Deliberative agents :

- Depth-first search : Expand to every successor until the final node. If it's not the goal node, backtrack to the last node and take another successor. Less memory is required compared to Breadth-first search but penalized by bad initial choice (heavy-tailed distribution). No certainty of finding the optimal plan. Can be improved by branch-and-bound by removing branches where the nodes have an higher cost than the top goal node found so far.

- Breadth-first search : Expand layers by layers. Find the optimal plan but need large memory to store all the possible nodes.

- Depth-limited search : Use Depth-first search but with a limited depth. The solution will be suboptimal.

- Iterative-deepening Depth-first search : Use Depth-first search with depth = 1. If the goal node is not discovered then use depth = 2, until the goal node is found. We can find than the complexity is only double compared to the Breadth-first search with the same depth.

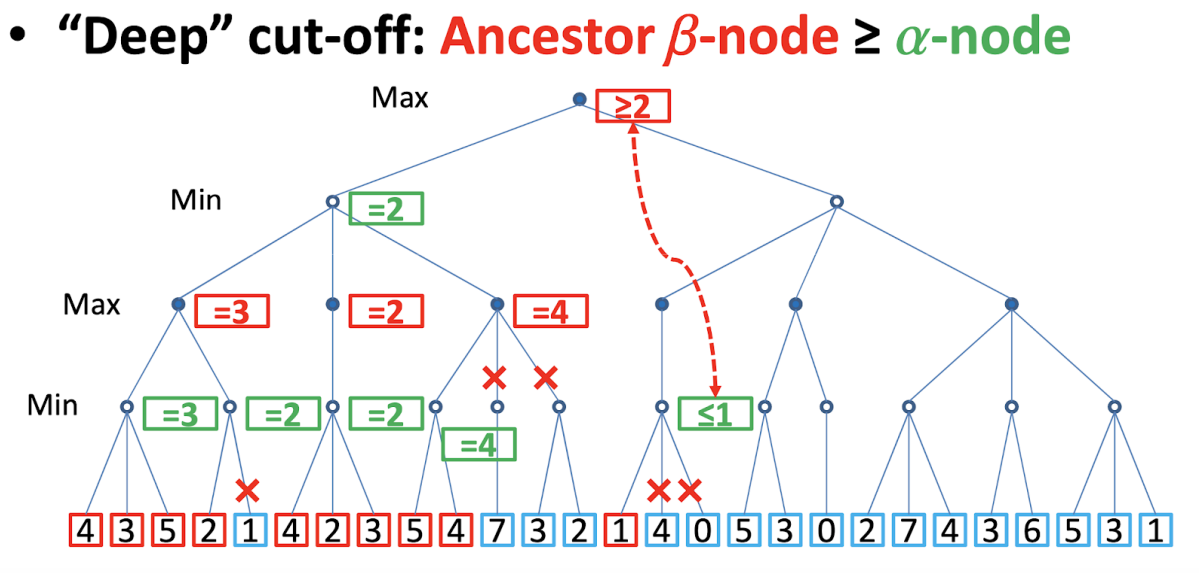
- A-star algorithm : Use an heuristic function h(n) to assess the cost until the goal node. In a map itinerary, we can use the direct distance to find the heuristic function. g(n) is the cost from the initial node to the actual node. Take the node with the lowest sum f(n) and compute its successor nodes. The heuristic function need not to overestimate the cost so that we can branch-and-bound and find the optimal plan.

- Beam search : A-star algorithm, sort and take only the n best nodes (n=beam width). However, we can't prove the optimality of the plan.

- Minimax search : Tree search for a 2-player game. Assume that you want to maximize your gain and your adversary want to minimize your gain (individuality rationality). Play the game until the end and evaluate the final node. Can perform backtracking until it's possible : evaluate all the other nodes to completely evaluate the ancestor node.

- Alpha-Beta pruning : Remove branches which are not possible either because the computer will not allow us or we it is not profitable for us. Reduce extremely the

complexity of the tree.



- Monte Carlo search : Use roll-out over the same node multiple times and performs statistics to evaluate this node. Try to remove useless nodes to avoid wasteful roll-outs. UCB can be a good way to select the nodes.

$$UCB_i = \underbrace{\frac{W_i}{N_i}}_{=Q(i)} + \underbrace{c\sqrt{\frac{log(N)}{N_i}}}_{\text{exceeded with prob.} < exp(-c^2 t/2)}$$

W_i = win with move i; N_i = roll outs with move i;
N = total roll outs; c = exploration parameter.

AlphaGo combines Monte-Carlo search with deep neural networks. The neural networks will evaluate the board position of the chess game.

- Counterfactual regret minimization : Use Monte Carlo search to evaluate the counterfactual regret (Regret between the outcome of two different nodes). Play strategies with probability proportional to their Counterfactual regret. Used by top poker algorithm. Try to converge to the Nash equilibrium for zero-sum games.

Planning with factored representations :

- Situation calculus : Use of situations to determine a plan. A situation is defined by an history of actions and a set of true/false predicates

  $On(A,B,\ s)$
  $On(B,C,\ s)$
  $On(C,Table,\ s)$ are situation-dependant predicates, also called fluents (they can change over time). The effect axioms represent the change in fluents after a specific action whereas the frame axioms are the predicates that are preserved after a specific action.

- STRIPS planner : Restricted representation language which decrease the complexity compared to situation calculus. Action can be made with preconditions (predicate true at present situation) and have effects (predicate true at situation after doing the action) with two lists add : new predicate and delete : previous predicates no more true for this situation

- Binarized Neural Networks : Learn compact models over discrete data. Allow to learn and train over factored state representation.

- Bayesian Networks : Represent each event by a node and causation by an arc $(x_3 = c) \rightarrow (x_7 = d)$. The arc is represented by a probability distribution. It is a directed acyclic graph (DAGs).

- Semantic networks : Node is represented as a concept in the world and creation of links (relations) between objects. It is a knowledge based network.

- Dynamic Bayesian Networks : Bayesian Networks with representation over different time steps. The causation need to be over different time step (can't create arc for the same time step). Calculate the transition probability between state as

$$pr(x_1' = v_1, .., x_n' = v_n | x_1 = w_1, .., x_n = w_n)$$
$$= \prod_{l=1}^{n} p_l(x_l' = v_l | x_j = w_j, .., x_k = w_k)$$

- Boolean Satisfiability problem (SAT) : A proposition is satisfiable if there is a set of variables such that the proposition is true. $(p \wedge \neg p)$ is not satisfiable.

- Value function factorization : Mapping the states vectors into return which correspond to a function. The function is usually approximated via basis functions like radial basis function (RBF) to reduce complexity. The weights are chosen so that the overall mean square error is minimized. An important aspect is to choose the correct basis function because the approximation depend on the quality of the basis functions.
  $\sum w_i b_i(S) \quad = \quad R(S, \pi(S)) + \gamma \sum_{S' \in \mathcal{S}} T(S, \pi(S), S') \sum w_i b_i(S')$ usually not solvable because many more states than basis functions.

- Least commitment principle : Some actions can be made in any order possible without having an effect on the result. A total order planning will create a combinatorial explosion of plans. The partial order (nonlinear) planning will give an order of actions only when it is necessary. Try to avoid conflicts of causal links by defining a set of actions that can be done in parallel. Try to create finite number of reachable propositions.

- Graphplan : Represented as a graph, the actions that can be carried out simultaneously. Not seeing as variables but as propositions which make it bigger but simpler (no worry about unification). Use iterative deepening algorithm. Level 0 : preconditions, level 1 : actions and level 2 : effects. The mutex consist of mutually exclusive actions, an arc is made between these actions to know that they can't be done in parallel. To find a solution, all the proposition at the final depth should not be mutex. After some number of steps, all levels of the graph will become identical. If no solution was found, the problem is said to be unsolvable. The number of nodes grow only polynomially. Graphplan will not give all solutions as it will not give the order.

- Fast Forward algorithm : A-star combined with graphplan.