# Informed search algorithms

Chapter 4

# Material

- Sections 3.5, 4.1
- Excludes memory-bounded heuristic search (3.5)

# Outline

- Best-first search
- Greedy best-first search
- A$^*$ search

- Heuristics

- Local search algorithms

- Online search problems

# Review: Tree search

**function** TREE-SEARCH( *problem*, *fringe*) **returns** a solution, or failure
  *fringe* ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)
  **loop do**
    **if** *fringe* is empty **then return** failure
    *node* ← REMOVE-FRONT(*fringe*)
    **if** GOAL-TEST[*problem*](STATE[*node*]) **then return** SOLUTION(*node*)
    *fringe* ← INSERTALL(EXPAND(*node*, *problem*), *fringe*)

- A search strategy is defined by picking the order of node expansion

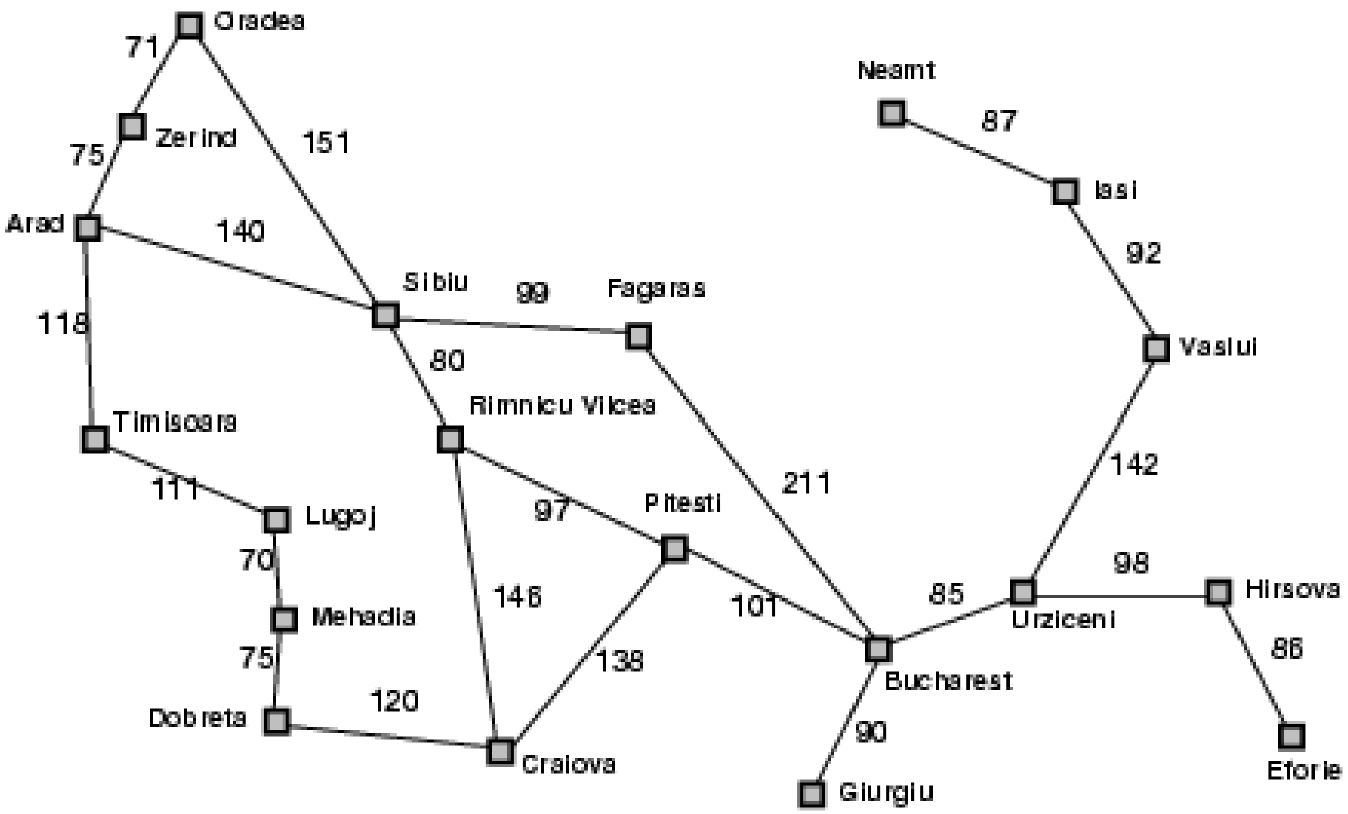
# Best-first search

- 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

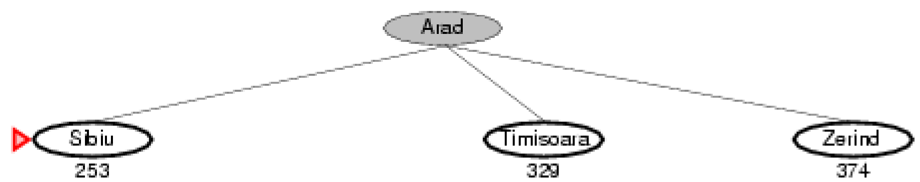
- Evaluation function $f(n) = h(n)$ (heuristic)
- = estimate of cost from $n$ to *goal*

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

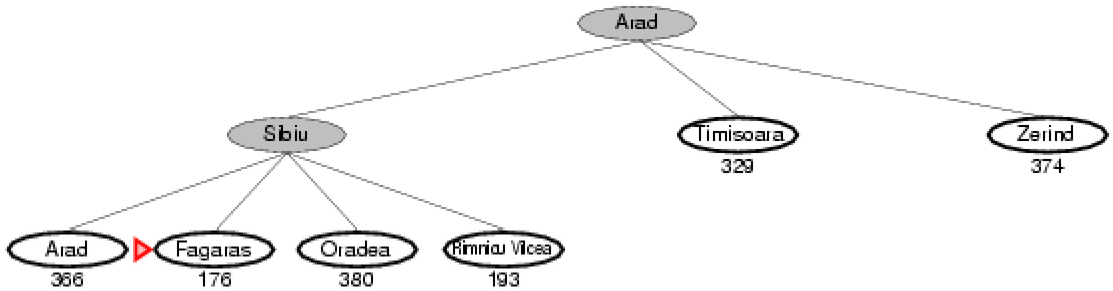
- Greedy best-first search expands the node that appears to be closest to goal

# Greedy best-first search example

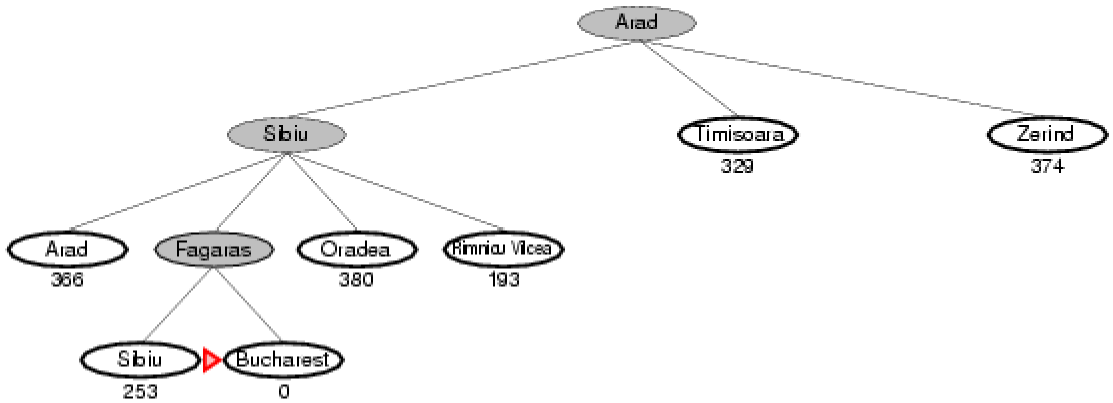
# Greedy best-first search example



# Greedy best-first search example



# Greedy best-first search example




Greedy Best-First demo?

# Properties of greedy best-first search

- Complete?

- Time?

- Space?
- Optimal?

# $A^*$ search

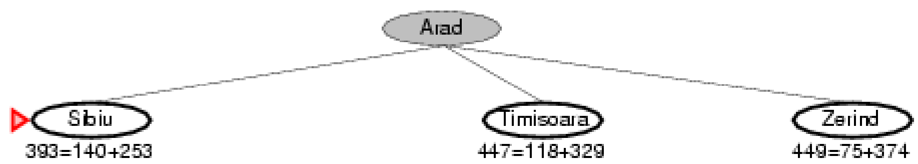
- Idea: avoid expanding paths that are already expensive

- Evaluation function $f(n) = g(n) + h(n)$

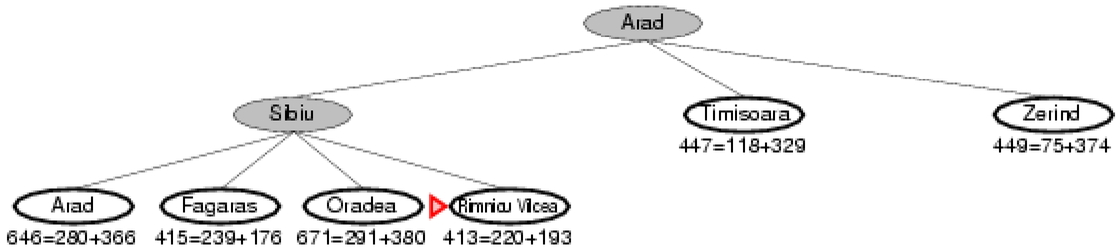
- $g(n)$ = cost so far to reach $n$
- $h(n)$ = estimated cost from $n$ to goal
- $f(n)$ = estimated total cost of path through $n$ to goal

# A* search example

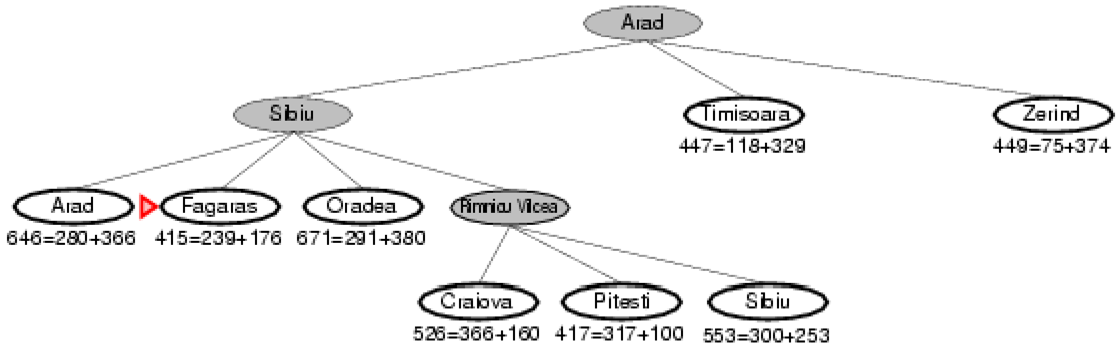
# A* search example



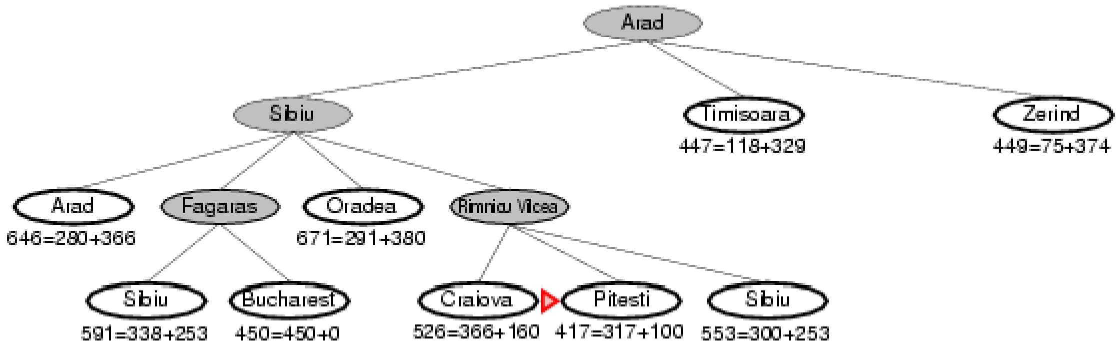
# A* search example



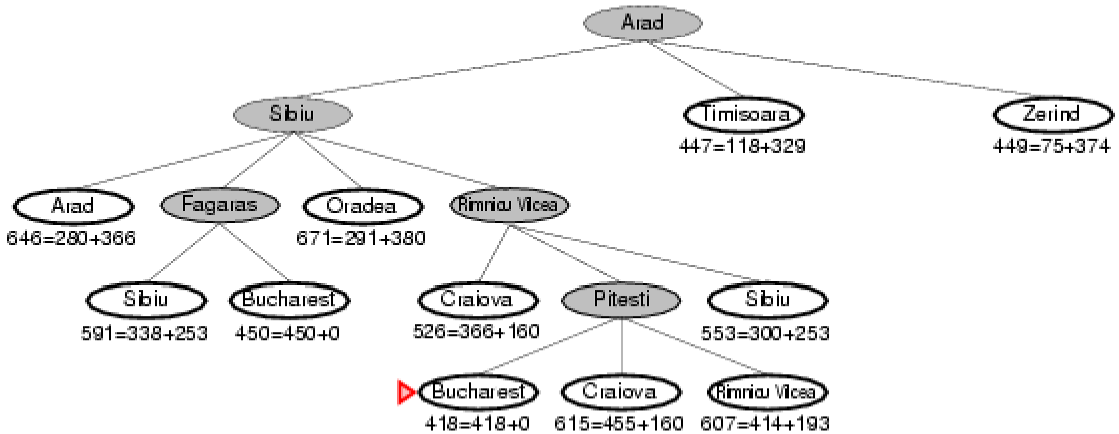
# A* search example



# A* search example



# A* search example



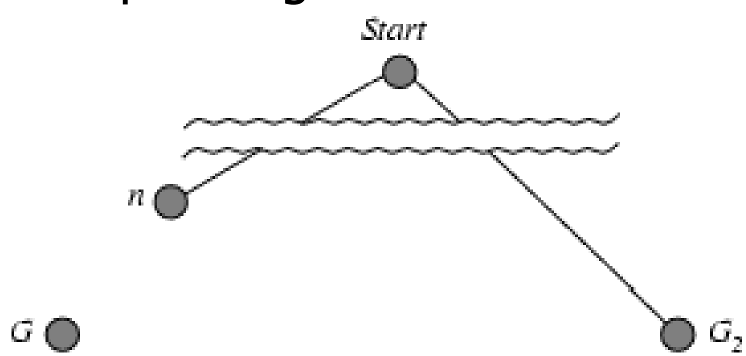

A* demo?

# 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* (proof)

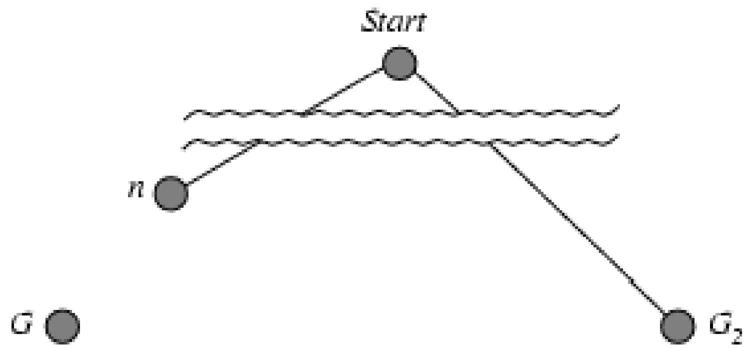
Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let $n$ be an unexpanded node in the fringe such that $n$ is on a shortest path to an optimal goal $G$.




- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since $G_2$ is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

# Optimality of A* (proof)

- Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.




- $f(G_2) > f(G)$ from previous

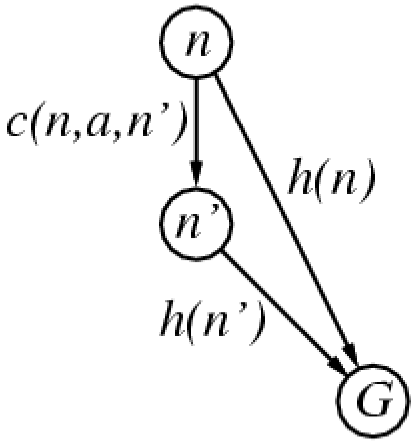
- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$

Hence $f(G_2) > f(n)$, and 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

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

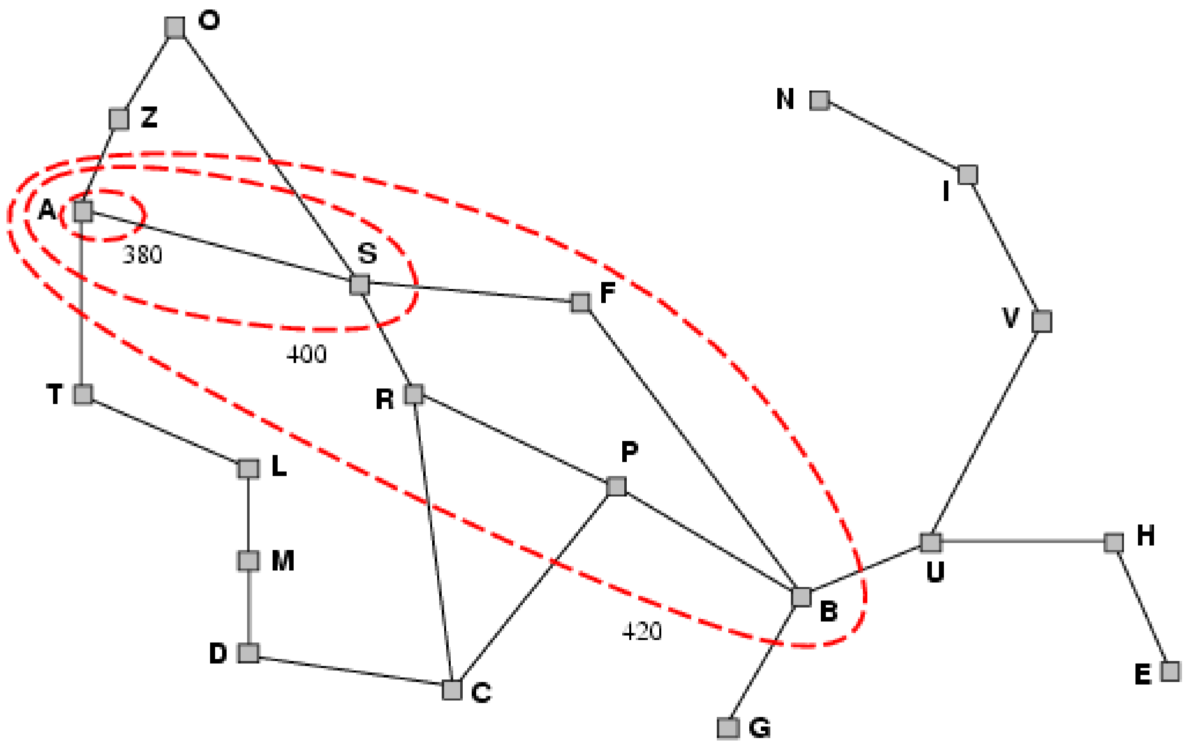
- i.e., *f(n)* is non-decreasing along any path.

- Theorem: If *h(n)* is consistent, A* using GRAPH-SEARCH is optimal

# Optimality of $A^*$

- $A^*$ 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?

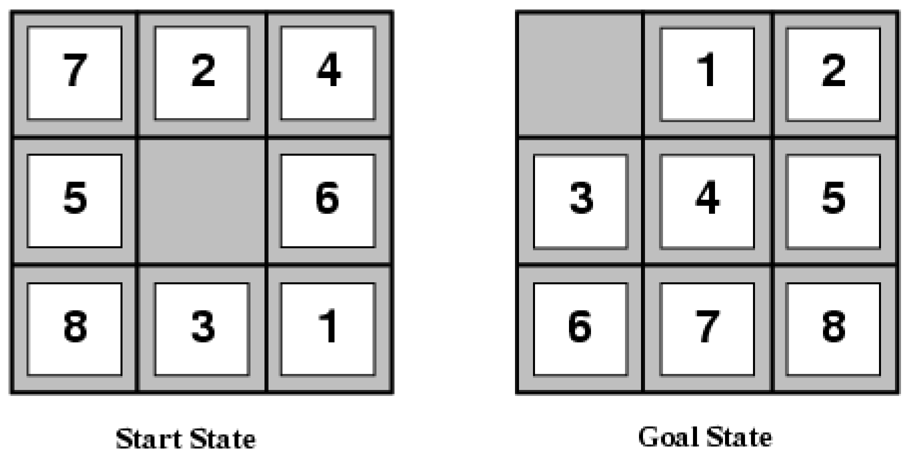
- Time?
- Space?
- Optimal?

# Admissible heuristics

E.g., for the 8-puzzle:

Average solution depth?

Average branching factor?



# 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

- $h_1(S) = ?$ 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

# 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

# Beyond Classical Search

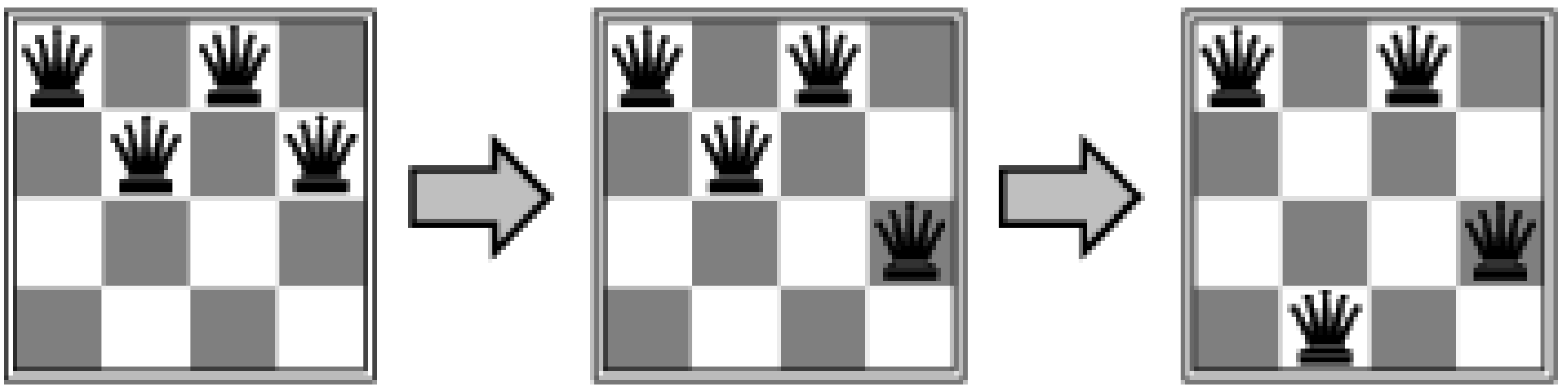
- Local Search

- Searching with non-determinism
- Searching with partial observations
- Online and exploratory search

# 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
- 

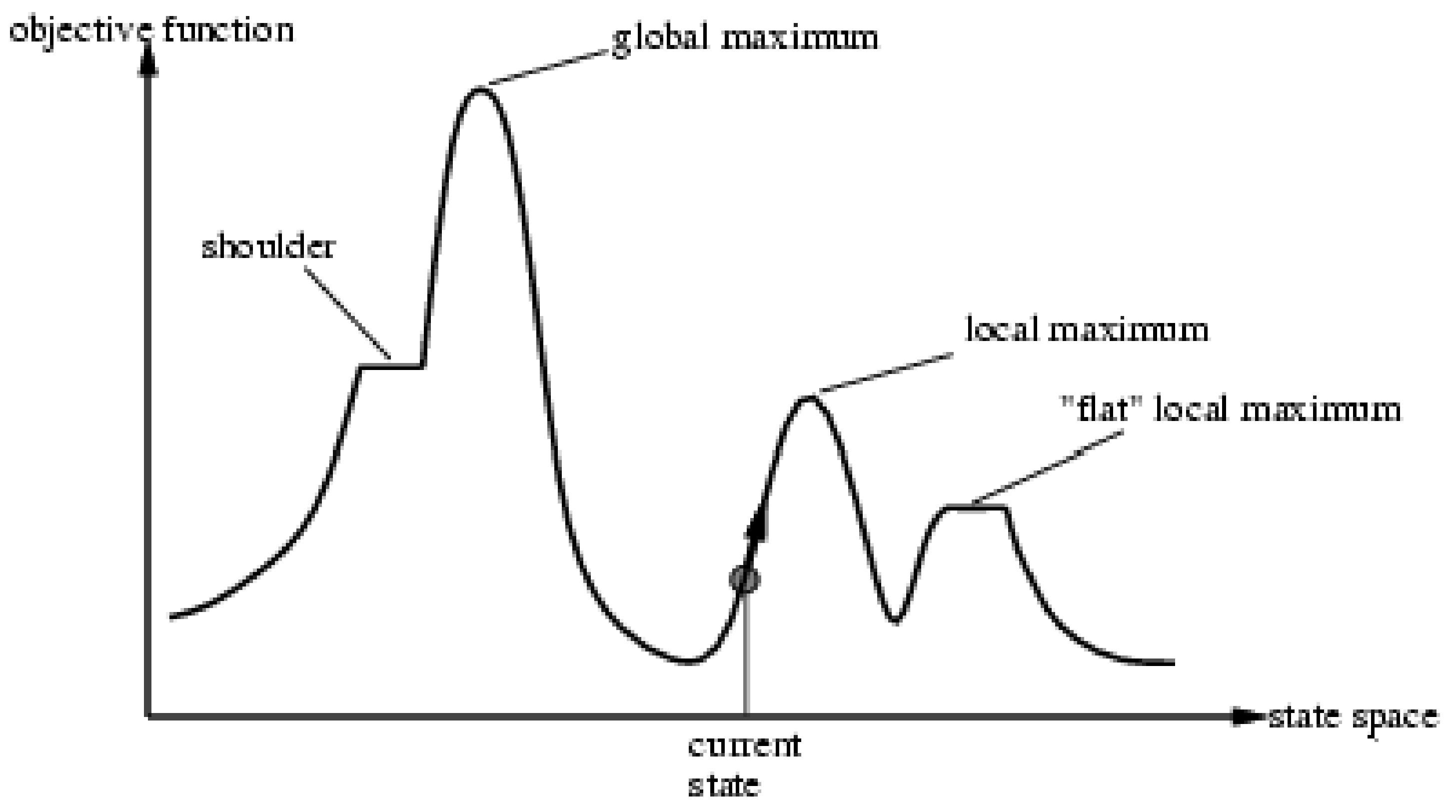
# Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

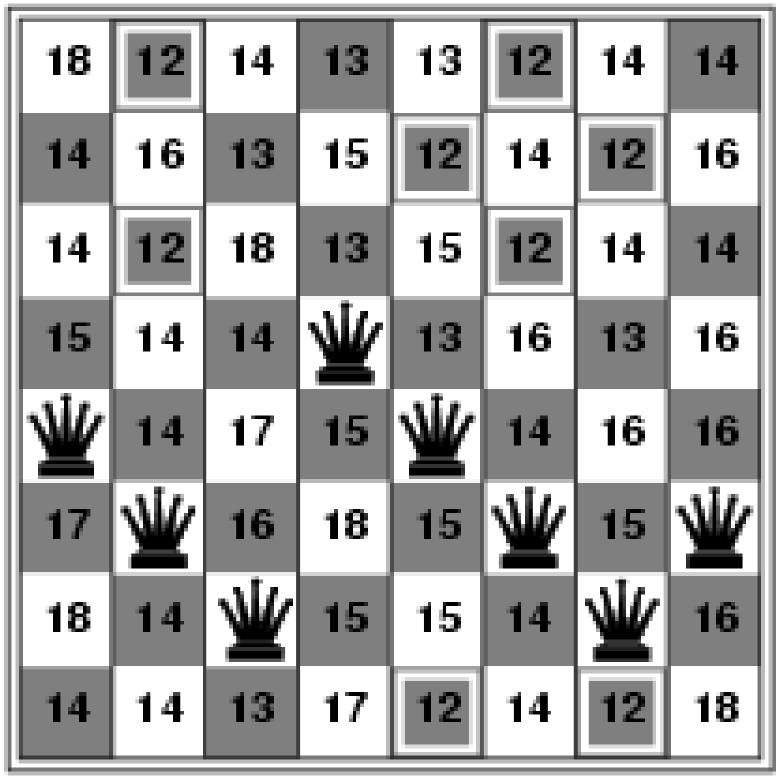
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   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

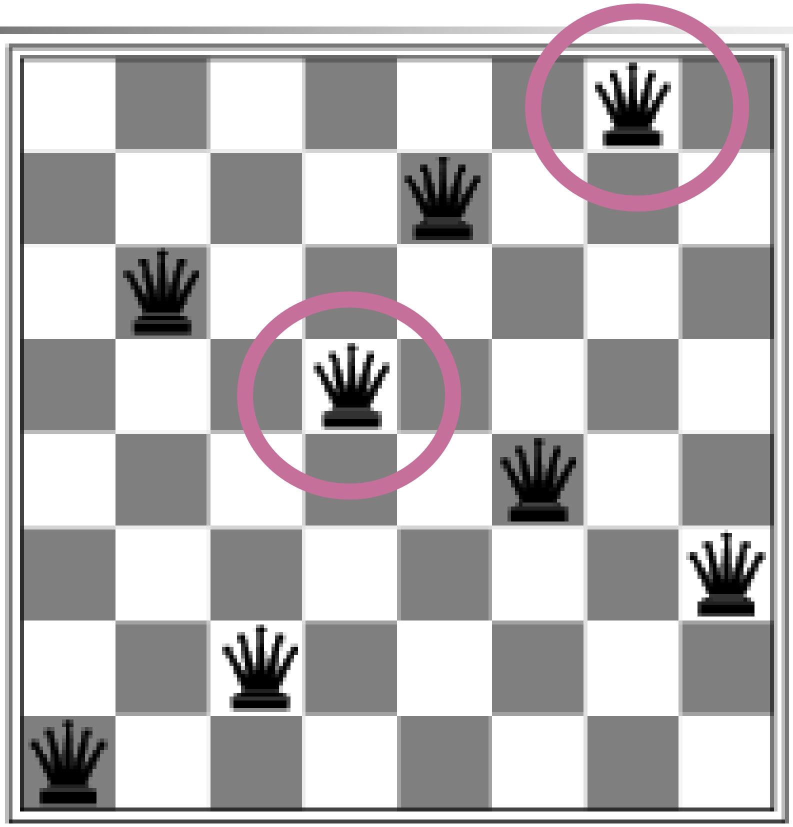
- Problem: depending on initial state, can get stuck in local maxima
- 

# 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

# Hill-climbing search: 8-queens problem



Hill climbing demo?

- 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)
```

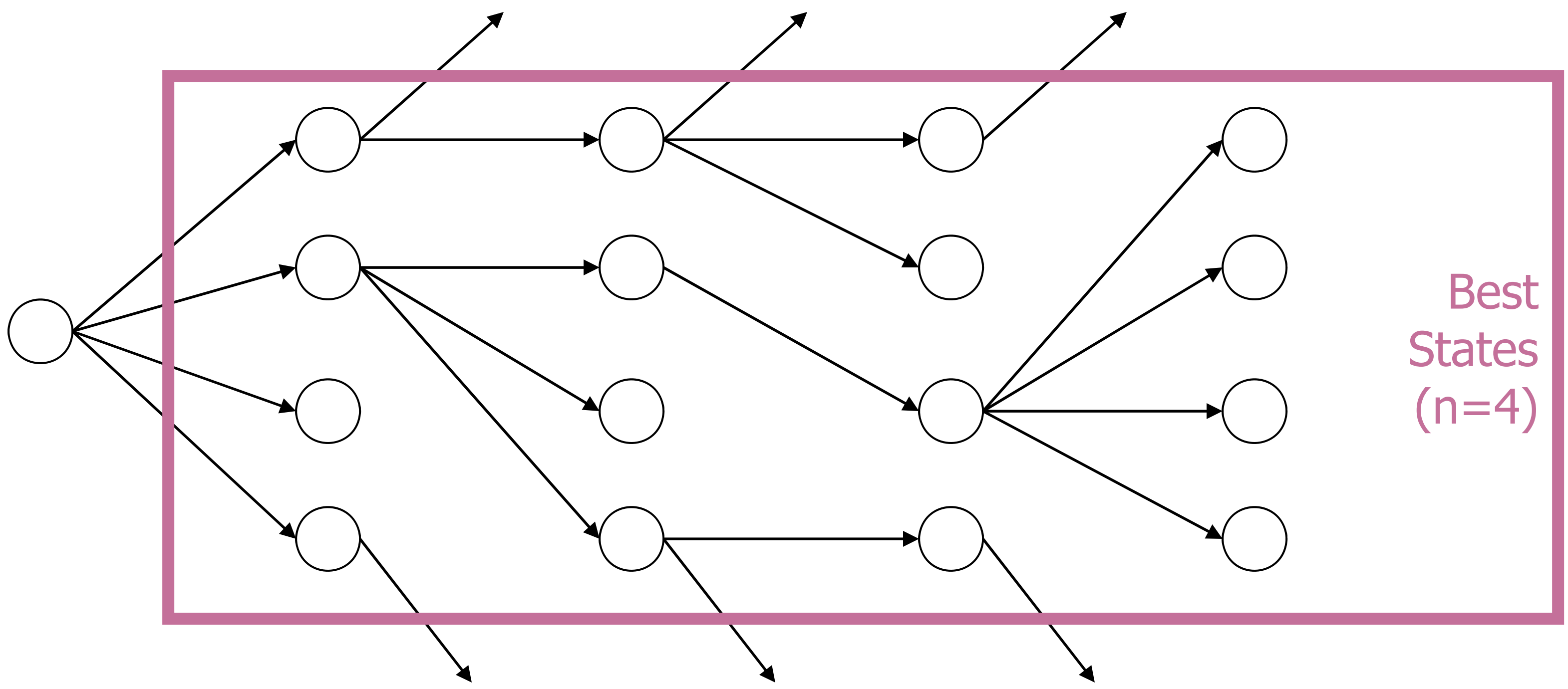
# Properties of simulated annealing search

- 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

- Why keep just one best state?

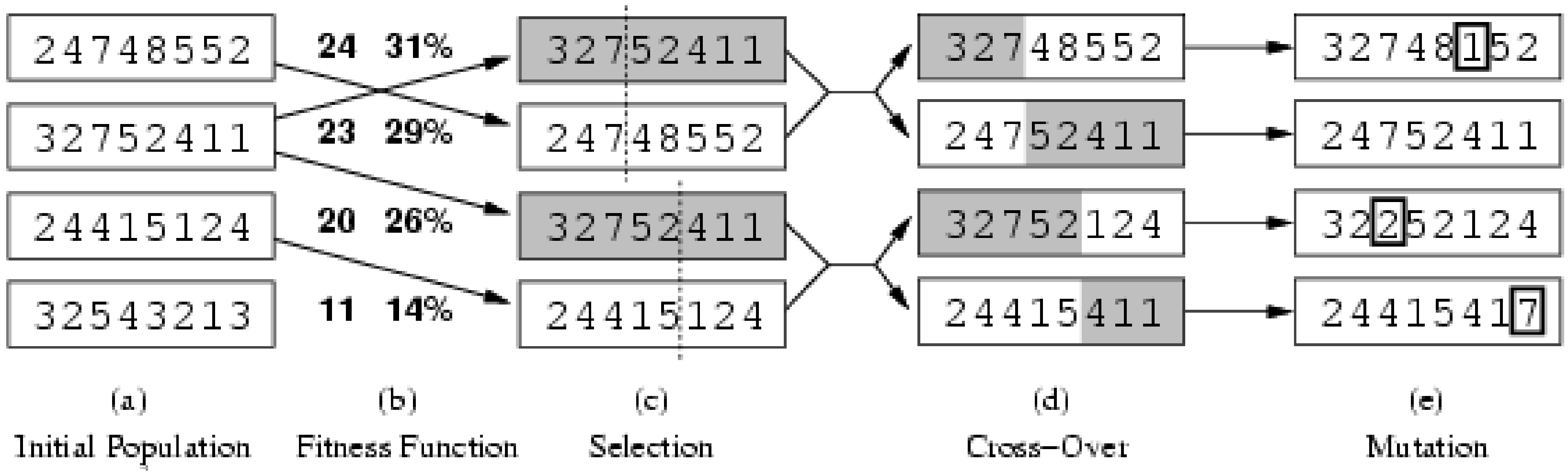



- Can be used with randomization too

# Genetic algorithms

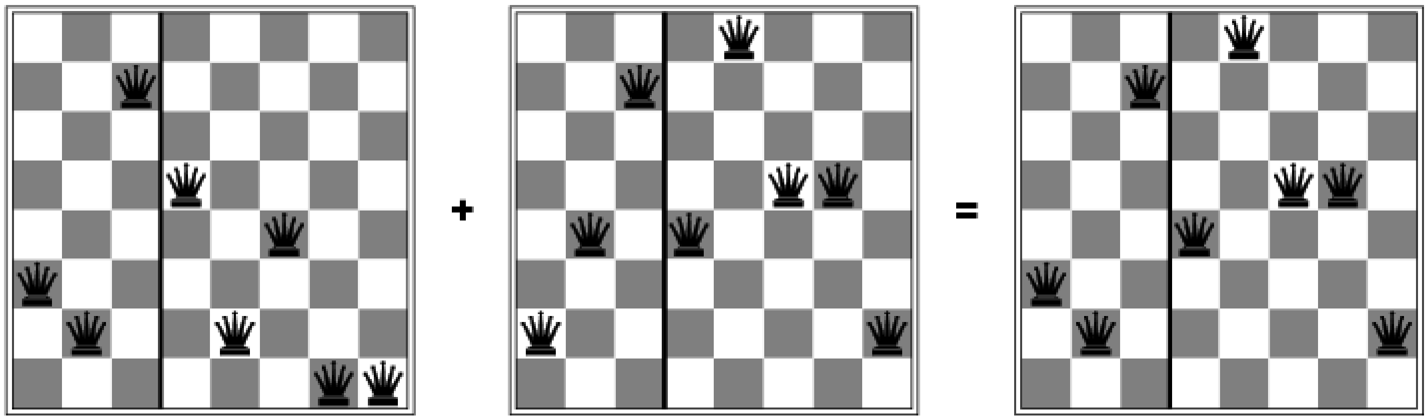
- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states
- Produce the next generation of states by selection, crossover, and mutation

# Genetic algorithms




- Fitness function: number of non-attacking pairs of queens (min = 0, max = $8 \times 7/2 = 28$)
- $24/(24+23+20+11) = 31\%$
- $23/(24+23+20+11) = 29\%$ etc.

# Genetic algorithms

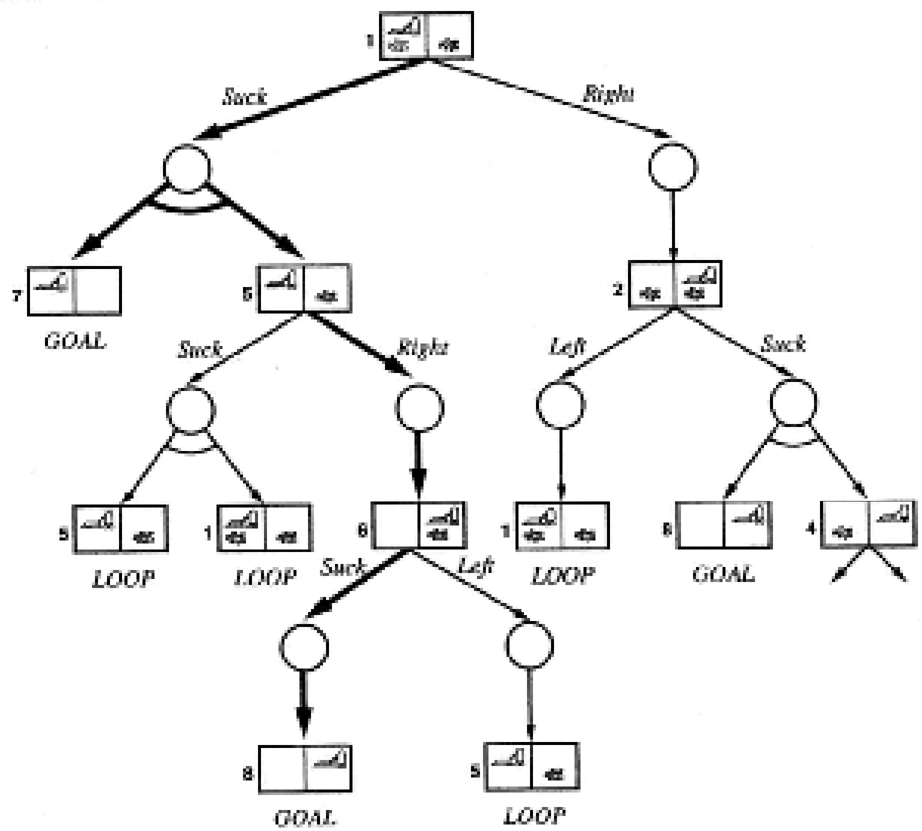


# Search w/ non-determinism

- Fully observable, deterministic environments
  - Sensors, precepts no use

- Consider erratic actuators
  - Action leads to a **set** of possible states
  - Plan will not be a set sequence, may have loops contingencies (if-then-else)

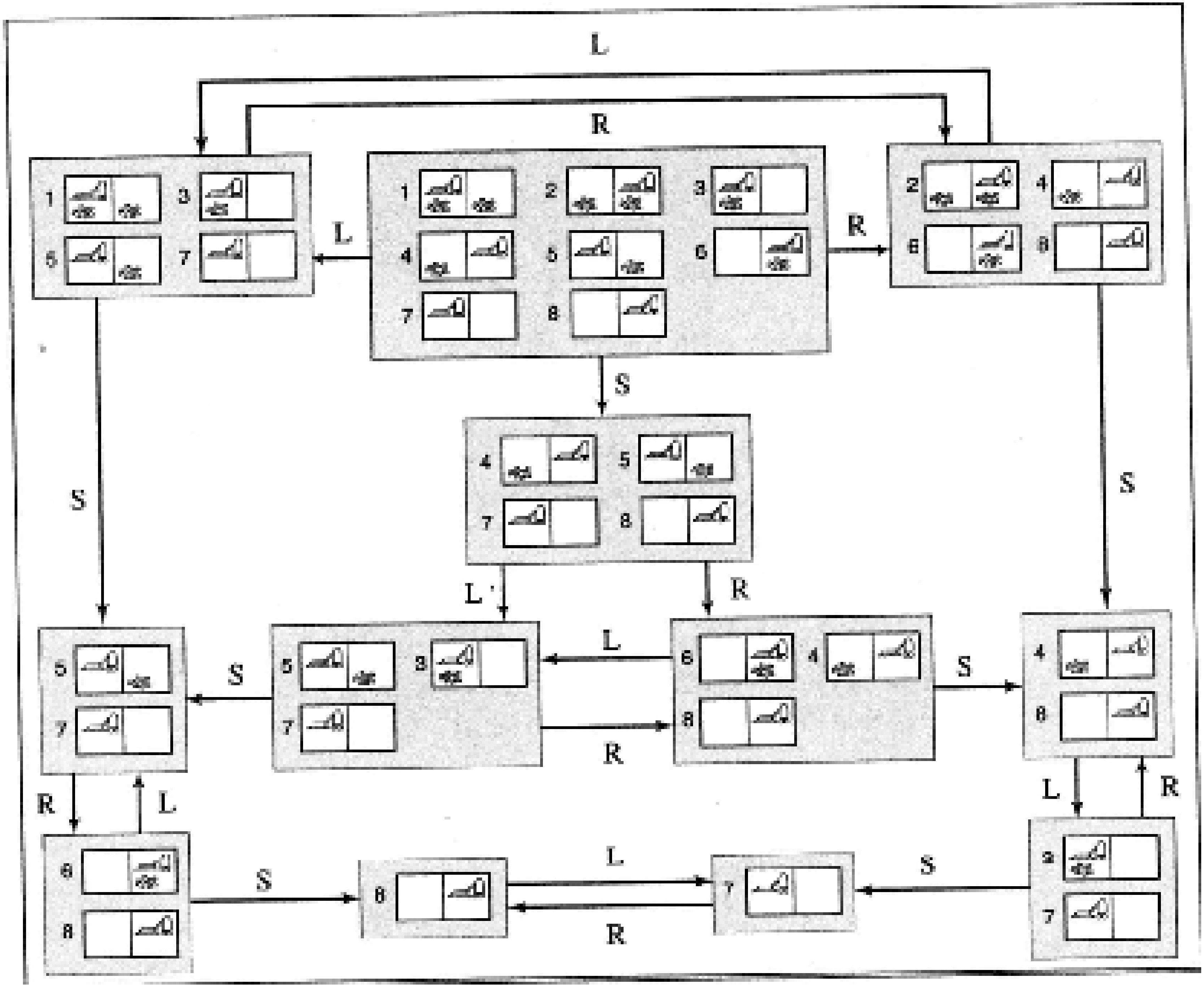
# And-Or Search Tree




- **Q**: what does the "LOOP" label mean here?

# Search w/ partial observations

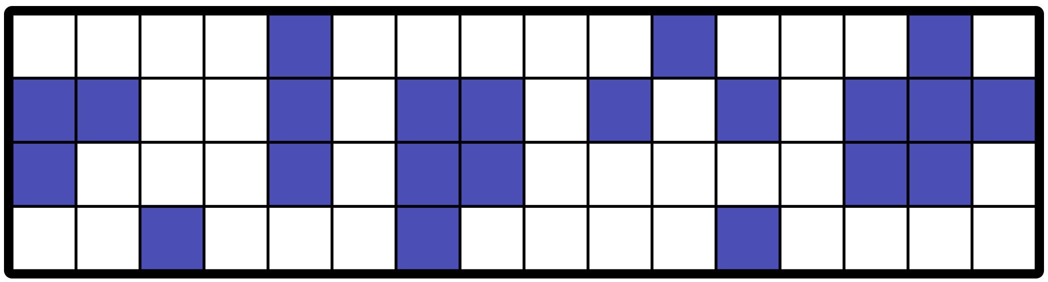
- Conformant problem – no observations
  - Useful! Solutions are independent of initial state
  - **Coerce** the state space into a subset of possible



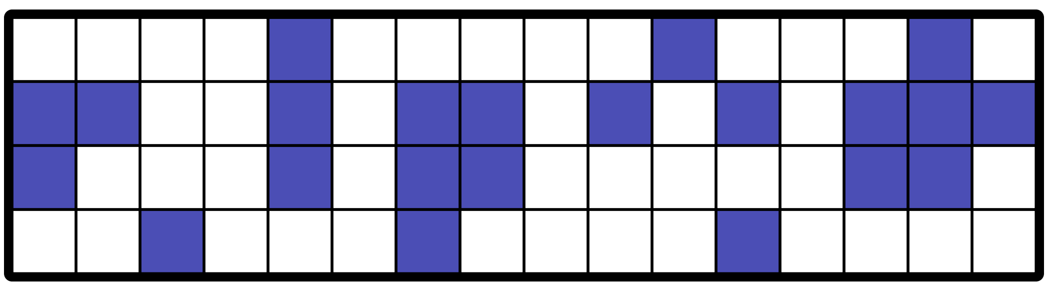
L
R
S
L
R
S
L
R
S
L
R
S
L
R
S
L
R
L
R
S
L
R
S

# Localization

Initial State:



After observing NSW:



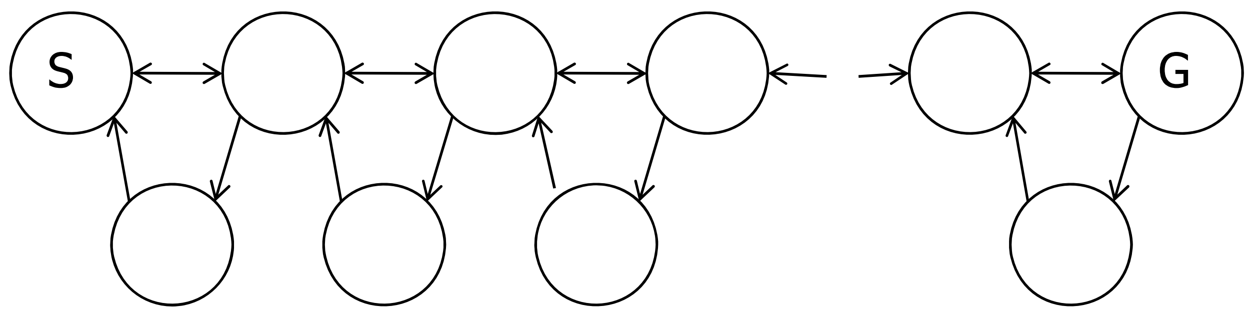
**Q**: What about a really big set of initial positions?

# Online search and exploration

- Many problems are offline
  - Do search for action and then perform action
- Online search interleave search & execution
  - Necessary for exploration problems
  - New observations only possible after acting

# Exploratory Search

- In an unknown state space, how to pick an action?
  - Any random action will do … but




  - Favor those that allow more exploration of the search space
    - ➔ Graph-search to track of states previously seen

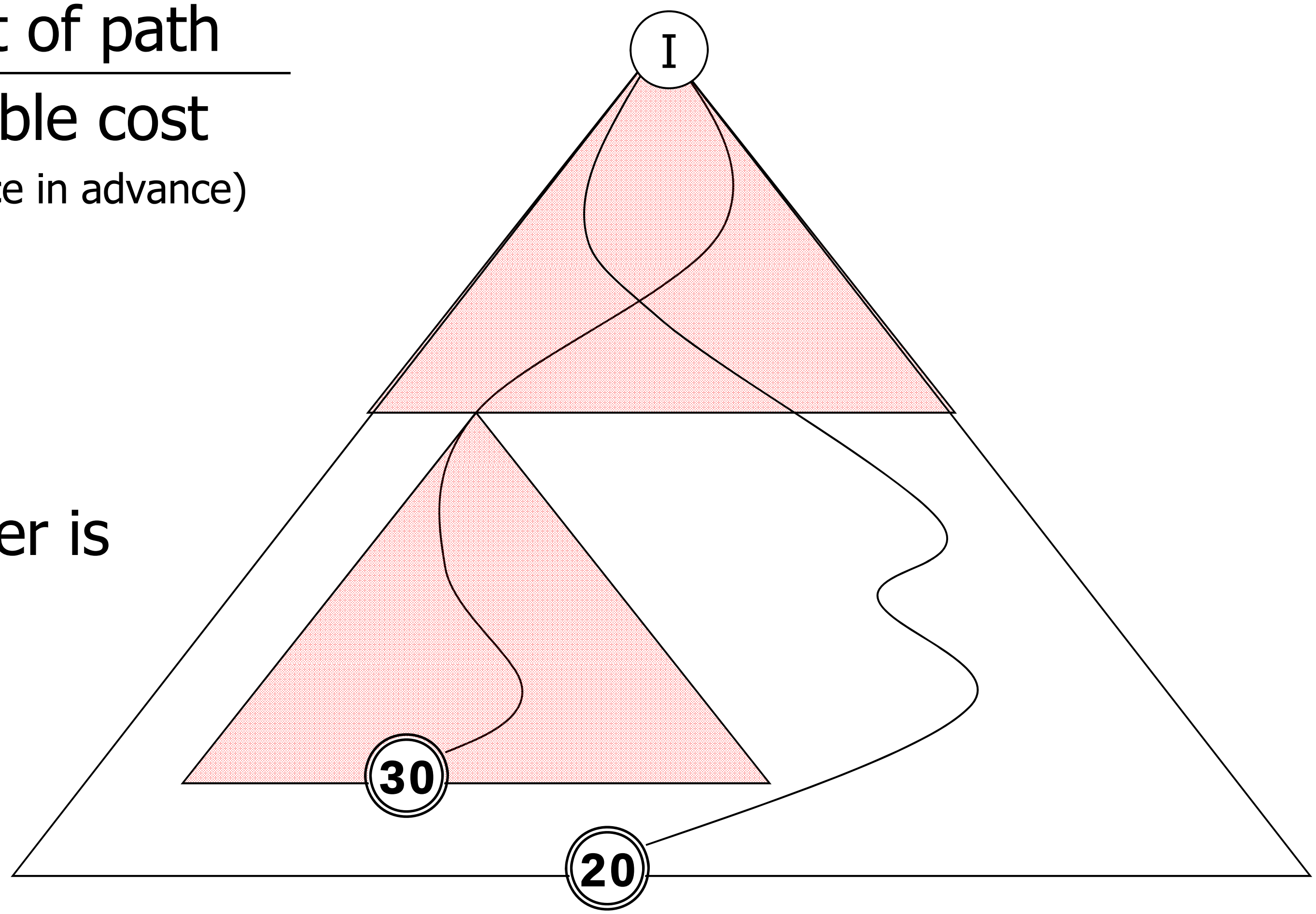
# Assessing Online Agents: Competitive Ratio

$$\frac{\text{Actual cost of path}}{\text{Best possible cost}}$$

(if agent knew space in advance)

30/20 = 1.5

For cost, lower is better



# Exploration problems

- Exploration problems: agent physically in some part of the state space.
  - e.g. solving a maze using an agent with local wall sensors
  - Sensible to expand states easily accessible to agent (i.e. local states)
    - Local search algorithms apply (e.g., hill-climbing)

# Assignment

- Build a game player
- Restricted by time per move (5 real-time secs)

- Interact with the game driver through the command line
  - Each turn, we will run your program, providing the board state as input.
  - Your output will be the pair of coordinates indicating the piece to move and its destination.

# Homework #1 - Ataxx

- Note: we haven't yet covered all of the methods to solve this problem

- This week: start thinking about it, discuss among yourselves (remember the Facebook Rule!)
  - Play the game, review past games by others.
  - What heuristics are good to use?
  - What type of search makes sense to use?