CS2109S Midterm cheatsheet

Classical AI: By Khoo Jing Hong,Derrick

Defining AI problems

PEAS template

- Performance Measure (Indicate **correctness** and **efficiency**)
- Environment

Factors:

1. **Fully observable(vs Partially observable)**: Agent's sensors able to give agent access to the complete state of environment at each point in time (able to detect all aspects that are relevant to decision making)
2. **Deterministic(vs Stochastic vs Strategic)**: The next state of environment is completely determined by current state and action executed by agent. **Strategic** if environment is deterministic except for actions of other agents. Stochastic if model deals with probabilities (eg Autonomous driving).
3. **Episodic(vs Sequential)**: Experience is divided into atomic "episodes", and choice of action in each episode depends only on the episode itself. **Sequential** means current action may affect all future decisions.
4. **Static(vs Dynamic vs Semi-dynamic)**: Environment is unchanged while agent is deliberating. **Dynamic** if environment can change, **Semi-dynamic** if environment is static but performance degrades over time (ie chess).
5. **Discrete(vs Continuous)**: Discrete for clearly defined inputs. Applies to state of environment, to how time is handled and to the percepts and actions of the agents.
6. **Single agent(vs Multi-agent)**: Multi-agent if object with best behaviour described as maximising a performance measure whose values depends on another agent's behaviour.

- Actuators (To perform action upon knowing environment)
- Sensors (To perceive environment agent is in)

Solving problems by searching

Types of search:

Single state(Fully observable, deterministic):

- **State space** (Legal states and state rep)
- **Initial state and goal state**
- **Actions or successor function** (finite set of actions that can be executed at state S)
- **Goal test** (Explicit eg **n = 10** or implicit eg **Checkmate(x)**)
- **Path cost** (additive, c(n, a, n') will give the cost from state n to state n')
- **Solution will be a sequence of actions leading from initial state to goal state**

Search strategies:

Uninformed search

- b: Branching factor, d: Depth of optimal solution, l/m: Maximum depth of search tree, C*: cost of optimal solution

1. **Breadth-first search(BFS)**
   - FIFO queue
   - **Intuition**: all the nodes at depth d is generated before nodes at depth d + 1.
   - Optimal if all nodes have same cost
2. **Uniform-cost search(Dijkstra)**
   - PriorityQueue, where f(n) = g(n)
3. **Depth-first search(DFS)**
   - LIFO stack
   - Optimal if all solutions have same depth
4. **Depth-limited search(DLS)**
   - DFS with depth l
5. **Iterative-Deepening search(IDS)**
   - BFS pretending to be DFS
   - Key idea: Try different depths
6. **Bidirectional search**
   - **Intuition:** 2 * O(b^d/2) is smaller than O(b^d). Different search strategies can be employed for either half. However, *pred(succ(n))* and *succ(pred(n))* must be equal. Optimal if the evaluation function is the path cost.

Note:
Graph-search is just Tree-search above + memoization! (Hash-table to store visited states)

Informed Search

1. **Greedy Best-First Search**
   - Priority Queue, f(n) = h(n)
   - Expand the first node that appears to be closest to goal. Tree: Not complete, Graph: Complete only in finite search spaces
   - **Problem**: Might get stuck in loops and does not consider cost incurred thus far.
2. **A* search**
   - f(n) = g(n) + h(n), where g(n) is cost incurred thus far
   - **Whether it is cost-optimal:**
     - **Admissibility:** h(n) <= h*(n). If h(n) is admissible, then A* using tree-search is optimal.
     - **Consistency:** h(n) <= c(n,a,n') + h(n'), ie triangle inequality holds. If h(n) is consistent, then A* using graph-search is optimal.
       - Consistent(h1) => f(n') >= f(n), which means that f(n) is **nondecreasing** along any path.
   - **Dominance:**
     - If h2(n) >= h1(n) for all n
       - h2(n) is better for search (dominant heuristic)
   - Notes on A*
     - Admissibility is a sufficient condition for optimality, but not necessary. If h(n) overestimates by a constant factor, A* search is still optimal.
     - If the optimal solution has cost C* and the second-best has cost C1 and if h(n) overestimates some costs, but never by more than C* - C1 then A* is guaranteed to return cost-optimal solution.
     - All nodes with f(n) < C* is expanded. No nodes with f(n) > C* is expanded. A* is efficient because it prunes away nodes that are not necessary for finding an optimal solution.
     - If h(n) = -2g(n), then f(n) = -g(n) which means A* = DFS $\Rightarrow$ not complete.

Proving Admissibility and Consistency

For admissibility:
Heuristic is a relaxed solution to <>;
Against admissibility:
Give counterexample
For consistency: Heuristic value is at most reduced by true cost (ie 1) and hence, obeys triangle inequality.
Against consistency: Give counterexample.

Variants of A*:

Iterative Deepening A* (IDA*)
- Use f-cost as cutoff, similar idea to IDS

Memory-bounded A* (Simplified MA*)
- When memory is full, drop the node(s) with worst f-value
- Tradeoff: Might accidentally prune only path to goal

Local Search Algorithms
Find best state according to objective function.
Hill Climbing Search Find local maxima by travelling to neighbouring states with steepest ascent. It can get stuck on local maxima and plateaus. Solutions are:
•**K-Sideways Move**.
•**Stochastic**. Chooses a random uphill move, with probabilities based on the steepness of the move.
•**Random-Neighbours**.Useful when a state has thousands of successors. May escape shoulders, but little chance of escaping local optima.
•**Random-Restart**. Perform a fixed number of steps from some randomly generated initial steps, then restart if no maximum found. Can escape shoulders, and high chance of escaping local optima.
Simulated Annealing Escape local maxima by allowing some bad moves, but gradually reduce their frequency. Accept bad moves with probability that decrease exponentially with the "badness" of the move.

Beam Search Perform k hill-climbing searches in parallel. In local beam search, information is shared between parallel threads, allowing the algorithm to abandon unfruitful searches. However, the k states may end up clustering together, making it k-times slower. Stochastic beam search alleviate this problem by choosing successors with probability proportional to their value.
Genetic algorithms A successor state is generated by combining two parent states. (1) selection (who gets to be parent), (2) crossover point (recombination procedure), (3) mutation (randomly flip bits).

Notes on Local vs Informed
- Informed search is preferred if search space is small, and there is **always** a solution (ie there is a goal state)
- Local search is preferred in constraint satisfaction problems, and **path to goal node does not matter**.

Adversarial Search

Minimax algorithm Start from leaf node, then traverse up to root.
Alpha-Beta pruning To optimize minimax when both b and m is large, we can introduce pruning. Pruning occurs when α ≥ β. Pruning does not affect the final outcome. With good move ordering, time complexity can be reduced to O(bm/2), allowing us to examine twice as deep given the same computation power. Generally, the "best" nodes should be visited first. We can further optimize by using transpositions table to cache previously seen states or pre-computing the Opening and Closing moves.
Notes on minimax
1. It is optimal only against an optimal opponent.
2. As long as the relationship between leaves are retained, a translation or scale will not affect the final outcome (not true if multiplying by negative value).
3. Assume left-to-right ordering for α - β pruning, the leftmost branch is always evaluated.

Information Theory and Decision Trees

Entropy(I) =

$$I(\frac{p}{p+n},\frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Information Gain(IG) =

$$remainder(A) = \sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i})$$

- Information Gain (IG) or reduction in entropy from the attribute test

↑ IG = better

$$IG(A) = I(\frac{p}{p+n},\frac{n}{p+n}) - remainder(A)$$

- Choose the attribute *A* with the largest *IG*

Entropy of this node (entropy of training set) Entropy of children nodes

45

Improve Tree-Search / Graph-Search (run faster)
We can do this by pruning off unwanted branches. ie Settle an additional constraint during search.

Summary of algorithms

Criterion	BFS	Uniform cost	DFS	DLS	IDS	A*	Greedy BFS	Minimax
Complete	Yes	Yes	No	No	Yes	Yes	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^m)$	$O(b^m)$	$O(b^m)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^m)$	$O(b^m)$	$O(bm)$
Optimal	Yes	Yes	No	No	Yes	Yes	No	Yes