Uninformed search

Christos Dimitrakakis

March 6, 2024

Outline

Search to find a solution

- Input: Problem specification (e.g. route-finding)
- Output: Solution (e.g. a policy)

Algorithms for finding solutions

- Must search through solution space
- Will ideally return an optimal solution

Graphs in Search Algorithms

Problem Graph

- Specifies the problem.
- An abstraction of a real-world problem

Search Graph

- Saves the state of the search.

Graph definitions

Graph $G = \langle N, E \rangle$

A graph G is defined by:

- Set of nodes N
- Set of edges E, with $\langle x, y \rangle \in E$ and $x, y \in N$

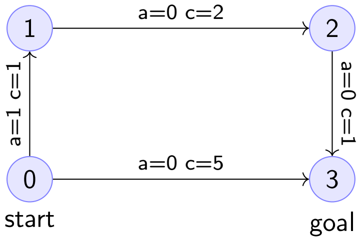
Labels and costs

- Nodes can be labelled as e.g. start and goal states.
- Arcs can be labelled according to actions.

Paths and cycles

- A path h from x to y in N is a sequence $\langle n_0, \ldots, n_k \rangle$ so that $n_0 = x, n_k = y$ and $\langle n_t, n_{t+1} \rangle \in A$.
- We write h_j for the j-th element of the path.
- A cycle is a path $\langle n_0, \ldots, n_k \rangle$ where $n_0 = n_k$.
- If a graph has no cycles, it is acyclic
- A state with no outgoing edges to other states is terminal

Graph labels



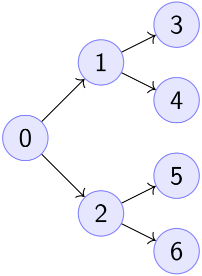
- Action labels a: Tell you which action traverses an edge
- Edge costs c: Optionally, c
- Node labels: Some nodes are goal or start nodes.

Notation

- For a node $i \in N$, we write $g(i) = 1$ if it's a goal label.
- For an edge $(i, j) \in E$, we write $c(i, j) \in \mathbb{R}$ for its cost.
- The total cost for a path $h = \langle x, \ldots, y \rangle$ from node x to a node y is

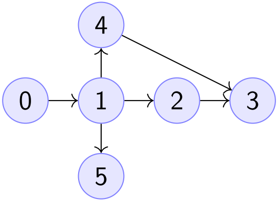
$$\sum_{k=1}^{|p|-1} c(p_k, p_{k+1}), \qquad \text{where } |p| \text{ is the length of } p$$

Tree example



- No matter where the goal is, there is a unique path to it.
- Depending on how the edges are ordered, finding it may require up to 6 steps.
- For binary trees of depth D, the worst-case complexity is 2^D.
- The branching factor is the number of children that each node can have.

Shortcut example



- There are two paths to node 3.
- Depending on how we search, we may find the longest path first.
- Why would we prefer one path to another?
- How should we search to find the *best* path?

State-space graphs

- $s \in S$: state space
- $a \in A$: action space (with $A_s \subset S$ available actions in state s)
- $\tau : S \times A \to A$: transition model (deterministic)
- When we reach a terminal state, we stop.

Graph specification

- Nodes: S
- Edges from node i: $\{(i, \tau(i, a)) | a \in A_s\}$

Problem specifications

One or more of the following:

- $g : S \to \{0, 1\}$: goal indicator
- $c : S \times A \to \mathbb{R}$: step cost or constraint.
- $r : S \times A \to \mathbb{R}$: step reward.

Solution specification

- $\pi : S \to A$ deterministic policy
- If both the problem and the policy are deterministic, the policy is open loop

The search graph S'

- Node 0 is root of the search graph.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \texttt{parent}(\texttt{parent}(s^i)), \texttt{parent}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\texttt{parent}(i)}$, with $d_0 = 0$.

Frontier: Keeping track of what to search next

At step 0, the frontier is $F_0 = \{0\}$ and set of searched nodes $S'_0 = \emptyset$.
At step $k = 0, 1, \ldots$:

- The frontier is F_k, and searched nodes S'_k.
- Select a node i, where $s^i \notin S'_k$.
- We select action a in node i, and observe $s' = \tau(s^i, a)$.
- $i + 1$ is now a child of i, with $s^{i+1} = s'$.
- Update the frontier $F_{k+1} = F_k \cup \{i + 1\} \setminus \{i\}$.
- In the end, no more nodes can be added: $F_k = \emptyset$ and $S'_k = S'_{k+1}$

Graph visualisation

Depth-first search

Generic depth-first search

global $S' = \emptyset$: Nodes searched
input $G = \langle N, E \rangle$: Graph.
input n : Current node
function $\texttt{DepthFirst}(G, n)$
$S' = S' \cup \{n\}$: mark n as searched
for $c \notin F : \langle c, j \rangle \in E$ **do**
  if $\texttt{DepthFirst}(G, j)$ **then**
    return 1.
  end if
end for

Discussion

- This function goes through all the nodes in the graph
- How can we use it to identify a paths to the goal?
- How can we modify it to identify all paths to the goal?
- How can we modify it to identify the shortest path to the goal?

Breadth-first search

Unlike Depth-First search, this cannot easily use a recursive function call implementation.

input $G = \langle N, E \rangle$: Graph.
input x : Start node
function $\texttt{BreadthFirst}(G, x)$
$S' = \emptyset$: Nodes searched.
$F = \{x\}$. Initialise the frontier
while $F \neq \emptyset$ **do**
  $s = \arg\min_{i \in F} d_i$. Select minimum depth node.
  $S' = S' \cup \{s\}$. Add s to the list of searched nodes.
  $F = F \setminus \{s\}$. Remove s from the frontier.
  $F = F \cup \texttt{child}(s)$. Add s's children to the frontier.
end while

Minimum-cost search

Note that DFS always adds the minimum depth node. We can instead add the minimum-cost node.

input $G = \langle N, E \rangle$: Graph.
input x : Start node
function $\texttt{BreadthFirst}(G, x)$
$S' = \emptyset$: Nodes searched.
$F = \{x\}$. Initialise the frontier
$c_x = 0$. Initialise the cost of node x
while $F \neq \emptyset$ **do**
  $n = \arg\min_{f \in F} c_f$. Select minimum cost node.
  $F = F \setminus \{n\}$. Remove n from the frontier.
  if $n \notin S'$ **then**
    $B = \texttt{child}(n) \setminus S'$. Get the set of unsearched children of n.
    $\forall b \in B$, $b_i = c_n + c(n, b)$. Calculate the total cost to each child b.
    $S' = S' \cup \{n\}$. Add n to the list of searched nodes.
    $F = F \cup B$. Add n's children to the frontier.
  end if
end while