3. Discrete Space Search

Course: Introduction to AI
Instructor: Saumya Jetley
Teaching Assistant(s): Raghav Awasty & Subhrajit Roy

October 10, 2022

What is a discrete space?

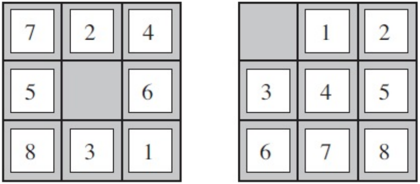
- A discrete space is one that is mapped by finite number or countably infinite number of states $s \in S$.
- Some states are *desirable*, some *undesirable* and some *neutral*.
- An agent can perform *actions* to move between states.

Goal of search

- To identify a sequence of actions that takes one from an initial state through to the goal state

Goal of search

- Example:
  - The 8-puzzle problem consists of a 3×3 grid contains eight tiles numbered one through eight.
  - A tile can be moved in to blank position adjacent to it, thus creating a blank in the tile's original position.
  - The goal is to go from the initial position to the final position through minimum number of moves.



Start State Goal State

Structure of search*

- Search starts from the initial state
- The edges out of this state are the possible actions
- The nodes at the output correspond to the resultant states

- Let's review this as a **tree** and think through concepts of:
  - **branching factor**, and
  - **reachability**

Deeper into Tree-Search*

- Let's consider a simpler example of traversing cities
- We want to go from Nagpur to Delhi, given a connectivity graph!

- Let's build this tree: is it **finite** or **infinite**?

Optimisations to Tree Search*

Those who forget their history, are bound to repeat it!

Let's divide the nodes into:

- Frontier set
- Explored set
- Unexplored set

Tree versus Graph Search

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Evaluation of search algorithms*

- Completeness
- Optimality
- Time complexity
- Space complexity

Uninformed Search Algorithms

The only knowledge we have is whether a state is goal state or non-goal state

- Breadth-first search
- Depth-first search
- Bidirectional search

1. Breadth-first search

- Search strategy that proceeds by expanding the shallowest unexpanded node
- Implemented by using a FIFO queue for the frontier

Performance analysis

- Is it **Complete**?
- Is it **Optimal**?
- What is the **Time-** and **Space-** complexity?

1. Breadth-first search

Performance analysis

- Is it **Complete**?
  DFS finds the shallowest goal node as the solution
- Is it **Optimal**?
  Not necessarily. Depends on path costs!
- What is the **Time-** and **Space-** complexity?
  $O(b^d)$ - can tweak graph-check to make this $O(b^{d-1})$

2. Depth-first search

- Search strategy that proceeds by expanding the deepest unexpanded node
- Implemented by using a LIFO queue for the frontier

Performance analysis

- Is it **Complete**?
- Is it **Optimal**?
- What is the **Time-** and **Space-** complexity?

2. Depth-first search

Performance analysis

- Is it **Complete**?
  In worst-case, will find the deepest goal state
- Is it **Optimal**?
  Not if solution cost increases with depth - which is often the case
- What is the **Time-complexity**
  $O(b^d)$
- What is the **Space-complexity**?
  $O(bm)$ - to enable backtracking

3. Bidirectional search

- Search strategy that proceeds by expanding from both the initial and goal state simultaneously
- Which do we use - BFS or DFS ?
  (Hint: *interacting frontiers or wavefronts*)

Performance analysis

- Is it **Complete**?
- Is it **Optimal**?
- What is the **Time-** and **Space-** complexity?

3. Bidirectional search

Performance analysis

- Is it **Complete**?
  Yes.
- Is it **Optimal**?
  Yes, need to wait until the connecting node is expanded.
- What is the **Time-** and **Space-** complexity?
  $O(b^{d/2})$

Quick review!!

Depth-first Search (DFS)

- DFS applies to trees or graphs?
- What is the main advantage of DFS?
- Is DFS guaranteed to find the minimum cost solution?

Cost is critical

If cost is the most critical concerns, then how about we search through cheaper action sequences first?

- Uniform cost search*
- Completeness — Optimality — Space and Time Complexity

Informed Search Algorithms

What are they? When we have some information about the goodness of states!

Example information?

Informed searches

- Greedy best-first search
- A* search

Greedy best-first search

- Relies on the heuristic $h(n)$ to prioritise expansion
- The cost evaluation function $f(n) = h(n)$
- Expands the node closest to the goal!

Greedy best-first search

Performance Analysis:

- Completeness
  No, can get stuck in loops
- Optimality
  Not guaranteed to render lowest cost solution
- Time complexity
  $O(b^d)$, (in worst case) but a good heuristic can give dramatic improvement (d is solution depth)
- Space complexity
  $O(b^d)$ – keeps all nodes in memory

A* search algorithm

- The cost evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is cost to reach node n and $h(n)$ is cost to reach goal from node n
- Thus, $f(n)$ is cheapest cost of solution through node n

A* search algorithm

Performance Analysis:

- Completeness
  Guaranteed, if $h(n)$ satisfies some criteria
- Optimality
  Guaranteed, if $h(n)$ satisfies some criteria
- Time Complexity
  $O(b^{\epsilon d})$, (in worst case) but a good heuristic can give dramatic improvement (m is max depth of search space)
- Space complexity
  $O(b^d)$ – keeps all nodes in memory

A* search algorithm

Core properties:

- Admissibility of $h(n)$
- Consistency of $h(n)$

Emergent properties:

- If $h(n)$ is consistent, then value of $f(n)$ along any path is non-decreasing
  No use continuing if initial cost is above the optimal cost
- When A* selects a node for expansion, it has selected the shortest path to that node
  Otherwise, would have selected path through some n' before n

Overview

1. **Introduction**
   - Searching in discrete spaces
   - Goal of search
   - Structure of search
2. **Tree vs. Graph search**
3. **Evaluating search algorithms**
4. **Uninformed search algorithms**
   - Breadth-first search
   - Depth-first search
   - Bidirectional search
5. **Cost is key: Uniform cost search**
6. **Informed search algorithms**
   - Greedy best-first search
   - A* search