Artificial Intelligence

Personal notes based on lecture material and assigned readings from Princeton's [COS 402: Artificial Intelligence](), taught by Xiaoyan Li, Sebastian Seung, and Elad Hazan.

Table of Contents

Tree and graph search

- Algorithm properties
  - Completeness – an algorithm is complete if it terminates with a solution when one exists
  - Optimality – an algorithm is optimal if it finds the lowest path cost solution, where the path cost is (as of now) the sum of the individual step costs along the path
- Tree vs. graph search
  - Both maintain a *frontier set* of nodes under current consideration
  - Graph search maintains an *explored set* of nodes that have been visited
- Breadth-first search
  - Complete
  - Optimal if path cost is non-decreasing function of node depth
  - Parameters – branching factor b, depth d
  - Time complexity is $O(b^d)$
  - Space complexity is $O(b^d)$
    - $O(b^{d-1})$ nodes in explored set and $O(b^d)$ nodes in frontier
    - Limiting factor for breadth-first search
- Depth-first search
  - Tree-search version
    - Not complete in finite or infinite state spaces
    - Not optimal
    - Time complexity is $O(b^m)$, where m is the max depth of any node

            - This value can be larger than the size of the state space
        - Space complexity is $O(bm)$
            - Need only store single path from root to leaf node, plus unexpanded sibling nodes for each node on path
    - Graph-search version
        - Complete in finite state space, but not in infinite
        - Not optimal
        - Time complexity bounded by size of state space
        - Space complexity – no advantage over BFS
- Depth-limited search
    - Not complete if depth limit $l < d$
    - Not optimal if $l > d$
    - Time complexity is $O(b^l)$
    - Space complexity is $O(bl)$
- Iterative deepening search
    - Combines benefits of breadth-first and depth-first search
    - Complete when branching factor is finite
    - Optimal when path cost is nondecreasing function of node depth
    - Space complexity is $O(bd)$
    - Time complexity is $O(b^d)$, asymptotically the same as breadth-first search
- Bidirectional search
    - Uses breadth-first search in both directions (to and from goal state)
    - Time complexity is $O(b^{d/2})$
    - Space complexity is $O(b^{d/2})$
- Best-first search
    - Greedy best-first search
        - Uses $f(n) = h(n)$
        - Not optimal
        - Completeness
            - Tree-search version is not complete, even in finite state spaces
            - Graph-search version is complete only in finite state spaces
        - Worst-case time and space complexity of $O(b^m)$
    - A* search
        - Uses $f(n) = g(n) + h(n)$
        - Is both complete and optimal, given certain conditions (see below)
        - In most cases, number of states within goal contour search space is exponential in the length of the solution
        - Space complexity even worse, since it keeps all generated nodes in memory (like other graph-search algorithms)
- Search heuristic properties
    - Admissibility
        - A heuristic is admissible if it never overestimates cost to reach goal
    - Consistency

            - A heuristic is consistent if it satisfies the triangle inequality – the heuristic value at node n is no greater than the heuristic value at node n’ plus the step cost from n to n’
    - Facts
        - All consistent heuristics are admissible, but all admissible heuristics are not consistent
        - Tree-search version of A* is optimal if h(n) is admissible
        - Graph-search version of A* is optimal if h(n) is consistent
- Comparison of tree-search algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First[5]	Depth-Limited	Iterative Deepening	Bidirectional	A*
Complete?	Yes[1]	Yes[1, 2]	No	No	Yes[1]	Yes[1, 4]	Yes
Time	$O(b^d)$	$O(b^{1+[C^*/e]})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$	$O(b^{\epsilon d})$[8]
Space	$O(b^d)$	$O(b^{1+[C^*/e]})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$	Exponential[9]
Optimal?	Yes[3]	Yes	No	No	Yes[3]	Yes[3, 4]	Yes[6, 7]

[1] complete if branching factor b is finite
[2] complete if step costs $\geq \epsilon$
[3] optimal if path cost is a non-decreasing function of depth d
[4] if both directions use BFS
[5] graph-search DFS is complete in finite state spaces, and its space/time complexities are bounded by the size of the state space
[6] tree-search version is optimal if $h(n)$ is admissible; graph-search version is optimal if $h(n)$ is consistent
[7] A* is *optimally efficient* – no other optimal algorithm is guaranteed to expand fewer nodes, except perhaps through tie-breaking among nodes with $f(n) = C^*$ (on goal contour)
[8] bound holds in state space with single goal and constant step costs; extra cost is proportional to number of near-optimal goals
[9] see IDA*, RBFS, (S)MA*

Adversarial search

- Minimax algorithm
    - Minimax value of a node is MAX’s utility from being in that state, assuming both players play optimally
    - Minimax values are backed up through the tree from the leaves
    - Time complexity of $O(b^m)$ and space complexity of $O(bm)$ or $O(m)$
        - Impractical for real games
- Alpha-beta pruning
    - Eliminates subtrees that could not possible affect solution

Propositional logic

- Terminology
  - If sentence α is true in m, then m satisfies α or m is a model of α
  - Entailment: α entails β if and only if in every model in which α is true, β is true, or, $M(\alpha) \subseteq M(\beta)$
  - Logical equivalence: two sentences α and β are logically equivalent if they are true in the same set of models
  - Validity: A sentence is valid if it is true in *all* models
  - Satisfiability: A sentence is satisifiable if its true in *some* model
- Basic results
  - Sentences α and β are equivalent if and only if $\alpha \vDash \beta$ and $\beta \vDash \alpha$
  - $\alpha \vDash \beta$ if and only if the sentence $\alpha \Longrightarrow \beta$ is valid
  - $\alpha \vDash \beta$ if and only if $\alpha \wedge \neg\beta$ is unsatisfiable
    - To prove that statement α entails β, just show that $\alpha \wedge \neg\beta$ is unsatisfiable
- Inference rules
  - Modus Ponens – if $\alpha \Longrightarrow \beta$ and α holds, then β is true
  - And-Elimination – if $\alpha \wedge \beta$ holds, then α is true
- Resolution (algorithm)
  - Couples resolution (inference rule) with any complete search algorithm
  - Unit resolution rule
    - Take a clause and a literal and produces new clause
  - Full resolution rule
    - Takes 2 clauses (with complementary literals l_i and m_j) and produces new literal containing all literals of 2 original clauses *except* the 2 complementary literals
    - Additionally, the resulting clause only contains one copy of each literal

$$\frac{l_1 \vee l_2 \vee \ldots \vee l_k, \qquad m_1 \vee m_2 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_i \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

  - Can be used to determine whether $KB \vDash \alpha$
    - Strategy: show that $KB \wedge \neg\alpha$ is unsatisfiable
    - Requires first converting KB, α to conjunctive normal form
      - Clauses contain only ORed literals (or their negation)
      - Clauses are ANDed together
    - Resolution rule is applied to pairs of clauses from $KB \wedge \neg\alpha$ with complementary literals
    - Process terminates when
      - No new clauses can be added – KB does not entail α
      - Two clauses resolve to the *empty* clause – KB entails α (sentence is unsatisfiable)
        - Empty clause represents a contradiction (i.e. $P \wedge \neg P$)
    - Is complete – proved through ground resolution theorem

- Clause types
  - Definite clauses – disjunction of literals of which *exactly one* is positive
  - Horn clause – disjunction of literals of which *at most one* is positive
- Forward Chaining
  - FC-ENTAILS? (KB, q) returns whether a knowledge base KB entails a query q
  - Algorithm steps
    - Begins with set of known facts (positive literals) in knowledge base
    - If all premises of an implication are known, its conclusion is added to set of facts
    - Process continues until query q is added (returns true) or until no further inferences can be made (returns false)
  - Time complexity of $O(n)$, where n is the number of clauses
  - Is sound – every inference is an application of Modus Ponens
  - Is complete – every entailed atomic sentence will be derived (can be proved)
- Backward Chaining
  - Algorithm steps
    - Works backward from query q
    - If q is true, no work needed (returns true)
    - Otherwise, finds implications whose conclusion is q
    - Works backward until it reaches set of known facts
  - Time complexity of $O(n)$ in theory, but often much less in practice
- DPLL
  - DPLL-SATISFIABLE? (s) returns true or false
  - Recursive, depth-first search of possible models
  - Uses three heuristics
    - *Early termination* – if a literal is true, entire clause is true
    - *Pure symbol heuristic* – symbol that appears with sign in all clauses (value can be assigned, so as to make all clauses true)
    - *Unit clause heuristic* – a clause with a single literal (value must be assigned)
  - Various tricks can be used to improve performance
    - Intelligent backtracking (combined with conflict clause learning)
    - Random restarts
    - Clever indexing
- WalkSAT
  - WALKSAT$(clauses,\ p,\ max_flips)$ returns satisfying model or failure
  - On each iteration, algorithm picks unsatisfied clause and flips a symbol
    - Flips symbol that minimizes number of unsatisfied clauses or, with probability p, picks randomly
  - When algorithm returns failure, two possible causes
    - Sentence is unsatisfiable
    - Algorithm needs more time

Bayesian networks

- Bayes Rule

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

- Bayesian networks are a representation of the full joint probability

$$P(X_1, X_2, \dots, X_n) = \prod_i^n P(x_i \mid \text{Parents }(X_i))$$

- Conditional independence in Bayes Nets
    - Node is conditionally independent of non-descendants given its parents
    - Node is conditionally independent of *all* other nodes given its Markov blanket
        - Markov blanket consists of a node's parents, children, and spouses
- Exact inference
    - Enumeration
        - Uses $P(X \mid e) = \alpha\, P(X, e) = \alpha \sum_y P(X, e, y)$ and full joint distribution to answer queries
        - ENUMERATION-ASK(X, e, bn) returns probability distribution $P(X \mid e)$
            - Uses depth-first recursion (similar to DPLL for satisfiability)
            - Space complexity is $O(n)$, where n is the number of Boolean variables in the network
            - Time complexity is $O(2^n)$
    - Variable elimination
        - Dynamic programming approach
        - Joint probability expressions are evaluated from right-to-left and intermediate results are stored
        - Uses matrices and pointwise product
        - Time and space complexity
            - Linear in size of the network for singly connected networks (polytrees)
            - Exponential for multiply connected networks
- Approximate inference
    - Direct sampling
        - Variables in probability expression are sampled in topological order
    - Rejection sampling
        - Generates samples from prior distribution specified by network, rejecting those that do not match evidence
        - Problem: rejects too many samples
            - Fraction of samples consistent with evidence e drops exponentially as number of evidence variables grows
    - Likelihood weighting
        - Fixes values for evidence variables E

        - Samples non-evidence variables, in any topological order
        - Each event is weighted by its likelihood (the product of the conditional probabilities of each evidence variable, given its parents)
    - Gibbs sampling (MCMC)
        - Evidence variables are fixed to their observed values
        - Nonevidence variables are randomly initialized
        - Generates a next state by randomly sampling for one of the nonevidence variables X_i
            - Sampling is done conditioned on the current values of the variables in the Markov blanket of X_i
        - Each state visited during process is a sample that contributes to the estimate for the query variable

Temporal models

- Satisfy Markov assumption
    - Current state depends only on fixed, finite number of previous states
- Satisfy sensor (Markov) assumption
    - Current state is sufficient to determine current output (sensor) value
- Inference tasks
    - Filtering and prediction – computing belief state, $P(X_t \mid e_{1:t})$, or future state, $P(X_{t+k} \mid e_{1:t})$, given all evidence to date

        - Determining $P(X_{t+1} \mid e_{1:t+1})$ from $P(X_t \mid e_{1:t})$

$$P(X_{t+1} \mid e_{1:t+1}) = \alpha\, P(e_{t+1} \mid X_{t+1})\, P(X_{t+1} \mid e_{1:t})$$

where

$$P(X_{t+1} \mid e_{1:t}) = \sum_{x_t} P(X_{t+1} \mid x_t)\, P(x_t \mid e_{1:t})$$

From this, define the FORWARD process as

$$f_{1:t+1} = \alpha\ \text{FORWARD}(f_{1:t}, e_{t+1})$$

        - Prediction is equivalent to filtering without evidence

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

    - Smoothing – computing past state, $P(X_k \mid e_{1:t})$, given all evidence to date

        - Use forward message, $f_{1:k}$, and *backward* message, $b_{k+1:t}$

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

where

$$b_{k+1:t} = \sum_{x_{k+1}} P(e_{k+1} \mid x_{k+1}) P(e_{k+2:t} \mid x_{k+1}) P(x_{k+1} \mid X_k)$$
$$= \text{BACKWARD}(b_{k+2:t}, e_{k+1})$$

- Time complexity of smoothing for a particular time step k with respect to evidence $e_{1:t}$ is $O(t)$

- To determine $P(X_k \mid e_{1:t})$ for all $1 \leq k \leq t$ in time $O(t)$, as opposed to time $O(t^2)$, and $O(|\boldsymbol{f}|t)$ space, where $|\boldsymbol{f}|$ is the size of the forward message, can use the *forward-backward algorithm*

    - FORWARD-BACKWARD($\boldsymbol{ev}$, $prior$) returns a vector of probability distributions $P(X_k \mid e_{1:t})$ for $1 \leq k \leq t$

        - Local variables
            - Vector of forward messages, **fv**
            - Representation of backward message, $\boldsymbol{b}$
            - Vector of smooth estimates for steps 1…t, **sv**
        - Initialization
            - **fv**[0] = $P(X_0)$
            - $\boldsymbol{b} = \vec{1}$
        - For i from 1 to t
            - **fv**[i] ⟵ FORWARD(**fv**[$i-1$], **ev**[i])
        - For i from t down to 1
            - **sv**[i] ⟵ NORMALIZE(**fv**[i] × $\boldsymbol{b}$)
            - $\boldsymbol{b}$ ⟵ BACKWARD($\boldsymbol{b}$, **ev**[i])
        - Return **sv**

- Most likely explanation – given sequence of observations, find most likely sequence of states, i.e. compute $\text{argmax}_{x1:t}\, P(x_{1:t} \mid e_{1:t})$

    - Viterbi algorithm
        - Time complexity (like filtering) is $O(t)$
        - Space complexity (unlike filtering) is $O(t)$

        - Steps
            - Final maximization

$$\max_{x_{0:t}} P(x_{0:t}, e_{1:t}) = \max_{x_t} \left[\max_{x_{0:t-1}} P(x_{0:t}, e_{1:t}) \right]$$

- Recursive step

$$\max_{x_{0:t-1}} P(x_{0:t}, e_{1:t})$$
$$= \max_{x_{t-1}} \left[P(x_t \mid x_{t-1})\, P(e_t \mid x_t) \max_{x_{0:t-2}} P(x_{0:t-1}, e_{1:t-1}) \right]$$

- Base case

$$\max_{x_{0:t-1}} P(x_{0:t}, e_{1:t}) = \mathrm{P}(\mathrm{x}_0)$$

- Hidden Markov Models (HMMs)
  - Temporal, probabilistic model in which state of process is described by a *single discrete* random variable
    - A model with 2 or more state variables can still be fit into the HMM framework by combining the variables into a single "megavariable"
  - Elegant matrix implementation of all basic algorithms possible
- Kalman filters
  - Filtering over continuous state variables
  - Required properties to do filtering
    - Probability distribution $P(X_t \mid e_{1:t})$ is Gaussian
    - Transition model $P(X_t \mid x_t)$ is linear Gaussian
    - Sensor model $P(e_t \mid X_t)$ is linear Gaussian
    - Consequence: one-step predicted distribution will also be Gaussian

$$P(X_{t+1} \mid e_{1:t}) = \int P(X_{t+1} \mid x_t)\, P(x_t \mid e_{1:t})\, d\boldsymbol{x_t}$$

    - If prediction $P(X_{t+1} \mid e_{1:t})$ is Gaussian and sensor model $P(e_{t+1} \mid X_{t+1})$ is linear Gaussian, then updated distribution is also Gaussian

$$P(X_{t+1} \mid e_{1:t+1}) = \alpha\, P(e_{t+1} \mid X_{t+1})\, P(X_{t+1} \mid e_{1:t})$$

- Dynamic Bayesian Networks (DBNs)
  - Bayesian network that represents temporal probability model
  - DBNs and HMMs
    - Every HMM can be represented as a DBN with a single state variable and single evidence variable
    - Every discrete-variable DBN can be represented as an HMM
    - Advantage of DBNs
      - By decomposing a complex state into its constituent variables, DBNs can take advantage of sparseness in temporal model
      - A DBN with 20 Boolean state variables, each with 3 parents, will have 20 x 2^3 = 160 probabilities in its transition model

            - Corresponding HMM has 2^{20} states or roughly a trillion probabilities in its transition matrix
    - DBNs and Kalman filters
        - Every Kalman filter model can be represented as a DBN with continuous variables and linear Gaussian conditional distributions
        - However, not every DBN can be represented as a Kalman filter model
    - Inference in DBNs
        - Possible approach: adapt variable elimination approach to filtering process, summing out state variables of previous time step to get distribution for new time step
            - Can achieve constant time and space per filtering update, but constant is exponential in number of state variables
        - Approximate inference solution: *particle filtering*
            - Algorithm steps
                - Initialize population of N samples by sampling from prior distribution $P(X_0)$
                - For each time step
                    - Propagate each sample forward by sampling next state value, given current, via $P(X_{t+1} \mid x_t)$
                    - Weight each sample by likelihood that it assigns to new evidence $P(e_{t+1} \mid x_{t+1})$
                    - Resample population to generate new population of N samples
                        - New samples are selected from current population, with probability of selection proportional to the weight of the sample
            - Is consistent – gives correct probabilities as N tends to infinity
            - Efficient performance in practice

Markov decision processes

- Key ideas
    - Discounted rewards
        - If discount factor γ is less than 1, utility of an infinite state sequence is still finite

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

    - Expected utility
        - Expected utility obtained by executing a policy π starting in s is

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

    - Optimal policy

        - Optimal policy is the policy with the highest expected utility

$$\pi_s^* = \text{argmax}_\pi \, U^\pi(s)$$

        - As a function of the state

$$\pi^*(s) = \text{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) \, U(s')$$

- Value iteration
    - Bellman equation for utilities

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

    - Algorithm
        - Bellman update step

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) \, U_i(s')$$

        - VALUE-ITERATION(mdp, ϵ) returns a utility function
            - Local variables
                - U, U' – vectors of utilities for states in S, initially zero
                - δ – max change in utility of any state in an iteration
            - Do while $(\delta < \epsilon(1 - \gamma)/\gamma)$
                - Initialization
                    - $U \leftarrow U'$
                    - $\delta \leftarrow 0$
                - For each state s in S
                    - $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) \, U'[s]$
                    - If $|U'[s] - U[s]| > \delta$
                        - Set $\delta \leftarrow |U'[s] - U[s]|$
            - Return U
    - Proof of convergence
        - Bellman update is a *contraction* by a factor of γ on the space of utility vectors with only 1 fixed point
- Policy iteration
    - Two main parts
        - Policy evaluation – given a policy π_i, compute the associated utility function $U_i = U^{\pi_i}$
            - Uses simplified version of Bellman equation (policy is fixed)

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi_i(s)) \, U_i(s')$$

                - Can do approximate policy evaluation by performing a fixed number k of simplified value iteration steps

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s))\, U_i(s')$$

        - Policy improvement – calculate a new policy, using a one-step look-ahead based on U_i
- Algorithm
    - POLICY-ITERATION(mdp) returns a policy
        - Local variables
            - U – vectors of utilities for states in S, initially zero
            - π – policy vector indexed by state, initially random
        - Repeat
            - Initialization
                - $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
                - unchanged? $\leftarrow$ true
            - For each state s in S
                - If $\max_{a} \sum_{s'} P(s' \mid s, a)\, U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\, U[s']$ then do
                    - $\pi[s] \leftarrow \mathrm{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
                    - unchanged? $\leftarrow$ false
            - Break if unchanged? $=$ false
        - Return π