Algoritmo de Dijkstra

Luis Servín
Taller TRA, Universidad Nacional Autónoma de México

June 20, 2016

Si se considera la situación mostrada en la figura 1 en la cual tenemos un objeto que va a desplazarse a través de su entorno. A continuación se ejemplifican dos situaciones que pueden presentarse:

- El objeto sigue una trayectoria directa hacia la meta. Ya que no existe indicio de que no deba seguir esta ruta debido a las lecturas del área que puede escanear (área roja) avanza con confianza. Cerca de la meta detecta un obstáculo y decide cambiar de dirección para evadirlo. Durante su movimiento descubre que se trata de un obstáculo en forma de "U", por lo cuál tuvo que realizar la trayectoria roja punteada mucho mas largo de lo previsto inicialmente.
- Previo a comenzar a moverse, el sistema obtiene un conocimiento del ambiente que lo rodea (área azul). Con base en esta información utiliza un algoritmo de planificación de ruta para encontrar un camino mas corto hacia la meta, y así evitar obstáculos en el ambiente.

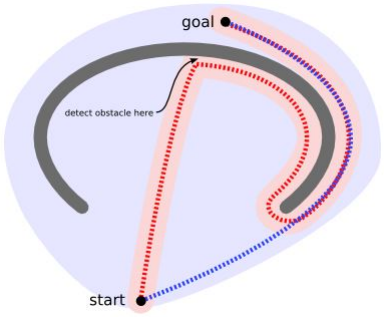

Figure 1: Ventaja de contar con una planeación de ruta previa

Gracias a lo descrito anteriormente podemos notar una de las principales ventajas de los algoritmos de planeación de ruta; Estos te permiten realizar un plan de las acciones a tomar con base en la información actual del ambiente, en lugar de esperar hasta el último momento para descubrir que existe un problema. Aunque la planeación anticipada suele ser mas lenta que los comportamientos reactivos llega a producir mejores resultados, si es que el ambiente no ha cambiado de manera drástica. Por lo tanto con el objetivo de aprovechar lo mejor de ambas propuestas, se recomienda utilizar ambos: Un planificador de ruta que tome en cuenta la información global en caminos largos; y un algoritmo reactivo para movimientos rápidos en situaciones no previstas en un inicio.

Estos algoritmos de planeación trabajan sobre gráficas, que se pueden definir como un conjunto de vértices con aristas que los conectan entre sí. Basándonos en esto podemos establecer una gráfica a partir de un mapa cuadriculado como el que se muestra en la figura 2. Este podemos definir a cada unidad o cuadro como un vértice y establecer una arista entre dos unidades continuas que comparten un lado. Con base en la información anterior podemos aplicar un algoritmo de planeación de ruta sobre un mapa cuadrículado como los el algoritmo de Dijkstra.

El algoritmo de Dijkstra resuelve el problema de encontrar la ruta mas corta de un punto dentro de la gráfica (la fuente) hacia un punto objetivo. Este algoritmo fue diseñado por el científico Edsger W. Dijkstra en 1956. Aunque en primera instancia el algoritmo puede encontrar el camino mas corto entre dos puntos dentro de la gráfica, una variante mas común consiste en establecer un punto como inicial y a partir del mismo encontrar la ruta mas corta a todos los puntos dentro de la gráfica.

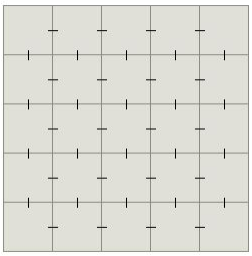

Figure 2: Representación de un mapa cuadriculado (grid-map)

EL algoritmo trabaja al visitando los vértices dentro de la gráfica comenzando en el punto inicial. Enseguida de manera repetitiva examina los vértices mas cercanos que no hayan sido visitados aún, añadiéndolos al conjunto de objetos que han sido visitados. De esta manera se expande desde el punto inicial hasta que encuentra el punto objetivo. Además tiene como característica que puede garantizar encontrar la ruta mas corta entre ambos puntos, siempre y cuando ninguna de las conexiones tenga un peso negativo.

Una vez explicado el proceso a seguir de manera sencilla, procedemos formalizar el proceso

Sea el nodo en el cual comenzamos conocido como **nodo inicial**. Ahora sea la **distancia del nodo Y** la distancia actual entre el nodo inicial y el nodo Y. Como primer paso se asignara un valor de distancia inicial, la cual se tratará de mejorar de manera iterativa.

1. Asignar a cada nodo un valor inicial de distancia. Establecer un valor de cero para nuestro nodo incial e infinito para todos los demás.
2. Establecer el nodo inicial como el actual. Los nodos restantes se establecerán como no visitados. Crear un conjunto que contenga todos los nodos no visitados conocido como *unvisited*.
3. Para el nodo actual, considerar todos los vecinos o nodos con un conexión directa que se encuentren dentro del conjunto *unvisited* y calcular su distancia tentativa. Compara la distancia tentativa obtenida con la que esta asignada hasta el momento, mantener el valor mas pequeño.
4. Una vez que se hayan calculado las distancia a todos los nodos vecinos, marcar el nodo actual como *visitado* y eliminarlo del conjunto *unvisited*. Un nodo que haya sido marcado como visitado no volverla a ser analizado de nuevo.
5. Si el nodo objetivo ha sido marcado como visitado (en caso que se busque una ruta entre dos nodos específicos) o si la distancia tentativa mas pequeña entre los nodos del conjunto *unvisited* es infinito (en caso de que no haya una conexión entre el nodo inicial y los nodos restantes) deternerse.
6. De lo contrario, seleccionar el nodo que tiene la distancia tentativa mas pequeña dentro del conjunto *unvisited*. Establecerlo como nodo inicial y regresar al paso 3

Una vez descrito el algoritmo, se establece un **pseudocódigo** que puede servir como base para implementar dicho algoritmo en un lenguaje de programación.

```
function DIJSKTRA(Graph, Source)
  create vertex set Q
  for each vertex v in Graph:                          ▷ Initialization
    dist[v] ← INFINITY                                 ▷ Unknown distance from source to v
    prev[v] ← UNDEFINED                                ▷ Previous node in optimal path from source
    add v to Q                                         ▷ All nodes initially in Q (unvisited nodes)
  dist[source] ← 0                                     ▷ Distance from source to source
  while Q is not empty:
    u ← vertex in Q with min dist[u]                   ▷ Source code will be selected first
    remove u from Q
    for each neigbor v of u:                           ▷ where v is still in Q
      alt ← dist[u] + length(u,v)
      if alt < dist[v]                                 ▷ A shorter path to v has been found
        dist[v] ← alt
        prev[v] ← u
  return dist[], prev[]
end function
```