

Set 3: Informed Heuristic Search

ICS 271 Fall 2016

Kalev Kask

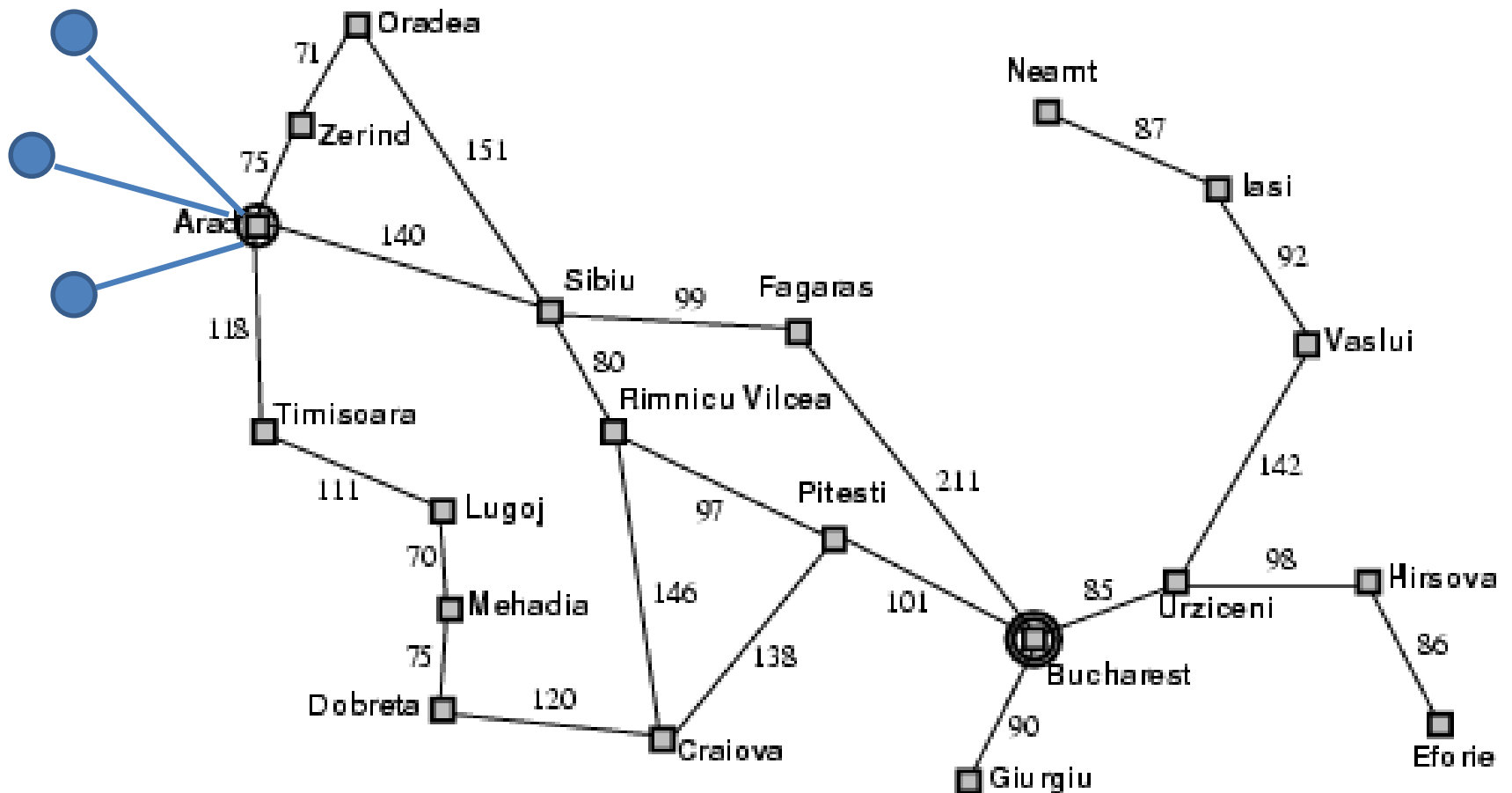
Basic search scheme

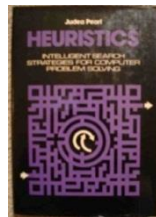
- We have 3 kinds of states
 - explored (past) – **only graph search**
 - frontier (current)
 - unexplored (future) – implicitly given
- Initially frontier=start state
- Loop until found solution or exhausted state space
 - pick/remove first node from frontier using search strategy
 - priority queue – FIFO (BFS), LIFO (DFS), g (UCS), f (A*), etc.
 - check if goal
 - add this node to explored,
 - expand this node, add children to frontier (**graph search : only those children whose state is not in explored list**)
 - Q: what if better path is found to a node already on explored list?

Overview

- Heuristics and Optimal search strategies (3.5-3.6)
 - heuristics
 - hill-climbing algorithms
 - Best-First search
 - A*: optimal search using heuristics
 - Properties of A*
 - admissibility,
 - consistency,
 - accuracy and dominance
 - Optimal efficiency of A*
 - Branch and Bound
 - Iterative deepening A*
 - Power/effectiveness of different heuristics
 - Automatic generation of heuristics

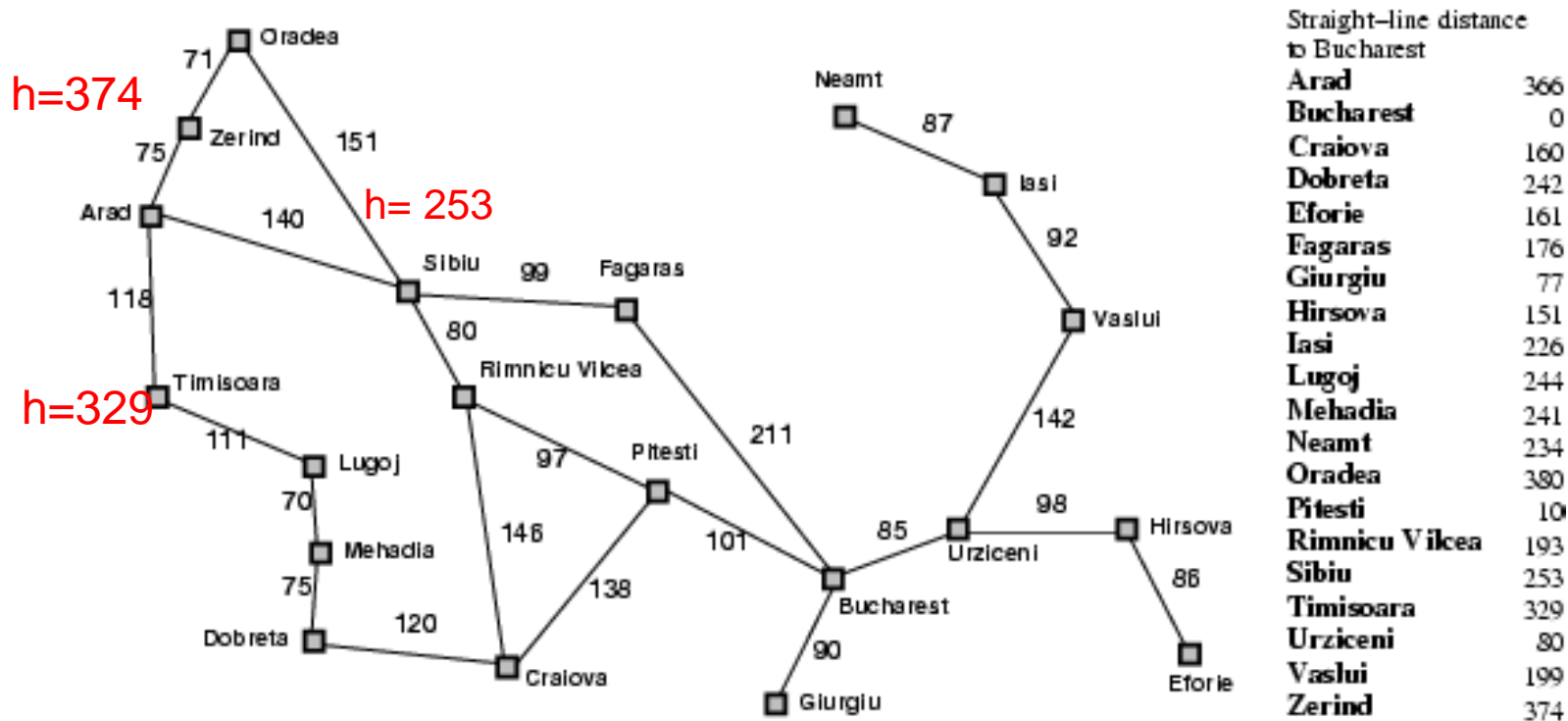
What is a heuristic?





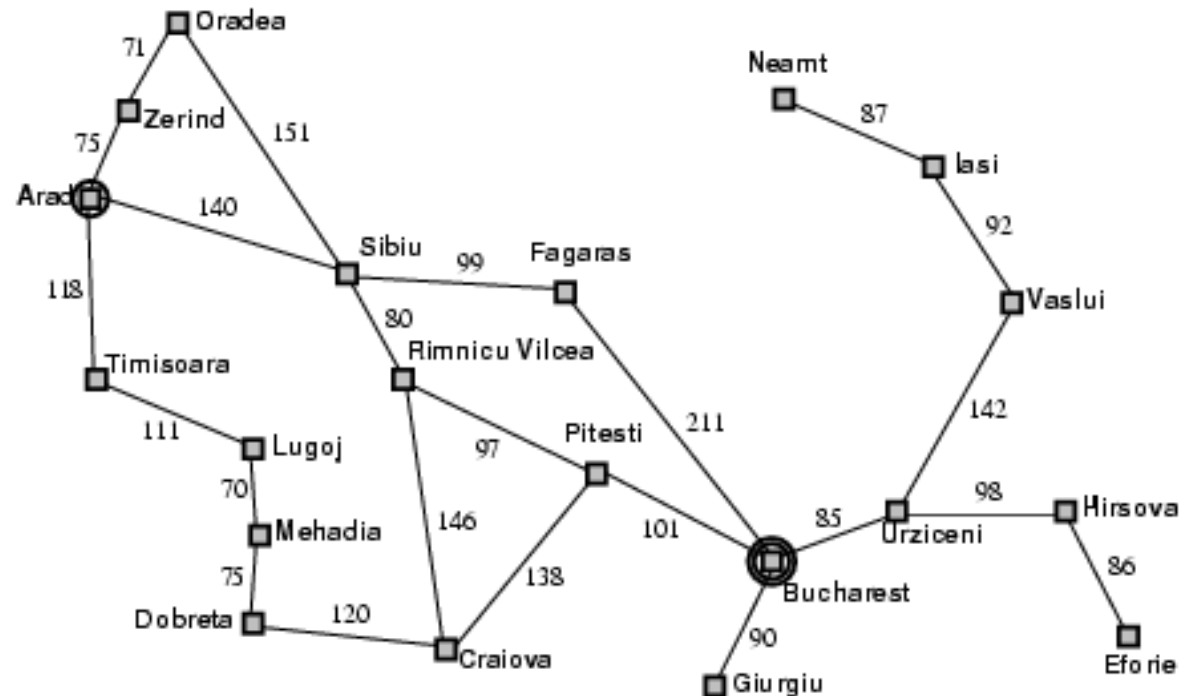
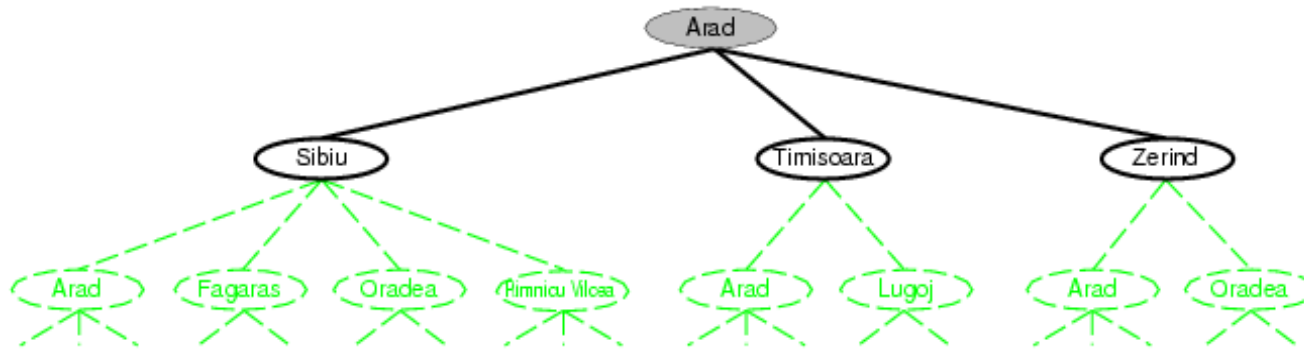
Heuristic Search

- State-Space Search: every problem is like search of a map
- A problem solving agent finds a **path** in a **state-space graph** from **start state** to **goal state**, using **heuristics**

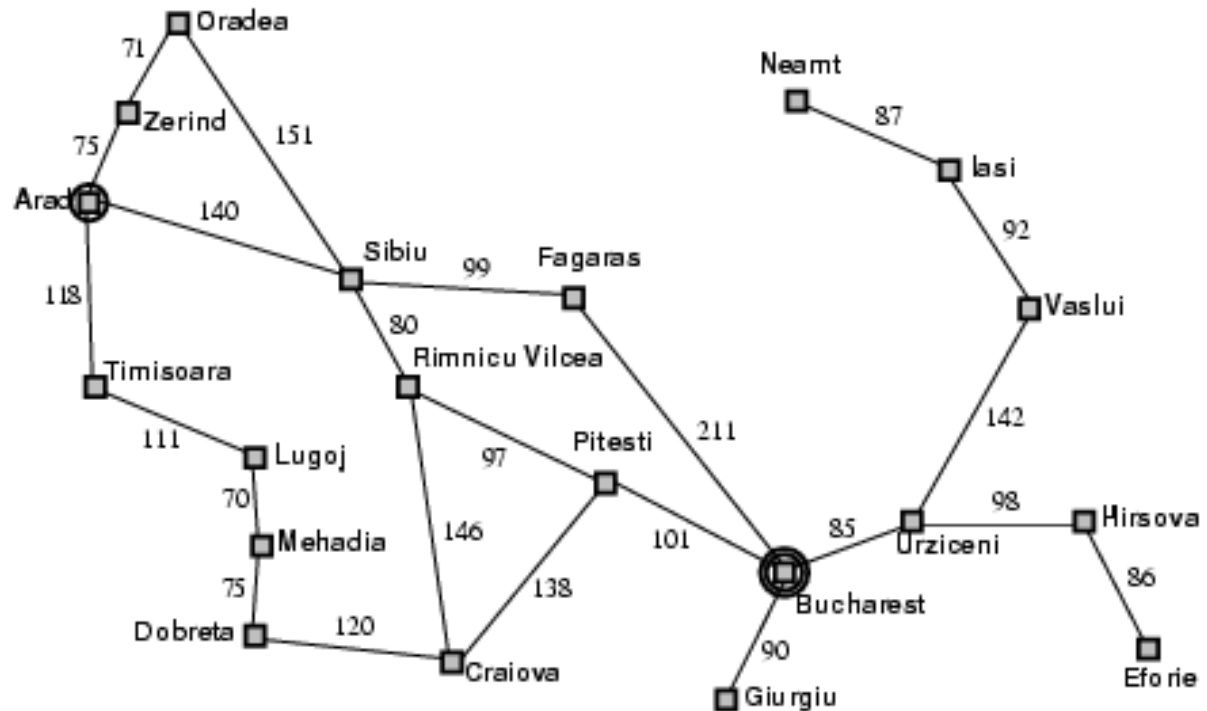
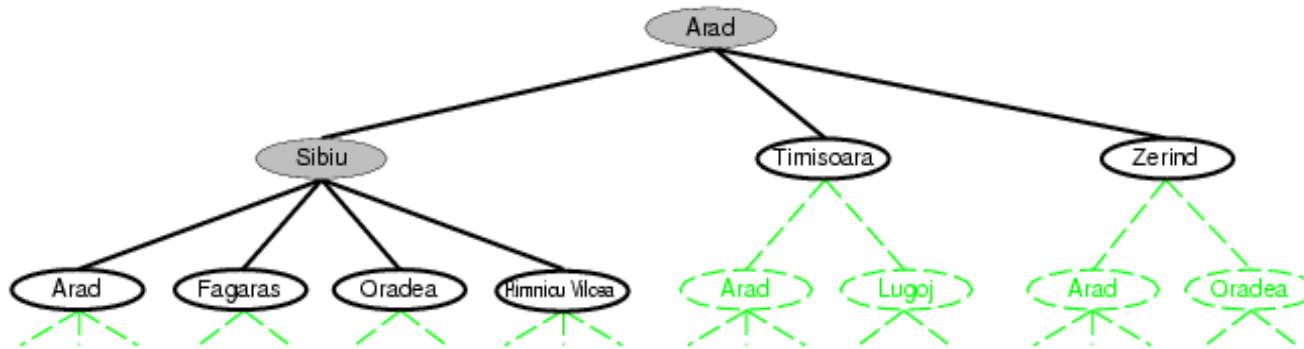


Heuristic = straight-line distance

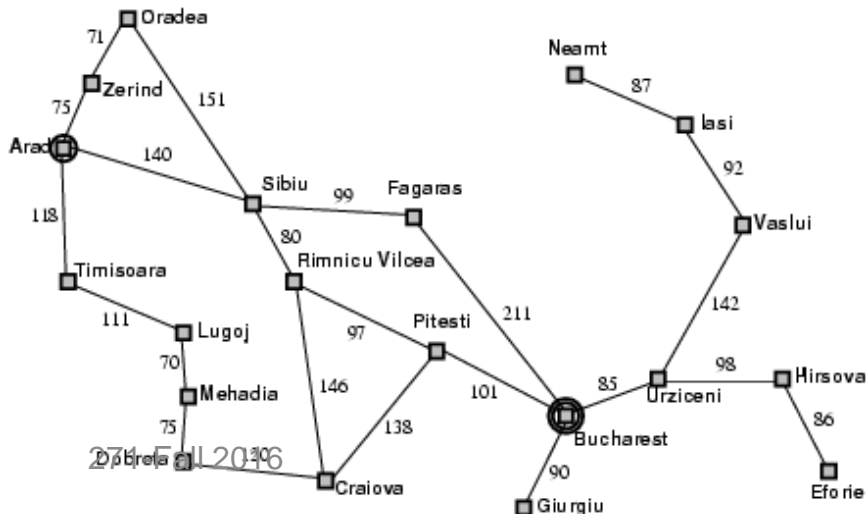
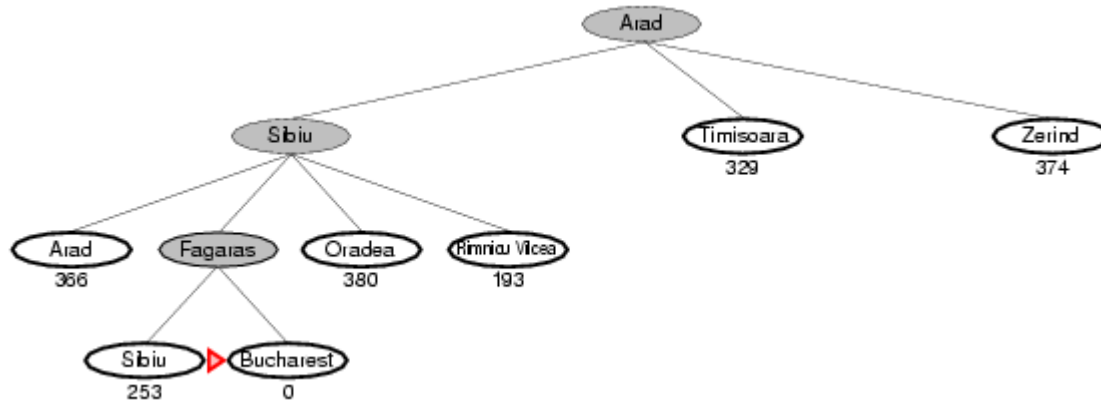
State Space for Path Finding in a Map



State Space for Path Finding on a Map



Greedy Search Example



State Space of the 8 Puzzle Problem

8-puzzle: 181,440 states
 15-puzzle: 1.3 trillion
 24-puzzle: 10^{25}

Search space exponential

Use Heuristics
 as people do

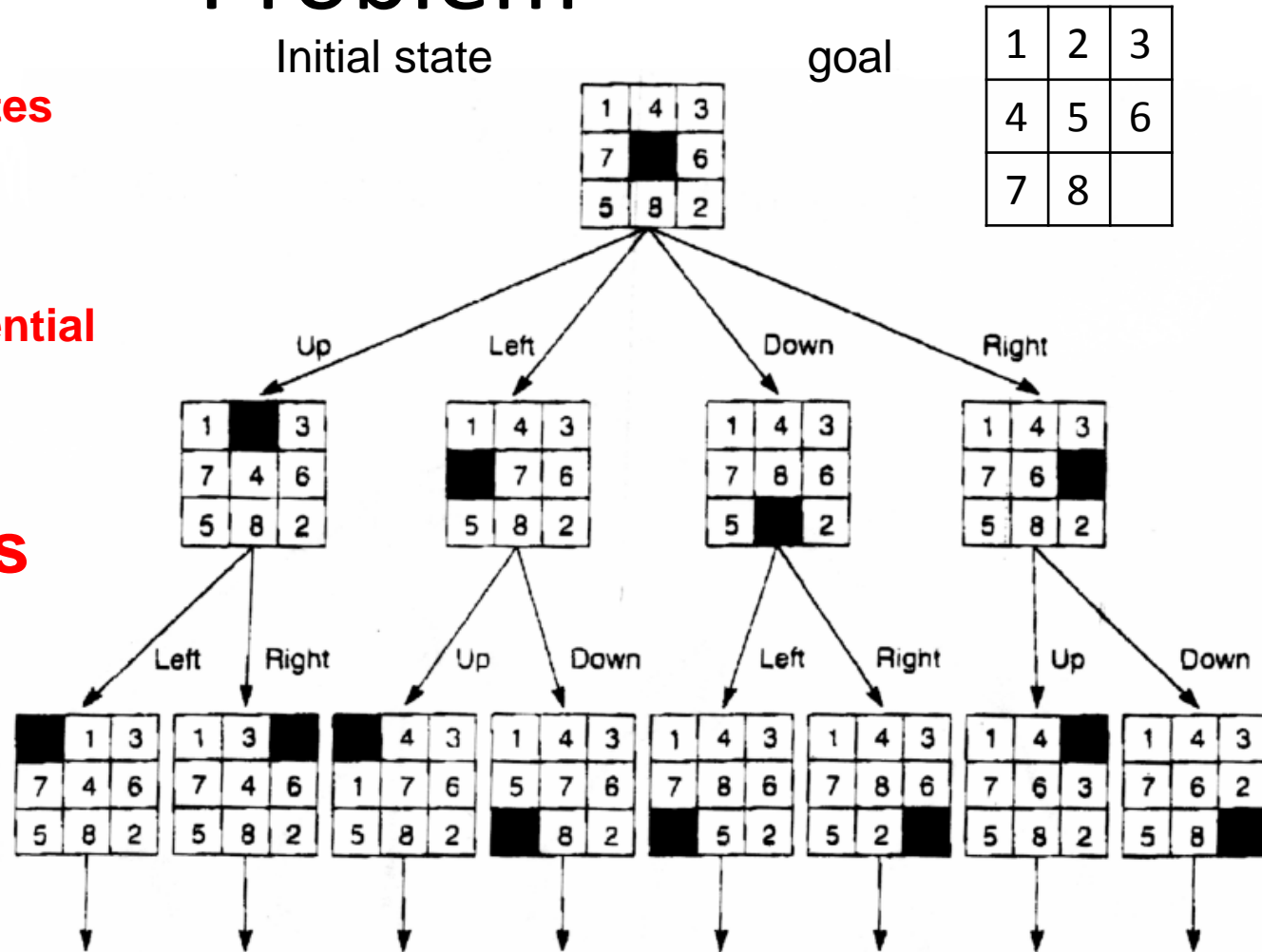


Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

State Space of the 8 Puzzle Problem

1	2	3
4	5	6
7	8	

h1 = number of misplaced tiles

h2 = Manhattan distance

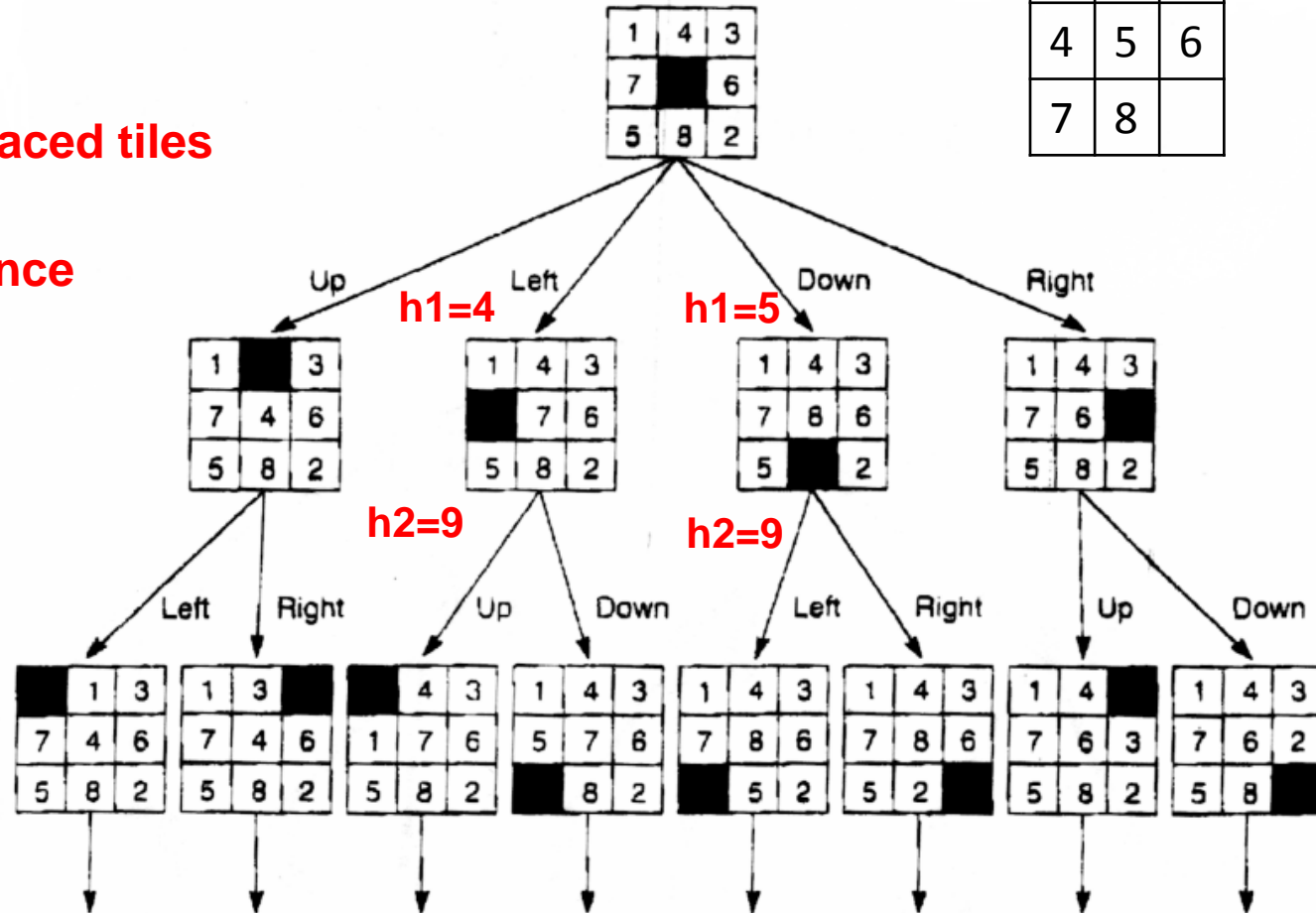
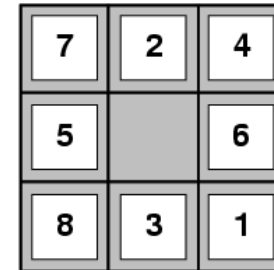


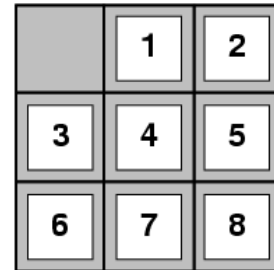
Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

What are Heuristics

- Rule of thumb, intuition
- A **quick way** to estimate how close we are to the goal. How close is a state to the goal..
- Pearl: “the ever-amazing observation of how much people can accomplish with that simplistic, unreliable information source known as *intuition*.”



Start State



Goal State

8-puzzle

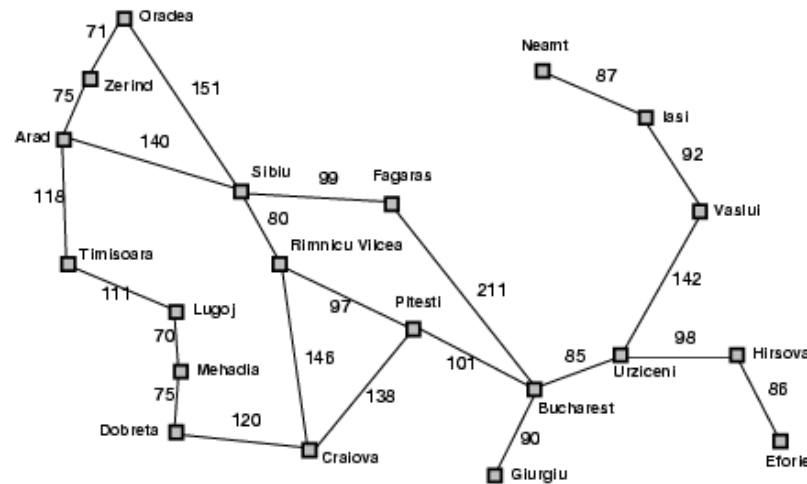
- $h_1(n)$: number of misplaced tiles
- $h_2(n)$: Manhattan distance
- $h_3(n)$: Gaschnig’s

$$h_1(S) = ? \quad 8$$

$$h_2(S) = ? \quad 3+1+2+2+2+3+3+2 = 18$$

$$h_3(S) = ? \quad 8$$

- Path-finding on a map
 - Euclidean distance



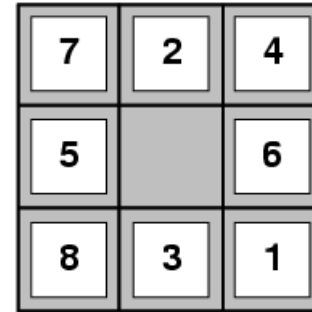
Problem: Finding a Minimum Cost Path

- Previously we wanted an path with minimum number of steps. Now, we want the minimum cost path to a goal G
 - Cost of a path = sum of individual steps along the path
- Examples of path-cost:
 - Navigation
 - path-cost = distance to node in miles
 - minimum => minimum time, least fuel
 - VLSI Design
 - path-cost = length of wires between chips
 - minimum => least clock/signal delay
 - 8-Puzzle
 - path-cost = number of pieces moved
 - minimum => least time to solve the puzzle
- Algorithm: Uniform-cost search ... still somewhat blind

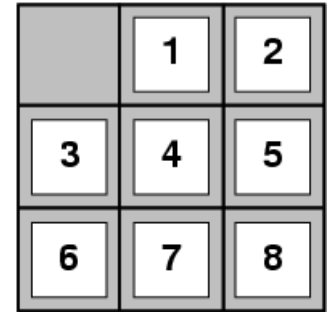
Heuristic Functions

- 8-puzzle

- Number of misplaced tiles
- Manhattan distance
- Gaschnig's



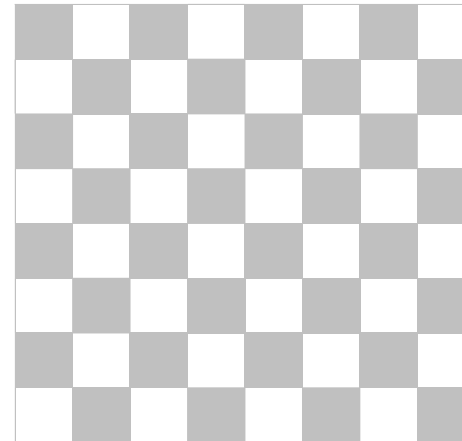
Start State



Goal State

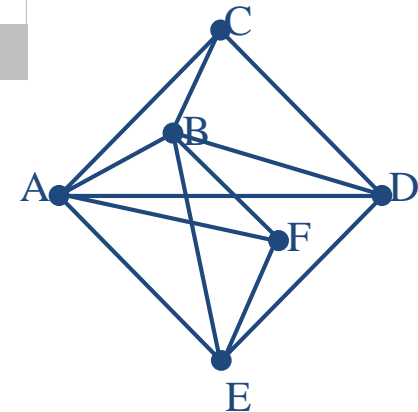
- 8-queen

- Number of future feasible slots
- Min number of feasible slots in a row
- Min number of conflicts (in complete assignments states)



- Travelling salesperson

- Minimum spanning tree
- Minimum assignment problem

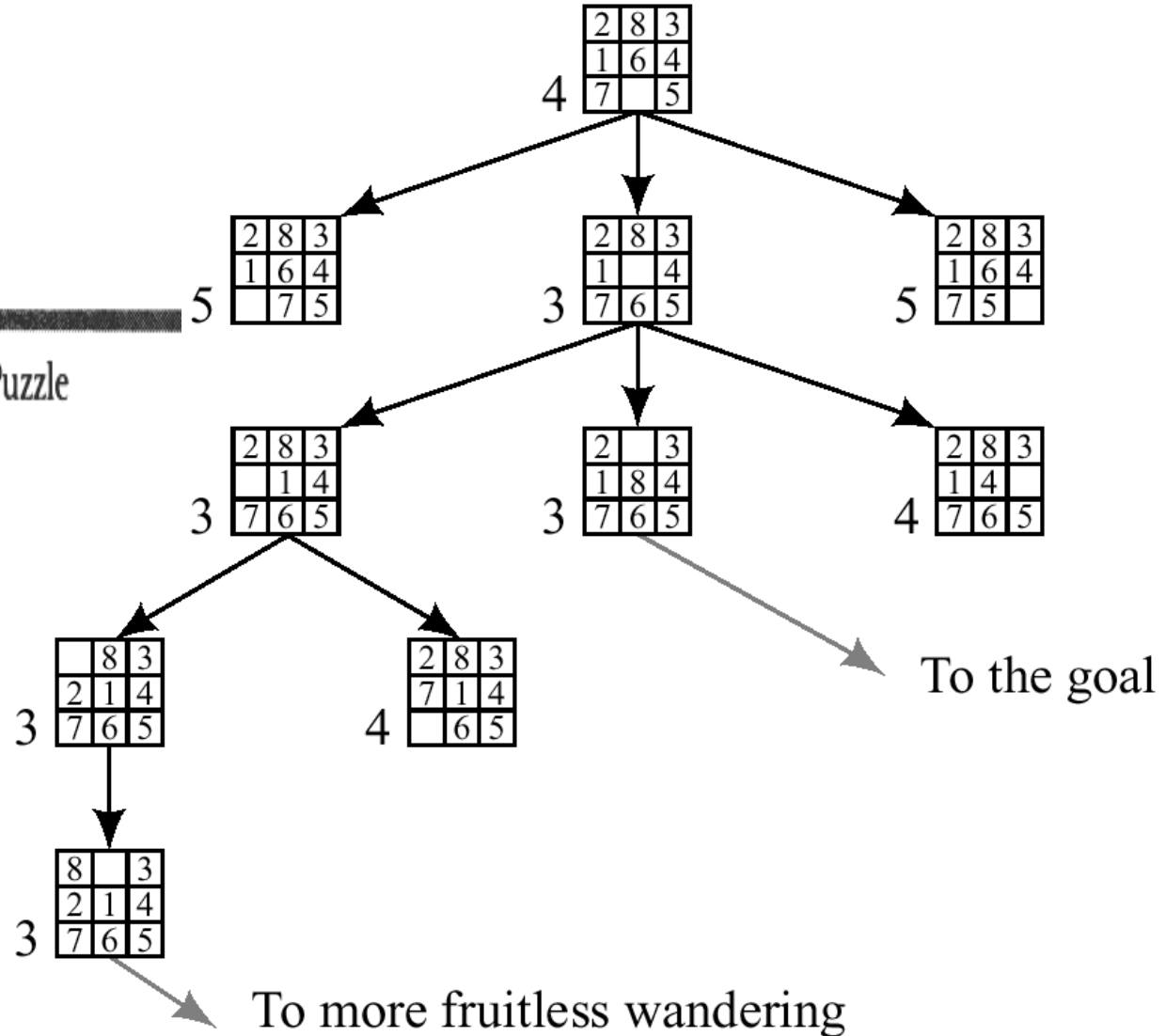


Best-First (Greedy) Search: $f(n)$ = number of misplaced tiles



Figure 8.1

Start and Goal Configurations for the Eight-Puzzle



Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (**h**euristic)
= estimate of cost from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that **appears** to be closest to goal

Greedy Best-First Search Example



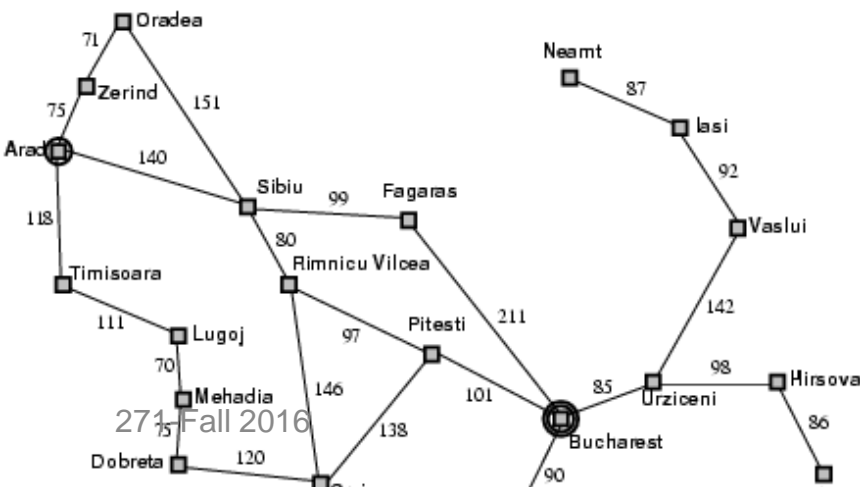
	Straight-line distance to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search Example

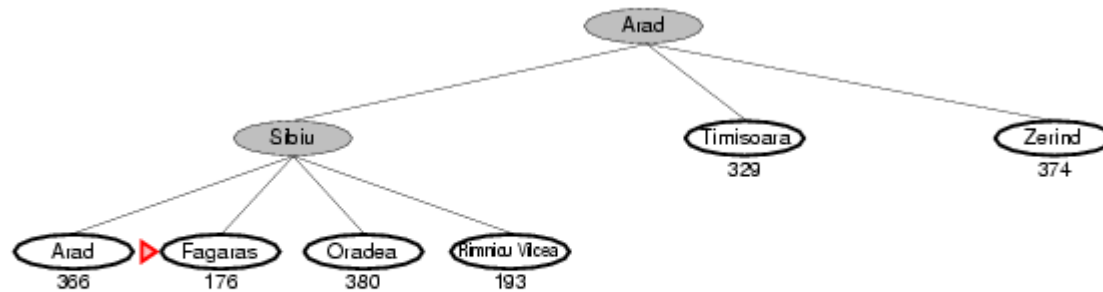


Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

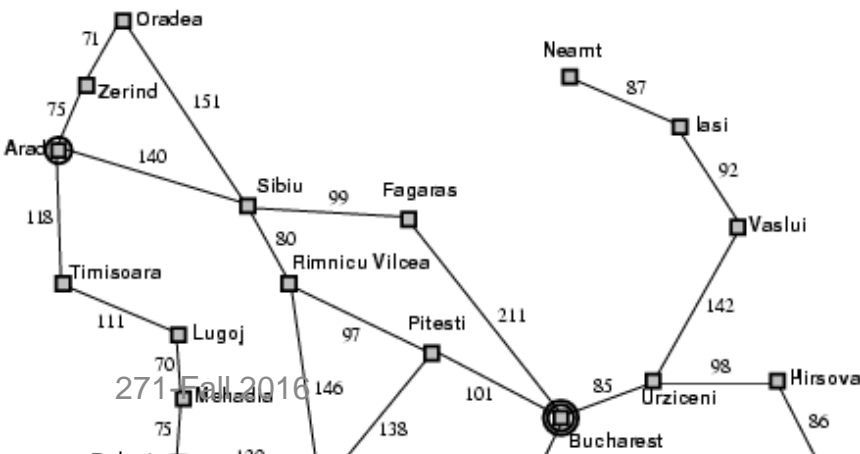


Greedy Best-First Search Example

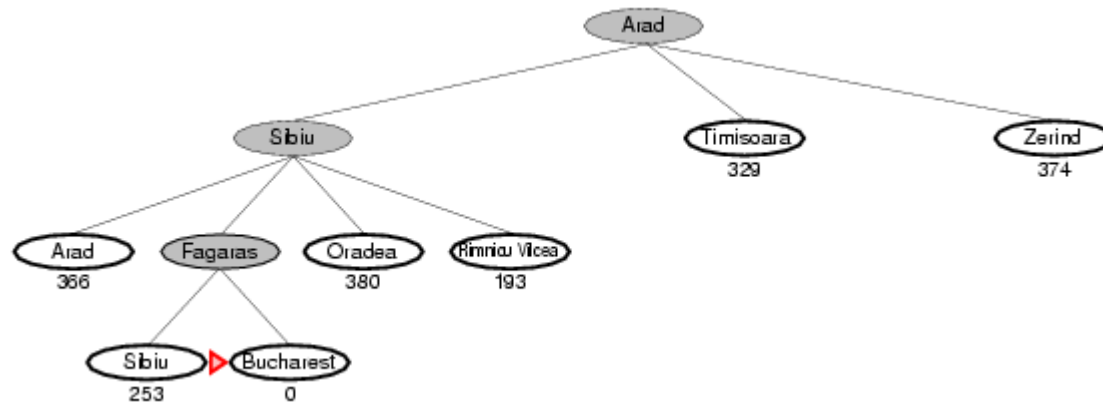


Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

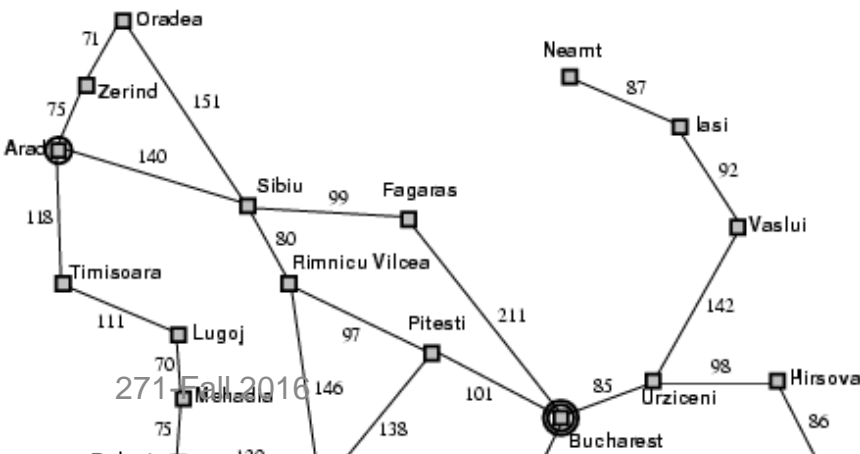


Greedy Best-First Search Example



Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Problems with Greedy Search

- Not complete
 - Gets stuck on local minimas and plateaus
- Infinite loops
- Irrevocable
- Not optimal
- Can we incorporate heuristics in systematic search?

Informed Search - Heuristic Search

- How to use heuristic knowledge in systematic search?
- Where? (in node expansion? hill-climbing ?)
- Best-first:
 - select the best from **all** the nodes encountered so far in OPEN.
 - “good” use heuristics
- Heuristic estimates value of a node
 - promise of a node
 - difficulty of solving the subproblem
 - quality of solution represented by node
 - the amount of information gained.
- **$f(n)$ - heuristic evaluation function.**
 - depends on n , goal, search so far, domain

Best-First Algorithm *BF* (*)

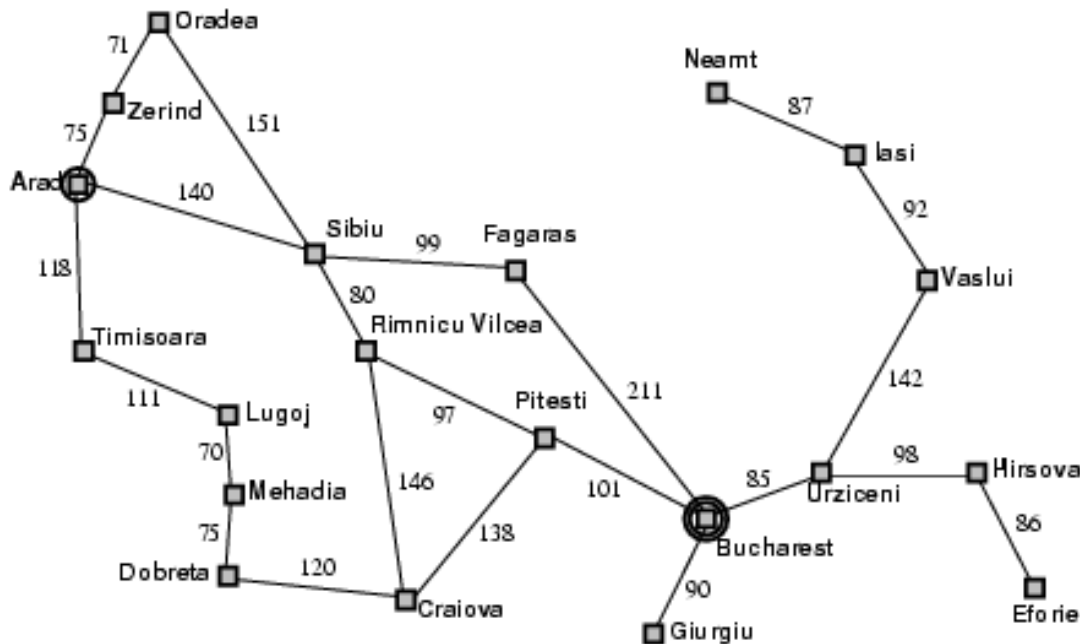
1. Put the start node s on a list called *OPEN* of unexpanded nodes.
 2. If *OPEN* is empty exit with failure; no solutions exists.
 3. Remove the first *OPEN* node n at which f is minimum (break ties arbitrarily), and place it on a list called *CLOSED* to be used for expanded nodes.
 4. If n is a goal node, exit successfully with the solution obtained by tracing the path along the pointers from the goal back to s .
 5. Otherwise expand node n , generating all it's successors with pointers back to n .
 6. For every successor n' on n :
 - a. Calculate $f(n')$.
 - b. if n' was neither on *OPEN* nor on *CLOSED*, add it to *OPEN*. Attach a pointer from n' back to n . Assign the newly computed $f(n')$ to node n' .
 - c. if n' already resided on *OPEN* or *CLOSED*, compare the newly computed $f(n')$ with the value previously assigned to n' . If the old value is lower, discard the newly generated node. If the new value is lower, substitute it for the old (n' now points back to n instead of to its previous predecessor). If the matching node n' resides on *CLOSED*, move it back to *OPEN*.
 7. Go to step 2.
- * With tests for duplicate nodes.

A* Search

- Idea:
 - avoid expanding paths that are already expensive
 - focus on paths that show promise
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

A* Search Example

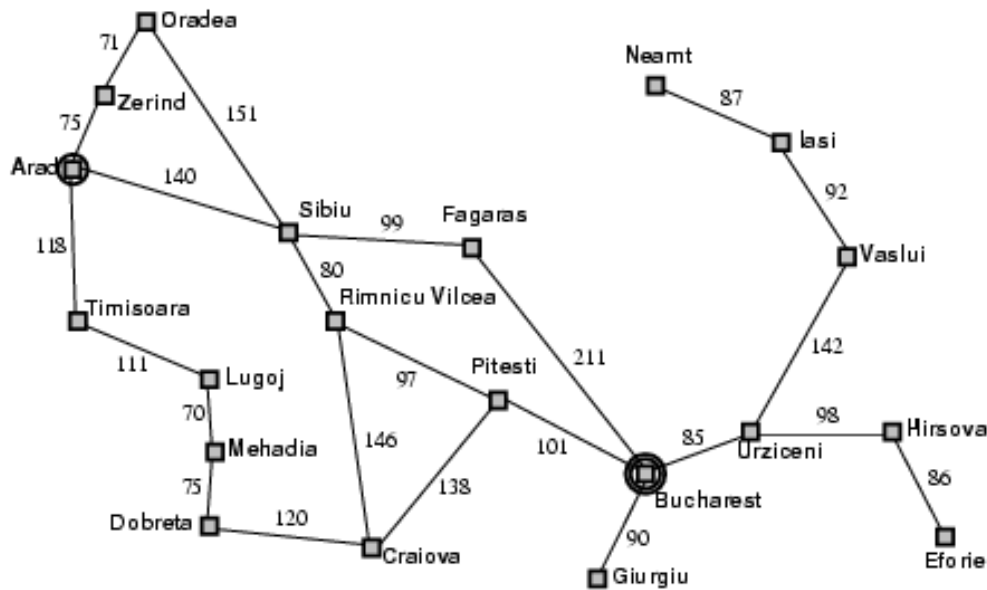
Arad
366=0+366



Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

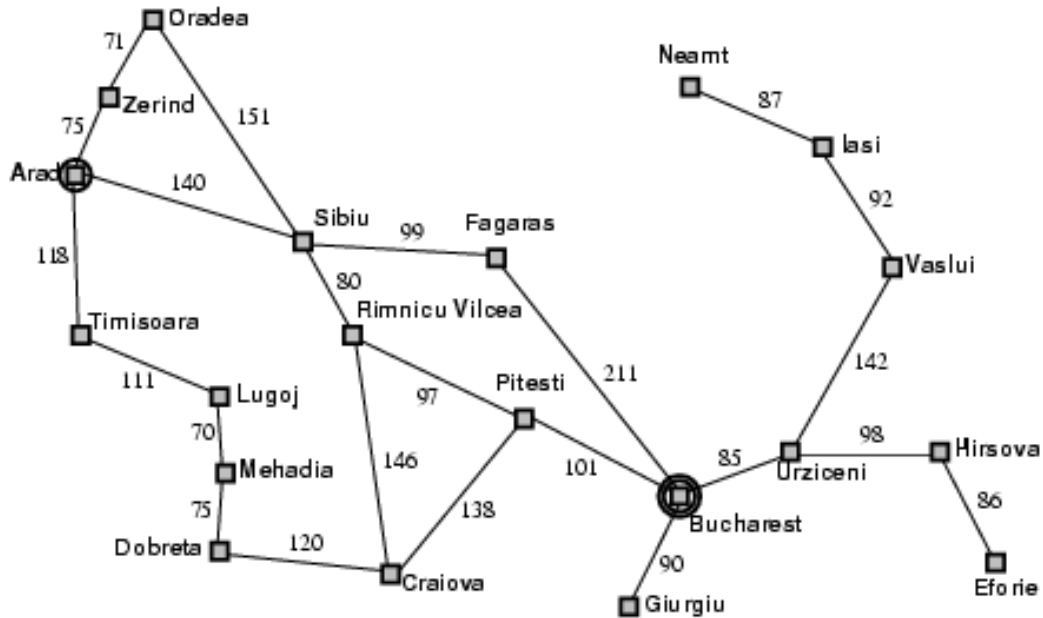
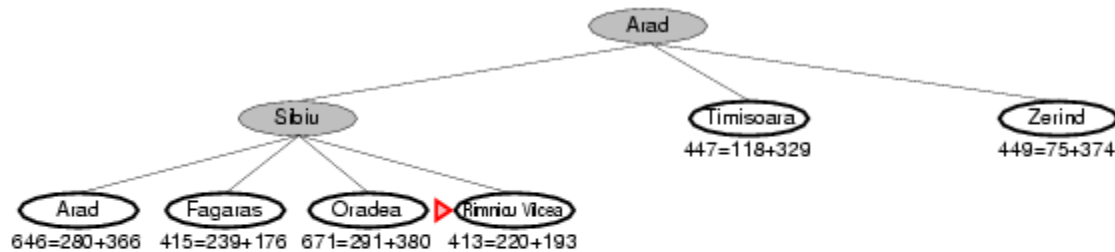
A* Search Example



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

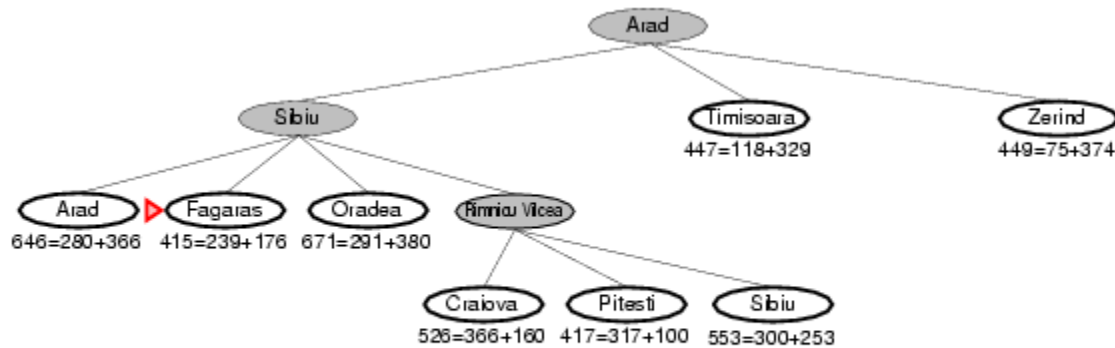
A* Search Example



Straight-line distance to Bucharest

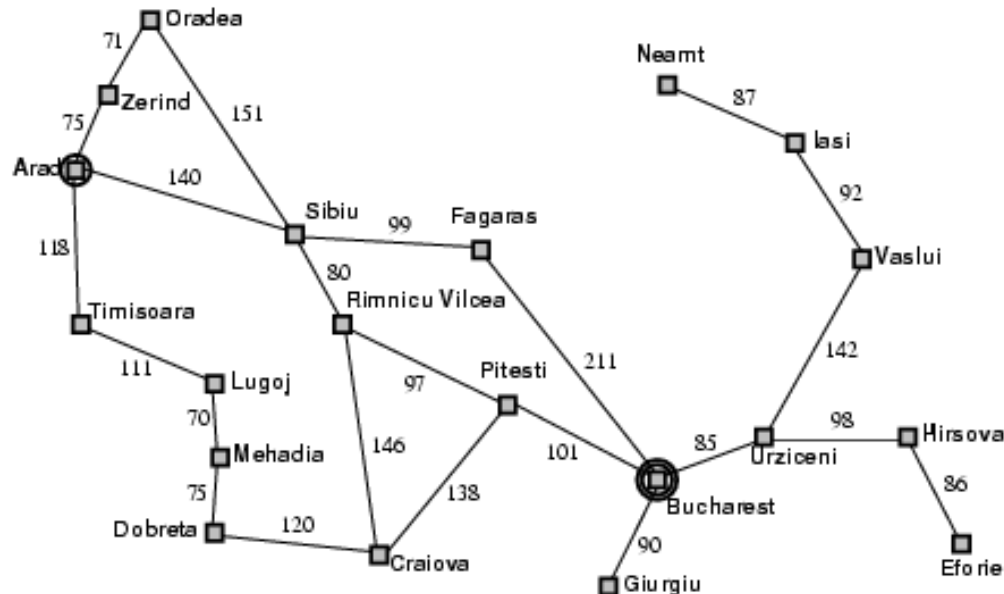
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search Example

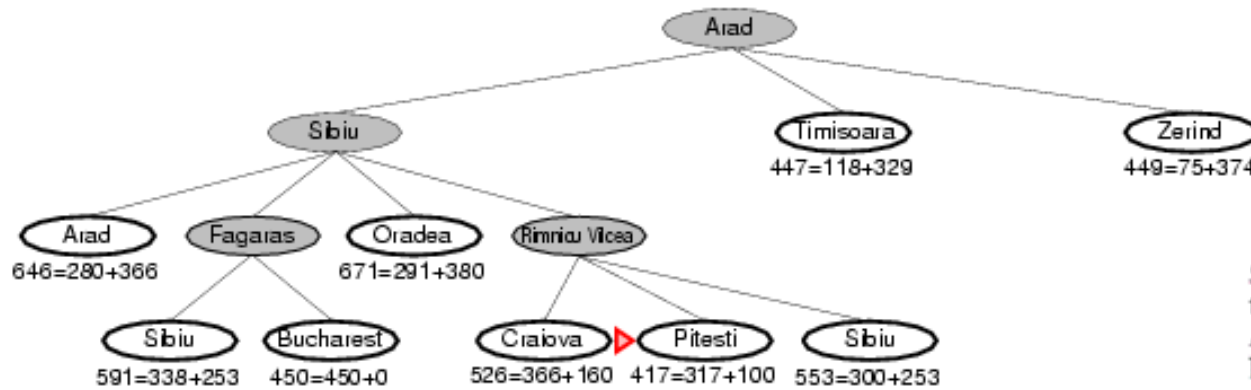


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

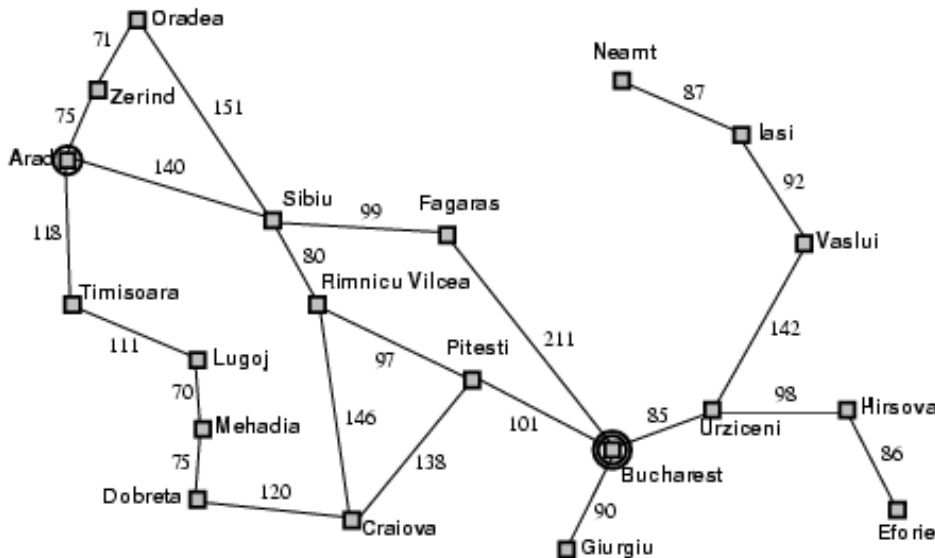


A* Search Example

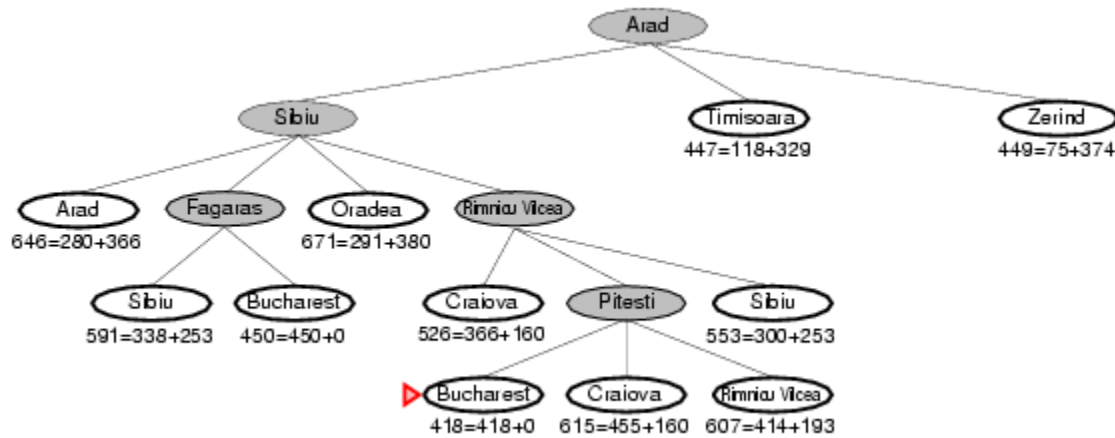


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

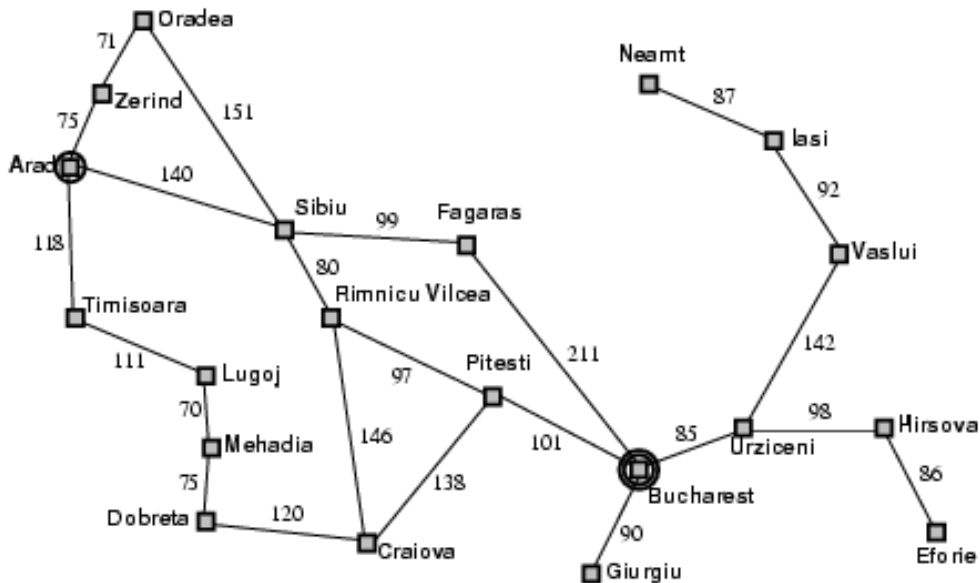


A* Search Example

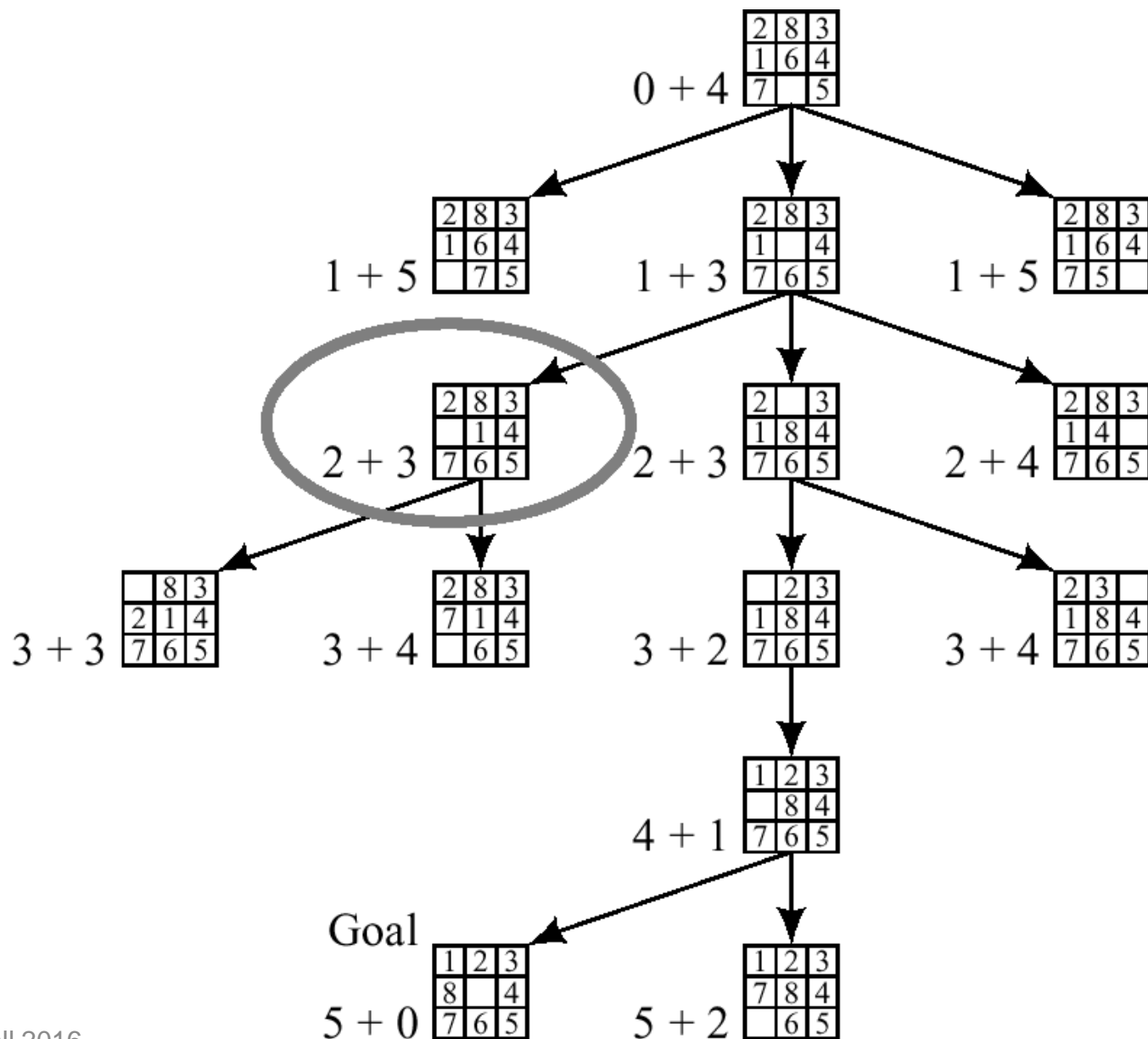


Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

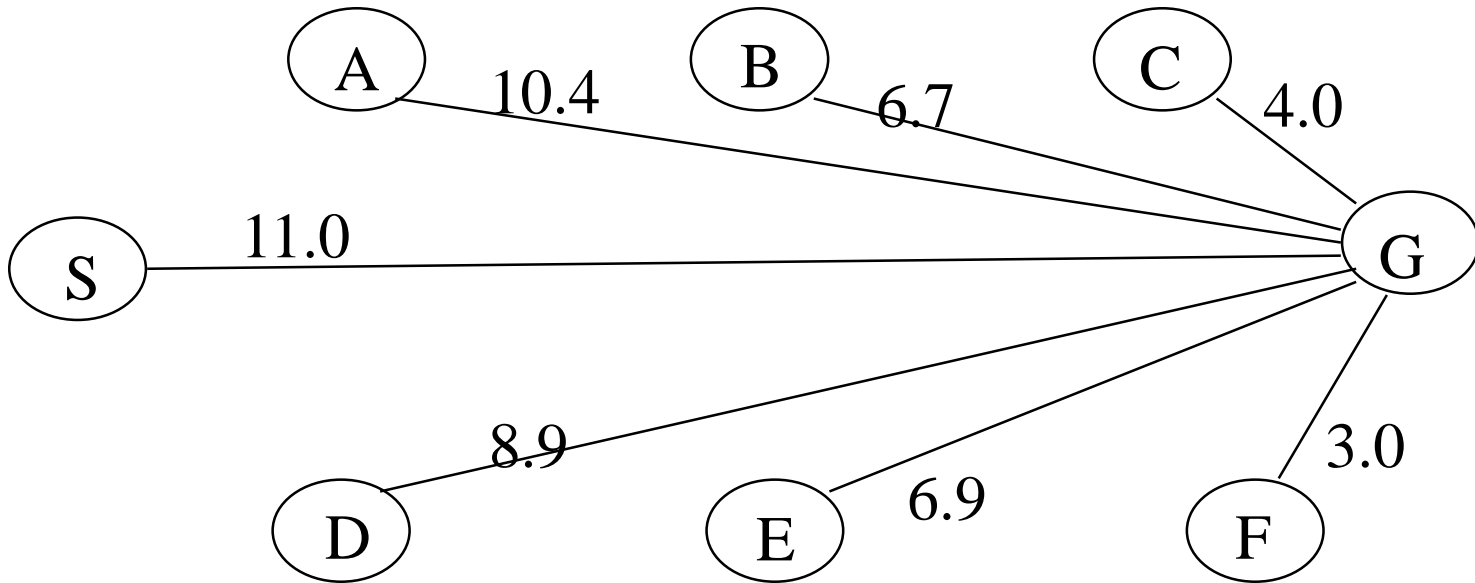
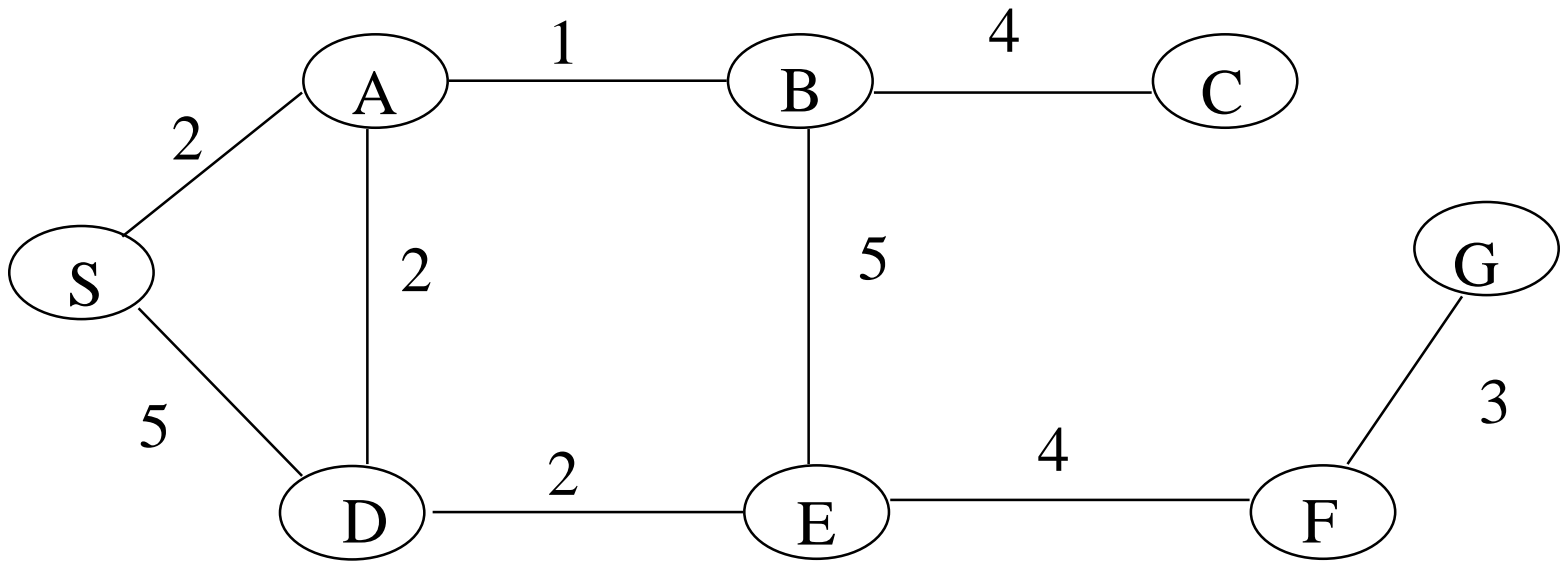


A* on 8-Puzzle with $h(n) = \#$ misplaced tiles

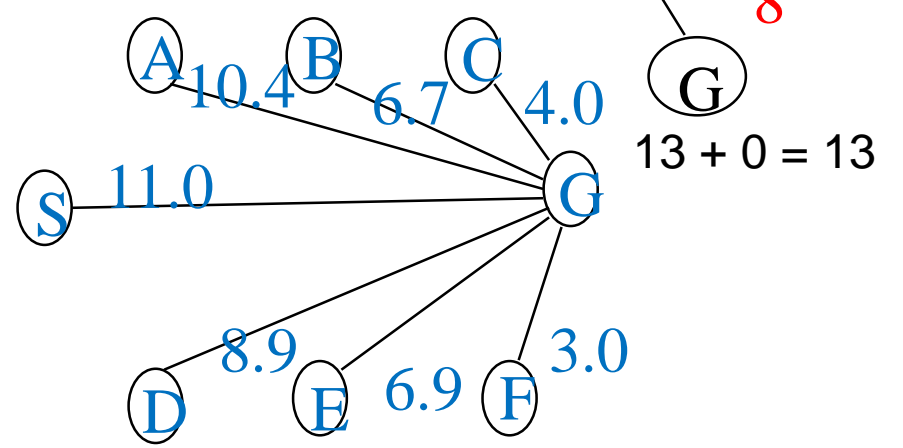
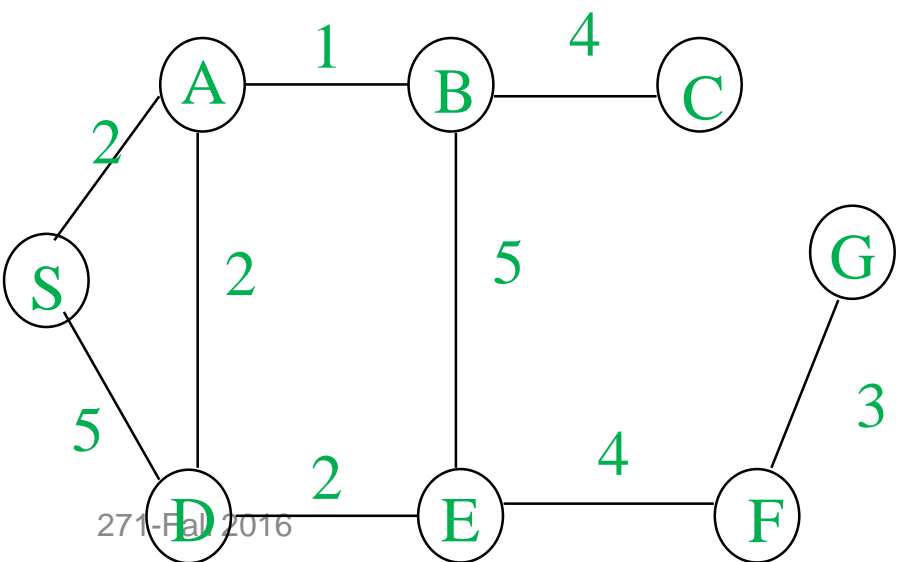
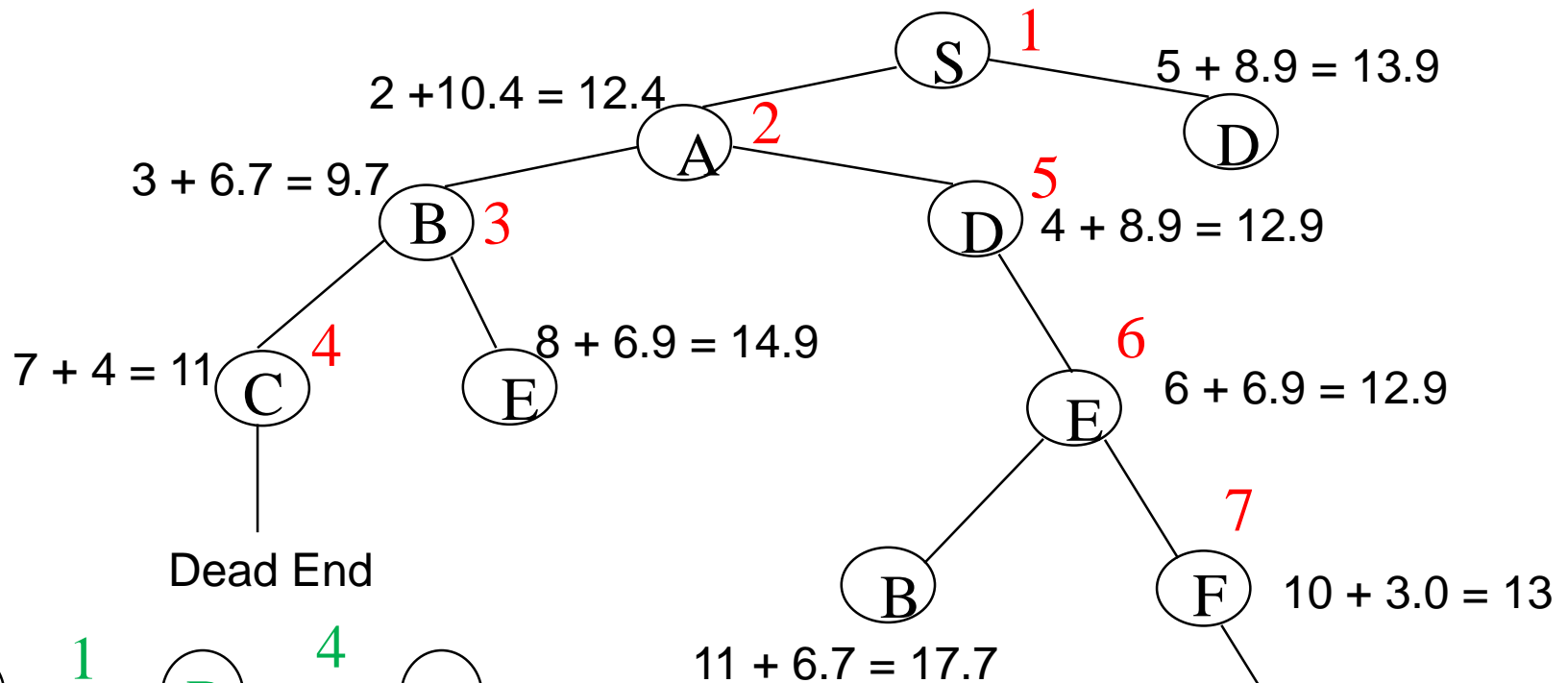


A* - a Special Best-First Search

- Goal: find a minimum sum-cost path
- Notation:
 - $c(n,n')$ - cost of arc (n,n')
 - $g(n)$ = cost of current path from start to node n in the search tree.
 - $h(n)$ = estimate of the cheapest cost of a path from n to a goal.
 - evaluation function: $f = g+h$
- $f(n)$ estimates the cheapest cost solution path that goes through n .
 - $h^*(n)$ is the true cheapest cost from n to a goal.
 - $g^*(n)$ is the true shortest path from the start s , to n .
 - C^* is the cost of optimal solution.
- If the heuristic function, h always underestimates the true cost ($h(n)$ is smaller than $h^*(n)$), then A* is guaranteed to find an optimal solution.



Example of A* Algorithm in Action



Algorithm A* (with any h on search Graph)

- Input: an implicit search graph problem with cost on the arcs
- Output: the minimal cost path from start node to a goal node.
 - 1. Put the start node s on OPEN.
 - 2. If OPEN is empty, exit with failure
 - 3. Remove from OPEN and place on CLOSED a node n having minimum f .
 - 4. If n is a goal node exit successfully with a solution path obtained by tracing back the pointers from n to s .
 - 5. Otherwise, expand n generating its children and directing pointers from each child node to n .
 - For every child node n' do
 - evaluate $h(n')$ and compute $f(n') = g(n') + h(n') = g(n) + c(n, n') + h(n')$
 - If n' is already on OPEN or CLOSED compare its new f with the old f . If the new value is higher, discard the node. Otherwise, replace old f with new f and reopen the node.
 - Else, put n' with its f value in the right order in OPEN
 - 6. Go to step 2.

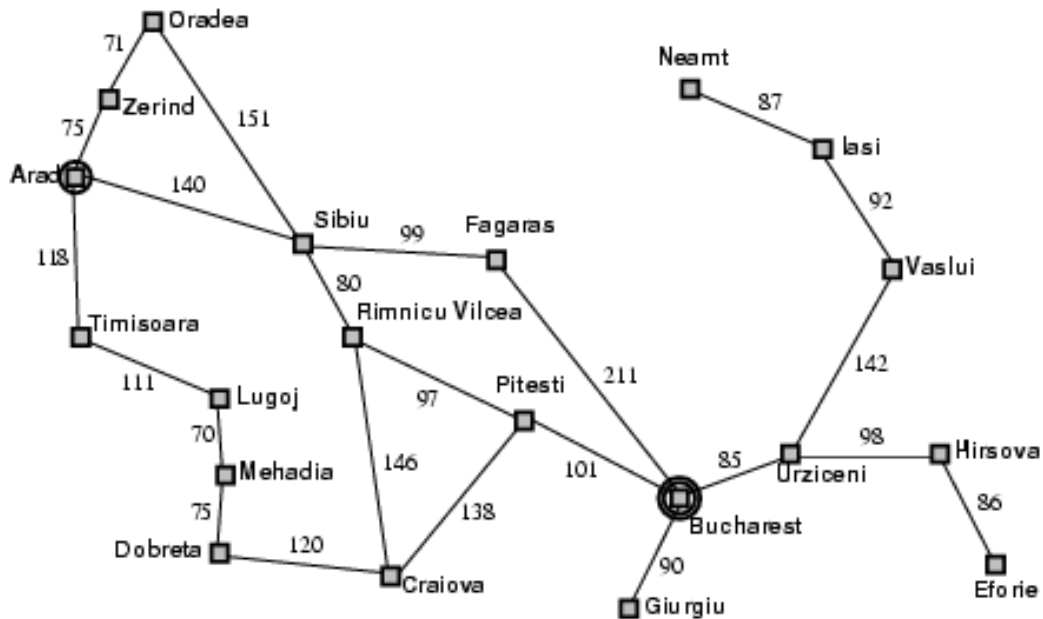
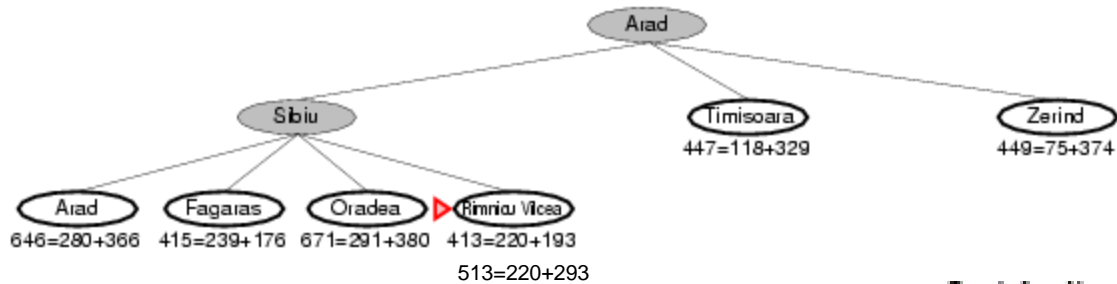
Behavior of A* - Termination/Completeness

- Theorem (completeness) (Hart, Nilsson and Raphael, 1968)
 - A* always terminates with a solution path (h is not necessarily admissible) if
 - costs on arcs are positive, above epsilon
 - branching degree is finite.
- Proof: The evaluation function f of nodes expanded must increase eventually (since paths are longer and more costly) until all the nodes on a solution path are expanded.

Admissible A*

- The heuristic function $h(n)$ is called admissible if $h(n)$ is never larger than $h^*(n)$, namely $h(n)$ is always less or equal to true cheapest cost from n to the goal.
- A* is admissible if it uses an admissible heuristic, and $h(\text{goal}) = 0$.
- If the heuristic function, h always underestimates the true cost ($h(n)$ is smaller than $h^*(n)$), then A* is guaranteed to find an optimal solution.

A* with inadmissible h



Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

→293

Consistent (monotone) Heuristics

- A heuristic is **consistent** if for every node n , every successor n' of n generated by any action a ,

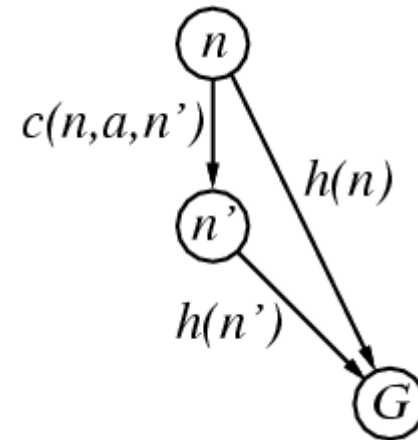
$$h(n) \leq c(n,a,n') + h(n')$$

- If h is consistent, we have

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

- i.e., $f(n)$ is non-decreasing along any path.

- Theorem:** If $h(n)$ is consistent, f along any path is non-decreasing.
- Corollary:** the f values seen by A* are non-decreasing.



Consistent Heuristics

- If h is consistent and $h(\text{goal})=0$ then h is admissible
 - Proof: (by induction of distance from the goal)
- An A^* guided by consistent heuristic finds an optimal paths to all expanded nodes, namely $g(n) = g^*(n)$ for any expanded n .
 - Proof: Assume $g(n) > g^*(n)$ and n expanded along a non-optimal path.
 - Let n' be the shallowest OPEN node on optimal path p to $n \rightarrow$
 - $g(n') = g^*(n')$ and therefore $f(n')=g^*(n')+h(n')$
 - Due to consistency we get $f(n') \leq g^*(n')+c(n',n)+h(n)$
 - Since $g^*(n) = g^*(n')+c(n',n)$ along the optimal path, we get that
 - $f(n') \leq g^*(n) + h(n)$
 - And since $g(n) > g^*(n)$ then $f(n') < g(n)+h(n) = f(n)$, contradiction

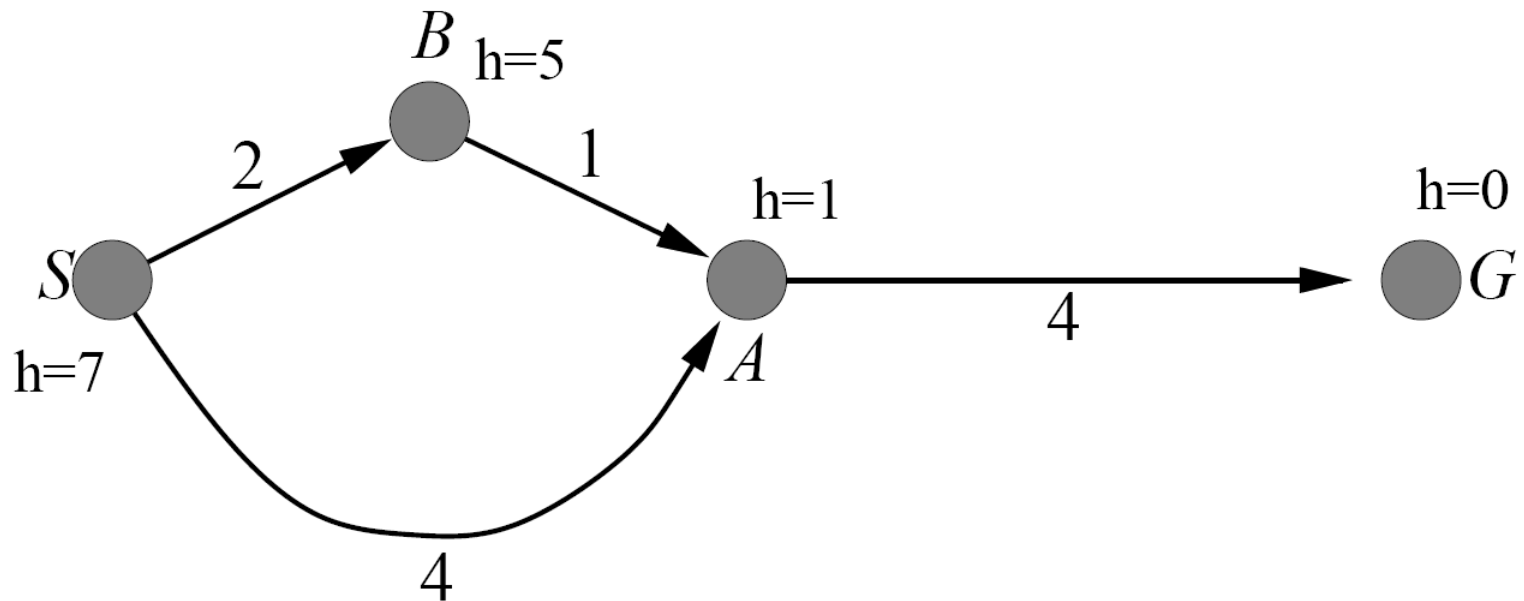
Behavior of A* - Optimality

- Theorem (completeness for optimal solution) (HNL, 1968):
 - If the heuristic function is
 - admissible (tree search or graph search with explored node re-opening)
 - consistent (graph search w/o explored node re-opening)
 - then A* finds an optimal solution.
- Proof:
 - 1. A*(admissible/consistent) will expand only nodes whose f-values are less (or equal) to the optimal cost path C^* ($f(n)$ is less-or-equal C^*).
 - 2. The evaluation function of a goal node along an optimal path equals C^* .
- Lemma:
 - Anytime before A*(admissible/consistent) terminates there exists an OPEN node n' on an optimal path with $f(n') \leq C^*$.

Requirements for Optimality

- Tree search
 - Need admissibility
- Graph search, without re-opening closed nodes
 - Need consistency
- Graph search, with re-opening closed nodes
 - Admissibility is enough

Inconsistent but admissible



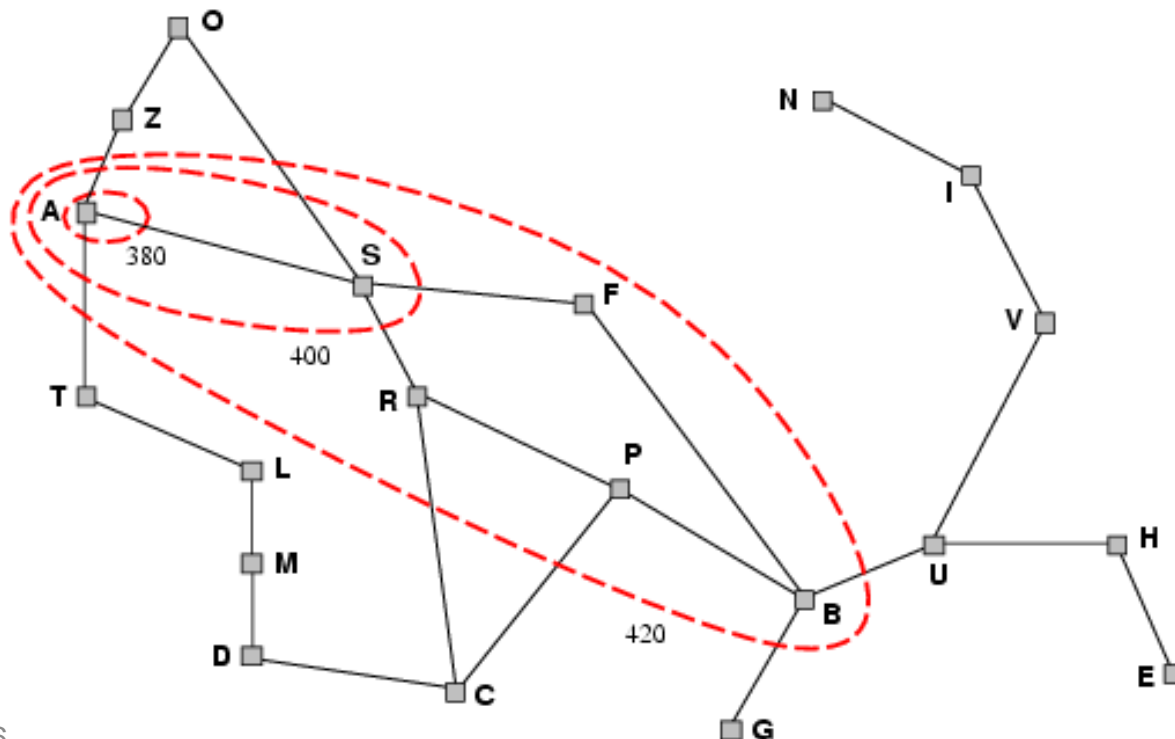
Consistency : $h(n_i) \leq c(n_i, n_j) + h(n_j)$
or $c(n_i, n_j) \geq h(n_i) - h(n_j)$
or $c(n_i, n_j) \geq \Delta h$

Summary so far

- Heuristic
- Best First Search : any f
- A^* : $f=g+h$
 - Admissible : $h \leq h^*$
 - Consistent : monotonic f

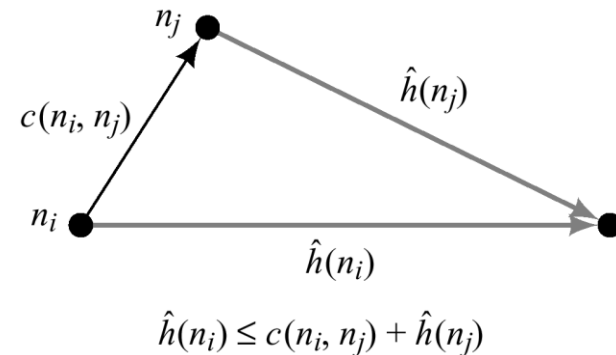
A* with Consistent Heuristics

- A* expands nodes in order of increasing f value
- Gradually adds " f -contours" of nodes
- Contour i has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Summary of Consistent Heuristics

- h is consistent if the heuristic function satisfies triangle inequality for every n and its child node n' : $h(n_i) \leq h(n_j) + c(n_i, n_j)$



- When h is consistent, the f values of nodes expanded by A^* are never decreasing.
- When A^* selected n for expansion it already found the shortest path to it.
- When h is consistent every node is expanded once.
- Normally the heuristics we encounter are consistent
 - the number of misplaced tiles
 - Manhattan distance
 - straight-line distance

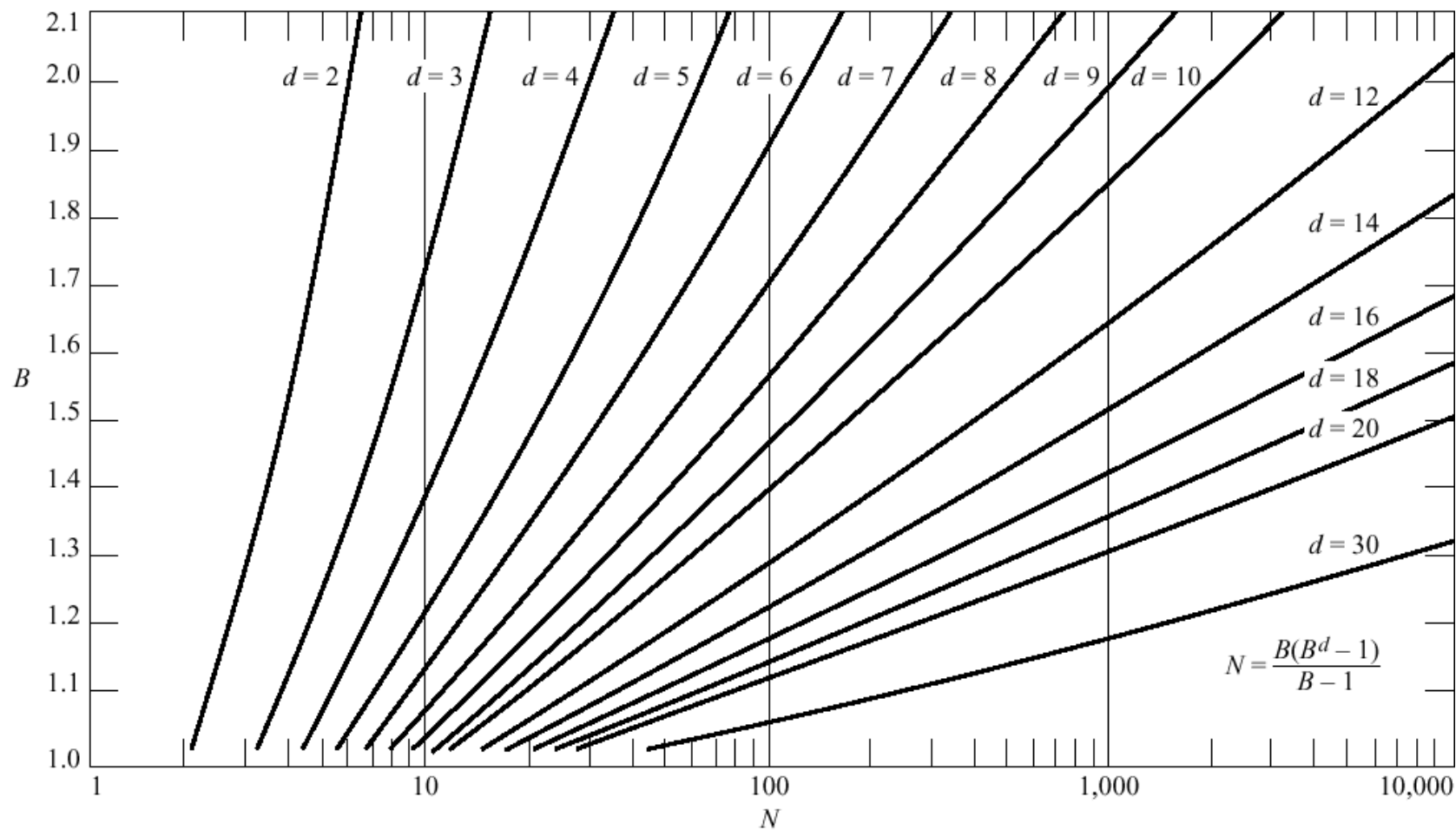
A* properties

- A* expands every path along which $f(n) < C^*$
- A* will never expand any node such that $f(n) > C^*$
- If h is consistent A* will expand any node such that $f(n) < C^*$
- Therefore, A* expands all the nodes for which $f(n) < C^*$ and a subset of the nodes for which $f(n) = C^*$.
- Therefore, if $h_1(n) < h_2(n)$ clearly the subset of nodes expanded by h_2 is smaller.

Complexity of A*

- A* is optimally efficient (Dechter and Pearl 1985):
 - It can be shown that all algorithms that do not expand a node which A* did expand (inside the contours) may miss an optimal solution
- A* worst-case time complexity:
 - is exponential unless the heuristic function is very accurate
- If h is exact ($h = h^*$)
 - search focus only on optimal paths
- Main problem:
 - space complexity is exponential
 - Not anytime; all or nothing ... but largest f expanded is lower bound on C^*
- Effective branching factor:
 - Number of nodes generated by a “typical” search node
 - Approximately : $b^* = N^{(1/d)}$
- Q: what if you are given a solution (not necessarily optimal); can you improve A* performance?

The Effective Branching Factor



Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in $h \times$ length of soln.]

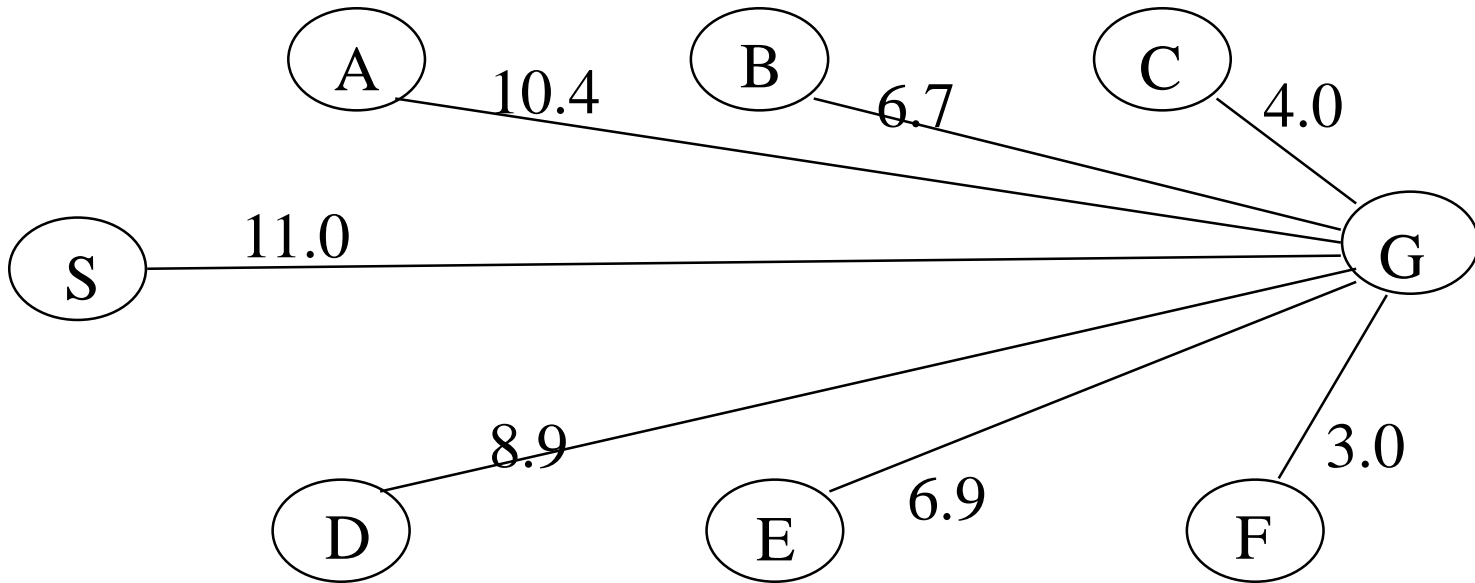
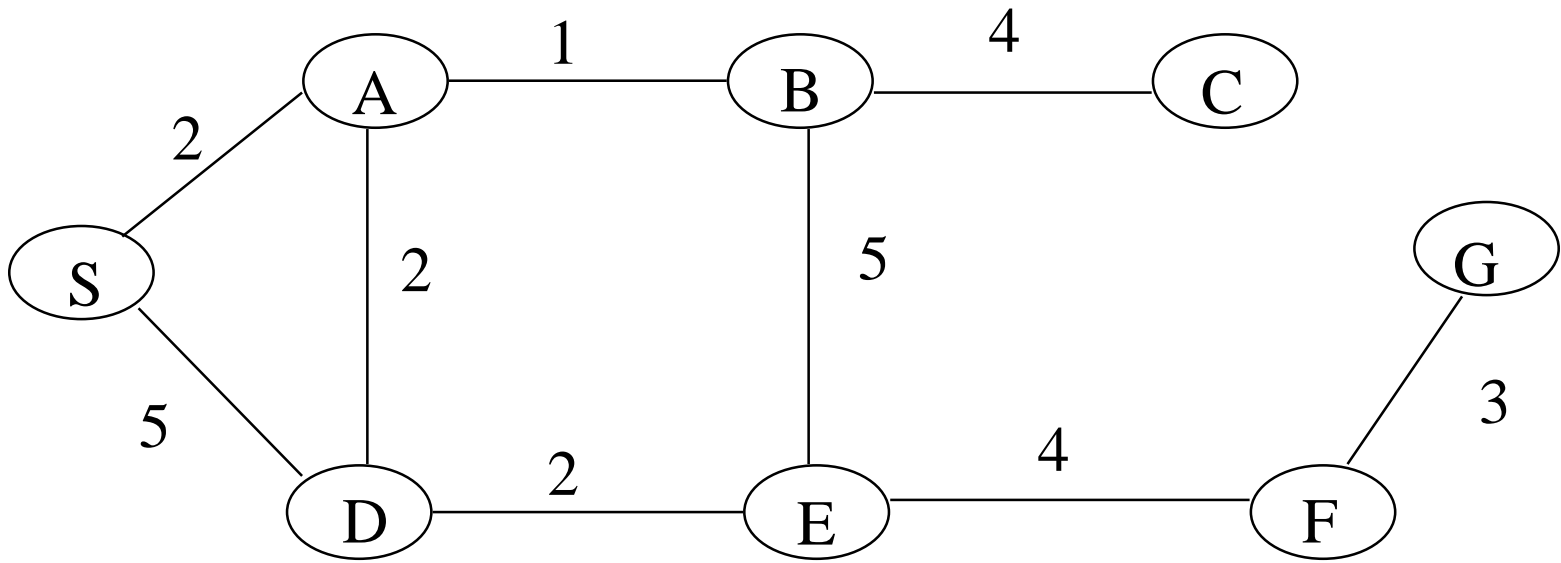
Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

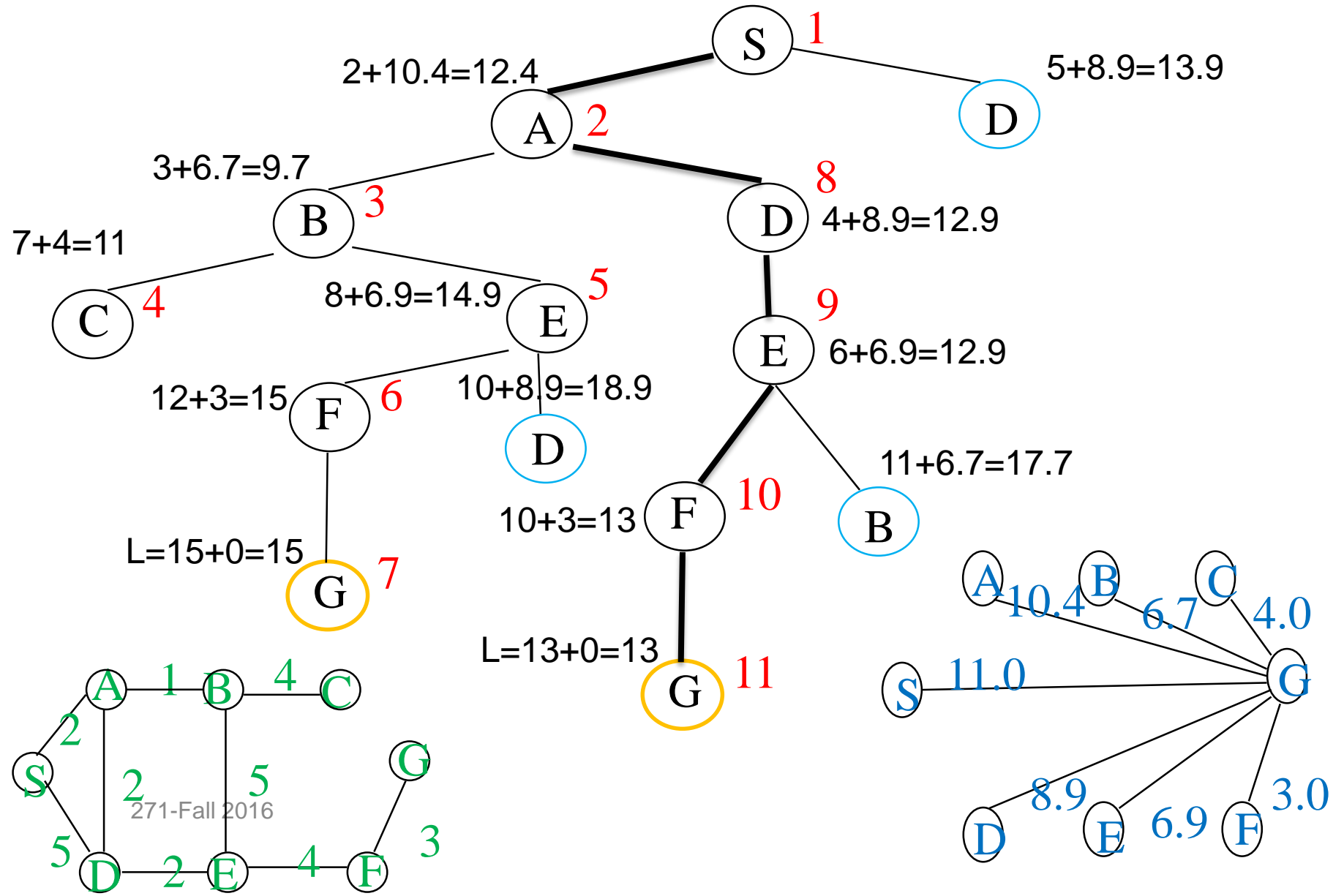
A* expands all nodes with $f(n) < C^*$

A* expands some nodes with $f(n) = C^*$

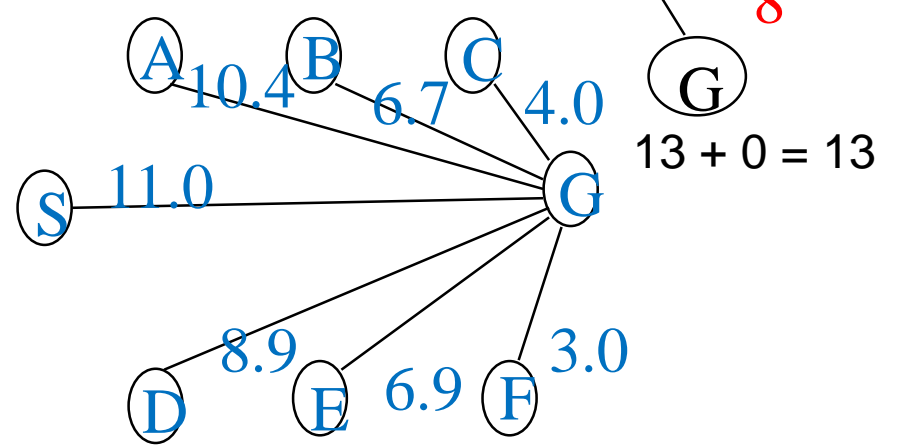
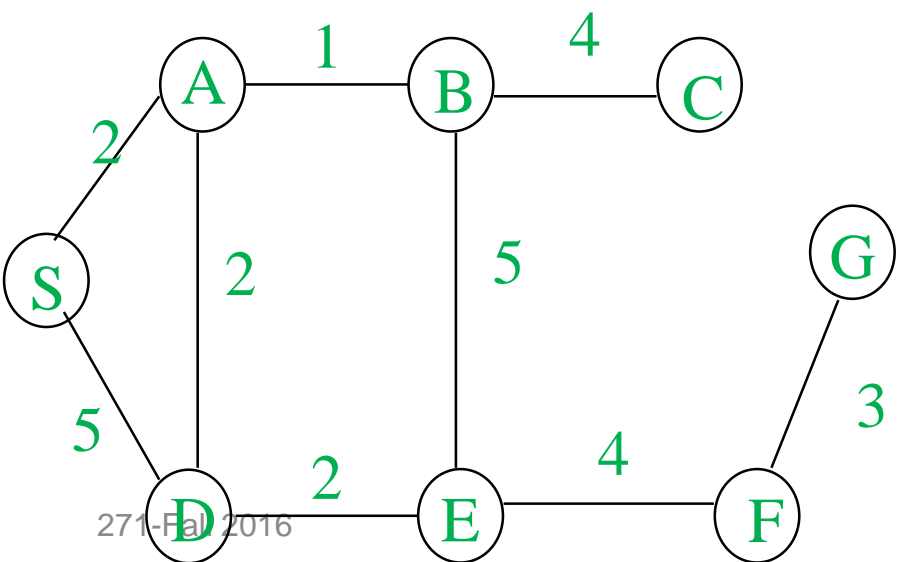
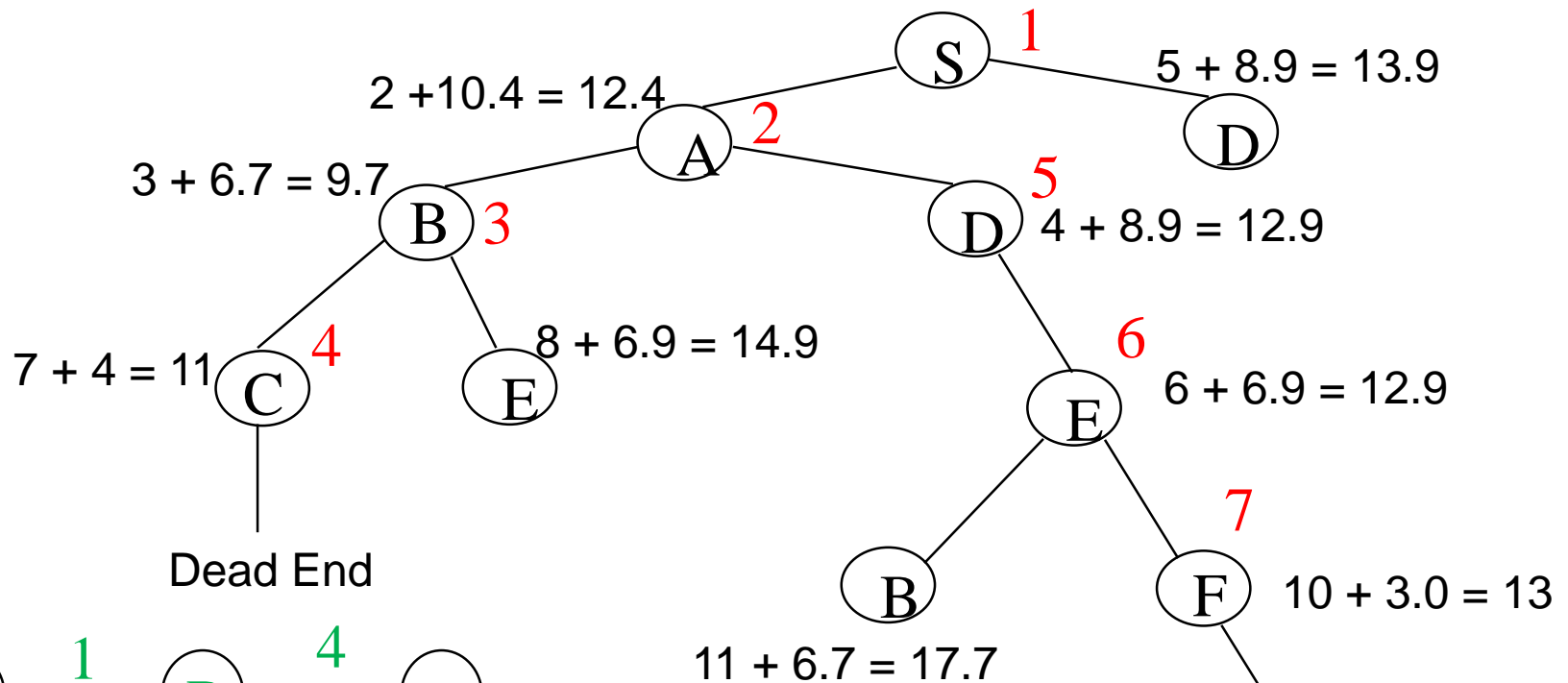
A* expands no nodes with $f(n) > C^*$



Example of Branch and Bound in Action



Example of A* Algorithm in Action



Pseudocode for Branch and Bound Search (An informed depth-first search)

Initialize: Let $Q = \{S\}$, $L = \infty$

While Q is not empty

 pull $Q1$, the first element in Q

 if $f(Q1) \geq L$, skip it

 if $Q1$ is a goal compute the cost of the solution and update

$L \leftarrow \text{minimum}(\text{new cost}, \text{old cost})$

 else

$\text{child_nodes} = \text{expand}(Q1)$,

 <eliminate child_nodes which represent simple loops>,

 For each child node n do:

 evaluate $f(n)$. If $f(n)$ is greater than L discard n .

 end-for

 Put remaining child_nodes on top of queue in the order of their f .

 end

Continue

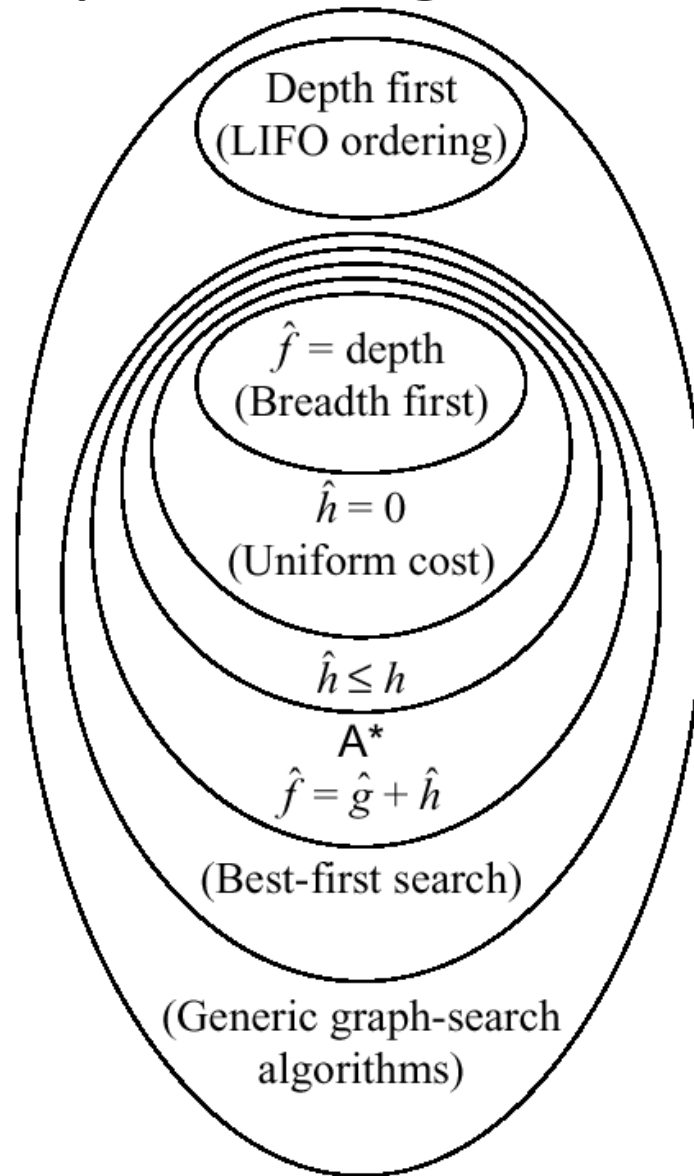
Properties of Branch-and-Bound

- Not guaranteed to terminate unless
 - has depth-bound
 - admissible f and reasonable L
- Optimal:
 - finds an optimal solution (f is admissible)
- Time complexity: exponential
- Space complexity: can be linear
- Advantage:
 - anytime property
- Note : unlike A^* , BnB may (will) expand nodes $f > C^*$.

Iterative Deepening A* (IDA*) (combining Branch-and-Bound and A*)

- Initialize: $f \leftarrow$ the evaluation function of the start node
- until goal node is found
 - Loop:
 - Do Branch-and-bound with upper-bound L equal to current evaluation function f .
 - Increment evaluation function to next contour level
 - end
- Properties:
 - Guarantee to find an optimal solution
 - time: exponential, like A*
 - space: linear, like B&B.
- Problems: The number of iterations may be large - Δf may be ϵ .

Relationships among Search Algorithms



Effectiveness of heuristic search

- How quality of the heuristic impacts search?
- What is the time and space complexity?
- Is any algorithm better? Worse?
- Case study: the 8-puzzle

Admissible and Consistent Heuristics?

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
 - $h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)
- The true cost is 26.
Average cost for 8-puzzle is 22. Branching degree 3.

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ $3+1+2+2+2+3+3+2 = 18$

Effectiveness of A* Search Algorithm

Average number of nodes expanded

d	IDS	A*(h1)	A*(h2)
2	10	6	6
4	112	13	12
8	6384	39	25
12	364404	227	73
14	3473941	539	113
20	-----	7276	676
24	-----	39135	1641

Average over 100 randomly generated 8-puzzle problems

h1 = number of tiles in the wrong position

h2 = sum of Manhattan distances

Dominance

- Definition: If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 **dominates** h_1
- Is h_2 better for search?
- Typical search costs (average number of nodes expanded):
 - $d=12$ IDS = 3,644,035 nodes
 $A^*(h_1) = 227$ nodes
 $A^*(h_2) = 73$ nodes
 - $d=24$ IDS = out of memory
 $A^*(h_1) = 39,135$ nodes
 $A^*(h_2) = 1,641$ nodes

Heuristic's Dominance and Pruning Power

- Definition:
 - A heuristic function h_2 (strictly) dominates h_1 if both are admissible and for every node n , $h_2(n)$ is (strictly) greater than $h_1(n)$.
- Theorem (Hart, Nilsson and Raphael, 1968):
 - An A^* search with a dominating heuristic function h_2 has the property that any node it expands is also expanded by A^* with h_1 .
- Question: Does Manhattan distance dominate the number of misplaced tiles?
- Extreme cases
 - $h = 0$
 - $h = h^*$

Inventing Heuristics automatically

- Examples of Heuristic Functions for A*
 - The 8-puzzle problem
 - The number of tiles in the wrong position
 - is this admissible?
 - Manhattan distance
 - is this admissible?
 - How can we invent admissible heuristics in general?
 - look at “relaxed” problem where constraints are removed
 - e.g., we can move in straight lines between cities
 - e.g., we can move tiles independently of each other

Inventing Heuristics Automatically (cont.)

- How did we
 - find h_1 and h_2 for the 8-puzzle?
 - verify admissibility?
 - prove that straight-line distance is admissible? MST admissible?
- Hypothetical answer:
 - Heuristic are generated from relaxed problems
 - Hypothesis: relaxed problems are easier to solve
- In relaxed models the search space has more operators or more directed arcs
- Example: 8 puzzle:
 - Rule : a tile can be moved from A to B, iff
 - A and B are adjacent
 - B is blank
 - We can generate relaxed problems by removing one or more of the conditions
 - ... if A and B are adjacent
 - ... if B is blank

Relaxed Problems

- A problem with fewer restrictions on the actions is called a **relaxed problem**
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then $h_1(n)$ (*number of misplaced tiles*) gives the shortest solution
- If the rules are relaxed so that a tile can move to **any h/v adjacent square**, then $h_2(n)$ (*Manhattan distance*) gives the shortest solution

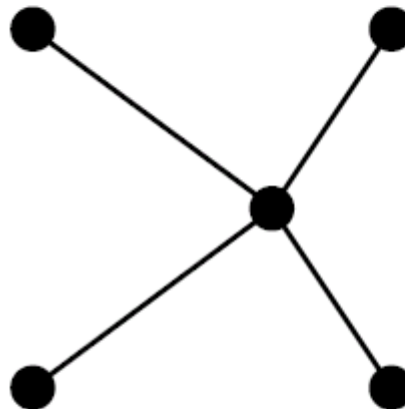
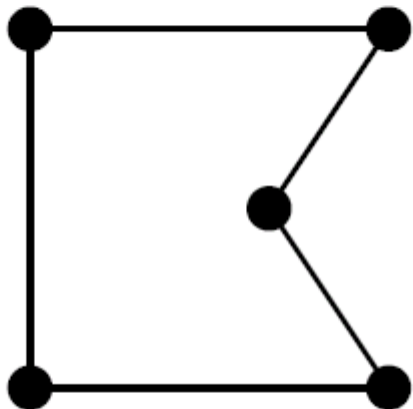
Generating heuristics (cont.)

- Example: TSP
- Find a tour. A tour is:
 - 1. A graph with subset of edges
 - 2. Connected
 - 3. Total length of edges minimized
 - 4. Each node has degree 2
- Eliminating 4 yields MST.

Relaxed problems contd.

Well-known example: [travelling salesperson problem](#) (TSP)

Find the shortest tour visiting all cities exactly once



[Minimum spanning tree](#) can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour

Automating Heuristic generation

- Use STRIPs language representation:
- Operators:
 - pre-conditions, add-list, delete list
- 8-puzzle example:
 - $\text{on}(x,y)$, $\text{clear}(y)$ $\text{adj}(y,z)$,tiles x_1,\dots,x_8
- States: conjunction of predicates:
 - $\text{on}(x_1,c_1),\text{on}(x_2,c_2)\dots\text{on}(x_8,c_8),\text{clear}(c_9)$
- $\text{move}(x,c_1,c_2)$ (move tile x from location c_1 to location c_2)
 - pre-cond: $\text{on}(x_1,c_1)$, $\text{clear}(c_2)$, $\text{adj}(c_1,c_2)$
 - add-list: $\text{on}(x_1,c_2)$, $\text{clear}(c_1)$
 - delete-list: $\text{on}(x_1,c_1)$, $\text{clear}(c_2)$
- Relaxation:
 - Remove from precondition: $\text{clear}(c_2)$, $\text{adj}(c_2,c_3) \rightarrow \#$ misplaced tiles
 - Remove $\text{clear}(c_2) \rightarrow$ Manhattan distance
 - Remove $\text{adj}(c_2,c_3) \rightarrow h_3$, a new procedure that transfers to the empty location a tile appearing there in the goal
- The space of relaxations can be enriched by predicate refinements
 - $\text{adj}(y,z) = \text{iff neighbour}(y,z)$ and $\text{same-line}(y,z)$

Heuristic generation

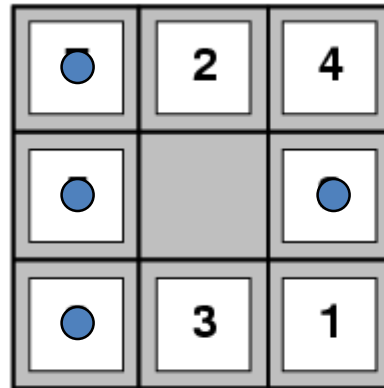
- Theorem: Heuristics that are generated from relaxed models are consistent.
- Proof: h is true shortest path in a relaxed model
 - $h(n) \leq c'(n, n') + h(n')$ (c' are shortest distances in relaxed graph)
 - $c'(n, n') \leq c(n, n')$
 - $\rightarrow h(n) \leq c(n, n') + h(n')$

Heuristic generation

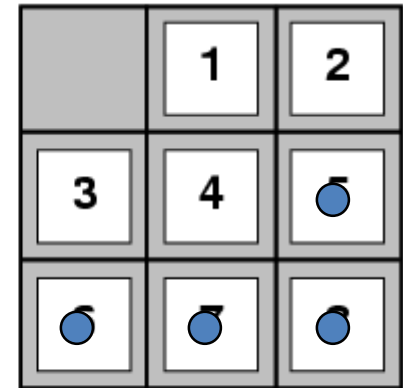
- Total (time) complexity = heuristic computation + nodes expanded
- More powerful heuristic – harder to compute, but more pruning power (fewer nodes expanded)
- Problem:
 - not every relaxed problem is easy
 - How to recognize a relaxed easy problem
 - A proposal: a problem is easy if it can be solved optimally by a greedy algorithm
- Q: what if neither h_1 nor h_2 is clearly better? $\max(h_1, h_2)$
- Often, a simpler problem which is more constrained is easier; will provide a good upper-bound.

Improving Heuristics

- Reinforcement learning.
- Pattern Databases: you can solve optimally a sub-problem



Start State



Goal State

Pattern Databases

- For sliding tiles and Rubic's cube
- For a subset of the tiles compute shortest path to the goal using breadth-first search
- For 15 puzzles, if we have 7 fringe tiles and one blank, the number of patterns to store are $16!/(16-8)! = 518,918,400$.
- For each table entry we store the shortest number of moves to the goal from the current location.
- Use different subsets of tiles and take the max heuristic during IDA* search. The number of nodes to solve 15 puzzles was reduced by a factor of 346 (Culberson and Schaeffer)
- How can this be generalized? (a possible project)

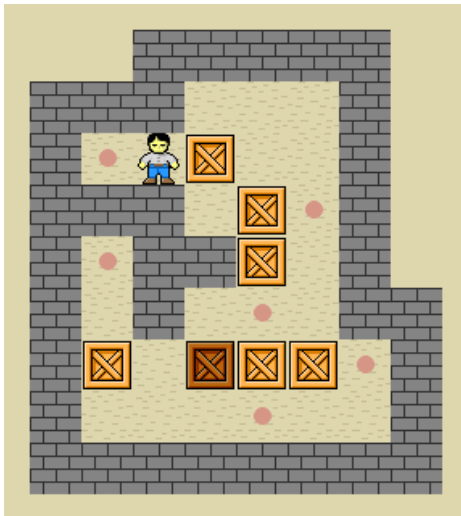
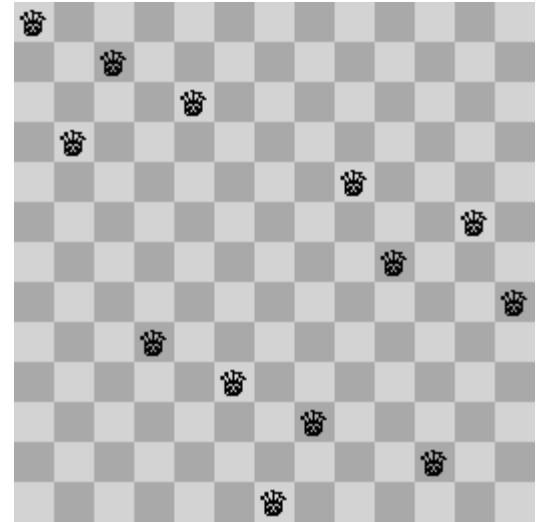
Beyond Classical Search

- AND/OR search spaces
 - Decomposable independent problems
 - Searching with non-deterministic actions (erratic vacuum)
 - Using AND/OR search spaces; solution is a contingent plan
- Local search for optimization
 - Greedy hill-climbing search, simulated annealing, local beam search, genetic algorithms.
 - Local search in continuous spaces
 - SLS : "Like climbing Everest in thick fog with amnesia"
- Searching with partial observations
 - Using belief states
- Online search agents and unknown environments
 - Actions, costs, goal-tests are revealed in state only
 - Exploration problems. Safely explorable

Course project

Choose one of the following

1. N-queens
2. (Classic) Sokoban
3. Sudoku
4. Mastermind



5	3			7			
6			1	9	5		
	9	8					6
8				6			3
4			8		3		1
7				2			6
	6					2	8
			4	1	9		5
				8			7
						7	9



Problem-reduction representations

AND/OR search spaces

- Decomposable production systems (language parsing)

Initial database: (C,B,Z)

Rules: R1: $C \rightarrow (D,L)$

R2: $C \rightarrow (B,M)$

R3: $B \rightarrow (M,M)$

R4: $Z \rightarrow (B,B,M)$

Find a path generating a string with M's only.

- Graphical models
- The tower of Hanoi

To move n disks from peg 1 to peg 3 using peg 2

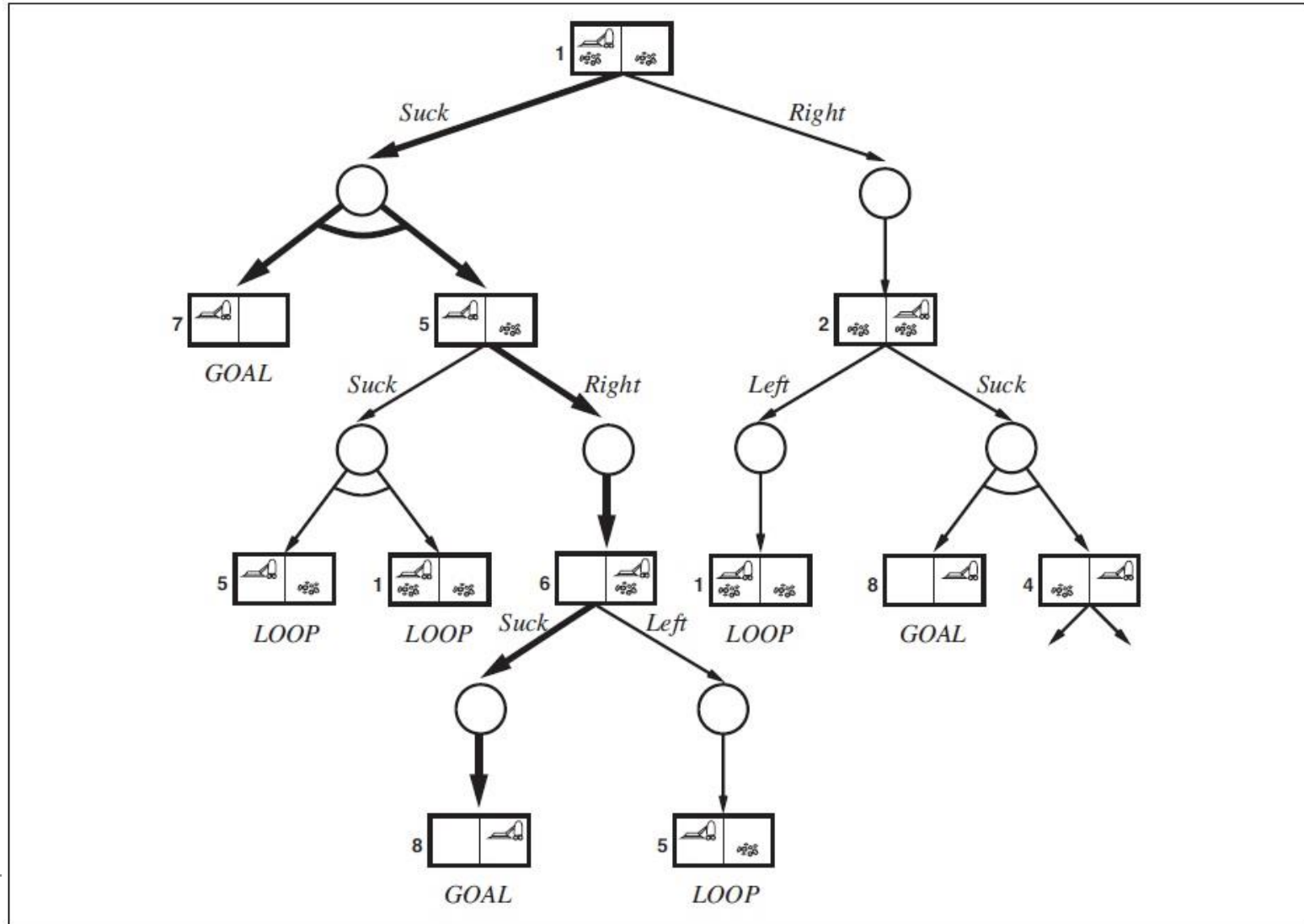
Move n-1 pegs to peg 2 via peg 3,

move the nth disk to peg 3,

move n-1 disks from peg 2 to peg 3 via peg 1.

AND/OR search spaces

non-deterministic actions : the erratic vacuum world



AND/OR Graphs

- Nodes represent subproblems
 - AND links represent subproblem decompositions
 - OR links represent alternative solutions
 - Start node is initial problem
 - Terminal nodes are solved subproblems
- Solution graph
 - It is an AND/OR subgraph such that:
 - It contains the start node
 - All its terminal nodes (nodes with no successors) are solved primitive problems
 - If it contains an AND node A, it must contain the entire group of AND links that leads to children of A.

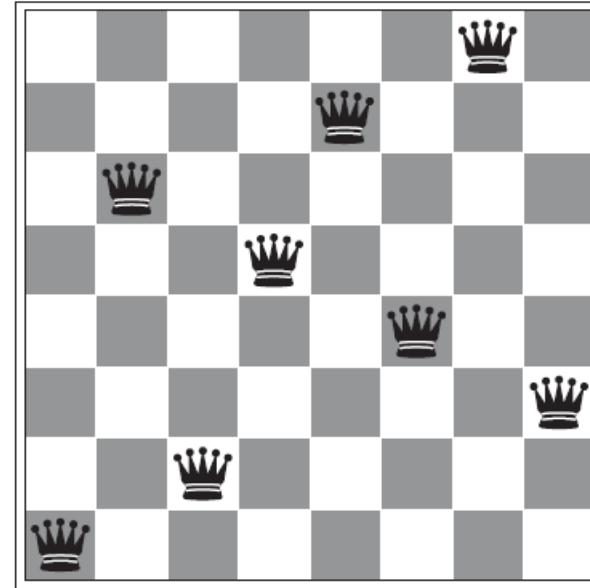
Algorithms searching AND/OR graphs

- All algorithms generalize using hyper-arc successors rather than simple arcs.
- AO*: is A* that searches AND/OR graphs for a solution subgraph.
- The cost of a solution graph is the sum cost of its arcs. It can be defined recursively as: $k(n,N) = c_n + k(n_1,N) + \dots + k(n_k,N)$
- $h^*(n)$ is the cost of an optimal solution graph from n to a set of goal nodes
- $h(n)$ is an admissible heuristic for $h^*(n)$
- Monotonicity:
- $h(n) \leq c + h(n_1) + \dots + h(n_k)$ where n_1, \dots, n_k are successors of n
- AO* is guaranteed to find an optimal solution when it terminates if the heuristic function is admissible

Local Search

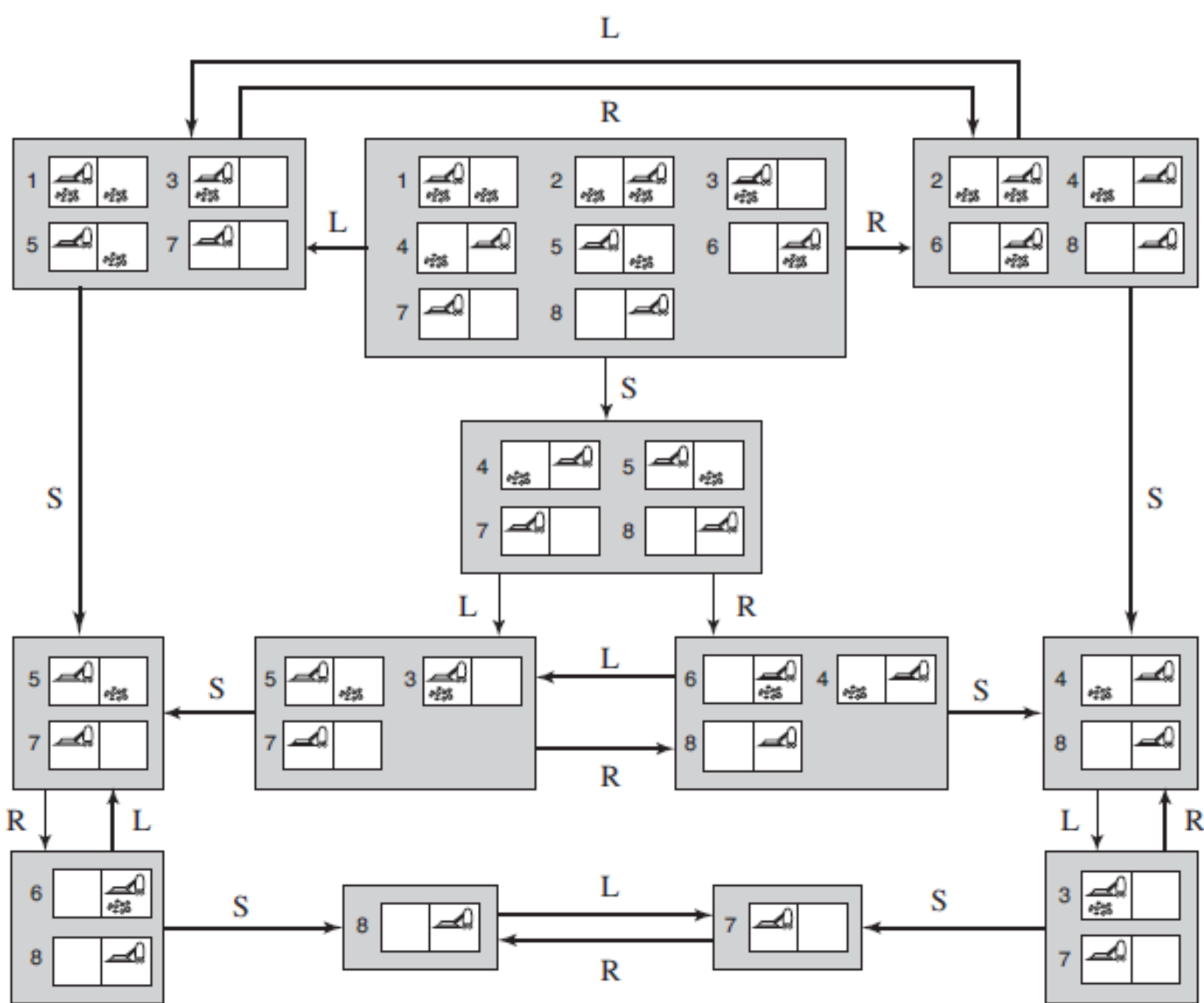
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♙	13	16	13	16
♙	14	17	15	♙	14	16	16
17	♙	16	18	15	♙	15	♙
18	14	♙	15	15	14	♙	16
14	14	13	17	12	14	12	18

(a)



(b)

Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.



Summary

- In practice we often want the goal with the minimum cost path
- Exhaustive search is impractical except on small problems
- Heuristic estimates of the path cost from a node to the goal can be efficient in reducing the search space.
- The A* algorithm combines all of these ideas with admissible heuristics (which underestimate) , guaranteeing optimality.
- Properties of heuristics:
 - admissibility, consistency, dominance, accuracy
- Reading
 - R&N Chapters 3-4