Artificial & Computational Intelligence

AIML CLZG557

M1 : Introduction
&
M2 : Problem Solving Agent using Search

Indumathi V

Guest Faculty,

BITS - WILP

BITS Pilani

Pilani Campus

Course Plan

M1	Introduction to AI
M2	Problem Solving Agent using Search
M3	Game Playing
M4	Knowledge Representation using Logics
M5	Probabilistic Representation and Reasoning
M6	Reasoning over time, Reinforcement Learning
M7	Ethics in AI

Searching for Solutions

Choosing the current state, testing possible successor function, expanding current state to generate new state is called Traversal. Choice of which state to expand – Search Strategy

Basic

Greedy

variations of A* — IDA* / RBFS

① IDS

② optimality

Completeness

Time comp } BFS & DFS & UCS

Space "

③ Applications of BFS & DFS

④ Tree vs graph.

Uninformed Search Overview

Application

"Find the path"

Home → Chinese Restaurant

Depth First Search

- Find the Connectedness in a graph
- Topological Sorting

① Tree search based Alg → Retain

② Graph " — " " → Remembered

Me
Dev Sw
Jenita

Application

Graph

Depth First Search

- Find the Connectedness in a graph
- Topological Sorting

eg Maze game

Segment

DFS

Application

easiest Path

Graph

easiest path → no. of edges

Breadth First Search

BFS

- Finding path in a graph (many solutions)
- Finding the Bipartitions in a graph

Me to Mukesh

P_1 : M-S-M → 2

P_2 : M-D-S-M → 3

P_3 : M-D-J-M → 3

M

Bur SD

P P

B

Recommended

Bipartitions

ACI

Search Algorithm – Uninformed Example - 2

Worst case

DFS
O(b.d)-TC

BFS
O(b^d+1)

Can you guess which algorithm are these ?

(1)
(1 2) (1 3) (1 4)
(1 2 3) (1 2 4) (1 2 5) (1 3) (1 4)
(1 2 3 4) (1 2 3 5) (1 2 4) (1 2 5) (1 3) (1 4)
(1 2 3 4 5) (1 2 3 5) (1 2 4) (1 2 5) (1 3) (1 4)

C(1-2-3-4-5) = 70 + 50 + 100 + 50 =270
Expanded : 4
Generated : 10
Max Queue Length : 6

(1)
(1 2) (1 3) (1 4)
TEST FAILED

(1 3) (1 4) (1 2 3) (1 2 4) (1 2 5)
(1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5)
~~TEST PASSED~~

C(1-2-5) = 70 + 125 =195
Expanded : 4
Generated : 10
Max Queue Length : 6

DS O(n)

n=5

for(i=0; i<n; i++)
{
5<5
}

X: Primitive operation

'n' times

i=0 → Yes
1 → Y
2 → Y
3 → Y

4→Y
5

Search Algorithm – Uninformed Example - 2

DFS

branching factor — max, Avg

d	# nodes	
0	1	1
1	3 4	b
2	3 4	b
3	3	b
4	1 3	b

b^0
b^1
b^2
b^3
b^4

$\boxed{b=3}$

13

bF : 4

4

3×4

$b \cdot d$

$O(b \cdot d)$

time complexity

$b \cdot d$

Search Algorithm – Uninformed Example - 2

Space DFS

1

3

3

3

3

m → depth @ which DFS is going

d → depth @ which goal is founded.

$O(b * m + 1)$

BFS @ end of queue.

Search Algorithm – Uninformed Example - 2

open list	closed list	Goal
(1) 1	–	F(1)
(12) (13) (14) 3	(1)	F(12)
(13) (14) (123) (124) (125) 5	(1) (12)	F(13)
(14) (123) (124) (125) (134) (135) 6	(1) (12) (13)	F(14)
(123) (124) (125) (134) (135) (145) 6	(1) (12) (13) (14)	F(123)
(124) (125) (134) (135) (145) (1234) (1235) 7	(1) (12) (13) (14) (123)	F(124)
(125) (134) (135) (145) (1234) (1235) (1245) 7	(11) + (124)	P(125)

Max queue length : 7

Search Algorithm– Uninformed Example - 2

Max bF = 3

		#nodes	d
b^d	3^0	1	0
b^d	3^1	3	1
b^d	3^2	9	2
b^d	3^3	27	3
b^d	3^4		-

BFS

3 3 3 3 3 3 3 3 3 9x3

$b^d + b^d + b^d + b^d$

$O(b^{d+1})$ Time

Search Algorithm – Uninformed Example - 2

$O(b^{d+1})$

Space complexity

BFS + DFS ⇒ IDS

DFS → Expansion

DFS → Goal

Iterative Deepening Depth First Search (IDS)

for (i=0 ...)

Restart

Goal

Regeneration nodes

cost of redo < BFS ✓

Iterative Deepening Depth First Search (IDS)

Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	Closed List	Goal Test
1.	(1)		Fail on (1)
2.			

Sample Evaluation of the Algorithm -BFS

Complete – If the shallowest goal node is at a depth d, BFS will eventually find it by generating all shallower nodes

Optimal – Not necessarily. Optimal if path cost is non-decreasing function of depth of node. E.g., all actions have same cost

Time Complexity – $G(b^d)$ b - branching factor, d – depth

- Nodes expanded at depth 1 = b
- Nodes expanded at depth 2 = b^2
- Nodes expanded at depth d = b^d
- Goal test is applied during generation, time complexity would be $G(b^{d+1})$

Space Complexity – $G(b^d)$

- $G(b^{d-1})$ in explored set
- $G(b^d)$ in frontier set

BFS

Uniform Cost Search – Evaluation

Completeness – It is complete if the cost of every step > small +ve constant ∈

- It will stuck in infinite loop if there is a path with infinite sequence of zero cost actions

Optimal – It is Optimal. Whenever it selects a node, it is an optimal path to that node.

Time and Space complexity – Uniform cost search is guided by path costs not depth or branching factor.

- If C* is the cost of optimal solution and ∈ is the min. action cost
- Worst case complexity = $G(b^{1+\frac{C^*}{\epsilon}})$,
- When all action costs are equal → $G(b^{d+1})$, the BFS would perform better
  - As Goal test is applied during expansion, Uniform Cost search would do extra work

$b^{d+1} \rightarrow d \Rightarrow \frac{C^*}{\epsilon}$

Tree Search Vs Graph Search

Oradea
71
Zerind
151
75
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Drobeta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
98
Hirsova
85
Urziceni
86
Bucharest
90
Giurgiu
Eforie

Coding Aspects

For each node n of the tree,

- **n.STATE** : the state in the state space to which node corresponds
- **n.PARENT** : the node in the search tree that generated this node
- **n.ACTION** : the action that was applied to parent to generate the node
- **n.PATH-COST** : the cost, denoted by g(n), of the path from initial state to node

Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	Goal Test
1.	(1)	Fail on (1)
2.	(1 3), (1 4), (1 2)	Fail on (1 3)

NO Closed list

Tree Search Algorithms

```
function Tree-Search (problem, strategy) returns a solution, or failure
        initialize the search tree using the initial state of problems
        loop do
                if there are no candidate for expansion
                        then return failure
                choose: leaf node for expansion according to strategy
                if the node contains a goal state
                        then return the corresponding solution
                else
                        Expand the node
                        Add the resulting nodes to the search tree
        end
```

Tree Search Vs Graph Search Algorithms

Coding Aspects

Need:

Redundant Path Problem : More than one way to reach a state from another.

Infinite Loop Path Problem

Start : Arad

Goal : Craiova

Tree Search Vs Graph Search Algorithms

Coding Aspects

Need:

Redundant Path Problem

Infinite Loop Path Problem: Repeated State generated by looped path existence.

Start : Arad

Goal : Craiova

Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	**Closed List**	Goal Test
1.	(1)		Fail on (1)
2.	(1 3), (1 4), (1 2)	(1)	Fail on (1 3)

Search

Coding Aspects

For each node n of the tree,

n.STATE : the state in the state space to which node corresponds

n.PARENT : the node in the search tree that generated this node

n.ACTION : the action that was applied to parent to generate the node

n.PATH-COST : the cost, denoted by g(n), of the path from initial state to node

n.VISITED : the boolean indicating if the node is already visited and tested (**or)** a global SET of visited nodes

0

1

Tree Search Vs Graph Search Algorithms

Coding Aspects

Graph-Search Algorithm

Augments the Tree-Search algorithm to solve redundancy by keeping track of states that are already visited called as **Explored Set**. **Only one copy of each state is maintained/stored.**

Graph Search Algorithms

closed

[A, Z]

function **Graph-Search** (problem, fringe) returns a solution, or failure

initialize the search space using the initial state of problems memory to store the visited fringe

closed an empty set

Insert(Make-Node(Initial-State[problem]), fringe)

fringe

loop if fringe is empty

do then return failure

node Remove-Front(fringe)

if the node contains a goal state

then return the corresponding solution

else

if the node is not in closed ie., not visited yet

Add the node to the closed set

Expand all the fringe of the node

Add all expanded sorted successors into the fringe

end

Informed Search

- Greedy Best First
- A*

T

or

G →

Greedy Best First Search

n	h(n)
1	60
2	120
3	30
4	40
5	0

(1)
(1 3) (1 4) (1 2)
(1 3 5) (1 3 4)

C(1-3-5) = 100 + 125 =225
Expanded : 2
Generated : 6
Max Queue Length : 3

Case Study – 1 Search in Treebanks

Source Credit :

https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html

https://ufal.mff.cuni.cz/pdt3.5

Case Study – 1 Search in Treebanks

Source Credit :

https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html

https://ufal.mff.cuni.cz/pdt3.5

A* Search

given

n	h(n)
1	60
2	120
3	70
4	40
5	0

good ness

(1)
(1 4) (1 2) (1 3)
(1 4 5) (1 2) (1 3)

C(1-4-5) = 100 + 150 =150
Expanded : 2
Generated : 5
Max Queue Length : 3

A* Search

A* Search

actual pathcost → $h^*(L)$

AI_analyst	h(L)
L2	50 ✓
L3	30

L.H.S

R.H.S

$h_G(L_2) < h^*_G(L_2)$

less informed

overestimated

50

20 → more informed w.r.t environment

not accept overestimate

$h_G(L_3) < h^*_G(L_3)$

underestimated ← 30

100 → more

less

accept ✓

A* Search

<u>Optimal on condition</u>

h(n) must satisfies two conditions:

- Admissible Heuristic – one that never overestimates the cost to reach the goal
- Consistency – A heuristic is consistent if for every node n and every successor node n` of n generated by action a, h(n) <= c(n, a, n`) + h(n`)

<u>Complete</u>

- If the number of nodes with cost <=C* is finite
- If the branching factor is finite
- A* expands no nodes with f(n) > C*, known as pruning

<u>Time Complexity</u> - $G(b^{\Delta})$ where the absolute error $\Delta = h^* - h$

A* Search

1-3 → P_1 → 1-3 (125)
P_2 → 1-2-3 (120)

Is the heuristic designed leads to optimal solution?

Assuming **node 3** as goal, taking only sample edges per node below is checked for consistency

	Best Path	n	h(n)	Is Admissible? h(n) <= h*(n)	Is Consistent? For every arc (i,j): h(i) <= g(i,j) + h(j)
125	1-2-3	1	80	80 ≤ 125 T → Yes	
50	2-3	2	60	60 ≤ 50 F → NO	
0	0	3	0		
245	4-5-1-2-3	4	200	200 ≤ 245 T yes	
195	5-1-2-3	5	190	190 ≤ 195 T yes	

A* Search

Is the heuristic designed leads to optimal solution?

Assuming node 3 as goal, taking only sample edges per node below is checked for consistency

good

n	h(n)	Is Admissible? h(n) <= h*(n)	Is Consistent? For every arc (i,j): h(i) <= g(i,j) + h(j)
1	80	Y	N (5→1) : 190<=155
2	60 ~~55~~ 48	N	Y (1→2) : 80<=130
3	0	Y	
4	200	Y	Y (1→4) : 80<=300 Y (2→4) : 60<=275
5	190	Y	Y (2→5) : 60<=315 Y (4→5) : 200<=240

Path finding Robot – Sample Planning Agent Design

Successor Function Design

N-W-E-S

A*

Demo

Required Reading: AIMA - Chapter #1, 2, 3.1

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials