Planificación Automática: Heurísticas y PDDL

Ignacio Navarro Blázquez
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
ignnavbla@alum.us.es

Fernando Rabasco Ledesma
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
ferrabled@alum.us.es

***Resumen*—El objetivo principal de este estudio inspirado en la heurística, persigue el poder estimar la distancia entre un estado y un objetivo expresados de acuerdo al formalismo PDDL. Además de implementar una nueva función para poder resolver los problemas de planificación heurística y comprobar su eficacia frente a la heurística estudiada en clase tras diversas puestas a prueba y análisis de los resultados.**

La planificación es una solución que se aplica a problemas cotidianos, en los que necesitamos conocer la mejor secuencia de acciones para llegar a la solución que tenemos cómo objetivo, siempre contrastando los resultados previamente obtenidos, comprobando que satisfacen todas nuestras necesidades.

***Palabras clave*—Inteligencia Artificial, Heurística, PDDL,Planning Dommain Definition Language, Planificación Automática, Estado, Objetivo, Acción, Precondición, Efecto, Búsqueda.**

I. Introducción

La Inteligencia Artificial (IA) ha buscado desde sus inicios desarrollar sistemas que resuelvan problemas automáticamente, contraponiéndose a la informática tradicional, que desarrolla programas específicos para los problemas. La planificación automática (PA) crea sistemas que resuelvan automáticamente los problemas con el objetivo de hallar las posibles acciones ejecutables en un entorno. Esto permite que a partir de un conjunto inicial de acciones, itere hasta un nuevo estado que satisfaga el objetivo o metas del problema.

La planificación heurística es uno de los paradigmas más exitosos en la última década. Estos problemas se suelen resolver como problemas de búsqueda, usando algoritmos guiados por la heurística. El éxito de este tipo de planificación se basa en la independencia del dominio, ya que permite solucionar problemas de dominios muy variables, cosa que antes no era posible con otros tipos de planificación. Sin embargo, este tipo de independencia deriva en un alto coste computacional, lo que dificulta la resolución de problemas "grandes".

Otro problema en ciertos dominios, es la aportación de la heurística al proceso de búsqueda, lo que conlleva no encontrar una solución en un tiempo razonable, puesto que la complejidad del algoritmo aumenta.

En este trabajo, nos centraremos en la creación de un modelo de planificación automática mediante heurística que pueda resolver problemas planteados con lenguaje PDDL de diferentes ámbitos y dominios.

También crearemos un nuevo modelo heurístico "prego" que nos devuelva la lista más corta con acciones y literales que se realicen en un problema, que es una variación de la heurística delta vista en clase, concretamente será la longitud de la lista obtenida tras aplicar el algoritmo modificado.

La solución propuesta será obtenida mediante diferentes técnicas y filtrados. Comenzaremos filtrando los datos obtenidos mediante PDDL para poder tratarlos de una manera más sencilla. Tras ello, podremos aplicar el método de la heurística delta implementado mediante bucles y llamadas recursivas a la función o el método implementado "prego" de una manera similar a la anterior, pero con una lista por los estados que pasa. Aplicando un algoritmo de búsqueda, en este caso de profundidad podremos llegar a la soluccción concreta del problema. Para complementar a los datos anteriormente obtenidos, calculamos el tiempo de ejecución de la heurística y de la solución, para poder hacer un análisis de los resultados obtenidos.

Gracias a la interfaz gráfica, es posible elegir que problema se desea ejecutar, además de poder ver los datos. También es posible elegir la heurística a aplicar, a elegir entre Delta o prego. Lo que resulta en poder ver los datos del problema y la solución de una manera más visual y amigable al usuario. La interfaz gráfica es opcional y también podemos ejecutar el problema mediante la consola, lo que resulta en más información intermedia entre el inicio y el fin del problema.

El documento presente está estructurado de la siguiente forma:

- Introducción
- Preliminares
  - Heurística
  - PDDL
  - Búsqueda en profundidad
  - Búsqueda hacia adelante en profundidad
  - Búsqueda hacia atrás
  - Tkinter
  - Trabajos relacionados
- Metodología
- Problemas encontrados
- Resultados
- Conclusiones
- Referencias

II. PRELIMINARES

En este trabajo, hemos usado PDDL para definir los problemas del mundo real a ordenador y a partir de ahí aplicar nuevos métodos, como ambas heurísticas, tanto le heurística delta para hallar las distancias entre estados, como la modificación aplicada en el sistema. A través de una búsqueda hacia adelante por profundidad, obtenemos la solución al problema, todo ello integrado en una interfaz gráfica gracias a Tkinter.

Aunque la información sobre este tema es escasa, hemos podido encontrar algunas tesis nombradas correctamente en la subsección de trabajos relacionados que nos han ayudado a avanzar.

A. Heurística

Algoritmo usado para ordenar los objetivos o estados. Estima la distancia entre estados y objetivos (número de acciones necesarias). Puede aplicarse mediante búsqueda hacia delante o hacia atrás. El objetivo es encontrar heurísticas independientes del dominio, basadas en la representación de los estados, objetivos y acciones. Para ello se suelen "relajar" algunos aspectos del problema, que consiste en desechar algunas restricciones como ignorar precondiciones y efectos negativos, suponer los literales como independientes o ignorar las precondiciones.

B. PDDL

"Planning Domain Definition Language", es un estándar para la representación de tareas de planificación, usado desde 1998. Está compuesto por:

- Objetos: Constantes que se usarán en el problema.
- Predicados: Estados que pueden tomar las constantes del problema.
- Estado inicial: Un estado es el conjunto de átomos cerrados. En el caso del inicial, como indica su nombre, es el punto de partida para llegar al objetivo propuesto.
- Objetivos: Los objetivos son la descripción de los estados finales. Se representan como una conjunción de literales, no tienen porque estar cerrados, puesto que no son estados concretos. Para que un estado satisfaga un objetivo se comprueba sustituyendo las variables objetivos por constantes.
- Acciones: están compuestas por precondiciones para realizarse y efectos que generan, que pueden ser positivos o negativos. Una acción será aplicable si se satisfacen sus precondiciones.

C. Búsqueda en profundidad

Es un algoritmo de búsqueda no informada utilizado para recorrer todos los nodos de un árbol de manera ordenada pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa por Backtracking, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

D. Búsqueda hacia adelante en profundidad

Es un algoritmo tipo Backtracking que ordena por la heurística los sucesores del estado actual. No garantiza la mejor solución, pero siempre termina y está correcto y completo. La complejidad dependerá de la máxima profundidad del árbol de búsqueda y la eficiencia dependerá de la bondad de la heurística.

E. Búsqueda hacia atrás

Se trata de empezar por el objetivo, aplicando las acciones a la inversa e intentar llegar al estado inicial, aplicando así solo las acciones relevantes para el objetivo.

F. Tkinter

Es un binding de la biblioteca gráfica Tcl/Tk para Python. Se considera un estándar para interfaces gráficas de usuario para Python y viene por defecto en Windows.

G. Trabajo Relacionado

Aunque la información sobre este tema es escasa, hemos podido basarnos y obtener ideas de varios de ellos, como pueden ser "Razonamiento basado en casos aplicado a la planificación heurística" de Tomás de la Rosa Turbides, "PDDL" por Fernando Berzal, la web de Drew V. McDermott, y Wikipedia en casos concretos de definiciones, entre otros.

Sobre librerías usadas, hemos usado la facilitada por la asignatura para las prácticas, Tkinter de Tcl/Tk, time y sys de Python.

III. METODOLOGÍA

- Si p aparece en e, $\Delta_0(e,p) = 0$
- Si p no aparece en e ni en los efectos positivos de ninguna acción, $\Delta_0(e,p) = +\infty$
- En otro caso, para cada acción A que tenga a p entre sus efectos positivos, calculamos $1 + \sum_{q \in precond^+(A)} \Delta_0(e,q)$ y definimos:

$$\Delta_0(e,p) = min_A\{1 + \sum_{q \in precond^+(A)} \Delta_0(e,q) | p \in efectos^+(A)\}$$

Fig. 1. Algoritmo para el cálculo de la heurística delta

Para tomar los datos de los problemas, primero hemos de inicializarlos como problema de planificación. Para ello usamos la librería proporcionada por la clase práctica de la siguiente forma. Ver figura 2.

Declaración de problema()
Entrada: Vacía
Salida: estados iniciales, operadores y el problema

```
 1  Creamos las listas que conforman los objetos;
 2  inicializamos los predicados con sus correspondientes
 3  variables, con probpl.Predicado();
 4  inicializamos el estado inicial con sus
 5  estados mediante probpl.Estado();
 6  definimos la lista de acciones:
 7  para cada acción:
 8        probpl.AcciónPlanificación( atributos)
 9        detro de la lista de atributos podemos poner:
10        Nombre;
11        Precondiciones (positivas y negativas);
12        Efectos(Positivos y negativos);
13        etc
14  Devolvemos el conjunto de las acciones;
15  Definimos el problema
16  Con probpl.ProblemaPlanificación()
17  asignando sus operadores o acciones,
18  el estado inicial y el objetivo;
19  Devolvemos las acciones, el estado inicial
20  y el problema.
```

Fig. 2. Pseudocódigo para la declaración del problema

Parseo de efectos(E)
Entrada: lista con los efectos E
Salida: lista con los efectos parseados EP

```
 1  Creamos la lista de efectos parseados EP;
 2  para cada e en E:
 3        si e no es una lista:
 4              lo convierto en una lista de un elemento;
 5        si no:
 6              obtengo la lista de elementos;
 7        para cada elemento de la nueva lista:
 8              reemplazamos las comillas;
 9              si tiene alguna coma antes de un parentesis
10                    la eliminamos;
11                    añadimos todo a la nueva lista;
12              devolvemos la lista.
```

Fig. 3. Pseudocódigo función de parseo

Para tratar el problema, hemos tenido que tratar los datos que nos proporcionaban la librería de PDDL del problema de planificación, y según si eran estados, objetivos o efectos.

Como todas las funciones para parsear datos son similares, vamos a representar mediante pseudocódigo el parseo de efectos. Ver figura 3.

También se ha realizado una interfaz gráfica mediante tkinter, gracias al cual, elegir problema a solucionar, la heurística por la que se soluciona y si mostrar los datos o no. Se puede apreciar en la Figura 6.

Heurística(SP, OP, EP, PA)
Entrada: una lista de estados SP, una lista de objetivos OP, una lista de efectos positivos EP, una lista de precondiciones PA)
Salida: un número entero SUM

```
 1  inicializamos una lista F ;
 2  para cada objetivo p en OP:
 3        si p está en SP entonces:
 4              añadimos un 0 a F ;
 5        si p no está en SP ni en EP entonces:
 6              añadimos infinito a F;
 7        si no entonces:
 8              inicializamos una lista V ;
 9              para cada objetivo p:
10                    obtenemos la lista de las precondiciones;
11                    para cada precondición pr:
12                          llamamos recursivamente a la función
13                          cambiando OP por las precondiciones
14                          de la acción.
15                          añadimos el valor a V
16                          sumándole 1;
17                    Añadimos el mínimo de V a F;
18  Sumamos los valores de F y lo devolvemos.
```

Fig. 4. Algoritmo de cálculo de Heurística por Delta 0

hprego($SP, OP, EP, PA, prego$)
Entrada: una lista de estados SP, una lista de objetivos OP, una lista de efectos positivos EP, una lista de precondiciones PA, una lista $prego$
Salida: un número entero SUM y una lista prego

```
 1  para cada objetivo p en OP:
 2        si p está en SP entonces:
 3              devolvemos prego lista vacía;
 4        si p no está en SP ni en EP entonces:
 5              añadimos a prego las acciones del problema;
 6        si no entonces:
 7              para cada objetivo p:
 8                    obtenemos la lista de las precondiciones;
 9                    para cada precondición pr:
10                          añadimos la acción a prego;
11                          llamamos recursivamente a la función;
12                    si prego tiene listas dentro:
13                          comparamos las longitudes de
14                          las listas y cogemos la menor;
15                    si no:
16                          prego es un solo elemento;
17                    imprimimos prego por pantalla;
18                    SUM será igual a la longitud de prego.
```

Fig. 5. Algoritmo de Heurística con prego(e,p)

Fig. 6. Interfaz gráfica del programa

IV. Problemas encontrados

A. Investigación

El primer inconveniente con el que nos enfrentamos al iniciar el estudio del problema, fue la falta de documentación sobre trabajos relacionados por internet en los que basarnos. Encontramos varias tesis sobre heurística o PDDL pero nada más allá de pura teoría. También pudimos encontrar algunas librerías de python sobre PDDL para realizar los problemas, pero tras valorar nuestras opciones, decidimos usar la enseñada en las prácticas de la asignatura, puesto que las otras no satisfacían del todo nuestras necesidades y optamos por evitar posibles problemas.

B. Tratamiento de la librería

El segundo gran inconveniente ha sido el tratamiento del contenido de los datos al definir un probelma PDDL. Debido a la librería escogida, los datos se devuelven de una manera difícil de seguir tratando con ellos para avanzar en el problema, por lo que se han tenido que crear varias funciones de parseo para poder usar los datos en las siguientes funciones.

C. Problema de bloques

Otro inconveniente encontrado nos ha supuesto todo un problema puesto que con el problema de bloques nos quedábamos sin memoria y alcanzaba un punto de máxima profundidad posible, llegando hasta 2300 bucles. Tras investigar sobre ello, se ha supuesto que se debía al problema en sí, ya que lo intentamos con diversos IDEs y hasta la consola. Por ello se decidió continuar con otros problemas que tuvieran menos requisitos y funcionaban sin ningún problema visible.

D. Complicaciones Sanitarias

Un gran inconveniente en el desarrollo del proyecto, ha sido las complicaciones sanitarias que ha sufrido un miembro del grupo tras la vacuna de la COVID-19. Debido a esto, ha estado varios días con fiebre y una conjuntivitis que ha durado hasta el final de la entrega, que le ha impedido en gran manera trabajar en ella.

E. Formato LaTex

Este formato para generar documentos, es algo complejo puesto que es programar dentro de un documento. El problema reside cuando no sabemos si por la web que hemos usado o por el formato en sí, no se respeta el orden en el que están escritas las cosas. Un ejemplo de este problema es que al escribir el pseudocódigo, si la figura era demasiado grande, saltaba a la página siguiente, y el apartado que la seguía, en este caso, los problemas encontrados, pasaban a estar en la página anterior, mezclando así los contenidos y desordenándolos. Hemos conseguido arreglarlo mediante el parámetro 'h' del elemento 'figure', que sitúa la imagen aproximadamente donde se encuentra posicionada en el código, de esta manera, se sitúa entre los dos textos que corresponden, aunque a veces no funcione correctamente.

V. Resultados

Los resultados que se han obtenido han sido mayoritariamente satisfactorios.

A. Problema rueda pinchada

Empezaremos con un ejercicio de la práctica sobre PDDL realizada en clase.

El problema de la rueda pinchada consiste en determinar los pasos a realizar para cambiar una rueda pinchada por una rueda de repuesto que se encuentra en el maletero, guardando finalmente la rueda pinchada en el maletero, para poder continuar el viaje. Las acciones que tenemos son:

TABLA I
Tabla acciones problema de rueda pinchada

Acción	Precondiciones	Efectos
Sacar	rep-maletero	¬rep-maletero, rep-suelo
Quitar	pin-eje	¬pin-eje, pin-suelo
Poner	rep-suelo, ¬pin-eje	rep-eje, ¬rep-suelo
Guardar	pin-suelo, ¬rep-maletero	¬pin-suelo, pin-maletero

Nuestro objetivo es cambiar la rueda pinchada y guardarla en el maletero.

El estado inicial es: pin-eje, rep-maletero.

Tras realizar la heurística Delta, el resultado obtenido es el apreciado en la Figura 7.

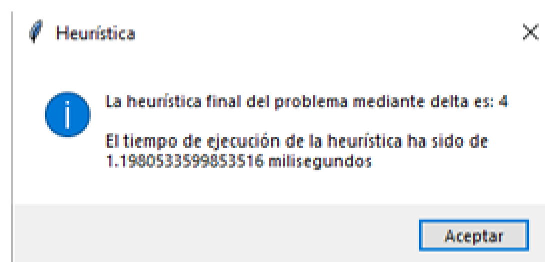

Fig. 7. Mensaje impreso por el programa con delta y el tiempo

Podemos comprobar que es correcto habiendo calculado la heurística manualmente con anterioridad, las 4 acciones a realizar serían:

1) *Sacar rueda de repuesto del maletero*
2) *Quitar rueda pinchada del eje*
3) *Poner rueda de repuesto en eje*
4) *Guardar rueda pinchada en el maletero*

Hemos comprobado, cambiando los datos del problema, que si no existen objetivos o manera de llegar al siguiente estado, el programa nos devuelve un numero muy grande (9999) , que representa al infinito, afirmando que funciona correctamente el código.

Para la heurísitca prego el resultado es el mismo que para la heurística anterior, 4, y además podemos comprobar que efectivamente las acciones de la lista son las acciones a realizar para llegar al estado final objetivo.

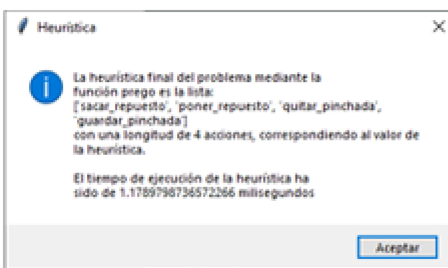

Fig. 8. Mensaje impreso por el programa con prego y el tiempo

Por último tras aplicar búsqueda hacia delante con profundidad, obtenemos lo representado en Figura 9.

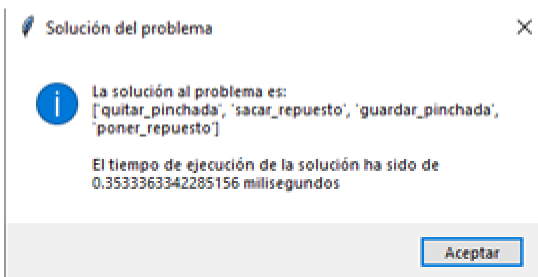

Fig. 9. Mensaje impreso por el programa con búsqueda y el tiempo

Sin duda, podemos observar que realizar la búsqueda es más eficiente, puesto que sus tiempos son menores a tiempos de ambas heurísticas, delta y prego, probablemente debido a la cantidad de bucles que contienen, que los hacen menos eficientes.

Hemos observado que los tiempos de ejecución cuentan con una gran diferencia, dependiendo del modo de ejecución del programa. Si ejecutamos el programa a través de un IDE los tiempos de ejecución son mucho menores que si lo ejecutamos directamente desde el intérprete de Python.

En la siguiente tabla se representan los tiempos obtenidos en 3 ordenadores diferentes, ejecutando el programa desde el intérprete de Python:

TABLA II
TABLA TIEMPOS RUEDA GUI

Por Prego	Por Delta	Por Búsqueda
1.104 ms	11.455 ms	0.088 ms, 0.096 ms
7.135 ms	22. 752 ms	8.914 ms, 8,468 ms
7.091 ms	9.498 ms	1.572 ms, 15.436 ms

TABLA III
TABLA TIEMPOS RUEDA CONSOLA

Por Prego	Por Delta	Por Búsqueda
7.314 ms	10.850 ms	0.336 ms, 0.361 ms
16.580 ms	22.602 ms	2.408 ms, 10.707 ms
4.781 ms	15.475 ms	5.506 ms, 9.340 ms

B. Problema robot

Este ejercicio lo hemos escrito a partir de un problema del boletín de la asignatura. Consiste en un robot que tiene como objetivo recoger los objetos que se encuentran en distintas, y para ello se tiene que desplazar. Las acciones que tenemos son:

TABLA IV
TABLA ACCIONES PROBLEMA DE ROBOT Y OBJETOS

Acción	Precondiciones	Efectos
Ir	robotEnX	¬robotEnX, robotEnY
Coger	robotEnX, objetoEnX	¬objetoEnX, tieneObjeto
Soltar	robotEnX, tieneObjeto	¬tieneObjeto, objetoEnX

Tenemos 2 objetos: Objeto1 y Objeto2. Y 3 salas : 'A', 'B', 'C'.

El estado inicial es: RobotenA, Objeto1EnB y Objeto2EnC.

En este caso particular del problema, no es necesario soltar los objetos, por lo que la acción de soltar no se aplicará.

Podemos comprobar que la solución a la heurística, calculada manualmente con anterioridad es de 4:

1) *Ir de sala A a sala B*
2) *Coger Objeto1*
3) *Ir de sala A a Sala C*
4) *Coger Objeto2*

Tras realizar delta, hemos comprobado que no da un valor fijo, si no que oscila entre 3 y 5. Esto se puede deber al orden de ejecución de los objetivos, ya que algunas veces comienza con coger(Objeto2), aunque el primer objetivo introducido sea coger(Objeto1). Suponemos que se debe a alguna casuística de la librería usada para PDDL que no hemos podido encontrar. Incluso, hemos llegado a valorar que este varianza es debida a la arquitectura de la cpu de cada ordenador, puesto mientras que a un compañero le sale siempre 4, a otro le varía continuamente y ambos tenemos procesadores de distinto

fabricante(AMD / Intel) . Otra opción puede ser el IDE ya que mientras uno trabaja en Spyder, otro lo hace en IntelliJ IDEA.

Con prego, no es un resultado oscilante, si no que da siempre 5 puesto que hace un paso intermedio, voliendo al inicio para ir de una sala a otra.

Para el algoritmo de búsqueda, nos devuelve el camino exacto a tomar ,obteniendo así la misma solución que la realizada manualmente.

De nuevo, ocurre que fuera del IDE los tiempos son mayores, con valores similares a los del problema anterior.

En la siguiente tabla se representan los tiempos obtenidos en 3 ordenadores diferentes:

TABLA V
TABLA TIEMPOS ROBOT GUI

Por Prego	Por Delta	Por Búsqueda
4.509 ms	15.494 ms	0.030 ms, 0.249 ms
17.854 ms	33.838 ms	2.074 ms, 14.855 ms
15.516 ms	29.473 ms	13.506 ms, 15.485 ms

TABLA VI
TABLA TIEMPOS ROBOT CONSOLA

Por Prego	Por Delta	Por Búsqueda
12.034 ms	15.999 ms	0.167 ms, 0.244 ms
19.602 ms	29.257 ms	1.844 ms, 5.537 ms
14.086 ms	27.162 ms	2.583 ms, 11.333 ms

VI. CONCLUSIONES

El objetivo de esta práctica ha sido obtener la solución de problemas de planificación y su heurística, por diferentes métodos. Tras la realización de esta, hemos sacado varias conclusiones que se explicarán a lo largo de esta sección.

La primera de ellas, que trabajar enfermo no es eficiente. Y esto se debe a que uno de los miembros del grupo, recibió la dosis de la vacuna y estuvo dos días con fiebre en los cuales los avances fueron mínimos, y cuando se recuperó, al día siguiente le surgió una conjuntivitis que le impedía mirar a la pantalla y la ha ido arrastrando hasta el final de la entrega del trabajo, lo que ha hecho que el avance de la última semana haya sido difícil.

Sobre planificación, nos hemos dado cuenta que es un problema de la vida cotidiana y es aplicable a prácticamente cualquier situación de manera que se obtiene la manera más óptima de realizar alguna situación, de manera que te permite no realizar pasos innecesarios o evitar el orden incorrecto, ahorrando así mucho tiempo en algunos casos. Un ejemplo sería: si tienes que ir de un punto A a un punto B, si existen dos caminos, uno que pasa por otro punto C pero es más corto, o uno directo que es más largo, se elige en que pasa por C. De esta manera hay muchos programas cotidianos como Google Maps que lo usan en sus algoritmos para ofrecernos la mejor ruta posible.

De los resultados hemos aprendido que puede haber más de una ruta similar, y que si el problema no está bien definido, ya sea por condiciones u objetivos, no es posible que devuelvan las soluciones correctas. Del mismo modo si una función falla, puede darse la posibilidad de que no devuelva una función correcta.

Como mejoras al programa, se podría buscar una mayor eficiencia de los algoritmos, debido a que una gran recursividad en python no es una buena práctica. Una opción para que el usuario pueda definir sus propios problemas y solucionarlos, e incluso una mejoría visual de la interfaz gráfica. Añadir métodos de búsqueda nuevos también sería una buena mejora, para poder contrastar entre varios, ya que no tienen porque dar todos el mejor resultado al problema descrito. También pensamos en programar una función que genere objetivos y estados iniciales aleatorios sobre un problema base, que por falta de tiempo y problemas externos no pudimos realizar.

Este trabajo nos ha ayudado a aprender y profundizar más sobre el funcionamiento de la heurística y a comprender como funciona el algortimo de obtención de esta.

Para la documentación, hemos seguido el formato recomendado por la asignatura ”LaTeX”, el cual se nos ha hecho un poco complejo de usar, puesto que no lo conocíamos. Lo hemos editado a través de la página web de Cocalc [7] la cual nos ofrecía un debugeador, con errores y avisos, además de una preview del pdf que se generará. Ha sido de mucha ayuda, aunque no ha sido cómodo y pensamos que se podría seguir el formato LaTeX mediante Word lo que lo haría mucho menos complejo. Para concluir el trabajo tuvimos que usar Overleaf [9], puesto que el compilador de Cocalc dejó de funcionar.

REFERENCIAS

[1] Departamento de ciencias de la computación e inteligencia artificial de la universidad de Sevilla, ”Práctica de planificación automática”.
[2] Tomás de la Rosa Turbides, Razonamiento basado en casos aplicado a la planificación heurística, Madrid 2009.
[3] Fernando Berzal, Universidad de Granada, ‘PDDL’.
[4] Wikipedia, ‘Búsqueda en profundidad’.
[5] Wikipedia, ‘Búsqueda en anchura’.
[6] http://cs-www.cs.yale.edu/homes/dvm/ , Drew V. McDermott.
[7] https://cocalc.com/ , editor latex.
[8] Wikipedia, ’Tkinter, biblioteca gráfica’.
[9] https://www.overleaf.com// , editor latex.
[10] Departamento de ciencias de la computación e inteligencia artificial de la universidad de Sevilla, ”Planificación”.