Artificial Intelligence

Lecture 3: Informed Search

Prof. Antonio Khalil Moretti

Week 3

SCIS 432
Spelman College

Today's Topics

Last week: Uninformed search

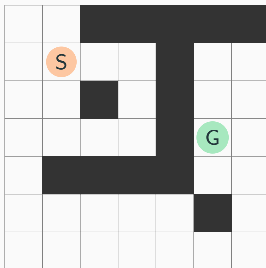
- BFS, DFS, UCS explore *blindly*
- No knowledge about goal location

This week: Informed search (heuristic search)

1. What is a heuristic?
2. Greedy Best-First Search
3. A* Search
4. Heuristic quality: admissibility and consistency

The Problem with Uninformed Search

Example: Finding path through a maze



Uninformed search (BFS, DFS) explores without direction!

Can we do better if we know *approximately* how far each position is from the goal?

What is a Heuristic?

Heuristic function $h(n)$: an *estimate* of the cost from state n to the goal

Example: Manhattan distance (ignores walls)

8	7					
7	S	5	4		2	3
6	5		3		1	2
5	4	3	2		G	1
6					1	2
7	6	5	4	3		3
8	7	6	5	4	3	4

Manhattan distance:

$$h(n) = |r_n - r_G| + |c_n - c_G| \qquad \text{(minimum grid moves, ignoring walls)}$$

Note: cells with smaller h may still be blocked by walls.

Greedy Best-First Search

Follow the heuristic

Strategy: Expand state that *appears* closest to goal

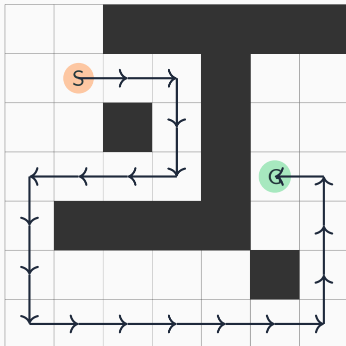
Implementation:

- Priority queue by $h(n)$
- Expand lowest $h(n)$

Algorithm 1 Greedy Best-First

```
1: frontier = PQ by h(n)
2: Add start to frontier
3: while frontier not empty do
4:     node = frontier.pop()
5:     if node is goal then
6:         return solution
7:     end if
8:     Add neighbors
9: end while
```

Greedy Best-First Search (Suboptimal)



Result: Finds a path, but it is *not optimal*.

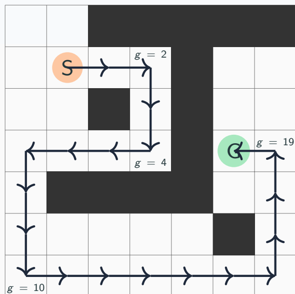
Greedy Best-First Search

- Expands the node with smallest $h(n)$
- Uses a closed set (no revisiting)
- Ignores path cost so far $g(n)$

What goes wrong?

- Heuristic looks promising locally
- Search commits to the wrong corridor
- Forced into a long detour

Problems with Greedy Best-First



Numbers show actual cost from start $g(n)$ along greedy path

Why Greedy Fails

- Greedy selects nodes using only $h(n)$
- It ignores how expensive the path so far is

What we see in the maze:

- Near the goal, $h(n)$ is small
- But $g(n)$ is already very large
- Greedy commits early and pays later

Key takeaway:

Minimizing $h(n)$ alone is not enough

Can we fix this? **Yes — A* Search**

A* Search

The optimal informed search

Key idea: Consider both actual and estimated costs

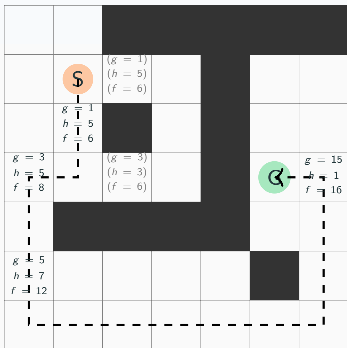
Evaluation function:

$$f(n) = g(n) + h(n)$$

- $g(n)$ = actual cost from start to n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost through n

Strategy: Expand state with lowest $f(n)$

A* Search Example



Legend: Black = eligible (OPEN / expanded)

Gray = discovered later

Key idea

A* may *explore* non-optimal branches, but

Important distinction

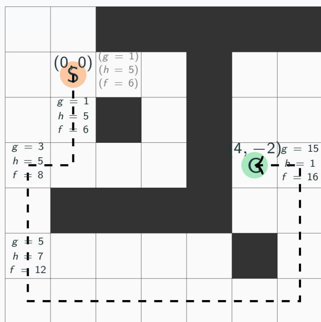
- The **dashed path** is the *final optimal solution*
- It is **not** the order in which nodes are expanded

Why doesn't A* move right first?

- From S, both right and down have $f = 6$
- A* may expand either one (tie-breaking)
- Exploring right is allowed, but does not lead to the best goal

the first goal returned has minimal $g(n)$

A* Expansion Order vs Final Solution



Black = expanded / eligible Gray = discovered later

OPEN list evolution (using grid coordinates)

Step 0
OPEN = $\{(0, 0)\}$

Step 1: expand $(0, 0)$
OPEN =

$(0, -1)$ $f = 6$
$(1, 0)$ $f = 6$

Step 2: expand one $f = 6$ **node (tie)**
OPEN =

$(0, -2)$ $f = 6$
$(1, 0)$ $f = 6$

Key observation

- OPEN contains candidates, not commitments
- Nodes off the optimal path may appear
- The first *goal* removed from OPEN is optimal

Takeaway: A* explores via a priority queue; optimality depends on when the goal is selected, not on early moves.

A* Algorithm

Algorithm 2 A* Search

```
1:  frontier = priority queue ordered by f(n) = g(n) + h(n)
2:  start.g = 0, Add start to frontier
3:  while frontier not empty do
4:      node = frontier.pop()                          ▷ Lowest f(n)
5:      if node is goal then return solution
6:      end if
7:      for each neighbor of node do
8:          neighbor.g = node.g + 1                    ▷ Cost per move = 1
9:          neighbor.f = neighbor.g + h(neighbor)
10:         Add neighbor to frontier
11:     end for
12: end while
```

What Does A* Actually Do? (One Iteration)

Three states a node can be in

- **Unseen:** we have not discovered this node yet
- **OPEN:** discovered, waiting in the priority queue
- **CLOSED:** already expanded (we processed its neighbors)

One iteration of A*

1. **Remove** the node with smallest $f(n)$ from **OPEN**
2. If it is the goal, **stop** (this path is optimal)
3. Otherwise, **expand** it:
   - generate its neighbors
   - update their g, h, and f values
   - add or update them in **OPEN**
4. Add the node to **CLOSED**

Key distinction: removed from OPEN $\neq$ returned as the solution

$A*$ may remove (expand) many nodes before it ever returns a path.

Check Your Understanding

Question 1

A node with $f = 6$ is in OPEN.

Another node with $f = 8$ is also in OPEN.

Which one will A* remove next?

Check Your Understanding

Question 1

A node with $f = 6$ is in OPEN.

Another node with $f = 8$ is also in OPEN.

Which one will A* remove next?

Answer: the node with $f = 6$

Question 2

If A* removes a node from OPEN, does that mean it is part of the final solution?

Check Your Understanding

Question 1

A node with $f = 6$ is in OPEN.
Another node with $f = 8$ is also in OPEN.
Which one will A* remove next?

Answer: the node with $f = 6$

Question 2

If A* removes a node from OPEN, does that mean it is part of the final solution?

Answer: No.

Removing a node only means "expand it next," not "commit to it."

Question 3

When does A* actually *commit* to a path?

Check Your Understanding

Question 1

A node with $f = 6$ is in OPEN.
Another node with $f = 8$ is also in OPEN.
Which one will A* remove next?

Answer: the node with $f = 6$

Question 2

If A* removes a node from OPEN, does that mean it is part of the final solution?

Answer: No.

Removing a node only means "expand it next," not "commit to it."

Question 3

When does A* actually *commit* to a path?

Answer: when the goal is removed from OPEN

At that moment, the returned path has minimal $g(n)$ among all possible paths.

Admissible Heuristics

When is A* optimal?

Definition: A heuristic $h(n)$ is **admissible** if

$$h(n) \leq h^*(n) \quad \text{for all } n$$

where $h^*(n)$ is the true shortest-path cost to the goal.

Key rule: Never overestimate the true cost.

Examples

- ✓ Manhattan distance $|x_n - x_G| + |y_n - y_G|$
- ✓ Straight-line (Euclidean) distance
- ✗ $2\times$ Manhattan distance

Why $2\times$ Manhattan fails:

$$h(n) = 2\big(|x_n - x_G| + |y_n - y_G|\big)$$

If the true cost is $h^*(n) = 3$, then:

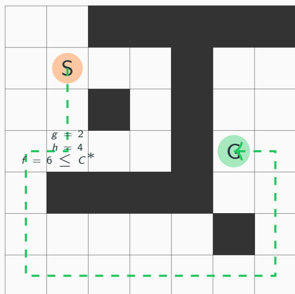
$$h(n) = 6 > 3 = h^*(n)$$

This overestimates the true cost $\Rightarrow$ not admissible.

Theorem: A* with an admissible heuristic is optimal.

Why A* Is Optimal: The Key Idea

Optimal path cost $C^* = 16$



Admissible heuristic

$$h(n) \leq h^*(n)$$

For any node n on an optimal path:

$$\begin{aligned} f(n) &= g(n) + h(n) \\ &\leq g(n) + h^*(n) \\ &= C^* \end{aligned}$$

Interpretation: Nodes on an optimal path never have $f(n) > C^*$, so A* will always expand them before any strictly worse alternative.

Key insight:

A* cannot "skip over" the optimal path when using an admissible heuristic.

Why A* Is Optimal: Proof Sketch

Assume, for contradiction, that A* returns a suboptimal solution.

- Let returned solution cost be C
- Suppose $C > C^*$ (a better path exists)

When A* terminates:

$$f(G) = g(G) = C$$

But along the optimal path:

- There exists a frontier node n
- From previous slide: $f(n) \leq C^*$

Contradiction:

$$f(n) \leq C^* < C = f(G)$$

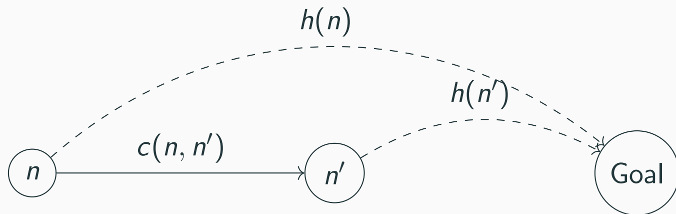
× **A* would expand n before the goal $\Rightarrow$ A* must return an optimal solution**

Consistent (Monotonic) Heuristics

Definition: $h(n)$ is **consistent** if:

$$h(n) \leq c(n, n') + h(n')$$

for every node n and successor n'



Triangle inequality! Consistent $\implies$ Admissible

Designing Good Heuristics

Goal: Heuristic should be:

- Admissible
- Close to actual cost
- Fast to compute

Technique: Relaxed Problems

8-puzzle example:

- h_1: Misplaced tiles
- h_2: Manhattan distance

Dominance:

If $h_2(n) \geq h_1(n)$ for all n, then h_2 **dominates** h_1

Better heuristic = fewer nodes

Heuristic	**Nodes**
h_1 (misplaced)	13,000
h_2 (Manhattan)	1,600

Concrete Example: 8-Puzzle with A*

Problem:

1	2	3
4	□	5
6	7	8

Start

1	2	3
4	5	6
7	8	□

Goal

Manhattan heuristic h_2:

$$h_2 = \sum_{\text{tiles } t \neq \square} \left(\left| r_t - r_t^{\text{goal}} \right| + \left| c_t - c_t^{\text{goal}} \right| \right)$$

Distances for this state (ignore blank):

- Tile 5: $(2,3) \to (2,2)$ $|2-2| + |3-2| = 1$
- Tile 6: $(3,1) \to (2,3)$ $|3-2| + |1-3| = 3$
- Tile 7: $(3,2) \to (3,1)$ $|3-3| + |2-1| = 1$
- Tile 8: $(3,3) \to (3,2)$ $|3-3| + |3-2| = 1$

$$h_2 = 1 + 3 + 1 + 1 = \boxed{6}$$

Initial cost: $g = 0, \ f = g + h_2 = 6$

A* Search Properties

Property	**A* with admissible h**
Complete?	Yes (finite graphs)
Optimal?	Yes
Time	$O(b^d)$ (exponential)
Space	$O(b^d)$ (exponential)

Optimally efficient: No algorithm with same info can do better!

Main limitation: Space complexity

- Keeps all nodes in memory
- Variants: IDA*, RBFS, SMA* reduce space

IDA*: Iterative Deepening A*

Motivation: A* is optimal but uses exponential memory.

Key idea

- Perform a **depth-first search**
- Use an f-cost threshold:

$$f(n) = g(n) + h(n)$$

- Only expand nodes with $f(n) \leq$ threshold
- Increase the threshold iteratively

Properties

- **Optimal** with admissible heuristic
- **Very low memory usage**
- Re-expands nodes many times (slower than A*)

Typical use: Large state spaces where memory is the main constraint (e.g., 15-puzzle).

SMA*: Simplified Memory-Bounded A*

Motivation: Run A* under a fixed memory limit.

Key idea

- Behaves like A* using a priority queue
- When memory is full:
  - Remove the node with the **worst** f-value
  - Back up its cost to its parent
- Regenerates nodes if needed later

Properties

- **Optimal** if enough memory is available
- **Complete** under reasonable assumptions
- More complex than A* or IDA*

Typical use: Situations with strict memory limits but where A*-like behavior is desired.

Comparing A*, IDA*, and SMA*

Algorithm	Optimal	Memory Use	Time Cost
A*	Yes (admissible h)	Very high	Low
IDA*	Yes (admissible h)	Very low	High (re-expansion)
SMA*	Yes (if memory allows)	Bounded	Medium

Key takeaway

- A* is fastest but memory-hungry
- IDA* trades time for memory
- SMA* trades complexity for bounded memory

Summary

Informed search uses heuristics

Greedy Best-First:

- Fast but not optimal
- Uses only $h(n)$

A* Search:

- Optimal and complete
- Uses $f(n) = g(n) + h(n)$
- Requires admissible h

Heuristic design matters!

- Better heuristics $\rightarrow$ fewer nodes
- Must stay admissible
- Relaxed problems help

Space is main limitation

- Memory-bounded variants
- Trade-offs exist

Next week: Advanced search