Agent

An agent is defined by:

**1. Action Space (A) :** The set of all action the agent can do

**2. Percept Space (P):** The set of all things that the agent can perceive in the world **3. State space (S) :**The set of all states that the agent can be in.

**4. Word dynamics (T: SXA**🡪**S):** the function mapping a state and a action to a next state **5. Percept Function( Z: S**🡪**P):** The function defining all the objects the agent can perceive at S. **6. Utility Function (S**🡪 **R):** Function defining the utility (desirability) of a certain state (mapping to a real number).

An agent is trying to optimize the actions it will take to maximise the utility function (i.e reach the more desirable states.)

An agent will be designed differently depending on its environment.

An Environment is defined by

**7. Discrete/Continuous:** Are the state/Action/percept space discrete (i.e is there a finite number of states ?)

**8. Deterministic / Stochastic(Non deterministic):** Does the agent always know exactly what the next state will be after performing an action.

Is the world dynamics a one-to-one relation.

9. **Fully observable vs Partially observable:** does the agent know the exact state of the world/itself or only has partial information.

10. **Static/Dynamic:** Can the world change while the agent is thinking?

Searches

To formulate a problem as a search problem, we define the Agent (A,P,S,T,Z,S) + a initial and goal state.

We start by generating a **State graph** defined by:

11. Vertex V: Represent states

12. Edges E: Represents world dynamics, each edge is labelled by its cost.

**The solution is a path:** within the graph from initial to goal vertices.

**The cost** associated with the solution is the sum of the cost of all edges in the path. **The optimal solution** is the path with the minimum cost.

We measure the performance of a search with:

**13. Completeness:** The algorithm will find a solution whenever one exists.

**14. Optimality:** The algorithm will return the minimum cost path whenever there is one. **15. Complexity:** Amount of time and memory needed by the algorithm to solve the problem.

We can represent the visited nodes in a graph with a **Search Tree**, with root = initial state and the tree expanding everytime we visit a new node.

Deterministic search in discrete space

Environment is Discrete, Fully observable, Deterministic and static.

Blind searches

Blind searches do not estimate the cost from the current vertex to the goal vertex

*Breadth first search(BFS)*

Explore all nodes off the current level of search tree before exploring deeper nodes. Uses FIFO queue to keep fringe nodes

**Cost:** #steps

**Algorithm:**

1. Set the initial vertex I as root of the search tree.

2. Push I to the queue.

3. Loop

a. t = queue.dequeue(), mark t as expanded

b. If t is the goal vertex

return.

c. For each v in successor(t)

i. If v is not in the tree yeT

1. Queue.push(v)

2. Put v as a child of t in the search tree

BFS is complete if b is **finite**

It will generate the **optimal** solution.

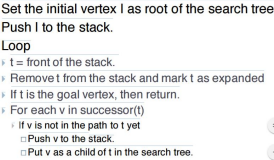
**Complexity:** Time&Space = O(b^d) (where b = branching factor d – shallowest goal node.)

*Depth first search(DFS)*

Explores ones full SubGraph before exploring the other.

Uses a stack (LIFO) to keep track of the fringe nodes.

**Cost:** #steps

**Algorithm:**

DFS is **complete** if b&m are finite (b = branching factor, m= max depth) DFS is **NOT optimal**

DFS has complexity: time: O(b^m) ;space: O(bm).

*Iterative deepening DFS*

Performing multiple DFS with a specified Depth limit that increments at every iteration until goal is found.

Iterative DFS is **complete** if b is finite

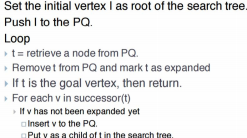
It is **optimal** in terms of #steps.

**Complexity:** Time O(b^d), Space: O(bd).

*Uniform cost search*

Expand fringe node with lowest cost from root, uses a Priority Queue (PQ) to keep fringe nodes. g(n) = Cost from root to node n, expand the n that minimises g(n).

**Algorithm:**

****

Uniform cost search is **Complete** if b is finite and all edges have cost> e ( where e is the min cost of a ste)

It is **Optimal** if all edges have a positive cost

**Complexity :** Time&Space : O(b^(1+floor(C/e)) where C is the cost of the optimal solution.

Informed searches

Informed search algorithms try and estimate the cost from the current node to the goal. **g(n)** = Cost from root to node n

**h(n)** = Estimated cost from n to goal (based on heuristics)

in an informed search **the next node selection is based on a function f(n)** which includes h(n).

*Greedy Best first search*

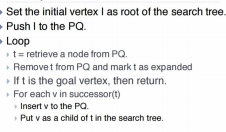
Expand fringe nodes with lowest estimated cost form current node to goal node. **f(n) = h(n).**

**g(n) is ignored.**

Expand node with lowest f(n).

Uses PQ to store fringe node, but the PQ is based on f(n) the node with the lowest f(n) value will have higher priority.

**Algorithm:**

****

**Not Complete**

**Not optimal**

**Complexity:** Worst case O(b^m) (b = branching factor, m = maximum depth).

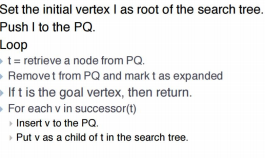
*A\* search*

The priority queue uses f(n) = g(n) +h(n) as priority, the lowest f(n) has the highest priority. g(n) = cost from root to node n

h(n) = estimated cost from node to goal

f(n) = g(n)+h(n).

**Algorithm:**

****

**An heuristic is admissible if it never overestimates the cost from n to goal. Complete** if all edges have cost >e

**Optimal** if Complete and heuristic is admissible.

**Complexity depends on heuristics.**

Deterministic search in continuous space/Motion Planning

Motion planning

Study of computational methods enabling agents to choose its own motions to get to a goal state from an initial state.

We need to discretise our infinite state graph.

2D motion planning

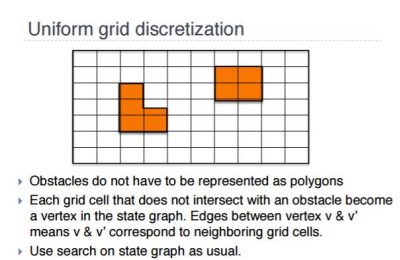
*Visibility graph*

**undirected graph** : nodes are vertices of obstacles, an edge between 2 vertices represents an edge for the obstacle or a collision free straight line between 2 vertices.

Given an initial and goal state: find the vertex Vi nearest to to init where the straight line segment between qi and init is collision free, same for the goal.

**Complexity:** Construction time = O(n^3); space = O(n^2)

*Uniform grid discretization*

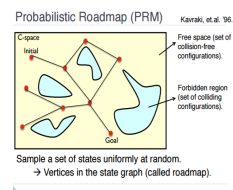
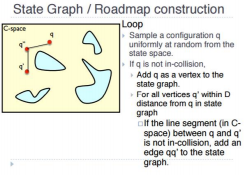
**

Higher dimensions.

Visibility graph and uniform grid still hold

To build the state graph we can use

*Probabilistic Road Map (PRM)*

**Once the graph is constructed usual search can be applied to find a path.

**Interleaving:** Doing graph construction and searching simultaneously, every n vertices added, search the graph.

**Sampling strategies:** Random, near obstacle, in between 2 obstacles….

**Multi-arm bandit problem:** Technique for trading off exploration vs exploitation, the strategy that would be the most useful in solving the problem is chosen with higher probability.

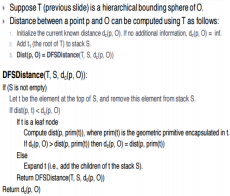
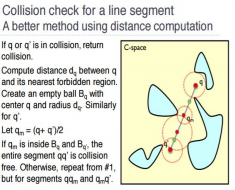
*Collision checks.*

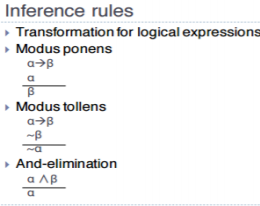
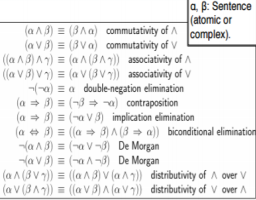
**Line collision check:** having 2 linep1q1, p2q2; if p1q1p2 & p1q1q2 have different orientations AND p2q2p1 &p2q2q1 have different orientation

**Hierarchical bounding volume (HBV):** Tree of bounding volume for each object to check, start by building the bounding volumes of the primitive lines and build up from there.

Then when checking for collision use the tree to quickly check for collision before going into **Line collision checks** (if one is found at a leaf node).

**Distance calculation with HBV:**

****Logic, representation, validity, satisfiability



Propositional logic

**Atomic sentences** that can be either **true or false.**

**Logic operators:** Not, AND, OR, implication, biconditional implication

**Valid sentence:** All possible interpretations of the sentence are true. (Eg P V ~P)

**Satisfiable sentence:** There exists an interpretation of the sentence such that it Is true. (Eg ~P) (everything that is valid is also satisfiable).

**Unsatisfiable sentence:** All possible interpretations of the sentence are false (Eg: P ^ ~P)

Propositional Logic Problem

To formulate propositional logic problems, we create a **Knowledge Base (KB). KB:** a set of sentences such that KB is false in models that contradict what the agents knows.

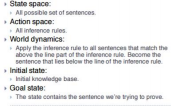
Validity problems

Is the given sentence valid?

*Model checking:*

Create truth table and check that the sentence is true given that all premises in KB are true.

*Theorem proving.*

Using Inference rules to reduce the sentence by using the KB. 

Theorem proving can also be done using searches in **Natural deduction.**

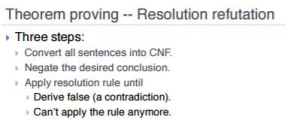
**Conjunctive Normal Form(CNF):** (AVB)^(CVD)^(EVF)… conjunction of

disjunctions.

When in CNF used we can use **Resolution.**

****

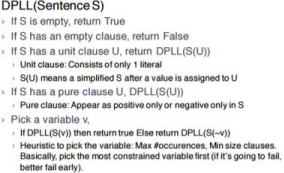
*RESOLUTION REFUTATION*

**

Satisfiability problems

There are several ways of solving Satisfiability problems

*Model checking: Davis Putnam Logeman Loveland*

**

*Model Checking: GSAT*

**AND/OR Tree

Fully observable, Non-deterministic, static, discrete Environment.

An and or tree is a tree interleaving And/Or 

levels.

Each Or level branching is introduced by the

agents choice.

Each And level branching is introduced by the

environments

Labeling

Node/Label Solved Closed Leaf(state) It is a goal state. It is not a goal. AND node (action) All its children are solved. At least one of its children is closed.

OR node (state, non-leaf) At least one of its children is closed.

All its children are closed.

Label from leaf nodes to root nodes if the root node is solved the problem is solved, if the root node is closed the problem is impossible to solve.

Solution

**The subtree that establishes that the root is solved is the solution to the AND/OR tree.** it defines a conditional plan to pick the next action.

AND/OR Tree search.



Min-Max

A min max tree is similar to an AND/OR tree but is designed to model adversarial situation. OR levels are agents moves, called **MAX** 

AND levels are Opponents move, called **MIN.**

Evaluation function

e: S🡪R

e>0 favourable to max, e<0 favourable to min, e=0 neutral.

MINIMAX ALGORTIHM



Alpha-Beta pruning



Decision/Utility theory

Decision theory

A framework for an agent to choose the best decision in a non-deterministic manner. (Calculus for decision making)



**Maximum Expected Utility(MEU):** assigns a utility value to each outcome representing preference “Best decision” maximises the MEU (utility theory)

Axioms of Utility Theory

16. **Orderability:**

****

17. **Transitivity:**

****

18. **Continuity:**

****

19. **Substitutability:**

****

20. **Monotonicity:**

****

21. **Decomposability:**

****

22. **Main theorem ( if the axioms are respected).**

****

**RISK in Utility theory**

1. Risk Neutral : the utility is the Expected utility

2. Risk Averse: we prefer a smaller reward for sure then a larger reward with low chance. The utility function will need to be redefined

3. Risk seeker, we prefer a uncertain outcome to a certain (not necessarily advantageous) outcome, same here

Decision Tree

The decision tree is represented as an AND/OR tree where:

∙ OR levels are Agents decisions

∙ AND levels are Lottery corresponding to the outcome of the decision made at the parent node.

Markov Decision Process (MDP)

Framework to find the best course of actions to perform when the outcome of each action is non deterministic.

Representation

A Markov decision process is framed as follows:

∙ **State space : S**

∙ **Action Space: A**

∙ **Transition function: T(s,a,s’)** = ��(����+1 = ��′|���� = ��, ���� = ��)

∙ **Reward function: R(s)** || R(s,a) || R(s,a,s’)

Can be represented by an AND/OR tree when the root is the initial state.

Solution

OFFLINE SOLUTIONS

Finding a solution ↔ finding a working strategy (called **policy**)

**Optimal policy** denoted as π’ is mapping from states to actions, for each state S π’ will tell us what is the optimal action to perform.

After performing each action the agent will receive a reward depending on what state he ends up in.

Therefore, the “best action” is trying to maximize the expected reward, called the value function. **Value function:**

****

VALUE ITERATION

POLICY ITERATION

ONLINE SOLUTIONS

Real Time Dynamic Programming (RTDP)



Labelled RTDP (LRTDP)



Monte Carlo Tree Search (MCTS)

Sampling based method to approximate the value of complicated functions, in MDP used to approximate the expected total reward.



SELECTION**:** Multi arm bandit is used to select an action



Where c is a variable indicating how to balance exploration vs exploitation n(s) = number of time the node has been visited

n(s,a) = number of times the out-edge of s with label a has been visited. SIMULATION: often called rollout**, use heuristic** (eg greedy, solution of deterministic case..)

Partially Observable MDP (POMDP) Partially observable, non-deterministic, static, discrete environment.

Representation

∙ **State Space**: S

∙ **Action Space**: A

∙ **Observable Space:** O

∙ **Transition function:** T

∙ **Observation function:** Z

∙ **Reward function:** R

**Belief:** mapping from a state to a probability, can be represented as a parametric distribution.

**Strategy/policy:** Mapping from beliefs to actions.

**Value function: **

**Computing belief:** using Bayes rule



Solutions

**Any solution applied to MDP can be applied to MDP but states become beliefs.** Machine learning- Supervised and

Unsupervised

**Definition:** Algorithms to enabling agents to improve their behaviour with experience. 3 main types of machine learning:

∙ **Supervised Learning:** Learn from examples (sometimes we are not sure if the examples are correct

∙ **Unsupervised learning:** Learn by finding structure in data

∙ **Reinforcement learning:** Learn by doing.



DECISION TREE

**Information gain:** 

∙ Entropy (measure of uncertainty)

∙ Trying to reduce the uncertainty.

**Greedy Decision tree learning:**

****

NEURAL NETWORK

A network of artificial neurons

**Neuron:** a function applied to a linear combination of the input attributes.



**Recurrent:** recurrent neural networks have at least one cycle to represent fe edback connection. Learning a neural network:

∙ Finding a suitable Linear combination

∙ Finding suitable structure, size and activation function.

BAYESIAN NETWORKS

A Graph related to conditional probability tables.

**The Graph:** Set of nodes representing a set of random variables (each with a finite set of values). **The Conditional Probability Table (CPT):** conditional probability of a node given its parent.

**Markov Blanket:** The Markov blanket of a node X is the parent of X the children of X and the Parents of Children of X

**Inference:** Compute the probability of an event knowing one or more have happened, answer queries by summing over the variables not involved in query.

Reinforcement learning

A Reinforcement learning problem is a **MDP where the transition and/or rewards functions are not known.**

**Solving:** Computing the best action to perform even though the transition& reward function might not be defined. In general it boils down to **Exploitation vs Exploration.**

Exploration vs Exploitation

Epsilon Greedy

∙ Assign weight to each strategy (starting with equal weight)

∙ Strategy with highest weight is selected with P = 1-e, the rest a selected with P = e/N ∙ Increment weight of strategy if it is successful.

Exp3



Upper Confidence Bound( UCB)



Approaches for solving

**Model Based:** Data is used to learn the missing components of the MDP, Once we know T&R we solve MDP, indirect learning but most efficient use of data.

**Model Free:** Data is used to learn the value function and policy directly, direct learn but not the most efficient use of data.

**Passive:** The agent is given a data set to learn from, agent does not need to decide what action to perform.

**Active:** The classical reinforcement learning, the agent select what action to perform then learns from the outcome and converges to the correct MDP

Model Based: Simple Frequentist



Model Based, Passive

1. Use a supervised learning method to compute the T&R,

2. Solve the MDP with computed T&R, compute difference between Data and the expected trajectory.

3. If the difference is large: improve the MDP model repeat from 2

Bayesian Reinforcement Learning (model based active)



Model Free

Monte Carlo



Temporal Difference (TD)learning

Iteratively reduce the difference between the value or the Q-value estimates. 

Values are updated after each step (instead of each

run for monte carlo)