Chapter 4: Informed Heuristic Search

ICS 171 Fall 2006

Summary

- **Heuristics and Optimal search strategies**
  - heuristics
  - hill-climbing algorithms
  - Best-First search
  - A*: optimal search using heuristics
  - Properties of A*
    - admissibility,
    - monotonicity,
    - accuracy and dominance
    - efficiency of A*
  - Branch and Bound
  - Iterative deepening A*
  - Automatic generation of heuristics

Problem: finding a Minimum Cost Path

- **Previously we wanted an arbitrary path to a goal or best cost.**
- **Now, we want the minimum cost path to a goal G**
  - Cost of a path = sum of individual transitions along 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

Best-first search

- **Idea: use an evaluation function *f(n)* for each node**
  - estimate of "desirability"

  ⇒ Expand most desirable unexpanded node

- **<u>Implementation</u>:**
  Order the nodes in fringe in decreasing order of desirability

- **Special cases:**
  - greedy best-first search
  - A* search

Heuristic functions

- 8-puzzle
- 8-queen
- Travelling salesperson

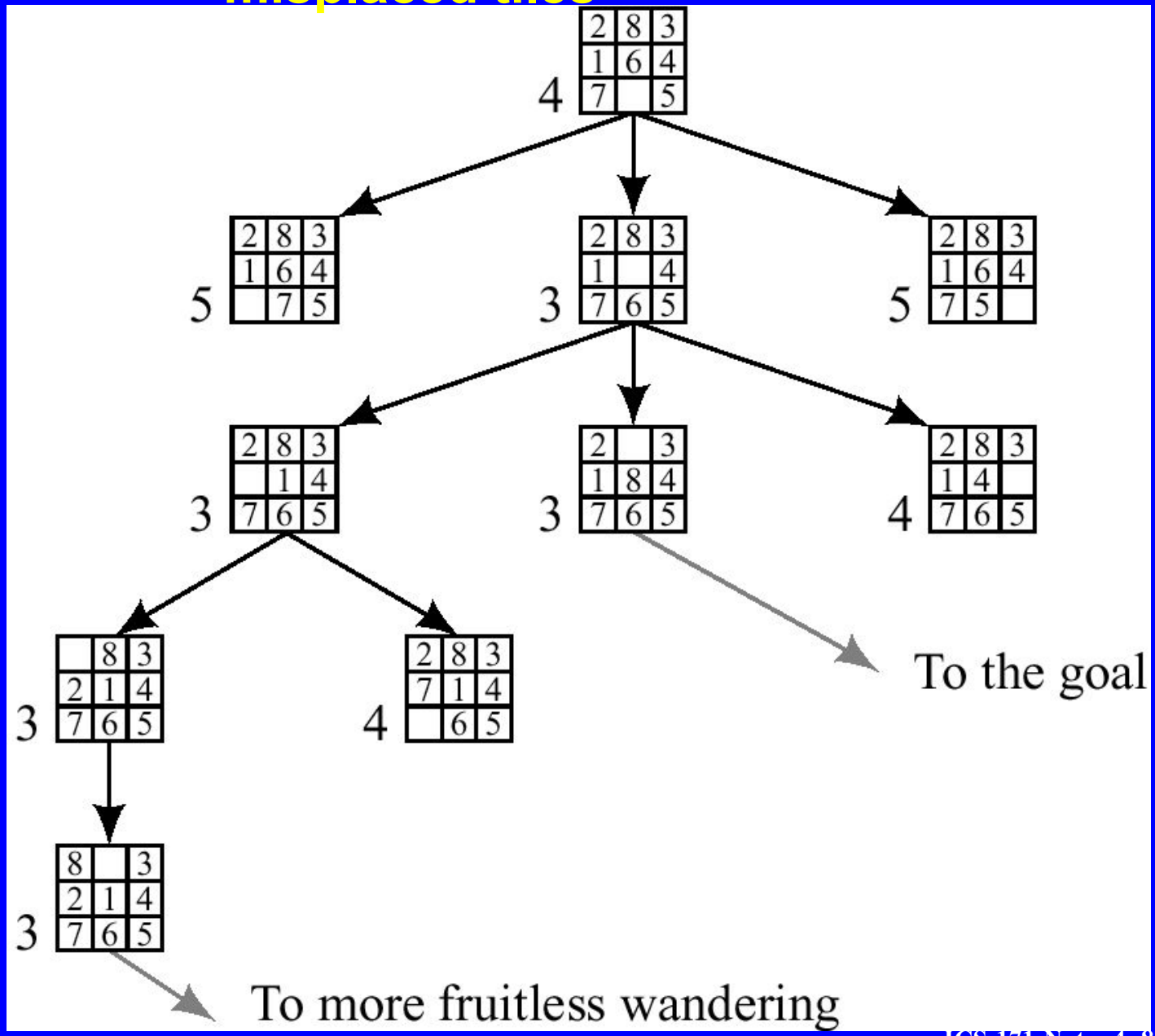
Heuristic functions

- **8-puzzle**
  - W(n): number of misplaced tiles
  - Manhatten distance
  - Gaschnig's
- **8-queen**
- **Travelling salesperson**

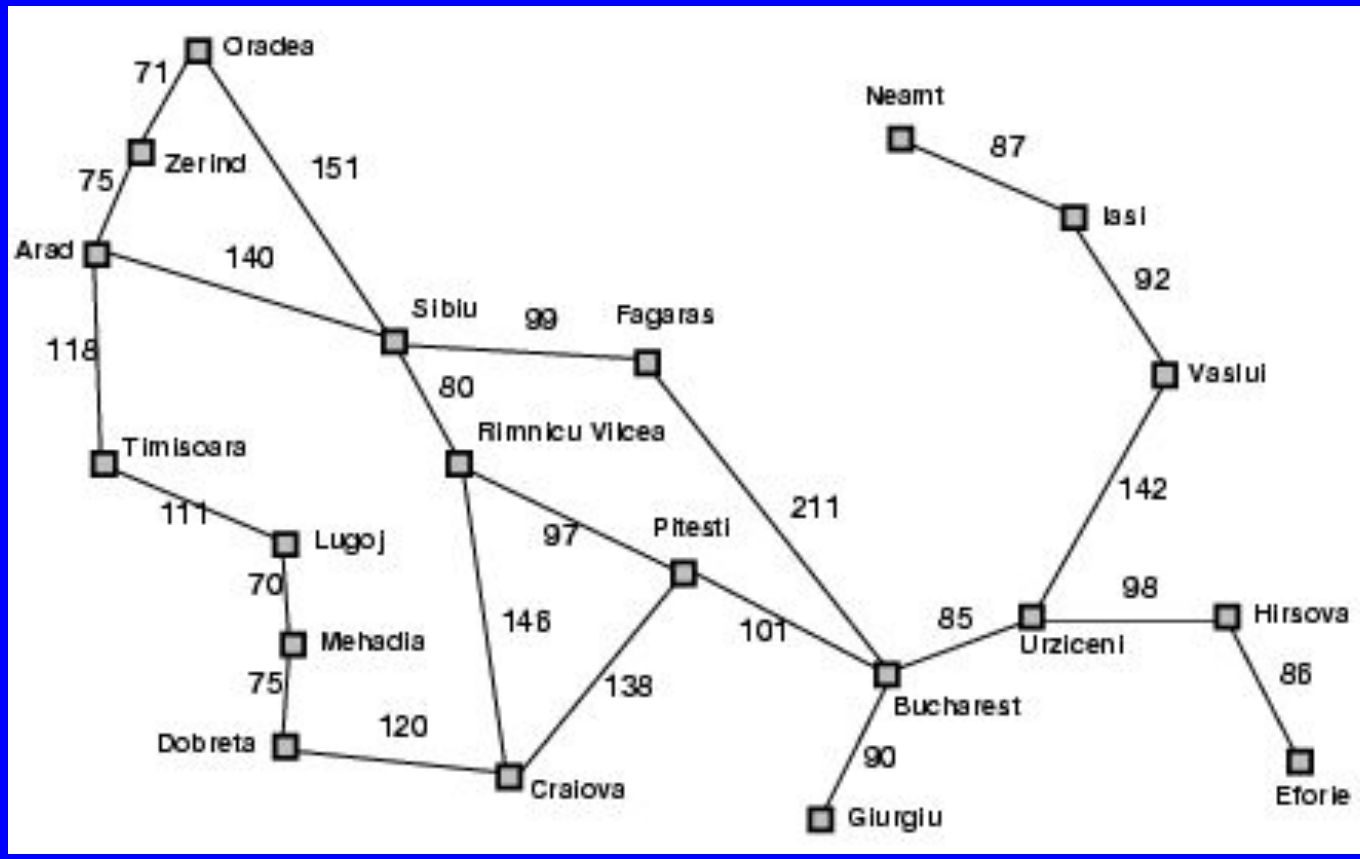
Heuristic functions

- **8-puzzle**
  - W(n): number of misplaced tiles
  - Manhatten distance
  - Gaschnig's
- **8-queen**
  - Number of future feasible slots
  - Min number of feasible slots in a row
- **Travelling salesperson**
  - Minimum spanning tree
  - Minimum assignment problem

Best first (Greedy) search: f(n) = number of misplaced tiles



Romania with step costs in km



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

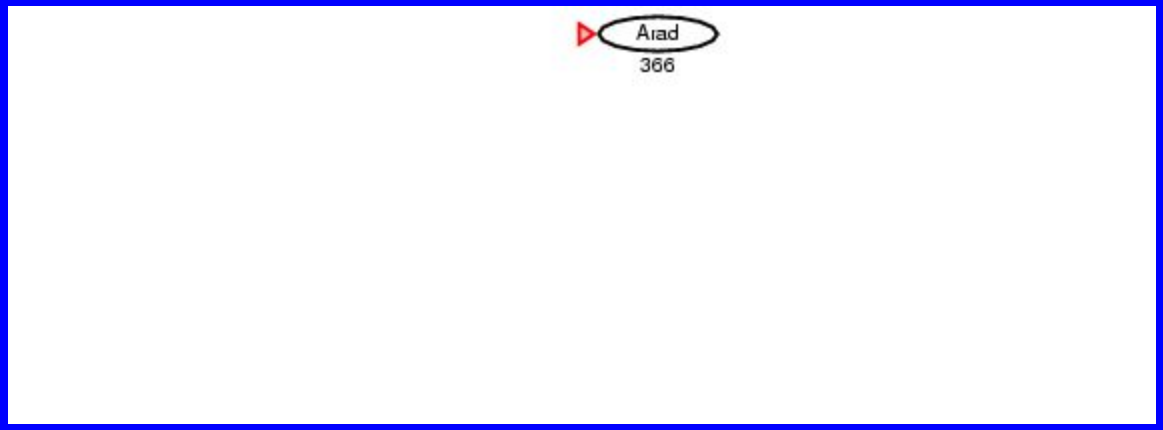
Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic)
- = estimate of cost from n to *goal*

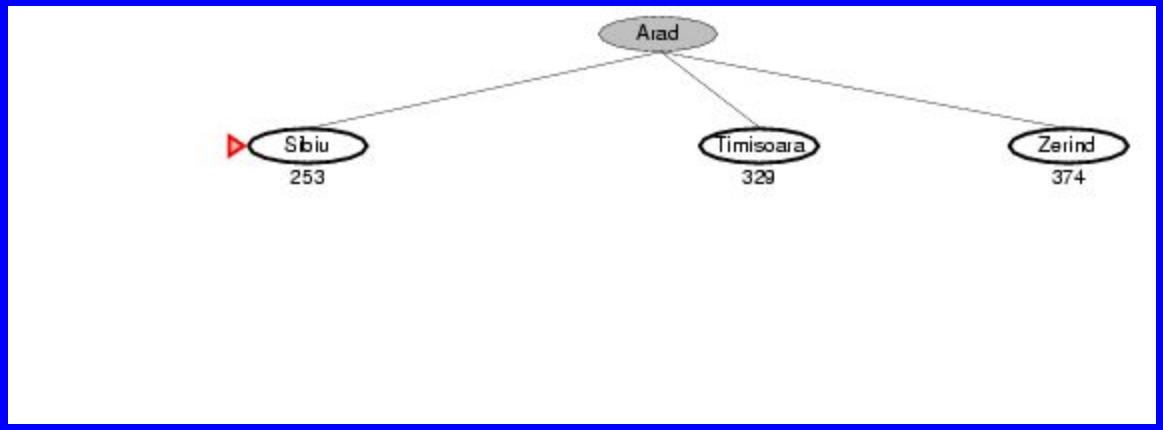
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

- Greedy best-first search expands the node that appears to be closest to goal

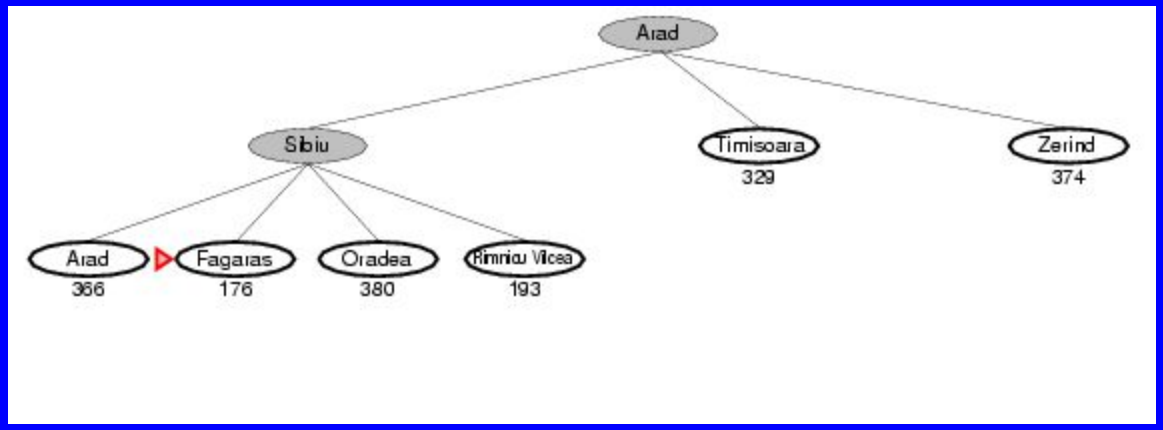
Greedy best-first search example



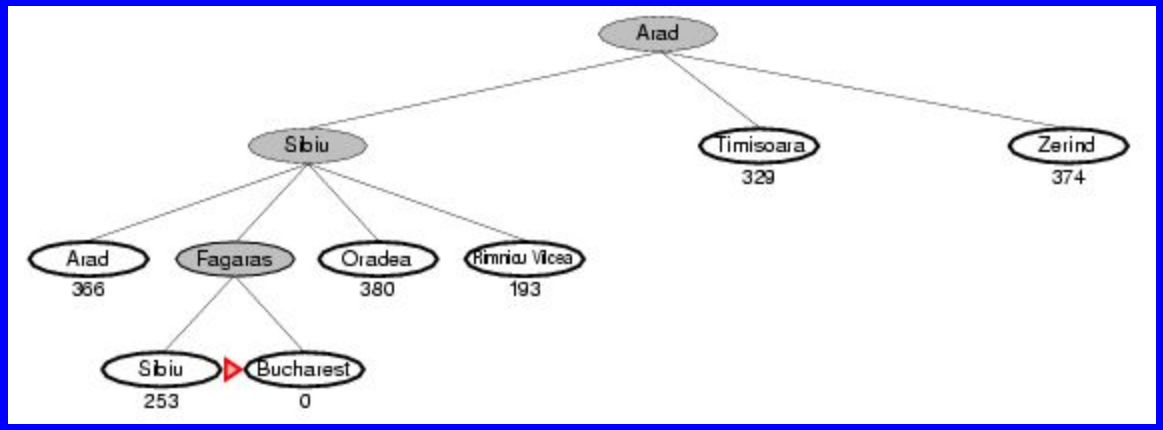
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example



Problems with Greedy Search

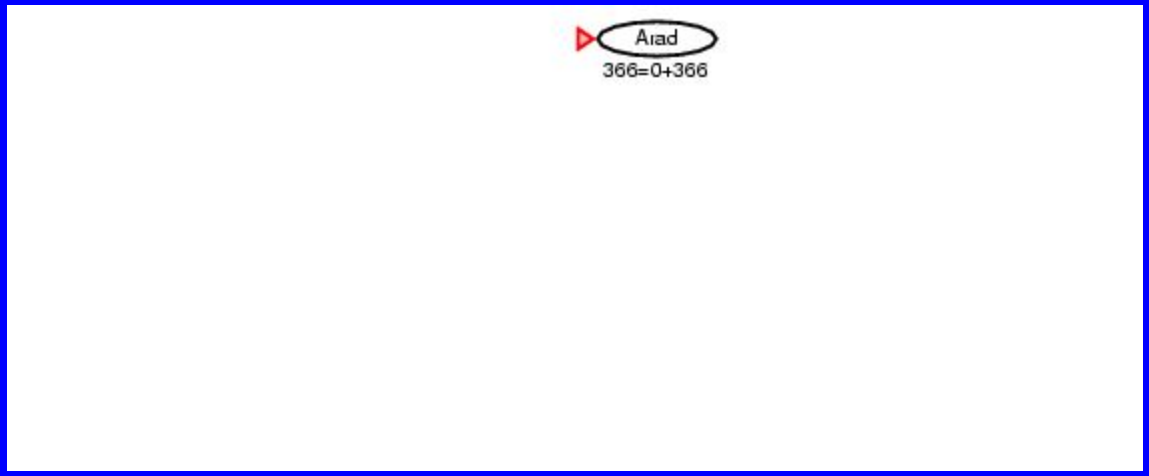
- Not complete

- Get stuck on local minimas and plateaus,
- Irrevocable,
- Infinite loops
- Can we incorporate heuristics in systematic search?

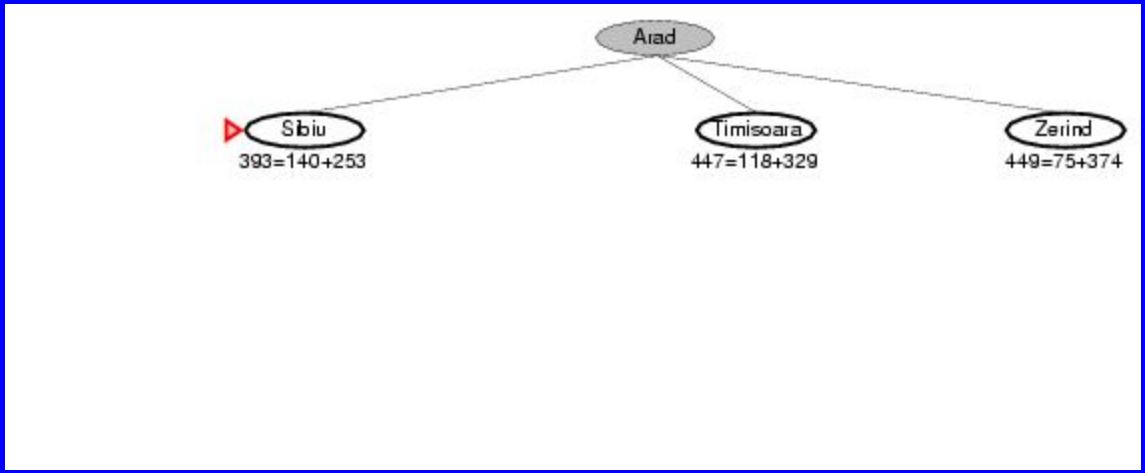
A* search

- Idea: avoid expanding paths that are already expensive
- 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

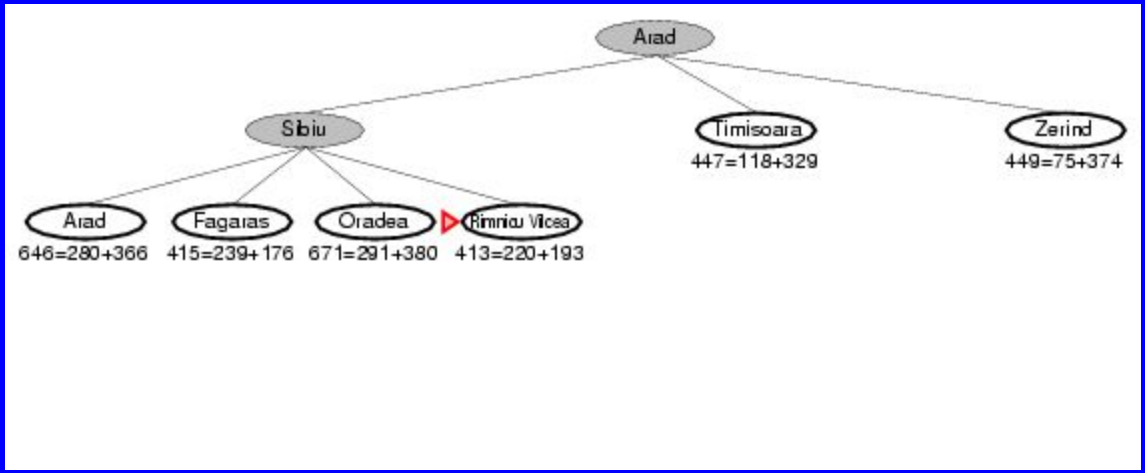
A* search example



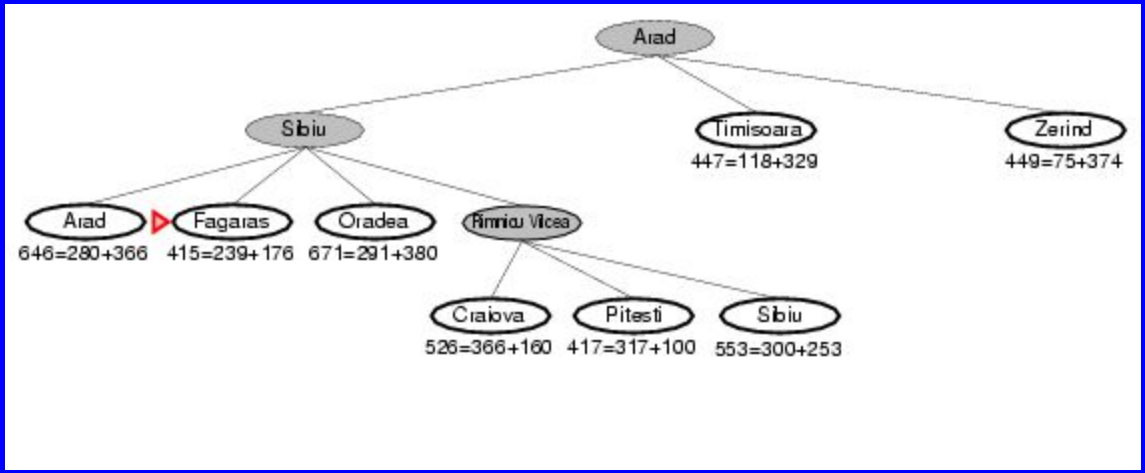
A* search example



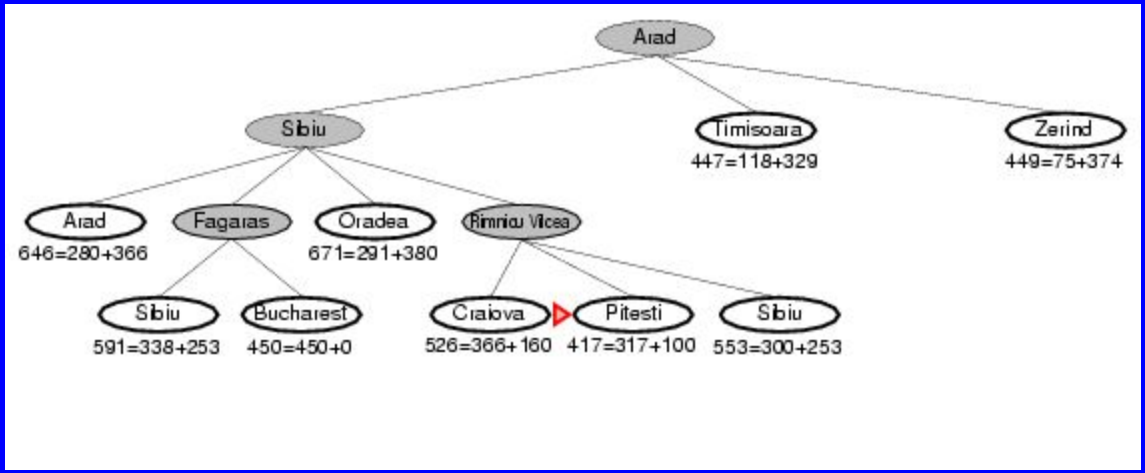
A* search example



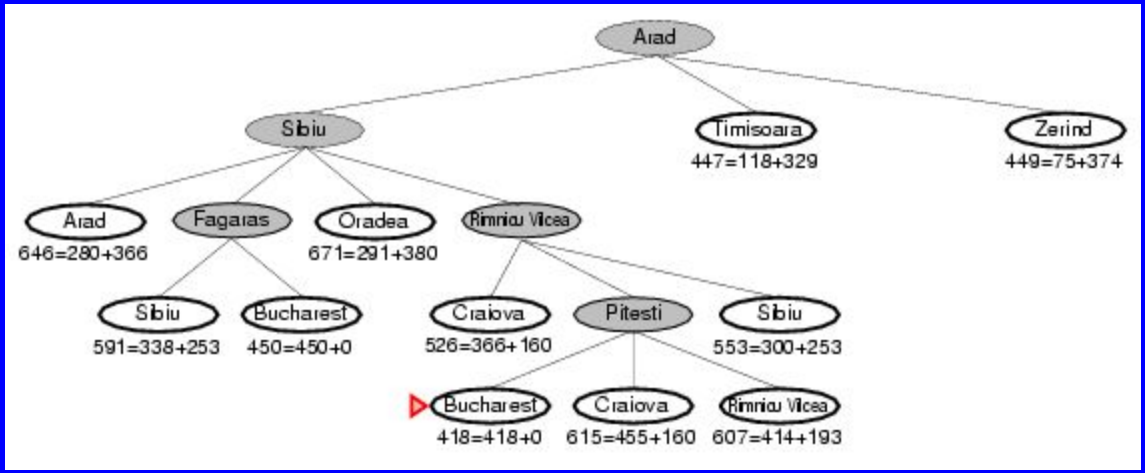
A* search example



A* search example



A* search example



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.
  - Special 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.

- **If the heuristic function, h always underestimate the true cost (**h(n) is smaller than h*(n)**), then A* is guaranteed to find an optimal solution.**

Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic

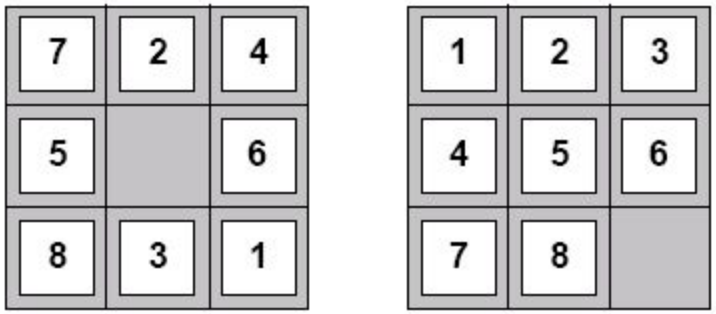
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
-

- Theorem: If $h(n)$ is admissible, A* using `TREE-SEARCH` is optimal

Admissible 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)

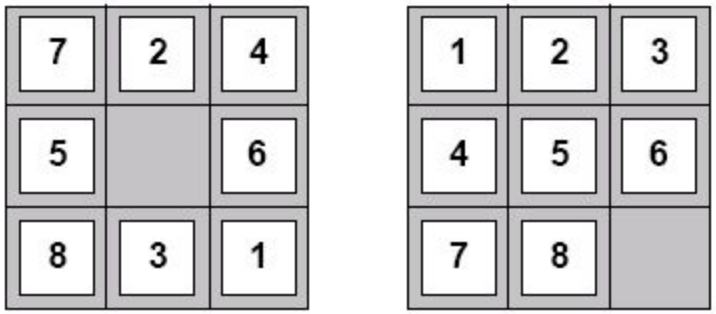


$h_1(S)$ =??
$h_2(S)$ =??

Admissible heuristics

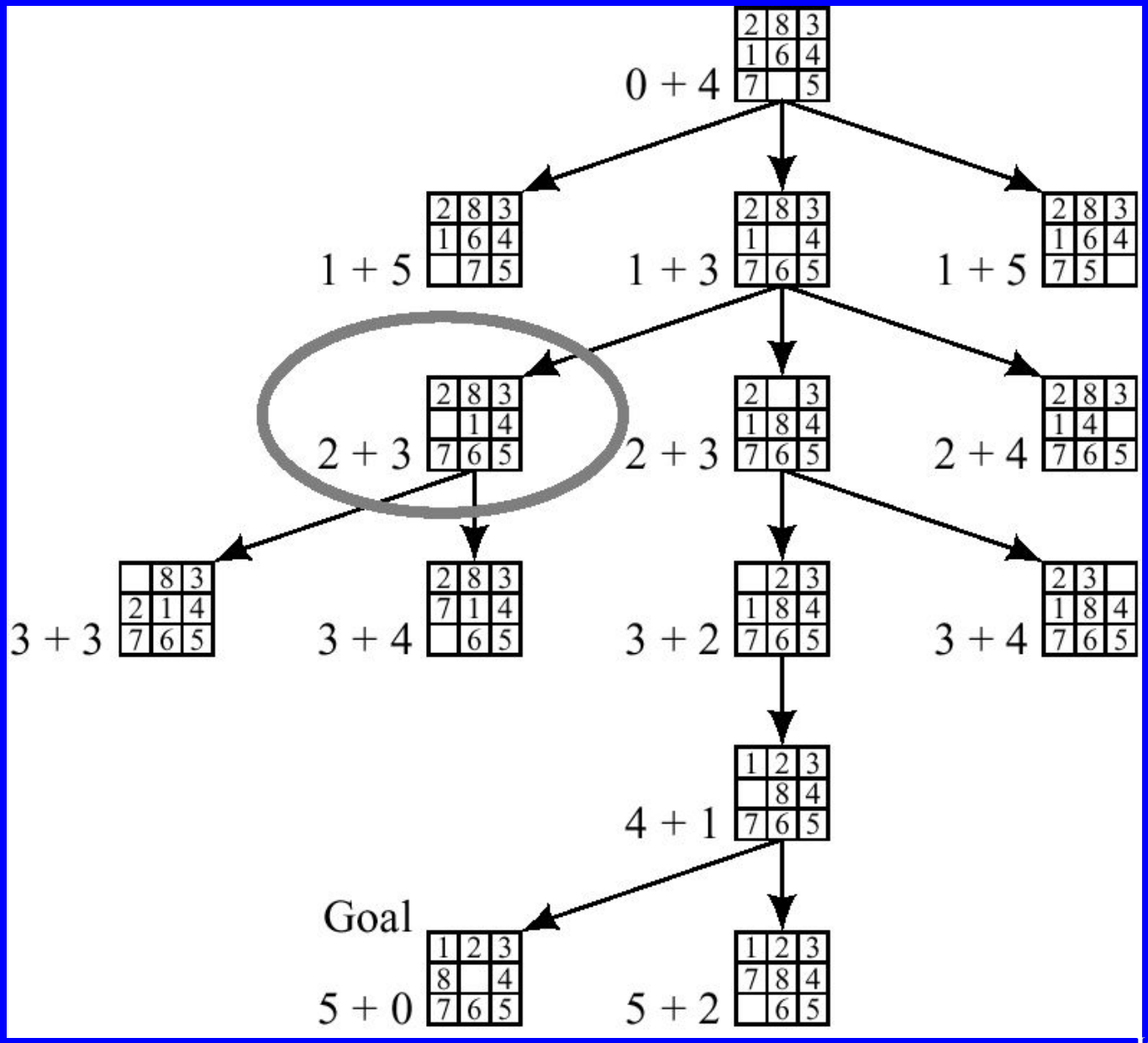
E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



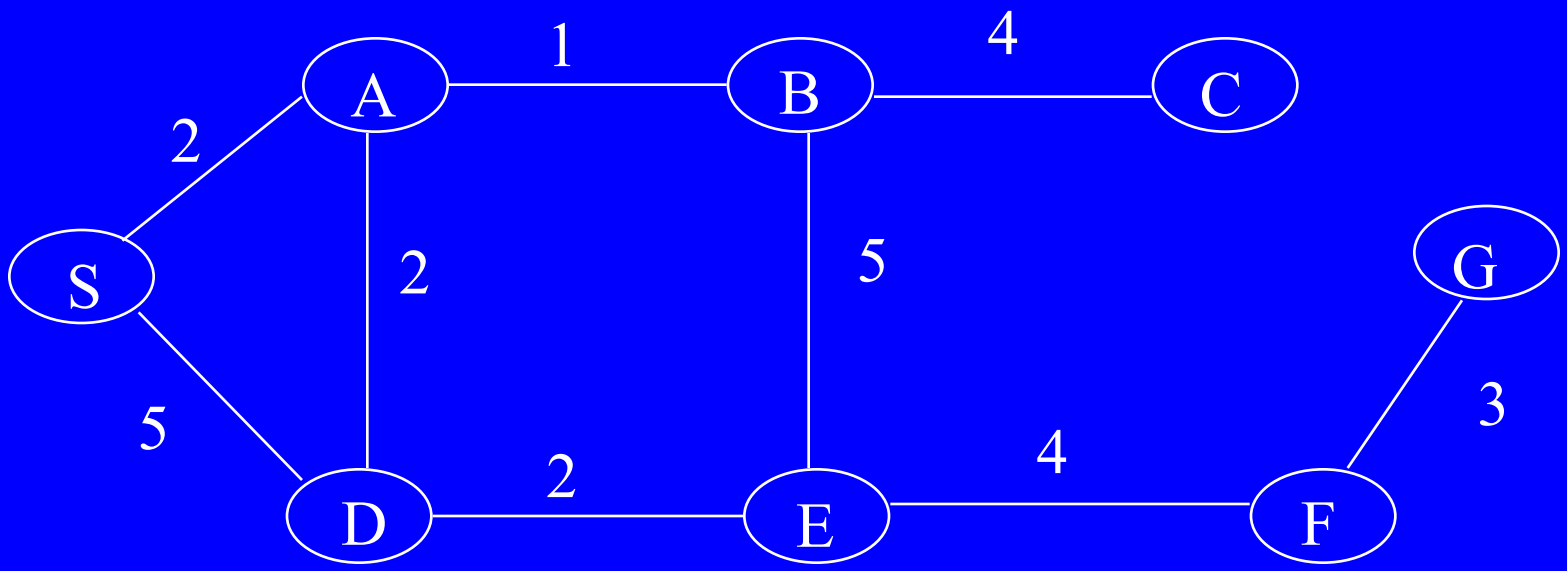
$h_1(S) =$?? 7
$h_2(S) =$?? 4+0+3+3+1+0+2+1 = 14

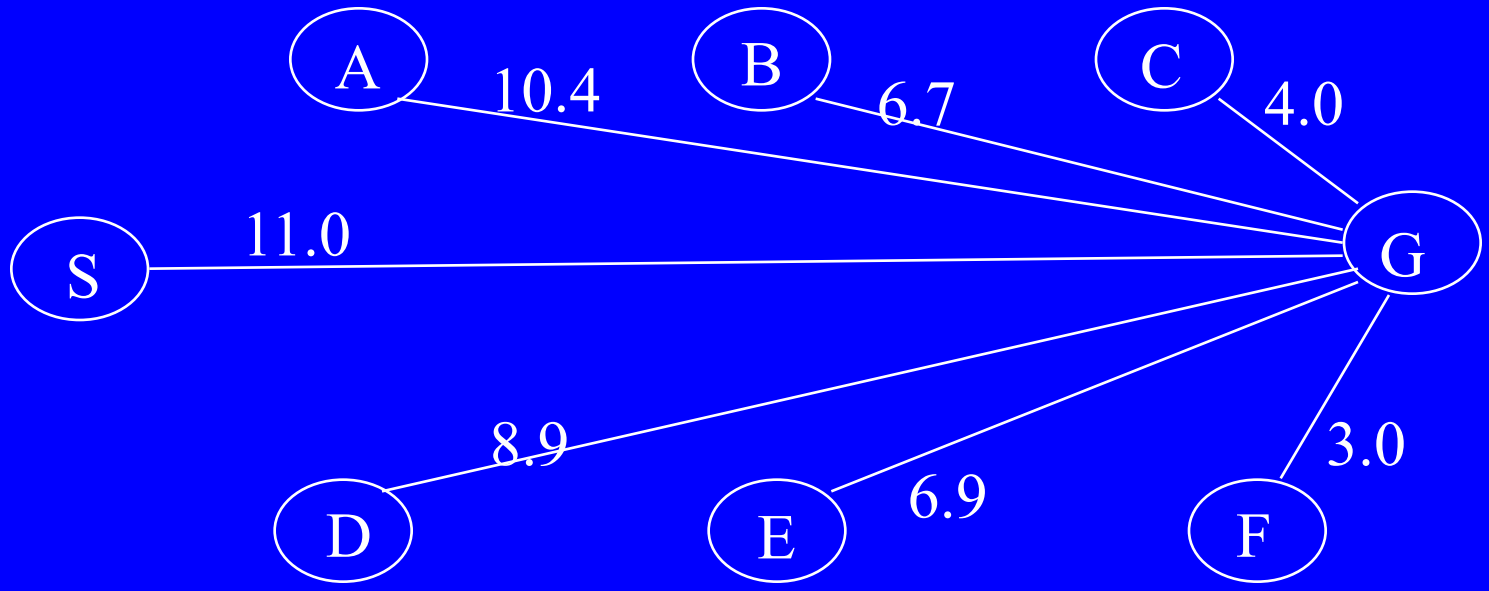
A* on 8-puzzle with h(n) = w(n)



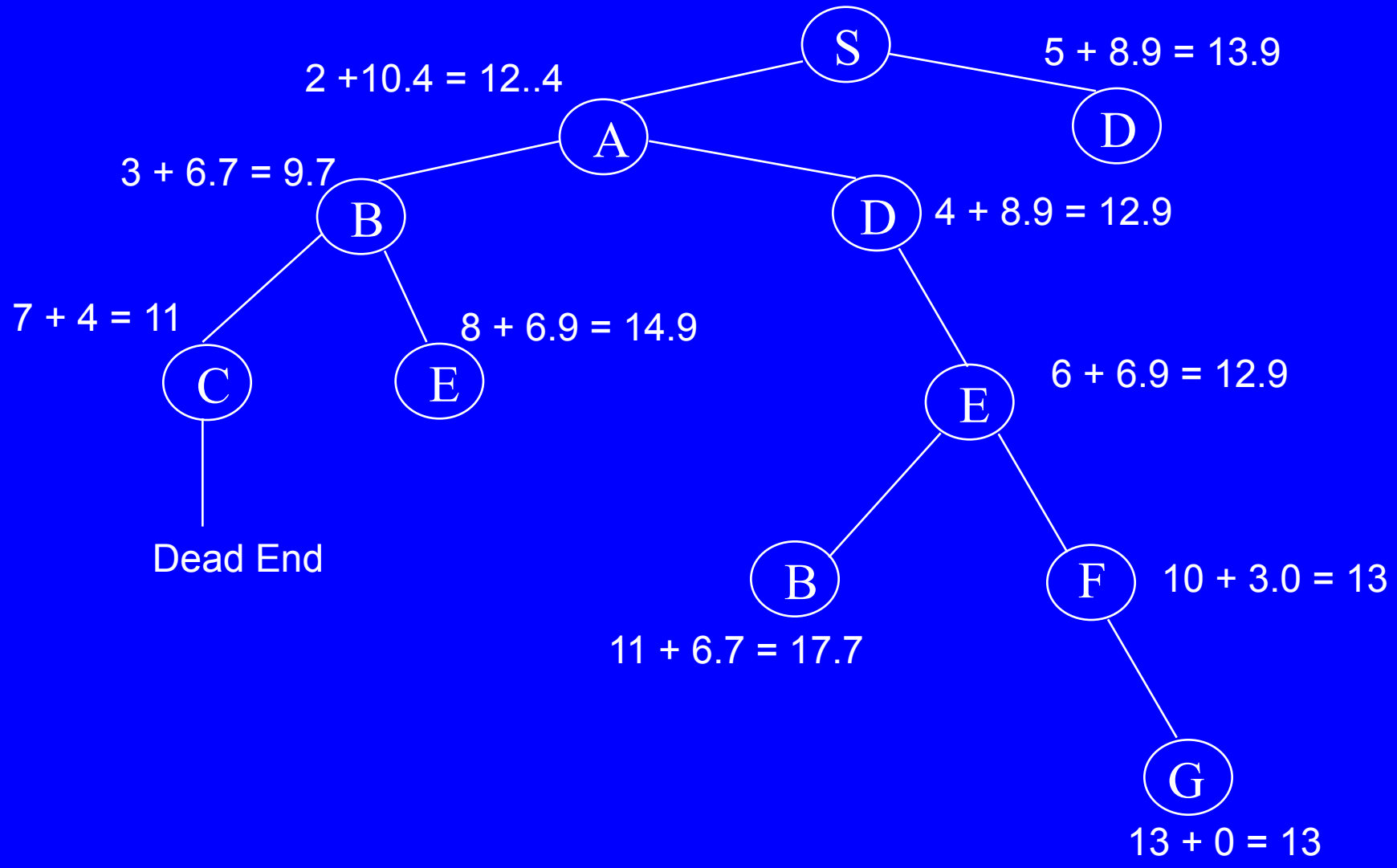
Algorithm A* (with any h on search Graph)

- **Input:** a 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 and attach the lowest f to n'.
      - put n' with its f value in the right order in OPEN
  - 6. Go to step 2.


A
B
C
S
D
E
F
G
1
4
2
2
5
5
2
4
3


A
B
C
S
D
E
F
G
10.4
6.7
4.0
11.0
8.9
6.9
3.0

Example of A* Algorithm in action



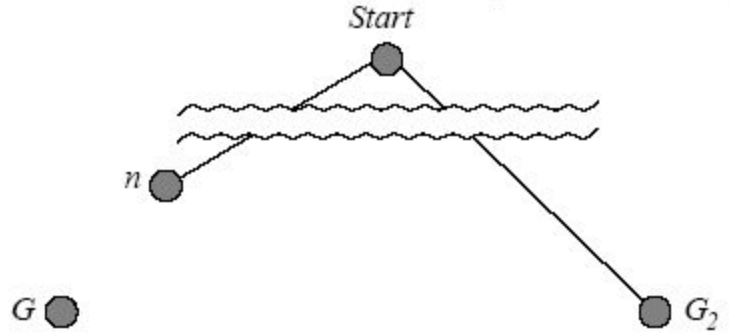
Behavior of A* - Completeness

- **Theorem (completeness for optimal solution) (HNL, 1968):**
  - If the heuristic function is admissible than A* finds an optimal solution.

- **Proof:**
  - 1. A* will expand only nodes whose f-values are less (or equal) to the optimal cost path C* (f(n) less-or-equal c*).
  - 2. The evaluation function of a goal node along an optimal path equals C*.
- **Lemma:**
  - Anytime before A* terminates there exists and OPEN node n' on an optimal path with f(n') <= C*.

Optimality of A* (standard proof)

Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G_1.

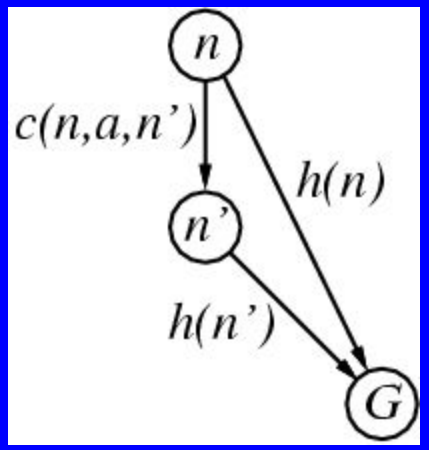


$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Consistent 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

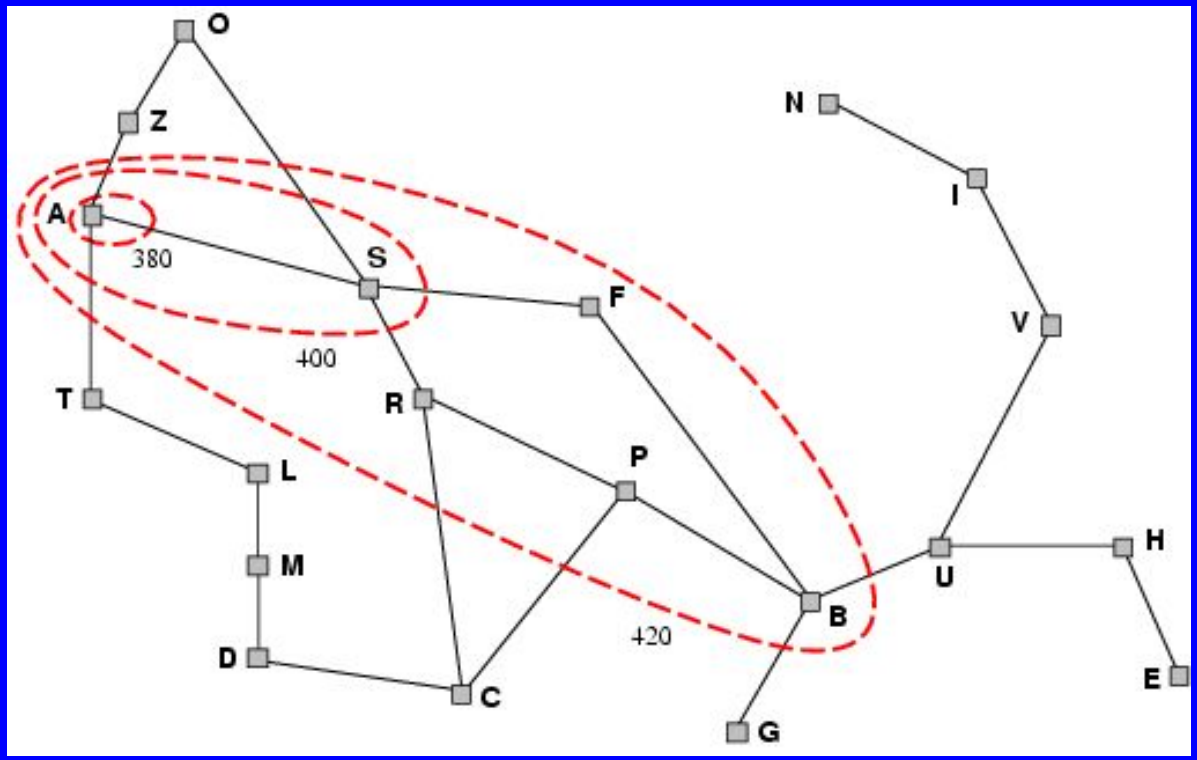
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

- i.e., *f(n)* is non-decreasing along any path.
- Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal

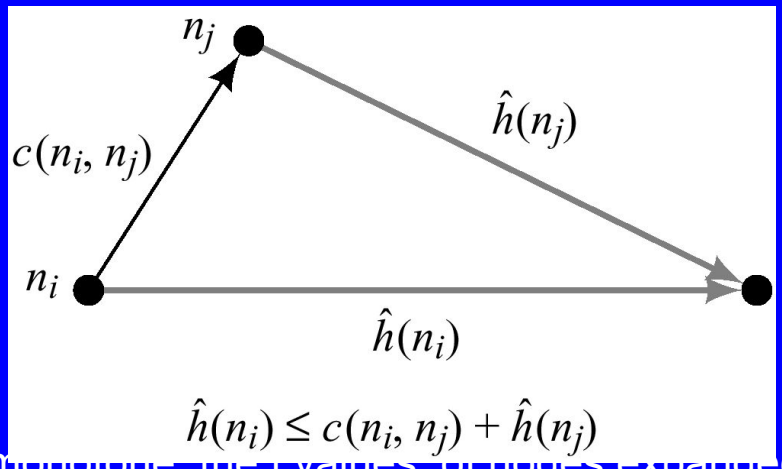
Optimality of A* with consistent h

- 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: Consistent (Monotone) Heuristics

- If in the search graph the heuristic function satisfies triangle inequality for every n and its child node n': h^(ni) less or equal h^(nj) + c(ni,nj)
  - 

- when h is monotone, 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 monotone every node is expanded once (if check for duplicates).
- Normally the heuristics we encounter are monotone
  - the number of misplaced ties
  - Manhattan distance
  - air-line distance

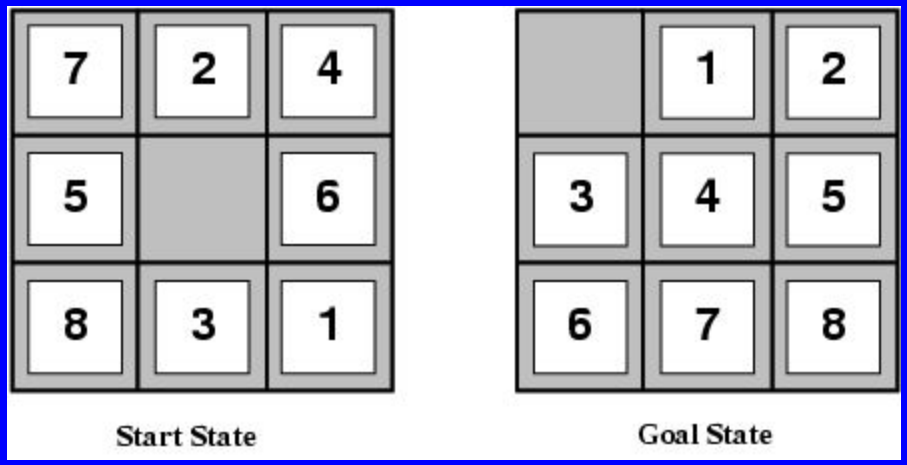
Admissible heuristics

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)

- $h_1(S) = ?$
- $h_2(S) = ?$



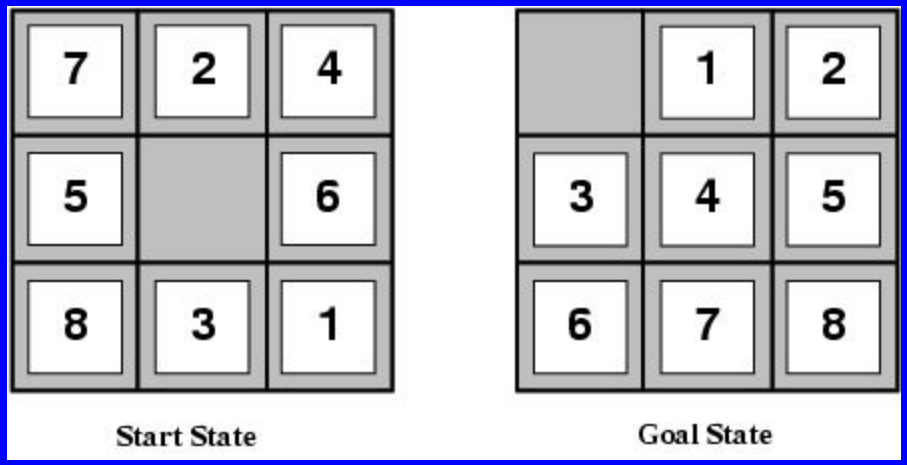
Admissible heuristics

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)

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



Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
- then h_2 dominates h_1
- h_2 is 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 = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

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**
- **Effective branching factor:**
  - logarithm of base (d+1) of average number of nodes expanded.

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

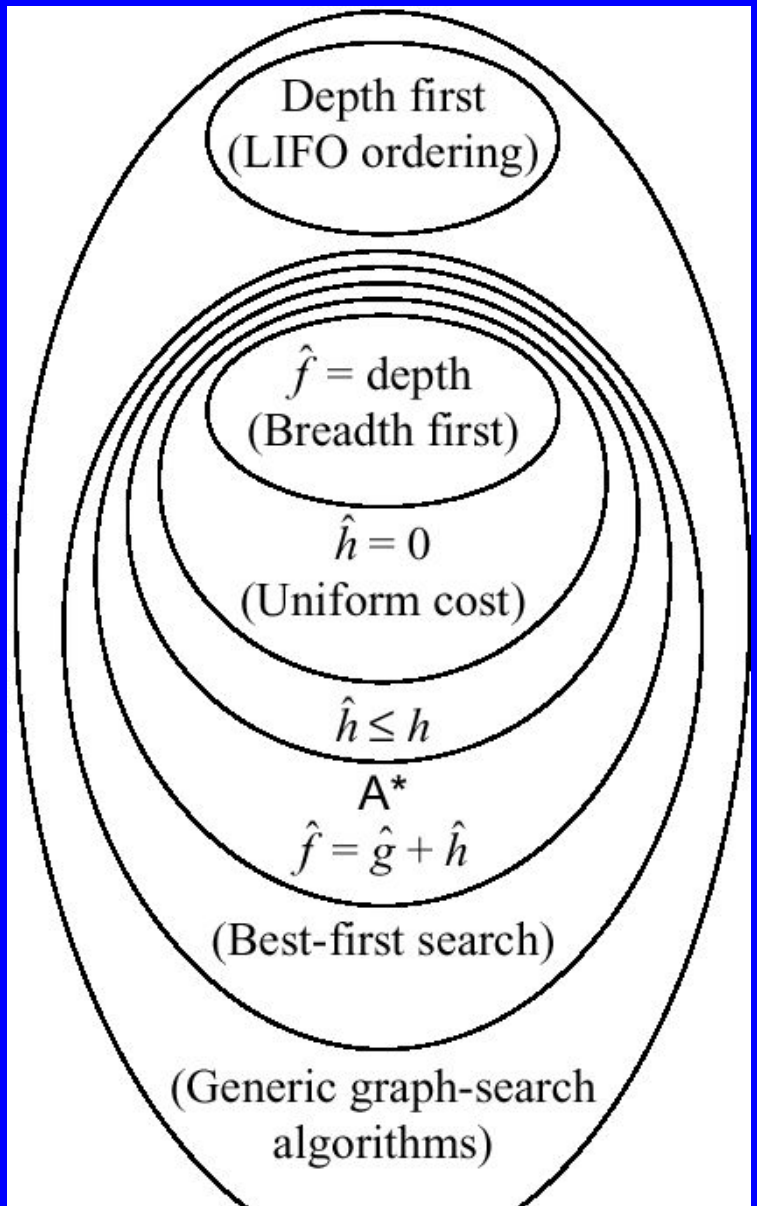
Average over 100 randomly generated 8-puzzle problems
h1 = number of tiles in the wrong position
h2 = sum of Manhattan distances

Properties of A*

- Complete? Yes (unless there are infinitely many nodes with f ≤ *f(G)*)
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

- A* expands all nodes having f(n) < C*
- A* expands some nodes having f(n) = C*
- A* expands no nodes having f(n) > C*

Relationships among search algorithms



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

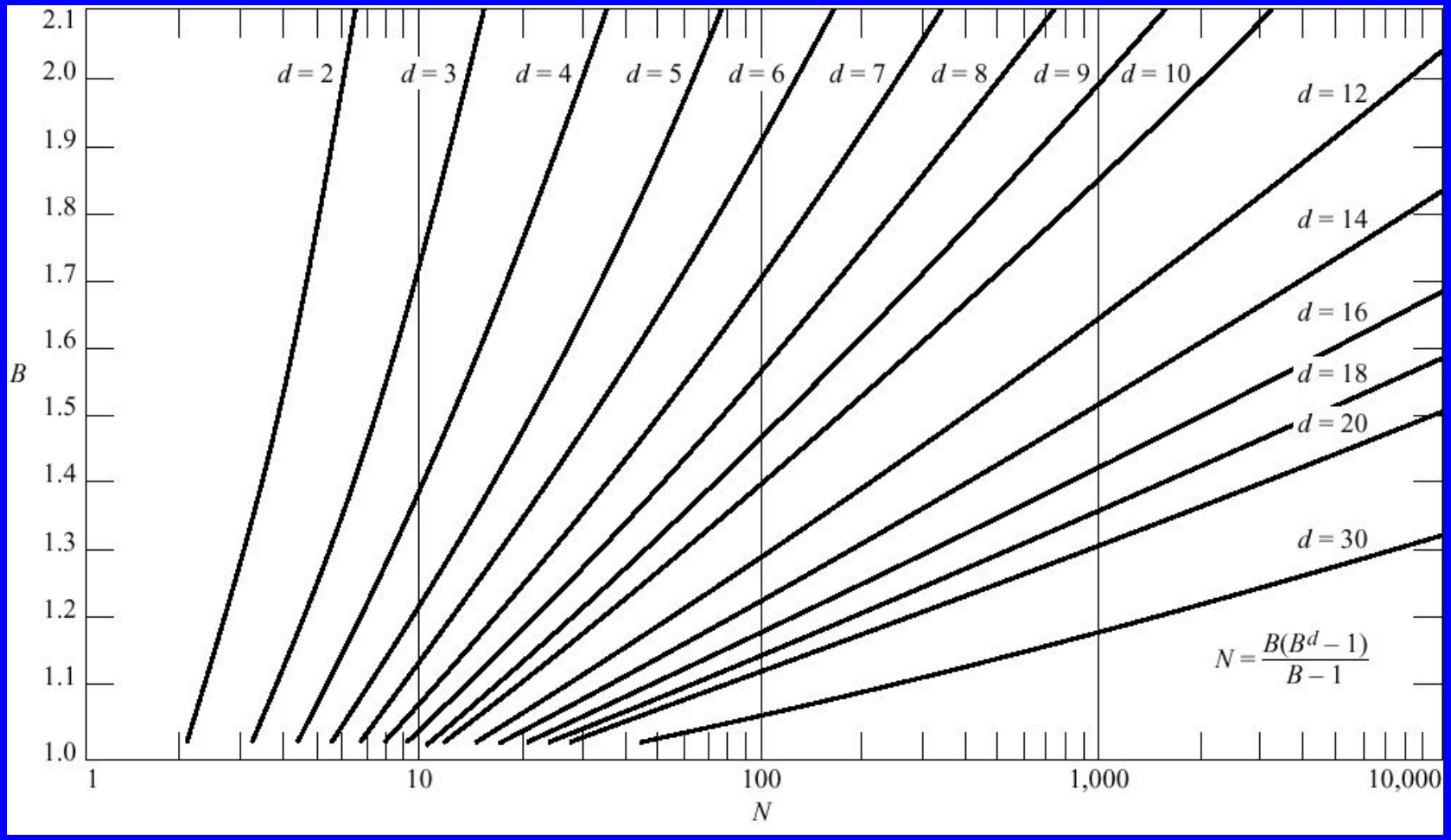
```
Initialize: Let Q = {S}
While Q is not empty
        pull Q1, the first element in Q
        if Q1 is a goal compute the cost of the solution and update
                L <-- minimum between new cost and old cost
        else
                child_nodes = expand(Q1),
                        <eliminate 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 evaluation function, f.

        end
Continue
```

Properties of Branch-and-Bound

- **Not guaranteed to terminate unless has depth-bound**
- **Optimal:**
  - finds an optimal solution
- **Time complexity: exponential**
- **Space complexity: linear**

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

- **Initialize:** f <-- the evaluation function of the start node
- **until goal node is found**
  - Loop:
    - Do Branch-and-bound with upper-bound L equal current evaluation function
    - Increment evaluation function to next contour level
  - end
- **continue**
- **Properties:**
  - Guarantee to find an optimal solution
  - time: exponential, like A*
  - space: linear, like B&B.


2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
B
d = 2
d = 3
d = 4
d = 5
d = 6
d = 7
d = 8
d = 9
d = 10
d = 12
d = 14
d = 16
d = 18
d = 20
d = 30
$N = \frac{B(B^d - 1)}{B - 1}$
1
10
100
1,000
10,000
N

Inventing Heuristics automatically

- **Examples of Heuristic Functions for A***
  - the 8-puzzle problem
    - the number of tiles in the wrong position
      - is this admissible?
    - the sum of distances of the tiles from their goal positions, where distance is counted as the sum of vertical and horizontal tile displacements ("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 (continued)

- **How did we**
  - find h1 and h2 for the 8-puzzle?
  - verify admissibility?
  - prove that air-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:**
  - A tile can be moved from A to B if A is adjacent to B and B is clear
  - We can generate relaxed problems by removing one or more of the conditions
    - A tile can be moved from A to B if A is adjacent to B
    - ...if B is blank
    - A tile can be moved from A to B.

Generating heuristics (continued)

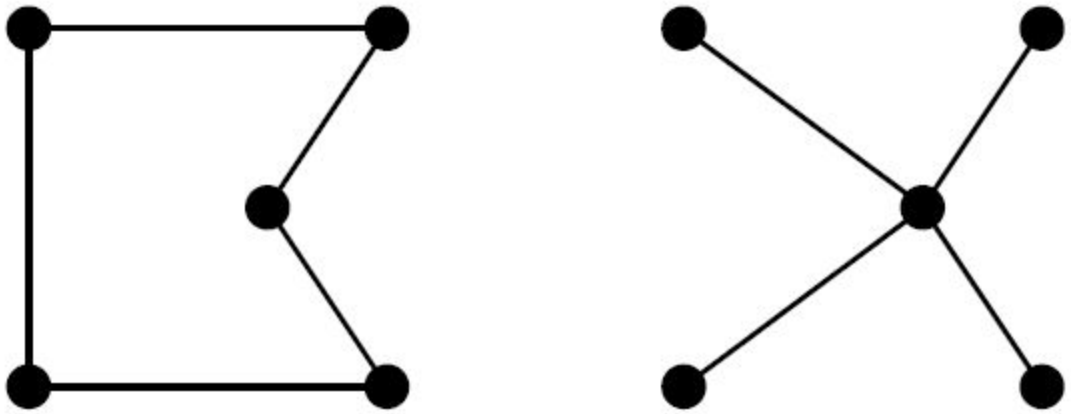
- **Example: TSP**
- **Finr a tour. A tour is:**
  - 1. A graph
  - 2. Connected
  - 3. Each node has degree 2.
- **Eliminating 2 yields MST.**

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 $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

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 representation:**
- **Operators:**
  - Pre-conditions, add-list, delete list
- **8-puzzle example:**
  - On(x,y), clear(y) adj(y,z) ,tiles x1,…,x8
- **States: conjunction of predicates:**
  - On(x1,c1),on(x2,c2)….on(x8,c8),clear(c9)
- **Move(x,c1,c2) (move tile x from location c1 to location c2)**
  - Pre-cond: on(x1.c1), clear(c2), adj(c1,c2)
  - Add-list: on(x1,c2), clear(c1)
  - Delete-list: on(x1,c1), clear(c2)
- **Relaxation:**
- **1. Remove from prec-dond: clear(c2), adj(c2,c3) □ #misplaced tiles**
- **2. Remove clear(c2) □ manhatten distance**
- **3. Remove adj(c2,c3) □ h3, a new procedure that transfer to the empty location a tile appearing there in the goal**

Heuristic generation

- **The space of relaxations can be enriched by predicate refinements**
- **Adj(y,z) iff neigbour(y,z) and same-line(y,z)**

- **The main question: how to recognize a relaxed problem which is easy.**
- **A proposal:**
  - A problem is easy if it can be solved optimally by agreedy algorithm
- **Heuristics that are generated from relaxed models are monotone.**

- **Proof: h is true shortest path l relaxed model**
  - H(n) <=c'(n,n')+h(n')
  - C'(n,n') <=c(n,n')
  - □ h(n) <= c(n,n')+h(n')
- **Problem: not every relaxed problem is easy, often, a simpler problem which is more constrained will provide a good upper-bound.**

Improving Heuristics

- If we have several heuristics which are non dominating we can select the max value.
- Reinforcement learning.

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

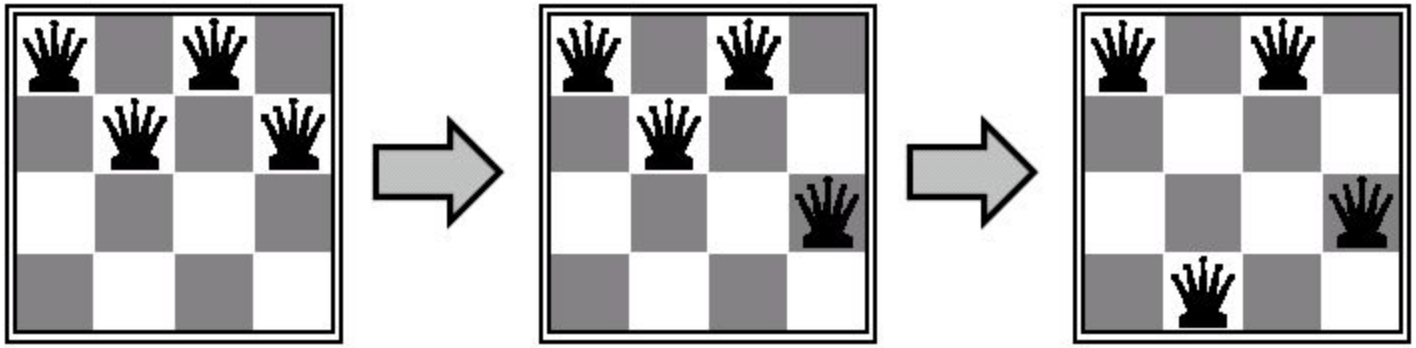
- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

- Constant space. Good for offline and online search

Example: n-queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Move a queen to reduce number of conflicts



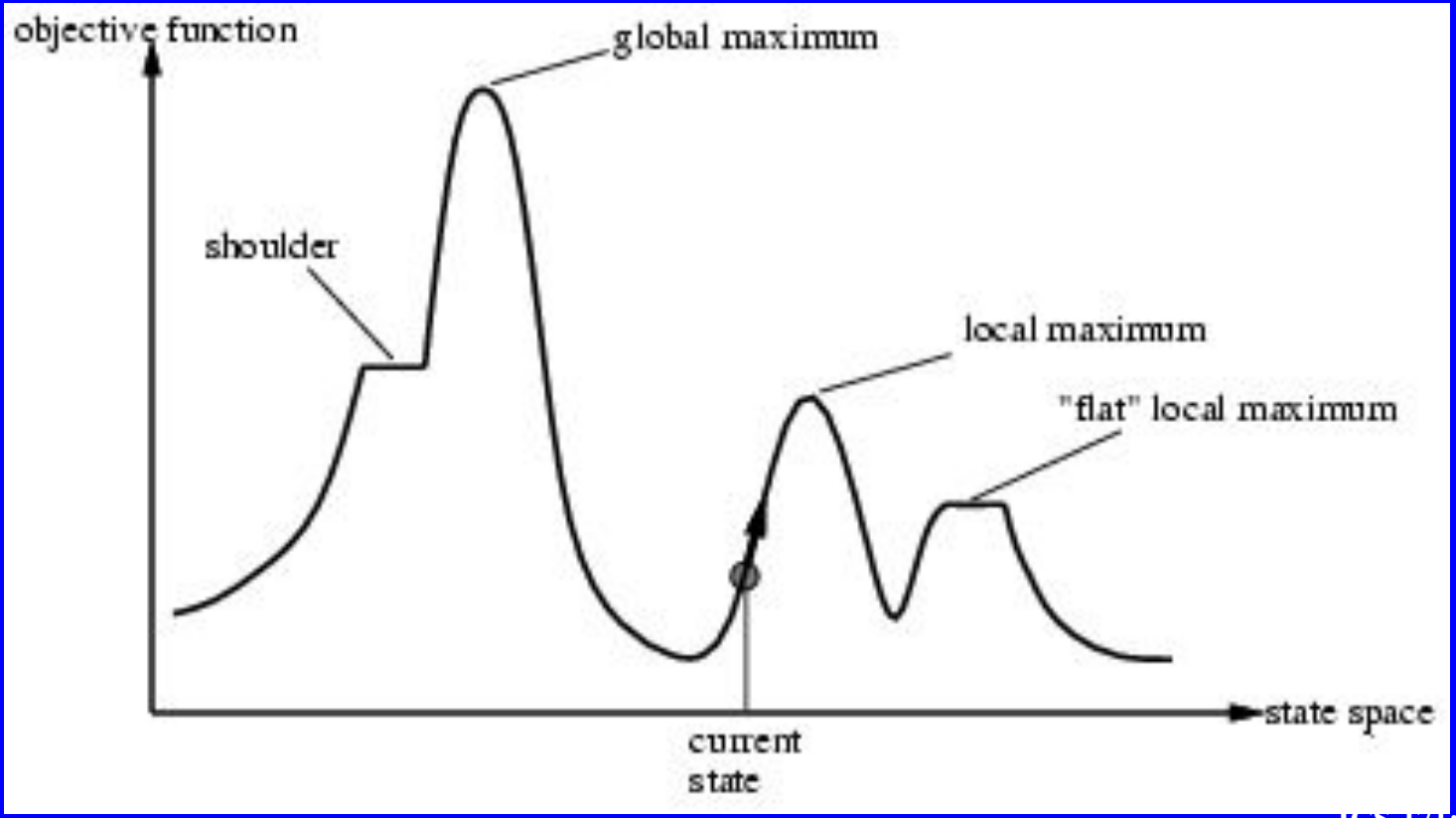
Hill-climbing search

- "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
```

Hill-climbing search

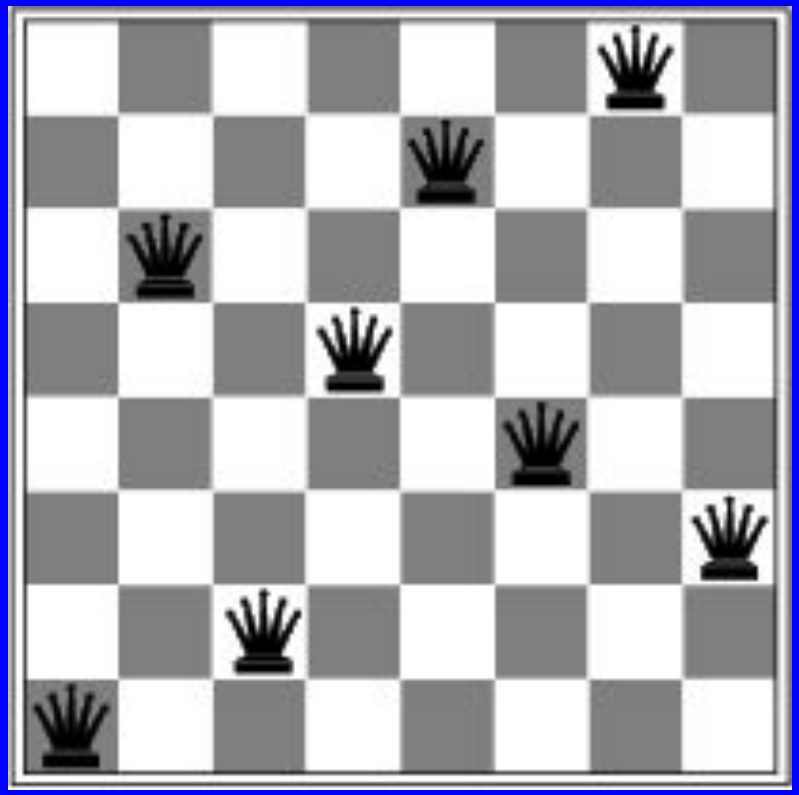
- **Problem: depending on initial state, can get stuck in local maxima**



Hill-climbing search: 8-queens problem

- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h* = *17* for the above state

Hill-climbing search: 8-queens problem



- A local minimum with $h = 1$

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(Δ E/T)
```

Properties of simulated annealing search

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc