CS 161 Intro. To Artificial Intelligence

Week 3, Discussion 1B

Qian Long

Today’s Topics

- Informed Search
  - Greedy Search
  - A* Search
    - Optimality
    - Heuristic Function
- Constraint Satisfaction Problem (CSP)
  - Formulation of CSPs
  - Backtrack Search
  - Techniques for improving CSP solution

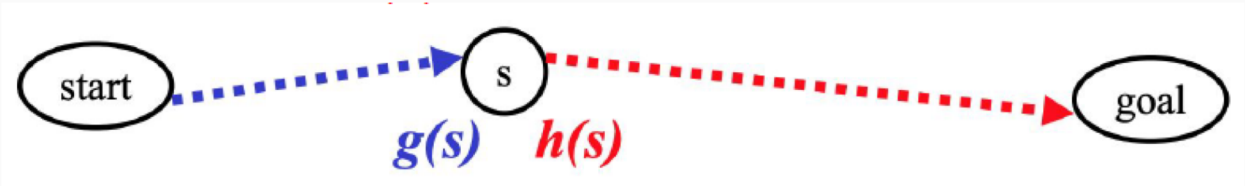
Informed Search

- Leverage problem-specific knowledge
- Also known as heuristic search

Informed Search

- Uninformed Search
  - Have access only to the problem definition
  - Knows the **actual path cost g(s)** from start to a node s in the fringe

- Informed Search
  - Have access to **heuristic function** as well as problem definition
  - The additional **h(s)** is the heuristic of **the cost from s to goal**



Informed Search

Q: Is uniform-cost search informed search?

A: **No!**

Reasons:

- It only looks backwards, it has no ability to predict future costs
- It doesn't use any domain knowledge

Best-first search

Best-first search:

- The general approach for informed search
- Choose the "best" (the most desirable) node to expand

How to determine which node is the best?

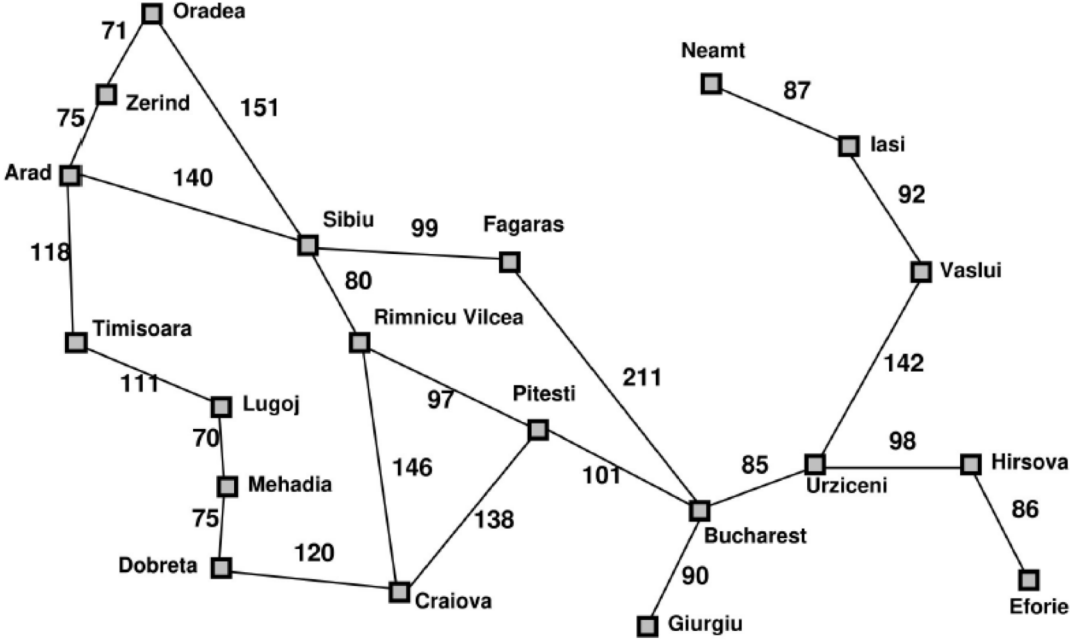
- **Evaluation function $f(n)$** (a cost estimation for node n) -- smallest $f(n)$ is usually the “best” node
- n is a node in search tree, not a state in state graph
- Often involves a **heuristic function $h(n)$**
- Fringe is a queue sorted in decreasing order of desirability

Special Cases:

- Greedy best first search
- A* search

Greedy Best-first Search

- $f(n) = h(n)$ **(heuristic)** = estimate cost from node n to the closest goal
  - E.g., h_{SLD} = straight-line distance from n to Bucharest



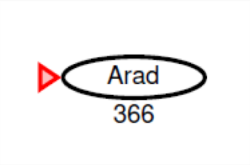
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-first Search

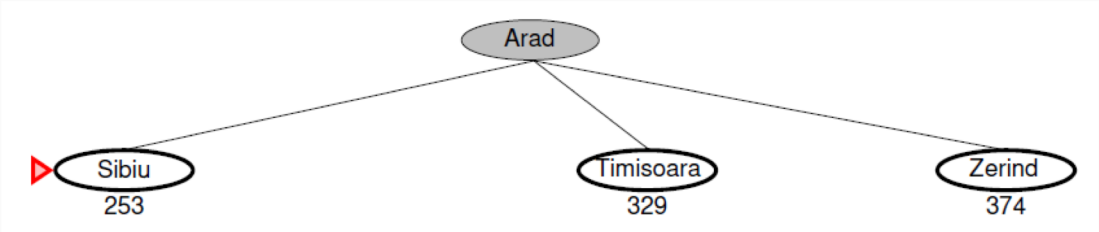
Greedy best-first search expands the node that **appears** to be closest to goal

$f(n) = h(n)$ = straight-line distance from n to Bucharest

Initial state:



After expanding Arad:



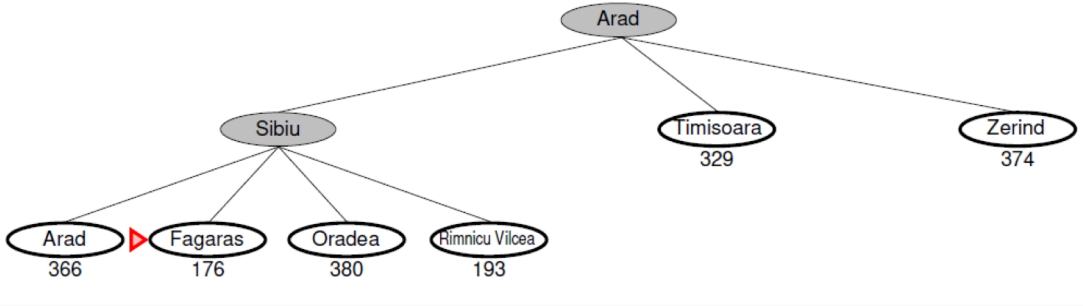
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-first Search

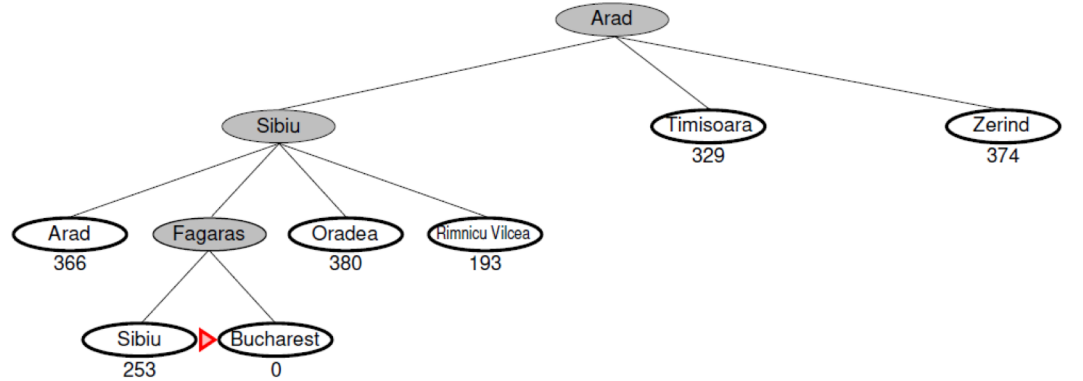
Greedy best-first search expands the node that **appears** to be closest to goal

$f(n) = h(n)$ = straight-line distance from n to Bucharest

After expanding Sibiu



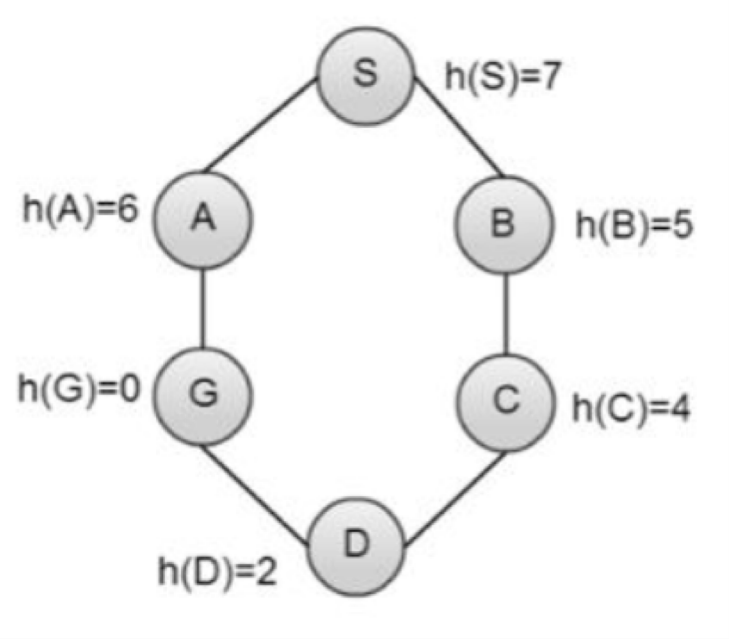
After expanding Fagaras:



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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

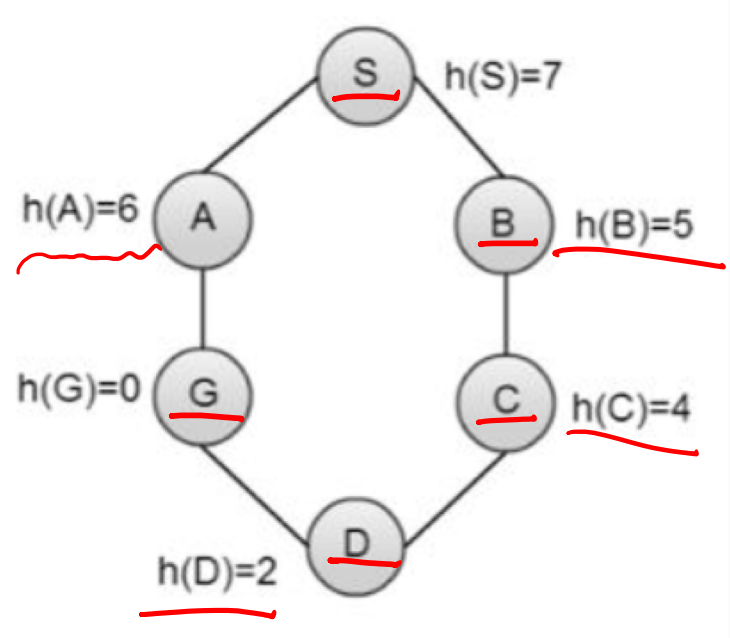
Greedy Best-first Search - Example

From S to G using Greedy best-first search:



Greedy Best-first Search - Example

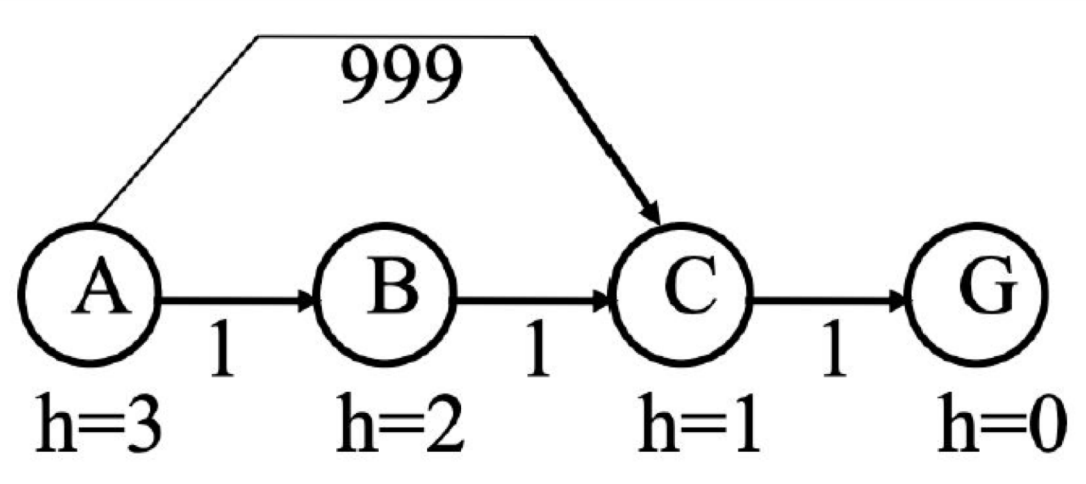
From S to G using Greedy best-first search:



S → B → C → D → G

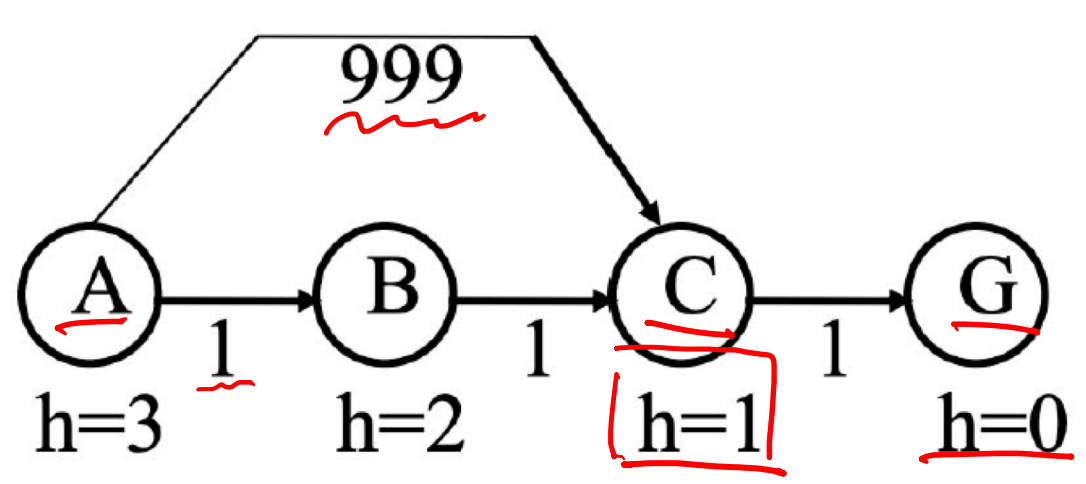
Greedy Best-first Search - Example

From A to G using Greedy best-first search:



Greedy Best-first Search - Example

From A to G using Greedy best-first search:



A → C → G

Problem:
Actual cost is too high!
→ Use A* search

A* Search

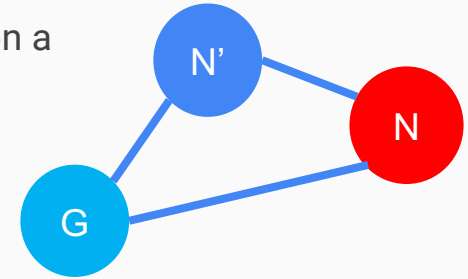
- Avoid expanding paths that are already expensive
- Goal test at expansion
- $f(n) = g(n) + h(n)$
  - $g(n)$: actual path cost so far to reach n
  - $h(n)$: estimated cost from n to goal
  - $f(n)$: estimated total cost of path through n to goal

A* Search

Theorem: A* search is optimal

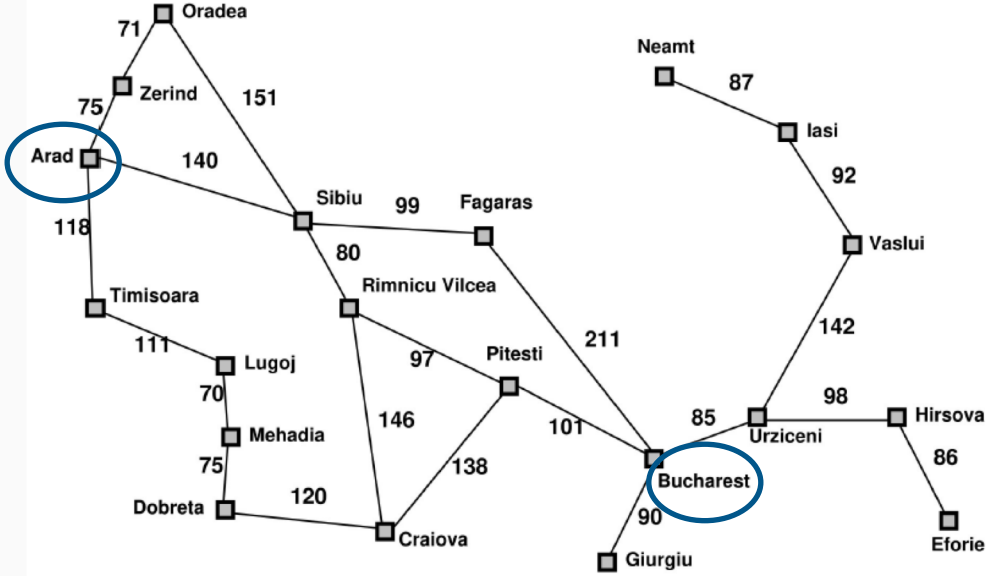
The properties of **heuristic h(n)** for A*: (they are the conditions for A*'s optimality)

- **Admissibility**
  - $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost from node n to goal state (also need h(n)>=0)
  - Admissible heuristic cost should **never overestimate** the actual cost of a node
- **Consistency (Monotonicity)**
  - $h(n) \leq c(n, a, n') + h(n')$
    - n' is a successor of n
    - $c(n, a, n')$ is the cost of path from n to n' by choosing action a
    - $f(n)$ is non-decreasing along any path
    - Alike to "triangle inequality"



A* Search

- Example: from Arad to Bucharest
- $f(n) = g(n) + h(n)$
  - $h(n) = h_{SLD}$ = straight-line distance from n to Bucharest

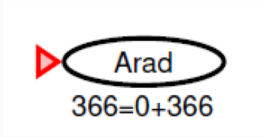


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

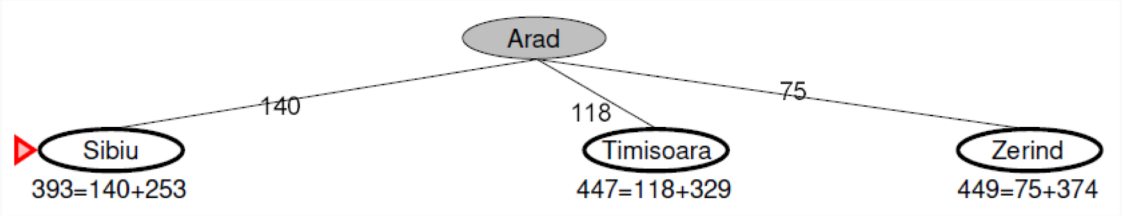
A* Search

- $f(n) = g(n) + h(n)$
  - $h(n) = h_{SLD}$ = straight-line distance from n to Bucharest

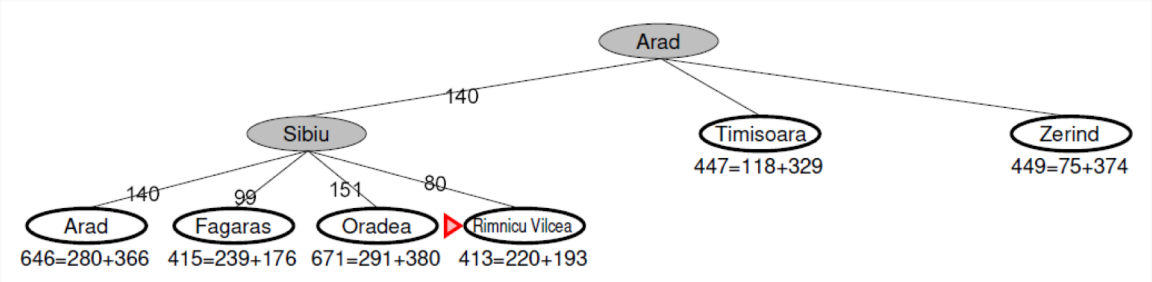
Initial state:



After expanding Arad:



After expanding Sibiu:

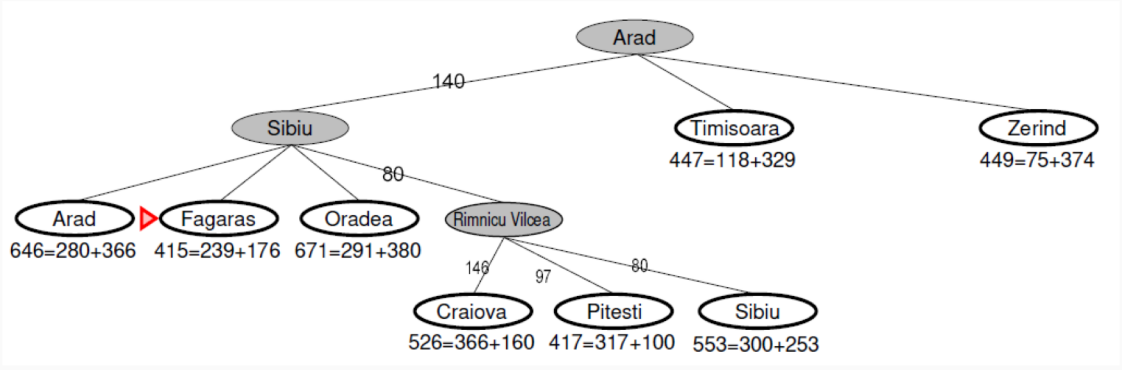


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

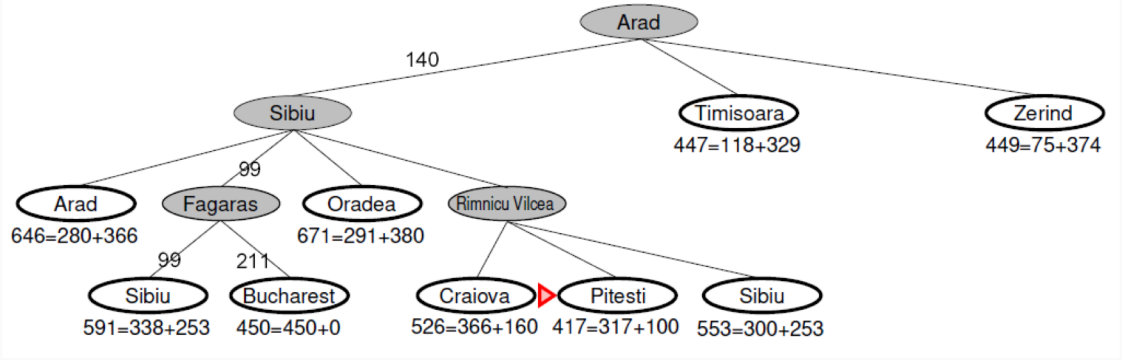
A* Search

- $f(n) = g(n) + h(n)$
  - $h(n) = h_{SLD}$ = straight-line distance from n to Bucharest

After expanding Rimnicu Vilcea:



After expanding Fagars:

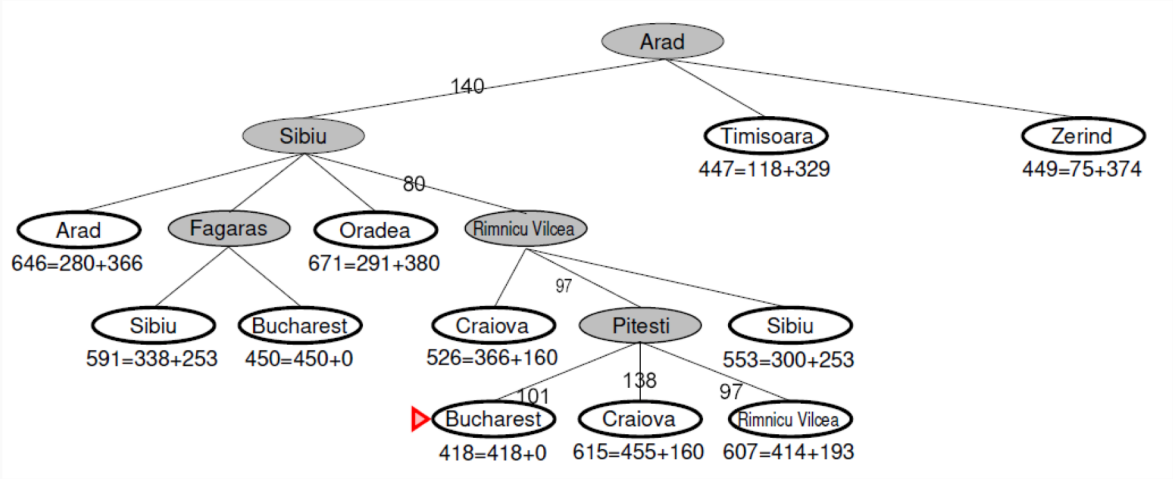


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search

- $f(n) = g(n) + h(n)$
  - $h(n) = h_{SLD}$ = straight-line distance from n to Bucharest

After expanding Pitesti:



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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search - Optimality

- Admissibility
  - $h(n) \leq h^*(n)$
    - $h^*(n)$: true cost from node n to goal state
  - Theorem: If *h(n)* is admissible, A* is optimal
- Consistency (Monotonicity)
  - $h(n) \leq cost(n, n') + h(n')$ (n' is successor of n)
    - $f(n)$ is non-decreasing along any path
- Consistency implies admissibility

A* Search - Optimality

- Consistency is stronger than admissible (**consistency implies admissibility**)
- If h(n) is consistent, the values of f(n) along any path are nondecreasing
  - It's because: f(n') = g(n') + **h(n')** = g(n) + **c(n,a,n') + h(n')** >= g(n) + **h(n)** = f(n)
  - Where c(n, a, n') is the cost of path from n to n' (a successor of n) by choosing action a

Heuristic Function

How to figure out good heuristics?

- A good heuristic should be admissible but as close to the true cost as possible
  - **Admissible** heuristic cost should **never overestimate** the actual cost of a node
  - I.e. it must be “optimistic” → we never overlook a node that is actually good

- Admissible heuristics can be derived from the exact solution cost of a **relaxed version** of the problem
  - Relaxed problems are problems with fewer restrictions on the actions (but with same initial state and goals)

CSP - Formulation

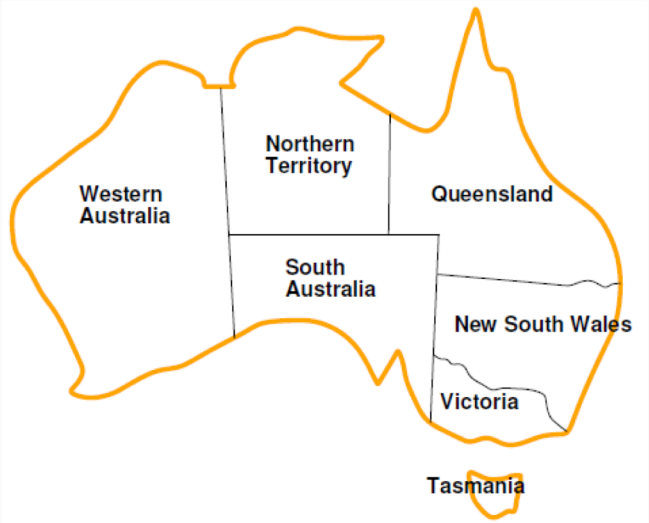
Components of Constraint Satisfaction Problem (CSP):

X is a set of variables, $\{X_1, \ldots, X_n\}$.
D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.
C is a set of constraints that specify allowable combinations of values.

- Each domain Di consists of a set of allowable values {v1,…,vk} for the corresponding Xi
- A **state** in CSP: an assignment of values to some or all variables
  - Partial assignment: assign values to only some of the variables
  - Complete assignment: every variable is assigned (otherwise partial assignment)
  - Consistent/Legal assignment: an assignment that does not violate any constraints
- A **solution** in CSP: a consistent, complete assignment

CSP – Formulation Example

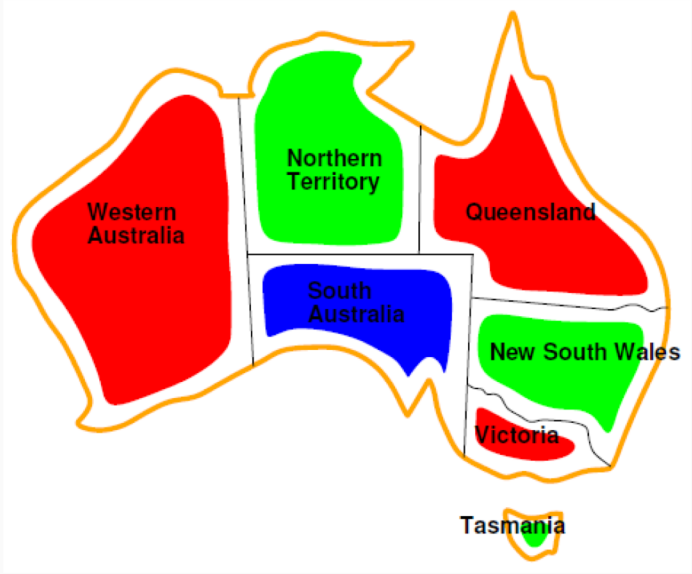
Map Coloring:



- **Variables**: WA, NT, Q, NSW, V , SA, T
- **Domains**: Di = {red, green, blue}
- **Constraints**: adjacent regions must have different colors
  - E.g. WA ≠ NT (if the language allows this),
  - E.g. (WA,NT) ∈ {(red, green), (red, blue), (green, red), (green, blue), . . .}

CSP – Formulation Example

Map Coloring:



- Solutions are assignments satisfying all constraints,
  - e.g., {WA=red, NT =green, Q=red, NSW =green, V =red, SA=blue, T =green}

Backtrack Search

Backtracking search: the basic uninformed algorithm for CSPs

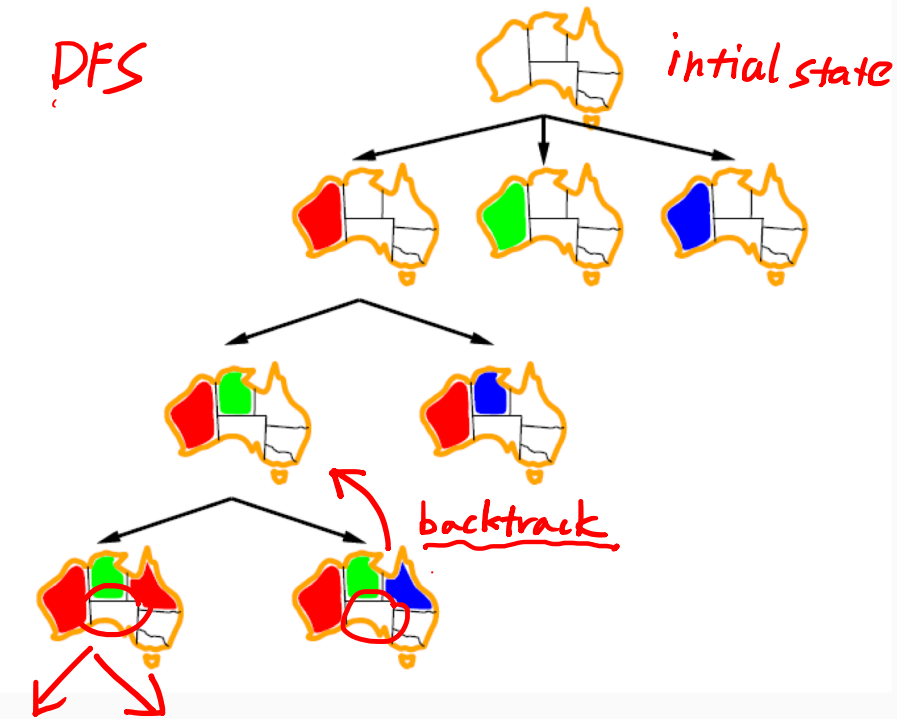
- It's basically a depth-first search for CSPs with single-variable assignments:
  - Chooses values for one variable at a time and **backtracks when a variable has no legal values left to assign**

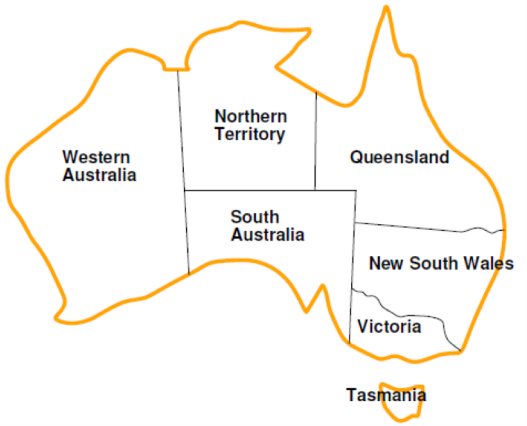
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Backtrack Search - Example

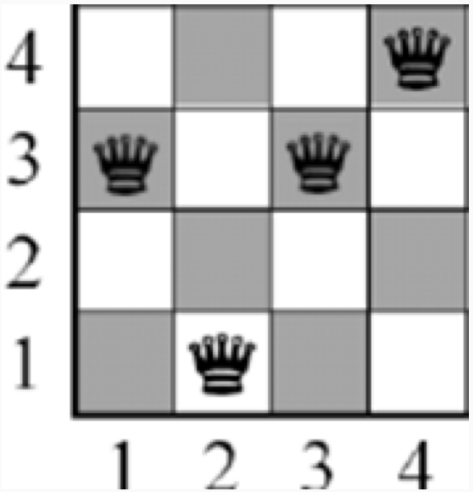
- When to backtrack? → when a variable has no legal values left to assign





Backtrack Search - Example

- 4-Queens Puzzle (assume each queen in each column) as a CSP:
  - Variables: Q1, Q2, Q3, Q4 -- row indices of each queen
  - Domains Di = {1, 2, 3, 4}
  - Constraints:
    - Qi ≠ Qj (cannot be in same row)
    - |Qi − Qj | ≠ |i − j| (or same diagonal)
- Backtracking search:
  - (Q1, Q2, Q3, Q4):
  - (1,X,X,X) → (1, 3,X,X) → No legal assign for Q3, backtracking
  - (1, 4,X,X) → (1, 4, 2,X) → No legal assign for Q4, backtracking
  - (1, 4, 3, X) → Not a legal assign for Q3, backtracking
  - (2,X,X,X) → (2, 4,X,X) → (2, 4, 1,X) → (2, 4, 1, 3), Bingo!



Improving Backtracking Efficiency

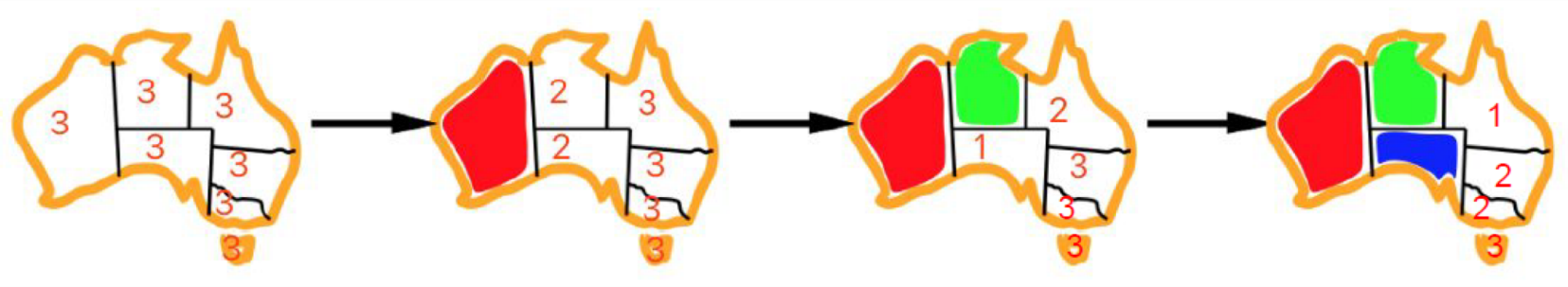
3 techniques (heuristics) to improve efficiency:

- How to select unassigned **variable**?
  - Most constrained variable / Minimum remaining values (MRV)
  - Most constraining variable / Degree heuristic

- In what order should we assign **values** to each variable?
  - Least constraining value

Improving Backtracking Efficiency

- How to select unassigned **variable**?
  - Most constrained variable / Minimum remaining values (MRV):
    - choose the **variable** with the fewest legal values
    - If no legal values left, fail immediately

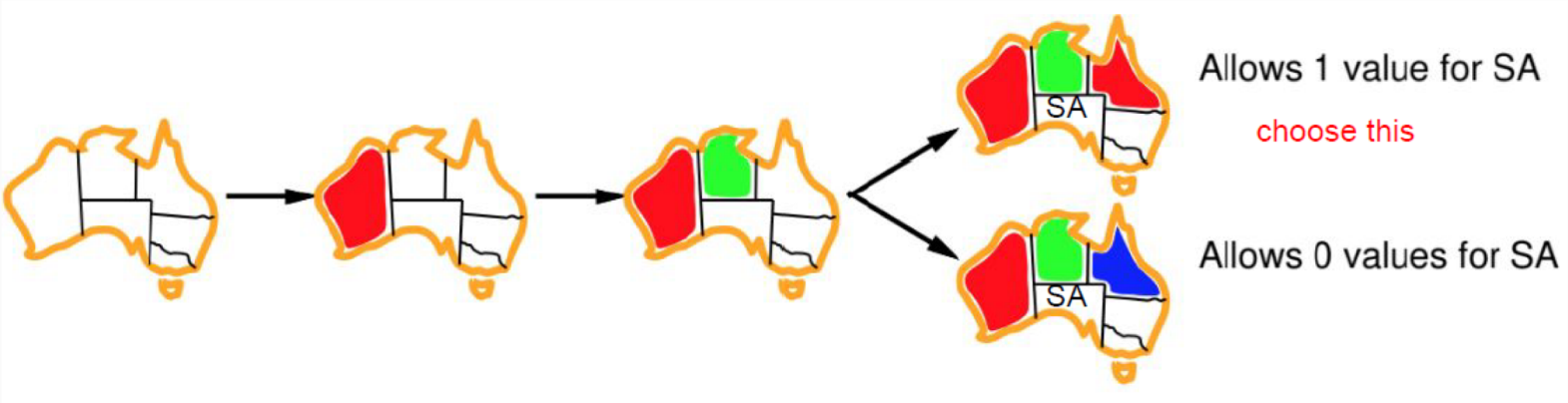


Improving Backtracking Efficiency

- How to select unassigned **variable**?
  - Most constrained variable / Minimum remaining values (MRV)
  - Most constraining variable / Degree heuristic:
    - Choose the **variable** with the most constraints on remaining variables
    - Attempt to reduce branching factor on future choice
    - **Useful as a tie-breaker**

Improving Backtracking Efficiency

- In what order should we assign **values** to each variable?
  - Least constraining value:
    - Choose the **value** that rules out the fewest values in the remaining variables
    - Leave the maximum flexibility for subsequent variable assignments



Questions?

- My slides take the following materials as references:
  - Xiao Zeng’s slide

Thank you!