CS 232: Artificial Intelligence

Spring 2024

Prof. Carolyn Anderson
Wellesley College


ASK ME ANYTHING
TUES JAN 30 | 3:30-4:30 | HCI LAB
with CS department co-chairs Orit and Sohie
ask a question here...
Fruit and dessert will be provided, so come ask Orit and Sohie your burning questions.
ALL students are welcome!!

Reminders

- Homework 2 will be released today
- Lepei has help hours Wednesday 12-2pm (Zoom)
- I have help hours Friday from 3:30-4:30pm
- Reading for next Tuesday: YLLATAILY Chapter 3-4

YLLATAILY Chapters 1-2

YLLATAILY Chapters 1-2

Signs of AI Doom

* Learning from Flawed Data
* The Problem Is Too Hard
* Sneaky Shortcuts
* The Problem is Not What We Thought

Example Tasks

- Self-driving cars
- Recipe generation
- Résumé screening
- Cockroach farming
- Tic-tac-toe
- Image recognition
- Joke generation
- Super Mario
- Writing news articles

- bias in person detection
- flawed data
- problem is too hard
- flawed data
- taking shortcuts - hard to evaluate
- problem is too hard
- sneaky shortcuts
- agent versus environment
- sneaky shortcuts & flawed data
- problem is too hard

Big Ideas

Rule-based programming

- Pro: we understand the rules the program is using
- Con: we have to write the rules

Supervised learning

- Pro: AI generates its own rules
- Con: hard to understand *why* it's doing what it's doing

Big Ideas

AI Weaknesses

- Remembering things
- Planning ahead
- Data- and computation-intensive

Recap

Agent Complexity

Problem-solving agent: capable of considering a sequence of actions that form a path to a goal state (planning ahead).

Formal Definition

1. ***States***: a set S
2. *An initial state* $s_i \in S$
3. ***Actions:*** *a set* A

   $\forall s$ $Actions(s)$ = *the set of actions that can be executed in* s*, that are applicable* in s.
4. ***Transition Model:*** $\forall s \forall a \in Actions(s)\ Result(s, a) \rightarrow s_r$

   s_r *is called a successor of* s

   $\{s_i\} \cup Successors(s_i)^*$ = *state space*
5. ***Path cost*** *(Performance Measure)*: Must be additive, e.g. sum of distances, number of actions executed, …

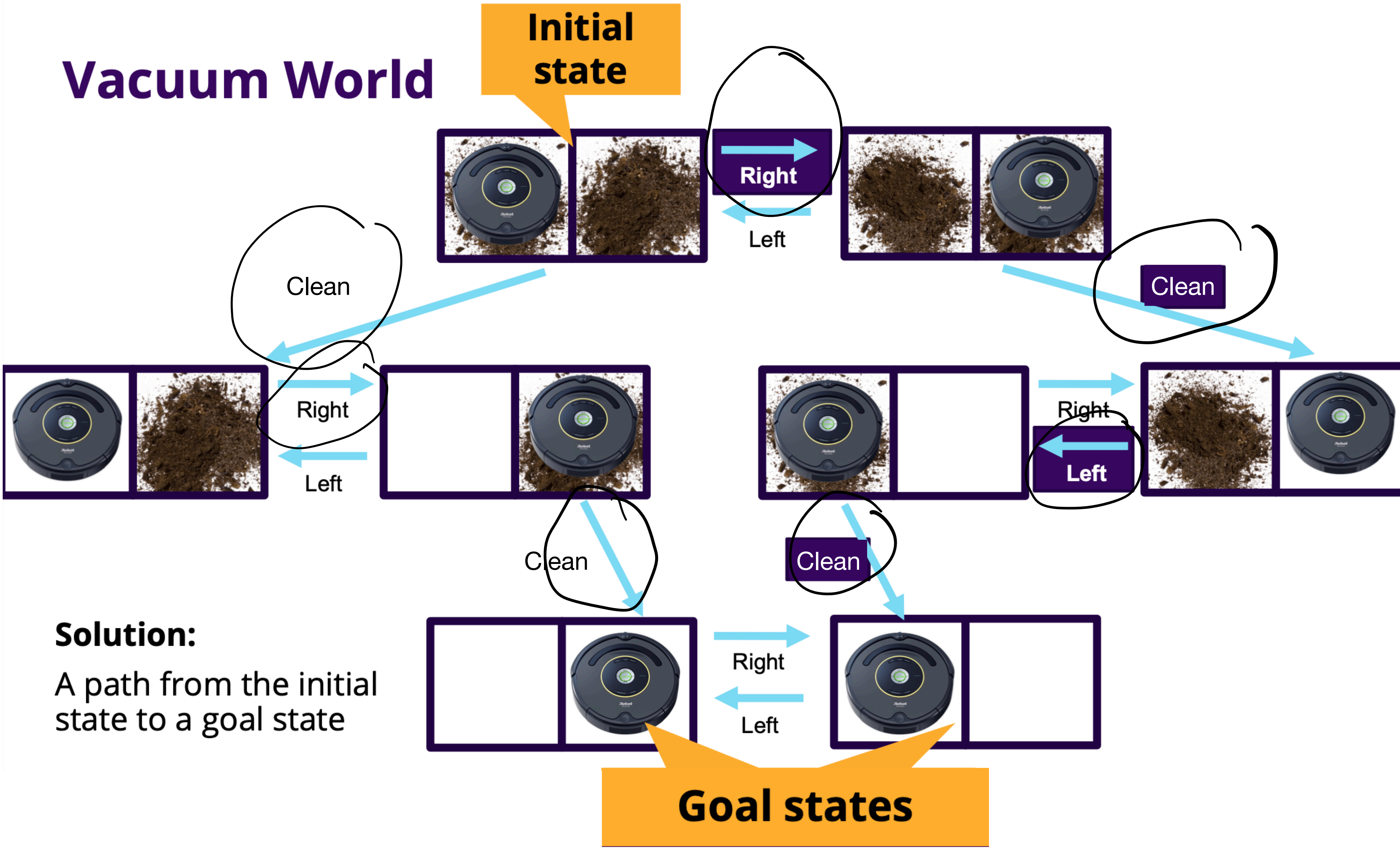
   $c(x,a,y)$ is the step cost, assumed ≥ 0

   - (where action a goes from state x to state y)
6. ***Goal test:*** $Goal(s)$

   Can be implicit, e.g. $checkmate(s)$

   s is a *goal state* if $Goal(s)$ is *true*

Vacuum World



Solution:

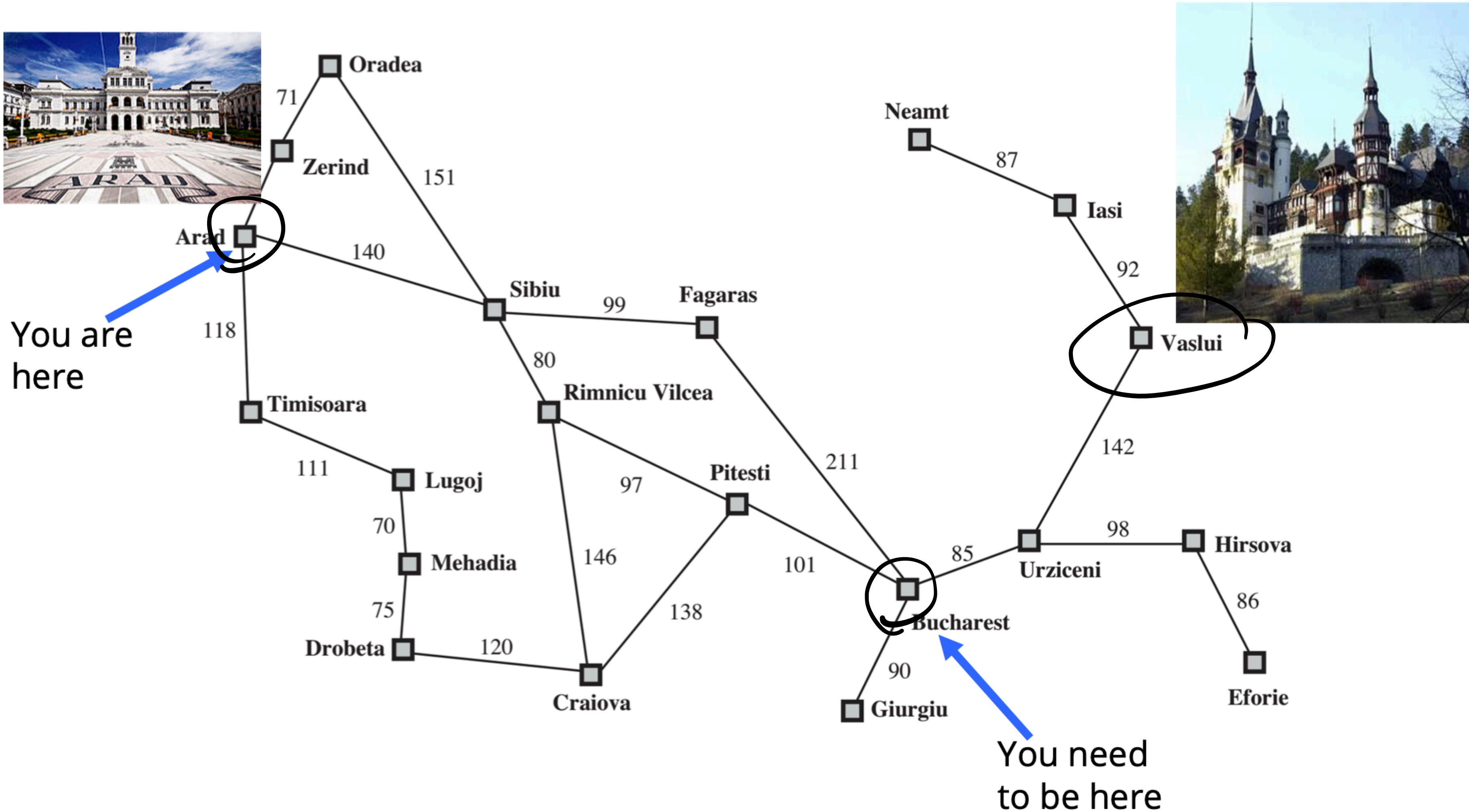
A path from the initial state to a goal state

Evaluating Solvers

- **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
- **Optimality**: Does the strategy find the optimal solution?
- **Time complexity**: How long does it take to find a solution?
- **Space complexity**: How much memory is needed to perform the search?

Search

Example search problem: Holiday in Romania



Slides adapted from Chris Callison-Burch

Holiday in Romania

On holiday in Romania; currently in Arad

- Flight leaves tomorrow from Bucharest

Formulate ***goal***

- Reach Bucharest

Formulate ***search problem***

- States: different cities
- Actions moving between cities
- Performance measure / cost: minimize distance

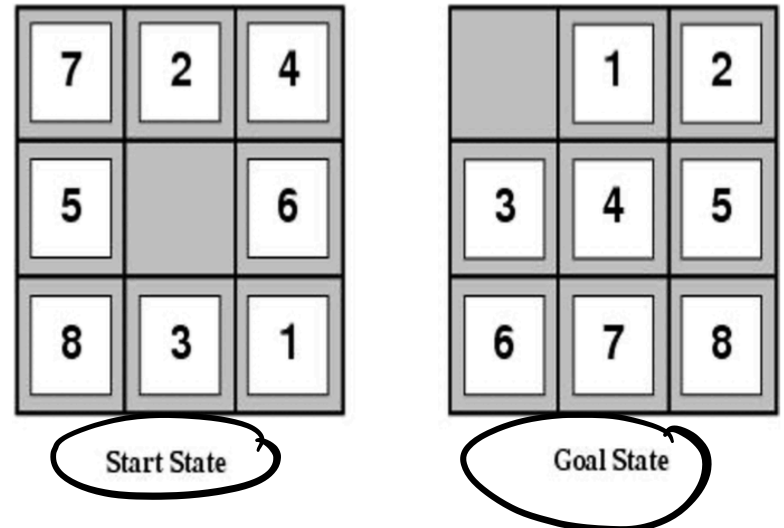
Find ***solution***

- Sequence of cities starting in Arad & ending in Bucharest

Example search problem: 8-puzzle

Formulate *goal*

- Arrange the tiles as shown in goal state



Formulate *search problem*

- States: an arrangement of the board
- Actions: left, up, right, down for a given tile
- Cost: # of moves

Find *solution*

List of different moves to take from start to goal

3 up, 8 Right, . . .

Search Algorithms

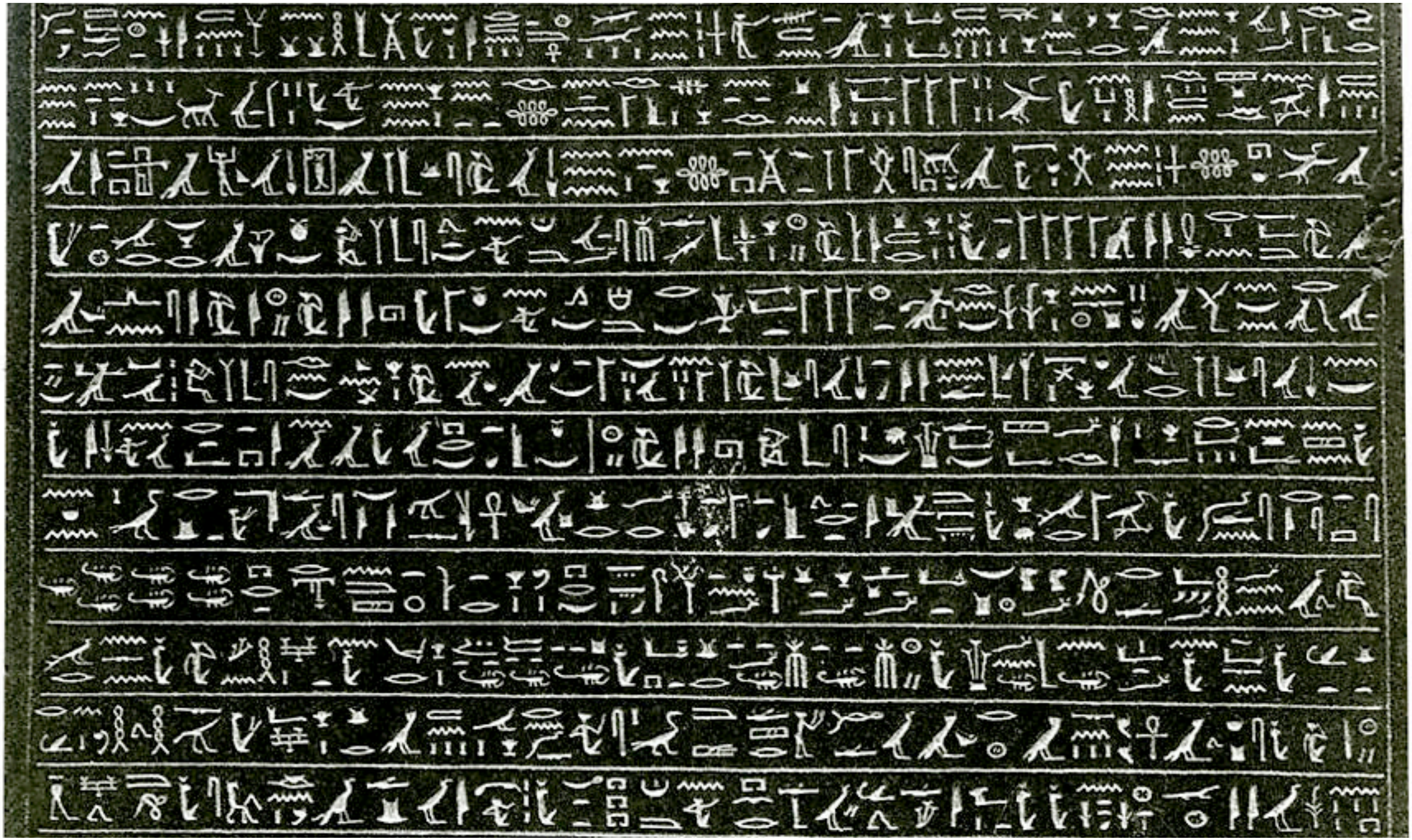
Backtracking Search

One of the simplest search strategies for puzzle-solving is **backtracking search**.

In backtracking search, we make a guess to eliminate possibilities (narrow the search space).

If we run into a logical inconsistency, we **backtrack** (undo our guess) and try out a different guess.

Backtracking Search Application



Magical Stela (Cippus of Horus), Metropolitan Museum of Art

Backtracking Search Application