Discrete State Search for Robotics

CS480
Nathan Sprague

Generic Graph Search Algorithm (without weighted edges)

From our book:

```
Procedure GraphSearch(start, goal)
    OPEN := {start}
    CLOSED := {}
    found := False
    while (OPEN not empty) and (not found)
        Select a node n from OPEN.
        OPEN := OPEN – {n}
        CLOSED := CLOSED U {n}
        if n ∈ goal then
            found := True
        else
            Let M be the set of all nodes
            directly accessible from n which
            are not in CLOSED.
            OPEN := OPEN U M
```

Determines the order that states are searched.

Depends on the problem

Depth First Search

```
Procedure DFS(start, goal)
    OPEN := An empty stack.
    OPEN.push(start)
    CLOSED := {}
    found := False
    while (OPEN not empty) and (not found)
        n = OPEN.pop()
        CLOSED := CLOSED U {n}
        if n ∈ goal then
            found := True
        else
            for each state m accessible from n
                if m ∉ CLOSED and m ∉ OPEN
                   OPEN.push(m)
```

Determines the order that states are searched.

Depends on the problem

“Correct” version of Figure 6.1

Chosen	Open	Closed
–	S	–
S	A	S
A	B, C, D	A, S
D	B, C, F	A, S, D
F	B, C, G	A, S, D, F
G	B, C	A, S, D, G, F

Breadth First Search

```
Procedure BFS(start, goal)
    OPEN := An empty Queue.
    OPEN.enqueue(start)
    CLOSED := {}
    found := False
    while (OPEN not empty) and (not found)
        n = OPEN.dequeue()
        CLOSED := CLOSED U {n}
        if n  ∈ goal then
            found := True
        else
            for each state m accessible from n
                if m ∉  CLOSED and m ∉ OPEN
                   OPEN.enqueue(m)
```

Determines the order that states are searched.

Depends on the problem

Search Nodes

```
Type Node
    State   state
    Node    parent_node
    Number  path_cost


Function CreateNode(State state, Node parent,
                    number step_cost)
    Return a node with
        state = state
        parent_node = parent
        path_cost = step_cost + parent.path_cost
```

Generic Graph Search With Nodes

```
Function GraphSearch(start, goal)
    OPEN := { CreateNode(start, NONE, 0) }
    CLOSED := {}
    found := False
    while (OPEN not empty) and (not found)
        Select a node n from OPEN.
        OPEN := OPEN – {n}
        CLOSED := CLOSED U {n.state}
        if n.state ∈ goal then
            found := True
        else
            Let M be the set of all nodes
            directly accessible from n which
            are not in CLOSED.
            OPEN := OPEN U M
    if found
        return a plan created by following
        parent links back from n
    else
        return FAILURE
```

Determines the order that states are searched.

Depends on the problem

Djikstra's Algorithm

```
Procedure Djikstra(start, goal)
    OPEN := An empty Priority Queue.
    n = CreateNode(start, NONE, 0)
    OPEN.enqueue(n, 0)
    CLOSED := {}
    found := False
    while (OPEN not empty) and (not found)
        n = OPEN.dequeue()
        CLOSED := CLOSED U {n.state}
        if n.state ∈ goal then
            found := True
        else
            for each state m accessible from n
                if m ∉ CLOSED and m ∉ Any Node in OPEN
                   m_node = CreateNode(m, n, Cost of n->m)
                   OPEN.enqueue(m_node, m_node.path_cost)
```

(Missing detail: If m is already in a node in OPEN, then that node should be replaced if m_node has a lower cost.)

A*

- Exactly like Djikstra's, Except, priority is calculated as:

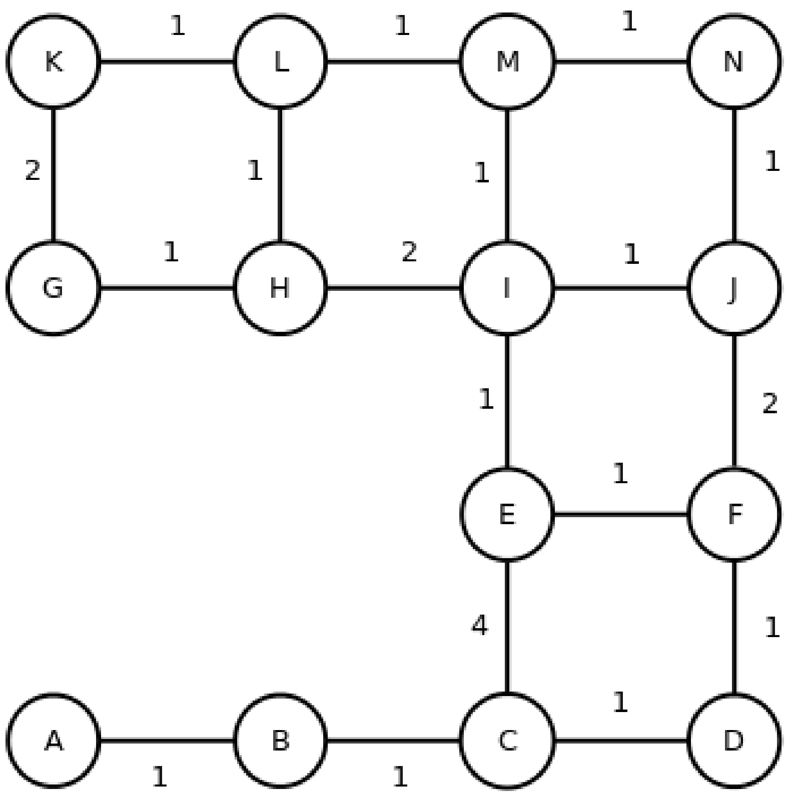
  $f(n) = g(n) + h(n)$

    $g(n)$ = Total path cost to that node

    $h(n)$ = Estimated cost to the goal

- As long as h(n) doesn't overestimate, A* is guaranteed to find an optimal path.

Homework



<u>A* Heuristic</u>

h(n) = Minimum number of edges between n and the goal.

For example:
h(C) = 2
h(G) = 6

Since all weights are at least 1, this is guaranteed not to overestimate the path cost.

Homework

- Fill out Chosen/Open/Closed tables (like figures 6.1-6.3) and the final path for:
  - DFS: start=I, goal=A
  - BFS: start=I, goal=A
  - Djikstra: start=E, goal=A
  - A*: start=E, goal=A
- All “ties” should be broken by alphabetical order: State 'A' is selected before state 'B'
- For DFS and BFS, the Open column should be formatted as follows:

  (state, parent), e.g. ('A', 'B')

- For Djikstra, the Open column should be formatted as follows:

  (state, parent, path_cost)

- For A*, the Open column should be formatted as follows:

  (state, parent, path_cost, f(n))