Artificial Intelligence

Week 4: Informed Search

COMP30024

April 1, 2021

THE UNIVERSITY OF MELBOURNE

Search

Search

Select action sequences to achieve a goal state in environments that are deterministic, observable, static with a completely specified state space.

- Uninformed/'brute force' search (last week):
  - Only generate successors and identify goals.
  - RN 3.5.
- Informed search (today):
  - Incorporate domain knowledge to prioritize exploration of more promising nodes.
  - RN 4.*.

Heuristic search is one of the pillars of artificial intelligence!

Dijkstra's Algorithm

- Used to find shortest paths in edge-weighted graphs G.
- Given start vertex s, finds the shortest path from s to every other vertex $v \in G$.
- Each round:
  - Expand the node x in the frontier with the lowest path cost $dist[x]$.
  - $dist(x) \leftarrow dist(u) + w(u \rightarrow x)$ if $dist(x) > dist(u) + w(u \rightarrow x)$.
  - Repeat until we expand target t.
- Optimal? If $w(u \rightarrow v) \geq \epsilon > 0 \, \forall \, (u, v) \in E$.
- Time/space complexity: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$.

Dijkstra's Algorithm

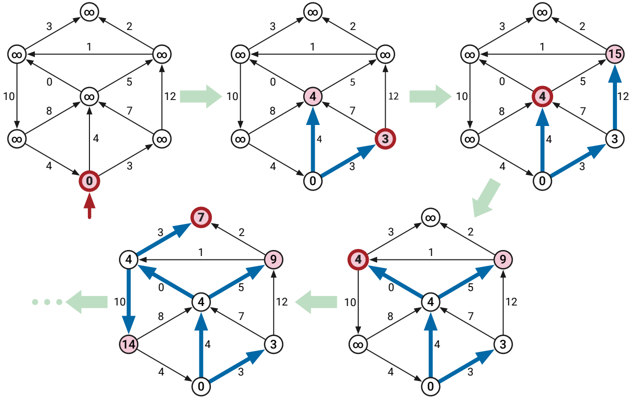


Figure 1: Jeff Erickson, Algorithms, Ch. 5, 2020

Idea: find shortest path from source s passing through v_i to unvisited vertex x minimizing $dist[x] = dist(s, v_i) + w(v_i \to x)$.

Heuristic Strategies

Incorporate domain knowledge via evaluation function $f(\sigma)$.

- $f(\sigma)$ orders nodes σ in the frontier by desirability of expansion.
    - Expand node $\sigma \in F$ with lowest $f(\sigma)$.
- Heuristic function $h(\sigma)$: Estimates cost of optimal path to goal state.
- Good heuristics avoid exploration of non-promising graph regions.
- Definitions:
    - σ: Search node - data structure describing a vertex in a graph.
    - Σ: Set of all possible nodes. $\sigma \in \Sigma$

Heuristic Strategies

$f(\sigma)$ orders nodes σ in the frontier by desirability of expansion.

- Admissibility:

$$h(\sigma) \leq \min(\text{cost}(\sigma \rightarrow \text{goal})) = h^*(\sigma),\ \forall\, \sigma \in \Sigma$$

  Optimistic; never overestimate cost to goal.

- Consistency/Monotony:

$$h(\sigma) \leq \text{cost}(\sigma, \sigma', A) + h(\sigma')\ \forall\, \sigma \in \Sigma, A(\sigma) = \sigma'$$

  Triangle inequality applied to costs of graph states.

- Domination:

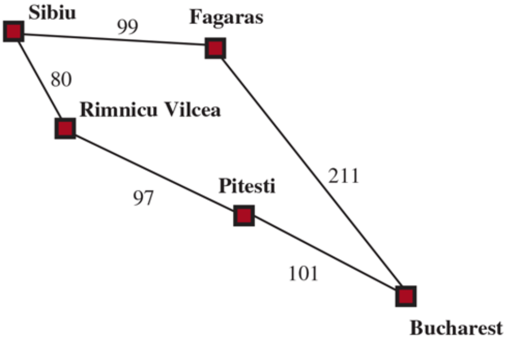
$$h_1(\sigma) \leq h_2(\sigma) \leq \text{goal cost}\ \forall\, \sigma \in \Sigma$$
$$\implies h_2 \text{ expands less nodes than } h_1$$

Want admissible h as close to the optimal solution h^* as possible, provided $h(\sigma) \leq h^*(\sigma) \quad \forall \sigma \in \Sigma$.

Generation v. Expansion

- Node Generation: Creation of data structure representing the node.
- Node Expansion: Generation of all children of current node.
- Uninformed search applies goal test upon generation.
- Informed search + (Dijkstra) applies goal test upon expansion.
  - Generation does not guarantee optimality.
  - Expand node with lowest $f(\sigma)$:
    - All other nodes in queue F have equal or larger path cost.
    - Goal test here guarantees optimality.

Generation v. Expansion



A* Search

- Rank nodes by the cost of the cheapest solution passing through each node.
- Evaluation function

$$
\begin{aligned}
f(\sigma) &= \text{cost}(\text{start} \to \sigma) + (\text{est.})\ \text{cost}(\sigma \to \text{goal}) \\
&= g(\sigma) + h(\sigma)
\end{aligned}
$$

- Completeness ✔
- (Graph) Optimality ✔ (provided $h(\sigma)$ is consistent)
- High space complexity $\mathcal{O}(b^d)$
- Rank nodes σ by the estimated total cost $f(\sigma)$
  - Search efficiency dependent on heuristic quality!

A* pseudocode

Start S, graph $G = (V, E)$, edge distance $e(u, v)$, heuristic $h(v)$.

$$d(v) \leftarrow \begin{cases} \infty & \text{if } v \neq S \\ 0 & \text{if } v = S \end{cases}$$

$Q :=$ the set of nodes in V, sorted by $d(v) + h(v)$

while Q not empty **do**

  $v \leftarrow Q.pop()$

  for all neighbours u of v **do**

    if $d(v) + e(v, u) \leq d(u)$ **then**

      $d(u) \leftarrow d(v) + e(v, u)$

    end if

  end for

end while

Practical

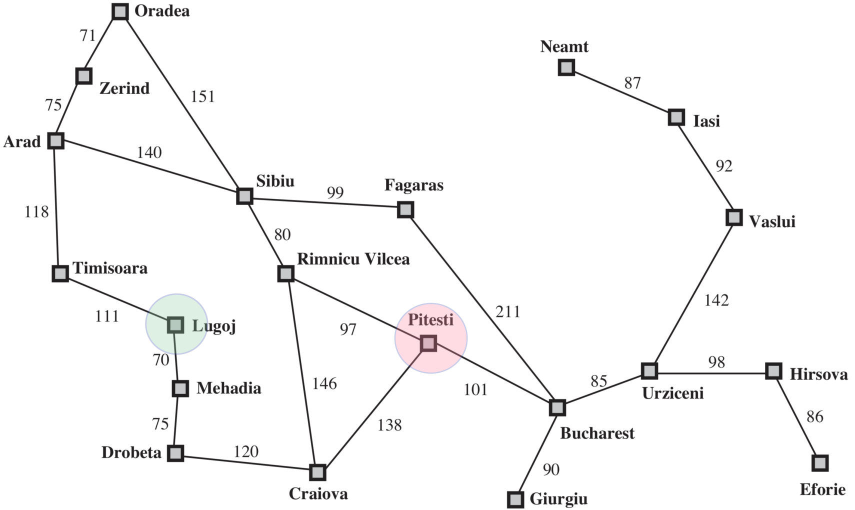


Figure 3.2 A simplified road map of part of Romania.

GBFS

- Greedy best-first search: Expand the node estimated to be closest to the goal - $f(\sigma) = h(\sigma)$.
- (Graph) Completeness ✔ (assuming finite state space)
- Optimality ✗ (requires perfect heuristic).
- What happens if the heuristic is set to the negative of the path cost? Assume graph search. What sort of blind search does this emulate?

$$h(\sigma) = -g(\sigma) \implies f(\sigma) = -g(\sigma)$$

GBFS

- What happens if the heuristic is set to the negative of the path cost? Assume graph search. What sort of blind search does this emulate?

$$h(\sigma) = -g(\sigma) \implies f(\sigma) = -g(\sigma)$$

GBFS

- What happens if the heuristic is set to the negative of the path cost? Assume graph search. What sort of blind search does this emulate?

$$h(\sigma) = -g(\sigma) \implies f(\sigma) = -g(\sigma)$$

- Lugoj $\rightarrow$ Timisoara $\rightarrow$ Arad $\rightarrow$ Sibiu $\rightarrow$ Oradea $\rightarrow$ Zerind $\rightarrow$ Arad $\rightarrow$?
- Graph search: Can't go from Arad to Sibiu because we already visited Sibiu with Arad as parent.

GBFS

- What happens if the heuristic is set to the negative of the path cost? Assume graph search. What sort of blind search does this emulate?

$$h(\sigma) = -g(\sigma) \implies f(\sigma) = -g(\sigma)$$

- Lugoj $\rightarrow$ Timisoara $\rightarrow$ Arad $\rightarrow$ Sibiu $\rightarrow$ Oradea $\rightarrow$ Zerind $\rightarrow$ Arad $\rightarrow$?
- Graph search: Can't go from Arad to Sibiu because we already visited Sibiu with Arad as parent.
- What do we do now?
  - Backtrack to deepest unexpanded node - Sibiu ...

GBFS

- What happens if the heuristic is set to the negative of the path cost? Assume graph search. What sort of blind search does this emulate?

$$h(\sigma) = -g(\sigma) \implies f(\sigma) = -g(\sigma)$$

- Lugoj $\rightarrow$ Timisoara $\rightarrow$ Arad $\rightarrow$ Sibiu $\rightarrow$ Oradea $\rightarrow$ Zerind $\rightarrow$ Arad $\rightarrow$?
- Graph search: Can't go from Arad to Sibiu because we already visited Sibiu with Arad as parent.
- What do we do now?
  - Backtrack to deepest unexpanded node - Sibiu ...
- What uninformed search method does this emulate?
  - Deepest node/furthest from root (assuming increasing path costs) gets expanded first.

A* practical

- Node σ: (State, Parent σ_P, $g(\sigma)$, $h(\sigma)$).
- Euclidean distance heuristic.

1. - Frontier (F): $\{\sigma_1\}$
     - σ_1 (root): (Lugoj, $\emptyset$, 0, ?)
   - F post-expansion: $\{\sigma_2, \sigma_3\}$
     - σ_2: (Mehadia, σ_1, 70, 280)
     - σ_3: (Timisoara, σ_1, 111, 320)
   - Explored set: $K = \{\sigma_1\}$

A* practical

- Node σ: (State, Parent σ_P, $g(\sigma)$, $h(\sigma)$).
- Euclidean distance heuristic.

1. - Frontier (F): $\{\sigma_1\}$
     - σ_1 (root): (Lugoj, $\emptyset$, 0, ?)
   - F post-expansion: $\{\sigma_2, \sigma_3\}$
     - σ_2: (Mehadia, σ_1, 70, 280)
     - σ_3: (Timisoara, σ_1, 111, 320)
   - Explored set: $K = \{\sigma_1\}$
2. - F: $\{\sigma_2, \sigma_3\}$
   - F post-expansion: $\{\sigma_3, \sigma_4\}$
     - σ_3: (Timisoara, σ_1, 111, 320)
     - σ_4: (Drobeta, σ_2, 145, 240)
   - $K \leftarrow K \cup \sigma_2$

A* practical

3.
- F: $\{\sigma_3, \sigma_4\}$
- F post-expansion: $\{\sigma_3, \sigma_5\}$
  - σ_3: (Timisoara, σ_1, 111, 320)
  - σ_5: (Craiova, σ_4, 265, 130)
- $K \leftarrow K \cup \sigma_4$

A* practical

3.
- F: $\{\sigma_3, \sigma_4\}$
- F post-expansion: $\{\sigma_3, \sigma_5\}$
  - σ_3: (Timisoara, σ_1, 111, 320)
  - σ_5: (Craiova, σ_4, 265, 130)
- $K \leftarrow K \cup \sigma_4$

4.
- F: $\{\sigma_3, \sigma_5\}$
- F post-expansion: $\{\sigma_3, \sigma_6, \sigma_7\} \leftarrow$ note goal generation
  - σ_3: (Timisoara, σ_1, 111, 320)
  - σ_6: (RV, σ_5, 411, 97)
  - σ_7: (Pitesti, σ_5, 403, 0)
- $K \leftarrow K \cup \sigma_5$

A* practical

3.
   - F: $\{\sigma_3, \sigma_4\}$
   - F post-expansion: $\{\sigma_3, \sigma_5\}$
     - σ_3: (Timisoara, σ_1, 111, 320)
     - σ_5: (Craiova, σ_4, 265, 130)
   - $K \leftarrow K \cup \sigma_4$

4.
   - F: $\{\sigma_3, \sigma_5\}$
   - F post-expansion: $\{\sigma_3, \sigma_6, \sigma_7\} \leftarrow$ note goal generation
     - σ_3: (Timisoara, σ_1, 111, 320)
     - σ_6: (RV, σ_5, 411, 97)
     - σ_7: (Pitesti, σ_5, 403, 0)
   - $K \leftarrow K \cup \sigma_5$

5.
   - F: $\{\sigma_3, \sigma_6, \sigma_7\}$
   - Upon goal expansion (σ_7), terminate. Note: Delay termination upon goal generation - more optimal path to the goal may exist.

A* practical

3.
- F: $\{\sigma_3, \sigma_4\}$
- F post-expansion: $\{\sigma_3, \sigma_5\}$
  - σ_3: (Timisoara, σ_1, 111, 320)
  - σ_5: (Craiova, σ_4, 265, 130)
- $K \leftarrow K \cup \sigma_4$

4.
- F: $\{\sigma_3, \sigma_5\}$
- F post-expansion: $\{\sigma_3, \sigma_6, \sigma_7\} \leftarrow$ note goal generation
  - σ_3: (Timisoara, σ_1, 111, 320)
  - σ_6: (RV, σ_5, 411, 97)
  - σ_7: (Pitesti, σ_5, 403, 0)
- $K \leftarrow K \cup \sigma_5$

5.
- F: $\{\sigma_3, \sigma_6, \sigma_7\}$
- Upon goal expansion (σ_7), terminate. Note: Delay termination upon goal generation - more optimal path to the goal may exist.

- Lugoj $\rightarrow$ Mehadia $\rightarrow$ Drobeta $\rightarrow$ Craiova $\rightarrow$ Pitseti

Uniform-cost Search / Dijkstra's Algorithm

Never expands a node with cost greater than the cumulative cost of the shortest path. Returns optimal path.

DA Pseudo-code

```
def dijkstra(graph, start, end, cost):
    frontier = [start]  # insert root node into queue
    for nodes in graph:
        total_cost[node] = infinity  # unknown path costs
    total_cost[start] = 0

    while frontier is not empty:
        # break ties randomly
        node = frontier.pop_lowest_cost()
        if node == end:
            return True
        for child in graph[node]:  # or neighbouring vertices
            cumulative_cost = total_cost[child] + cost(node, child)
            if cumulative_cost < total_cost[child]:
                # update path cost if lower than best recorded
                total_cost[child] = cumulative_cost

    return False
```

A* Search

A* Pseudo-code

```
def A*(graph, start, end):
    g(node) = ...  # returns path cost from start to node
    h(node) = ... # estimated cost of cheapest path from node to goal
    f(node) = g(node) + h(node)  # estimated cost of cheapest solution through node
    open = [start]  # nodes to be expanded, ordered by f(node), ascending
    closed = []  # nodes visited and expanded
    min_g = inf

    while open:
        node = open.popmin()  # extract node with lowest evaluation function
        if node not in closed or g(node) < min_g:
            # re-open expand node
            closed.append(node)
            min_g = g(node)

            if node == end:
                return True

            for each node_successor in successor(node):
                if h(node).isfinite():
                    open.append(node)

    return False
```