Search - Lecture 0

Search, knowledge, uncertainty, optimalization, learning, neural network

agent: entity that perceives its environment and acts upon that environment

state: a configuration (initial state)

actions: choices that can be made in a state, Actions(s) (s: state)

transition model: a description of what state results from performing any applicable action in any state
Result(s, a)

state space: the set of all states reachable from the initial state by any sequence of actions

goal test: way to determine whether a given state is a goal state

path cost: numerical cost associated with a given path

frontier: all the possible paths, we haven't taken yet

A Search problem has 5 things

initial state, actions, transition model, goal test, path cost funct

We would like to find the optimal solution

Node - data structure:
- a state
- a parent (a node that generated this node)
- an action (action applied to parent to get this)
- path cost (from initial state to node)

Approach

- Start with frontier = initial state
- Repeat
  - if frontier is empty then not possible ↩ if/else
  - remove a node from the frontier
  - if node contains goal state, return the solut
  - Expand node, add resulting nodes to the frontier ↩ if/else

Drawbacks: you can lose time on repeating states

Revised Approach

- Start with a **frontier** that contains the initial state.
- Start with an empty **explored set**. ← it is important how we choose
- Repeat:
  - If the frontier is empty, then no solution.
  - Remove a node from the frontier. ← we use a stack
  - If node contains goal state, return the solution.
  - Add the node to the explored set.
  - **Expand** node, add resulting nodes to the frontier if they aren't already in the frontier or the explored set.

Depth first search

DFS

not always optimal

Breadth-first search - BFS finds the shortest

search algorithm that always expands the shallowest node in the frontier

look at maze.py

Uninformed Search vs. Informed Search

DFS, BFS

DFS no, BFS yes

search strategy that uses problem specific knowledge to find solutions more efficiently

Greedy Best-First Search

search algorithm that expands the node that is closest to the goal, as estimated by a heuristic function h(n)

we need to define it

Manhattan distance

x-y distance

doesn't allways find best route

A* Search

search algorithm that expands node with lowest value of $g(n) + h(n)$

$g(n)$ = cost to reach node
$h(n)$ = estimated cost to goal - Manhattan, etc.

this is optimal if

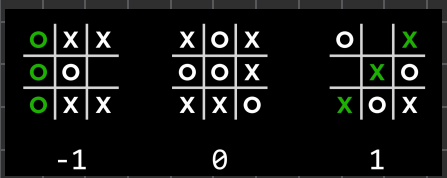
- $h(n)$ is admissible
- $h(n)$ is consistent
  $h(n) \leq h(n') + c$

uses more memory

Adversarial Search

the agent has to work against something eg: tic tac toe

Minimax



- Max (x) aims to maximise score
- Min (x) aims to minimise score

– score

Min Player

Max Player

Recursive

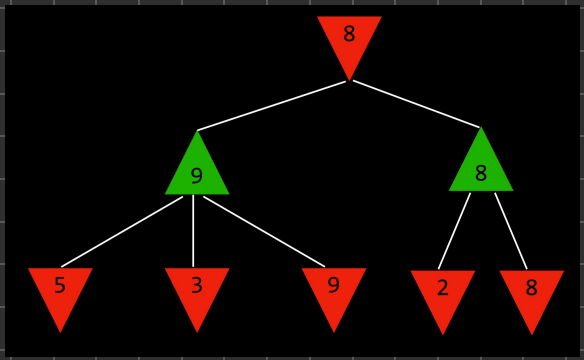
Game

- S_0 : initial state
- PLAYER(s) : returns which player to move in state s
- ACTIONS(s) : returns legal moves in state s
- RESULT(s, a) : returns state after action a taken in state s
- TERMINAL(s) : checks if state s is a terminal state
- UTILITY(s) : final numerical value for terminal state s

empty game array

– this is for the AI

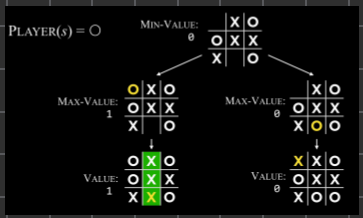
– score



we put ourselves in the opposite players shoes

min - max - min - max ....

recursively



Pseudo Code Minimax

- Given a state *s*:
  - MAX picks action *a* in ACTIONS(*s*) that produces highest value of MIN-VALUE(RESULT(*s*, *a*))
  - MIN picks action *a* in ACTIONS(*s*) that produces smallest value of MAX-VALUE(RESULT(*s*, *a*))

```
function MAX-VALUE(state):
  if TERMINAL(state):
    return UTILITY(state)
  v = -∞
  for action in ACTIONS(state):
    v = MAX(v, MIN-VALUE(RESULT(state, action)))
  return v
```

Min-Value is the opposite

Optimalization – Alpha Beta Pruning

this is inefficient, you need to skip branches, that are worse, than what we found so far.

There are too many states to → Depth-limited Minimax ← this needs an Evaluation Function

Knowledge - lecture 1

knowledge-based agents: agents that reason by operating on internal representations of knowledge

sentence: an assertion about the world in a knowledge representation language

propositional logic:

propositional symbols: P, Q, R – facts

logical connectives: ¬ (not), ∧ (and), ∨ (or), → (implication), ↔ (biconditional)

not

P	$\neg P$
false	true
true	false

and

P	Q	$P \wedge Q$
false	false	false
false	true	false
true	false	false
true	true	true

or

P	Q	$P \vee Q$
false	false	false
false	true	true
true	false	true
true	true	true

implication

P	Q	$P \rightarrow Q$
* false	false	true
* false	true	true
true	false	false
true	true	true

* makes no claim

biconditional

P	Q	$P \leftrightarrow Q$
false	false	true
false	true	false
true	false	false
true	true	true

model: assignment of a truth value to every propositional symbol ("a possible world")

knowledge base: a set of sentences known by a knowledge based agent

entailment: in every model in which sentence α is true, β is also true

$\alpha \models \beta$

inference: the process of deriving new sentences from old ones

Model Checking

- To determine if $\text{KB} \models \alpha$:
  - Enumerate all possible models.
  - If in every model where KB is true, α is true, then KB entails α.
  - Otherwise, KB does not entail α.

P: It is a Tuesday. Q: It is raining. R: Harry will go for a run.

KB: $(P \wedge \neg Q) \rightarrow R$ P $\neg Q$

Query: R

P	Q	R	KB
false	false	false	F
false	false	true	T
false	true	false	F
false	true	true	T
true	false	false	F
true	false	true	T
true	true	false	F
true	true	true	T

Knowledge Engineering: describe real world problems with logical symbols

logic puzzles can be solved by models

Model checking isn't very efficient

Inference Rules

Modus Ponens	$\alpha \to \beta$; α	β
And Elimination	$\alpha \wedge \beta$	α
Double Negation Elimination	$\neg(\neg\alpha)$	α
Implication Elimination	$\alpha \to \beta$	$\neg\alpha \vee \beta$
Biconditional Elimination	$\alpha \leftrightarrow \beta$	$(\alpha \to \beta) \wedge (\beta \to \alpha)$
De Morgan's Law	$\neg(\alpha \wedge \beta)$	$\neg\alpha \vee \neg\beta$
Distributive Law	$(\alpha \wedge (\beta \vee \gamma))$ $(\alpha \vee (\beta \wedge \gamma))$	$(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$ $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
	$P \vee Q, \neg P$	Q
	$P \vee Q, \neg P \vee R$	$Q \vee R$

clause: a disjunction of literals

↑ connected with OR conjunction = AND

conjunctive normal form: logical sentence that is a conjunction of clauses

Theorem Proving

- initial state: starting knowledge base
- actions: inference rules
- transition model: new knowledge base after inference
- goal test: check statement we're trying to prove
- path cost function: number of steps in proof

Conversion to CNF conjuctive normal form

- Eliminate biconditionals
  - turn $(\alpha \leftrightarrow \beta)$ into $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$
- Eliminate implications
  - turn $(\alpha \rightarrow \beta)$ into $\neg\alpha \vee \beta$
- Move $\neg$ inwards using De Morgan's Laws
  - e.g. turn $\neg(\alpha \wedge \beta)$ into $\neg\alpha \vee \neg\beta$
- Use distributive law to distribute $\vee$ wherever possible

$(P \vee Q) \rightarrow R$

$(\neg P \vee R) \wedge (\neg Q \vee R)$

$P \vee Q \vee S \quad \neg P \vee R \vee S$

$(Q \vee R \vee S)$

eliminate duplicate
eliminate opposite

$P \quad |\neg P| \quad ( ) \quad$ empty

Inference by Resolution

- To determine if $\mathbf{KB} \models \alpha$:
  - Convert $(\mathbf{KB} \wedge \neg\alpha)$ to Conjunctive Normal Form.
  - Keep checking to see if we can use resolution to produce a new clause.
    - If ever we produce the **empty** clause (equivalent to False), we have a contradiction, and $\mathbf{KB} \models \alpha$.
    - Otherwise, if we can't add new clauses, no entailment.

:3 yaay

:3 yaay

Does $(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C)$ entail A?

$(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C) \wedge (\neg A)$

$(A \vee B) \quad (\neg B \vee C) \quad (\neg C) \quad (\neg A) \quad (\neg B) \quad (A) \quad ( )$

First Order Logic

$Person(Minerva)$	Minerva is a person.
$House(Gryffindor)$	Gryffindor is a house.
$\neg House(Minerva)$	Minerva is not a house.
$BelongsTo(Minerva, Gryffindor)$	Minerva belongs to Gryffindor.

Universal Quantification

$\forall x.\ BelongsTo(x, Gryffindor) \rightarrow \neg BelongsTo(x, Hufflepuff)$

For all objects x, if x belongs to Gryffindor, then x does not belong to Hufflepuff.

Anyone in Gryffindor is not in Hufflepuff.

Existential Quantification

$\exists x.\ House(x) \wedge BelongsTo(Minerva, x)$

There exists an object x such that x is a house and Minerva belongs to x.

Minerva belongs to a house.