15-780: Graduate AI
Lecture 12: LLM pipeline

Aditi Raghunathan

Recap

- Going from a base-model to a deployed chatbot
  - "Alignment" to make the model follow instructions, human preferences about safety toxicity

- A natural strategy is to collect examples of behaviors we do want
  - Minimize loss on this data (supervised learning)
  - Language model has a rich initialization (trained on trillions of tokens)

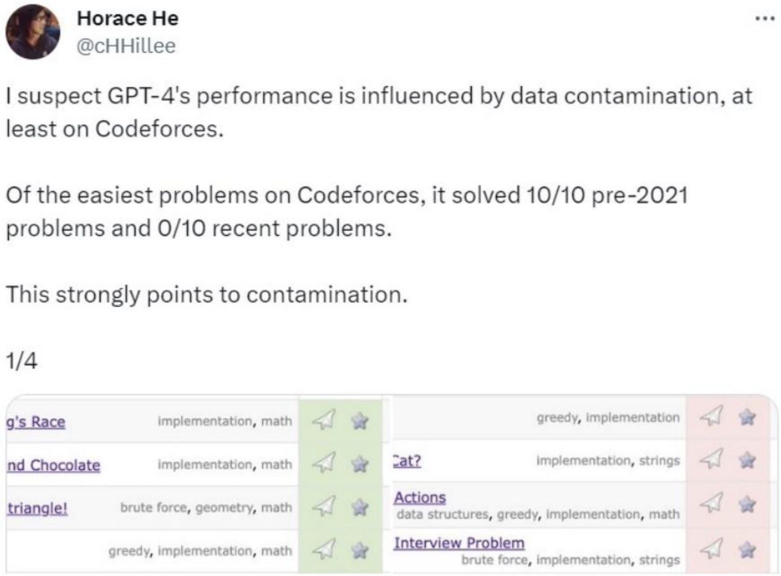
Learning from preferences

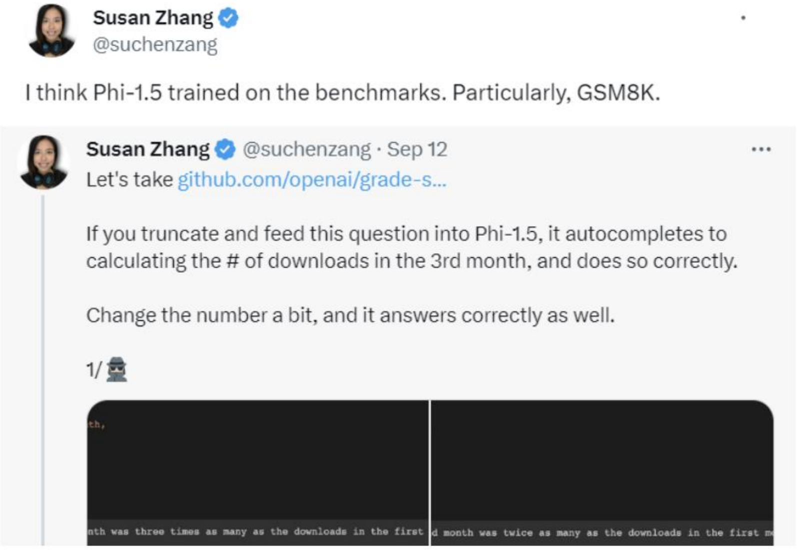
$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right].$$

- Increase likelihood of ”good” responses and lower likelihood of ”bad” responses
- There is a formal derivation in terms of learning a reward model and maximizing reward
  - We will get into this shortly in the course

Evaluation

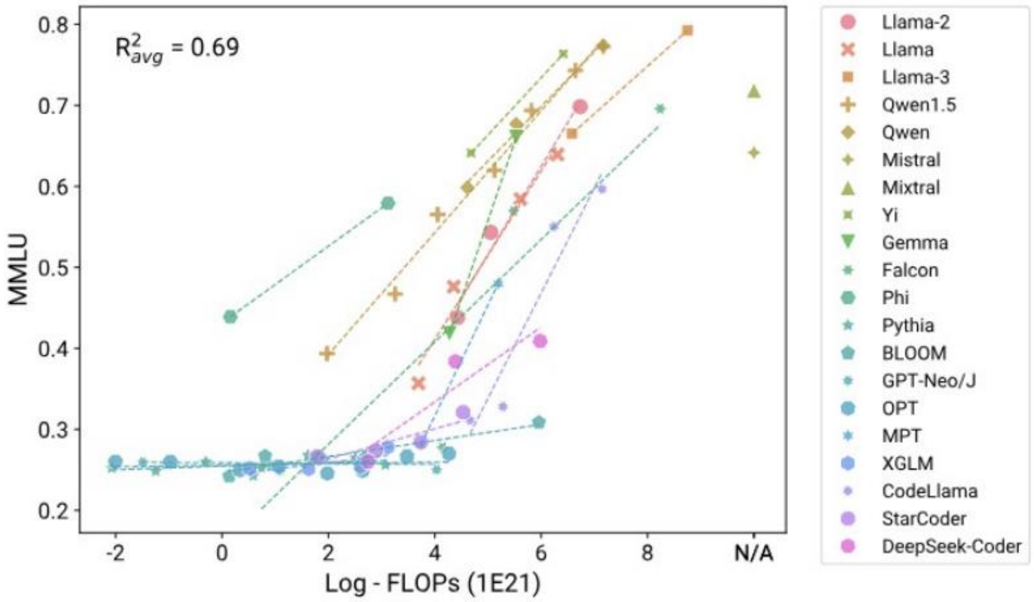
- Standard benchmarks
  - Sensitivity to prompt format
  - Issues of contamination



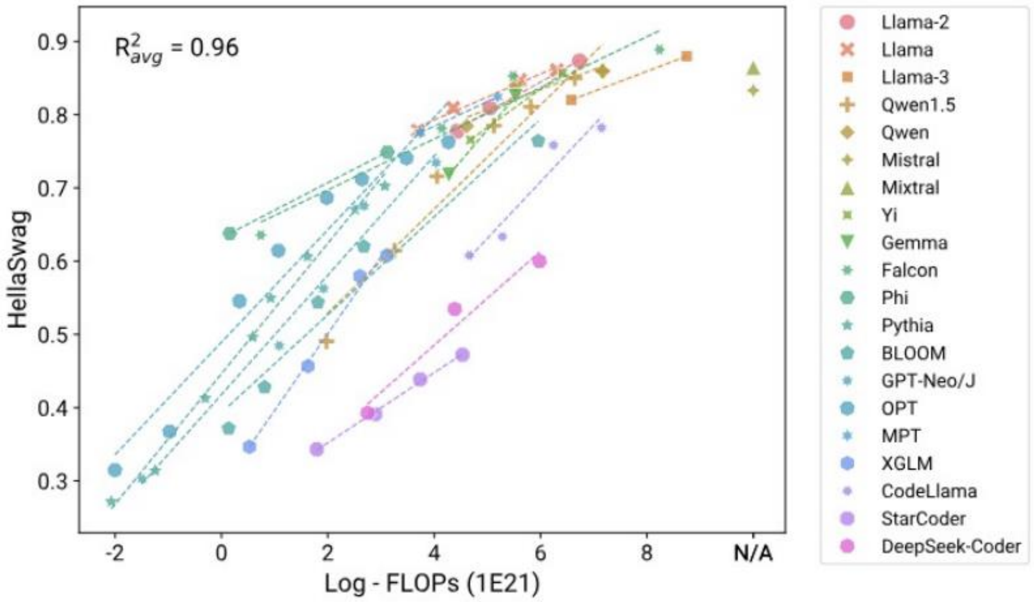


- Saturation: Hard to make ever increasingly tough benchmarks

Effect of scale on benchmarks

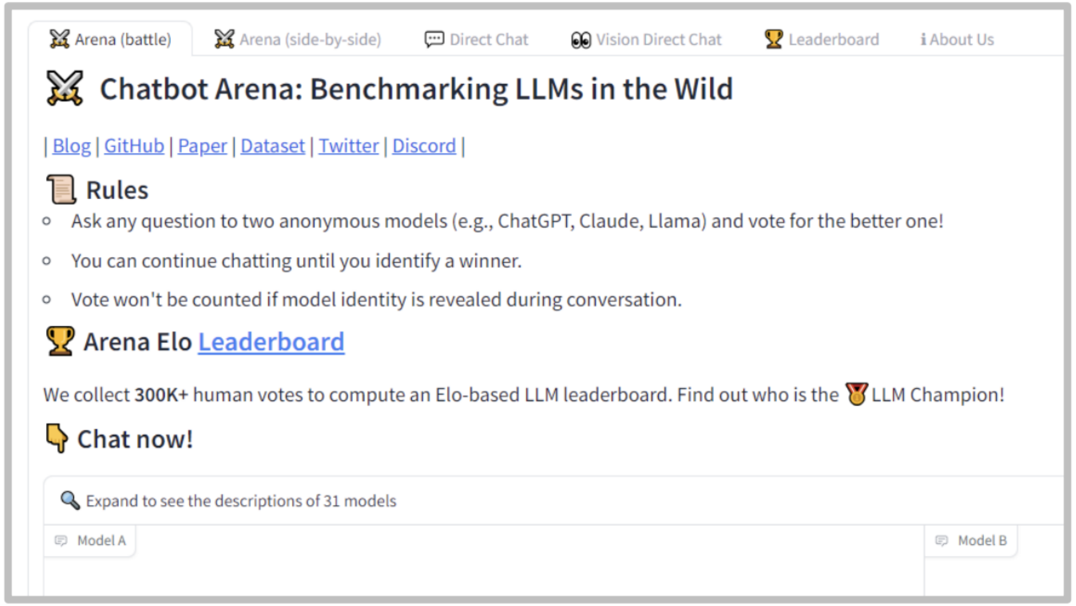


(a) MMLU



(b) HellaSwag

Evaluation: user-facing system



- Blind user rates which model's responses are better
- LMSYS-Chat-1M: one million real-world conversations

What is missing with prediction systems?

Chain-of-thought prompting

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27.

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9.

Can language models "reason"?

A model is just shown Q, A pairs and asked to ***predict*** the answer

vs

A model ***reasons*** about the questions and comes up with answer

[Hendrycks et al., 2021c]

AIME

For any finite set X, let $|X|$ denote the number of elements in X. Define

$$S_n = \sum |A \cap B|,$$

where the sum is taken over all ordered pairs (A, B) such that A and B are subsets of $\{1, 2, 3, \cdots, n\}$ with $|A| = |B|$. For example, $S_2 = 4$ because the sum is taken over the pairs of subsets

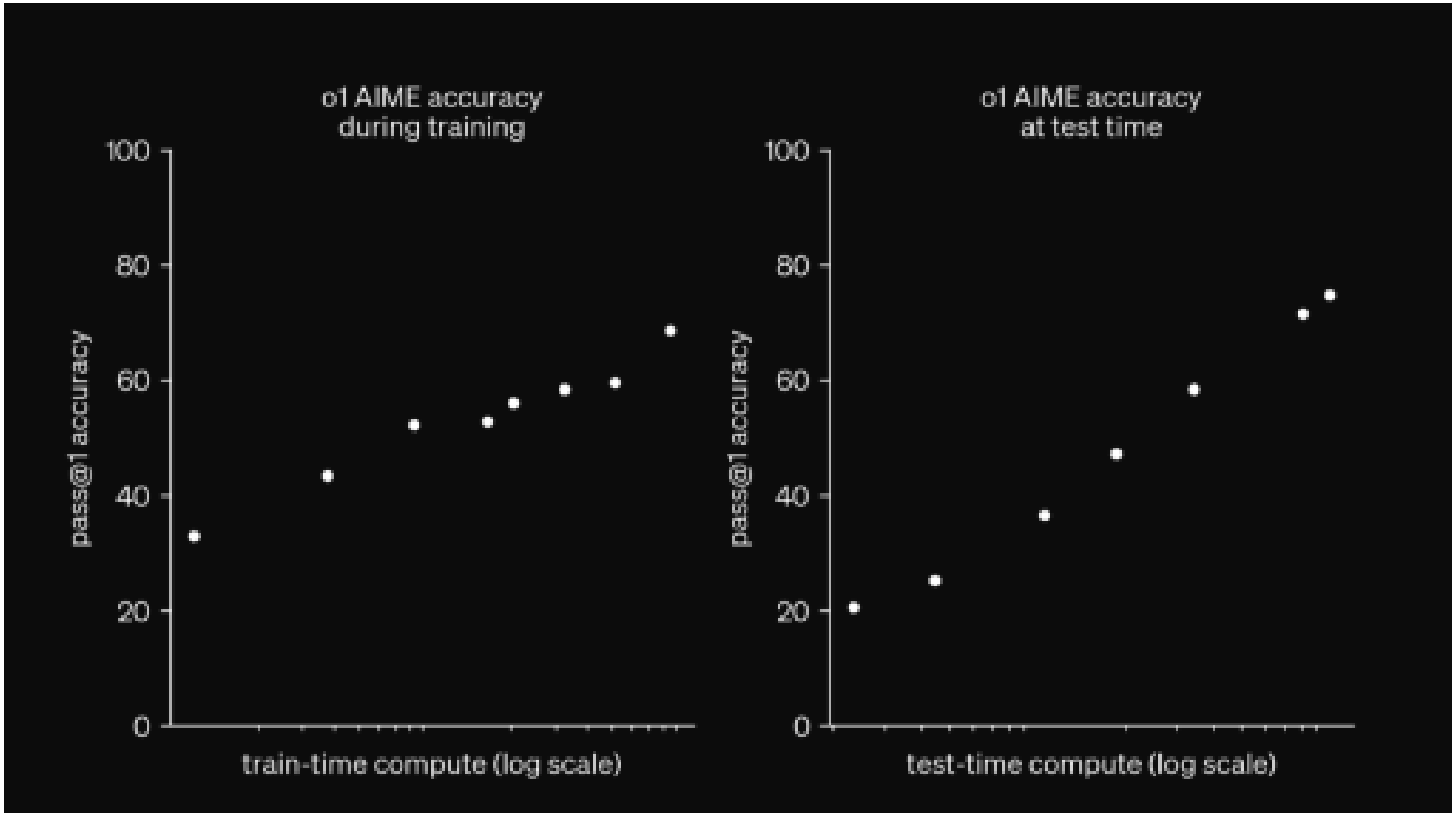
$$(A, B) \in \{(\emptyset, \emptyset), (\{1\}, \{1\}), (\{1\}, \{2\}), (\{2\}, \{1\}), (\{2\}, \{2\}), (\{1, 2\}, \{1, 2\})\}$$

giving $S_2 = 0 + 1 + 0 + 0 + 1 + 2 = 4$. Let $\frac{S_{2022}}{S_{2021}} = \frac{p}{q}$, where p and q are relatively prime positive integers. Find the remainder when $p + q$ is divided by 1000.

The bitter lesson

- "**Search** and **learning** are the two most important classes of techniques for utilizing massive amounts of computation in AI research"
- Learning: scaled up using larger models and training on trillions of tokens

Scaling up test-time compute



Classifiers vs search

- Classifiers: x -> single output y

- Search problem: x -> action sequence

Key: need to consider future consequence of action

- *Can you just repeatedly use a reflex model?*

- What about generating one token at a time?
  - In language models, you condition on the history
  - Can encode state information
  - Might need special post-training for some of the more sophisticated algos to emerge

Coming up

- Search
- MDPs and reinforcement learning
- Game theory

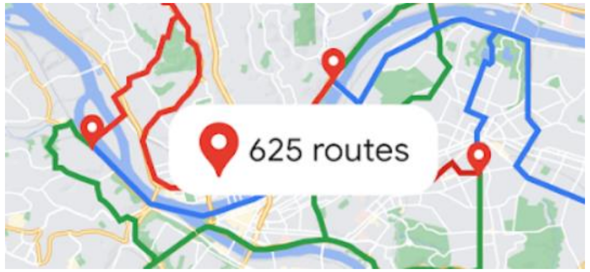
Search

Search

- Formulation
- Tree search
- Dynamic programming
- Uniform cost search
- A*
- A* relaxations

Examples

- Route finding
  - Minimize time, fuel etc



- Robot motion planning
  - Find the fastest path
  - Popular search algorithms developed in the context of robots

- Solving puzzles like Rubik's cube
  - Want to get to a certain configuration
  - https://www.youtube.com/watch?v=7RvdTWM9sJA

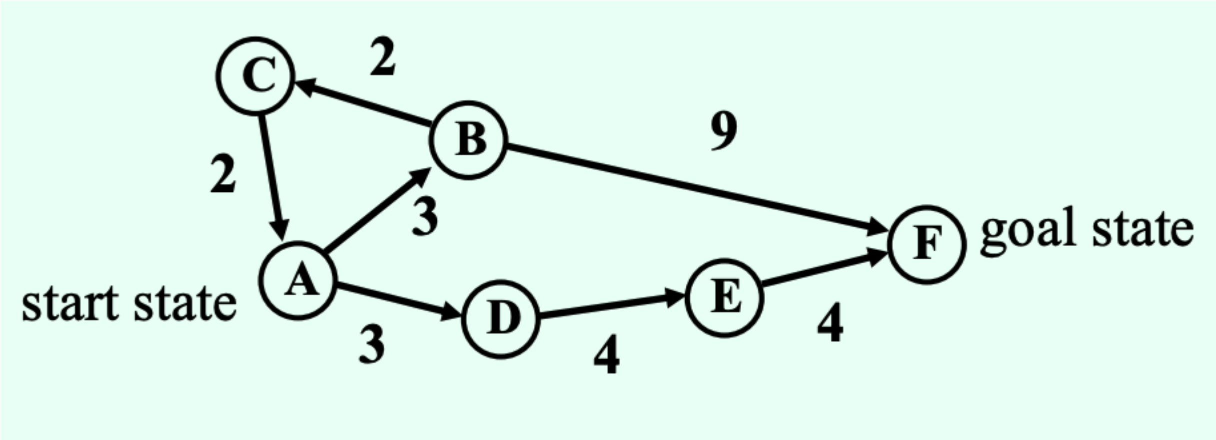
Search problem

- We have some actions that can change the state of the world
  - Change induced by an action is perfectly predictable
- Goal: try to come up with **a sequence of actions** that will lead us to a goal state while minimizing the cost
- Do not need to execute actions in real life while searching for solution!
  - Everything perfectly predictable anyway

Search problem

- s_{start}: start state
- Action(s): possible actions at state s
- Cost (s, a): cost of taking action a at state s
- Succ (s, a): state you end up in after you take action a at state s
- IsEnd(s): reached end state?

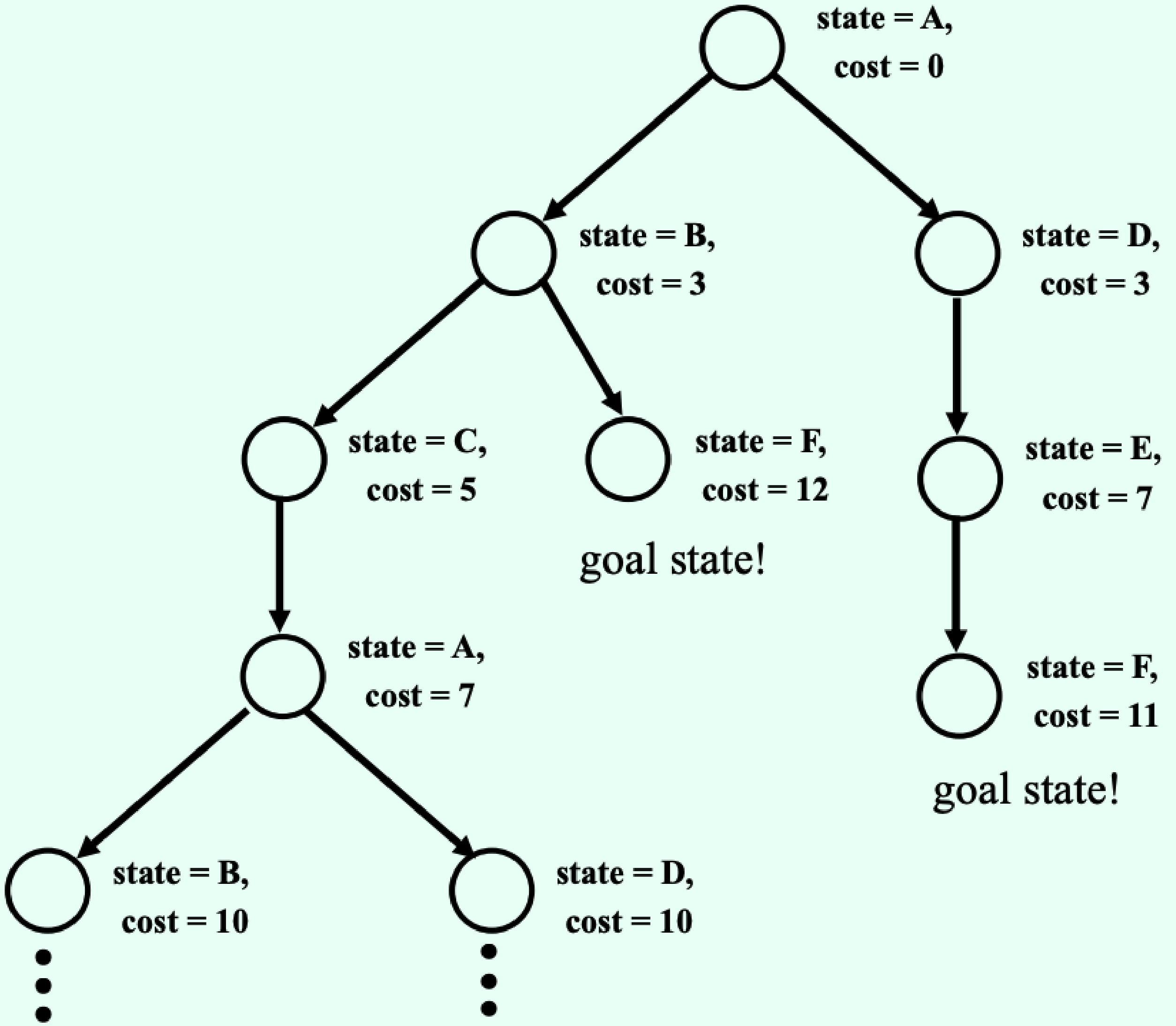
Traveling a graph



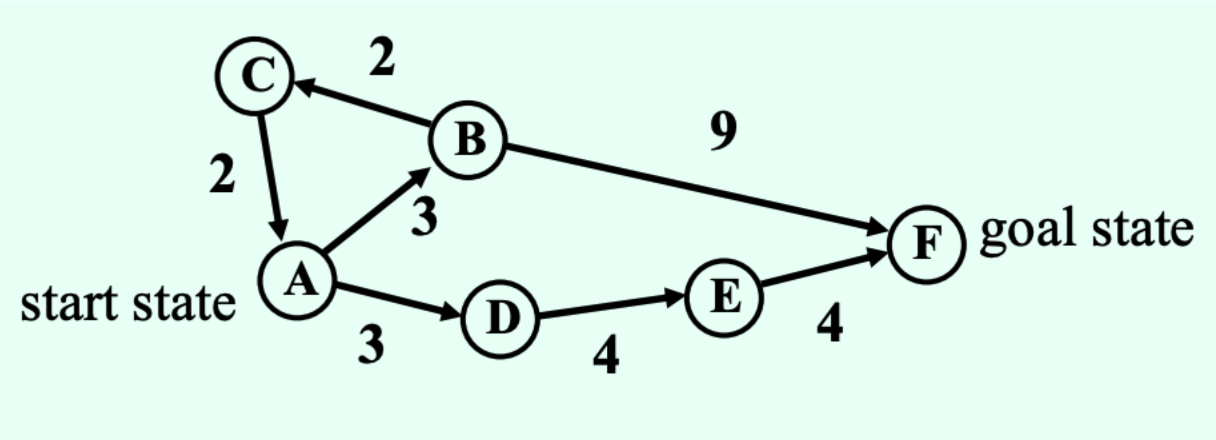
Find minimum cost path from start to goal

State formulation

Full search tree



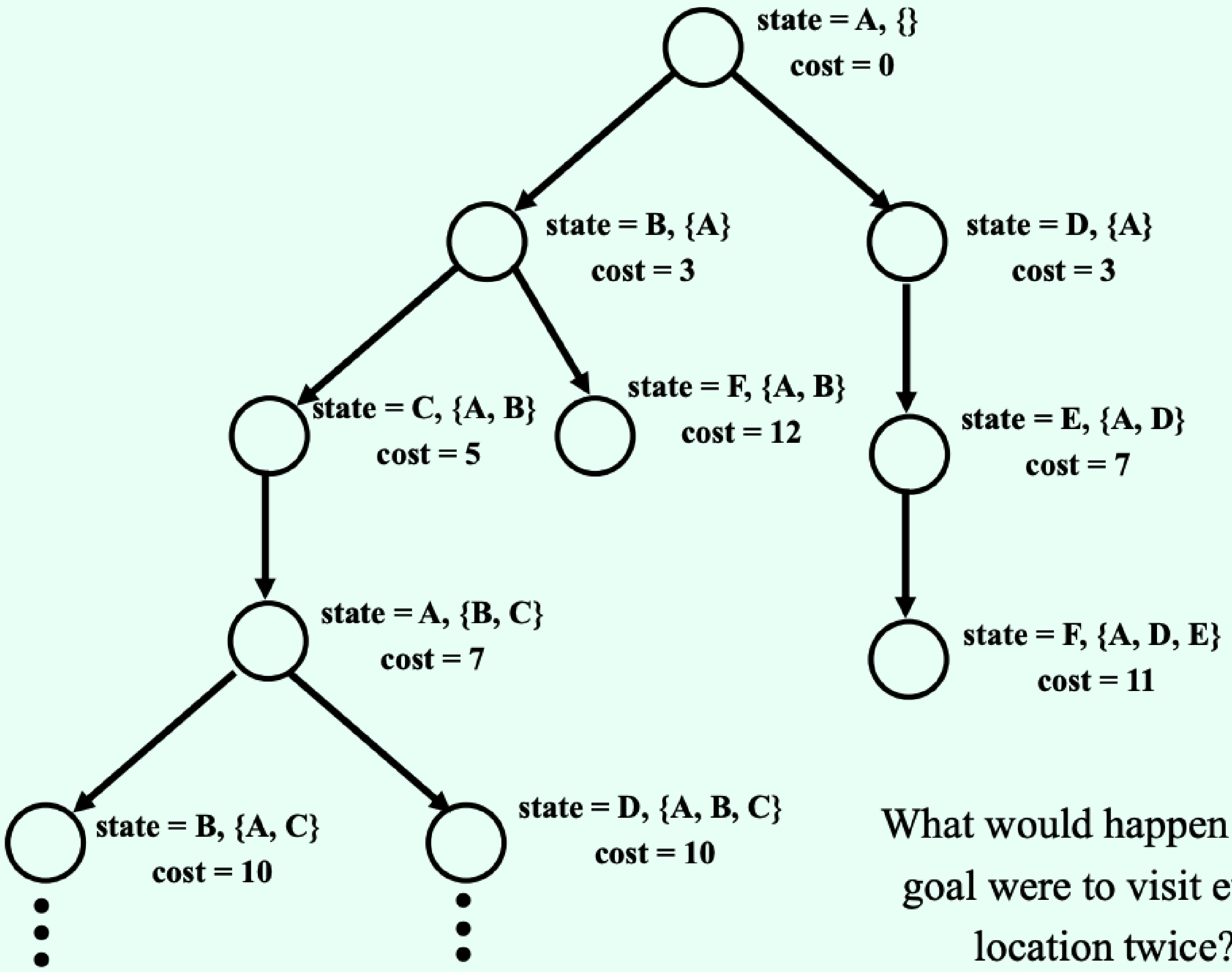
New goal



Minimum cost path that traverses all the nodes

State formulation

Full search tree



What would happen if the goal were to visit every location twice?

Uninformed search

- Given a state, we only know whether it's a goal state or not
  - Cannot compare non-goal states
- Traverse the space "blindly" in the hope of finding the goal
- Blindly but **systematically**

Measuring performance

- **Completeness:** is the algorithm guaranteed to find *a* solution?
- **Cost optimality:** does it find lowest cost solution of all?
- **Time complexity:** how long to find a solution
  - Measured abstractly by number of states and actions
- **Space complexity:** how much memory is needed
- **Notation:**
  - Branching factor: b
  - Max depth of tree: m
  - Depth of shallowest solution: d

Breadth-first search

- Root first, then all successors of root node, and then their successors and so on

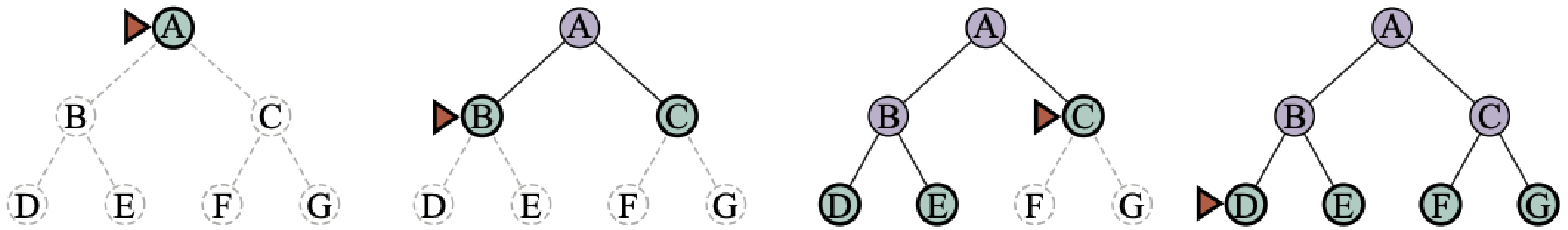


Figure 3.8 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

Breadth-first search: piazza poll

- (A) BFS is complete and cost-optimal
- (B) BFS is complete but not necessarily cost-optimal
- (C) BFS is cost-optimal but not complete
- (D) BFS is neither cost-optimal nor complete

Breadth-first search

- **Complete:** yes, we will eventually search all paths exhaustively
- **Cost-optimal:** only if all actions have the same cost
  - Complete in either case

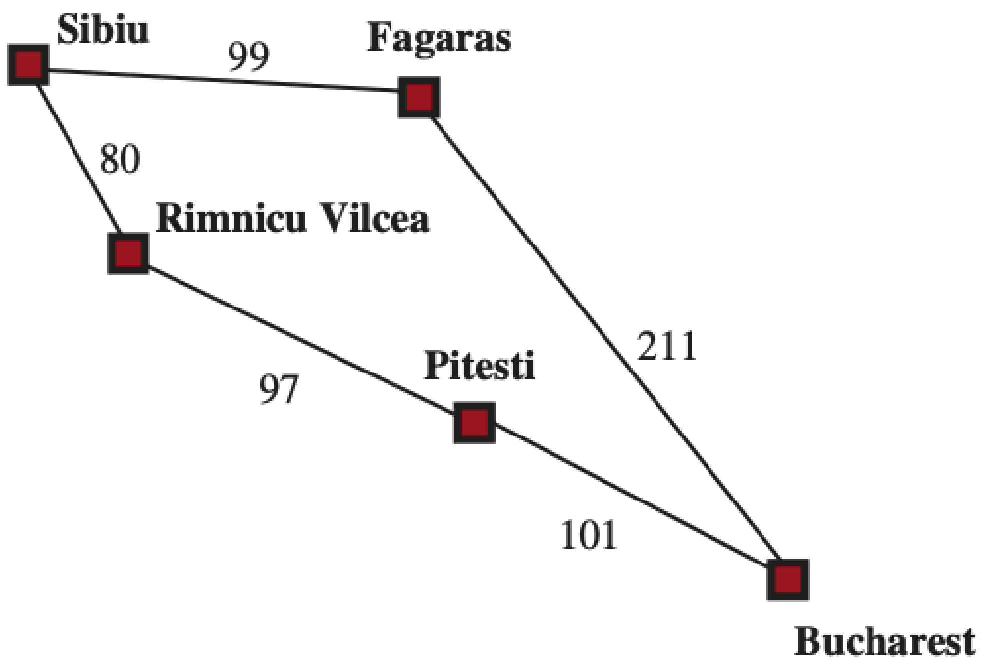
- **Implementation**
  - BFS maintains a queue of states to be explored. It pops a state off the queue, then pushes its successors back on the queue

- **Time-complexity:** suppose solution is at depth d, total nodes generated are $1 + b^2 + b^3 \dots b^d = O(b^d)$

- **Space-complexity:** All nodes remain in memory so also $O(b^d)$

Djikstra or uniform-cost search

- Expand the node with a shortest path from root



Sibiu to Bucharest

- Sibiu is added to the queue and expanded
- Candidates: R (80) and F(99)
  - Least cost is R
  - R is expanded and added to queue
- Candidates: P (177), F(99)
  - Least cost is F, added to queue and expanded
- Candidates: P(177), **B(310)**
  - Least cost is P, added to queue and expanded
- Candidates: B'(278), B(310)
  - Least cost is B'
  - Goal reached

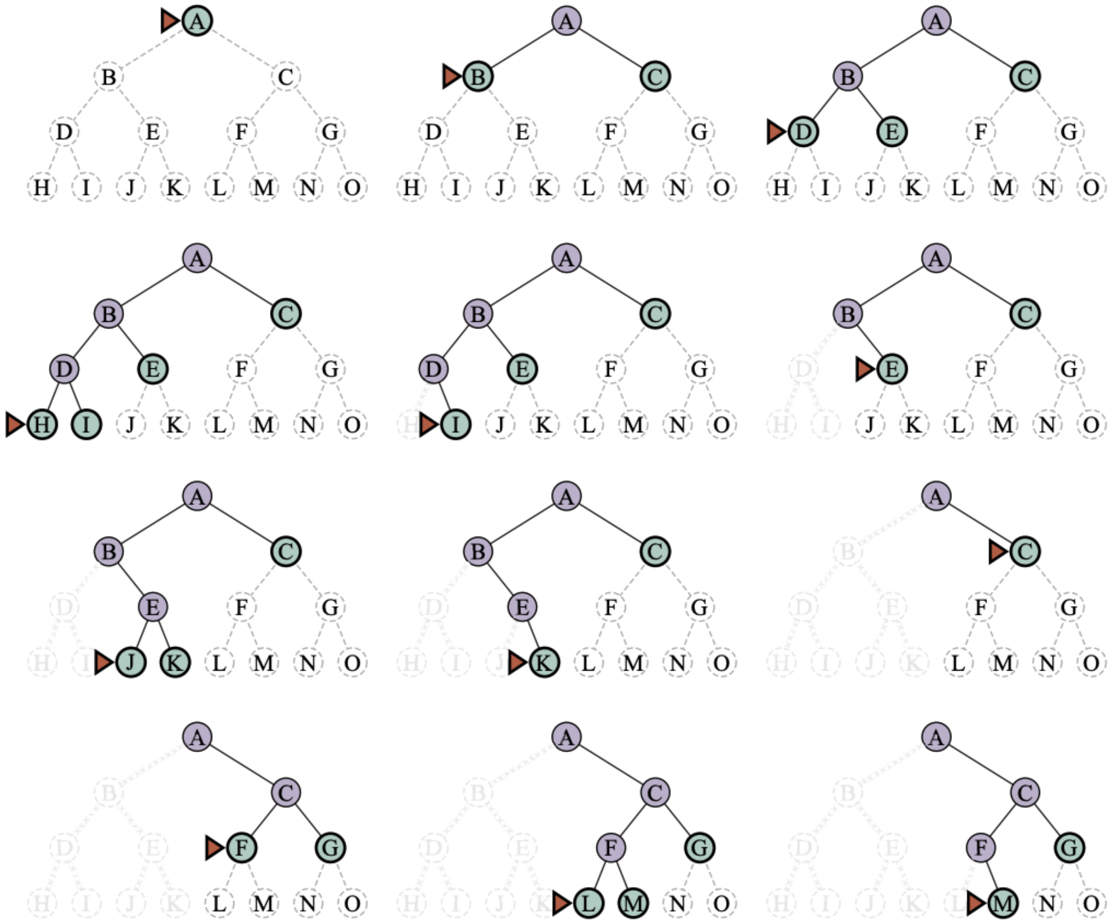
Djikstra or UCS

- **Complete:** yes, we will eventually search all paths exhaustively
- **Cost-optimal:** yes!
- **Time-complexity:** $O(b^{1+C^*/\epsilon})$
  - $\boldsymbol{C^*}$ is the cost of the optimal path
  - $\boldsymbol{\epsilon}$ is a lower bound on the cost of the path

- Longer than BFS potentially
  - Explore larger depth paths in search for a shorter path
- BFS spreads out in waves of uniform depth, Djikstra spreads out in waves of uniform cost

Depth-first search

- Expand deepest node



Depth-first search

- **Complete:** No, can get stuck in cyclic states
  - Some implementations check for ”new” states
- **Cost-optimal:** Also no
- **Memory:** O(bm)
  - m is max-depth of the tree
  - can be implemented as O(m) memory
- **Time:** $O(b^m)$
- Really saving on memory!
  - Slight worse time over BFS $O(b^d)$

Iterative deepening

- *Best of both*
  - Prevent DFS from wandering down an infinite path
- Depth limited: fix depth to be l with DFS
  - Time complexity is $O(b^l)$
  - Space complexity is $O(b\ l)$
- **Iterative deepening search:** keep increasing l from 1, 2, … d
  - Time complexity is $O(b^d)$
  - Space complexity is $O(bd)$

Summary

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