Lecture 08
Graph Search Algorithm
Review & D* Algorithm

Instructor: Jingjin Yu
Outline

Basic graph search algorithms

⇒ Graph
⇒ BFS
⇒ DFS
⇒ Uniform cost
⇒ A*

All pairs shortest path

A word on the principle of dynamic programming

D* algorithm intro
Discrete Search Algorithms

What are discrete search algorithms?

⇒ An algorithm whose **data structure** is a graph
⇒ That is, a set of “nodes” with “neighborhoods”
⇒ The graph may be explicit
⇒ Or it may be implicit
  ⇒ What are the nodes here?
  ⇒ And neighborhoods?
⇒ And even not fully known!

![8 Queens](image1.png)

![Sodoku](image2.png)

![C&C: Red alert](image3.png)
Applications of Discrete Search Algorithms

- Navigation
- Robot motion planning
- Competitive chess
- Game AI and go
Graph Basics

A graph $G = (V, E)$ is a set of vertices $V$ and a set of edges $E$

Example

- $V = \{A, B, C, G, S\}$
- $E = \{(A, B), (A, C), (A, G), (A, S), (B, S), (B, C), (C, G)\}$

Variations

- A graph may be directed
- There can be multi-edges between two vertices
  - This is called a multi-graph
  - We will not consider multi-graphs in our course

Basic properties

- An undirected graph with $n$ vertices has at most $\frac{n^2 - n}{2}$ edges
  - When this happens, the graph is a complete graph
- A graph is connected if there is a path between any two vertices
- A connected graph with $n - 1$ edges is a tree
A Generic Graph Search Algorithm

input: \( G = (V, E), x_I, x_G \)
AddToQueue \((x_I, Queue)\);  // Add \( x_I \) to a queue of nodes to be expanded
while (!IsEmpty (Queue))
    \( x \leftarrow \text{Front} (Queue) \);  // Retrieve the front of the queue
    if (\( x.expanded \) == true) continue;  // Do not expand a node twice
    \( x.expanded = \) true;  // Mark \( x \) as expanded
    if (\( x \) == \( x_G \)) return solution;  // Return if goal is reached
    for each neighbor \( n_i \) of \( x \) // Add all neighbors of to the queue
        if (\( n_i.expanded \) == false) AddToQueue \((n_i, Queue)\)
return failure;

Different graph search algorithms (breadth first, depth-first, uniform-cost, ... ) differ at the function AddToQueue

To retrieve the actual path, use back pointers
Classical Search Algorithms

- **Breadth-first search (BFS)**
  - Always add new nodes at the end of the queue

- **Depth-first search (DFS)**
  - Always add new nodes in the front of the queue

- **Uniform-cost (Dijkstra’s)**
  - Always keep the node with the best cost in the front of the queue

- **A***
  - Similar to uniform-cost, but also uses a guess of distance to goal
Breadth First search

Problem graph (a weighted directed graph)

Running BFS graph search (we do not use weights here)

Path can be retrieved by storing S in A and A in G as back pointers.
Depth First search

Running DFS graph search (again we do not use weights here)
Uniform-Cost Search

Maintain queue order based on current cost

Produces **optimal** path!

⇒ This is basically the Dijkstra’s algorithm
A* Search

Maintain queue order based on current cost + guess

State | $h(x)$
--- | ---
S | 7
A | 6
B | 4
C | 2
G | 0
All Pairs Shortest Paths (Floyd-Warshall)

Floyd-Warshall is a type of dynamic programming

⇒ So are most other optimal search algorithms (e.g., Dijkstra, A*)

⇒ Pseudo-code

```
1 let dist be a |V| × |V| array of minimum distances initialized to ∞
2 for each vertex v
3  dist[v][v] ← 0
4 for each edge (u, v)
5  dist[u][v] ← w(u, v) // the weight of the edge (u, v)
6 for k from 1 to |V|
7  for i from 1 to |V|
8    for j from 1 to |V|
9      if dist[i][j] > dist[i][k] + dist[k][j] then
10         dist[i][j] ← dist[i][k] + dist[k][j]
11 end if
```

⇒ Does not directly produce paths

⇒ A path tree from any vertex can be constructed by adding back pointers

⇒ Runs in time $O(|V|^3)$

Algorithm outline adapted from Wikipedia page on Floyd-Warshall
Principle of Dynamic Programming

Dynamic programming is a type of recursion

- It breaks a big problem into smaller pieces
  - E.g., $P(n) = P(n_1) + P(n_2)$ with $n = n_1 + n_2$
  - May need to try all combinations of $n_1$ and $n_2$

- The problem must have structures that allow computation to be reused
  - E.g., for path optimality, any segment of an optimal path must also be optimal

- Divide-and-conquer search algorithms are special cases
  - $P(n) = n_1 + n_2$ for $n_1 = n_2 = \frac{n}{2}$
  - Dijkstra’s, A*, Floyd-Warshall are all types of dynamic programming
Dynamic Programming in Search

Recall **AddToQueue** is the crucial step of graph search

- BFS and DFS do not care about edge costs
- Uniform cost and A* do
- This is in fact dynamic programming!
- Priority of unvisited = cost-to-come + estimated cost-to-go

I.e., \( f = g + h \)

- \( g \), cost-to-come, is fixed
- \( h \), estimated cost-to-go, determines algorithm behavior
- Uniform cost: \( h = 0 \)
- A*: \( h \) is consistent
- Other behaviors are possible by changing \( h \)
D* Algorithm Intro

D* and D*-lite stand for “dynamic” A*

- It runs over an environment with potentially unknown obstacles
- Initially, for parts of the environment that is unknown, assume no obstacle
- Runs A* backwards to find an initial optimal solution
- Then, execute the path

- If we hit an obstacle along the way
  - Update the node itself to be unavailable
  - Put all its descendant nodes on the queue for search again
  - Do the above step recursively
  - Restart the A* search to find an optimal path

- Repeat the previous two steps

- One may view D* as running many A* searches

- A new A* search will be run as a previously unknown obstacle is met