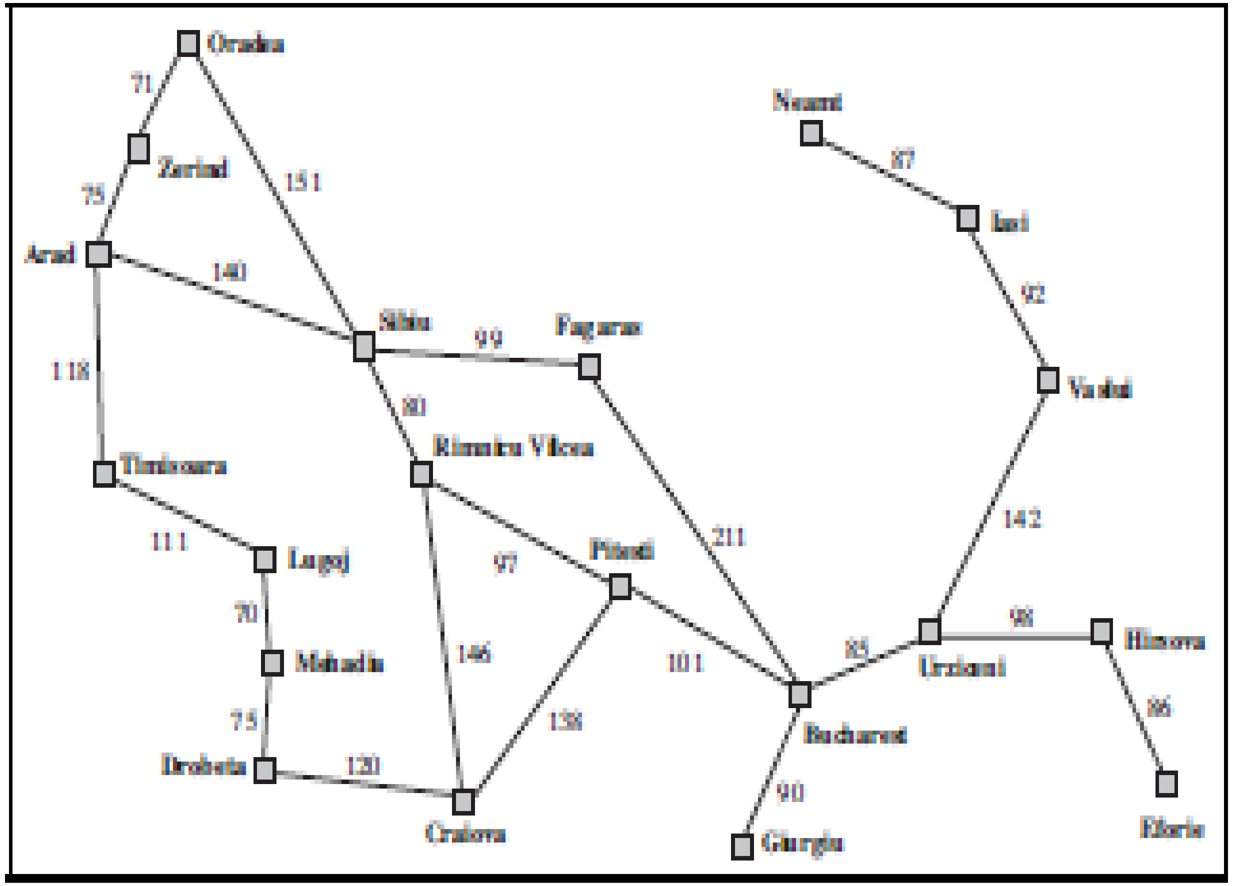
Example – Driving In Romania



Goal: Drive to Bucharest

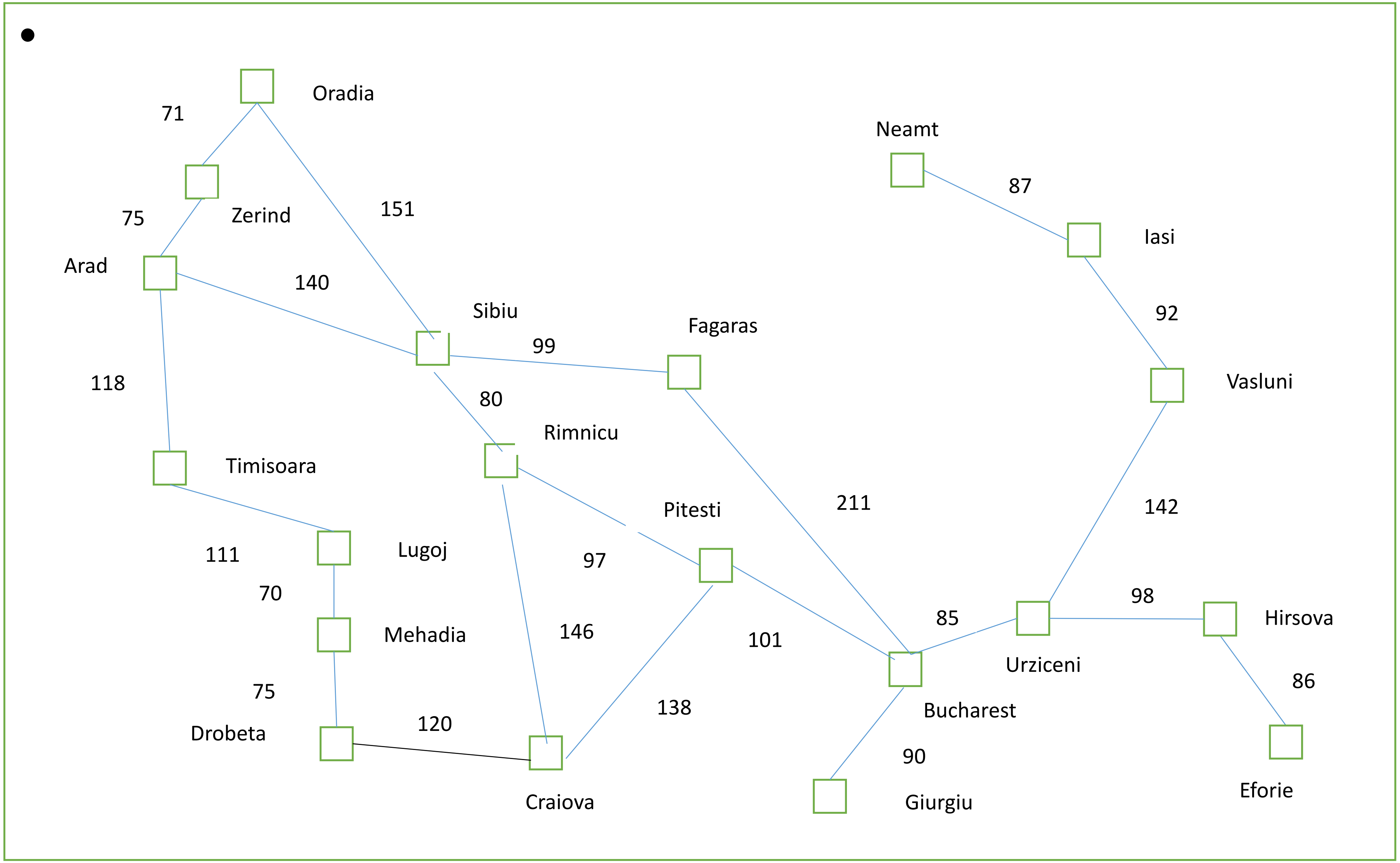
Current State: Arad

3 roads from Arad

Arad → Zerind
Arad → Sibiu
Arad → Timisoara

None achieve the goal!

Example – Driving In Romania



Tree and Graph Search Algorithms – handling repeated states

function TREE-SEARCH(problem) **returns** a solution, or failure

initialize the frontier using the initial state of problem

loop do

if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the corresponding solution

expand the chosen node, adding the resulting nodes to the frontier

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

initialize the frontier using the initial state of problem

initialize the explored set to be empty

loop do

if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the corresponding solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to the frontier - ***only if not in the frontier or explored set***

Types of searches

Uninformed search strategies / blind searches

strategies have no additional information about the states, beyond that provided in the problem definition

All they do is generate successors and distinguish a goal state from a non-goal state

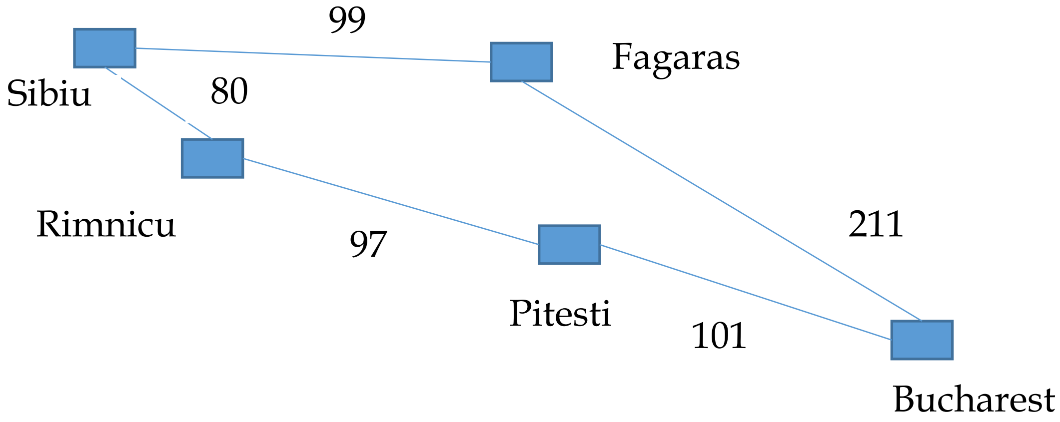
all search strategies are distinguished by the order in which the nodes are expanded

BFS, DFS, DLS, UCS,

Informed search strategies / heuristic search strategies

strategies that know whether one non-goal state is 'more promising' than another

Uniform Cost Search Approach

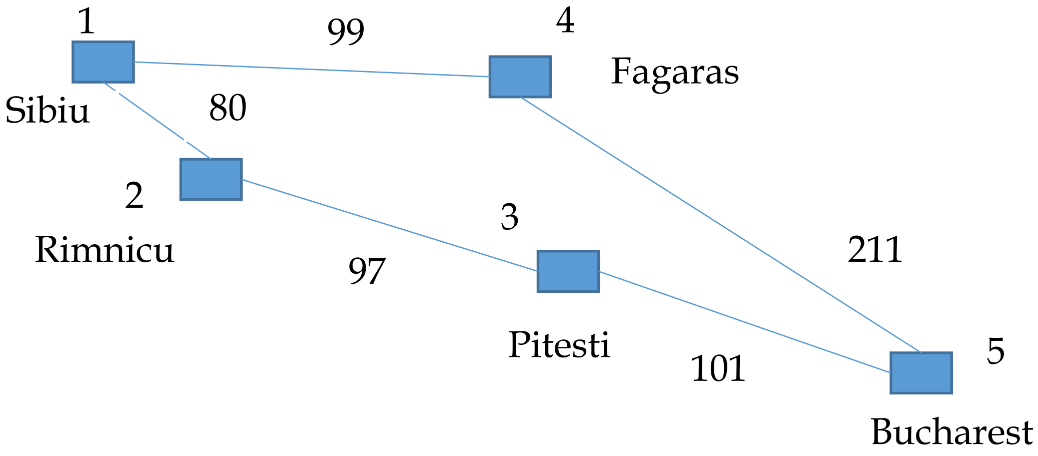


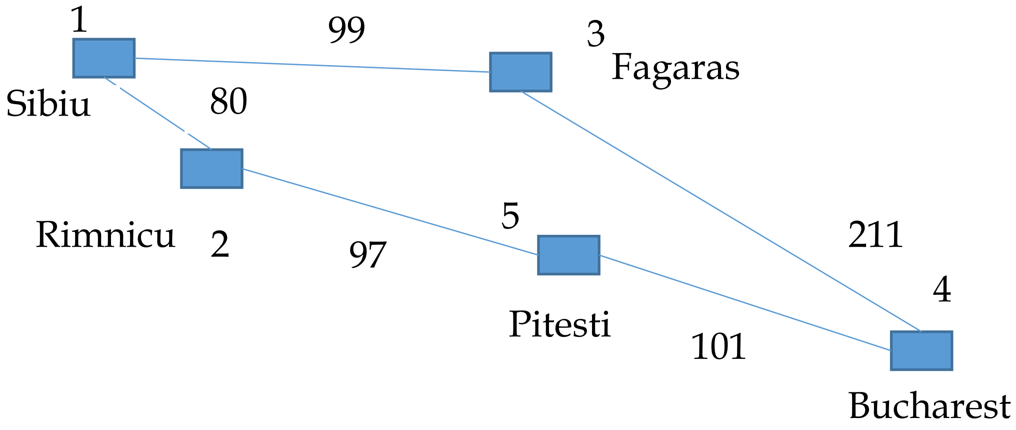
BFS	UCS
Frontier (R80, F99)	Frontier (R80, F99)
Frontier (F99, P97)	Frontier (R80 +P97, F99)
Frontier (P97, B211)	Frontier (**F99**, RP177)
Frontier (B211)	Frontier (**RP177**, F99+ **B211**)
	Frontier (**RP177+B101**, FB310)
	Frontier (**RPB278**, FB310)

UCS: goal node generated 1-2-3-4

adds a second path 1-2-5-4

checks if the second path is better and returns it





Uniform Cost Search Approach

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
    node ←a node with STATE = problem.INITIAL-STATE, PATH-COST = 0

    frontier ←a priority queue ordered by PATH-COST,
                    with node as the only element
    explored ←an empty set
    loop do
        if EMPTY?( frontier) then return failure
        node←POP( frontier ) /* chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return
        SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ←CHILD-NODE(problem, node, action)
            if child .STATE is not in explored or frontier then
            frontier ←INSERT(child , frontier )
            else if child .STATE is in frontier with higher PATH-COST
            then
            replace that frontier node with child
```

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
        node ←a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        frontier ←a FIFO queue with node as the only element
        explored ←an empty set
    loop do
        if EMPTY?( frontier) then return failure
        node←POP( frontier ) /* chooses the shallowest node in frontier */

            add node.STATE to explored
            for each action in problem.ACTIONS(node.STATE) do
                child ←CHILD-NODE(problem, node, action)
                if child .STATE is not in explored or frontier then
                if problem.GOAL-TEST(child .STATE) then
                return SOLUTION(child )
                frontier ←INSERT(child , frontier )
```

Comparing uninformed search strategies (using tree search)

Criterion	BFS	UCS	DFS	DLS	IDS	BDS
Complete?	Yes (if b is finite)	Yes (if b is finite, step costs >= ε for positive ε	No (yes if graph search is used and state space is finite)	No	Yes (if b is finite)	Yes (if b is finite, if both directions use BFS)
Time	$O(b^d)$	$O(b^{1+C^*/e})$	$O(b^m)$ (for graph search, bound by size of state space)	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+C^*/e})$	O(bm) (for graph search, bound by size of state space)	O(bl)	(O(bd)	$O(b^{d/2})$
Optimal?	Yes (If step costs are all identical)	Yes	No	No	Yes (If step costs are all identical)	Yes (If step costs are all identical, if both directions use BFS)

Informed/Heuristic Search strategies: Best First Search

Uses problem specific knowledge, beyond the problem definition itself

more efficient than uninformed search strategy

Best First Search - BeFs

instance of tree search/Graph Search

A node is selected for expansion based on an evaluation function, f(n)

f(n), a cost estimate - used to order priority queue (similar to UCS, g(n))

Choice of f(n) – determines search strategy

f(n) = h(n) + … ;

h(n) heuristic function – estimated cost of the cheapest path from the node n to a goal node

h(n) depends only on the state at that node and not on path cost

Ex: Map of Romania:

h(n), cost estimate from Arad to Bucharest – straight line distance from Arad to Bucharest

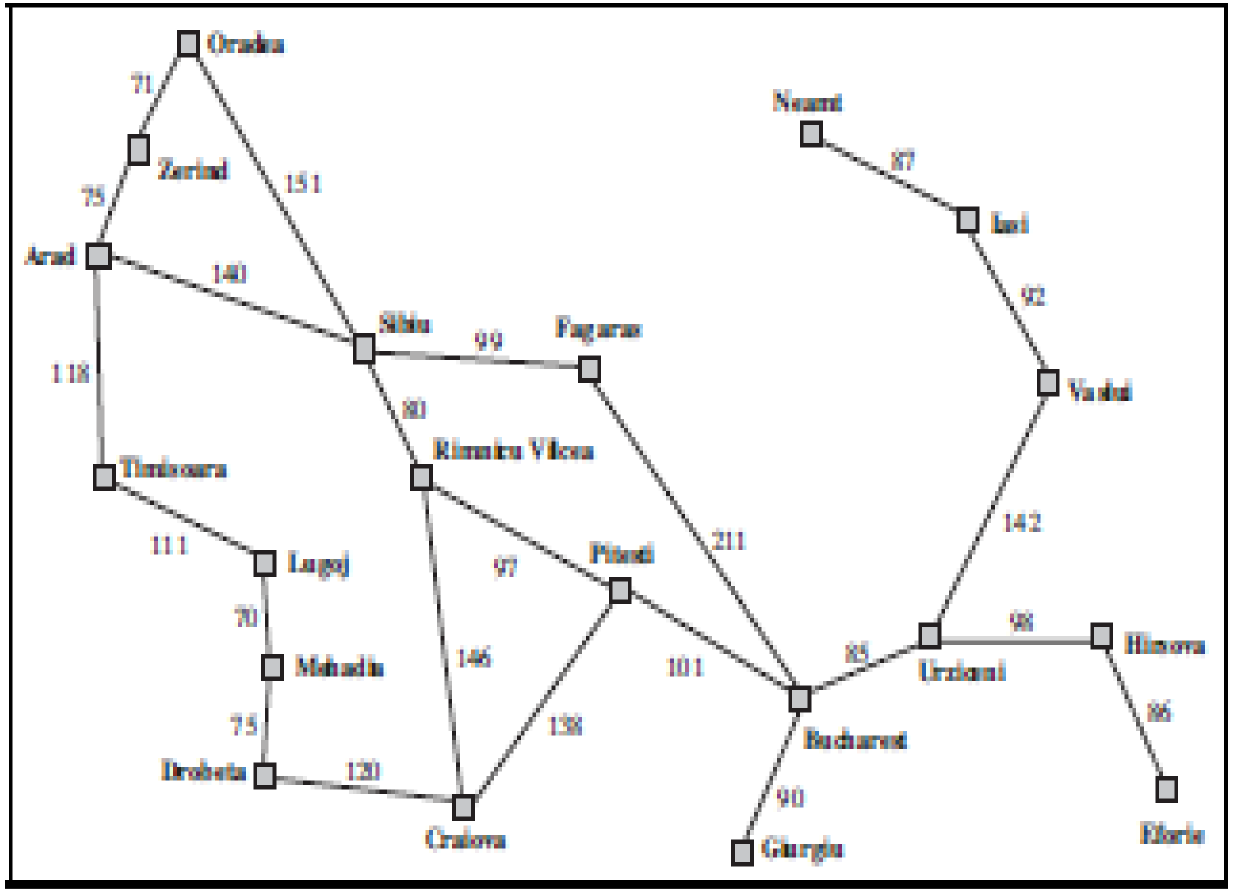
h(n) = 0 at goal node

Informed/Heuristic Search strategies: Best First Search

Greedy Best First Search

expand the node that is closest to the goal (likely to reach the goal quickly)

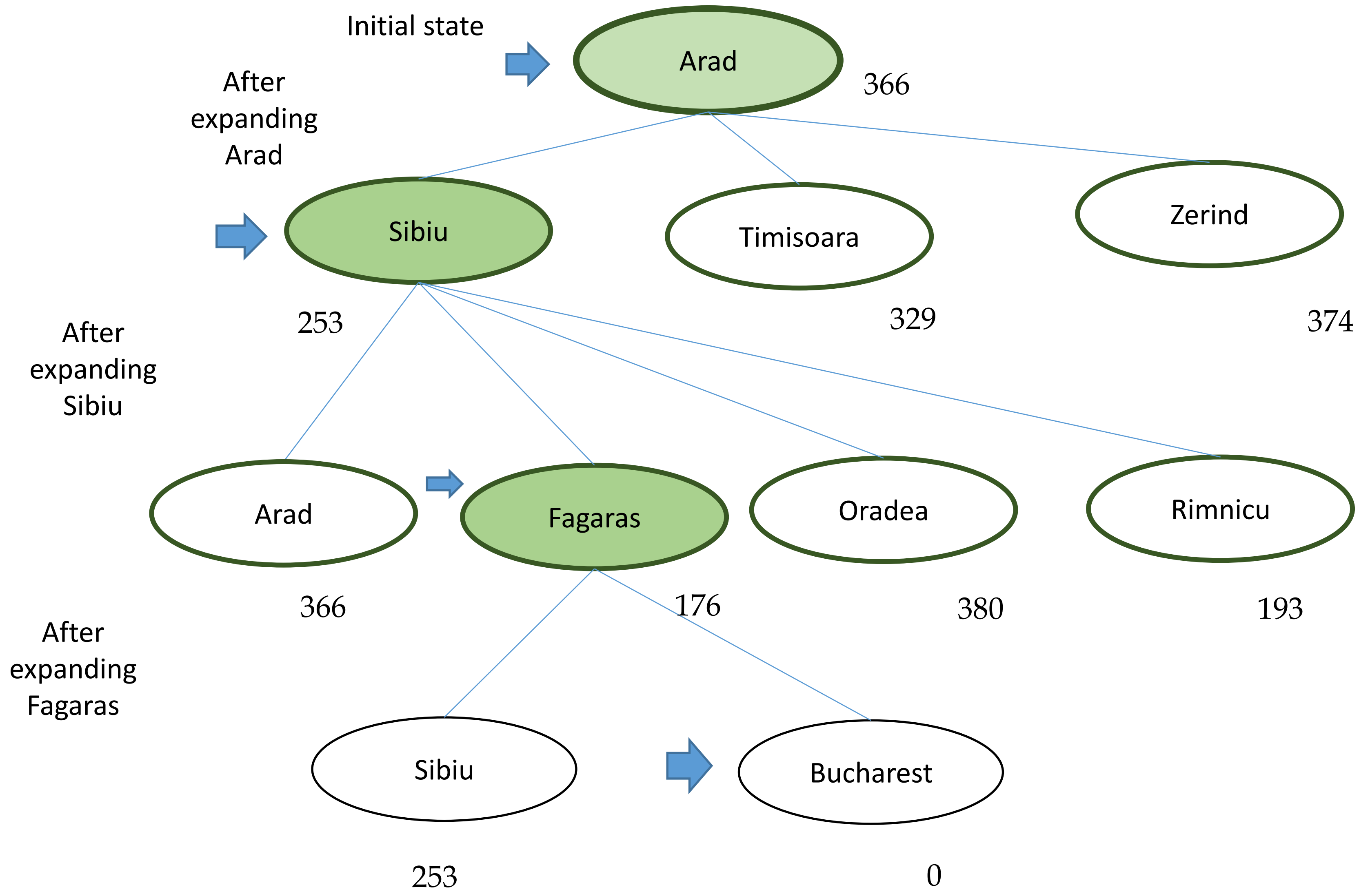
Ex: route finding in Romania: values of h_{sld} to Bucharest



city	hsld to Bucharest	city	hsld to Bucharest
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Droheta	242	Pitesti	100
Eforie	161	Rimnicu	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Lasi	226	Urziceni	80
Lugoj	244	Vaslui	199
		Zerind	374

Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

•



Informed/Heuristic Search strategies: Best First Search

Not optimal – Arad – Sibiu – Fagaras –Bucharest – longer by 32 km (than Arad – Sibiu – Rimnicu – Pitesti – Bucharst)

Search cost is minimal in this case, as no node that is not on the solution path is expanded.

At each step it tries to get as close to it's goal as it can. - greedy approach

GBeFS incomplete even in finite state space, much like DFS

Ex: Iasi to Fagaras - if tree search is used, gets in to infinite loop,

Graph search leads to solution in finite spaces – complete

Time, Space complexity in worst case - $O(b^m)$ m – maximum depth of state space

Informed/Heuristic Search strategies: A* search

Minimize the total estimated solution cost – (a form of BeFS)

$f(n) = g(n) + h(n)$ - estimated cost of the cheapest solution through node n with lowest $f(n)$

$g(n)$ – cost to reach the node - computed from step costs

$h(n)$ - estimate of the cost to get from the node to the goal

A* identical to UCS

A* search – complete, optimal, provided $h(n)$ satisfies certain conditions:

Conditions for optimality: Admissibility, Consistency

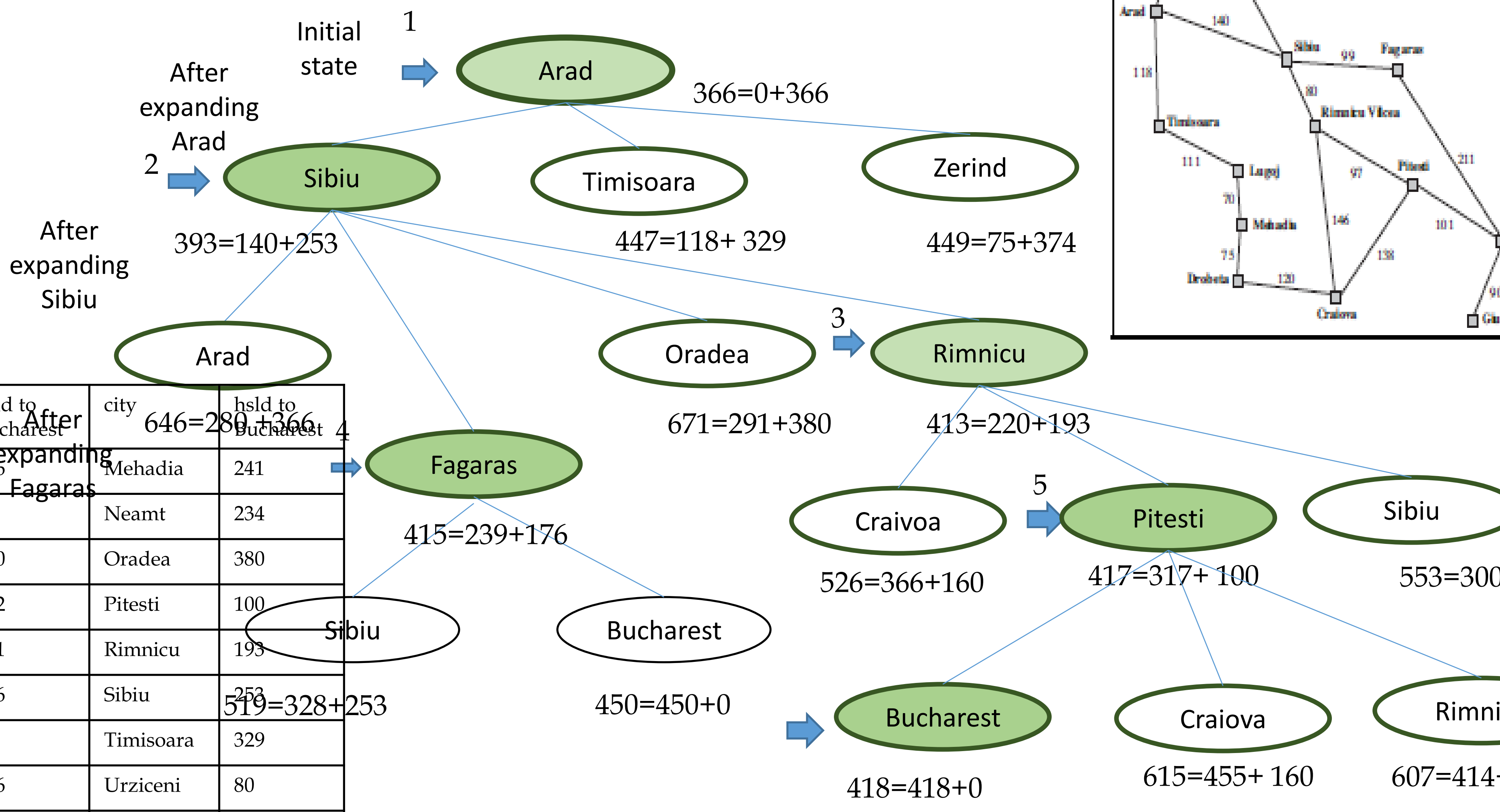
$h(n)$ is admissible if it does not over estimate the cost to reach a goal

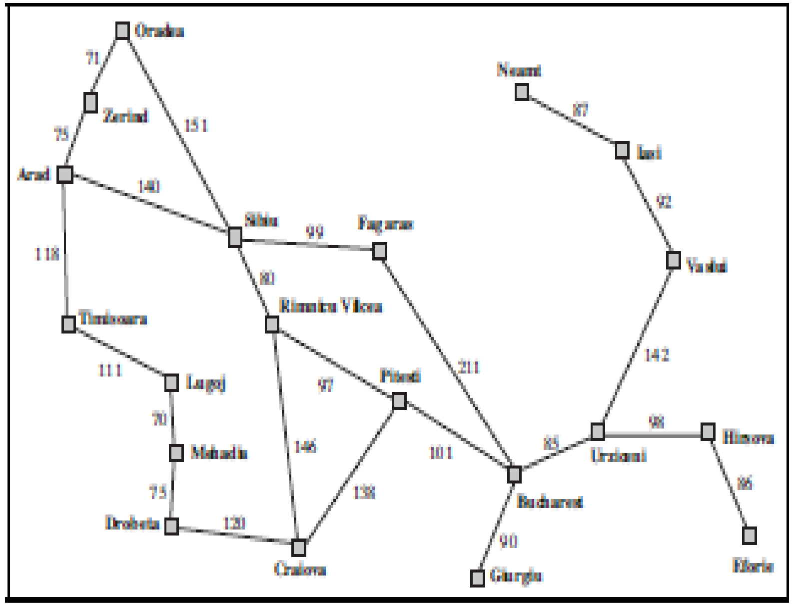
Admissible heuristics are optimistic as g(n) is the actual cost and h(n) is admissible and does not over estimate the cost, hence f(n) never overestimates the true cost of a solution along the current path through the node n.

Ex: h_{sld} in Romania map : SLD – is shortest path between two cities/points and can not be an over estimate

Stages in A* search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with $f(n) = g(n) + h(n)$.

•





city	hsld to Bucharest	city	hsld to Bucharest
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Droheta	242	Pitesti	100
Eforie	161	Rimnicu	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Lasi	226	Urziceni	80
Lugoj	244	Vaslui	199
		Zerind	374

Informed/Heuristic Search strategies: A * Search

Step 0 : Frontier{Arad(366)}

Explored { }

Step 1: Frontier{Sibiu(393), Timisoara(447), Zerind(449)}

Explored { Arad(366)}

Step 2: Frontier{Rimnicu(413), Fagaras(415), Timisoara(447), Zerind(449), Arad(646), Oradea(671)}

Explored { Arad(366), Sibiu(393)}

Step 3: Frontier{Fagaras(415), Pitesti(417), Timisoara(447), Zerind(449), Craiova(526), Sibiu(553), Arad(646), Oradea(671)}

Explored { Arad(366), Sibiu(393), Rimnicu(413) }

Step 4: Frontier{Pitesti(417), Timisoara(447), Zerind(449), **Bucharest(450),** Craiova(526), Sibiu(519), Arad(646), Oradea(671)}

Explored { Arad(366), Sibiu(393), Rimnicu(413), Fagaras(415) }

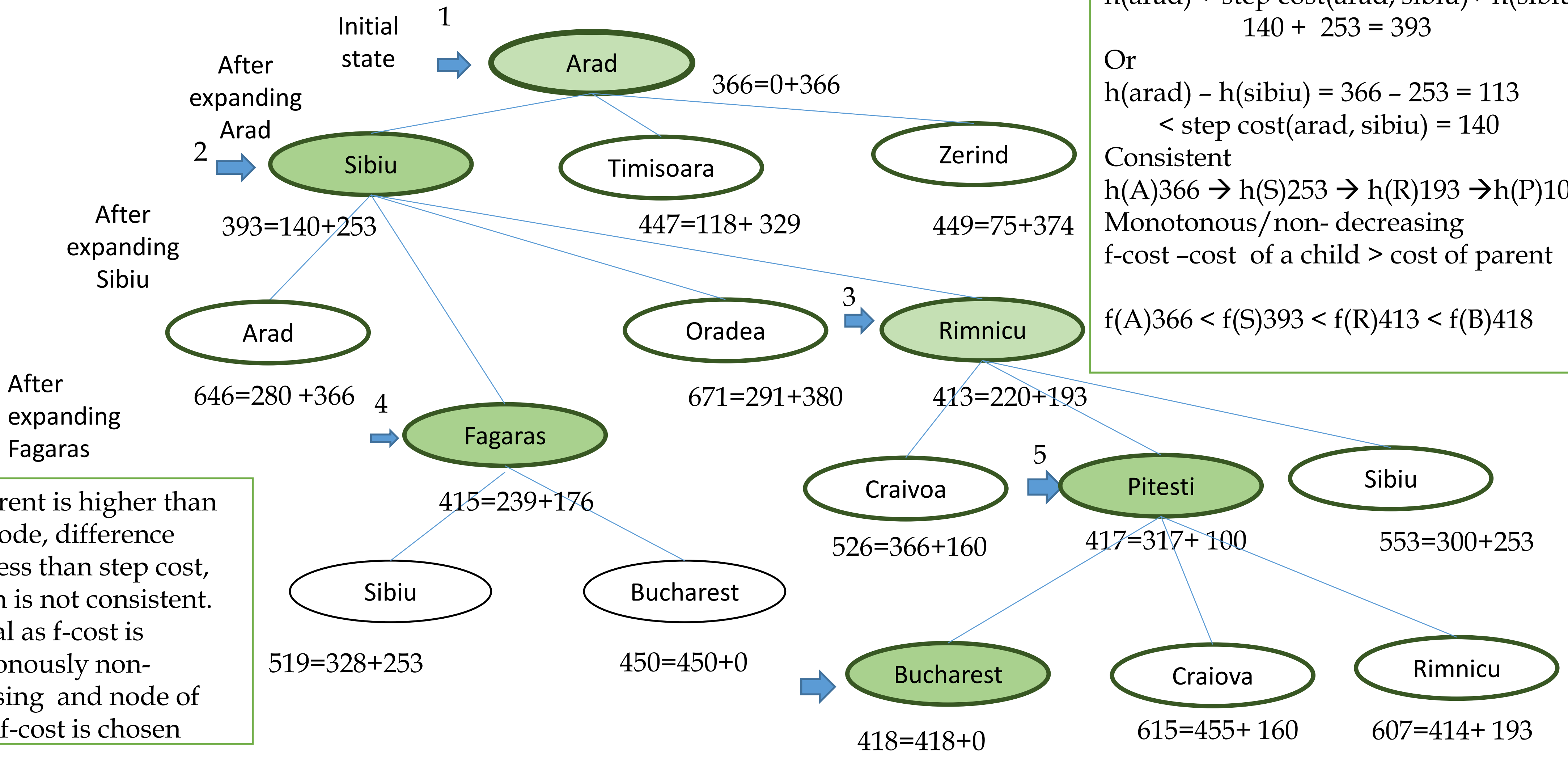
Step 5: Frontier{**Bucharest(418),** Timisoara(447), Zerind(449), Craiova(526), Sibiu(519), Rimnicu(607), Arad(646), Oradea(671)}

Explored { Arad(366), Sibiu(393), Rimnicu(413), Fagaras(415), Pitesti(417) }

Step 6: Frontier{Timisoara(447), Zerind(449), Sibiu(519), Craiova(526), Rimnicu(607), Arad(646), Oradea(671)}

Explored { Arad(366), Sibiu(393), Rimnicu(413), Fagaras(415), Pitesti(417), **Bucharest(418)**}

Stages in A* search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with $f(n) = g(n) + h(n)$.



f(A) = g(A)+ h(A) = 0+366 < solution cost = path cost+ h(B) = 418+0

- admissible

h(arad)< step cost(arad, sibiu)+ h(sibiu)

140 + 253 = 393

Or

h(arad) – h(sibiu) = 366 – 253 = 113

< step cost(arad, sibiu) = 140

Consistent

h(A)366 → h(S)253 → h(R)193 →h(P)100 →h(B)0

Monotonous/non- decreasing

f-cost –cost of a child > cost of parent

f(A)366 < f(S)393 < f(R)413 < f(B)418

h of parent is higher than child node, difference being less than step cost, if not, h is not consistent. Optimal as f-cost is monotonously non-decreasing and node of lowest f-cost is chosen

h(A)366 - h(S)253 < C(A,S)140; → h(S)253 - h(R)193 < C(S,R)80 ; → h(R)193 - h(P)100 <C(R,P)97; → h(P)100 - h(B)0 <C(P,B)101

Informed/Heuristic Search strategies: A* Search

Goal Node, Bucharest first appears on the frontier at step 4 ,

but it is not selected for expansion (goal test done at expansion, not at generation)

As its cost 450 is higher than that of Pitesti (417),

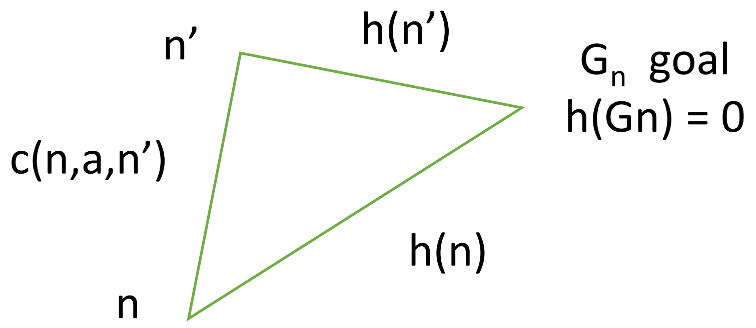
there might be a solution through Pitesti whose cost is as low as 417,

the algorithm does not settle for a solution that costs 450

2. Consistency (monotonicity)

h(n) is consistent, if $h(n) \leq c(n.a.n') + h(n')$

(general triangular inequality: each side of the triangle can not be longer than the sum of the other two sides)



H(n) is consistent if, for every node n and every successor n′ of n generated by an action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n′ plus the estimated cost of reaching the goal from n′

For an admissible heuristic, if there were a route from n to Gn via n′ that was cheaper than h(n), that would violate the property that h(n) is a lower bound on the cost to reach Gn.

Hence, Every consistent heuristic is also admissible. Consistency is a stricter requirement

Informed/Heuristic Search strategies: Best First Search

Consistent heuristics are called monotone because, the estimated final cost of a partial solution is monotonically non decreasing along the best path to the goal

Ex: h_{sld} is a consistent heuristic as

$h(n) \leq c(n.a.n') + h(n')$ (since SLD between n and n′ ≤ $c(n,a, n')$)

Optimality of A*

A* is optimal with tree search if h(n) is admissible; A* is optimal with graph search if h(n) is consistent

Proof:

1. if h(n) is consistent, then the values of f(n) along any path are non-decreasing. (from definition of consistency)

Suppose n′ is a successor of n, then

$g(n') = g(n) + c(n,a,n')$ for some action a

And $f(n') = g(n') + h(n')$

$= g(n) + c(n,a,n') + h(n') \geq g(n) + h(n) = f(n)$

Therefore $f(n') \geq f(n)$

If $f(n) = g(n) + W\, h(n)$ where $W>1$ non monotonic

Informed/Heuristic Search strategies: A* search

2. Whenever A* selects a node n for expansion , the optimal path to that node has been found.

Were this not the case,

there would have to be another node, n′ in frontier on the optimal path from the start node to n.

Because f is non-decreasing along any path, n′ would have lower cost than n and would have been selected first

From 1 and 2, it follows that sequence of nodes expanded by A* using graph search is in non-decreasing order of f(n).

The first goal node selected for expansion must be an optimal solution,

because f is the true cost for goal nodes (h(Gn) = 0) and all later goals will be at least as expensive

Contours of f costs - if C* is the cost of the optimal solution path

1. A* expands all nodes with f(n) < C*
2. A* might then expand some of the nodes right on the goal contour (f(n) = C*), before selecting a goal node

Informed/Heuristic Search strategies: A*

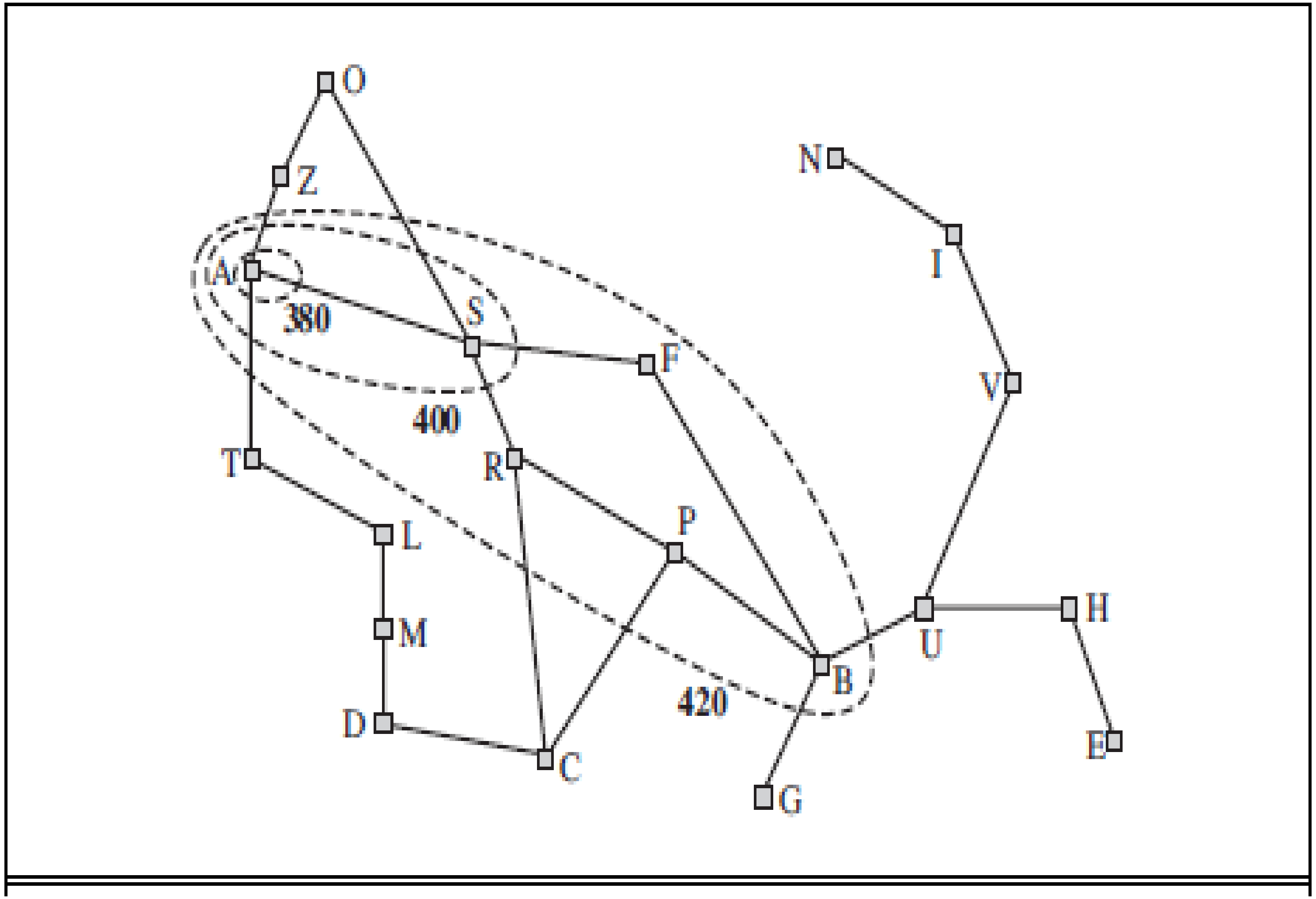
Map of Romania showing contours

at f = 380, f = 400, and f = 420,

with Arad as the start state.

Nodes inside a given contour have f-costs

less than or equal to the contour value.



Informed/Heuristic Search strategies: A* performance

Completeness

number of nodes with cost < C * needs to be finite.

True if all step costs > ε and b is finite

A* expands no nodes with f(n) > C*

ex: Timisoara is not expanded (sub tree of Timisoara is pruned)

Optimality

Because h_{sld} is admissible, Timisoara can be pruned.

The algorithm can safely ignore this subtree while still generating optimality

A* is optimally efficient for any given consistent heuristic

no other optimal algorithm is guaranteed to expand fewer nodes than A*

A* is complete, Optimal, Optimally efficient

Informed/Heuristic Search strategies: A* performance

Complexity

number of nodes within the goal contour

search space is exponential to the length of solution

absolute error: $\Delta = h^* - h$: h^* - actual cost of getting from initial state to goal state, h – heuristic

relative error: $\varepsilon = (h^* - h)/h^* = \Delta / h^*$

Time complexity

with single goal : $O(b^{\Delta}) \sim O(b^{\varepsilon d}) \sim O(b^{\varepsilon})^d$

b^{ε} is effective branching factor ;

Space Complexity

A* (graph search) keeps all nodes generated in memory

Informed/Heuristic Search strategies : Memory bounded – IDA*

IDA* - Iterative deepening A * for reducing memory requirements

Cutoff ? length?

f - cost : (g+ h)

At each iteration, cutoff value used is

smallest f cost of any node that exceeds the cutoff on the previous iteration

IDA* avoids substantial overhead associated with keeping a sorted queue of nodes

Retains only the current f-cost limit between iterations

may end up re-expanding the same states many times over

Informed/Heuristic Search strategies : Memory bounded – IDA*

Memory limitation of A* is addressed by iterative deepening (ID)

ID performs a series of depth-first searches, pruning branches when their cost exceeds a threshold for that iteration.

Initial threshold is the cost of the root node

Threshold for each succeeding iteration is the minimum node cost that exceeded the previous threshold

Informed/Heuristic Search strategies: Recursive Best First Search

Mimics Best First Search

Similar to recursive DFS

> instead of continuing indefinitely down the current path, uses f-limit variable to keep track of the f-value of the best alternative path available from any ancestor of the current node.
>
> if the current node exceeds this limit, recursion unwinds back to the alternative path
>
> as the recursion unwinds, f-value of each node along the path is replaced with a backed up value – the best f-value of its child nodes.

RBFS remembers f-value of the best leaf in the forgotten subtree and can decide whether it is worth re-expanding the subtree at some later time

Start :	Arad → Sibiu → Rimnicu
Back track:	Sibiu → Fagaras →Bucharest
Re-expand:	Rimnicu → Pitesti → Bucharest

Stages in A* search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with f(n) = g(n) + h(n).

•

Initial state 1 → Arad
366=0+366

After expanding Arad 2 → Sibiu | Timisoara | Zeri
393=140+253 | 447=118+ 329 | 449=

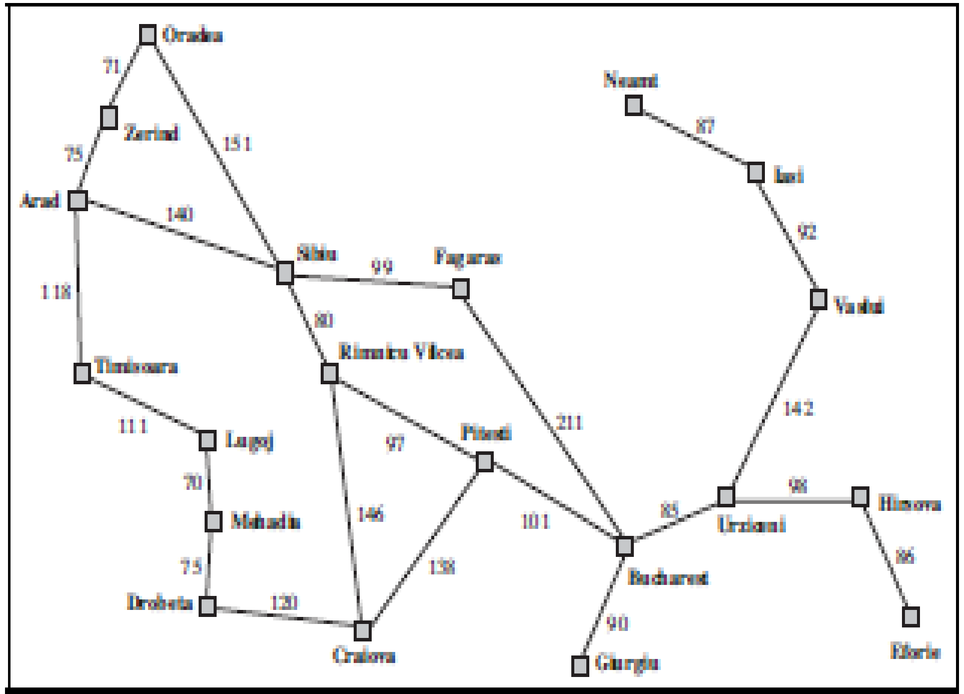
After expanding Sibiu: Arad | 4 → Fagaras | Oradea | 3 → Rim
646=280+366 | 415=239+176 | 671=291+380 | 413=220+193

After expanding Fagaras: Sibiu | Bucharest
519=328+253 | 450=450+0

Craivoa | 5 → Pitesti | Sibiu
526=366+160 | 417=317+ 100 | 553=300+253

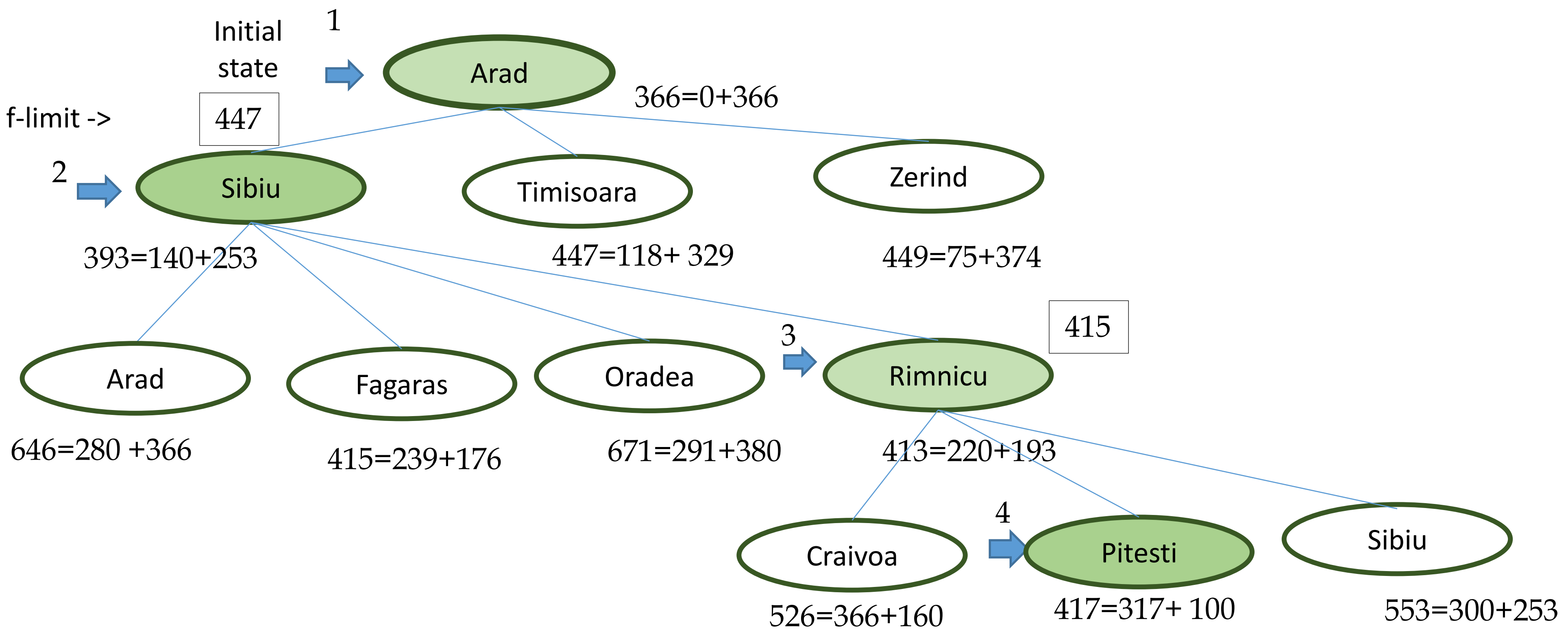
→ Bucharest | Craiova | Rimnicu
418=418+0 | 615=455+ 160 | 607=414+ 193

city	hsld to Bucharest	city	hsld to Bucharest
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Droheta	242	Pitesti	100
Eforie	161	Rimnicu	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Lasi	226	Urziceni	80
Lugoj	244	Vaslui	199
		Zerind	374



Stages in RBFS search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with f(n) = g(n) + h(n).

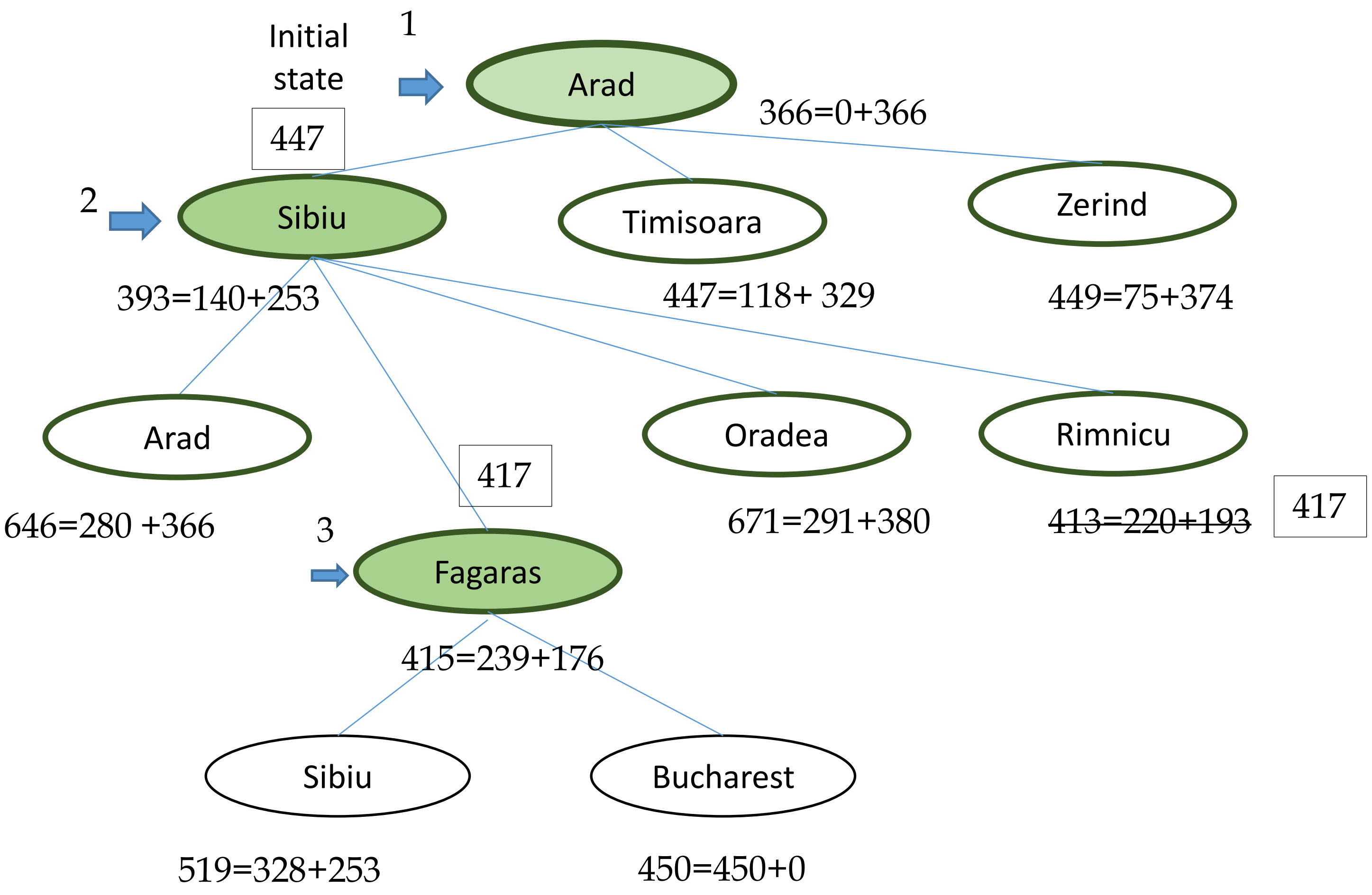
•



After expanding Arad, Sibiu, Rimnicu

Stages in RBFS search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with f(n) = g(n) + h(n).

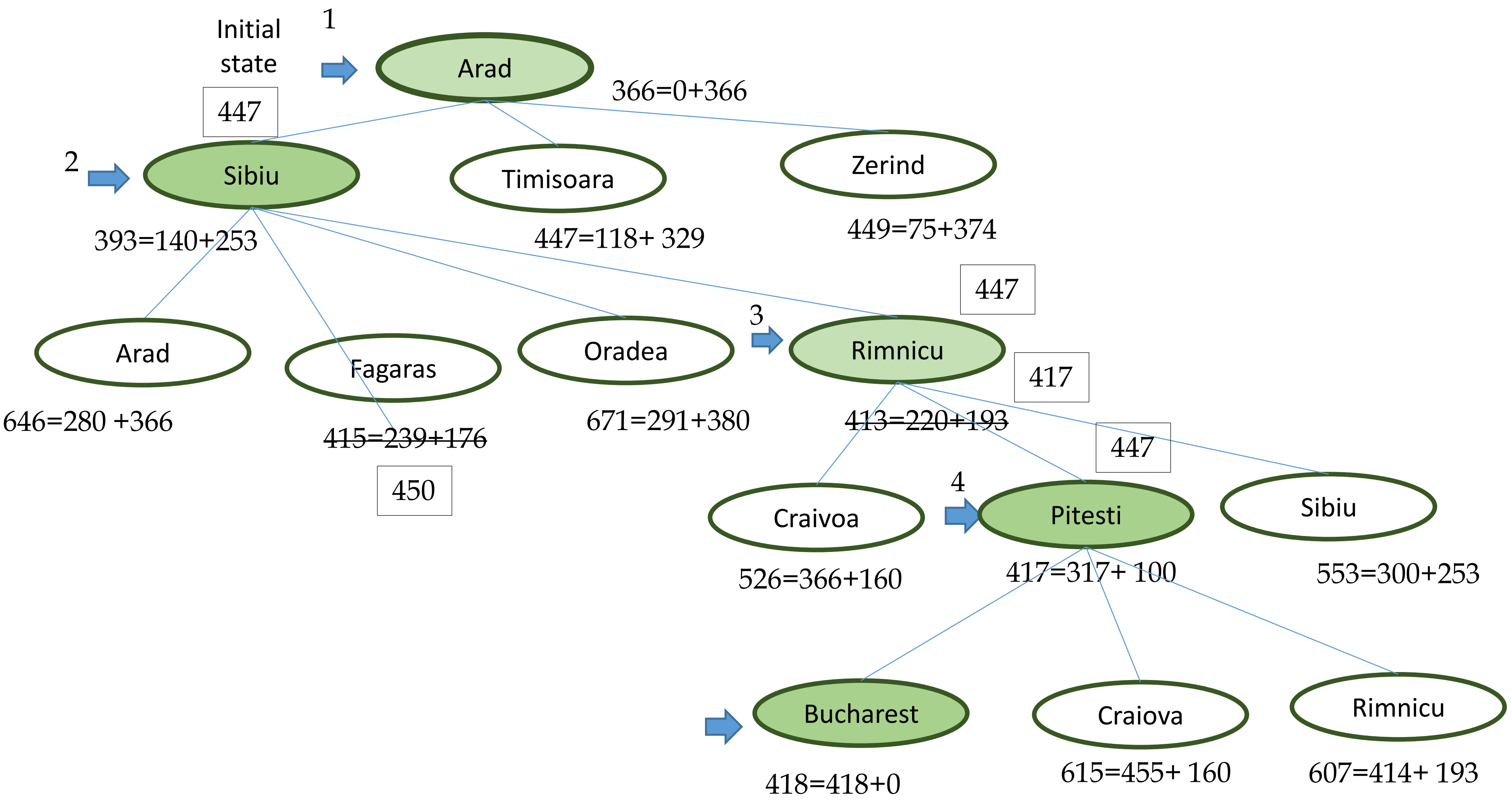
•



After unwinding back to Sibiu and expanding Fagaras

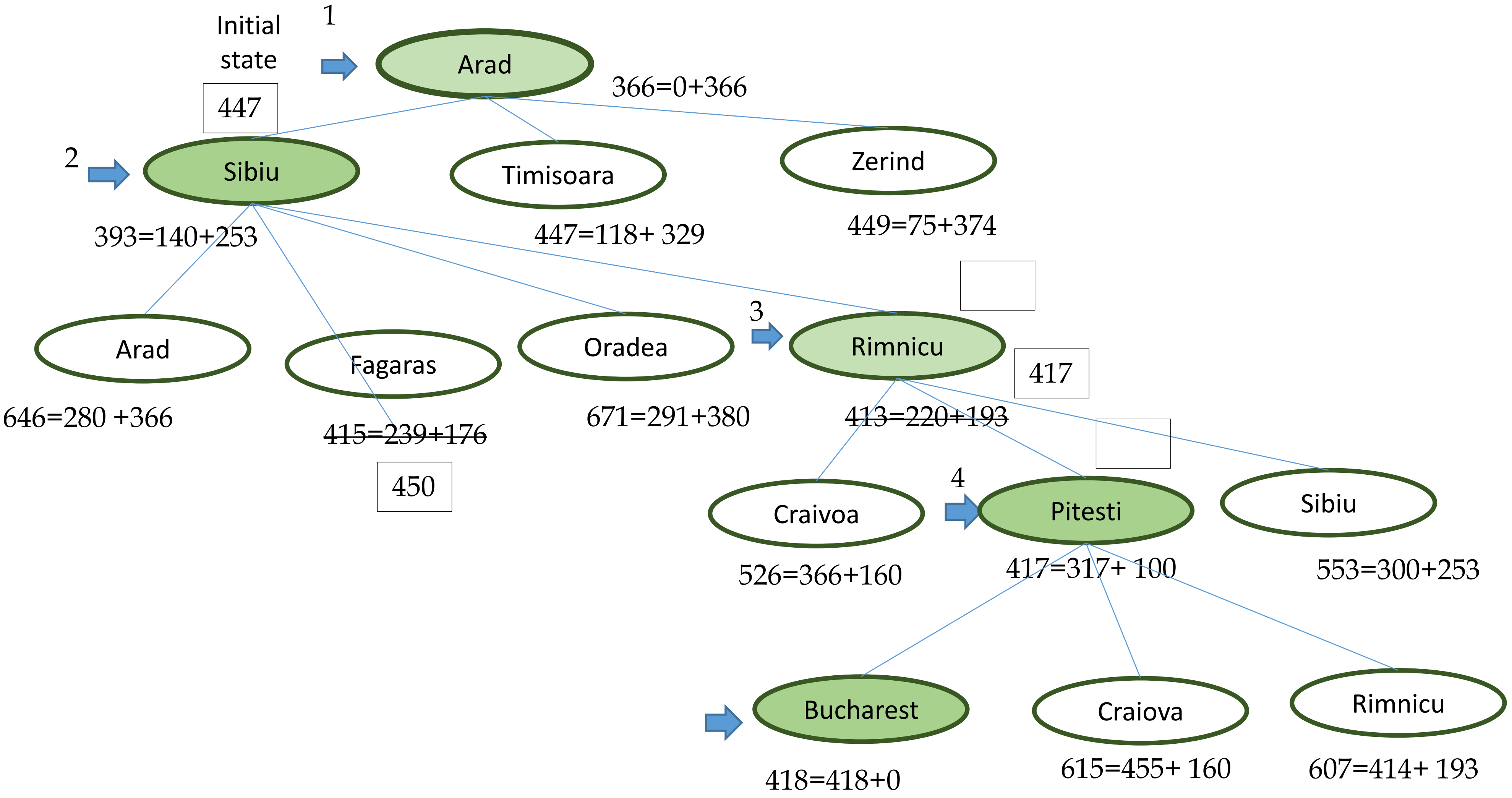
Stages in RBFS search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with f(n) = g(n) + h(n).

•



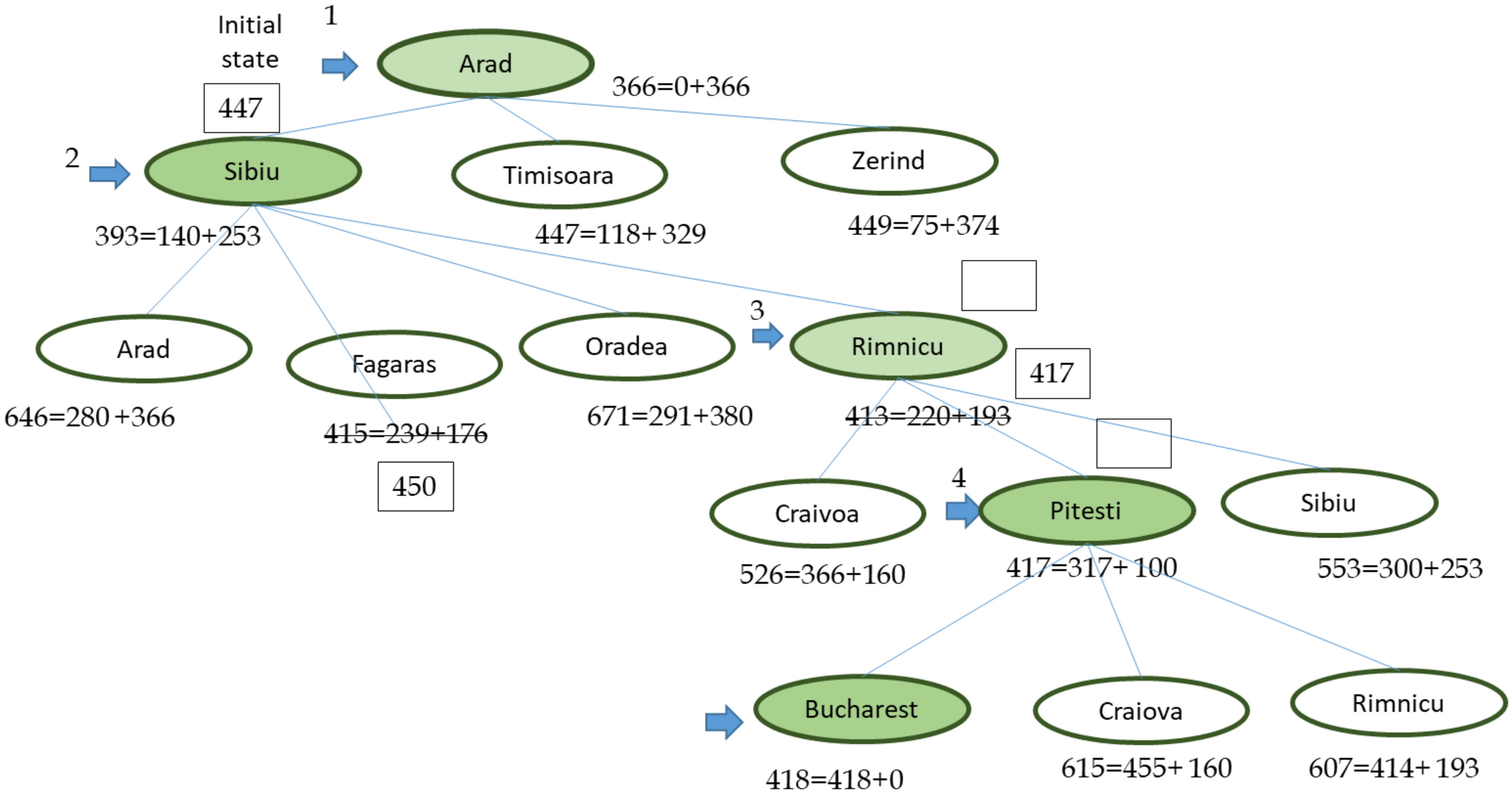
After switching back to Rimnicu and expanding Pitesti

•



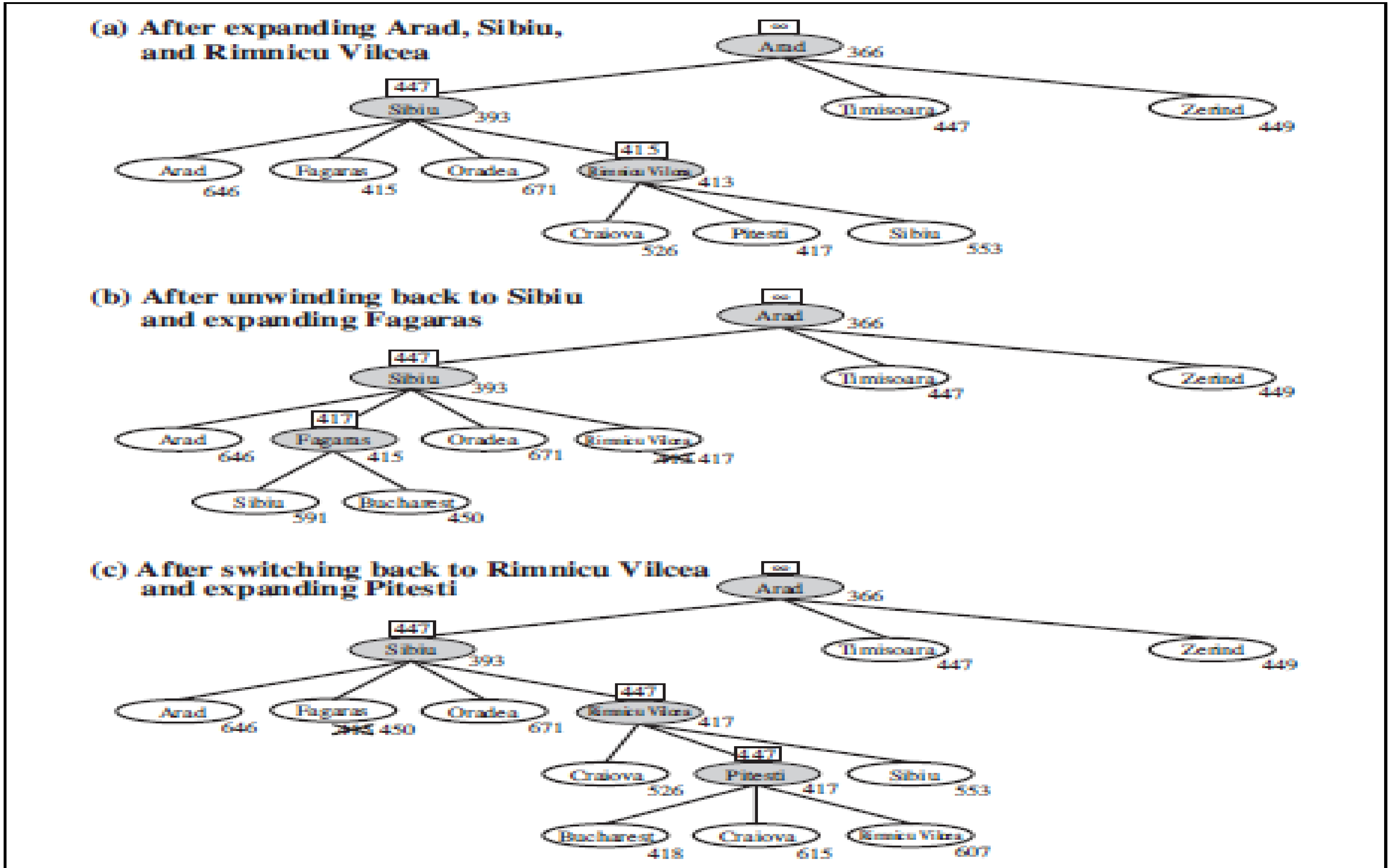
Initial
state
1
Arad
366=0+366
447
2
Sibiu
Timisoara
Zerind
393=140+253
447=118+ 329
449=75+374
3
Arad
Fagaras
Oradea
Rimnicu
417
646=280 +366
~~415=239+176~~
671=291+380
~~413=220+193~~
450
4
Craivoa
Pitesti
Sibiu
526=366+160
417=317+ 100
553=300+253
Bucharest
Craiova
Rimnicu
418=418+0
615=455+ 160
607=414+ 193

•



Initial
state
1
Arad
366=0+366
447
2
Sibiu
Timisoara
Zerind
393=140+253
447=118+ 329
449=75+374
Arad
Fagaras
Oradea
3
Rimnicu
417
646=280 +366
~~415=239+176~~
671=291+380
~~413=220+193~~
450
Craivoa
4
Pitesti
Sibiu
526=366+160
417=317+ 100
553=300+253
Bucharest
Craiova
Rimnicu
418=418+0
615=455+ 160
607=414+ 193

Informed/Heuristic Search strategies : Recursive Best First Search



Informed/Heuristic Search strategies: Best First Search

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
        return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE),∞)
function RBFS(problem, node, f limit ) returns a solution, or failure and a new f-cost limit
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        successors ←[ ]
        for each action in problem.ACTIONS(node.STATE) do
                add CHILD-NODE(problem, node, action) into successors
        if successors is empty then return failure,∞
        for each s in successors do /* update f with value from previous search, if any */
                s.f ←max(s.g + s.h, node.f ))
        loop do
                best ←the lowest f-value node in successors
                if best .f > f limit then return failure, best .f
                alternative ←the second-lowest f-value among successors
                result , best .f ←RBFS(problem, best , min( f limit, alternative))
                if result ≠ failure then return result
```

Recursive Best First Search - Performance

RBFS

- more efficient than IDA* , but still suffers from excessive node generation
- Optimal, like A*, if h(n) is admissible
- Space complexity
  - Proportional to d , linear in the depth of the deepest optimal solution
- Time Complexity depends
  - on the accuracy of heuristic function and
  - on how often best path changes as the nodes are expanded!

Informed/Heuristic Search strategies: MA*, SMA*

Memory Bounded A* - MA*

Simplified Memory Bounded A* - SMA*

SMA*

- proceeds like A*, expanding the best leaf until memory is full
- drops the worst leaf node – the one with highest f - value
- when it can not add a new node to the search tree, without dropping an old one
- backs up the value of the forgotten node to its parent node (similar to RBFS)
  - quality of the best path in that subtree is preserved in the backed up value with the ancestor

regenerates the subtree only when all other paths have been shown to look worse than the path it has forgotten

Heuristic Functions

8 – Puzzle

Solution 26 steps long.

7	2	4
5	-	6
8	3	1

-	1	2
3	4	5
6	7	8

Average solution cost for a randomly generated 8-puzzle instance is about 22 steps.

b ~ 3

Exhaustive tree search to depth 22 => states: 3^{22}

With Graph search 9!/2 (= 181440) distinct states are reachable

Good heuristics for finding shortest solutions: that never over estimate the number of steps to the goal

h1: number of misplaced tiles

ex: above initial state – all eight are out of position , h1 = 8 ,

h1 admissible as any tile that is out of place must be moved at least once

h2: sum of the horizontal and vertical distances(city block or Manhattan distance) of the tiles from their goal positions

Ex: h2 = 3+1+2+2+2+3+3+2= 18

h2 is admissible as any move can only move one tile one step closer to the goal.

h1, h2 do not over estimate as true solution cost is 26

Heuristics accuracy

Effect of heuristic accuracy on performance

Quality of heuristic? Can be characterized by effective branching factor b*

If the total number of nodes generated for a particular problem is N, and the solution depth is d, then the effective branching factor, b* that a uniform tree of depth d would have to contain N+1 nodes, is given by

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

Ex: if A* finds a solution at depth 5 using 52 nodes

$$52 + 1 = 1 + b^* + (b^*)^2 + (b^*)^3 + (b^*)^4 + (b^*)^5$$

$$b^* = 1.92$$

A well designed heuristic would have a value of b* close to 1

Heuristics – case study

8 – slide bar puzzle – 1200 random problems (initial state → goal state)

Solution lengths: 2 to 24 (100 problems for each even number). Solved with IDS, A* with h1, h2

	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

h2 better than h1
A* far better than IDS

Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A_* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

Heuristic Functions

Generating admissible heuristics from relaxed problems

h1: misplaced tiles, h2: Manhattan distance –

estimates of the remaining path length for the 8-puzzle

Relaxed problem: problem with fewer restrictions on the actions

Ex: 8 slide bar puzzle –

tile could move anywhere instead of just to the adjacent empty square

➔ h1 gives exact number of steps shortest solution

tile could move one square in any direction, even onto an occupied square

➔ h2 gives exact number of steps in the shortest solution

Heuristic functions

The state space graph of the relaxed problem is a super graph of the original state space,

as the removal of restrictions creates added edges in the graph

Any optimal solution in the original problem is also a solution in the relaxed problem.

Relaxed problem may have better solution, if the added edges provide short cuts.

The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem,

As the derived heuristic is an exact cost for the relaxed problem, it must satisfy/obey triangular inequality and therefore is consistent.

Heuristic Functions

It is possible to construct relaxed problems automatically, if the problem definition is written down in formal language

8- puzzle actions

A tile can move from square A to square B

if A is horizontally or vertically adjacent to B
and B is blank

Three Relaxed problems generation by removing one or both conditions

1. A tile can move from square A to square B

    if A is adjacent to B - (h2)

2. A tile can move from square A to square B

    if B is blank

3. A tile can move from square A to square B - (h1)

Heuristic functions

Sub Problem of 8-slide bar puzzle:

*	2	4
*	-	*
*	3	1

→

-	1	2
3	4	*
*	*	*

Task: get 1,2,3,4 into their correct positions without worrying about what happens to other tiles.

The cost of optimal solution of this sub problem is a lower bound on the cost of the complete problem.

ABSOLVER can generate heuristics automatically from problem definition using relaxed problem method (and other techniques)

Pattern databases: sub problems

Meta state space, Object state space

Heuristic search space compared to BFS

Artificial_Intelligence Luger.pdf 168 / 912 50%

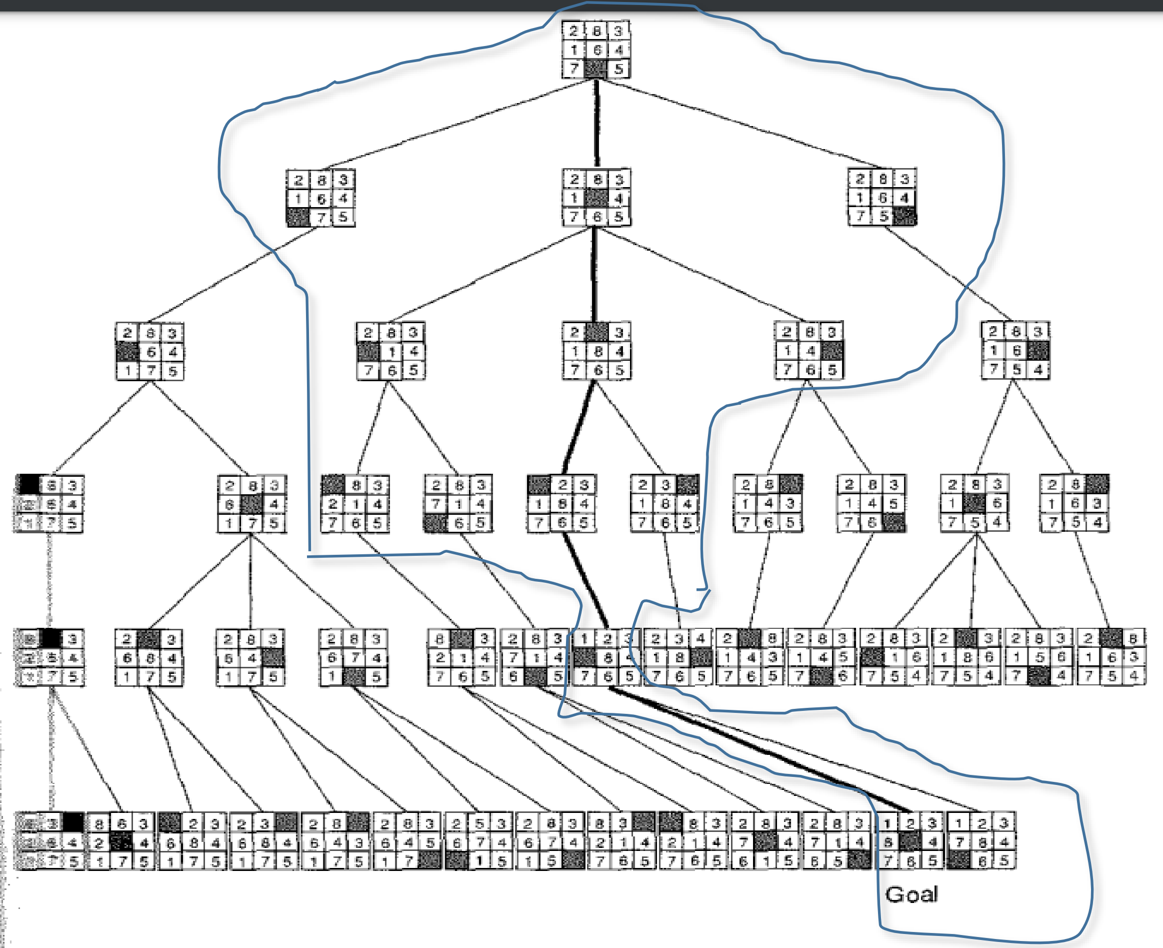


Figure 4.18 Comparison of state space searched using heuristic search with space searched by breadth-first search. The portion of the graph searched heuristically is shaded. The optimal solution path is in bold. Heuristic used is