uc3m

Práctica 2: *Satisfacción de Restricciones y Búsqueda Heurística*
Heurística y Optimización
Grado de Ingeniería Informática, 2019 - 2020

1. Objetivo

El objetivo de esta práctica es que el alumno aprenda a modelar y resolver problemas tanto de satisfacción de restricciones (CSP) como de búsqueda heurística.

2. Enunciado del problema

Un importante colegio de la Comunidad de Madrid se ha puesto en contacto con alumnos del curso de Heurística y Optimización. Tras varias entrevistas os han seleccionado a tu compañero y a ti para dar solución a varios de sus problemas.

2.1. Parte 1: Generación de horarios mediante CSP

La elaboración de horarios por parte de los colegios se ha convertido en una tarea compleja que requiere tener en cuenta los profesores disponibles, sus preferencias, el número de materias a impartir y las restricciones entre éstas. En el caso particular que nos ocupa, el colegio se ha puesto en contacto con nosotros para que elaboremos el horario de clases de 4º curso de primaria. Para ello, el colegio nos proporciona información que deberemos tener en cuenta. En concreto, nos comenta que en este curso los alumnos tienen 6 materias: Ciencias de la Naturaleza, Ciencias Sociales, Lengua Castellana y Literatura, Matemáticas, Inglés y Educación Física. Además, el colegio nos informa que dispone de 3 profesores para impartir estas asignaturas: Lucía, Andrea, y Juan. En cuanto al horario de clases, los lunes, martes, y miércoles, se deben impartir 3 horas que se distribuyen de 9:00 a 12:00. Los jueves, en cambio, sólo hay clases de 9:00 a 11:00. Las restricciones que el colegio nos impone a la hora de elaborar este horario son las siguientes:

- La duración de cada una de las clases es de 1 hora de duración, en la que sólo se puede impartir una única materia.
- Para todas las materias se deben impartir 2 horas semanales, excepto para Educación Física que sólo tiene asignada 1 hora semanal.
- Las 2 horas dedicadas a cada materia podrían impartirse de forma no consecutiva, e incluso en días diferentes, excepto las 2 horas dedicadas a Ciencias de la Naturaleza que sí se deben impartir de forma consecutiva el mismo día.
- La materia de Matemáticas no puede impartirse el mismo día que Ciencias de la Naturaleza e Inglés.
- Además, la materia de Matemáticas debe impartirse en las primeras horas, y la de Ciencias Sociales en las últimas.
- Cada profesor debe impartir 2 materias, que son diferentes a las de sus compañeros.
- Lucía solo se encargará de Ciencias Sociales, si Andrea se encarga de Educación Física.

- Juan no quiere encargarse de Ciencias de la Naturaleza o de Ciencias Sociales, si algunas de sus horas se imparte a primera hora los lunes y jueves.

Para esta parte se pide:

- Modelar el problema como un problema de satisfacción de restricciones CSP.
- Utilizando la librería *python-constraint*, desarrollar un programa que codifique el modelo anterior, y permita determinar a qué horas se imparten cada una de las materias y qué profesor se encarga de cada una de ellas.

  El programa deberá ejecutarse desde una consola o terminal con el siguiente comando:

  ```
  python CSPScheduling.py
  ```

- Cada grupo debe generar sus propios casos de prueba, por ejemplo, modificando el caso propuesto añadiendo/quitando profesores/asignaturas/restricciones, ampliando/reduciendo el horario de clases, con el fin de analizar el comportamiento del programa.

2.2. Parte 2: Búsqueda Heurística

En esta segunda parte debemos encargarnos de un problema diferente. En concreto, debemos planificar la ruta de un autobús escolar que recoge a los estudiantes en varias paradas y los lleva a sus respectivos colegios. La figura 1 muestra un ejemplo de problema que debemos resolver.

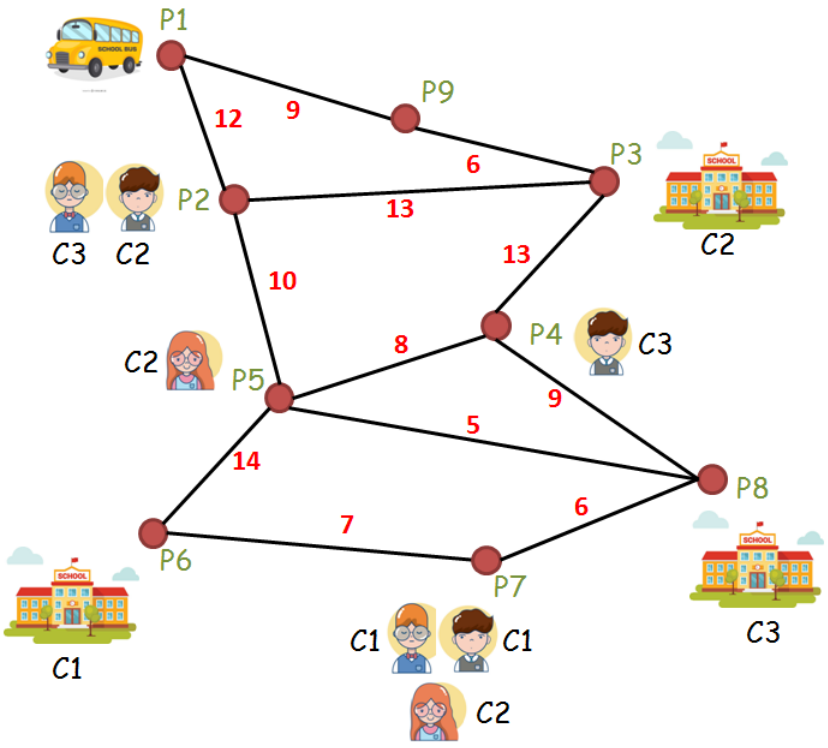


Figura 1: Ejemplo de problema con nueve paradas, tres colegios y siete estudiantes.

La figura 1 representa un problema con nueve paradas $(P_1, P_2, \ldots, P_9)$, tres colegios (C_1, C_2, C_3) y siete estudiantes repartidos entre las paradas P_2, P_4, P_5 y P_7. Como se puede ver, el problema se puede representar como un grafo, donde cada uno de los nodos representa una parada de autobús, y los arcos representan los trayectos entre estas paradas. Cada uno de estos trayectos tiene un coste asociado que representa lo que le

```
    P1  P2  P3  P4  P5  P6  P7  P8  P9
P1  --  12  --  --  --  --  --  --   9
P2  12  --  13  --  10  --  --  --  --
P3  --  13  --  13  --  --  --  --   6
P4  --  --  13  --   8  --  --   9  --
P5  --  10  --   8  --  14  --   5  --
P6  --  --  --  --  14  --   7  --  --
P7  --  --  --  --  --   7  --   6  --
P8  --  --  --   9   5  --   6  --  --
P9   9  --   6  --  --  --  --  --  --
C1: P6; C2: P3; C3: P8
P2: 1 C2, 1 C3; P4: 1 C3; P5: 1 C2; P7: 2 C1, 1 C2
B: P1 5
```

Tabla 1: Example of an input file

cuesta al autobús moverse de una parada a otra. Además, para cada estudiante se sabe a qué colegio debe ir. En el caso de ejemplo de la figura 1, hay dos estudiantes en la parada P_2 (el primero va al colegio C_2 y el otro al C_3), uno en la parada P_5 (que va al colegio C_3), tres en P_7 (dos van a C_1 y uno a C_2), y uno en P_4 (que va a C_3). Cuando el autobús pasa por sus paradas, los estudiantes pueden entrar en el autobús, y ya no vuelven a bajar hasta que no llegan a su colegio de destino.

La capacidad del autobús está limitada a un número máximo de estudiantes, que no se debe superar en ningún momento del trayecto. Inicialmente el autobús se encuentra en un parada, y debe acabar en esta misma parada una vez haya llevado a los estudiantes a sus colegios. En esta ruta, algunas paradas sin estudiantes podrían quedar sin servicio, es decir, el autobús podría no pasar por ellas. El objetivo de esta segunda parte es calcular la ruta que debería seguir el autobús para trasladar a los estudiantes que esperan en sus paradas a sus respectivos colegios de forma que se minimice el coste de transporte.

En particular, se pide:

- Modelar el problema del cálculo de rutas propuesto como un problema de búsqueda heurística.
- Implementar el algoritmo A* (en el lenguaje de programación que se considere) 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:

  ```
  ./BusRouting.sh <problema.prob> <heurística>
  ```

  donde:

  - `BusRouting.sh`: Nombre del script que permite invocar al programa desarrollado.
  - `problema.prob`: Nombre del fichero que contiene el problema que queremos resolver. Un ejemplo de fichero de entrada que modela el problema de la figura 1, se muestra en la tabla 1.

    En la tabla 1 se muestra en primer lugar la matriz de costes entre todas las paradas de autobús. Estas paradas están identificadas desde P_1 a P_n, donde n es el número de paradas de autobús que vamos a considerar. Después de esta matriz de costes, debe aparecer una línea que identifica las paradas donde se encuentran los colegios. En esta línea, para cada colegio, se sigue el formato, $C_i : P_j$, para identificar que el colegio C_i, $i \in \{1, \ldots, m\}$, está en una parada P_j, $j \in \{1, 2, \ldots, n\}$. Se puede comprobar que las paradas de cada colegio están separadas por punto y coma. Después de esta línea, aparece otra donde se especifica el número de alumnos de cada colegio que esperan en las paradas. Si en una parada hay

alumnos esperando se especifica como $P_j : L\ C_i$, indicando que en la parada P_j hay L alumnos que van al colegio C_i. Si en una parada hay estudiantes esperando de varios colegios se separan con coma como se puede ver en el ejemplo. En cambio, las diferentes paradas con alumnos esperando se separan con punto y coma. Por último, en la última línea se especifica la localización inicial y la capacidad del autobús con el formato $B : P_j\ q$, que indica que el autobús inicialmente se encuentra en la parada P_j y no puede llevar más de q estudiantes.

- `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 problema de entrada y deben tener el mismo nombre del problema (extensión incluida) más una extensión adicional. Los ficheros son los siguientes:

- Solución del problema. Debe contener la ruta realizada por el autobús para recoger a todos los estudiantes, llevarlos a sus colegios y regresar al punto de partida. Para ello, se mostrará en primer lugar el problema de entrada, e inmediatamente debajo la ruta seguida por el autobús, y por cada parada, los estudiantes que suben y bajan, por ejemplo, `P1` $\rightarrow$ `P2 (S: 1 C2, 1 C3)` $\rightarrow$ `P3 (B: 1 C2)` $\rightarrow \ldots \rightarrow$ `P1`, que indica que el autobús sale de la parada $P1$, va a la parada $P2$ donde se sube 1 persona que va al colegio $C2$ y 1 al colegio $C3$ (y que se denota con la letra `S`), después va hasta la parada $P3$ donde se baja una persona que va al colegio $C2$ (denotado con la letra `B`), y finalmente, después de realizar el resto de la ruta, regresa a la parada $P1$. La extensión de este fichero debe ser '.output'.

  Ejemplo: `problema.prob.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
Paradas visitadas: 27
Nodos expandidos: 132
```

  La extensión de este fichero debe ser '`.statistics`'. Ejemplo: `problema.prob.statistics`.

- Proponer casos de prueba con diversas configuraciones de problemas (distinto número de colegios y paradas, distinto número de estudiantes en las paradas, . . .) 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.).

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. 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 (4 puntos)
   - Modelización del problema (1.25 puntos)
   - Implementación del modelo (1.5 puntos)
   - Resolución y análisis de los casos de prueba (1.25 punto).
2. Parte 2 (6 puntos)
   - Modelización del problema (0.75 puntos)
   - Implementación del algoritmo (2.25 puntos)
   - Resolución y análisis de los casos de prueba (1.5 puntos)
   - Análisis comparativo de las heurísticas implementadas (1.5 puntos)

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 15 de Diciembre a las 23:55. El deadline es firme y no se extenderá en ningún caso.

Sólo un miembro de cada pareja de estudiantes debe subir:

- Un único fichero `.zip` a la sección de prácticas de Aula Global denominado *Entrega Práctica 2*.
- 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. Ejemplo: `p2-054000-671342.zip`.
- La memoria en formato pdf debe entregarse a través del enlace Turnitin denominado *Entrega Memoria Práctica 2*. La memoria debe entregarse en formato pdf y debe llamarse `NIA1-NIA2.pdf` — después de sustituir convenientemente el NIA de cada estudiante. Ejemplo: `054000-671342.pdf`

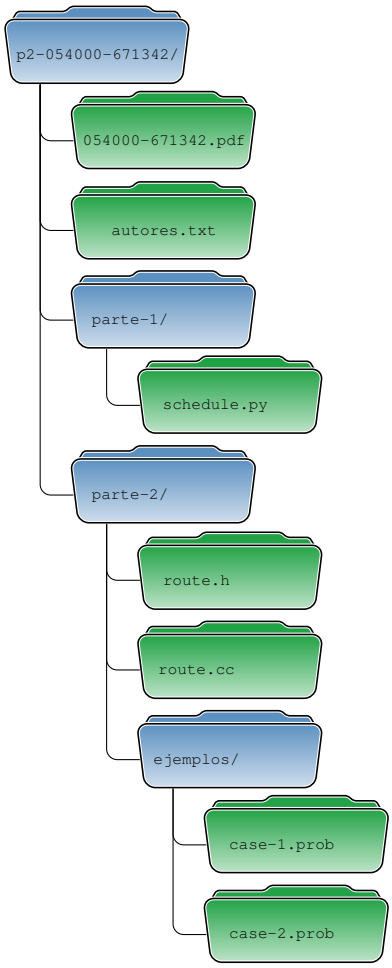
La descompresión del fichero entregado en primer lugar debe generar un directorio llamado `p2-NIA1-NIA2`, donde `NIA1` y `NIA2` son los últimos 6 dígitos del NIA (rellenando con 0s por la izquierda si fuera preciso) de cada miembro de la pareja. Este directorio debe contener: primero, la misma memoria en formato pdf que ha sido entregada a través de Turnitin, y debe llamarse `NIA1-NIA2.pdf` — después de sustituir convenientemente el NIA de cada estudiante; segundo, un fichero llamado *autores.txt* que identifique a cada autor de la practica en cada línea con el formato: NIA Apellidos, Nombre. Por ejemplo:

```
054000 Von Neumann, John
671342 Turing, Alan
```

La descompresión de este fichero debe producir al menos dos directorios llamados exactamente "`parte-1`" y "`parte-2`", que contengan los archivos necesarios para ejecutar correctamente cada una de las partes.

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

Se muestra a continuación una distribución posible de los ficheros que resultan de la descompresión:


p2-054000-671342/
054000-671342.pdf
autores.txt
parte-1/
schedule.py
parte-2/
route.h
route.cc
ejemplos/
case-1.prob
case-2.prob