Agent is anything that can be viewed as perceiving its **environment** through **sensors** and **acting** upon that environment through **actuators**.

Agent Model:
- **Percepts/Obs** of the env, made by sensors
- **Actions** which may affect the env, made by actuators
- **Env** in which the agent exists
- **Performance measure** of the desirability of env states.

The above characteristics dictate techniques for selecting rational actions

Agent: Automated packing robot
Performance measure: Percentage of items correctly packed
Environment: Conveyor belt with parts, bins
Actuators: Jointed arm and hand
Sensors: Camera, joint angle sensors

Agent is specified by an **agent fn** $f : P \rightarrow A$ that maps a sequence of percept vectors $P = [p_0, ..., p_t]$ to an action a from a set $A = \{a_0, ..., a_k\}$

A **rational agent** chooses whichever action max the expected value of the performance measure given the percept sequence to date and prior environment knowledge.

Limitations:
- Rationality $\neq$ omniscience (Percepts may not provide all the required information)
- Rationality $\neq$ clairvoyant (Actual outcome of actions may not be as expected)
- Thus, we aim for a bounded (based on limitations) rationality based on expected performance, not actual performance

Fully observable (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.

Deterministic (vs. stochastic): The next state of the env is completely determined by the current state and the action executed by the agent. If the env is deterministic except for the actions of other agents, then the environment is strategic

Episodic (vs. sequential): The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.

Static (vs. dynamic): The environment is unchanged while an agent is deliberating. (The environment is semidynamic if the environment itself does not change with the passage of time but the agent's performance score does)

Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.

Single agent (vs. multiagent): An agent operating by itself in an environment.

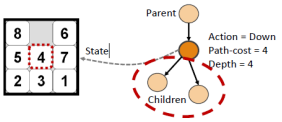
Env Type	Solitaire	Crossword	Online Shopping	Basketball	Chess (w clock/ wo)	Poker	Taxi Driving	Medical Diagnosis	Image Analysis	Robot Part picking
Observable?	F	F	P	P	F	P	P	P	F	F
Deterministic?	Y	Y	Y	N	N	N	N	N	Y	Y
Episodic?	N	N	N	N	N	N	N	Y	Y	Y
Static?	Y	Y	Semi	N	Semi/Y	Y	N	Y	Semi	Semi
Discrete?	Y	Y	Y	N	Y	Y	N	N	Y	Y
Single-agent?	Y	Y	N	N	N	N	N	Y	Y	Y

Standard Search problem

State space: Number tiles in each cell position
Initial state: [8,-,6,5,4,7,2,3,1]
Actions: Move tile {Left, Right, Up, Down}
Transition Model: Update tiles in current & target cell pos
Path cost: Number of moves
Goal test: Compare to positions in goal state

State: rep. of a **physical configuration**
Node: data struct, **part of a search tree**
- Comprising state, parent-node, child-node(s), action, path-cost, depth unlike a state.

Solution: Seq of act from init to goal state



Expand function **creates new nodes** (& their various fields) using the Act & Transition Model to create the corresponding states

Search strategy: picking the **order of node expansion**

b: Max branching factor of the search tree
d: depth of the least-cost solution
m: max depth of the state-space (inf possible)

Tree Search:
- Offline, simulated exploration of state-space by expanding states via generation of successors of already-explored states
- However repeated states results in redundant paths that can cause a tractable problem to become intractable (inf loop, m = inf)

Graph Search:
- Modified tree search to keep track of previously visited states (to not revisit) but a potentially large number of states to track)

Types of Search:
a) Uninformed Search
No additional information about states beyond that in the problem definition (blind search)

b) Informed Search
Uses problem-specific knowledge beyond the definition of the problem itself

c) Adversarial Search
Used in a multi-agent environment where the agent needs to consider the actions of other agents and how they affect its own performance.

BFS (FIFO, Graph): Init: A
Expand & pop A: AB, AC
Expand & pop B: AC, ABD, ABE
Expand & pop C: ABD, ABE, ACF
Expand & pop D: ABE, ACF, ABDG, E visited
Expand & pop E: ACF, ABDG, ABEH, F visited
Expand & pop F: ABDG, ABEH, H visited
Expand & pop G: ABEH, ABDGX, H visited
Expand & pop G: ABDGX, H visited
Expand & pop X: X is goal state, return ABDGX

Uninformed Searches

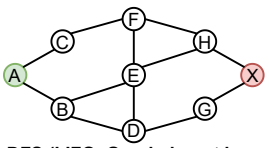
Breadth First Search:
Expand shallowest unexpanded node, implement using (FIFO) Queue
- **Completeness:** Yes (if b is finite)
- **Optimality:** Yes (Not optimal in general)
- **Time complexity:** $O(b^d)$
- **Space:** Same (keeps every node in mem)

Recall BFS checks level by level, if the step cost != 1/ not uniform, then we might miss out certain nodes with a cheaper step cost

Uniform Cost Search: BFS w g(n)=1
Expand unexpanded node with lowest path cost g(n) implement using Queue ordered by g(n). If visited node, only replace if cost is lower
- **Completeness:** Yes (if step cost some positive constant else lead to loop pattern)
- **Optimality:** Yes
- **Time/Space complexity:** $O\left(b^{\frac{C^\star}{\varepsilon}}\right)$

$C^\star$ is the cost of opt solution
Goal test after popping only, since there might be some shorter path to that node. Only do goal test when popping that node.

Depth First Search:
Expand deepest unexpanded node, implement using (LIFO) Queue
- **Completeness:** No (if m inf, keep going down inf path)
- **Optimality:** No (might be searching down non-optimal path)
- **Time complexity:** $O(b^m)$ terrible if m>>d, as need to back track all the way back up
- **Space:** `O(bm)` linear , store m depth, and at each level, keep track of b child nodes



DFS (LIFO, Graph, insert lowest alphabetical order 1st): Init: A
Expand & pop A: AB, AC
Expand & pop C: AB, ACF
Expand & pop F: AB, ACFE, ACFH
Expand & pop H: AB, ACFE, ACFHG, ACFHX
Expand & pop X: X is goal state, ret ACFHX

DLS (with l = 2): Init: A
Expand & pop A: AB, AC
Expand & pop C: AB, ACF
Expand & pop F: AB, depth limit = 2
Expand & pop B: ABD, ABE
Expand & pop E: ABD, depth limit = 2
Expand & pop D: Empty, depth limit = 2
Frontier empty, ret no solution

Iterative-Deepening Search:
Use increasing DLS to find best depth limit
- **Completeness:** Yes
- **Optimality:** Yes
- **Time complexity:** $O(b^d)$
- **Space:** $O(bd)$

Best of BFS (optimality) + DFS (space complex). Not wasteful to keep generating states with each increasing DLS since most of the work (node expansion) happens at lower depths)

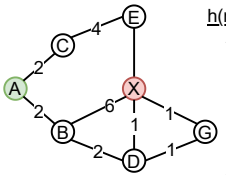
Depth-Limited Search:
DFS with predetermined depth limit l
- **Completeness:** No (l<d)
- **Optimality:** Chance of No (if l>d), might find a solution but is not the least-cost
- **Time complexity:** $O(b^l)$
- **Space:** `O(bl)`

Solves the infinite problem of DFS

Greedy Best-First Search:
Expand node with lowest h(n), implement using Queue ordered by h(n)
- **Completeness:** No (can get stuck in loop unless we keep track of visited states)
- **Optimality:** No
- **Time complexity:** $O(b^m)$
- **Space:** Same (keeps every node in memory)

UCS complete & optimal but may "waste searches" if search in the wrong direction. G-BFS searches generally in the "right" direction but not complete & optimal since a shorter path from the initial state does not imply we are nearer to the goal state.

A-Star Search:
Expand node that has incurred least cost & is nearest to goal state, implement using Queue ordered by eval. fn, $f(n) = g(n) + h(n)$
- **Completeness:** Yes (no need to keep track of explored steps due to path cost g(n))
- **Optimality:** Yes (if adm/consistent h(n))
- **Time/Space complexity:** UCS



h(n) to X
A: 7
B: 4
C: 5
D: 3
E: 4
F: 2
X: 0

A* becomes UCS if heuristic cost of every node is 0. So, UCS is a special case of A* algorithm

Informed Searches

$g(n)$ ignores direction of the search, instead we use $h(n)$ heuristic fn: est dist from node n to goal state (how close a state n is to the goal state)

$h(n)$ **admissible** if $h(n) \leq h^\star(n)$
$h^\star(n)$ = true cost from node n to goal state (e.g. straight line distance $\leq$ the actual road distance)

$h(n)$ **consistent** if
$h(n) \leq h(n\prime) + c(n, a, n\prime)$
$c(n, a, n\prime)$ = cost of getting from node n to n'

$h_1(n)$ **dominates** $h_2(n)$ if
$h_1(n) \geq h_2(n)$

Provided that both $h(n)$ are admissible
More dominant $h(n)$ potentially explores lesser branches, hence better

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem

e.g. Relaxed: Tiles can be directly moved to any square. $h_1(n)$: # of misplaced tiles
OR Relaxed: Tiles can be moved to any adjacent square. $h_2(n)$: total Manhattan distance

Admissible heuristics can be derived from the solution cost of a sub-problem of the problem (e.g. Solve 4 tiles first)

Greedy-BFS (Graph): Init: A(7)
Expand & pop A: AB(4), AC(5)
Expand & pop B: ABX(0), ABD(3), AC(5)
Expand & pop X: X is goal state, ret ABX

A⋆: Init: A(0+7=7)
Expand & pop A: AB(2+4=6), AC(2+5=7)
Expand & pop B: ABD(2+2+3=7), AC(7), ABX(2+6+0=8)
Expand & pop D: ABDX(2+2+1+0=5), ABDF(2+2+1+2=7), AC(7), ABX(8)
Expand & pop X: X is goal state, ret ABDX

Constraint Satisfaction Problem:
State: Variables, X_i with values from doman, D_i
Goal Test: Set of constraint, C_i specifying allowable combinations of values for subsets of var

(Nodes) Finite set of variables, $X = \{X_1, ..., X_n\}$
Non-empty domain D of k possible values for each variable, $D_i = \{v_1, ..., v_k\}$
(Edges) Finite set of constraints, $C = \{C_1, ..., C_m\}$ where they limit the values variables can take

Complete assignment: every var assigned a val
Consistent assignment meets all constraint
CSP solution = complete & consistent for all var

Formal representation language that can be used to formalize many problems types

Able to use ***general-purpose solver***, which are generally more efficient than standard search.
- Constraints allow us to focus the search to valid branches, invalid branches removed
- Non-trivial to do this for standard search (need manual selection of actions)

Varieties of CSP:
o Discrete variables:
- **Finite domains**: $O(d^n)$ complete assignments for n variables, domain size d (e.g. map-colouring, scheduling with time limits)
- **Infinite domains**: Int, str, etc. (e.g. job scheduling (e.g. StartJob1 + 5 < StartJob3)

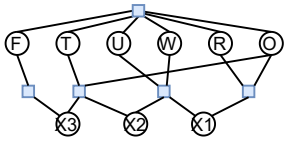
o Conti. variables: (e.g. start/end times of an obs)

o Unary/Binary/Higher-order Constraints:
# of variables involved in constraint

o Preference (soft) Constraints:
(e.g. red is better than green represented by some cost of each var, aka Constr. Opti Problem)

E.g. of CSP: Variables: F, T, U, ..., X1, X2, X3
Domains: {0, 1, 2, ..., 9}
Constraints: all_diff(F, T, ..., O), O + O = R + 10* (X1), ..., X3 = F

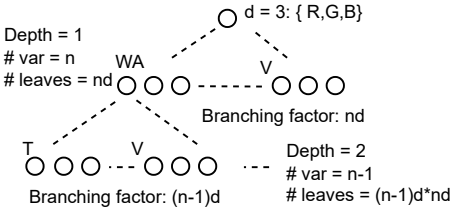
X3 X2 X1
T W O
+ T W O
F O U R



Variables: WA, NT, Q, NSW, V, SA, T
Domains: $D_i = \{R, G, B\}$
Constraints: Adj regions diff colours, ie. WA!=NT
CSP Solution: {WA=R, NT =G, ..., T=G}

CSP as Std Search:
Initial State: {}
Successor fn (actions): Assign val to unassign var
Path cost: Constant cost for every step
Goal test: Current assignment complete & consist.

Sequence of actions/path not impt, only final goal state. Also solution will appear at depth n for n var, potentially $n!d^n$ leaves in the search tree. Can reduced using backtracking search



Backtracking Search:
(DFS with single-var assignment)
CSP var assignments are **commutative** (order not impt)

Hence only consider assignment to 1 var at each level/depth, branch factor at all levels d, d^n leaves

CSP: $O(d^n)$
If subproblem have c var out of n total, worst case is $O\left(\frac{n}{c} \cdot d^c\right)$

Variable Assignment Order:
1) Minimum remaining values
- Choose var with fewest legal values left (most constraint var, most likely to fail)

If same MRV, **Degree Heuristic**:
- Choose var with the most constraint on remaining var

2) Least constraining value:
- Choose one that rules out the fewest values in the remaining var

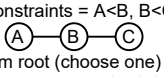
Early Detection:
1) Fwd Checking Algo
- Keep track of remaining lv for unassigned var, terminate search when any var has no lv
- Propagates info from assigned to unassign var only, only provide early detection for some failures. Need to repeatedly enforce constraints locally using constraint propagation.

2) AC-3 Algo $O(n^2 d^3)$
- Start with queue of all directed arcs
- **RemoveInconsistent(X,Y) :**
Check for consistency (X-> Y is consistent iff for every val of X there is some allowed y)
- If removed any, add the neighbours (into X) into queue
- Continue till queue is empty

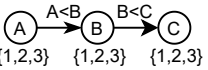
$O(n^2)$: Need to check all (potentially) n^2 edges (directional)
$O(d^2)$: For every edge, compare both domains
$O(d)$: Each var change, repropagate to its neighbour at most max d times

Tree Structured CSP:
If constraint graph has no loop: $O(nd^2)$ time since tree with n nodes has at most n edges, compare 2 nodes which each side at most d values

Domain = {1,2,3}
Constraints = A<B, B<C



Order var from root (choose one) to leaves st every node's parent precedes it in the ordering



From j = n to 2, apply $RemoveInconsistent(parent(X_j, X_j))$

RC(B,**C**): B{1,2,~~3~~}, C{1,2,~~3~~}
RC(A,**B**): B{1,2}, A{1,~~2~~,~~3~~}

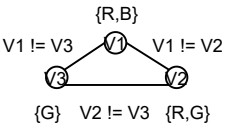
From j = 1 ton, assign X_j consistently with $Parent(X_j)$

A{**1**}
B{1,**2**}, only 2 is consistent
C{1,2,**3**}, only 3 is consistent

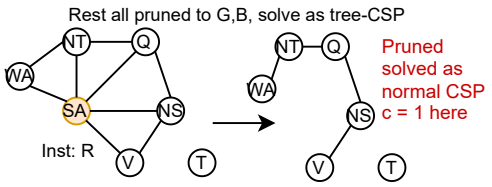
Nearly Tree Structured CSP:
Conditioning: Instantiate a var, prune its neighbours' domain

Cutset conditioning: instantiate (in all ways) a set of variables st the remaining constraint graph is a tree

Cutset size c implies $O(d^c \cdot (n-c)d^2)$, very fast for small c



Pure Backtracking:
V1: R
V2: R (Inconsist. assign, backtrack)
V2: G
V3: G (Inconsist. assign., backtrack)
V1: B (V2 lv none left)
V2: R
V3: G, all var assigned, return
V1: B, V2: R, V3: G



Backtracking with Fwd Checking: V1: R
V2: G (R inconsist. removed)
V3: Empty (G inconsist. removed)
V1: B
V2: R (Choose R)
V3: G all var assigned, return
V1: B, V2: R, V3: G

AC3: V1 -> V2: D1 = RB, D2 = RG
V2 -> V1: D2 = RG, D1 = RB
V2 -> V3: **D2** = R~~G~~, D3 = G
V3 -> V2: D3 = G, D2 = R
V1 -> V3: D1 = RB, D3 = G
V3 -> V1: D3 = G, D1 = RB
(+) **V1** -> **V2**: **D1** = ~~R~~B, D2 = R
(+) V3 -> **V1**: D3 = G, D1 = B
(+) V2 -> **V1**: D2 = R, D1 = B
Queue empty, all arc consistent, ret ...

Game Types	Deterministic	Chance
Perfect Info (Fully Obs)	Chess, Checker, Go	Backgammon
Imperfect Info (Partially Obs)	Battleship	Bridge, Poker, Scrabble

Game as Standard Search:
Def strategic two-player game by:
- Initial State and Actions
- Terminal Test (Win / Lose / Draw)
- Utility Function (num reward for the outcome e.g. Tictac Toe: +1, 0, -1)

0-sum game with 2 players: each player's utility for a state are equal and opposite (+1/-1, sum = 0)

ply: action by a player (2-ply, means 1 action followed by 1 counter-action: 1 move & game terminates)

Minimax-Algo:
(Deterministic + Fully obs game), Idea: choose move to position with highest minimax val

Completeness: Yes if tree is finite (exist terminal states)
Optimality: Yes against a rational (opti) opponent
Time Complexity: $O(b^m)$
Space Complexity: $O(bm)$ similar to DFS

Yet, for chess e.g. branching factor (possible moves), $b \approx 35$ & depth (number of plys) $m \approx 100$, exact solution is infeasible

Resource Limitations:
Standard Approaches:
a) **Cutoff test**: On the depth limit

b) **Evaluation fn**: Heuristic scoring on the est. desirability of current state

Note for the evaluation fn values, behaviour is preserved under any monotonic transformation of itself, only the order matters

$\alpha - \beta$ pruning
α: best value (to MAX) found so far off the current path
β: best value (to MIN) found so far off the current path
Prune if $\alpha \geq \beta$

- Pruning does not affect final result only the size of the search tree

- Good move ordering improves effectiveness of pruning

- 'Perfect ordering': $O\left(b^{\frac{m}{2}}\right)$, hence we can double the depth of search

Watch https://www.youtube.com/watch?v=l-hh51ncgDI for example, lecture one is too simplistic

Non-deterministic Game:
Adversarial search where the actions/transitions are non-deterministic

ExpectiMiniMax, similar to **MiniMax** but accounts for chance nodes