COSC 4368
Fundamentals of Artificial Intelligence

Search 3: Informed Search
August 30th, 2023

Informed (Heuristic) Search

- Search strategies
  - Greedy best-first search
  - A* search
  - Memory bounded heuristic search
- Heuristic functions

Introduction

- Informed search strategies
  - Use problem-specific knowledge beyond the problem definition itself
  - Can find solutions more efficiently than uninformed search strategies

- What are heuristics
  - Rule of thumb, intuition

  "*the ever-amazing observation of how much people can accomplish with that simplistic, unreliable information source known as intuition*"
  --- Judea Pearl, ACM Turing Award Laureate

  - A quick way to estimate how close we are to the close; how close is a state to the goal

Introduction

- Best-first search (BFS)
  - Select the node based on an **evaluation function** (cost estimate)
    - Depend on node , goal, search so far, domain
  - Identical to uniform-cost search when is the path cost
  - Usually include a heuristic function
    - estimated cost of the cheapest path from the state at node to a goal state
    - the most common form to incorporate problem-specific knowledge
    - Nonnegative, and if is a goal node
- Two popular algorithms:
  - Greedy best-first search
  - A* search

Graph Search Algorithm for BFS

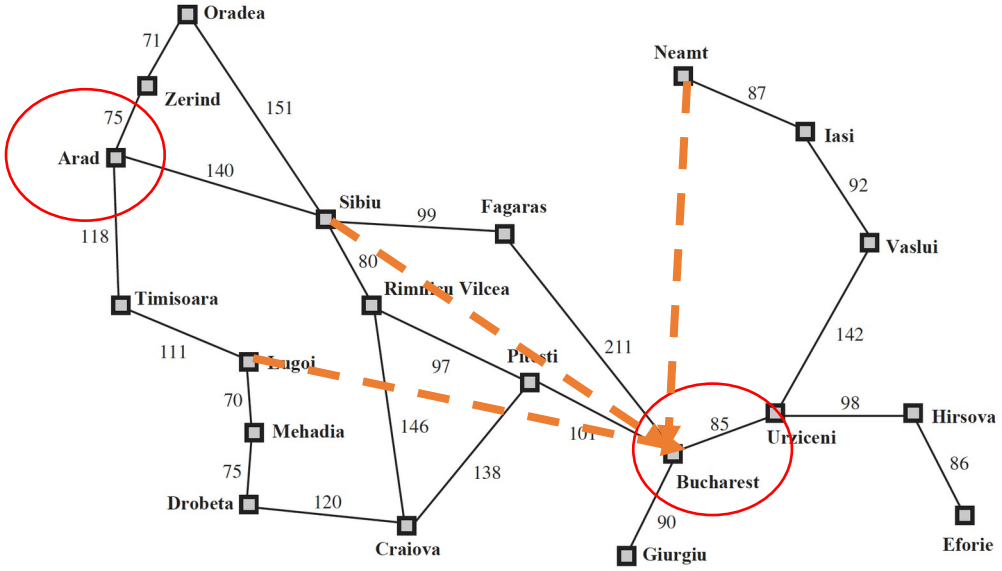
1. Put the start node in *Frontier* of unexpanded nodes.
2. If *Frontier* is empty exit with failure; no solutions exists.
3. Remove the first node in *Frontier* at which is minimum (break ties arbitrarily), and place it into *Explored set* for expanded nodes.
4. If is a goal node, exit successfully with the solution obtained by tracing the path along the pointers from the goal back to .
5. Otherwise expand node , generating all its successors with pointers back to .
6. For every successor on :
    1. If is in *Explored set,* ignore.
    2. Calculate
    3. If was neither in *Frontier* or *Explored set*, add it to *Frontier*. Attach a pointer from back to . Assign the newly computed to node .
    4. if is already in *Frontier*, compare the newly computed with the value previously assigned to . If the old value is lower, discard the newly generated node. If the new value is lower, substitute it for the old (now points back to instead of to its previous predecessor).
7. Go to step 2.

Overview

- Last time:
  - Uninformed search
    - Breadth-first, Uniform-cost, Depth-first, Depth-limited, Iterative deepening, Bidirectional
  - Informed search
    - Greedy best-first search
- Today:
  - Informed search
    - Greedy best-first search
    - A* search
    - Memory bounded search
    - Heuristic functions

Greedy Best-first Search (Greedy BFS)

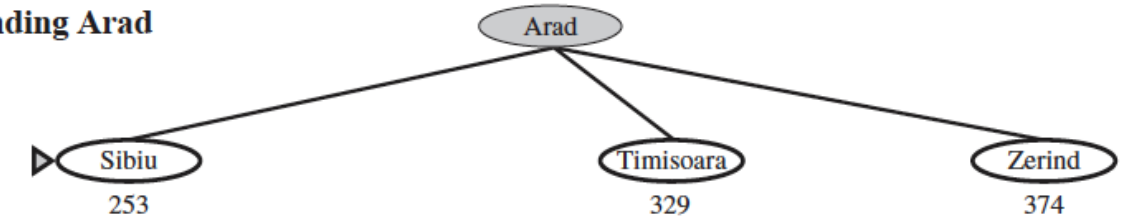
- Expand the node that appears to be closest to the goal (in some metrics)
- Ordering with
  - Estimate of cost from node *n* to the goal
- e.g., the **straight-line distance heuristic** (from node *n* to the Bucharest) in the route-finding problem



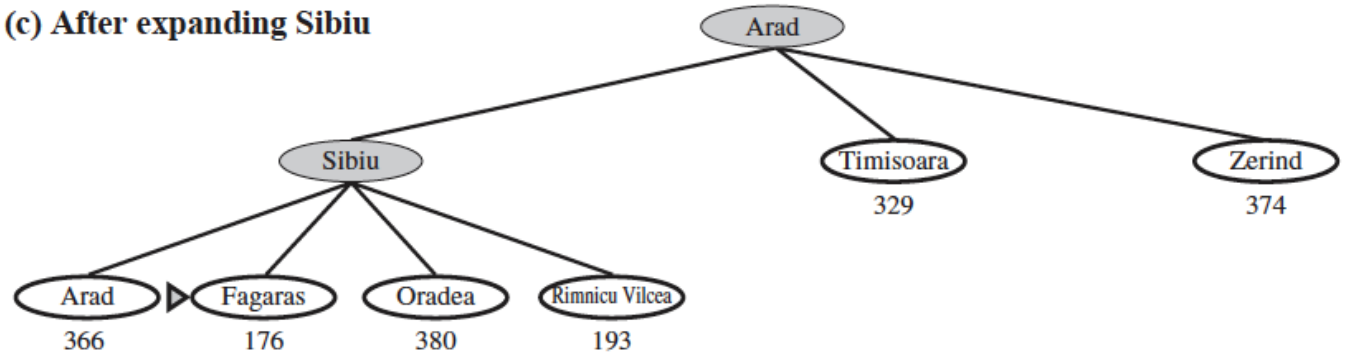
Greedy Best-first Search (Greedy BFS)

(a) The initial state

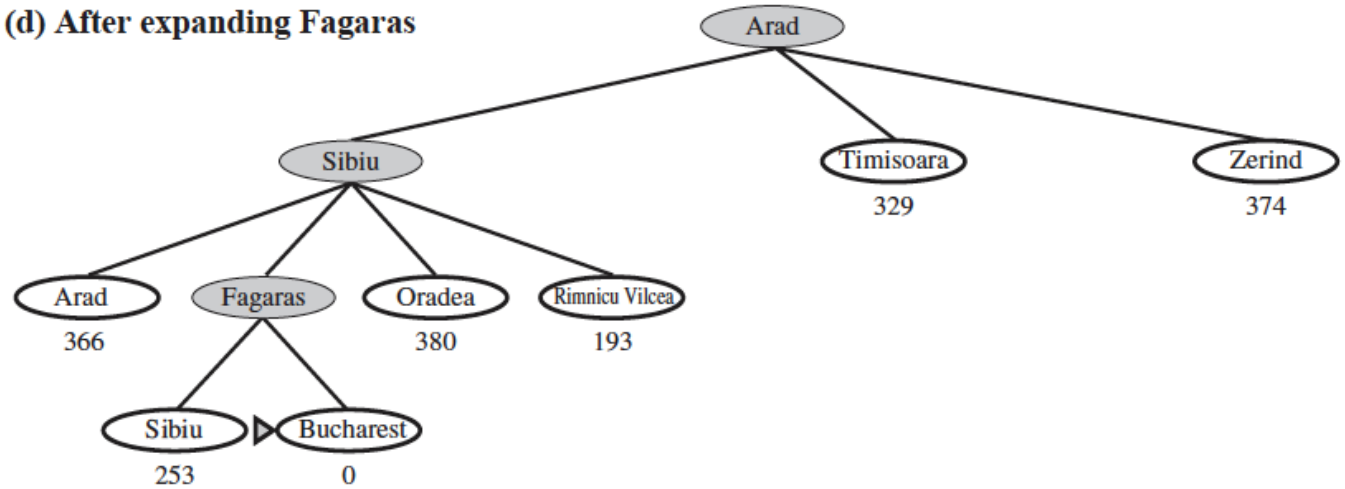
(b) After expanding Arad

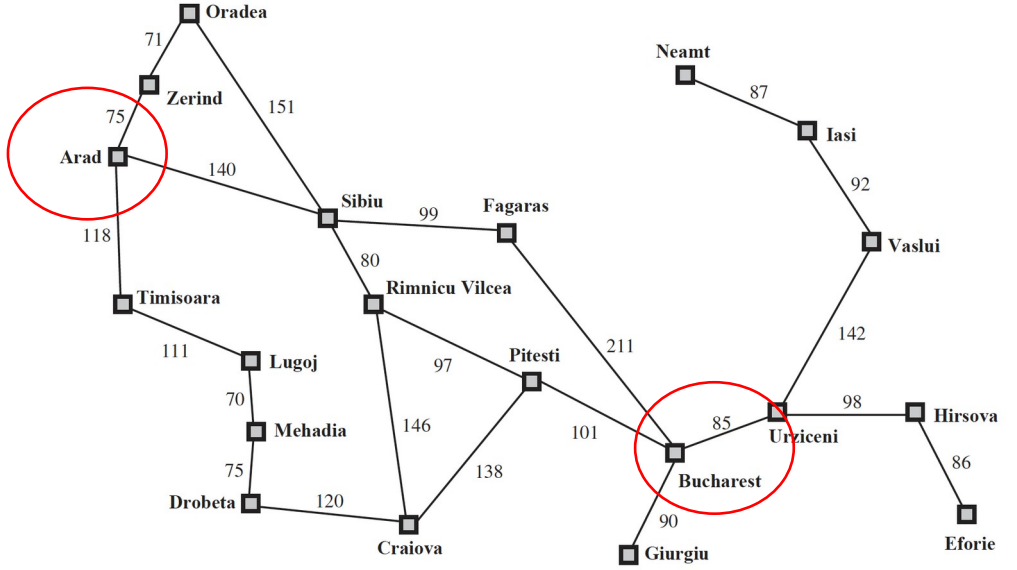


(c) After expanding Sibiu



(d) After expanding Fagaras



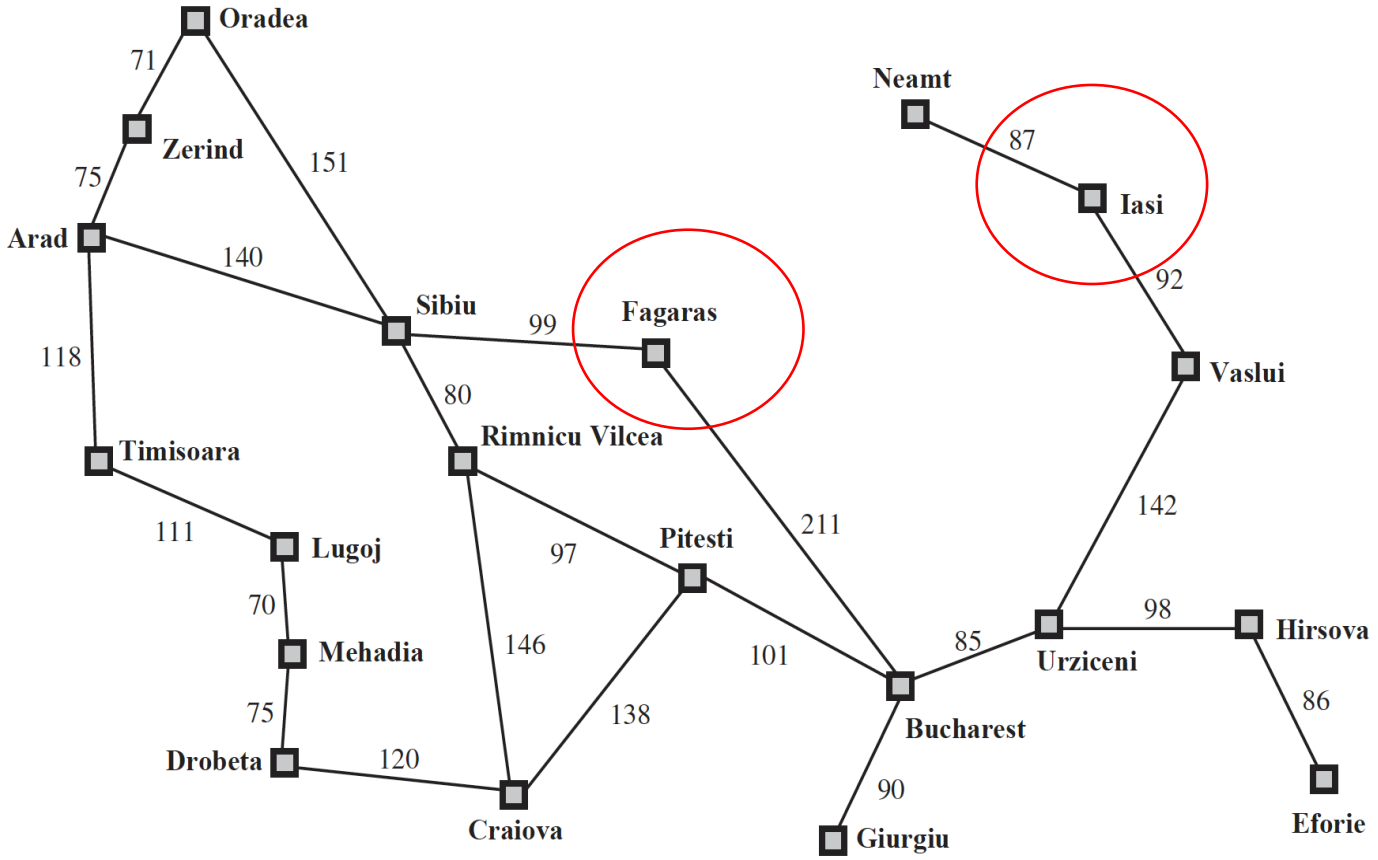


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

Properties of Greedy BFS

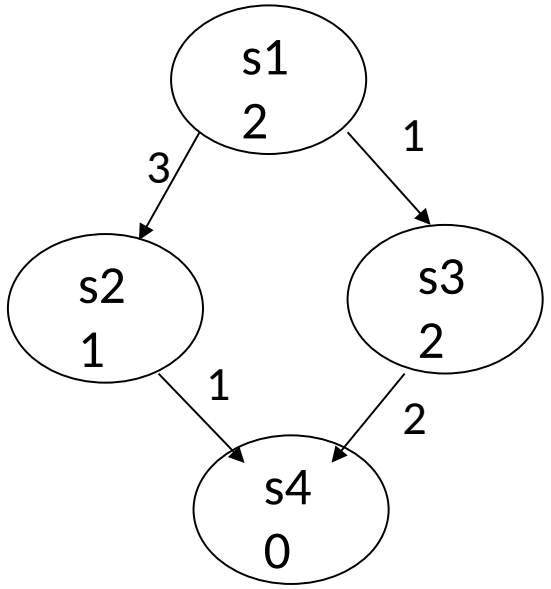
- Complete only for graph search in finite state spaces (like depth-first)



From Iasi to Fagaras

Properties of Greedy BFS

- Not optimal



s4: a goal state;
edge weight: step cost;
h(s1)=2, h(s2)=1;
Greedy BFS: s1-s2-s4, not optimal
Only consider h(s2), but ignores the cost of reaching s2 from the initial state, i.e., g(s2)

- Complexity depends on the problem and the quality of the heuristic
  - Worst case time/space complexity

Some Discussion about Greedy BFS

- Fast and therefore attractive to solve NP-hard and other problems with high complexity
- A lot of successful and popular algorithms in Computer Science are greedy algorithms
- Makes locally optimal choices at each stage
- They do not backtrack: if they make a bad decision (based on local criteria), they never revise the decision
- They are not guaranteed to find the optimal solution(s), and sometimes can get deceived and find really bad solutions

A* Search

- Minimize the total estimated solution cost
- Avoid expanding paths that are already expensive

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

$f(n)$: Estimated cost of cheapest path from node to goal

$g(n)$: Path cost from the root to node

A* search is both complete and optimal given that satisfies certain conditions

Optimality of A* Search

Theorem 1. The tree-search version of A* is optimal if *is admissible.*

Theorem 2. The graph-search version of A* is optimal if *is consistent.*

Admissible Heuristics

- Optimistic: never overestimate the cost to reach the goal

Definition 1. A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n

- Example: the straight-line distance
  - Straight-line distance never overestimates the actual road distance

Proof of Optimality

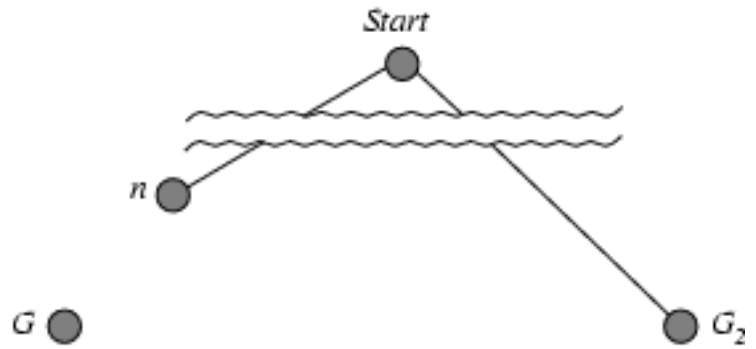
- Prove by contradiction
- *Proof:*

1) Suppose is the optimal goal with a path cost of
2) Suppose is a suboptimal goal with a path cost of
3) Suppose node is on an optimal path to
4) The algorithm selects instead of from Frontier

- Based on 3) and that is admissible, we know

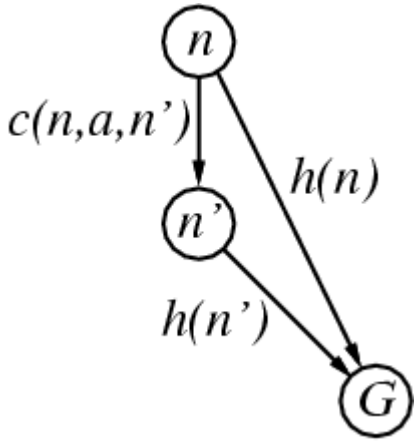
- Together with 2), we know

- Based on 4), we know
- Now we arrive at the contradiction:



→ A* never chooses a suboptimal path

Consistent (Monotonic) Heuristics



- A slightly stronger condition

Definition 2. *A heuristic is consistent if for every node , every successor of generated by any action , we have*

.

step cost

- The estimated cost of reaching goal from is no greater than the step cost of getting to plus the estimated cost of reaching goal from
- A form of the general triangle inequality
- When is consistent, every node is expanded once

Consistent (Monotonic) Heuristics

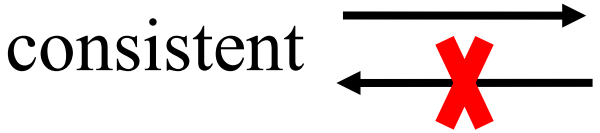
- **Proposition 1.** *If a heuristic is consistent, the value of along any path is non-decreasing.*
- *Proof:* If is consistent, we can have

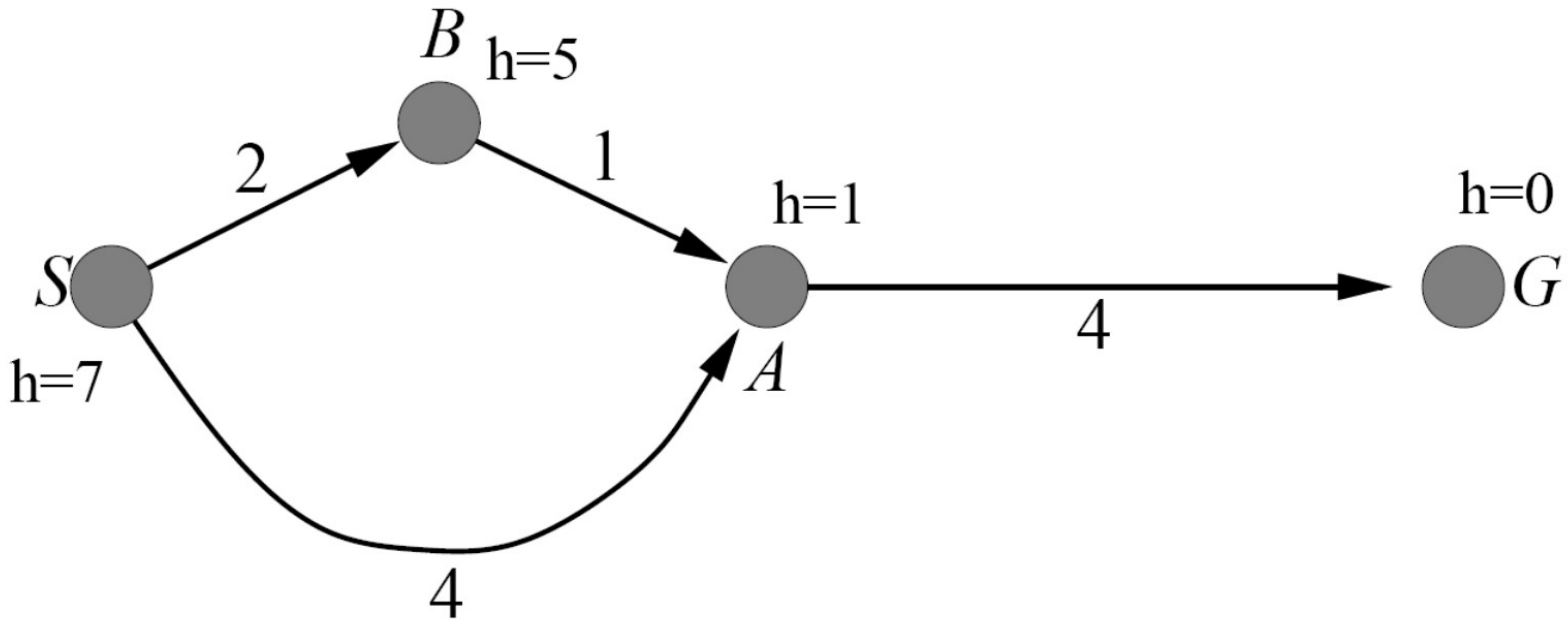
Consistent (Monotonic) Heuristics

- **Proposition 2.** *If a heuristic is consistent and , then it must be admissible.*
- *Proof:* By induction of distance from the goal.
  - For any goal state , we have ;
  - For any predecessor of the goal by taking action , we have
  - For any predecessor of the node by taking action , we have
  - Continue the induction until the root
  - So we can have for any node , , i.e., is admissible.

Consistent (Monotonic) Heuristics

- But the fact that is admissible does not mean that is consistent





Admissible but not consistent

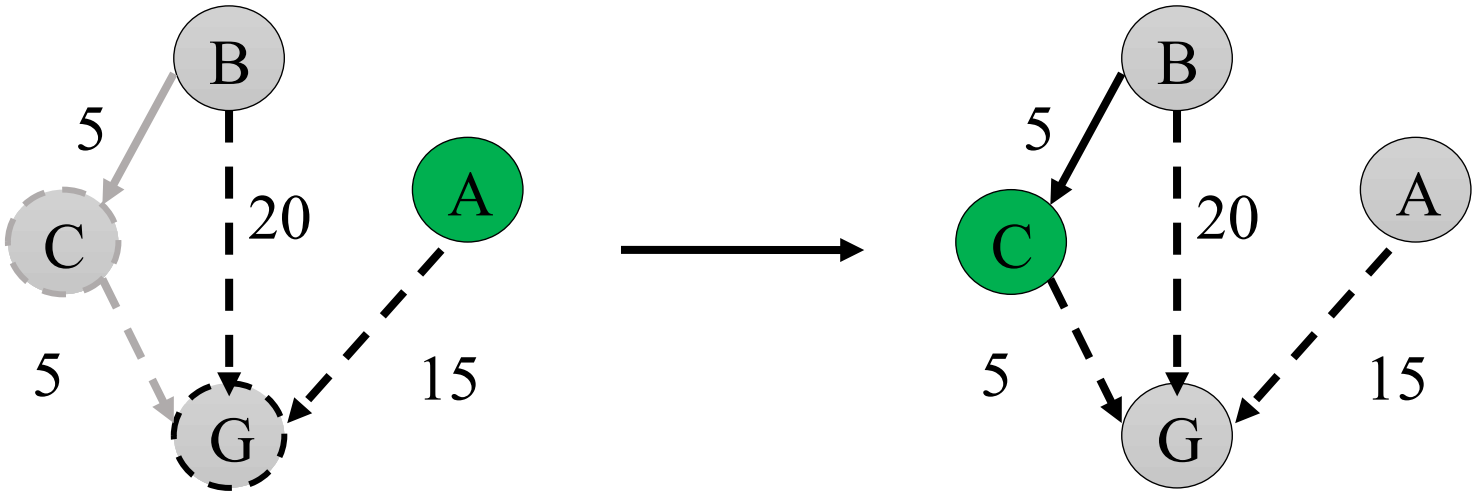
Consistency:

or

Optimality for Graph Search

Theorem 2. The graph-search version of A* is optimal if *is consistent.*

- Why is consistency needed for graph-search?

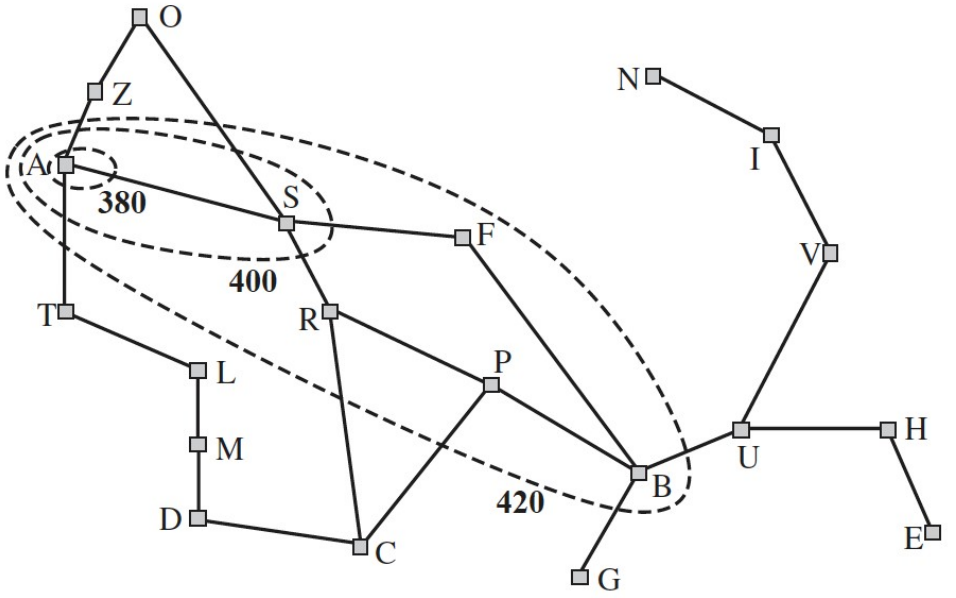


A is better than B

After discovering C, B is better, but can not be revisited, because it is in the explored set

Completeness of A* Search

- Complete for finite and step cost > some positive value
  - The fact that is non-decreasing along any path: draw contours in the state space
  - A* expands nodes in order of increasing value
  - Contour has all nodes with , where
  - Gradually adds “-contours” of nodes
  - A* expands all nodes with (cost of the optimal path), some nodes with , no nodes with



Other Properties of A* Search

- A* is optimally efficient for any given consistent heuristics and suitable tie-breaking rules
  - No other optimal algorithms expand fewer nodes than A*

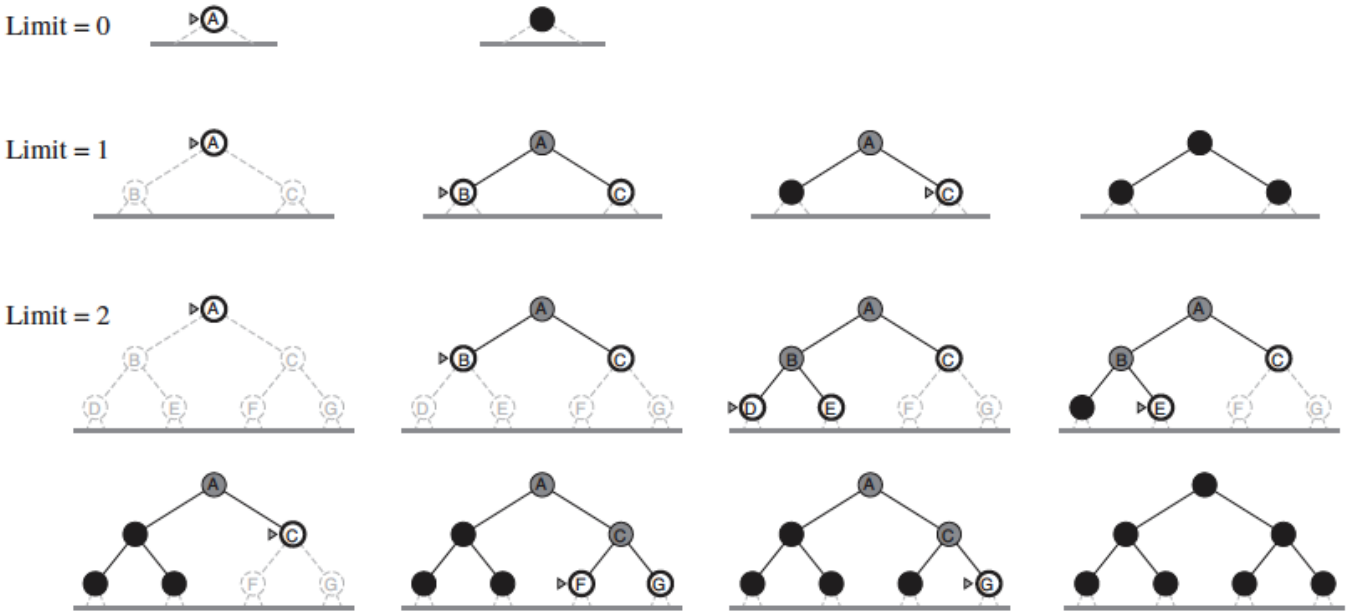
- Complexity
  - Exponential time complexity
  - Exponential space complexity

Overview

- Last time:
  - Informed search
    - Greedy best-first search:
      - Complete only for graph search in finite state spaces
      - Not optimal
      - worst case time/space complexity
    - A* search:
      - Tree search version is optimal for admissible
      - Graph search version for consistent
      - Complete for finite and step cost > some positive value
      - Optimally efficient
      - Exponential time/space complexity
- Today:
  - Informed search
    - Memory bounded search
    - Heuristic functions
  - Local search

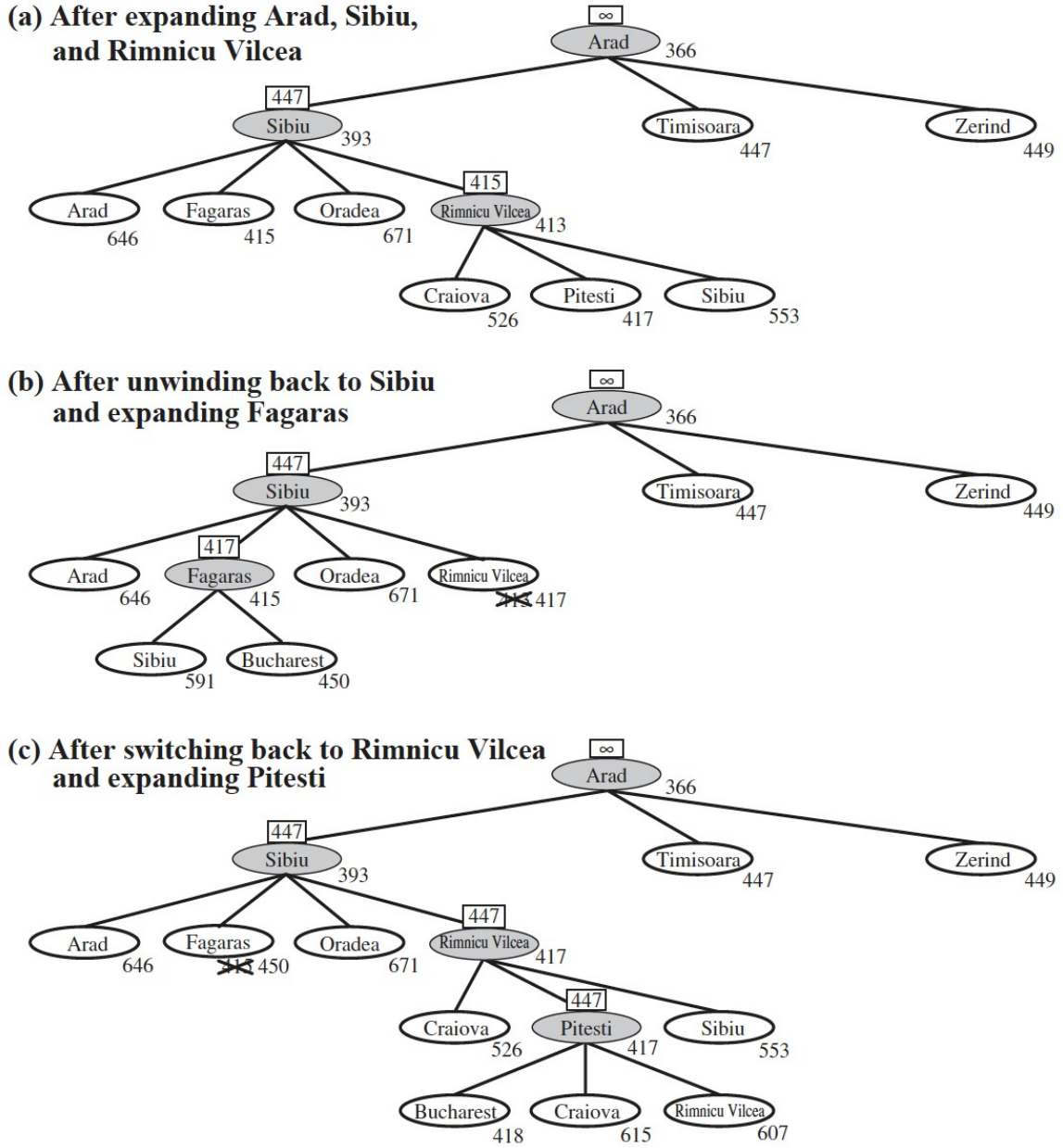
Memory Bounded Search

- Integrate with depth-first
  - Iterative-deepening A* (IDA*)
    - iterative deepening, cutoff value is the smallest f-cost of any node that is greater than the cutoff of the previous iteration
  - When is IDA* complete?
  - Exponential time complexity
  - Linear space complexity



Recall Iterative deepening with depth-limit search (not A*)

Memory Bounded Search



- Integrate with depth-first
  - Recursive best-first search (RBFS)
    - mimics best-first search,
    - uses f_limit to keep track of the best alternative path
    - backtracks if the current path is not promising and a better path exist;
    - advantage: limited size of the frontier
    - disadvantage: excessive node regeneration

Memory Bounded Search

- Add a memory bound
  - Memory-bounded A* (MA*), Simplified MA* (SMA*)
    - Before the memory is full, act like A*
    - After the memory is full, drop the worst leaf node, back up the value of the forgotten node to its parent

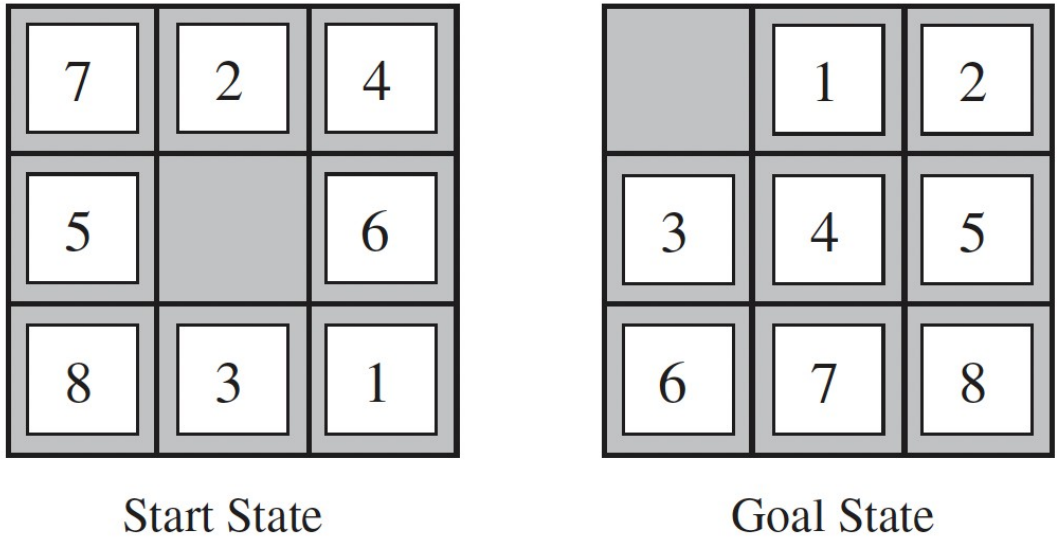
Informed (Heuristic) Search

- Search strategies
  - Greedy best-first search
  - A* search
  - Memory bounded heuristic search
- **Heuristic functions**
  - How quality of the heuristic impacts search
  - How to generate heuristics

The 8-Puzzle Problem

- Commonly used candidates:
  - number of misplaced tiles
  - sum of Manhattan distances from each tile to its goal position

  (i.e., no. of squares from the current position to the goal position)



True solution cost is 26 steps

Effectiveness of A* Search

- Average over 100 randomly generated 8-puzzle problems
- is better than , and is far better than using iterative deepening search

	Search Cost (nodes generated)		
d	IDS	A*(h_1)	A*(h_2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	3644035	227	73
14	–	539	113
16	–	1301	211
18	–	3056	363
20	–	7276	676
22	–	18094	1219
24	–	39135	1641

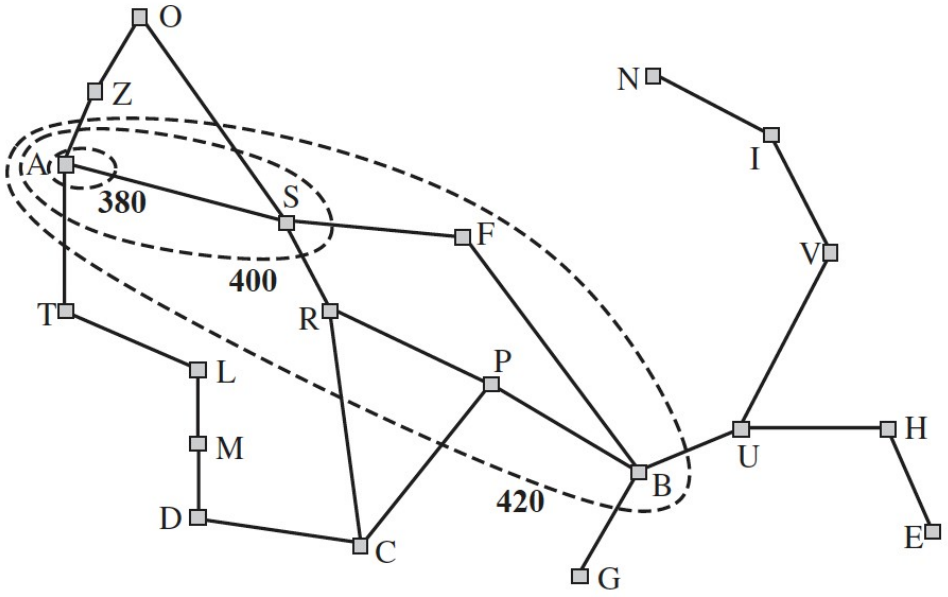
Dominance

- **Definition.** *Suppose both and are admissible. If for any node , then dominates .*

- A* using will never expand more nodes than A* using (except possibly for some nodes with)

- Every node with will surely be expanded

- **Theorem.** *An A* search with a dominating heuristic function has the property that any node it expands is also expanded by A* with .*



How to Generate Heuristics

- Generate admissible heuristics from relaxed problems
  - A problem with fewer restrictions on the actions is **a relaxed problem**
  - **8-puzzle rule: a tile can be moved from A to B, iff**
    - **A and B are adjacent horizontally or vertically**
    - **B is blank**
  - Relaxed P1: remove the condition on adjacency, direction and emptiness
    - a tile can move anywhere instead of just to the adjacent empty square
  - Relaxed P2: remove the condition on direction and emptiness
    - a tile can move one square in any direction, even onto an occupied square

- The relaxed rules allow the problem to be decomposed into independent subproblems, such that the relaxed problems should be easy to solve without search

How to Generate Heuristics

- The state-space graph of the relaxed problem is a supergraph of the original state space
  - The removal of restrictions creates new edges in the graph
  - Any optimal solution to the original problem is a solution in the relaxed problem (which may have better solutions)

- The cost of the optimal solution to a relaxed problem is an admissible heuristic for the original problem
  - gives the exact number of steps in the shortest solution for relaxed P1
  - gives the exact number of steps in the shortest solution for relaxed P2

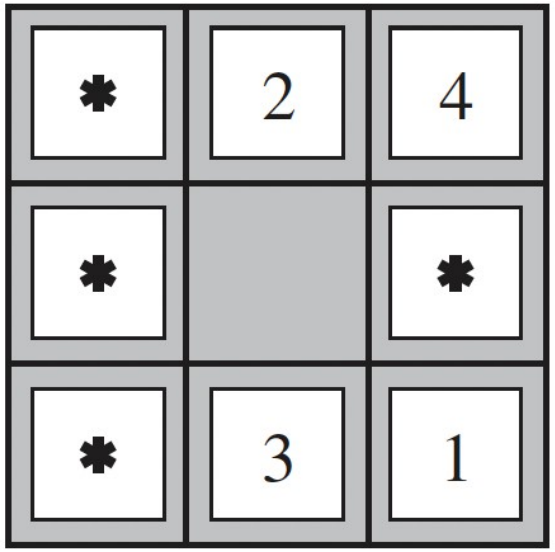
How to Generate Heuristics

- **Theorem.** *Heuristics that are generated from relaxed models are consistent.*

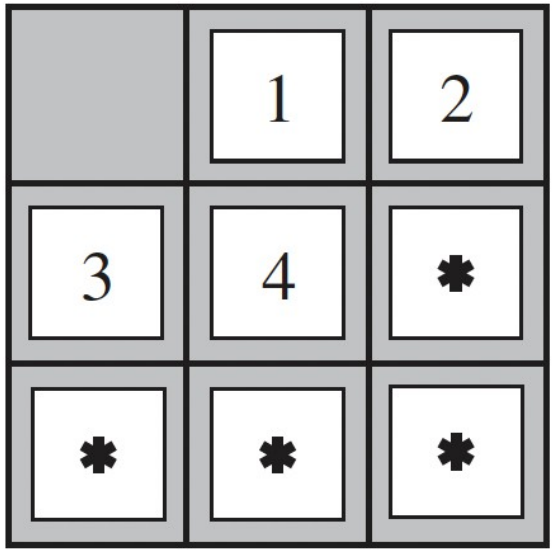
- Proof. H is the true shortest path in a relaxed model
  (1) (is the shortest distance in the relaxed graph)
  (2)
  (3)

How to Generate Heuristics

- Generate admissible heuristics from subproblems
  - Pattern databases: store the exact solution costs for the subproblems
  - Subproblem: e.g., get tiles 1, 2, 3, 4 into the goal positions (other four tiles are irrelevant to the goal, but moves of them will count toward the cost)
  - The cost for the optimal solution of the subproblem is a lower bound on the cost of the original problem



Start State



Goal State

How to Generate Heuristics

- Ask the domain expert (if there is one)
- Try to develop evaluation functions that measure the closeness of a state to a goal state
- Use machine learning, e.g., reinforcement learning to construct a function that predicts the future costs for other states

Summary

- Informed (Heuristic) search: have access to a heuristic function that estimates the cost of a solution from node
- Best-first search: greedy best-first search and A* search
- Memory bounded search
- Quality of heuristics and how to generate heuristics