CSE3013 Artificial Intelligence

Module – 2

Dr. SRIDHAR RAJ S, SCOPE, VIT - VELLORE

Problem solving by search

- An important aspect of intelligence is *goal-based* problem solving.
- The solution of many problems (e.g. timetabling, chess) can be described by finding a sequence of actions that lead to a desirable goal.
- Each action changes the *state* and the aim is to find the sequence of actions and states that lead from the initial (start) state to a final (goal) state.

Problem solving by search

A well-defined problem can be described by:

- **Initial state**
- **Operator or successor function** - for any state x returns s(x), the set of states reachable from x with one action
- **State space** - all states reachable from initial by any sequence of actions
- **Path** - sequence through state space
- **Path cost** - function that assigns a cost to a path. Cost of a path is the sum of costs of individual actions along the path
- **Goal test** - test to determine if at goal state

Search Terminology

- **Problem Space** − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
- **Problem Instance** − It is Initial state + Goal state.
- **Problem Space Graph** − It represents problem state. States are shown by nodes and operators are shown by edges.
- **Depth of a problem** − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
- **Space Complexity** − The maximum number of nodes that are stored in memory.
- **Time Complexity** − The maximum number of nodes that are created.
- **Admissibility** − A property of an algorithm to always find an optimal solution.
- **Branching Factor** − The average number of child nodes in the problem space graph.
- **Depth** − Length of the shortest path from initial state to goal state.

To Build a System to Solve a Particular Problem, The Following Four Things are Needed

1. Define the problem precisely- specify both initial and final situations(state)
2. Analyze the problem
3. Isolate and represent the task knowledge that is necessary to solve the problem
4. Choose the best problem solving technique and apply it

State space search Problem = Searching for a goal state

- It is a process in which successive configurations or states of an instance are considered , with the goal of finding a goal state with a desired property .
- State space- a set of states that a problem can be in.
  - The group consisting of all the attainable states of a problem
  - ex: Customers in a line would have state space {0,1,2….}

Search problem

- S: the full set of states
- S_0 :the initial state
- A: S ->S set of operators
- G : the set of final states.
- G is subset of S
- Search problem: Find a sequence of actions which transforms the agent from the initial state to goal state.

Problem formulation

- A single state problem formulation is defined by four items
Initial state, successor function, goal test and path cost

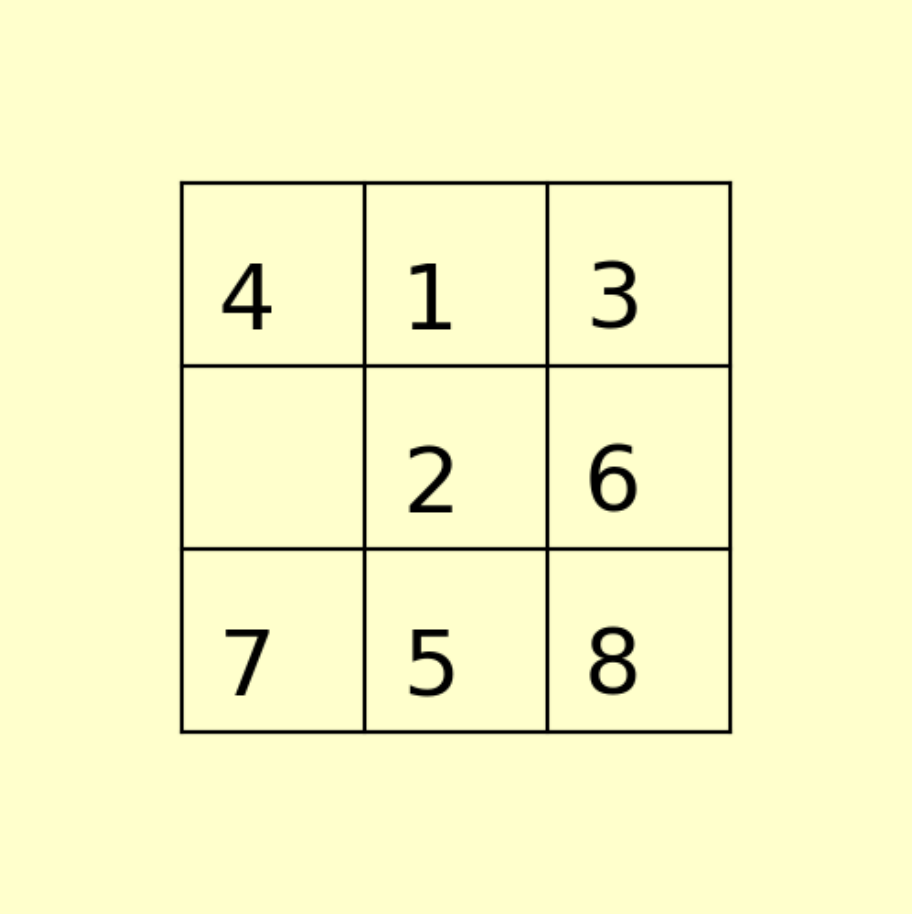
- Problem formulation means choosing a relevant set of states to consider, and a feasible set of operators for moving from one state to another

- Search is the process of imagining sequences of operators applied to the initial state and checking which sequence reaches a goal state.

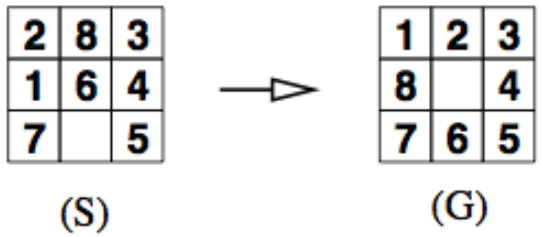
Examples.
Problem: On holiday in Singapore; currently in Mysore. Flight leaves tomorrow from Bangalore. Find a short route to drive to Bangalore.

Formulate problem:
states: various cities
actions: drive between cities
solution: sequence of cities
Path Cost: distance travelled

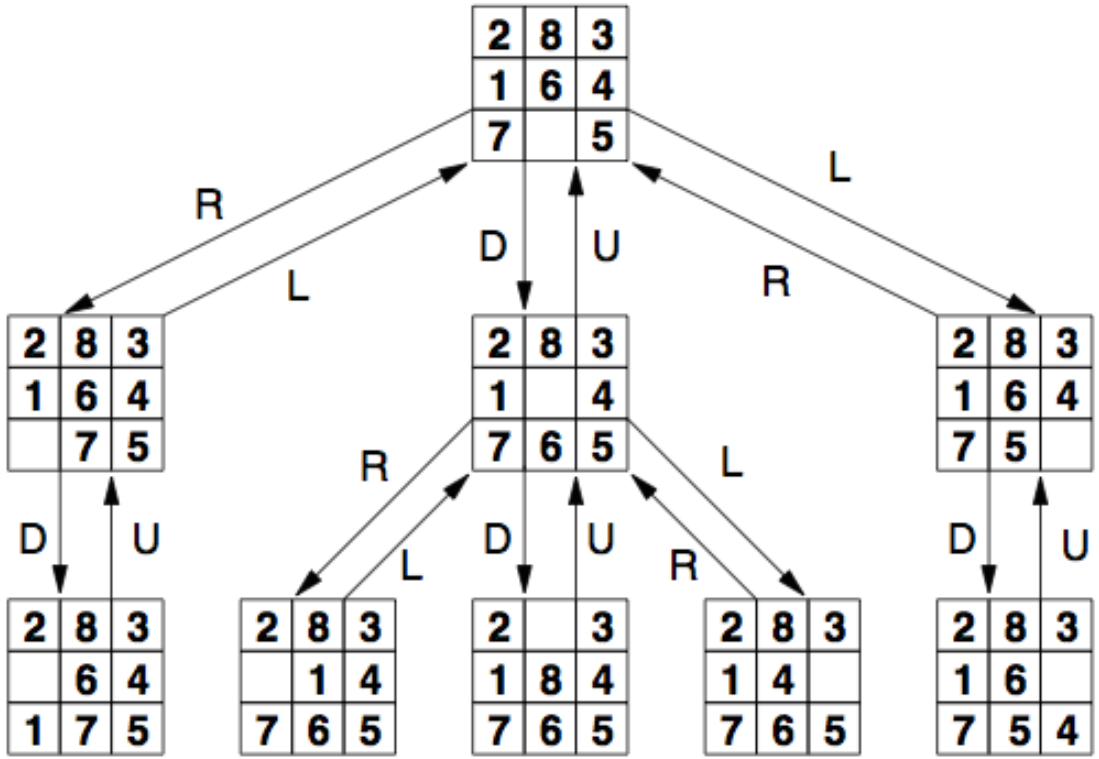
The 8 - Puzzle



Problem: Go from state S to state G.



States: Locations of tiles
Actions: Move blank left, right, up, down
Goal test: Given
Path cost: 1 per move

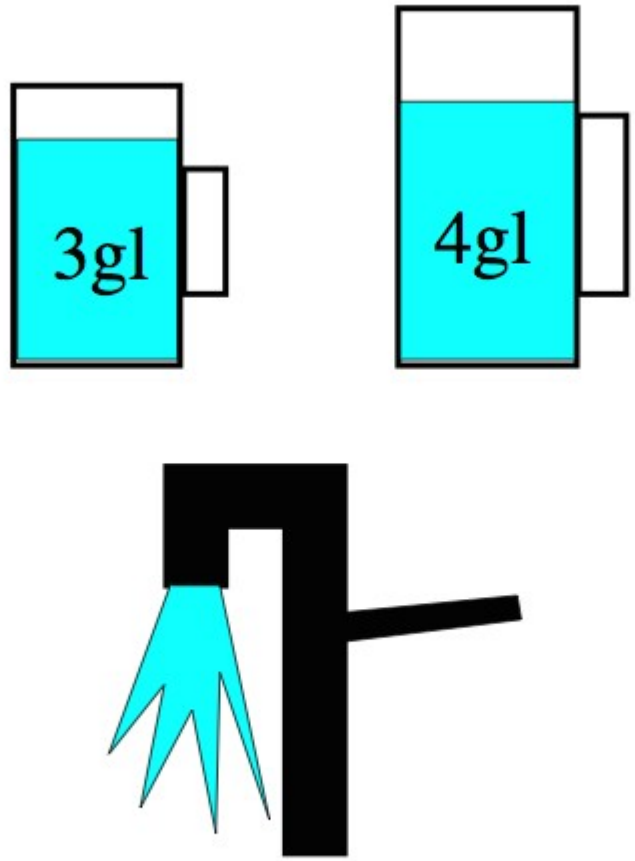


State space search: Playing Chess

- State space is a set of legal positions.
- Starting at the initial state.
- Using the set of rules to move from one state to another.
- Attempting to end up in a goal state.

Water Jug problem

- We have 2 jugs, one of 4-gallon, second of 3-gallon. No measurement or no marking is there on jugs. We have fill 4-gallon jug with 2 gallon water.
- Assumptions
  - X -> range from 0 to 4 for 4-gallon jug
  - Y -> range from 0 to 3 for 3-gallon jug
  - We can fill water through pump.
  - We can pour waste water on ground.
  - We can put water from one jug to another



Get exactly 2 gallons of water into the 4gl jug.

Sl No	Current state	Next State	Descritpion
1	(x,y) if x < 4	(4,y)	Fill the 4 gallon jug
2	(x,y) if y <3	(x,3)	Fill the 3 gallon jug
3	(x,y) if x > 0	(x-d, y)	Pour some water out of the 4 gallon jug
4	(x,y) if y > 0	(x, y-d)	Pour some water out of the 3-gallon jug
5	(x,y) if x>0	(0, y)	Empty the 4 gallon jug
6	(x,y) if y >0	(x,0)	Empty the 3 gallon jug on the ground
7	(x,y) if x+y >= 4 and y >0	(4, y-(4-x))	Pour water from the 3 –gallon jug into the 4 –gallon jug until the 4-gallon jug is full

8	(x, y) if x+y >= 3 and x>0	(x-(3-y), 3)	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if x+y <=4 and y>0	(x+y, 0)	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if x+y <= 3 and x>0	(0, x+y)	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	(0,2)	(2,0)	Pour the 2 gallons from 3-gallon jug into the 4-gallon jug
12	(2,y)	(0,y)	Empty the 2 gallons in the 4-gallon jug on the ground

To solve the water jug problem

- The start state is (0,0)
- The goal state is (2,n)

- Required a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen
- The appropriate change to the state is made as described in the corresponding right side
- The resulting state is checked to see if it corresponds to goal state.

Gallons in the 4-gallon jug	Gallons in the 3-gallon jug	Rule applied
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5 or 12
0	2	9 0r 11
2	0	

Production System

A production system consists of:

- A set of rules, each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if that rule is applied.
- One or more knowledge/databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem.
- A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
- A rule applier

Control strategies

- How to decide which rule to apply next during the process of searching for a solution to a problem?
- The two requirements of good control strategy are that
  - It should cause motion.
  - It should be systematic

Search algorithms

- Breadth first search: Representing water jug problem starting with its initial state along with its off springs i.e. various possible solutions until for reaching a goal state.
  ◦ It gives local as well as global motion.
  - Depth first search: One node of tree is explored until either dead end or goal state is not found.
  - In case of dead end chronological back tracking is done to explore various other ways to solve that problem. It also keeps current searching path in memory.

Breadth first search

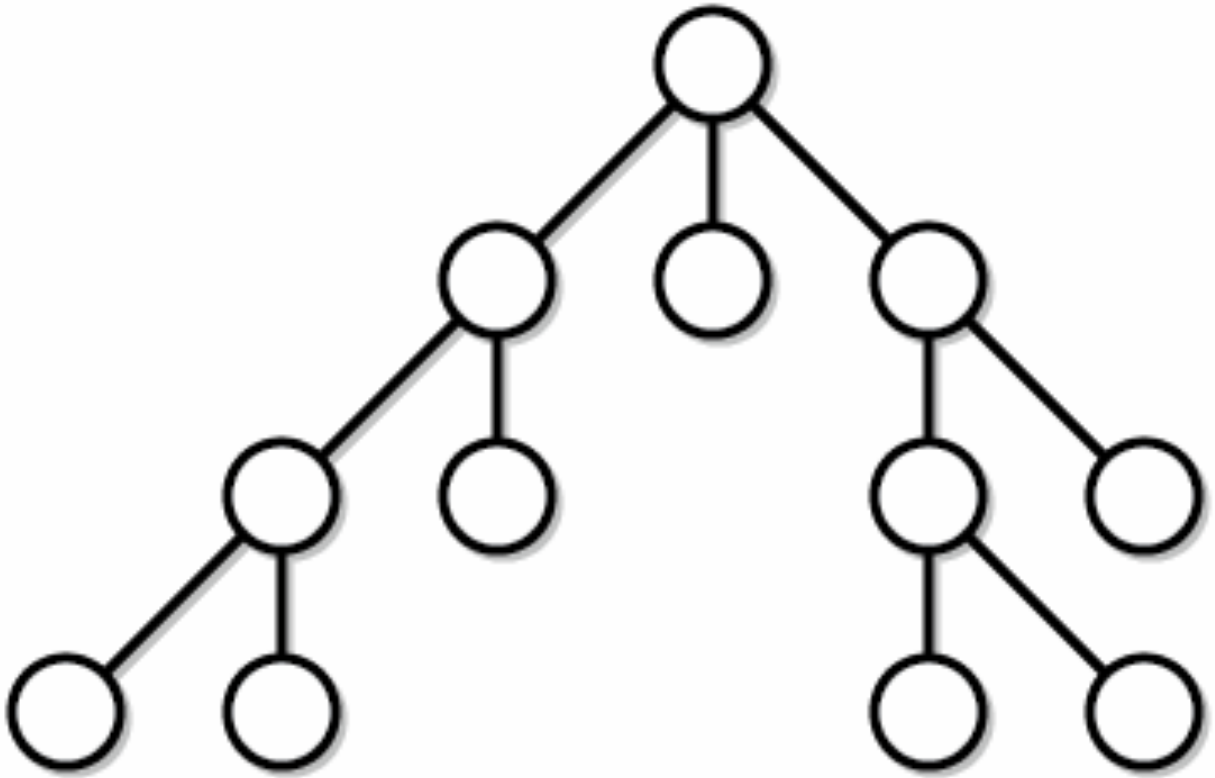
- Explore *nodes* in tree order: library, school, hospital, factory, park, newsagent, uni, church. (conventionally explore left to right at each level)

Algorithm:
1. Create a variable called NODE-LIST and set it to initial state
2. Until a goal state is found or NODE-LIST is empty do
a. Remove the first element from NODE-LIST and call it E. If NODELIST was empty, quit
b. For each way that each rule can match the state described in E do:
i. Apply the rule to generate a new state
ii. If the new state is a goal state, quit and return this state
iii. Otherwise, add the new state to the end of NODE-LIST

Advantages of BFS

- It does not takes us to the blind way.
- It will definitely give us the solution, if that exists.
- It gives us solution with minimum value.
- It guarantees that a solution with minimum value will be explored first, only at last all nodes with maximum values will be explored.

Depth first search



Depth first search

Algorithm:
1.If the initial state is a goal state, quit and return success
2.Otherwise, do the following until success or failure is signalled:
a. Generate a successor, E, of initial state. If there are no more successors, signal failure.
b. Call Depth-First Search, with E as the initial state
c. If success is returned, signal success. Otherwise continue in this loop.

Advantages of DFS

- It takes less memory as compared to BFS as BFS requires entire tree to be stored but this requires only one path.
  - Sometimes solution lies in earlier stages then DFS is better.
  - If there are multiple solutions then DFS stops when first solution is found. Where as BFS gives all the solutions at the same time.

Disadvantages of DFS

- Sometimes it gives dead end after searching a lot.
- It can or cannot give solution to a problem.

Uniform Cost

- Breadth first search is optimal when all step costs are equal.
- Uniform Cost search is optimal when step costs varies.
- Uniform-cost search expands the node n with lowest path cost.

Uniform Cost Search:example

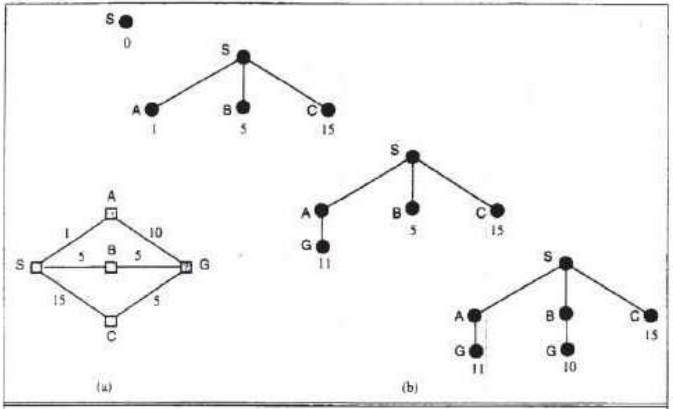


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Depth Limited Search

- **Depth-limited search** avoids the pitfalls of depth-first search by imposing a cutoff on the maximum depth of a path.

- This cutoff can be implemented with a special depth-limited search algorithm, or by using the general search algorithm with operators that keep track of the depth.

- To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don’t expand beyond depth L.

Iterative deepening depth-first search

- To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don't expand beyond depth L

  ->Depth first search.

- What of solution is deeper than L? -->Increase L iteratively.

  ->Iterative Deepening Search

Contd...

- When the initial depth cut-off is one, it generates only the root node and examines it.
- If the root node is not the goal, then depth cut-off is set to two and the tree up to depth 2 is generated using typical depth first search.
- Similarly, when the depth cut-off is set to m, the tree is constructed up to depth m by depth first search.

Contd…

- **Iterative deepening search** is a strategy that sidesteps the issue of choosing the best depth limit by trying all possible depth limits: first depth 0, then depth 1, then depth 2, and so on.

Algorithm: IDS

Procedure Iterative-deepening

Begin

1. Set current depth cutoff =1;
2. Put the initial node into a stack, pointed to by stack-top;
3. **While** the stack is not empty and the depth is within the given depth cut-off do

   Begin

   Pop stack to get the stack-top element;

   if stack-top element = goal, return it and stop

   else push the children of the stack-top in any order into the stack;

   End While;

4. Increment the depth cut-off by 1 and repeat through step 2;

End.

Example : Iterative deepening

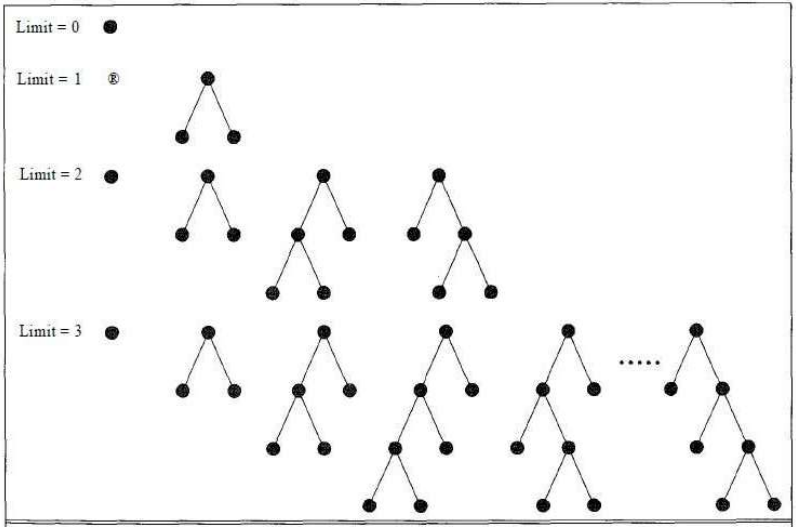


Figure 3.16 Four iterations of iterative deepening search on a binary tree.

Properties: IDS

Complete? Yes

Time? $O(b^d)$

Space? $O(bd)$

Optimal? Yes, if step cost = 1 or increasing function of depth.

Bi-Directional Search

- Alternate searching from the start state toward the goal and from the goal state toward the start.
- Stop when the frontiers intersect.
- Works well only when there are unique start and goal states.
- Requires the ability to generate “predecessor” states.
- Time complexity: $O(b^{d/2})$. Space complexity: $O(b^{d/2})$.

Bidirectional search: figure

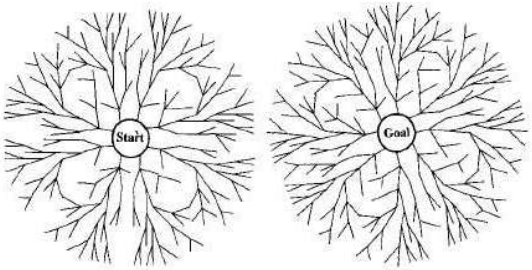


Figure 3.17 A schematic view of a bidirectional breadth-first search that is about to succeed, when a branch from the start node meets a branch from the goal node.

Comparing Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l > d$	Yes	Yes

Figure 3.18 Evaluation of search strategies. b is the branching factor; d is the depth of solution; m is the maximum depth of the search tree; l is the depth limit.

Heuristic search

- A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed.
- A Heuristic (or a heuristic function) takes a look at search algorithms. At each branching step, it evaluates the available information and makes a decision on which branch to follow.
- It does so by ranking alternatives. The Heuristic is any device that is often effective but will not guarantee work in every case.

Heuristic search

- So *why do we need heuristics*? One reason is to produce, in a reasonable amount of time, a solution that is good enough for the problem in question. It doesn't have to be the best- an approximate solution will do since this is fast enough.
- Most problems are exponential. Heuristic Search let us reduce this to a rather polynomial number. We use this in AI because we can put it to use in situations where we can't find known algorithms.
- We can say Heuristic Techniques are weak methods because they are vulnerable to combinatorial explosion.

Heuristic search techniques

- Direct Heuristic Search Techniques in AI
  - Blind Search, Uninformed Search, and Blind Control Strategy. These aren't always possible since they demand much time or memory.
  - They search the entire state space for a solution and use an arbitrary ordering of operations. Examples of these are Breadth First Search (BFS) and Depth First Search (DFS).
- Weak Heuristic Search Techniques in AI
  - Informed Search, Heuristic Search, and Heuristic Control Strategy. These are effective if applied correctly to the right types of tasks and usually demand domain-specific information.
  - We need this extra information to compute preference among child nodes to explore and expand. Each node has a heuristic function associated with it. Examples are Best First Search (BFS) and A*.

Informed Search Algorithms

- So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space.
- But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc.
- This knowledge help agents to explore less to the search space and find more efficiently the goal node.

Informed search algorithm

- The informed search algorithm is more useful for large search space.
- Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristic function

- Heuristic is a function which is used in Informed Search, and it finds the most promising path.
- It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- The heuristic method, however, might not always give the best solution, but it is guaranteed to find a good solution in reasonable time.
- Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states.
- The value of the heuristic function is always positive.

Admissibility of the heuristic function

- Admissibility of the heuristic function is given as

$$h(n) <= h^*(n)$$

  here h(n) is heuristic cost and h*(n) is the estimated cost.

- Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search

- Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have not yet been expanded.
- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list.
- The algorithm continues until a goal state is found.

Best First Search Algorithm (Greedy search)

- Greedy best-first search algorithm always selects the path which appears best at that moment (most promising node).
- It is the combination of depth-first search and breadth-first search algorithms. Utilizes the pros of both.
- The greedy best first algorithm is implemented by the priority queue.

Best First Search Algorithm (Greedy search)

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

Greedy Best First Search Algorithm

- For eg., Consider the below search problem, and we will traverse it using greedy best-first search.
- At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.
- *Refer book for the example "Artificial Intelligence: A modern approach" by Russell S*

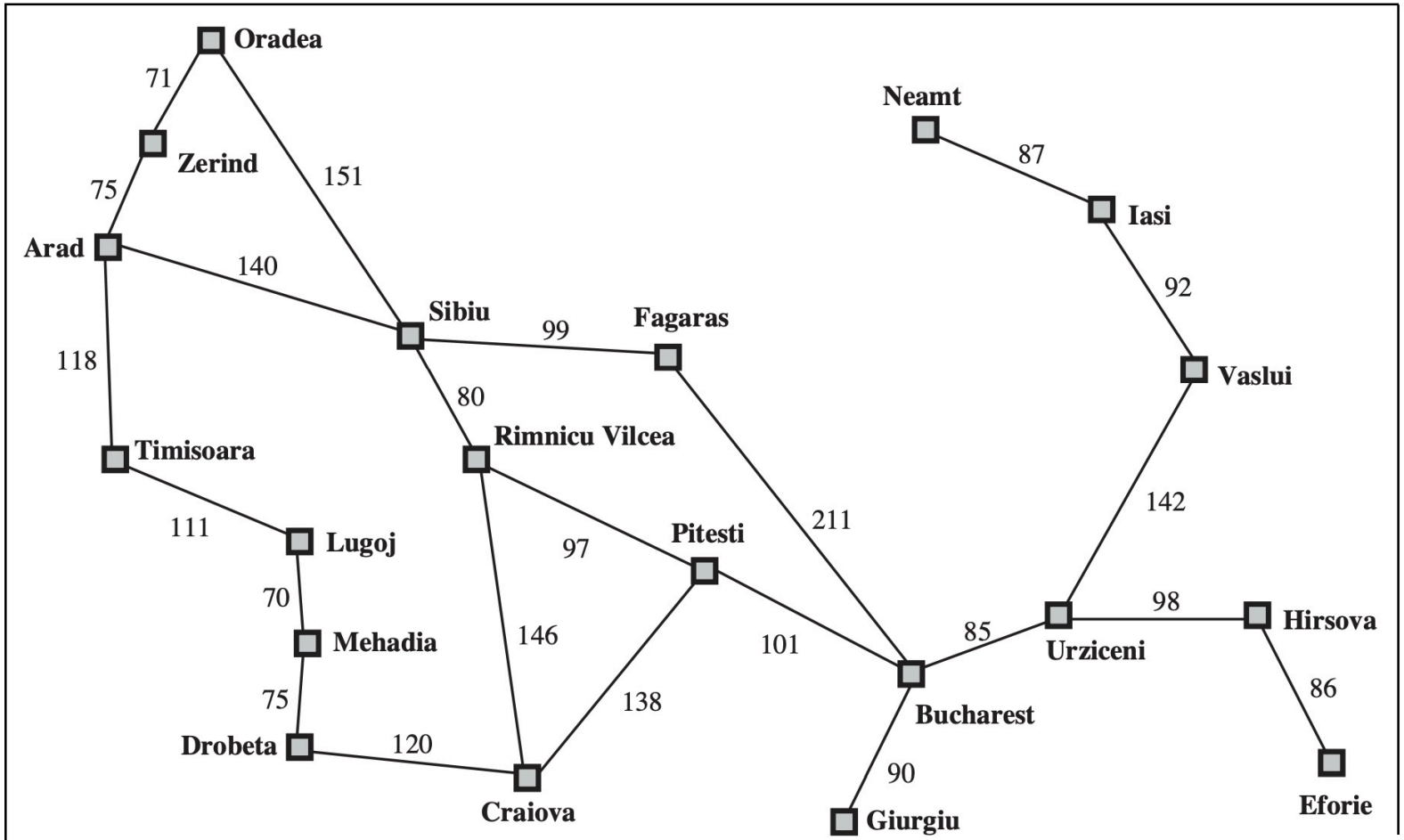


Figure 3.2 A simplified road map of part of Romania.

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

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

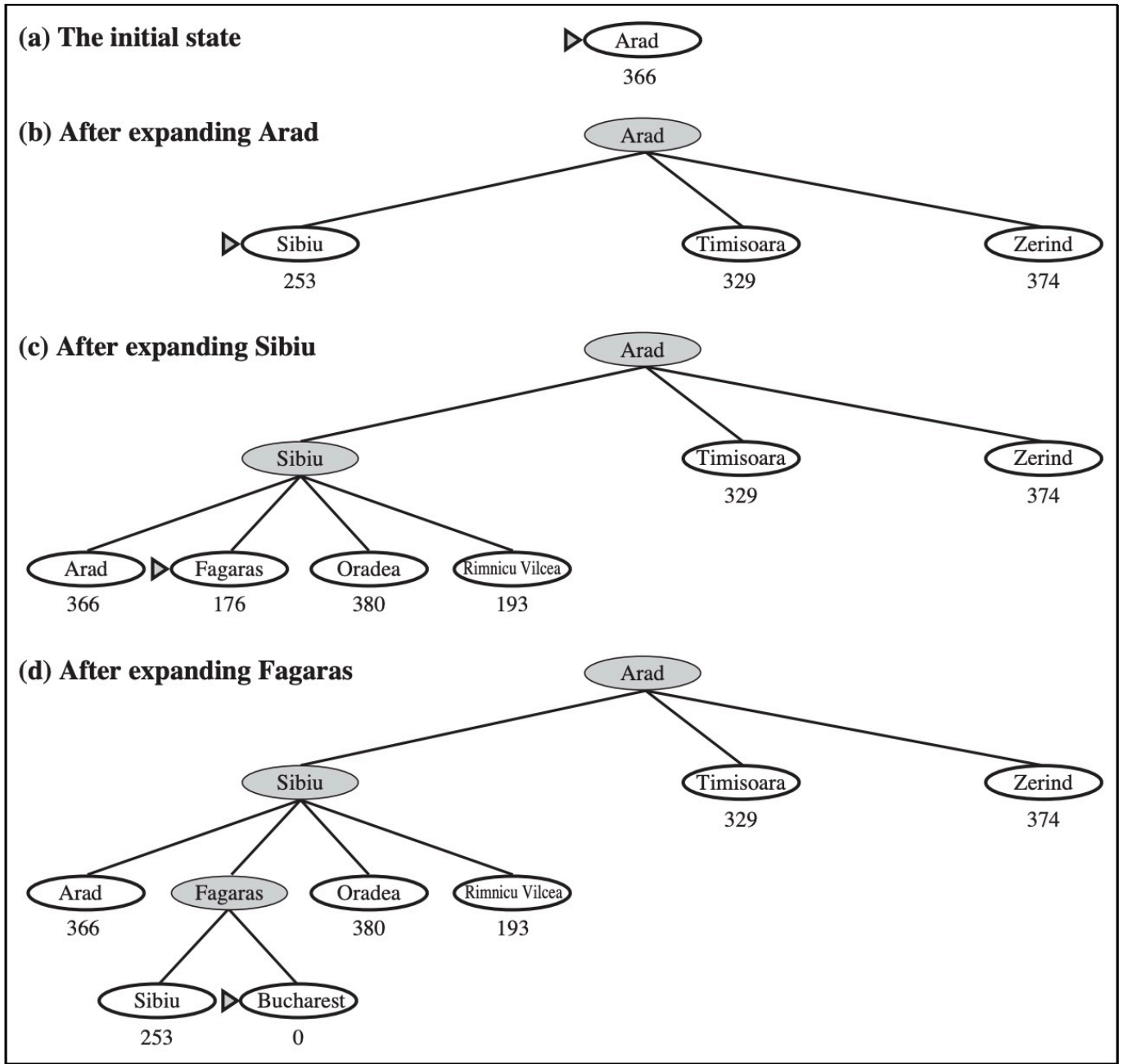
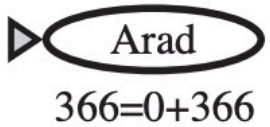
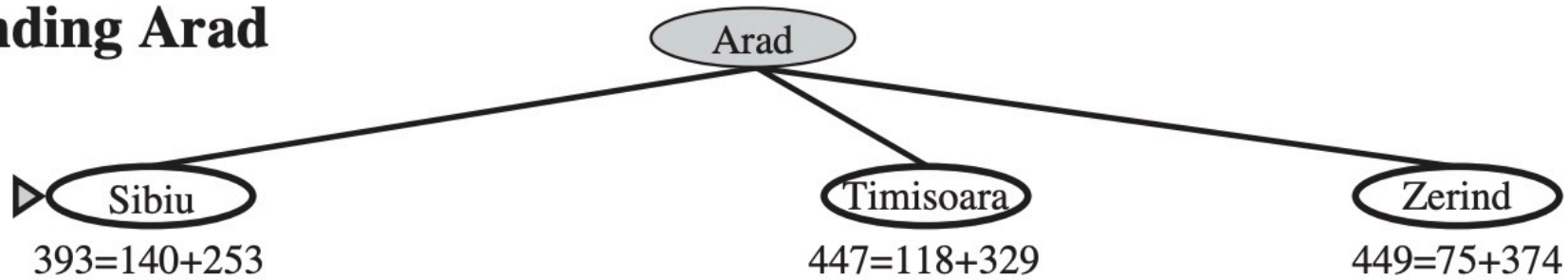


Figure 3.23 Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

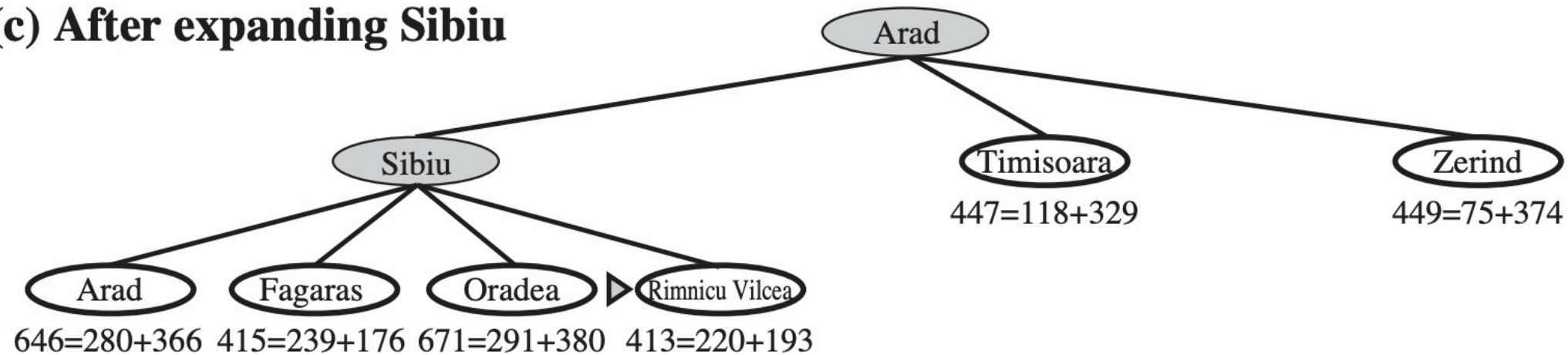
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu



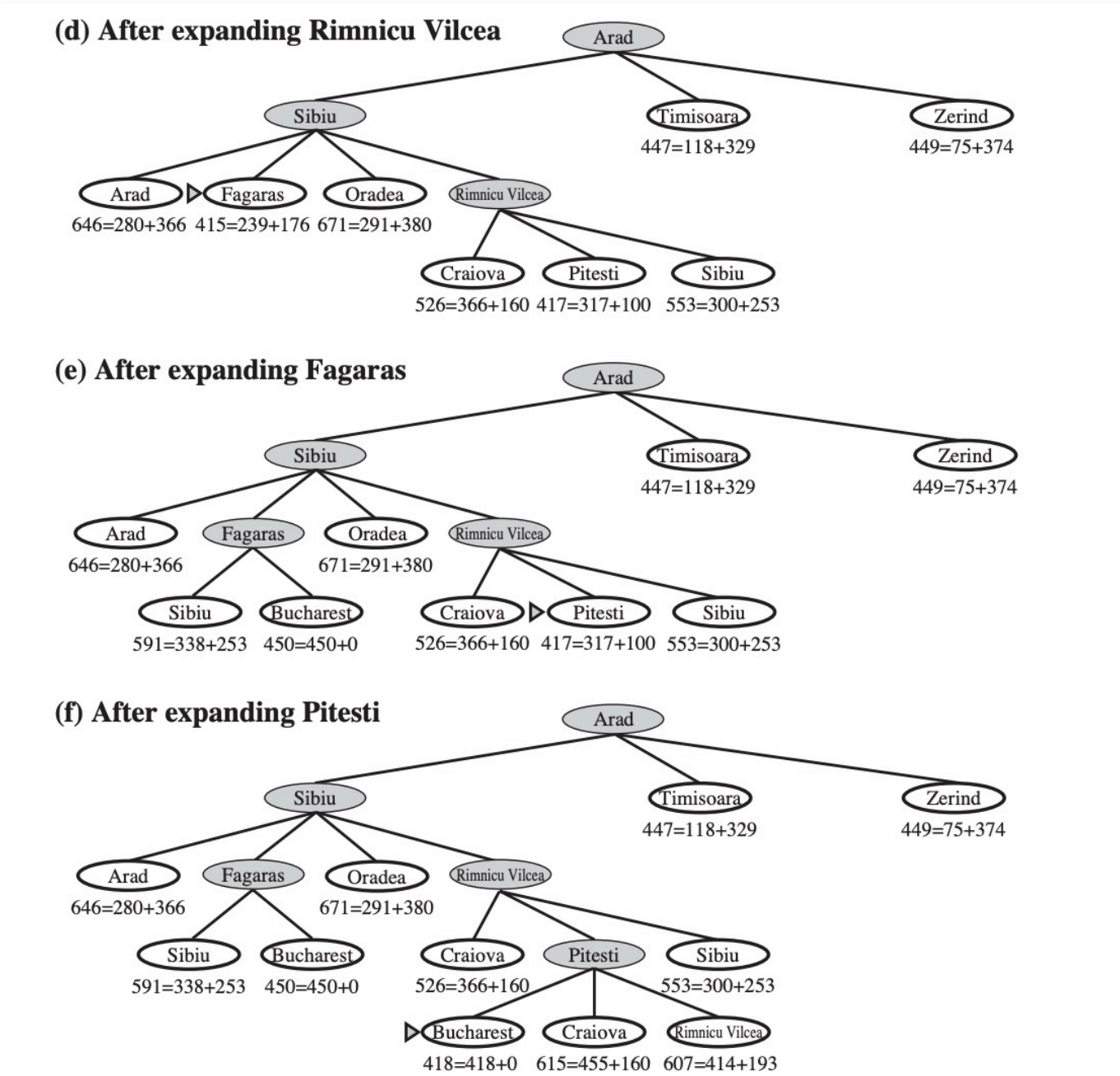


Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The