Daniel Robayo 18-11086
Valeria Vera 16-11233

Informe Proyecto III - CI5437

1. Introducción

En este informe se presentan los detalles de la implementación y los resultados obtenidos del proyecto. En este proyecto se pidió utilizar un SAT-SOLVER para resolver un problema de planificación de un torneo tal que se cumplieran ciertas restricciones. El proyecto consta de 3 partes. La primera es la traducción de las restricciones al lenguaje CNF. La segunda es la llamada al SAT-SOLVER y la última es la creación de un archivo iCalendar que contuviera la planificación del torneo.

Se usó el lenguaje de programación Python 3.10.6 y el SAT-SOLVER usado fue Glucose 4.2.1. Para la representación de las restricciones y la traducción de estas a CNF nos basamos en los artículos que se encuentran en la carpeta *papers* del repositorio. Por último, se entrega un cliente el cual se encarga de la ejecución de las distintas partes.

2. Detalles de la implementación.

a. Representación.

Para la representación de las restricciones se define el predicado X(j1, j2, d, h) que se traduce a que el jugador *j1* (local) juega contra el jugador *j2* (visitante) el día *d* en la hora *h*. Esté predicado toma valores 1 y 0 que se traducen a True y False. Dado un juego de N jugadores, D dias y S juegos disponibles por dia, se tienen N*(N-1)*D*H posibles juegos que son nuestros literales. Se usó la notación de enteros para definir las restricciones. La definición de cada restricción es la siguiente:

- **Todos los participantes deben jugar dos veces con cada uno de los otros participantes, una como "visitantes" y la otra como "locales":**

Para los jugadores j1 y j2 (j1 y j2 son distintos) la suma de las veces que juega j1 como local contra j2 como visitante durante todo el torneo es igual a 1.

$$(\forall\, j1,\ j2 |\ j1 != j2 \colon \sum_{d,\ h} X(j1,\ j2,\ d,\ h)\ = 1)$$

- **Dos juegos no pueden ocurrir al mismo tiempo**.

Para cada par (día, hora) la suma de juegos que suceden en ese momento es menor o igual a 1. Se toma que la suma es menor o igual y no igual ya que puede haber un momento del torneo en donde no se esté jugando algún juego.

$$(\forall\, d, h | \colon \sum_{j1,\, j2\ (j1\ != j2)} X(j1,\ j2,\ d,\ h)\ <= 1)$$

- **Un participante puede jugar a lo sumo una vez por día**.

Para un jugador j1 y un día d, la suma de los juegos de ese jugador en ese día debe ser menor o igual a 1.

$$(\forall\, j1,\ d|: \sum_{j2,\, h\ (j1\ !=\ j2)} X(j1,\ j2,\ d,\ h)\ +\ \sum_{j2,\, h\ (j1\ !=\ j2)} X(j2,\ j1,\ d,\ h)\ <=\ 1)$$

- **Un participante no puede jugar de “visitante” en dos días consecutivos, ni de “local” dos días seguidos.**

Esta restricción consta de dos partes, una es un jugador siendo local y otra siendo visitante. En el caso de local, para un jugador j1 y un dia d, la suma de juegos jugador por ese jugador como local en ese dia mas la suma de juegos de ese jugador como local el día siguiente es menor o igual a 1. Para el caso de visitante es análogo

$$(\forall\, j1,\ d|\ d\ <\ numeroDias: \sum_{j2,\, h\ (j1\ !=\ j2)} X(j1,\ j2,\ d,\ h)\ +\ \sum_{j2,\, h\ (j1\ !=\ j2)} X(j1,\ j2,\ d\ +\ 1,\ h)\ <=\ 1))$$

- **Todos los juegos deben ocurrir entre una fecha inicial y una fecha final especificadas. Pueden ocurrir juegos en dichas fechas.**

Para obtener los días de juego, se hizo ∆(end_date - start_date) + 1, puesto que se debía garantizar que se jugara también en el día inicial.

- **Todos los juegos deben ocurrir entre un rango de horas especificado, el cuál será fijo para todos los días del torneo.**

En el caso del manejo de horas, también se realizó ∆(end_time - start_time) para conocer las horas disponibles que se tenían por día de juego.

- **Todos los juegos deben empezar en horas "en punto".**

Se creó la función *o_clock()* que permite ver si la hora estipulada en el .json contiene minutos, segundos o microsegundos distintos a cero y adelantar/atrasar una hora según el caso.

- **A efectos prácticos, todos los juegos tienen una duración de dos horas.**

Se utilizaron *slots* o espacios de tiempo que consistió en dividir las horas totales entre 2 para así conocer la cantidad de espacios de tiempo con los que se contaba en el día. De esta manera, al generar las restricciones, cada hora h de las variables corresponde a la hora en la que inicia uno de los *slots*.

3. Complejidad en tiempo

El programa tiene una complejidad de tiempo que se ve afectada de dos maneras. Una parte es la traducción de las restricciones a CNF y la ejecución de Glucose. Con respecto a la traducción de las restricciones tenemos lo siguiente:

- Se anidan 4 ciclos FOR que dan una total de N*(N-1)*D*S iteraciones, siendo N la cantidad de jugadores, D la cantidad de días y S la cantidad de juegos disponibles en cada día.
- En los segundos FOR anidados se usa una función que se encarga de crear subconjuntos de cierto tamaño. El conjunto en el que se obtienen los subconjuntos es de nuestro conjunto que tiene todas los posibles juegos posibles. Esta operación tiene un tiempo de ejecución de a lo sumo $O(2^M)$, siendo M la cantidad de juegos posibles.

- Aplicando propiedades de la notación O, podemos concluir que el tiempo de ejecución de la creación de las restricciones es $O(2^M)$ en promedio. Esto es así ya que el orden en el que se anidan los ciclos varía.

Luego, en el peor de los casos el SAT-SOLVER tiene un tiempo de ejecución igual a $O(2^M)$ siendo M el número de literales del problema. Como se tiene tantos literales como juegos disponibles, el tiempo de ejecución del cliente es igual a $O(2^{N*(N-1)*D*S})$.

4. Pruebas realizadas

Las pruebas fueron realizadas en los siguientes computadores:

- Computadora 1.
  - Intel i3 10ma generación.
  - 8 Gb de RAM.
  - WSL Ubuntu 22.04.2
- Computadora 2.
  - Intel i3 6ta generación.
  - 12 Gb de RAM.
  - Ubuntu 22.04.

Para estas pruebas, se optó por dejar fijos dos cantidades de jugadores (4 y 7 respectivamente) y analizar los cambios en el número de variables y cláusulas así como del tiempo de ejecución de creación de restricciones y de las llamadas al solver Glucose.

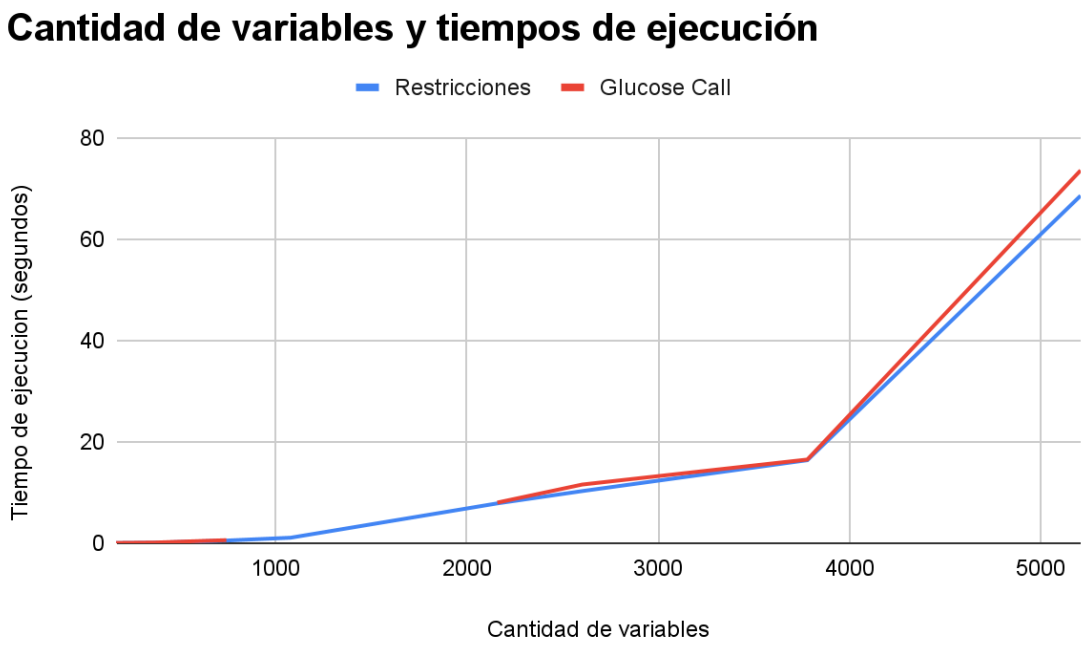
La cantidad de días que se decidió utilizar en ambos casos constan de un mes y dos meses. Sin embargo, en el caso de Test_1_1, se optó por realizar el torneo durante una semana ya que era la cantidad mínima que se necesitaba en un torneo de cuatro jugadores para poder obtener un resultado satisfactorio.

En esta suite de pruebas, para todos los problemas se encontraron soluciones.

<table>
<tr><td colspan="8">Pruebas Realizadas</td></tr>
<tr><td colspan="4">VARIABLES DE LA PRUEBA</td><td colspan="2">DIMACS CNF</td><td colspan="2">AVG EXEC TIME</td></tr>
<tr><td>Nombre Test</td><td>No. Jugadores</td><td>No. Horas diarias</td><td>No. días</td><td>Variables</td><td>Cláusulas</td><td>Restricciones</td><td>Glucose Call</td></tr>
<tr><td>Test_1_1</td><td rowspan="5">4</td><td rowspan="2">4</td><td>7</td><td>168</td><td>7044</td><td>0,03</td><td>0,05</td></tr>
<tr><td>Test_1_2</td><td rowspan="2">31</td><td>744</td><td>50820</td><td>0,516</td><td>0,6</td></tr>
<tr><td>Test_1_3</td><td rowspan="2">1</td><td>372</td><td>13098</td><td>0,09</td><td>0,13</td></tr>
<tr><td>Test_1_4</td><td rowspan="2">92</td><td>1080</td><td>70092</td><td>1,08</td><td>1.16</td></tr>
<tr><td>Test_1_5</td><td>4</td><td>2160</td><td>275964</td><td>7,85</td><td>8,01</td></tr>
<tr><td>Test_2_1</td><td rowspan="3">7</td><td>4</td><td rowspan="2">31</td><td>2604</td><td>308658</td><td>10,28</td><td>11,57</td></tr>
<tr><td>Test_2_2</td><td>8</td><td>5208</td><td>1145634</td><td>68,56</td><td>73,55</td></tr>
<tr><td>Test_2_3</td><td>2</td><td>92</td><td>3780</td><td>369558</td><td>16,37</td><td>16,5</td></tr>
</table>

De esta tabla se visualiza que, al tener dos torneos con el mismo número de días, aquellos que tengan mayor número de horas diarias tomarán mayor tiempo en ejecutarse que los que tienen un tiempo diario reducido. A su vez y como era de esperarse, mientras menos días para realizarse se tengan, más rápida será la ejecución de las creación de restricciones y de Glucose.

Un caso curioso en particular es el de Test_2_2, ya que es el que tiene mayor número de cláusulas y variables así como más grandes tiempos de ejecución en promedio. Sin embargo, esto no significa que sea la prueba con mayor número de días puesto que ocurre en un mes mientras que Test_2_3 tiene 3 meses para desarrollarse.



En el primer gráfico, podemos observar que los tiempos de ejecución tanto de la llamada a Glucose como de la creación de las restricciones crecen de una manera casi lineal con respecto al aumento de la cantidad de variables en el .dimacs. Asimismo, los tiempos de ejecución de ambos son casi idénticos, con poca variación entre ellos.

Por el segundo gráfico, notamos que la cantidad de cláusulas generadas con respecto a las variables generadas tiene un crecimiento también similar al del gráfico 1. Es decir, que mientras existen más variables, también existen más cláusulas.

5. Conclusiones

Se concluye que los resultados de tiempo se corresponden a lo esperado por la teoría y crear las restricciones tiene un tiempo de ejecución cercano o igual a la ejecución del SAT-SOLVER. Además, se pudo observar cómo pequeños cambios en las condiciones del torneo como la extensión de horas o más largos plazos de días inducen cambios considerables en el tiempo de ejecución del programa.

En esté proyecto aprendimos sobre la representación de problemas usando lógica y se pudo observar la eficiencia del SAT-SOLVER escogido.