Block 1 – Knowledge Representation and Search

Chapter 6:
Heuristic Search

Block 1, Chapter 6- Index

Bibliography

S. Russell, P. Norvig. *Artificial Intelligence . A modern approach.* Prentice Hall, 3rd edition, 2010 (Chapter 3) http://aima.cs.berkeley.edu/

Alternatively:
S. Russell, P. Norvig. *Artificial Intelligence . A modern approach.* Prentice Hall, 2nd edition, 2004 (Chapters 3 and 4) http://aima.cs.berkeley.edu/2nd-ed/

1. Heuristic search

Informed (heuristic) search: one that uses problem-specific knowledge to guide search.

Heuristic search can find solutions more efficiently than an uninformed search. Heuristic search is recommended for complex problems of combinatorial explosion(e.g.: traveling salesman problem).

Why use heuristics? In order to efficiently solve a problem we must sometimes give up using a systematic search strategy which guarantees optimality and use instead a search strategy that returns a good solution though not the optimal one.

Heuristic search performs an intelligent search guidance and allows for pruning large parts of the search tree.

Why heuristics are appropriate?

1. Usually, we don't need to get the optimal solution in complex problem, but a *good* solution is just enough.
2. Heuristics may not return a good solution for the worst-case problem but this case seldom happens.
3. Understanding how heuristics work (or don't work) helps people understand and go deeply into the problem

1. Heuristic search

We will consider a general approach called best-first search:

- an instance of the TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an evaluation function $f(n)$
- $f(n)$ is a cost estimate so the node with the lowest cost is expanded first from the priority queue (in uninformed search, f(n) is a different function for each strategy so that nodes are inserted in the OPEN LIST according to the expansion order of the strategy)
- In fact, all search strategies can be implemented by using $f(n)$; the choice of **f** determines the search strategy
- Two special cases of best-first search: *greedy best-first search* and *A**

2. Greedy search

Most best-first algorithms include as a component of **f** a heuristic function $h(n)$

Heuristic function: *A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood*

$h(n)$= *estimated cost* of the cheapest path from the state at node **n** to the goal state

If **n** is the goal state then $h(n)=0$

Greedy best-first search expands the node that appears to be closest to goal; it expands the closest node to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function: $f(n)=h(n)$

2. Greedy search: the Romania example
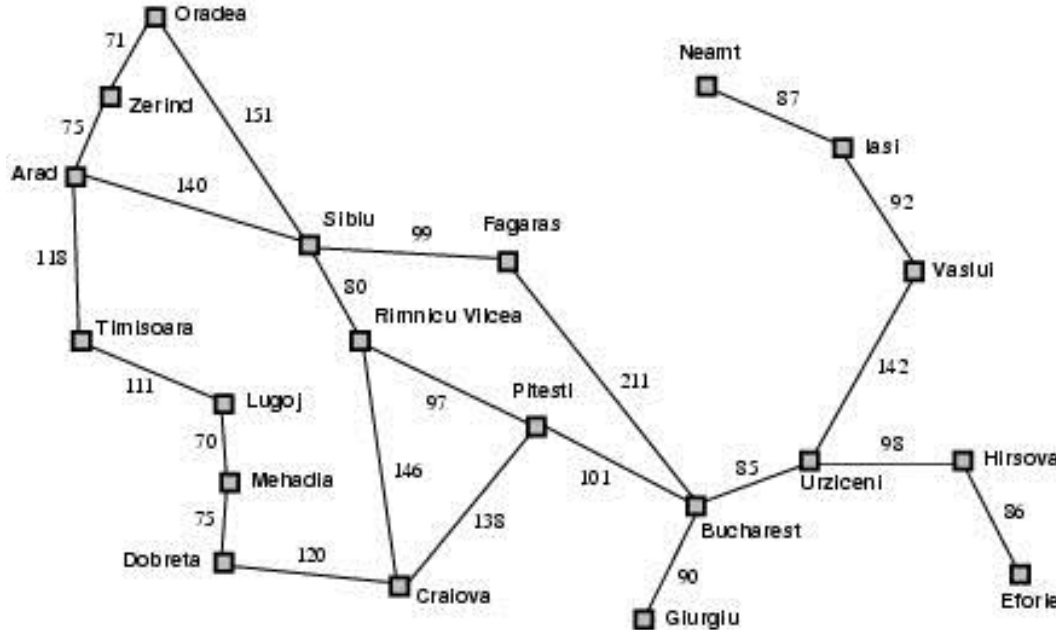
h_{SLD} = straight-line distance heuristic from a city to Bucharest

h_{SLD} can **NOT** be computed from the problem description itself

Examples:
    h(Arad)=366
    h(Fagaras)=176
    h(Bucharest)=0

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	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

2. Greedy search: the Romania example

In this example $f(n)=h(n)$

- expand node that is closest to goal
- greedy best-first search
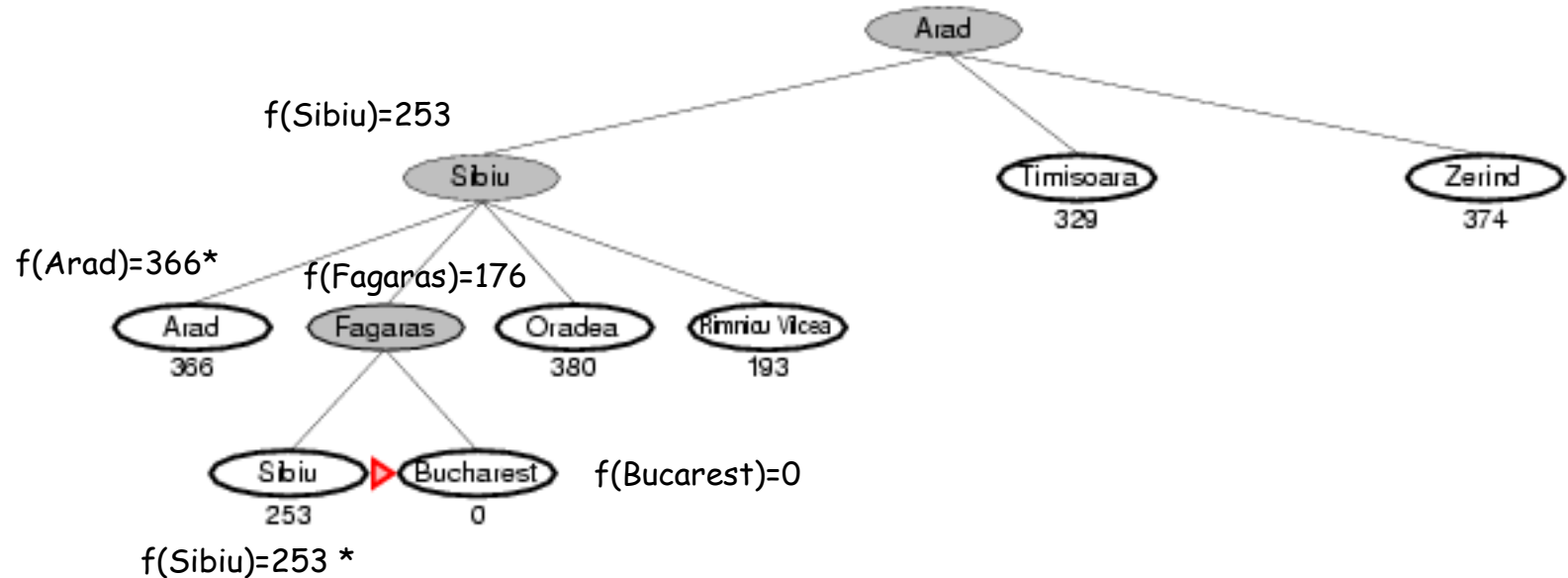
Reached goal: non-optimal solution
(alternative solution: Arad – Sibiu - Rimnicu Vilcea – Pitesti - Bucharest)

* By using the GRAPH-SEARCH version, we could check repeated states in the CLOSED list and the new appearance of the Arad node would not be inserted in the OPEN list.

2. Greedy search: evaluation

- Complete:
  - NO, it can get stuck in loops, e.g. (consider the problem to go from Iasi to Fagaras): Iasi $\to$ Neamt $\to$ Iasi $\to$ Neamt (dead end, infinite loop)
  - Resembles depth-first search (it prefers to follow a single path to the goal)
  - The GRAPH-SEARCH version is complete (check on repeated states in the CLOSED LIST)

- Optimal:
  - No, at each step it tries to get as close to the goal state as it can; this is why it is called greedy!

- Time complexity (execution time):
  - The worst-case time complexity is $O(b^m)$ where m is the maximum depth of the search space
  - Like worst case in depth-first search
  - Good heuristic can give dramatic improvement
  - The amount of the reduction depends on the particular problem and on the quality of the heuristic

- Space complexity (memory requirements):
  - $O(b^m)$ where m is the maximum depth of the search space

3. A* search

A* search is the most widely-known form of best-first search

It evaluates nodes by combining $g(n)$, the cost to reach the node n, and $h(n)$, the cost to get from the node to the goal state. $f(n)=g(n)+h(n)$

The idea is to avoid paths that are already expensive. $f(n)$ is the estimated total cost of the cheapest solution through n.

Algorithms that use an evaluation function of the form $f(n)=g(n)+h(n)$ are called A algorithms

3. A* search

A* search uses an admissible heuristic function

A heuristic is admissible if it never overestimates the cost to reach the goal, i.e., it is optimistic

Formally:

- a heuristic $h(n)$ is admissible if for every node n, $\mathbf{h(n) \leq h^*(n)}$, where $h^*(n)$ is the true cost to reach the goal state from **n**

- Because A* search uses an admissible heuristic, it returns the optimal solution
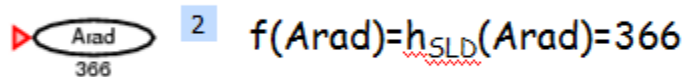
- $h(n) \geq 0$ so $h(G)=0$ for any goal G

e.g. $h_{SLD}(n)$ never overestimates the actual road distance
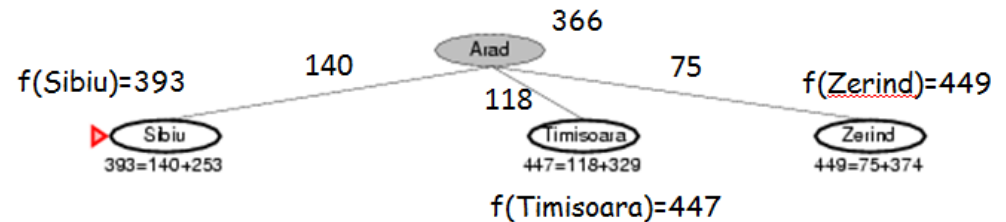
3. A* search: the Romania example

In this example $f(n)=g(n)+h(n)$

- $h(n)=h_{SLD}(n)$
- expand node with the lowest estimated total cost to the goal state
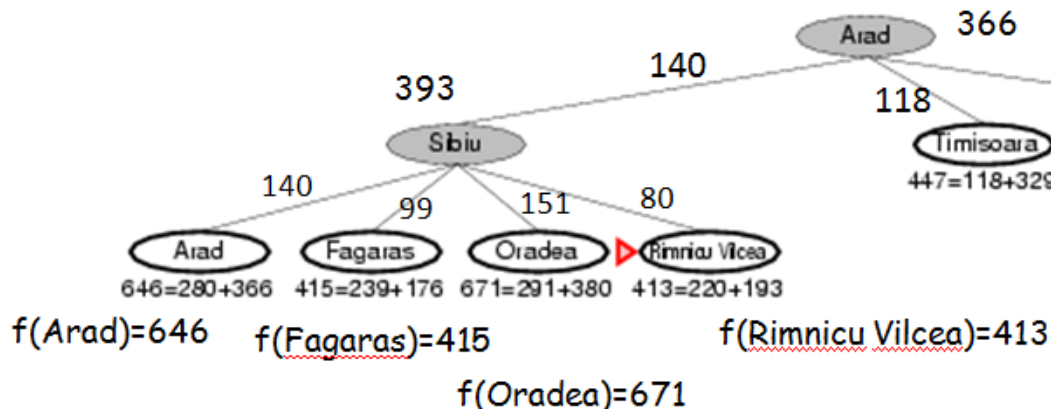- A* search

Iteration 1:

$f(Arad)=h_{SLD}(Arad)=366$

Iteration 2:

$f(Sibiu)=393$

$f(Zerind)=449$

$f(Timisoara)=447$

Iteration 3:

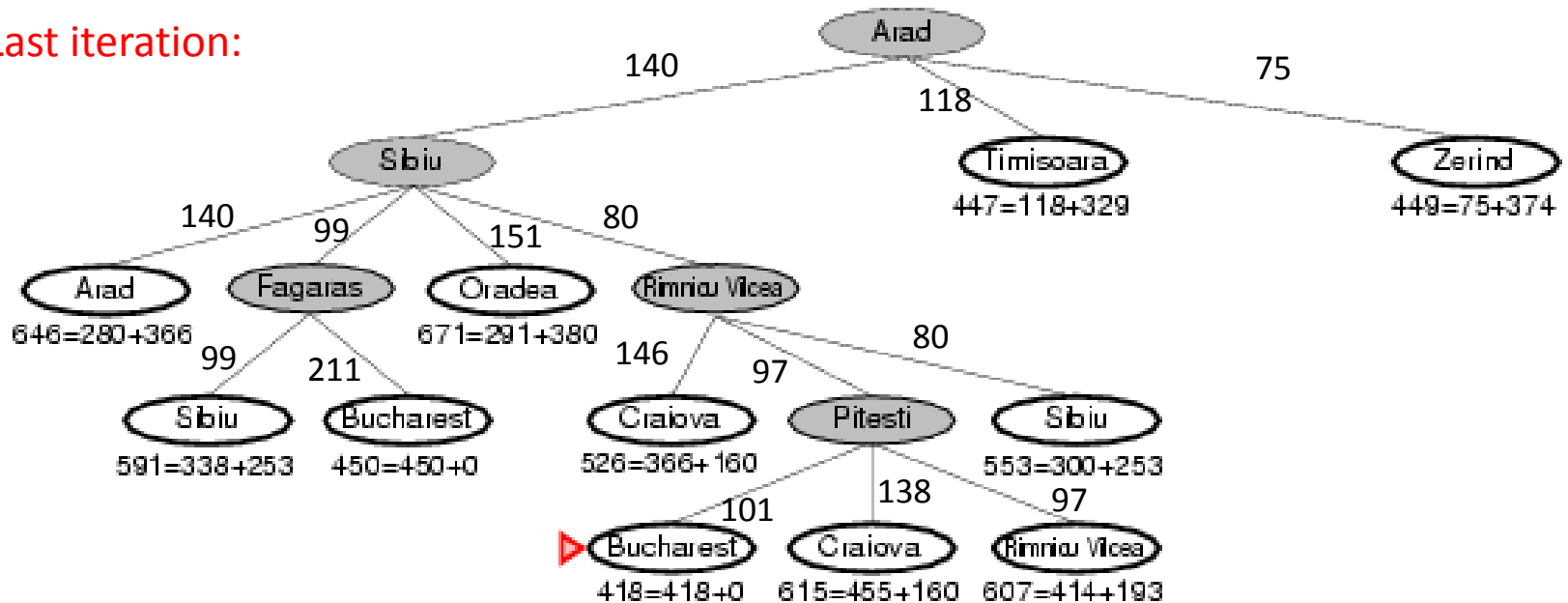$f(Arad)=646$ $f(Fagaras)=415$ $f(Rimnicu\ Vilcea)=413$

$f(Oradea)=671$

In GRAPH-SEARCH version: Arad is a repeated node; the Arad node in the CLOSED list has a better f-value.

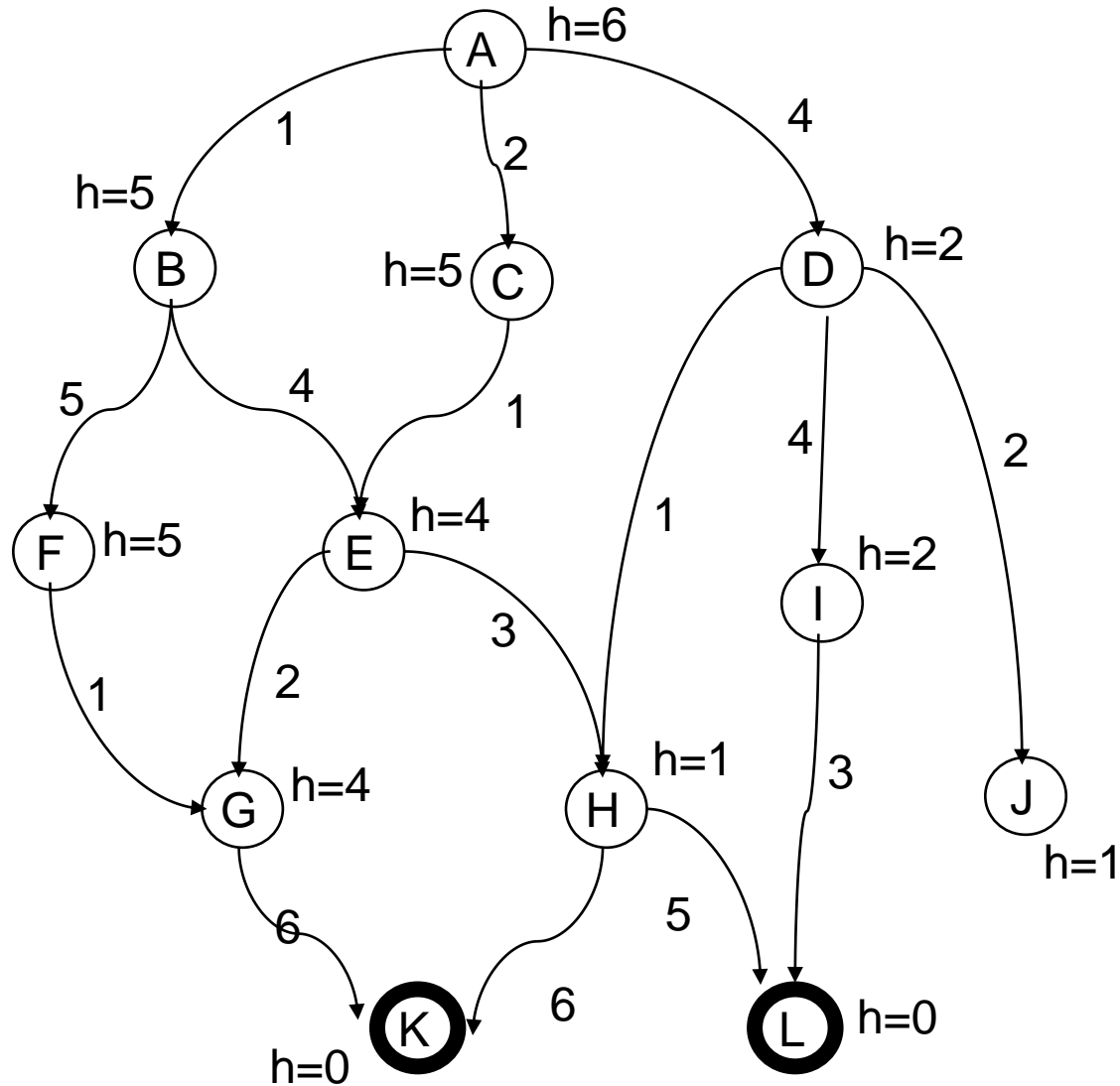# 3. A* search: the Romania example

OPEN list= {Bucharest(418), Timisoara(447), Zerind(449), Craiova(526), Oradea (671)}
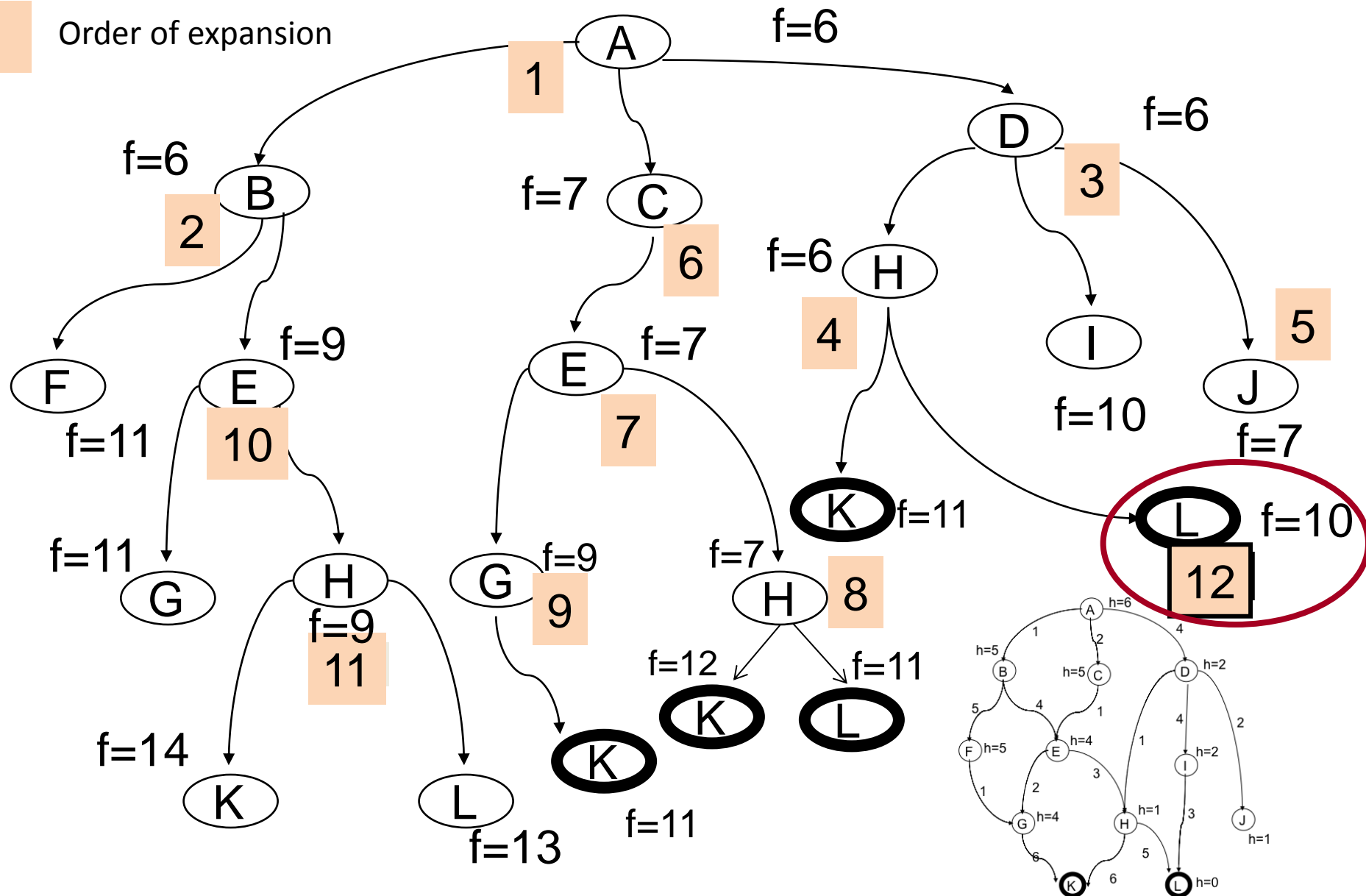
CLOSED list = {Arad, Sibiu, Rimnicu Vilcea, Fagaras, Pitesti}

Bucharest is already in the OPEN LIST (cost=450). The newly generated node has a lower estimated total cost (f-value=418) than the one in the OPEN LIST. We replace the Bucharest node in OPEN with the new found instance.
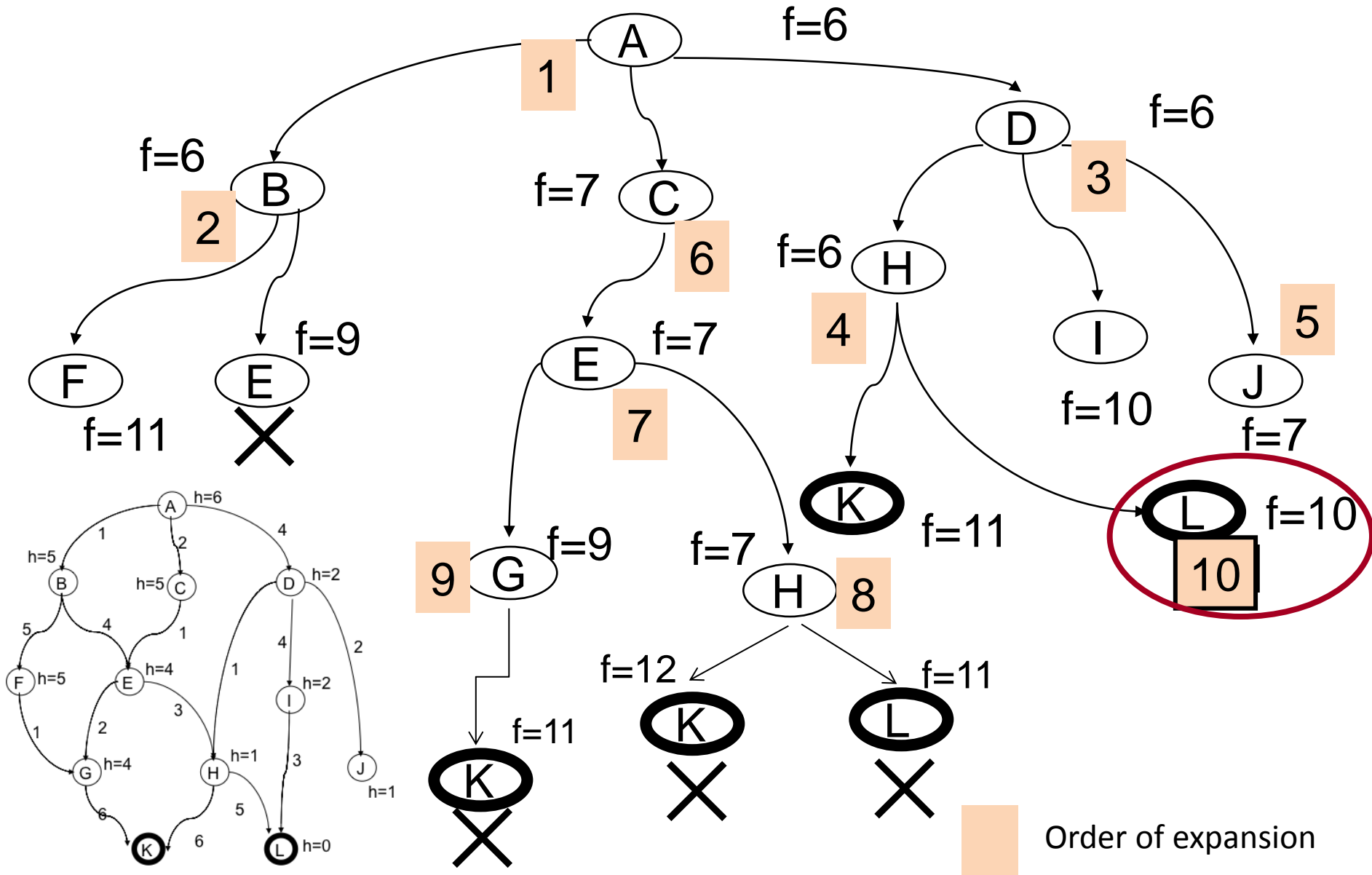
3. A* search: another example

Order of expansion

A f=6

1

D f=6

3

f=6 B

2

f=7 C

6

f=6 H

4

5

F

f=9 E

10

f=11 E f=7

7

I f=10

J f=7

f=11

K f=11

L f=10

12

f=11 G

f=9 H f=9

11

f=9 G

9

f=7 H 8

L f=10

f=14

f=12 f=11

K L

K f=13

K

f=11

Order of expansion

3. A* search: GRAPH-SEARCH version

Order of expansion
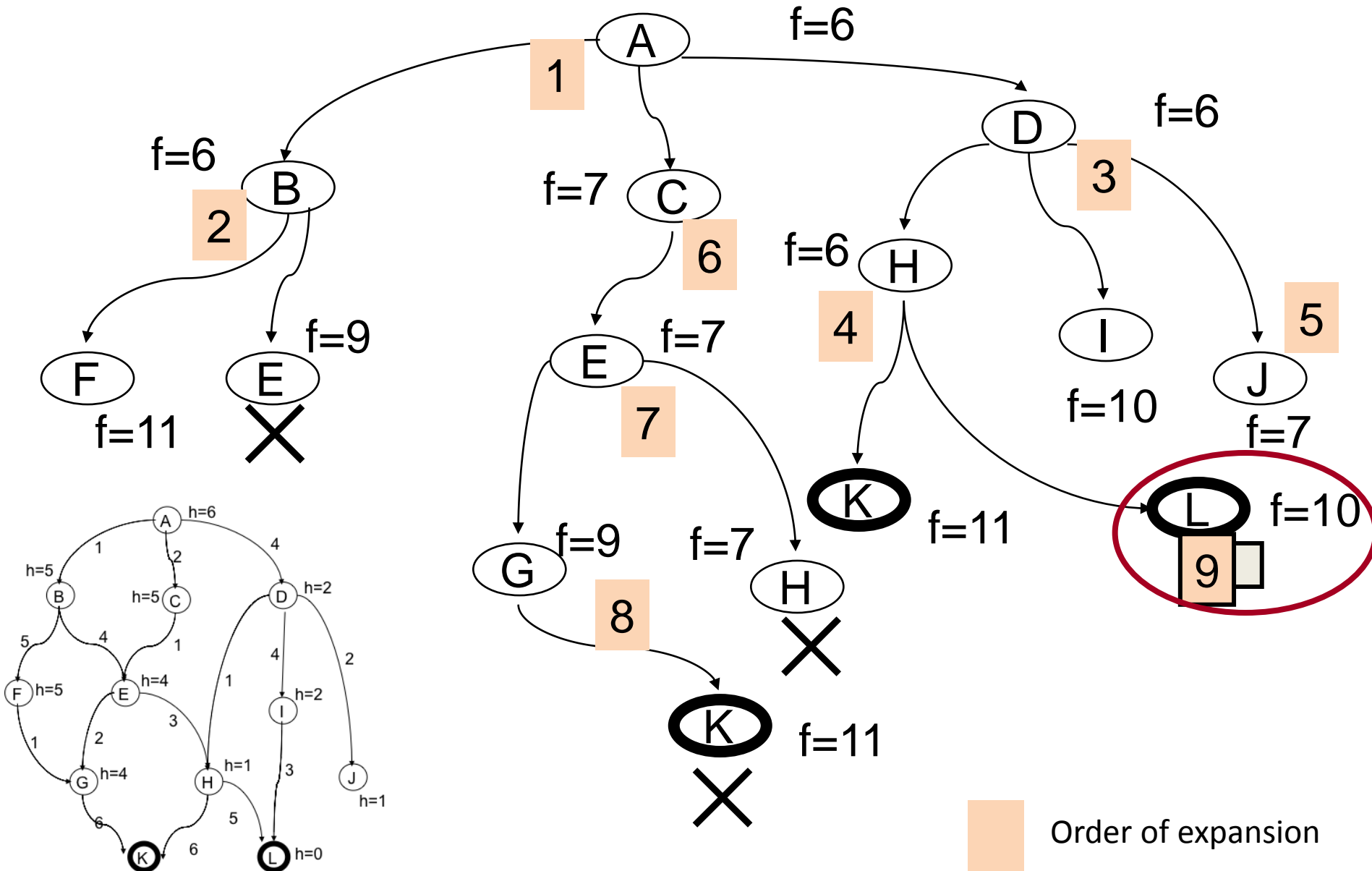
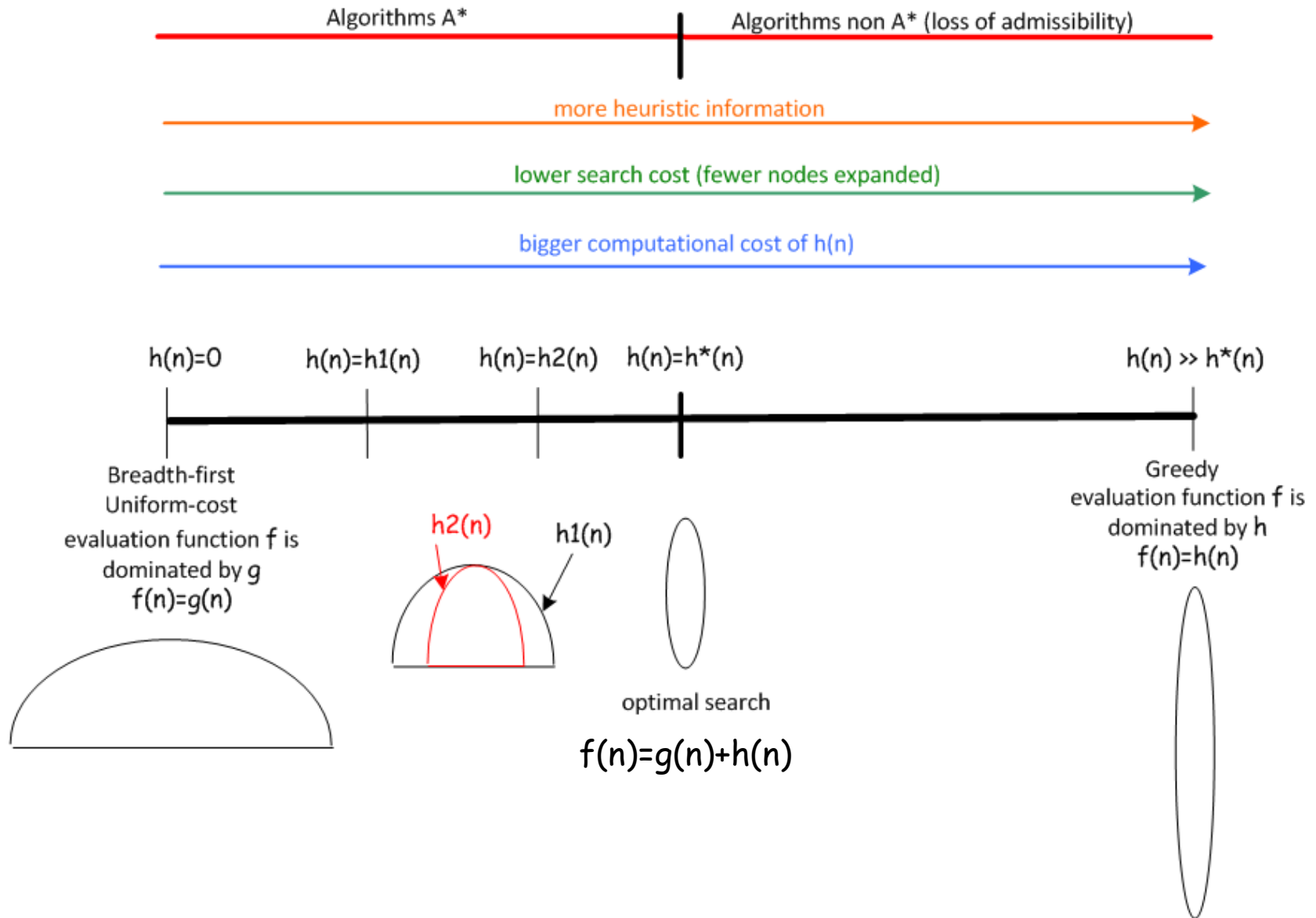# 3. A* search: comparison and analysis

- Comparison to other search strategies:
  - Breadth-first search (optimal if all operators have the same cost). Equivalent to $f(n)=level(n)+0$, where $h(n)=0 < h^{*}(n)$
  - Uniform-cost (optimal). Equivalent to $f(n)=g(n)+0$ where $h(n)=0 < h^{*}(n)$
  - Depth-first (non-optimal). Not comparable to A*

- Heuristic knowledge:
  - $h(n)=0$, lack of knowledge
  - $h(n)=h^{*}(n)$, maximal knowledge
  - if $h2(n) \geq h1(n)$ for all n (both admissible) then h2 dominates h1 (h2 is more informed than h1); h2 will never expand more nodes than h1

3. A* search: comparison and analysis

3. A* search: comparison and analysis

h(n) = 0: no computational cost of h(n). (Slow search. Admissible.)
h(n)=h*(n): large computational cost of h(n). (Fast search. Admissible).
h(n)> h*(n): very high computational cost of h(n). (Very fast search. No admissible)

In general, h^* is not known but it is possible to state whether h is a lower bound of h^* or not.

For difficult problems, it is recommended to highly reduce the search space. In these cases, it is worth using h(n) > h*(n) to eventually find a solution at a reasonable cost even though this is not the optimal solution.

For each problem, find a balance between the search cost and the cost of the solution path found

3. A* search: optimality

Conditions for A* optimality:

1. $h(n)$ must be an admissible heuristic
2. $h(n)$ must be a consistent heuristic (only required for GRAPH-SEARCH version of A* if nodes in the CLOSED list are not re-expanded)

Admissibility:

- $h(n)$ is admissible if it never oversestimates the cost to reach the goal ($\forall n, h(n) \leq h^*(n)$)
- Let G be an optimal goal state, and n a node on an optimal path to G. $f(n)$ never overestimates the true cost of a solution through n
  - $f(n) = g(n)+h(n) \leq g(n)+h^*(n) = g(G) = f(G) \Rightarrow f(n) \leq g(G)$ (1)

Prove that if $h(n)$ is admissible, then A* is optimal
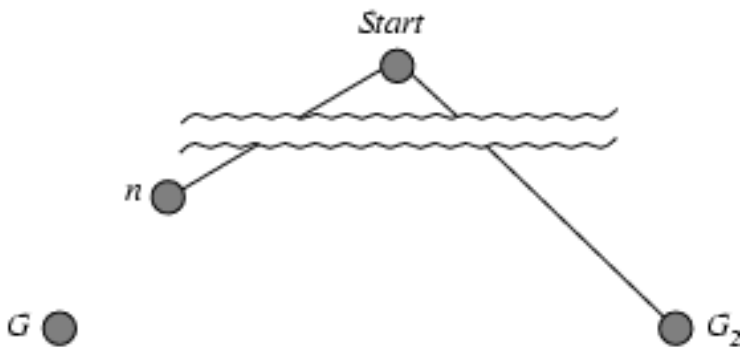
G is an optimal goal state with path cost $f(G) = g(G)$

$G2$ is a suboptimal goal state in the OPEN LIST with path cost $f(G2) = g(G2) > g(G)$ (2)

n is a node in the OPEN LIST on an optimal path to G

if n is not chosen for expansion then $f(n) >= f(G2)$ (3)

if we combine (1) and (3) we get
$f(G2) <= f(n) <= g(G) \Rightarrow g(G2) \leq g(G)$
this contradicts (2) so n is chosen first for expansion

3. A* search: optimality

Consistency, also called monotonicity is a slightly stronger condition than admissibility:

$h(n)$ is consistent if, for every node **n** and every successor **n'** of **n** generated by any action **a**:
$$h(n) <= h(n')+c(n,a,n')$$

If:
- we are applying GRAPH-SEARCH of A* with $f(n)=g(n)+h(n)$, and
- $h(n)$ is consistent

then the GRAPH-SEARCH algorithm returns the optimal solution because

- the sequence of expanded nodes by GRAPH-SEARCH version of A* is in non-decreasing order of $f(n)$

- it is guaranteed that when a node **n** is expanded, we have found the optimal solution from the initial state to **n** <u>so it is not necessary to re-expand **n** in case of a repeated instance</u>; i.e. the cost of the first instance of **n** will never be improved by any repeated node (they will be at least as expensive).

3. A* search: evaluation

- Complete:
  - YES; if there exists at least one solution, A* finishes (unless there are infinitely many nodes with $f < f(G)$)
- Optimal:
  - YES, under consistency condition (for GRAPH-SEARCH version)
  - There always exists at least a node **n** on the OPEN LIST that belongs to the optimal solution path from the start node to a goal node
  - If $C*$ is the cost of the optimal solution:
    - A* expands all nodes with $f(n) < C*$
    - A* might expand some nodes with $f(n)=C*$
    - A* expands no nodes with $f(n) > C*$
- Time complexity (execution time):
  - $O(b^{C*/min\_action\_cost})$: exponential with path length
  - Number of nodes expanded is still exponential in the length of the solution

- Space complexity (memory requirements):
  - Exponential: it keeps all generated nodes in memory (as do all GRAPH-SEARCH algorithms)
  - A* usually runs out of space long before it runs out of time
  - Hence, space is the major problem, not time

4. Heuristics for the 8-puzzle problem

Initial state	Goal state
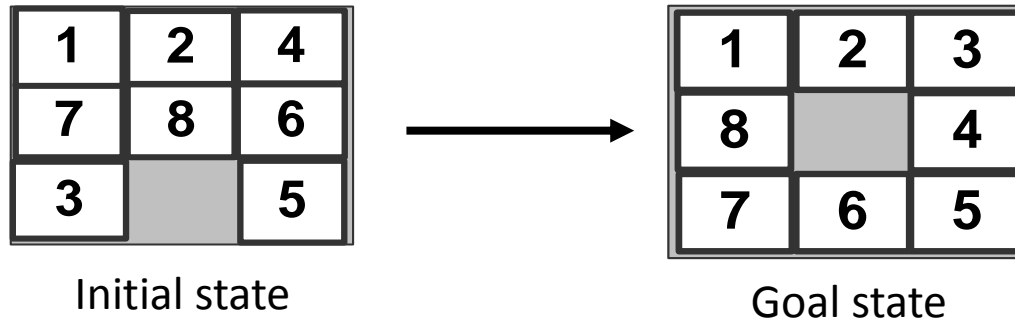
- The 8-puzzle problem
  - Average solution cost is about 22 steps (branching factor $\approx$ 3)
  - Exhaustive search to depth 22: $3^{22} \approx 3.1 \times 10^{10}$ states.
  - A good heuristic function can reduce the search process.

- **Heuristic 1:**
  - h1(n): number of misplaced tiles
  - h1(n)=5 for the example

4. Heuristics for the 8-puzzle problem

1	2	4
7	8	6
3		5

Initial state

1	2	3
8		4
7	6	5

Goal state
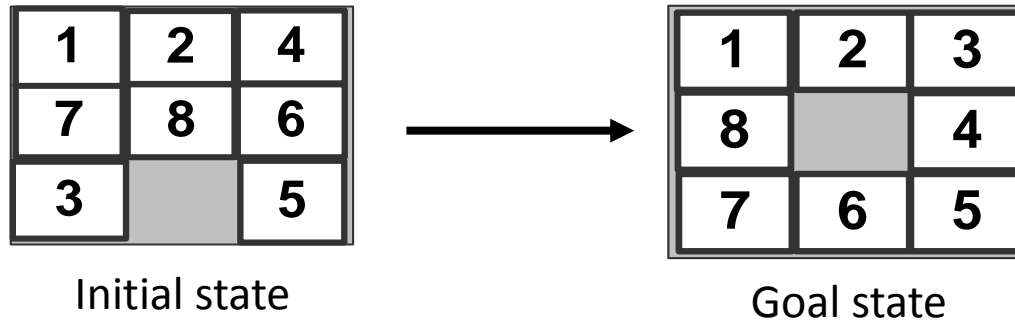
- **Heuristic 2:**
    - h2(n): Manhattan distance (the sum of the distances of the tiles from their goal positions)
    - h2(n)=1+2+4+1+1=9 for the example

Manhattan distance dominates misplaced tiles **(h2(n)>= h1(n), $\forall$n)**

h2 is better for search

4. Heuristics for the 8-puzzle problem

A* search ($f(n)=g(n)+h(n)$) using
h(n)= misplaced tiles