Segunda práctica de Inteligencia Artificial
Curso 2021-2022

Departament d'Informàtica i Enginyeria Industrial
Universitat de Lleida
carlos@diei.udl.cat
eduard.torres@udl.cat
josep.alos@udl.cat

1. Enunciado

El objectivo de esta práctica es evaluar el conocimiento del alumno sobre la modelización de problemas de optimización como fórmulas de Máxima Satisfactibilidad (MaxSAT) y la implementación de sistemas que permitan describir y resolver estos problemas de optimización. En el campus virtual, en la carpeta `Laboratorio/maxsat` encontraréis disponible el código que hemos presentado y desarrollado durante las clases de laboratorio, y que os servirá de base para la realización de esta práctica.

1.1. Modelización de problemas de grafos: 2 puntos

En clase de laboratorio hemos mostrado cómo traducir el problema del Minimum Vertex Cover en una fórmula MaxSAT. Siguiendo el mismo esquema propuesto en clase, se os pide ahora que traduzcáis los problemas Maximum Clique y Max-Cut, cuya descripción se encuentra en las transparencias de clase.

Las funciones que tenéis que implementar (`min_vertex_cover`, `max_clique` y `max_cut`) las podéis encontrar en el fichero *graph.py*.

1.2. Transformación (1,3) Weighted Partial MaxSAT: 2 puntos

Implementad el procedimiento `to_13wpm` del fichero *wcnf.py* que transforme cualquier fórmula MaxSAT en una fórmula (1,3) Weighted Partial MaxSAT.

1.3. Modelización de *Software Package Upgrades* como WPMS: 4 puntos

Implementad una aplicación que permita resolver el problema **Software Package Upgrades** mediante su conversión a Weighted Partial MaxSAT.

1.3.1. Tareas a desarrollar

Deberéis proporcionar un fichero Python llamado *spu_ solver.py*, que recibirá como primer parámetro la ruta al binário del MaxSAT solver que se quiera utilizar para resolver el problema y como segundo parámetro la instancia del problema a resolver. La aplicación que desarrolléis deberá cumplir con la siguiente funcionalidad:

- Parsear el fichero de entrada (su especificación está detallada en la sección 1.3.2).
- Traducir el problema a una fórmula Weighted Partial MaxSAT siguiendo la traducción que podéis encontrar en las transparencias.
- Resolver la fórmula con un MaxSAT solver.
- Interpretar la solución del MaxSAT solver y devolver como resultado el listado de paquetes que no se podrán instalar (especificado en la sección 1.3.3).

1.3.2. Formato de entrada de las instancias

Dentro de la carpeta `Laboratorio/maxsat/spu` podréis encontrar instancias del problema que podéis utilizar para comprobar vuestra implementación. Dichas instancias tienen el siguiente formato:

1. `p spu <n>`, dónde n es un número natural: Primera línea del fichero. n indica el número total de paquetes que se consideran en la instancia.
2. `n <pkg-name>`: Descripción del nombre del paquete (e.g. `n mypckg`). El número de líneas de este tipo deberá ser igual al número natural especificado en la primera línea del fichero mencionada en el punto anterior.
3. `d <pckg-name><dependency>[dependency ...]`: Especificación de dependencias. En este tipo de línea se especifica una dependencia del paquete <pkg-name>con otro paquete, por ejemplo `d myapp python2` describiría la dependencia del paquete *myapp* con el paquete *python2*.

   Un paquete puede depender de una disyunción de paquetes, en cuyo caso se especificará dentro de la misma línea, por ejemplo `d myapp python2 python3` representaría la dependencia del paquete *myapp* con los paquetes *python2* **o** *python3*.

   Se pueden incluir tantas líneas de dependencias para un paquete como sean necesarias, siendo todo el conjunto de líneas asociadas a un paquete una conjunción de dependencias. Por ejemplo:

   ```
   d myapp gcc
   d myapp python2 python3
   ```

   representaría la dependencia del paquete *myapp* con los paquetes $gcc \wedge (python2 \vee python3)$. También es posible tener paquetes sin dependencias.

   Considerad que todos los paquetes referenciados en las líneas de dependencias ya han sido préviamiente declarados con anterioridad. De no ser así podéis lanzar un error.
4. `c <pkg-name><conflict>`: Especificación de conflictos. En este tipo de línea se especifica un conflicto del paquete <pkg-name>con otro paquete, por ejemplo `c myapp rust` describiría el conflicto del paquete *myapp* con el paquete *rust*.

   Se pueden incluir tantas líneas de conflictos para un paquete como sean necesarias, siendo todo el conjunto de líneas asociadas a un paquete una conjunción de conflictos. Por ejemplo:

   ```
   c myapp rust
   c myapp R
   ```

   representaría el conflicto del paquete *myapp* con los paquetes $rust \wedge R$. También es posible tener paquetes que no tengan conflictos.

   Considerad que todos los paquetes referenciados en las líneas de conflictos ya han sido préviamiente declarados con anterioridad. De no ser así podéis lanzar un error.
5. `# <comment>`: Comentario. Las líneas cuyo primer carácter sea "#"se consideran comentarios de la instancia y deben ser ignoradas.

1.3.3. Formato de salida de la solución

Vuestra aplicación deberá imprimir por la salida estándar la siguiente cadena de carácteres como solución:

```
o <n>
v [pckgs ...]
```

dónde n será el número de paquetes que no podrán ser instalados, y *pckgs* será una **lista en orden alfabético ascendiente** de dichos paquetes, separados por espacios.

1.3.4. Ejemplo

Dada la siguiente instancia a la que llamaremos *sample.spu*:

```
p spu 4
n pkg1
n pkg2
n pkg3
n pkg4
d pkg1 pkg2 pkg3
d pkg2 pkg3
d pkg3 pkg2
d pkg4 pkg2
d pkg4 pkg3
c pkg1 pkg4
c pkg2 pkg4
```

y considerando que nuestro solver se llama *EvalMaxSAT*, la llamada

```
python3 spu_solver.py ./EvalMaxSAT ./sample.spu
```

nos retornará la siguiente solución:

```
o 1
v pkg4
```

1.4. Implementación: 1 punto

En este apartado se valora la calidad y eficiencia de los algoritmos implementados. Es necesario tener en cuenta los siguientes aspectos de la implementación:

- El lenguaje de programación es Python 3.
- La utilización de un diseño orientado a objetos.
- La simplicidad y legibilidad del código.
- Se recomienda seguir la guía de estilo [1].

1.5. Documentación: 1 punto

Documento en pdf (máximo $\approx$ 4 páginas) que incluya:

- una descripción de las decisiones que se han tomado en la implementación de los algoritmos

La página inicial ha de contener el nombre de los integrantes del grupo. La práctica puede realizarse en grupos de dos personas o individualmente.

Se valorará la redacción y presentación del documento.

[1] `https://www.python.org/dev/peps/pep-0008/`

1.6. (Opcional) Comprobación de soluciones

Como tarea opcional, se os pide ampliar vuestra implementación del *spu_ solver.py* para comprobar si la solución reportada es una solución válida. Para ello deberíais realizar las siguientes modificaciones:

- Vuestro script *spu_ solver.py* deberá recibir un parámetro opcional adicional que sea `--validate`. En caso de que el usuario active dicho parámetro deberéis activar la comprobación de la solución retornada.
- La comprobación de la solución tendrá que asegurar que todos los paquetes que se especifican como instalados puedan ser instalados. Por cada paquete marcado como instalado tendréis que comprobar:
  - Que al menos una de las dependencias por cada conjunción de dependencias esté instalada.
  - Que no haya ningún paquete conflictivo instalado.

  Podéis añadir otras comprobaciones si lo creeis necesario.
- Como output, justo después de imprimir la solución, deberéis especificar si la solución es válida o no utilizando el siguiente formato:

```
c VALIDATION [OK|WRONG]
```

1.6.1. Ejemplo

Para la instancia *sample.spu* descrita en el ejemplo de la sección 1.3.4, activaríamos la validación con el siguiente comando:

```
python3 spu_solver.py ./EvalMaxSAT ./sample.spu --validate
```

La solución retornada sería la que se muestra a continuación:

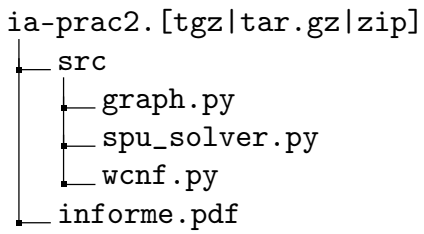
```
o 1
v pkg4
c VALIDATION OK
```

1.7. Material a entregar

El material evaluable de esta práctica es:

- Todos los archivos de código fuente, modificados o añadidos.
- Documento de la práctica.

Todo el material requerido se entregará en el paquete de nombre `ia-prac2.[tgz|tar.gz|zip]`



NOTA: Tened el cuenta que para desarrollar la práctica también necesitaréis el fichero **msat_runner.py**, el cual nos permite ejecutar el MaxSAT solver dentro de Python de manera sencilla, y el binario del própio MaxSAT solver, en nuestro caso **EvalMaxSAT**, pero NO es necesario que los incluyáis dentro del material a entregar.