)p(m \mid a)p(a \mid e, B)
\]

Distribution over variable $B$ – i.e. over states \{b,\neg b\}

Sum is over states of variable $A$ – i.e. \{a,\neg a\}
What is the posterior conditional distribution of our query variables, given that fever was observed?

\[
P(A, B|d) = \alpha \sum_c P(A, B, c, d) = \alpha P(A)P(B) \sum_c P(c|A,B)P(d|c) + P(\neg c|A,B)P(d|\neg c) \\
= \alpha P(A)P(B) \times \{ .95 \times .95 + .05 \times .002 \} \approx \alpha .00903 \approx .014
\]

\[
\begin{align*}
P(\neg a, \neg b|d) &= \alpha P(\neg a)P(\neg b) \sum_c P(c|\neg a, \neg b)P(d|c) \\
&= \alpha P(\neg a)P(\neg b) \times \{ .95 \times .95 + .05 \times .002 \} \approx \alpha .00627 \approx .096
\end{align*}
\]

\[
\alpha \approx 1 / (.000903+.0162+.0419+.00627) \approx 1 / .06527 \approx 15.32
\]
Mid-term Review
Chapters 2-5, 7, 13, 14

• Review Agents (2.1-2.3 ? 1-2)
• Review State Space Search
  • Problem Formulation (3.1, 3.3 ? 3.1-3.4)
  • Blind (Uninformed) Search (3.4)
  • Heuristic Search (3.5 ? 3.5-3.7)
  • Local Search (4.1, 4.2)
• Review Adversarial (Game) Search (5.1-5.4)
• Review Propositional Logic (7.1-7.5)
• Review Probability & Bayesian Networks (13, 14.1-14.5)
• Please review your quizzes and old CS-171 tests
  • At least one question from a prior quiz or old CS-171 test will appear on the mid-term (and all other tests)