VE492 Midterm Recitation Class

Yunpeng Jiang, Zhengjie Ji

UMJI

{*jyp9961, jizhengjie*}*@sjtu.edu.cn*

June 22, 2021

Table of Contents

Search and Planning - Outline

Background knowledge

- Agent Types: Simple Reflex Agents, Model-based Reflex Agents, Goal-based Agents, Utility-based Agents, Learning Agents
- Environment Types: Observable/Partially Observable, Single agent/Multiple agents, Deterministic/Non-deterministic, Static/Dynamic, Discrete/Continuous, Episodic/Sequential
- Complexity Theory

Search

- Search Problems: Action set, Transition model, Cost function, Start state / goal state
- Search Methods: Uninformed Search, Informed Search

Background knowledge

Rationality

maximizing "expected utility" Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

Rational Agents

- The **performance measures**, which determine the degree of success.
- Agent's **Percept Sequence** till now.
- The agent's **prior knowledge about the environment**.
- The **actions** that the agent can carry out.

Task Environment to Design a Rational Agent

Performance Measure, Environment, Actuators, and Sensors (PEAS).

Define the following terms

1. Initial State
2. Successor Function
3. Path Cost Specification
4. Goal Test

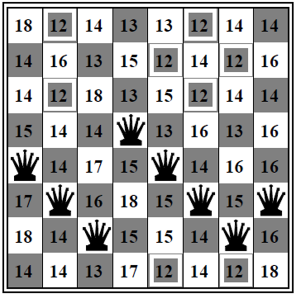


Figure: 8-queens

Search Methods

Uninformed Search

- Depth-First Search (Tree Search / Graph Search)
- Breadth-First Search
- Iterative Deepening Search
- Uniform-Cost Search

Informed Search

- Greedy Search
- A* Search (Tree Search / Graph Search)

Search Methods - Uninformed Search

	DFS(Tree)	DFS(Graph)	BFS	UCS	IDS
Completeness	No	Yes	Yes	Yes	Yes
Time	$O(b^m)$	$O(b^m)$	$O(b^d)$	$O(b^{\frac{C^*}{\epsilon}})$	$O(b^d)$
Space	$O(bm)$	$O(bm)$	$O(b^d)$	$O(b^{\frac{C^*}{\epsilon}})$	$O(bd)$
Optimal	No	No	No	Yes	No

-**b** branch factor (max num of successors of any node)

-**d** depth of optimal solution

-**m** maximum length of a path in the spate space (could be ∞)

-**C*** solution cost

-ϵ minimum arc cost

Search Methods - Informed Search

	Greedy	A*(Tree / Graph)
Completeness	No	Yes
Time	/	Exponential in length of solution
Space	/	Keeps all nodes in memory
Optimal	No	with admissible / consistent heuristics

Admissible Heuristics

heuristic cost $\leq$ actual cost to goal

Consistency of Heuristics

triangular inequality, heuristic "arc" cost $\leq$ actual cost for each arc

Game Trees - Overview

- Type of Games: Zero-Sum Games / General Games
- Adversarial Search with Minimax ($O(b^m)$)
- Resource Limits: Bounded lookahead / Game Tree Pruning $O(b^{m/2})$

Practice: Alpha-Beta Pruning

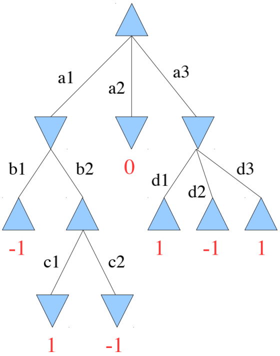
function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in ACTIONS(*state*) with value v

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v



Practice: Alpha-Beta Pruning

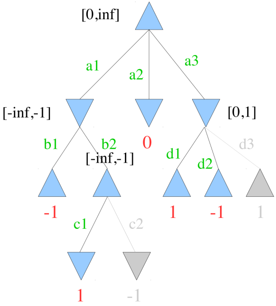
function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v



Decision Theory and Game Theory - Outline

- Multi-agent search: Minimax, Expectimax
- Games with chance
- Decision Theory
- Game Theory: Strategy, Solution

Practice: Nash Equilibrium

Morra Game

Morra is a hand game that dates back thousands of years to ancient Roman and Greek times. We will study a simplified version of this game. Each player simultaneously reveals their hand, extending one or two fingers, and calls out a number (2, 3 or 4). If one player guessed correctly the total number of extended fingers and the other was wrong, the latter pays that number in dollars to the former. In all other cases, nobody pays anything. We denote (k, s) the pure strategy that consists in extending k fingers and guessing s as the total number.

- Is it a zero-sum game? Express this game in normal form.
- Assume the row player plays a mixed strategy where the probabilities for strategies (1, 2),(1, 3),(2, 3), and (3, 4) are respectively denoted α, β, γ and δ. Write a system of inequalities that expresses that this mixed strategy is a Nash equilibrium.

Practice: Nash Equilibrium

- Yes, therefore we can only give the payoffs for one of the player. Here's table for the row player.

	(1, 2)	(1, 3)	(2, 3)	(2, 4)
(1, 2)	0	2	-3	0
(1, 3)	-2	0	0	3
(2, 3)	3	0	0	-4
(2, 4)	0	-3	4	0

- The inequalities are:

$$
\begin{aligned}
& -2\beta + 3\gamma \geq 0 \\
& 2\alpha - 3\delta \geq 0 \\
& -3\alpha + 4\delta \geq 0 \\
& 3\beta - 4\gamma \geq 0 \\
& \alpha + \beta + \gamma + \delta = 1 \\
& \alpha \geq 0; \beta \geq 0; \gamma \geq 0; \delta \geq 0
\end{aligned}
$$

Markov Decision Process(MDP), Reinforcement Learning(RL)

The bellman equations

$$Q(s_t, a_t) = \sum_{s'} T(s_t, a_t, s_{t+1})[R(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1})]$$

$$V(s_t) = \max_{a_t} \sum_{s_{t+1}} T(s_t, a_t, s_{t+1})[R(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1})]$$

MDP

MDP Definition

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition function $T(s, a, s')$, $P(s'|s, a)$
- A reward function $R(s, a, s')$
- A discount factor γ
- A start state
- Maybe a terminal state
- looking for an optimal policy $\pi^* : S \to A$

RL definition

- A set of states $s \in S$
- A set of actions (per state) A
- A model $T(s, a, s')$
- A reward function $R(s, a, s')$
- A discount factor γ
- looking for a policy $\pi(s)$
- In RL, we don't know the transition function T function or the reward function R.

How to Solve MDP

How to solve MDP problems

- value iteration
- policy iteration: policy evaluation + policy improvement

value iteration

- Start with $V_0(s) = 0$
- Use bellman equation to update the value of each state
- Repeat until convergence

policy iteration

- Policy evaluation: For current policy π, find the value of each state
- Policy improvement: In every state s_t, take the action a_t that maximize $Q(s_t, a_t)$.

How to Solve RL

How to solve RL problems

- Q-learning

Q-learning

- receive a sample (s_t, a_t, s_{t+1}, r_t)
- target Q value $Q_{target}(s_t, a_t) = r_t + \gamma \underset{a_{t+1}}{max} Q(s_{t+1}, a_{t+1})$
- update the original Q value $Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha Q_{target}$, where α is the learning rate.

ϵ-greedy

With probability ϵ, act randomly; otherwise, act according to the current policy.

Formulas for MDP

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall s$

Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall s$

CSP

CSP

- The state is defined by variables X_i with values form a domain D
- The goal test is a set of constraints specifying allowable combinations of values for subsets of variables.
- Varieties of constraints: unary constraints, binary constraints; higher-order constraints

How to solve CSPs

How to solve CSPs

- Backtracking search: DFS+variable-ordering+fail-on-violation
- Forward checking: cross off avlues that violate a constraint when added to the existing assignment
- Arc consistency: Given an arc $X \to Y$, for every x in the tail X, there is some y in the head Y which could be assigned without violating a constraint. Otherwise, delete x from the tail.
- Minimum remaining values: choose the variable with the fewest legal left values in its domain.
- Least constraining value: choose the variable that rules out the fewest values in the remaining variables.

How to solve CSPs

Backtracking search

function BACKTRACKING-SEARCH(*csp*) **returns** solution/failure
  return RECURSIVE-BACKTRACKING({ }, *csp*)

function RECURSIVE-BACKTRACKING(*assignment, csp*) **returns** soln/failure
  if *assignment* is complete **then return** *assignment*
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment, csp*)
  for each *value* **in** ORDER-DOMAIN-VALUES(*var, assignment, csp*) **do**
    if *value* is consistent with *assignment* given CONSTRAINTS[*csp*] **then**
      add $\{var = value\}$ to *assignment*
      result ← RECURSIVE-BACKTRACKING(*assignment, csp*)
      if $result \neq failure$ **then return** *result*
      remove $\{var = value\}$ from *assignment*
  return *failure*

How to solve CSPs

Arc Consistency

```
function AC-3(csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function REMOVE-INCONSISTENT-VALUES(Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi]; removed ← true
   return removed
```

Questions

- What are the bellman equations?
- In value iteration, how do we update the value of each state?
- In policy improvement, how do we compute the value of the current state?
- In Q learning, how do we compute the target value of the current state-action pair?
- What does ϵ-greedy mean?
- How can we use feature representation to approximate the Q function?
- How can we use backtracking search to solve CSP?
- When can we check the arc consistency in the process of backtracking search?

The End