INFORMED (HEURISTIC) SEARCH STRATEGIES

Course Code: **CSC4226** Course Title: **Artificial Intelligence and Expert System**

Dept. of Computer Science
Faculty of Science and Technology

Lecture No:	**Four (4)**	**Week No:**	**Four (4)**	**Semester:**	
Lecturer:	*Dr. Abdus Salam*		*abdus.salam@aiub.edu*		

LECTURE OUTLINE

1. **BEST-FIRST SEARCH**
2. **GREEDY BEST-FIRST SEARCH**
3. **A* SEARCH**
4. **CONDITIONS FOR OPTIMALITY**
5. **OPTIMALITY OF A***

INFORMED SEARCH STRATEGY

Informed search strategy—

one that uses **problem-specific knowledge** beyond the **definition of the problem itself**

can find solutions more efficiently than can an uninformed strategy.

problem-specific knowledge is the extra bit of information the program uses rather than the problem formulation, thus known as **Informed Search**

BEST-FIRST SEARCH

- The general approach to informed search is called **best-first search**
- Best-first search is an instance of the general **TREE-SEARCH** or **GRAPH-SEARCH** algorithm in which a node is selected for expansion based on an **evaluation function**, ***f(n).***
- The evaluation function is construed as a cost estimate, so the node with the *lowest* evaluation is expanded first
- The implementation of best-first graph search is identical to that for **uniform-cost search**, except for the **use of f instead of g** to order the priority queue.
- The choice of f determines the search strategy.

HEURISTIC SEARCH COMPARED WITH BLIND SEARCH

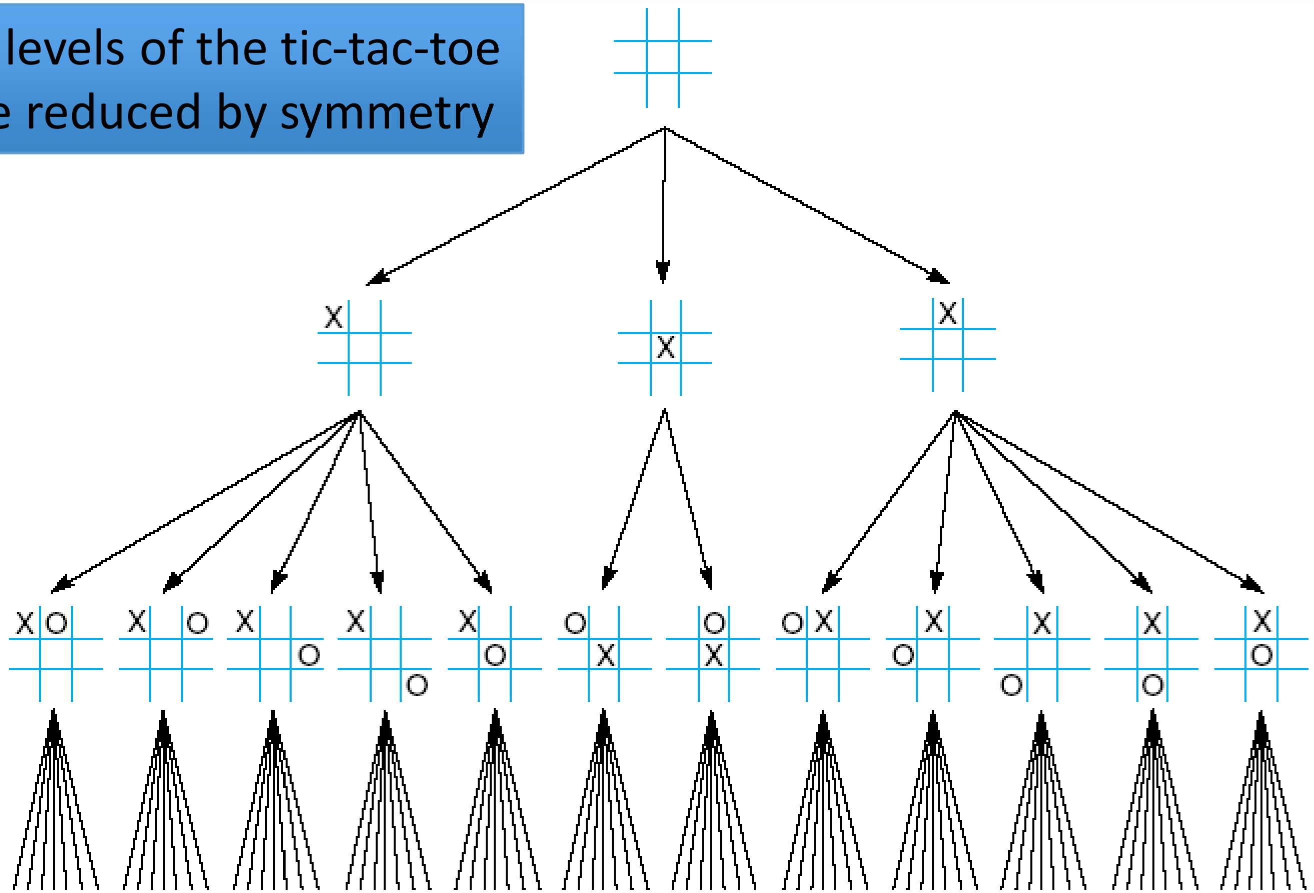
Brute force / Blind search	Heuristic search
◊ Only have knowledge about already explored nodes	◊ Estimates "distance" to goal state
◊ No knowledge about how far a node is from goal state	◊ Guides search process toward goal state
	◊ Prefer states (nodes) that lead close to and not away from goal state

HEURISTIC FUNCTION

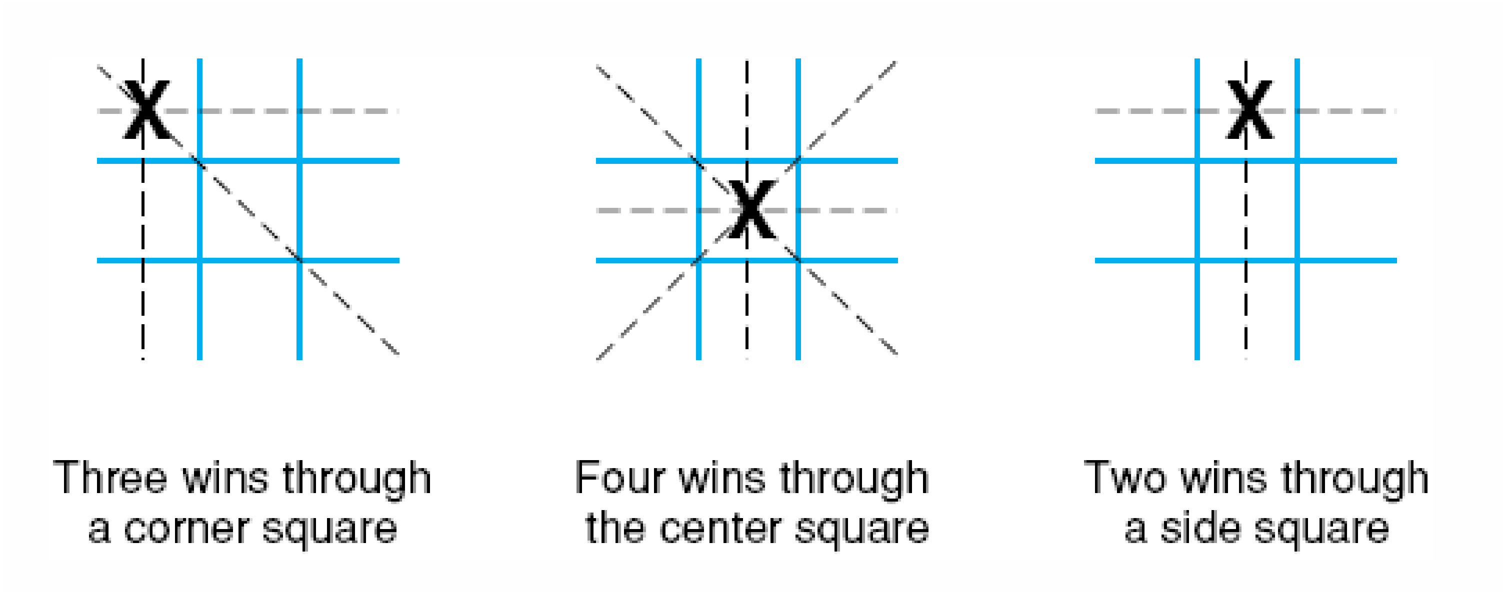
- The choice of **f(evaluation function)** determines the search strategy
- Best-first algorithms include as a component of **f** a **heuristic function**, denoted **h(n)**
- h(n) = estimated cost of the cheapest path from the state at node *n* to a goal state.
- h(n) takes a *node* as input, but, unlike g(n), it depends only on the *state* at that node.
- Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm
- Consider them to be arbitrary, nonnegative, problem-specific functions, with one constraint: **if n is a goal node, then h(n)=0.**

HEURISTIC FUNCTION

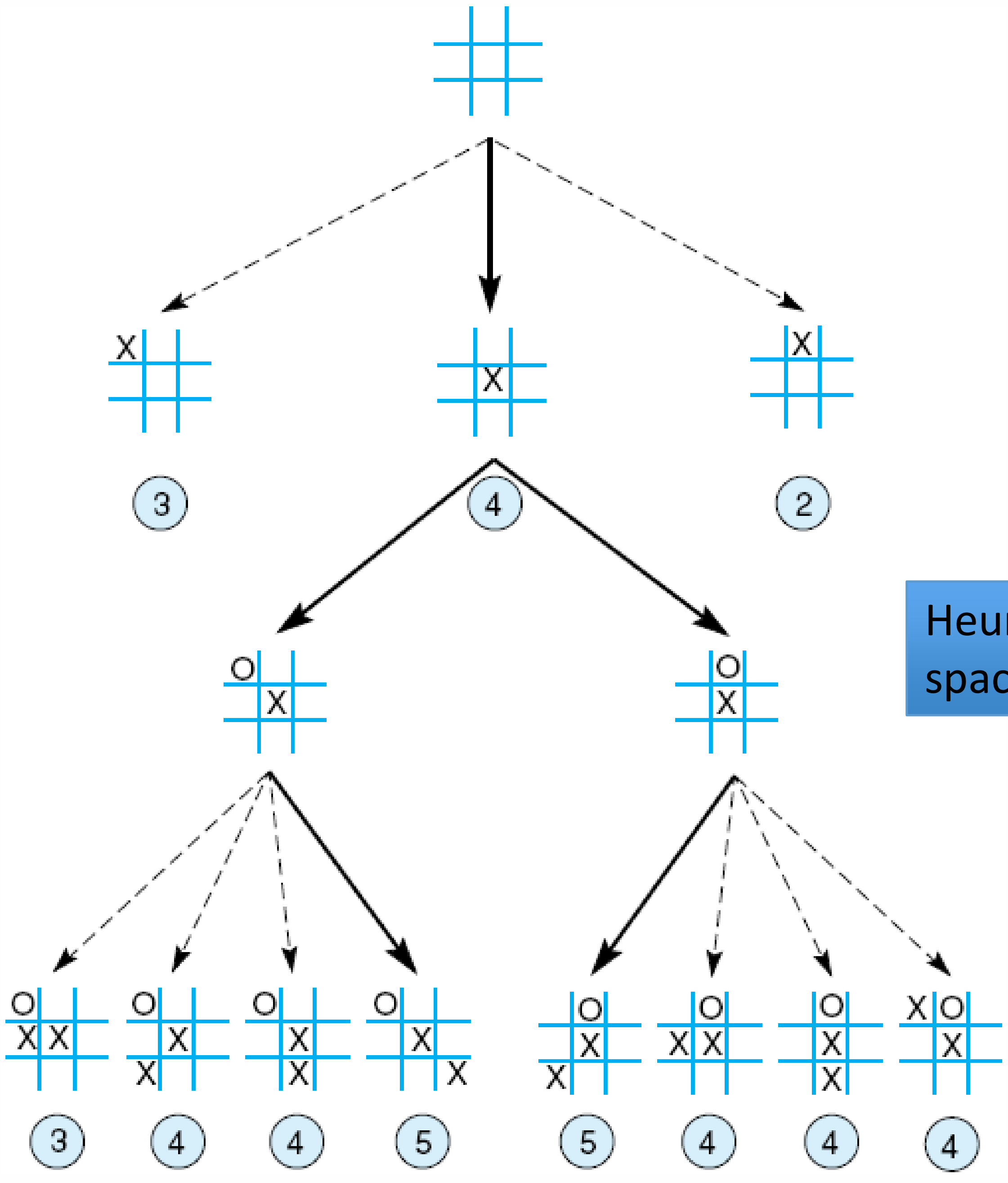
First three levels of the tic-tac-toe state space reduced by symmetry



HEURISTIC FUNCTION



The “most wins” heuristic applied to the first children in tic-tac-toe.



Heuristically reduced state space for tic-tac-toe

GREEDY BEST-FIRST SEARCH

- **Greedy best-first search** tries to expand the node that is closest 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; that is, $f(n) = h(n)$.

- "greedy"—at each step it tries to get as close to the goal as it can.

- Its search cost is minimal. It is not optimal.

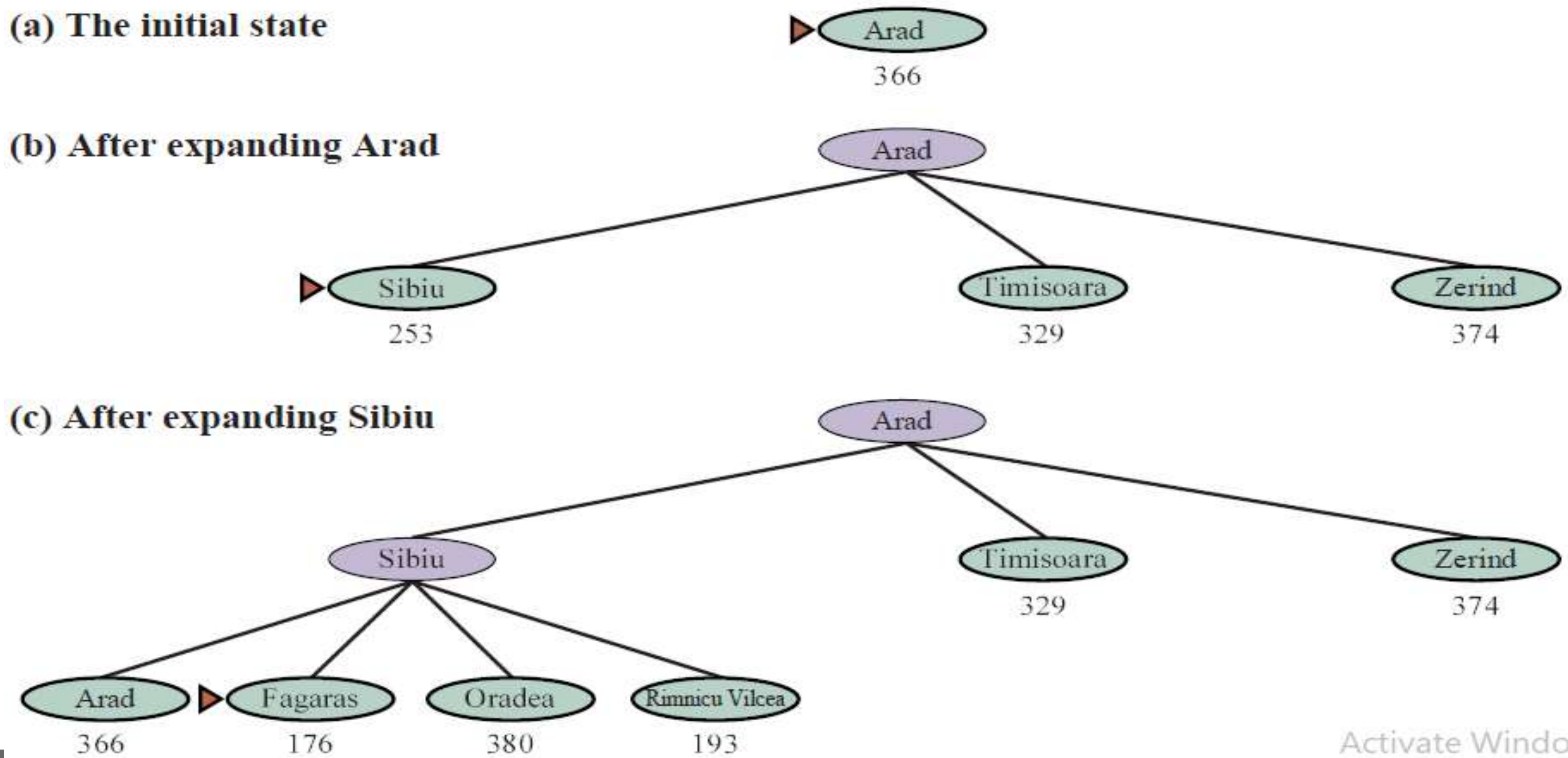
- Greedy best-first tree search is also incomplete even in a finite state space, much like depth-first search.

STRAIGHT-LINE DISTANCE HEURISTIC

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

GREEDY BEST-FIRST TREE SEARCH



GREEDY BEST-FIRST TREE SEARCH

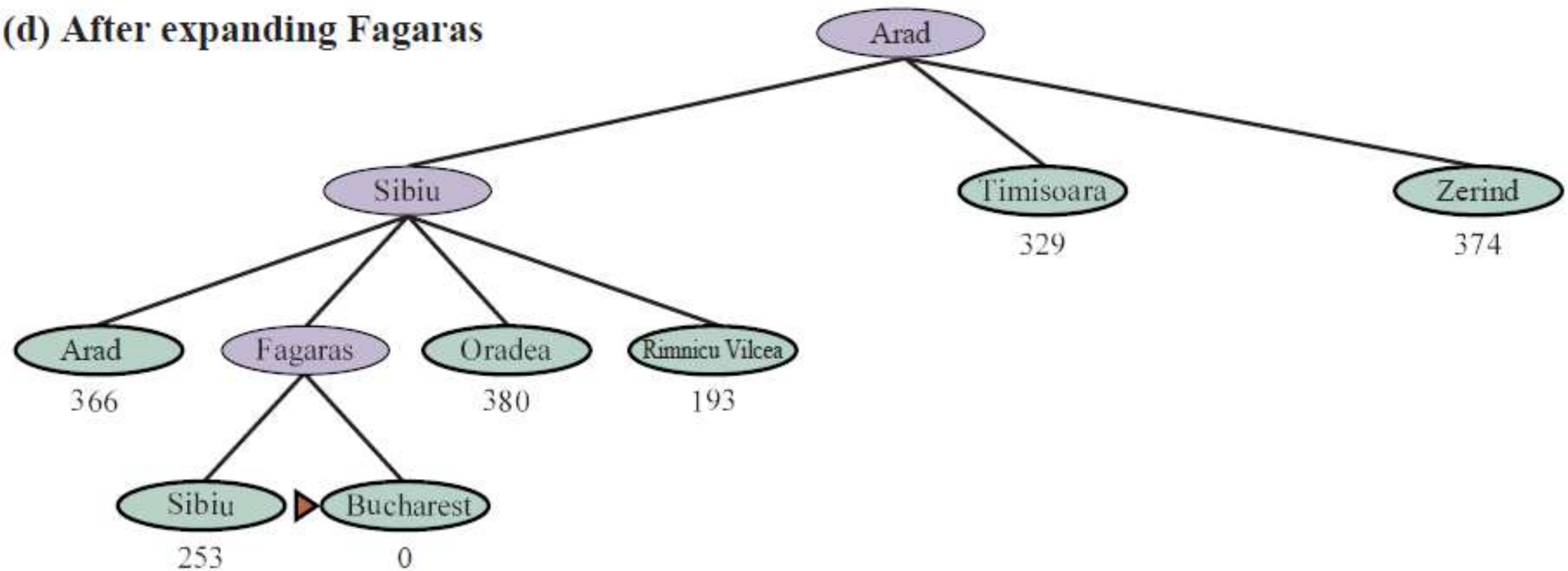


Figure 3.17 Stages in a greedy best-first tree-like search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

A* SEARCH:

MINIMIZING THE TOTAL ESTIMATED SOLUTION COST

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

It evaluates nodes by combining

g(n), the cost to reach the node, and

h(n), the cost to get from the node to the goal:

$$f(n) = g(n) + h(n) .$$

Since

g(n) gives the path cost from the start node to node n, and

h(n) is the estimated cost of the cheapest path from n to the goal,

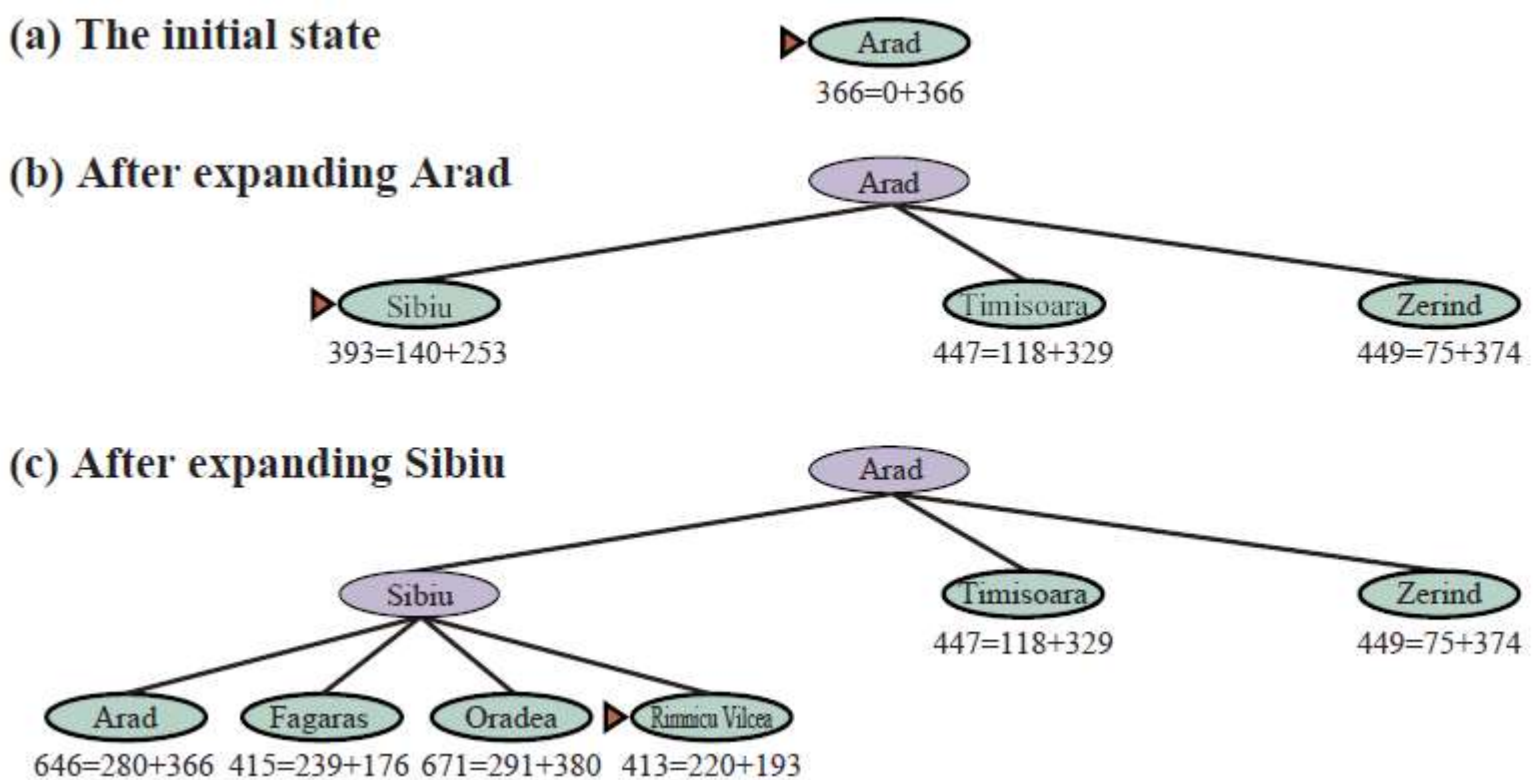
we have f(n) = estimated cost of the cheapest solution through n .

A∗ search is both complete and optimal.

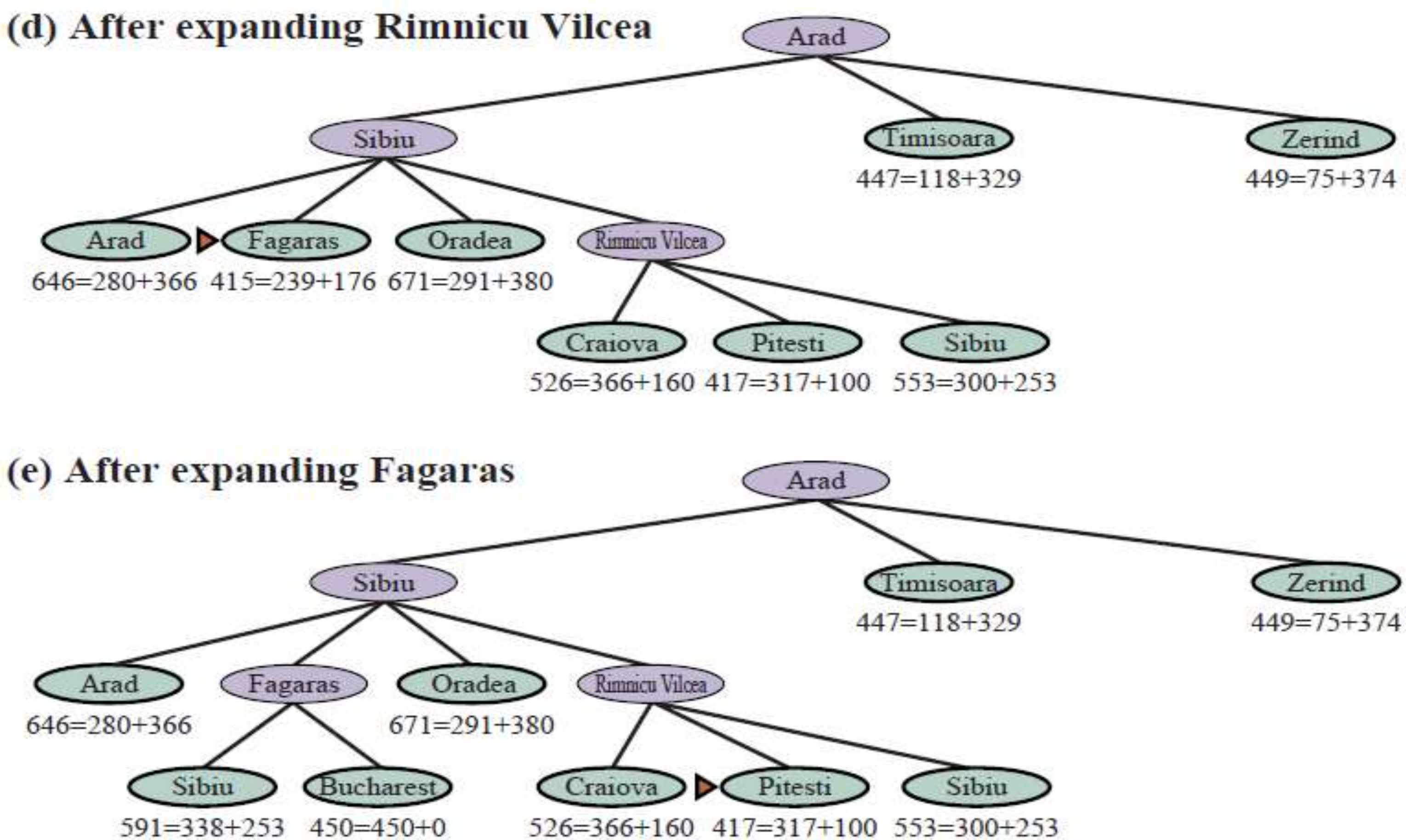
The algorithm is identical to UNIFORM-COST-SEARCH except that

A∗ uses **g + h** instead of **g**.

A* SEARCH FOR BUCHAREST



A∗ SEARCH FOR BUCHAREST



A∗ SEARCH FOR BUCHAREST

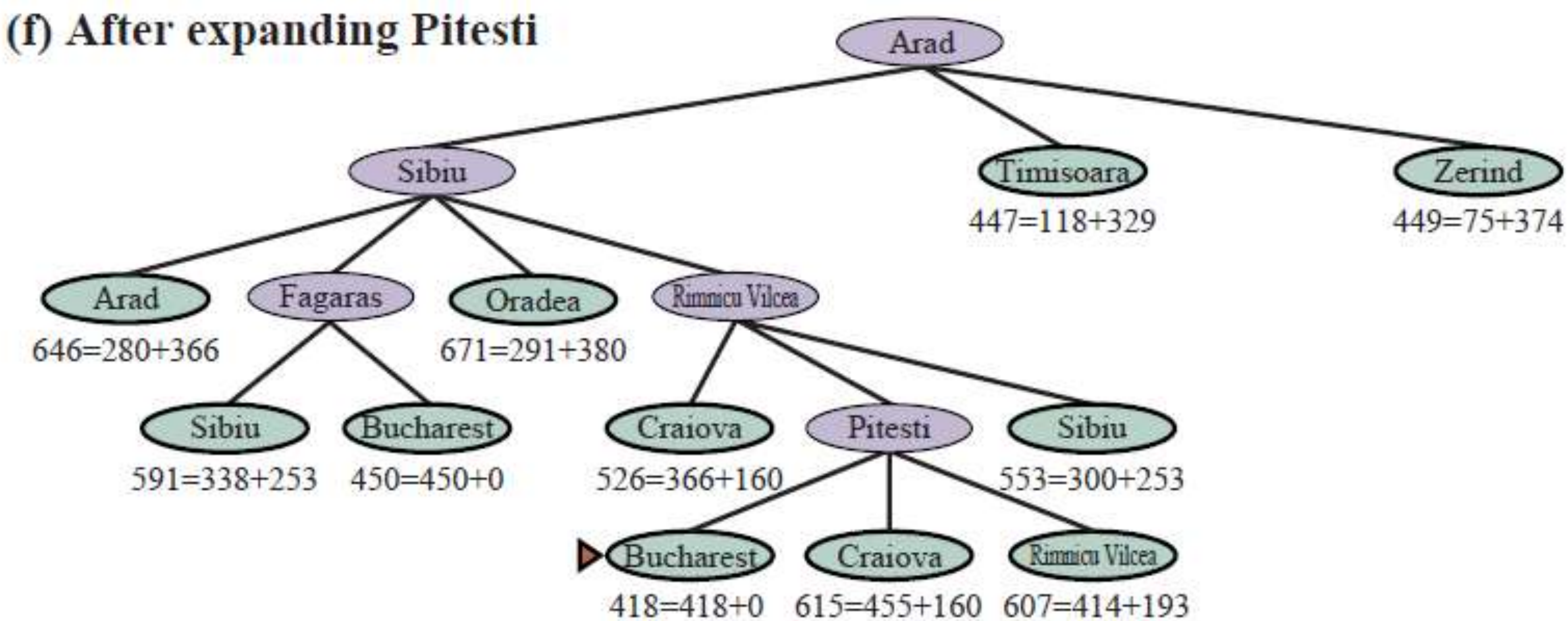


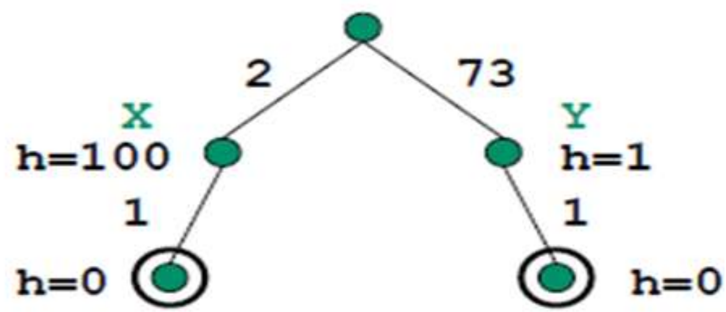
Figure 3.18 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure ??.

CONDITIONS FOR OPTIMALITY: ADMISSIBILITY AND CONSISTENCY

- The first condition we require for optimality is that h(n) be an **admissible heuristic**
- An admissible heuristic is one that *never overestimates* the cost to reach the goal
- Because g(n) is the actual cost to reach n along the current path, and f(n)=g(n) + h(n), we have as an immediate consequence that f(n) never overestimates the true cost of a solution along the current path through n.
- Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.

ADMISSIBILITY

- What must be true about h for A^* to find optimal path?
- A^* finds optimal path if h is admissible; h is admissible when it never overestimates.
- In this example, h is not admissible.
- In route finding problems, straight-line distance to goal is admissible heuristic.



$g(X)+h(X) = 102$

$g(Y)+h(Y) = 74$

Optimal path is not found!

CONSISTENCY

- A second, slightly stronger condition called **consistency** (or sometimes **monotonicity**) is required only for applications of A* to graph search.
- A heuristic h(n) is consistent if, for every node n and every successor n’ of n generated by any 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’ :

$$\mathbf{h(n) \leq c(n, a, n') + h(n')}.$$

- This is a form of the general **triangle inequality**, which stipulates that each side of a triangle cannot be longer than the sum of the other two sides

OPTIMALITY OF A*

A∗ has the following properties: *the tree-search version of* A* *is optimal if* h(n) *is admissible, while the graph-search version is optimal if* h(n) *is consistent.*

if h(n) *is consistent, then the values of* f(n) *along any path are nondecreasing.*
The proof follows directly from the definition of consistency. Suppose n is a successor of n; then g(n')=g(n) + c(n, a, n') for some action a, and we have

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

= g(n) + c(n, a, n') + h(n') because g(n') = g(n) + c(n, a, n')

≥ g(n) + h(n) because c(n, a, n') + h(n') >= h(n)

= f(n)

So, f(n') >= f(n)

OPTIMALITY OF A*

The next step is to prove that *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 frontier node n' on the optimal path from the start node to n; because f is nondecreasing along any path, n' would have lower f-cost than n and would have been selected first.

Local search algorithms

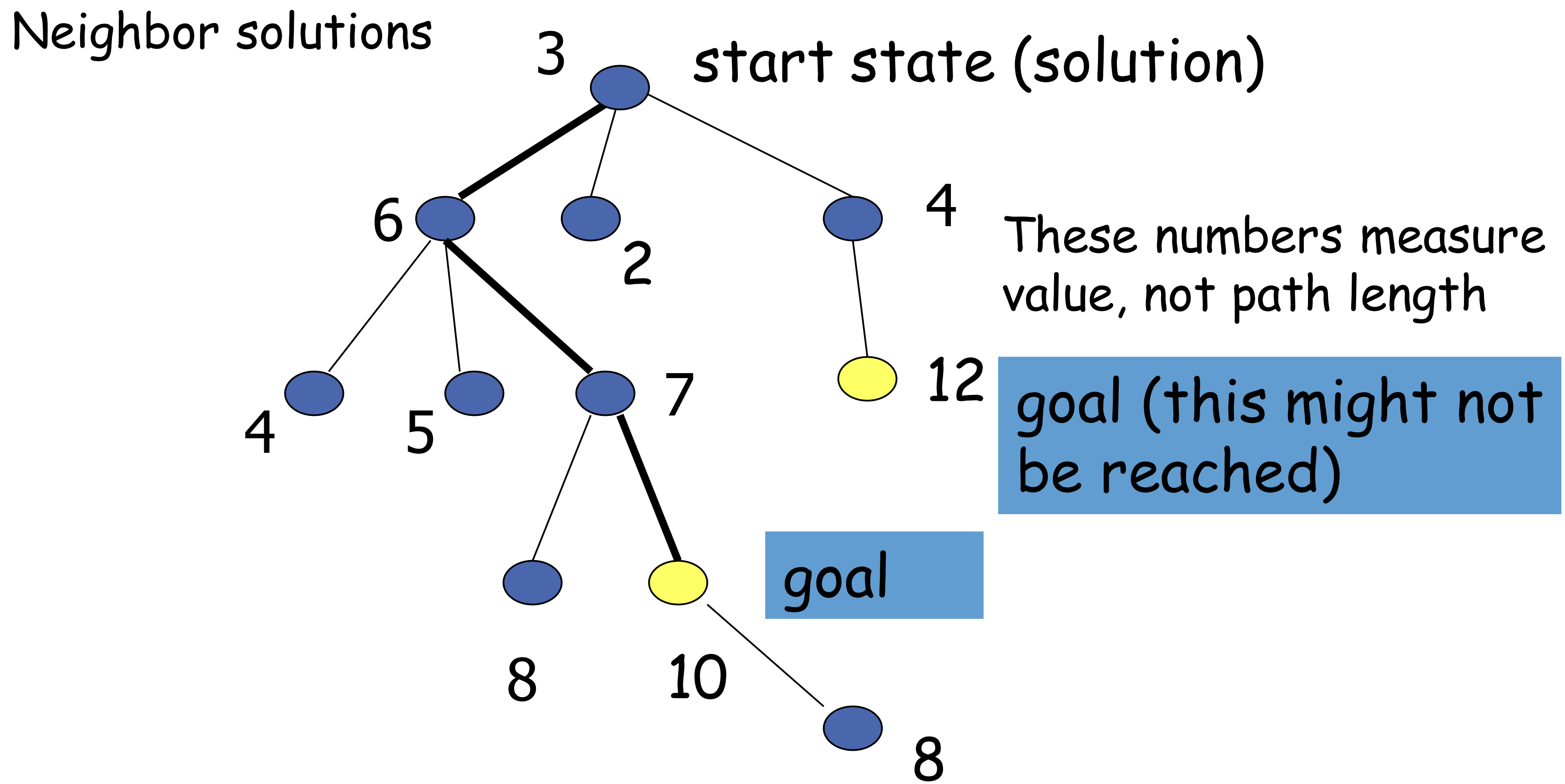
- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

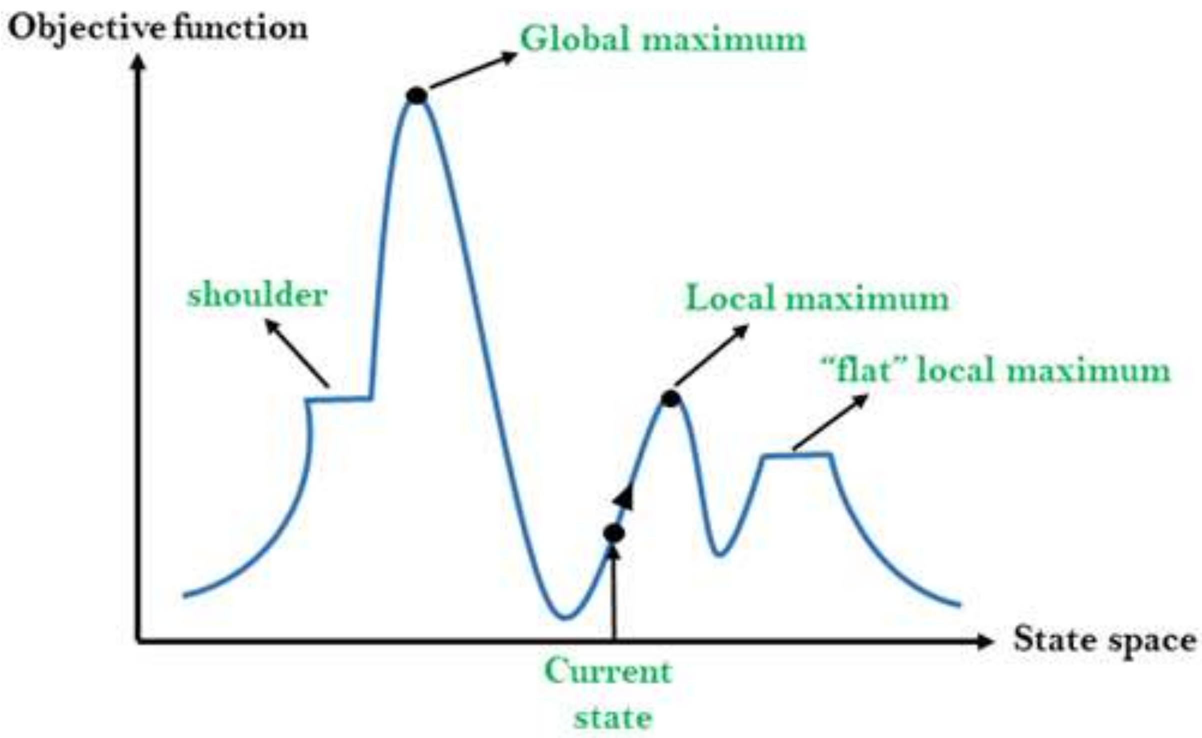
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Example of hill-climbing



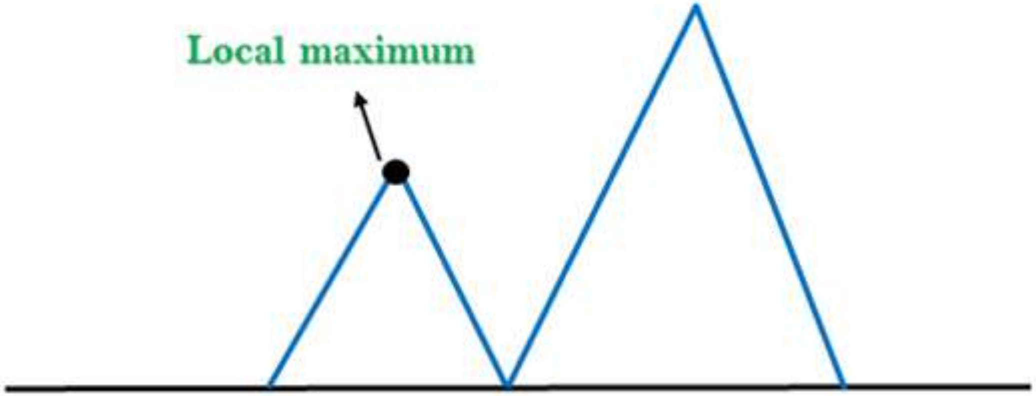
Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima



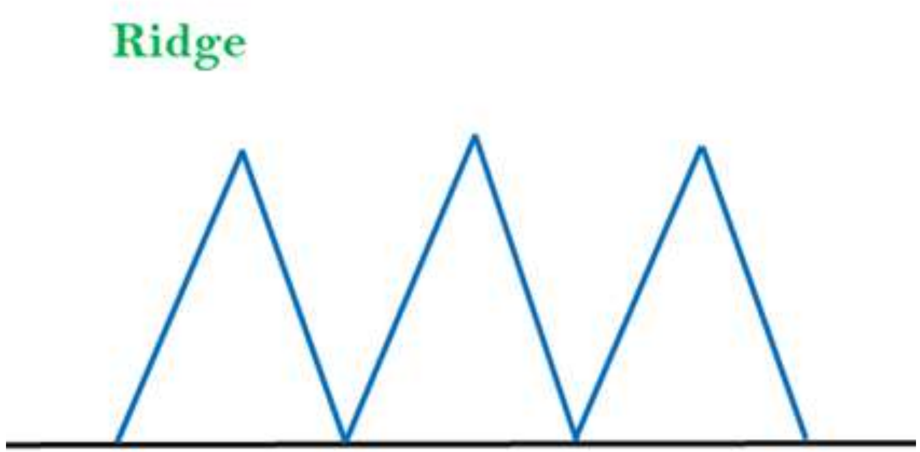
Hill-climbing search

- **Local maximum problem**



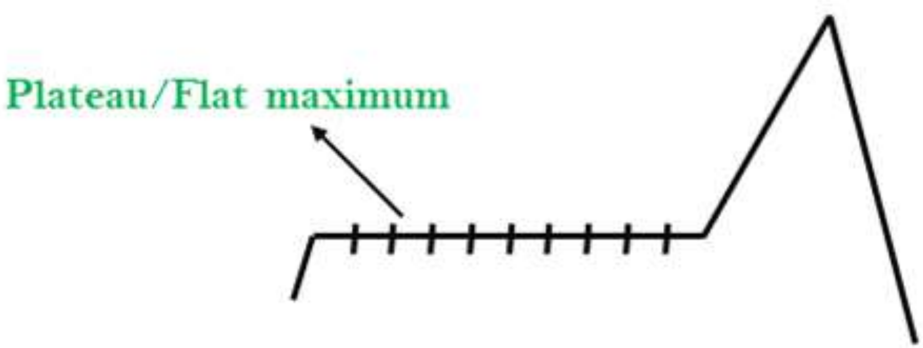
Hill-climbing search

- **Ridge:** Result in a sequence of local maxima that is very difficult for a greedy algorithms to navigate.
- **Ridge:** then jump a little bit.



Hill-climbing search

- **Plateau:** The evaluation function is flat. It can be a flat local maximum, from which no uphill exit exists or a shoulder, from which it is possible to make progress.
- **Plateau:** may be side way move. However, may not be a sol^n when no uphill exists. So need some limitation of steeping sideway.



References

1. Chapter 3: Solving Problem by Searching , Pages 92-97
"Artificial Intelligence: A Modern Approach," by Stuart J. Russell and Peter Norvig,

Books

1. “Artificial Intelligence: A Modern Approach,” by Stuart J. Russell and Peter Norvig.
2. "Artificial Intelligence: Structures and Strategies for Complex Problem Solving", by George F. Luger, (2002)
3. "Artificial Intelligence: Theory and Practice", by Thomas Dean.
4. "AI: A New Synthesis", by Nils J. Nilsson.
5. “Programming for machine learning,” by J. Ross Quinlan,
6. “Neural Computing Theory and Practice,” by Philip D. Wasserman, .
7. “Neural Network Design,” by Martin T. Hagan, Howard B. Demuth, Mark H. Beale, .
8. “Practical Genetic Algorithms,” by Randy L. Haupt and Sue Ellen Haupt.
9. “Genetic Algorithms in Search, optimization and Machine learning,” by David E. Goldberg.
10. "Computational Intelligence: A Logical Approach", by David Poole, Alan Mackworth, and Randy Goebel.
11. “Introduction to Turbo Prolog”, by Carl Townsend.