Search

Formalization

Introduction to Artificial Intelligence

Chandra Gummaluru
University of Toronto

Version W22.1

Acknowledgements

- The following is based on material developed by many individuals, including (but not limited to):
  - Sheila McIlraith
  - Bahar Aameri
  - Fahiem Bacchus
  - Sonya Allin

- We will first consider a special type of game called a "search problem".
- It makes the following assumptions:
  - There is only one-player.
  - The set of terminal states, $T \subseteq S$, is replaced with a set of goal states, $G \subseteq S$, where $u(s)$ is constant across G; all goal states are equally beneficial.

Example: Slider Puzzle

- Move 8 tiles numbered from 1 to 8 on a 3×3 board so that the tiles are in order.
- The search is over the set of all board configurations.
- The element we seek is the specific configuration in which the tiles are in order.
- Finding such board configurations is non-trivial.
- However, given a board configuration, it is easy to check that it is a valid solution.

1	2	3
4		6
7	5	8

s_0

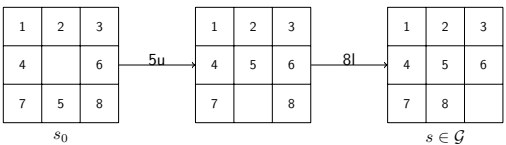
1	2	3
4	5	6
7	8	

$s \in \mathcal{G}$

- One way to perform a search is as follows:
  1. Check if the current state, s is a goal.
  2. If not, perform an action, $a \in A(s)$, resulting in a new state, $s' = a(s)$.
  3. Set the current state to s' and repeat until a goal is found.

Example: Searching in the Slider Puzzle

- In the slider puzzle, we define $\mathcal{S}$ as the set of all possible board configurations, and $\mathcal{G}$ as the configuration where the tiles are in order.
- Given some initial configuration, we can search for a goal by moving the tiles one at a time until they are in order.

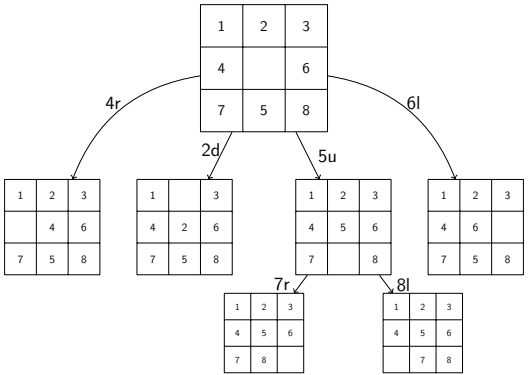


- We define a set, $\mathcal{O}$, called the **open**, which holds known but unexplored states.
- To search, we remove a state, s, from $\mathcal{O}$. If s is the goal, the search is complete. Otherwise, we add the successors of s to $\mathcal{O}$ and search again.

```
1: O ← {s_0}                                  ▷ initialize O with s_0
2: procedure SEARCH(O)
3:     if O = ∅ then    ▷ the search failed to find a goal
4:         return NULL
5:     s ← REMOVE(O)
6:     if s ∈ G then
7:         return s
8:     O ← O ∪ S(s)                           ▷ add s's successors to O
9:     SEARCH(O)
```

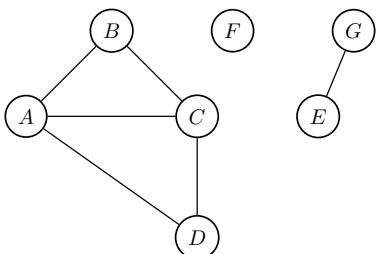
- It turns out that the search takes place on a tree.

Example: Search Tree for the Slider Puzzle

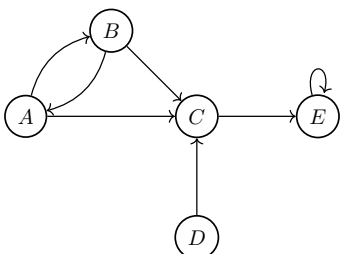


- A **tree** can be thought of as a type of graph.
- A **graph** is used to model relationships between elements of a set, $\mathcal{V}$, called its **vertices** by connecting them through a set of **edges**, $\mathcal{E}$.
- An edge between two vertices, $u, v \in \mathcal{V}$ can be
  - **directed**, in which case, each edge is represented as a ordered-pair, (u, v)
  - **undirected**, in which case, each edge is represented as a pair, $\{u, v\}$.
- An ordered sequence of edges, $\langle e_1, \ldots, e_n \rangle$, where $e_i = (v_i, v_{i+1})$ defines a **path** between v_1 and v_{n+1}; sometimes we represent the path with the corresponding vertex sequence, $\langle v_1, \ldots, v_{n+1} \rangle$.
- Two vertices are **related** if there exists at least one path between them.
  - If v follows u in a path, then u is an **ancestor** of v and v is a **successor** of u.
  - If v immediately follows u in a path, then u is a **parent** of v and v is a child of u.
- Each edge, e, may be associated with a weight $w(e)$.

- Below is a graph with:
  - $\mathcal{V} = \{A, B, C, D, E, F, G\}$
  - $\mathcal{E} = \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{C, D\}, \{E, G\}\}$.



- Below is a graph with:
  - $\mathcal{V} = \{A, B, C, D, E\}$
  - $\mathcal{E} = \{(A, B), (A, C), (B, A), (B, C), (C, E), (D, C), (E, E)\}$.



- A **tree** is a directed acyclic connected graph.
- A graph is **cyclic** if there exist some path, $\langle e_1, \ldots, e_n \rangle$, for which $e_1 = (v_1, \ldots, v_2)$, and $e_n = (v_{n-1}, v_n)$, where $v_n = v_1$.
- A graph is **connected** if from every vertex, there is a path to every other vertex.

- The order in which the successors are explored (i.e., the removal order) changes the search algorithm's properties.

- There are four properties of particular interest:

  - **Time Complexity**: the order of the number of states that must be generated before the algorithm terminates.
  - **Space Complexity**: the order of the maximum number of states that exist on the open at any given iteration.
  - **Completeness**: the algorithm always find a goal if one exists.
  - **Optimality**: the algorithm always finds the cheapest solution if multiple exists.

- The ideal search algorithm is complete, optimal, and minimizes the time and space complexity, but we will see that this is difficult to achieve.

- We can reduce the time complexity if all actions have positive costs, i.e., $c(a) > 0$.
- In this case, we can prune paths as follows:
  - When a path $p = \langle s_0, \ldots, s_n \rangle$ is being expanded to $p' = \langle s_0, \ldots, s_n, s_{n+1} \rangle$ if $s_{n+1} = s_i$ for some $1 \leq i \leq n$, do not add p' to the open.
  - The rationale is that if actions have positive costs, then $p_{0:i}$ is a cheaper way to get to s_i than p', i.e., $c(p') \geq c(p_{0:i})$.
- This is called **local path checking**.

- We can reduce the time complexity even further if the first time the search algorithm explores a path to any given state, s, the path is the cheapest one to s.
- In this case, we can prune paths as follows:
  1. Define a list of **closed** (i.e., explored) states, $\mathcal{C}$.
  2. Initially, $\mathcal{C} = \emptyset$.
  3. Whenever we ever discover a path, $p = \langle s_0, \ldots, s_n \rangle$:
     - If $s_n \in \mathcal{C}$, do not add p to $\mathcal{O}$.
     - If $s_n \notin \mathcal{C}$, then add s_n to $\mathcal{C}$.
- This is called **global path checking**.

- It turns out that the general search algorithm is complete on finite trees:
  - A tree is finite if it has a finite branching factor and depth.
  - The **branching factor**, denoted b, is the number of *children* of the vertex with the maximum number of children.
  - The **depth**, denoted m, is number of *parents* (not necessarily immediate) of the vertex with the *maximum* number of parents.