Assignment Information

Rick Valenzano and Sheila McIlraith

UNIVERSITY OF TORONTO

Assignment Format

- Investigating the impact of tie-breaking and re-expansions on A*, WA*, and GBFS

- Given a codebase with A* implemented
  - Will have to add WA* and GBFS
  - Will have to add different tie-breaking rules
  - Will have to add re-expansion options

- Three proofs as well

Tie-Breaking

- In A*, you can have two nodes with the same f-cost
  - Which should you prefer?

- What about in WA* and GBFS?

Re-Expansions

- Comparing WA* and GBFS when you reopen nodes and when you do not

```
def OCL(s_I):
    OPEN ← {s_I}, CLOSED ← {},
    g(s_I) = 0, parent(s_I) = ∅
    while OPEN ≠ {}:
        p ← SelectNode(OPEN)
        if p is a goal, return path to p
        for c ∈ children(p):
            if c ∉ OPEN ∪ CLOSED:
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                OPEN ← OPEN ∪ {c}
            else if g(c) > g(p) + κ(p, c):
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                if c ∈ CLOSED:
                    OPEN ← OPEN ∪ {c}
                    CLOSED ← CLOSED − {c}
        OPEN ← OPEN − {p}, CLOSED ← CLOSED ∪ {p}
    return No solution exists
```

```
def OCL(s_I):
    OPEN ← {s_I}, CLOSED ← {},
    g(s_I) = 0, parent(s_I) = ∅
    while OPEN ≠ {}:
        p ← SelectNode(OPEN)
        if p is a goal, return path to p
        for c ∈ children(p):
            if c ∉ OPEN ∪ CLOSED:
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                OPEN ← OPEN ∪ {c}
            else if g(c) > g(p) + κ(p, c):
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                if c ∈ CLOSED:
                    OPEN ← OPEN ∪ {c}
                    CLOSED ← CLOSED − {c}
        OPEN ← OPEN − {p}, CLOSED ← CLOSED ∪ {p}
    return No solution exists
```

Node Reopening

def OCL(s_I):

$OPEN \leftarrow \{s_I\}, CLOSED \leftarrow \{\},$

$g(s_I) = 0, parent(s_I) = \emptyset$

while $OPEN \neq \{\}$**:**

p ← SelectNode($OPEN$)

if p is a goal, **return** path to p

for $c \in children(p)$:

if $c \notin OPEN \cup CLOSED$:

$g(c) = g(p) + \kappa(p,c)$

$parent(c) = p$

$OPEN \leftarrow OPEN \cup \{c\}$

else if $g(c) > g(p) + \kappa(p,c)$:

$g(c) = g(p) + \kappa(p,c)$

$parent(c) = p$

~~**if** $c \in CLOSED$:~~

~~$OPEN \leftarrow OPEN \cup \{c\}$~~

~~$CLOSED \leftarrow CLOSED - \{c\}$~~

$OPEN \leftarrow OPEN - \{p\}, CLOSED \leftarrow CLOSED \cup \{p\}$

return No solution exists

```
def OCL(s_I):
    OPEN ← {s_I}, CLOSED ← {},
    g(s_I) = 0, parent(s_I) = ∅
    while OPEN ≠ {}:
        p ← SelectNode(OPEN)
        if p is a goal, return path to p
        for c ∈ children(p):
            if c ∉ OPEN ∪ CLOSED:
                g(c) = g(p) + κ(p, c)
                parent(c) = p
                OPEN ← OPEN ∪ {c}
            else if g(c) > g(p) + κ(p, c) and
                    c ∈ OPEN:
                g(c) = g(p) + κ(p, c)
                parent(c) = p
        OPEN ← OPEN − {p}, CLOSED ← CLOSED ∪ {p}
    return No solution exists
```

Re-Expansions

- Comparing WA* and GBFS when you reopen nodes and when you do not
  - How does this impact performance?

Grid Pathfinding

- Pathfinding in a grid
- 4-connected means can move N, E, S, W
  - Every move costs 1
- 8-connected means can also move NE, SE, SW, NW
  - Diagonal moves cost square root 2
- Heuristics
  - Manhattan distance for 4-connected
  - Octile distance for 8-connected

Sliding Tile Puzzle

- Classic grid puzzle where you slide tiles
  - All slides cost 1
- Using Manhattan distance heuristic

A* Implementation

- Dijkstra's is $O(|V|\log|V| + |E|)$
  - Why?

A* Implementation

- NodeTable for OPEN and CLOSED list
  - Nodes are assigned a StateID
  - Hash table for Open-Closed list checking
- Priority Queue for OPEN list