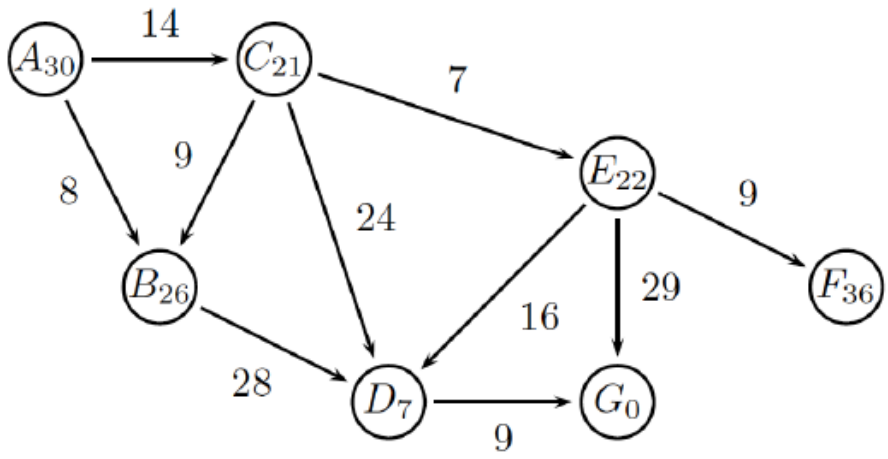
Tomé Maseda Dorado, Julián Barcia Facal

Ejercicio 1. Búsqueda informada.



Búsqueda Avara

Paso	*Frontera*	*Explorados*
1	A(30)	-
2	C(21),B(26)	A(0)
3	B(26),D(7),E(22)	A(0),C(14)
4	B(26),E(22),**G_0(0)**	A(0),C(14),D(38)
Solución: **A-C-D-G**		

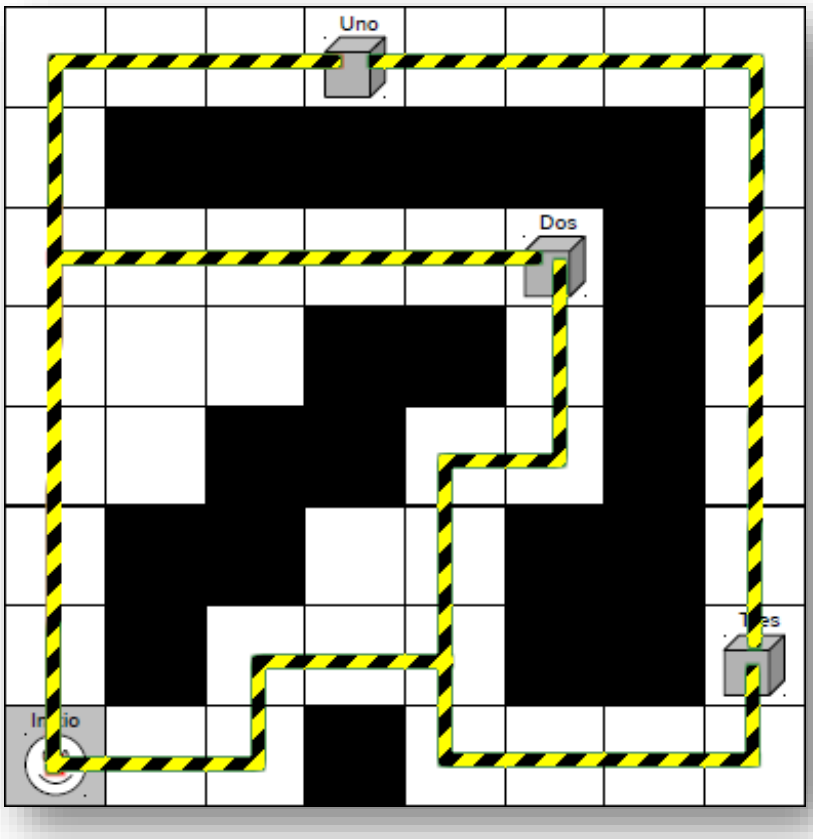
*Búsqueda A**

Paso	*Frontera*	*Explorados*
1	A(0+30)	-
2	B(8+26),C(14+21)	A(0)
3	C(14+21),D(36+7)	A(0),B(8)
4	D(36+7),E(21+22)	A(0),B(8),C(14)
5	E(21+22),G_0(0+45)	A(0),B(8),C(14),D(36)
6	**G_0(0+45)**,F(30+36)	A(0),B(8),C(14),D(36),E(21)
7	F(30+36)	A(0),B(8),C(14),D(36),E(21),**G_0(45)**
Solución: **A-B-D-G**		

Comparación de resultados

Tal y como reflejan las tablas, podemos ver que la búsqueda avara es un algoritmo más eficiente (obtiene una solución en menos pasos), sin embargo, A* obtiene la mejor solución (aunque no tiene por qué ser siempre así), es decir, es un algoritmo óptimo mientras que la búsqueda avara no. Con una heurística consistente, si queremos encontrar la mejor solución, deberíamos usar A*, en cambio, si el tamaño del problema es muy grande usaremos búsqueda avara por razones de eficiencia (A* se queda sin espacio antes de encontrar la solución para tamaños muy grandes).

Ejercicio 2. Búsqueda con Graph Search.



1. Estados

Representación del problema

- 0 = Sede
- 1 = Localidad del paquete 1
- 2 = Localidad del paquete 2
- 3 = Localidad del paquete 3
- P1 = Paquete 1
- P2 = Paquete 2
- P3 = Paquete 3
- X = Localidad actual del robot
- C = Conjunto de paquetes recogidos
- [C] = Nº de paquetes del conjunto C
- (X, C) = Estado actual del problema
- Número "n" = Acción ejecutada

Conjunto de estados iniciales

I = (0, {})

Conjunto de metas o soluciones aceptables

M = {(0, {P1, P2}), (0, {P1, P3}), (0, {P2, P3})}

Conjunto de operaciones permitidas

O = {op1, op2, op3, op4, op5, op6, op7}

2. Acciones

Número de orden	Precondiciones	Acción
1	[C] < 2 y P1$\notin$C	Coger P1
2	[C] < 2 y P2$\notin$C	Coger P2
3	[C]< 2 y P3$\notin$C	Coger P3
4	P1$\in$C	Dejar P1
5	P2$\in$C	Dejar P2
6	P3$\in$C	Dejar P3
7	[C] = 2	Dejar 2 paquetes

3. Heurística

min(p) ---------> Peso mínimo*1

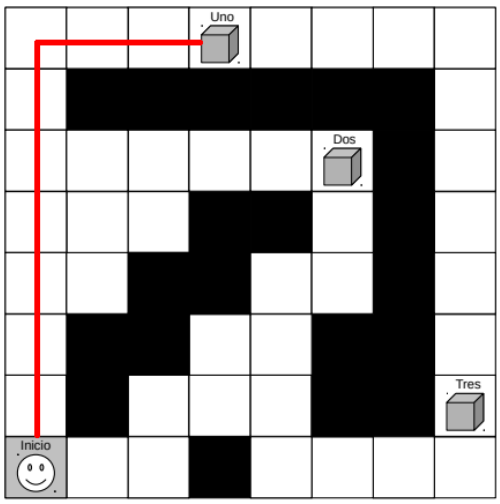
∑ min(d(n, i)) ---------> Sumatorio de todas las distancias mínimas *2

*1: El peso mínimo es el menor peso que podemos recoger de todas las localidades en las que queden paquetes o la sede (si no estamos en ella), en la sede el peso será cero ya que no hay paquetes a recoger en ella.

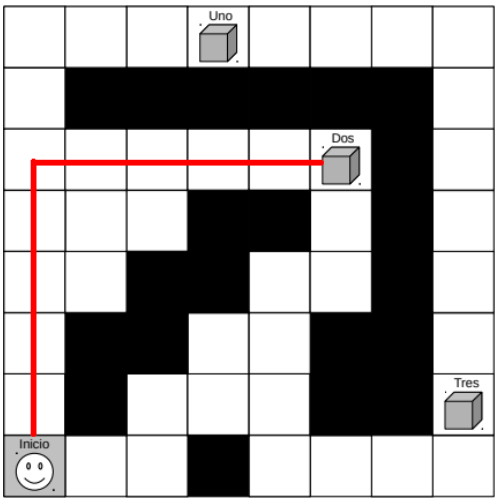
*2: Entendemos como distancia mínima la distancia en casillas de una localidad a otra (incluida la sede) sin tener en cuenta los obstáculos, es decir, el robot puede atravesar las casillas negras. d(n, i) es la distancia mínima desde el nodo n al nodo i, siendo i cualquier otro nodo del grafo en el que queden paquetes o la sede (si no está en ella).

$h(n) = \sum \min(d(n, i)) + \min(p)$

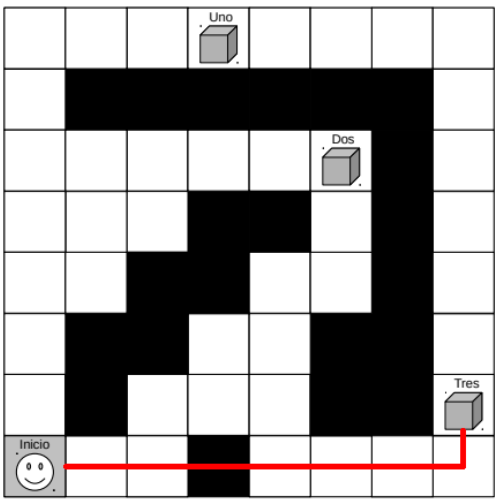
h(n) es una buena heurística porque representa (apróximadamente) la situación real del problema, cuando ya hemos recogido un paquete y volvemos a la sede a dejar dicho paquete (alejándonos de la meta), los nodos que representan dicho estado tienen una heurística mayor que los nodos que representan el estado en el que, tras la misma situación, vamos a por el segundo paquete (movimiento más óptimo ya que estamos acercándonos a la meta). Es una heurística admisible ya que NUNCA sobreestima el coste real y además es una heurística consistente, porque cumple que para cada nodo n el coste estimado de alcanzar la meta desde este (h(n)) no es mayor que la suma entre el coste de llegar a n' (sucesor de n) y el coste estimado de alcanzar la meta desde n' (h(n')).



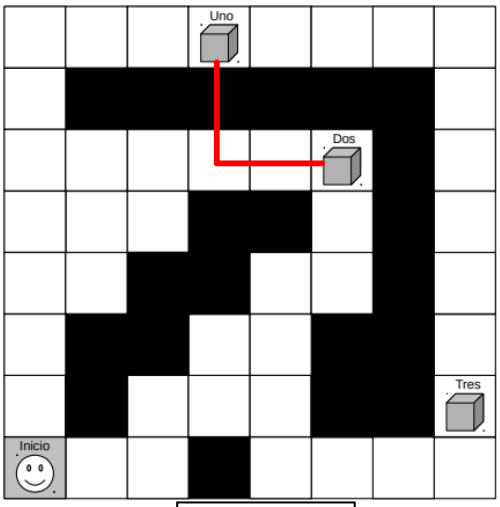
d(0,1) = 10



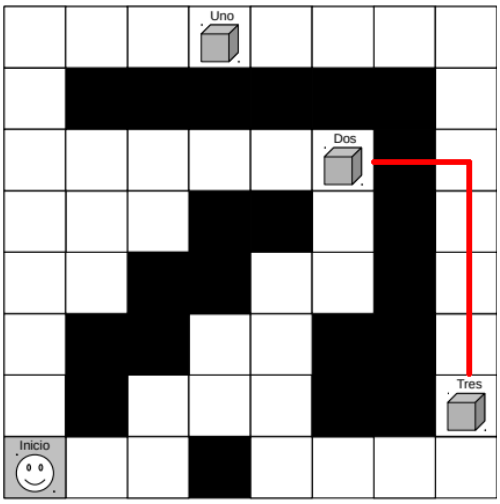
d(0,2) = 10



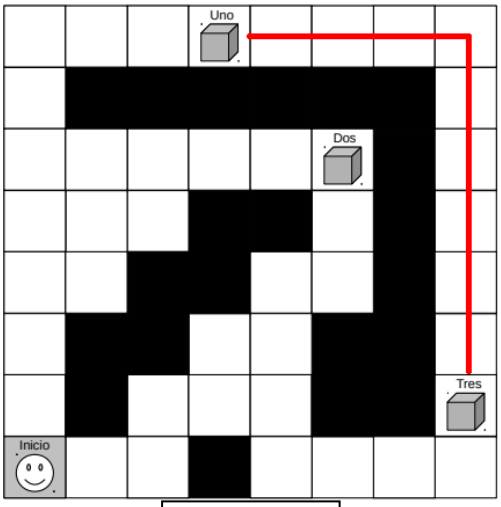
d(0,3) = 8



d(1,2) = 4

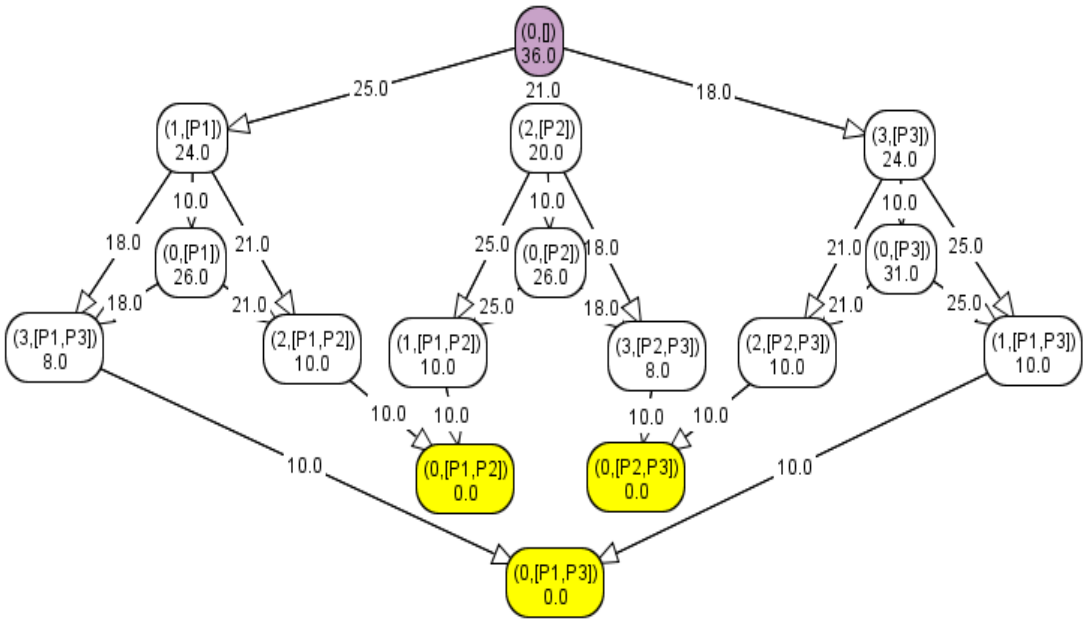


d(2,3) = 6



d(1,3) = 10

4. Grafo



5. Resultados

Algoritmo de búsqueda	*Camino obtenido*	*Coste del camino*	*Nº de nodos expandidos*
Depth First	(0,[]) --> (1,[P1]) --> (0,[P1]) --> (2,[P1,P2]) --> (0,[P1,P2]) (Goal)	66	5
Breadth First	(0,[]) --> (1,[P1]) --> (2,[P1,P2]) --> (0,[P1,P2]) (Goal)	56	16
Lowest Cost First	(0,[]) --> (3,[P3]) --> (2,[P2,P3]) --> (0,[P2,P3]) (Goal)	49	16
Best First	(0,[]) --> (2,[P2]) --> (3,[P2,P3]) --> (0,[P2,P3]) (Goal)	49	4
A*	(0,[]) --> (2,[P2]) --> (3,[P2,P3]) --> (0,[P2,P3]) (Goal)	49	7

*: (En caso de empate se ha escogido por orden alfabético)

6. Conclusiones y discusión

Si observamos los resultados obtenidos en la anterior tabla comprobamos que la mejor solución corresponde con la del algoritmo Best First. Esto se debe a que, aunque comparte el mismo coste de camino con A* y Lowest Cost First, sus nodos expandidos se reducen simplemente a 4 decrementando la parte del grafo que se debe recorrer para obtener la solución. Los tres son óptimos (en este caso) pero Best First gana en eficiencia.

En el caso del algoritmo Depth First el algoritmo no resulta óptimo ya que se regresa a devolver el paquete antes de recoger dos lo que resulta en la meta, lo cual incrementa considerablemente el coste del camino.

Tanto el algoritmo Depth First como Breadth First ignoran la heurística calculada de cada nodo y solo recorren el grafo en profundidad y anchura respectivamente (esto incrementa el coste de la solución).

El algoritmo de Lowest Cost First calcula el camino usando el coste de los vértices , eligiendo el mínimo en cada caso, por tanto, siempre obtendrá la mejor solución, pero no sabemos en cuantos pasos (es óptimo pero no eficiente).

El caso Best-First busca explorar el grafo priorizando el nodo más prometedor el cual se calcula gracias a la heurística.

Por último, A* explora el grafo priorizando el nodo más prometedor, pero a su vez de menor coste.

En cuanto a variaciones, a cada algoritmo le afectarán diferentes variaciones. Depth First y Breadth First cambiarán su comportamiento si variamos el orden de los nodos. Sin embargo, Best First y A* son mucho más afectados por los costes de los caminos (solo A*) y la función heurística (a pesar de que también son afectados por el orden de los nodos). Por último, Lowest Cost First es afectado mayoritariamente por el coste de los caminos también, aunque estos cambios se verán reflejados en la eficiencia del algoritmo y no en su solución.