State Spaces and Search

CS 4710 Course Notes

January 28, 2019

Search Problems

Many **problems** can be **represented** as a set of possible states Ω. **Goal** (or solution) states are defined as some set $G \subseteq \Omega$. To **pose** a problem is to determine:

(1) a **test** for identifying solution states

(2) a **generator** of possible solutions

Formal description of search: let Ω be the set of states (if a metric on this set exists, call this a **state space**). A **search problem** consists of:

(1) an initial state S_1

(2) transitions between states ("admissible moves") – can be given all at once (batch), or a rule for generation can be given (online/live)

(3) a **test function**

$$f(s \in \Omega) = \begin{cases} \text{True} & \text{if } s \in G \\ \text{False} & \text{otherwise} \end{cases}$$

Example. the "8-puzzle"

Definition of π. When dealing with ordered sets, let function π_i select the i'th member of our set. For instance: $\pi_1(\langle 1, 2, 3\rangle) = 1$. If $i < 1$, $\pi_i = \pi_1$, and if $i > |S|$, then $\pi_i(S) = \pi_{|S|}(S)$.

Define a tile as an ordered triple $\langle c, x, y\rangle$ such that $c \in C = \{\blacksquare, 1, 2, \ldots, 8\}$, and x, y are indices such that $1 \leq x, y \leq 3$.

Define a state of our 8-puzzle as a set of tiles T such that:

(1) $|T| = 9$

(2) For all $t_i, t_j \in T$: $\pi_1(t_i) \neq \pi_1(t_j)$

(3) For all $t_i, t_j \in T$: $\langle \pi_2(t_i), \pi_3(t_i)\rangle \neq \langle \pi_2(t_j), \pi_3(t_j)\rangle$

Rather than define each state transition, we'll describe what an admissible move is. Let $\langle x_s, y_s\rangle$ be π_2 and π_3 applied to the tile t_b containing $\blacksquare$. Let T_a be defined as the set of tiles adjacent to t_b, that is, the tiles t_a such that one of the following holds:

(1) $\pi_2(t_b) = \pi_2(t_a)$ and **either** $\pi_3(t_b) = \pi_3(t_a) - 1$ **or** $\pi_3(t_b) = \pi_3(t_a) + 1$

(2) $\pi_3(t_b) = \pi_3(t_a)$ and **either** $\pi_2(t_b) = \pi_2(t_a) - 1$ **or** $\pi_2(t_b) = \pi_2(t_a) + 1$

Example of admissible move

■	2	3		5	2	3
5	7	1	⇔	■	7	1
4	8	6		4	8	6

Note we can move in either direction – our admissible moves can all be reversed. Not every set of transitions has this property.

We define our goal states directly as the singleton set containing:

1	2	3
8	■	4
7	6	5

Our test function is a function that simply tests for equality with the single goal state described above. **(end example)**

Note that Ω can be finite or infinite. Even if Ω is finite, a search algorithm over it may still fail to halt.

Search through Ω can be represented as a **directed graph**. We define a directed graph as a set of nodes $N = \{n_1, \ldots, n_m\}$ (representing states), and a set of ordered pairs E with each element a member of N (representing state transitions).

Sometimes we wish to attach a parameter to a transition associated with how costly that move is. This requires a weighted digraph, where each edge is a triple, consisting of two nodes and a number w called a **weight** (or **cost**).

Search Trees and Types

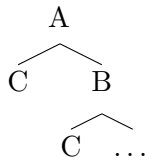
Let the **in-degree** of a node n refer to the number of edges E in our digraph such that $\pi_2(E) = n$. In other words, the in-degree is how many edges point into n. We say a node is a **root** iff its in-degree is 0. Define a **cycle** as two edges E_i, E_j such that $E_i = \langle n_x, n_y \rangle$ and $E_j = \langle n_y, n_x \rangle$, or any sequence of nodes arrived at via admissible moves where the same node appears twice or more.

Define a **tree** as a digraph with a root and no cycles.

We can represent a set of states and its transitions through a digraph. In searching such a digraph, we can generate a **search tree** which traces admissible moves in the order made.

Let our starting state be A and $\Omega = \{A, B, C\}$ with transitions $A \to C, B \to C, A \to B, B \to A$.

From this state digraph and starting node, we can generate a search tree:



Notice once we take A to B, we can go back to A again. The existence of this cycle in our state digraph implies that our search tree is infinite.

Finite sets of states can be searched through brute-force in order to solve a search problem. Notice that given a set of statements in PC, we can construct a finite number of valuations which represent all possible valuations. Thus, we can treat truth-tables in PC as a sort of brute-force search strategy.

If we use a brute-force strategy, we call this "uninformed" search. "Informed" search is when we take some available shortcut. We say a problem is **hard** if it admits of no shortcut (every state must be checked), and brute-force is all we have available. Otherwise we say it's **easy**.

Uninformed Search Example

Consider the traveling salesman problem (TSP). In TSP, we are presented with a set of cities, each pair of which has a defined distance between them. The goal is to find the path through each city that is shortest.

Refer to Wikipedia for a thorough treatment of TSP. For our purposes it is enough to note that every pairwise distance will have to be checked. Given this, as the number of cities grows, we have an exponentially larger set of distances to handle (consider how one might prove this). The reason this is true is that there's no way to intuit the road's length, say, between New York and Los Angeles without actually measuring it (or looking it up) at some point.

In these cases, we say that our search space doesn't contain a shortcut, or that it doesn't contain information relevant to our search.

Informed Search Example

For an informed example of search, we'll look at the opposite case – when a state space has a lot of relevant information for our search. Consider Newton's method, defined below, as a method for finding the roots of a real-valued function.

Newton's method: find x such that $f(x) = 0$ given $x, f(x) \in \mathbb{R}$.

Define $f'(x)$ as the derivative of $f(x)$ (and assume it exists).

(1) Pick some initial guess x_1.

(2) Calculate a better guess using the information present in a derivative (that is, slope):

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

(3) Continue until, for some chosen δ close to 0, $|f(x_n)| < \delta$.

This can be viewed as a heuristic search. Our initial function $f(x)$ can be viewed as an infinite state space. The derivative $f'(x)$ can be used to construct a move in the "right direction", that is, toward the zero. When all goes according to plan, we converge to the function's zero (under arbitrary tolerance δ).

While Newton's method, as presented above, is efficient (it exploits a shortcut), it is not **complete**. We can show its incompleteness by giving an example of a function, as stated above, for which it does not hold. (Note that typically Newton's method is presented with all of the assumptions made on $f(x)$ necessary to guarantee completeness.)

Pick $f(x) = 1 - x^2$, and $x_1 = 0$. Note that $f'(x) = -2x$, and so $f'(0) = 0$. This, however, means $x_2 = 0 - \frac{1}{0}$! So, as we can see, some f such that $f(x) \in \mathbb{R}$ are not suitable for Newton's method (the method is undefined).

Completeness for a search algorithm means it finds a solution if one exists.