Artificial Intelligence

Heuristic (Informed) Search

Prof. Dr. habil. Jana Koehler

Dr. Sophia Saller, M. Sc. Annika Engel

Summer 2020

Deep thanks goes to
Prof. Jörg Hoffmann for
sharing his course material

Agenda

- Using Knowledge during Search
  - evaluation of search states
  - admissible and consistent (monotone) heuristics

- Algorithms
  1) Greedy (Best-First) Search
  2) A* and IDA*
  3) Bidirectional Search

- Finding good heuristics

Recommended Reading

- AIMA Chapter 3: Solving Problems by Searching
  - 3.4 Uninformed Search Strategies, the following subchapters:
    - 3.4.6 Bidirectional search
  - 3.5 Informed (Heuristic) Search Strategies
    - 3.5.1 Greedy best-first search
    - 3.5.2 A* search: Minimizing the total estimated solution cost
  - 3.6 Heuristic Functions
    - 3.6.1 The effect of heuristic accuracy on performance
    - 3.6.2 Generating admissible heuristics from relaxed problems
    - 3.6.3 Generating admissible heuristics from subproblems: Pattern databases

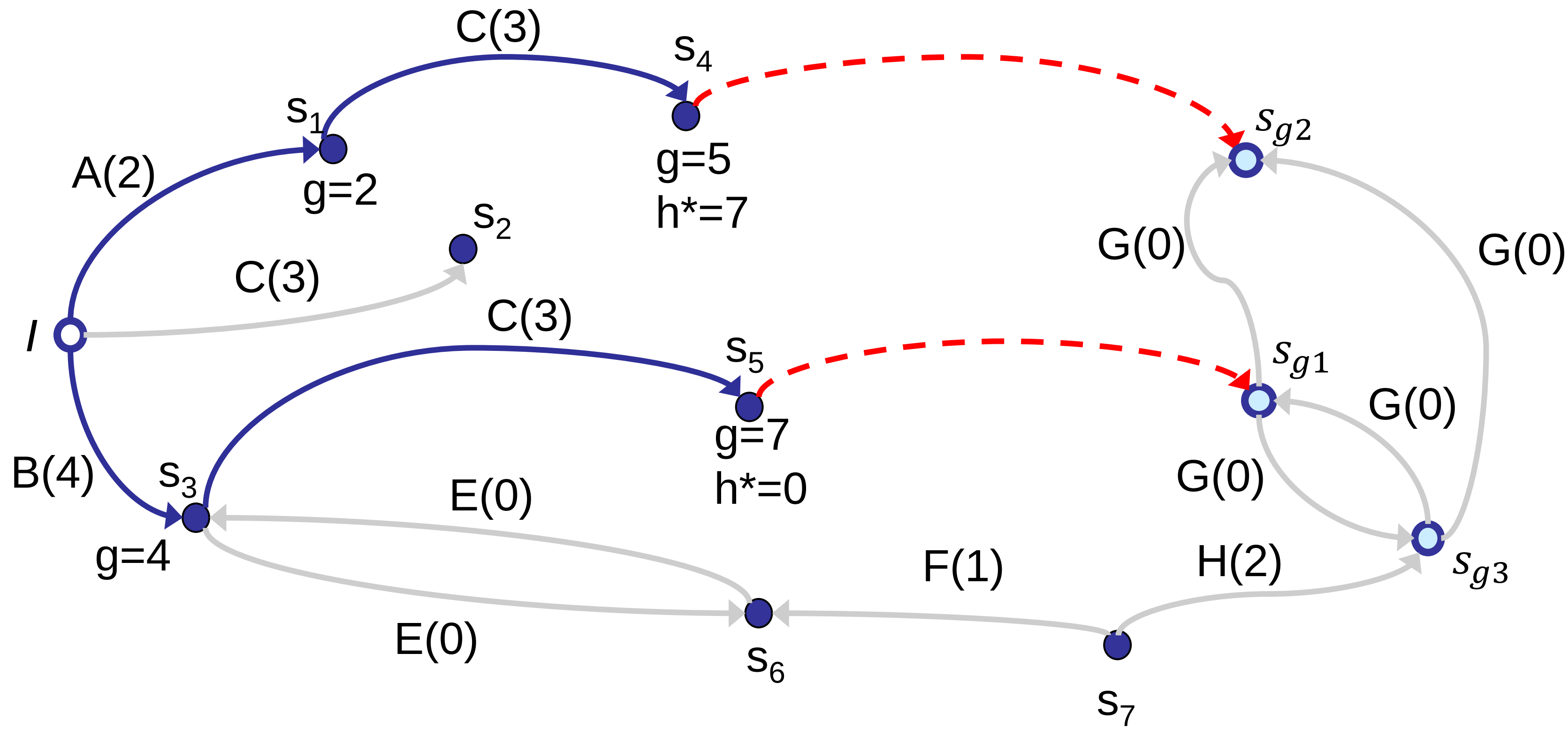
- Optional reading:
  - R. C. Holte, A. Felner, G. Sharon, N. R. Sturtevant: Bidirectional Search That Is Guaranteed to Meet in the Middle, AAAI-2016

How to Determine the next Node for Expansion?

- Uninformed Search
  - rigid procedure, no knowledge of the cost from a node to the goal
  - e.g. FIFO, LIFO queues

- Informed Search
  - "value" of expanding a node (state) used as guidance that steers the search algorithm through the search space
  - evaluation function $f(s)$ assigns a number to each state

The Evaluation Function $f(s) = g(s) + h(s)$



$g(s)$ corresponds to the costs from the initial state to the current state s

- ➢ a precise value

+

$h(s)$ corresponds to the estimated costs from the current state s to the goal state s_g

- ➢ an estimated value

Heuristic Functions h and h^*

Let Π be a problem with state space Θ. A heuristic function, short heuristic, for Θ is a function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ so that, for every goal state s, we have $h(s) = 0$.

The perfect heuristic h^* is the function assigning every $s \in S$ the cost of a cheapest path from s to a goal state, or ∞ if no such path exists.

- $h(s) = \begin{cases} 0, & \text{if } s \text{ is a goal state} \\ > 0, & \text{otherwise} \end{cases}$
- $h^*(s) = \infty$ for dead-end states, from which the goal is unreachable
- $h^*(s)$ is also called the goal distance of s
- The value of h depends only on the state s, not on the path that we followed so far to construct the partial solution (and the costs g of this path)

Desirable Properties of Heuristic Function $h(s)$

1) Efficient to compute ($h(s) = 0$ as extreme case)
2) Informative ($h(s) = h^*(s)$ as extreme case)

3) $$h(s) = \begin{cases} 0, & \text{if } s \text{ is a goal state} \\ > 0, & \text{otherwise} \end{cases}$$

4) h is admissible

5) $h(s_d) = \infty$ for dead-end states s_d

6) h is consistent

- GOOD heuristics should satisfy a balanced compromise of properties (1) to (4) at least, better of all 6
- Properties (5) ensures effective dead-end recognition and (6) is a prerequisite for algorithms to guarantee minimal-cost (optimal) solutions

Admissiblity of $h(s)$

Let Π be a problem with state space Θ and let h be a heuristic function for Θ. We say that h is admissible if, for all $s \in S$, we have $h(s) \leq h^*(s)$.

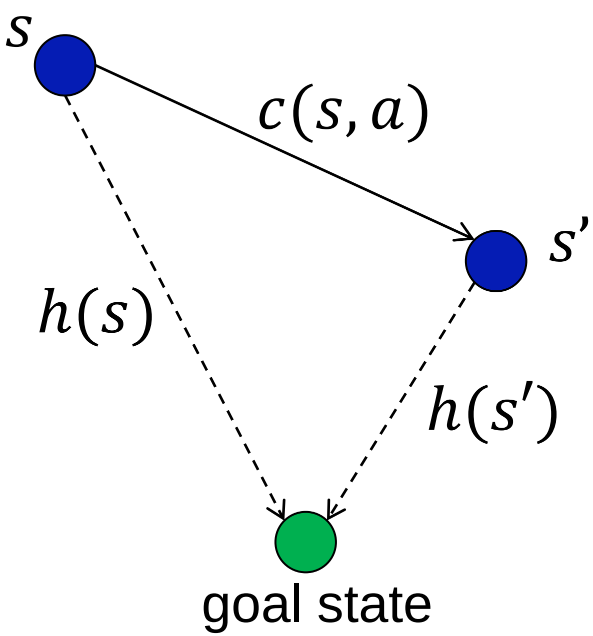
The function $h^*(s)$ corresponds to the real cost of the optimal path from node n to a goal state.

The function h is an optimistic estimation of the costs that actually occur. It underestimates the real costs and provides the search algorithm with a lower bound on the goal distance.

Consistency (Monotonicity) of *h(s)*

Let Π be a problem with state space Θ, and let h be a heuristic function for Θ. We say that h is consistent if, for all transitions $s \xrightarrow{a} s'$ in Θ, we have $h(s) - h(s') \leq c(s,a)$.

The value $c(s,a)$ is the action cost of getting from s to s' with action a.
We reformulate the inequality from above to: $h(s) \leq c(s,a) + h(s')$.



Triangle inequality: The sum of the lengths of any two sides of a triangle must be greater or equal than the length of the remaining side.

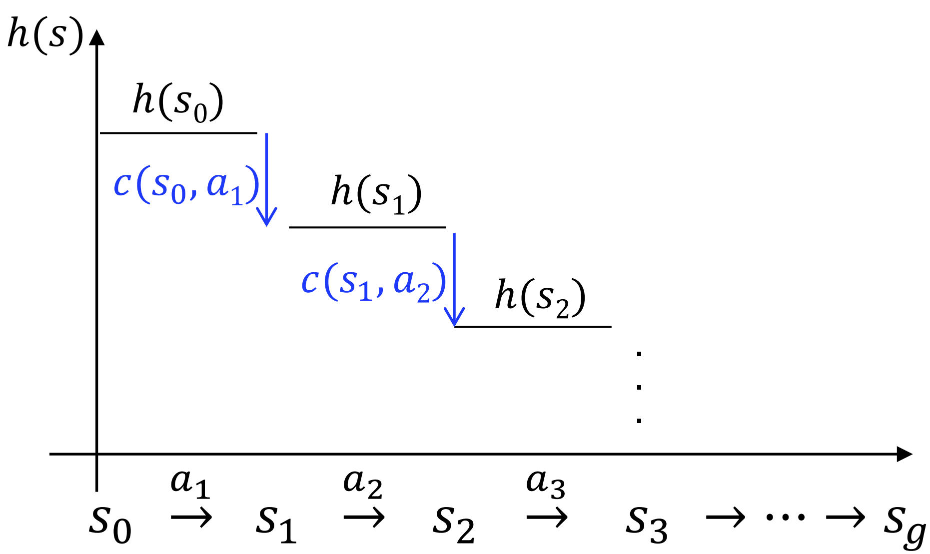
Applying an action a to the state s, the heuristic value cannot decrease by more than the cost $c(s,a)$ of a.

Consistency $\Longrightarrow$ Admissibility

Let Π be a problem with state space Θ and let h be a heuristic function for Θ.
If h is consistent, then h is admissible.

To show: $h(s) - h(s') \leq c(s,a),\ \forall s \xrightarrow{a} s' \Longrightarrow h(s) \leq h^*(s),\ \forall s \in S$.
This means that we need to show that a consistent heuristic never overestimates the costs to the goal.

Observation: The value of h can at most decrease by the action costs.



$$
\begin{aligned}
& h(s) - h(s') \leq c(s,a) \\
\Leftrightarrow\ & h(s) \leq c(s,a) + h(s')
\end{aligned}
$$

Proof: We need to show that $\boldsymbol{h(s) \leq h^*(s)}$ for all $\boldsymbol{s}$.
For states s (dead ends) where $h^*(s) = \infty$, this is trivial as any number is $\leq \infty$.
Now let $\mathcal{S}_k$ be the set of non dead-end states with a shortest cheapest path to a goal state of length k.
We will prove for all k that $h(s) \leq h^*(s)$ for all $s \in \mathcal{S}_k$ by induction over k.

Base case: s is a goal state, so $s \in \mathcal{S}_0$ $(k = 0)$. By the definition of heuristic functions then $h(s) = 0$ and so $h(s) \leq h^*(s) = 0$ as required.

Inductive Hypothesis: For all $s \in \mathcal{S}_k$ we have that $h(s) \leq h^*(s)$.

Inductive step: Let $s \in \mathcal{S}_{k+1}$. Then the cheapest path from s to a goal state has length $k + 1$. Let s' be the successor state of s in this cheapest path, so $s \xrightarrow{a} s'$. We thus know that $s' \in \mathcal{S}_k$ and therefore

1) By the consistency of h we have: $h(s) - h(s') \leq c(s, a)$
2) By the Inductive Hypothesis: $h(s') \leq h^*(s')$
3) Since the cheapest path has the cheapest costs: $h^*(s) = h^*(s') + c(s, a)$

Combining these three statements, we get

$$\underbrace{\boldsymbol{h(s)} \leq h(s') + c(s,a)}_{1)} \leq \underbrace{h^*(s')}_{2)} + c(s,a) = \underbrace{\boldsymbol{h^*(s)}}_{3)}$$

QED

(1) Greedy Best-First Search

- Uses only the heuristic part of the evaluation function

$$f(s) = h(s)$$

- Expands the node first that is estimated as being closest to the goal

- Does not consider the current path costs
  – "counterpart" to uniform-cost search, which uses

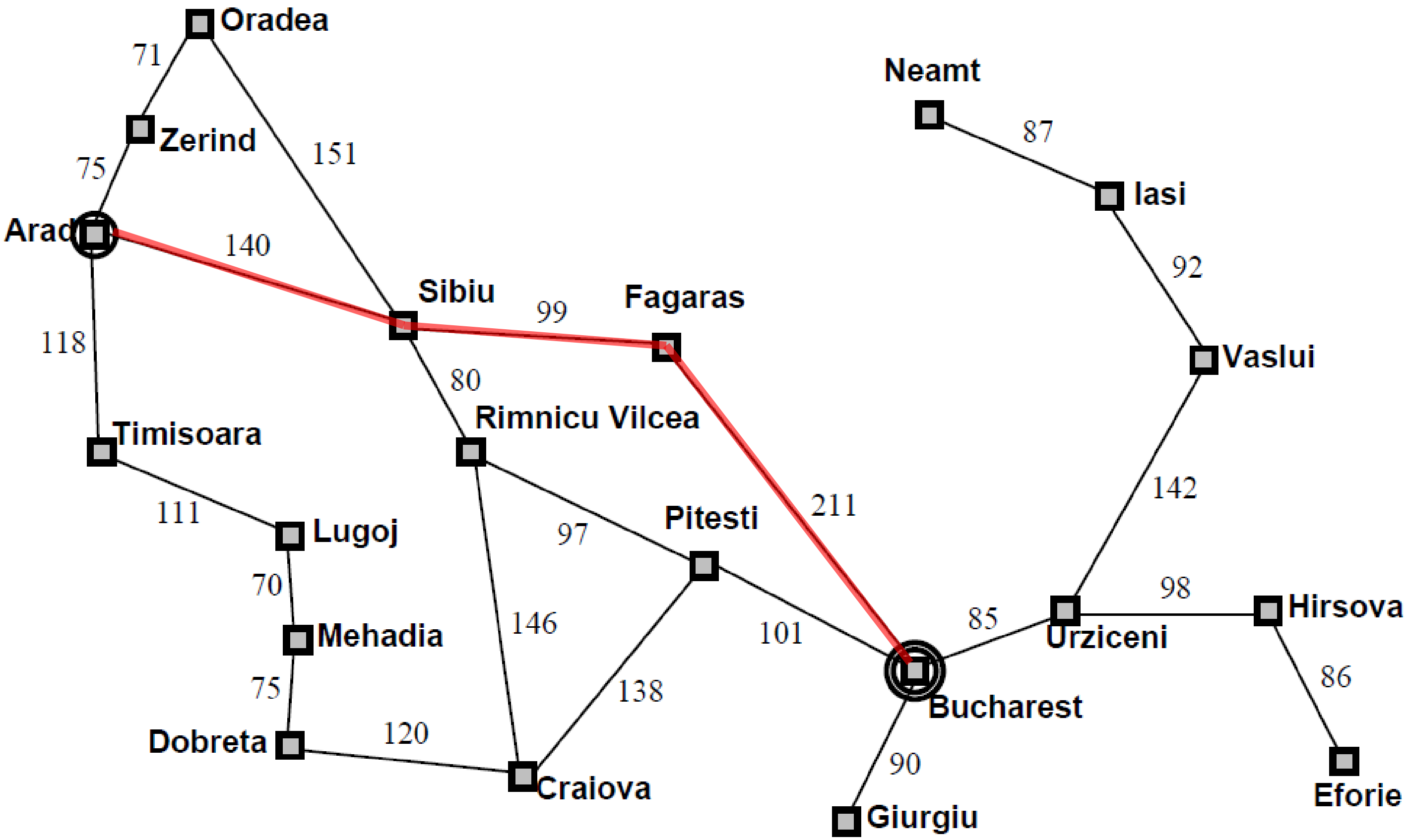
$$f(s) = g(s)$$

GBFS Algorithm

function GREEDY BEST-FIRST-SEARCH(*problem*) **returns** a solution or failure
  $node \leftarrow$ a node n with n.State $=$ *problem*.InitialState
  $frontier \leftarrow$ a priority queue ordered by ascending h, only element n
  $explored \leftarrow$ empty set of states
  loop do
    if Empty?(*frontier*) **then return** failure
    $n \leftarrow$ Pop(*frontier*)
    if GoalTest(n.State) **then return** *Solution*(n)
    $explored \leftarrow explored \cup n$.State
    for each action a **in** Actions(n.State) **do**
      $n' \leftarrow$ ChildNode(problem, n, a)
      if n'.State $\notin explored \cup$ States(*frontier*) **then** Insert(n', $h(n')$, *frontier*)

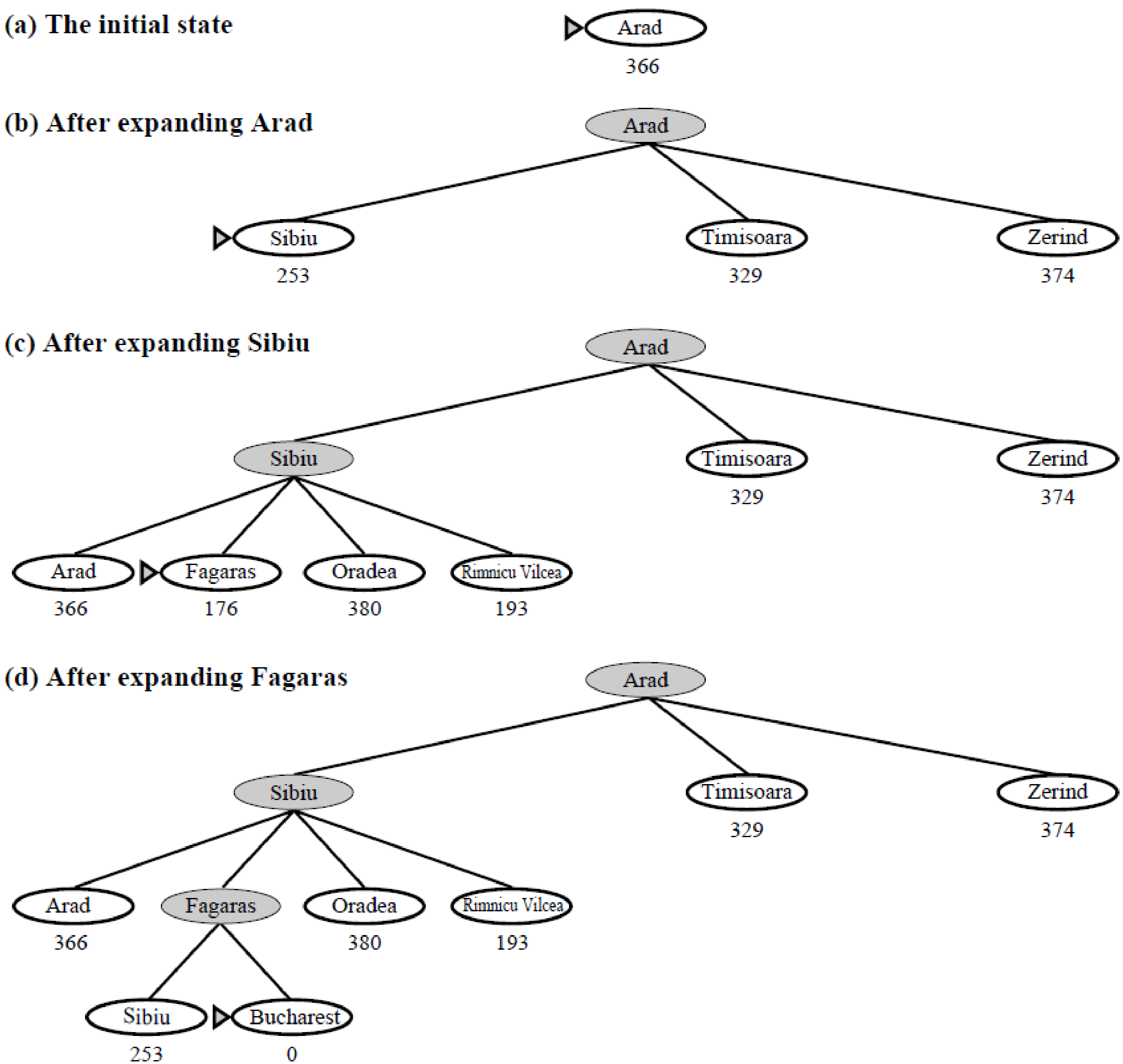
- Frontier ordered by ascending h
- Duplicates checked at successor generation, against both the frontier and the explored set

GBFS on the Romania Travel Example



<u>h: Aerial Distances to Bucharest</u>

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



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

Properties of GBFS

- **Complete**
  - for finite state spaces and with duplicate elimination

- **Not optimal**

- **Time complexity** is $O(b^m)$
- **Space complexity** is $O(b^m)$

where m is the maximum depth of the search space

UNIVERSITÄT DES SAARLANDES

(2) A* (Hart, Nilsson, Raphael 1968)

- Greedy search only uses $h(s)$
- Uniform-Cost search uses $g(s)$
  - finds an optimal solution if path costs grow monotonically:

$$g(s) \leq g(s')$$

- A* uses $f(s) = g(s) + h(s)$
- *A** combines both using preferably **admissible and consistent heuristics**
- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html gives a good introduction

A* Algorithm

function A*(*problem*) **returns** a solution or failure
  node ← a node n with n.State = *problem*.InitialState
  frontier ← a priority queue ordered by ascending $g + h$, only element n
  explored ← empty set of states
  loop do
    if Empty?(*frontier*) **then return** failure
    n ← Pop(*frontier*)
    if GoalTest(n.State) **then return** Solution(n)
    explored ← *explored* ∪ n.State
    for each action a **in** Actions(n.State) **do**
      n' ← ChildNode(problem, n, a)
      if n'.State ∉ *explored* ∪ States(*frontier*) **then**
        Insert(n', $g(n') + h(n')$, *frontier*)
      else if ex. n'' *in frontier* s.t. n'.State = n''.State and $g(n') < g(n'')$ **then**
        replace n'' in *frontier* with n'

Frontier ordered by ascending $g + h$, duplicates handled as in UCS (nodes replaced by duplicates with cheaper costs)

Properties of A*

- A* is **complete**
  - if a solution exists, A* will find it provided that
    1) every node has a finite number of successor nodes, and
    2) each action has positive and finite costs

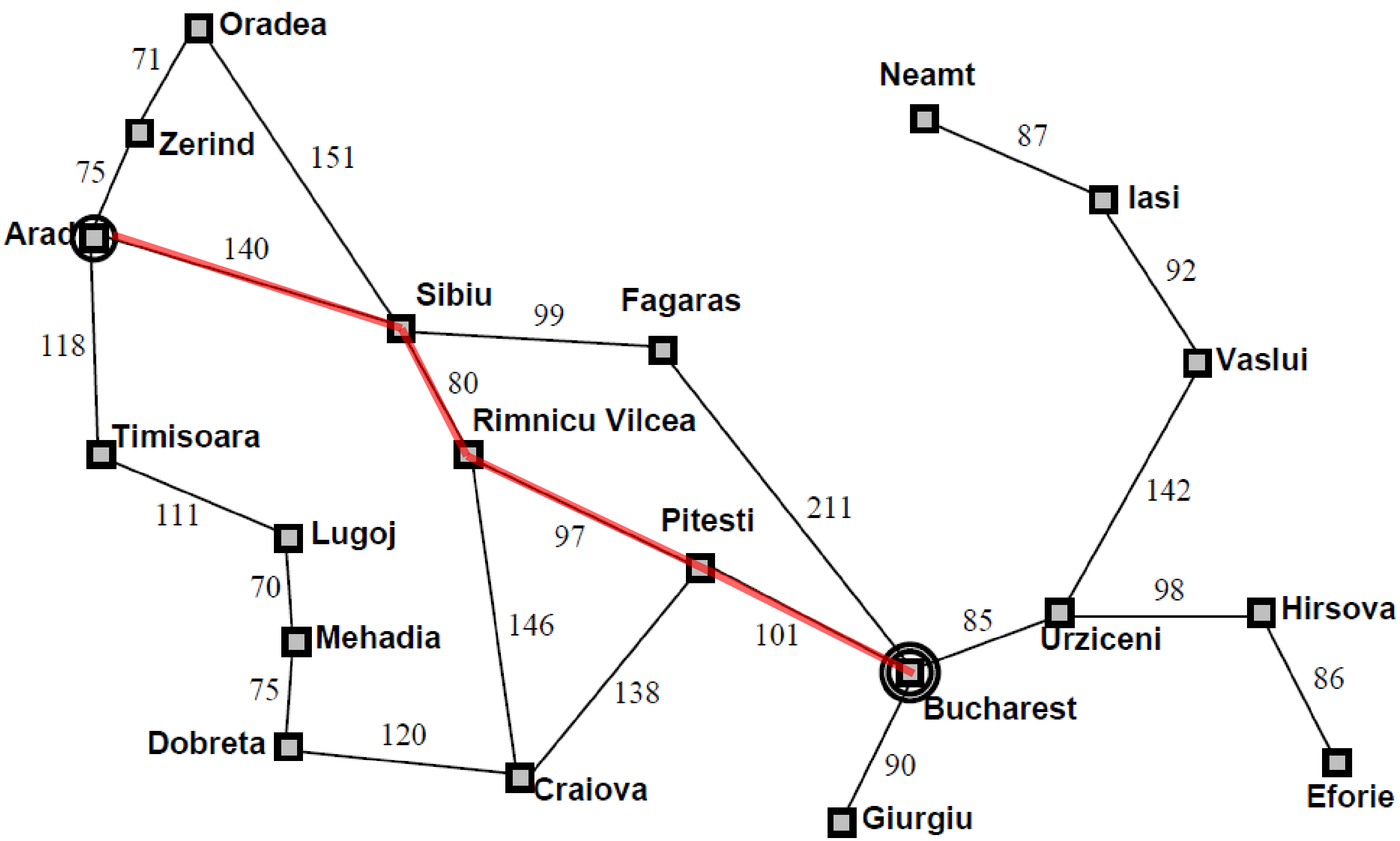
- A* is **optimal**
  - first solution found has minimum path cost if h is admissible (on trees) or if h is consistent (on graphs)
    - under an admissible heuristics on graphs, A* needs to expand all nodes with $f(n) \leq C^*$ (the cost of an optimal solution), called "re-opening"
    - For any path, re-opening checks if it is the cheapest to a state s and puts explored states back into the frontier if a cheaper path was found

Properties of A*

- **Time Complexity** is $O(b^d)$
- **Space Complexity** is $O(b^m)$, where m is the maximum depth of the search space
  - subexponential growth requires that the error in the heuristic function grows no faster than the logarithm of the actual path cost

$$|h(s) - h^*(s)| \leq O(\log h^*(s))$$

A* on the Romania Travel Example

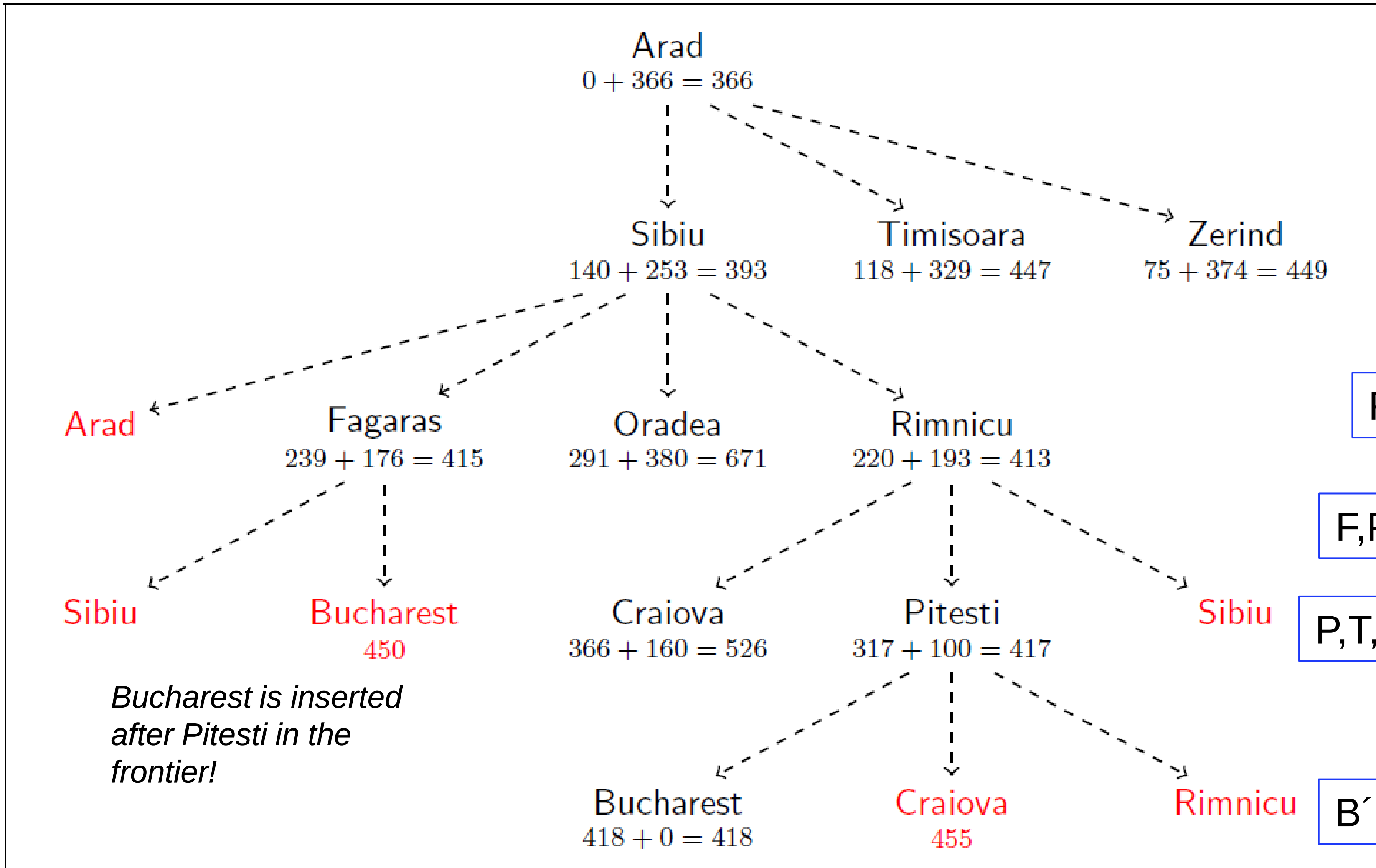


h: Aerial Distances to Bucharest

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

Computed f-Values in the Example

Frontier:



A

S,T,Z

R,F,T,Z,O

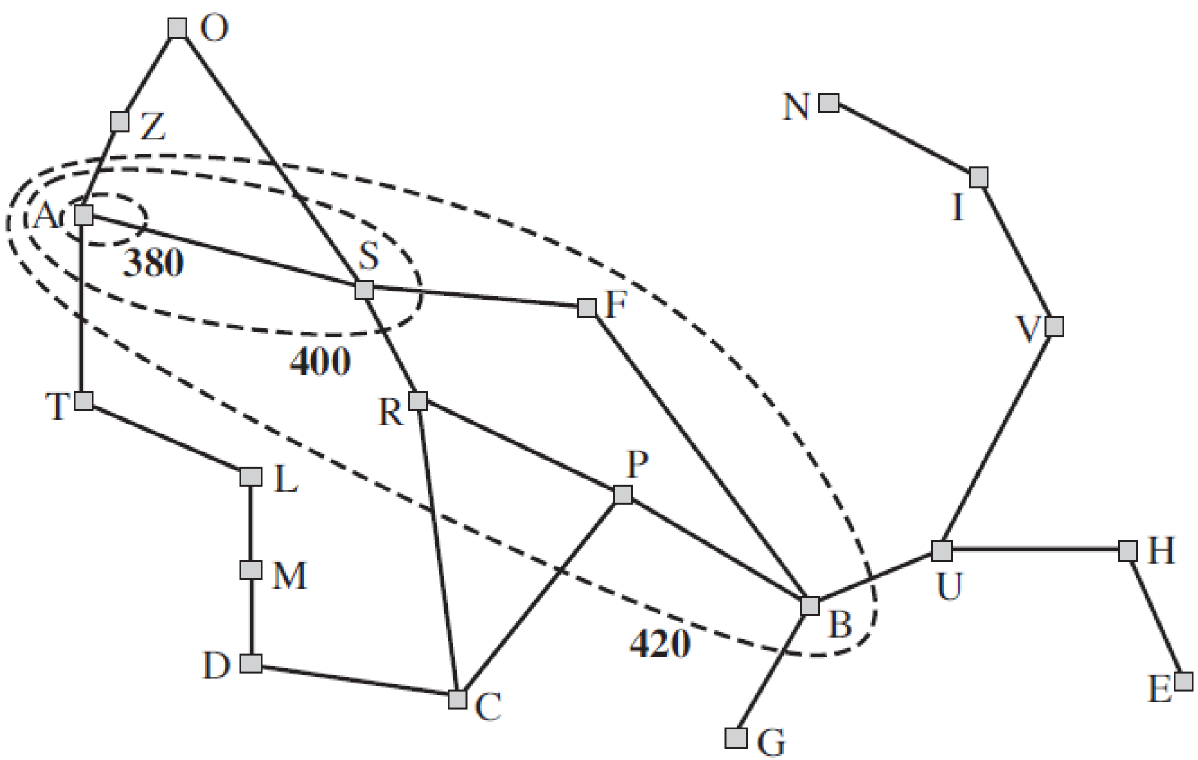
F,P,T,Z,C,O

P,T,Z,B,C´,O

B´,T,Z,C´,O

f-based Contours in the Search Space

- A* fans out from the start node, adding nodes in concentric bands of increasing f-costs
  - with good heuristics the bands stretch towards the goal state and are more narrowly focused around the optimal path



Proof of Optimality of A* under Consistent Heuristics

- The general idea for the proof is to encode the **consistent** heuristic function as action costs and establish a correspondence to uniform cost search
  - UCS is optimal for non-negative action costs (Dijkstra´s algorithm)
- We then show that the original and the transformed problem have the same optimal solutions and isomorphic search spaces
- Finally, we can prove that every optimal solution found by A* on the transformed problem, is also an optimal solution for the original problem

Step 1: Encoding Heuristic Values as Action Costs

Definition: Let Π be a problem with state space $\Theta = (S, A, c, T, I, S^G)$, and let h be a consistent heuristic function for Π. We define the h-weighted state space as $\Theta^h = (S, A^h, c^h, T^h, I, S^G)$ where:

- $A^h := \{a[s, s'] \mid a \in A,\ s, s' \in S,\ (s, a, s') \in T\}$,
- $c^h: A^h \to \mathbb{R}_0^+$ is defined by $c^h(a[s, s']) := c(s, a) - \boxed{[h(s) - h(s')]}$,
- $T^h = \{(s, a[s, s'], s) \mid (s, a, s') \in T\}$.

Remember consistency of h:

$$h(s) \leq c(s, a) + h(s')$$
$$= h(s) - h(s') \leq c(s, a)$$

Lemma: Θ^h is well-defined, i.e.

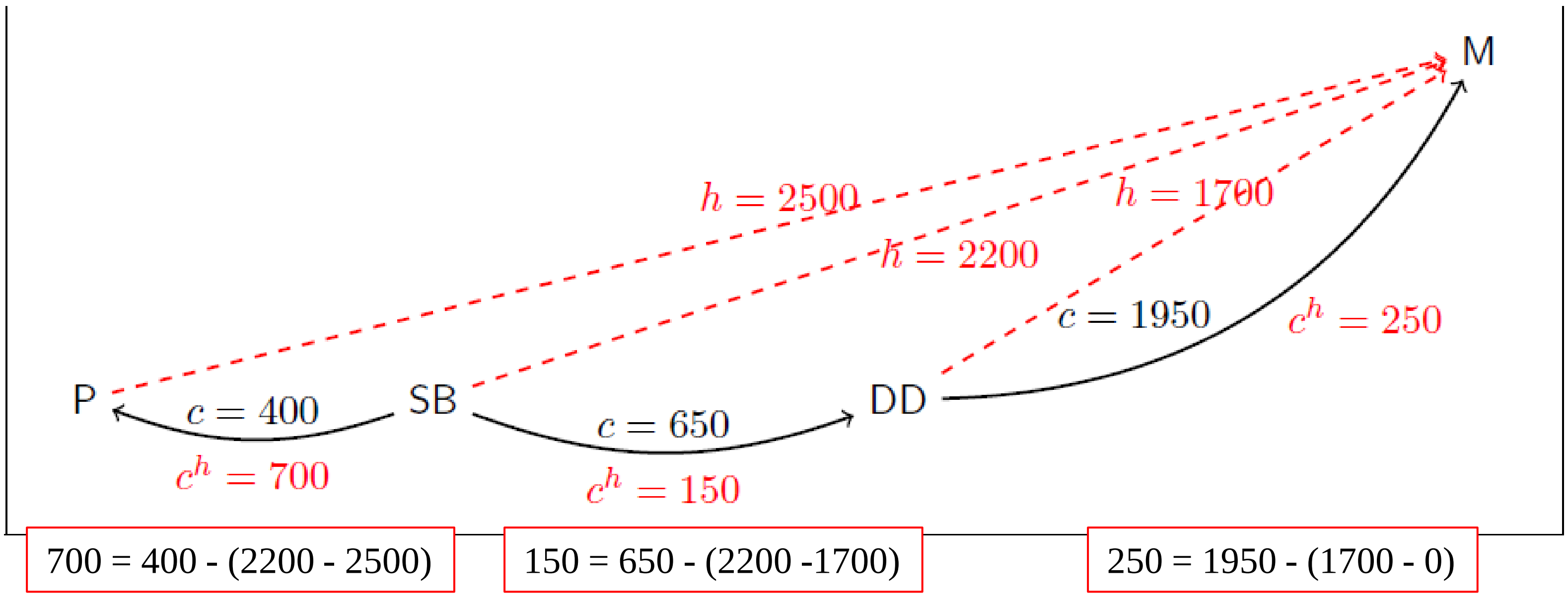
$$c(s, a) - [h(s) - h(s')] \geq 0.$$

Proof: The assumption follows immediately from consistency.

Illustration of Encoding

Example: Finding a route from SB to Moscow

- States: P (Paris), SB, DD (Dresden), M (Moscow).
- Actions: $c(\text{SBtoP}) = 400$, $c(\text{SBtoDD}) = 650$, $c(\text{DDtoM}) = 1950$.
- Heuristic (straight line distance): $h(\text{Paris}) = 2500$, $h(\text{SB}) = 2200$, $h(\text{DD}) = 1700$.



Optimal Solution: SB - DD - M $\underbrace{650 + 1950}_{\text{in } \Theta} = 2600 = \underbrace{150 + 250}_{\text{in } \Theta^h} + \underbrace{2200}_{h(\text{SB})}$

Identify the Correspondence (1)

Lemma A: Θ and Θ^h have the same optimal solutions.

Proof: Let $s_0 \xrightarrow{a_1} s_1, \ldots, s_{n-1} \xrightarrow{a_n} s_n$ be the corresponding state path of a solution in Θ, $s_n \in S^G$. The cost of the same path in Θ^h is

$Note: -[h(s) - h(s')] = -h(s) + h(s')$

$$[-h(s_0) + c(s_0, a_1) + h(s_1)] + [-h(s_1) + c(s_1, a_2) + h(s_2)] + \cdots + [-h(s_{n-1}) + c(s_{n-1}, a_n) + h(s_n)]$$

$$= -h(s_0) + \underline{c(s_0, a_1)} + \boxed{[h(s_1) - h(s_1)]} + \underline{c(s_1, a_2)} + \boxed{[h(s_2) - h(s_2)]} + \cdots \boxed{- h(s_{n-1})} + \underline{c(s_{n-1}, a_n)} + h(s_n)$$

$h(s_i) - h(s_i) = 0$
$\forall s_i \in S$

$$= \sum_{i=1}^{n} \underline{c(s_{i-1}, a_i)} - h(s_0) + h(s_n) = \left[\sum_{i=1}^{n} c(s_{i-1}, a_i)\right] - h(s_0),$$

since s_n is a goal state, it holds $h(s_n) = 0$.

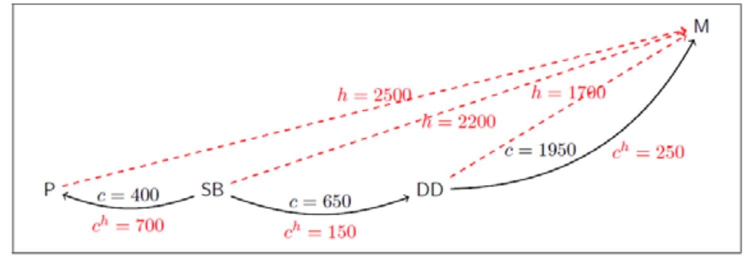
Thus, the costs of solution paths in Θ^h are those of Θ, minus a constant. The claim follows.

Identify the Correspondence (2)

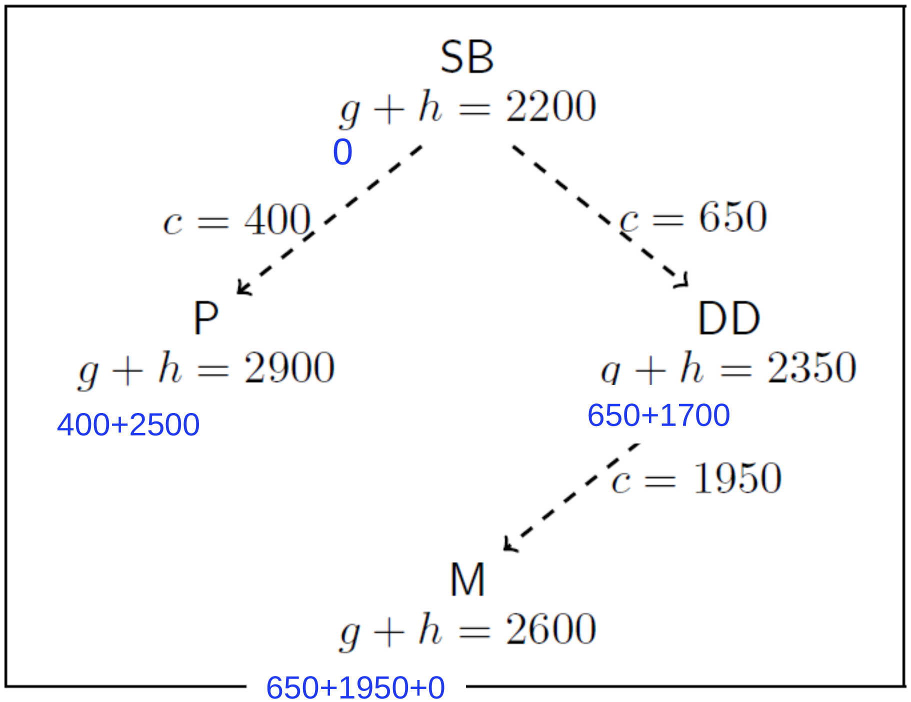
Lemma B: The search space of A* on Θ is isomorphic to that of uniform-cost on Θ^h.

Proof: Let $s_0 \xrightarrow{a_1} s_1, \ldots, s_{n-1} \xrightarrow{a_n} s_n$ be any state path in Θ. The $g + h$ value, used by A*, is $[\sum_{i=1}^{n} c(s_{i-1}, a_i)] + h(s_n)$. The g value in Θ^h, used by uniform-cost search on Θ^h, is $[\sum_{i=1}^{n} c(s_{i-1}, a_i)] - h(s_0) + h(s_n)$ (see Proof of Lemma A). The difference $-h(s_0)$ is constant, so the ordering of the frontier (open list) is the same. As the duplicate elimination is identical, the assumption is shown.

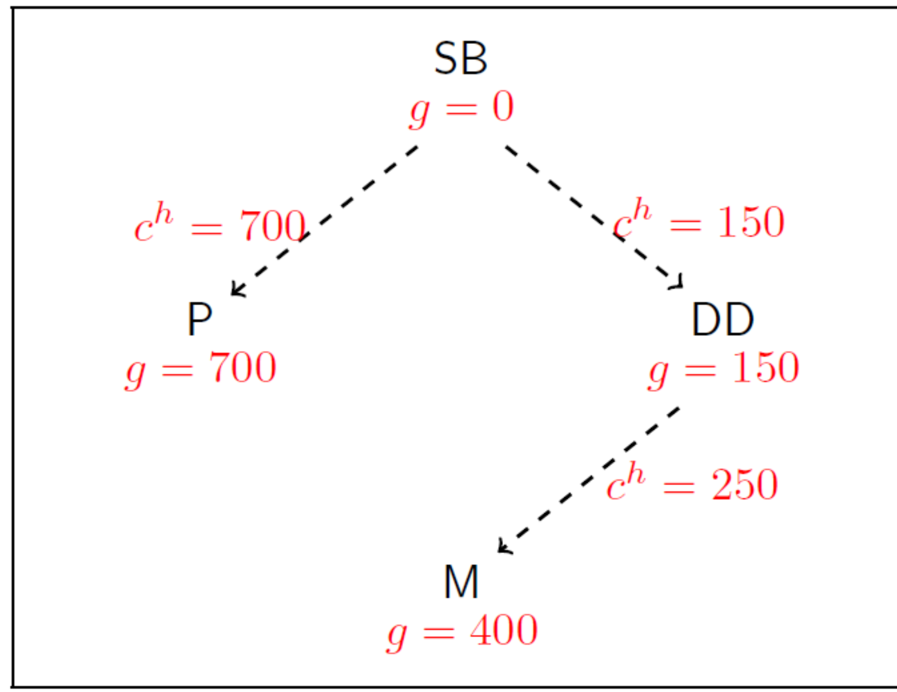
Illustration of A* and UCS Searches



$\mathbf{A}^*$ on Θ



uniform-cost search on Θ^h



Proving the Final Theorem

Theorem (Optimality of A*)
Let Π be a problem with state space Θ, and let h be a heuristic function for Θ. If h is consistent, then the solution returned by A* (if any) is optimal.

Proof Let ρ^{A^*} be the solution returned by A* run on Θ. Denote by $\mathbb{S}^{UCS}$ the set of all solutions that can possibly be returned by uniform cost search run on Θ^h.
By Lemma B we know that $\rho^{A^*} \in \mathbb{S}^{UCS}$.
By optimality of Uniform-Cost-Search, every element of $\mathbb{S}^{UCS}$ is an optimal solution for Θ^h.
Thus ρ^{A^*} is an optimal solution for Θ^h.
Together with Lemma A, this implies that ρ^{A^*} is an optimal solution for Θ.

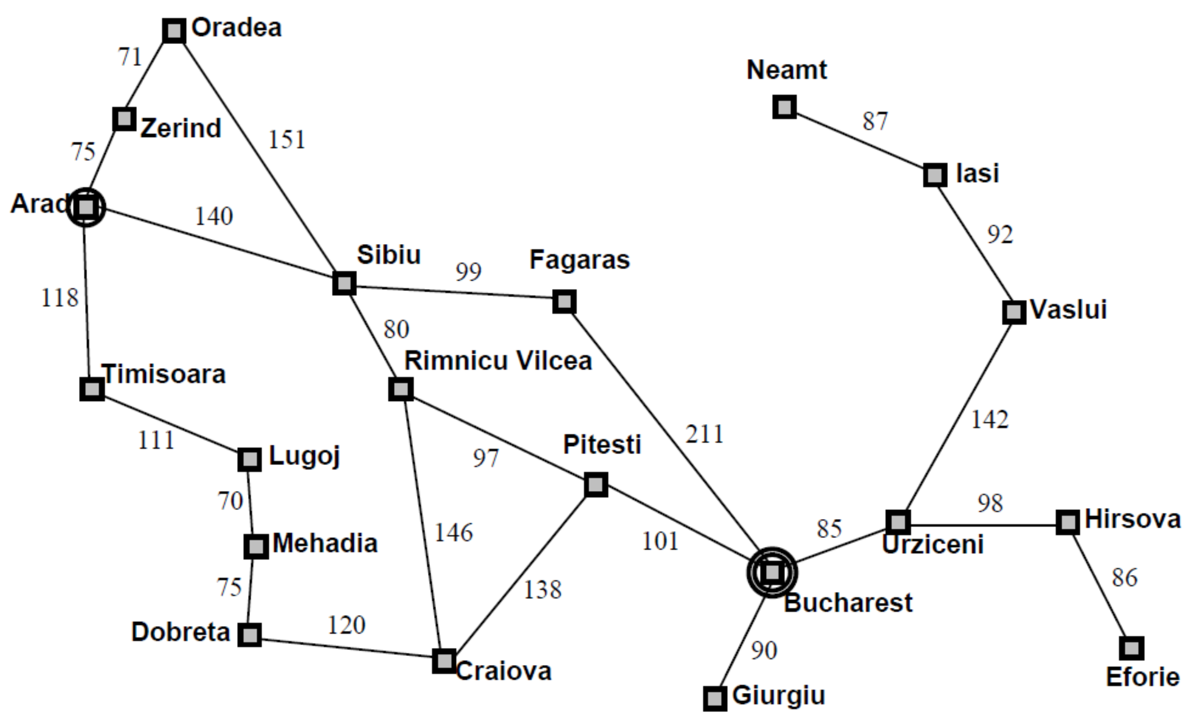
IDA* (Korf 1985)

- A* requires exponential memory in the worst case
  - combine with Depth-Limited Search
- **Idea:** use successive iterations with increasing f-costs
  - use f-bounds instead of bounding the length of the path

- At each iteration, perform a depth-first search, cutting off a branch when its total cost $(g + h)$ exceeds a given threshold
  - threshold starts at the estimate of the cost of the initial state, and increases for each iteration of the algorithm
  - at each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold

Example of IDA*

- initial f-cost limit = 366 (f-costs of the initial state)
  - first expansion: T=118 +329=447, S=140+253=393, Z=75+374=449
- next f-cost limit = 393 (S)



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

Properties of IDA*

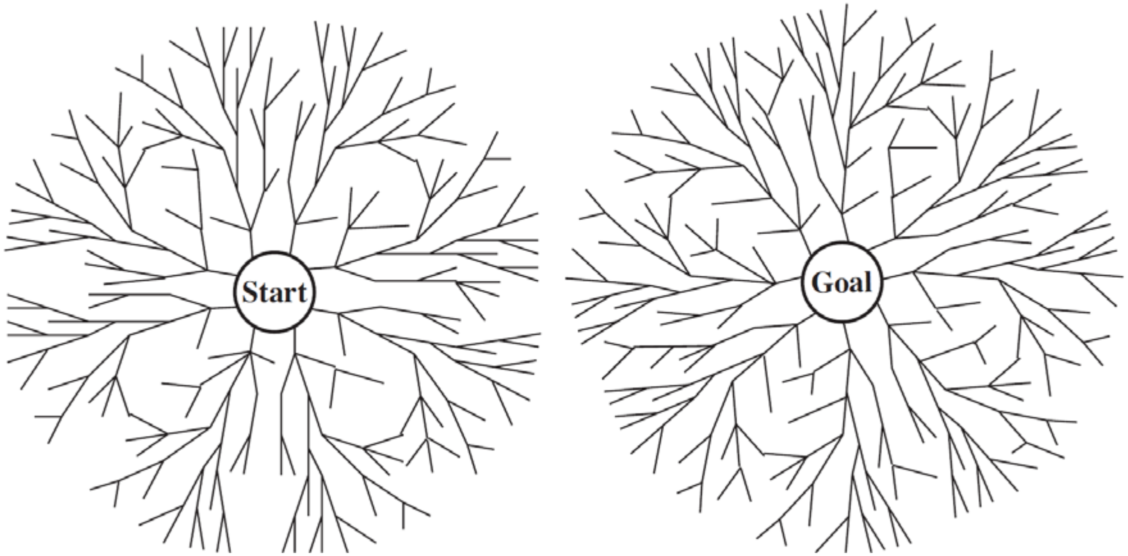
- IDA* is **complete** if every node has a finite number of successor nodes and if each action has positive and finite costs

- IDA* is **optimal**. The first solution found has minimal path costs if h is admissible (tree) or consistent (graph)

- **Time Complexity** is $O(b^d)$

- **Space Complexity** is $O(d)$

(3) Bidirectional Search (Concept of Searching)

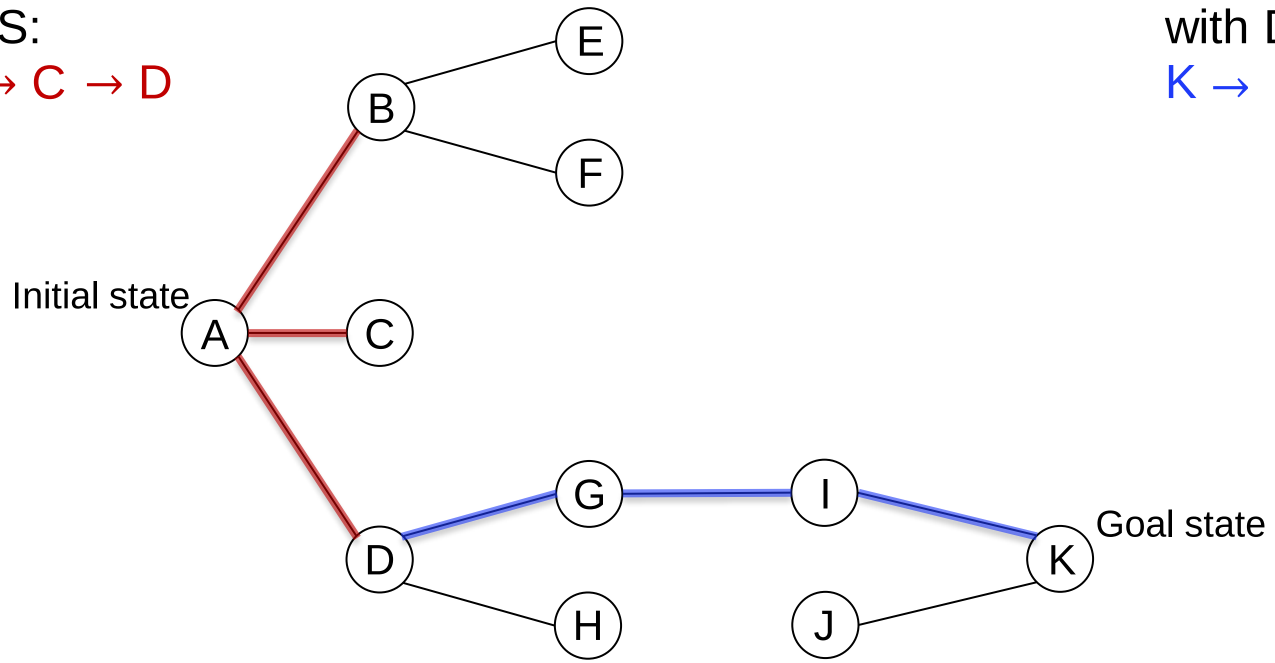


- 2 simultaneous searches: 1 forward (starting at initial state) and 1 backward (starting at goal state)
- Hoping that the two searches meet in an intersection state in the **middle**
- Motivation: $O(b^{d/2})$ is exponentially less than $O(b^{d})$
- Any search technique can be used
- Goal test ⟶ test if two searches have intersection state

Example

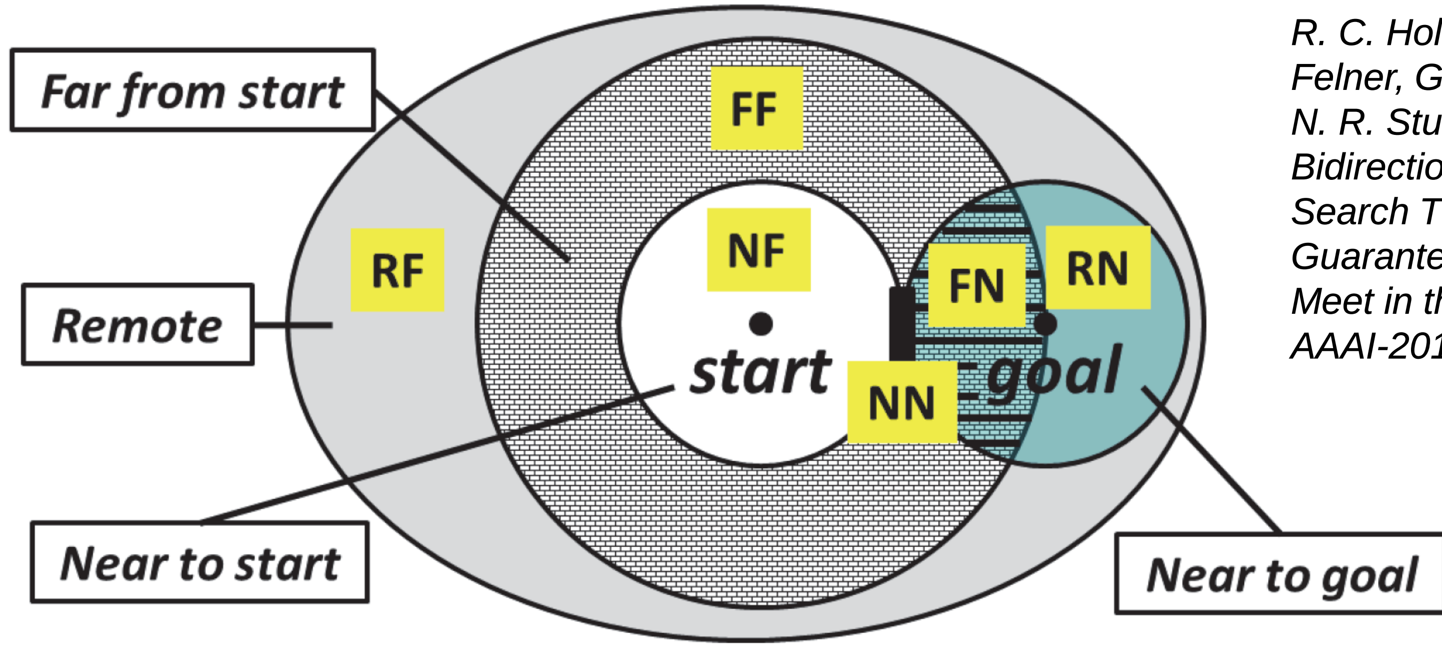
Forward search
with BFS:
A → B → C → D

Backward search
with DFS:
K → I → G → D



⟹ **Problem:** We do not necessary meet in the middle and there is no guarantee that the solution is optimal

(3) Bidirectional Search – MM Algorithm (Holte et al 2016)



R. C. Holte, A. Felner, G. Sharon, N. R. Sturtevant: Bidirectional Search That Is Guaranteed to Meet in the Middle, AAAI-2016

- First letter indicates the distance from start (N=near, F=far, R=remote) and the second letter indicates the distance from goal (N=near, F=far)
- NN includes only those states at the exact midpoint of optimal solutions

Measuring Heuristic Distances in Bidirectional Search

- A* search in both directions
- Each search direction uses an admissible **front-to-end heuristic** that directly estimates the distance from node n to the target of the search (target for forward search is the goal, target for backward search is the start state)
- $d(u, v)$ is the distance (cost of a least-cost path) from state u to state v
- $C^* = d(\text{"start"}, \text{"goal"})$ is the cost of an optimal solution
- State s is "near to *goal*" if $d(s, \text{goal}) \leq C^*/2$, and "far from *goal*" otherwise. For *start*, we make a 3-way distinction: s is "near to *start*" if $d(\text{start}, s) \leq C^*/2$, "far from *start*" if $C^*/2 < d(\text{start}, s) \leq C^*$, and "remote" if $d(\text{start}, s) > C^*$

Properties of MM

- MM is **complete**

- MM is **optimal** for non-negative action costs, the first solution found has minimal path costs if h is admissible (tree) or consistent (graph)

- If there exists a path from *start* to *goal* and MM's heuristics are consistent, MM never expands a state twice

Overview on Properties of Heuristic Algorithms

Criterion	Greedy Best-First Search	A* search	IDA* search	MM Algorithm (Bidirectional search)
Evaluation function $f(s)$	$h(s)$	$g(s) + h(s)$	$g(s) + h(s)$	$g(s) + h(s)$
Complete?	Yes[a,b]	Yes[c,d]	Yes[c,d]	Yes[e]
Time	$O(b^m)$	$O(b^d)$	$O(b^d)$	?
Space	$O(b^m)$	$O(b^m)$	$O(d)$[g]	?
Optimal?	No	Yes[e]	Yes[e]	Yes[f]

Where:

d depth of solution

m maximum depth of the search space

Superscripts:

a for finite state spaces

b with duplicate elimination

c if every node has a finite number of successor nodes

d if each action has positive and finite costs

e 1st solution found has minimal path costs if h is admissible (tree) or consistent (graph)

f for non-negative action costs

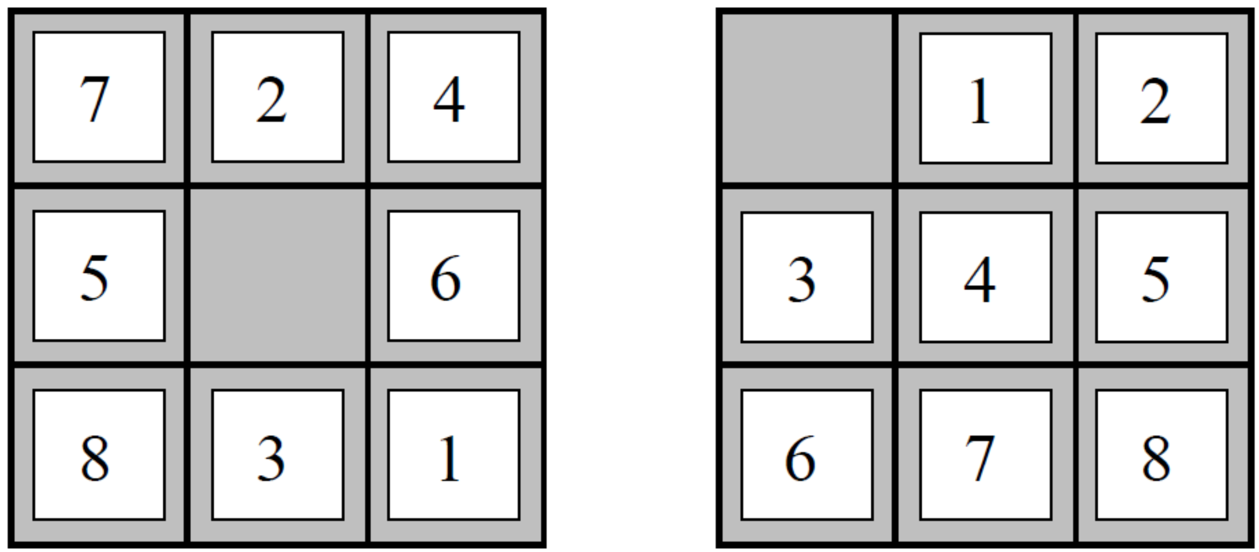
g with backtracking search, else $O(bd)$

Designing Heuristic Functions

- The ***informedness*** of the heuristic is critical for the success of the search algorithm
  - steer the algorithm towards the most promising parts of the search space
  - recognize dead ends early
  - find a near-optimal solution under practical conditions

- Requires an understanding of the application domain
  - keep heuristic and search approach separate
  - try out different variants in empirical tests
  - an art form and a hot topic in AI research

Heuristic Functions Example

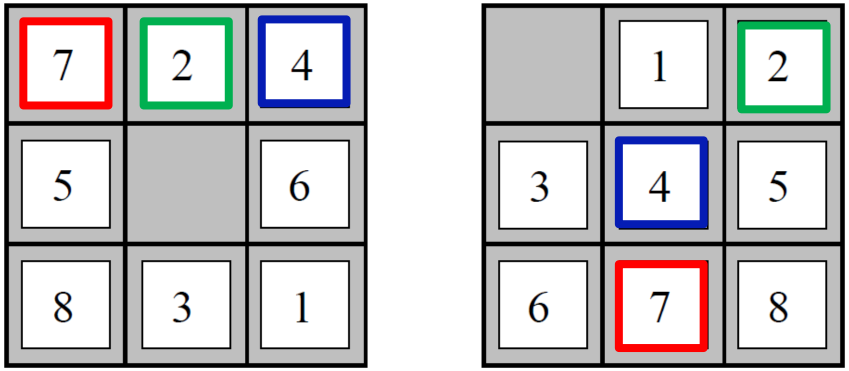


Start State Goal State

h_1 corresponds to the number of tiles in the wrong position (Misplaced Tiles)

h_2 corresponds to the sum of the distances of the tiles from their goal positions (Manhattan distance)

Misplaced Tiles vs. Manhattan Distance



initial state *goal state*

$h_1 = 8$ (all tiles are misplaced)

$h_2 = 3 + 1 + 2 + \ldots = 18$

- Distance between two points measured along axes at right angles

goal position

$$h(s) = \sum_{i=1}^{s} \left(|x_i(s) - \bar{x_i}| + |y_i(s) - \bar{y_i}| \right)$$

current position

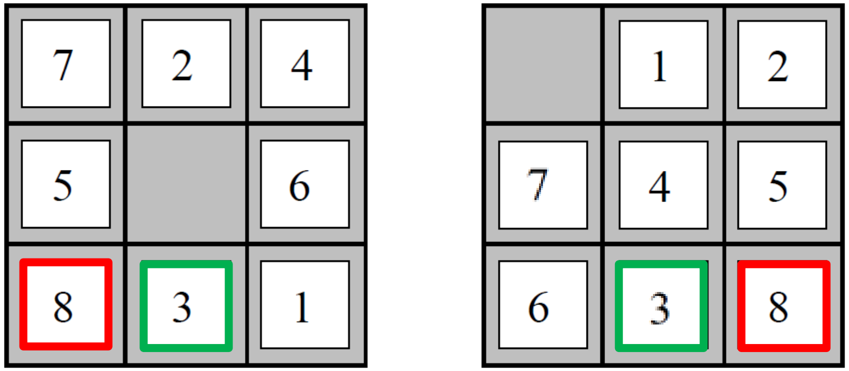
- **Disadvantage:** considers all tiles independently

Empirical Comparison of Both Example Heuristics

- d = distance from goal
- Average over 100 instances

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A* (h_1)	A* (h_2)	IDS	A* (h_1)	A* (h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	-	539	113	-	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.47
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

Linear Conflict Heuristic vs. Manhattan Distance

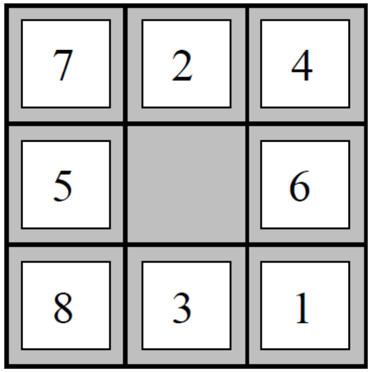


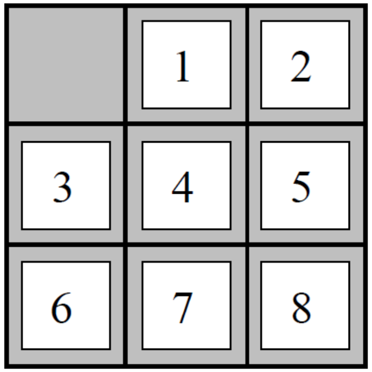
LC: $2 + 2 = 4$

MH: $2 + 0 = 2$

- Two tiles t_j and t_k are in a linear conflict if t_j and t_k are in the same line, the goal positions of t_j and t_k are both in that line, t_j is to the right of t_k and the goal position of t_j is to the left of the goal position of t_k
  - LC will add a cost of 2 moves for each pair of conflicting tiles to the Manhattan Distance
  - Motivation: Tiles need to surround one another
- LC is consistent and more informative than MH

Gaschnig's Heuristic





9 stands for the blank

transform	724596831
into	912345678

Relaxed Move: Any tile can be moved to the blank position (count the number of swaps)

loop until solution is found:

If 9 is not at the 1st position

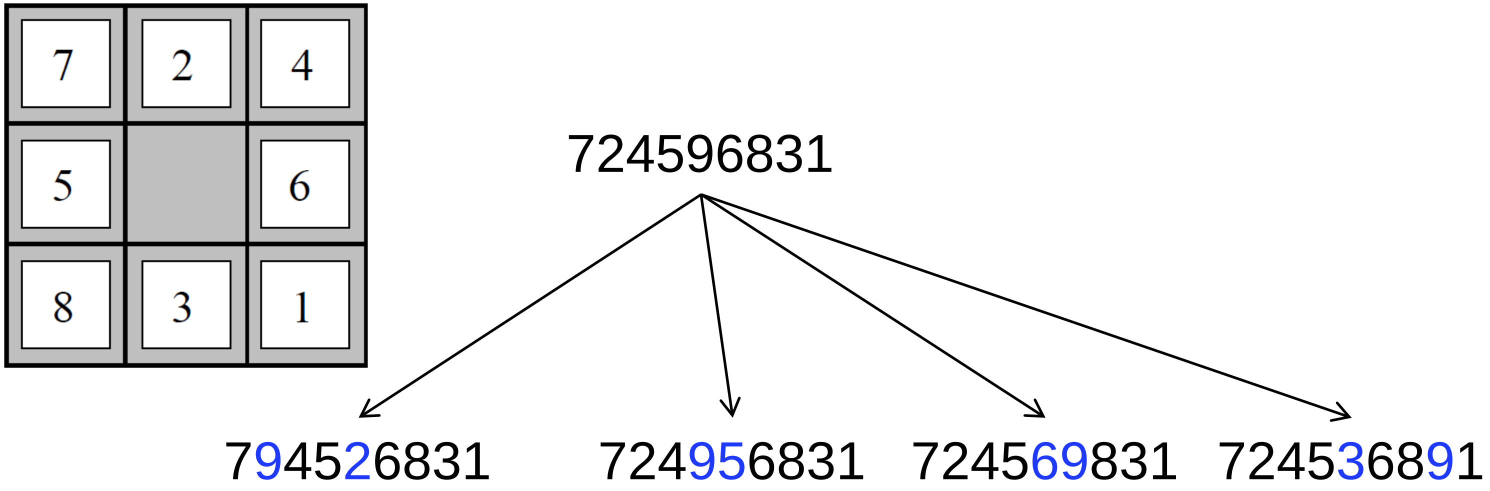
then swap 9 with the element whose target position 9 is taking

else swap 9 with the rightmost element that is not in its proper place

724596831 – 729546831 – 792546831 – 712546839 – 712546938 – 712549638 – 712945638 – 712345698 – 912345678

= 8 swaps

Applying Gaschnigs Heuristic During Search



- 4 successor nodes
- Compute the number of swaps for each node
- Expand the node with the fewest number of swaps first

➢ Problem relaxation is a powerful idea and very successfully used to derive good heuristics

Problem Relaxation on Whitebox Description

- Primitive Predicates in the N-Puzzle
  - ON(t, y) : tile t is on cell y
  - CLEAR(y) : cell y is clear of tiles
  - ADJ(y, z) : cell y is adjacent to cell z

- **Move(t, y, z)**

  preconditions : ON(t, y) & CLEAR(z) & ADJ(y, z)

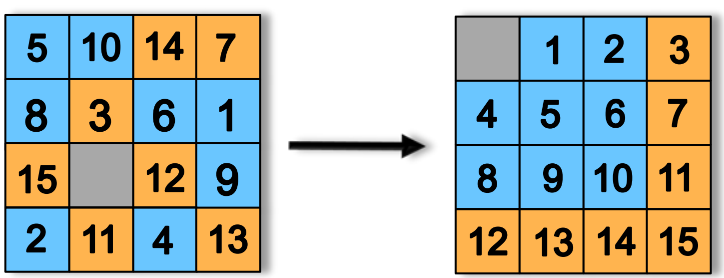
  effects : ON(t, z) & CLEAR(y) &

  NOT ON(t, y) & NOT CLEAR(z)

- Remove CLEAR(z) & ADJ(y, z) – Misplaced Tile heuristic
- Remove CLEAR(z) – Manhattan Distance heuristic
- Remove ADJ(y, z) – Gaschnig's heuristic

Pattern Database

- Apply the Divide and Conquer principle
  - decompose the problem into subproblems (subgoals)
  - store solutions to the subproblems with associated costs (patterns)
  - reuse these solutions



Divide the 15 puzzle into Fringe + 8 puzzle

- map the current location of the fringe tiles into an index of the database
- the data base tells us the minimal number of moves to achieve the fringe
- achieve the fringe + solve the remaining 8 puzzle

- Famous example: end game libraries in chess

Learning Heuristic Functions

- Relaxed problem heuristic
  - problem with fewer restrictions on the actions is called a relaxed problem
  - cost of an optimal solution path to a relaxed problem is an admissible heuristic for the original problem

- Modern search algorithms analyze the domain <u>and</u> the given problem instance
  - learn a problem-instance specific heuristic before they start searching

Summary

- Heuristic search is the preferred search method for medium size search spaces
- The effectiveness of heuristic search depends on the properties of the heuristic function: efficient to compute, informative, admissible, consistent
- Only recently, a bidirectional search algorithm was developed, which is guaranteed to meet in the middle, but does not gurantee to reduce search and memory costs in the worst case
- Greedy best-first search often quickly finds good solutions in practice
- A* is optimal on graphs when using consistent heuristics
- Finding good heuristics can be done by analyzing the search problem, e.g. using relaxation methods

Working Questions

1. Explain the underlying ideas of greedy (best-first) search, A* and IDA*.
2. What are the properties of A*?
3. Why can IDA* be more efficient than A* in practice?
4. What are heuristics, which role do they play in informed search?
5. Which properties have good heuristics?
6. What is an admissible/consistent heuristic function?
7. What can happen with A* if the heuristic function is non-admissible / non-consistent?
8. What is important for bidirectional search to work?