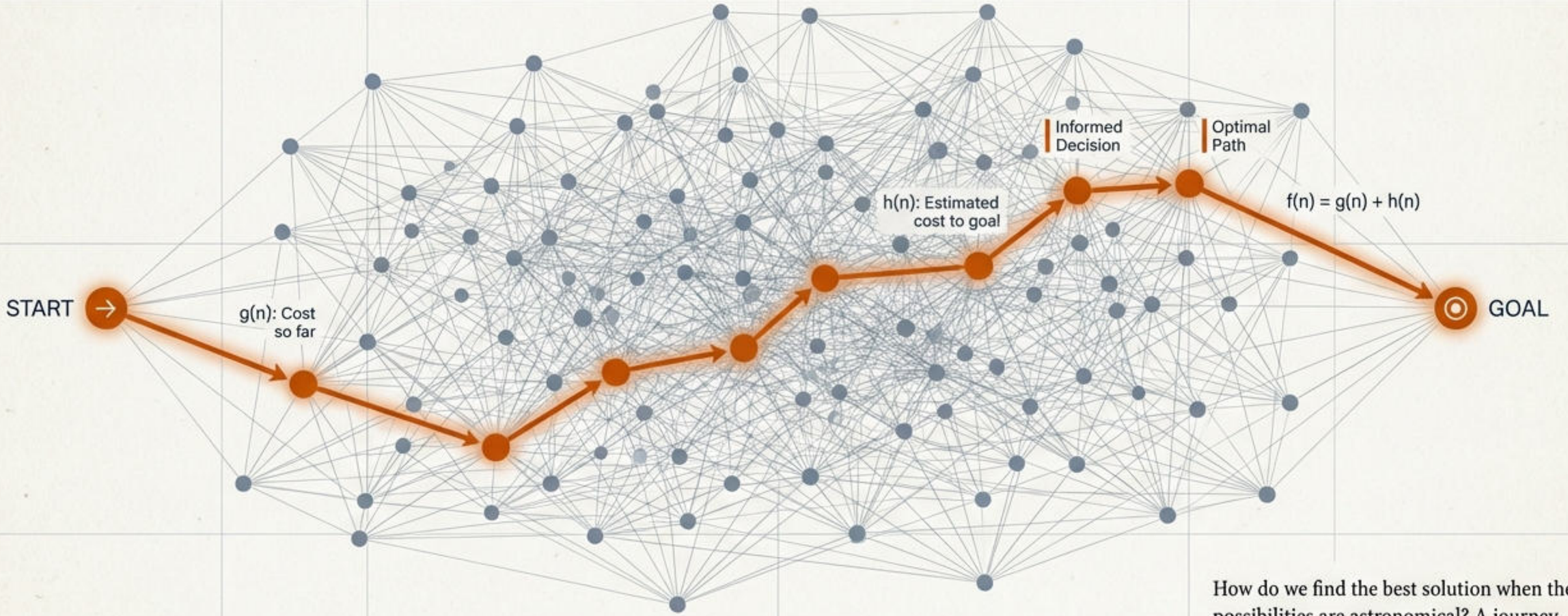
Heuristic Search: Navigating Complexity

From Blind Wandering to Intelligent Planning

AI Lecture Series 3.2 | Based on lectures by Wlodzislaw Duch



How do we find the best solution when the possibilities are astronomical? A journey from brute force to the elegance of A*.

The Villain: Combinatorial Explosion

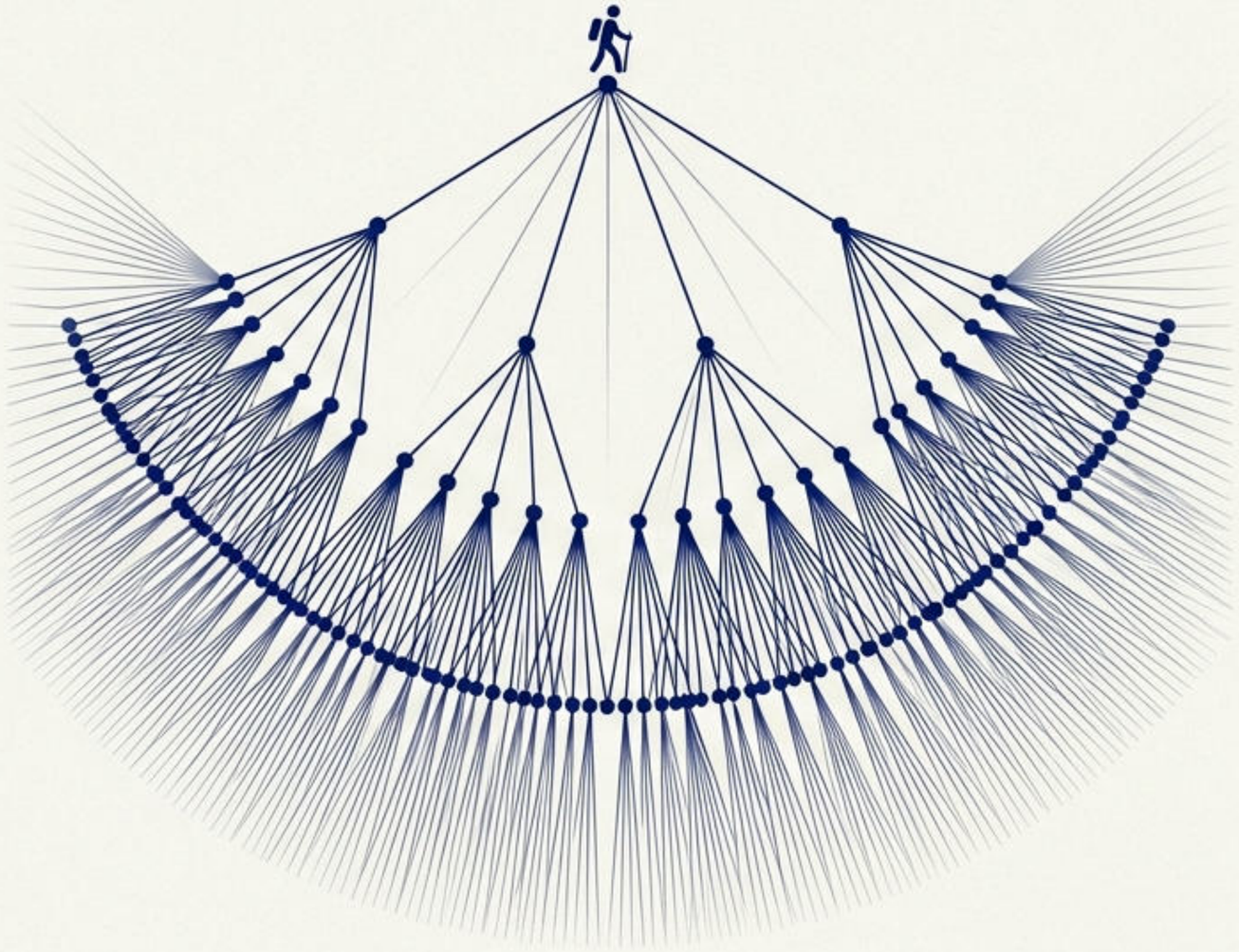
Why “Blind” Search Fails

In an uniformed search, the algorithm cannot distinguish “hot” from “cold.” It wanders aimlessly through every possibility.

The Traveling Salesman Problem (TSP):
For N cities, routes = N! (factorial).

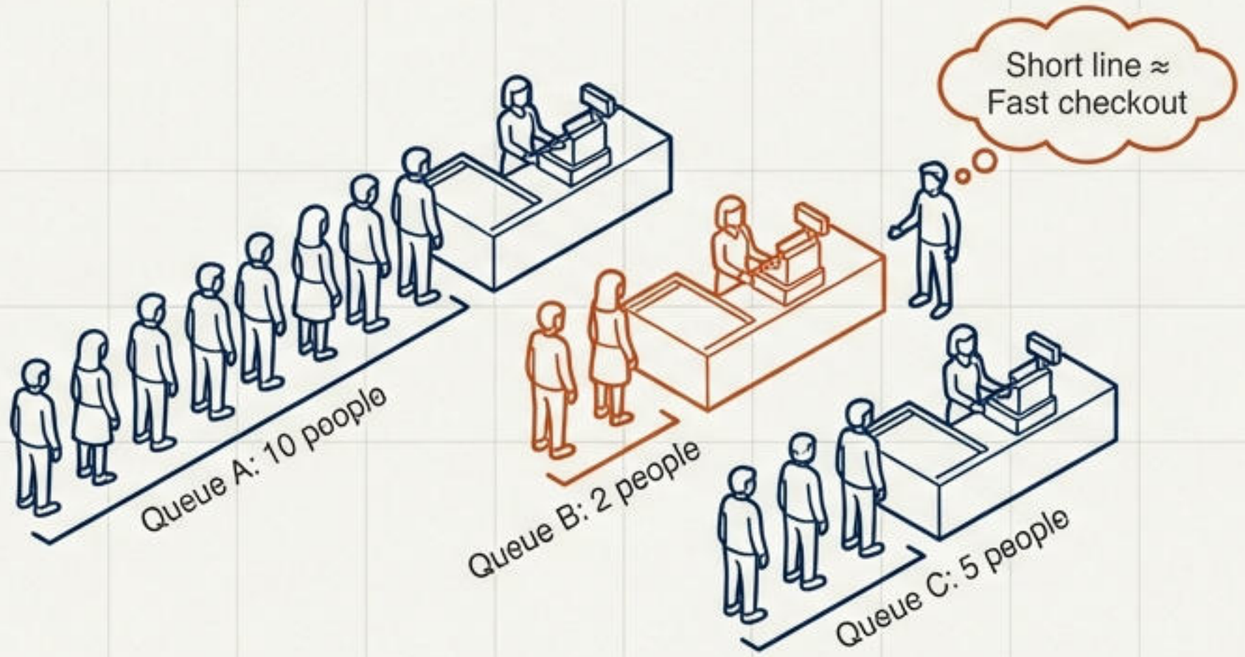
Example: 20 cities = 2,432,902,008,176,640,000 possibilities.

Conclusion: Brute force is mathematically impossible for complex problems.



The Spark: What is a Heuristic?

Definition: A function h(s) that estimates the cost from state s to the goal. It maps states to real numbers to evaluate relative benefit.



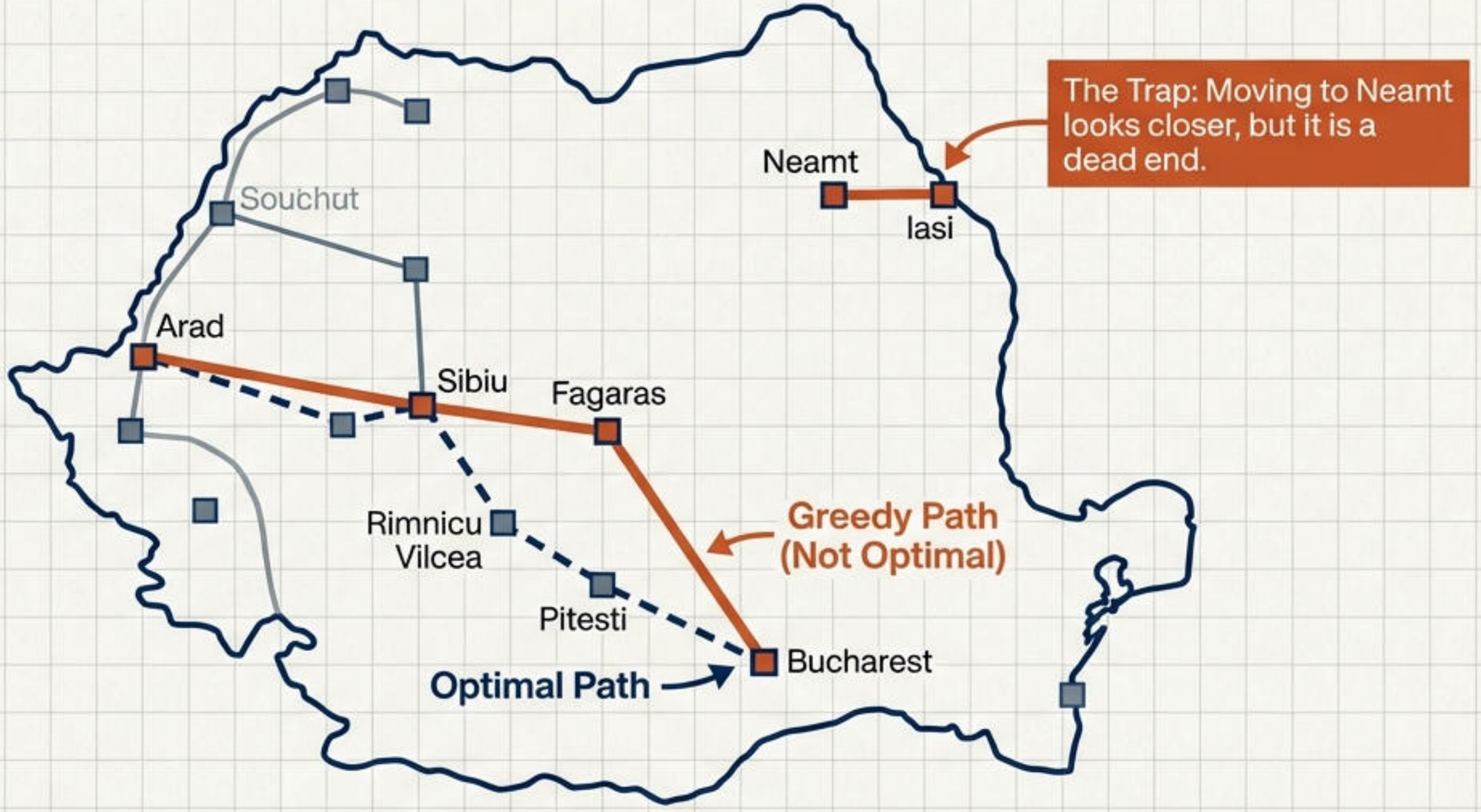
The Intuition: A “rule of thumb.” It doesn’t guarantee perfection, but it speeds up decisions.

The Caveat: Heuristics can be wrong. The shortest line might have a customer with a full cart (hidden cost), or the register might be cash-only. It is an *estimate*, not a fact.

Strategy 1: The Impatient Traveler (Greedy Search)

- **Logic:** Best-First Search (BestFS). Always choose the node that *looks* closest to the goal right now.
- **The Math:** Minimize h(n) (estimated distance).
- **Behavior:** Like driving toward a mountain peak because you can see it, ignoring road signs that might indicate a necessary detour.
- **Metaphor:** Impatience over planning.

The Flaw of Greed: Fast, But Not Optimal

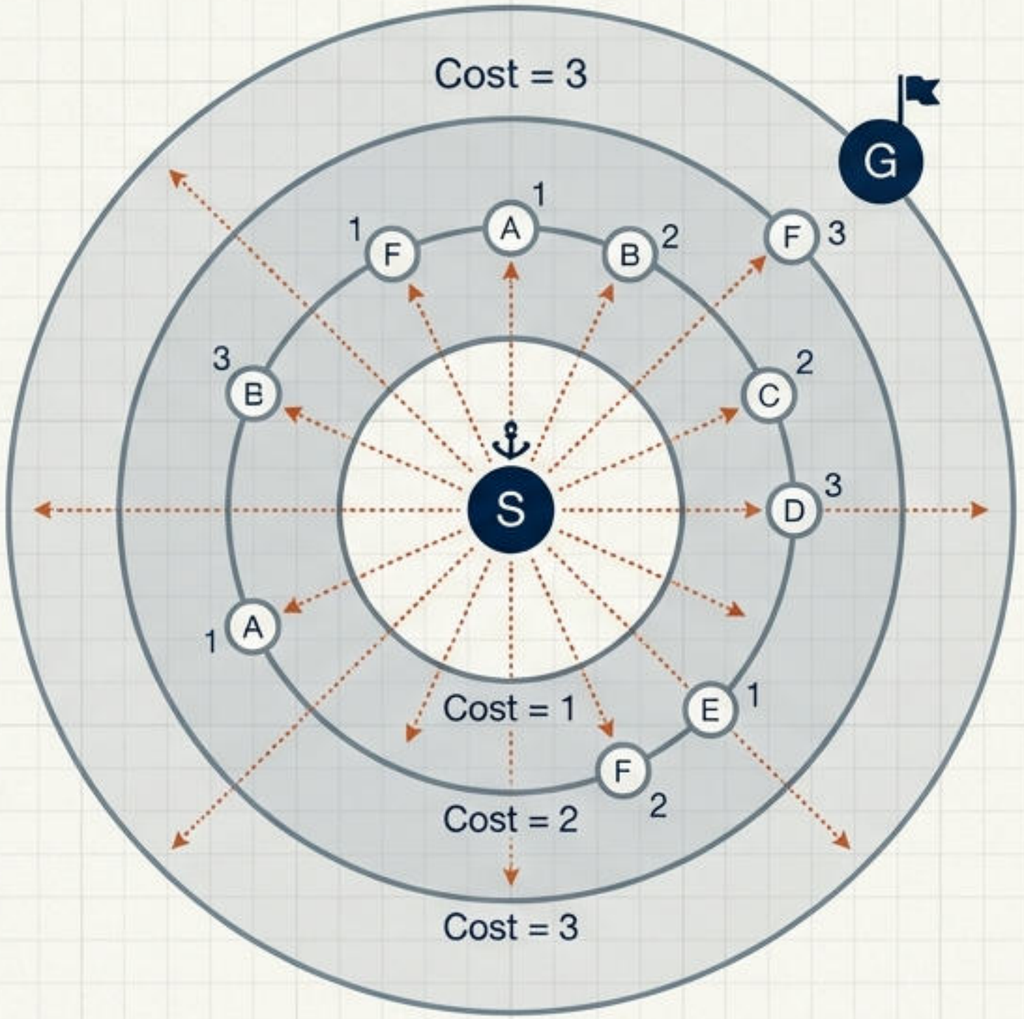


The Result: Greedy search creates a path that is often longer than necessary (suboptimal). In the worst case (Iasi to Fagaras), it rushes into dead ends, forcing costly backtracking.

Verdict:
Not Complete. Not Optimal.

Strategy 2: The Cautious Accountant (Uniform Cost Search)

- **Logic:**
  Ignore predictions. Focus only on spent budget.
- **The Math:**
  Minimize g(n) (actual cost from start).



- **Pros:**
  Guaranteed Optimal & Complete.
- **Cons:**
  Painfully slow. Explores irrelevant directions just because they are “cheap”.
- **Complexity:**
  Time and memory can be high: $O(b^{(1 + \text{floor}(C^*/\text{epsilon}))})$.

Interlude: The Value of Memory (Dynamic Programming)

The Principle: “Those who cannot remember the past are condemned to repeat it” (Bellman, 1957).

Methodology: Break complex problems into sub-problems, solve them once, and store the result (Memoization).

Relevance to Search: This logic underpins the **calculation of g(n)**—ensuring we know the true cost of the path traveled so far without recalculation.

	1	2	3	4	5
1	∞	4	6	8	11
2	3	∞	2	4	7
3	8	12	∞	2	5
4	10	14	4	∞	7
5	3	7	9	6	∞

Calculation:
D[1,3] = D[1,2] + D[2,3] = 4 + 2 = 6

The Synthesis: A* Search

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

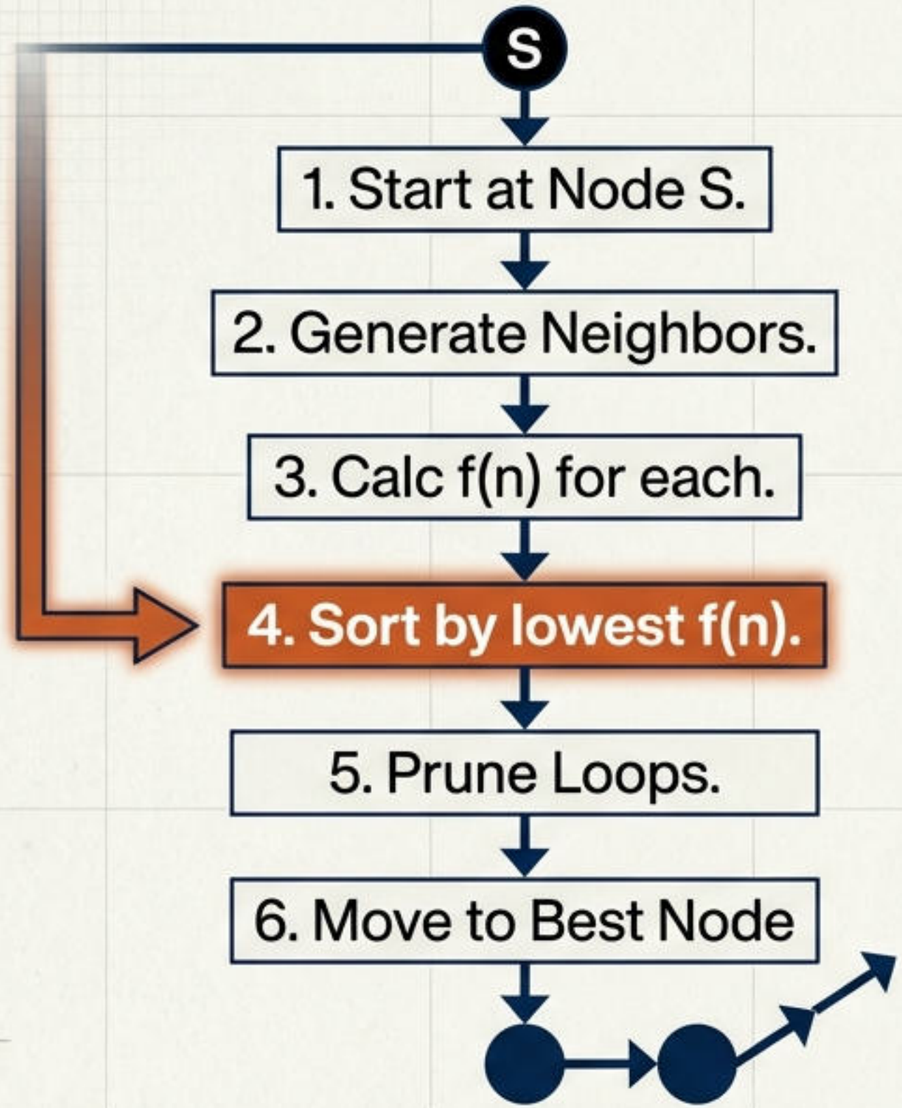
History (Cost so far).
From Uniform Cost Search.

Prediction (Heuristic).
From Greedy Search.

Estimated Total Cost.

A* prioritizes the most promising path (h) but uses historical data (g) to ensure optimality. It is the best of both worlds.

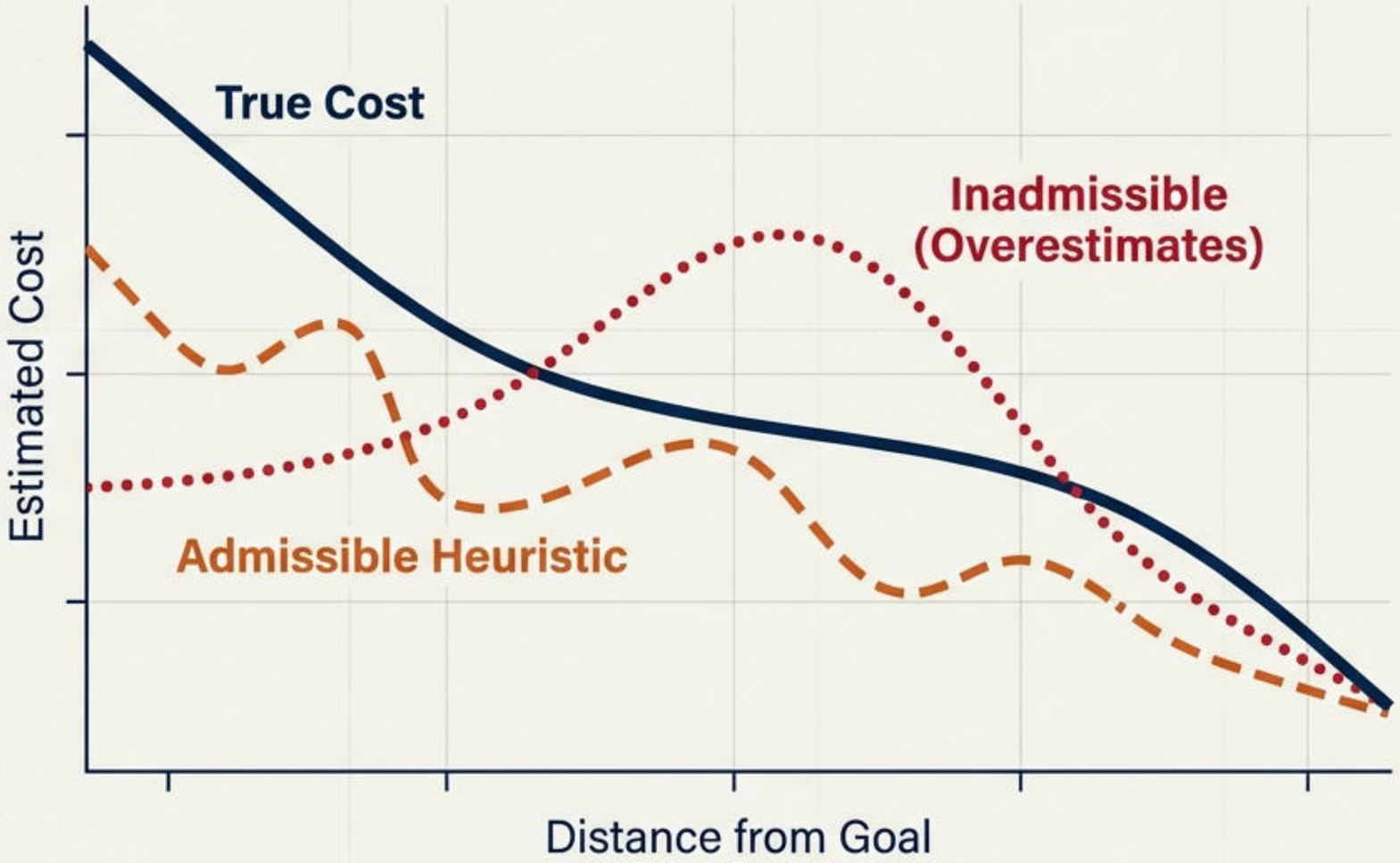
A* in Action: Intelligent Navigation



Why It Works:

- Unlike Greedy, A* won't be fooled by a short-term gain if the total cost (g+h) becomes too high.
- Unlike UCS, it pushes toward the goal rather than expanding everywhere.
- **Monotonicity:** Since it always picks the cheapest path to a node, **no other** path can lower the cost later.

The Condition for Perfection: Admissibility



The Rule: A* is Optimal only if h(n) is **Admissible**.

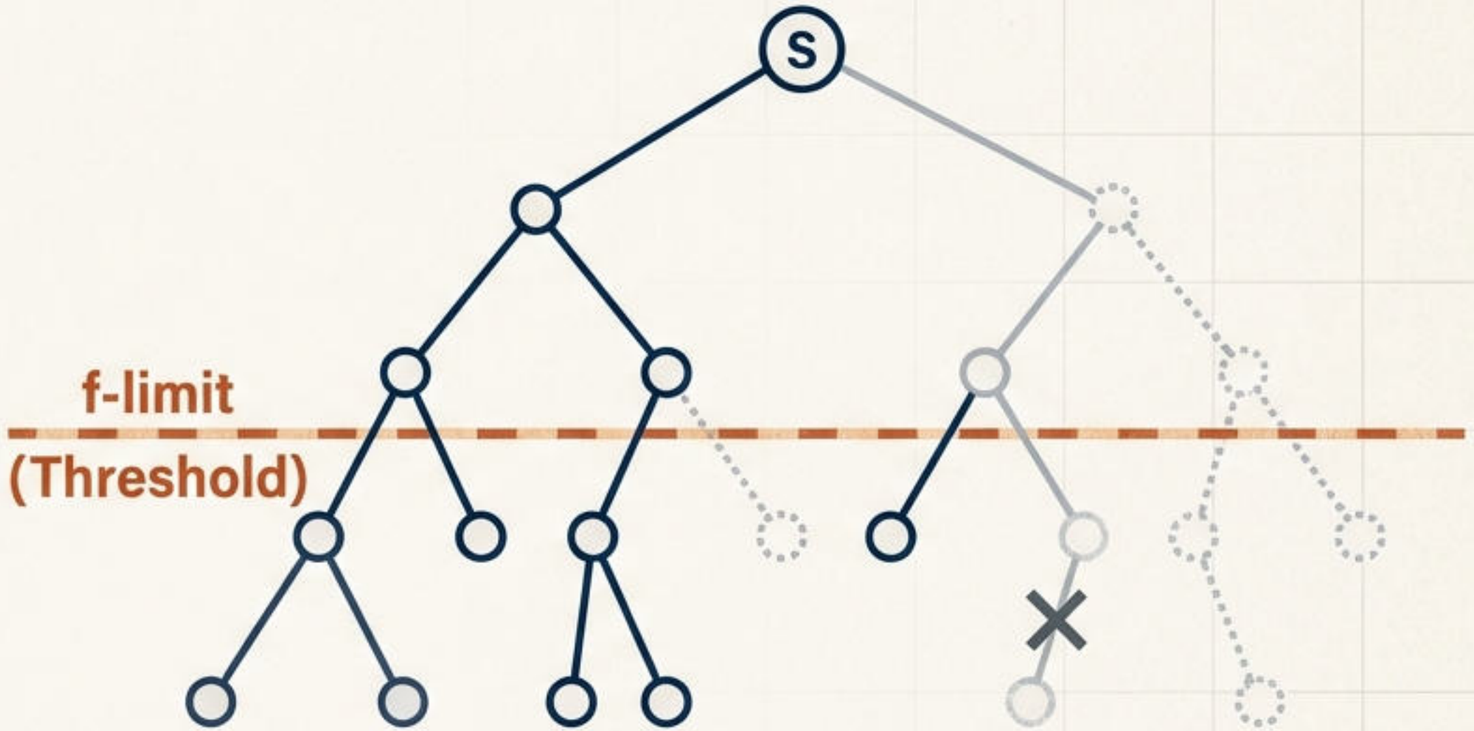
Definition: The heuristic must never overestimate the cost to the goal. It must be optimistic.

Philosophy: It is better to think the goal is closer than it is, than to mistakenly discard a good path because you thought it was too far.

Handling Constraints: IDA* (Iterative Deepening A*)

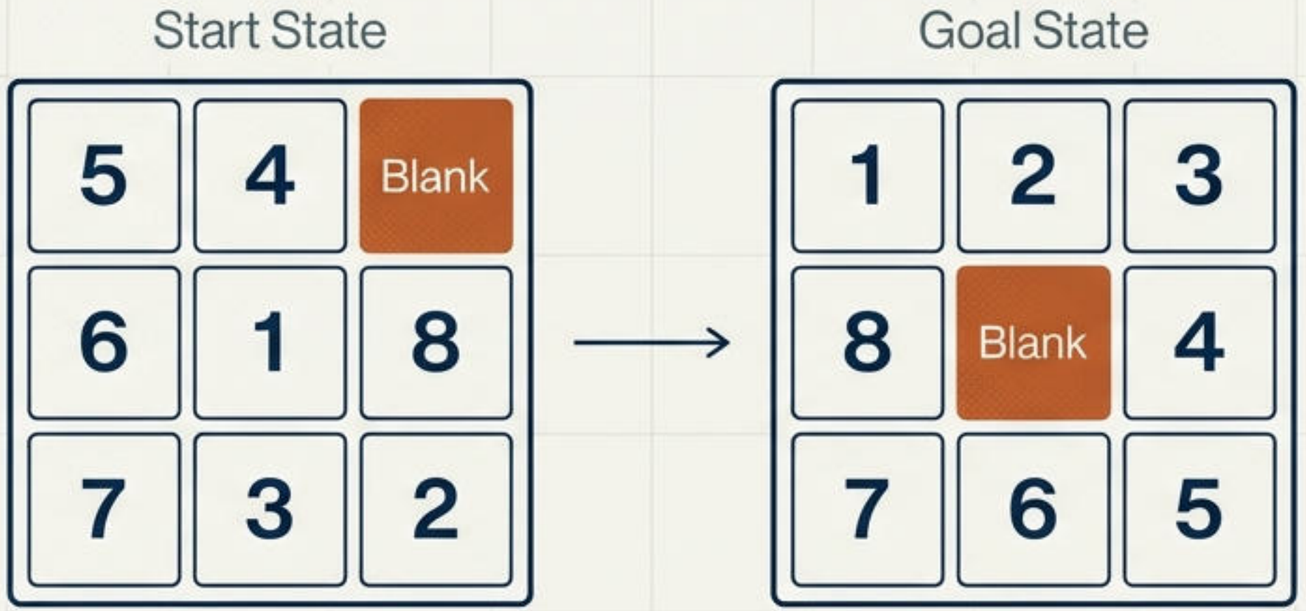
The Bottleneck: A* keeps all generated nodes in memory. For huge graphs, memory exhaustion occurs quickly (Complexity $O(b^d)$).

The Solution: IDA* uses Depth-First Search logic (linear memory) but cuts off branches that exceed a specific cost threshold.



Pruning branches that are too expensive, then increasing the limit iteratively.

Case Study: The 8-Puzzle



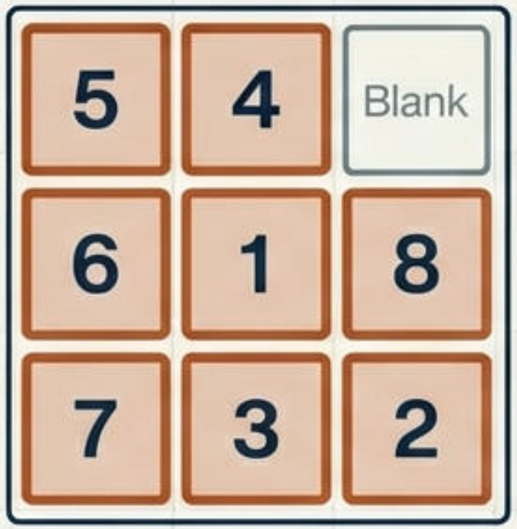
The Challenge: Sliding tiles to match the goal pattern.

Complexity: 181,440 possible states for the 8-puzzle. For the 15-puzzle, it jumps to 653 billion.

The Goal: Find a heuristic to reduce the search from thousands of states to fewer than 50.

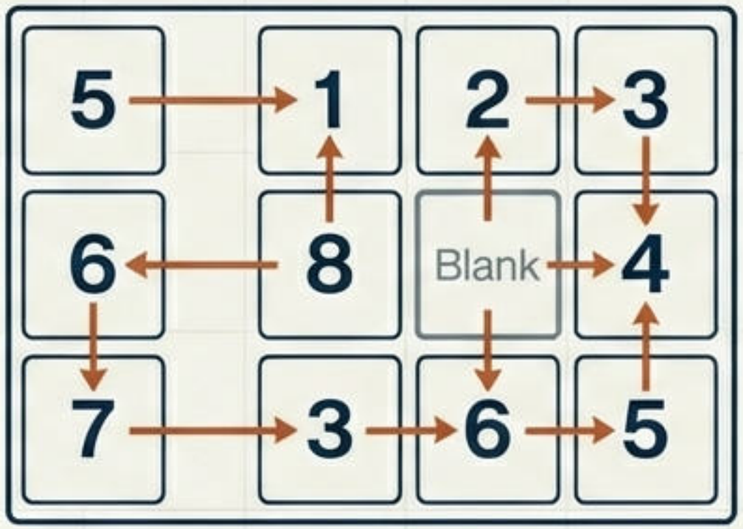
The Quality of Insight: Comparing Heuristics

h1: Misplaced Tiles



Count of tiles not in their goal position.
h1 = 8. (Weak Guidance).

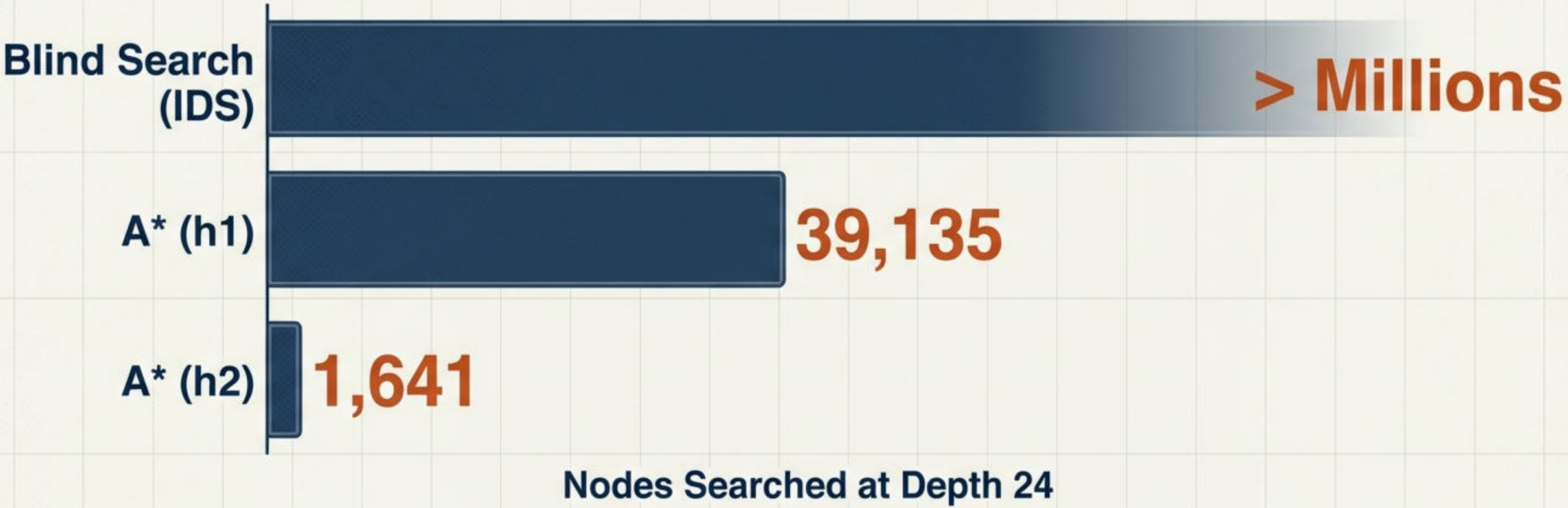
h2: Manhattan Distance



Sum of distances each tile must move.
h2 = 18. (Strong Guidance).

Both are admissible, but h2 is closer to the true cost, making it more efficient.

The Efficiency Gap: ‘Smart’ Beats ‘Hard’



Effective Branching Factor: h2 reduces the branching factor to 1.26, meaning the tree grows much slower.

Key Insight: A better heuristic drastically cuts computational cost.

Summary: The Art of Search

1. **Blind Search:** “Wanders aimlessly.”

2. **Greedy Search:** “Fast but risky (h only).”

3. **A* Search:** “The Enlightened Balance.”
   - Complete? Yes.
   - Optimal? Yes.
   - Efficient? Yes.

The Equation: f(n) = g(n) + h(n) remains the gold standard.

The Future: Large Language Models are now providing even better heuristics for complex search processes.