Lecture 1

Introduction

{Intelligent agent: takes the best possible action in a solution}

AI divided in 4 cartegories: (1)thinking humanly, (2)thinking rationally, (3)acting humanly, (4)action rationally

(1)Turing Test[…]-->major component of AI:natural language undestanding, knowledge representation, automatic reasoning, learnign, computer vision and robotics

(1)congnitive model, now cognitivescience!=AI

(2)"laws of thought", this study initiated a field called logic

[socrates is a man, all men are mortal--->socrates is mortal]

(3)

Agent: entity that preceives and acts

Rational agent: agent that acts so as to achive the best outcome(best expected outcome) / thinking shuold be in the service of rational action

Our goal is to design rational agents, but due to compotutional limitations perfect rationality is unachievable

[historical introduction]

State of the art "the AI planning techniques generated in hours a plan thtat would have taken weeks with older methods.

[examples]

Lecture 2

Agents

An agent is anything that can be viewed as percieving its enviroment through sensors and acitng upon thta enviroment through actuators.

Ex: human agent (sensors: eyes, ears, ..-----actuators:hands, legs, ...)

The agent function maps percept histories to actions [f:p-->A]

The agent program runs on the physical architecture to produce f

An agent should strive to "do the right thing" based on what it can perceive and the actions it can perform. The right action is the one that will cause the agent to be the most successful.

So we need: --> **Performance measure** : an objective criterion for success of an agent behavior.

Rational agent : for each possible percept sequence it should select an action that is expected to maximize its performance maesure, given the evidence provided by the percept sequence, given whatever built-in knowledge the agent has.

Rationally!=omniscience

Agents can perform actions to obtain information(useful)

It is autonomous if its behavior is determined by its own experience.

Task enviroment : the problems for which artificial agents are solutions. It's specified by PEAS (performance measure, enviroment, actuators, sensors).

Ex: autonomous taxi driver:

P: safe, fast, legal... E:roads, others cars in traffic, ... A:steering, accelerator S:cameras, sonar, ..

Types:

- Fully observable : agents sensors give it access to the complete state of the enviroment at each point in time

Partially obs : due to noisy

No sensors : unobservable enviroment

- Deterministic : next state determied by current state and action executed by the agent

Strategic: det enviroment except for other agents enviroment

Stochastic: uncertainty on the next state, expressed w. probabilities

Non –det: no probabilities available

Uncertain: not fully observable + not deterministic

- Episodic: agent experience divided into atomic episodes(in each agent percives and performs a single action)

Sequential: a current decision may affect future decisions

- Static: env unchanged while an agent is deliberating(no worrying about time)

Dynamic: continuously asking the agent what it wants to do

- Discrete: finite number of states, percepts, actions
- Single agent: agent operating by itself in an env

Multi-agent: competitive (2 playing chess) vs cooperative (taxis avoid each other)

- Know: depends on the knowledge of the agent or the designer of the agent of the rules governing the enviroment

The real world is partially observable, stochastic, sequential, dynamic, continuos, multi-agent

#2

Lecture 3/4

Problem solving agents: is a particular type of goal-based agent. They use atomic representation of the enviroment(states w.out internal structure). The solution is a fixed sequence of actions

Intelligent agents are supposed to maximize thier performance measure (simplified if the agent can select a goal).

Problem formulation (for problem solving agents) : we know that if the enviroment is observable, know, deterministic --> a solution to a problem is a fixed sequence of actions

Search : process of looking for such a sequence

Search algorithm : input(problem), output (an action sequence)

(scheme slide)

This simple problem solving agent:

1. Formulates a goal and a problem
2. Searches for a sequence of actions that would solve the problem
3. Execute tha actions one at the time

When this is complete, it formulates another goal and starts over

A problem can be formally defined by:

- initial state that the agents starts in
- actions available to the agent
- transition model: retrun the state s reached by doing action a in state s'

  Problem state space = (initial state, actions, transition model)

  A **state space** is the set of all states reachable from the initial state by any sequence of actions

- Goal test: allows to check if a state is a goal

The state space can be depicted asd a directed graph: noeds=states, edges=actions, path=sequence of states connected by a sequence of actions

Path cost: numeric value associated to each path reflecting the desired performance measure, step cost is c(x,a,y) from state x to y by action a, >=0

A solution is a sequence of actions leading from the initial state to a goal state.

An optimal solution is a solution with minimal path-cost

Appliance to real world: real world is too complex--> state space must be abstracted for problem solving

So we have that: (abstract)state=set of real states, (abstract)action=complex combination of real actions, (abstract)solution=set of real paths that are solutions in the real world

Of course each abstractation should be easier that the original problem

Uniformed search algorithms: search algorithms consider possibile sequencecs of actions. Possible sequences of actions from initial state form a **search tree.**

search tree: root=initial state, nodes=states, braches=actions(outgoing edges from a node)

The same state can appear multiple time

Tree search algorithm:

Function tree-search(problem) returns a solution, or failure
Initialize the frontier using the initial state of problem
Loop do
    If the frontier is empty then return failure
    Choose a leaf node and remove it from the frontier
    If the node contains a goal state then return the correponding solution
    Expand the chosen node, adding the resulting nodes to the frontier

(frontier= the set of all leafs nodes available for an expansion at any given point)

Redundant path: when there si more than one way to get from one state to another. To avoid them the algorithm is augmented with explored set that remembers every expanded node.

Graph search algorithm:

Function graph search(problem) returns a solution, or failure

    Initialize the frontier using the initial state of problem

    Initialized the explored set to be empty

Loop do

    If the frontier is empty then return failure

    Choose a leaf node and remove it from the frontier

    If the node contains a goal state then return the correponding solution

    Add node to explored set

    Expand the choosen node, adding the resulting nodes to the frontier only if not in the frontier or explored set

#3/4

Lecture 5

Infrastructure for search algorithm: search algorithm require a data structure to keep track of the search tree. For each node of the tree, we have a structure w.:

- n.**STATE**(state –into state space- configuration in the world)
- N-**PARENT**(node that generate this node)
- n.**ACTION**(action that was applied to the parent to generate the node
- n.**PATH**_COST(cost, g(n9), of path from initial node to this node

Node=data stuct used to represent tree

Frontier: needs to be stored in such a way that search alg can easily choose th next node to exapnd. The appropiate data structure is a queue.

Queue op: EMPTY, POP,(remove first +return) INSERT(elem+return queue).

Type of queue: FIFO, LIFO, priority queue

Search strategy is defined by picking the order of the expansion, and evaluated along: completeness, time complexity, space complexity, optimality.

2nd and 3rd are measured in terms of branching factor, depth, maximum depth

Uniformed search strategies: are the whothat use only info available in the problem definiton, like: breadth first search, uniform cost search, depth first search, depth limited search, iterative deepening search. Lets see em all in details:

Breadth first search: expand shallowest unexpanded node. Impl: FIFO. Goal control in each generated node, no when selected for expansion.

Properties: n(node)finte, time n^d(depth)(space same), optimal(yes w. cost =1). Space is problem (n^d...)

Uniform cost search: expand least cost unexanded node. Impl: priority queue order by path cost. Goal test at expansion time. (if all cost=1 is like breadth...). Complete if cost>epsilon.

Time-space(copy)

Optimal(yes)

Depth first search: expande deepest unexpanded node. Impl: LIFO, successor at front.

Complete NO

Time O(b^m) b=brnching factor, m=max length of any path

Space: O(bm), linear

Optimal NO

Depth limit search: same as before w. depth limit l(node w. depth l no successor). It solves infinite path problems. This return a solution or failure(no sol exist, no sol in depth limit, but it exist..). incomplete/non optimal.

Iterative deepeing serch : it applies depth limit search w. increasing limit l. it terminates w. solution or failure(no sol exist)

Complete if b finite

Time O(b^d), space O(bd), optimal if step cost =1

[comparing these strateies, see above...]

#5

Lecture 6 informed search alg: use problem specific knowledge to speed up the search process.

[outline: Best fisrt search: greedy, A* search Heuristic]

Best first search: idea: use an evaluation function f(n) for each node (estimate of desiderability).

Special cases are: *Greedy* - BFirstS- and *A* search*

Most alg use a heuristc function h(n) as a component on f, h(n)=estimate of cost of the cheapest path from the state of the node n to a goal state. Assumptions are:h>0 and h=0 at every goal state.

[romania example]

Greedy version: expands the node that appears to be closet to goal. Properties:

Complete: NO(tree search). Would be complete with repeated state checking

Time O(b^m), good heuristic huge improvement.

Space O(b^m), all node in memory

Optimal: no

A* search : idea: avoid expanding paths that are already expensive. Evaluation function: f(n)=g(n)+h(n) , where g=path cost from start to n, h=estimated cost from n to goal, f=estimated solution cost.

But ah heuristic is always ammissible?NO, is ammissible if h(n)<=h*(n)(that is the true cost from n to goal). An ammissible one can't overstimate the cost to reach the goal!!!

[proof...]

An heuristic is consistent when:

If for every node n

for every successor n' of n geenrated by an action a,

h(n)<= c(n,a,n') + h(n'), where:

.h(n) si the estimated cost from node n to goal

.c(n,a,n') is the cost from node n to n' performing action a

.h(n)' is the cost from n' to goal

---->triangular inequality

If is consistent -->is ammissible

Th: is h is consistent -->A* GRAPH SEARCH is optimal.

Properties of A*: complete YES, time EXPONENTIAL, space ALL NODE MEMORY, optimal YES

[ex....]

There are many ways to define the heuristic, for istance:

In the 8 puzzle we can define:

H1=number of misplaced tiles

H2=total manhattan distance

Which is better???

If h2(n)>=h1(n) for all node n (both ammissible)--->then h2 dominates h1

So h2 is better than h1 for search (h1 will expand all the nodes h2 and possibly more)

[little comparasion of the 8puzzle example]

Generating admissible heuristic from relaxed problem: [relaxed problem: problem w. fewer restrictions on the actions]

A superset of actions are available at each state. The obtained graph is a supergraph

Any optimal solution of the original is also solution of the relaxed problem. Cost in optimal solution in relaxed problem is same or lower than optimal in original problem.

Admissible heuristic can be drived from the esact solution cost of a relaxed version of the problem

#6

Lecture 7

LECTURE 6 REVIEW(CHECK)

#7

Lecture 8 Local search algorithms

We saw in last lecture that the solution of a problem was the path. But there's also other type of algorithms.

In many optimization problem the path from start to end is irrelevant, the goal state is itself the solution.

State space=set of complete configurations

Landscape: A landscape has location(def by state) and elevation(def by value). The aim is to find: a global minimum(if elevatino=cost), or a global maximum (if elevation=obj function).

In such cases we can use local search algorithm, which keep a single current state and try to imporve it.

Principal local search alg: Hill climbing search, simulated annealing search, local beam search, genetic algorithms.

Hill climbing search: assume the elevation correspond to the obj function. At each step the current node is replaced w. the best neighbor

Function hill.climbing(problem) returns a state that is a local maximun

Current<--make-node(problem.initial-state)

Loop do

Neighbor<--a highest valued successor of current

If neighbor.value<=current.value then return current.state

Current<--neighbor

This method often gets stuck in local in local maxima or plateaux

*There are many variants of this algorithm : **stochastic, fist-choice, random-restart**.*

#8