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 enironment
- 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