COMP3211 Tutorial 3: Search

Fengming ZHU

Feb. 26&29, 2024

Department of CSE
HKUST

Outline

Search

Before Formulations

Formulation

Search Diagram

Exercise

Search

Simple Agents

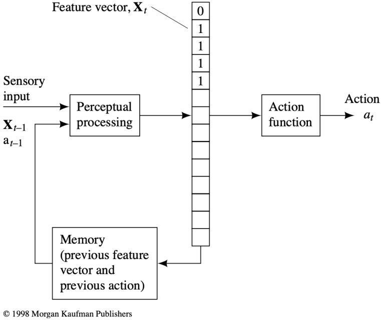


Figure 1: Simple agents

Simple Agents

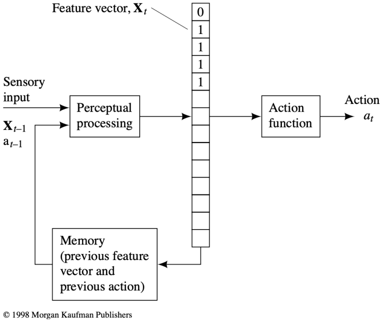


Figure 1: Simple agents

Key points:

- Respond to the environment,

Simple Agents

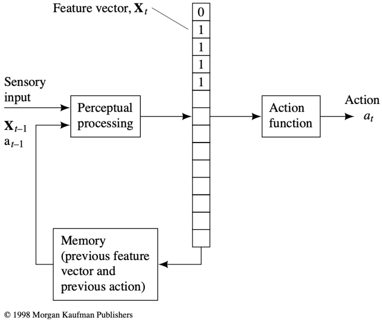


Figure 1: Simple agents

Key points:

- Respond to the environment,
- Consider how the world IS, or HAVE BEEN,

Simple Agents

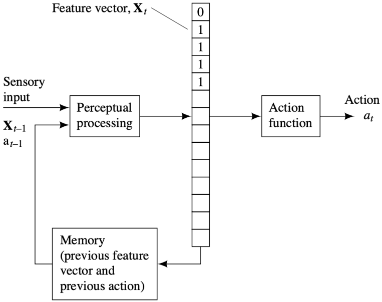


Figure 1: Simple agents

Key points:

- Respond to the environment,
- Consider how the world IS, or HAVE BEEN,
- Cannot imagine how the world WOULD BE.

Formulation

Notations:

- A set of states $\mathcal{S}$
- An initial state $I \in \mathcal{S}$
- A goal state $G \in \mathcal{S}$ (sometimes a goal test)
- A set of actions $\mathcal{A}$
- Deterministic transitions $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$
- Cost function $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- A solution (path) is a sequence of actions from I to G.

Follow-up Questions

Question #1:
Can reactive agents or state machines do search?

Follow-up Questions

Question #1:
Can reactive agents or state machines do search?

– Neither, should be agents that can plan ahead (transitions matter).

Follow-up Questions

Question #1:
Can reactive agents or state machines do search?

– Neither, should be agents that can plan ahead (transitions matter).

Question #2:
For agent 1, who can compute a feasible plan (path), if you extract her plan and deploy to agent 2 under the same setting, who has no sensing ability and no computing power, can she successfully reach the goal?

Follow-up Questions

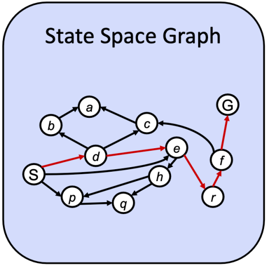
Question #1:
Can reactive agents or state machines do search?

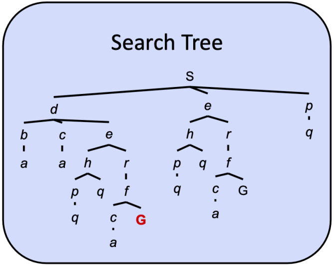
– Neither, should be agents that can plan ahead (transitions matter).

Question #2:
For agent 1, who can compute a feasible plan (path), if you extract her plan and deploy to agent 2 under the same setting, who has no sensing ability and no computing power, can she successfully reach the goal?

– Yes, once computed, just blindly execute it.

General Idea: Graph/Tree Search





A node can only be expanded once, while it may be visited multiple times!

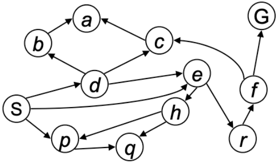
Key concepts:

- Fringe (frontier)
- Expansion
- Exploration strategy

Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue



Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

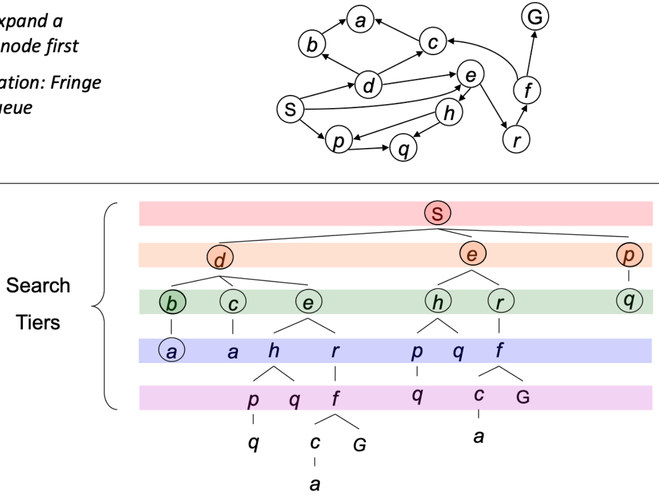
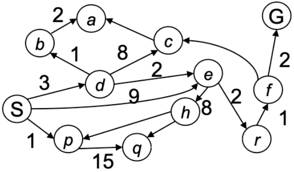


Figure 2: Breadth-First Search

Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



Uniform Cost Search

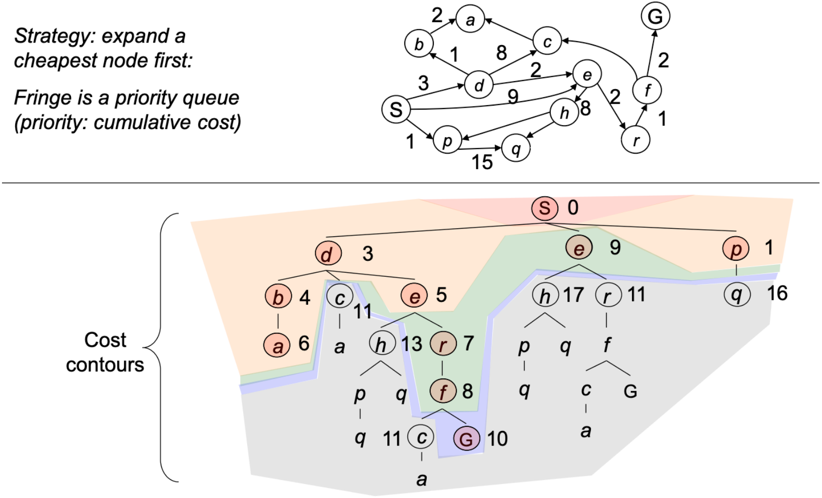


Figure 3: Uniform Cost Search

Greedy Search

- Strategy: expand a node that you think is closest to a goal state.
- Best case: every time you make a perfect guess.
- Worst case: turn around until you get into a dead end.

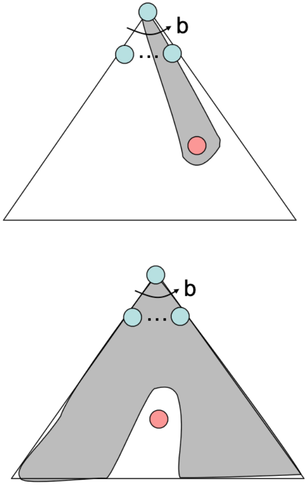


Figure 4: Uniform Cost Search

A^* Search

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)

A^* Search

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)
- Strategy: expand a node that is best so far.

A^* Search

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)
- Strategy: expand a node that is best so far.
- Implementation: priority queue, f(n) = g(n) + h (n).

A^* Search

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)
- Strategy: expand a node that is best so far.
- Implementation: priority queue, f(n) = g(n) + h (n).
- Admissable heuristic: $h(n) \leq cost(n, G)$.

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)
- Strategy: expand a node that is best so far.
- Implementation: priority queue, f(n) = g(n) + h (n).
- Admissable heuristic: $h(n) \leq cost(n, G)$.

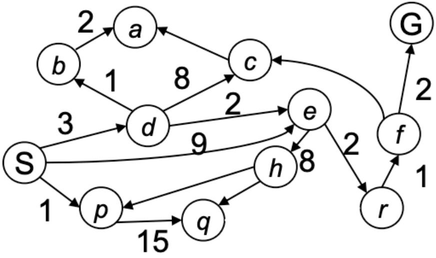


Figure 5: $h(n) = shorst\_path\_length(n, G)$

A^* Search

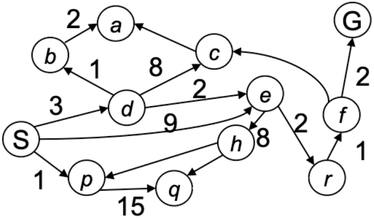
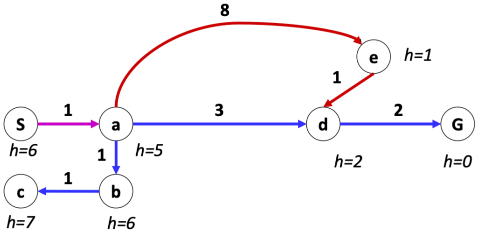
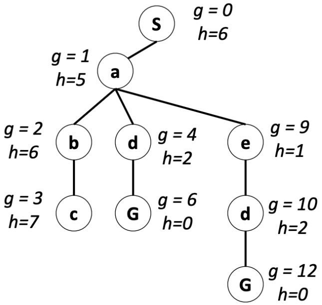


Figure 6:
$h(n) = shorst\_path\_length(n, G)$

A^* Search – another example





A^* Search – demo

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)
- Strategy: expand a node that is best so far.
- Implementation: priority queue, f(n) = g(n) + h (n).
- Admissable heuristic: $h(n) \leq cost(n, G)$.
- Live demo: https://www.movingai.com/SAS/index.html

Exercise

A^* search:

- An offline process – when the map changes: need to replan the whole solution.

Exercise

A^* search:

- An offline process – when the map changes: need to replan the whole solution.
- Can we reuse any historical data to be more efficient?

Exercise

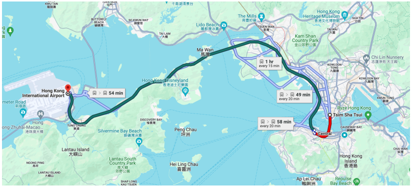
A^* search:

- An offline process – when the map changes: need to replan the whole solution.
- Can we reuse any historical data to be more efficient?

Exercise

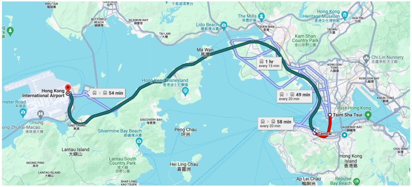
A^* search:

- An offline process – when the map changes: need to replan the whole solution.
- Can we reuse any historical data to be more efficient?



A^* search:

- An offline process – when the map changes: need to replan the whole solution.
- Can we reuse any historical data to be more efficient?



A^* search:

- An offline process – when the map changes: need to replan the whole solution.
- Can we reuse any historical data to be more efficient?

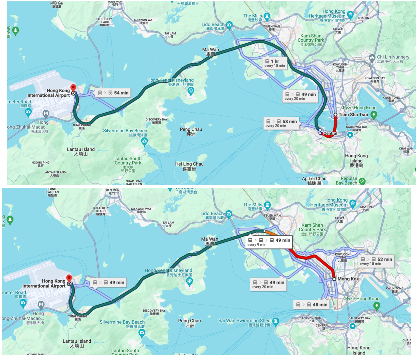


Figure 7: Plan backwrads from the goal to the start: D^* Lite

Thanks!