Chapter 04

Informed Search

Instructor

LE Thanh Sach, Ph.D.

Instructor's Information

LE Thanh Sach, Ph.D.

Office:

Department of Computer Science,
Faculty of Computer Science and Engineering,
HoChiMinh City University of Technology.

Office Address:

268 LyThuongKiet Str., Dist. 10, HoChiMinh City, Vietnam.

E-mail: LTSACH@hcmut.edu.vn

E-home: http://cse.hcmut.edu.vn/~ltsach/

Tel: (+84) 83-864-7256 (Ext: 5839)

Acknowledgment

The slides in this PPT file are composed using the materials supplied by

- **Prof. Stuart Russell and Peter Norvig**: They are currently from University of California, Berkeley. They are also the author of the book "Artificial Intelligence: A Modern Approach", which is used as the textbook for the course

- **Prof. Tom Lenaerts**, from Université Libre de Bruxelles

Outline

- Recap
- Best-First Search
- A* Search
- Heuristic

Recap

Recap

- Uninformed search
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
  - DLS,
  - IDS
  - Etc
- Uninformed searches DO NOT use specific knowledge of the problems => HAVE TO explore the state systematically and exhaustively
  - ⇒ Need much memory
  - ⇒ Spend much time

Best-first search

Other names:

- Greedy Search;
- Greedy Best-First Search;
- Best-First Strategy

Best-first search's idea

function TREE-SEARCH(*problem*,*fringe*) **return** a solution or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)
  loop do
    if EMPTY?(*fringe*) **then return** failure
    node ← REMOVE-FIRST(*fringe*)
    if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds
      then return SOLUTION(*node*)
    fringe ← INSERT-ALL(EXPAND(*node*, *problem*), *fringe*)

Best-first search's idea

function **TREE-SEARCH**(*problem*,*fringe*) **return** a solution or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)
  loop do
    if EMPTY?(*fringe*) **then return** failure
    node ← REMOVE-FIRST(*fringe*)
    if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds
      then return SOLUTION(*node*)
    fringe ← INSERT-ALL(EXPAND(*node*, *problem*), *fringe*)

- **Best-First Strategy**: selects the node which appears best
- Use an evaluation function $f(n) = \mathrm{h(n)}$ to give the score for state n
- *Fringe* is a priority queue (heap) to sort states decreasingly in order of desirability

Best-first search's idea

function GRAPH-SEARCH(*problem*,*fringe*) **return** a solution or failure

closed ← an empty set

fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

loop do

if EMPTY?(*fringe*) **then return** failure

node ← REMOVE-FIRST(*fringe*)

if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds

then return SOLUTION(*node*)

if STATE[*node*] is not in *closed* **then**

add STATE[*node*] to *closed*

fringe ← INSERT-ALL(EXPAND(*node*, *problem*), *fringe*)

- **Best-First Strategy**: selects the node which appears best
- Use an evaluation function $f(n) = \mathrm{h(n)}$ to give the score for state n
- *Fringe* is a priority queue (heap) to sort states decreasingly in order of desirability

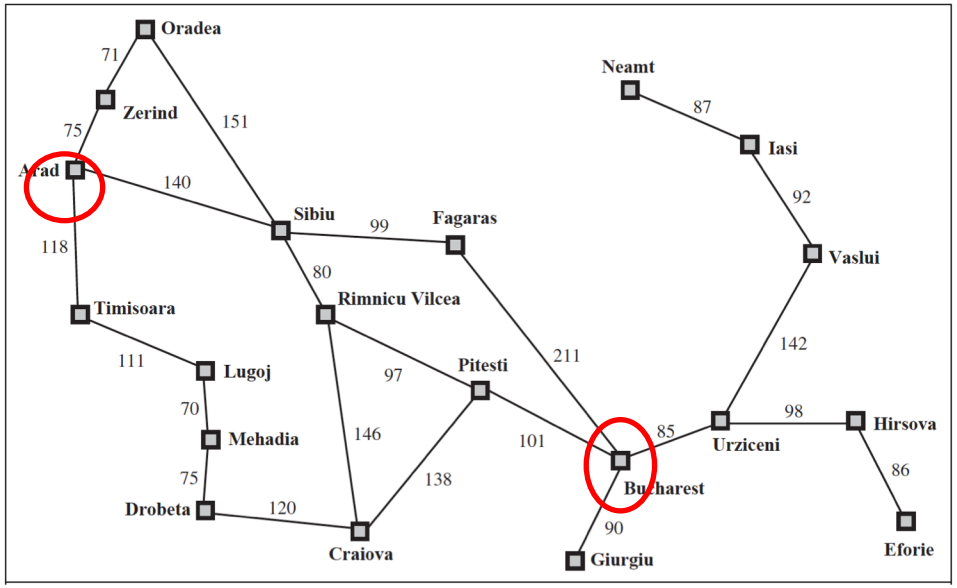
A heuristic function

- [dictionary] *"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 node *n* to goal node.
  - If *n* is goal then $h(n)=0$
  - Best-First Search: $f(n) = h(n)$

More information later.

Romania with step costs in km

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



- h_{SLD}=straight-line distance heuristic.
- h_{SLD} can **NOT** be computed from the problem description itself
- In this example $f(n)=h(n)$
  - Expand node that is closest to goal

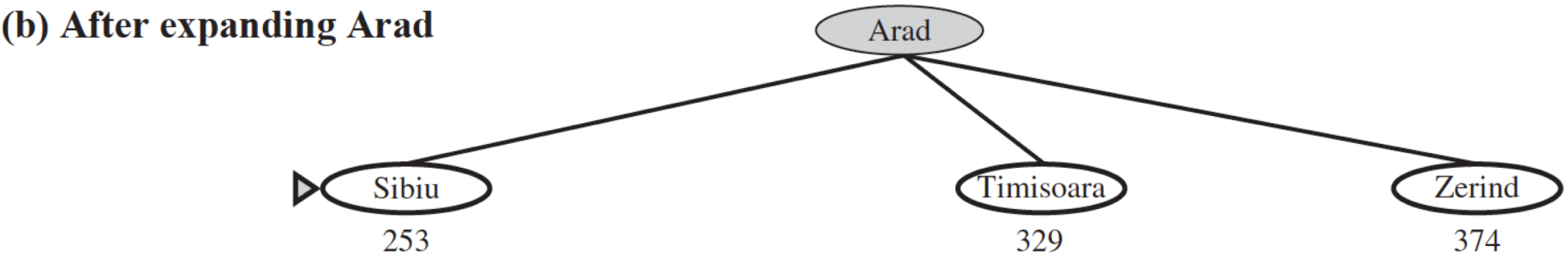
  = Greedy best-first search

Greedy search example

(a) The initial state

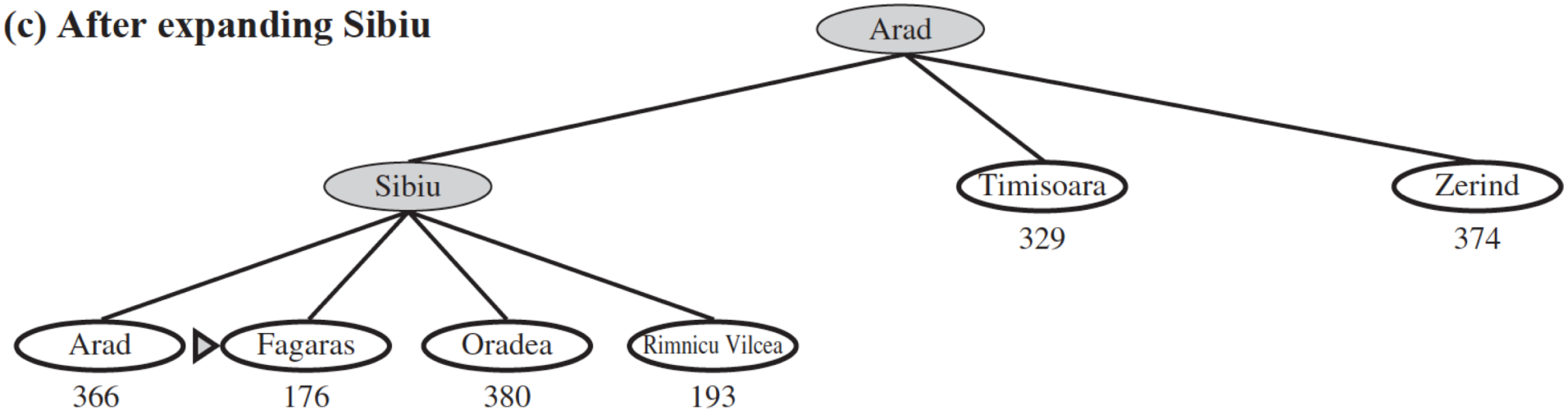
- Assume that we want to use greedy search to solve the problem of travelling from Arad to Bucharest.
- The initial state=Arad

Greedy search example



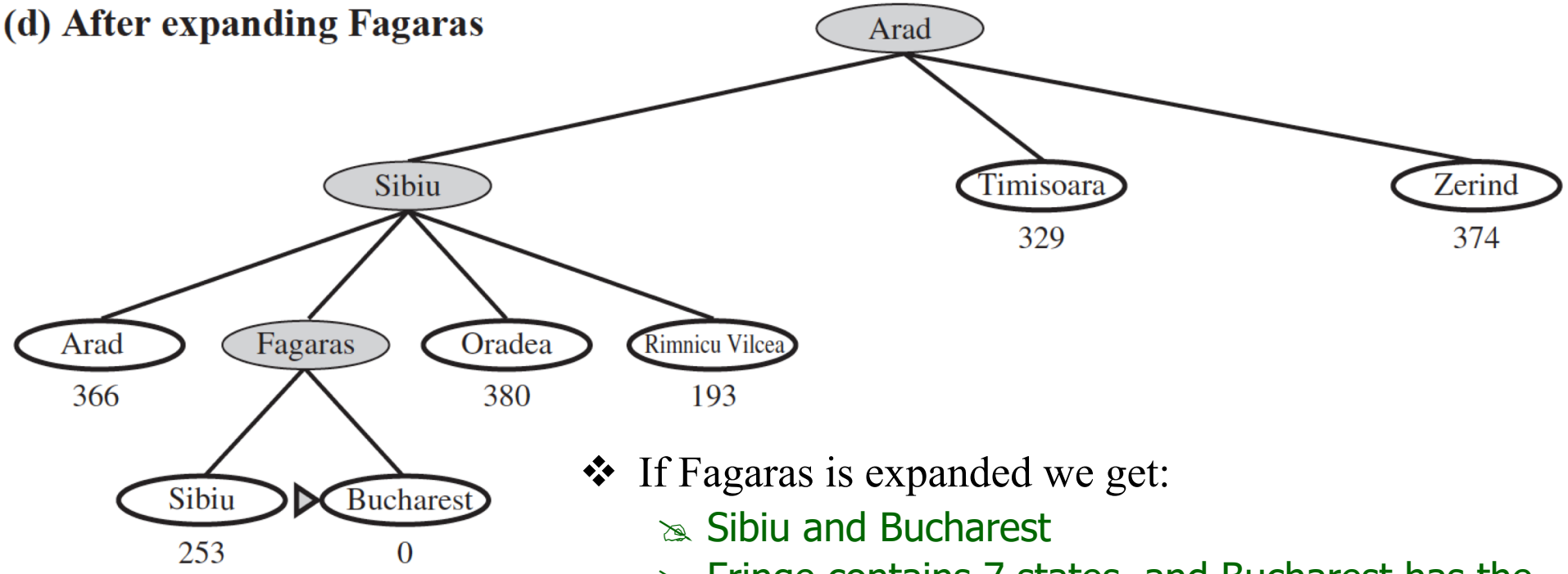
- The first expansion step produces:
  - Sibiu, Timisoara and Zerind
- Greedy best-first will select Sibiu, because its f-cost is smallest.

Greedy search example



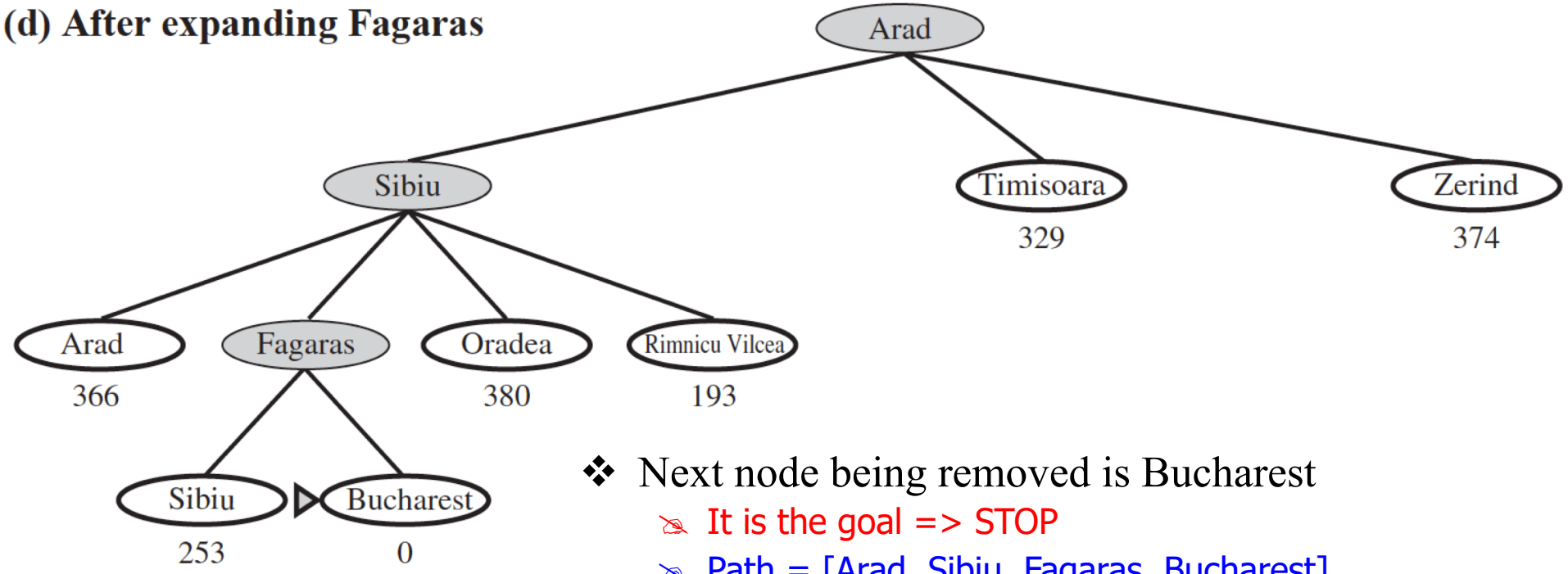
- If Sibiu is expanded we get:
  - Arad, Fagaras, Oradea and Rimnicu Vilcea (using Tree-Search, and not detected for repeated state Arad)
- Greedy best-first search will select: Fagaras
  - Before removing Fagaras for processing, there are 6 states in fringe.

Greedy search example



- If Fagaras is expanded we get:
  - Sibiu and Bucharest
  - Fringe contains 7 states, and Bucharest has the smallest f-cost

Greedy search example

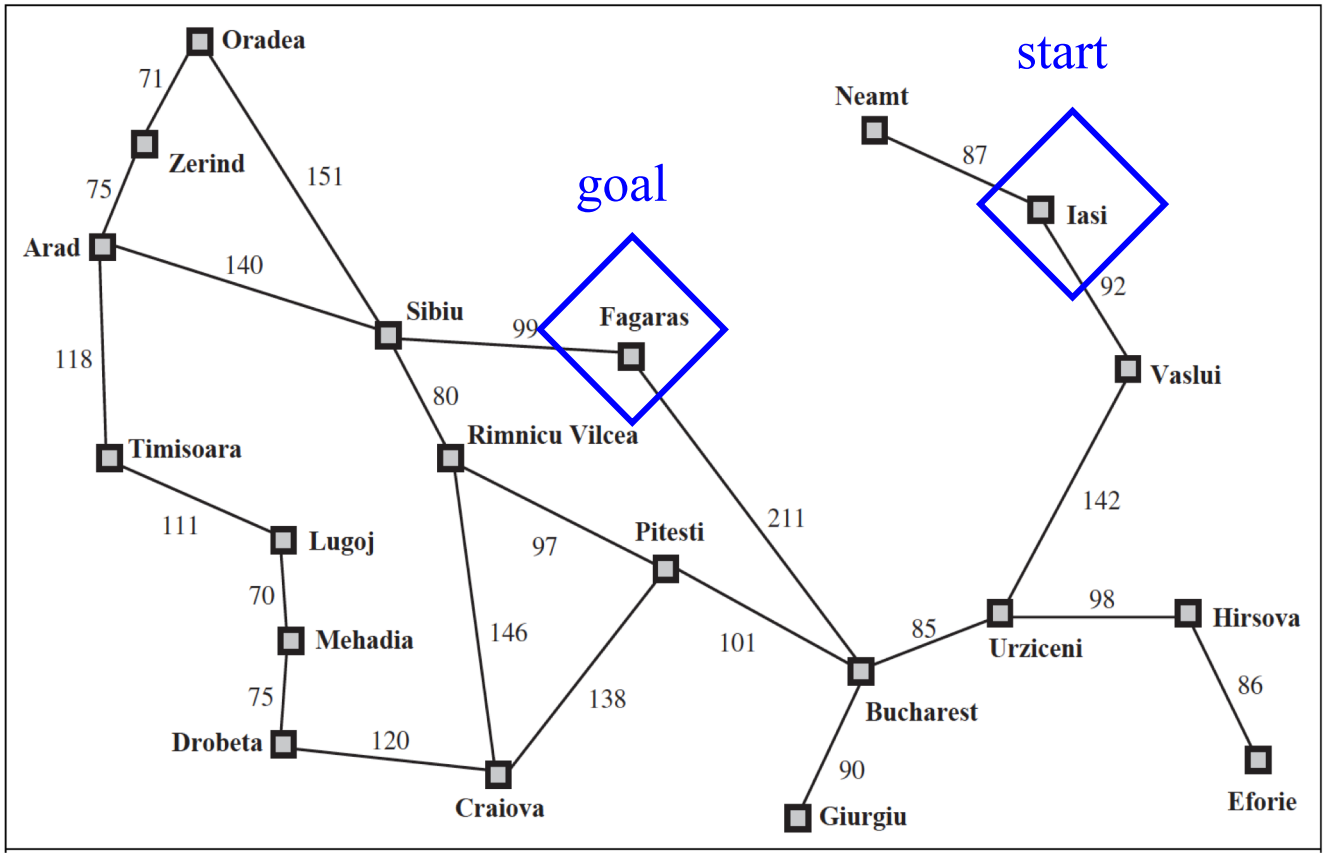


- Next node being removed is Bucharest
  - It is the goal => STOP
  - Path = [Arad, Sibiu, Fagaras, Bucharest]
  - Path-Cost = 140 + 99 + 211 = 450

- Yet not optimal
  - Optimal path = [Arad, Sibiu, Rimnicu Vilcea, Pitesti]
  - Optimal-path's cost = 140 + 80 + 97 + 101 = 418

Greedy search, evaluation

- Completeness:
  - Using Best-First Stategy with Tree-Search: not complelete
    - ✓ Because of repeated states

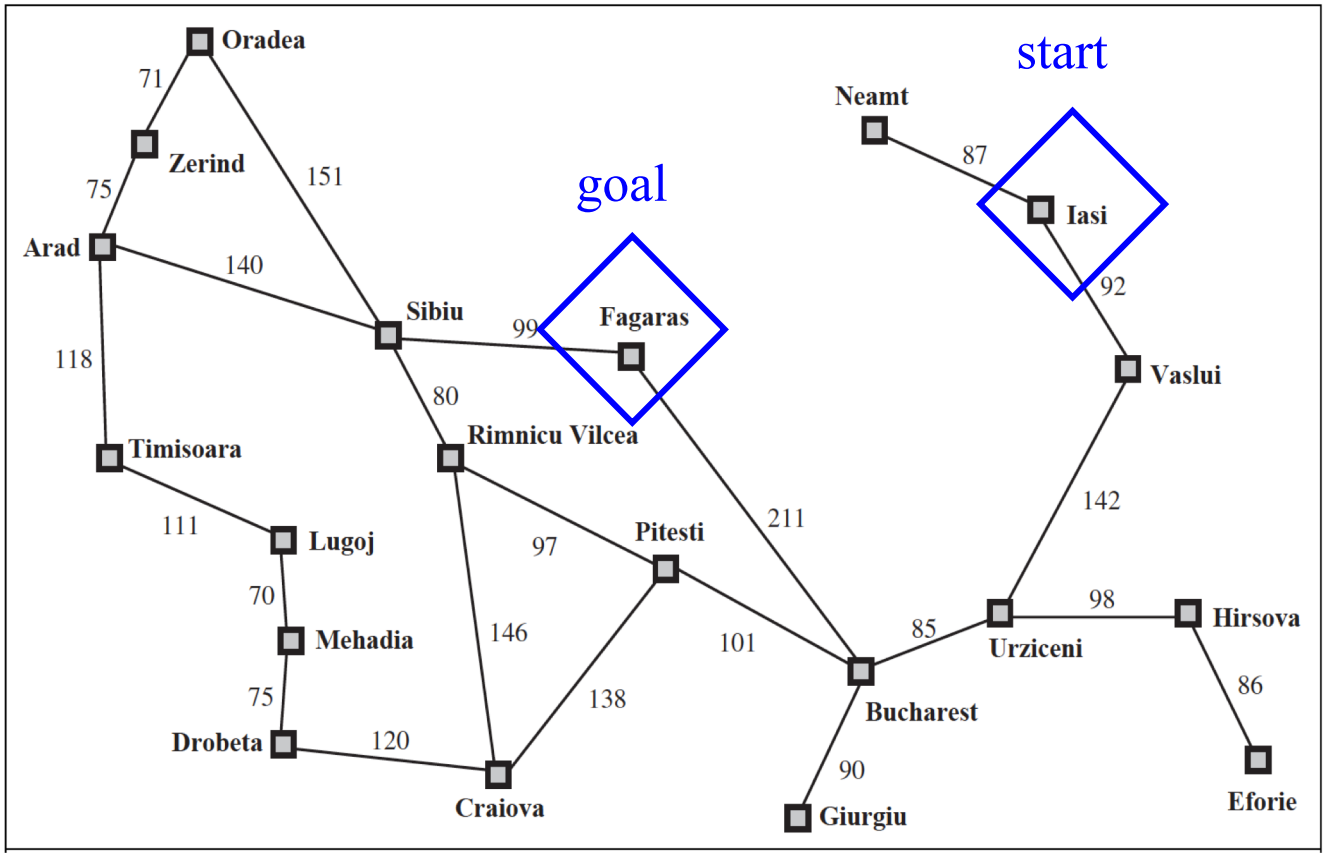


States being explored:

Iasi → Neamt
→ Iasi → Neamt
→ Iasi → Neamt
→ … (forever)

Greedy search, evaluation

- Completeness:
  - Using Best-First Stategy with Graph-Search: complelete
    - ✓ Because of removing repeated states



States being explored:

Iasi
→ Neamt
→ Vaslui
→ Urziceni
→ Bucharest
→ Fagaras

Greedy search, evaluation

- Completeness: Yes/No
- Time complexity?
  - Cfr. Worst-case DF-search $O(b^m)$

    (with m is maximum depth of search space)
  - Good heuristic can give dramatic improvement.

Greedy search, evaluation

- Completeness: Yes/No
- Time complexity: $O(b^m)$
- Space complexity: $O(b^m)$
  - Keeps all nodes in memory

Greedy search, evaluation

- Completeness: Yes/No
- Time complexity: $O(b^m)$
- Space complexity: $O(b^m)$
- Optimality? NO
  - Same as DF-search

Greedy search, evaluation

- Completeness: Yes/No
- Time complexity: $O(b^m)$
- Space complexity: $O(b^m)$ ⟸ (1) A big problem (out-of-memory error soon)
- Optimality? No

(2) The peformance of Best-First Search depends heavily heuristics and the state space itself.

(3) In worst case, Best-First Search combines the worst features of Breadth-First Search and Dept-First Search

A-Star Search (A*)

- Idea
- Example

A-Star's idea

function **TREE-SEARCH**(*problem*,*fringe*) **return** a solution or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)
  loop do
    if EMPTY?(*fringe*) **then return** failure
    node ← REMOVE-FIRST(*fringe*)
    if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds
      then return SOLUTION(*node*)
    fringe ← INSERT-ALL(EXPAND(*node*, *problem*), *fringe*)

- **A***: selects the node which appears best
- Use an evaluation function $f(n) = g(n) + h(n)$ to give the score for state n
- *Fringe* is a priority queue (heap) to sort states decreasingly in order of desirability

A-Star's idea

function GRAPH-SEARCH(*problem*,*fringe*) **return** a solution or failure

closed ← an empty set

fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

loop do

if EMPTY?(*fringe*) **then return** failure

node ← REMOVE-FIRST(*fringe*)

if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds

then return SOLUTION(*node*)

if STATE[*node*] is not in *closed* **then**

add STATE[*node*] to *closed*

fringe ← INSERT-ALL(EXPAND(*node*, *problem*), *fringe*)

- **A***: selects the node which appears best
- Use an evaluation function $f(n) = g(n) + h(n)$ to give the score for state n
- *Fringe* is a priority queue (heap) to sort states decreasingly in order of desirability

A* search

- Evaluation function $f(n)=g(n) + h(n)$
  - $g(n)$ the cost (so far) to reach the node.
  - $h(n)$ estimated cost to get from the node to the goal.
  - $f(n)$ estimated total cost of path from the start to the goal state through state n
- Best-known form of best-first search.
- Idea: avoid expanding paths that are already expensive.

A* search

- A* search uses an admissible heuristic
  - A heuristic is admissible if it *never overestimates* the cost to reach the goal
  - Are optimistic

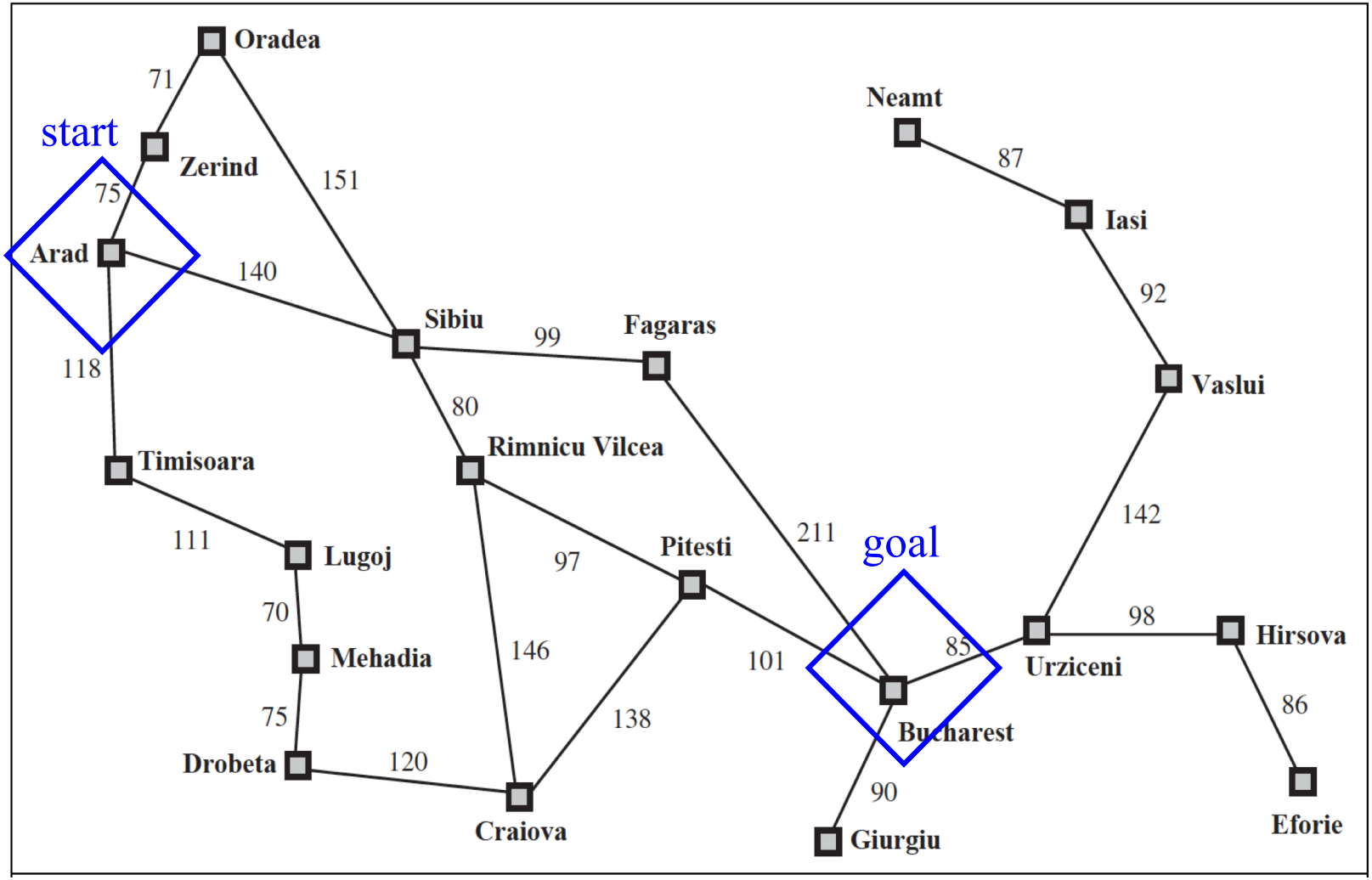
Formally:

1. $h(n) <= h^*(n)$ where $h^*(n)$ is the true cost from n
2. $h(n) >= 0$ so $h(G)=0$ for any goal G.

e.g. $h_{SLD}(n)$ never overestimates the actual road distance

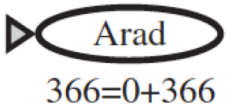
(admissible: chấp nhận được)

Romania example



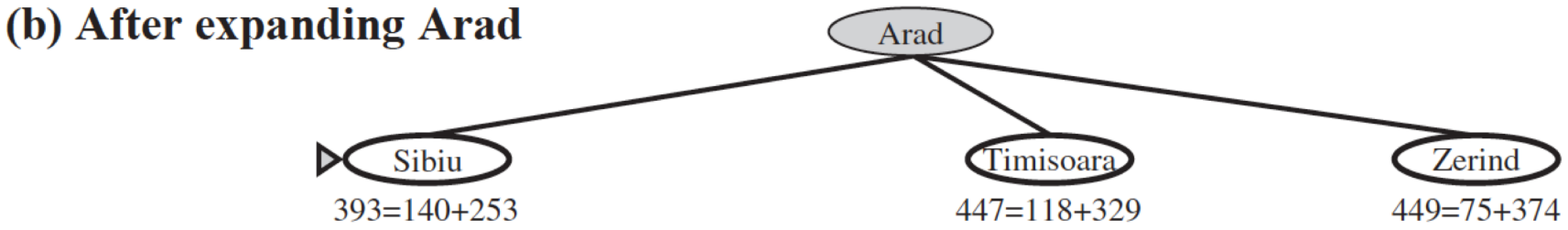
A* search example

(a) The initial state



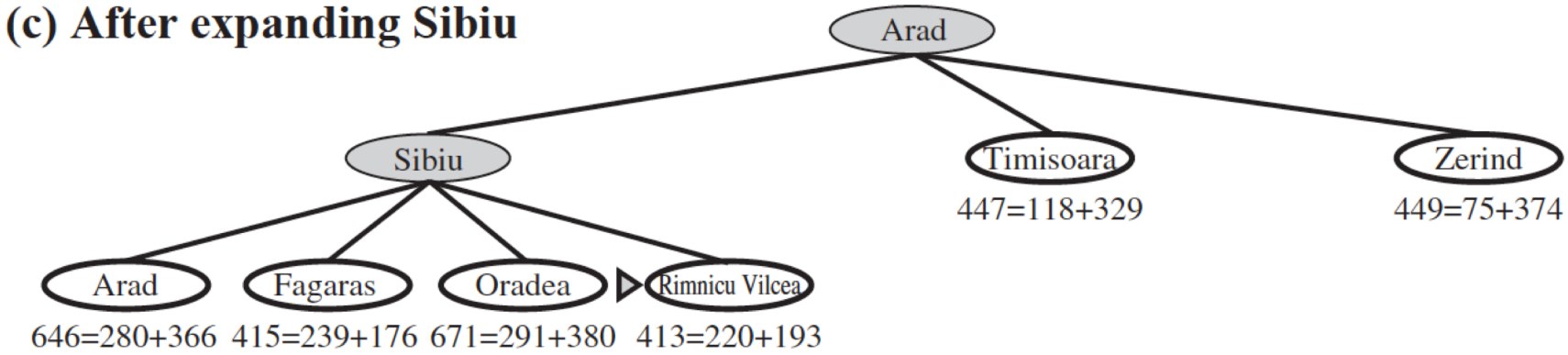
- Find Bucharest starting at Arad
  - f(Arad) = g(Arad)+h(Arad)= 0+366= 366

A* search example



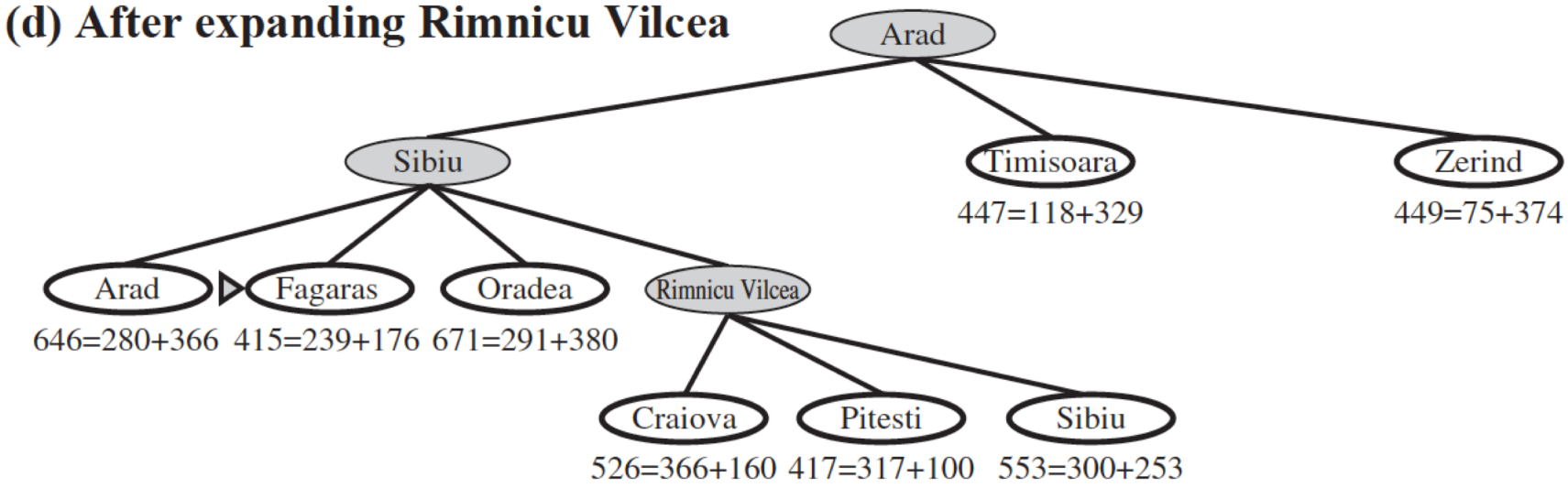
- Expand Arad and determine *f(n)* for each node
  - f(Sibiu)=g(Sibiu)+h(Sibiu)=140+253=393
  - f(Timisoara)=g(Timisoara)+h(Timisoara)=118+329=447
  - f(Zerind)= g(Zerind)+h(Zerind)=75+374=449
- Best choice is Sibiu
  - Before removing Sibiu for processing, there are 3 states in fringe

A* search example



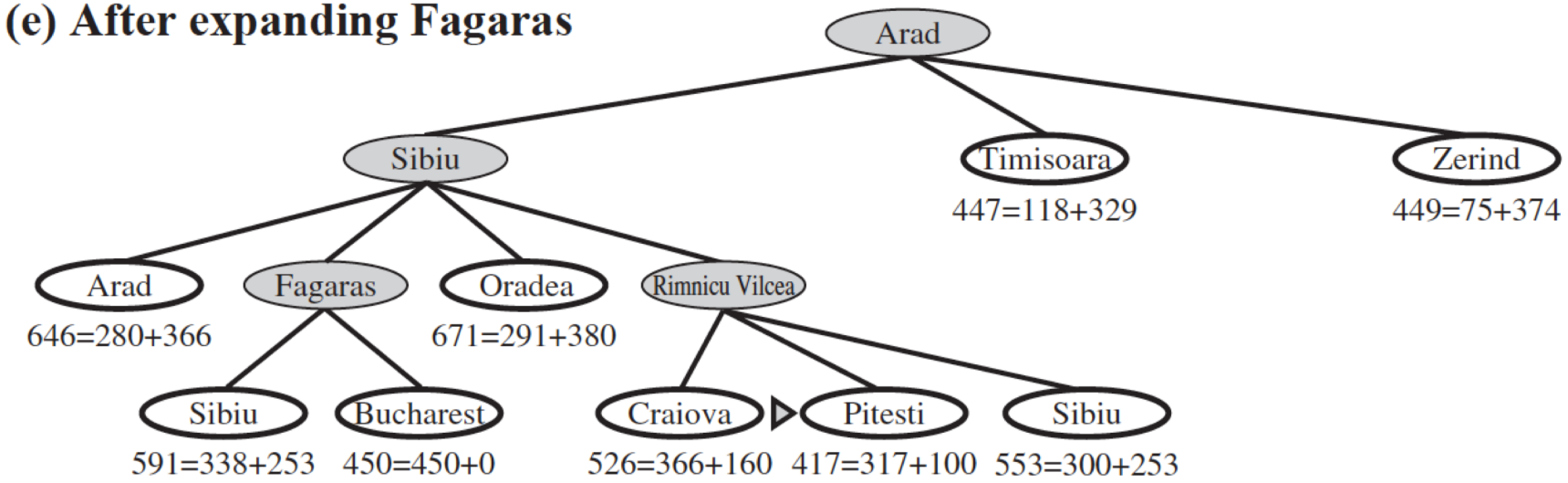
- Expand Sibiu and determine *f(n)* for each node
  - f(Arad)=g(Arad)+h(Arad)=280+366=646
  - f(Fagaras)= g(Fagaras)+h(Fagaras)=239+179=415
  - f(Oradea)= g(Oradea)+h(Oradea)=291+380=671
  - f(Rimnicu Vilcea)=g(Rimnicu Vilcea)+ h(Rimnicu Vilcea)=220+192=413
- Best choice is Rimnicu Vilcea

A* search example



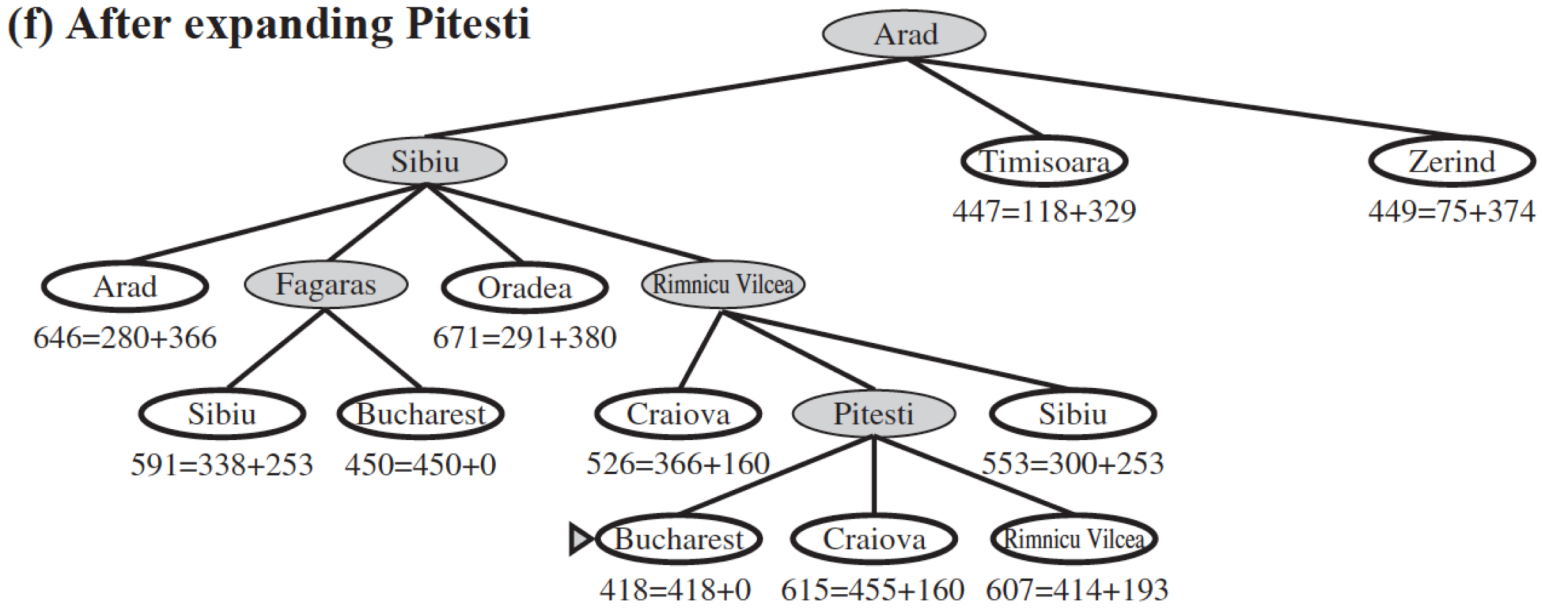
- Expand Rimnicu Vilcea and determine *f(n)* for each node
  - f(Craiova)=g(Craiova)+h(Craiova)=360+160=526
  - f(Pitesti)=g(Pitesti)+h(Pitesti)=317+100=417
  - f(Sibiu)= g(Sibiu)+h(Sibiu)=300+253=553
- Best choice is Fagaras

A* search example



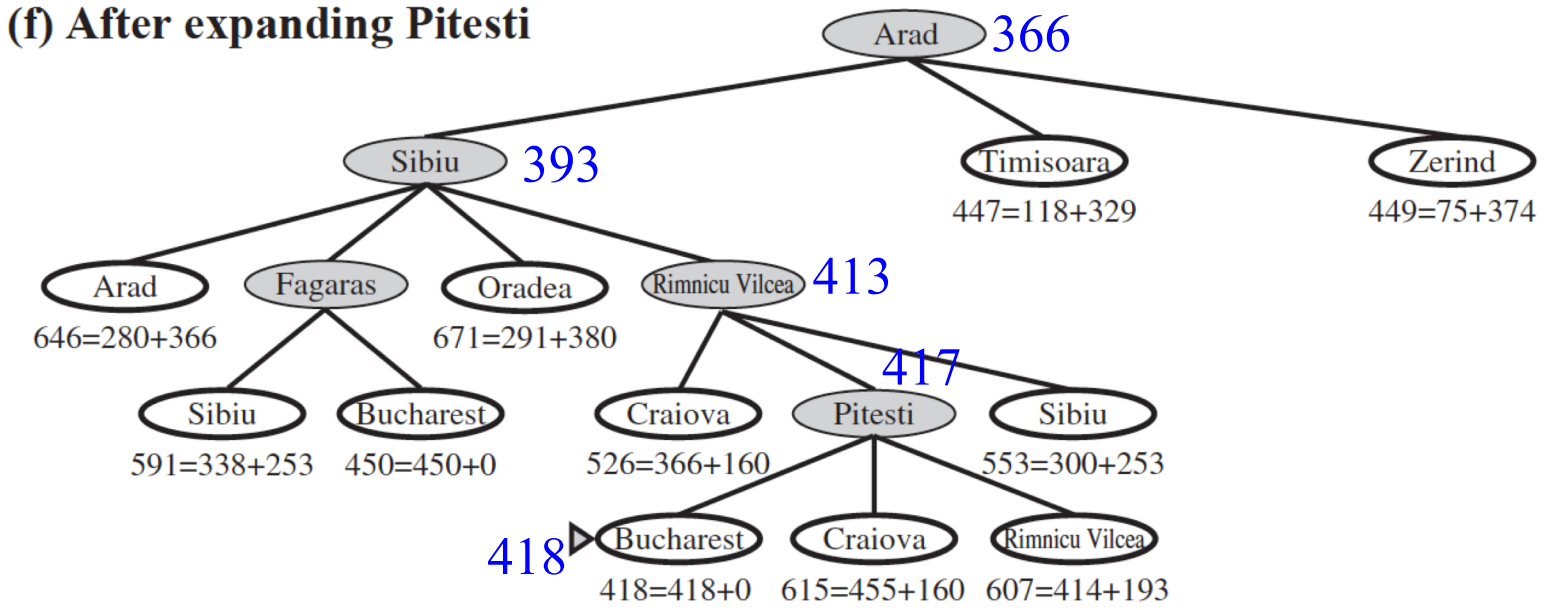
- Expand Fagaras and determine *f(n)* for each node
  - f(Sibiu)= g(Sibiu)+h(Sibiu)=338+253=591
  - *f(Bucharest)=g(Bucharest)+h(Bucharest)=450+0=450*
- Best choice is Pitesti !!!
  - Even Bucharest (the goal) has been generated and placed into fringe
  - But, it is not selected!
  - Before removing Pitesti, there are 9 states in fringe

A* search example



- Expand Pitesti and determine *f(n)* for each node
  - *f(Bucharest)=g(Bucharest)+h(Bucharest)=418+0=418*
- Best choice is Bucharest !!!
  - Optimal solution (only if *h(n)* is admissable)

A* search example

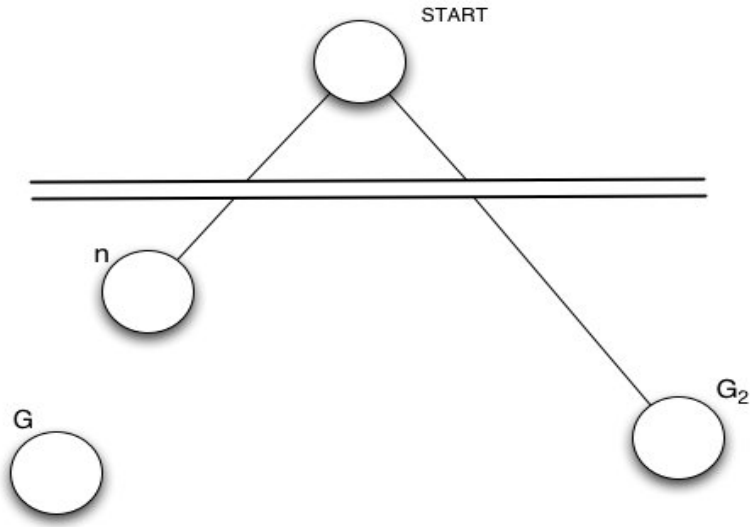


- Expand Pitesti and determine *f(n)* for each node
  - *f(Bucharest)=g(Bucharest)+h(Bucharest)=418+0=418*
- Best choice is Bucharest !!!
  - Optimal solution (only if *h(n)* is admissable)
- Note values along optimal path !!

A-Star Search (A*)

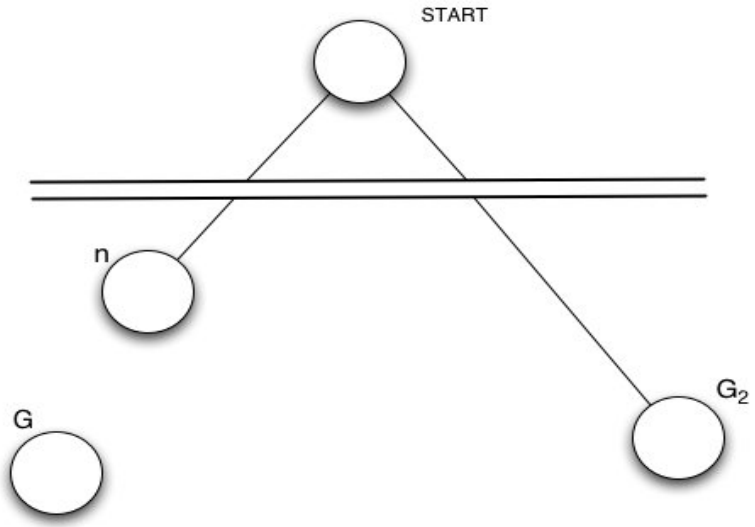
- Optimality

Optimality of A* (with Tree-Search)



- Assume that
  - G and G_2 are goals and path[start → G] is shorter than path[start → G_2] (or G2: suboptimal)
  - n is any node on path[start → G]
- We need to prove:
  - A* selects n instead of G_2
  - Or, $f(G_2) > f(n)$

Optimality of A* (with Tree-Search)



- Steps:

$f(G_2)$	$= g(G_2)$	since $h(G_2)=0$
	$> g(G)$	since G_2 is suboptimal
	$> g(n) + c(n, G)$	since $g(G) = g(n) + c(n, G)$; $c(n,G)$: cost $n \rightarrow G$
	$> f(n)$	since $f(n) = g(n) + h(n)$; $c(n,G) >= h(n)$ since h is admissible

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Optimality of A* (with Graph-Search)

function GRAPH-SEARCH(*problem*,*fringe*) **return** a solution or failure

closed ← an empty set

fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

loop do

if EMPTY?(*fringe*) **then return** failure

node ← REMOVE-FIRST(*fringe*)

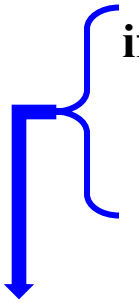
if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds

then return SOLUTION(*node*)

if STATE[*node*] is not in *closed* **then**

add STATE[*node*] to *closed*

fringe ← INSERT-ALL(EXPAND(*node*, *problem*), *fringe*)



❖ Discards new paths to repeated state.

- Previous proof breaks down

Optimality of A* (with Graph-Search)

- Solution:
  - Add extra bookkeeping i.e. remove more expensive of two paths.
  - Ensure that optimal path to any repeated state is always first followed.
    - ✓ Extra requirement on *h(n)*: consistency (monotonicity)

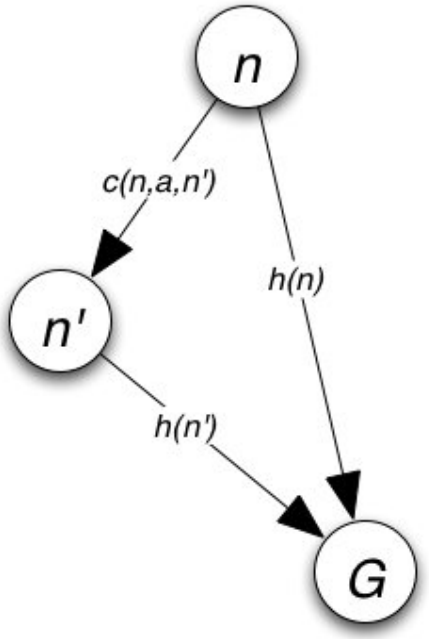
Optimality of A* (with Graph-Search)
consistent (nhất quán)

- A heuristic is consistent if

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

- If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &\geq f(n) \end{aligned}$$



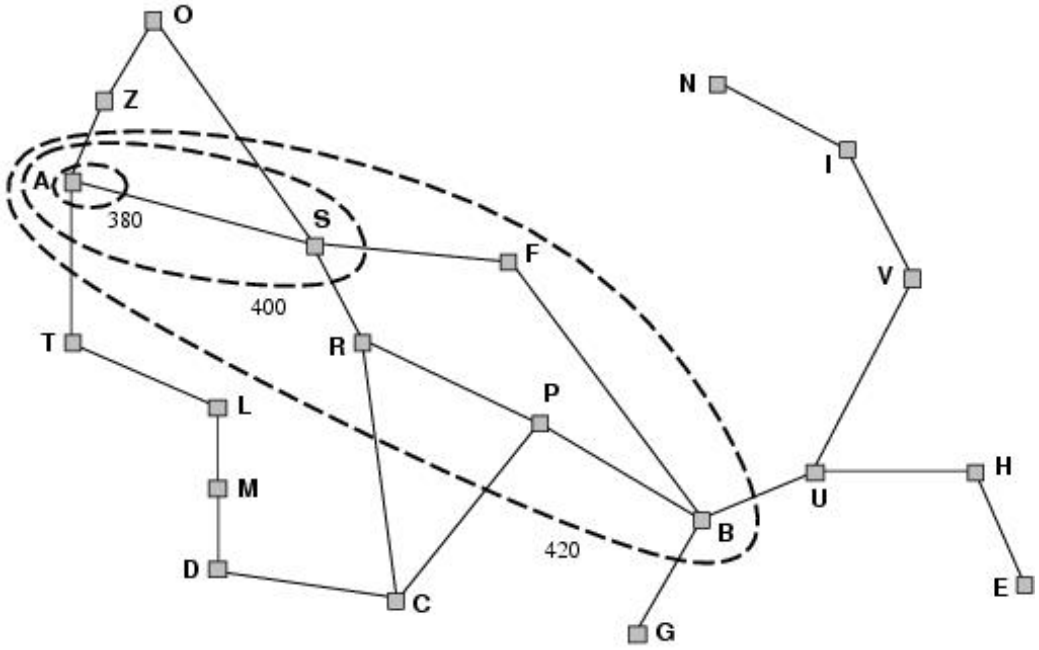
i.e. *f(n)* is nondecreasing along any path.

Optimality of A*(more useful)

- A* expands nodes in order of increasing *f* value
  - Whener A* hits a goal, it has arrived the goal on the optimal path

- Contours can be drawn in state space
  - Uniform-cost search adds circles.

  - f-contours are gradually added:
    1) nodes with $f(n)<C^*$
    2) Some nodes on the goal Contour ($f(n)=C^*$).



A* search, evaluation

- Completeness: YES
  - Since bands of increasing *f* are added
  - Unless there are infinitily many nodes with *f<f(G)*

A* search, evaluation

- Completeness: YES
- Time complexity:
  - Number of nodes expanded is still exponential in the length of the solution.

A* search, evaluation

- Completeness: YES
- Time complexity: (exponential with path length)
- Space complexity:
  - It keeps all generated nodes in memory
  - Hence space is the major problem not time

A* search, evaluation

- Completeness: YES
- Time complexity: (exponential with path length)
- Space complexity:(all nodes are stored)
- Optimality: YES
  - Cannot expand f_{i+1} until f_i is finished.
  - A* expands all nodes with $f(n) < C^*$
  - A* expands some nodes with $f(n) = C^*$
  - A* expands no nodes with $f(n) > C^*$

Also *optimally efficient* (not including ties)

A-Star Search (A*)

Example with 8-Puzzle

8-puzzle heuristics

n

4	6	5
7	2	
1	3	8

Goal

3	1	8
7	4	2
6	5	

Tile	n	Goal
1	1	0
2	1	0
3	1	0
4	1	0
5	1	0
6	1	0
7	0	0
8	1	0
$h_1(n)$	7	
$h_1(Goal)$	0	

Heuristic function: $h_1(n)$ = the number of misplaced tiles

8-puzzle heuristics

n

4	6	5
7	2	
1	3	8

Goal

3	1	8
7	4	2
6	5	

$h_1(n)$: admissible?
Yes!
Why?

Tile	n	Goal
1	1	0
2	1	0
3	1	0
4	1	0
5	1	0
6	1	0
7	0	0
8	1	0
$h_1(n)$	7	
$h_1(Goal)$	0	

Heuristic function: $h_1(n)$ = the number of misplaced tiles

8-puzzle heuristics

n

4	6	5
7	2	
1	3	8

Goal

3	1	8
7	4	2
6	5	

Tile	n	Goal
1	3	0
2	1	0
3	3	0
4	2	0
5	3	0
6	3	0
7	0	0
8	2	0
h_2(n)	**17**	
h_2(Goal)	**0**	

Heuristic function: h_2(n)= the sum of the distances of the tiles from their goal positions

8-puzzle heuristics

n

4	6	5
7	2	
1	3	8

Goal

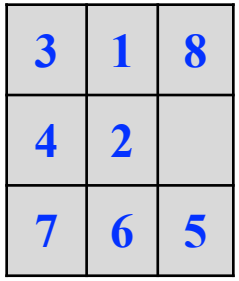
3	1	8
7	4	2
6	5	

$h_2(n)$: admissible?
Yes!
Why?

Tile	n	Goal
1	3	0
2	1	0
3	3	0
4	2	0
5	3	0
6	3	0
7	0	0
8	2	0
$h_2(n)$	**17**	
$h_2(Goal)$	**0**	

Heuristic function: $h_2(n)$= the sum of the distances of the tiles from their goal positions

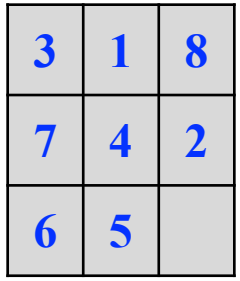
8-puzzle with A*



Start

Heuristic function: $h_1(n)$ = the number of misplaced tiles

Steps of A*?



Goal

3	1	8
4	2	
7	6	5

h(start): 5

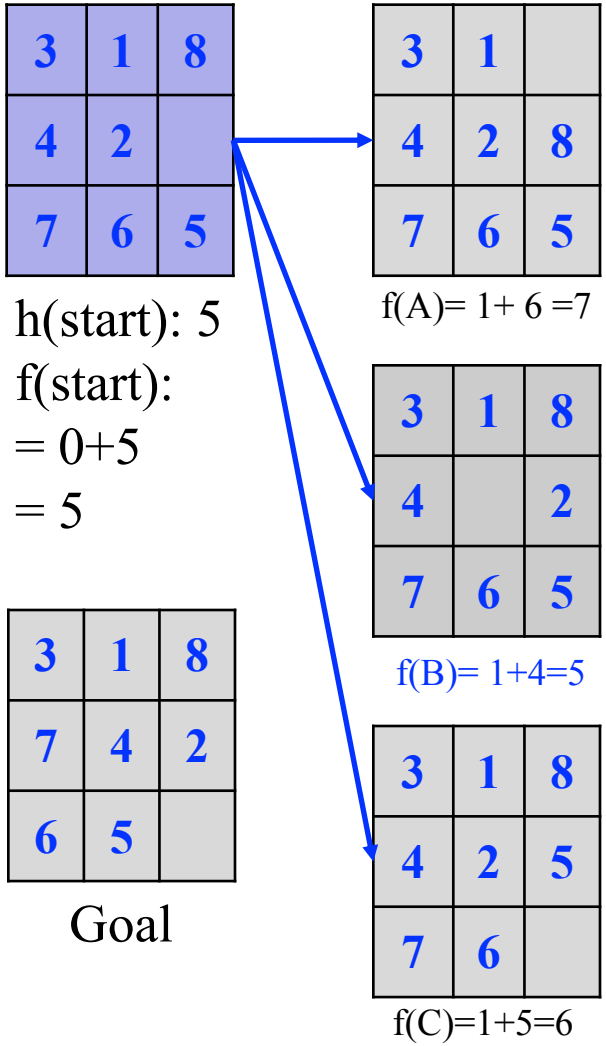
f(start):

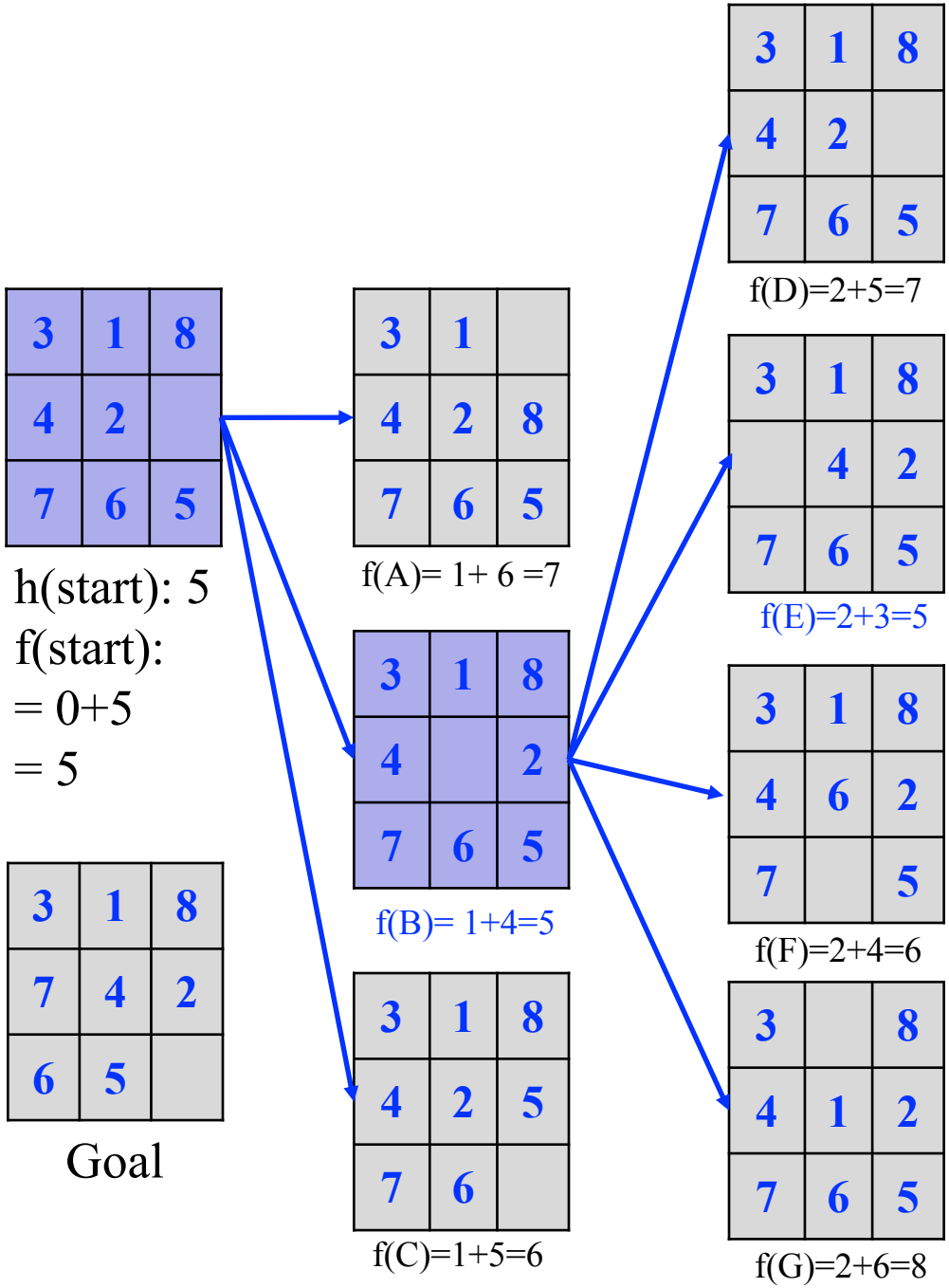
= 0+5

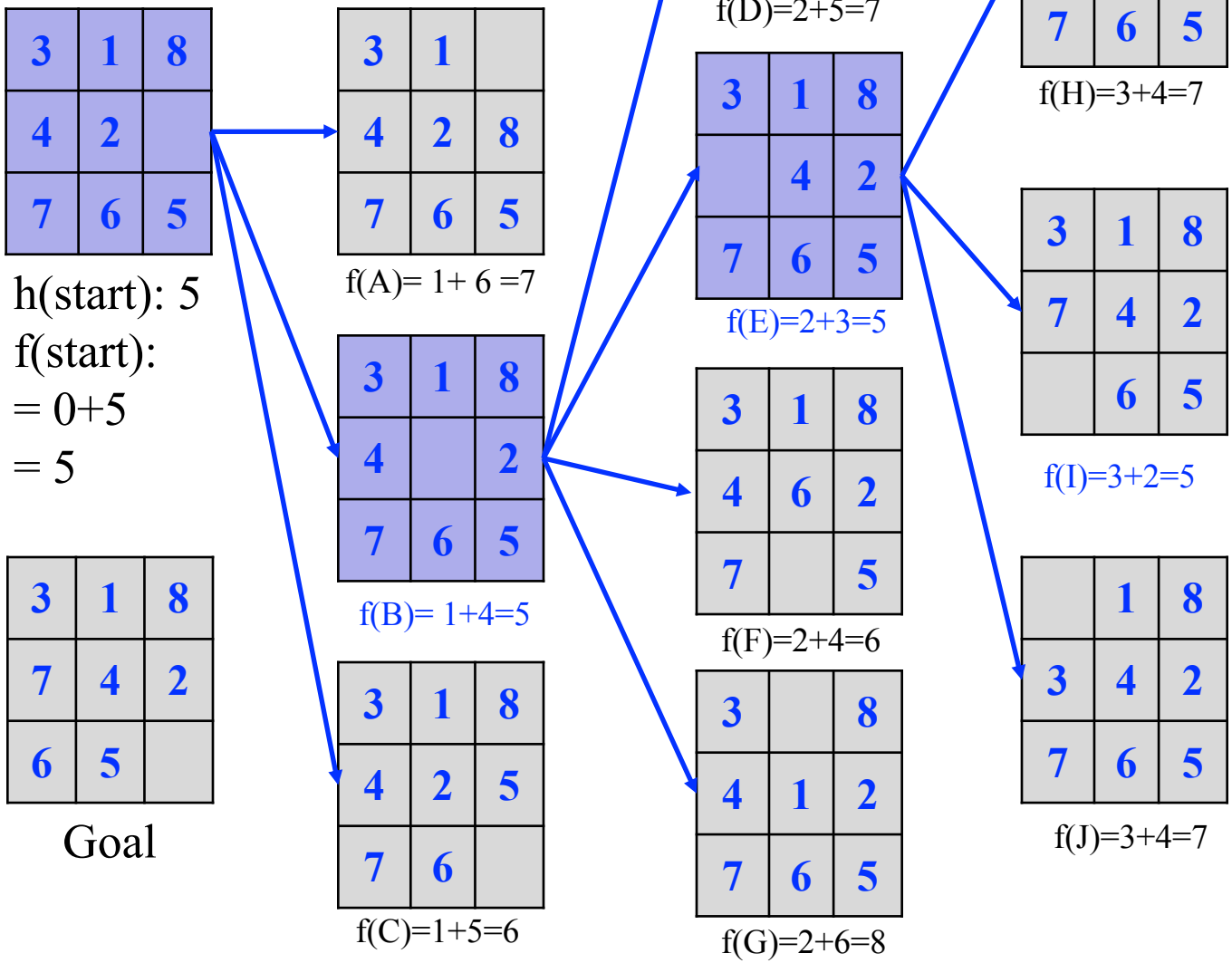
= 5

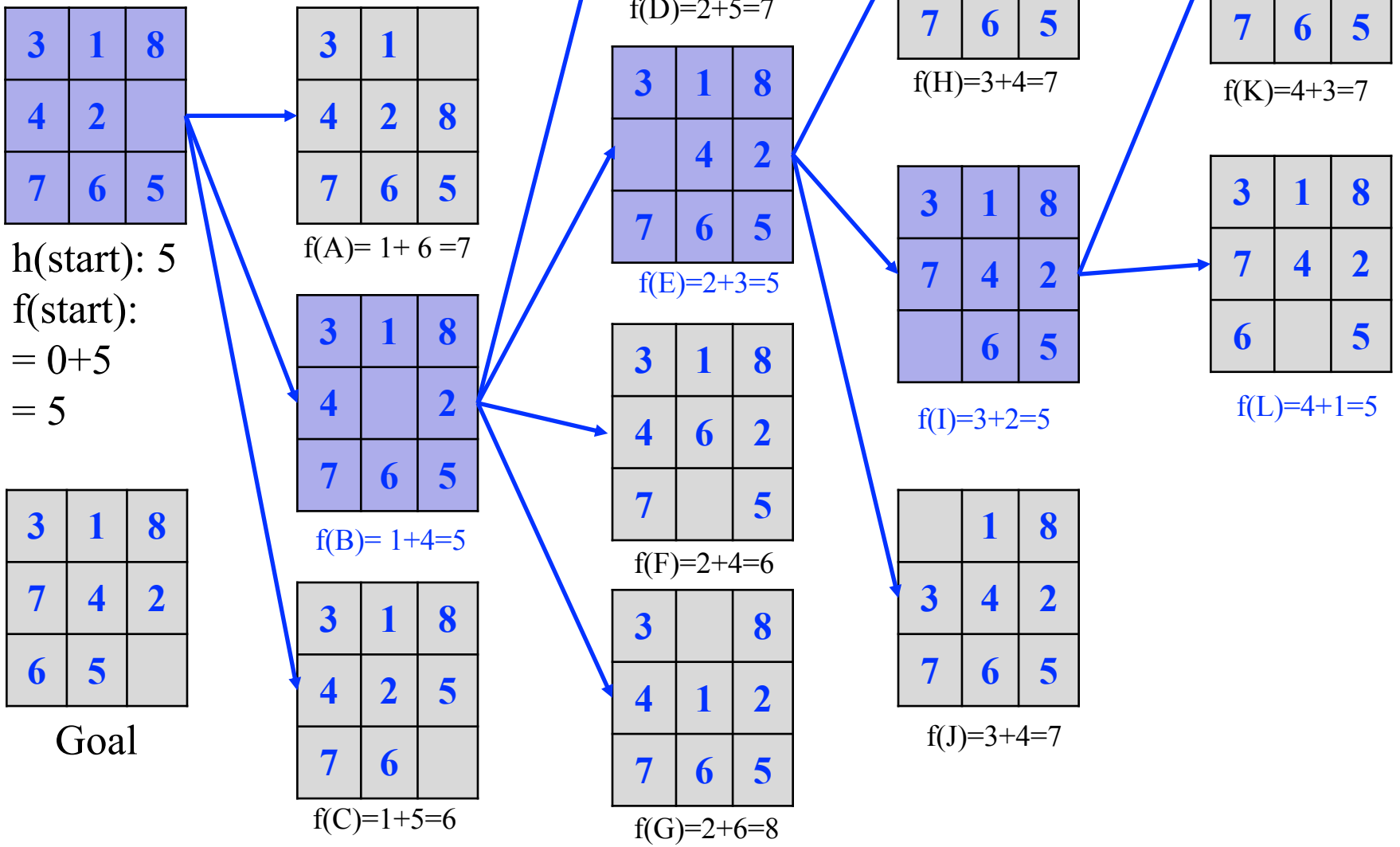
3	1	8
7	4	2
6	5	

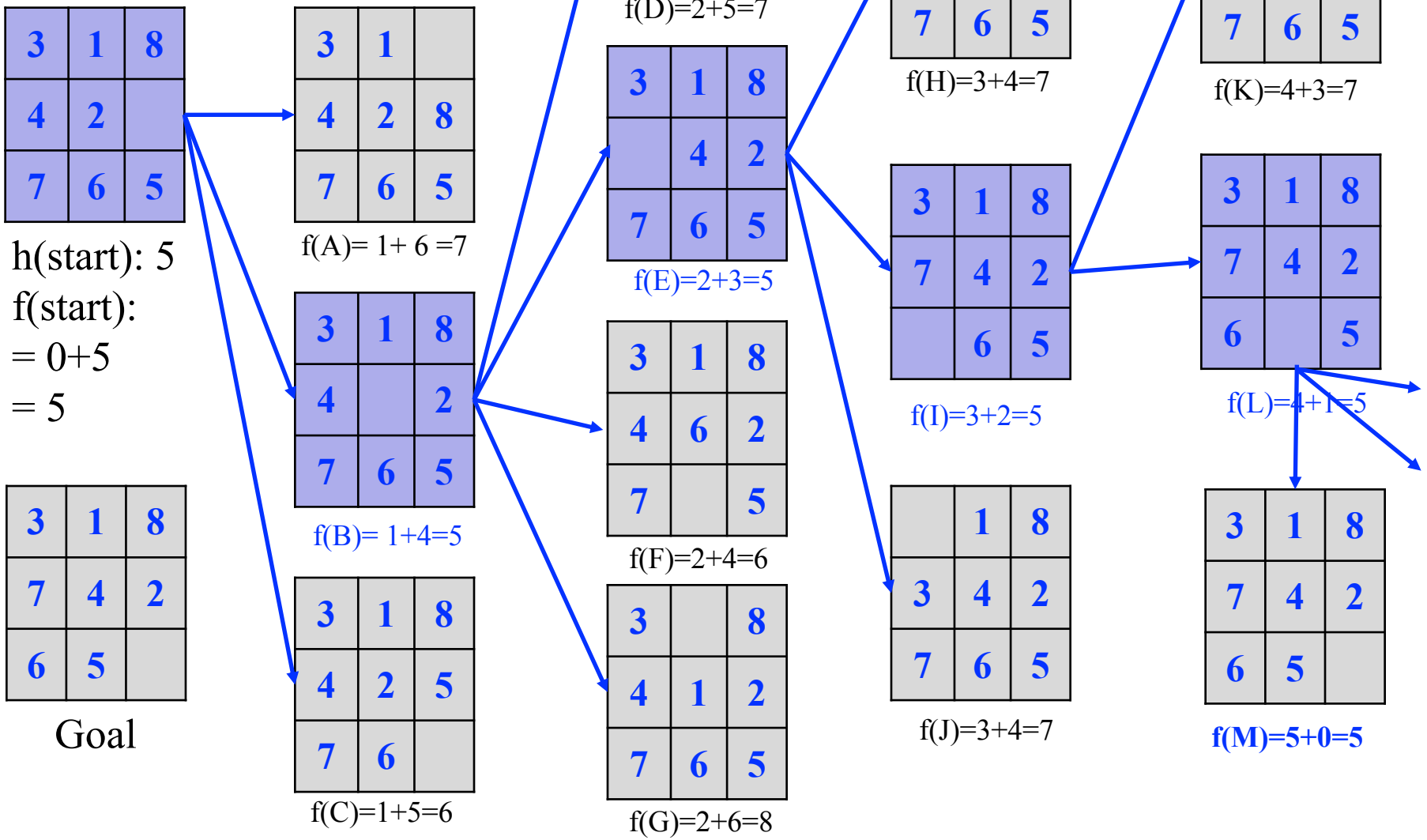
Goal

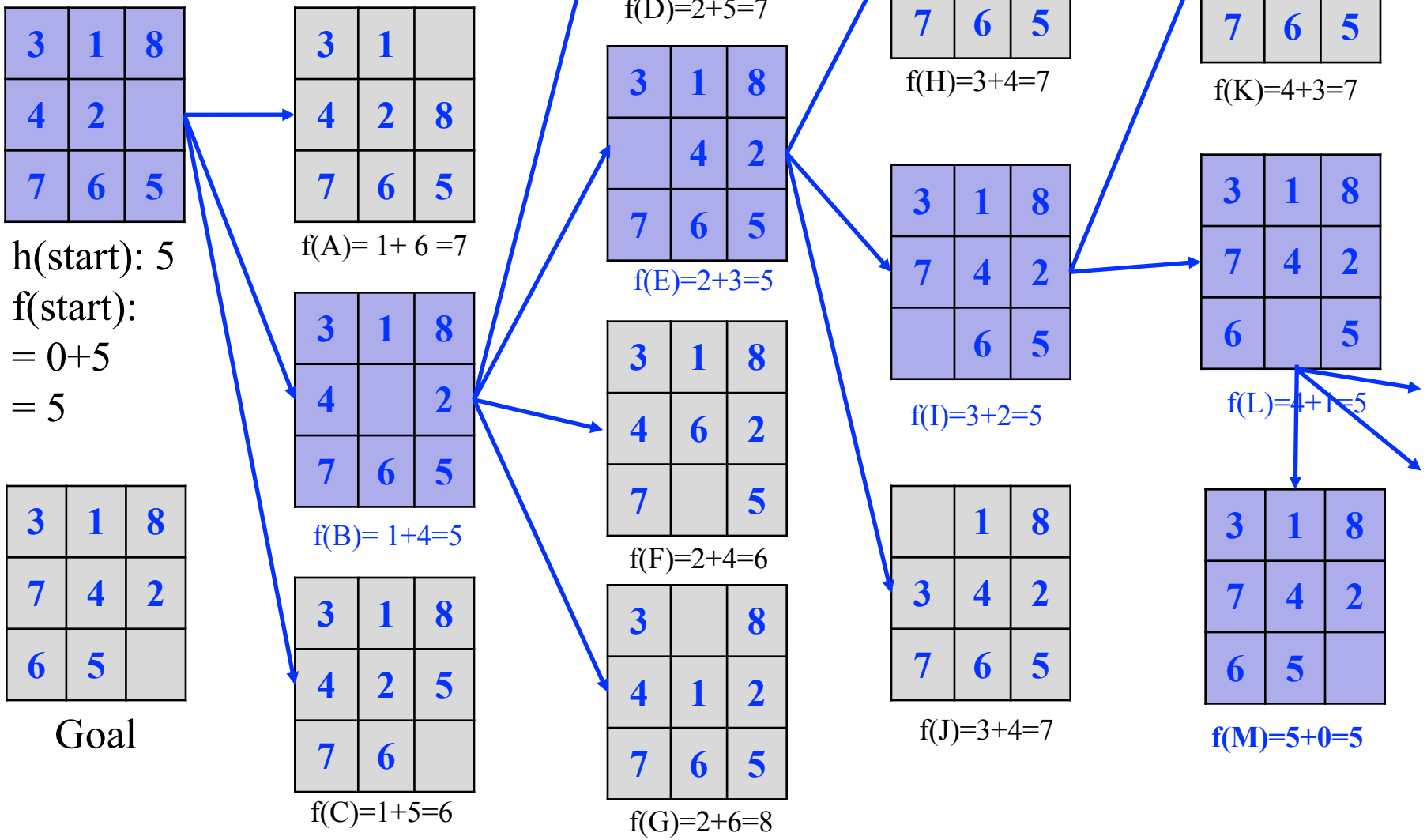

3 1 8
4 2
7 6 5
h(start): 5
f(start):
= 0+5
= 5
3 1 8
7 4 2
6 5
Goal
3 1
4 2 8
7 6 5
f(A)= 1+ 6 =7
3 1 8
4 2
7 6 5
f(B)= 1+4=5
3 1 8
4 2 5
7 6
f(C)=1+5=6


3 1 8
4 2
7 6 5
h(start): 5
f(start):
= 0+5
= 5
3 1 8
7 4 2
6 5
Goal
3 1
4 2 8
7 6 5
f(A)= 1+ 6 =7
3 1 8
4 2
7 6 5
f(B)= 1+4=5
3 1 8
4 2 5
7 6
f(C)=1+5=6
3 1 8
4 2
7 6 5
f(D)=2+5=7
3 1 8
4 2
7 6 5
f(E)=2+3=5
3 1 8
4 6 2
7 5
f(F)=2+4=6
3 8
4 1 2
7 6 5
f(G)=2+6=8


h(start): 5
f(start):
= 0+5
= 5
Goal
f(A)= 1+ 6 =7
f(B)= 1+4=5
f(C)=1+5=6
f(D)=2+5=7
f(E)=2+3=5
f(F)=2+4=6
f(G)=2+6=8
f(H)=3+4=7
f(I)=3+2=5
f(J)=3+4=7

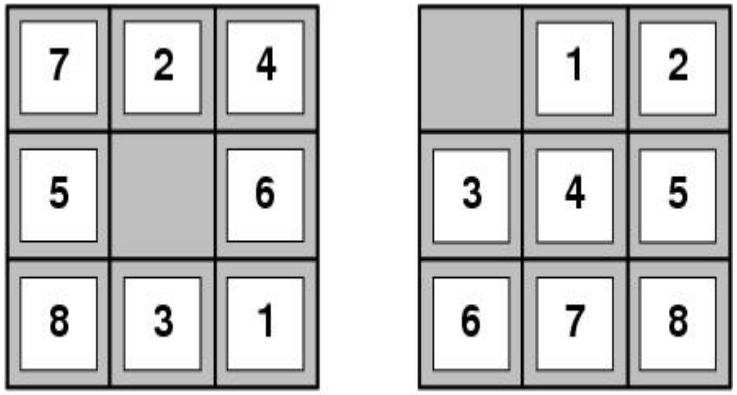

h(start): 5
f(start):
= 0+5
= 5
Goal
f(A)= 1+ 6 =7
f(B)= 1+4=5
f(C)=1+5=6
f(D)=2+5=7
f(E)=2+3=5
f(F)=2+4=6
f(G)=2+6=8
f(H)=3+4=7
f(I)=3+2=5
f(J)=3+4=7
f(K)=4+3=7
f(L)=4+1=5


h(start): 5
f(start):
= 0+5
= 5
Goal
f(A)= 1+ 6 =7
f(B)= 1+4=5
f(C)=1+5=6
f(D)=2+5=7
f(E)=2+3=5
f(F)=2+4=6
f(G)=2+6=8
f(H)=3+4=7
f(I)=3+2=5
f(J)=3+4=7
f(K)=4+3=7
f(L)=4+1=5
f(M)=5+0=5


h(start): 5
f(start):
= 0+5
= 5
Goal
f(A)= 1+ 6 =7
f(B)= 1+4=5
f(C)=1+5=6
f(D)=2+5=7
f(E)=2+3=5
f(F)=2+4=6
f(G)=2+6=8
f(H)=3+4=7
f(I)=3+2=5
f(J)=3+4=7
f(K)=4+3=7
f(L)=4+1=5
f(M)=5+0=5

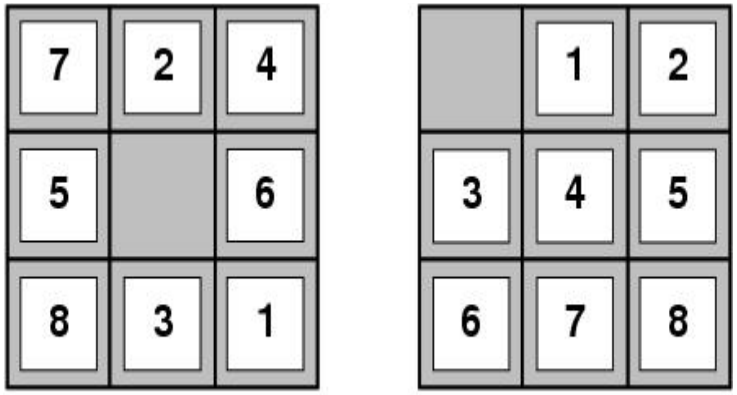
Heuristic functions

Heuristic functions



- E.g for the 8-puzzle
  - Avg. solution cost is about 22 steps (branching factor +/- 3)
  - Exhaustive search to depth 22: 3.1 x 10^{10} states.
  - A good heuristic function can reduce the search process.

Heuristic functions



- E.g for the 8-puzzle knows two commonly used heuristics
- h_1 = the number of misplaced tiles
  - $h_1(s)=8$
- h_2 = the sum of the distances of the tiles from their goal positions (manhattan distance).
  - $h_2(s)=3+1+2+2+2+3+3+2=18$
- h_1 or h_2 is better?

Heuristic quality

- Effective branching factor b*
  - Is the branching factor that a uniform tree of depth *d* would have in order to contain *N+1* nodes.

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

  - Measure is fairly constant for sufficiently hard problems.
    - ✓ Can thus provide a good guide to the heuristic's overall usefulness.
    - ✓ A good value of b* is 1.

Heuristic quality and dominance

- 1200 random problems with solution lengths from 2 to 24.

	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	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

- If $h_2(n) >= h_1(n)$ for all n (both admissible) then h_2 *dominates* h_1 and is better for search

Inventing admissible heuristics

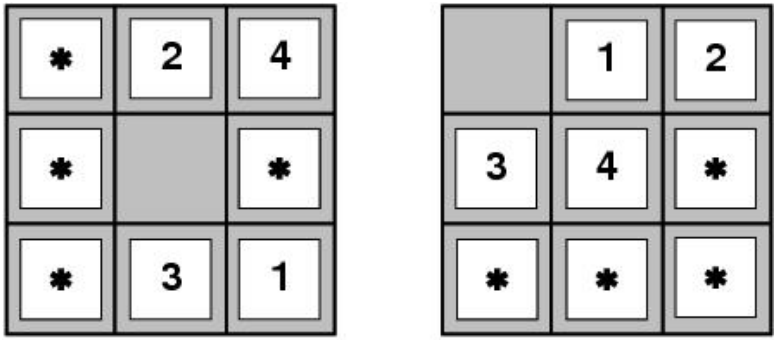
- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem:
  - Relaxed 8-puzzle for h_1 : a tile can move anywhere
    As a result, $h_1(n)$ gives the shortest solution
  - Relaxed 8-puzzle for h_2 : a tile can move to any adjacent square.
    As a result, $h_2(n)$ gives the shortest solution.

The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.

ABSolver found a usefull heuristic for the rubic cube.

Inventing admissible heuristics

- Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem.
- This cost is a lower bound on the cost of the real problem.
- Pattern databases store the exact solution to for every possible subproblem instance.
  - The complete heuristic is constructed using the patterns in the DB



Inventing admissible heuristics

- Another way to find an admissible heuristic is through learning from experience:
  - Experience = solving lots of 8-puzzles
  - An inductive learning algorithm can be used to predict costs for other states that arise during search.