CSCE 580: Introduction to AI

Lecture 9: Search Continued – Local Search

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

17TH SEP 2024

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

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

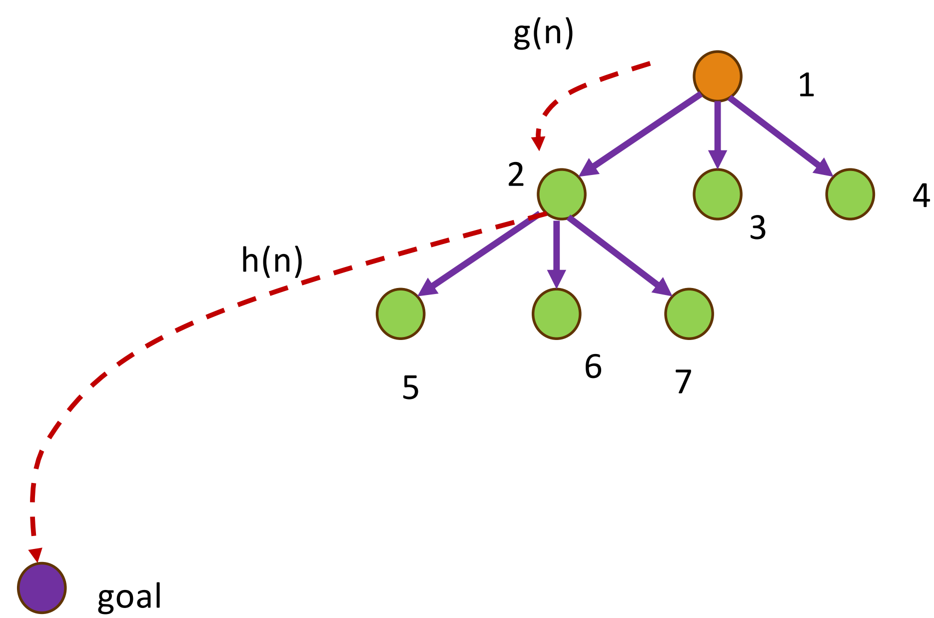
Organization of Lecture 9

- Introduction Segment
  - Recap of Lecture 8
- Main Segment
  - Hill climbing
  - Simulated Annealing
  - Genetic programming
  - Search in complex environments
- Concluding Segment
  - Course Project Discussion
  - About Next Lecture – Lecture 9
  - Ask me anything

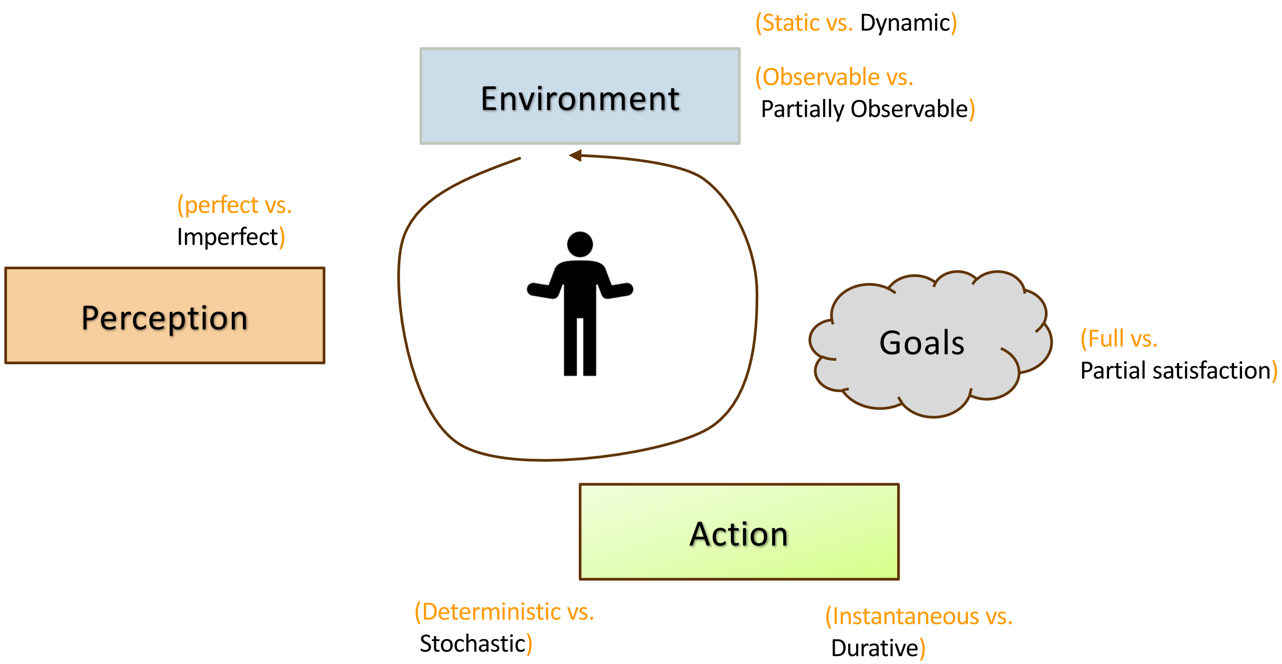
Introduction Section

Recap of Lecture 8

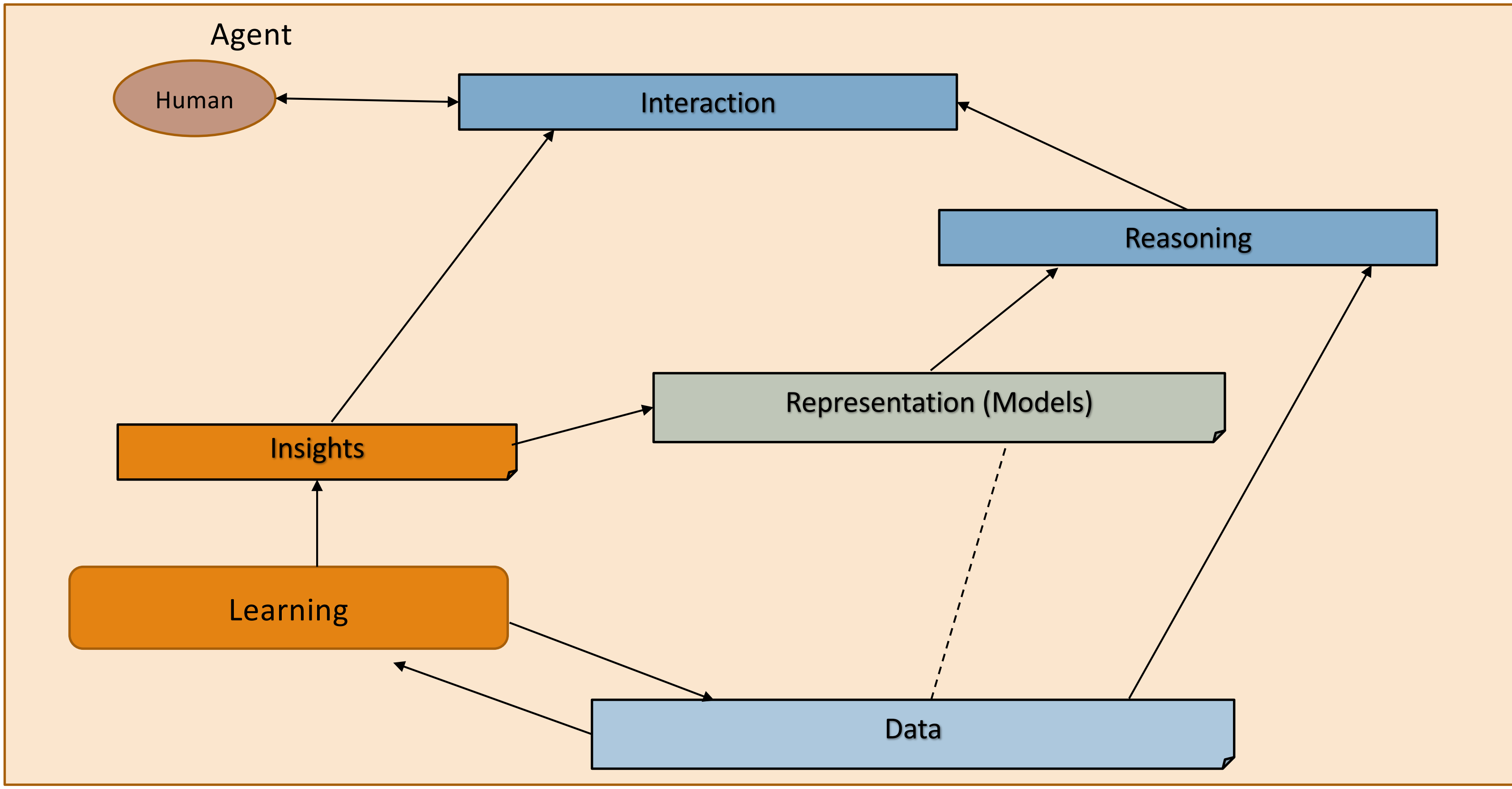
- Informed Search
- Heuristics and Properties
- Designing Heuristics



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

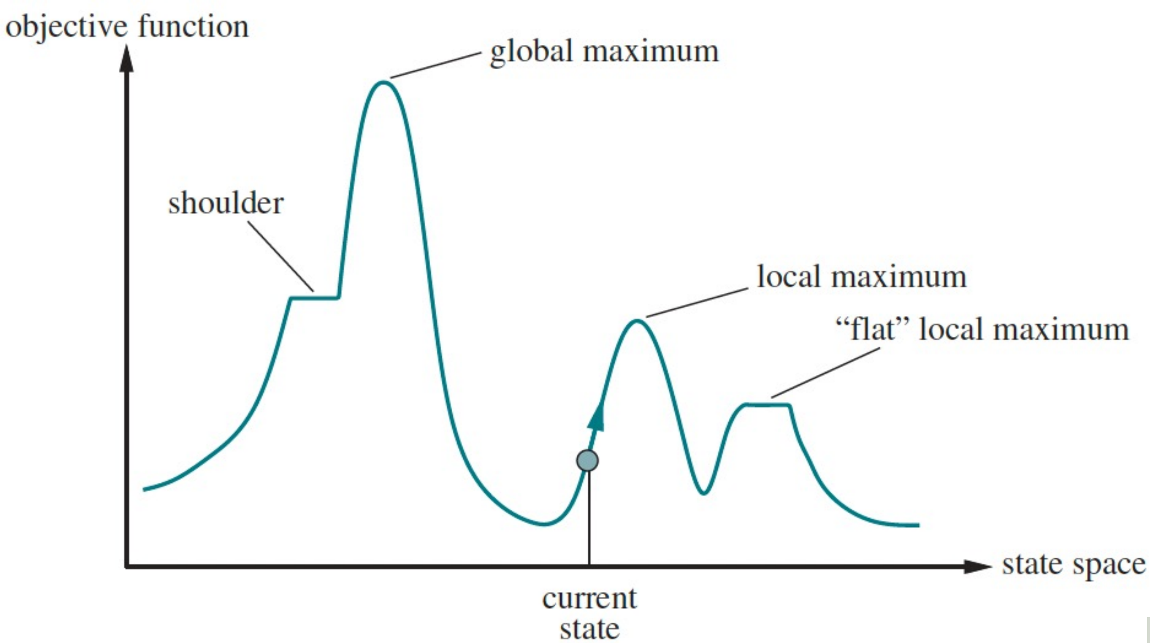
Main Section

Local Search

- Systematic search
  - Path matters [Store search trajectory]
- Non-systematic search
  - Solution matters, not path

- Settings
  - States: Discrete, continuous
  - Non-deterministic actions
  - Partial observability

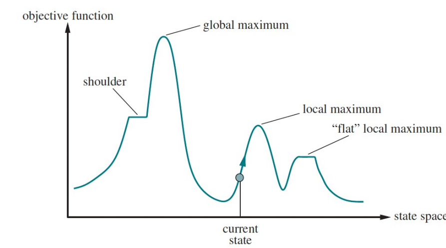
State Space Landscape



- Setup: find maxima

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

Hill Climbing /Greedy Local Search

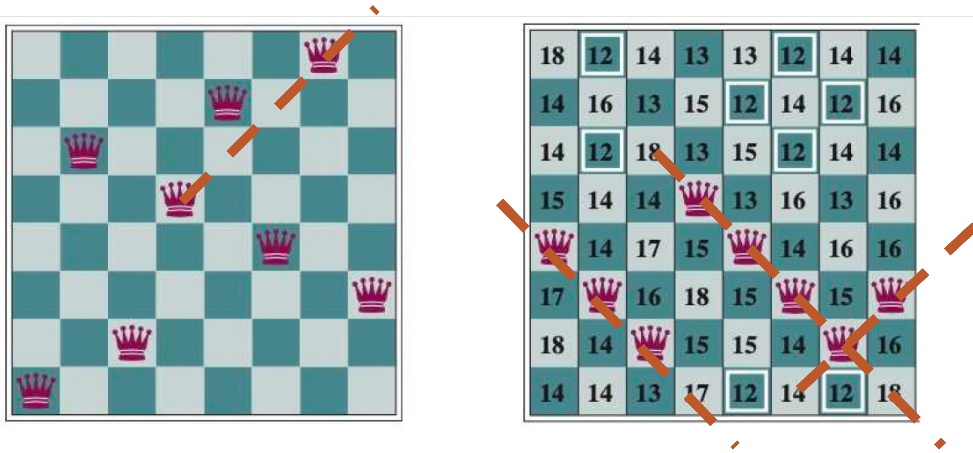


function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  current ← *problem*.INITIAL
  while *true* **do**
    neighbor ← a highest-valued successor state of *current*
    if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
    current ← *neighbor*

At each step, replace the current node with the **best** neighbor.

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

Hill Climbing Illustration



(a) (b)

Figure 4.3 (a) The 8-queens problem: place 8 queens on a chess board so that no queen attacks another. (A queen attacks any piece in the same row, column, or diagonal.) This position is almost a solution, except for the two queens in the fourth and seventh columns that attack each other along the diagonal. (b) An 8-queens state with heuristic cost estimate $h = 17$. The board shows the value of h for each possible successor obtained by moving a queen within its column. There are 8 moves that are tied for best, with $h = 12$. The hill-climbing algorithm will pick one of these.

State representation:

- Complete state formulation

Next Action:

- Any queen in the same column (8 x 7 = 56 children)

State space: 8^8 = 17 million (appx)

Steepest ascent:

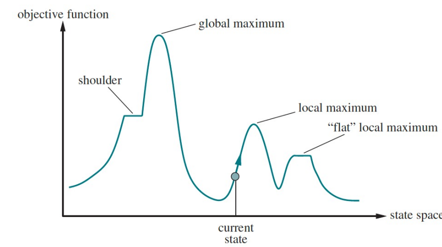
* Gets stuck 86% times in 3 steps
* Solves 14% times in 4 steps

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

Hill Climbing Variations

- **Stochastic hill climbing**: chooses an uphill node with prob. depending on steepness of increase
- **First-choice hill climbing**: choose first that is uphill
- **Random-restart hill climbing**: restart after a few tries
  - If p is chance of success. restarts needed = 1/p
  - For 8-queens, p=.14
    - Restart needed = 7 (6 failure, 1 success)
    - Total steps for finding a solution = 4 +((1-p) / p) * 3 = 22 steps

Simulated Annealing



function SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  $current \leftarrow problem$.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow$ VALUE($current$) – VALUE($next$)
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

- Setup: find minima
- T: temperature
- A bad successor is chosen will prob. that decreases with temperature
- Schedule: cooling schedule

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

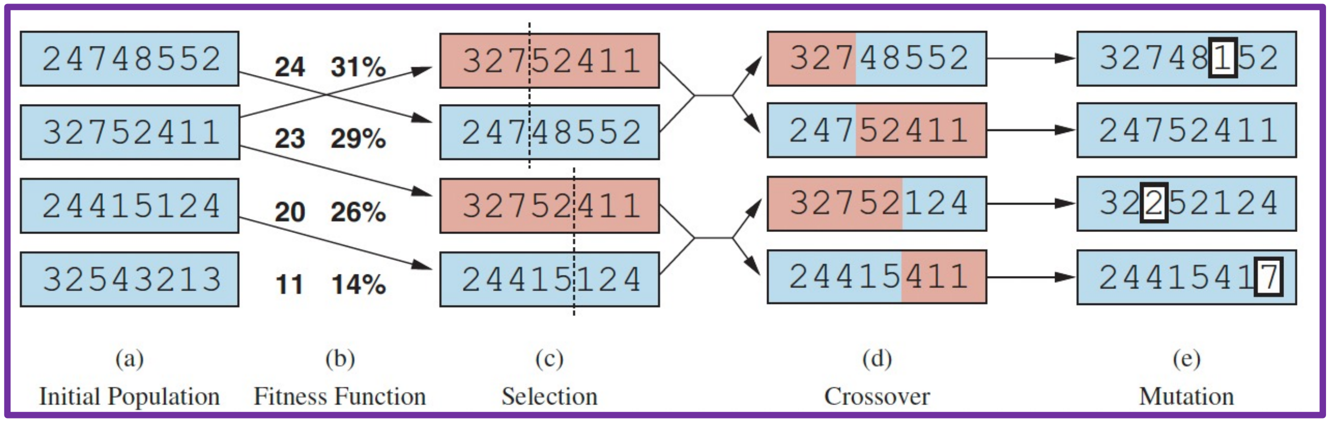
Related Algorithms

- **Local beam search**: keeps track of **k** states rather than 1
  - Generate k randomly generated states
  - Repeat
    - Generate all successors of k states generated
    - If one is a goal, done
    - Select k best successors
- Stochastic beam search
  - Chooses k successors with probability proportional to the successors value

Evolutionary Algorithms (EAs)

Basic idea

- A population of individuals (states)
- Fittest (highest value) produce offsprings (successor states) - recombination
  - Cross-over
  - mutation



Digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Fitness function: non-attacking pairs of queens

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

Comparing EA with Local Search

- Idea of cross-over
  - Useful if traits of parents are useful in children
- Idea of mutation
  - Random changes can help escape local minima
- Selection of parameters (e.g., generations) affects performance
- # Parents
  - =1 : stochastic beam search
  - =2 : similar to nature
  - > 2 : not common in nature, but possible to simulate

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
     weights ← WEIGHTED-BY(population, fitness)
     population2 ← empty list
     for i = 1 to SIZE(population) do
        parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
        child ← REPRODUCE(parent1, parent2)
        if (small random probability) then child ← MUTATE(child)
        add child to population2
     population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Figure 4.7 A genetic algorithm. Within the function, *population* is an ordered list of individuals, *weights* is a list of corresponding fitness values for each individual, and *fitness* is a function to compute these values.

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

Local Search With Non-Deterministic Actions

- Systematic search
  - Path matters [Store search trajectory]
- Non-systematic search
  - Solution matters, not path
- Settings
  - States: Discrete, continuous
  - **Non-deterministic actions***
  - Partial observability*

Erratic Vacuum World

- When applied to a dirty square, the robot cleans that room and sometimes the adjacent room
- When applied to a clean square, the robot throws dirt in the room

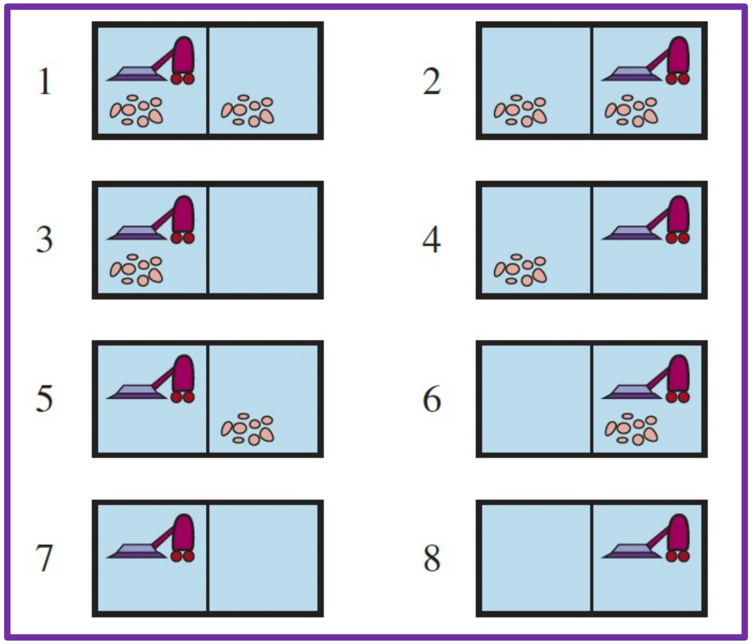
Adapted from:
Russell & Norvig, AI: A Modern Approach

* Solutions are not nodes but conditional plans/ strategies.

Local Search With Non-Deterministic Actions

Erratic Vacuum World

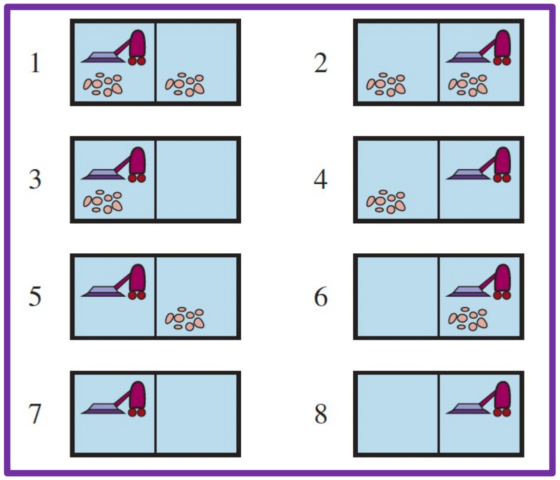
- When applied to a dirty square, the robot cleans that room and sometimes the adjacent room
- When applied to a clean square, the robot throws dirt in the room

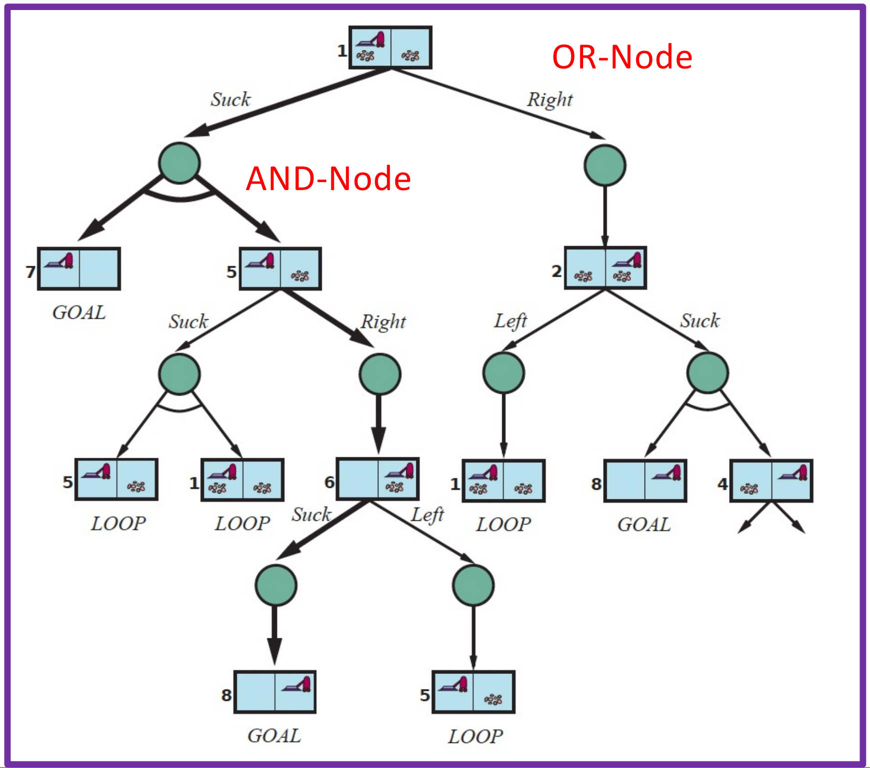


Adapted from:
Russell & Norvig, AI: A Modern Approach

Local Search With Non-Deterministic Actions

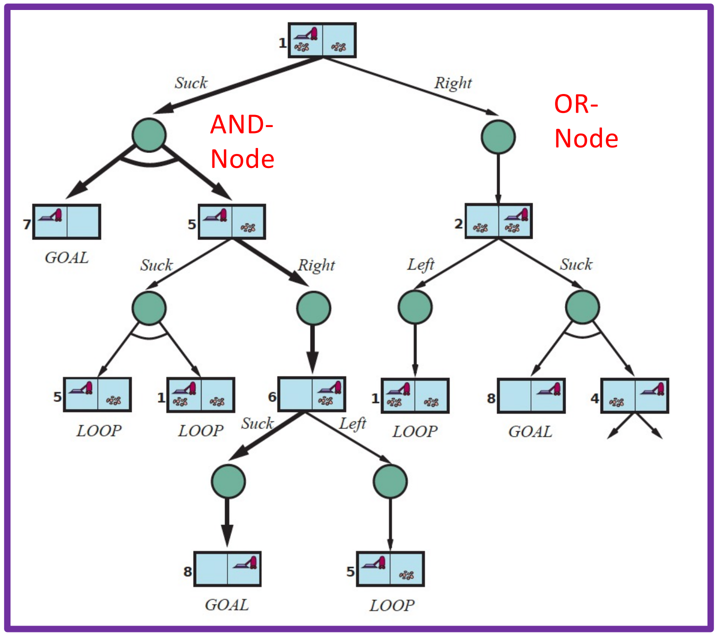
Erratic Vacuum World
- When applied to a dirty square, the robot cleans that room and sometimes the adjacent room
- When applied to a clean square, the robot throws dirt in the room





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

Local Search With Non-Deterministic Actions



function AND-OR-SEARCH($problem$) **returns** a conditional plan, or $failure$
  return OR-SEARCH($problem$, $problem$.INITIAL, [ ])

function OR-SEARCH($problem, state, path$) **returns** $a\ conditional\ plan,\ or\ failure$
  if $problem$.IS-GOAL($state$) **then return** the empty plan
  if IS-CYCLE($path$) **then return** $failure$
  for each $action$ **in** $problem$.ACTIONS($state$) **do**
    $plan \leftarrow$ AND-SEARCH($problem$, RESULTS($state, action$), $[state] + path$])
    if $plan \neq failure$ **then return** $[action] + plan$]
  return $failure$

function AND-SEARCH($problem, states, path$) **returns** $a\ conditional\ plan,\ or\ failure$
  for each s_i **in** $states$ **do**
    $plan_i \leftarrow$ OR-SEARCH($problem, s_i, path$)
    if $plan_i = failure$ **then return** $failure$
  return [**if** s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else** ... **if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

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

Coding Example

- 8-Puzzle – code notebook
  - https://github.com/biplav-s/course-ai-tai-f23/blob/main/sample-code/Class6-To-Class9-search.md

Lecture 9: Summary

- We talked about
  - Hill climbing
  - Simulated Annealing
  - Genetic programming
  - Search in complex environments

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%]**

About Next Lecture – Lecture 10

Lecture 10: Adversarial Games

- Game tree
- Adversarial games