The Motion Planning Problem

• A special case of the more general planning problem

• The goal is to develop techniques that would allow a robot or robots to automatically decide how to move from one position or configuration to another.
  • Specifically concerned with planning motions – get robot from place A to B
Motion Planning for Robotics
An Example – the PacMan problem

• How does the computer guide the ghosts back to their lair when they are eaten?
Planning on a grid

• In this example the robot can move between adjacent cells on the grid
• The dark squares indicate obstacles that the robot cannot traverse.
Graph Structure

• We can think of the unoccupied cells as nodes and draw edges between adjacent cells as shown here.

• This set of nodes and edges constitutes a graph.
• A graph, $G$, consists of a set of vertices, $V$, and a set of Edges, $E$, that link pairs of vertices.

• The edges are often annotated with numerical values to indicate relevant quantities like distances or costs.
Examples of Graphs in the Wild – Toll Chart

Chicago

Philadelphia

Pittsburgh

Richmond

New York

Washington

$20

$8

$11

$25

$4

$15

$20

$8

$13

$8
Graph Structure

• In this grid graph we will implicitly associate a cost or distance of 1 with every edge in the graph since they link adjacent cells.
Planning on a grid

• The goal is to construct a path through the grid/graph from the start to the goal
Planning on a grid

• Typically there are many possible paths between two nodes.
• We are usually interested in the shortest paths
Planning on a grid

- Goal:
  - Construct the shortest path between the start and the goal location.
Planning Procedure – Grassfire Algorithm

• Begin by marking the destination node with a distance value of 0
Planning Procedure – Grassfire Algorithm

- On every iteration find all the unmarked nodes adjacent to marked nodes and mark them with that distance value + 1.
## Planning Procedure – Grassfire Algorithm

![Grassfire Algorithm Diagram]

### START

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Grassfire Algorithm steps and progression indicated by numbers and color change.)
Planning Procedure – Grassfire Algorithm

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**START**

- Green square indicates the starting point.
- Black squares represent the spread of the grassfire.
- Red square marks the destination or goal.
- Numbers indicate the distance from the start to the current cell.
Planning Procedure – Grassfire Algorithm
Planning Procedure – Grassfire Algorithm

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **START**
Planning Procedure – Grassfire Algorithm

- On every iteration the marking radiates outward from the destination like a fire spreading – hence the name

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
Planning Procedure – Grassfire Algorithm

START

0

1

2

3

4

5

6

7

6

5

4

3

2

1

0

1

2

3

2

1

2
Planning Procedure – Grassfire Algorithm
Planning Procedure – Grassfire Algorithm

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>9</th>
<th>8</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>6</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

START
Planning Procedure – Grassfire Algorithm

START

9  10  9  8
8  9  8  7  6
7
6  5
5  4
3
2  1  0
3  2  1  2
Planning Procedure – Grassfire Algorithm

• The distance values produced by the grassfire algorithm indicate the smallest number of steps needed to move from each node to the goal.
Grassfire algorithm – pseudo code

• For each node n in the graph
  • n.distance = Infinity
• Create an empty list.
• goal.distance = 0, add goal to list.
• While list not empty
  • Let current = first node in list, remove current from list
  • For each node, n that is adjacent to current
    • If n.distance = Infinity
    • n.distance = current.distance + 1
    • add n to the back of the list
Tracing a path to the destination

- To move towards the destination from any node simply move towards the neighbor with the smallest distance value, breaking ties arbitrarily.
Another Example – Grassfire Algorithm
Another Example – Grassfire Algorithm
Planning Procedure – Grassfire Algorithm

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

START
Planning Procedure – Grassfire Algorithm
Planning Procedure – Grassfire Algorithm

• In this case the procedure terminates before the start node is marked indicating that no path exists
Grassfire Algorithm

• It will find the shortest path between the start and the goal if one exists.
• If no path exists that fact will be discovered.
Computational Complexity - Grassfire

- The computational effort required to run the grassfire algorithm on a grid increases linearly with the number of edges.

- This can be expressed more formally as follows.

\[ \mathcal{O}(|V|) \]  \hspace{1cm} (1)

Where \(|V|\) denotes the number of nodes in the graph
Computational Complexity - Grassfire

• Number of nodes in a 2D grid 100x100 = $10^4$
• Number of nodes in a 3D grid 100x100x100 = $10^6$
• Number of nodes in a 6D grid 100 cells on side = $10^{12}$
Video 9.3

CJ Taylor
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm

A, 0

B, 1

C

D, 2

E

F, 5

G
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm

A, 0
B, 1
C, 5
D, 2
E
F, 5
G, 10
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm
Planning shortest paths – Dijkstra’s Algorithm
Dijkstra’s algorithm – pseudo code

• For each node n in the graph
  • n.distance = Infinity

• Create an empty list.

• start.distance = 0, add start to list.

• While list not empty
  • Let current = node in the list with the smallest distance, remove current from list
  • For each node, n that is adjacent to current
    • If n.distance > current.distance + length of edge from n to current
    • n.distance = current.distance + length of edge from n to current
    • n.parent = current
    • add n to list if it isn’t there already
Planning shortest paths – Dijkstra’s Algorithm

A, 0 B, 1
C, 5
D, 2
E, 6
F, 5
G, 9
Computational Complexity of Dijkstra’s algorithm

- A naive version of Dijkstra’s algorithm can be implemented with a computational complexity that grows quadratically with the number of nodes.

\[ O(|V|^2) \] (1)

- By keeping the list of nodes sorted using a clever data structure known as a priority queue the computational complexity can be reduced to something that grows more slowly

\[ O((|E| + |V|) \log(|V|)) \] (2)

- \(|V|\) denotes the number of nodes in the graph and \(|E|\) denotes the number of edges
Video 9.4

CJ Taylor
A* Procedure

• Improving on Dijkstra/Grassfire using heuristic search
• Example of Best First search strategy
Dijkstra/Grassfire Algorithm

• When applied on a grid graph where all of the edges have the same length, Dijkstra’s algorithm and the grassfire procedure have similar behaviors.

• They both explore nodes in order based on their distance from the starting node until they encounter the goal.
A* Search

• A* Search attempts to improve upon the performance of grassfire and Dijkstra by incorporating a heuristic function that guides the path planner.
Heuristic Functions

• Heuristic functions are used to map every node in the graph to a non-negative value

• Heuristic Function Criteria:
  • $H(\text{goal}) = 0$
  • For any 2 adjacent nodes $x$ and $y$
    • $H(x) \leq H(y) + d(x,y)$
    • $d(x,y) =$ weight/length of edge from $x$ to $y$

• These properties ensure that for all nodes, $n$
  • $H(n) \leq \text{length of shortest path from } n \text{ to goal.}$
Example Heuristic Functions

- For path planning on a grid the following 2 heuristic functions are often used

  - Euclidean Distance

    \[ H(x_n, y_n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \] (1)

  - Manhattan Distance

    \[ H(x_n, y_n) = |(x_n - x_g)| + |(y_n - y_g)| \] (2)

  - where \((x_n, y_n)\) denotes the coordinates of the node \(n\) and \((x_g, y_g)\) denotes the coordinate of the goal
A* algorithm – pseudo code

• For each node n in the graph
  • n.f = Infinity, n.g = Infinity

• Create an empty list.

• start.g = 0, start.f = H(start) add start to list.

• While list not empty
  • Let current = node in the list with the smallest f value, remove current from list
  • If (current == goal node) report success
  • For each node, n that is adjacent to current
    • If (n.g > (current.g + cost of edge from n to current))
    • n.g = current.g + cost of edge from n to current
    • n.f = n.g + H(n)
    • n.parent = current
    • add n to list if it isn’t there already