UNIVERSITÄT DES SAARLANDES

Artificial Intelligence

Heuristic (informed) Search

Prof. Dr. habil. Jana Koehler

Dr. Sophia Saller, M. Sc. Annika Engel

Summer 2021

Agenda

- Using Knowledge during Search
  - Evaluation of search states
  - Admissible and consistent (monotone) heuristics
- Heuristic (Informed) Algorithms
  - Greedy (Best-First) Search
  - A* and IDA*
  - 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

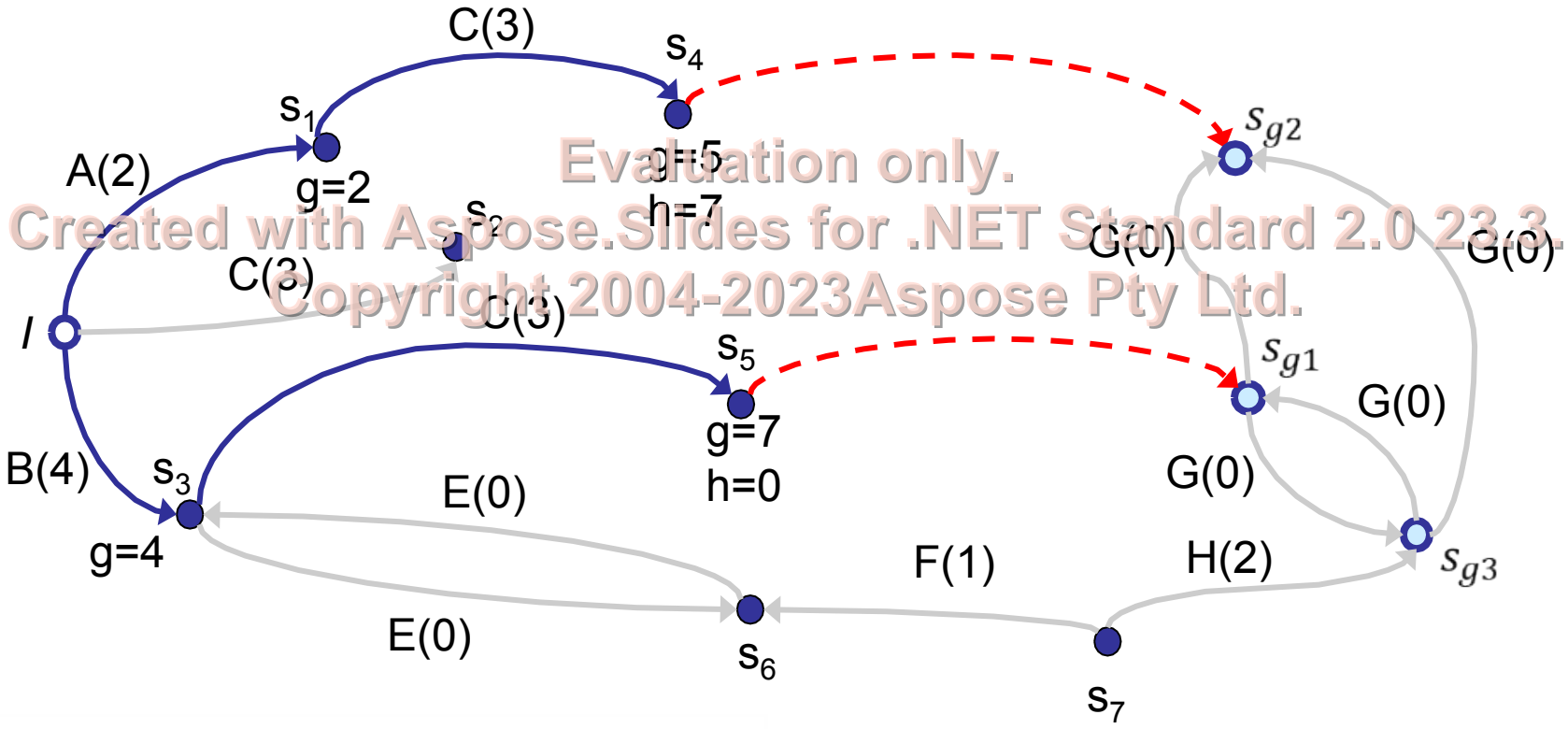
Using Knowledge during Search

How to Determine the next Node for Expansion?

- Uninformed Search
  - Systematic rigid procedure
  - No knowledge used to estimate the distance from a current state to a goal
  - Frontier organized as FIFO, LIFO, or priority queue
- Informed Search
  - “Value" of expanding a node used as guidance that steers the search algorithm through the search space
  - Evaluation function $f(n)$ assigns a number to each node

The Evaluation Function

$$f(n) = g(n) + h(n.\text{State})$$



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

- a precise value

+

$h(n.\text{State})$ corresponds to the estimated costs from the current state s to the goal state s_g

- an estimated value

Heuristic Function

Let Π be a problem with state space Θ. A heuristic function, short heuristic, for Θ is a function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$,

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

The value of h depends only on the state s, not on the path that the algorithm followed so far to construct the partial solution (and the costs g of this path)

Perfect Heuristic Function

The perfect heuristic h^* is the function assigning every $s \in S$ the cost of a cheapest path from s to a goal state

- $h^*(s) = \infty$ for dead-end states, from which the goal is unreachable
- $h^*(s)$ is also called the goal distance of s

Desirable Properties of Heuristic Function

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

ØGOOD heuristics satisfy a balance between (1) and (2) and satisfy properties (3) to (4) at least
ØProperty (5) ensures effective dead-end recognition
ØProperty (6) is a prerequisite for algorithms to guarantee minimal-cost (optimal) solutions

Admissiblity of Heuristic Functions

- Let Π be a problem with state space Θ and let h be a heuristic function for Π. 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 a state s to a goal state.

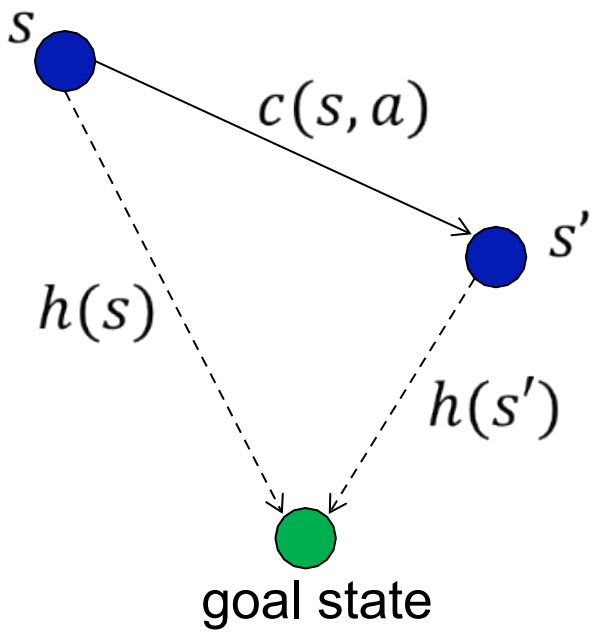
- The function h is an optimistic estimation of the costs that actually occur. It underestimates the real costs and provides the search algorithm with a lower bound on the goal distance.

Consistency (Monotonicity) of Heuristic Functions

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

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

The value $c(s,a)$ is the action cost of getting from s to s' with action a.



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

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.

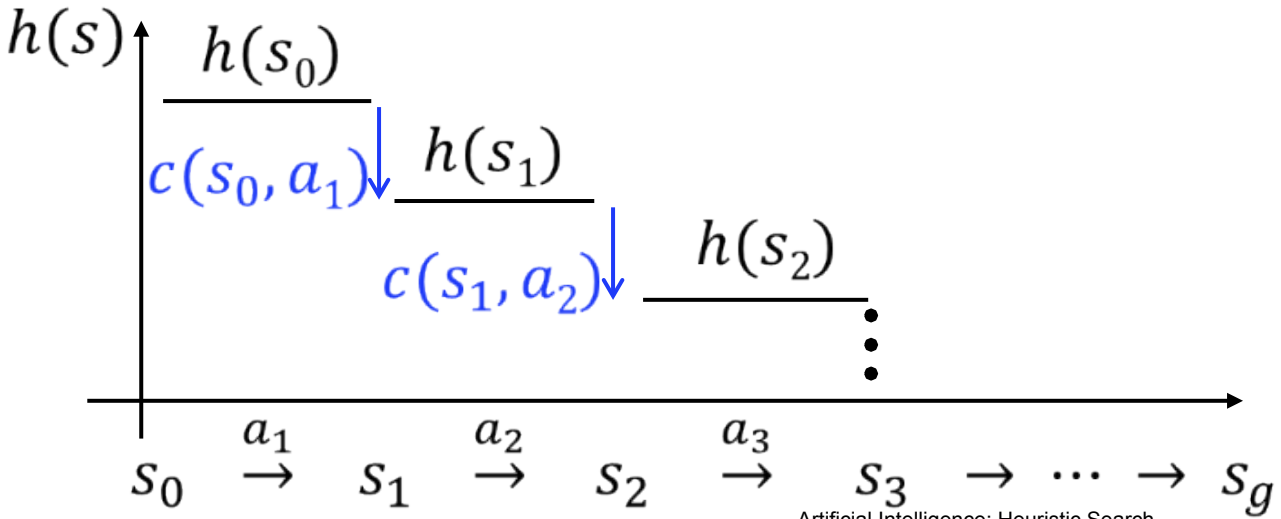
Consistency implies Admissibility

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.



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

UNIVERSITÄT DES SAARLANDES

Proof that Consistency implies Admissibility

We need to show that it holds $h(s) \leq h^*(s)$ for all 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: $h(s) \leq h(s') + c(s,a)$
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

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

QED

Heuristic (Informed) Algorithms

Greedy Best-First Search (GBFS)

- Uses only the heuristic part of the evaluation function

$$f(n) = h(n.\text{State})$$

- 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

$$f(n) = g(n)$$

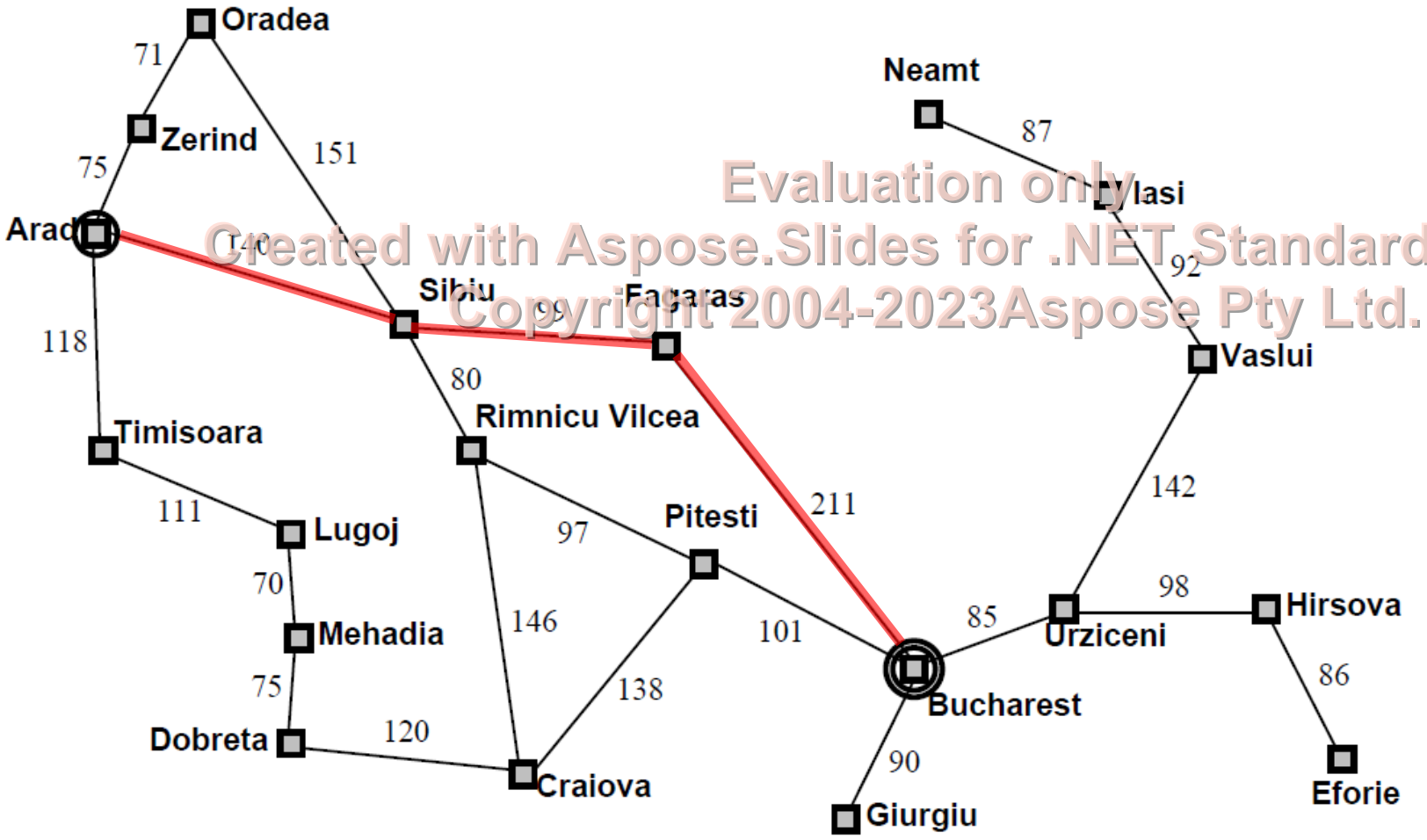
Greedy Best-First Search (GBFS) Algorithm

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

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

UNIVERSITÄT DES SAARLANDES

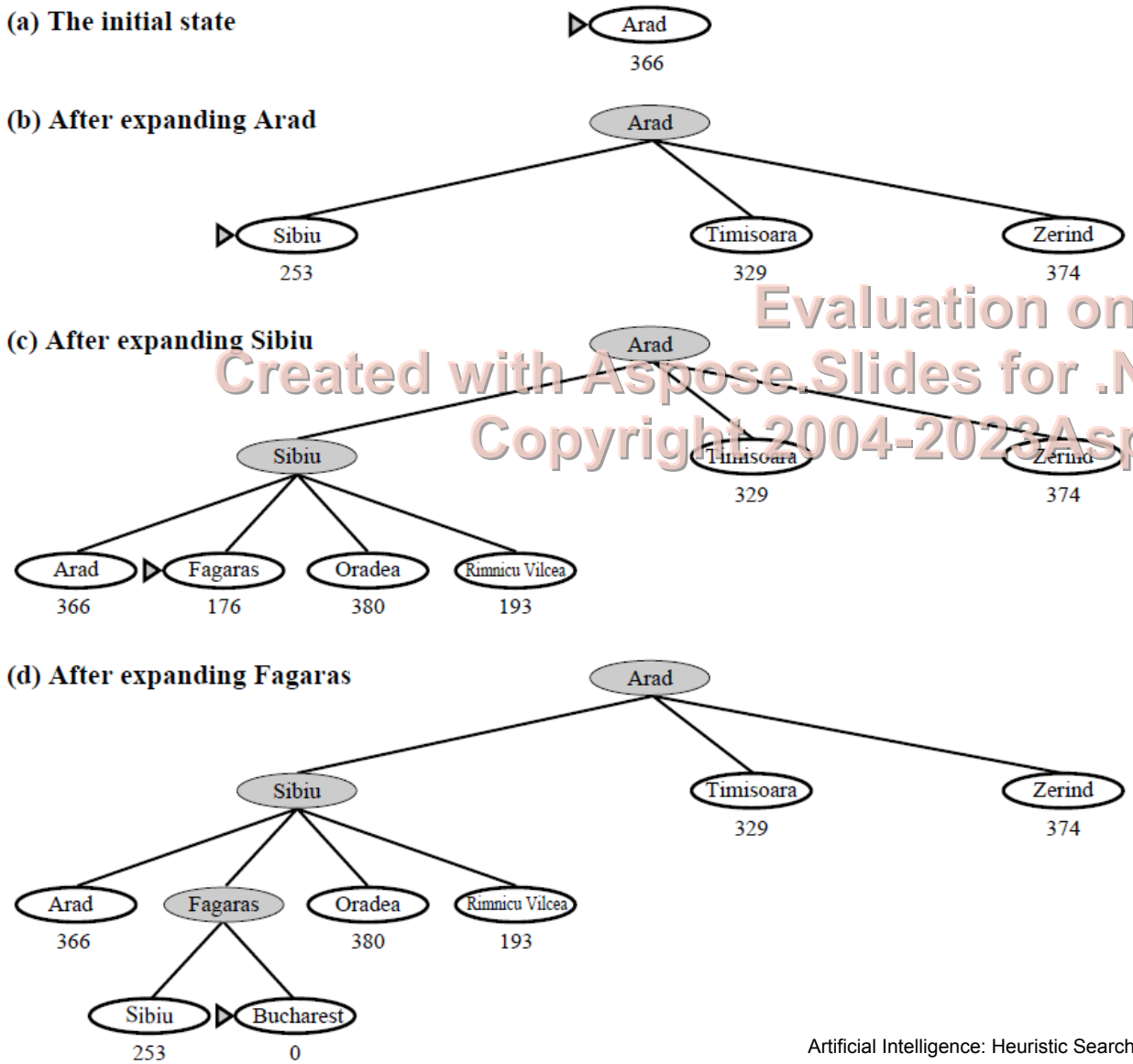
Greedy Best-First Search (GBFS) on the Romania Travel Example



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

Greedy Best-First Search (GBFS) on the Romania Travel Example



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 and Complexity of Greedy Best-First Search (GBFS)

- **Complete**
  - for finite state spaces and with duplicate elimination
- **Not optimal**
- **Exponential time complexity** is $O(b^m)$
- **Exponential space complexity** is $O(b^m)$

where m is the maximum depth of the search space

A* (Hart, Nilsson, Raphael 1968)

- Greedy search only uses $h(n.\text{State})$
- Uniform-Cost search uses $g(n)$
  - finds an optimal solution if path costs grow monotonically: $g(n) \leq g(n')$
- A* uses $f(n) = g(n) + h(n.\text{State})$
- *A** combines current path costs and estimated goal distance 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 with *node*.State = *problem*.InitialState
  frontier ← a priority queue ordered by ascending $g + h$, with *node* as the only element
  explored ← empty set of states
  loop do
    if Empty?(*frontier*) **then return** failure
    node ← Pop(*frontier*)
    if GoalTest(*node*.State) **then return** Solution(*node*)
    explored ← *explored* ∪ *node*.State
    for each *action* **in** Actions(*node*.State) **do**
      child ← ChildNode(problem, *node*, *action*)
      if *child*.State ∉ *explored* ∪ States(*frontier*) **then**
        Insert(*child*, $g(child) + h(child)$, *frontier*)
      else if there exists *otherNode* ∈ *frontier* s.t. *child*.State = *otherNode*.State
          and $g(child) < g(otherNode)$ **then** replace *otherNode* in *frontier* with *child*

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

Properties of A*

- A* is **complete**
  - If a solution exists, A* will find it provided that
    1) each 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 costs if h is admissible on trees or if h is consistent on graphs
  - If h is admissible and the search space is a graph it is not guaranteed to be optimal, but by using a technique called re-opening we can make A* optimal even in this case

Re-opening of Nodes in A*

If the heuristic is only admissible and the search space is a graph

- A* needs to expand all nodes with $f(n) \leq C^*$ (the cost of an optimal solution) to ensure optimality

- It is possible that the path found to a state s at the time it is inserted into the explored set is not the optimal path to this state

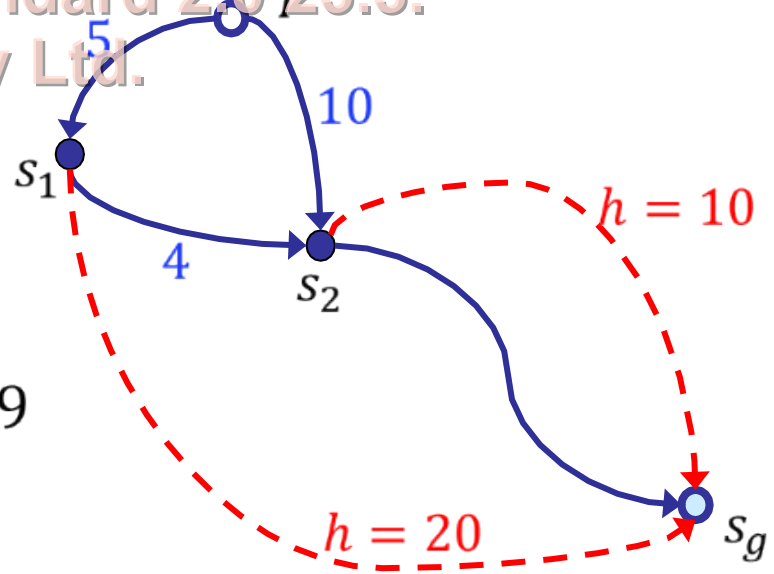
- To ensure optimality, the algorithm needs to add a node from the explored set back into the frontier if it finds a shorter path to the node during search

Example of Re-opening in A*

Red numbers show values of an admissible heuristic, blue numbers are action costs

- Nodes of s_1 and s_2 with evaluation value $5 + 20 = 25$ and $10 + 10 = 20$ resp. are in the frontier
- First, s_2 with $g = 10$ is inserted into the explored set
- As the algorithm proceeds, it finds a shorter path from I to s_2 via s_1 with $g = 9$
  $\Longrightarrow$ s_2 is removed from the explored set and a new node for state s_2 with evaluation $5 + 4 + 10 = 19$ is added to the frontier

cheaper path to s_2 having costs 9 instead of 10



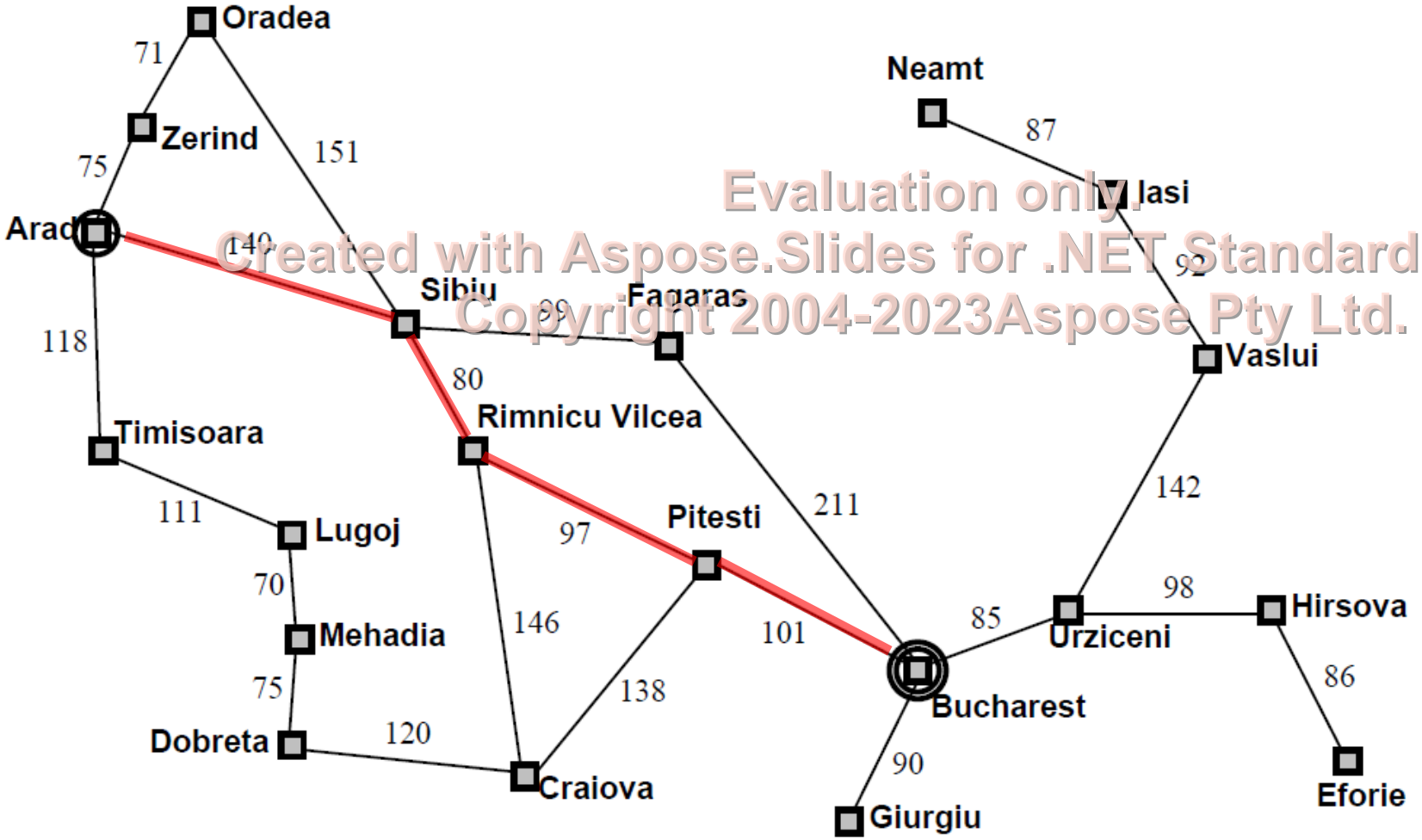
Complexity of A*

- **Exponential time complexity:** $O(b^d)$

- **Exponential space complexity** is $O(b^m)$, where m is the maximum depth of the search space
  - If state space is a tree (only one path to every state) and there is only one goal state: space complexity is linear in the length of the solution [Gaschnig 1977]
  - But if Gaschnig´s additional restrictions do not hold: space complexity is exponential even for very simple problems and for $c = 1$ with $|h(n.\text{State}) - h^*(n.\text{State})| \leq c$ for a constant c (absolute error) [Helmert and Röger 2008]

Gaschnig, John. “Exactly good are heuristics?: Towards a realistic predictive theory of best-first search". In Proceedings of the 5th International Joint Conference on Artificial Intelligence (*IJCAI’77)*, pp. 434-441, 1977.
Helmert, Malte and Röger, Gabriele. “How good is almost perfect?". In Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI’08), pp. 944-949, 2008.

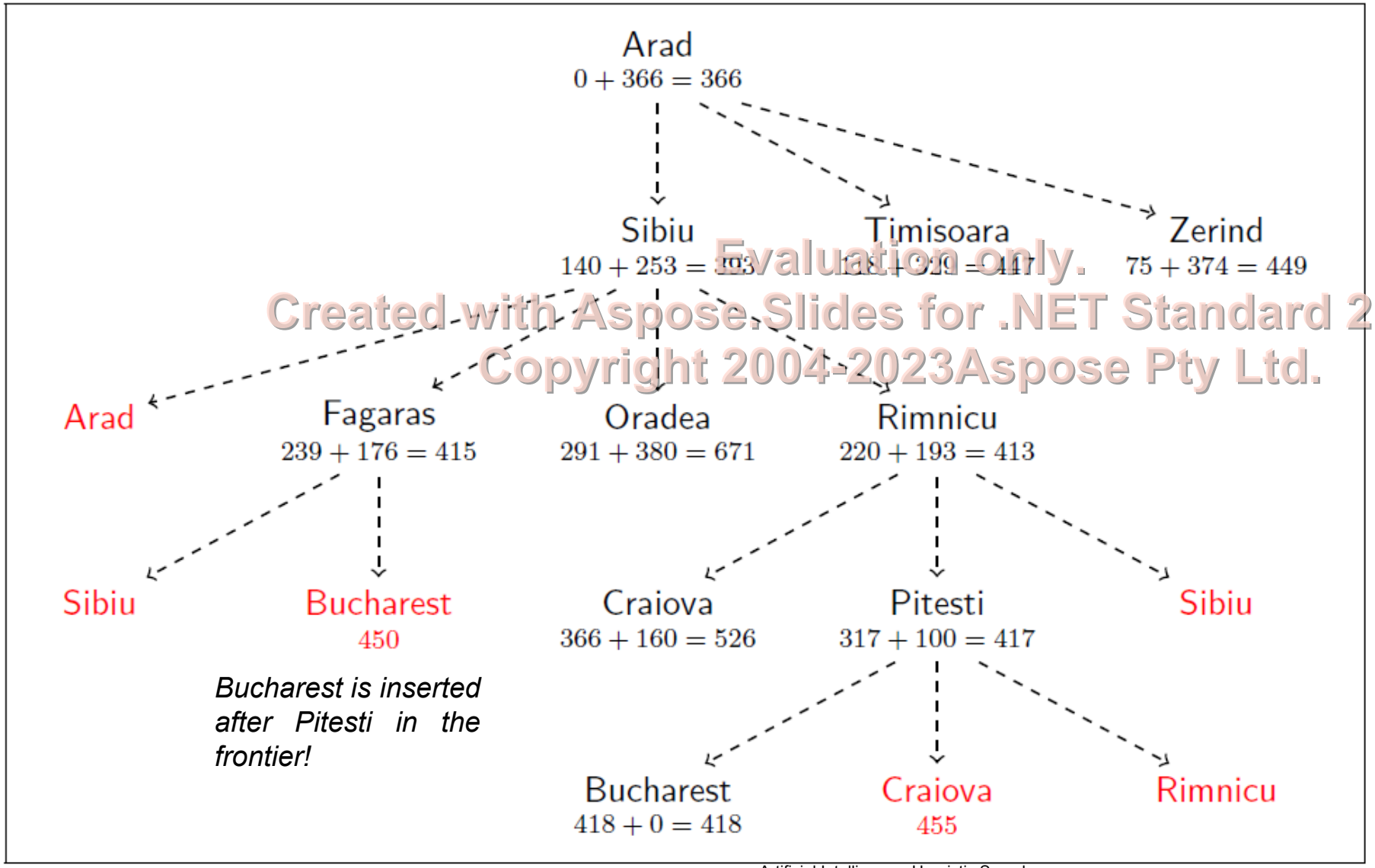
A* on the Romania Travel Example



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 by A* in the Romania Travel Example



Nodes in Frontier ordered by increasing f-values:

A

S,T,Z

R,F,T,Z,O

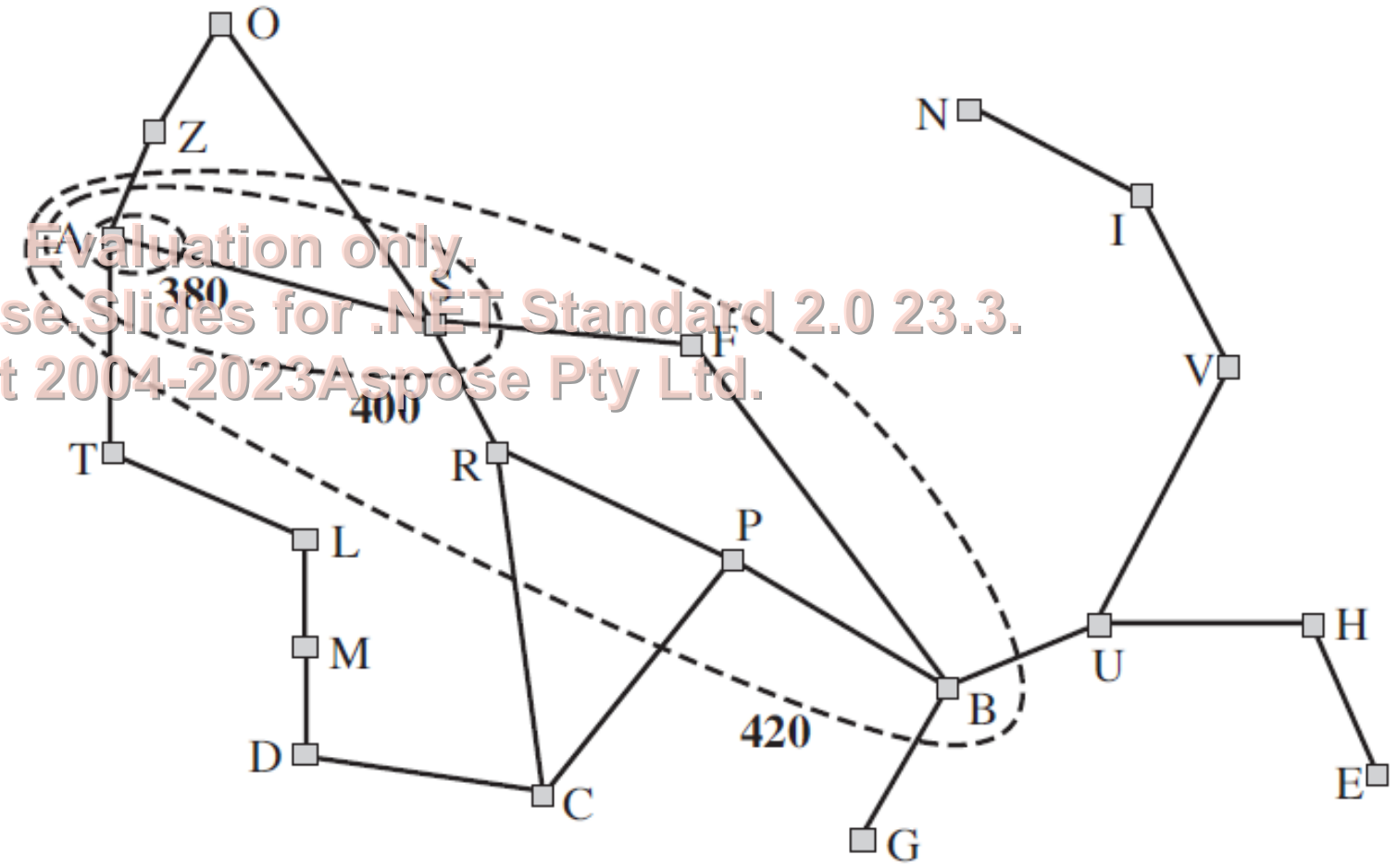
F,P,T,Z,C,O

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 contours stretch towards the goal state and are narrowly focused around the optimal path



Proof of Optimality of A* under Consistent Heuristics

- Step 1: 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)

- Step 2: We show that the original and the transformed problem have isomorphic search spaces and the same optimal solutions

- Step 3: 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 (S, A, c, T, I, S^G), and let h be a consistent heuristic function for Π. We define the h-weighted problem as $\Pi^h = (S, A^h, c^h, T^h, I, S^G)$ 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 $c^h(a[s, s']) := c(s, a) - [h(s) - h(s')]$
- $T^h = \{(s, a[s, s'], s') \mid (s, a, s') \in T\}$

Remember consistency of h:

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

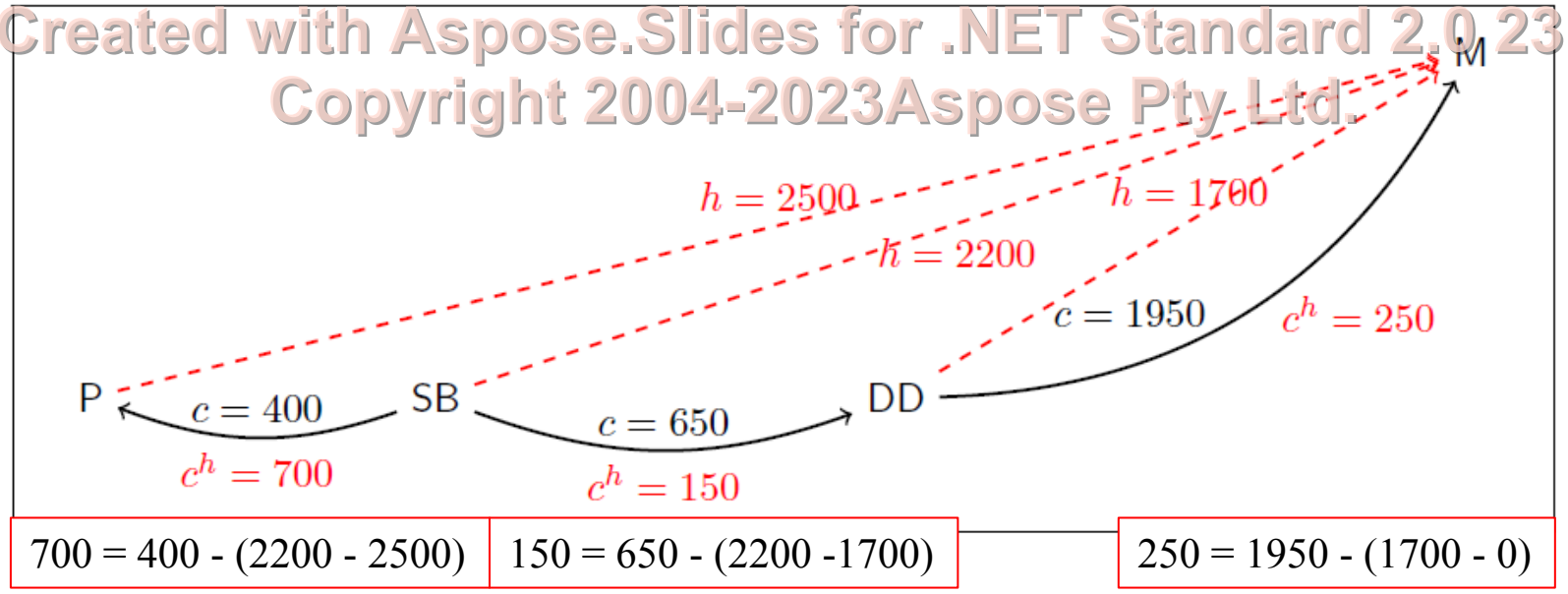
Lemma: Π^h is well-defined, i.e.,

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

Proof: The assumption follows immediately from consistency.

Step 1: Illustration of Encoding

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



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

Step 2: Identify the Correspondence

Lemma A: Π and Π^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 Π, $s_n \in S^G$. The cost of the same path in Π^h is

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

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

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

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

since s_n is a goal state, it holds $h(s_n) = 0$. Thus, the costs of solution paths in Π^h are those of Π, minus a constant.

The claim follows.

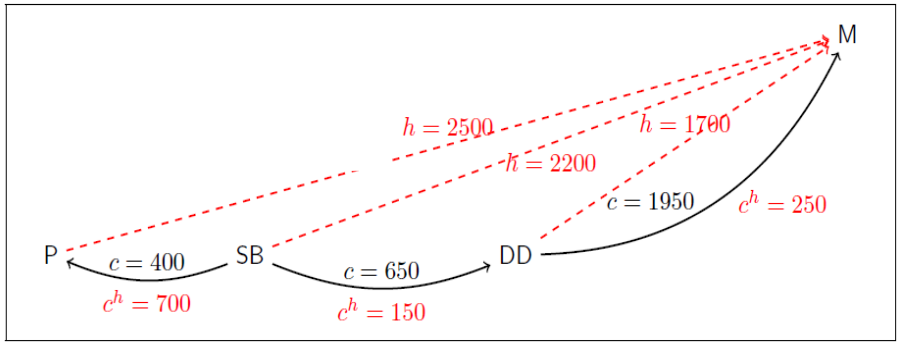
Note: $-[h(s) - h(s')] = -h(s) + h(s')$

Step 2: Proof of Isomorphism

Lemma B: The search space of A* on the problem Π is isomorphic to the search space of uniform-cost search on the transformed problem Π^h.

Proof: Let $s_0 \xrightarrow{a_1} s_1, \ldots, s_{k-1} \xrightarrow{a_k} s_k$ be any path in Π to some state s_k. The $g + h$ value, used by A*, is $\left[\sum_{i=1}^{k} c(s_{i-1}, a_i)\right] + h(s_k)$. The g value in Π^h, used by uniform-cost search on Π^h, is $\left[\sum_{i=1}^{k} c(s_{i-1}, a_i)\right] - h(s_0) + h(s_k)$ (see Proof of Lemma A). The difference $-h(s_0)$ is a constant, so the ordering of the frontier is the same. As duplicate elimination is identical, the lemma is proven: both algorithms will expand the same nodes in the same order.

Illustration of A* and UCS Searches



A* on Π

SB
$g + h = 2200$
0

$c = 400$ | $c = 650$

P
$g + h = 2900$
400 + 2500

DD
$g + h = 2350$
650 + 1700

$c = 1950$

M
$g + h = 2600$
650+1950+0

Uniform-Cost Search on Π^h

SB
$g = 0$

$c^h = 700$ | $c^h = 150$

P
$g = 700$

DD
$g = 150$

$c^h = 250$

M
$g = 400$

Step 3: Proving the Final Theorem

Theorem (Optimality of A*)

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

Proof Let ρ^{A^*} be the solution returned by A* for Π. Denote by $\mathbb{S}^{UCS}$ the set of all solutions that can possibly be returned by uniform cost search for Π^h.

By Lemma B we know that $\rho^{A^*} \in \mathbb{S}^{UCS}$ because search spaces are isomorphic.

By optimality of uniform cost search, every element of $\mathbb{S}^{UCS}$ is an optimal solution for Π^h.

Thus, ρ^{A^*} is an optimal solution for Π^h.

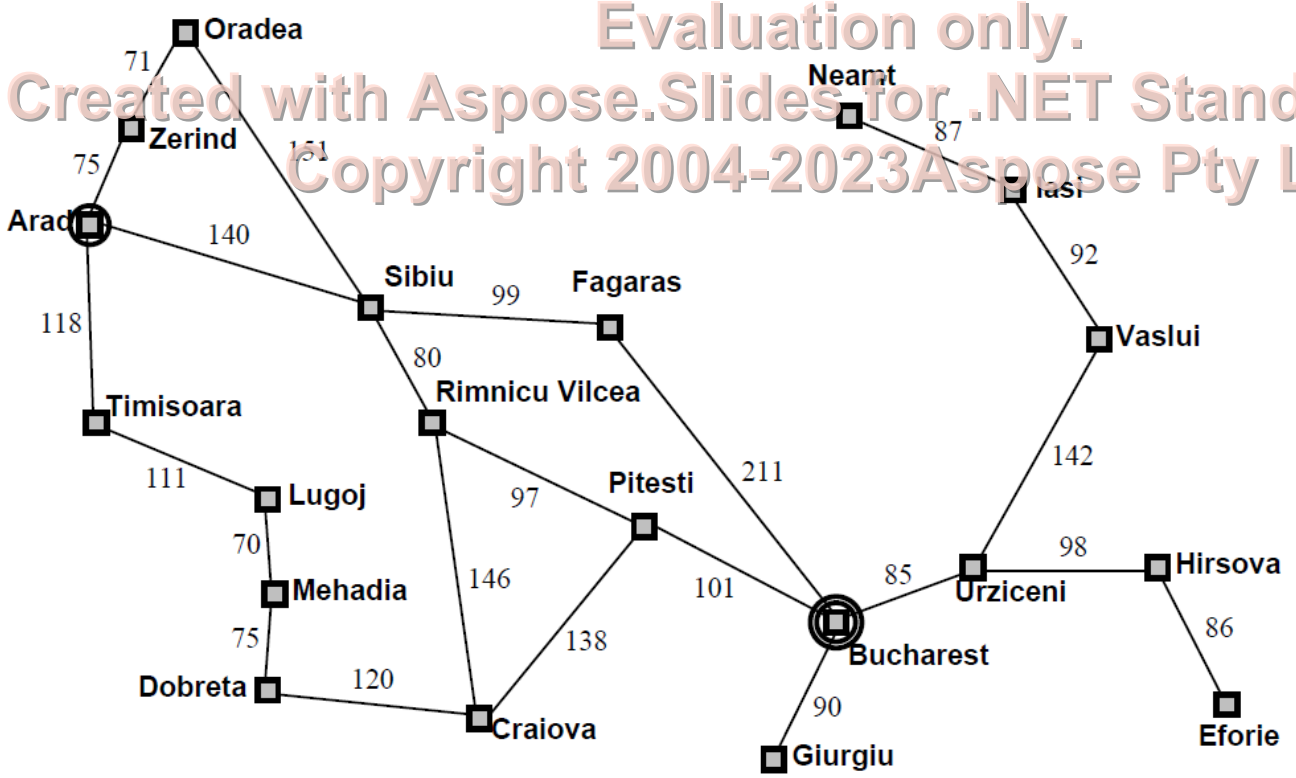
Together with Lemma A, this implies that ρ^{A^*} is an optimal solution for Π, because both problems have the same optimal solutions.

Iterative Deepening A* - IDA* (Korf 1985)

- A* requires exponential memory in the worst case
  - Combine node evaluation with algorithm only requiring linear space (e.g. DFS)
- **Idea:** use successive iterations with increasing f-costs
  - Use f-bounds instead of binding the length of the path
- At each iteration, perform a depth-first search, cutting off a branch when its total costs $(g + h)$ exceeds a given threshold
  - Threshold starts at the estimate of the costs 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 of all values of nodes in the frontier that exceed the current threshold

F-cost Limit Computation by IDA*on Romania Travel Example

- Initial f-cost limit = 366 (f-costs of the initial state)
  - After 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 and Complexity of Iterative Deepening A* (IDA*)

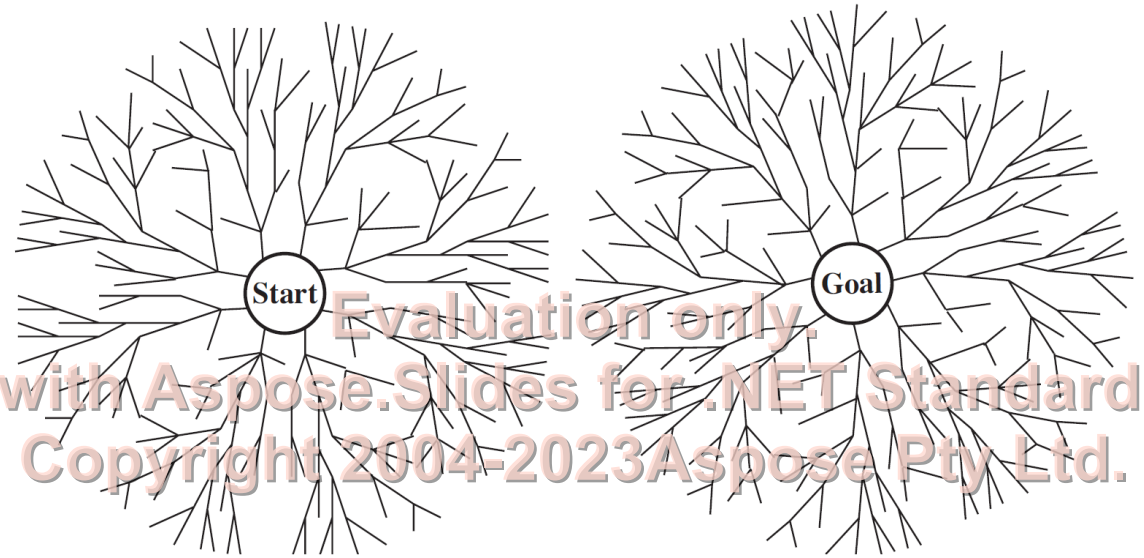
- IDA* is **complete** if every node has a finite number of successor nodes and if each action has positive and finite costs

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

- **Exponential time complexity:** $O(b^d)$

- **Linear space complexity:** $O(d)$

Bidirectional Search

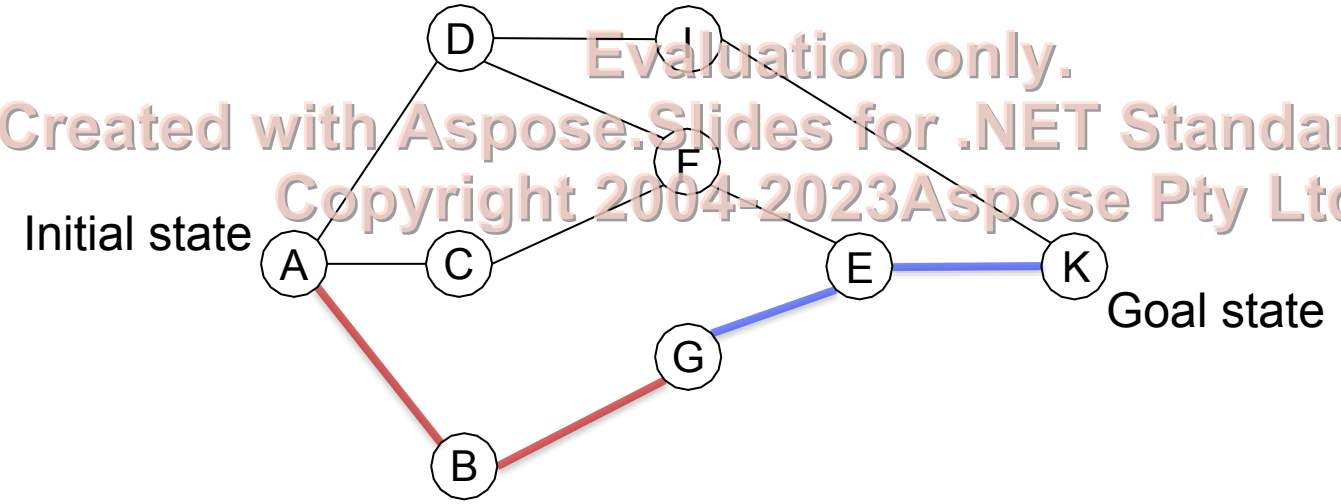


- 2 simultaneous searches: forward (from initial state) + backward (from goal state)
- Hoping that the two searches meet in a common state in the **middle**
- Motivation: $O(b^{d/2})$ is exponentially less than $O(b^d)$
- Any search technique can be used

Illustration of Bidirectional Search

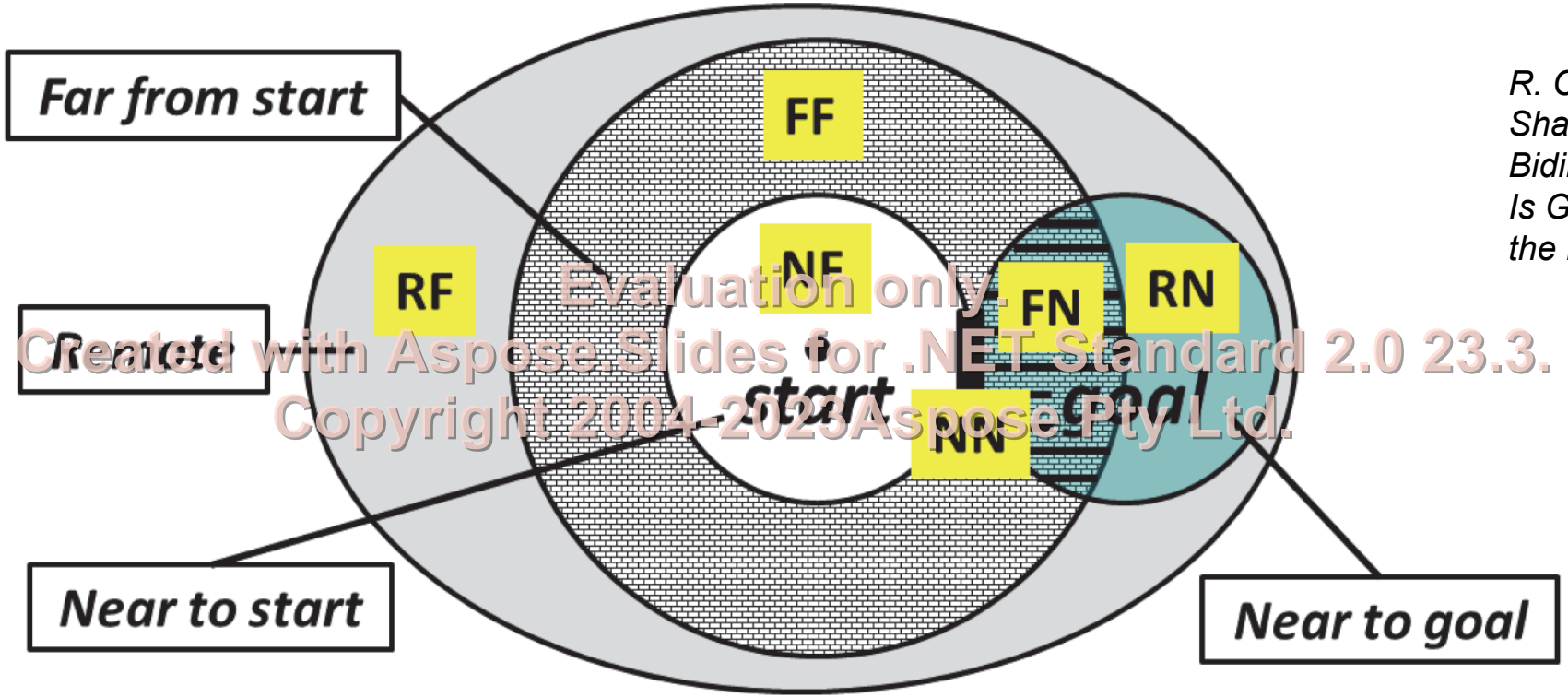
BFS as Forward search
States in the explored set: A, B
States in the frontier: C,D,G

DFS as Backward search
States in the explored set: K, E
States in the frontier: G,F,I



Challenge: How to ensure that both searches meet in the “middle” and that the combined solution is optimal?

Bidirectional Search – Node Sets in 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 the initial state (here called *start*) (N=near, F=far, R=remote) and the second letter indicates the distance from the goal state (N=near, F=far)
- NN includes only those states at the exact midpoint of optimal solutions

MM Algorithm - 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 state s to the target of the search (target for forward search is the goal state, target for backward search is the initial state)
- $d(u, v)$ is the distance (cost of a cheapest path) from state u to state v
- $C^* = d(\text{"initial"}, \text{"goal"})$ 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 the "initial" state, we make a 3-way distinction: s is "near to initial" if $d(\text{"initial"}, s) \leq C^*/2$, "far from initial" if $C^*/2 < d(\text{"initial"}, s) \leq C^*$, and "remote" if $d(\text{"initial"}, s) > C^*$

Properties of MM

- MM is **complete**, because it is based on A* searches that are guaranteed to meet
  - Each single A* search would find all solutions due to completeness of A*
- 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

- Time and space complexity improvements (if any) of MM are unknown

UNIVERSITÄT DES SAARLANDES

Overview on Properties of Heuristic Algorithms

Criterion	Greedy Best-First Search	A*	IDA*	MM Algorithm (Bidirectional search)
Evaluation function $f(n)$	$h(n.State)$	$g(n)+h(n.State)$	$g(n)+h(n.State)$	$g(n)+h(n.State)$
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
- b branching factor 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 $O(bd)$

Finding Good Heuristics

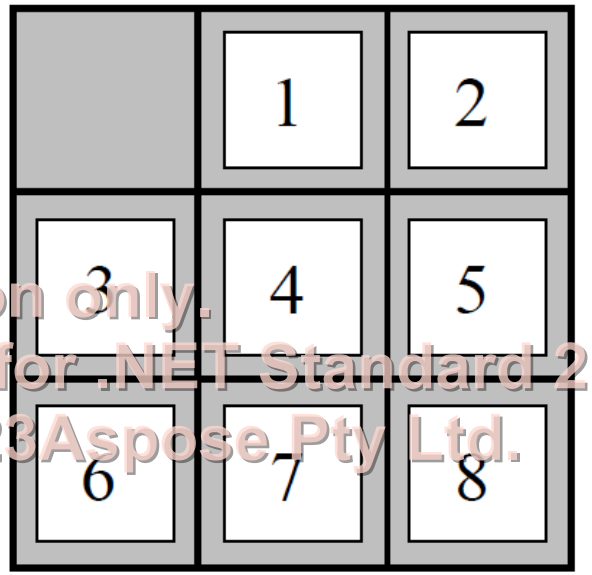
Designing Heuristic Functions

- 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
  - Automatic computation of heuristics is a hot topic in AI research

Heuristic Functions for the 8-Puzzle

Initial State

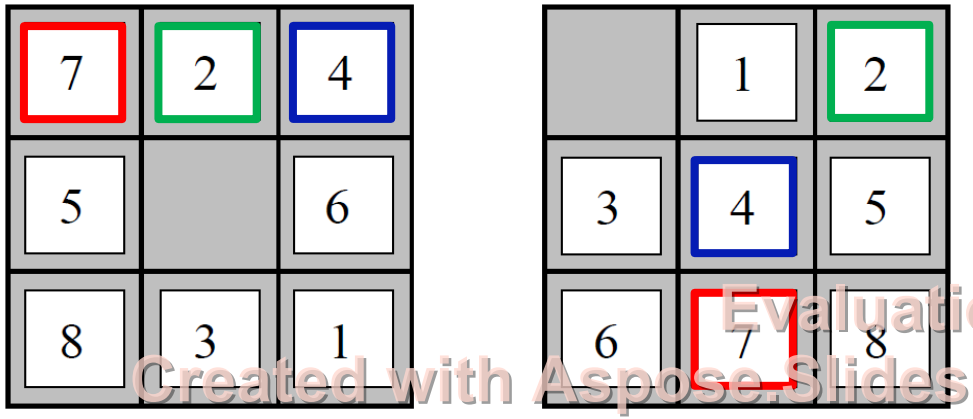


Goal State

Misplaced Tiles heuristic h_1: number of tiles in the wrong position

Manhattan Distance heuristic h_2: sum of the distances of the tiles from their goal positions

Misplaced Tiles vs. Manhattan Distance Heuristics



Initial State Goal State

h_1 = 8 (all tiles are misplaced)

h_2 = 3 + 1 + 2 + … = 18

- Distance between two points measured along axes at right angles

$$h(s) = \sum_{i=1}^{s} (|x_i(s) - \bar{x}_i| + |y_i(s) - \bar{y}_i|)$$

goal position

current position

- **Disadvantage:** considers all tiles independently

Effective Branching Factor

- The effective branching factor b^* is a characterization of the quality of a heuristic

- If the total number of nodes generated by A* for a particular problem is N and the solution depth is d, then b^* is the branching factor that a uniform tree (where every node has b^* successors) of depth d would have to have in order to contain $N + 1$ nodes

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

- There is no closed form to compute b^*, an educated guess is $N^{\left(\frac{1}{d}\right)}$

- Testing heuristics on a small set of problems and determining b^* can provide a good guide to the heuristic's overall usefulness

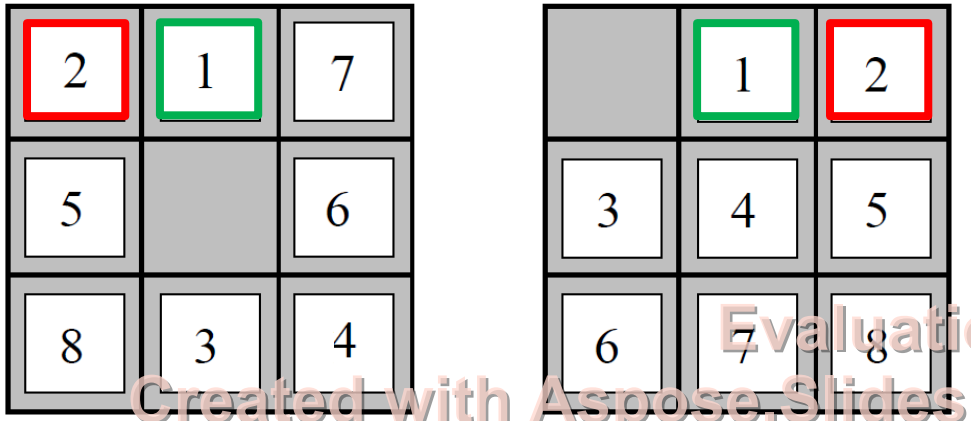
Empirical Comparison of the Effective Branching Factor of h_1 and h_2

- To test the heuristic functions h_1 and h_2, 1200 random problems with solution length (d) from 2 to 24 (100 instances of the 8-puzzle for each even d) are generated

- Iterative deepening search IDS and A* tree search using h_1 and h_2
- Average number of nodes generated by each strategy and the effective branching factor
- A* using h_2 works best on the test set

	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



MH: $h_2 = 2 + 0 = 2$

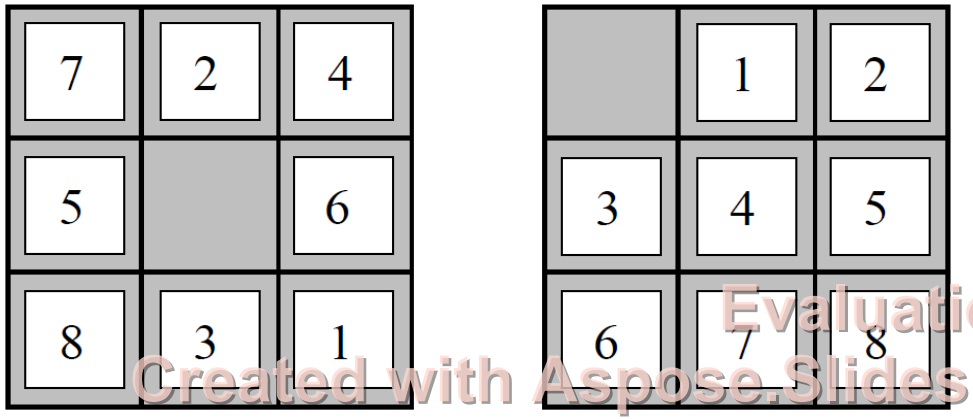
LC: $h_3 = 2 + 2 = 4$

- 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
  - The linear conflict heuristic LC will add a cost of 2 moves for each pair of conflicting tiles to the Manhattan Distance
  - Motivation: Tiles need to go around one another
- LC is consistent and more informative than MH

UNIVERSITÄT DES SAARLANDES

Gaschnig's Heuristic

Gaschnig, John. "A Problem Similarity Approach to Devising Heuristics: First Results", International Joint Conferences on Artificial Intelligence, 1979. 301-307.



Initial State Goal State

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:

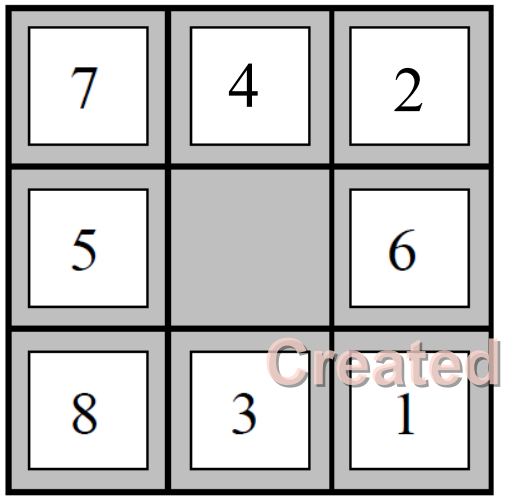
If 9 is not at the 1st position:

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

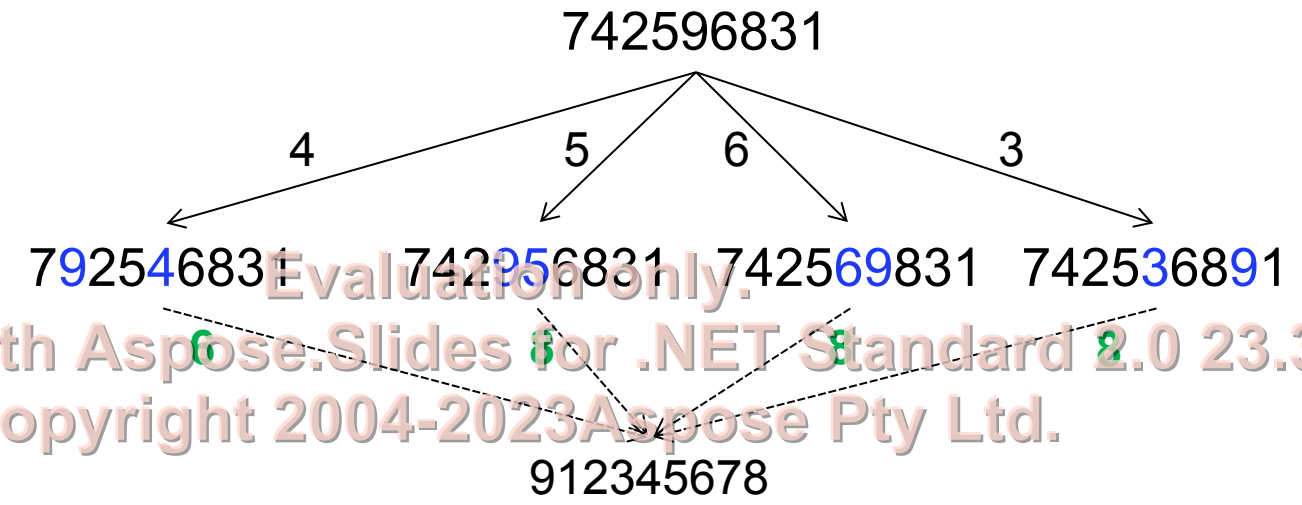
else: swap 9 with the rightmost element that is not in its proper place

724596831 – 729546831 – 792546831 – 712546839 – 712546938 – 712549638 – 712945638 – 712345698 – 912345678 = **8 swaps**

Applying Gaschnigs Heuristic During Search



Initial State



Cells to be moved in blue

- 4 successor nodes
- Compute the number of swaps for each node yielding values in green
- Expand the node with the fewest number of swaps to the goal state
  - Move cell 4 to the middle of the grid

Problem Relaxation on a Whitebox Description of the 8-Puzzle

- Primitive Predicates
  - 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)

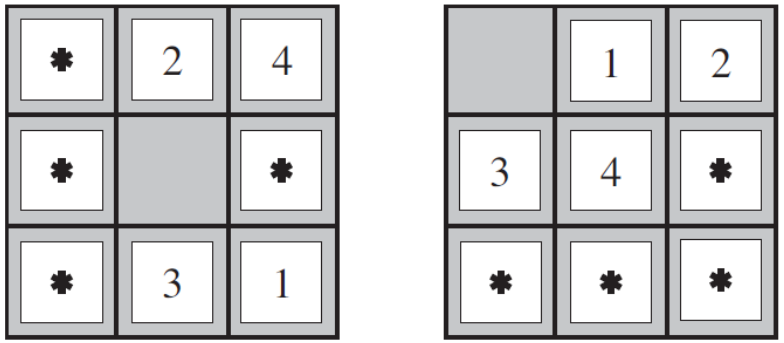
  effects: CLEAR(y) & ON(t, z) & NOT ON(t, y) & NOT CLEAR(z)

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

Admissible Heuristics from Subproblem Solutions: Pattern Database

- Decompose the problem into subproblems (subgoals)
  - Costs of the optimal solution to a subproblem are an admissible heuristic and a lower bound for the complete problem solution
  - Note that the sum of the subproblem costs is not an admissible heuristic
    - Ignores potentially shared moves and may overestimate

- Subproblem solutions can also be reused directly
  - Opening and end game libraries in chess
  - Combining subproblem solutions may not yield the optimal solution to a problem as synergies when solving the subproblems are lost



Subproblem: get tiles 1,2,3,4 into their correct positions

Learning Heuristic Functions by Problem Relaxation

- 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 application domain and the given problem instance to learn a domain- and 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
- Greedy best-first search often quickly finds good solutions in practice
- A* is optimal on graphs when using consistent heuristics
- Good heuristics can be developed by analyzing a search problem using relaxation methods
- Only recently, AI research has renewed its interest in bidirectional search algorithms, which can now be guaranteed to meet in the middle, but if these algorithms achieve a signficant reduction in search effort is an open question

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 and which role do they play in heuristic search?
5. What are the properties of good heuristics?
6. What is an admissible or consistent heuristic function?
7. What can happen with A*, if the heuristic function is inadmissible or inconsistent when performing tree or graph search?
8. What are important properties of bidirectional search to work better than unidirectional search?