Motion planning: Beyond Navmeshes

CMSC425.01 Spring 2019
Administrivia

• Exam being graded ...

• Project 2b concepts out, write up soon (add animations to 2a)
Today’s questions

Big question: Making intelligent agents
First question: Navigation
Finding paths in polygonal configuration space

• Version 1: Navmesh

• Others?

• Version 7: Randomized placement (sampling)
Finding paths in polygonal configuration space

• Version 1: Navmesh
• Others?

• Version 8: Rapidly-expanded Random Trees (RRTs)
Computing shortest path

• Reduce navigation to path finding in graphs
  • Directed?
  • Weighted?

• $G = (V, E)$
  • Vertices $V = \{ u, v, \ldots \}$
  • Edges $E = \{ (u, v), \ldots \}$
  • Weight function $w(u, v) \to \text{reals}$
Computing shortest path

- Reduce navigation to path finding in graphs
  - Directed?
  - Weighted?

- $G = (V, E)$
  - Vertices $V = \{ u, v, \ldots \}$
  - Edges $E = \{ (u, v), \ldots \}$
  - Weight function $w(u, v) \to \text{reals}$

- Path sequence of nodes
  - $P = \langle u_0, u_1, \ldots, u_k \rangle$

- Path cost
  - $\text{cost}(P) = \sum_{i=0}^{k} w(u_i, u_{i+1})$

- Lowest cost path $\partial(s, t)$
First: what's the problem?

• Compute one shortest path?  
• Compute all shortest paths to store?
First: what's the problem?

- Compute path here to there?

- Find fastest way to home base?
  - Reverse edges
  - Find shortest path to all from home

- Find closest facility (health, etc)?
  - Add Supernode connected to all facilities.

- Compute all shortest paths to store?
  - Floyd-Warshall

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>2</td>
<td>10</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
<td>13</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>D</td>
<td>12</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>7</td>
<td>3</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>
```
First: what's the problem?

• Find closest facility (health, etc)?
  • Add Supernode connected to all facilities.
Uninformed vs. informed search

- Uninformed – follow weights
  - Pick next node on distance to $d[u]$

- Informed – add bias towards destination

- Heuristic
  - Pick next node on distance to goal $h(u)$
Informed search

• Distance functions
  • \( w(u,v) \) - distance node \( u \) to \( v \)
  • \( d[u] \) - distance traversed from start to node \( u \)
  • \( \text{dist}(u,t) \) - distance from \( u \) to \( t \)

\[
\begin{align*}
\text{w}(s,1) &= \_\_\_ \quad \text{dist}(1,t) &= \_\_\_
\end{align*}
\]

\[
\begin{align*}
\text{w}(s,2) &= \_\_\_ \quad \text{dist}(1,t) &= \_\_\_
\end{align*}
\]
Informed search

• Distance functions
  • $w(u,v)$ - distance node $u$ to $v$
  • $d[u]$ - distance traversed from start to node $u$
  • $\text{dist}(u,t)$ - distance from $u$ to $t$

• $w(s,1) = 3$ \hspace{1cm} $\text{dist}(1,t) = 6$
• $w(s,2) = 3$ \hspace{1cm} $\text{dist}(1,t) = 4$

• $\text{dist}(u,t)$ is a heuristic
Less perfect information?

• Can't see rest of graph until you expand it
• Need guess on what's to come
• dist(u, t) as Euclidean distance
• Approximates actual cost
Footnote

• Euclidean distance
  • $\text{distE}(p1,p2) = \sqrt{(x1-x2)^2 + (y1-y2)^2}$

• Manhattan distance
  • $\text{distM}(p1,p2) = |x1-x2| + |y1-y2|$
Dijkstra’s Algorithm

Dijkstra(G, s, t) {
    foreach (node u) { // initialize
        d[u] = +infinity; mark u undiscovered
    }
    d[s] = 0; mark s discovered // distance to source is 0
    repeat forever { // go until finding t
        let u be the discovered node that minimizes d[u]
        if (u == t) return d[t] // arrived at the destination
        else {
            for (each unfinished node v adjacent to u) {
                d[v] = min(d[v], d[u] + w(u,v)) // update d[v]
                mark v discovered
            }
            mark u finished // we’re done with u
        }
    }
}
Example

• $w(u,v)$ as given

• Start with $d$ array as

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INF</td>
<td>INF</td>
<td>INF</td>
<td>INF</td>
<td>INF</td>
</tr>
</tbody>
</table>

• End with?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

• \( w(u,v) \) as given

• Start with \( d \) array as

\[
\begin{array}{cccccc}
a & b & c & d & e & z \\
0 & \text{INF} & \text{INF} & \text{INF} & \text{INF} & \text{INF}
\end{array}
\]

• End with?

\[
\begin{array}{cccccc}
a & b & c & d & e & z \\
0 & 3 & 4 & 7 & 5 & 14
\end{array}
\]
BestFirst(G, s, t) {
    foreach (node u) { // initialize
        d[u] = +infinity; mark u undiscovered
    }
    d[s] = 0; mark s discovered // distance to source is 0
    repeat forever { // go until finding t
        let u be the discovered node that minimizes dist(u,t)
        if (u == t) return d[t] // arrived at the destination
        else {
            for (each unfinished node v adjacent to u) {
                d[v] = min(d[v], d[u] + w(u,v)) // update d[v]
                mark v discovered
            }
            mark u finished // we’re done with u
        }
    }
}
Best first bad case ...

- Trapped in local minimum
A*

- Pick next node to expand based on sum of distance so far and heuristic

\[ f(u) = d[u] + h(u) = d[u] + \text{dist}(u, t) \]
A-Star(G, s, t) {
    foreach (node u) { // initialize
        d[u] = +infinity; mark u undiscovered
    }
    d[s] = 0; mark s discovered // distance to source is 0
repeat forever { // go until finding t
    let u be the discovered node that minimizes d[u] + dist(u,t)
    if (u == t) return d[t] // arrived at the destination
    else {
        for (each unfinished node v adjacent to u) {
            d[v] = min(d[v], d[u] + w(u,v)) // update d[v]
            mark v discovered
        }
        mark u finished // we’re done with u
    }
}
A* Example

- Manhattan distance
A* Search – Each entry is $d[u] : f(u)$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(u)$</td>
<td>15</td>
<td>13</td>
<td>15</td>
<td>17</td>
<td>12</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Init</td>
<td>0:15</td>
<td>∞:13</td>
<td>∞:15</td>
<td>∞:17</td>
<td>∞:12</td>
<td>∞:10</td>
<td>∞:9</td>
<td>∞:8</td>
<td>∞:5</td>
<td>∞:0</td>
</tr>
<tr>
<td>1: $s$</td>
<td>0</td>
<td>8:13</td>
<td>−</td>
<td>2:17</td>
<td>3:12</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>2: $d$</td>
<td>8:13</td>
<td>−</td>
<td>2:17</td>
<td>3</td>
<td>5:10</td>
<td>6:9</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>3: $e$</td>
<td>8:13</td>
<td>−</td>
<td>2:17</td>
<td>−</td>
<td>5</td>
<td>6:9</td>
<td>7:8</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>4: $f$</td>
<td>8:13</td>
<td>−</td>
<td>2:17</td>
<td>−</td>
<td>6</td>
<td>7:8</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>15:0</td>
</tr>
<tr>
<td>5: $t$</td>
<td>8:13</td>
<td>−</td>
<td>2:17</td>
<td>−</td>
<td>−</td>
<td>7:8</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>15</td>
</tr>
<tr>
<td>Final</td>
<td>0</td>
<td>8</td>
<td>∞</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>∞</td>
<td>15</td>
</tr>
</tbody>
</table>
Good heuristics

• For A* to compute correctly the heuristic $h(u)$ must be:

  • Admissible: $h(u)$ never overestimates the graph distance from node $u$ to goal $t$

  • Consistent: $h(u') \leq \delta(u',u'') + h(u'')$