Wang Xiyu

# 1 Overview

- Week 1-3: Classical AI, search algorithms
  1. Uninformed search
  2. Local search: hill climbing
  3. Informaed search: $A^*$
  4. Adversarial search Minimax
- Week 4-7: Classical ML
  1. Decision trees
  2. Linear/Logistic regression
  3. Kernels and support vector machines
  4. "Classical" unsuperivese learning
- Week 10-12: Modern ML
  1. Neural networks
  2. Deep learning
  3. Sequential data
- Week 13: Misc.

# 2 AI: Computers Trying to Behave Like Humans

- **PEAS Framework:**
  - **Performance measure:** define "goodness" of a solution
  - **Environment:** define what the agent can and cannot do
  - **Actuators:** outputs
  - **Sensors:** inputs
- Agent function is sufficient.
- Common agent structures (to define an AI agent):
  - Reflex
  - Goal-based
  - Utility-based
  - Learning
  - (Others possible; can mix and match!)
- Exploration vs exploitation

# 3 Problem Statement

fully observable $\wedge$ deterministic $\wedge$ static $\wedge$ discrete $\implies$ only need to observe once
To solve a prob using search:

- A goal or a set of goals
- a model of the eironment
- a search algorithm

goal formulation -> problem formulation -> search -> execute

1. goal formulation
2. problem formulation, eg. path finding
   - states: nodes representation invariant:: abstract states should correspond to concrete states
   - initial state: starting node
   - goal states/test: dest node
     Goal test: define the goal using a function $is\_goal$
   - actions: move along an edge :: $|actions(state)| \leq branching\_factor$
   - transition model: $f(curr\_state, action) \implies next\_state$
   - action cost function: see edges
3. Important facts:
   - Representation Invariant: ensure that the abstract states correspond to concrete states
   - Goal Test: Goal defined via a function $is\_goal$
   - Action: a set of $action(state)$, $|actions(state)| \leq branching\_factor$
   - Transition model: $f(curr\_state, action) \implies next\_state$

## Search

### Uninformed search

No information that could guide the seaech: no clue how good a state is

```
create frontier
// create visited // with vsited memory
insert Node(initial_state) to frontier
while frontier is not empty:
    node = frontier.pop()
    if node.state is goal:
        return solution
// if node.state in visited: // with vsited memory
//    continue
// visited.add(state)
    for action in actions(node.state):
    next_state = transition(node.state, action)
    frontier.add(Node(next_state))
return failure
```

Different subvariant of tree search uses differen DS for the frontier.

| Search Type | Data Structure for Frontier |
|---|---|
| BFS | Queue |
| DFS | Stack |
| UCS (Uniform-cost Search) | Priority Queue |

## Depth limited search

limit the search to depth l
backtrack when the limit is hit.
time complexity: exponential to search depth
space complexity: size of the frontier

```
create frontier
tier = 0
insert Node(initial_state) to frontier
while (!empty(frontier)) && (tier <= limit):
    node = frontier.pop()
    tier++
    if node.state is goal:
        return solution
    for action in actions(node.state):
    next_state = transition(node.state, action)
    frontier.add(Node(next_state))
return failure
```

## Iterative deeptening search

search with depth from 0 to inf
return soln when found. Both complete

```
create frontier
tier = 0
insert Node(initial_state) to frontier
while (!empty(frontier)) && (tier <= limit):
    node = frontier.pop()
    tier++
    if node.state is goal:
        return solution
    for action in actions(node.state):
    next_state = transition(node.state, action)
    frontier.add(Node(next_state))
return failure
```

## Summary

| **Name** | **Time Complexity*** | **Space Complexity*** | **Complete?** | **Optimal?** |
|---|---|---|---|---|
| Breadth-first Search | Exponential | Exponential | Yes | Yes |
| Uniform-cost Search | Exponential | Exponential | Yes | Yes |
| Depth-first Search | Exponential | Polynomial | No# | No |
| Depth-limited Search | Exponential | Polynomial** | No** | No** |
| Iterative Deepening Search | Exponential | Exponential** | Yes | Yes |

# Not complete if not tracking visited nodes, search may stuck in loop before visiting all nodes.
* In terms of some notion of depth/tier
** If used with DFS

# 4 Local Search

Systematic search: typically complete and optimal under certain constraints. However intractable sometimes
Local search: typically incomplete and suboptimal, but has anytime property, ie. longer time -> better solution. Able to provide good enough solution under reasonable amount of time.

## 4.1

1. Start at random position in the state space
2. iteratively move from a state to another neighouuring state vie perturbation or construction
3. solution is the final state

$$\text{State space: all possible configuration} \tag{1}$$
$$\text{Search space: a subset of state space that will be explored} \tag{2}$$

### 4.1.1 Perturbation search

- Search space: complete candidate solutons
- search step: modification of one or more solution

For example: swap a path with another path

### 4.1.2 Constructive search

- partial candidate soluton
- extension of one or more solution

For example: path finding

## 4.2

goal formulation -> problem formulation -> search -> execute

1. goal formulation
2. problem formulation, eg. path finding
   - states: nodes representation invariant:: abstract states MAYNOT directly correspond to concrete states
   - initial state: starting node, a candidate solution
   - goal states/test: dest node [optional]
     Goal test: define the goal using a function $is\_goal\ f(curr\_state, action) \implies next\_state$
   - Successor function: a funciton that generates neighbouring states by applying modifications from the curren tstate. This defines the local search space

## 4.3 Evaluation function

A math function that assess the quality or desireability of the solution.
Some solutions may be unacceptable but there are some less bad than the others.

### 4.4 Hill Climbing/ Greedy Local Search

```
curr_state = init_state
while 1:
    best_succ = best(successor(curr_state))
    if (eval(best) <= eval(curr_state))
        return curr_state
    curr_state = best_succ
```

### 4.5 State space landscape

- Global max:
- Local max:
- shoulder:

## 5 Adversarial search

### 5.1 Classical adversarial games

- Fully observable
- Deterministic
- discrete
- No infinite run
- 2-player zero-sum
- turn taking

#### termns

- Player: agent
- Turn:
- Move
- End state
- winning conditon
-

### 5.2 Problem formulation in adversarial search

- states: nodes representation invariant:: abstract states MAYNOT directly correspond to concrete states
- initial state: starting node, a candidate solution
- Terminal State: state the outcome of the game when it terminates
- Utility function: output the value of a state from the perspetive of our agent

## 5.3 Minimax

In the view of A, A try to maximize the outcome of the game, B will try to minimize A's outcome, as the gam is zero sum

- expend(state) => [a]

```
max_value(state):
    if is_terminal(state): return utility(state)
    v = -∞
    for next_state in expend(state):
        v = max(v, min value for player A in next_state)
    return v

min_value(state):
    if is_terminal(state): return utility(state)
    v = ∞
    for next_state in expend(state):
        v = min(v, max value for player A in next_state)
    return v

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

## 5.4 Alpha-beta prunning

From the viewpoint of the MAX player:

- $\alpha$: the value of the best choice for the MAX player so far
- $\beta$: the value of the best choice for the MIN player so far

An optimized version of the Minimax algorithm using prunning:

```
max_value(state, α, β):
    if is_terminal(state): return utility(state)
    v = -∞
    for next_state in expend(state):
        v = max(v, min(v, α, β))
    return v

min_value(state, α, β):
    if is_terminal(state): return utility(state)
    v = ∞
    for next_state in expend(state):
        v = min(v, max(v, α, β))
    return v

alpha-beta search(state):
    v = max_value(state, -∞, ∞) // initialized α to be -∞, β to ∞
    return action in expend(state) with value v
```

# 6 Learning agent

For problems that the function are difficult to specify, solutions re intractale to compute in general. Typically episodic,

$$DL \subset ML \subset AL$$

## 6.1 supervised

Learn the mapping input, feedback given -> output, given a dataset, it minimizes the difference betwenn the prediction and the provided correct ans using a leanign algorithm e.g. image ientification

1. Train phase: try minimizing the diff between pred and correct ans given using the training set, resulting in a trainign agen function, known as the model/hypothesis
2. Testing/evaluation phase: using a test set to measure the performance of the model. The performance on unseen data measures the generalization of the model

**Task**

- Classification: to predict discreate labels or catagories on the input features
- Regression: predict continuous numetical value based on input features

**Dataset**

$$D = \bigcup_{[1,n]} \{(x^{(i)}, y^{(i)})\}$$

**True data generation function**

$$y = f^*(x) + \epsilon$$

where $f^*(x)$ is true but unknow, which generates the label from the input features; $\epsilon$ is some noise or error term, which account for the randomness or imperfection in the date generating process.
the goal is to find a function that best approximately $f^*(x)$

**Hypothesis class**

THE set of models or functions that maps from inputs to outputs $h : X \implies Y$ that can be learned by a learing algorithm. Each element of the hypo clas $h \in \mathcal{H}$

**LEarnign algorithm**

$$A(D_{train}, H_{hypo}) = h(x), h \in \mathcal{H} \approx f^*(x)$$

**PErformance measure**

$$h(x) \approx f^*(x)$$

$$PM(D_{test}, h \in H_{hypo}) \mapsto$$

Try the hypothesis h on a new set of examples (test data)

**Regression: error**

$$\text{Absolute error} = |\hat{y} - y| \tag{3}$$

$$\text{Squared error} = (\hat{y} - y)^2 \tag{4}$$

Mean squared error

$$MSE = \frac{1}{N}\sum_{i=0}^{N}(\hat{y}^{(i)} - y^{(i)})^2$$

Mean absolute error

$$MAE = \frac{1}{N}\sum_{i=0}^{N}|\hat{y}^{(i)} - y^{(i)}|$$

where

$$\hat{y}^{(i)} = h(x^{(i)})$$

Accuracy:

$$A = \frac{1}{N}\sum_{i=1}^{n}\mathbb{1}_{\hat{y}^{(i)}=y^{(i)}}$$

**confusion matrix**

True positive, false positive, false negatve, true negative (2, 1, 3, 4 quadrnt)

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

$$Precision = \frac{TP}{(TP + FP)}$$

maximize if FP is costly

$$Recall = \frac{TP}{(TP + FN)}$$

maximize recall if FN is dangerous

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

## 6.2 Decision Tree

Greedy, top-down, recursive, use

```
DTL(examples, attributes, default):
    if (examples = ∅): return default
    if (∀e ∈ example, e has the same classification c): return c
    if (attributes = ∅): return mode(examples)

    best = choose_attribute(attributes, examples)
    tree = new decision tree with root test best
    for v_i of best do:
        examples_i = {e|e ∈ examples, e.best = v_i}
        subtree = DTL(examples, attributes \ best, mode(examples))
        tree.add(v_i: subtree)
```

$$f : [\text{attribute vector}] \mapsto Boolean$$

Basically nested if-else

**Expresssioveness**

Decision trees can express any function of the input attributess. Trivially, Consistent trainign set $\rightarrow$ Consistent decision tree, but unlikely to generalize to new examples
Each row in the truth table is represented as a path in the decison tree

**Size of the hypothesis class**

For n boolean attributes, there are n boolean func, n distinct truth tables rach with $2^n$ rows $\rightarrow 2^{2^n}$ decision trees

**Informativeness**

Ideally we want to sekect an attibute that aplits the examples into all positive or all negative.
Entropy: The higher the more random, thus less informative

$$I(P(v_1)...P(v_k)) = -\sum_{i=1}^{k} P(v_i) log_2 P(v_i)$$

For data set contains boolean outputs,

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

**Information gain**

Information gain = entropy of this node - entropy of children nodes

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

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

**Decision tree pruning**

- By sample size
- by max search depth

## 6.3 unsupervised

FInd pattern. No feedback given. e,g, group images by char

## 6.4 Reinforment

Trial and error, reward given based on observation adn action. eg. chess