Segunda práctica de Inteligencia Artificial
Curso 2015-2016

Departament d'Informàtica i Enginyeria Industrial
Universitat de Lleida
carlos@diei.udl.cat
jponfarreny@diei.udl.cat

En esta práctica, tendréis que construir un sistema que sea capaz de resolver el problema de las subastas combinatorias. En particular, una vez recibidas las pujas de los agentes os encargaréis de seleccionar aquellas que son compatibles y que maximizan el beneficio del subastador. Para ello, tendréis que resolver el problema de las subastas combinatorias mediante su traducción al problema de la Máxima Satisfactibilidad y la utilización de un *MaxSAT solver*.

1. Generación de subastas combinatorias

Para generar subastas combinatorias, tal y como hemos visto en clase de laboratorio, debéis utilizar la herramienta CATS (Combinatorial Acution Test Suite), disponible en `https://www.cs.ubc.ca/~kevinlb/CATS/`.

Recordad que tenéis la versión para Linux 64 bits en la carpeta de la sesión 5 en campus virtual [1].

2. Codificación de una subasta combinatoria como una fórmula Max-SAT (4 puntos)

En este apartado vamos a describir cómo codificar el problema de la subasta combinatoria como una instancia del problema de la Máxima Satisfactibilidad (MaxSAT).

Primero definimos las variables booleanas:

- $X = \{x_1, \ldots, x_M\}$, donde $x_i = 1$ significa que la puja i ha sido aceptada.

Entonces, la formulación estándar expresada usando cláusulas con pesos es:

- `[1 punto]` Soft (*coste de la puja*):
  Por cada puja i, añadimos:
  $$(x_i, w_i)$$
- `[1 punto]` Hard (*pujas incompatibles*):
  Por cada pareja de pujas i y j, tal que $S_i \cap S_j \neq \emptyset$, añadimos:
  $$(\neg x_i \vee \neg x_j, \infty)$$

Además, se os pide que extendáis esta formulación para los siguientes casos:

- `[1 punto]`: al menos se debe aceptar una puja de cada agente.
- `[1 punto]`: como máximo podemos aceptar una puja de cada agente.

Opcional: Utilizar una codificación lineal para implementar las restricciones "At most one".

[1] `https://cv.udl.cat/access/content/group/102020-1617/laboratorio/sesion%205/cats-linux-x64`

3. Implementación (2 puntos)

En este apartado se valorará la calidad y eficiencia de las funciones implementadas. Es necesario tener en cuenta los siguientes aspectos de la implementación:

- El lenguaje de programación es **Python 2.7**.
- La utilización de un diseño orientado a objetos.
- La simplicidad y legibilidad del código.
- **Se recomienda** seguir la guía de estilo [2].

4. Evaluación Experimental (4 puntos)

En este apartado se evaluará la traducción y las extensiones comentadas en el apartado 2:

1. `[1 punto]` Formulación estándar
2. `[1 punto]` Extensión: al menos se debe aceptar una puja de cada agente.
3. `[1 punto]` Extensión: como máximo podemos aceptar una puja de cada agente.
4. `[1 punto]` Codificar el problema como una instancia 1,3-WPM

Para comprobar que vuestra solución es correcta los instructores generarán un conjunto de subastas combinatorias. Para evaluar las tres primeras partes, se utilizará vuestra herramienta para traducir cada subasta combinatoria en una instancia MaxSAT. Se considerará que vuestra traducción es correcta si al aplicarle un *MaxSAT solver* proporciona un óptimo que se corresponde con el de la subasta combinatoria que se haya traducido. Para evaluar la cuarta parte, se comprobará que efectivamente se trata de una instancia 1,3-WPM y que el óptimo es correcto.

La puntuación de cada tipo de traducción se calculará de la siguiente manera:

$$mark = 1{,}0 * \frac{|correct\ translations|}{|instances|}$$

4.1. Entorno de evaluación

Para automatizar la evaluación experimental, deberéis utilizar como *main* de vuestra solución el archivo Python *ca2msat.py* que se adjunta con la práctica. Los parámetros que acepta y su significado son:

- -a/--auction: nombre del archivo que contiene la subasta combinatoria a resolver.
- -f/--formula: nombre del archivo donde guardar la fórmula wcnf.
- -r/--result: nombre del archivo donde guardar el resultado de la subasta.
- -s/--solver: ruta del solver WPM3.
- -alo/--accept-at-least-one: Añadir a la fórmula las cláusulas necesarias para aceptar al menos una puja de cada agente.
- -amo/--accept-at-most-one: Añadir a la fórmula las cláusulas necesarias para aceptar como máximo una puja de cada agente.
- -t13/--transform-to-1-3-wpm: El formato de la fórmula debe ser 1,3-WPM.

El solver que se le proporcionará a la herramienta será el **WPM3** que tenéis en la carpeta de la sesión 5 en el campus virtual [3].

[2] `https://www.python.org/dev/peps/pep-0008/`
[3] `https://cv.udl.cat/access/content/group/102020-1617/laboratorio/sesion%205/WPM1-2012`

El formato del resultado debe ser, una sola línea que empezará con la letra **b** y a continuación, si la subasta combinatoria tiene solución, la lista de los identificadores de las pujas aceptadas. Por ejemplo, si se aceptan las pujas $0, 3, 5, 6$ y 7 se escribirá:

```
b 0 3 5 6 7
```

Si la subasta combinatoria no tiene solución, la herramienta deberá escribir:

```
b NO SOLUTION
```

5. Material a entregar

El material evaluable de esta práctica es, todo el código fuente que hayáis escrito. Todo el material requerido se entregará en un paquete comprimido de nombre ia-prac2.[tgz|tar.gz|zip].

```
ia-prac2.[tgz|tar.gz|zip]
└── src
    ├── ca2msat.py
    └── *.py
```

Nota: Si debido a problemas con la implementación (la herramienta no utiliza los *flags* especificados) o con el paquete de la solución (no sigue la estructura especificada), el entorno de evaluación no es capaz de ejecutar vuestra práctica, esto supondrá un 0 de la parte de evaluación experimental.