**Artificial Intelligence**

Personal notes based on lecture material and assigned readings from Princeton's [COS 402: Artificial Intelligence](http://www.cs.princeton.edu/courses/archive/fall15/cos402/), taught by Xiaoyan Li, Sebastian Seung, and Elad Hazan.

Table of Contents

[Tree and graph search 1](#_Toc485211028)

[Adversarial search 3](#_Toc485211029)

[Propositional logic 3](#_Toc485211030)

[Bayesian networks 5](#_Toc485211031)

[Temporal models 7](#_Toc485211032)

[Markov decision processes 10](#_Toc485211033)

# 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 , depth
  + Time complexity is
  + Space complexity is
    - nodes in explored set and 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 , where is the max depth of any node
      * This value can be larger than the size of the state space
    - Space complexity is
      * 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
  + Not optimal if
  + Time complexity is
  + Space complexity is
* 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
  + Time complexity is , asymptotically the same as breadth-first search
* Bidirectional search
  + Uses breadth-first search in both directions (to and from goal state)
  + Time complexity is
  + Space complexity is
* Best-first search
  + Greedy best-first search
    - Uses
    - 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
  + A\* search
    - Uses
    - 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 is no greater than the heuristic value at node plus the step cost from to
  + 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-First5 | Depth-Limited | Iterative Deepening | Bidirectional | A\* |
| Complete? | Yes1 | Yes1, 2 | No | No | Yes1 | Yes1, 4 | Yes |
| Time |  |  |  |  |  |  | 8 |
| Space |  |  |  |  |  |  | Exponential9 |
| Optimal? | Yes3 | Yes | No | No | Yes3 | Yes3, 4 | Yes6, 7 |

1 complete if branching factor is finite

2 complete if step costs

3 optimal if path cost is a non-decreasing function of depth

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 is admissible; graph-search version is optimal if 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 (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 and space complexity of or
    - Impractical for real games
* Alpha-beta pruning
  + Eliminates subtrees that could not possible affect solution

# Propositional logic

* Terminology
  + If sentence is true in , then satisfies or is a model of
  + Entailment: entails if and only if in every model in which is true, is true, or,
  + Logical equivalence: two sentences and 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 and are equivalent if and only if and
  + if and only if the sentence is valid
  + if and only if is unsatisfiable
    - To prove that statement entails , just show that is unsatisfiable
* Inference rules
  + Modus Ponens – if and holds, then is true
  + And-Elimination – if holds, then 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 and ) 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
  + Can be used to determine whether
    - Strategy: show that is unsatisfiable
    - Requires first converting , 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 with complementary literals
    - Process terminates when
      * No new clauses can be added – does not entail
      * Two clauses resolve to the *empty* clause – entails (sentence is unsatisfiable)
        + Empty clause represents a contradiction (i.e. )
    - 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
  + returns whether a knowledge base entails a query
  + 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 is added (returns true) or until no further inferences can be made (returns false)
  + Time complexity of , where 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
    - If is true, no work needed (returns true)
    - Otherwise, finds implications whose conclusion is
    - Works backward until it reaches set of known facts
  + Time complexity of in theory, but often much less in practice
* DPLL
  + 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
  + 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 , picks randomly
  + When algorithm returns failure, two possible causes
    - Sentence is unsatisfiable
    - Algorithm needs more time

# Bayesian networks

* Bayes Rule
* Bayesian networks are a representation of the full joint probability
* 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 and full joint distribution to answer queries
    - returns probability distribution
      * Uses depth-first recursion (similar to DPLL for satisfiability)
      * Space complexity is , where is the number of Boolean variables in the network
      * Time complexity is
  + 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 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
      * Sampling is done conditioned on the current values of the variables in the Markov blanket of
    - 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, , or future state, , given all evidence to date  
    - Determining from

where

From this, define the Forward process as

* + - Prediction is equivalent to filtering without evidence
  + Smoothing – computing past state, , given all evidence to date
    - Use forward message, , and *backward* message,

where

* + - Time complexity of smoothing for a particular time step with respect to evidence is
    - To determine for all in time , as opposed to time , and space, where is the size of the forward message, can use the *forward-backward algorithm*
      * returns a vector of probability distributions for
        + Local variables

Vector of forward messages,

Representation of backward message,

Vector of smooth estimates for steps 1…t,

* + - * + Initialization
        + For from 1 to
        + For from down to 1

, )

* + - * + Return
  + Most likely explanation – given sequence of observations, find most likely sequence of states, i.e. compute
    - Viterbi algorithm
      * Time complexity (like filtering) is
      * Space complexity (unlike filtering) is
      * Steps
        + Final maximization
        + Recursive step
        + Base case
* 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 is Gaussian
    - Transition model is linear Gaussian
    - Sensor model is linear Gaussian
    - Consequence: one-step predicted distribution will also be Gaussian
    - If prediction is Gaussian and sensor model is linear Gaussian, then updated distribution is also Gaussian
* 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 23 = 160 probabilities in its transition model
      * Corresponding HMM has 220 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
        + For each time step

Propagate each sample forward by sampling next state value, given current, via

Weight each sample by likelihood that it assigns to new evidence

Resample population to generate new population of 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 tends to infinity
      * Efficient performance in practice

# Markov decision processes

* Key ideas
  + Discounted rewards
    - If discount factor is less than 1, utility of an infinite state sequence is still finite
  + Expected utility
    - Expected utility obtained by executing a policy starting in is
  + Optimal policy
    - Optimal policy is the policy with the highest expected utility
    - As a function of the state
* Value iteration
  + Bellman equation for utilities
  + Algorithm
    - Bellman update step
    - returns a utility function
      * Local variables
        + – vectors of utilities for states in S, initially zero
        + – max change in utility of any state in an iteration
      * Do while (
        + Initialization
        + For each state in

If

Set

* + - * Return
  + Proof of convergence
    - Bellman update is a *contraction* by a factor of on the space of utility vectors with only 1 fixed point
* Policy iteration
  + Two main parts
    - Policy evaluation – given a policy , compute the associated utility function
      * Uses simplified version of Bellman equation (policy is fixed)
* Can do approximate policy evaluation by performing a fixed number of simplified value iteration steps
  + - Policy improvement – calculate a new policy, using a one-step look-ahead based on
  + Algorithm
    - returns a policy
      * Local variables
        + – vectors of utilities for states in S, initially zero
        + – policy vector indexed by state, initially random
      * Repeat
        + Initialization
        + For each state in

If

then do

* + - * + Break if
      * Return