|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Notation | | | | | |  | | |  | |
| [indicator function]  {set of unique elements}  Product | | | | | |  | | |  | |
| Week 1 | | | | | |  | | |  | |
| Week 2 | | | | | |  | | |  | |
| Week 3 Search | | | | | |  | | |  | |
| **Reflex** > **States** (search, MDPs, games) > **Variables** (CSP, Markov Networks, Bayesian Network)> **Logic**   * Search is powerful iff well understood world states and actions. So key skill is defining and decomposing problems into states * State is a summary of past actions sufficient to choose future actions optimally. State collapses tree into having only info that we use to choose future actions optimally to avoid the exponential blow ups * Search problem is an abstraction that provides a clean interface to the world to find optimality * Paradigm is the trifecta: modelling, learning (structured perceptron), inference | | | | | | Definition of search problem   1. Sstart Start state 2. Action(s): possible actions from states 3. Cost(s,a): cost of the action from state s 4. Succ(s,a): successor state from state s given action a 5. IsEnd(s): reached end state? | | | | |
| **1. Tree search**   * Enumerating all states and actions but not done in practice so we build algorithms to help instantiate a search tree. * Tree search is memory efficiency but exponential time complexity | | | | | | **2. DP**  DFS with reuse | | | | **3. UCS** The analog of DP for BFS |
| **1a. Exhaustive**  If b actions per state and max depth of D | **1b. DFS**  Key idea is backtracking search plus terminate when it finds first end state | **1c. BFS**  Key idea: explore all nodes in order of increasing depth  Space > DFS because @ lowest level have to remember the queue of nodes to explore | **1d. DFS-ID** | | | Definition of DP   1. Recurse 2. If already computed for s, return cached answer   Effect: recasts tree search problem as a DAG. If not DAG then DP breaks! | | | | 1. Expand sates close to the start 2. Use past-cost to re-use computation   Key idea: UCS enumerates sates in order of increasing past cost.  Implementation diff: UCS start to end, Djikstras all nodes |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Exhaustive | DFS | BFS | DFS-ID | | Edge Cost | Any | 0 | ≥ 0 | ≥ 0 | | Time | O(bD) | O(bD) | O(bd) | O(bd) | | Space | O(D) which is small | O(D) | O(bD) > DFS | O(d) | | | | | | | |  |  |  | | --- | --- | --- | |  | DP | UCS | | Cycles | No | Yes | | Edge cost | Any | ≥ 0 | | Time | O(n) | O(nlogn) | | Space | ? | ? | | | | | |
| **4. A\*** Key idea is to distort edge costs to favour certain end states  A\* explores in order of past cost(s,a) and future cost (h(s))  Cost’(s,a) = cost(s,a) + h(succ(s,a)) –h(s)  Definition of h(s) is any estimate of future cost (s). If h(s) =0 means I’m on the optimal path. | | | | | |  | | | | |
| Week 4 MDP | | | | | |  | | |  | |
| Relationship to search. Many similarities but one main one and one minue one. Major is the transition probabilities minor is changed from minimising cost to maximising rewards.   1. succ(s,a) becomes T(s,a,s’). Transition distribution for each state, s is . Introduced via the chance nodes. 2. cost(s,a) becomes Reward(s,a,s’). Reward may be positive or negative   Definitions   * Policy: A mapping from each state s \in States to an action a \in Actions * Utility: Following a policy yields a random path. The utility of a policy is the discounted sum of the rewards on the path. This is a random variable. * Path: s0 → a1,r1,s1 → a2,r2,s2 … (action, reward, new state) * Utility: r1 + r2 + r3 + … * Value: The value of a policy at a state is the expected utility. Also known as expected utility | | | | | | Definition   1. Sstart Start state 2. Action(s): possible actions from states 3. T(s,a,s’): transition prob of s’ by taking action a from a 4. R(s,a,s’): reward for taking T(s,a,s’) 5. Succ(s,a): successor state from state s given action a 6. IsEnd(s): reached end state? 7. 0 ≤ ≤ 1: discount factor (default 1) | | | | |
| What is a solution for MDP?   * Policy * Utility | | | | | |  | | |  | |
|  | | | | | |  | | |  | |
| Week 6 Adversarial Games | | | | | |  | | |  | |
|  | | | | | |  | | |  | |
|  | | | | | |  | | |  | |
| Alpha beta pruning is a technique to minimise search of minimax   * When beta <= alpha then we break * The recursion is DFS. It evaluates the first leaf as a baseline. | | | | | |  | | |  | |
| Week 7 CSPs | | | | | | CSP 2: probabilistic CSPs, hidden markov model | | | | |
| Factor graphs   * Variables: * Values: the set of all values for each variable * Domain: Set of remaining values for each variable   + Value is an element in the domain   + Domain mutates whereas values don’t. It can reduce with each look ahead. * Factor vs. constraint: Factor is a weight. Constraint is a Boolean. * Constraints/factor: scope * Constraints/factor: expression * Number of consistent values = num of values that satisfies factors   + Most consistent var = var with most num of vals in domain   + Lease consistent variable = var with fewest num of vals in domain | | | | | |  | | | | |
| Backtracking search is exponential time with number of variables n.  What does that union notation mean? x is the partial assignment of all the variables. | | | | | | 1. Beam search (most important algos for language. Used by google translate to hold K num of best translation solusions). Extends partial assignemnts    1. Exhaustive, K = inf is BFS so tie O(bn)    2. **Greedy** **K=1**: not global optimum but fast. DFS only one path for one partial assignment. K=1. Time = O(nb)    3. Beam search. Considers K candidates of partial assignments rather than one partial assignment. Beam size K controls tradeoff between efficiency and accuracy.    4. n variables (depth), branching factor: b = domain len, beam size = K, Time = O(nKblogK) 2. Local search. Modify complete assignments. 3. Gibbs sampling (probabilistic) 4. Conditional independence | | | | |
| Which variable to assign next? **Most constrained variable.**   1. Given a list of the remaining unassigned variable, which one to assign next? 2. Choose the variable that has the *fewest* consistent values, in other words, which variable has the fewest remaining options left in their domain? 3. Must assign **every** variable = fail early = more pruning. When to use? **When SOME factors are constraints.** | | | | | | Hints from section   * Valid assignment = multiplication of all factors is non-zero * find probability of CSP= multiplication of all factors / Z, where z is the sum of all factors for all values for all variables. * State the arity of each factor: the number of variables as parameters into the factor function. * Factors are always functions of the variables! F(X1, X2). * Indicator function needs a big 1 notation 1[function] * Domains can add a **null** to domain reflect no choice. Remember that scheduling appended [0] to list? * Which CSP technique will guarantee the maximum weight solution? Least constrained value, most constrained variable, iterated conditional modes, backtracking search. Answer: Backtracking only and MCV. | | | | |
| Dynamic Ordering. **Least constrained variable**   1. Order values of selected variable by decreasing number of consistent values of neighbouring variables. Need to choose **some** value most likely to lead to a solution. 2. **Use when ALL factors are constraints (CSPs). If some are preferences.** | | | | | |  | | |  | |
| Look ahead   * Forward checking (enforces arc consistency on neighbours). Need to actually prune domains to make heuristics useful. If constraints are long distance from each other or rely on many many variables no good! Look ahead: assign to current variable, then look @ its neighbours. Figure out what values would violate the factors and remove them from the domain of the set for that particular neighbour. Reason why it’s useful is because it allows you to exit early out of incompatible scenarios. If domain empty for a variable then terminate early * AC3 takes forward look ahead to the extreme by enforcing arc consistency on neighbours and their neighbours etc. Can solve large problems. Limitation: only looks at pairwise constraints (local structure) not global structures | | | | | |  | | |  | |
| Modelling: binary constraints easiest to reason about, lots of tools. Every n-ary CSP can be reduced to binary CSPs.   * Modelling NxN chess board HW6. Each queen assigned column. | | | | | |  | | |  | |
| Inference: compute marginal probabilities (P(X=x). Gibbs sampling sample one variable at a time a count visitations. More generally Markov chain Monte Carlo (MCMC) powerful toolkit of randomised procedures.  Search vs. sampling    Independence. Let A and B be a partitioning of variables X. A and B are independent if there are no edges between A and B.  Conditioning: To condition on variable X =v, consider all factors f1, …, fk that depend on Xi. Remove Xi and f1,…,fk. Add g(x) = fj(xU{Xi=v}) for j=1,…,k  Conditional independence: Let A, B, C be a partitioning of variables. A and B are conditionally independent given C if conditioning on C produces a graph in which A and B are independent. In other words, every path from A to B goes through C.  Markov blanket of a node = all its parents and children, and all the parents of its children. What to condition on to make A conditionally independent of the rest. Why? Its purpose is to find minimal set without losing useful info for feature extraction. | | | | | |  | | |  | |
| **Bayesian Networks. Why? Handle heterogeneously missing info, incorporate prior knowledge, interpret all intermediate variables. Precursor to causal models** | | | | | | | | | | |
| ELI5: BN = world model. Inference is asking a query using that world model. How to create BN? Requires domain knowledge.  BNs allow us to define a joint probability over many variables e.g P(C,A,H,I) by specifying local conditional probabilities e.g p(i|a).  BNs uses the language of factor graphs.  Markow Networks are a set of preferences. BNs are a generative story. It evolves states.  Markov Networks have arbitrary factors. BN have local conditional probabilities.  Markov models are limiting because they don’t have a way of talking about noisy sensor readings. Hence Hidden Markov Models are its extension.  Unlike HMM, BNs are directed and acyclic.  Unlike NN, BN’s vertices and edges have meaning. The structure describes conditional dependence. In NNs structures does not tell you anything. BN > NN for explainability – you can interrogate the conditional probabilities.  Unlike NN, BNs goes from outputs to inputs. NNs goes from inputs to outputs | | | | | Property 1: Consistency of sub-bayesian networks    Property 2: local conditional distributions | | | | | |
| Fundamentals: it’s important to think about an assignment to random variables as capturing the state of the world | | | | | | | | | | |
| Joint distribution: probabilty for each assignment to state of the world. Upper case to denote the random variable. Lower case to denote value P(S=s, R=r), which is a number whereas P(S,R) is a distribution (table of probabilities).  Marginal distribution: Pick subset of variables we care about (query variables) and induce a distribution over that. E.g. P(S,R,T,A) but query is P(R=1 | T=1, A=1) means S is marginalised out.  Conditional distribution: **YOU MUST NORMALISE**  Local conditional probability:  Conditional independence⫫ If checking A B independent, does knowing B tell me something about A? vice versa because it’s birectional. It’s sufficient to ask one way.  If checking A⫫B | C: ‘Given C, does the extra knowledge of knowing B tell me something new about A and vice versa? Be careful of v-structures, and inverted v-structures.  BN definition: Let X = (X\_1, … , X\_N), BN is a DAG that specifies joint distribution over X as a product of local conditional distributions, one for each node.  Probabilistic Inference definition: P(R=query| conditions )  Bayes rule P(A|B) = P(B|A)P(A) / P(B). In other words, the probability of my query is the same as the probability of my query given the evidence for all valuesof the  Inference by converting BNs to markov networks: ‘it’s important to remember that a single factor connects all parents. There isn’t one per arrow - this is a common mistake. We run any inference algo for markov networks (e.g. gibbs sampling) on this so called markov network. But one important thing is missing, which is the ability to condition on the evidence.  HMM: HMM goal is to infer what the hidden object/state is given observable evidence. | | | | | | | | End-to-End HMM | | |
| **Probabilitistic inference:** three techniques to ask two types of queries: filtering (conditioned on evidence up to that point and can’t see the future e.g. object tracking) and smoothing (conditioned on all evidence, when you have collected all the data).  What is the belief distribution of position given our observations? | | | | | | | | | | |
| Forward-backward for HMMs (**exact**). Use DP to exhaustively enumerate all assignments. First convert HMM to lattice representation. Smoothing question = the nomalised version of Si = the weighted fraction of paths pathing through a given node.    Particle filtering query  for HMMs (**approx)**. Exhaustive search is expensive.Akin to beam search but random sampling. **Very lightweight, practical algorithm because it only depends on the last timestep.** Partial assignement because each particle = guess of current state.  1) Propose (transitions): given currParitialAssignments get transition probability because it’s fast  2) Weight (emisions) old particles based on evidence: w(hi) = ∏P(ej|hi) for all emissions 1,…j at hi. Weights are assigned to each particle as a tuple pair.  3) Resample with replacement kills off particles with low weights, then particles behave as if they were sampled from the distribution we care about p(H=h| latest evidence). Reallocate the *same number of particles* at the higher weighted locations.      Gibbs sampling (GS)(**general**, **approx**): GS isn’t practical for HMMs because it  GS works best when variables are independent from each other. GS can only update variables by a small amount so takes a long time for convergence. HMMs are the case where variables are highly dependent on each other.  For GS to work on BNs, need to initialise the first hiddden variable in the BN either as a fixed value or specify a prior distribution P(B1) from which to sample.  PF: Linear wrt to particle counts. Set that to be at least as big as the domain size. Having more particles is always better.  GS: slow to converge | | | | | | |  | | | |
| Learning Where do the local conditional probabilities come from? These local conditional distributions are the parameters of the BN.  Three learning techniques. Surprisingly for BNs, learning in fully-supervised setting is easier than inference. Parameter sharing. | | | | | | | | | | |
| **Supervised:** where training data is complete assignments. Just need naively *count* and *normalise* = Maximum likelihood technique.  **Laplace Smoothing:** mechanism to guard against overfitting. Involves small tweak to existing algorithms Just add lambda to each possible value whether observed or not.  **Unsupervised with EM:** Unsupervised or missing data setting where training consists of partial assignments. The workhorse is the EM algorithm which breaks the problem into probabilistic inference + supervised algorithm.   * Estep: easiest way is to write joint distribution table * Mstep: add the weighted number of times that param was used in the data and normalize to get probabilities. | | | | In E-step, each q(g) needs to multiply by the number of times the states were observed in the evidence set. In M-step counts need to get normalised by the count of all observations of g. | | | | | | |
| Propositional logic | | | |  | | | | |  | |
| Reinterpret implications as CNF  **Conjunctive Normal Form**. Is a re-expression with conjunctions (ands). (v …v) ∧ (v… v)… we want the conjunctions.Modus Ponens isn’t permitted. Rewrite P→Q into ¬P v Q  **De morgans’ law:** ¬(¬want ∧ ¬ invite). If you have negation outside an expression with you can push negation into it (want v invite) and flip the inside operator  Inference  **Resolution rule. Get rid of p and** ¬P and join the two statements. It is a single inference rule that operates on CNFs.  **Prove f is entailed KB via resolution**  **Equivalent to proving KB U** ¬f. Add ¬f to the KB then use resolution to prove contradiction. | | | | * KB entails f means that f didn’t add any more info. It is already known from KB. f is a superset of KB. No satisfying assignment. | | | | | Resolution tree | |
| First order logic | | | |  | | | | |  | |
| **Has predicates, variables and quantifiers.**   * **If** ∀ **then comes with implication** * **If** ∃ **then comes with conjunction or disjunction** * **If you see a verb then it is a predicate**   **De morgans’ law flips ∀ into** ∃ **and vice versa**  **Propositionisation: converts first order logic into propositional logic by converting infinite predictes into instances, otherwise intractable.**  **CNFs permitted by ignoring the** ∀ **and** ∃  To check if new statement changes size of KB need to check if it is equivalent to an existing statement in KB. If so then adds nothing. If distinct then adds something.  Resolution doesn’t work for first order logic | | | |  | | | | | | |