Intelligent Search

EBE Digital Business track

TILBURG UNIVERSITY

Understanding Society

What is Search?

Search is a class of techniques for systematically finding or constructing solutions to problems.

Example technique: generate-and-test.

Example problem: Combination lock.

Generate-and-test:

1. Generate a possible solution.
2. Test the solution.
3. If solution found THEN done ELSE return to step 1.

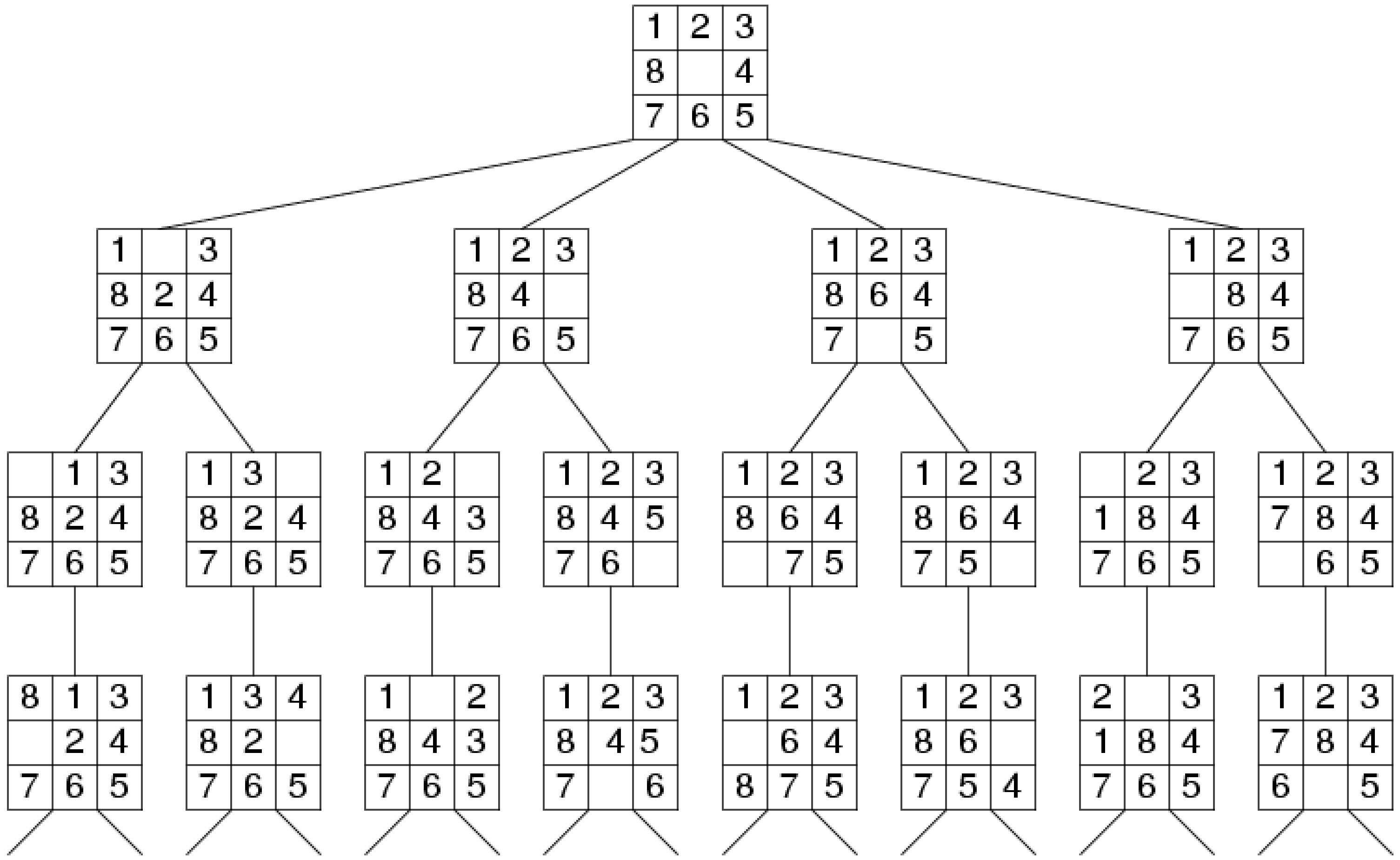
Why is search interesting?

- Many (all?) AI problems can be formulated as search problems!

Examples:

- Labyrinth
- Path planning
- Games
- Natural Language Processing
- Machine learning
- Genetic algorithms

Search Tree Example:
Fragment of 8-Puzzle Problem Space



Search through a Problem Space/ State Space

Input:

- Set of states
- Operators [and costs]
- Start state
- Goal state [test]

Output:

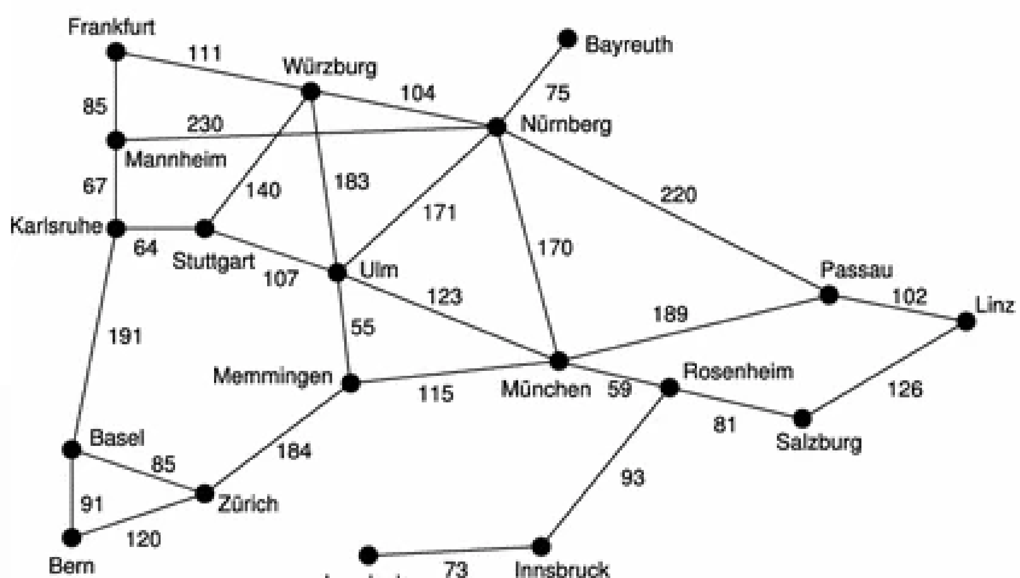
- Path: start $\Rightarrow$ a state satisfying goal test
- [May require shortest path]

Example: Route Planning

Input:

- Set of states
- Operators [and costs]
- Start state
- Goal state (test)

Output?

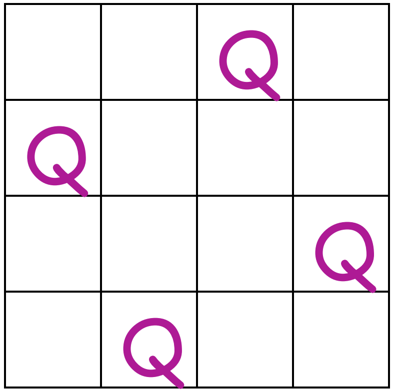


Basel	204
Bayreuth	207
Bern	247
Frankfurt	215
Innsbruck	163
Karlsruhe	137
Landeck	143
Linz	318
München	120
Mannheim	164
Memmingen	47
Nürnberg	132
Passau	257
Rosenheim	168
Stuttgart	75
Salzburg	236
Würzburg	153

Example: N Queens

Input:

- Set of states
- Operators [and costs]
- Start state
- Goal state (test)



Output?

Classifying Search

- GUESSING ("Tree Search")
  - Guess how to extend a partial solution to a problem.
  - Generates a tree of (partial) solutions.
  - The leaves of the tree are either "failures" or represent complete solutions
- SIMPLIFYING ("Inference")
  - Infer new, stronger constraints by combining one or more constraints (without any "guessing")

    Example: $X+2Y = 3$

    $X+Y = 1$

    subtract $Y = 2$
- WANDERING ("Markov chain")
  - Perform a (biased) random walk through the space of (partial or total) solutions

Search Strategies

- Blind Search
    - Depth first search
    - Breadth first search
    - Iterative deepening search
    - Iterative broadening search
- Informed Search
- Constraint Satisfaction
- Adversary Search

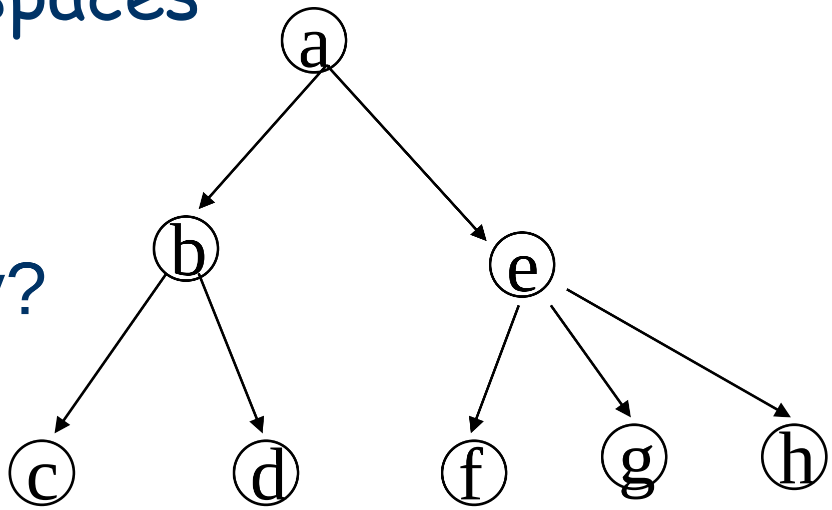
Depth First Search

- Maintain stack of nodes to visit
- Evaluation
  - Complete?

    Not for infinite spaces
  - Time Complexity?

    O(b^d)
  - Space Complexity?

    O(d)



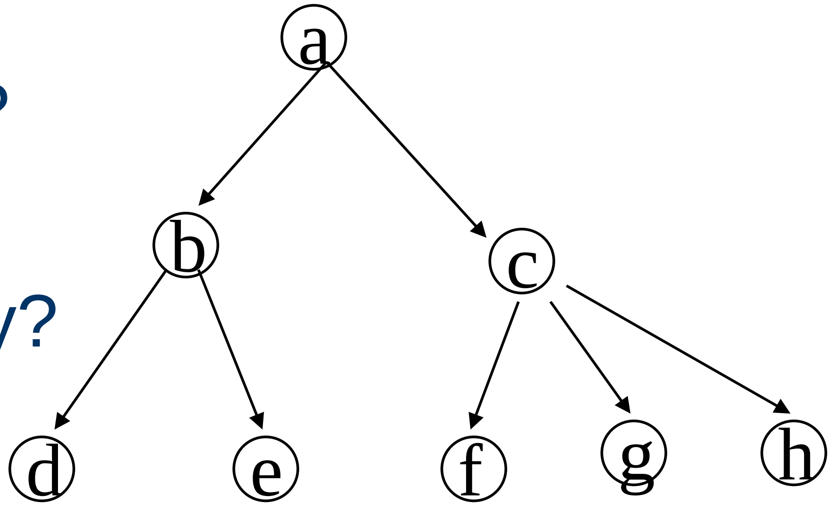
Breadth First Search

- Maintain queue of nodes to visit
- Evaluation
  - Complete?

    Yes
  - Time Complexity?

    O(b^d)
  - Space Complexity?

    O(b^d)



Tilburg University

Memory a Limitation?

- Suppose:

  2 GHz CPU

  1 GB main memory

  100 instructions / expansion

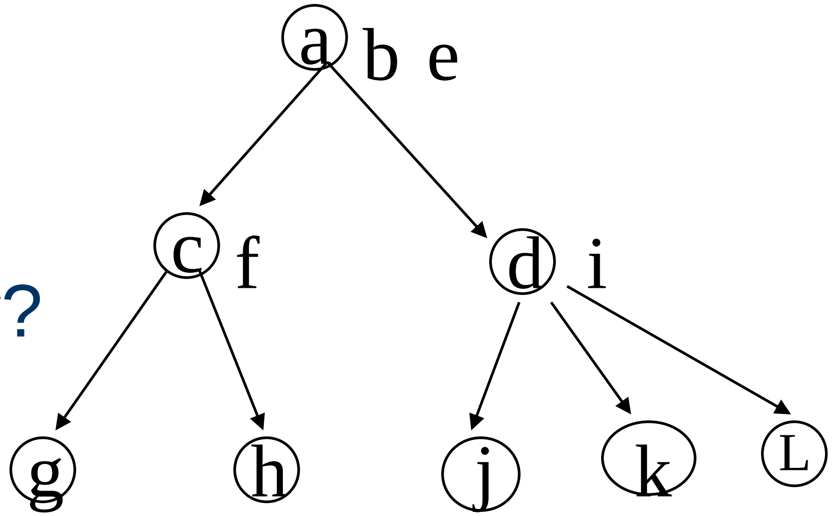
  5 bytes / node

  200,000 expansions / sec

  Memory filled in 100 sec … < 2 minutes

Iterative Deepening Search

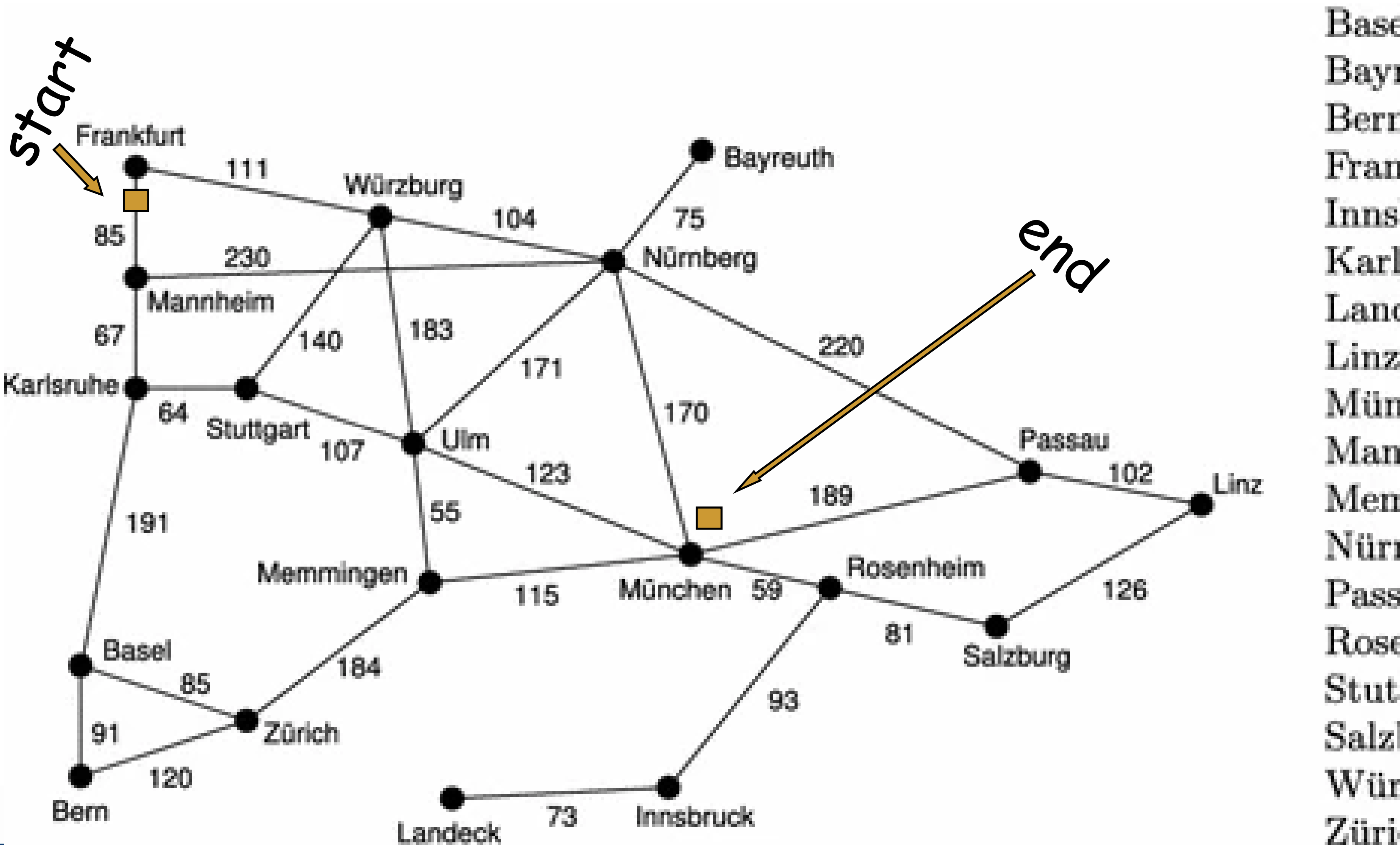
- DFS with limit; incrementally grow limit
- Evaluation
  - Complete?
    Yes
  - Time Complexity?
    $O(b^d)$
  - Space Complexity?
    $O(d)$



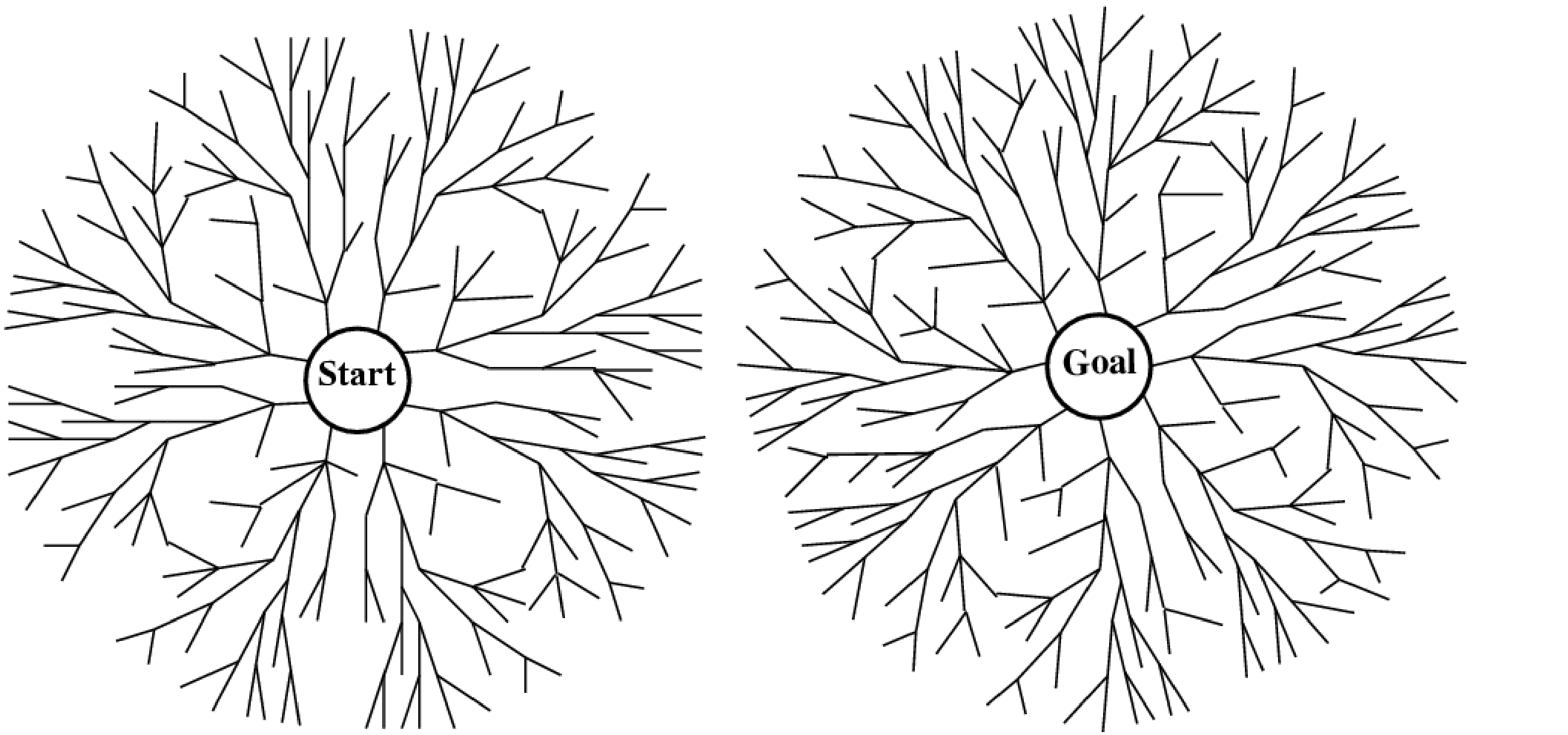
Cost of Iterative Deepening

b	ratio ID to DFS
2	3
3	2
5	1.5
10	1.2
25	1.08
100	1.02

Forwards *vs.* Backwards



vs. Bidirectional



Problem

- All these methods are slow (blind)

- Solution → add guidance ("heuristic estimate")

  → "informed search"

Intelligent Search

- Best-first search
  - Greedy best-first search
- Heuristics
  - **A* search**
- **Local search algorithms**
- Hill-climbing search
- **Simulated annealing search**
- Local beam search
- Genetic algorithms

Best-first search

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

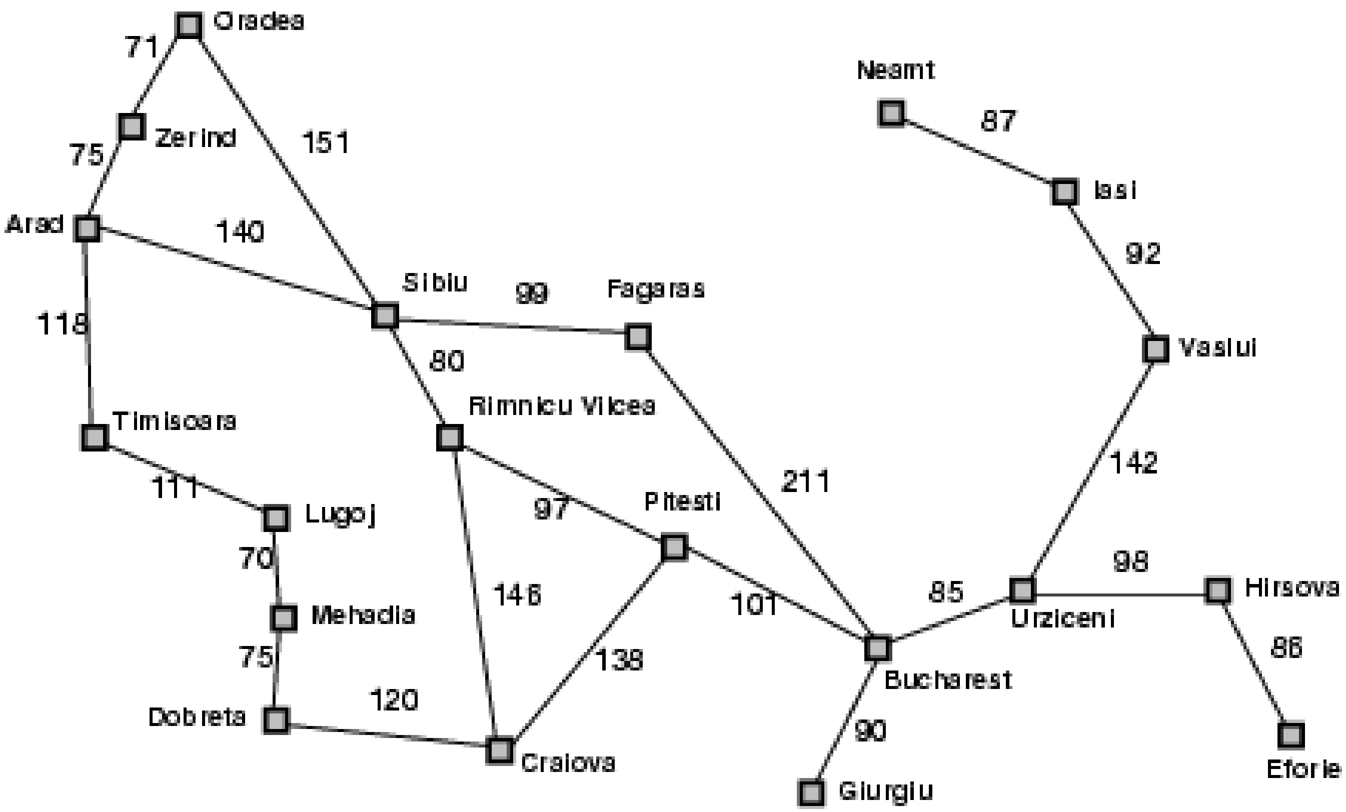
→Expand most desirable unexpanded node

- Implementation:

  Order the nodes in fringe in decreasing order of desirability

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

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)

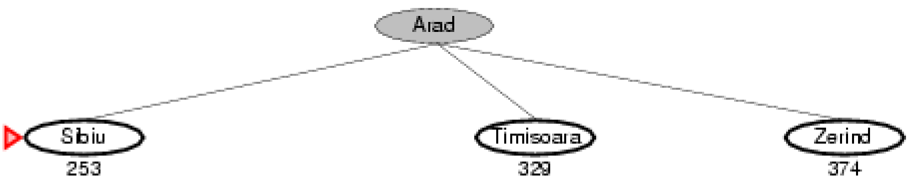
= estimate of cost from n to *goal*

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

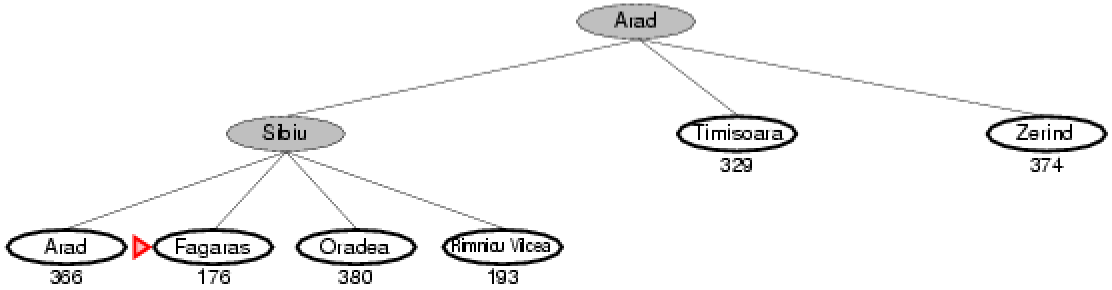
Greedy best-first search expands the node that appears to be closest to goal

Greedy best-first search example

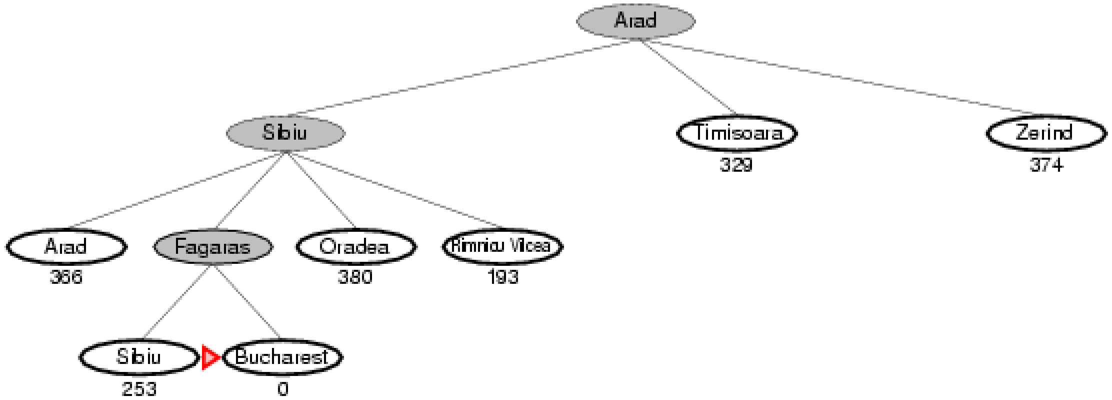
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example



Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

A* search

Idea: avoid expanding paths that are already expensive

Be greedy *and* reflective

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

Fig. 6:16. Two shapshots of A*: Frankfurt to Ulm

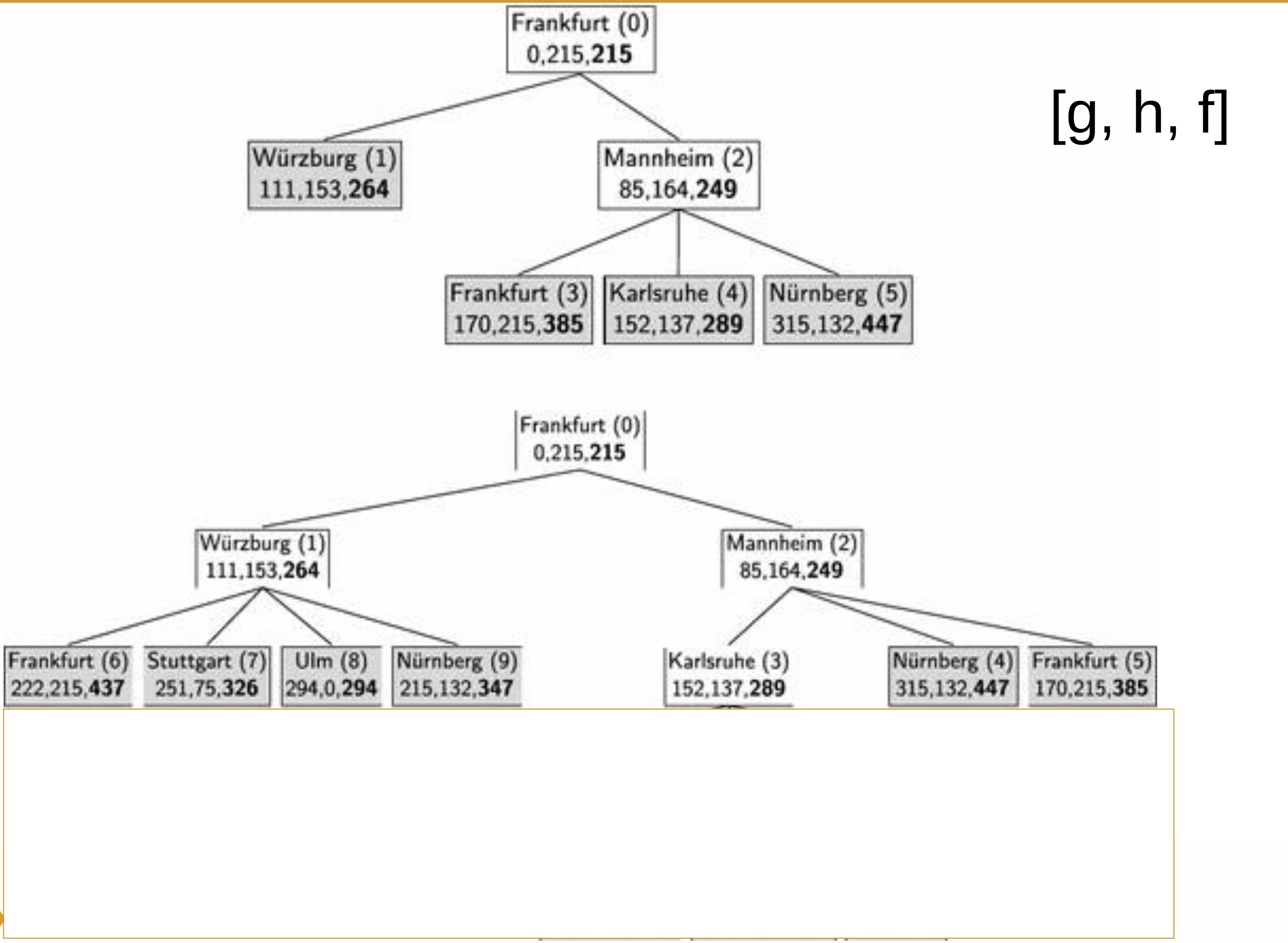
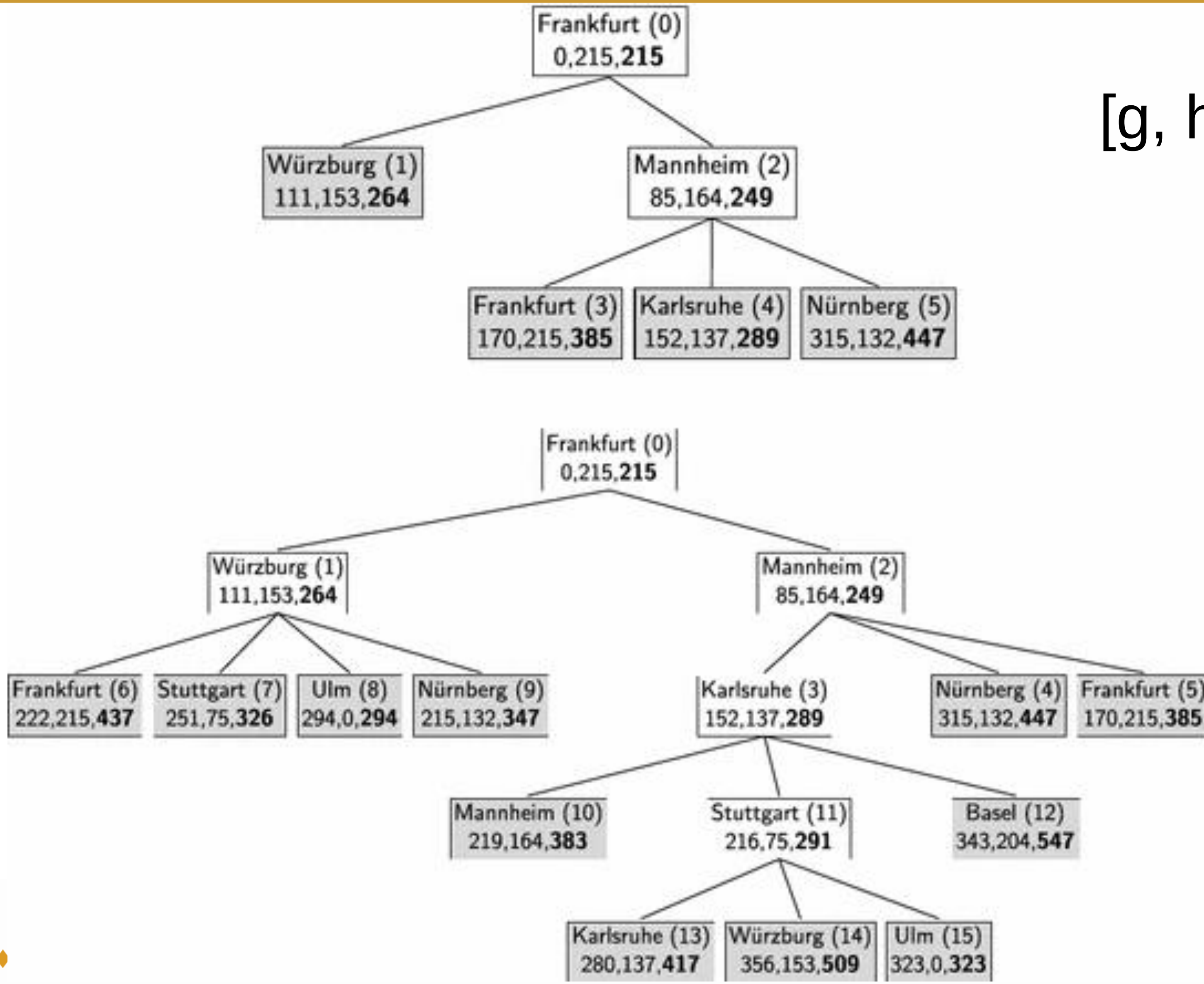


Fig. 6:16. Two shapshots of A*: Frankfurt to Ulm

[g, h, f]



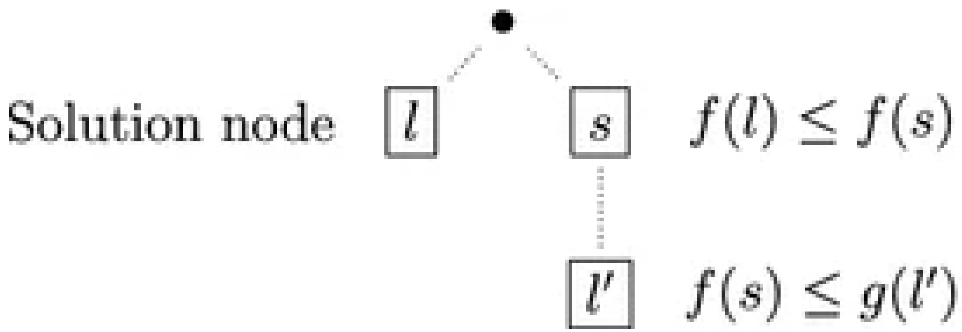
Admissible heuristics

- 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

Optimality of A^*

- **Theorem 6.2**
- *The $A^\star$ algorithm is optimal. That is, it always finds the solution with the lowest total cost if the heuristic h is admissible.*

Optimality of A* (proof)



If you choose the node with the lowest f (l in this case), and l is a solution, then it is also the optimal solution

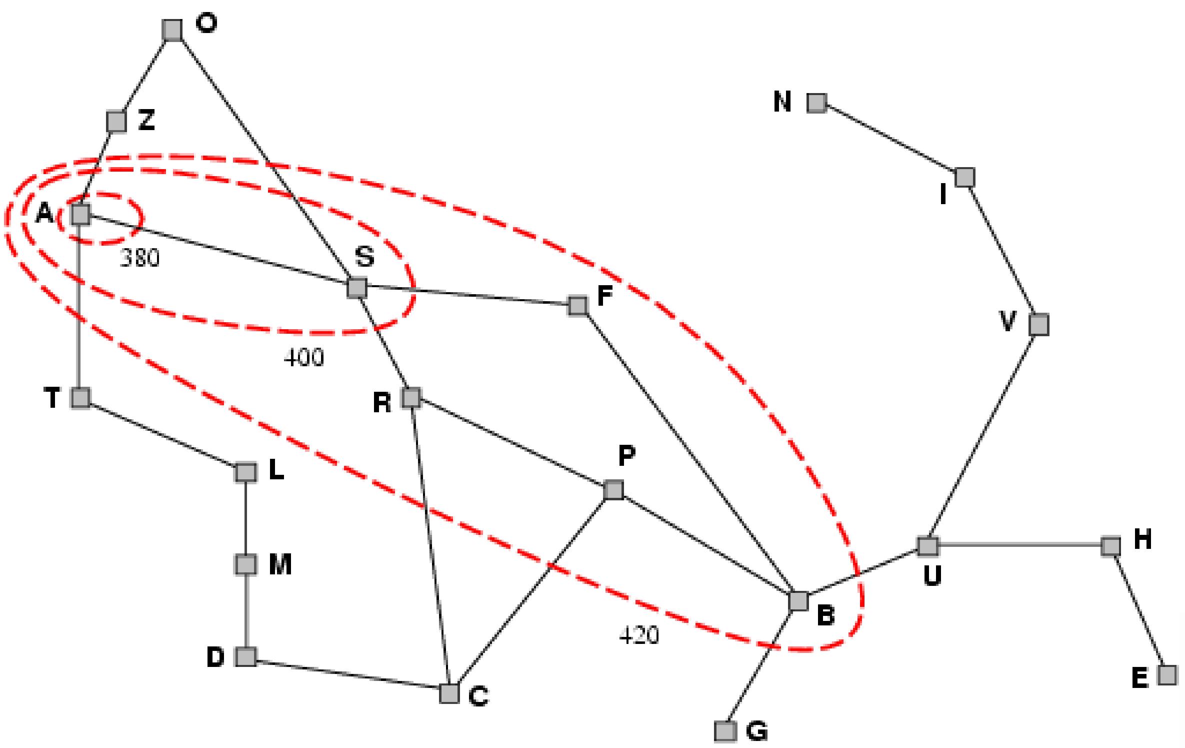
$$g(l) = g(l) + h(l) = f(l) <= f(s) = g(s) + h(s) <= g(l')$$

because

1) *l* is a solution node, so *h(l)* = 0
2) definition of *f*
3) definition of A*: *l* is the first to choose
4) definition of *f*
5) *h* is admissible

Optimality of A*

- 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 $f_i < f_{i+1}$



Properties of A*

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

- Time? Exponential

- Space? Keeps all nodes in memory

- Optimal? Yes

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)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

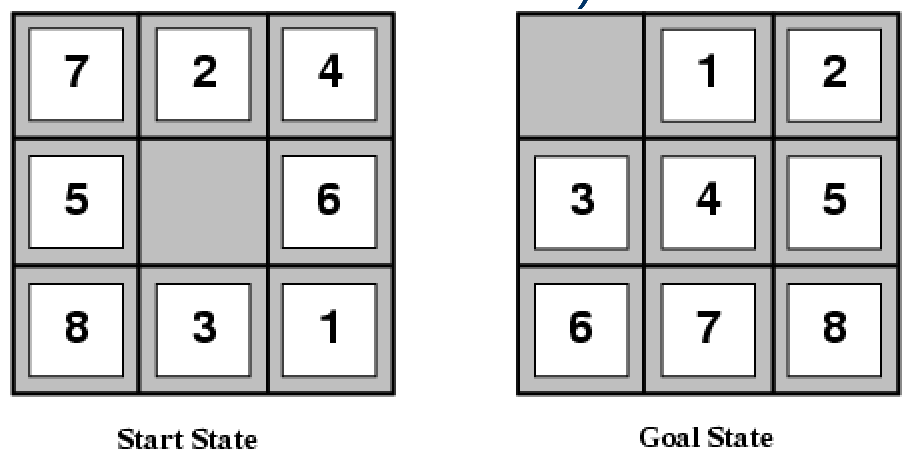
- $h_1(S) = ?$
- $h_2(S) = ?$

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

Relaxed problems

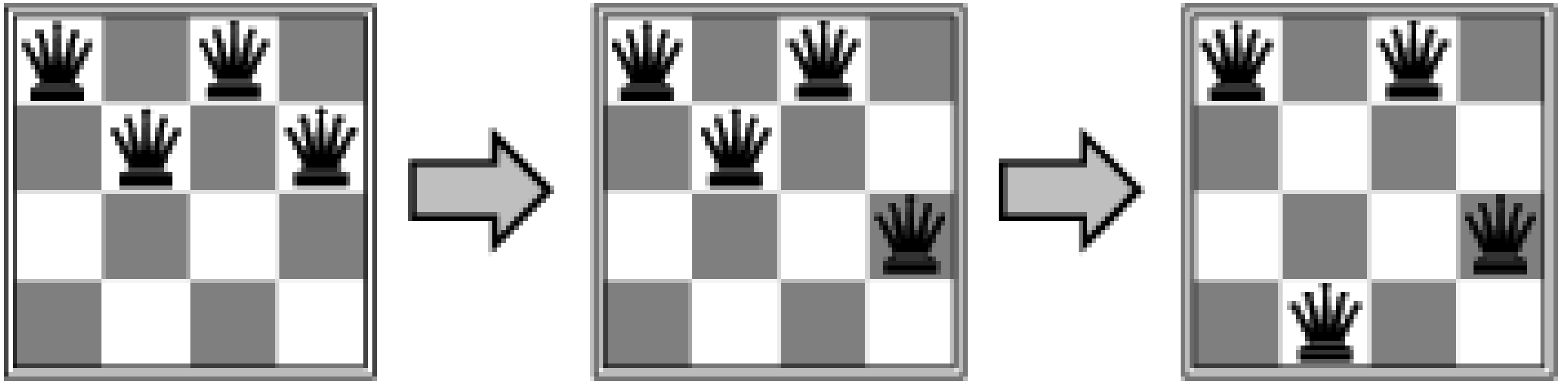
- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution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

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

Example: *n*-queens

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



Hill-climbing search

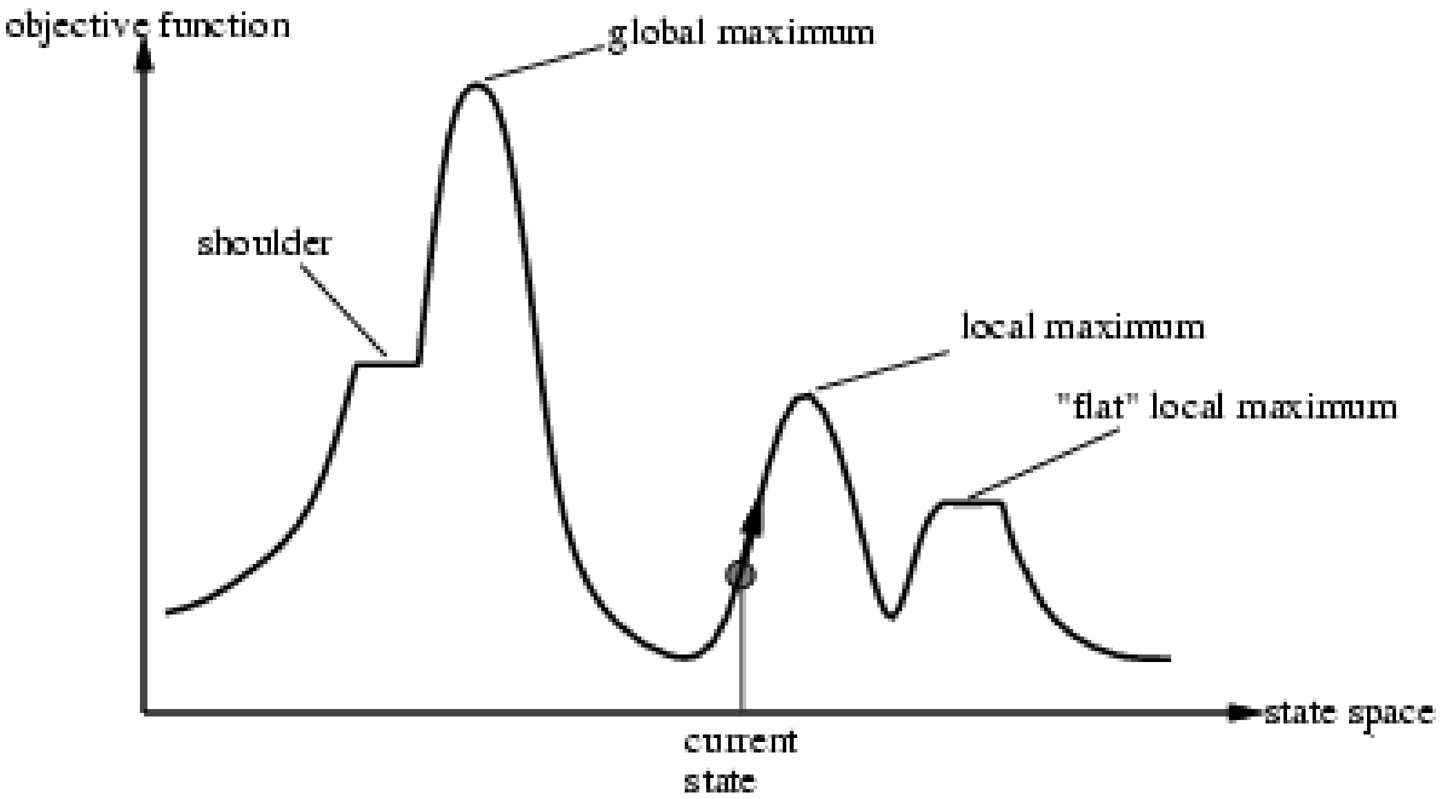
- "Like climbing Everest in thick fog with amnesia"

```
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
```

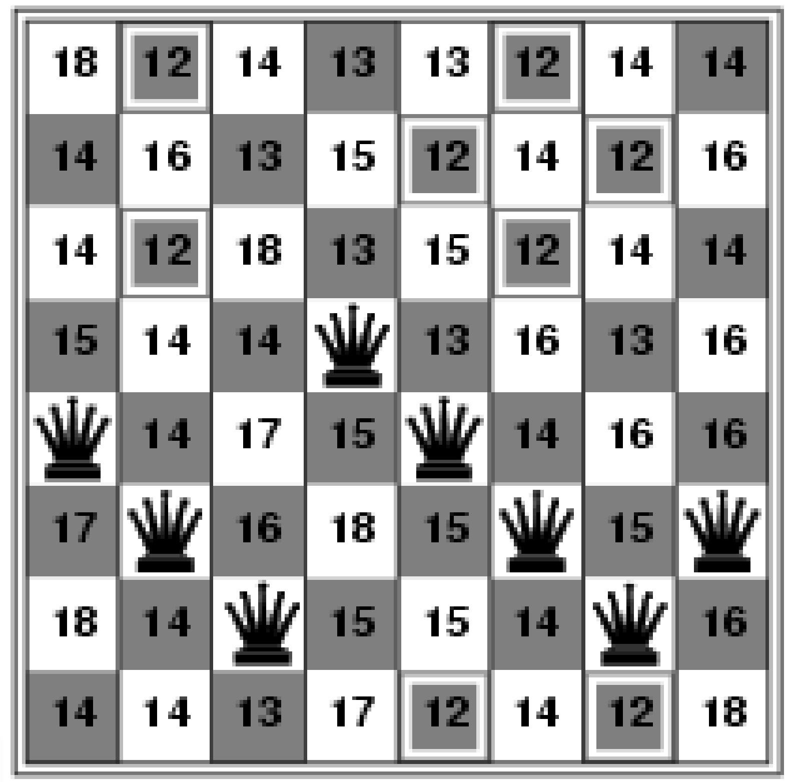
Hill-climbing search

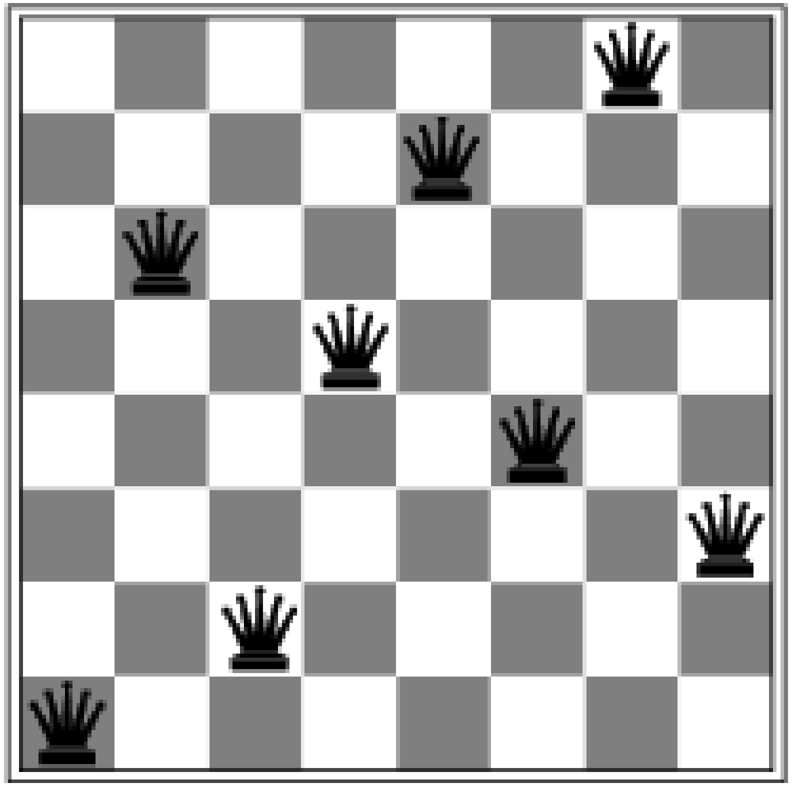
- Problem: depending on initial state, can get stuck in local maxima



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





- 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

Local beam search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

Final remarks

- Route planning algorithms in navigation systems use *landmarks* and recorded distances rather than calculations.

	Unidirectional		Bidirectional	
	Tree Size [nodes]	Comp. time [msec.]	Tree Size [nodes]	Comp. time [msec.]
No heuristic	62000	192	41850	122
Straight-line distance	9380	86	12193	84
Landmark heuristic	5260	16	$\infinity$ or b^d	16

- We have not discussed other search algorithms, for instance
  - Genetic algorithms
  - Adverserial (minimax) search