CMPT 310: Artificial Intelligence

A Course Overview

Jeffrey Leung
Simon Fraser University

Spring 2019

Contents

1 Introduction

- **Artificial intelligence:** Systems which may think like humans, think rationally, act like humans, or act rationally/correctly
  - **Strong AI:** System which is functionally and cognitively equal to a human
  - **Weak AI:** System which can solve a specific problem well
- **Agent:** Entity which interacts with another entity (if mechanically, through sensors and actuators)
  - E.g. Humans, animals, robots, Roombas
  - **Percept:** Input from sensors
  - **Agent function:** Mapping from the percept history to an action by actuators
    - Notation: $f : P^* \to A$
    - **Agent program:** Agent function in software form
  - **Performance measure:** Arbitrary calculation of the success of actions
  - **Rational agent:** Agent which acts to maximize the result of the performance measure
    - Is neither omniscient (not all information is known) nor clairvoyant (outcomes may not be as expected)
    - **Task environment:** Performance measure, Environment, Actuators, and Sensors of a rational agent (PEAS)
      - **Fully observable:** Environment where all aspects can be perceived at all times, at any time (alternate: *partially observable* or *unobservable*)
      - **Deterministic:** Environment with no randomness (antonym: *stochastic*)
      - **Episodic:** Environment which occurs in a series of independent tasks, none of which affect each other (antonym: *sequential*)
      - **Static:** Environment which does not change while the agent is between actions (antonym: *dynamic*)
      - **Discrete:** Environment which occurs in distinct stages (antonym: *continuous*)
      - **Single-agent:** Environment where only one agent acts upon or influences any other entity (antonym: *multi-agent*)
    - Basic types of agents:
      - **Simple reflex agent:** Agent which acts only based on immediate input
      - **Reflex agent with state:** Agent which acts based on immediate input and an modelled understanding of the environment and how it changes over time
      - **Goal-based agent:** Agent which acts based on immediate, input, a modelled understanding of the environment, and a desired situation to achieve
      - **Utility-based agent:** Agent which acts based on immediate, input, a modelled understanding of the environment, a desired situation to achieve, and a performance measure to maximize

2 Problem Solving

- Environment: Observable, deterministic, discrete
- **State**: Possible arrangement of entities and agents in the environment
  - Transition: Change from an initial state to another state
    - Successor function: Mapping from the current state and action to a new state
      - Notation: $(\text{state}, \text{action}) \rightarrow \text{state}$
    - Goal state: Desired end state
    - Goal test: Condition which determines whether or not the current state is the goal state
      - Notation: $(\text{state}) \rightarrow \text{true/false}$
    - Path cost: Numeric cost to take a path
    - Includes the possibility of no change
  - E.g. For a two-square grid of a possibly dirty carpet with a vacuum cleaner, there are 4 possibilities for the dirty squares and 2 possibilities for the vacuum cleaner

3 Search Algorithms

3.1 Uninformed Algorithms

- **Breadth-first search:** Utilizes a queue
  - Time and space complexity: Bounded by the number of nodes the same distance away from the root as the goal node
  - Complete if branching factor is finite
- **Uniform-cost search:** Utilizes a priority queue to follow the path with least cost
  - Time and space complexity: Bounded by the branching factor to the power of the average action cost
  - Complete if branching factor is finite and step costs $\leq \epsilon$
- **Depth-first search:** Utilizes a stack
  - Time complexity: Bounded by the maximum depth of the tree
  - Space complexity: Bounded by the length of the maximum-length path, and its remaining unexpanded child nodes for each node on the path
  - Not complete
  - More space-efficient than breadth-first search
- **Iterative deepening (depth-first) search:** Depth-first search with maximum limited depth, which increases until the goal is found
  - Complete if branching factor is finite
  - Time complexity: Bounded by the number of nodes the same distance away from the root as the goal node
  - Space complexity: Bounded by the length of the goal path, and its remaining unexpanded child nodes for each node on the path

3.2 Informed Algorithms

- **Admissible heuristic:** Underestimation of the true cost from a given node to the goal
  - Notation for a function evaluating a heuristic: $h(n)$
  - Will provide improved time and space complexity over the worst time
  - To create an admissible heuristic, relax a rule/constraint and find the shortest solution given
  - **Manhattan distance:** Number of unrestricted 4-directional moves from the goal state
  - **Dominance:** Characteristic of an admissible heuristic which has a result greater than or equal to another admissible heuristic for all n
  - Given admissible heuristics $h_a(n)$ and $h_b(n)$, $h(n) = \max(h_a(n), h_b(n))$ is also admissible and dominates $h_a(n)$ and $h_b(n)$
- **Greedy search:** Best-first search algorithm which chooses subsequent nodes by minimizing direct cost to the goal
  - Complete: Only if state is checked for repeats
  - Time and space complexity: At worst, bounded by all nodes

- **A* search:** Best-first search algorithm which expands only the paths with less cost than the optimal cost to the goal
  - Evaluation function:

$$\begin{aligned} f(n) &= g(n) + h(n) \\ \text{where } f(n) &= \text{ estimated total cost of the path through } n \text{ to the goal} \\ g(n) &= \text{ cost so far to reach } n \\ h(n) &= \text{ heuristic-estimated cost from } n \text{ to the goal} \end{aligned}$$

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

3.3 Complexity of Algorithms

Figure 1: Complexity of Algorithms

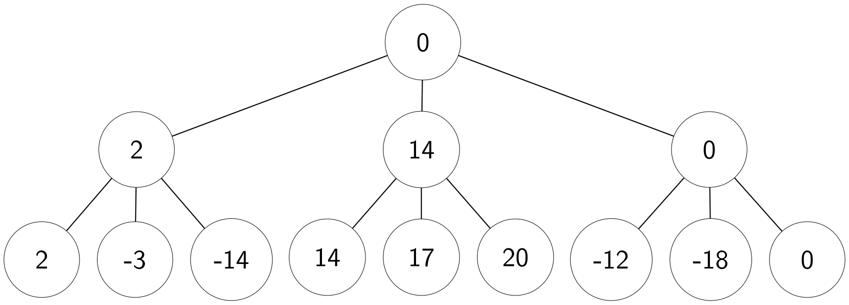
	Time	Space	Is the Goal Path Optimal?
Breadth-first search	$O(b^{d+1})$	$O(b^d)$	Only when $d = 1$ or when paths have no cost
Uniform-cost search	$O(b^{\lfloor \frac{C^*}{\epsilon} \rfloor})$	$O(b^{\lfloor \frac{C^*}{\epsilon} \rfloor})$	Yes
Depth-first search	$O(b^m)$	$O(b \cdot m)$	No
Iterative deepening search	$O(b^d)$	$O(b \cdot d)$	Yes
Greedy search	$O(b^m)$	$O(b^m)$	No
A* search	$O(b^m)$	$O(b^m)$	Yes
Minimax algorithm	$O(b^m)$ (with α/β, $O(b^{\frac{m}{2}})$)	$O(b \cdot m)$	Yes (with rational adversary)
Backtracking search			Yes
Hill climbing search			No
Simulated annealing search			Yes

$$\begin{aligned} \text{where } b &= \text{ branching factor} \\ d &= \text{ depth of the goal} \\ C^* &= \text{ cost of the optimal solution} \\ \epsilon &= \text{ the minimum cost of an action on the goal path} \\ m &= \text{ maximum depth} \end{aligned}$$

4 Game Playing and Adversarial Search

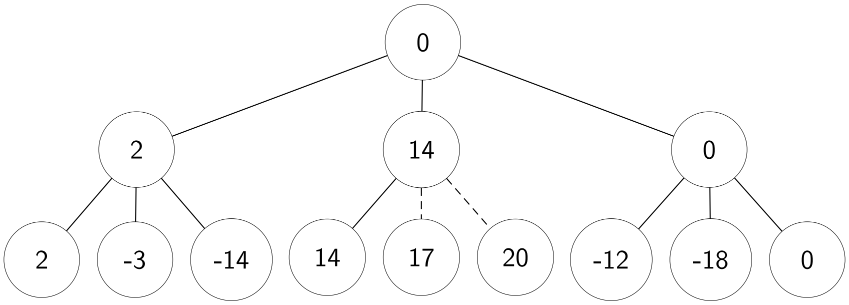
- Game playing: Scenario where a game with discrete states is played with two players, each seeking to maximize their position which minimizes the opponent's position
  - E.g. Chess, checkers
  - **Heuristic/evaluation function:** Estimate of the optimality of a game state
  - **Terminal state:** Game state which has no further states
  - Example diagram of game state evaluations in a tree: See figure 2

Figure 2: Game Tree (Minimizer first)



- **Minimax algorithm:** Decision-making rule which minimizes the potential loss for a worst-case scenario, given two entities who are alternatively trying to maximize and minimize a score
  - Performs the equivalent of a depth-first search
  - Each alternating level of the decision tree is minimized or maximized from the values in the levels below it
  - Complete: Yes, as long as the tree is finite
- α/β **pruning:** Keeping the maximum/minimum searched values at a given level so far to eliminate nodes which will not exceed a minimum/maximum
  - α: Lower bound on the potential value of a maximum node
  - β: Upper bound on the potential value of a minimum node
  - Resource for practice
  - See figure 3 for an example of pruned nodes (notated by dashed edges)

Figure 3: Game Tree with α/β Pruning (Minimizer first)



- Time, space, and optimal complexity of algorithms: See figure 1

5 Constraint Satisfaction Problems

- **Constraint satisfaction problem (CSP):** Problem where a set of constraints acts on a set of variables, and the solution is a set of variable values which satisfies all constraints
  - E.g. Sudoku, assiging resources to tasks, map colorings
  - Formulation of a CSP:
    - Specify the variables
    - Specify the domain D of possible values for the variables
    - Specify contraints
- **Backtracking search:** Depth-first search which assigns possible values to variables, and backtracks when a variable has no valid values
  - Complete: Yes
  - Runtime: $O(n!D^n)$ (actually $O(k^n)$ where $k =$ number of values, $n =$ number of variables)
    - For each of n variables, there are D options
      - Unary constraint: Restriction which only applies to a single variable
  - Space: $O(m)$ where m is the maximum depth
  - Improvements:
    - **Forward checking:** Assigning to each variable an early set of possibilities restricted by constraints, independent of other variables
    - **Arc consistency:** Whether for each value of a variable there is a valid value for all other variables sharing a binary constraint
      - Runtime complexity of enforcing arc consistency first: $O(n^2D^2)$
      - k consistency has runtime $O(n^kD^k)$
    - Heuristics:
      - **Minimum-remaining-values heuristic:** Assigning variables starting from those with the least possible number of values, to maximize early failure in order to minimize the number of changed values
      - **Maximum-degree heuristic:** Assigning variables starting from those which share the most constraints with other variables, to prune future options faster
        - Used for tiebreakers or starting
      - **Least-constraining-value heuristic:** Assigning variable values starting from the values which constrain other variables least, to minimize value failure
- **Local search:** Heuristic which uses iterative improvement on an arbitrary sub-optimal solution
  - Choose candidates by the variable(s) which violate constraints, using all domain values
  - Complete: No
  - Runtime: Unknown
  - Space: $O(n)$
  - Methods to choose the next solution:
    - **Hill-climbing search or gradient descent/ascent:** Local search algorithm which chooses following candidate which violates the fewest constraints

    - Terminates when it reaches a peak where no neighbor has a higher value
    - **Simulated annealing:** Hill-climbing search algorithm which allows for moderate sub-optimal moves (with decreased frequency over time) to escape local maxima and reach a global maximum
        - Complete: Yes

- **Genetic algorithm:** Search algorithm maintains a fixed-size set of solutions, from which the optimal individuals are selected to create the successive set of solutions, evolving to a global optimal state over time

6 Logical Inference

- **Entailment:** Logical connective which means the result is a consequence of the input
  - Notation: $\alpha \models \beta$
- **Satisfiability:** Whether a model exists where the logical sentence is true
  - Includes constraint satisfaction problems
  - Searches such as: Backtracking, local
  - **DPLL algorithm:** Backtracking algorithm which permutes the variables until all clauses are satisfied
    - **Pure symbol:** Symbol which is either never negated or always negated in each clause it appears
    - **Unit clause:** Clause which only contains a single literal
  - **WalkSAT algorithm:** Local non-deterministic algorithm which chooses a random false variable and changes it to true to attempt to find a solution
- **Deduction/Inference:** Given a knowledge base, whether a logical sentence is true (i.e. $KB \models S$)
  - $KB \models S$ if and only if $KB \land \neg S$ is unsatisfiable
  - **Knowledge base:** Collection of known true sentences
- **Conjuctive Normal Form (CNF):** Form of a logical statement which is a conjunction of disjunction(s) of literals
  - Steps to reduce a statement to CNF:
    - Replace equivalencies with a conjunction of implications
    - Replace implications with $\neg A \lor B$
    - Use DeMorgan's Law to apply negations to clauses
    - Use the Distributive Law to reduce clauses to CNF
- **Inference by resolution:** Process to reduce a set of CNF statements
  - Given a pair of clauses with complementary literals, resolve them to combine the clauses

7 Probability

7.1 Fundamentals of Probability

- Fundamentals:
  - Continuous variable: Variable which can be any continuous value between two given points defining the domain
    - Uniform distribution: Variable which has a equal possibility of being any value in the domain
  - Inverse of a probability: $P(a) = 1 - P(\neg a)$
  - Probability of at least one of two events: $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$
  - Probability of both events: $P(a \wedge b) = P(a, b) = P(a) + P(b) - P(a \vee b)$
- **Factor/probability table:** Set of probabilities of all possible values of a specific set of random variables
  - May not sum to 1 (in contrast with a probability table)
    - **Normalization constant:** Value equaling the sum of the probabilities, by which every probability is divided by in order to reduce a probability function to a total probability of 1
  - E.g. See figure 4.

Figure 4: Example Probability Table

	Toothache		¬ Toothache	
	Rough	¬ Rough	Rough	¬ Rough
Cavity	0.108	0.012	0.072	0.008
¬ Cavity	0.016	0.064	0.144	0.578

- **Probabilistic inference:** Using previous information to compute specific probabilities
- **Conditional probability:** Probability of one event occurring given the guarantee of another event happening
  - Notation: Probability of a given $b = P(a|b) = \frac{P(a,b)}{P(b)}$
  - Inverted: $P(a \wedge b) = P(a|b) \times P(b)$
  - **Conditional Probability Table (CPT):** Probability table showing the probability of the query variable being true given all possible combinations of values of the condition variables
    - E.g. See figure 5

Figure 5: Example of a Conditional Probability Table

A	B	P(C | A, B)
T	T	0.5
T	F	0.14
F	T	0.29
F	F	0.001

  - **Query variable:** Variable for which the value is sought

        - E.g. For variables A, B, C, D and the conditional probability $P(a|b,c)$, the query variable is a

    - **Evidence/condition variable:** Variable which is known

        - E.g. For variables A, B, C, D and the conditional probability $P(a|b,c)$, the condition variables are b, c

    - **Hidden variable:** Variable which is unrelated and not included

        - E.g. For variables A, B, C, D and the conditional probability $P(a|b,c)$, the hidden variable is d

- **Chain rule:** Derivation of the probability of two events:

$$P(a,b) = P(a)P(b|a) = P(b)P(a|b)$$
$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|x_1, \ldots, x_{i-1})$$

    - When conditional dependence is added:

$$P(a,b|c) = P(a|c)P(b|a,c)$$

        - Derived from the formula for conditional probability

- **Bayes' Rule:** Equation which allows conditional probability calculation given the independent probabilities and inverse conditional probability

$$P(B|A) = \frac{P(A|B) \times P(B)}{P(A)}$$

    - $P(A|B)$: Causal probability (e.g. probability that symptoms occur given a disease is present)
    - $P(B|A)$: Diagnostic probability (e.g. probability that a disease is present given the symptoms)

$$\begin{aligned} P(A \wedge B) &= P(A|B) \times P(B) = P(B|A) \times P(A) \\ P(B|A) \times P(B) &= P(A|B) \times P(A) \\ P(B|A) &= \frac{P(A|B) \times P(B)}{P(A)} \end{aligned}$$

- **Joint probability distribution:** Probability model consisting of all the possible variables in a model
    - Size: $d^n - 1$ where $d =$ size of the domain, $n =$ number of variables
    - Runtime: $O(d^n)$ where $d =$ size of the domain, $n =$ number of variables
    - Requires all data (amount proportional to d^n) to create parameters beforehand

- **Marginalization/summing out:** Determining the probability of a variable from a joint probability distribution by adding up the probabilities for all values of other variables, effectively eliminating the other variables
    - E.g. For variables A, B, C, $P(a,b) = P(a,b,c) + P(a,b,\neg c)$

- **Independence:** Property of two variables which do not rely on each other (i.e. knowing if one event occurred has no effect on the probability of the other event)

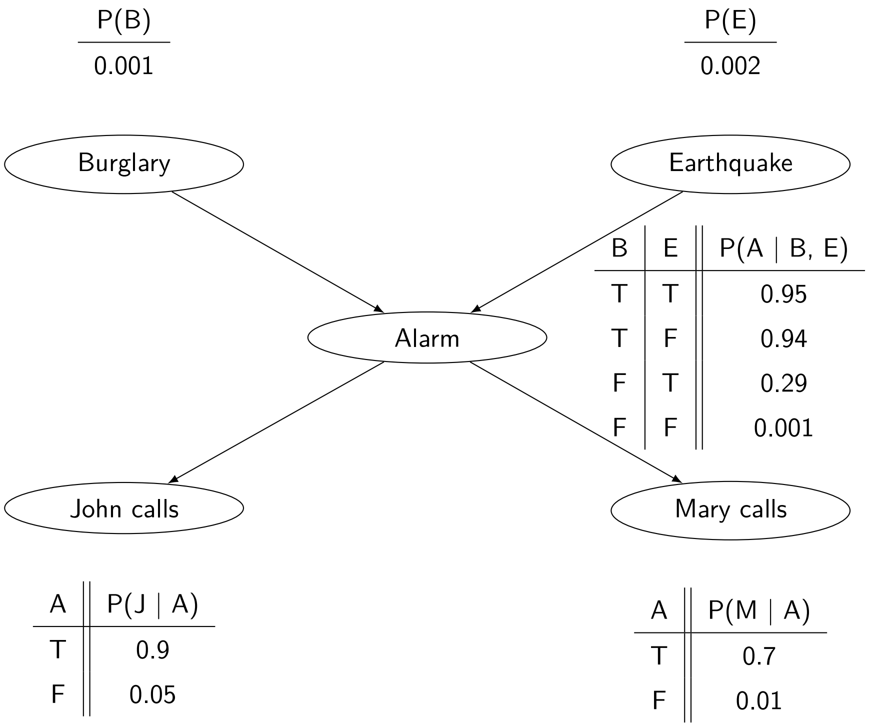
  - Definition: A, B are independent iff $P(A|B) = P(A)$ or $P(A,B) = P(A)P(B)$
  - In a Bayesian network, two variables are dependent if they share a parent nodes (including themselves)

- **Conditional independence:** Property of two variables given a third variable where, if the third variable is known, then knowing if one event occurred has no effect on the probability of the other event
  - Definition: A, B are conditionally independent given C if $P(A, B|C) = P(A|C)P(B|C)$
  - In a Bayesian network, two variables are conditiionally dependent if they share a child node (including themselves)

7.2 Bayesian Networks

- **Bayesian Network:** Acyclic digraph where each node (representing a variable) has a probability conditional on its parents
  - Nodes represent random variables, directed edges represent conditional dependence
  - Each node is independent of all nodes which do not share a child node with the original node
  - Each node is conditionally independent of all nodes which are not its descendents
  - Runtime: $O(nd^p(d-1))$ where n = number of variables, d = values in the domain, and p = number of parents
  - For an example, see figure 6

Figure 6: Example of a Bayesian Network



P(B)
0.001

P(E)
0.002

B	E	P(A | B, E)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(J | A)
T	0.9
F	0.05

A	P(M | A)
T	0.7
F	0.01

7.3 Exact Inference

- **Exact inference:** Methods of finding the exact probabilities of a specific solution to specific variables, through application of the chain rule to a probability distribution
  - Runtime: Exponential
  - Methods: Joint probability table, enumeration
  - Equation:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | parents(x_i))$$

- **Factor multiplication:** Determining the probability of a specific solution to specific variables by multiplying together the probabilities of the relevant relationships in the Bayesian network
  - E.g. Given the network in figure 6, to find the probabilities of the conditionally dependent variables M, A, B, E, see figure 7

Figure 7: Factor multiplication example

M	A	B	E	Probability
m	a	b	e	$P(m\|a)P(a, b\|e)$
m	a	b	$\neg e$	$P(m\|a)P(a, b\|\neg e)$
m	a	$\neg b$	e	$P(m\|a)P(a, \neg b\|e)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

  - **Inference by enumeration:** Determining the probability of a specific solution to all variables by multiplying together the probabilities of all relationships in the Bayesian network
    - E.g. Given the network in figure 6:

$$\begin{aligned} P(b, \neg e, \neg a, j, \neg m) &= P(b)P(\neg e)P(\neg a|b, \neg e)P(j|\neg a)P(\neg m|\neg a) \\ P(B|j, m) &= \alpha P(B) \sum_e P(e) \sum_a P(a|B, e)P(j|a)P(m|a) \\ &= \alpha P(B) \sum_e P(e) \Big[P(a|B, e)P(j|a)P(m|a) + P(\neg a|B, e)P(j|\neg a)P(m|\neg a) \Big] \\ &= \alpha P(B) \Big[P(a|B, e)P(j|a)P(m|a) + P(\neg a|B, e)P(j|\neg a)P(m|\neg a) \\ &\quad + P(a|B, \neg e)P(j|a)P(m|a) + P(\neg a|B, \neg e)P(j|\neg a)P(m|\neg a) \Big] \end{aligned}$$

  - Runtime: $O(nd^n)$ (n probability tables, d^n entries in the probability tables)

- **Inference by variable elimination:** Algorithm for efficiently executing inference by enumeration which combines relationships between probabilities and marginalizes out hidden variables to reduce factor size
  - Order can affect runtime; prefer starting from nodes which have no parents
  - Runtime: Exponential (worst-case) but often better than creating the entire table

- E.g. Given variables A, B, C, query and condition variables A, B, and factors $(C), (A, B, C)$, all factors involving hidden variable C are collected and combined in the first iteration of the algorithm and C is marginalized out to create the factor A, B
- E.g. Given the network in figure 6: (warning: expanded version is not actually correct)

$$P(B|j,m) = \alpha P(B) \sum_e P(e) \sum_a P(a|B,e) P(j|a) P(m|a)$$

$$\begin{aligned}
\text{Let } f_1(B,E,j,m) &= \sum_a P(a|B,e) P(j|a) P(m|a) \\
&= P(a|B,e)P(j|a)P(m|a) + P(\neg a|B,e)P(j|\neg a)P(m|\neg a)
\end{aligned}$$

$$P(B|j,m) = \alpha P(B) \sum_e P(e) f_1(B,E,j,m)$$

$$\begin{aligned}
\text{Let } f_2(B,j,m) &= \sum_e P(e) f_1(B,E,j,m) \\
&= P(e) f_1(B,e,j,m) + P(\neg e) f_1(B,\neg e,j,m)
\end{aligned}$$

$$P(B|j,m) = \alpha P(B) f_2(B,j,m)$$

$$\begin{aligned}
\text{Expanded: } P(B|j,m) &= P(B) f_2(B,j,m) \\
&= \alpha P(B) \Big[P(e) f_1(B,e,j,m) + P(\neg e) f_1(B,\neg e,j,m) \Big] \\
&= \alpha P(B) \Bigg\{ P(e) \Big[P(a|B,e)P(j|a)P(m|a) + P(\neg a|B,e)P(j|\neg a)P(m|\neg a) \Big] \\
&+ P(\neg e) \Big[P(a|B,\neg e)P(j|a)P(m|a) + P(\neg a|B,\neg e)P(j|\neg a)P(m|\neg a) \Big] \Bigg\} \\
&= \alpha \cdot 0.001 \Bigg\{ 0.002 \cdot \Big[0.95 \cdot 0.90 \cdot 0.7 + 0.05 \cdot 0.05 \cdot 0.01 \Big] \\
&+ 0.998 \Big[0.94 \cdot 0.90 \cdot 0.7 + 0.06 \cdot 0.05 \cdot 0.01 \Big] \Bigg\} \\
&= \begin{pmatrix} 0.284 \\ 0.716 \end{pmatrix}
\end{aligned}$$

- Algorithm:

```
FUNCTION order(vars)
        RETURN vars in a specific order
END FUNCTION

SET baynet to bayesian_network
SET hidden to baynet.hidden_vars
SET factors to baynet.factors_of_relationships
```

```
FOR each var in order(hidden)
        SET current_factors to factors.find(f WHERE f contains var)
        REMOVE current_factors from factors
        SET combined_factors to product_of(current_factors)
        SET combined_factors to marginalize(combined_factors, var)
        # var no longer appears in any factor
        ADD combined_factors to factors
        END IF
END FOR
RETURN factors
```

7.4 Approximate Inference

- **Approximate/sampling-based inference:** Methods of inferring the probabilities of a specific solution to specific variables by sampling to create an approximate solution
- **Rejection sampling:** Determining the approximate solution to specific variables by iterating through variables and for each, sampling the result of a probability multiple times (given all previous probabilities) and dividing the number of preferred results by the number of all results
  - Parents must be processed before their children (i.e. topological ordering)
  - Runtime: $O(NVD)$ where $N =$ number of samples, $V =$ number of variables, and $D =$ size of the domain
- **Gibbs sampling:** Determining the approximate solution to specific variables by fixing evidence variables, randomly assigning values to non-evidence variables, taking a non-evidence variable, sampling given the other assigned values, assigning the result, and repeating with another non-evidence variable
  - Results converge to the probability
  - **Markov blanket:** The parents, children, and all parents of the children of a node, which the node is dependent on, and so the node is independent of all other variables
  - **Markov Chain Monte Carlo (MCMC) algorithm:** Algorithm which alters a solution, uses the solution to create a next state, and repeats

7.5 Temporal Bayesian Networks

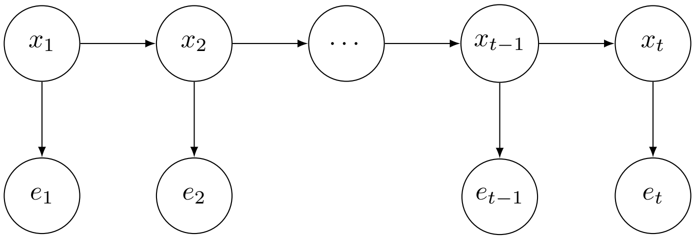
- **Temporal Bayesian Network:** Bayesian network where nodes represent the value of a variable at a specific time, and where each node may be a repeat of an antecedent node which represents the variable at a later time
  - **Tree width:** Number of edges crossed by a cut of the graph

7.6 Markov Models

- **Markov model:** State-change model which uses probabilities based on previous states
  - Notation: $e_t : e_1, e_2, \ldots, e_n$ where $e_t =$ known evidence of the state of the model at time t
  - **First-order Markov model:** Markov model which depends only on the previous state (i.e. $P(x_t|x_{t-1})$)
  - n**th-order Markov model:** Markov model which depends on the previous n states (i.e. $P(x_t|x_{t-1}, \ldots, x_{t-n})$)

- **Hidden Markov model:** Markov model which contains hidden states which are conditional upon evidence states
  - Notation: $P(x_t|e_{t-1}, e_{t-2}, \dots)$ where x_t = unknown state of the model at time t;
    - **Initial distribution:** $P(e_1)$
    - **Transition distribution:** $P(x_t|x_{t-1})$
    - **Emission distribution/corrector:** $P(e_t|x_t)$
  - For a diagram, see figure 8

Figure 8: Diagram of a Hidden Markov Model



- **Filtering:** Method of hidden Markov model inference which finds an approximate hidden probability given evidences of all previous probabilities (i.e. finding x_t given $e_{1:t}$)
  - Derivation: Given $P(x_{t-1}|e_{1:t-1})$, find $P(x_t|e_{1:t})$. (See equation 9.)

Figure 9: Derivation of Calculation for Filtering

$$
\begin{aligned}
P(x_t|e_{1:t}) &= P(x_t|e_{1:t-1}, e_t) && \\
&= \frac{1}{z} P(x_t, e_t|e_{1:t-1}) && \text{Move } e_t \text{ out of condition} \\
&= \frac{1}{z} \sum_{x_{t-1}} P(x_{t-1}, x_t, e_t|e_{1:t-1}) && \text{Reverse marginalization} \\
&= \frac{1}{z} \sum_{x_{t-1}} P(x_{t-1}, x_t|e_{1:t-1}) \cdot P(e_t|x_{t-1}, x_t, e_{1:t-1}) && \text{Chain rule} \\
&= \frac{1}{z} \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1}, e_{1:t-1}) \cdot P(e_t|x_{t-1}, x_t, e_{1:t-1}) && \text{Chain rule} \\
&= \frac{1}{z} \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1}) \cdot P(e_t|x_t) && \text{Conditional independence} \\
&= P(e_t|x_t) \cdot \frac{1}{z} \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1}) &&
\end{aligned}
$$

where $P(e_t|x_t)$ is the emission distribution,
$P(x_{t-1}|e_{1:t-1})$ is the provided probability and evidence, and
$P(x_t|x_{t-1})$ is the transition distribution.

  - Runtime: $O(TD^2)$ where $T =$ number of time periods and $D =$ degree of the hidden variable
  - Space complexity: $O(D)$ where $D =$ degree of the hidden variable
    - Only the last two time periods of probabilities of each degree is necessary

- **Smoothing (forward/backward algorithm):** Method of hidden Markov model inference which finds the approximated previous hidden probabilities given evidence of previous and future probabilities (i.e. finding x_t given $e_{1:T}$ where $1 \leq t < T$)
  - Derivation: Given $e_{1:t}$, find $P(x_k|e_{1:t})$. (See equation 10.)

Figure 10: Derivation of Calculation for Smoothing

$$
\begin{aligned}
P(x_k|e_{1:t}) &= \frac{1}{z}P(x_k, e_{k+1:t}|e_{1:k}) && \text{Chain rule}\\
&= \frac{1}{z}P(x_k|e_{1:k}) \cdot P(e_{k+1:t}|x_k, e_{1:k}) && \text{Chain rule}\\
&= \frac{1}{z}P(x_k|e_{1:k}) \cdot P(e_{k+1:t}|x_k) && \text{(Step 3) Conditional independence}\\
&= \frac{1}{z}P(x_k|e_{1:k}) \cdot \sum_{x_{k+1}} P(x_{k+1}, e_{k+1:t}|x_k) && \text{Reverse marginalization}\\
&= \frac{1}{z}P(x_k|e_{1:k}) \cdot \sum_{x_{k+1}} P(x_{k+1}|x_k) \cdot P(e_{k+1:t}|x_{k+1}, x_k) && \text{Chain rule}\\
&= \frac{1}{z}P(x_k|e_{1:k}) \cdot \sum_{x_{k+1}} P(x_{k+1}|x_k) \cdot P(e_{k+1:t}|x_{k+1}) && \text{Conditional independence}\\
&= \frac{1}{z}P(x_k|e_{1:k}) \cdot \sum_{x_{k+1}} P(x_{k+1}|x_k) \cdot P(e_{k+1}|x_{k+1}) \cdot P(e_{k+2:t}|x_{k+1}) && \text{Separating multiplied probabilities}
\end{aligned}
$$

where $P(x_k|e_{1:k})$ can be found as the result of filtering,
$P(x_{k+1}|x_k)$ is the transition distribution,
$P(e_{k+1}|x_{k+1})$ is the emission distribution, and
$P(e_{k+2:t}|x_{k+1})$ is recursively solved from Step 3.

- **Prediction:** Method of hidden Markov model inference which finds the approximated future hidden probabilities given evidence of all previous probabilities (i.e. finding x_t given $e_{1:t-\delta}$)
- **Most likely sequence of states:** Method of hidden Markov model inference which finds the most likely sequence of states given evidence of their probabilities (i.e. finding $\underset{x_{1:t}}{arg\,max}\ P(x_{1:t}|e_{1:t})$, the most likely states $x_{1:t}$ given $e_{1:t}$)
  - **Viterbi algorithm:** Algorithm which finds the most likely sequence of states
    - Algorithm:

```
# Initialize arrays
SET vitr[num_observ][num_states] to [][]
SET prev[num_observ][num_states] to [][]

FOR each state from 1 to n
        SET vitr[0][state] to original_prob[state] ÃŮ emission(state i,
            observation 0)
        SET prev[0][state] to 0
END FOR

# For each observation and state
FOR each observation current_observ from 1 to num_observ−1
        FOR each state current_state from 0 to num_states−1

                # Calculate all possible probabilities towards current
                    state
                SET max_prob to 0
                FOR each prev_state from 0 to n−1
                        SET prob to vitr[current_observ −1][prev_state] ×
                            transition(prev_state, current_state) ×
                            emission(current_state, current_observ)
                        IF prob > max_prob THEN
                                SET max_prob to prob
                        END IF
                END FOR

                # Set the max and argmax
                SET vitr[current_observ][current_state] to max_prob
                SET prev[current_observ][current_state] to vitr[
                    current_observ].index(max_prob)
        END FOR
END FOR

# Trace back to find most likely sequence
SET sequence to []
SET current_state to vitr[current_observ].index(max(vitr[current_observ])
    )
FOR each observation current_observ from t−1 to −1
        SET sequence to [current_state] + sequence
        SET current_state to prev[current_observ][current_state]
END FOR
```

```
RETURN sequence
```

8 Machine Learning

8.1 Introduction

- **Machine learning:** Approximation of a function which detects the classification of a problem given attributes of data
- **Classification:** Category of data
- **Attribute:** Characteristics of a problem using a variable (boolean, discrete, continuous, etc.)
- Epoch: Round of training data
- Types of machine learning:
  - Supervised learning
  - **Semi-supervised learning:** Machine learning technique which classifies data through processing a combination of pre-classified data and unclassified data
  - **Unsupervised learning:** Machine learning technique which classifies data and finds patterns through processing previously unclassified data
    - E.g. Clustering
  - **Reinforcement learning:** Machine learning technique which takes actions to obtain a score, and repeats to find the correct actions to obtain the optimal score
    - Used for robotics, games

8.2 Supervised Learning

- **Supervised learning:** Machine learning technique which classifies data through processing provided pre-classified data
- **Episodic problem (ML):** Situation in supervised learning where a dataset is built from discrete data points
  - E.g. Spam checker
- **Inductive learning:** Approximation of a function on the attributes of a problem which produces a label
  - Notation: Ideal function $f(features) \to label$; find $h \approx f$
  - **Label:** Resulting classification
  - Highly similar to linear regression
  - E.g. Finding a best-fit line to a set of datapoints
  - Choose the simplest and most generalizable model h (Occam's razor)
- **Hypothesis space:** Set of models h which are valid
  - Given n boolean functions, there are 2^n values in the dataset and 2^{2^n} unique possible assignments of results to the dataset
  - E.g. Neural networks, logistic regressions, support vector machines, random forests
- Dataset training:
  - Training data: Dataset which is applied to a neural network to provide neurons with weights and biases
    - The greater the data for training, the better the model created

  - **Validation data:** Dataset which is applied to a neural network with known optimal results to test accuracy
  - **Testing data:** Dataset which is applied to a neural network after training and validation
    - Cannot be the same as training data
    - The greater the data for testing, the better the estimate of accuracy
  - Cross-validation: Splitting ('folding') the original dataset into equal subsets, one for testing, one for validation, and the remaining for training
    - Strategy: Cross-validate multiple times with each fold to find the most accurate fold
  - **Overfitting:** Being overly specific with training data such that testing data is no longer accurate

8.3 Decision Trees

- **Decision tree:** Tree where each node represents a decision with multiple choices and each leaf represents a unique choice
  - Restricting hypothesis space: Limit on depth
  - Split paths on most informative features (which result in the lowest resulting entropy)
    - **Entropy:** Uncertainty of information (in this context, calculated as similarity of ratio between choices)
      - Notation: H where $0 \leq H \leq 1$
      - Unit: Bits
      - Calculation: $H(p_1, p_2, \dots, p_n) = -\sum_i p_i \log_2 p_i$
      - E.g. $H(0.5, 0.5) = -\frac{1}{2} \log \frac{1}{\frac{1}{2}} = 1$
      - E.g. $H(0, 1) = -0 \cdot \log \frac{1}{0} - 1 \cdot \log \frac{1}{1} = \lim_{x \to 0} x \log \frac{1}{x} = 0$
      - E.g. $H(\frac{1}{3}, \frac{2}{3}) = 0.918\dots$
    - To find the reduction in uncertainty, calculate:

$$H(root) - \sum_{i \in \text{children}} \frac{\text{samples of } i}{\text{samples of root}} H(i)$$

    - See diagram and patrons graph

8.4 Neural Networks

- **Neural network:** Combination of neuron structures through which data flows and is manipulated, to create a set of output values
  - **Bias:** Value which is added as a standalone neuron input (typically -1)
  - **Weight:** Value by which any neuron input is multipled
  - Neural network simulation
- **Activation function:** Simple function applied to the sum of weighted neuron inputs to create a neuron output value
  - Notation: $g : \mathbb{R} \to \mathbb{R}$
  - Types of functions:

- **Threshold function:** Activation function which has the output value -1 for negative input values, and the output value 1 for positive input values

$$a = \begin{cases} -1 & \text{if } in < 0 \\ 1 & \text{otherwise} \end{cases}$$

- **Sigmoid/logistic function:** Activation function which changes output value gradually from 0 to 1, and has output value 0.5 at input value 0

$$a = \frac{1}{1 + exp_e(-in)}$$

- **Rectifier Linear Unit (ReLU) function:** Activation function which has output value 0 for negative values, and output value equal to the input value for positive values

$$a = max(0, in) = \begin{cases} 0 & \text{if } in \leq 0 \\ in & \text{otherwise} \end{cases}$$

- Process of training a network:
  - **Squared error:**

$$Err = y - h_w(x)$$

$$\begin{aligned} \text{Squared error } E &= \frac{Err^2}{2} \\ &= \frac{(y - h_w(x))^2}{2} \end{aligned}$$

$$\begin{aligned} \text{where } x = &\ \text{input value} \\ y = &\ \text{true/ideal output value} \end{aligned}$$

- Using gradient descent to find the optimal value (one iteration per attribute):

$$\begin{aligned} \frac{\partial E}{\partial W_j} &= Err \times \frac{\partial E}{\partial W_j} \\ &= Err \times \frac{\partial}{\partial W_j}(y - g(\sum_{j=0}^{n} W_j x_j)) \\ &= -Err \times g'(in) \times x_j \end{aligned}$$

- Given the error, update the weight by:

$$W_j \leftarrow W_j + (\alpha \times Err \times g'(in) \times x_j)$$

$$\text{where } \alpha = \ \text{rate of learning correction}$$

  - **Back propagation:** Process of training a neural network by altering weights and biases to provide the intended results, using the outputs of training data

        - Involves calculus

$$\delta_{out} = (y - a_{out}) \cdot g'(in_{out})$$

- **Single-layer perceptron:** Neural network consisting of a single neuron
    - **Linearly separable:** Property of a multidimensional dataset which can be classified using a single linear function
        - A function can be represented by a single-layer perceptron if and only if its data is linearly separable
- **Multi-layer perceptron:** Neural network consisting of sets of multiple interconnected neurons where every node in each layer depends on at least one node in the layer immediately prior, and does not depend on nodes in other layers
    - **Hidden layer:** Layer of neurons which is neither the input layer nor the output layer
    - Acyclic digraph (i.e. has a topological order)
    - Multi-layer perceptron with 2 hidden layers (3 total layers of processing) can emulate any possible function
    - **Multi-task learning:** Model of multi-layer neural network which has consistent hidden layer processing but has multiple unique outputs
    - E.g. Image content differentiator
- **Convolutional layer:** Layer component of a neural network which handles prestructured input by abstracting and processing components of the input, often to fewer inputs
    - E.g. Image consisting of a 2-d grid of pixels, squares of which are taken and compressed
    - **Filter:** Structured subset of input values
    - **Pooling:** Combination of multiple inputs (often the depth of a 3D matrix) into a single output (e.g. summation, multiplication, averaging, maximization)
- **Recurrent network:** Acyclic digraph which is structured as a 2D matrix, with a set of input nodes flowing to a set of output nodes
    - E.g. Conversational bot

9 Flashcard Questions

- What characteristics does a task environment consist of?
    - Performance measure, Environment, Actuators, Sensors (PEAS)
- What does a problem formulation consist of?
    - Initial state, goal test, successor function, and cost function
- How is uniform-cost search different from breadth-first search?
    - Uniform-cost search uses a priority queue to follow the path of least cost
- Which search algorithm is iterative deepening search derived from?
    - Depth-first search
- How is iterative deepening search unique?
    - Maximum depth of depth-first search increases until the goal is found
- What is an admissible heuristic?
    - Function which underestimates the true cost to the goal
- What is a dominant heuristic?
    - Admissible heuristic which is greater than or equal to another admissible heuristic
- How is greedy/heuristic search different from A* search?
    - Heuristic search does not consider the distance already travelled
- What is the equation for A* search?

$$\begin{aligned} f(n) &= g(n) + h(n) \\ \text{where } f(n) &= \text{ estimated total cost of the path through } n \text{ to the goal} \\ g(n) &= \text{ cost so far to reach } n \\ h(n) &= \text{ heuristic-estimated cost from } n \text{ to the goal} \end{aligned}$$

- Which of α/β is the upper bound on the potential minimum node, and which is the lower bound on the potential maximum node?
    - α: Lower bound on the potential value of a maximum node
    - β: Upper bound on the potential value of a minimum node
- When does backtracking search backtrack?
    - When a variable has no possible valid values, given the assignments of the other variables
- What algorithm is DPLL derived from?
    - Backtracking search
- How does WalkSAT 'walk'?
    - The value of a random variable from a false clause is flipped
- What is a normalization constant?
    - A value which every probability is divided by, to reduce a probability function to a total probability of 1

- What is the difference between dependence and conditional dependence?
  - Dependence: Whether knowing a variable has an effect on the probability of another variable (share a parent node in Bayesian network)
  - Conditional dependence: Given an evidence variable(s), whether knowing a variable has an effect on the probability of another variable (share a child node in Bayesian network)
- What is the formula for the Chain Rule?

$$P(a,b) = P(a)P(b|a) = P(b)P(a|b)$$
$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i|x_1,\ldots,x_{i-1})$$

- What is the formula for Bayes' Rule?

$$P(B|A) = \frac{P(A|B) \times P(B)}{P(A)}$$

- What is rejection sampling?
  - Sampling each variable multiple times to find approximate probabilities of all variables
- What is Gibbs sampling?
  - Fixing evidence variables, randomly assigning values to non-evidence variables, sampling a non-evidence variable, and iterating through all non-evidence variables
- What is filtering?
  - Finding a hidden probability given evidence of all previous probabilities (i.e. finding x_t given $e_{1:t}$)
- What is smoothing?
  - Finding previous hidden probabilities given evidence of previous and future probabilities (i.e. finding x_t given $e_{1:T}$ where $1 <= t < T$)
- What does the Viterbi algorithm do?
  - Find the most likely sequence of states ending in xt
- How do you avoid overfitting?
  - Use relatively less training data
- What is the equation used for entropy?
  - $H(p_1,p_2,\ldots,p_n) = -\sum_i p_i \log_2 p_i$
- What is the equation used for reduction in uncertainty?
  - $H(root) - \sum_{i\in\text{children}} \frac{\text{samples of } i}{\text{samples of root}} H(i)$
- What is the entropy of a 50/50 choice?
  - 1
- What is the entropy of a 0/100 choice?
  - 0
- Do you make a choice at a decision tree to increase or decrease entropy?
  - Decrease entropy (i.e. maximize reduction of entropy)

- What is the equation for a threshold function?

$$a = \begin{cases} -1 & \text{if } in < 0 \\ 1 & \text{otherwise} \end{cases}$$

- What is the equation for a ReLU function?

$$a = max(0, in)$$