SISTEMAS DE CONTROL

PARA ROBOTS

Universidad de Alcalá

Planificación Global

A*

Pedro Barquín Ayuso

Miguel Ballesteros García

Índice

Introducción

El algoritmo de búsqueda A* fue presentado por primera vez en 1968 por Peter E. Hart, Nils J. Nilsson y Bertram Raphael.

El algoritmo A* es un algoritmo completo, siempre encontrará la solución si es que esta existe. En resumen encuentra el camino de menor coste entre dos puntos conocidos, siempre que se cumplan las condiciones necesarias.

Es un algoritmo que puede ser empleado para calcular el camino optimo en un escenario concreto, en nuestro caso sería el mapa que recorrería nuestro robot. Se va a tratar de un algoritmo heurístico, pues una de sus principales características es que hará uso de una función de evaluación heurística, mediante la cual estudiara los diferentes nodos del mapa y determinara cuál de ellos pertenecerán al camino óptimo hacia nuestro destino.

Esta función de evaluación estará compuesta a su vez por otras dos funciones. Una de ellas indicará la distancia actual desde el nodo origen hasta el nodo a estudiar, y la otra expresará la distancia estimada desde este nodo a estudiar hasta el nodo destino hasta el que se pretende encontrar un camino óptimo.

Es muy usado en videojuegos, el primero en usarlo fue el Pac-Man (para la planificación de los fantasmas).

Algoritmo

El algoritmo A* pertenece a la familia de los algoritmos informados, frente a los desinformados o por fuerza bruta, que son aquellos que poseen una información extra sobre la estructura objeto de estudio, la cual explotan para alcanzar más rápidamente su objetivo final, siguiendo un camino óptimo como recorrido. En nuestro caso tendremos un mapa conocido además de los dos puntos de inicio y fin.

El algoritmo A* evalúa una función para clasificar nodos. Este algoritmo usa principalmente la función f (n) = g (n) + h (n) para guiarse y seleccionar los nodos abiertos para expandirse, donde:

-G(n) es el coste actual del nodo inicial al siguiente nodo.

-H(n) es una estimación del coste óptimo del nodo n hasta el nodo objetivo, dependiendo de la información de la heurística.

Si para todo nodo n del grafo se cumple g (n) = 0, nos encontramos ante una búsqueda voraz. El algoritmo voraz se guía exclusivamente por la función heurística lo que puede llevar a encontrar un camino que no sea el más corto hasta el punto de destino.

Si para todo nodo n del grafo se cumple h(n) = 0, A* pasa a ser una búsqueda de coste uniforme no informada, con lo que el algoritmo pasa a llamarse A y como en el caso anterior puede llevarnos a un camino que no sea el más corto hasta el punto de destino.

Proceso de búsqueda del algoritmo A*

1.- Calcular el valor H(n) para todo el mapa, conocida la posición de destino.

2.- Marcar el nodo inicial y expandir los nodos consecutivos sin marcar.

3.-Calcular la función de evaluación para cada nodo adyacente, clasificarlos de acuerdos valor de la función de coste e identificar y marcar el nodo con el mínimo valor de la función de coste. La búsqueda no termina hasta que el nodo actual es el nodo objetivo.

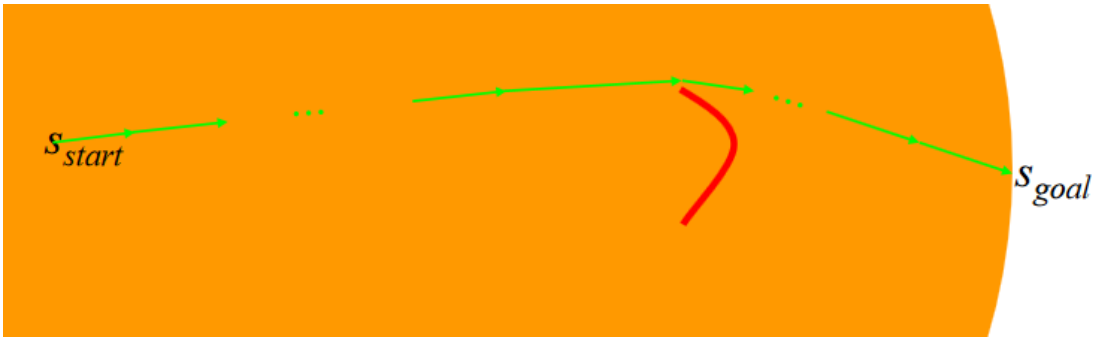
El paso más importante del algoritmo es elegir la función de coste. Esto ayudara a encontrar el camino óptimo si la función se ha elegido correctamente.

Mejoras del algoritmo A*

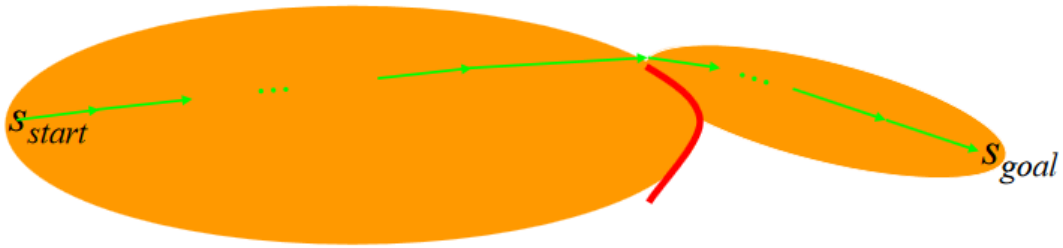
Weighted A*

Esta mejora afecta directamente en la función de coste. Para comprender dicha mejora la comparamos con la función del algoritmo A* y de Dijkstra.

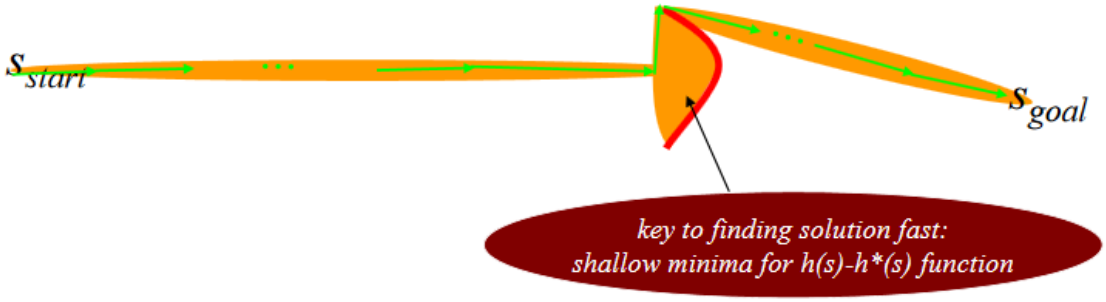
- El algoritmo de Dijkstra se expande según la función f (n) = g (n), tal como se ve en la imagen



- El algoritmo A* se expande según la función f (n) = g (n) + h (n) como hemos visto anteriormente.



El algoritmo Weighted A* se expande según la función f (n) = g (n) + ε h (n), con ε>1= dirección hacia nodos cercanos al destino.

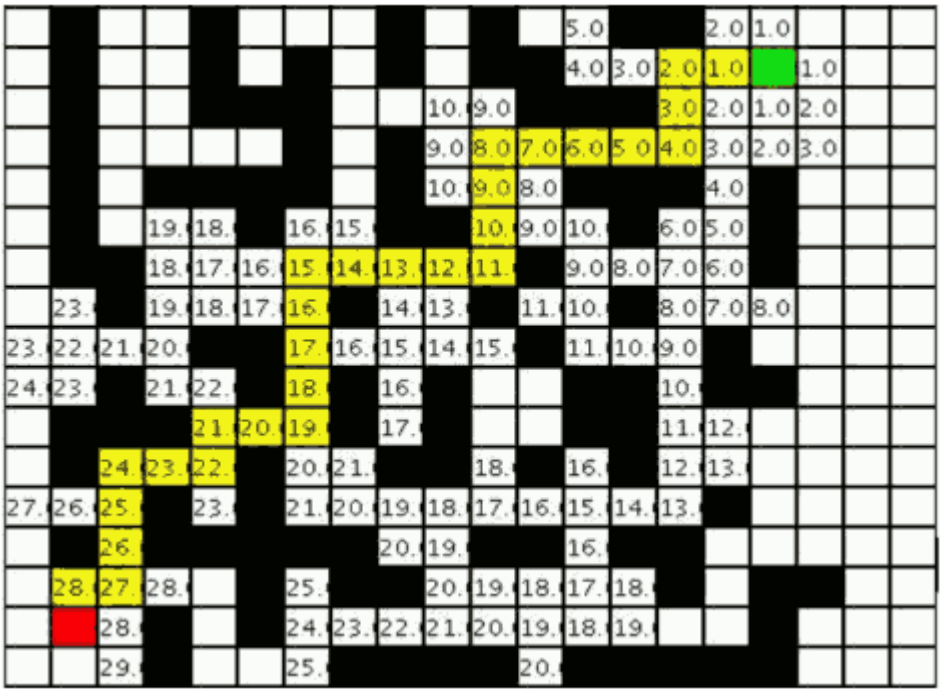


Como podemos ver en las imágenes anteriores existen varias diferencias con el algoritmo A*. Renuncia a encontrar la solución óptima por ser más rápido y se ha comprobado que en muchas ocasiones es más rápido que el algoritmo A*. Busca la mejora de la función heurística para reducir los mínimos locales.

Near-Optimal Hierarchical Pathfinding (HPA*)

Como hemos comentado al inicio del documento el Algoritmo A* ha sido utilizado en algunos juegos y dada la gran demanda en la industria del videojuego en estos últimos años muchas empresas han estado investigando nuevos algoritmos y mejorando algunos ya creados, dado que el movimiento de un personaje dentro de un videojuego es fundamental.

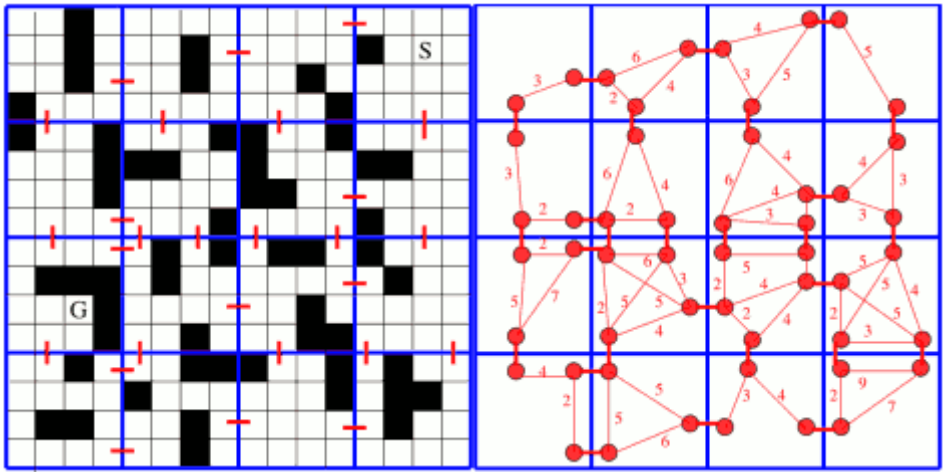
El algoritmo A* tradicional debe completar la búsqueda antes de devolver una ruta final, lo que ocupa varios marcos del juego y un tiempo sustancial en varios juegos teniendo en cuenta que la mayoría de juegos en la actualidad son online. Además, gran parte de la información calculada por A* en una búsqueda individual es descartada para no volver a ser usada, lo que supone una gran pérdida de procesamiento.



HPA * nos da un enfoque jerárquico para reducir la complejidad del problema en la búsqueda de rutas en mapas basados en la red. Esta técnica abstrae un mapa en clústeres locales vinculados. A nivel local, las distancias óptimas para cruzar cada grupo son precalculadas y almacenadas en caché. A nivel global, los arcos se atraviesan en un solo paso. Los pequeños grupos se agrupan para formar grupos más grandes, es la típica estrategia de divide y vencerás. El cálculo de las distancias de cruce para un grupo grande usa distancias calculadas para los clústeres contenidos.

Este método no depende de una topología específica y funciona bien en escenarios con un entorno dinámicamente cambiante. La técnica también tiene la ventaja de la simplicidad y es fácil de implementar. Si se desea, se puede refinar y dotar de mejoras para aumentar su rendimiento.

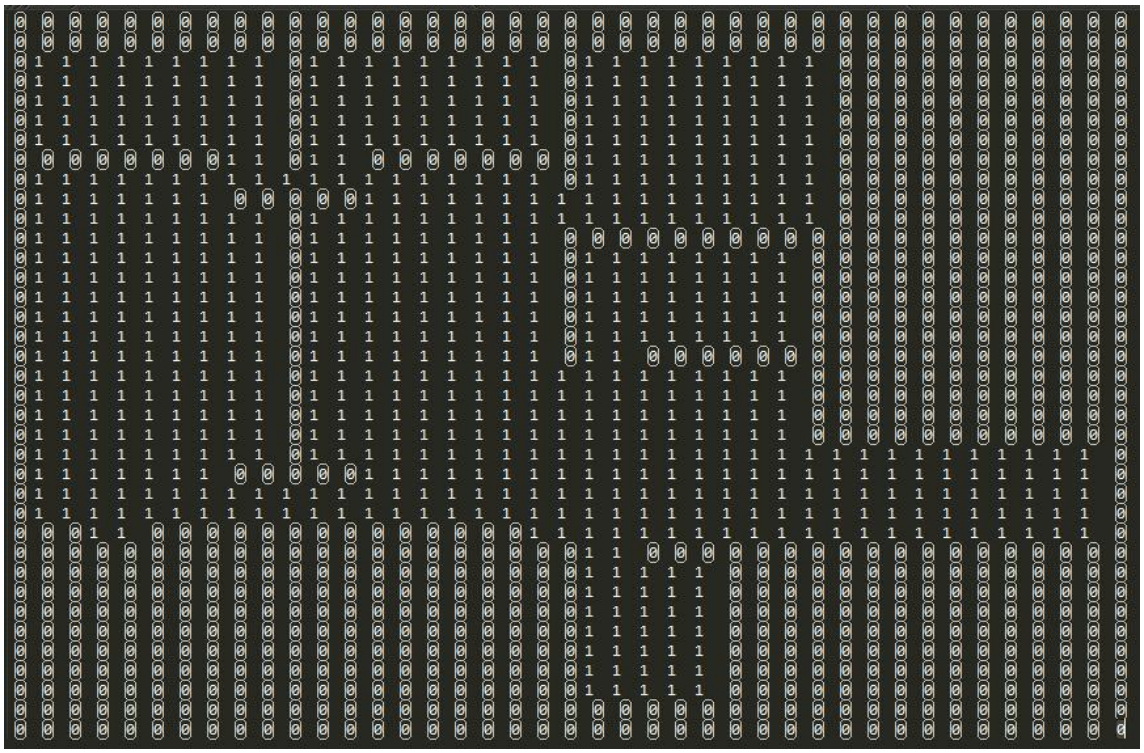
En comparación con un A* optimizado, HPA * se muestra hasta 10 veces más rápido, mientras que la búsqueda de caminos están dentro de 1% de lo óptimo.



Implementación

Para la implementación del algoritmo en playerstage, han sido necesarios unos pasos previos para su correcto funcionamiento.

- Convertir el mapa a una matriz de ocupación. Para ello se ha tomado el doble de la resolución, es decir, que entre el punto 1 y el 2 hay un punto intermedio. Además, para representar las zonas ocupadas se ha usado un valor de 1 y 0 para las libres tal como se ve en la imagen



Una vez que tenemos la matriz que representa la ocupación del mapa procedemos a pasar el seudo-codigo descrito arriba a C para que realice la búsqueda.

Lo primero que necesitamos es la estructura de la matriz que representa el mapa para ello hemos optado por utilizar los siguientes valores.

```
//Estructura
struct nodo{
    int x;
    int y;
    int ocupado;
    int peso;
    int g;
    int h;
    int visitado;
    int padre[1][2];
    int principio;
    int fin;
    int analizado;
};

//variables globales
struct nodo almacen[38][41];
```

También hemos implementado las funciones necesarias para que funcione el algoritmo A*

- Alrededor: esta función es la encargada de rellenar las celdas de alrededor con la información necesaria, como es el peso (resultado de g(n)+h(n)), el valor de g, que se calcula con el movimiento 10 ortogonal 14 diagonal, y la información del padre para poder regresar y trazar el camino.

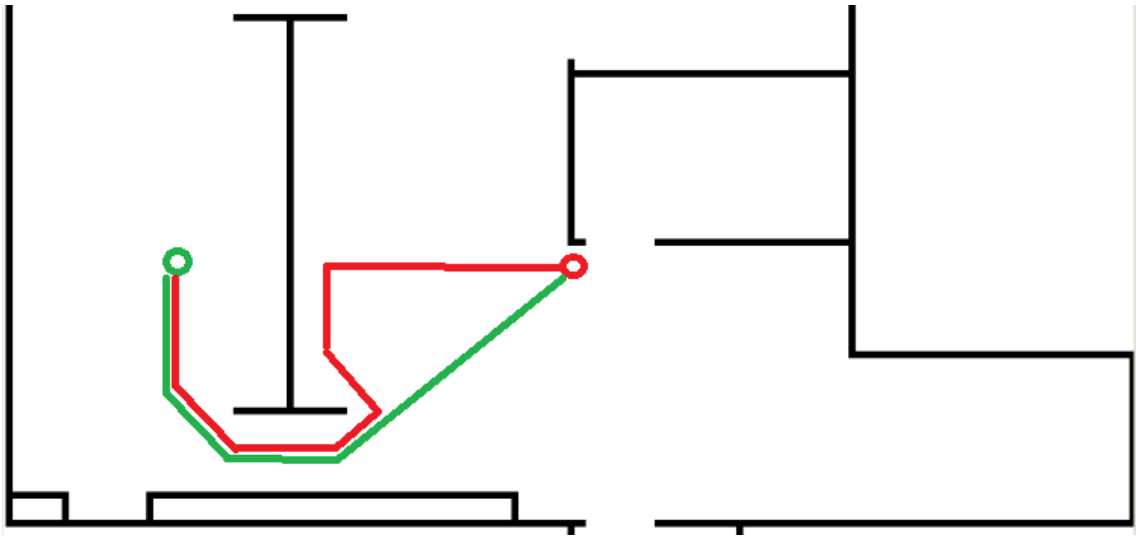
- Buscar: esta función busca el valor más pequeño de las celdas visitadas y que no estén utilizadas para la propagación para evitar que se atasque en un mínimo local.
- robotMapaX/Y: sirve para pasar la información de las coordenadas del robot a la referencia del mapa.
- mapaRobotX/Y: nos devuelve el valor de la referencia del mapa del algoritmo a las coordenadas para el robot.

Resumen de pasos a seguir:

- Inicializamos la estructura con los valores por defecto para todas las casillas.
- Pedimos por teclado la posición inicial del robot, la transformamos al mapa para operar con ella y realizamos los ajustes para esa casilla.
- Pedimos la posición de destino y realizamos la misma acción para la casilla final.
- Calculamos el valor de H para todas las casillas teniendo en cuenta el destino.
- Realizamos la aproximación “Alrededor” para la casilla de salida.
- Entramos en el bucle de búsqueda del camino, en el cual repetimos la misma acción hasta que la casilla actual sea el destino. La acción es buscar la casilla con menos valor y propagar alrededor de ella.
- Sacamos los nodos padres partiendo desde el final lo que nos da el recorrido optimo calculado,
- Invertimos dicho recorrido para poder pasárselo al robot y pasamos dichas coordenadas con “mapaRobotX/Y” para obtener los puntos del camino.
- Por último, realizamos el recorrido con el robot con ayuda del VHF para evitar las colisiones al pasar cerca de las esquinas.

Conclusión

Aunque el algoritmo busca el camino óptimo desde un punto inicial a un destino, tiene sus fallos al ser la implementación básica. En el caso de la imagen mostrada a continuación, el robot realiza el recorrido pintado en rojo, mientras que el recorrido optimo real seria el verde. Esto se debe a que la implementación básica busca el camino óptimo dentro de unos rangos, pero no tiene por qué ser siempre el mejor camino, como es el caso de la imagen.



En este segundo caso sí que coinciden el camino óptimo y el trazador por el algoritmo.