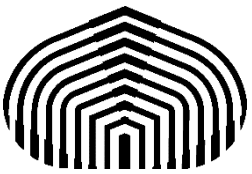


Universidad Simón Bolívar

CI-5437 Inteligencia Artificial

Prof. Carlos Infante

INFORME DEL PROYECTO 3

Simon Puyosa 18-10717

Ana Shek 19-10096

Juan Cuevas 19-10056

Caracas, Marzo del 2024.

Tabla de Contenido

Introducción

En el ámbito de la investigación operativa y la ciencia de la computación, la capacidad de modelar y resolver problemas complejos es fundamental. Este informe describe el enfoque para abordar un desafío específico: la modelación de un problema de programación de torneos en formato CNF y la utilización de un SAT solver (Glucose) para encontrar soluciones viables.

Este proyecto se centra en la organización de un torneo con restricciones específicas de programación. El objetivo es asignar fechas y horas para los juegos de manera eficiente, asegurando el cumplimiento de todas las reglas establecidas.

La complejidad de este problema radica en la necesidad de considerar múltiples factores simultáneamente, como los horarios de los equipos participantes y las reglas específicas del torneo. Además, es crucial garantizar que no se produzcan conflictos entre los eventos programados, lo que puede ser un desafío en sí mismo. Al modelar este problema como una instancia de SAT, podemos aprovechar la potencia y eficiencia de los solvers modernos para encontrar soluciones factibles en un tiempo razonable.

En este sentido, nuestro enfoque para este proyecto implica una cuidadosa traducción de las restricciones del problema a una fórmula lógica en formato CNF (Forma Normal Conjuntiva). Esta fórmula representa todas las condiciones que deben cumplirse para que una solución sea válida. Una vez que la instancia de SAT está lista, se utiliza el solver Glucose para encontrar una asignación de valores que satisfaga la fórmula. Esta asignación representa una solución factible para el problema de programación del torneo.

Los experimentos fueron realizados en la siguiente plataforma:

- Una computadora con procesador Intel Core i7-8700, disco SSD, 16GB de memoria RAM y Linux Mint 21.1 x86_64
- Un laptop con procesador AMD Ryzen 3 3200U, disco SSD, 20GB de memoria RAM y WSL 2 Ubuntu.
- Un laptop con procesador AMD Ryzen 5 5500U, disco SSD, 8GB de memoria RAM y WSL 2 Ubuntu.

Modelación del problema en CNF

Variables

Definiremos la siguiente variable J_{abdh}

J_{abdh} indica "El participante a juega como local contra el participante b que juega como visitante en el día d a la hora de inicio h".

Tenemos que $J_{abdh} \in True,\ False$.

Los participantes los nombraremos con un entero de 0 a n-1, el cual, $a, b \in [0,\ n)$, $donde\ n \geq 2$. Deben existir al menos dos equipos para que compitan.

Para t, esta indica si el participante juega como "local" o como "visitado". Para ello, se define $t = 0$ si es local, y $t = 1$ si es visitante. Por lo tanto $t \in [0,\ 1]$

Los días d también lo vamos a enumerar con un entero: el primer día del torneo será $d = 0$, el segundo día será $d = 1$, y así sucesivamente. Sea LastDay el número que le toca al último día del torneo tenemos que $d \in [0,\ LastDay)$.

De igual manera, las horas de inicio h estarán numeradas con un entero, donde $h = 0$ será la hora de inicio de cada juego para cada día, $h = 1$ corresponde a la próxima hora, y así sucesivamente. Sea LastHour el número que le toca a la última hora para cada día de torneo, tenemos que $h \in [0,\ LastHour)$.

Sin embargo, también tenemos que: Todos los juegos tienen una duración de dos horas. Entonces el último juego para cualquier día no puede comenzar a la hora de LastHour, sino a más tardar a la hora LastHour- 1. Entonces $h \in [0,\ LastHour - 1)$

De esto es directamente deducible que:

$$|J_{atdh}| = n^2 \times LastDay \times (LastHour - 1)$$

Restricciones

Ahora, vamos a modelar cada restricción en lógica proposicional:

- Todos los participantes deben jugar dos veces con cada uno de los otros participantes, una como "visitantes" y la otra como "locales". Esto significa que, si hay 10 equipos, cada equipo jugará 18 veces.

  Para traducirlo en lógica proposicional, vamos a traducirlo como: Para cada participante a y b tal que a !=b, se tiene que existe un día d y hora h en que a juega como local contra b que juega como visitante.

$(\forall a,\ b \mid a\ \neq\ b : (\exists\ d,\ h|: J_{abdh}))$

La cantidad de cláusulas para esta restricción es n*(n-1)

- Dos juegos no pueden ocurrir al mismo tiempo.

  Esta restricción lo vamos a modelar como: Para cada partido que ocurra en un mismo día, si uno comienza a la hora h1, entonces no puede haber otro que comience a la hora h1 o a la h1+1 (Esto porque todos los juegos duran dos horas, y por lo tanto, el otro partido debe comenzar al menos dos horas después para empezar).

$(\forall a, b, d, h_1 \mid a \neq b \wedge J_{abdh_1} : \neg(\exists x, y,\ h_2 | x \neq y \wedge (a \neq x \vee b \neq y) \wedge (h_2 = h_1 \vee h_2 = h_1 + 1) : J_{xydh_2}))$

Esto es equivalente a que

$(\forall a, b, x, y, d, h_1, h_2 | a \neq b \wedge x \neq y \wedge (a \neq x \vee b \neq y) \wedge (h2 = h1 \vee h2 = h1 + 1) : \neg J_{abdh_1} \vee \neg J_{xydh_2})$

Esta restricción tiene

$n * (n-1)^2 * LastDay * LastHour * (n - (1/(n-1))) * (2 - 1/LastHour)$ cláusulas.

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

Para esta restricción, necesitamos definir que LastDay $\geq$ 2, ya que aparte de que cada participante debe jugar como “visitante” y como “local”.

Ahora, esta restricción lo modelamos como: Para cada participante que juega como local contra b que juega como visitante en un día d y en hora h1, no existe otro partido tal que esté alguno de estos dos participantes y tenga el mismo día d a la hora h2.

$$(\forall a, b, d, h_1 | \ a \neq b \ \wedge J_{abdh_1} : \neg(\exists c, h_2 \mid h_1 \neq h_2 : \ J_{cadh_2} \vee \ J_{acdh_2} \ \vee J_{cbdh_2} \vee \ J_{bcdh_2}))$$

Esto es

$$(\forall a, b, c, d, h_1, h_2 | \ a \neq b \ \wedge h_1 \neq h_2 : \neg J_{abdh_1} \vee \ \neg(J_{cadh_2} \vee \ J_{acdh_2} \ \vee J_{cbdh_2} \vee \ J_{bcdh_2}))$$

Luego

$$(\forall a, b, c, d, h_1, h_2 | \ a \neq b \ \wedge h_1 \neq h_2 : \neg J_{abdh_1} \vee (\neg J_{cadh_2} \wedge \neg J_{acdh_2} \ \wedge \ \neg J_{cbdh_2} \wedge \neg J_{bcdh_2}))$$

Entonces

$$(\forall a, b, c, d, h_1, h_2 | \ a \neq b \ \wedge h_1 \neq h_2 : (\neg J_{abdh_1} \vee \neg J_{cadh_2}) \wedge (\neg J_{abdh_1} \vee \neg J_{acdh_2}) \ \wedge (\neg J_{abdh_1} \vee \neg J_{cbdh_2}) \wedge (\neg J_{abdh_1} \vee \neg J_{bcdh_2}))$$

Esta restricción tiene $4 * n^2 * (n - 1) * LastDay * LastHour * (LastHour - 1)$ cláusulas.

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

  La traducción lo haremos como: Para cada participante que juegue como local contra otro participante b en un día d y hora h1, no existe otro partido con el participante a que juegue como local en un día d + 1 y hora h2, donde h2 puede ser igual a h1 o no. Tampoco existe otro partido con el participante b que juegue como visitante en un día d +1 y hora h2.

$$(\forall a, \ b, d, h_1 | \ a \neq b \ \wedge J_{abdh_1} : \neg(\exists \ c, h_2 \mid : \ J_{ac(d+1)h_2} \vee J_{cb(d+1)h_2}))$$

Observemos que a esta restricción, en el cuantificador existencial le falta que c sea distinto de a en caso de que exista un partido donde juegue a como local contra c en el día siguiente o que c sea distinto de b y exista un partido donde juegue c contra b que juega como visitante. Para ello, vamos a agregarle una nueva restricción: *Cada participante no puede jugar contra sí mismo:* $(\forall a, d, h | : \neg(\exists J_{aadh}))$, el cual va a tener $n * LastDay * LastHour$ cláusulas.

Ahora bien, $(\forall a, \ b, d, h_1 | \ a \neq b \ \wedge J_{abdh_1} : \neg(\exists \ c, h_2 \mid : \ J_{ac(d+1)h_2} \vee J_{cb(d+1)h_2}))$, esto es equivalente a que

$$(\forall a,\ b, c, d, h_1, h_2 |\ a \neq b: \neg J_{abdh_1} \vee \neg(J_{ac(d+1)h_2} \vee J_{cb(d+1)h_2}))$$

Luego

$$(\forall a,\ b, c, d, h_1, h_2 |\ a \neq b: \neg J_{abdh_1} \vee (\neg J_{ac(d+1)h_2} \wedge \neg J_{cb(d+1)h_2}))$$

Finalmente

$$(\forall a,\ b, c, d, h_1, h_2 |\ a \neq b: (\neg J_{abdh_1} \vee \neg J_{ac(d+1)h_2}) \wedge (\neg J_{abdh_1} \vee \neg J_{cb(d+1)h_2}))$$

Esta restricción tiene $2 * n^2 * (n - 1) * (LastDay - 1) * Lasthour^2$ cláusulas

- 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).

  La traducción de esta restricción en lógica viene gratis por el dominio de h, el cual, se definió h como un entero.

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

  La manera en que se modela el problema ya garantiza que todos los juegos ocurran entre una fecha inicial y final especificada, y que puedan ocurrir varios juegos en dichas fechas dependiendo de las restricciones anteriores.

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

  Así mismo, el modelaje del problema ya asegura esta restricción.

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

  Ya se consideró esta restricción al momento de definir las variables y las restricciones anteriores.

Ahora, sea d = LastDay y h = LastHour

En total, tenemos $-n^3h^2 - n^2h^2 + 2dn^4h - 4dn^3h + 4dn^2h^2$

$-\ 4dn^2h\ +\ 3dnh\ -\ dn^4\ +\ 2dn^3\ +\ n^2\ -\ n\ -\ dn$ cláusulas.

Implementación de la solución

Se eligió Python como el lenguaje de programación para este proyecto debido a su extensa biblioteca estándar, el cual proporciona una amplia gama de módulos y funcionalidades listos para usar.

La implementación de la solución al problema consiste en obtener primero el nombre del archivo JSON a través de los argumentos de línea de comandos. Este archivo JSON contiene la información necesaria para planificar el torneo.

El siguiente paso sería intentar abrir y leer el archivo para cargar los datos en unas variables. Si el archivo no existe o hay un problema con el formato de los datos JSON, se muestra un mensaje de error y se sale del programa. Si no ocurre ningún problema, se guardan los datos del torneo en variables individuales, como el nombre del torneo, la fecha de inicio y finalización, la hora de inicio y finalización de los juegos, y la lista de participantes.

A continuación, se calcula la cantidad de participantes *n*, obteniendo la longitud de la lista de participantes. Se verifica que si hay al menos 2 participantes en el torneo. Además, se calcula la cantidad de días que durará el torneo con las fechas de inicio y finalización, igualmente se calcula la diferencia en horas entre la hora de inicio y finalización en cada día del torneo.

Se verifica también si hay suficientes días y horas para planificar los partidos del torneo. Para que cada equipo juegue al menos dos veces (como local y visitante), debe haber al menos 2*(n-1) días del torneo. Además, para cada equipo juega 2*(n-1) veces, por lo que debe haber suficiente días y horas para acomodar todos los partidos según las restricciones que se ha dicho anteriormente.

Con la cantidad de participantes, los días y horas que transcurren en el torneo, se crea una tabla de variables de 4 dimensiones ($n\ \times\ n\ \times\ días\ \times\ horas$) para enumerar y almacenar cada variable en esta tabla.

Posteriormente, se crea un archivo en formato DIMACS CNF y se abre en modo de escritura para colocar de cabecera en el archivo "`p cnf cantidad_variables cantidad_claúsulas`". Seguidamente, se coloca cada cláusula correspondiente de cada restricción, iterando sobre los índices de la tabla de variables.

Cada cláusula se escribe en forma normal conjuntiva, donde cada una representa una serie de números separados por espacios terminando con un 0. Cada número representa una variable o su negación. Si el número es positivo, representa la variable correspondiente. Si el número es negativo, representa la negación de la variable correspondiente.

Al terminar de colocar todas las cláusulas en el archivo de formato DIMACS, se guarda y se cierra el archivo. Luego, se utiliza Glucose (versión 4.2.1), que está integrado en el marco de trabajo OptiLog en Python, con el fin de resolver el problema en SAT y determinar si existe una asignación de valores True a las variables de la fórmula de tal manera que la fórmula completa se evalúe como verdadera . Si Glucose encuentra una solución al problema, se obtiene el modelo de la solución llamando al método `model()` del solver.

Si todos los valores en el modelo son negativos, significa que no hay solución al problema y se muestra un mensaje de error y se sale del programa. De lo contrario, se guarda en una lista los valores que resultaron positivos en el modelo. Estos valores corresponden a las variables del modelo y se procede a utilizarlos para crear el calendario de las fechas del torneo a través de la biblioteca `ics`.

Se itera sobre cada posible combinación de partidos entre los participantes. Para cada combinación, se itera sobre los días y las horas del torneo. Verifica si el elemento del modelo en la posición correspondiente de la tabla es positivo. Si es así, se crea un evento y se configura el nombre del partido, una descripción, las fechas y horas correspondientes. Y se agrega este evento al calendario creado.

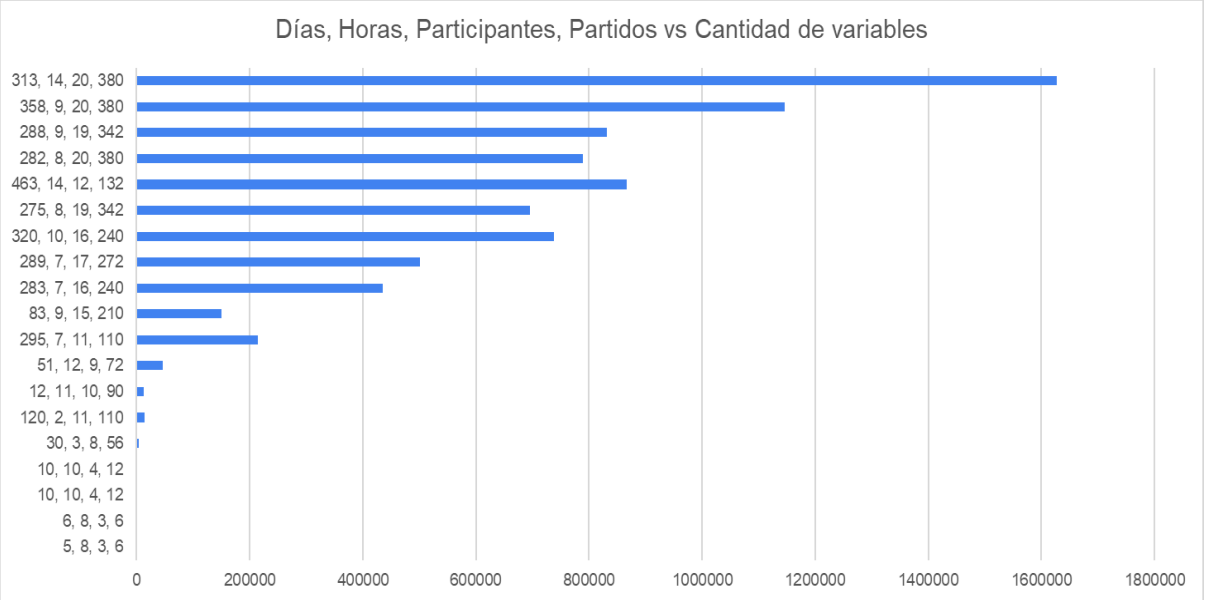
Análisis de los resultados

En el siguiente enlace al spreadsheet se encuentran los resultados del tiempo en que tarda en traducir las restricciones a formato DIMACS, en resolver el problema con

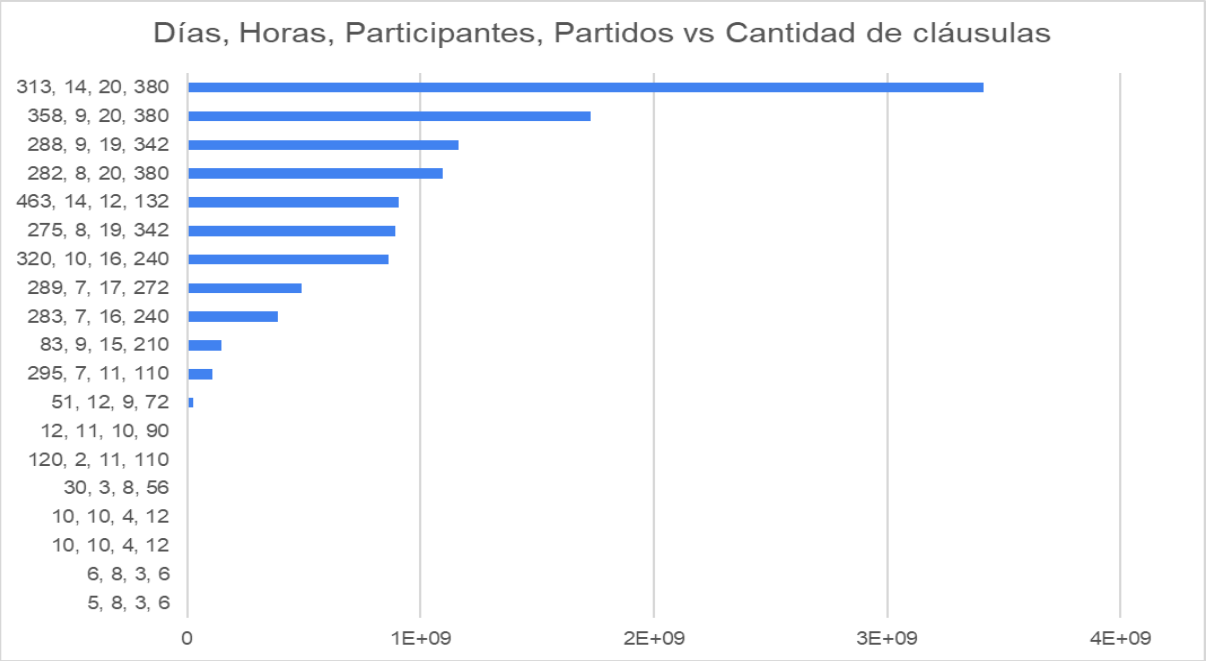
Glucose y el tiempo total según la cantidad de días, de las horas por día, el número de participantes y de partido de cada prueba de caso que se generó:

https://docs.google.com/spreadsheets/d/15XtJl7Snf5oiEZowgR9KLT036ARYt0guKgIb0clULSM/edit?usp=sharing

- Gráfico de la cantidad de días, horas, participantes y partidos vs cantidad de variables:

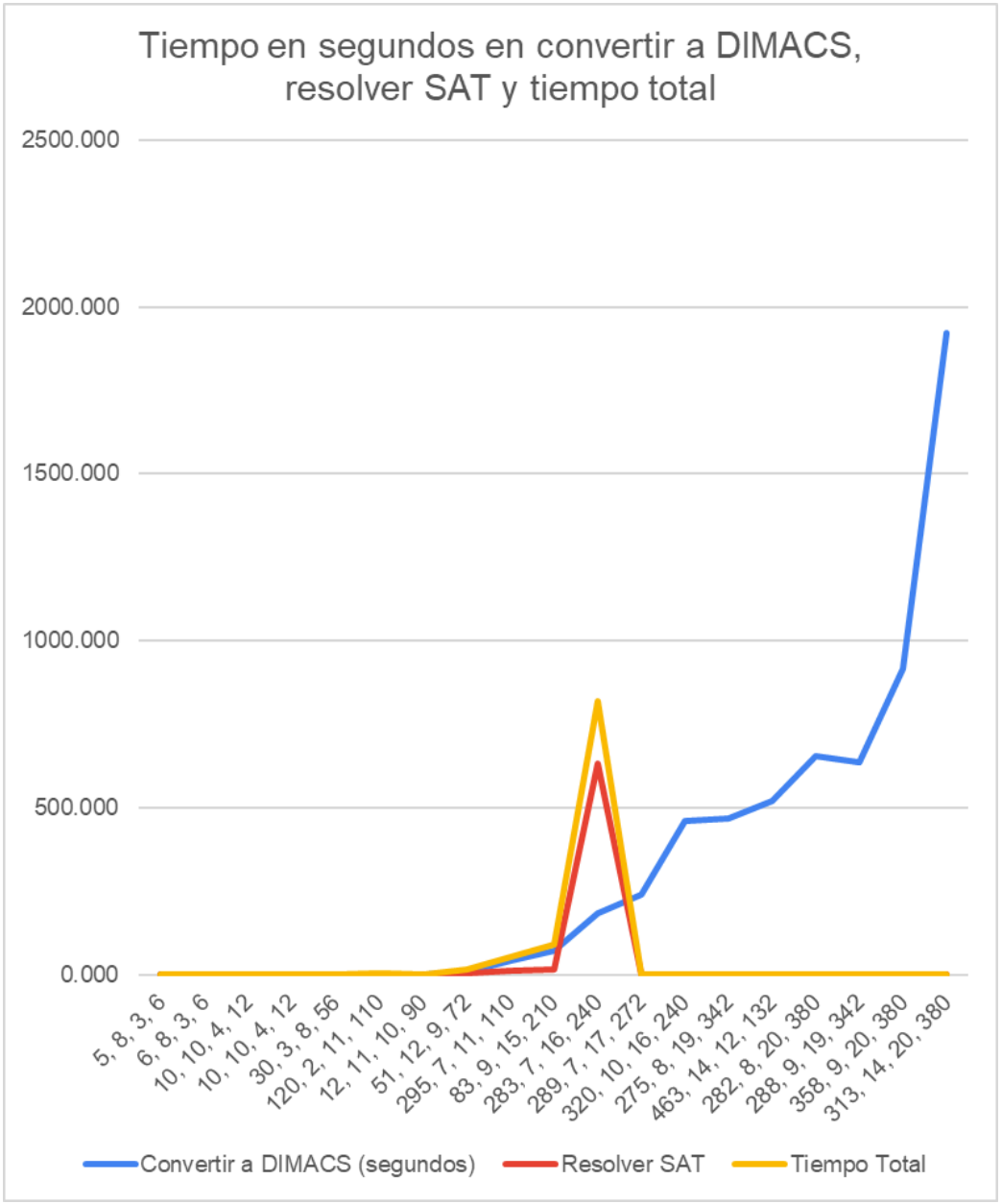


- Gráfico de la cantidad de días, horas, participantes y partidos vs cantidad de cláusulas:



De ambos gráficos de arriba, se puede observar que la cantidad de días/de horas/de participantes es directamente proporcional a la cantidad de variables y de cláusulas. También se observa que cuando se tiene la misma cantidad de participantes en dos pruebas de casos, se generan más cláusulas y variables para el caso de prueba que tiene más cantidad de horas, pero si en un caso la cantidad de días es enormemente más grande en comparación con la cantidad de días de otro caso, entonces éste último va a tener menos cantidad de variables.

- Gráfico de la cantidad de días, horas, participantes y partidos vs el tiempo en convertir las restricciones a DIMACS, resolver el problema con Glucose y el tiempo total de ejecución:



Cabe resaltar que los valores en el eje x del gráfico de arriba, están ordenados según la cantidad de cláusulas que se generan.

Se puede apreciar que para la línea azul que representa el tiempo en segundos para convertir las restricciones de cada prueba de caso en formato DIMACS, este tiempo aumenta de acuerdo a la cantidad de cláusulas, excepto para los dos siguientes casos:

Días	Horas	Participantes	Partidos	Número de variables	Número de cláusulas	Convertir a DIMACS
288	9	19	342	831744	1161760661	0:10:35.685141
282	8	20	380	789600	1097363579	0:10:56.436190

Acá tenemos que aunque la primera tenga más cantidad de cláusulas y de variables que el segundo, el tiempo en convertir las restricciones a DIMACS es menor que el tiempo de la segunda, con una diferencia de 20,75 segundos.

Otro detalle importante, es que existen dos casos donde no se generaron los archivos en formato DIMACS, debido a que la cantidad de días y horas no eran suficientes para planear el calendario de los partidos, y por lo tanto, en esos casos, es fácilmente deducible que el problema no es satisfacible.

Días	Horas	Participantes	Partidos	Número de variables	Número de cláusulas	Convertir a DIMACS	Resolver SAT	Tiempo total	
12	11	10	90	12000	7695570	0	0	0:00:00.007810	UNSAT
30	3	8	56	3840	489192	0	0	0:00:00.000008	UNSAT

Para el primero se tiene 12 días para el torneo y existen 10 participantes donde cada uno de ellos juega 18 veces, y como cada equipo solo puede jugar una sola vez al día, es imposible que un equipo pueda jugar los l8 partidos que le toca en 12 días.

Para el segundo, se tiene 30 días y 8 participantes, donde cada equipo juega 14 veces. En este caso, los días son suficientes para que al menos un equipo pueda jugar las 14 veces, sin embargo, las horas por día no son suficientes para tener los 56 partidos necesarios, pues como cada partido dura dos horas y tenemos 30 días y 3 horas por día, solamente se permite un partido por día, o sea, solo se podrá hacer 30 partidos en total.

Pero también se tiene otro caso en que el problema es no satisfacible pero se genera el archivo DIMACS, donde se tiene los siguientes datos:

Días	Horas	Participantes	Partidos	Número de variables	Número de cláusulas	Convertir a DIMACS	Resolver SAT	Tiempo total	
5	8	3	6	315	24237	0:00:00.060657	0:00:00.472353	0:00:00.539431	UNSAT

En este caso, se tiene que cada equipo juega 2 veces y entonces en total se tiene 6 partidos, y como tenemos 5 días y 8 horas, se puede hacer 4 partidos al día, o sea, 20 partidos en total. A simple vista, el problema parece ser satisfactorio. Sin embargo, en realidad no lo es, por ejemplo tenemos a los siguientes tres participantes: A, B y C.

Si el primer día A juega como local contra B, se tiene que C no puede jugar con nadie en este día pues no hay nadie que pueda jugar contra él ya que A y B ya jugaron el primer día y ambos solo pueden jugar a lo sumo una vez al día. De la misma forma con los siguientes 4 días, solo existe un partido al día donde habrá uno quien no podrá jugar. Y entonces, solo se tendrá 5 partidos al final y faltará uno que le falta jugar contra otro como local/visitante. Por ello, este caso es insatisfacible.

Ahora bien, en el último gráfico también se puede apreciar que luego del caso donde hay 283 días, 7 horas al día, 16 participantes y 240 partidos, la línea roja y amarilla que corresponde al tiempo en resolver el problema con Glucose y el tiempo total, se cae y baja hasta llegar a 0 segundos. Esto se debe a que para cada prueba de caso que seguía de esta, el proceso murió por falta de memoria a la hora de resolver el problema con Glucose.

También cabe resaltar que los archivos DIMACS que se generaban eran muy grandes y pesados, particularmente, para aquellos pruebas de caso que tenía más de

100 millones de cláusulas. Por ejemplo, para el archivo JSON: `benchmarks/bundesliga.json`, que tiene 289 días, 7 horas al día, 17 participantes y 272 partidos, con un total de 501.126 variables se generaba 490.603.782 cláusulas en el archivo DIMACS y se tenía que dicho archivo pesaba alrededor de 8,1 GB.

Conclusión

En este proyecto, abordamos el desafío de modelar y resolver un problema de programación de torneos utilizando un solver SAT. Nuestro enfoque consistió en traducir cuidadosamente las restricciones del problema a una fórmula lógica en formato CNF, para luego emplear el solver Glucose para encontrar una asignación de valores que satisfaga dicha fórmula.

A través de múltiples experimentos con diferentes cantidades de días, horas y participantes, pudimos analizar el rendimiento y las limitaciones de nuestro enfoque. Si bien los tiempos para convertir las restricciones a formato DIMACS eran razonables, el solver Glucose mostraba dificultades para manejar instancias de SAT con un gran número de cláusulas, especialmente aquellas que superan las 100,000,000 cláusulas. En algunos casos, el proceso murió debido a la falta de memoria, lo que destaca la necesidad de optimizaciones adicionales o el uso de solvers más potentes.

Otro aspecto a destacar es el tamaño de los archivos DIMACS generados, los cuales podían alcanzar varios gigabytes para instancias con una gran cantidad de cláusulas. Esto plantea desafíos en términos de almacenamiento y manipulación de estos archivos, lo que podría requerir el desarrollo de técnicas más eficientes para representar y procesar las instancias de SAT.

En general, este proyecto nos permitió comprender mejor las complejidades involucradas en la modelación y resolución de problemas de programación de torneos utilizando el solver SAT Glucose. Si bien los resultados obtenidos son prometedores, existen oportunidades para mejorar y optimizar nuestro enfoque, lo que podría conducir a soluciones más escalables y eficientes para este tipo de problemas.