CSCE 590-1: From Data to Decisions with Open Data: A Practical Introduction to AI

Lecture 16: Reasoning and Optimal Decisions

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

9TH MAR, 2021

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

Organization of Lecture 16

- Introduction Segment
  - Quiz 2 marked and posted
  - Recap/ Discussion of Lecture 15

- Main Segment
  - Search continued
  - Planning
  - Optimal decisions
    - What is optimal
    - Methods

- Concluding Segment
  - About Next Lecture – Lecture 17
  - Ask me anything

Introduction Segment

Quiz 2 Marked and Posted

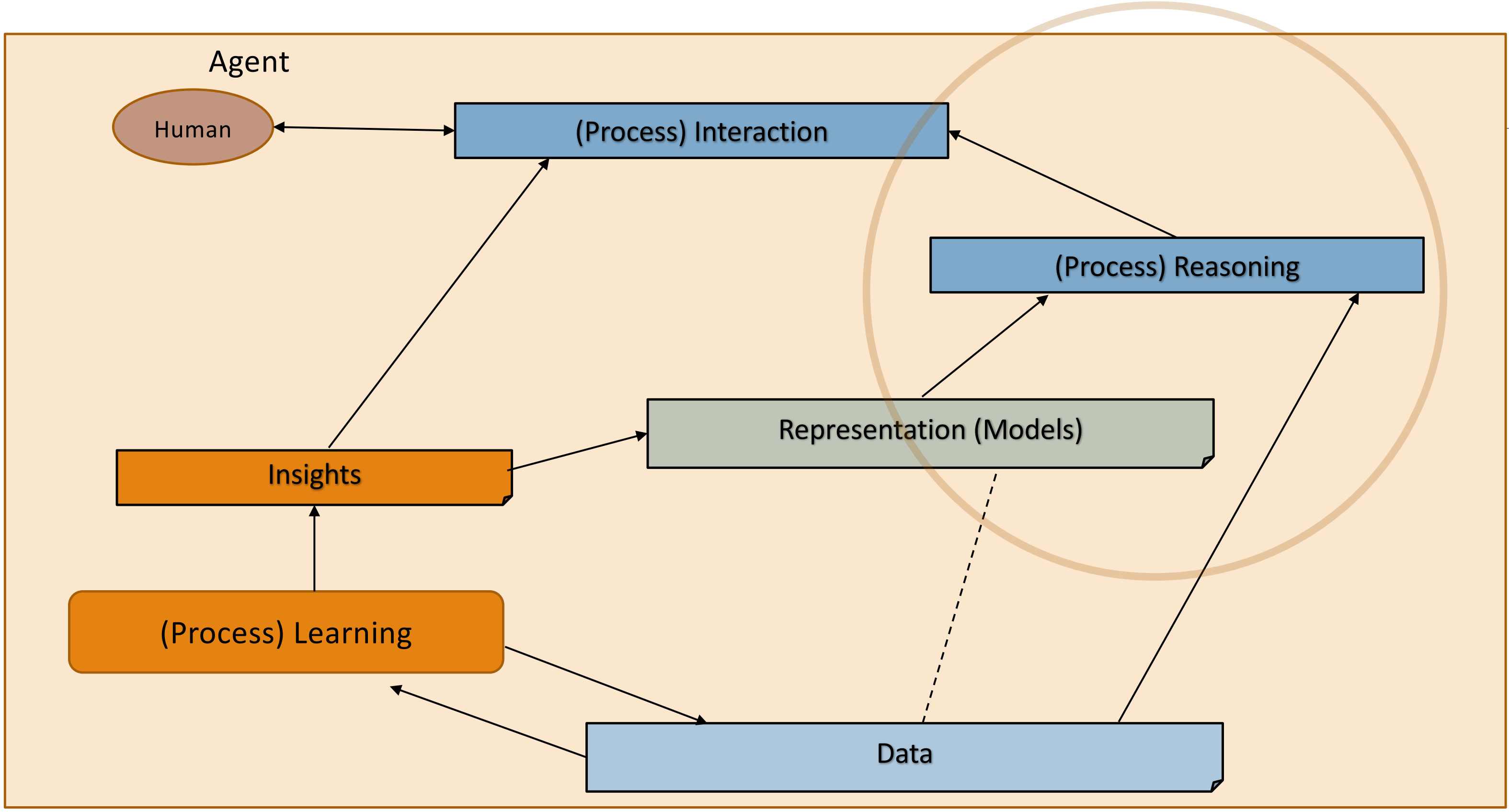
- Q1: classification - All got fine
- Q2: clustering – Needed to code (with Python); Weka did not provide V-scores
- Q3: bonus – a few tried

Recap of Lecture 15

- We revisited what it means to have an AI system as Intelligent Agent
- Understood knowledge representation and relationship to facts in the world
- Discussed logic procedures to draw inferences
- Discussed search methods to solve problems
- Saw a planning problem – generating sequence of actions to meet a goal

** Spillover topics*

Relationship Between AI Processes



Example Situation – Course Selection

- A person wants to pass an academic program in two majors: A and B
- There are three subjects: A, B and C, each with three levels (*1, *2, *3). There are thus 9 courses: A1, A2, A3, B1, B2, B3, C1, C2, C3
- To graduate, at least one course at beginner (*1) level is needed in major(s) of choice(s), and two courses at intermediate levels (*2) are needed
- Answer questions
  - Q1: How many <u>minimum</u> courses does the person have to take ?
  - Q2: Can a person graduate in 2 majors studying 3 courses only?
  - …

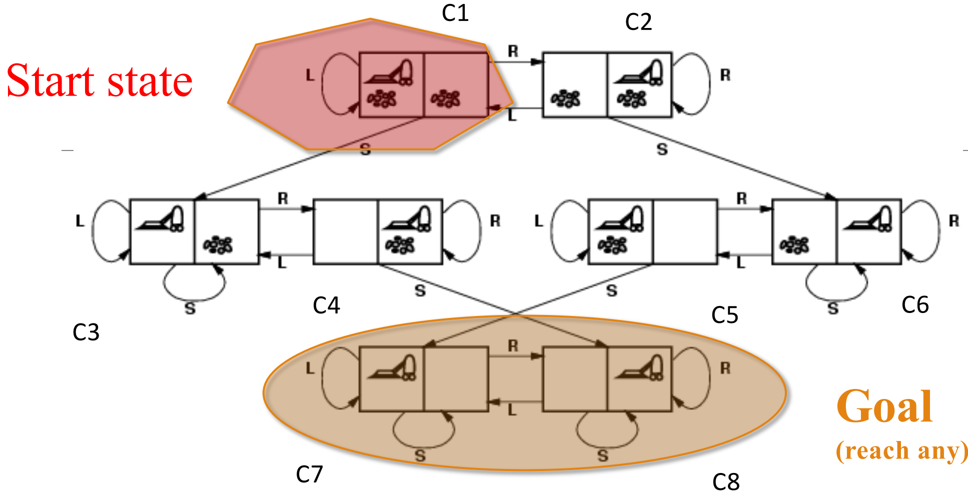
Exercise and Code

- Solving “Course” Example in Propositional Logic
  - Code: https://github.com/biplav-s/course-d2d-ai/blob/main/sample-code/l15-l16-l17-l18-agents/l15-logic.ipynb
  - Adapted from code of Book: AI – A Modern Approach

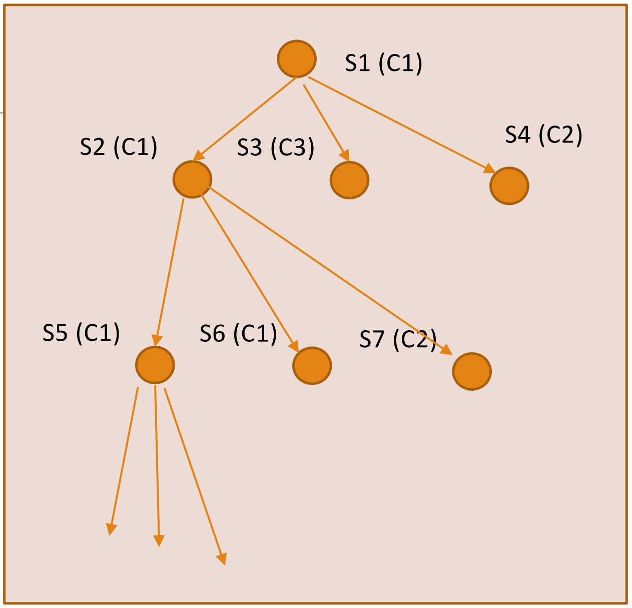
Acknowledgement: Russell & Norvig, AI: A Modern Approach

Search Techniques

Example: Vacuum World State Space Graph



Example Search Tree



States: The agent is in one of 8 possible world states.

Actions: *Left, Right, Suck [simplified: left out No-op]*

goal test: No dirt at all locations

path cost: 1 per action

Minimum path from Start to Goal state: **3 actions**

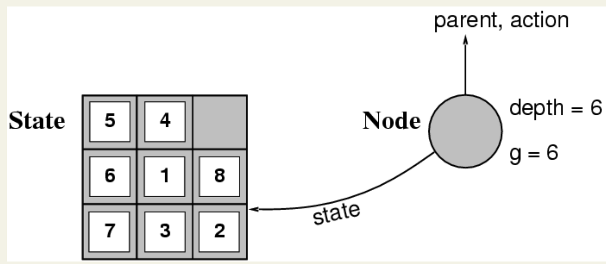
Alternative, longer plan: **4 actions**

Adapted from: Bart Selman's CS 4700 Course

Search Concepts: States and Nodes

A state is a representation of a physical configuration.

A node is a data structure constituting part of a search tree includes state, tree parent node, action (applied to parent), path cost (initial state to node) *g(x)*, depth



The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Fringe is the collection of nodes that have been generated but not (yet) expanded. Each node of the fringe is a leaf node.

Source: Bart Selman's CS 4700 Course

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
```

Source: Bart Selman's CS 4700 Course

Implementation: General Tree Search

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
   successors ← the empty set
   for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
   return successors
```

A search strategy is defined by picking the order of node expansion.

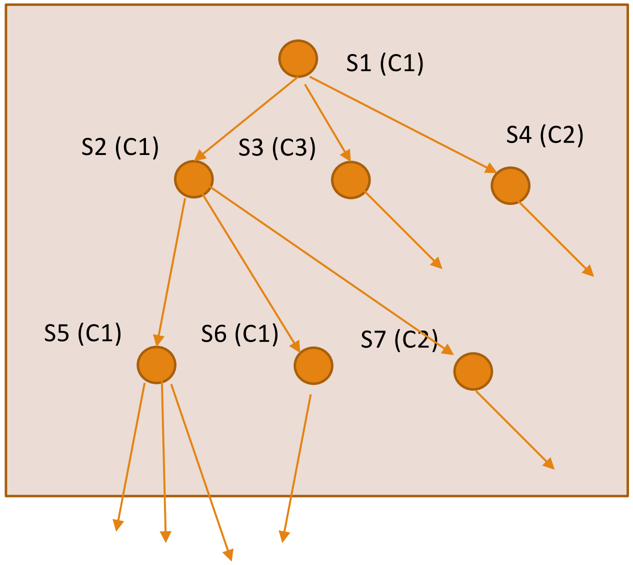
Uninformed Search Strategies

Uninformed (blind) search strategies use only the information available in the problem definition:

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

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

Example Search Tree



Source: Bart Selman's CS 4700 Course

Search Strategies

- A search strategy is defined by picking the **order of node expansion.**

- Strategies are evaluated along the following dimensions:
  - **completeness**: does it always find a solution if one exists?
  - **time complexity**: number of nodes generated
  - **space complexity**: maximum number of nodes in memory
  - **optimality**: does it always find a least-cost solution?

- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

Source: Bart Selman's CS 4700 Course

Comparison of Uninformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l > d$	Yes	Yes

Figure 3.18 Evaluation of search strategies. b is the branching factor; d is the depth of solution; m is the maximum depth of the search tree; l is the depth limit.

Source: Russell & Norvig, AI: A Modern Approach

Exercise and Code

- Search Methods
  - Search algos: https://github.com/biplav-s/course-d2d-ai/blob/main/sample-code/l15-l16-l17-l18-agents/l16-search.ipynb
  - Original code from Book: AI – A Modern Approach, https://github.com/aimacode/aima-python/blob/master/search.ipynb

Acknowledgement: Russell & Norvig, AI: A Modern Approach

Goal-Based Agents
Generating Sequence of Actions

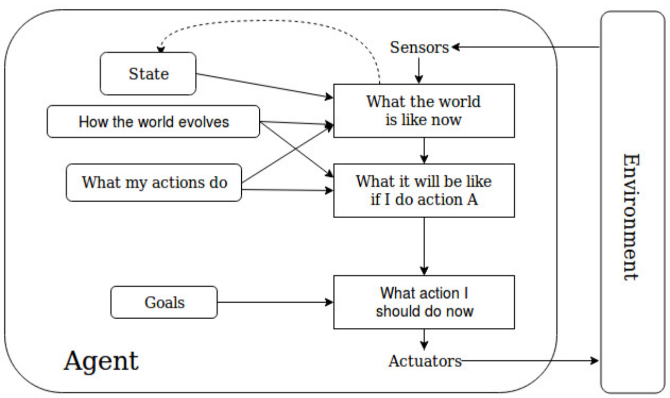
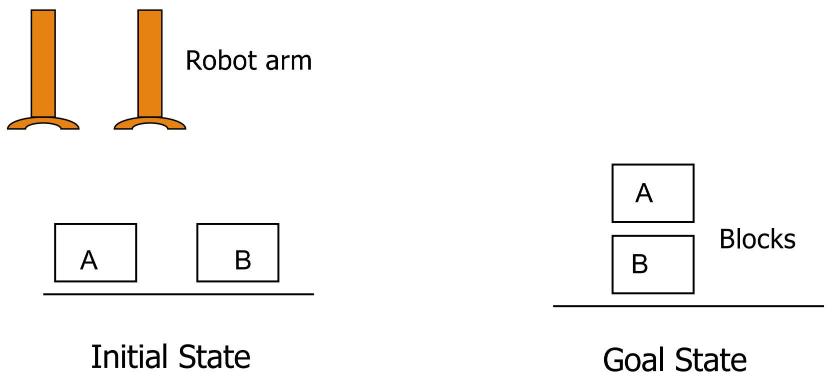


Figure Source: Russell & Norvig, AI: A Modern Approach

Reasoning Illustration - Planning Example

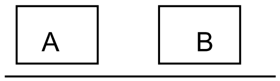
Blocks World



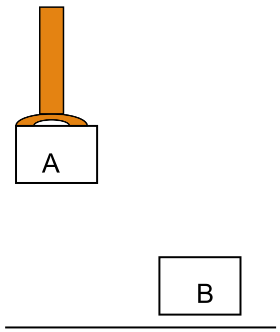
All robots are equivalent

Reasoning Illustration - Representation

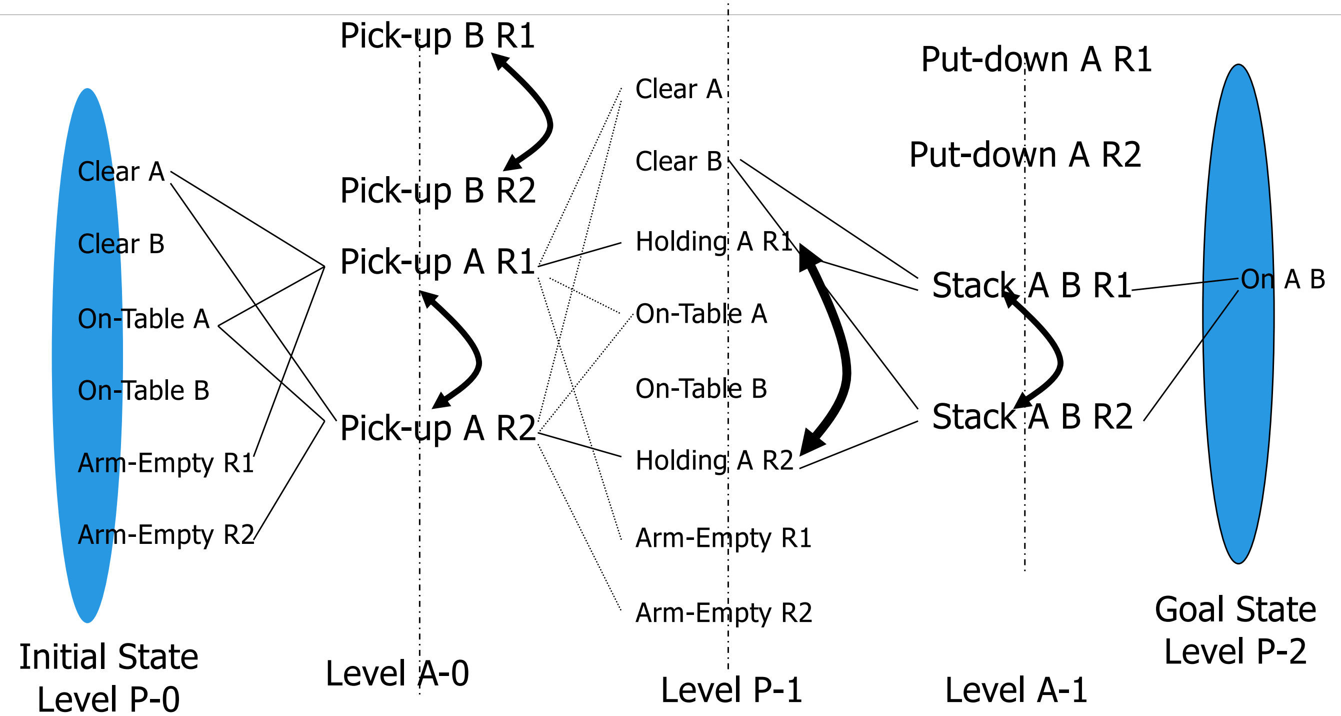
States: ((On-Table A) (On-Table B) …)



Actions: ((Name: (Pickup ?block ?robot)
Precondition: ((Clear ?block)
(Arm-Empty ?robot)
(On-Table ?block))
Add: ((Holding ?block ?robot))
Delete: ((Clear ?block)
(Arm-Empty ?robot)))…)



Reasoning Illustration - Planning Process



Active Area of Research

Considerations

- What to find:
  - Any workable plan
  - Optimal plan – but then what is the criteria
  - All plans
  - Diverse plans
- How to find
  - Plan at the end
  - Plan anytime
- How to represent problem
- How to explain solution

Making Optimal Decisions

Optimal Decision

- What is it? There is no absolute answer. In AI, there is the concept of a **rational** agent.
- Acting rationally: acting such that one ca achieve one’s goals given one’s beliefs (and information)
  - But what are one’s goals
  - Are the always of achievement?
- Some options
  - Perfect rationality: maximize expected utility at every time instant
    - Given the available information; can be computationally expensive
    - “Doing the right thing”
  - Bounded optimality: do as well as possible given computational resources
    - Expected utility as high as any other agent with similar resources
  - Calculative rationality: *eventually* returns what would have been the rational choice

What Is It?

- As a working principle
  - Bounded or Calculative Rationality
- In observable and deterministic scenarios
  - Maximize utility: (benefit – cost)
- In scenarios with uncertainty and/ or unobservable
  - Maximize ***expected*** utility: (benefit – cost)

Example Situation – Course Selection

- A person wants to pass an academic program in two majors: A and B
- There are three subjects: A, B and C, each with three levels (*1, *2, *3). There are thus 9 courses: A1, A2, A3, B1, B2, B3, C1, C2, C3
- To graduate, at least one course at beginner (*1) level is needed in major(s) of choice(s), and two courses at intermediate levels (*2) are needed
- **Optimality considerations** in the problem
  - Least courses, fastest time to graduate, class size, friends attending together, …
- Answer questions
  - Q1: How many <u>minimum</u> courses does the person have to take ?
  - Q2: Can a person graduate in 2 majors studying 3 courses only?
  - …

Algorithms for Optimality

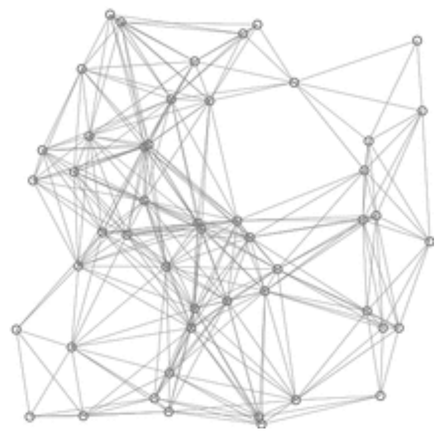
- Problem specific methods
  - Path finding
  - Linear programming
  - Constraint satisfaction and optimization

- General Purposed - methods for optimality in search

Optimality: Example - Path Finding

Main steps

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances through the current node. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one.
4. When we are done considering all of the unvisited neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the *unvisited set* is infinity, then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.



A demo of Dijkstra's algorithm based on Euclidean distance

Djikstra's Algorithm with positive numbers or labels that are monotonically non-decreasing.

Source: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Exercise and Code

- Linear Programming Methods

  - Link - https://github.com/biplav-s/course-d2d-ai/blob/main/sample-code/l16-optimal/Optimization.ipynb

A* Search

- Cheapest cost (f), via a search node (n), to reach the goal is represented as:
  - $f(n) = g(n) + h(n)$
  - Where: g is the cost to arrive at node (n) and h is the ***estimate*** of the cost to reach goal
- h is called a **heuristic** function
- When h is a lower-bound to the actual cost h*, the heuristic is called **admissible**

Properties of A* Search

- **Properties**
  - Completeness: Will find a solution if one exists
    - Expand node in the order of increasing f-cost. As long as the branching factor at each node is finite, a solution, if it exists, will be found.
  - Optimal: Will provably find the lowest cost of solution ***if the heuristic function is admissible***
    - f(p) be the cost of parent and f(c_i) be the cost of a child c_i of p.
    - If h is admissible, f(c_i) never decreases compared to f(p). If it did, that means there was a shorter path to go to goal via child than the parent since heuristic is a lower bound. But since the only way to reach child is via parent, we can use parent's estimate and ensure the estimate is non-decreasing.

- **Problem with A***
  - Memory – need to store nodes since we do not know if a future node may have a lower f-cost

Exercise and Code

- Search Methods
  - Search algos: https://github.com/biplav-s/course-d2d-ai/blob/main/sample-code/l15-l16-l17-l18-agents/l16-search.ipynb
  - Original code from Book: AI – A Modern Approach, https://github.com/aimacode/aima-python/blob/master/search.ipynb

Acknowledgement: Russell & Norvig, AI: A Modern Approach

Lecture 16: Concluding Comments

- Continued discussion on Search
- Sequential decision making: Planning
- Explored optimal decisions
  - Domain-specific optimal methods
    - Path finding
    - Linear programming
  - A* for general search

Concluding Segment

Adjustment to Upcoming Schedule

15	**Mar 4 (Th)**	**Reasoning and Search**	**Semester - Midpoint**
16	Mar 9 (Tu)	Agent – Optimization	
17	Mar 11 (Th)	Agent – Handling Uncertain World	
18	Mar 16 (Tu)	Agent – Learning	
19	Mar 18 (Th)	Text: Data Prep (NLP)	Quiz 3
20	Mar 23 (Tu)	Text: Analysis - Supervised (NLP)_	
21	Mar 25 (Th)	Review, Paper presentations, Discussion	
22	Mar 30 (Tu)	Text: Advanced – Summarization, Sentiment	
23	Apr 1 (Th)	Text: Visualization, Explanation	
24	Apr 6 (Tu)	Multimodal Agents: Structured+Text: Examples	
25	Apr 8 (Th)	Case Study 1: Water	Quiz 4
26	Apr 13 (Tu)	Case Study 2: Finance	

Focus on Integrated Agent Behavior (Lectures 17, 18)

Reduce focus on Case-studies (1 per domain)

About Next Lecture – Lecture 17

Lecture 17: Reasoning, Agents Continued

- Reasoning – advanced settings
  - Agents Making Decisions Under Uncertainty
- Kind of uncertainties
- What is the best decision possible: Maximize Expectation
- Some methods