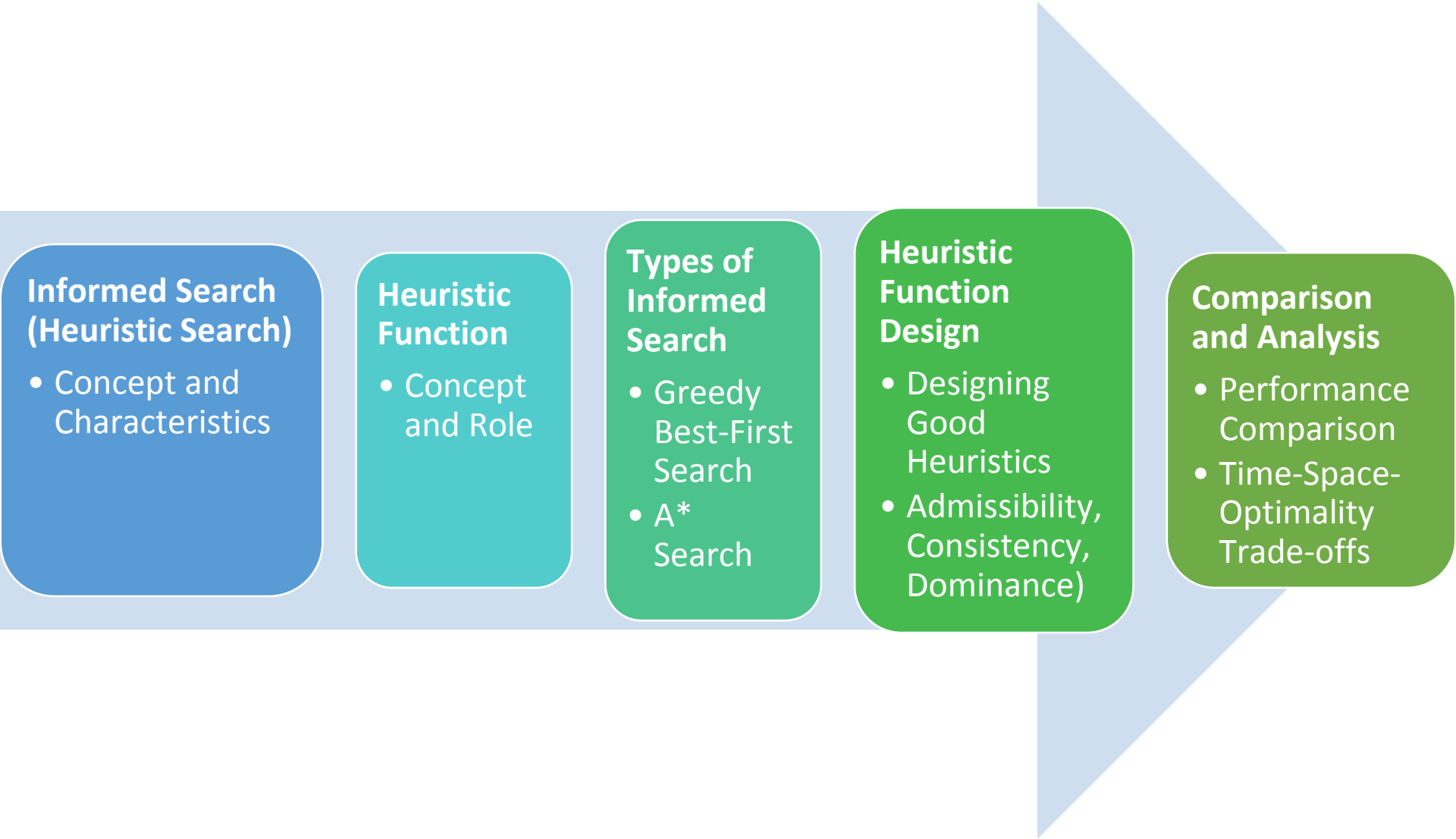
Artificial Intelligence (CS280)

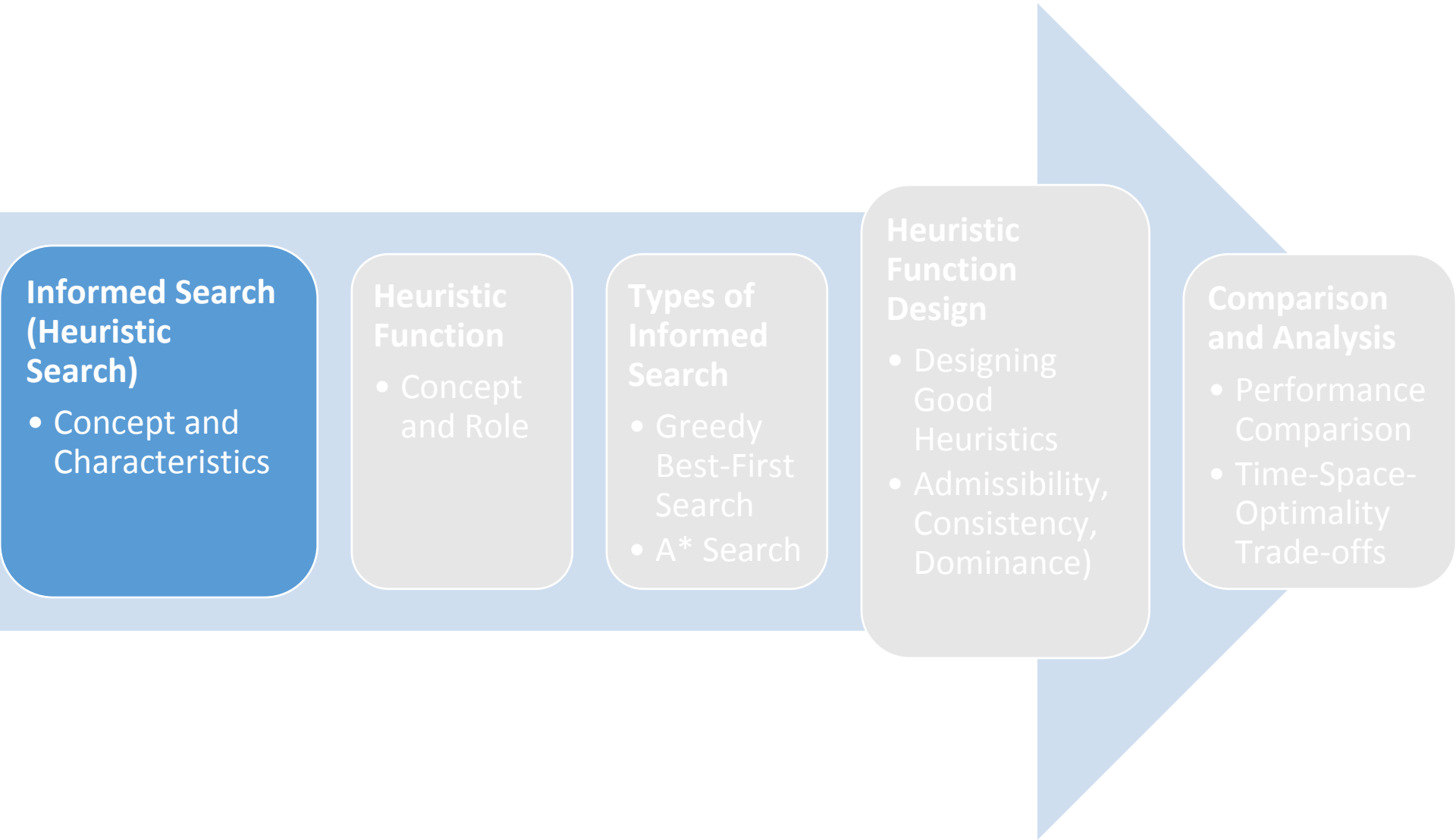
Noushin Saba

Lecture 5
Heuristic Search: Informed Search Techniques and Frameworks

Outline


Informed Search (Heuristic Search)
• Concept and Characteristics
Heuristic Function
• Concept and Role
Types of Informed Search
• Greedy Best-First Search
• A* Search
Heuristic Function Design
• Designing Good Heuristics
• Admissibility, Consistency, Dominance)
Comparison and Analysis
• Performance Comparison
• Time-Space-Optimality Trade-offs

Outline


Informed Search (Heuristic Search)
• Concept and Characteristics
Heuristic Function
• Concept and Role
Types of Informed Search
• Greedy Best-First Search
• A* Search
Heuristic Function Design
• Designing Good Heuristics
• Admissibility, Consistency, Dominance)
Comparison and Analysis
• Performance Comparison
• Time-Space-Optimality Trade-offs

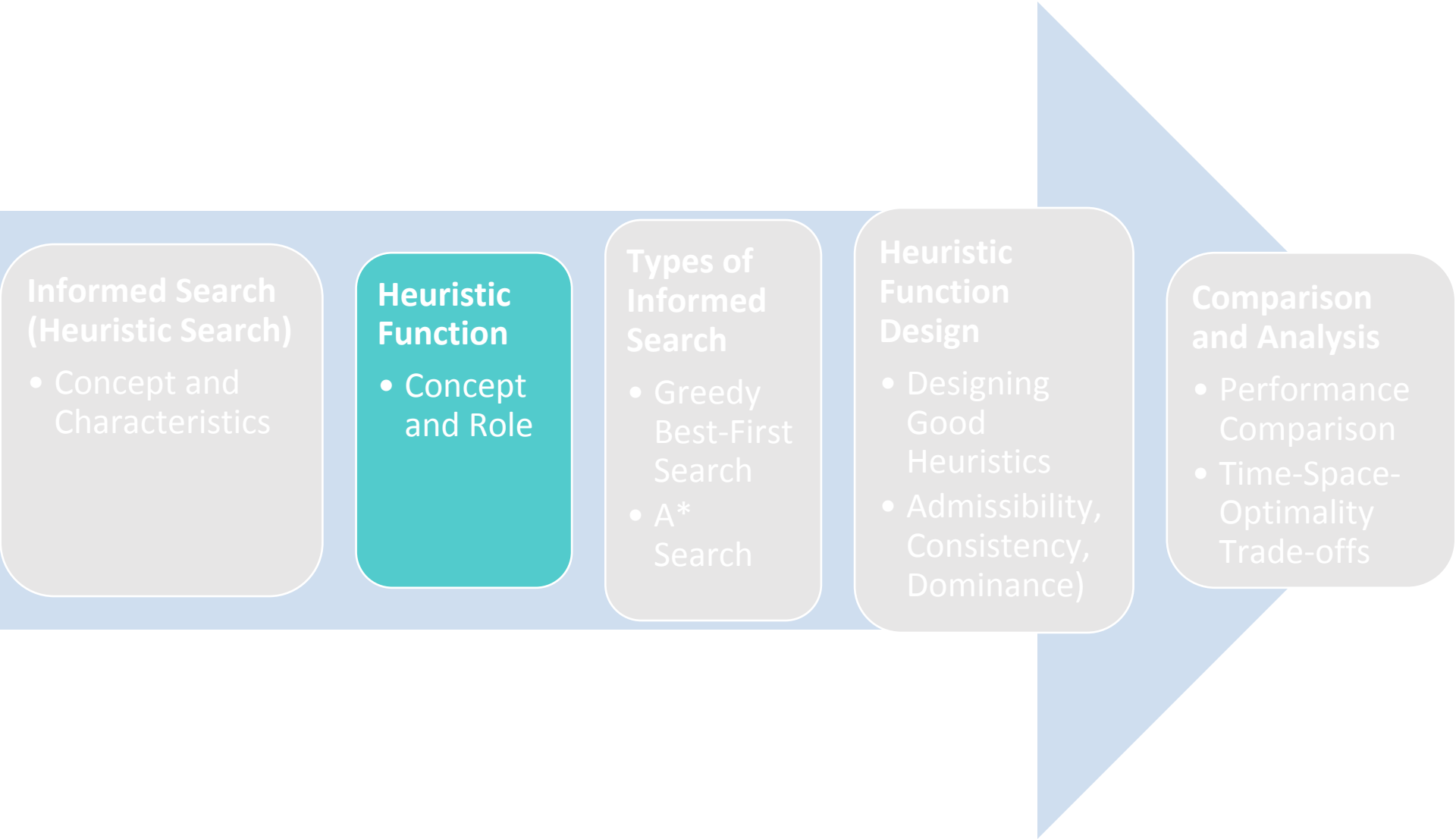
Informed search

- AI search problems typically have a very large search space. We would like to improve efficiency by **expanding as few nodes as possible.**

- The agent can use **additional information** in the form of "hints" about how promising different states/nodes are to lead to the goal. These hints are derived from
  - information the agent has (e.g., a map) or
  - percepts coming from a sensor (e.g., a GPS sensor).

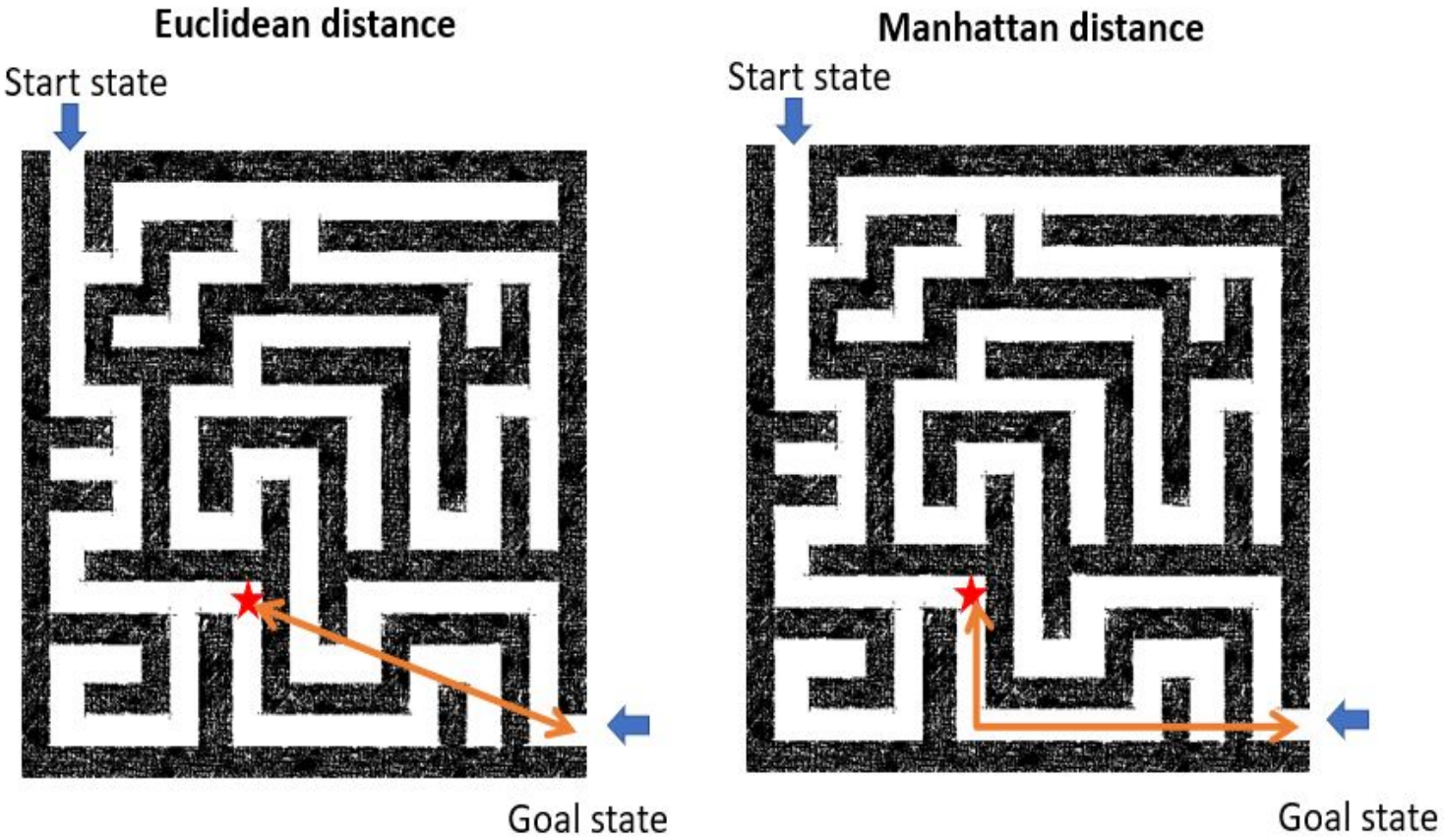
- **Goal:** Find the least-cost path efficiently by using *knowledge* to guide search.

Outline


Informed Search (Heuristic Search)
• Concept and Characteristics
Heuristic Function
• Concept and Role
Types of Informed Search
• Greedy Best-First Search
• A* Search
Heuristic Function Design
• Designing Good Heuristics
• Admissibility, Consistency, Dominance)
Comparison and Analysis
• Performance Comparison
• Time-Space-Optimality Trade-offs

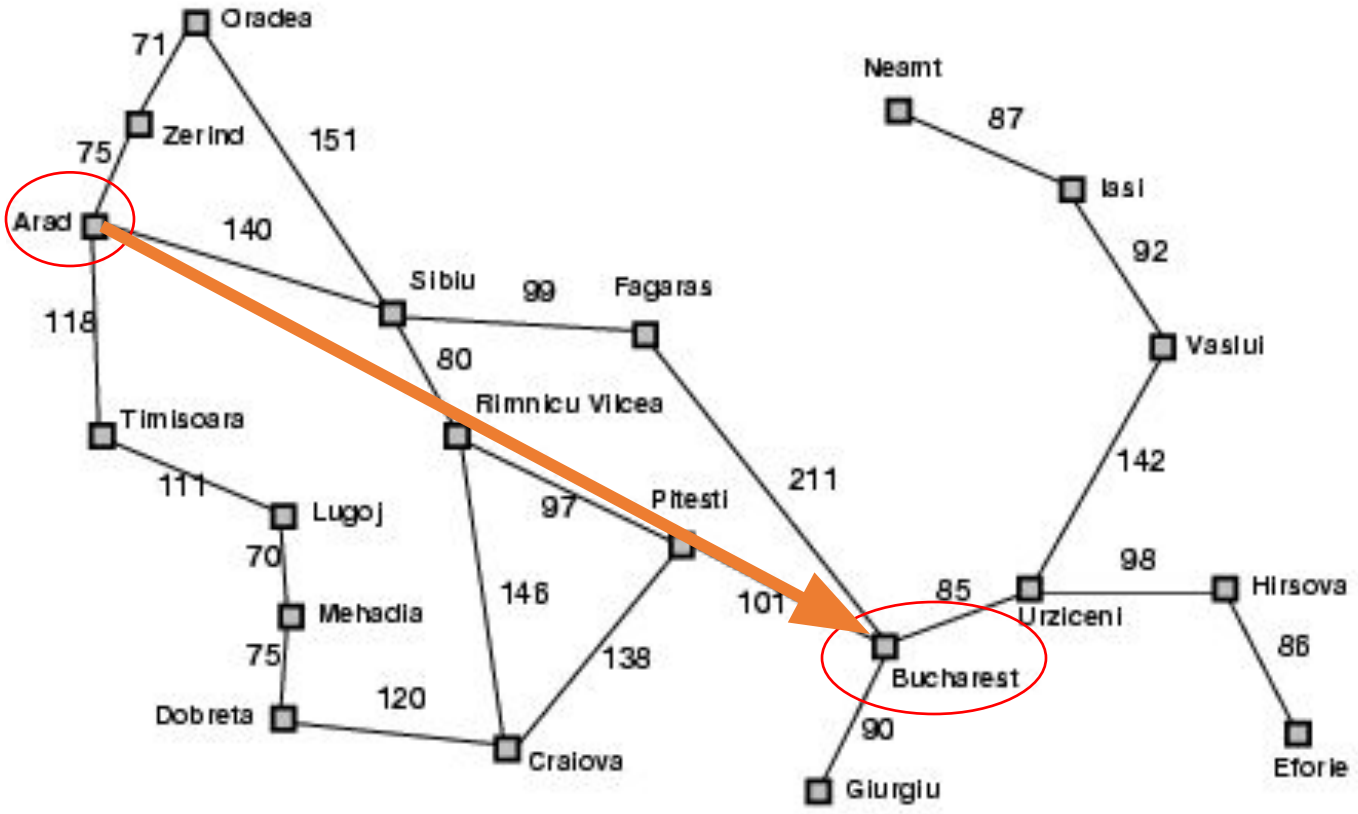
Heuristic function

- h(n)=estimated cost from node n to the goal.
- A heuristic is **problem-specific knowledge** added to guide search.
- **Smaller h(n)** → closer to the goal.
- Must be **computationally simple** yet informative.



Heuristic for the Romania problem

Drive from Arad to Bucharest using a table with straight-line distances.



Straight–line distance to Bucharest	$h(n)$
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Understanding Heuristics — Intuition

- A **heuristic** is a *rule of thumb* or *educated guess* to guide search.
- It helps estimate **how far or costly** it is to reach the goal from the current state.
- Not guaranteed to be correct — but helps **reduce exploration time**.
- *Example*
  When driving to a city, your **straight-line distance** gives a *good guess* of how far the city is, even though the road may curve.

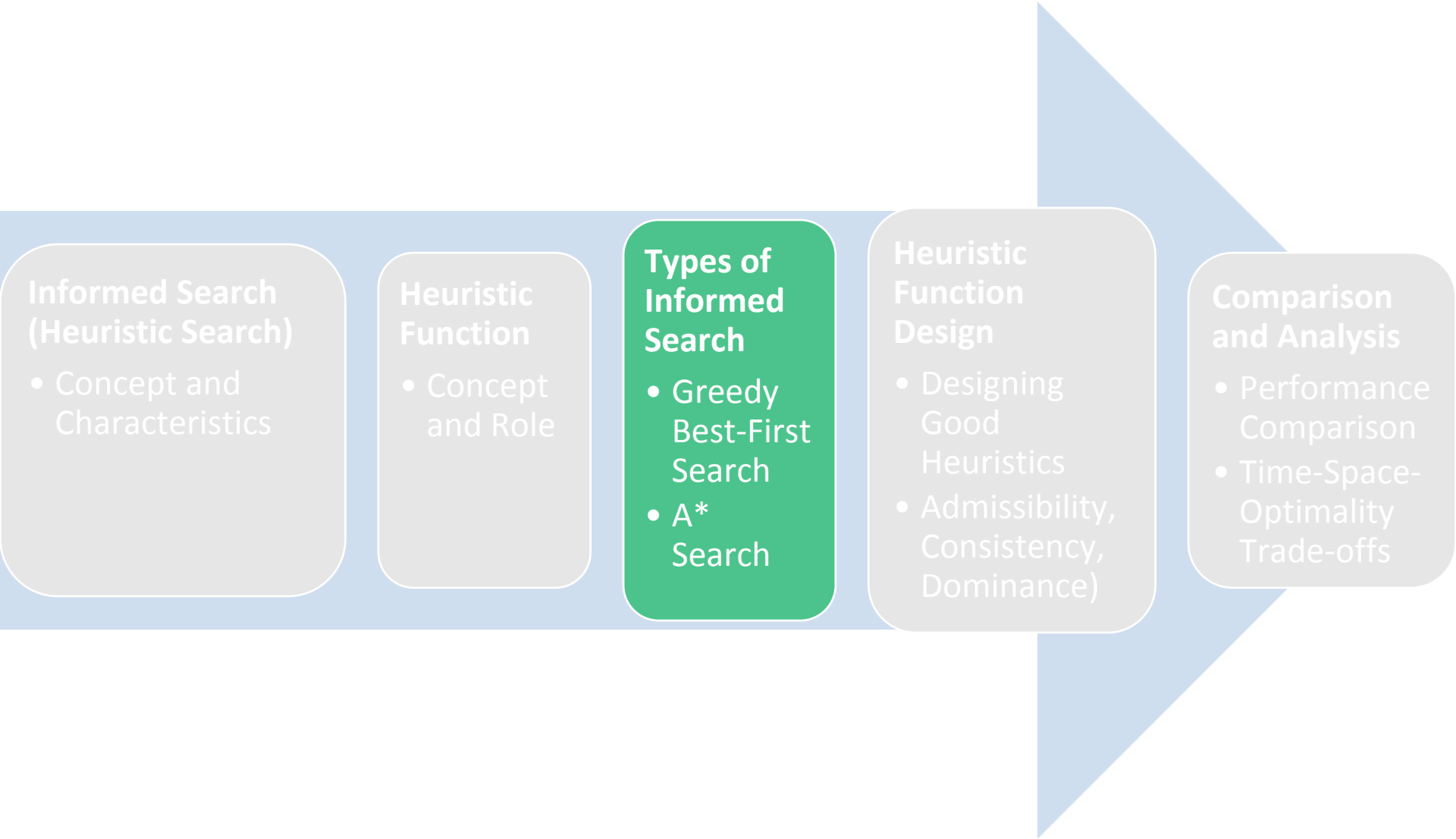
Role of Heuristic Function (h(n))

- A heuristic provides **problem-specific knowledge** that is not part of the problem definition.
- It acts as **a measure of goal proximity** — a smaller h(n) means *closer to the goal*.
- Used by informed algorithms (e.g., A*, Greedy) to decide **which node to expand next**.
- Formula → $h(n) =$ *estimated cost from node* n *to the nearest goal.*

Characteristics of a Good Heuristic

- A good heuristic should:
- **Approximate the true cost** to goal (not random).
- **Be easy to compute.**
- **Be admissible** — never overestimates true cost.
- **Be consistent (monotonic)** — cost estimates do not decrease along a path.
- **Reduce search effort** — fewer nodes expanded.
- *Trade-off*
  - Simplicity vs. Accuracy — complex heuristics may take longer to compute.

Outline


Informed Search (Heuristic Search)
• Concept and Characteristics
Heuristic Function
• Concept and Role
Types of Informed Search
• Greedy Best-First Search
• A* Search
Heuristic Function Design
• Designing Good Heuristics
• Admissibility, Consistency, Dominance)
Comparison and Analysis
• Performance Comparison
• Time-Space-Optimality Trade-offs

Types of Informed Search Algorithms

- Heuristic (Informed) Search algorithms use the heuristic function $h(n)$ to guide the search process.
- They differ in how they combine the actual cost so far (g(n)) and estimated cost to goal (h(n)).
  - Greedy Best-First Search
  - A* Search

Greedy best-first

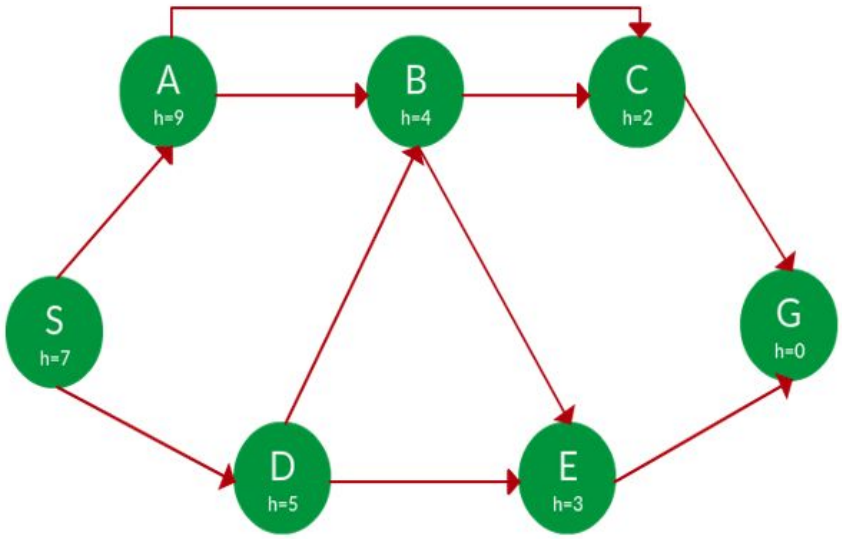
- In greedy search, we expand the node closest to the goal node. The "**closeness**" is estimated by a heuristic h(x) .

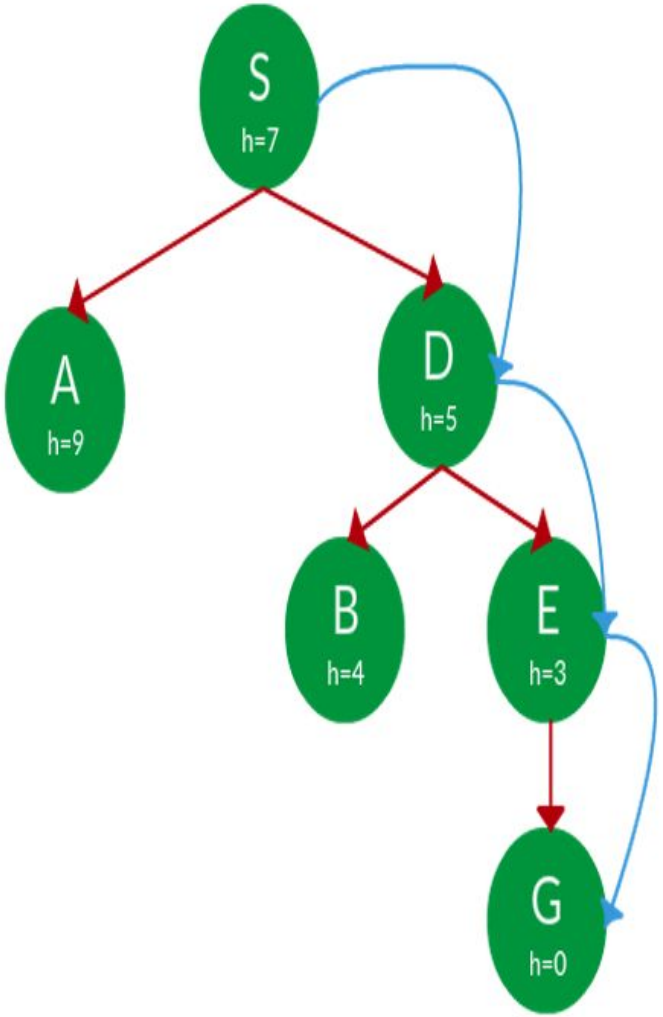
- **Strategy**: Expand the node closest to the goal state, i.e. expand the node with lower h value.

- Evaluation function: $f(n) = h(n)$

Greedy best-first Example

Find the path from S to G using greedy search.

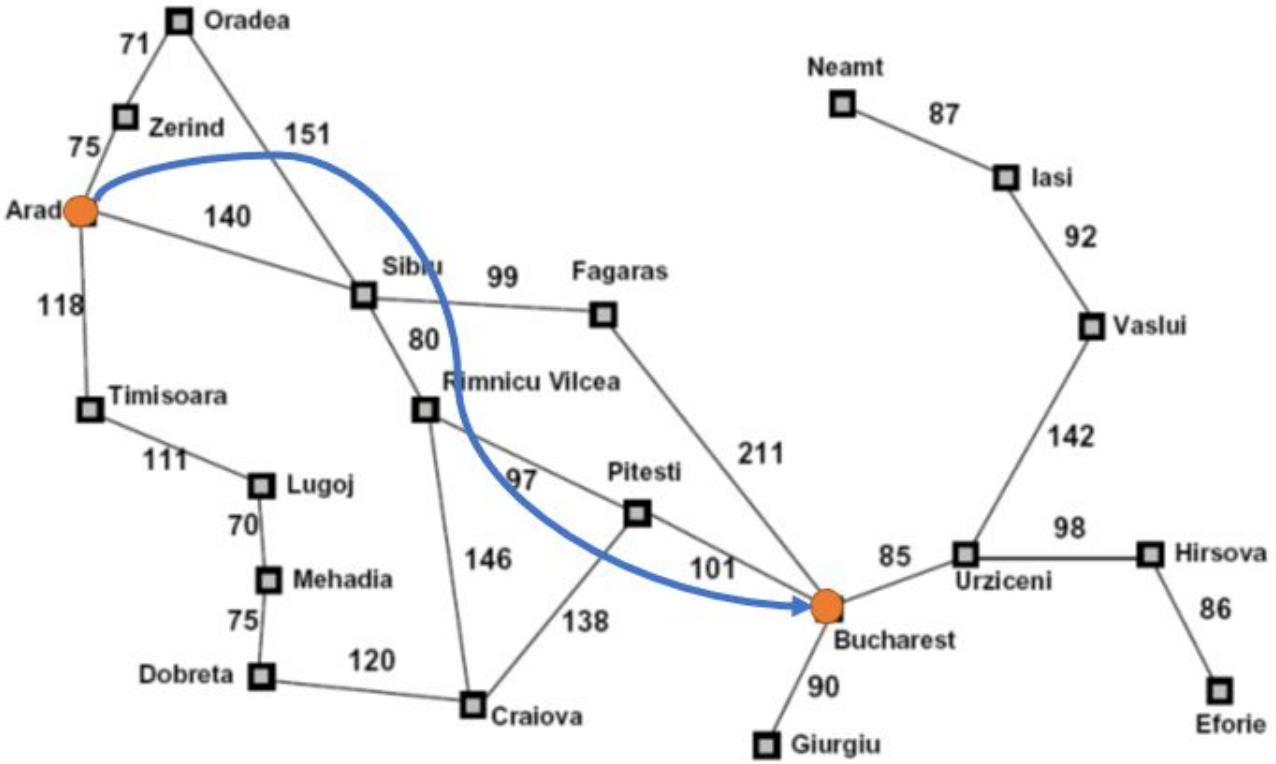




Starting from **S**, we can traverse to A(h=9) or D(h=5). We choose D, as it has the lower heuristic cost. Now from D, we can move to B(h=4) or E(h=3). We choose E with lower heuristic cost. Finally, from E, we go to G(h=0).

Path: S -> D -> E -> G

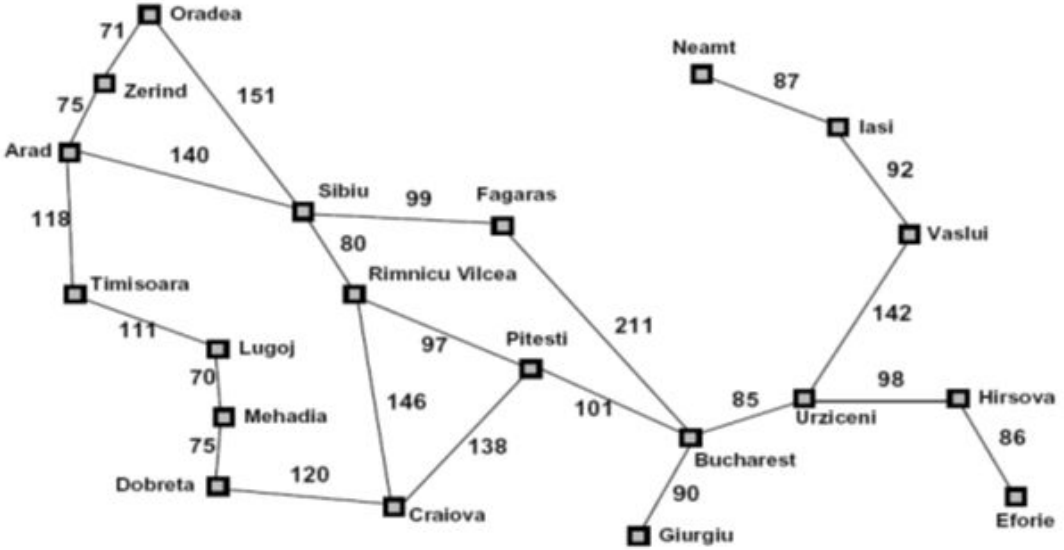
Travel Sales Man Problem using Greedy Search

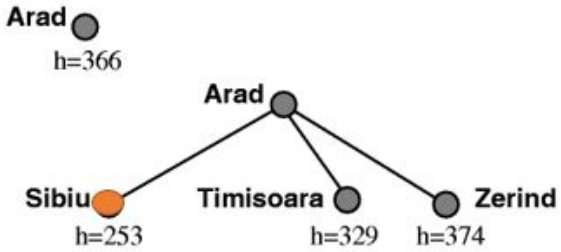


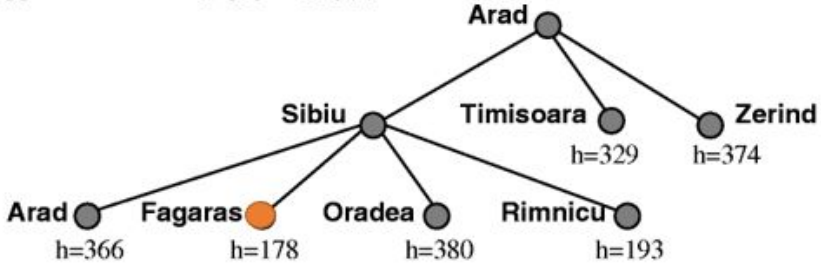
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

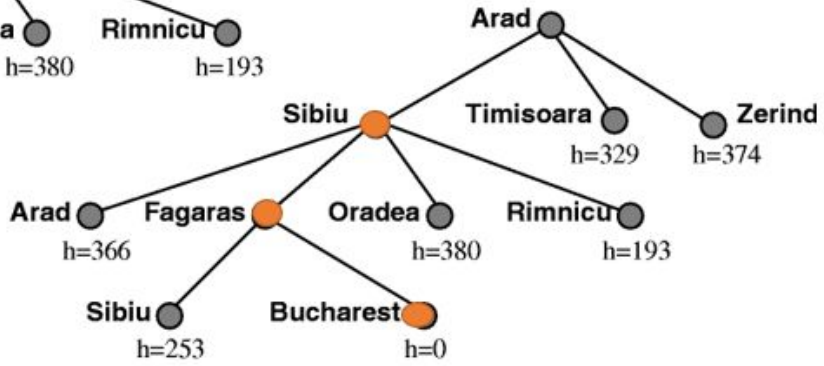
h(x)

Greedy Search……

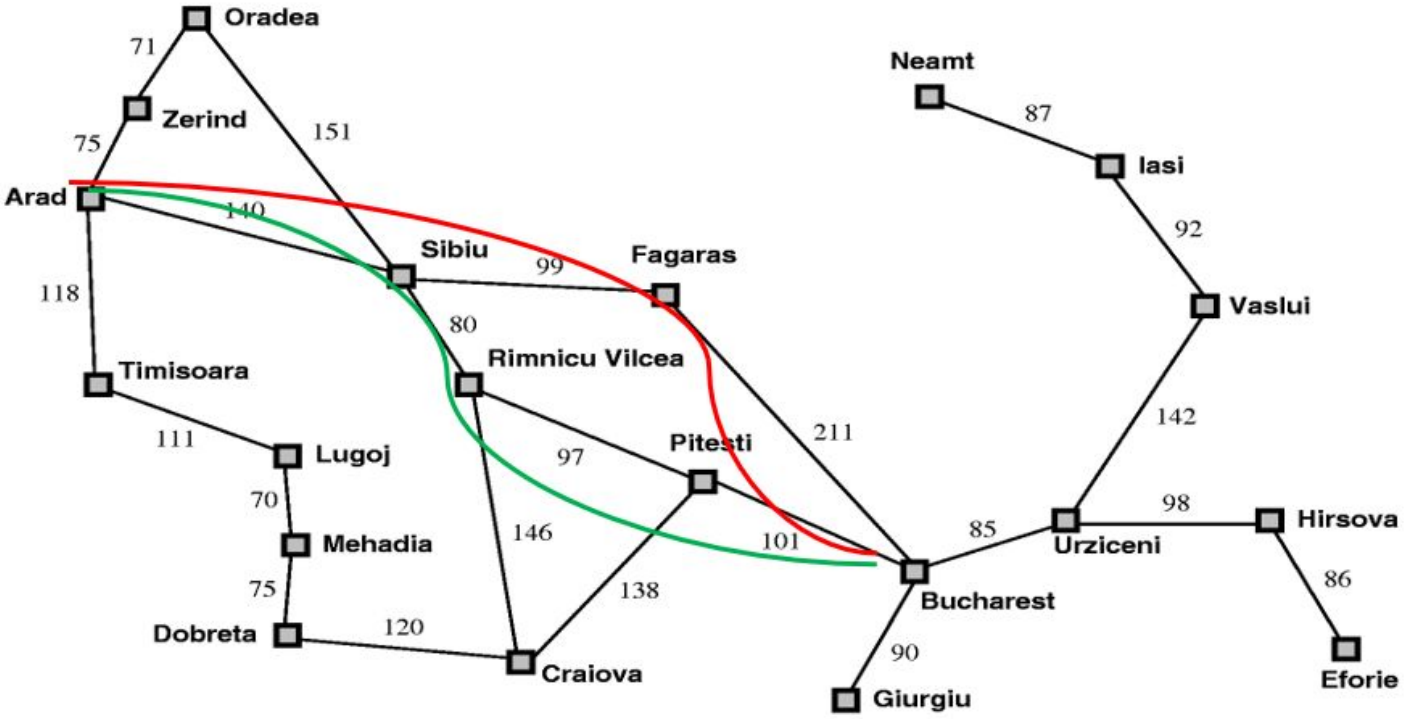








- **What can go wrong?**
  - **Not optimal**



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Red Path (GBF)=431

Green Path (UCS)= 418

- Greedy search leads to a minimal cost search because no node off the solution path is expanded.
- However, it does not find the optimal path:
  - The path it found via Sibiu and Fagaras to Bucharest is 13 miles longer than the path through Pimnicu Vilcea and Pitesti.

Implementation of greedy best-first search

Heuristic $h(n)$ so we expand the node with the lowest estimated cost

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

The order for expanding the frontier is determined by *f(n)*

This check is the different to BFS! It visits a node again if it can be reached by a better (cheaper) path.

See BFS for function EXPAND.

Properties of greedy best-first search

- **Complete?**

  Yes – Best-first search if complete in finite spaces.

- **Optimal?**

  No

- **Time?**

  Worst case: $O(b^m)$ ⇔ like DFS

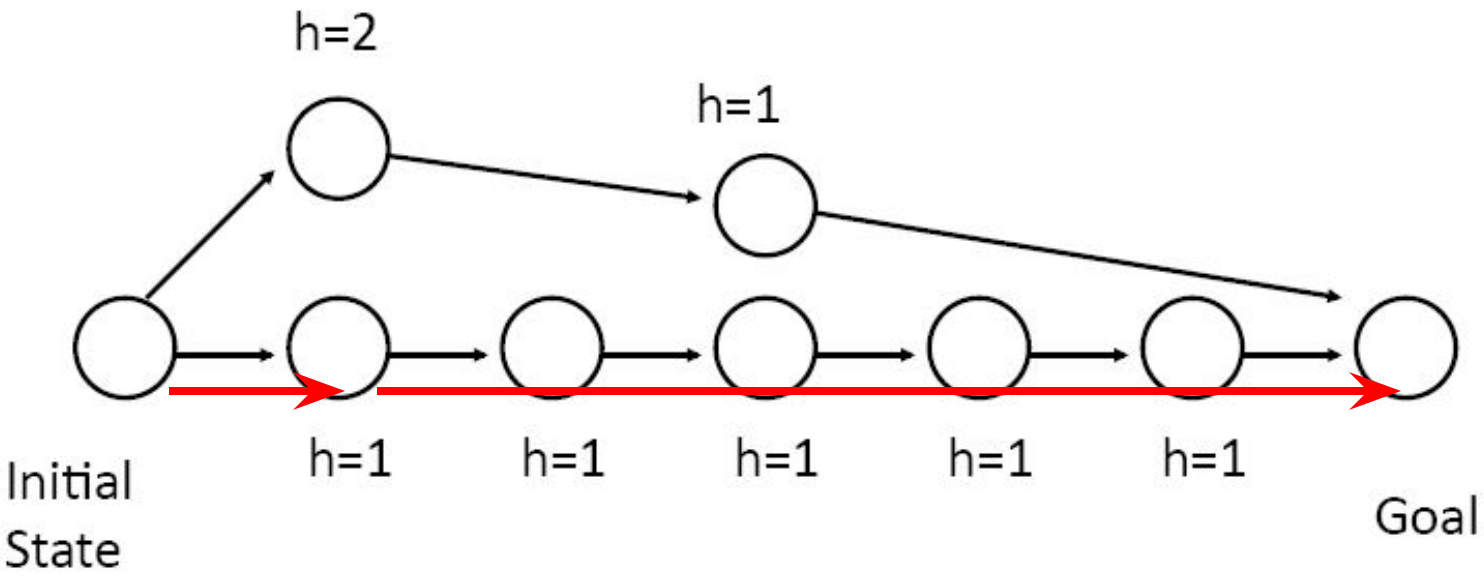
  Best case: $O(bm)$ – If $h(n)$ is 100% accurate

- **Space?**

  Same as time complexity.

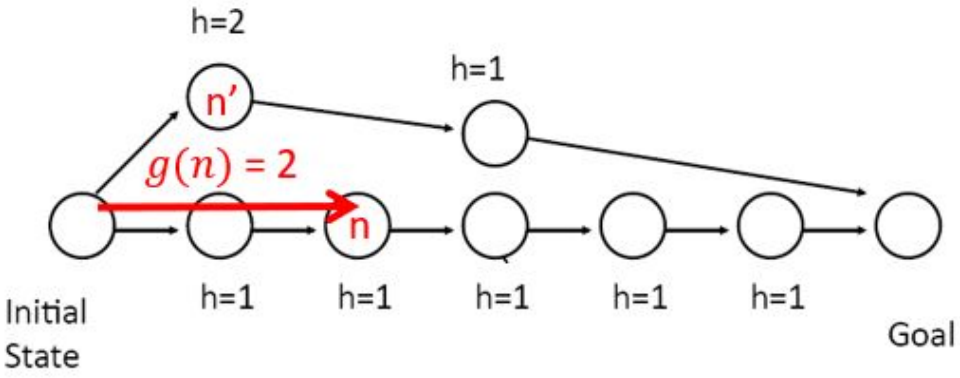
d: depth of the optimal solution
m: max. depth of tree
b: maximum branching factor

How can we fix the optimality problem with greedy best-first search?



$h = 1$ is always better than $h = 2$.
Greedy best-first will go this way
and never reconsider!

A* Search



- A* Search, combines the strengths of uniform-cost search and greedy search. In this search, the *heuristic is the summation of the cost in UCS*, denoted by g(x), and the **cost in greedy search**, denoted by h(x). The *summed cost* is denoted by f(x).

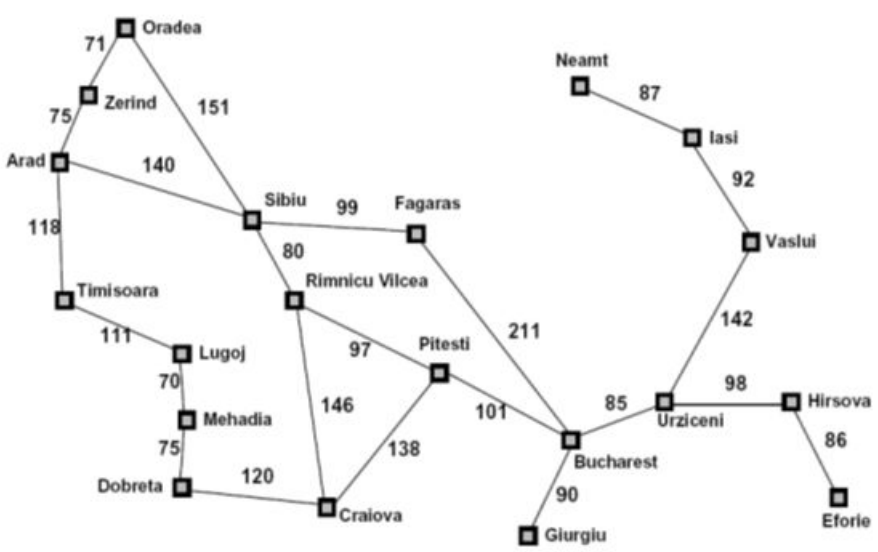
- Evaluation Function: $f(n) = g(n) + h(n)$

  $g(n)$: cost so far to reach *n* (path cost)

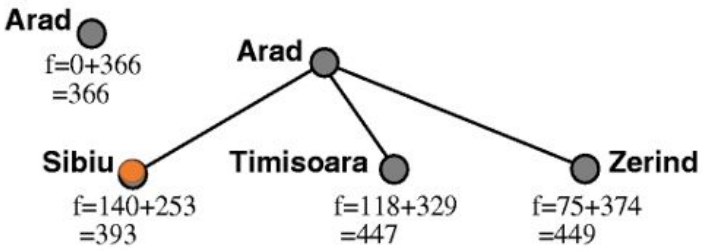
  $h(n)$: estimated cost from *n* to goal (heuristic)

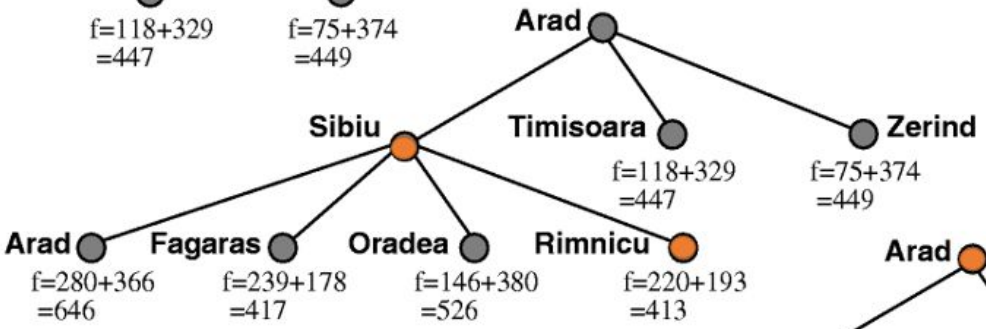
- The agent in the example above will stop at n with $f(n) = 3$ and chose the path up with a better $f(n') = 2$

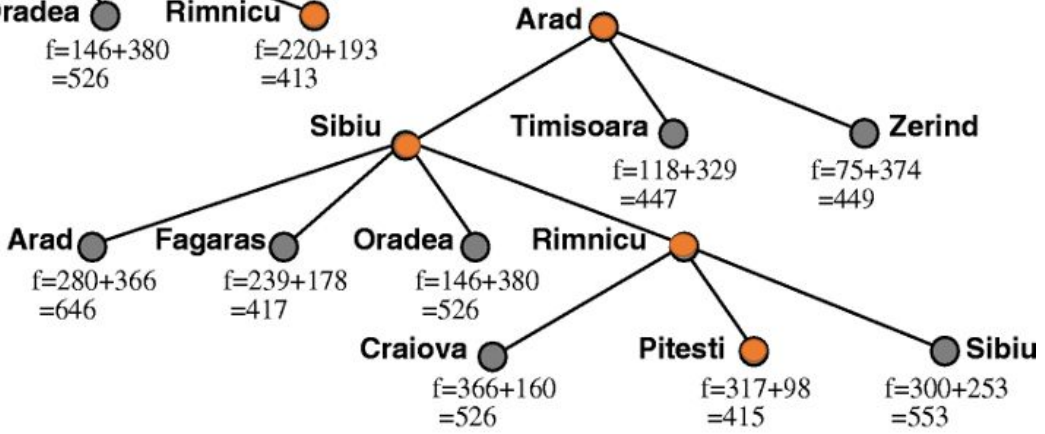
A* Search Example



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374







Implementation of A* Search

Path cost to n + heuristic from n to goal = estimate of the total cost

$$g(n) + h(n)$$

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

The order for expanding the frontier is determined by $f(n)$

This check is different to BFS! It visits a node again if it can be reached by a better (cheaper) redundant path.

See BFS for function EXPAND.

Properties of A*

- **Complete?**
  Yes – if branching factor is finite

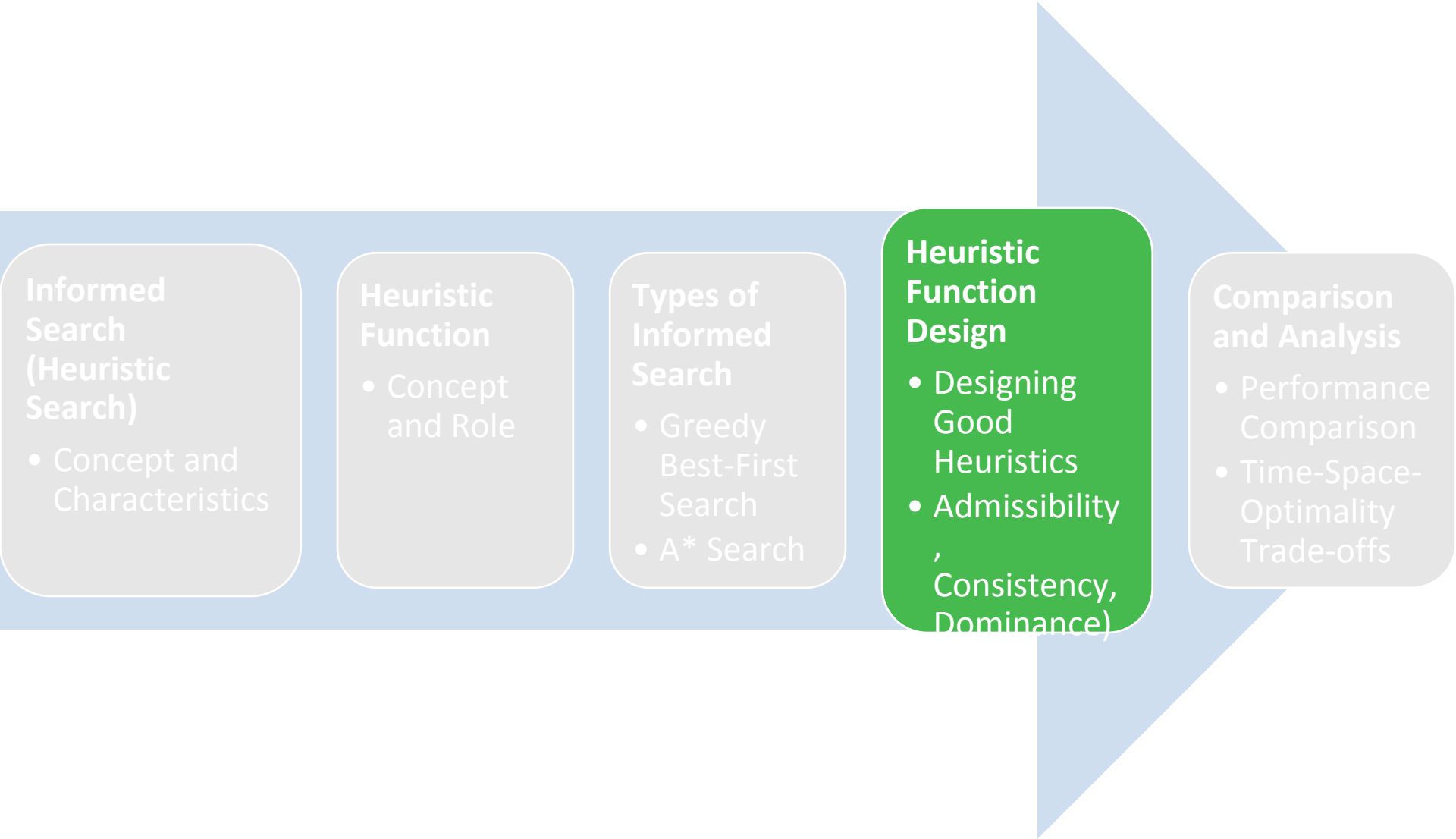
- **Optimal?**
  Yes

d: depth of the optimal solution
m: max. depth of tree
b: maximum branching factor

- **Time?**
  Depends on heuristic function and number of nodes expanded is exponential to the depth of solution d ? $O(b^d)$

- **Space?**
  Same as time complexity.

Outline


Informed Search (Heuristic Search)
• Concept and Characteristics
Heuristic Function
• Concept and Role
Types of Informed Search
• Greedy Best-First Search
• A* Search
Heuristic Function Design
• Designing Good Heuristics
• Admissibility, Consistency, Dominance)
Comparison and Analysis
• Performance Comparison
• Time-Space-Optimality Trade-offs

Why Heuristic Design Matters?

- We saw that **A*** can find *optimal* paths when guided by a *good heuristic*.
- But what makes a heuristic *good* or *bad*?
- Designing an effective heuristic is **crucial** because it directly affects:
  - The **speed** of the search
  - The **optimality** of the result
  - The **memory** used during computation
- **Goal:**
  Understand the properties that make heuristics *useful, admissible,* and *consistent*.

Admissible Heuristics

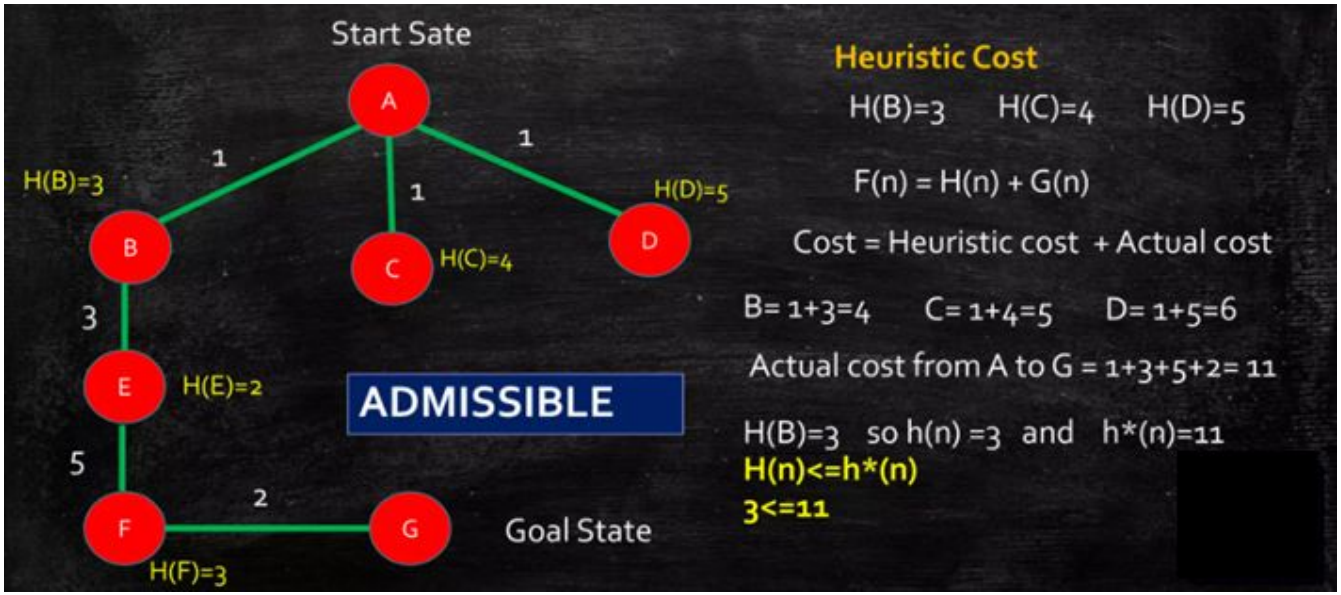
- A heuristic **h(n)** is *admissible* if it **never overestimates** the actual minimal cost to reach the goal.
- $h(n) \leq h^*(n)$

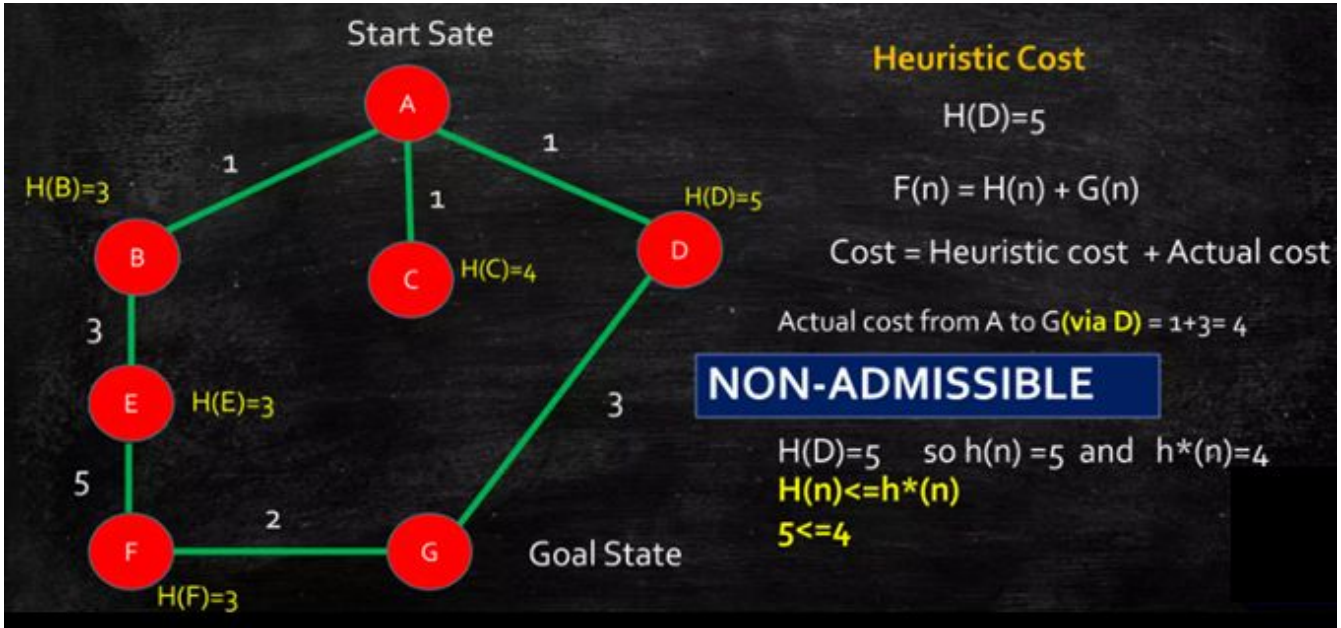
Where $h^*(n)$ =true minimal cost from *n* to the goal.

- **Intuition**
  - Admissible heuristics are *optimistic*.
  - They assume the goal might be *closer* than it really is, but never farther.
- In the Romania map problem,
- *h(n)* = straight-line distance to Bucharest
- Always ≤ real road distance ⇒ *Admissible*

Theorem: If h is admissible, A* is optimal.

Admissible/ Non-Admissible Heuristics





Consistent (Monotonic) Heuristics

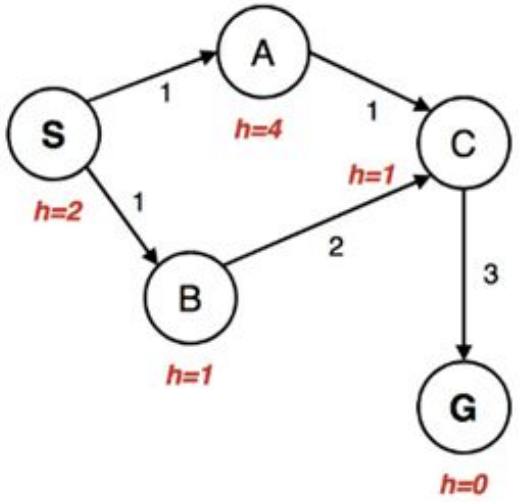
- A heuristic is **consistent** if its estimate satisfies:
- $h(n) \leq c(n, n') + h(n')$

for every node *n* and its successor *n'*, where *c(n, n')* = cost from *n* to *n'*.

- The heuristic value always decreases **gradually** along a path.
- It ensures $f(n) = g(n) + h(n)$is **non-decreasing**.
- **Every consistent heuristic is admissible**, but not every admissible heuristic is consistent.
- **Why Consistency Matters**
  - Simplifies implementation of A*
  - No need to re-expand nodes once visited
  - Ensures **optimality** and **efficiency**
- **Without consistency**, A* may:
  - Revisit nodes unnecessarily
  - Increase computation time

Consistent/ Non-Consistent Heuristics

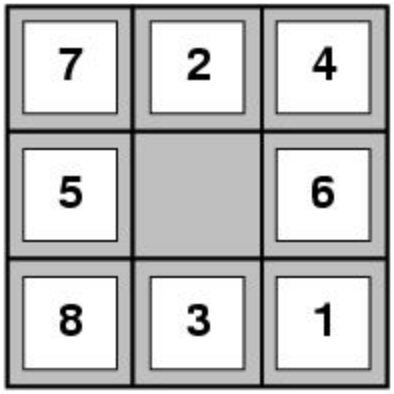
$$h(n) \leq c(n, n') + h(n')$$



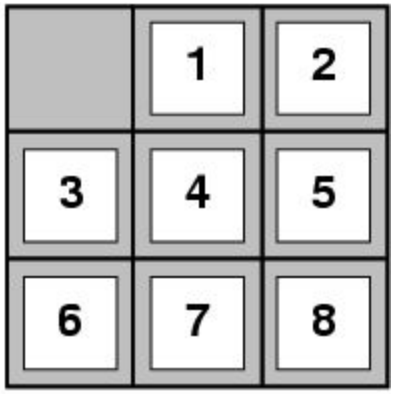
Edge	Cost (c(n, n'))	(h(n))	(h(n'))	Condition	Result
S → A	1	2	4	2 ≤ 1+4 (5)	Consistent
S → B	1	2	1	2 ≤ 1+1 (2)	Consistent
A → C	1	4	1	4 ≤ 1+1 (2)	Non-Consistent
B → C	2	1	1	1 ≤ 2+1 (3)	Consistent
C → G	3	1	0	1 ≤ 3+0 (3)	Consistent

Designing heuristic functions

- Heuristics for the 8-puzzle
  - $h_1(n)$ = number of misplaced tiles
  - $h_2(n)$ = total Manhattan distance (number of squares from desired location of each tile)



Start State



Goal State

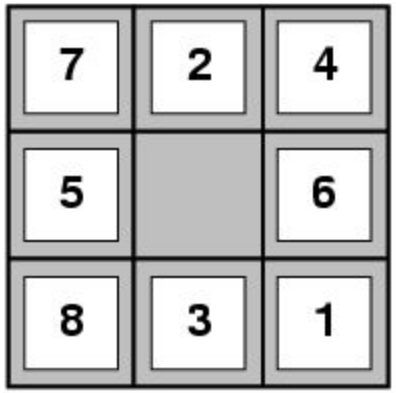
$$h_1(start) = 8$$
$$h_2(start) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

1 needs to move 3 positions

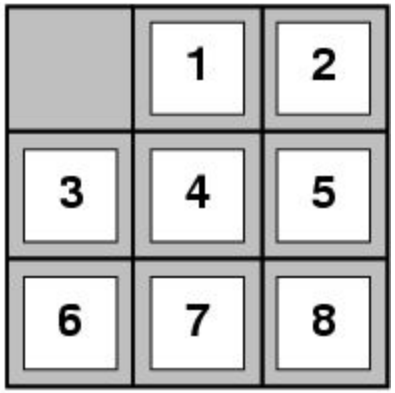
Are h_1 and h_2 admissible?

Heuristics from relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem.
- **The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem. I.e., the true cost is never smaller.**
- h_1: If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.
- h_2: If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.



Start State



Goal State

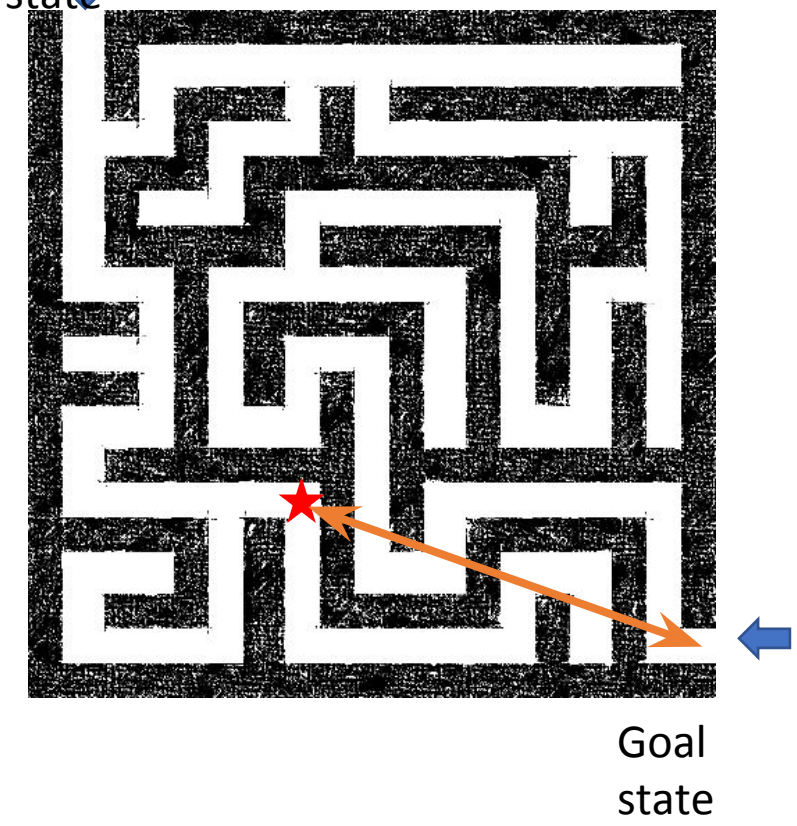
$h_1(start) = 8$

$h_2(start)$
$= 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2$
$= 18$

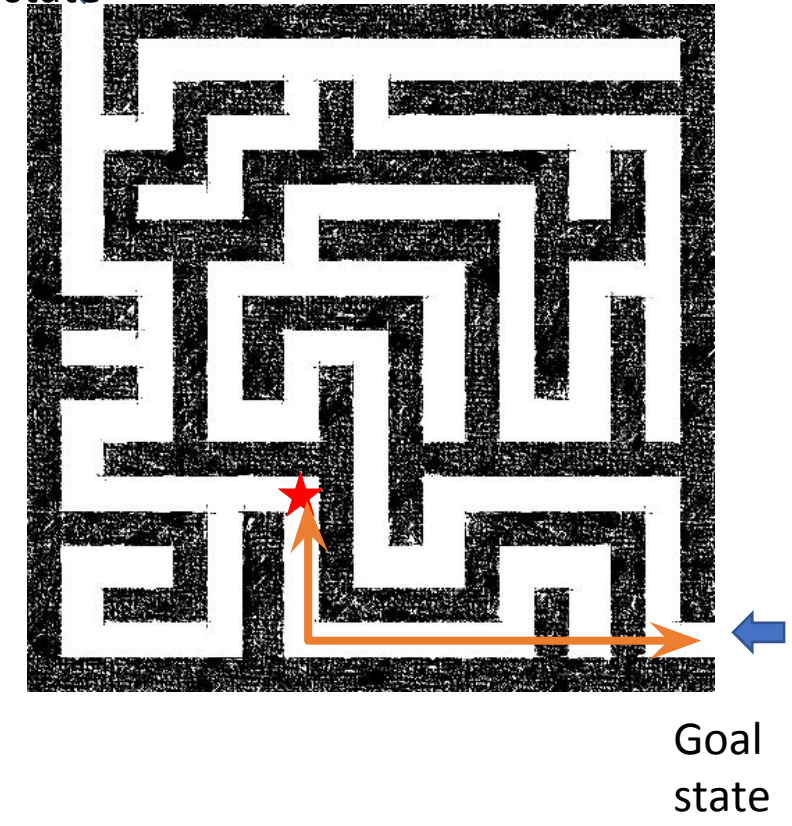
Heuristics from relaxed problems

What relaxations are used in these two cases?

Euclidean distance

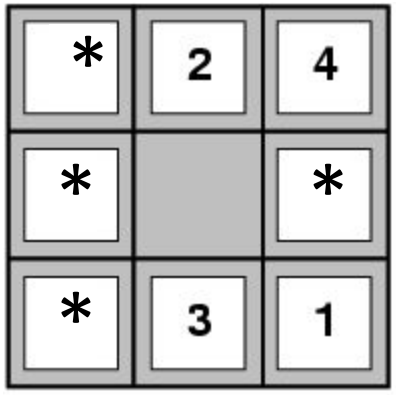


Manhattan distance

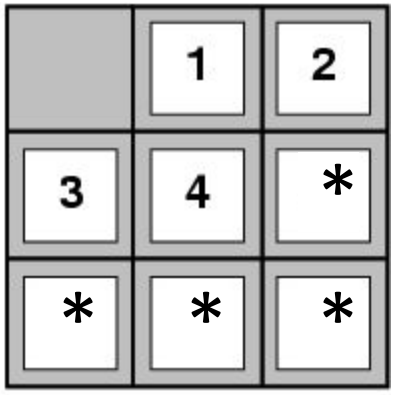


Heuristics from Subproblems

- Let $h_3(n)$ be the cost of getting a subset of tiles (say, 1,2,3,4) into their correct positions. The final order of the * tiles does not matter.
- Small subproblems are often easy to solve.
- Can precompute and save the exact solution cost for every or many possible subproblem instances – ***pattern database***.



Start State



Goal State

Dominance: What heuristic is better?

Definition: If h_1 and h_2 are both admissible heuristics and $h_2(n) \geq h_1(n)$ for all n, then h_2 dominates h_1

Is h_1 or h_2 better for A* search?

- A* search expands every node with $f(n) < C^* \Leftrightarrow h(n) < C^* - g(n)$
- h_2 is never smaller than h_1. A* search with h_2 will expand less nodes and is therefore better.

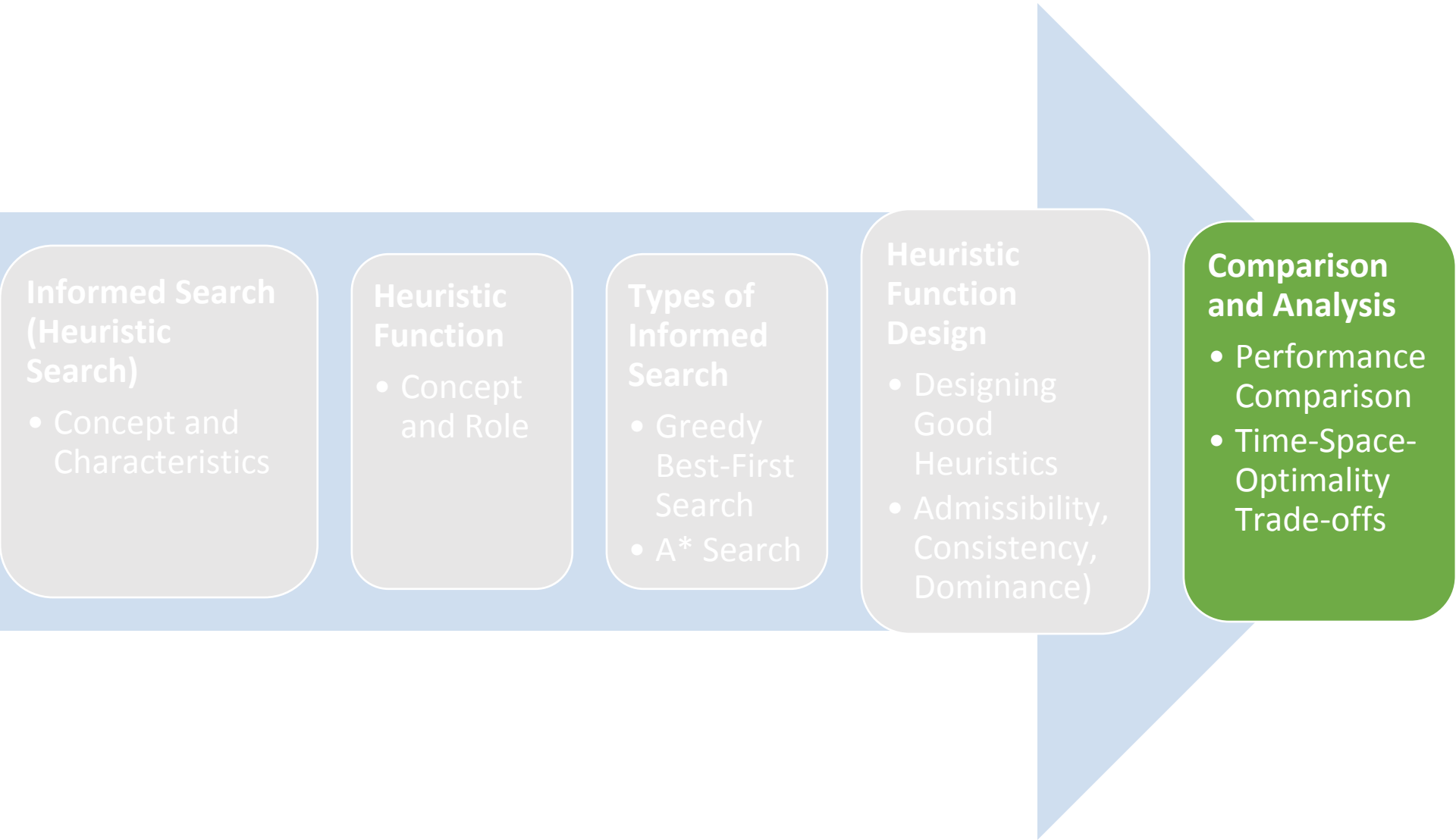
Dominance

- Typical search costs for the 8-puzzle (average number of nodes expanded for different solution depths d):

- $d = 12$ IDS = 3,644,035 nodes
  A*(h_1) = 227 nodes
  A*(h_2) = 73 nodes

- $d = 24$ IDS ≈ 54,000,000,000 nodes
  A*(h_1) = 39,135 nodes
  A*(h_2) = 1,641 nodes

Outline


Informed Search (Heuristic Search)
• Concept and Characteristics
Heuristic Function
• Concept and Role
Types of Informed Search
• Greedy Best-First Search
• A* Search
Heuristic Function Design
• Designing Good Heuristics
• Admissibility, Consistency, Dominance)
Comparison and Analysis
• Performance Comparison
• Time-Space-Optimality Trade-offs

Comparison of Uninformed Search Algorithms

d: depth of the optimal solution
m: max. depth of tree
b: maximum branching factor

Algorithm	Completeness	Optimality	Time Complexity	Space Complexity
Greedy best-first Search	No (May get stuck in loops)	No		
A* Search	Yes	Yes	$O(b^d)$	$O(bd)$

Trade-offs Among Informed Searches

- **Greedy Best-First Search** Fast and memory-light, but not optimal — may take misleading paths.
- **A* Search** Most effective and optimal (if *h(n)* admissible) — high memory and time cost.

Summary

- We explored
  - **Heuristic (Informed) Search:** How intelligent agents use domain knowledge, through heuristic functions, to guide search efficiently toward the goal.
  - **Heuristic Function:** Defined $h(n)$as an estimate of the cost from a state to the goal, emphasizing its role in prioritizing promising nodes and improving search performance.
  - **Informed Search Algorithms:** Examined **Greedy Best-First Search** and **A*** — algorithms that integrate heuristic knowledge to balance speed and optimality.
  - **Heuristic Properties:** Discussed **admissibility**, **consistency**, and **dominance** — principles ensuring that A* remains both efficient and optimal.
- **Trend:**

Informed search transforms brute-force exploration into goal-directed reasoning — a step closer to intelligent planning and decision-making.

"If uninformed search shows how to explore, heuristic search teaches where and why to explore — guiding intelligence with knowledge."