SEARCH ALGORITHMS

SEARCH ALGORITHMS

- Node: State in the state space
- Edge/branch: actions

- Parent node
- Child/Successor node
- Leaf node
- Frontier node

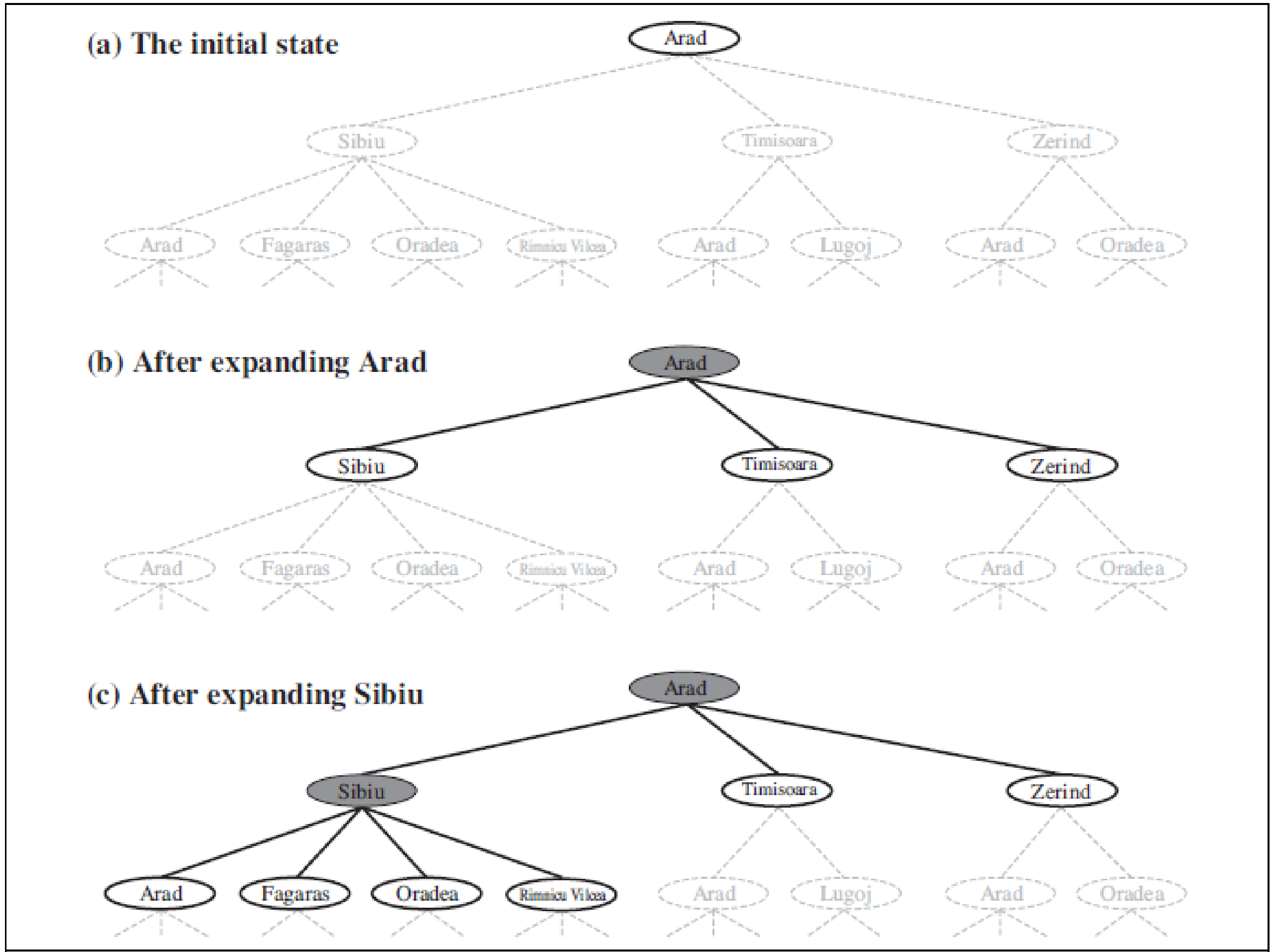

Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

TREE-SEARCH ALGORITHM

```
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
```

SEARCH TREE

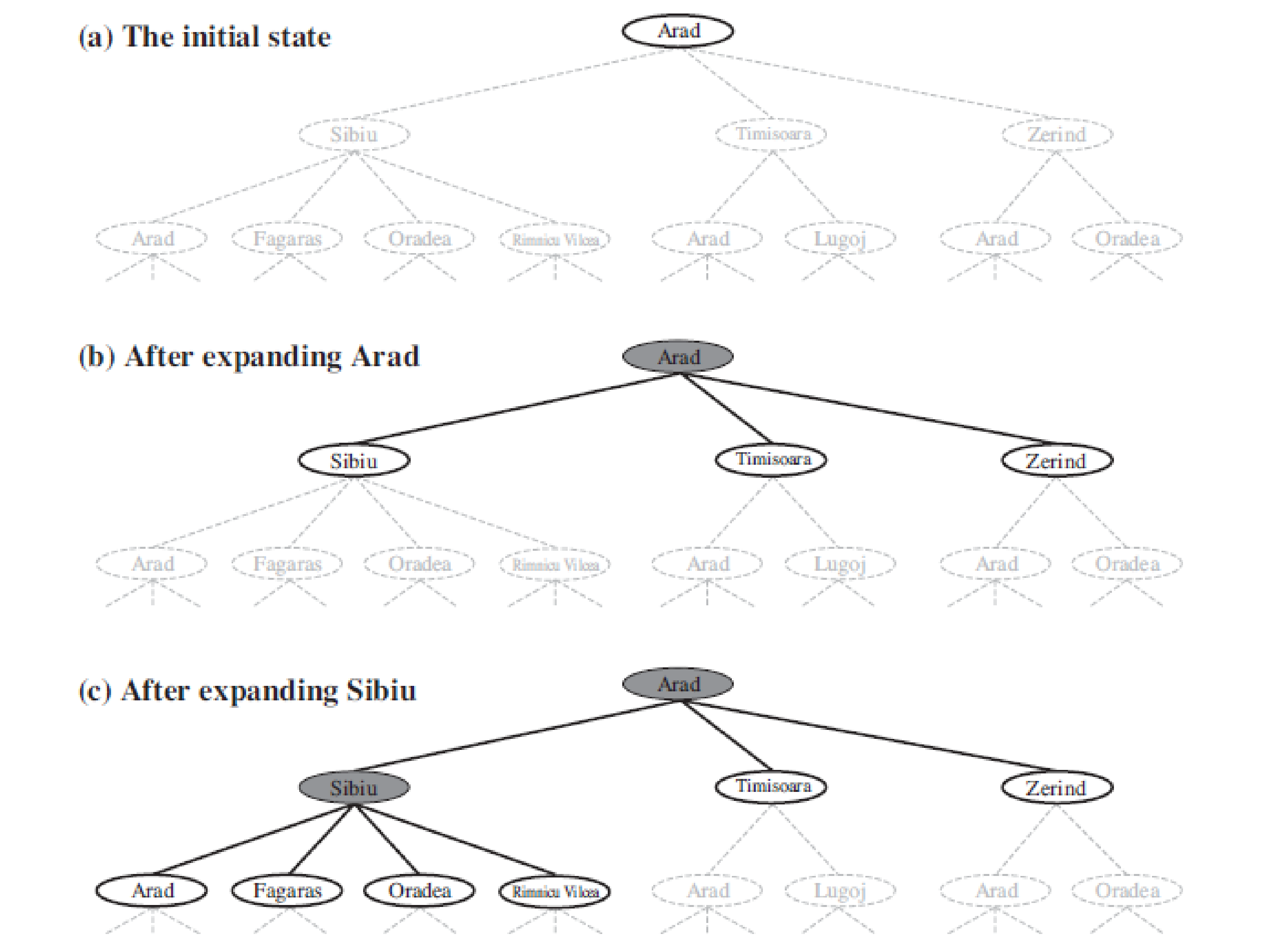

Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

REDUNDANT PATHS

- Problem: Loopy path
- Solutions:

1. Remember all previously reached states.
2. Not worry about repeating the past.
   - Graph search: Algorithm that checks for redundant paths.
   - Tree-like search: Algorithm that does not check for redundant paths.
3. Check for cycles but not for the redundant paths.

GRAPH-SEARCH ALGORITHM

```
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
```

GRAPH-SEARCH

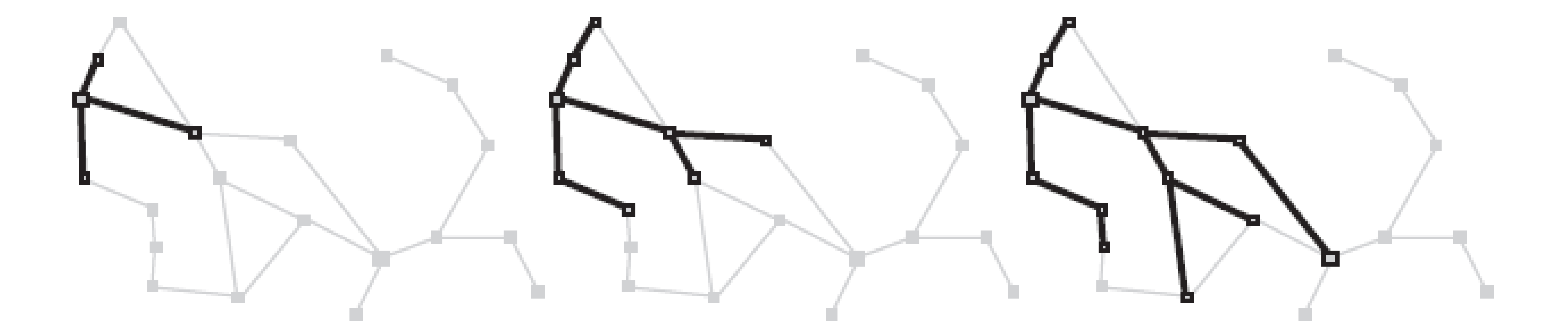

Figure 3.8 A sequence of search trees generated by a graph search on the Romania problem of Figure 3.2. At each stage, we have extended each path by one step. Notice that at the third stage, the northernmost city (Oradea) has become a dead end: both of its successors are already explored via other paths.

GRAPH-SEARCH

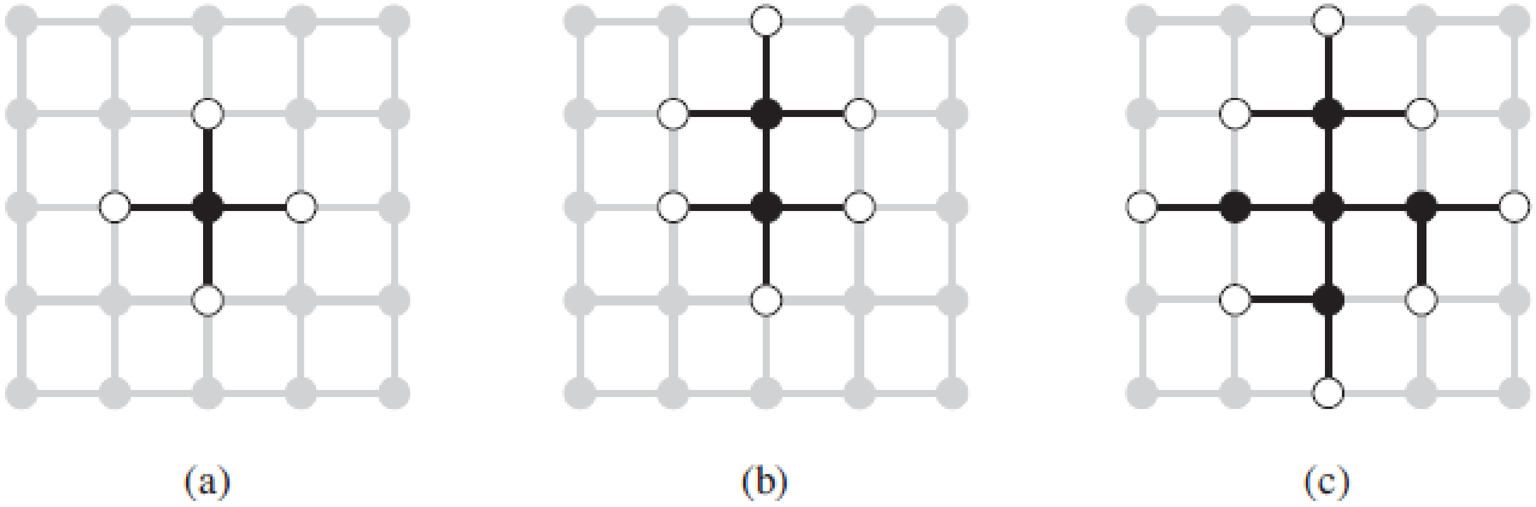

Figure 3.9 The separation property of GRAPH-SEARCH, illustrated on a rectangular-grid problem. The frontier (white nodes) always separates the explored region of the state space (black nodes) from the unexplored region (gray nodes). In (a), just the root has been expanded. In (b), one leaf node has been expanded. In (c), the remaining successors of the root have been expanded in clockwise order.

SEARCH DATA STRUCTURE (NODE)

- node.STATE: The state to which the node corresponds
- node.PARENT: The node in the tree that generated this node
- node.ACTION: The action that was applied to the parent's state to generate this node
- node.PATH-COST: The total cost of the path from the initial state to this node.

CHILD-NODE FUNCTION

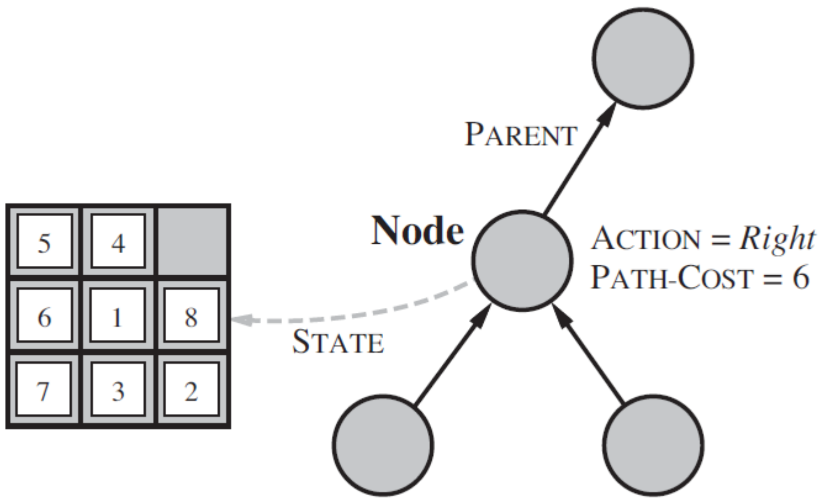

Figure 3.10 Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

function CHILD-NODE(*problem*, *parent*, *action*) **returns** a node
  return a node with
      STATE = *problem*.RESULT(*parent*.STATE, *action*),
      PARENT = *parent*, ACTION = *action*,
      PATH-COST = *parent*.PATH-COST + *problem*.STEP-COST(*parent*.STATE, *action*)

SEARCH DATA STRUCTURE (FRONTIER)

- IS-EMPTY(frontier) returns true only of there are no nodes in the frontier.
- POP(frontier) removes the top node from the frontier and returns it.
- TOP(frontier) returns (but does not remove) the top node of the frontier.
- ADD(node, frontier) inserts node into its proper place in the queue.

SEARCH DATA STRUCTURE (QUEUE)

- Priority queue: used in best-first search
- FIFO queue: used in breadth-first search
- LIFO queue: used in depth-first search

MEASURING PROBLEM-SOLVING PERFORMANCE

- Completeness
- Cost optimality
- Time complexity
- Space complexity

How do we decide which node from the frontiers to expand first?