Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-5437 - Inteligencia Artificial I
Abril - Julio 2021

Proyecto 3: Satisfacibilidad Booleana

Jesús De Aguiar 15-10360
Neil Villamizar 15-11523
Jesús Wahrman 15-11540

Problema

El objetivo de este proyecto es aprender a modelar un problema en CNF, y a usar un SAT solver para resolverlo, así como traducir la salida del SAT solver a un formato legible, otorgándole importancia al rendimiento en las transformaciones y modelados.

Solución Planteada

La solución planteada fue implementada en cuatro módulos de python que se encuentran en el repositorio del proyecto. Además, se implementó un script que encadena las distintas funcionalidades

Modelado del problema

Para modelar el problema, en primer lugar se ejecuta un preprocesamiento de los parámetros. En este punto se leen los parámetros de la entrada y se calculan la cantidad de días hábiles donde pueden ocurrir partidos y la cantidad de bloques en los cuales puede ocurrir un partido, haciendo cumplir las reglas de que un partido debe comenzar a en una hora "en punto" y de que un partido debe durar dos horas. Teniendo estos dos parámetros, la construcción de las cláusulas en el SAT que modela a este problema se realiza de una manera más sencilla. En caso de que los parámetros dados impidan que haya partidos, entonces se reporta como entrada inválida.

Modelado del SAT

Para modelar el problema como un problema SAT, en primer lugar se definen los literales que serán parte de nuestras cláusulas de la siguiente forma:

$$x(i, j, d, b) = \text{El equipo } i \text{ juega como local vs el equipo } j \text{ (visitante) el día } d \text{ en el bloque } b$$

Por lo tanto, el problema se reduce a encontrar cuáles de esos literales tendrán valor *true* al final de la ejecución del SAT solver.

Para hacer cumplir las reglas del torneo, se modelan las siguientes reglas:

Todos vs Todos. Todos los participantes deben jugar dos veces con cada uno de los otros participantes, una como "visitantes" y la otra como "locales". Para modelar esta restricción, se crearon los siguientes tipos de cláusulas:

- En primer lugar, se estableció que dos equipos deben jugar al menos una vez. De esta manera, para cada par de equipos i y j, se crearon las cláusulas compuestas por los literales x(i, j, d, b) donde d y b toman todas las posibles combinaciones de valores de días y horarios.
- Seguidamente, se estableció que dos equipos deben jugar a lo sumo una vez. Para ello, para cada par de equipos i y j, se crearon las clausulas $(\neg x(i, j, d, b) \vee \neg x(i, j, d', b'))$ para cualquier combinación día y bloque d' b' distintas a las originales. De esta manera, se modela la implicación de que si un equipo juega contra otro en algún día y en determinado bloque, entonces no juega con el mismo equipo en algún otro día en cualquier otro bloque.

Dos juegos no pueden ocurrir al mismo tiempo. Para modelar esta restricción, se generaron las implicaciones que indican que si dos equipos juegan en un día determinado y hora determinada, entonces para cualquier otro par de equipos, ellos no juegan el mismo día y a la misma hora. Las cláusulas lucirían de la forma $(\neg x(i, j, d, b) \vee \neg x(i', j', d, b))$ para cualquier para de equipos i,j y cualquier otro par de equipos i',j'

Una vez al día. Se debe cumplir que un participante puede jugar a lo sumo una vez por día. Para esto, se generaron las implicaciones que indican que si dos equipos juegan en un día determinado y hora determinada, entonces para cualquier otro tercer equipo se debe cumplir que tanto el equipo local como el equipo visitante no puede jugar el mismo día en contra de este tercer equipo (de local o visitante). Adicionalmente, se agregó la implicación de que estos dos mismos equipos no podían jugar el mismo día cambiando la localía.

Cambio de localía. Finalmente, se debe cumplir que un participante no puede jugar de "visitante" en dos días consecutivos, ni de "local" dos días seguidos. Para modelar esto, se construyeron las cláusulas que representan la implicación donde si un par de equipos juega a un día determinado y bloque determinado, entonces se cumple que el local no juega como local para cualquier otro equipo y cualquier otro bloque en el día siguiente (y lo mismo para el visitante). De esta manera, se modelarían las clausulas de la siguiente manera: $(\neg x(i, j, d, b) \vee \neg x(i, k, d+1, b'))$ y $(\neg x(i, j, d, b) \vee \neg x(k, j, d+1, b'))$ para cualquier otro equipo k.

Implementación

El proyecto fue implementado en Python. Los módulos que implementan la solución se encuentran en el repositorio del proyecto.

Para ejecutar el programa, se debe instalar inicialmente el paquete ics, el cuál se utilizará para

la creación de los archivos en formato iCalendar. El paquete está disponible utilizando PyPI y para instalarlo es suficiente con ejecutar en un ambiente virtual:

```
@> pip install -r requirements.txt
```

Para generar el ejecutable de glucose, es suficiente con ejecutar en el directorio raíz del respositorio:

```
@> make
```

Seguidamente, para ejecutar la generación del archivo que contiene el modelo del problema SAT en formato DIMACS CNF, es suficiente con ejecutar:

```
@> python main.py --sat <input-json-file> <output-cnf-file>
```

Donde `<input-json-file>` es la ruta del archivo de entrada y `<output-cnf-file>` es la ruta del archivo de salida.

Por otro lado, si desea ejecutar el procedimiento completo, generando el archivo .ics en formato ICalendar, es suficiente con ejecutar:

```
@> python main.py --ical <input-json-file> <output-Ical-file>
```

Donde `<input-json-file>` es la ruta del archivo de entrada y `<output-Ical-file>` es la ruta del archivo de salida.

Los módulos que implementas las funcionalidades son los siguientes:

- `preprocess.py` implementa las funciones para preprocesar la entrada y calcular el número de bloques y días
- `sat_generator.py` implementa la generación de las cláusulas y el archivo en formato DIMACS CNF. Una de las prioridades de este código es evitar la repetición de cláusulas ya que se genera un número importante de estas y queremos evitar afectar el rendimiento en la ejecución al agregar duplicados.
- `postprocess.py` implementa el procesamiento de la salida del SAT solver luego que el problema fue resuelto
- `ical_generator.py` implementa la creación del archivo .ics en formato ICal donde se encuentra la programación de los eventos. Para la creación de este módilo, se utilizó el paquete ics.
- `main.py` contiene el flujo principal de ejecución, enlazando las distintas funcionalidades de los distintos módulos y, al mismo tiempo, haciendo uso del SAT solver Glucose.

Tests y Resultados

Se diseñó una serie de experimentos donde se ejecutó el programa con una serie de entradas, algunas con posible solución y otra sin ellas. A continuación se muestra una tabla de comparación de la entrada y el rendimiento para cada uno de los casos ejecutados:

Experimento	Participantes	Días	Bloques	Número de Clausulas	Tiempo total de ejecución(s)	Tiempo de Ejecución Glucose (s)	Solución Encontrada
1	7	15	5	754467	7.36	2.85	Sí
2	7	10	4	371742	43	40.4	No
3	9	20	5	221479	22.02	20.74	Sí
4	9	10	4	646632	35.19	31.49	No
5	9	30	12	22685256	33.97	30.45	Sí

Table 1: Resultados obtenidos por los algoritmos tras recorrer el árbol del juego desde la profundidad 12

Podemos darnos cuenta que el problema el modelado del problema genera una cantidad importante de cláusulas incluso para entradas que representan torneos pequeños. La mayor parte del tiempo de ejecución total se emplea en resolver el problema SAT asociado, incluso con un resolvedor tan eficiente como Glucose, y que la generación de un número tan grande de cláusulas es mucho menor en tiempo. Aún así, este es capaz de calcular una respuesta correcta en un tiempo aceptablemente corto pese a dicha cantidad de cláusulas.

Resolver este tipo de problema de scheduling con un SAT solver nos puede otorgar una respuesta correcta en un tiempo corto para torneos donde el número de días, horas y participantes involucrados es bajo. Un problema que se encontró es que al aumentar estos valores (al triplicar los valores actuales, por ejemplo) la cantidad de cláusulas generadas era tan grande que no se poseían los recursos de memoria para ser capaces de cargar el problema en el resolvedor, por lo cuál no se recomienda utilizar este programa para torneos con muchos participantes o con una cantidad grande de días y horas donde los partidos pueden ocurrir.