Artificial Intelligence Foundation – JC3001

Lecture 6: Search II: Informed Search I

Prof. Aladdin Ayesh (aladdin.ayesh@abdn.ac.uk)

Dr. Binod Bhattarai (binod.bhattarai@abdn.ac.uk)

Dr. Gideon Ogunniye, (g.ogunniye@abdn.ac.uk)

September 2025

Material adapted from:

Russell and Norvig (AIMA Book): Chapter 3 (3.5–3.6)

Malte Helmert (University of Basel)

Course Progression

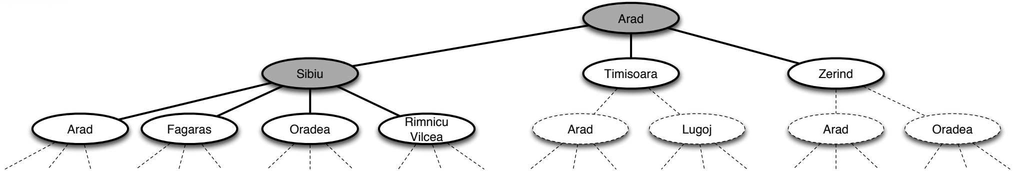
- Part 1: Introduction
  1. Introduction to AI ✓
  2. Agents ✓
- Part 2: Problem-solving
  1. Search 1: Uninformed Search ✓
  2. **Search 2: Heuristic Search**
  3. Search 3: Local Search
  4. Search 4: Adversarial Search
- Part 3: Reasoning and Uncertainty
  1. Reasoning 1: Constraint Satisfaction
  2. Reasoning 2: Logic and Inference
  3. Probabilistic Reasoning 1: BNs
  4. Probabilistic Reasoning 2: HMMs
- Part 4: Planning
  1. Planning 1: Intro and Formalism
  2. Planning 2: Algos and Heuristics
  3. Planning 3: Hierarchical Planning
  4. Planning 4: Stochastic Planning
- Part 5: Learning
  1. Learning 1: Intro to ML
  2. Learning 2: Regression
  3. Learning 3: Neural Networks
  4. Learning 4: Reinforcement Learning
- Part 6: Conclusion
  1. Ethical Issues in AI
  2. Conclusions and Discussion

Objectives

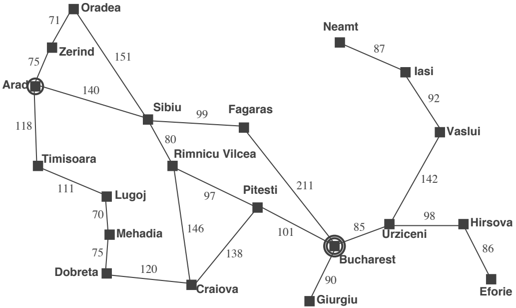
- Informed Search
  - Greedy Best First Search
  - A^* Search
- Heuristic Functions

- Recap
- Informed Search

Tree search example



After expanding Sibiu



- We have covered a number of search strategies which operate by systematically generating new states and testing them against a goal
- These are typically highly inefficient:
  - Apart from the state space and cost, they do not take any knowledge of the problem into account
  - They represent **uninformed search**
- By using problem specific knowledge (i.e. an **informed search**), we can perform better than the algorithms already encountered

Recap – Tree versus Graph

- Tree search
  - Maintains an **open list** (the fringe)
  - Expands nodes according to a strategy
- Graph search
  - Maintains an open-list and a **closed list**
  - Similar strategies to tree-search

What search strategies did we have?

Recap – Uninformed Search Strategies

What search strategies did we have?

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

- Breadth-first search is optimal only when **all step costs are equal** as it always expands the shallowest unexpanded nodes.
- UCS allows different **step-cost** functions
  - Expands the node with the lowest path cost (so far), $g(n)$
    $g(n)$ – This is the cost from the initial state to the current node
  - For completeness, we must have $g(n) > 0$
- Implementation: *fringe* is a **FIFO** ordered by cost, lowest cost first (priority queue)

function graphSearch(problem p, strategy s)

    $closed \leftarrow \{\}$ ▷ *We now keep a list of explored states*

    $frontier$.add$(newNode(p.initial))$

    loop

        if $frontier$ is empty **then**

            return $\boldsymbol{fail}$

        $n \leftarrow s$.removeChoice$(frontier)$

        $closed$.add$(n.state)$ ▷ *Where we keep states we already visisted*

        if p.goalTest$(n.state)$ **then**

            return getPath(n)

        for all $a \in p$.actions(n) **do**

            $n' \leftarrow a$.result$(n.state)$

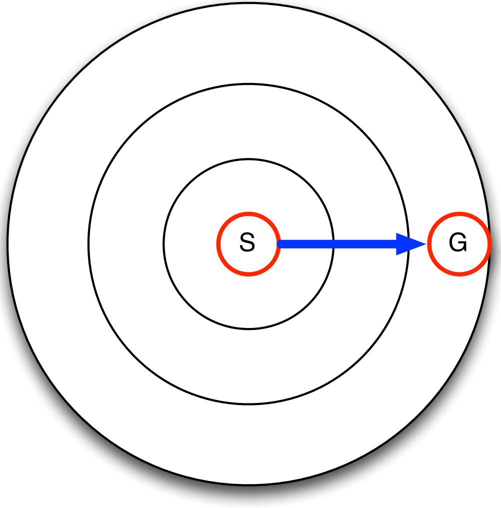
            if $n'.state \notin closed$ **then**

                $frontier$.add(n') ▷ *And only explore states we haven't visited*

- Recap
- Informed Search

Uniform Cost Search – Contours

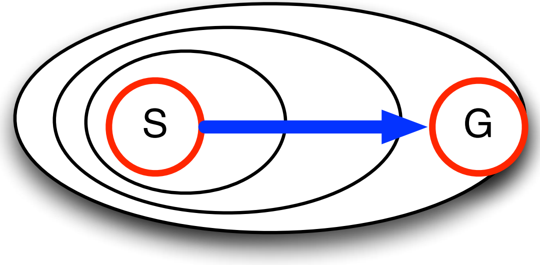
Uniform Cost Search – Contours



- Idea: select node for expansion based on an **evaluation function**, $f(n)$
  - estimate of “desirability”
  - expand most desirable unexpanded node
- This function could measure distance from the goal, so expand node with lowest evaluation
- Typically implemented via a **priority queue**
- Our search is a best-first one; we expand the node that (we believe) is the best one first

Greedy Best-first Search – Contours

Greedy Best-first Search – Contours



- Different informed searches use different evaluation functions.
- A key component is the heuristic function $h(n)$ which estimates the cost of the cheapest path from the current node to a goal node.
  Example: the straight line path from the current node to the destination in the route finding problem.
- One rule for heuristic functions: if we're at the goal node n, $h(n) = 0$.

Greedy Best-first Search

- This search tries to expand the node that is closest to the goal
- Why?

- This search tries to expand the node that is closest to the goal
- Why?
  - If we're close, then we'll probably get to the goal soon
- $f(n) = h(n)$

function graphSearch(problem p, strategy s)

    $closed \leftarrow \{\}$ ▷ *We keep a collection of explored states*

    $frontier$.add$(newNode(p.initial))$

    loop

        if $frontier$ is empty **then**

            return $fail$

        $n \leftarrow s$.removeChoice$(frontier)$

        $closed$.add$(n.state)$ ▷ *Where we keep states we already visisted*

        if p.goalTest$(n.state)$ **then**

            return getPath(n)

        for all $a \in p$.actions(n) **do**

            $n' \leftarrow a$.result$(n.state)$

            if $n'.state \notin closed$ **then**

                $frontier$.add(n') ▷ *And only explore states we haven't visited*

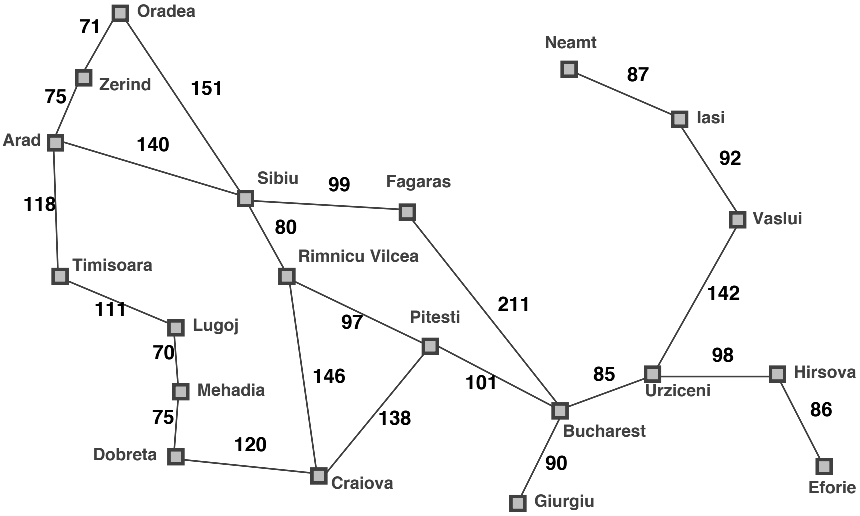
function uniformCostSearch(problem p, strategy s)
    $frontier$.add$(newNode(p.initial))$
    loop
        if $frontier$ is empty **then**
            return $fail$
        $n \leftarrow s$.removeChoice$(frontier)$
        $closed$.add$(n.state)$ ▷ *Where we keep states we already visisted*
        if p.goalTest$(n.state)$ **then**
            return getPath(n)
        for all $a \in p$.actions(n) **do**
            $n' \leftarrow a$.result$(n.state)$
            if $n'.state \notin closed$ **then**
                $v \leftarrow n'.cost$ ▷ *Order nodes by cost from initial*
                $frontier$.add(n', v) ▷ *And only explore states we haven't visited*

```
function greedySearch(problem p, heuristic h)
    closed ← {}                                   ▷ We keep a collection of explored states
    frontier.add(newNode(p.initial), 0)
    loop
        if frontier is empty then
            return fail
        n ← frontier.get
        closed.add(n.state)                       ▷ Where we keep states we already visited
        if p.goalTest(n.state) then
            return getPath(n)
        for all a ∈ p.actions(n) do
            n′ ← a.result(n.state)
            if n′.state ∉ closed then
                v ← h(n′.state)
                frontier.add(n′, v)               ▷ And only explore states we haven’t visited
```

function greedySearch(problem p, heuristic h)

- $closed \leftarrow \{\}$ ▷ *We keep a collection of explored states*
- $frontier$.add$(newNode(p.initial), 0)$
- **loop**
  - **if** $frontier$ is empty **then**
    - **return** $\boldsymbol{fail}$
  - $n \leftarrow frontier$.get
  - $closed$.add$(n.state)$ ▷ *Where we keep states we already visisted*
  - **if** p.goalTest$(n.state)$ **then**
    - **return** getPath(n)
  - **for all** $a \in p$.actions(n) **do**
    - $n' \leftarrow a$.result$(n.state)$
    - **if** $n'.state \notin closed$ **then**
      - $v \leftarrow h(n'.state)$ ▷ *Order priority queue by the heuristic estimate*
      - $frontier$.add(n', v) ▷ *And only explore states we haven't visited*

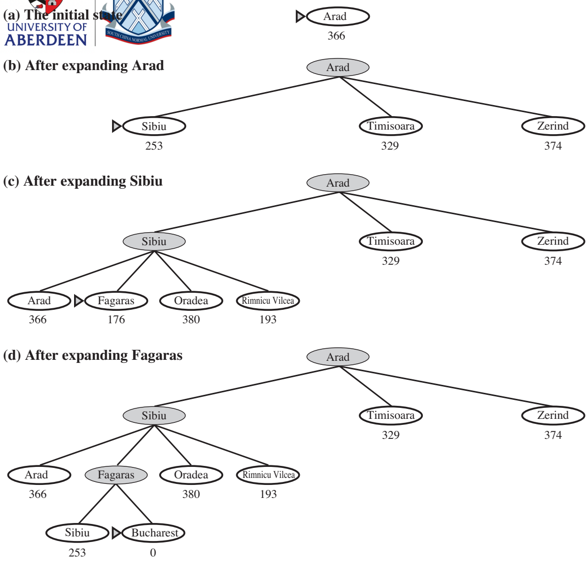
Example: Greedy Search



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
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

Example: Greedy Search

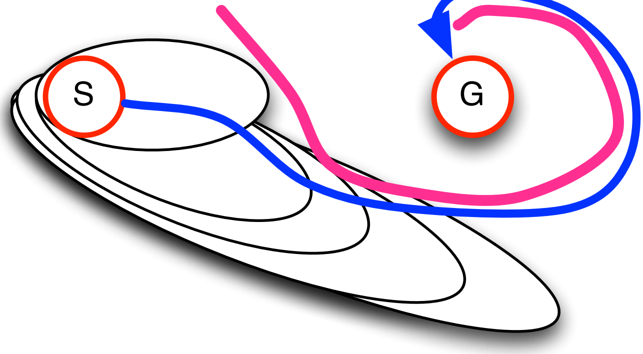


Example: Greedy Search

- We get all the way to Bucharest without expanding a single unnecessary node!
  - Is it optimal?
  - Is it complete?
- What if we want to go from Iasi to Fagaras?

Greedy Best-first Search – Contours

Greedy Best-first Search – Contours



Greedy best-first search is like DFS in that it prefers to follow a single path all the way to the goal, but will back up when it hits a dead end

To continue in the next session.