Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5437 - Inteligencia Artificial I
Ene-Mar 2024
Prof. Carlos Infante

Gabriela Panqueva 18-10761
Daniela Ramírez 16-10940

Informe Proyecto 3

1. Introducción

Este informe detalla la implementación y los resultados del proyecto que implicó el uso de un SAT-SOLVER para planificar un torneo, cumpliendo con ciertas restricciones. El proyecto se dividió en tres fases. La primera fase consistió en la conversión de las restricciones a la forma normal conjuntiva (CNF). La segunda fase implicó la ejecución del SAT-SOLVER. Finalmente, la tercera fase consistió en la generación de un archivo iCalendar con la programación del torneo. Para la implementación del proyecto, se utilizó Python 3.10.6 y el SAT-SOLVER Glucose 4.2.1.

2. Especificaciones del equipo usado

AMD Ryzen 7 5800H 3.20 GHz, 40,0 GB (39,4 GB usable) de RAM y se ejecutaron en una WSL2 de Ubuntu 22.04.3 LTS.

3. Detalles de implementación

3.1 Representación

Para el modelado del problema se definió las siguientes variables y restricciones:

$X_{i,j,d,h}$: Representa cuando el jugador ***i*** juega de local contra el jugador ***j*** que juega de visitante el día ***d*** a la hora ***h***. Esta variable puede tomar valores booleanos, es decir, True y False.

Dado un juego de N jugadores, D días y H horas, se tiene que *i* y *j* pertenecen al conjunto de los N jugadores, *d* al conjunto de los D días y *h* al conjunto de las H horas. Así pues, se tienen $N * (N - 1) * D * H$ posibles juegos.

Usando lógica proposicional se definen las restricciones de la siguiente manera:

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

$$(\forall i, j \mid i, j \in N \ \wedge \ i \ \neq j : (\exists!_1 \, d, h \mid d \in D \ \wedge h \in H : X_{i,j,d,h})$$

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

$$(\forall i, j, d, h \mid i \neq j : X_{i,j,d,h} \Rightarrow \neg(\exists \ n, m \mid (i \neq n \ \vee j \neq m) \wedge n \neq m : X_{n,m,d,h})) \wedge$$
$$(\forall i, j, d, h \mid i \neq j \wedge h < H - 2 : X_{i,j,d,h} \Rightarrow \neg(\exists \ n, m \mid (i \neq n \ \vee j \neq m) \wedge n \neq m : X_{n,m,d,h+1}))$$

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

$$(\forall i, j, d, h \mid i \neq j : X_{i,j,d,h} \ \Rightarrow \ \neg(\exists \, k, l \mid k \in N \wedge l \in H \wedge \ l \neq h : X_{i,k,d,l} \vee X_{k,j,d,l} \vee X_{j,k,d,l} \vee X_{k,i,d,l}))$$

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

$$(\forall i, j, d, h \mid i \neq j \ \wedge \ d < D - 1 : X_{i,j,d,h} \ \Rightarrow \ \neg(\exists \, k, l \mid k \in N \wedge l \in H : X_{i,k,d+1,l} \vee X_{k,j,d+1,l}))$$

- **Todos los juegos deben empezar en horas "en punto" (por ejemplo, las 13:00:00 es una hora válida pero las 13:30:00 no).**

  El programa genera las horas intermedias en incrementos de dos horas, comenzando desde la hora de inicio hasta la hora de finalización especificada. Esto asegura que todos los juegos comiencen en horas “en punto”.

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

  El programa genera todas las fechas desde la fecha de inicio hasta la fecha de finalización. Esto asegura que todos los juegos ocurran dentro del rango de fechas especificado.

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

El programa genera las horas intermedias en incrementos de dos horas, comenzando desde la hora de inicio hasta la hora de finalización especificada. Esto asegura que todos los juegos ocurran dentro del rango de horas especificado.

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

  Como el programa genera las horas intermedias en incrementos de dos horas, esto implica que cada juego tiene una duración de dos horas.

4. Ejecución del programa

Para poner en marcha el proyecto, es necesario seguir una serie de pasos que garantizan la correcta instalación de las herramientas necesarias y la ejecución del programa. A continuación, se detallan estos pasos:

Actualización del sistema: Antes de comenzar la instalación de cualquier software, es recomendable actualizar el sistema. Esto se puede hacer con el siguiente comando:

sudo apt-get update

Instalación de Python: El lenguaje de programación utilizado para este proyecto es Python 3.10.6. Si no está instalado en tu sistema, puedes hacerlo con el siguiente comando:

sudo apt-get install python3

Instalación de pip: Pip es un gestor de paquetes para Python, que se utilizará para instalar las dependencias del proyecto. Si no está instalado en tu sistema, puedes hacerlo con el siguiente comando:

sudo apt install pip

Instalación de las dependencias del proyecto: Este proyecto requiere de ciertos paquetes de Python para su correcta ejecución. Estos paquetes están listados en el archivo requirements.txt. Para instalarlos, puedes utilizar el siguiente comando:

python3 -m pip install -r requirements.txt

Ejecución del programa: Finalmente, para poner en marcha el programa, debes ejecutar el archivo main.py con el siguiente comando:

python3 main.py

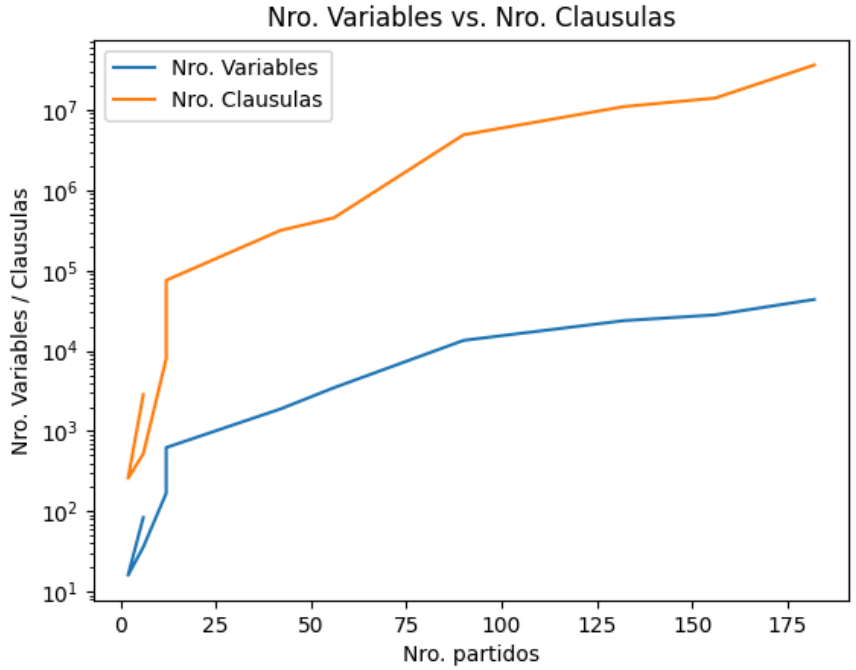
Una vez iniciado el programa, se te solicitará que introduzcas el nombre del archivo de prueba, el cual debe estar en formato JSON.

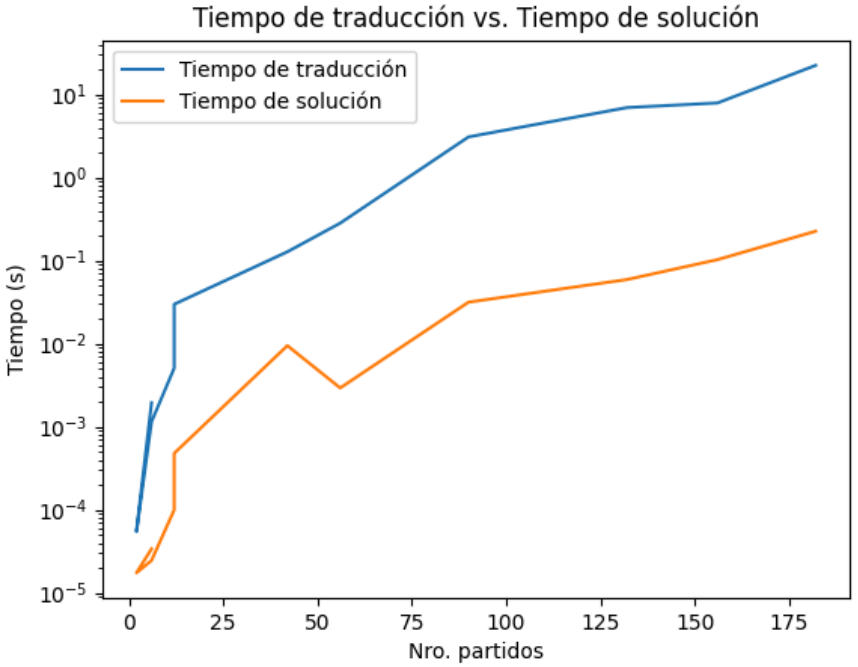
5. Resultados

Se generaron pruebas de diversa complejidad, tanto sencillas como desafiantes, con el objetivo de verificar la correcta operación del programa. Los casos de prueba se encuentran en la carpeta “Tests” del repositorio.

Test	Partidos	Variables	Cláusulas	Tiempo traducción del JSON	Tiempo en la creación de cláusulas	Tiempo del solver	Solución
0	6	84	2850	0.001431	0.000546	3.385543	Si
1	2	16	258	0.001348	5.459785	1.740455	Si
2	6	36	522	0.001942	0.000148	2.455711	Si
3	12	168	7956	0.002553	0.002473	9.965896	Si
4	12	624	75612	0.002410	0.027798	0.000478	Si
5	42	1890	316806	0.001930	0.127596	0.009463	Si
6	56	3472	454104	0.002411	0.280309	0.002928	Si
7	90	13500	4913010	0.004690	3.080974	0.031586	Si
8	132	23760	11022396	0.009758	6.748533	0.059098	Si
9	156	28080	14090700	0.006729	7.883396	0.102627	Si
10	182	43680	36389990	0.007688	22.379068	0.225556	Si
11	-	-	-	0.030128	-	-	No (killed)
12	-	-	-	-	-	-	No (killed)
14	-	5400	2027790	0.010266	1.054067	-	No (killed)
15	-	9450	4985910	0.009772	2.635415	-	No (killed)
16	-	8100	3304890	0.009205	1.761756	-	No (killed)
17	-	16200	12360690	0.011407	6.216549	-	No (killed)

A continuación, se muestran unas gráficas para poder analizar los resultados obtenidos.





Según los resultados mostrados en la tabla previa, la implementación de la conversión a CNF de las instancias demuestra ser altamente eficiente y no constituye un obstáculo para el rendimiento del programa.

Además, se puede observar que la cantidad de partidos, variables y cláusulas aumenta con la complejidad del problema. A medida que aumenta el número de partidos, también lo

hacen las variables y las cláusulas, lo que indica que se están considerando más escenarios posibles.

Con respecto al tiempo de creación de las cláusulas, se puede ver que aumenta con la complejidad de los tests. Esto tiene sentido ya que a medida que aumenta el número de variables y cláusulas, también lo hace el tiempo necesario para crear estas cláusulas.

El tiempo del solver también aumenta con la complejidad del problema, pero no de manera tan pronunciada como el tiempo de creación de cláusulas. Esto podría indicar que el solver es bastante eficiente, incluso para problemas más complejos.

Sin embargo, a partir del test 11, no fue posible hallarle solución a los tests proporcionados, ya que el sistema operativo mataba el proceso.

Los escenarios que pueden presentar mayor desafío para el SAT solver son aquellos en los que, a pesar de cumplir con las condiciones de coherencia, no existe una solución. En estos casos, es probable que el solver deba realizar una exploración más detallada del espacio de búsqueda para confirmar la ausencia de una solución.

6. Conclusiones

Cualquier problema que se resuelve con SAT enfrenta dos desafíos principales: la creación de la fórmula en CNF y la demostración de satisfacibilidad. Aunque el segundo desafío ha sido en gran medida superado gracias a la eficiencia de los SAT Solvers, como Glucose, el primer desafío aún recae en el desarrollador.

Además, es importante destacar que la eficacia de la solución depende en gran medida de cómo se plantea el problema en CNF. Un planteamiento eficiente puede reducir significativamente el tiempo de resolución. Por lo tanto, aunque los SAT Solvers han avanzado mucho, la habilidad y la creatividad del desarrollador siguen siendo fundamentales para resolver problemas con SAT de manera eficiente. Esta dualidad entre la herramienta y el usuario subraya la importancia de una comprensión profunda tanto del problema como de la herramienta utilizada para resolverlo.