CSCE 580: Introduction to AI

Lecture 7: Search Continued

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

10TH SEP 2024

Carolinian Creed: "I will practice personal and academic integrity."

Credits: **Copyrights of all material reused acknowledged**

Organization of Lecture 7

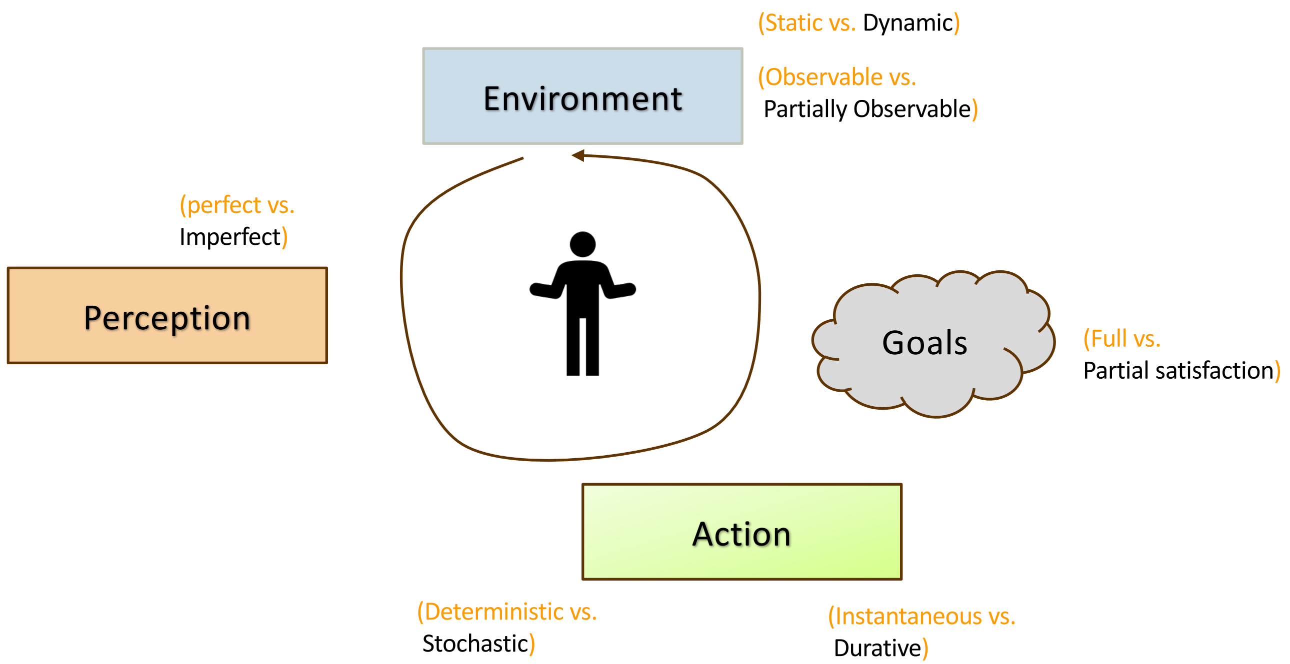
- Introduction Segment
  - Recap of Lecture 6
- Main Segment
  - Problems: vacuum, sliding tile, N-queens
  - Search – uninformed
  - Analyzing search performance
  - Informed search
  - Quiz 1
- Concluding Segment
  - Course Project Discussion
  - About Next Lecture – Lecture 8
  - Ask me anything

Introduction Section

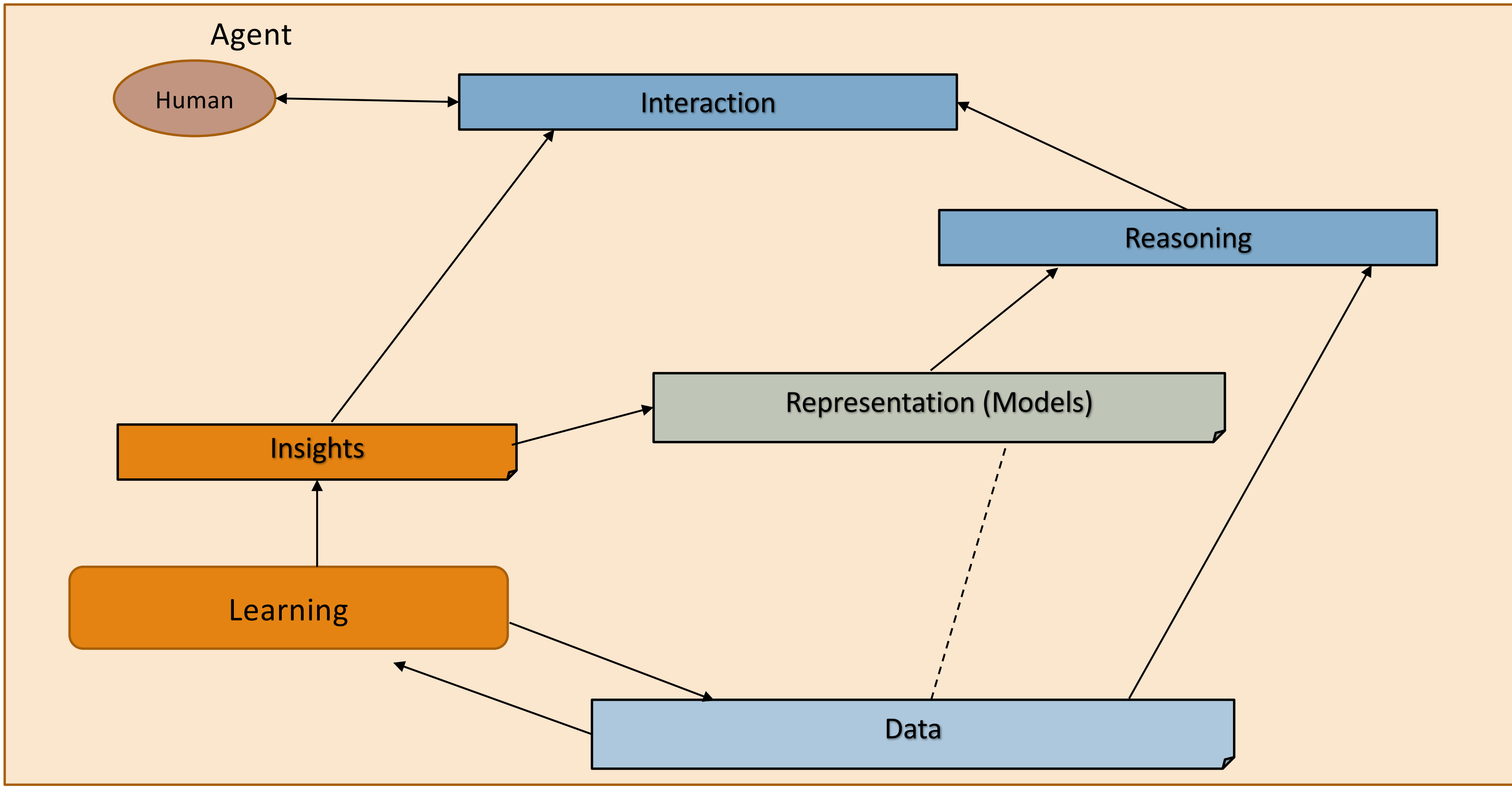
Recap of Lecture 6

- Problem solving agent – goal directed
- Problem formulation – abstraction, type of problems
- Search approach of problem solving

Intelligent Agent Model



Relationship Between Main AI Topics



Where We Are in the Course

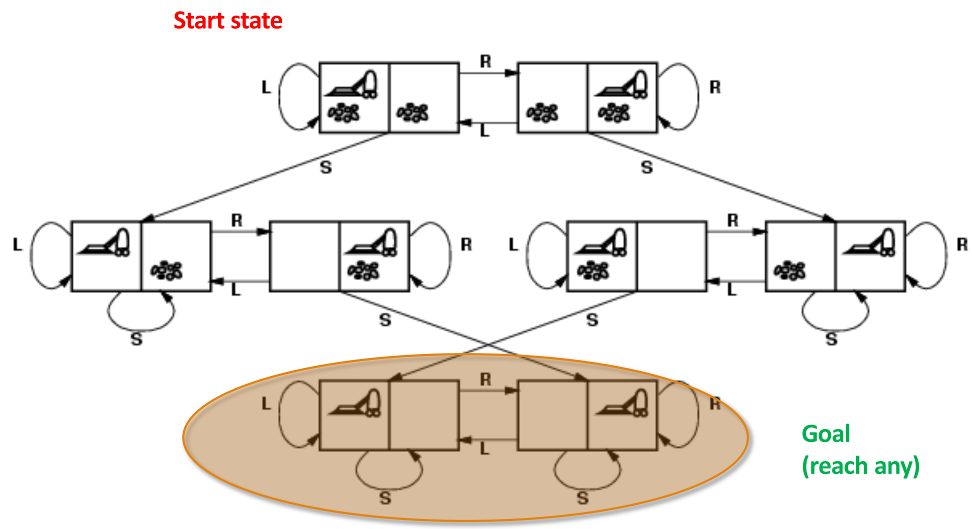
CSCE 580/ 581 – In This Course

- Week 1: Introduction, Aim: Chatbot / Intelligence Agent
- Weeks 2-3: Data: Formats, Representation and the Trust Problem
- Week 4-5: Search, Heuristics - Decision Making
- Week 6: Constraints, Optimization – Decision Making
- Week 7: Classical Machine Learning – Decision Making, Explanation
- Week 8: Machine Learning - Classification
- Week 9: Machine Learning - Classification – Trust Issues and Mitigation Methods
- Topic 10: Learning neural network, deep learning, Adversarial attacks
- Week 11: Large Language Models – Representation, Issues
- Topic 12: Markov Decision Processes, Hidden Markov models - Decision making
- Topic 13: Planning, Reinforcement Learning – Sequential decision making
- Week 14: AI for Real World: Tools, Emerging Standards and Laws; Safe AI/ Chatbots

Main Section

Example: Vacuum World

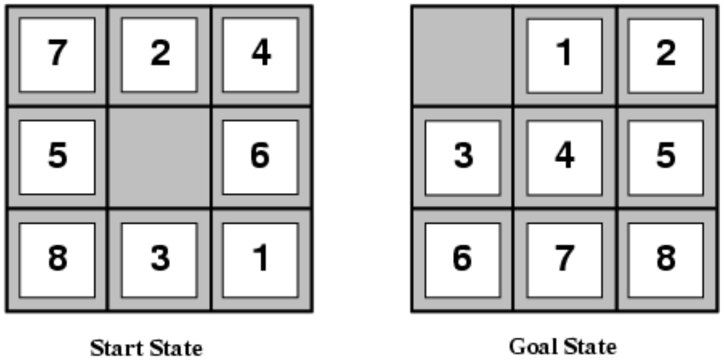
States	8 possible world states *(2room x 2dirt location x 2clean?)*
• Initial state • Goal state	• Any • No dirt at all locations
Actions • Transition model • Action cost	Left, Right, Suck • Action transition (edges) • 1



Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Example: Sliding 8-tile Puzzle

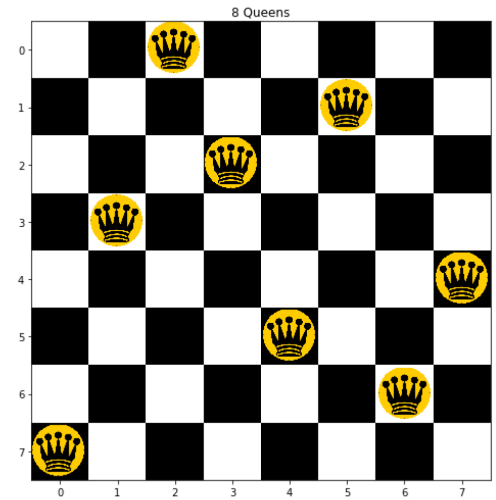
States • Initial state • Goal state	Location of tiles • Any (given) • All numbers sorted, Empty tile in corner (given)
Actions • Transition model • Action cost	move blank left, right, up, down • Blank transition (edges) • 1



Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Exercise: N-Queen Puzzle

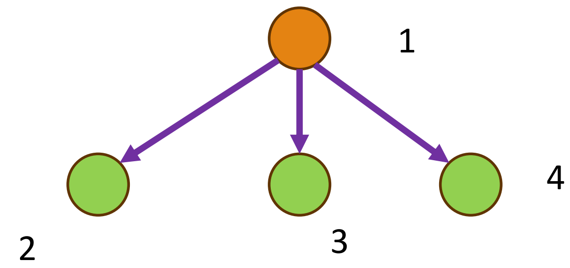
States • Initial state • Goal state	
Actions • Transition model • Action cost	



Adapted from:
Russell & Norvig, AI: A Modern Approach

Tree-search Algorithms

Basic idea: simulated exploration of state space by generating successors of already-explored states (a.k.a. ~ expanding states)



function TREE-SEARCH(*problem, strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree

Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

Uninformed Search Strategies

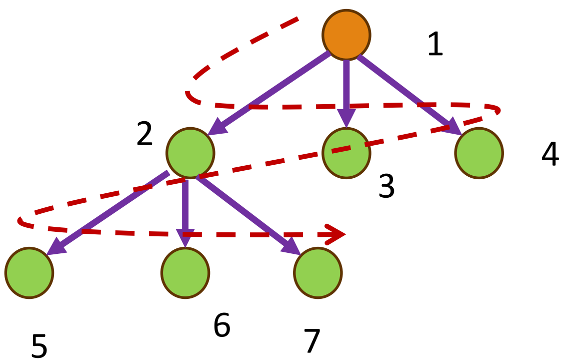
Search strategies use only the information available in the problem definition. They do not use a measure of distance to goal (*uninformed*).

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search
- Bidirectional search

Consideration: type of queue used for the fringe of the search tree
(collection of tree nodes that have been generated but not yet expanded)

Adapted from:
1. Russell & Norvig, AI: A Modern Approach
2. Bart Selman's CS 4700 Course

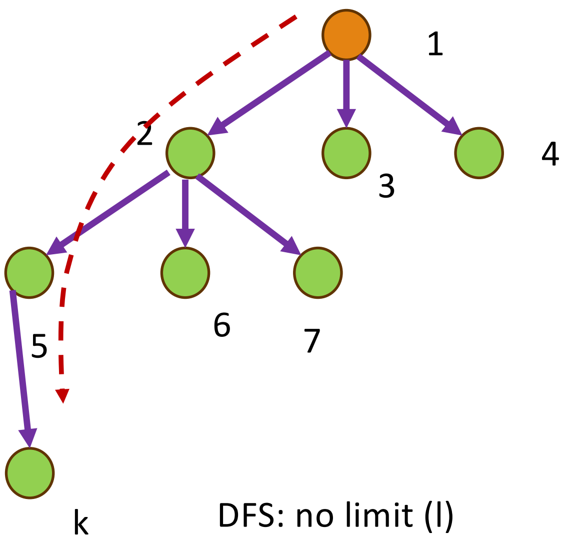
Breadth First Search (BFS)




```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Adapted from: Russell & Norvig, AI: A Modern Approach

Depth First Search (DFS) and Depth Limited Search (DLS)




```
function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

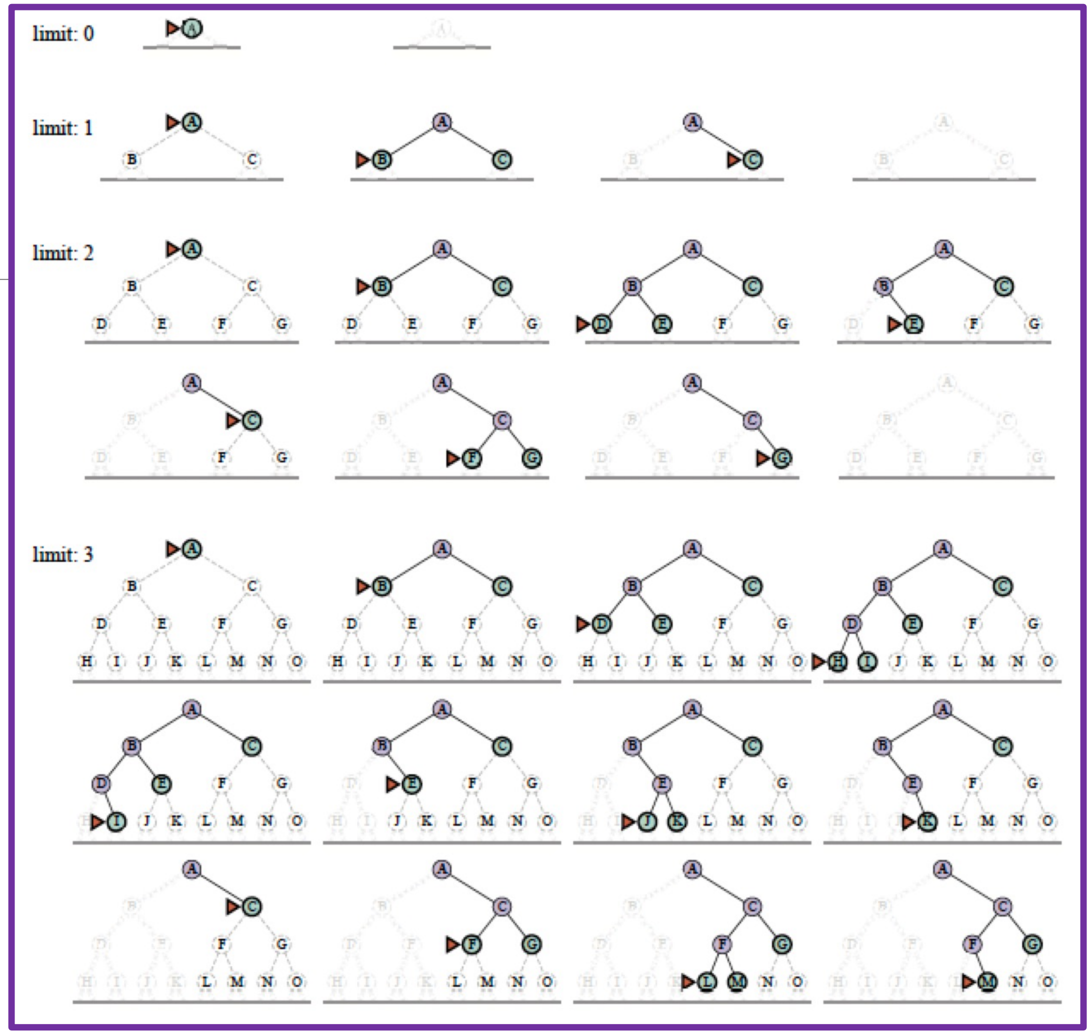
DFS: no limit (l)

Cutoff: when result is cutoff due to l

(result may be there if l increased)

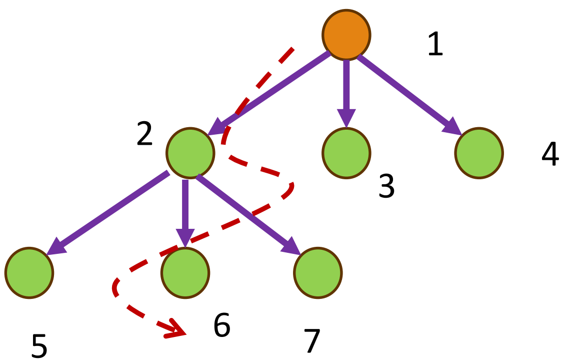
Adapted from: Russell & Norvig, AI: A Modern Approach

Illustration: DLS



Adapted from: Russell & Norvig, AI: A Modern Approach

Best-First Search

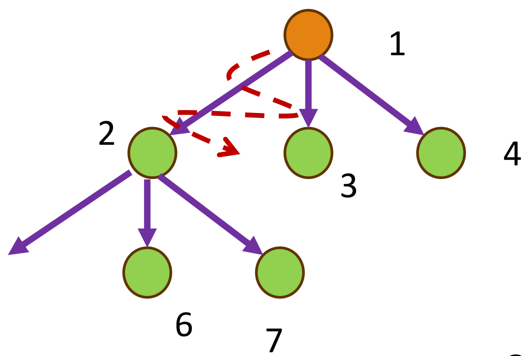


function BEST-FIRST-SEARCH(*problem*, *f*) **returns** a solution node or *failure*
  node ← NODE(STATE=*problem*.INITIAL)
  frontier ← a priority queue ordered by *f*, with *node* as an element
  reached ← a lookup table, with one entry with key *problem*.INITIAL and value *node*
  while not IS-EMPTY(*frontier*) **do**
    node ← POP(*frontier*)
    if *problem*.IS-GOAL(*node*.STATE) **then return** *node*
    for each *child* **in** EXPAND(*problem*, *node*) **do**
      s ← *child*.STATE
      if *s* is not in *reached* **or** *child*.PATH-COST < *reached*[*s*].PATH-COST **then**
        reached[*s*] ← *child*
        add *child* to *frontier*
  return *failure*

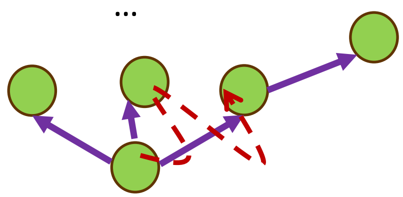
function EXPAND(*problem*, *node*) **yields** nodes
  s ← *node*.STATE
  for each *action* **in** *problem*.ACTIONS(*s*) **do**
    s′ ← *problem*.RESULT(*s*, *action*)
    cost ← *node*.PATH-COST + *problem*.ACTION-COST(*s*, *action*, *s′*)
    yield NODE(STATE=*s′*, PARENT=*node*, ACTION=*action*, PATH-COST=*cost*)

Source: Russell & Norvig, AI: A Modern Approach

Bi-Directional Search



…



Q: Why do this?
Q: When do this?

Adapted from: Russell & Norvig, AI: A Modern Approach

```
function BIBF-SEARCH(problem_F, f_F, problem_B, f_B) returns a solution node, or failure
  node_F ← NODE(problem_F.INITIAL)                       // Node for a start state
  node_B ← NODE(problem_B.INITIAL)                       // Node for a goal state
  frontier_F ← a priority queue ordered by f_F, with node_F as an element
  frontier_B ← a priority queue ordered by f_B, with node_B as an element
  reached_F ← a lookup table, with one key node_F.STATE and value node_F
  reached_B ← a lookup table, with one key node_B.STATE and value node_B
  solution ← failure
  while not TERMINATED(solution, frontier_F, frontier_B) do
    if f_F(TOP(frontier_F)) < f_B(TOP(frontier_B)) then
      solution ← PROCEED(F, problem_F frontier_F, reached_F, reached_B, solution)
    else solution ← PROCEED(B, problem_B, frontier_B, reached_B, reached_F, solution)
  return solution

function PROCEED(dir, problem, frontier, reached, reached_2, solution) returns a solution
        // Expand node on frontier; check against the other frontier in reached_2.
        // The variable "dir" is the direction: either F for forward or B for backward.
  node ← POP(frontier)
  for each child in EXPAND(problem, node) do
    s ← child.STATE
    if s not in reached or PATH-COST(child) < PATH-COST(reached[s]) then
      reached[s] ← child
      add child to frontier
      if s is in reached_2 then
        solution_2 ← JOIN-NODES(dir, child, reached_2[s]))
        if PATH-COST(solution_2) < PATH-COST(solution) then
          solution ← solution_2
  return solution
```

Figure 3.14 Bidirectional best-first search keeps two frontiers and two tables of reached states. When a path in one frontier reaches a state that was also reached in the other half of the search, the two paths are joined (by the function JOIN-NODES) to form a solution. The first solution we get is not guaranteed to be the best; the function TERMINATED determines when to stop looking for new solutions.

Analyzing Search Performance

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either has a solution or is finite. [2] complete if all action costs are $\geq \epsilon > 0$; [3] cost-optimal if action costs are all identical; [4] if both directions are breadth-first or uniform-cost.

Adapted from: Russell & Norvig, AI: A Modern Approach

Coding Example

- N-Queens – code notebook
  - https://github.com/biplav-s/course-ai-tai-f23/blob/main/sample-code/Class6-To-Class10-search.md

Informed Search – Greedy best-first

Uses domain/problem specific hints to guide search

f(n) = h(n)

- f: estimated cost of best path via n to goal
- h: estimated cost to goal from n   // h is also called heuristic function

Informed Search – A* search

Uses domain/problem specific hints to guide search

f(n) = g(n) + h(n)

- f: estimated cost of best path via n to goal
- g: cost of best path to n
- h: estimated cost to goal from n

Lecture 7: Summary

- We talked about
  - Goal-directed problem solving agents
  - How to formulate problem formulations
  - Search concepts
    - Problems of controlled robot navigation, 8-tile
  - Search strategies

Concluding Section

Course Project

Discussion: Projects

- New: two projects
  - Project 1: model assignment
  - Project 2: single problem/ llm based solving / fine-tuning/ presenting result

Project Discussion

1. Go to Google spreadsheet against your name
2. Enter model assignment name and link from (http://modelai.gettysburg.edu/)

1. Create a private Github repository called "CSCE58x-Fall2024-<studentname>-Repo". Share with Instructor (biplav-s) and TA (vishalpallagani)
2. Create Google folder called "CSCE58x-Fall2024-<studentname>-SharedInfo". Share with Instructor (prof.biplav@gmail.com) and TA (vishal.pallagani@gmail.com)
3. Create a Google doc in your Google repo called "Project Plan" and have the following by next class (Sep 5, 2024)

Timeline

1. Title:
2. Key idea: (2-3 lines)
3. Data need:
4. Methods:
5. Evaluation:
6. Milestones
    1. // Create your own
7. Oct 3, 2024

Reference: Project 1 Rubric (30% of Course)

Assume total for Project-1 as 100

- **Project results** – 60%
  - Working system ? – 30%
  - Evaluation with results superior to baseline? – 20%
  - Went through project tasks completely ? – 10%
- **Project effort**s – 40%
  - Project report – 20%
  - Project presentation (updates, final) – 20%
- **Bonus**
  - Challenge level of problem – 10%
  - Instructor discretion – 10%
- **Penalty**
  - Lack of timeliness as per your milestones policy (right) - up to 30%

Milestones and **Penalties**

- Project plan due by Sep 5, 2024 **[-10%]**
- Project deliverables due by Oct 3, 2024 **[-10%]**
- Project presentation on Oct 8, 2024 **[-10%]**

Discussion on Quiz 1

About Next Lecture – Lecture 8

Lecture 8: Searching for Problem Solving

- Informed Search
  - Heuristics for efficient search

- Class 9: Local search
- Class 10: Adversarial games and search