

# Fundamentals of Artificial Intelligence – Informed Search

Matthias Althoff

TU München

Winter semester 2023/24

# Organization

- 1 Informed Search Strategies
  - Greedy Best-First Search
  - A\* Search
- 2 Heuristic Functions
  - Relaxed Problems
  - Pattern Databases
  - Reachability Analysis

The content is covered in the AI book by the section “Solving Problems by Searching”, Sec. 5-6.

# Learning Outcomes

- You can describe the difference between informed and uninformed search.
- You can apply Greedy Best-First Search and A\*.
- You can discuss the influence of a heuristic on the search result.
- You can decide whether a heuristic dominates another heuristic.
- You can explain methods how to systematically obtain heuristics for search problems.
- You can create simple heuristics to improve the performance of search problems.

# Informed Search

Informed search strategies can find solutions more efficiently using domain-specific hints about the location of goals using a **heuristic function**  $h(n)$ :

- $h(n)$  gives the **estimated cost of the cheapest path** from  $n$ . State to a goal state;
- $h(n)$  is problem-specific with the only constraints that it is **nonnegative** and  $h(\hat{n}) = 0$ , where  $\hat{n}$  is a goal node.

All presented informed search strategies are identical to **best-first search** with a specific evaluation function  $f(n)$  based on  $h(n)$ .

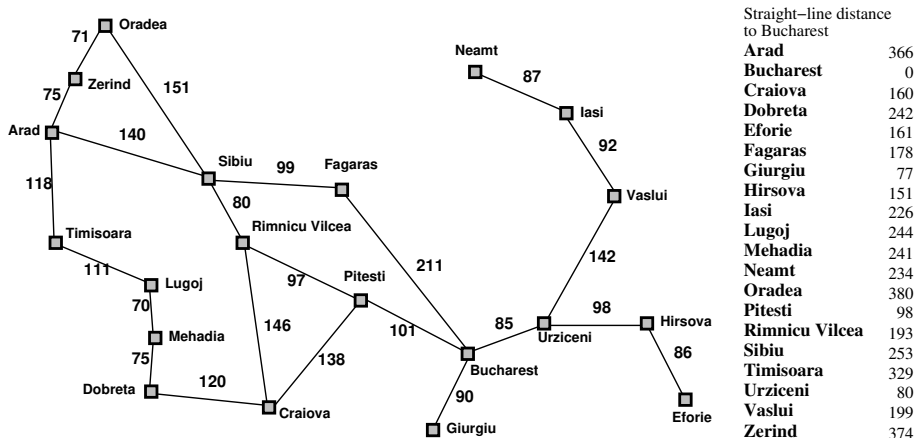
## Heuristics in general

Heuristics refers to the art of achieving good solutions with limited knowledge and time based on experience.

# Greedy Best-First Search: Idea InformedSearch.ipynb

Expands the node that is estimated to be closest to the goal by using just the heuristic function so that  $f(n) = h(n)$ .

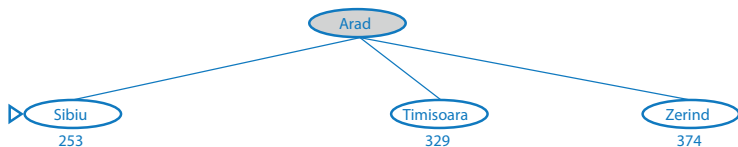
**Romania example:** We use the straight line distance to the goal as  $h(n)$ :



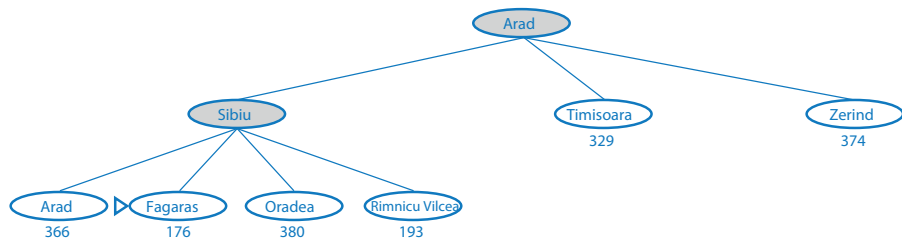
# Greedy Best-First Search: Example (Step 1)



## Greedy Best-First Search: Example (Step 2)

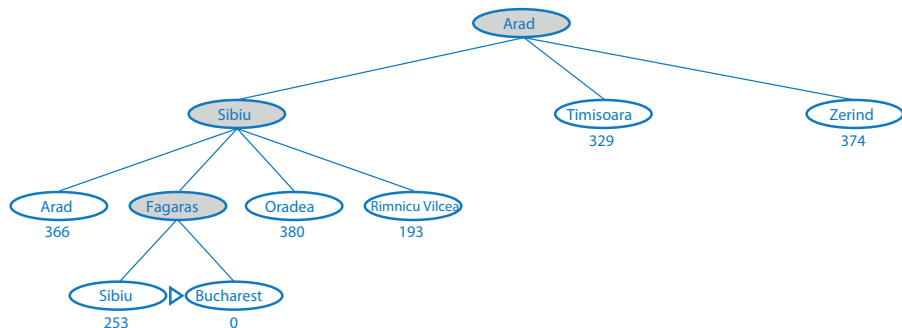


## Greedy Best-First Search: Example (Step 3)





## Greedy Best-First Search: Example (Step 4)



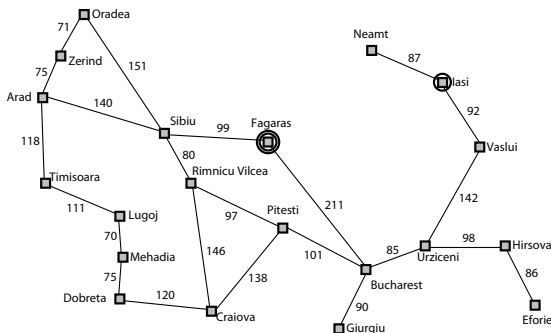
Note that the solution is not optimal since the path is 32 km longer than through Rimnicu Vilcea and Pitesti.

## Tweedback Question

Greedy best-first search using **tree-like search**; start: Iasi, goal: Faragas.

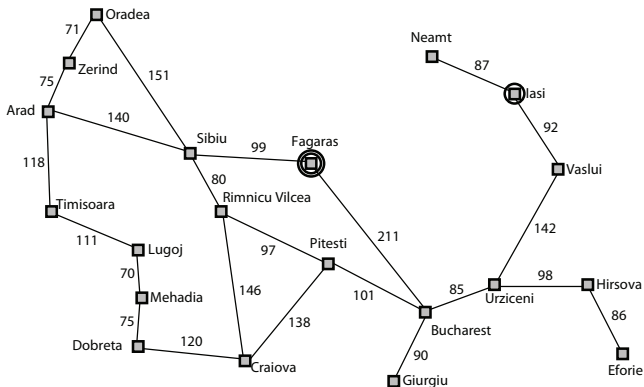
Neamt is first expanded. Next,

- A no further node is expanded.
- B Iasi is expanded. Afterwards, Neamt is again expanded due to the closest straight line distance.
- C Iasi is expanded. Afterwards, Vaslui is expanded, since Neamt has been expanded before.



## Tweedback Question: Solution

Neamt is expanded first due to closer straight line distance and then constantly moves between Iasi and Neamt.



Greedy best-first search is incomplete when not using a reached set (tree-like search), even on finite state spaces.

# Tweedback Questions

- Is greedy best-first search optimal? ✗
- What is the time complexity of greedy best-first search?
  - A  $\mathcal{O}(b^m)$ .
  - B  $\mathcal{O}(b^d)$ .

**Reminder:** Branching factor  $b$ , depth  $d$ , maximum length  $m$  of any path.

# Greedy Best-First Search: Performance

Reminder: Branching factor  $b$ , depth  $d$ , maximum length  $m$  of any path.

- **Completeness:** Yes, if graph search is used.
- **Optimality:** No (see previous example).
- **Time complexity:** The worst-case is that the heuristic is misleading the search such that the solution is found last:  $\mathcal{O}(b^m)$ . But a good heuristic can provide a dramatic improvement.
- **Space complexity:** Equals time complexity since all nodes are stored:  $\mathcal{O}(b^m)$ .

# Greedy Best-First Search: CommonRoad Example



- Concatenation of motion primitives
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](https://github.com/malthoff/cr_informed_search_tutorial.ipynb).

# Greedy Best-First Search: CommonRoad Example



- Concatenation of motion primitives
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](https://github.com/malthoff/cr_informed_search_tutorial.ipynb).

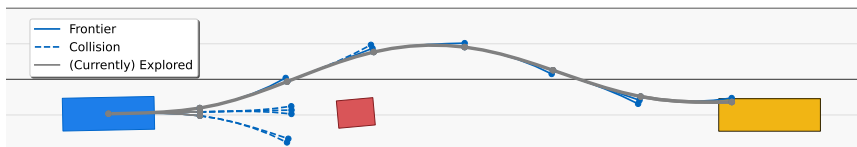
# Greedy Best-First Search: CommonRoad Example



- Concatenation of motion primitives
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](#).



# Greedy Best-First Search: CommonRoad Example



- Concatenation of motion primitives
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](https://github.com/malthoff/cr_informed_search_tutorial.ipynb).

## A\* Search: Idea

The most widely known informed search is A\* search (pronounced “A-star search”). It uses

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

where  $h(n)$  has to be **admissible**. An admissible heuristic is an underestimation, i.e., it has to be less than or equal to the actual cost.

→  $f(n)$  never overestimates the cost to the goal and thus the algorithm keeps searching for paths with a lower cost to the goal.

# A\* Search: Example (Step 1)

( InformedSearch.ipynb)

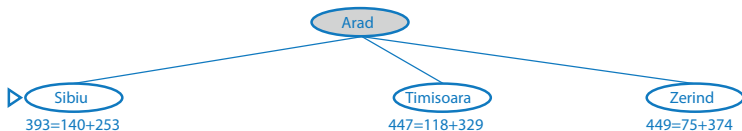
Straight line distance is an underestimation of the cost to the goal, which is used for  $h(n)$ .



Nodes are labeled with  $f = g + h$

## A\* Search: Example (Step 2)

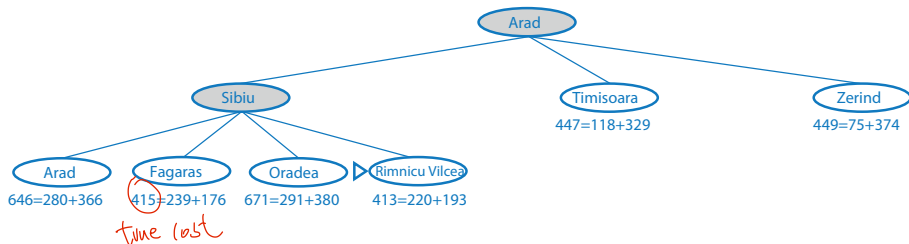
Straight line distance is an underestimation of the cost to the goal, which is used for  $h(n)$ .



Nodes are labeled with  $f = g + h$

# A\* Search: Example (Step 3)

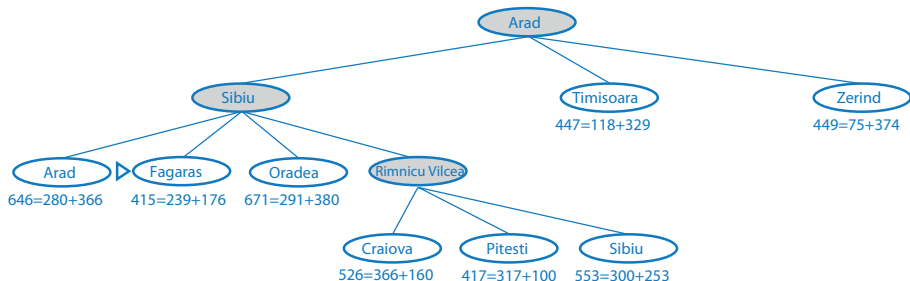
Straight line distance is an underestimation of the cost to the goal, which is used for  $h(n)$ .



Nodes are labeled with  $f = g + h$

## A\* Search: Example (Step 4)

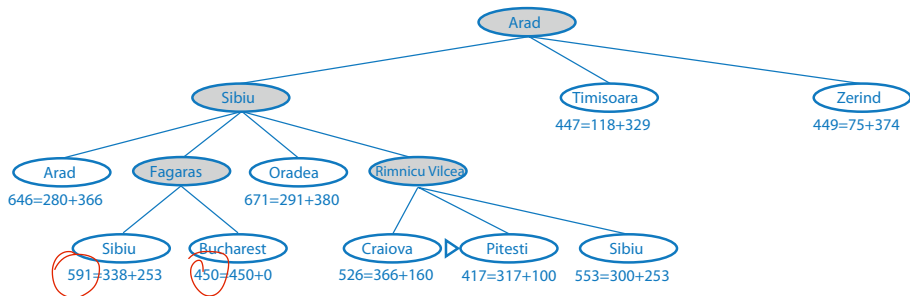
Straight line distance is an underestimation of the cost to the goal, which is used for  $h(n)$ .



Nodes are labeled with  $f = g + h$

## A\* Search: Example (Step 5)

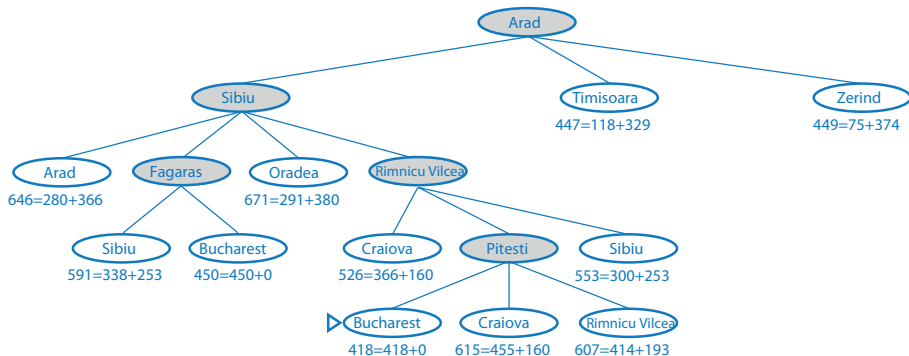
Straight line distance is an underestimation of the cost to the goal, which is used for  $h(n)$ .



Nodes are labeled with  $f = g + h$

# A\* Search: Example (Step 6)

Straight line distance is an underestimation of the cost to the goal, which is used for  $h(n)$ .

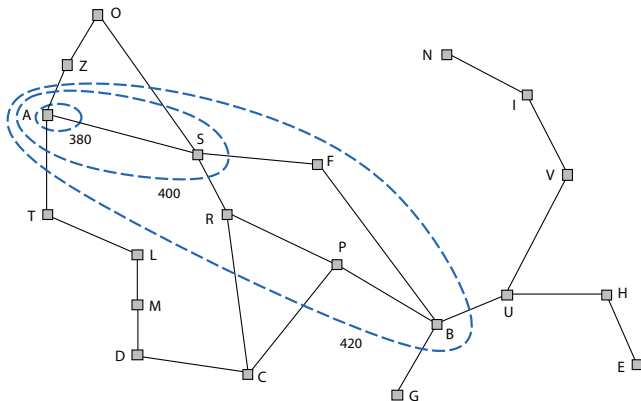


Nodes are labeled with  $f = g + h$



# A\* Search: Effects of the Heuristic

- The heuristic “steers” the search towards the goal.
- A\* expands nodes in order of increasing  $f$  value, so that “ $f$ -contours” of nodes are gradually added.
- Each contour  $i$  includes all nodes with  $f \leq f_i$ , where  $f_i < f_{i+1}$ .



# Tweedback Question

Given the cost  $C^*$  of the optimal path.

Does A\* expand nodes, where  $f(n) > C^*$ ?

# A\* Search: Pruning

- Given the cost  $C^*$  of the optimal path, only paths with  $f(n) \leq C^*$  are expanded (equality occurs when the heuristic returns the exact remaining costs).  
→ A\* never expands nodes, where  $f(n) > C^*$ .
- For instance, Timisoara is not expanded in the previous example. We say that the subtree below Timisoara is pruned.
- The concept of pruning – eliminating possibilities from consideration without having to examine them – brings enormous time savings and is similarly done in other areas of AI.

# Consistent Heuristics



A slightly stronger condition than admissibility is consistency. A consistent heuristic fulfills

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

for each node  $n$ , each action  $a$  and each direct successor  $n'$  (remember:  $c(n, a, n')$  is the action cost).

→ This is a form of the general triangle inequality.

→ It is fairly easy to show that every consistent heuristic is also admissible.

# Admissibility vs Consistency

What are consistent heuristics useful for?

**Admissible** heuristic:

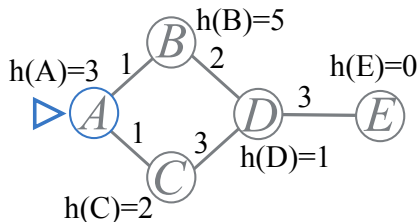
Any **goal node** is only expanded if it is located on an optimal path to its state.

**Consistent** heuristic:

Any **node** is only expanded if it is located on an optimal path to its state.



Inconsistent heuristics can result in re-adding a node to the frontier that was already expanded before:



# Admissibility vs Consistency

What are consistent heuristics useful for?

**Admissible** heuristic:

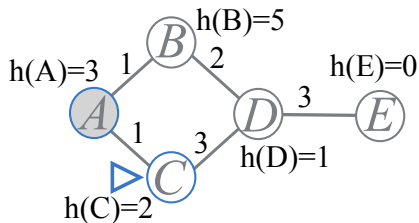
Any **goal node** is only expanded if it is located on an optimal path to its state.

**Consistent** heuristic:

Any **node** is only expanded if it is located on an optimal path to its state.



Inconsistent heuristics can result in re-adding a node to the frontier that was already expanded before:



# Admissibility vs Consistency

What are consistent heuristics useful for?

**Admissible** heuristic:

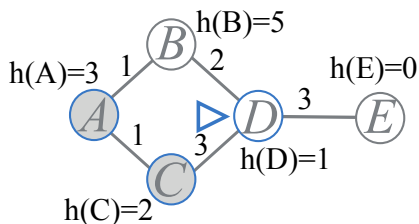
Any **goal node** is only expanded if it is located on an optimal path to its state.

**Consistent** heuristic:

Any **node** is only expanded if it is located on an optimal path to its state.



Inconsistent heuristics can result in re-adding a node to the frontier that was already expanded before:



# Admissibility vs Consistency

What are consistent heuristics useful for?

**Admissible** heuristic:

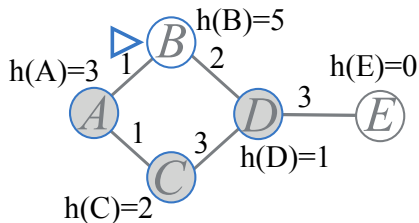
Any **goal node** is only expanded if it is located on an optimal path to its state.

**Consistent** heuristic:

Any **node** is only expanded if it is located on an optimal path to its state.



Inconsistent heuristics can result in re-adding a node to the frontier that was already expanded before:





# Admissibility vs Consistency

What are consistent heuristics useful for?

**Admissible** heuristic:

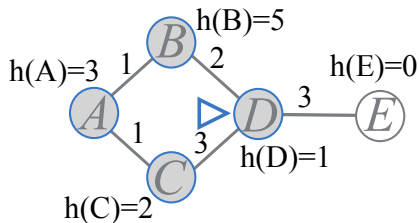
Any **goal node** is only expanded if it is located on an optimal path to its state.

**Consistent** heuristic:

Any **node** is only expanded if it is located on an optimal path to its state.



Inconsistent heuristics can result in re-adding a node to the frontier that was already expanded before:



# Admissibility vs Consistency

What are consistent heuristics useful for?

**Admissible** heuristic:

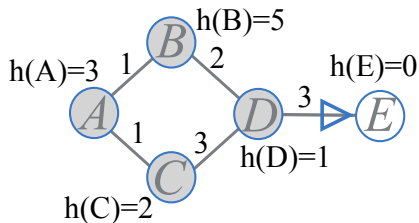
Any **goal node** is only expanded if it is located on an optimal path to its state.

**Consistent** heuristic:

Any **node** is only expanded if it is located on an optimal path to its state.



Inconsistent heuristics can result in re-adding a node to the frontier that was already expanded before:



But: in practice, inconsistent heuristics are not too bad...

# A\* Search: Performance

**Reminder:** Branching factor  $b$ , depth  $d$ , maximum length  $m$  of any path.

Time and space complexity of A\* is quite involved, and we will not derive the results. They are presented in terms of the relative error

$\epsilon = (h^* - h)/h^*$ , where  $h$  is the estimated and  $h^*$  is the actual cost from the root to the goal.

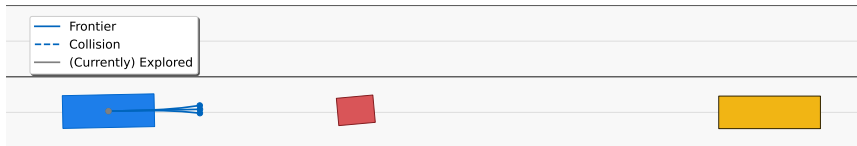
- **Completeness:** Yes, if costs are greater than 0 (otherwise infinite optimal paths of zero cost exist).
- **Optimality:** Yes (if costs are positive) and heuristic is admissible.
- **Time complexity:** We only consider the easiest case: The state space has a single goal and all actions are reversible:  $\mathcal{O}(b^{\epsilon d})$ .
- **Space complexity:** Equals time complexity since all nodes are stored.

# A\* Search: CommonRoad Example



- Concatenation of motion primitives.
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $g(n)$ : Time to reach current state.
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](#).

# A\* Search: CommonRoad Example



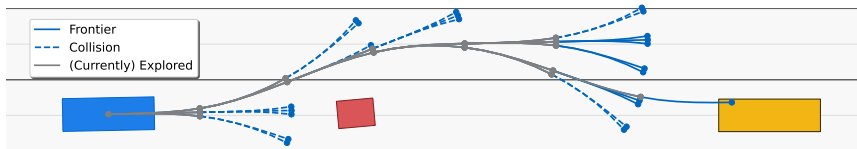
- Concatenation of motion primitives.
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $g(n)$ : Time to reach current state.
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](#).

# A\* Search: CommonRoad Example



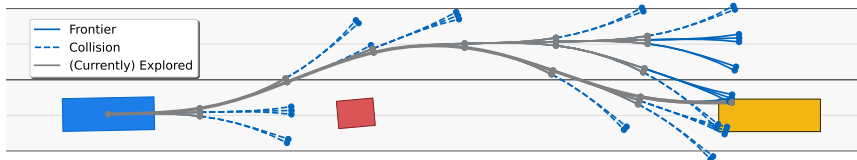
- Concatenation of motion primitives.
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $g(n)$ : Time to reach current state.
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](#).

# A\* Search: CommonRoad Example



- Concatenation of motion primitives.
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $g(n)$ : Time to reach current state.
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](#).

# A\* Search: CommonRoad Example



- Concatenation of motion primitives.
- Note: Colliding states are not further considered (collision with obstacle or out of road boundary).
- $g(n)$ : Time to reach current state.
- $h(n)$ : Euclidean distance to the goal region divided by the velocity of the current state.
- Link to tutorial: [cr\\_informed\\_search\\_tutorial.ipynb](#).




## Alternatives of A\* Search

One of the big disadvantages of A\* search is the possibly huge space consumption. This can be alleviated by extensions, e.g.:

- **Iterative-deepening A\***: Iterative deepening search, where the  $f$ -cost ( $g + h$ ) is used for cutoff, rather than the depth.
- **Recursive best-first search**: a simple recursive algorithm with linear space complexity. Its structure is similar to the one of recursive-depth-first search, but keeps track of the  $f$ -value of the best alternative path.
- **Memory-bounded A\*** and **simplified memory-bounded A\***: These algorithms work just like A\* until the memory is full. The algorithms drop less promising paths to free memory.

# Heuristic Functions

For the shortest route in Romania, the straight line distance is an obvious underapproximating heuristic.



7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

This is not always so easy, as we will show for the 8-puzzle.

# Heuristic Functions for the 8-Puzzle

Two commonly used candidates that underestimate the costs to the goal:

- $h_1$ : the number of misplaced tiles (e.g., in the figure  $h_1 = 8$ ).

**Why admissible?** Misplaced tile has to be moved at least once.

- $h_2$ : the sum of the horizontal and vertical distances to the goal.

**Why admissible?** All a move can do is bring the tile one step closer.

In the figure below:

tile	1	2	3	4	5	6	7	8	sum
steps	3	1	2	2	2	3	3	2	18

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

(the true cost is 26)

## Effective Branching Factor

One way of characterizing the quality of a heuristic is the effective branching factor  $b^*$ .

**Given:**

- Number of nodes  $N$  generated by the  $A^*$  search.
- A uniform tree with depth  $d$  (each node has the same fractional number  $b^*$  of children)

Thus,

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

E.g., if  $A^*$  generates a solution at depth 5 using 52 nodes,  $b^* = 1.92$  since

$$53 \approx 1 + 1.92 + (1.92)^2 + \dots + (1.92)^5.$$

The branching factor makes it possible to compare heuristics applied to problems of different size. Why?

# Comparison of the Heuristic for the 8-Puzzle

100 random problems for each depth  $d$ :

d	Search Cost (nodes generated)			Effective Branching Factor		
	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	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

# Tweedback Question

Is  $h_2$  always better or equally good as  $h_1$ ?

## Reminder:

- $h_1$ : the number of misplaced tiles.
- $h_2$ : the sum of the distances to the goal positions using horizontal and vertical movement.

# Domination of a Heuristic

**Question:** Is  $h_2$  always better or equally good as  $h_1$ ?

**Answer:** Yes.

**Reason:**

- For every node, we have that  $h_2(n) \geq h_1(n)$ . We say that  $h_2$  dominates  $h_1$ .
- A\* using  $h_2$  will never expand more nodes than with  $h_1$  (except possibly for some nodes with  $f(n) = C^*$ ):

A\* expands all nodes with

$$f(n) < C^* \leftrightarrow h(n) < C^* - g(n),$$

where  $g(n)$  is fixed. Since  $h_2(n) > h_1(n)$ , fewer nodes are expanded.

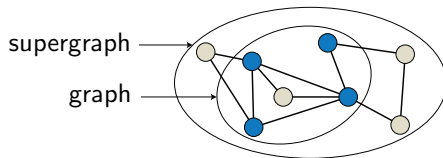
# Heuristics from Relaxed Problems

**Idea:** The previous heuristics  $h_1$  and  $h_2$  are perfectly accurate for **simplified** versions of the 8-puzzle.

**Method:** Formalize a problem definition and remove restrictions.

**Result:**

- One obtains a **relaxed** problem, i.e., a problem with more freedom, whose state-space graph is a supergraph of the original one (see figure).
- The optimal solution in the original problem is automatically a solution in the relaxed problem, but the relaxed problem might have a better solution due to added edges.
- Hence, the cost of the optimal solution in the relaxed problem is underapproximative.





# Heuristics from Relaxed Problems: 8-Puzzle Example

A tile can move from square A to B if  $\Phi_1 \wedge \Phi_2$ , where

- $\Phi_1$ : B is blank
- $\Phi_2$ : A is adjacent to B

We generate three relaxed problems by removing one or two conditions:

- 1 remove  $\Phi_1$ : A tile can move from square A to B if A is adjacent to B.
  - 2 remove  $\Phi_2$ : A tile can move from square A to B if B is blank.
  - 3 remove  $\Phi_1$  and  $\Phi_2$ : A tile can move from square A to B.
- From the first relaxed problem, we can generate  $h_2$  and from the third relaxed problem, we can derive  $h_1$ .
  - If one is not sure which heuristic is better, one can apply in each step

$$h(n) = \max\{h_1(n), h_2(n), \dots, h_M(n)\}.$$

Why?

# Heuristics from Pattern Databases

- Underapproximative heuristics can also be obtained from subproblems.
- Those solution costs are underapproximations and can be stored in a database.
- The number of subproblems has to be much less than the original problems to not exceed storage capacities.

The figure shows a subproblem of an 8-puzzle, where only the first 4 tiles have to be brought into a goal position:

*	2	4
*		*
*	3	1

Start State

	1	2
3	4	*
*	*	*

Goal State

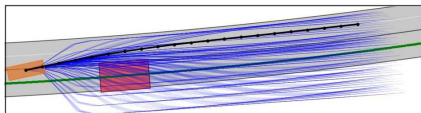
# Heuristics from Reachability Analysis

## (Own Research)

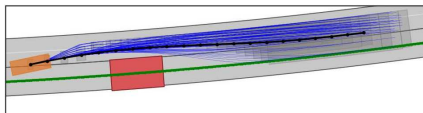
Not  
relevant for  
the exam

Reachability analysis can effectively guide search in continuous state spaces.

The figures (and video) show how we can prune the search space for a motion planner using reachable sets:



standard search space



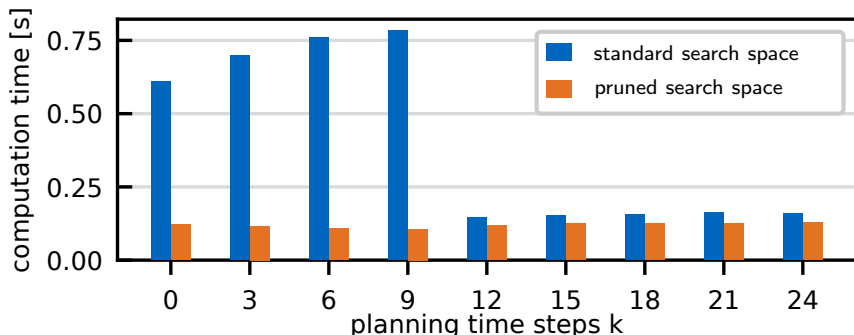
pruned search space

Reachability analysis is taught in our lecture *Formal Methods for Cyber-Physical Systems*.

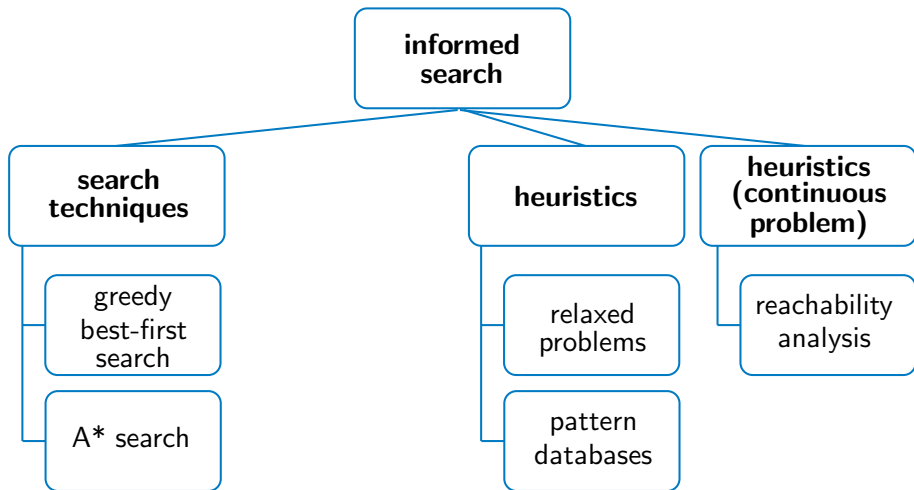
# Heuristics from Reachability Analysis: Example Continued

Not  
relevant for  
the exam

Pruning the search space before planning can significantly reduce the computation time of the motion planner:



# Overview of Informed Search Methods



# Summary

- Informed search methods require a heuristic function that estimates the cost  $h(n)$  from a node  $n$  to the goal:
  - **Greedy best-first search** expands nodes with minimal  $h(n)$ , which is not always optimal.
  - **A\* search** expands nodes with minimal  $f(n) = g(n) + h(n)$ . A\* is complete and optimal for underapproximative  $h(n)$ .
- The performance of informed search depends on the quality of the heuristic. Possibilities to obtain good heuristics are **relaxed problems**, **pattern databases**, and **reachability analysis**.