CS2109S CheatSheet AY24/25 —— @Jin Hang

Basic Concepts

PEAS Framework:
- **Performance measure:** define "goodness" of a solution
- **Environment:** define what the agent can and cannot do
- **Actuators:** outputs
- **Sensors:** inputs

Properties of Task Environment

Fully observable: (vs. **Partially** observable) An agent's sensors give it access to the **complete** state of env. at each point in time.
Deterministic: (vs. **Stochastic**) Next state of env. is completely determined by current state and action executed by the agent.
Strategic: env. is **deterministic** except for actions of other agents.
Episodic: (vs. **Sequential**) Agent's experience is divided into atomic "episodes". Each episode consists of agent perceiving and then performing a single action. The choice of action in each episode depends **only** on the episode itself.
Static: (vs. **Dynamic**) The env. is unchanged while an agent is deliberating.
Semi-dynamic: env. itself does not change with 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. Multi.) An agent operating by itself in an env.

Agents
- The agent function maps from percept histories to actions.
- An agent is completely specified by the agent function.
- A rational agent will choose actions that maximize 'P'
- **Exploration:** Learn more about the world
- **Exploitation:** Maximize gain based on current knowledge

Uninformed Search

Trick: Assume we know correct max depth for IDS/DLS
- Terminate (have sol): BFS/DLS/IDS
- Terminate (No sol): DLS
- Find Answer: BFS/DLS/IDS

Graph Search: Additional $O(b^m)$ memory to store all visited nodes to avoid revisiting states.

BFS: Queue

```
def BFS:
    insert initial state to frontier
    while frontier is not empty:
        state = frontier.pop()
        if state is goal: return solution
        for action in actions(state):
            next state = transition(state, action)
            frontier.add(next state)
    return failure
```

- $T(n) = 1 + b + b^2 + \cdots + b^d = O(b^d)$ (# nodes generated)
- $S(n) = O(b^d)$
- **Worst case:** Expand the last child in a branch
- **Complete:** Yes, if B is finite. If a solution **exists**, then the depth of the shallowest node s must be finite, so BFS must eventually search this depth. Hence, it's complete.
  - Apply goal-test when **PUSHING** a successor state to the frontier to preserve completeness.
- **Optimal:** Generally not optimal because it simply does not take costs into consideration when determining which node to replace on the frontier.
  - Optimal is if all edge costs are equivalent.

DFS: Stack
- $T(n) = O(b^m) \Rightarrow O(b^l)$ for DLS
- $S(n) = O(bm) \Rightarrow O(bl)$ for DLS
- **Complete:** No, when depth is infinite or can go back. There exists the possibility that DFS will faithfully yet tragically get "stuck" searching for the deepest node in an infinite-sized search tree ⇒ **may never find a solution**.
- **Optimal:** No, DFS simply finds the "**leftmost**" solution in the search tree without regard for path costs.

Iterative Deepening Search (IDS)
- $T(n) = b^0 + (b^0 + b^1) + \cdots + (b^0 + \cdots + b^d) = (d+1)b^0 + db^1 + (d-1)b^2 + \cdots + 2b^{d-1} + b^d = O(b^d)$
- $S(n) = O(bd)$
- Complete: Yes
- Optimal: Yes, if step cost is the same everywhere
- Overhead = (#IDS - #DLS)/#DLS

Uniform-cost Search (UCS): Priority Queue (path cost)

```
insert initial state to frontier
while frontier is not empty:
    state = frontier.pop()
    if state is goal: return solution
    for action in actions(state):
        next state = transition(state, action)
        frontier.add(next state)
return failure
```

- $T(n) = O(b^{C^*/\epsilon})$
- $S(n) = O(b^{C^*/\epsilon})$
- **Complete:** Yes, if $\epsilon > 0$ and C^* finite. If a goal state **exists**, it must have some finite length shortest path; ⇒ must find shortest path.
- **Optimal:** Yes, if $\epsilon > 0$
  - C^* cost of optimal solution
  - ϵ minimum edge cost. $\epsilon = 0$ may cause zero cost cycle

Bidirectional Search: Forward (from start) and Backward (from goal): $\Rightarrow 2 \times O\left(b^{d/2}\right) < O(b^d)$

Greedy Best-first Search: Priority Queue $(f(n) = h(n))$
- $T(n), S(n) = O(b^m)$, good heuristic gives improvement
- Complete and Optimal: **No**. Particularly when useing a bad heuristic function. It acts unpredictably from scenario to scenario, and can range from going straight to a goal state to acting like a badly-guided DFS and exploring all wrong areas.

A* Search: Priority Queue $(f(n) = g(n) + h(n))$
- $T(n), S(n) = O(b^m)$, good heuristic gives improvement
- Complete: Yes
- Optimal: Yes
  - If $h(n)$ **admissible** and using **tree search**.
  - if $h(n)$ is **consistent** and using **graph search**.
  - Work with negative edge weights.
  - A* **DO NOT** work with negative heuristics even admissible

Admissible: A heuristic $h(n)$ is admissible if $\forall n,\ h(n) \le h*(n)$
- $h*(n)$ is the true cost to reach the goal state from n.
- **Conservative:** An admissible heuristic **never** overestimates the cost to reach the goal

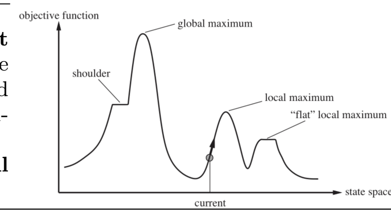
Manhattan Distance: $\mathrm{MD}(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$
Let p_i be the current location
- $\frac{1}{k}\sum_{i=1}^{k} \mathrm{MD}(p_i, g) \le \max\{\mathrm{MD}(p_i, g)\} \le h^*(n)$
- $\frac{1}{2}(\max\{\mathrm{MD}(p_i, g)\} + \min\{\mathrm{MD}(p_i, g)\}) \le \max\{\mathrm{MD}\} \le h^*(n)$
- $\max\{\min\{\mathrm{MD}(p_i, g)\}\} \le h^*(n)$

Consistent: $\forall n$, every successor n' of n generated by any action a, $h(n) \le c(n, a, n') + h(n')$, and $h(G) = 0$. h is consistent ⇒ $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \ge g(n) + h(n) = f(n) \Rightarrow f(n)$ is **non-decreasing** along any path
- **Theorem:** If $h(G) = 0$, then $h(n)$ is consistent ⇒ admissible
- Admissibility **does NOT** imply consistency.

Dominance: $\forall n, h_2(n) \ge h_1(n)$, then h_2 dominates $h_1 \Leftrightarrow h_2$ is better for search if admissible.
- No dominance relationship if $h(n)$ is not admissible.

Local Search
- **Goal:** minimizes the number of conflict violations
- Almost constant time, but **incomplete and sub-optimal**
- We are not interested in obtaining the solution path, but rather, reaching the goal state.
- When the search space is huge, using informed search could take a very long time.

Hill climbing algorithm
- The algorithm **does not** maintain a search tree but only the states and the corresponding values of the objective.
- May be trapped in **local maxima**




```
current = initial state
while True:
    neighbor = a highest-valued successor of current
    if value(neighbor) <= value(current):
        return current
    current = neighbor
```

Simulated Annealing: $\mathbf{P}(\text{curr, next, T}) = e^{\frac{\text{value(next)} - \text{value(curr)}}{T}}$

```
current = initial state
T = a large positive value
while T > 0:
    next = a randomly selected successor of current
    if value(next) > value(current): current = next
    else with probability P(current, next, T): current = next
    decrease T
return current
```

Theorem: if T decreases slowly enough, simulated annealing will find a global optimum with high probability.

Mini-Max

```
def minimax(state):
    v = max_value(state)
    return action in successors(state) with value v
```

```
def max_value(state):
    if is_terminal(state): return utility(state)
    v = -inf
    for action, next_state in successors(state):
        v = max(v, min_value(next_state))
    return v
def min_value(state):
    if is_terminal(state): return utility(state)
    v = inf
    for action, next_state in successors(state):
        v = min(v, max_value(next_state))
    return v
```

- $T(n) = O(b^m)$, $S(n) = O(bm)$, with depth first exploration
- **Complete:** Yes, if tree is finite
- **Optimal:** Yes, against optimal opponent

Alpha-beta Pruning: Won't change the decision

```
def alpha_beta_search(state):
    v = max_value(state, −inf, inf)
    return action in successors(state) with value v

def max_value(state, alpha, beta):
    if is_terminal(state): return utility(state)
    v = −inf
    for action, next_state in successors(state):
        v = max(v, min_value(next_state, aplha, beta))
        alpha = max(alpha, v)
        if v >= beta: return v
    return v

def min_value(state, alpha, beta):
    if is_terminal(state): return utility(state)
    v = inf
    for action, next_state in successors(state):
        v = min(v, max_value(next_state, alpha, beta))
        beta = min(beta, v)
        if v <= alpha: return v
    return v
```

- Good move order improves effectiveness of pruning $\Rightarrow O\left(b^{\frac{m}{2}}\right)$

Evaluation Functions: There is **no** notion of admissibility and consistency in local search and adversarial search.

Supervised Learning

- **Regression:** predict **continuous** output
- **Classification:** predict **discrete** output

Formalism Assume that y is generated by $f : x \to y$. We want to find a hypothesis h:$x \to \widehat{y}$ (from a hypothesis class H) s.t. $h \approx f$ given a training set $\{(x_1, f(x_1)), ..., (x_N, f(x_N))\}$ We use a **learning algorithm** to find this hypothesis

Error: If the output of the hypothesis is a continuous value

- MSE= $\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$, where $\hat{y}_i = h(x_i)$ and $y_i = f(x_i)$
- MAE= $\frac{1}{N}\sum_{i=1}^{N}\|\widehat{y}_i - y_i\|$

Decision Tree

- **Continues-valued Attributes:** Define a discrete-valued input attribute to partition the values into a discrete set of intervals.
- **Missing Values:**
  - Assign the most common value of the attribute
  - Assign the most common value of attribute with same output
  - Assign probability to each possible value and sample
  - Drop the attribute
  - Drop the rows

```
def DTL(examples, attributes, default):
if examples is empty: return default
if examples have the same classification: return classification
if attributes is empty: return mode(examples)
best = choose_attribute(attributes, examples)
tree = a new decision tree with root best
for each value vi of best:
    examples_i = {rows in examples with best = vi}
    subtree = DTL(examples_i, attributes−best, mode(examples))
    add a branch to tree with label vi and subtree subtree
```

- **Accuracy:** $\frac{\text{TP+TN}}{\text{TP+FN+FP+TN}}$
- **Recall:** $R = TP/(TP + FN)$
- **Precision:** $P = TP/(TP + FP)$
- **F1 Score:**

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

Predicted Label \ Actual Label	Positive	Negative
Positive	TP	FP
Negative	FN	TN

- **Entropy:** $I(P(v_1), ..., P(v_n)) = -\sum_{i=1}^{n} P(v_i)\log_2 P(v_i)$
- **Remainder:** Entropy of children nodes
  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:** $IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \text{remainder}(A)$

Overfitting: DT performance is perfect on training data, but worse on test data. DT captures data perfectly, including the noise.

Occam's Razor: Prefer short/simple hypotheses. In favor:

- Short/simple hypothesis that fits the data is unlikely to be coincidence
- Long/complex hypothesis that fits the data may be coincidence

Against:

- Many ways to define small sets of hypotheses (e.g., trees with prime number of nodes that uses attribute beginning with "Z")
- Different hypotheses representations may be used instead

Appendix

Magic Entropy Number

- $I(\frac{1}{2}, \frac{1}{2}) = 1$
- $I(\frac{1}{4}, \frac{3}{4}) = 0.811$
- $I(\frac{1}{5}, \frac{4}{5}) = 0.722$
- $I(\frac{1}{6}, \frac{5}{6}) = 0.650$
- $I(\frac{1}{7}, \frac{6}{7}) = 0.592$
- $I(\frac{3}{7}, \frac{4}{7}) = 0.985$
- $I(\frac{3}{10}, \frac{7}{10}) = 0.881$
- $I(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) = 1.585$
- $I(\frac{1}{6}, \frac{1}{3}, \frac{1}{2}) = 1.459$
- Math: $\log_2 N = \lg N / \lg 2$
- $I(1, 0) = 0$
- $I(\frac{1}{3}, \frac{2}{3}) = 0.918$
- $I(\frac{2}{5}, \frac{3}{5}) = 0.971$
- $I(\frac{2}{7}, \frac{5}{7}) = 0.863$
- $I(\frac{3}{8}, \frac{5}{8}) = 0.954$
- $I(\frac{2}{12}, \frac{5}{12}, \frac{5}{12}) = 1.483$
- $I(\frac{1}{7}, \frac{2}{7}, \frac{4}{7}) = 1.379$

Decision Tree Data Table Work Space

#							
0							
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							