uc3m

Práctica 2: *Satisfacibilidad Lógica y Búsqueda Heurística*
Heurística y Optimización
Grado de Ingeniería Informática, 2018 - 2019

1. Objetivo

El objetivo de esta práctica es que el alumno aprenda a modelar y resolver problemas tanto de satisfacibilidad lógica (SAT) como de búsqueda heurística.

2. Enunciado del problema

Paganitzu [1] es un videojuego arcade lanzado en los años 90, que está basado en el clásico juego del *Sokoban* [2]. En él, el protagonista llamado Alabama o "Al" Smith se encuentra en un laberinto tipo *grid* tal como se muestra en la Figura 1. En la versión simplificada del juego que consideraremos, el objetivo es conseguir que Al recorra el laberinto recogiendo todas las llaves y, una vez recogidas, salga del laberinto. Para conseguir este objetivo, Al puede empujar las rocas en forma de bolas que aparecen y que obstaculizan sus movimientos. En el laberinto, también aparecen serpientes que disparan de forma horizontal si el protagonista cruza por delante, aunque no disparan si tienen un obstáculo (llave o roca) que obstaculice su visión. Por lo tanto, estas rocas también se pueden colocar delante de las serpientes para evitar ser disparados. Para entender mejor la dinámica del juego se recomienda jugar algunas partidas [3], aunque, como se ha comentado, en esta práctica consideraremos la versión simplificada del juego anteriormente descrita (i.e., no se consideran gemas, arañas, ni demás elementos del juego).

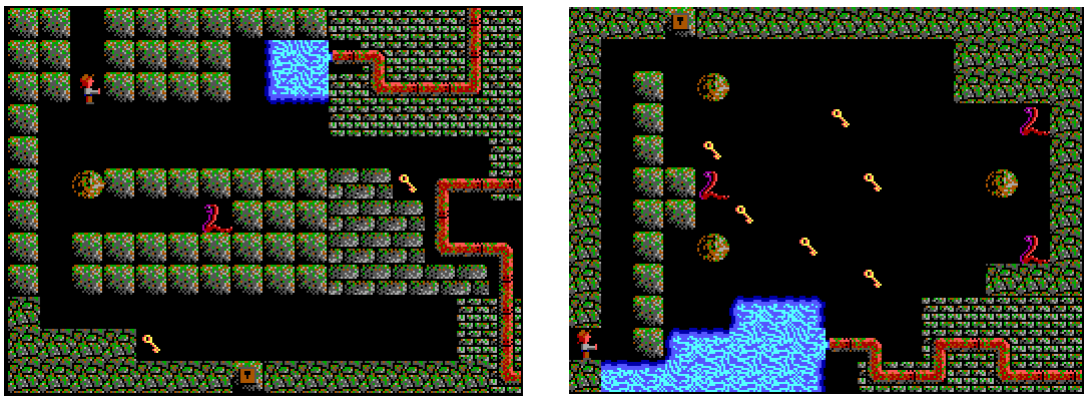

Figura 1: Ejemplo de laberintos para el juego del Paganitzu.

El laberinto está conformado por diferentes celdas, y en cada celda se puede encontrar un único elemento del juego: el protagonista, una serpiente, una roca, una pared o un obstáculo que no podemos atravesar, las llaves, la salida, o una celda vacía. En los mapas que se elaboren en esta práctica cada uno de estos elementos

[1] https://en.wikipedia.org/wiki/Paganitzu
[2] https://es.wikipedia.org/wiki/Sokoban
[3] https://www.playdosgames.com/online/paganitzu/

se va a representar con una letra diferente: 'A' para el protagonista, 'S' para las serpientes, 'O' para las rocas, 'K' para las llaves, 'E' para la salida del laberinto, '%' para los obstáculos que no podemos atravesar, y un espacio en blanco para las celdas vacías. De esta forma, los mapas correspondientes a los laberintos de la Figura 1 serían los que se muestran en la Figura 2.

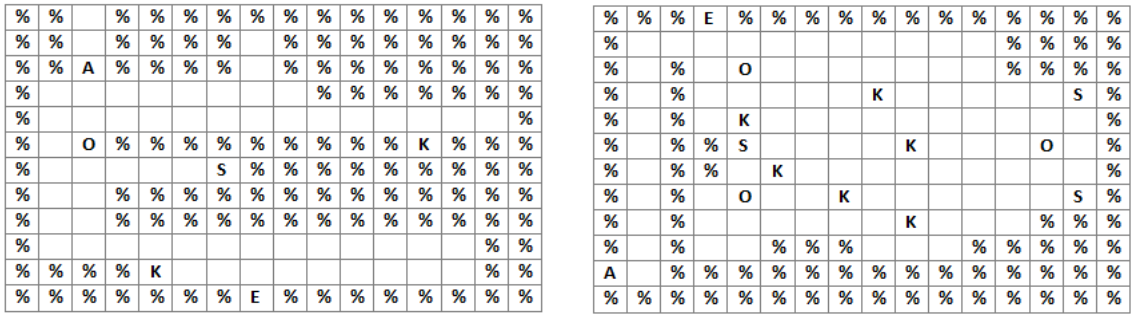

Figura 2: Traducción a letras de los laberintos de la Figura 1.

2.1. Parte 1: Diseño de laberintos con SAT

En el diseño de los laberintos, uno de los problemas que surgen es dónde situar inicialmente los diferentes elementos del juego de forma que el laberinto sea atractivo para los jugadores. Para ello, se ha decidido automatizar la colocación del protagonista y las serpientes en un laberinto dado (i.e., las paredes, rocas, llaves, y la salida vienen definidos inicialmente). Las restricciones a tener en cuenta en este proceso son:

- Al y las serpientes sólo se pueden colocar en celdas vacías.
- Una serpiente no puede estar en la misma fila que otra serpiente.
- No puede haber ninguna serpiente ni en la misma fila ni en la misma columna que Al.

Para esta parte se pide:

- Modelar el problema como un problema de satisfacibilidad lógica, en Forma Normal Conjuntiva (CNF).
- Utilizando JaCoP, desarrollar un programa que codifique el modelo anterior y determine dónde colocar a Al y las serpientes. La implementación desarrollada se deberá ejecutar desde una consola o terminal con el siguiente comando:

  ```
  java SATPaganitzu <laberinto> n
  ```

  donde:

  1. *laberinto*: Es el nombre del fichero que contiene un laberinto en el formato indicado en la Figura 2 pero vacío, es decir, sólo indicando las paredes, llaves, salida y las rocas. Un ejemplo de nombre para este fichero podría ser *lab1_parte1.lab*.
  2. *n*: Número de serpientes a colocar en el laberinto.

  En el caso de que el problema sea satisfacible, el programa generará como salida un laberinto que se escribirá en un fichero utilizando el formato de la Figura 2 que contendrá, además de los elementos iniciales, las n serpientes y a Al. El nombre del fichero generado debe ser el nombre del fichero de entrada con extensión *'.output'* (por ejemplo, *lab1_parte1.lab.output*). En el caso de que el problema no sea satisfacible, se imprimirá un mensaje por pantalla informando de este hecho.

- Los alumnos deben generar sus propios casos de prueba, es decir, sus propios laberintos vacíos de varios tamaños y formas y con un número de serpientes diferente a incluir en ellos.

2.2. Parte 2: Búsqueda Heurística

El problema que se propone en esta parte consiste en calcular la ruta de coste mínimo que permite a Al recoger todas las llaves y salir del laberinto, mientras evita ser disparado por las serpientes. Para ello, Al puede desplazarse entre dos celdas contiguas horizontales o verticales siempre y cuando no se lo impida una pared u obstáculo que no puede atravesar. Además, puede empujar las rocas que obstaculizan su camino, o bien puede empujarlas para situarlas en la línea de visión de las serpientes y evitar así ser disparado. Para poder empujar una roca, Al tiene que estar en una celda contigua, y la celda a la que se desea mover la roca debe estar vacía. Cuando Al empuja una roca pasa a ocupar la celda que ocupaba la roca. Los costes que se contemplan son los siguientes:

- Una celda vacía puede atravesarse con coste 2.
- Mover una roca tiene coste 4.

En esta parte se pide:

- Modelar el problema del cálculo de rutas propuesto como un problema de búsqueda heurística.
- Implementar el algoritmo A* que permite resolver el problema e implementar dos funciones heurísticas informadas y admisibles que estimen el coste restante, de modo que sirvan de guía para el algoritmo. La implementación desarrollada se deberá ejecutar desde una consola o terminal con el siguiente comando:

  ```
  java AstarPaganitzu <laberinto> <heurística>
  ```

  donde:

  - *laberinto:* Nombre del fichero que contiene el laberinto en el formato que se especifica en la Figura 2. Ejemplo: `lab1_parte2.map`.
  - *heurística:* Nombre de la heurística.

  Los posibles valores para el parámetro *heurística* deben ser detallados en la memoria y en la ayuda de la implementación desarrollada. El programa debe generar dos ficheros de salida. Ambos se deben generar en el mismo directorio dónde se encuentre el laberinto de entrada y deben tener el mismo nombre del laberinto (extensión incluida) más una extensión adicional. Los ficheros son los siguientes:

  - Laberinto de salida. Debe contener la ruta realizada por Al para recoger todas las llaves y llegar hasta la salida. Para ello, se mostrará en primer lugar el laberinto de entrada, e inmediatamente debajo el camino seguido por Al, por ejemplo, *(0,0)→(0,1)→(1,1)→...*, donde cada una de las posiciones (x, y) indica la fila x y la columna y donde se encuentra Al. La extensión de este fichero debe ser '`.output`'. Ejemplo: `lab1_parte2.lab.output`.
  - Fichero de estadísticas. Este fichero debe contener información relativa al proceso de búsqueda, como el tiempo total, coste total, longitud de la ruta, nodos expandidos, etc. Por ejemplo,

    ```
    Tiempo total: 145
    Coste total: 54
    Longitud de la ruta: 27
    Nodos expandidos: 132
    ```

    La extensión de este fichero debe ser '`.statistics`'. Ejemplo: `lab1_parte2.lab.statistics`.

- Proponer casos de prueba sobre diversos laberintos de entre los generados en la primera parte y los que se quieran generar expresamente para esta parte, y resolverlos con la implementación desarrollada. Estos casos se deben generar razonablemente dependiendo de la eficiencia alcanzada en la implementación.

- Realizar un estudio comparativo utilizando las dos heurísticas implementadas (número de nodos expandidos, tiempo de cómputo, etc.).
- Como **parte opcional** de la práctica se propone modelar e implementar el movimiento diagonal entre dos casillas contiguas, implementar una heurística que considere dicho movimiento y un breve análisis comparativo como complemento al anterior. Dicha parte se calificaría con un punto extra sobre la nota final de la práctica.

3. Directrices para la Memoria

La memoria debe entregarse en formato pdf y tener un máximo de 15 hojas en total, incluyendo la portada, el índice y la contraportada. Al menos, ha de contener:

1. Breve introducción explicando los contenidos del documento.
2. Descripción de los modelos, argumentando las decisiones tomadas.
3. Análisis de los resultados.
4. Conclusiones acerca de la práctica.

Importante: Las memorias en un formato diferente a pdf serán penalizadas con 1 punto.

La memoria **no debe incluir código fuente** en ningún caso.

4. Evaluación

La evaluación de la práctica se realizará sobre 10 puntos. Sin embargo, es posible obtener un punto adicional si se realiza la parte opcional. En este caso, la nota máxima de la práctica sería de 11 puntos.

Para que la práctica sea evaluada deberá realizarse al menos la primera parte de la práctica y la memoria.

La distribución de puntos es la siguiente:

1. Parte 1 (3 puntos)
   - Modelización del problema del diseño de laberintos (0.75 puntos)
   - Implementación del modelo (1.25 puntos)
   - Resolución y análisis de los casos de prueba (1 punto).
2. Parte 2 (7 puntos)
   - Modelización del problema (0.75 puntos)
   - Implementación del algoritmo (2.25 puntos)
   - Resolución y análisis de los casos de prueba (1.75 puntos)
   - Análisis comparativo de las heurísticas implementadas (2.25 puntos)
3. Parte opcional (1 punto)

En la evaluación de la modelización de los problemas, un modelo correcto supondrá la mitad de los puntos. Para obtenerse el resto de puntos, la modelización del problema deberá:

- Ser formalizada correctamente en la memoria.
- Ser, preferiblemente, sencilla y concisa.

- Estar bien explicada (ha de quedar clara cuál es la utilidad de cada variable/restricción).
- Justificarse en la memoria todas las decisiones de diseño tomadas.

En la evaluación de la implementación de los modelos, y cualquier aspecto relacionado con ella, sólo se corregirán implementaciones que compilen y que respondan a lo que se solicite en el enunciado. Además, una implementación correcta supondrá la mitad de los puntos. Para obtenerse el resto de puntos, la implementación del problema deberá:

- Corresponderse íntegramente con el modelo propuesto en la memoria.
- Entregar código fuente correctamente organizado y comentado a lo que se pide en el enunciado. Los nombres deben ser descriptivos.
- Contener casos de prueba que aporten diversidad para la validación de la implementación.

En la evaluación del análisis de resultados, se valorará positivamente la inclusión de conclusiones personales sobre la dificultad de la práctica y sobre lo que se ha aprendido durante el desarrollo de la misma.

5. Entrega

Se tiene de plazo para entregar la práctica hasta el 16 de Diciembre a las 23:55. Sólo un miembro de cada pareja de estudiantes debe subir a la sección de prácticas de Aula Global un único fichero `.zip`. Es importante cumplir las siguientes instrucciones para la entrega.

El fichero debe nombrarse `p2-NIA1-NIA2.zip`, donde `NIA1` y `NIA2` son los últimos 6 dígitos del NIA (rellenando con 0s por la izquierda si fuera preciso) de cada miembro de la pareja. Ej: `p2-054000-671342.zip`.

La descompresión de este fichero debe producir al menos dos directorios llamados exactamente "`parte-1`" y "`parte-2`". Además, si se entregara la parte extra (la que implementa los desplazamientos en diagonal), entonces debe incluirse un tercer directorio "`parte-3`" que la contenga.

La memoria en formato .pdf debe incluirse en la raíz de estos directorios y debe llamarse `NIA1-NIA2.pdf`, mientras que las soluciones de cada parte deben incluirse en el subdirectorio que corresponda.

Importante: no seguir las normas de entrega puede suponer una pérdida de hasta 1 punto en la calificación final de la práctica.