FACULTAD DE INGENIERÍA

TALLER 1
Modelamiento e Implementación de CSP’s

LENIN ESTEBAN CARABALÍ MORENO - 202310025
HÉCTOR LUIS DÍAZ HURTADO - 2310001
ALEJANDRO GUERRERO CANO - 202179652

DOCENTE: ROBINSON DUQUE

PROGRAMACIÓN CON RESTRICCIONES
INGENIERÍA DE SISTEMAS - 3743
2025

INTRODUCCIÓN

El presente informe reúne el análisis, modelado e implementación de una serie de problemas formulados como Problemas de Satisfacción de Restricciones, en el marco del curso de Programación con Restricciones. Cada problema ha sido abordado utilizando el lenguaje de modelado MiniZinc, el cual permite expresar de forma declarativa las variables, dominios, restricciones y estrategias de búsqueda necesarias para resolver este tipo de problemas.

A lo largo del taller se desarrollaron modelos para diversos problemas. El objetivo principal de cada ejercicio fue formular un modelo correcto, eficiente y expresivo, que permitiera encontrar soluciones válidas en un espacio de búsqueda acotado por las restricciones impuestas.

Cada modelo incluye una descripción de los parámetros, variables y restricciones utilizadas, así como una explicación de las decisiones de modelado tomadas. Se realizaron pruebas con distintas configuraciones y estrategias de búsqueda, analizando el impacto de estas en el desempeño del solver. Cuando fue pertinente, se integraron restricciones redundantes o restricciones para romper simetrías, con el fin de optimizar el proceso de búsqueda de soluciones.

Este informe también presenta un análisis comparativo de las estrategias utilizadas y una reflexión final sobre los resultados obtenidos, su validez y eficiencia, así como las posibilidades de mejora en términos de modelado y rendimiento.

METODOLOGÍA DE TRABAJO

Se realizaron pruebas considerando diferentes configuraciones según nuestras propias consideraciones para cada pregunta, aplicando casos con y sin redundancia, y con y sin simetría cuando fuese pertinente. Se desarrolló un script en Python que permitió automatizar las ejecuciones, manejar distintas estrategias de búsqueda y comparar el rendimiento de dos solvers (Gecode y Chuffed).

Los resultados fueron organizados en hojas de cálculo anexas para facilitar su análisis. A partir de estos datos, se tomaron decisiones sobre los mejores valores obtenidos según la estrategia y el solver más adecuado para cada ejemplo.

DESARROLLO DE PROBLEMAS USANDO CSP'S

SUDOKU

DESCRIPCIÓN DEL PROBLEMA

El Sudoku es un rompecabezas numérico de lógica que consiste en rellenar una cuadrícula de 9×9 con dígitos del 1 al 9, cumpliendo tres restricciones fundamentales: cada fila, cada columna y cada una de las nueve sub cuadrículas de 3×3 deben contener todos los números del 1 al 9 sin repeticiones. Algunas celdas están previamente llenas, sirviendo como pistas.

Desde el punto de vista de la programación por restricciones, el Sudoku se modela como un CSP, donde cada celda es una variable con dominio 1..9, y las restricciones de unicidad se imponen mediante restricciones de tipo alldifferent. La solución debe cumplir todas las restricciones y, en general, se espera que sea única.

DEFINICIÓN FORMAL DEL MODELO

PARÁMETROS

- $tabla\_incial$ **:** Representa la instancia del juego de Sudoku. Ya con su dominio definido

$$\forall i,j \in [1,9],\ tabla\_inicial_{i,j} \in [0,9]$$

  - Los valores distintos de cero indican pistas iniciales ya definidas en el tablero.
  - Los ceros representan celdas vacías que deben ser completadas por el modelo.

VARIABLES

- **tabla:** representa el estado del tablero de Sudoku en solución, modelado como una matriz de 9 filas por 9 columnas, donde cada celda de la matriz tabla $(tabla_{i,j})$ es una variable de decisión. Ya con su dominio definido.

$$\forall i,j \in [1,9],\ tabla_{i,j} \in [1,9]$$

RESTRICCIONES

- **Restricción de filas:** Garantiza que cada fila contenga los números del 1 al 9 sin repetirse.

$$\forall i \in [1,9],\ alldifferent(tabla_{i,j} | j \in [1,9])$$

- **Restricción de columnas:** Garantiza que cada columna contenga los números del 1 al 9 sin repetirse.

$$\forall j \in [1,9],\ alldifferent(tabla_{i,j} | \ i \in [1,9])$$

- **Restricción de columnas:** Garantiza que cada columna contenga los números del 1 al 9 sin repetirse.

$$\forall j \in [1,9],\ alldifferent(tabla_{i,j} | \ i \in [1,9])$$

- **Fijación de valores predefinidos**: Mantiene los valores iniciales dados en la cuadrícula sin permitir cambios en esas posiciones.

$$\forall i,j \in [1,9],\ tabla\_inicial_{i,j} = tabla_{i,j} \mid tabla\_inicial_{i,j} \neq 0)$$

RESTRICCIONES REDUNDANTES

Se incluyeron tres restricciones redundantes que imponen una suma fija de 45 en cada fila, columna y subcuadrícula de 3×3. Aunque estas restricciones son lógicamente redundantes, ya que se encuentran implícitas en las restricciones del problema. Fueron añadidas con el objetivo de mejorar la eficiencia del modelo. Su incorporación permite una propagación más efectiva, acotando los dominios de las variables con menos iteraciones, lo cual reduce el tamaño del árbol de búsqueda y el tiempo de resolución en muchos casos.

- **Suma de cada fila:** Cada fila contiene los números del 1 al 9 sin repetirse, lo que significa que la suma de todos sus elementos siempre debe ser 45

$$\forall i \in [1,9],\ \sum_{i=1}^{9} tabla_{i,j} = 45$$

- **Suma de cada columna:** Cada columna también debe contener los números del 1 al 9 sin repetirse, asegurando la misma suma de 45

$$\forall j \in [1,9],\ \sum_{j=1}^{9} tabla_{i,j} = 45$$

- **Suma de cada subcuadrícula 3×3:** Cada una de estas subcuadrículas debe contener los números del 1 al 9 sin repetirse, asegurando que su suma sea 45.

$$\forall j \in [1,9],\ alldifferent(tabla_{i,j} | \ i \in [1,9])$$

ESTRATEGIAS DE BÚSQUEDA

Para las pruebas sobre el modelo de Sudoku se utilizaron distintas estrategias de heurísticas de selección de variables

- *first_fail,*
- *dom_w_deg*
- *input_order*
- *satisfy (que el solver decida por sí mismo)*

Combinadas con heurísticas de selección de valores:

- *indomain_min*
- *indomain_split*
- *indomain_median*
- *satisfy (que el solver decida por sí mismo)*

Estas combinaciones se evaluaron utilizando dos solvers diferentes: **Gecode** y **Chuffed**, con el fin de analizar si el uso de un solver distinto generaba alguna diferencia significativa en el rendimiento del modelo, tanto en tiempo de resolución como en la cantidad de nodos explorados, fallos, profundidad del árbol de búsqueda y soluciones

PRUEBAS

Se realizaron un total de 48 pruebas con restricciones de redundancia y 48 pruebas sin restricciones de redundancia par aun total del 96 pruebas, resultado de combinar 4 heurísticas de selección de variables con 4 heurísticas de selección de valores, aplicadas a dos solvers distintos: Gecode y Chuffed. Para cada combinación de estrategias y solver se ejecutaron 4 pruebas, lo que permitió observar el comportamiento del modelo bajo diversas configuraciones.

Durante las ejecuciones se recopilaron métricas relevantes como el tiempo de inicialización, tiempo de resolución, tiempo total de ejecución, así como el número de nodos explorados, fallos registrados y la profundidad máxima del árbol de búsqueda. Esta información sirvió de base para un análisis detallado del impacto de las estrategias sobre el rendimiento del modelo.

A partir de este conjunto de pruebas, se seleccionaron 8 combinaciones representativas de heurísticas en cada solver para ser analizadas con mayor detalle. Cada combinación fue ejecutada con ambos solvers, con el fin de comparar directamente su desempeño bajo las mismas condiciones. Los resultados se resumen en la siguiente tabla, en la que se destacan las diferencias en tiempo,

nodos, fallos y profundidad alcanzada, evidenciando cómo influye el uso del solver y la estrategia de búsqueda en la eficiencia de resolución.

Total de pruebas sudoku: Resultados Sodoku.xlsx

PRUEBAS CON RESTRICCIONES DE REDUNDANCIA

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	totalTime	solutions	nodes	failures	peakDepth
example-e.dzn	Gecode	satisfy	satisfy	0,0233156	1	289	144	13
example-e.dzn	Chuffed	input_order	indomain_min	0,015	1	24	18	4
example-h2.dzn	Gecode	dom_w_deg	indomain_min	0,0202074	1	481	240	13
example-h2.dzn	Chuffed	satisfy	satisfy	0,024	1	71	66	7
example-m.dzn	Gecode	satisfy	satisfy	0,0143312	21	211	85	13
example-m.dzn	Chuffed	first_fail	indomain_min	0,026	21	43	36	7
example1.dzn	Gecode	first_fail	indomain_median	0,0028064	1	1	0	0
example1.dzn	Chuffed	satisfy	satisfy	0,016	1	1	1	0

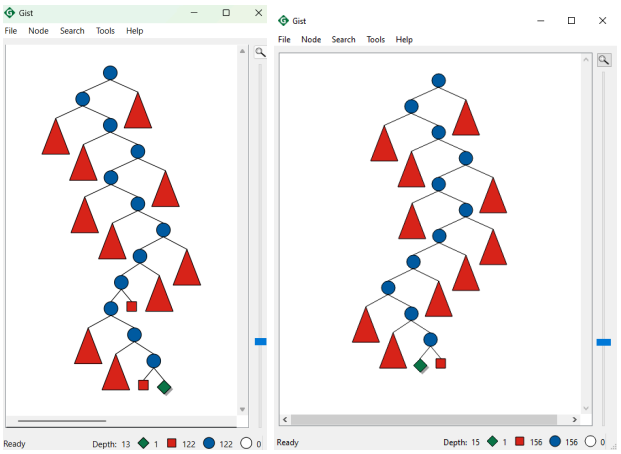
PRUEBAS SIN RESTRICCIONES DE REDUNDANCIA

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	totalTime	solutions	nodes	failures	peakDepth
example-e.dzn	Gecode	satisfy	satisfy	0,006752	1	297	148	12
example-e.dzn	Chuffed	dom_w_deg	indomain_min	0,009	1	24	18	4
example-h2.dzn	Gecode	satisfy	satisfy	0,009344	1	567	283	12
example-h2.dzn	Chuffed	satisfy	satisfy	0,01	1	79	76	7
example-m.dzn	Gecode	dom_w_deg	indomain_min	0,00912	21	179	69	12
example-m.dzn	Chuffed	first_fail	indomain_min	0,014	21	43	36	7
example1.dzn	Gecode	first_fail	indomain_median	0,0028064	1	1	0	0
example1.dzn	Chuffed	satisfy	satisfy	0,016	1	1	1	0

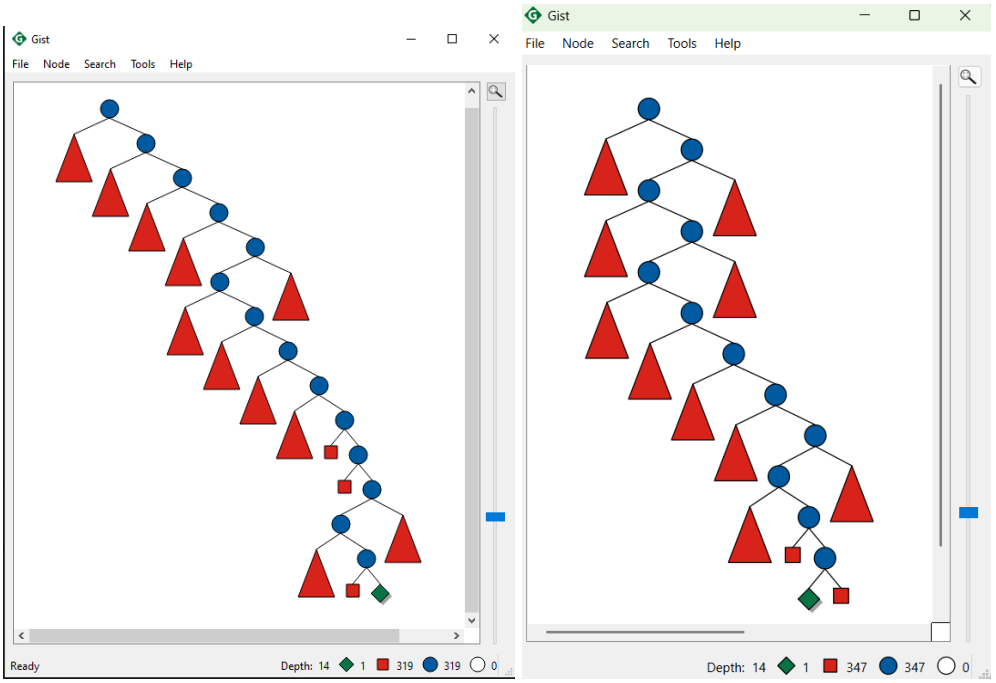
ÁRBOLES GENERADOS

A partir de los ejemplos previamente seleccionados en la tabla, se generan los árboles de búsqueda utilizando Gecode Gist. **Sin restricciones de redundancia y con restricciones de redundancia respectivamente.**

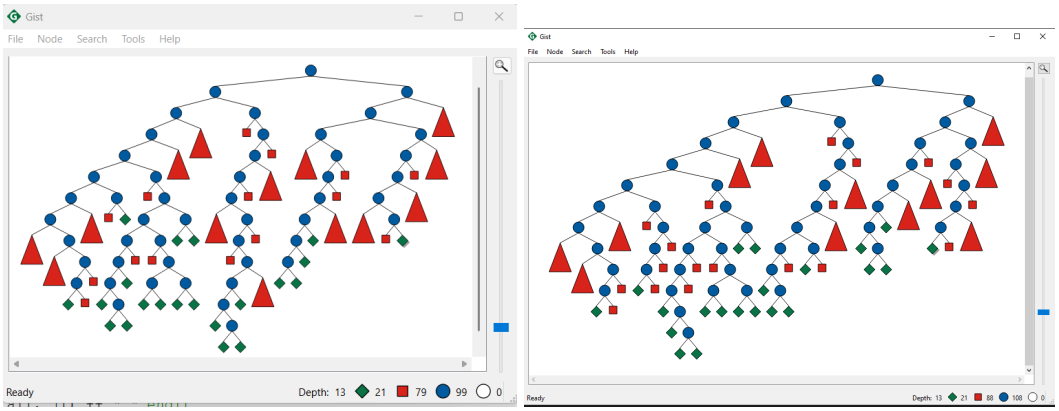
example-e.dzn



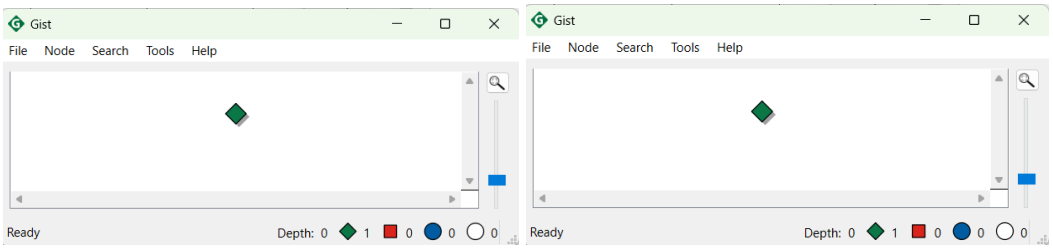
example-h2.dzn



example-m.dzn



example1.dzn



ANÁLISIS

- Podemos notar que los valores en las estrategias cambian en algunos casos al ejecutar sin restricciones de redundancia, y que mejoran bastante cuando se aplican las restricciones de redundancia. (Visualización en el anexo Resultados Sodoku.xlsx).
- En este caso, el **mejor solver fue Chuffed** al dar árboles más reducidos y conservando las mismas soluciones.
- A partir de las mejores combinaciones de heurísticas evaluadas por archivo y solver, se observa que la estrategia **satisfy / satisfy** fue una de las más efectivas tanto en Gecode como en Chuffed, resolviendo varias instancias con mínimos nodos explorados y fallos. En general, la elección del solver y la estrategia de búsqueda influye notablemente en el rendimiento. Sin embargo, en algunas instancias bien restringidas, el uso de heurísticas específicas no aporta mejoras, siendo solver satisfy una opción más simple y eficiente.
- **Restricciones de Redundancia:** Decidimos usar restricciones de redundancia porque ayudan al solver a reducir el espacio de búsqueda al incorporar conocimiento adicional del problema (como que cada fila, columna y subcuadro debe sumar 45), para guíar mejor la exploración y mejora notablemente la eficiencia. Al final sí se pudo evidenciar en los árboles más pequeños generados cuando se corrieron las pruebas del sudoku.
- **Restricciones de Simetría**: En este caso no fue necesario aplicar rompimiento de simetría, ya que las pistas iniciales del Sudoku suelen restringir suficientemente el dominio, reduciendo significativamente la aparición de soluciones simétricas. Aunque no siempre se garantiza una solución única, las simetrías estructurales del tablero tienden a romperse naturalmente con las pistas dadas, por lo que añadir restricciones adicionales para este fin no resulta necesario ni aporta mejoras significativas al desempeño del modelo.

KAKURO

DESCRIPCIÓN DEL PROBLEMA

Kakuro es un juego de lógica numérica similar a un crucigrama, donde las pistas son sumas en lugar de palabras. El objetivo es rellenar una cuadrícula con números del 1 al 9 de manera que, en cada fila o columna indicada, los números sumen un valor específico sin repetir dígitos dentro de cada grupo. Las combinaciones numéricas deben cumplir las sumas dadas en las casillas negras, y la disposición debe respetar las restricciones tanto horizontales como verticales. La solución debe ser coherente con todas las pistas.

DEFINICIÓN FORMAL DEL MODELO

PARÁMETROS

- $N \in Z^{+}$: Un valor entero que define el tamaño de la cuadrícula $(N\ x\ N)$
- $cell\_type$: Matriz de tamaño $N\ *\ N$ con un valores $\in \{0, 1, 2\}$ que indica si la casilla está bloqueado (0), dispositivo (1) o si es una pista horizontal y/o vertical (2)
- $horizontal\_sum[i, j] \in Z \cup \{-\ 1\}$: Matriz de tamaño $N\ *\ N$ con valores de la suma horizontal indicada en la pista en (i, j). Si no hay pista, vale −1.
- $vertical\_sum[i, j] \in Z \cup \{-\ 1\}$: Matriz de tamaño $N\ *\ N$ con valores de la suma vertical indicada en la pista en (i, j). Si no hay pista, vale −1.

VARIABLES

- $grid$: Una matriz de tamaño $N\ *\ N$ que representa el valor en la celda (i, j). Con dominio del 0 al 9.

RESTRICCIONES

- **Valores válidos (celdas jugables):** Esto asegura que solo las celdas jugables pueden tener valores del 1 al 9.

$$\forall i, j\ \in\ [1, N],\ si\ cell\_type[i, j]\ =\ 1\ \Rightarrow\ \ grid[i, j]\ \in\ [1,\ 9]$$

- **Valores válidos (celdas no jugables):** Esto garantiza que las celdas negras (no jugables) y de pista no contengan números del 1 al 9.

$$\forall i, j\ \in\ [1, N],\ si\ cell\_type[i, j]\ \neq\ 1\ \Rightarrow\ \ grid[i, j]\ =\ 0$$

- **Valores horizontales iguales a la pista y diferentes:** Garantiza que la suma da el valor de la pista y no se repite ningún número.Sea k la cantidad de celdas jugables a la derecha

$$\sum_{l=1}^{k} grid[i, j + l] = horizontal\_sum[i, j]$$ **and**

$$grid[i, j + 1] \neq grid[i, j + 2] \neq \ldots \neq grid[i, j + k]$$

- **Valores verticales iguales a la pista y diferentes:** Garantiza que la suma da el valor de la pista y no se repite ningún número.Sea k la cantidad de celdas jugables hacia abajo

$$\sum_{l=1}^{k} grid[i + l, j] = vertical\_sum[i, j]$$ **and**

$$grid[i + 1, j] \neq grid[i + 2, j] \neq \ldots \neq grid[i + k, j]$$

RESTRICCIONES REDUNDANTES

Restringen los valores que hay en $horizontal\_sum[i, j]$ (es decir, la suma objetivo de ese bloque horizontal) y $vertical\ sum[i, j]$ debe estar entre la suma mínima y máxima posibles con n (la cantidad de celdas jugables) valores distintos del 1 al 9. Esperando que se encuentre dentro del rango de sumas posibles para un bloque de n celdas jugables.

- $\frac{n(n+1)}{2}$ representa la suma mínima posible que se obtiene al usar los n números más pequeños (1, 2, …, n) sin repetir.
- $\frac{n(19-n)}{2}$ representa la suma máxima posible al usar los n números más grandes (9, 8, …, 10−n) sin repetir.

- **Restricción de rango válido de suma horizontal**

$$\frac{n(n+1)}{2} \leq horizontal\_sum[i, j] \leq \frac{n(19-n)}{2}$$

- **Restricción de rango válido de suma vertical**

$$\frac{n(n+1)}{2} \leq vertical\ sum[i, j] \leq \frac{n(19-n)}{2}$$

ESTRATEGIAS DE BÚSQUEDA

Para las pruebas sobre el modelo del Kakuro se utilizaron distintas estrategias de heurísticas de selección de variables

- *first_fail,*
- *dom_w_deg*
- *input_order*
- *satisfy (que el solver decida por sí mismo)*

Combinadas con heurísticas de selección de valores:

- *indomain_min*
- *indomain_split*
- *indomain_median*
- *satisfy (que el solver decida por sí mismo)*

Estas combinaciones se evaluaron utilizando dos solvers diferentes: **Gecode** y **Chuffed**, con el fin de analizar si el uso de un solver distinto generaba alguna diferencia significativa en el rendimiento del modelo, tanto en tiempo de resolución como en la cantidad de nodos explorados, fallos, profundidad del árbol de búsqueda y soluciones.

PRUEBAS

Se realizaron un total de 60 pruebas (120 en total incluyendo las de redundancia y las que no lo incluyen), resultado de combinar 4 heurísticas de selección de variables con 4 heurísticas de selección de valores, aplicadas a dos solvers distintos: Gecode y Chuffed. Para cada combinación de estrategias y solver se ejecutaron 5 pruebas, lo que permitió observar el comportamiento del modelo bajo diversas configuraciones.

Durante las ejecuciones se recopilaron métricas relevantes como el tiempo de inicialización, tiempo de resolución, tiempo total de ejecución, así como el número de nodos explorados, fallos registrados y la profundidad máxima del árbol de búsqueda. Esta información sirvió de base para un análisis detallado del impacto de las estrategias sobre el rendimiento del modelo.

A partir de este conjunto de pruebas, se seleccionaron 10 combinaciones representativas de heurísticas en cada solver para ser analizadas con mayor detalle. Cada combinación fue ejecutada con ambos solvers, con el fin de comparar directamente su desempeño bajo las mismas condiciones. Los resultados se resumen en la siguiente tabla, en la que se destacan las diferencias en tiempo, nodos, fallos y profundidad alcanzada, evidenciando cómo influye el uso del solver y la estrategia de búsqueda en la eficiencia de resolución.

Total de pruebas sudoku: Resultados Kakuro.xlsx

PRUEBAS CON RESTRICCIONES DE REDUNDANCIA

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	totalTime	solutions	nodes	failures	peakDepth
ejemplo1.dzn	Gecode	first_fail	indomain_split	0,000659	1	1	0	0
ejemplo1.dzn	Chuffed	satisfy	satisfy	0,002	1	1	1	0
ejemplo2.dzn	Gecode	first_fail	indomain_split	0,0016119	48	109	7	7
ejemplo2.dzn	Chuffed	first_fail	indomain_min	0,002	48	50	50	2
ejemplo3.dzn	Gecode	satisfy	satisfy	0,0029426	1	7	3	3
ejemplo3.dzn	Chuffed	satisfy	satisfy	0,003	1	4	4	1
ejemplo4.dzn	Gecode	input_order	indomain_min	0,0006055	8	17	1	1
ejemplo4.dzn	Chuffed	input_order	indomain_min	0,001	8	9	9	1
ejemplo5.dzn	Gecode	input_order	indomain_min	0,0006665	1	3	1	1
ejemplo5.dzn	Chuffed	satisfy	satisfy	0,002	1	2	2	1

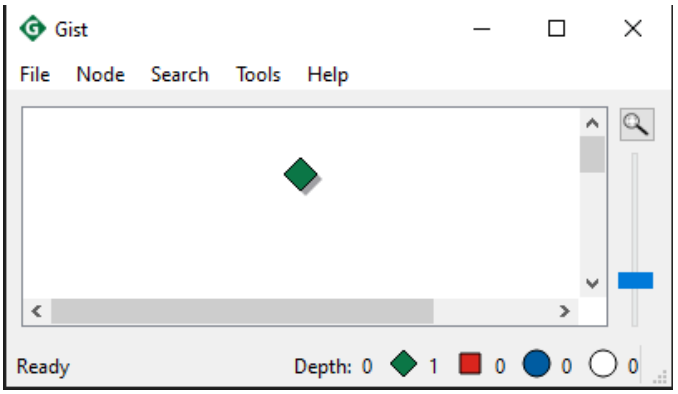
Las pruebas se ejecutaron usando las restricciones de redundancia, aunque en este no mostró mejoras en este caso, representan las mismas salidas que las pruebas sin usar las restricciones de redundancia. La comparación se puede evidenciar en el anexo y del los resultados obtenidos (Resultados Kakuro.xlsx) por lo tanto representan los mismos árboles de búsqueda en este caso.

PRUEBAS SIN RESTRICCIONES DE REDUNDANCIA

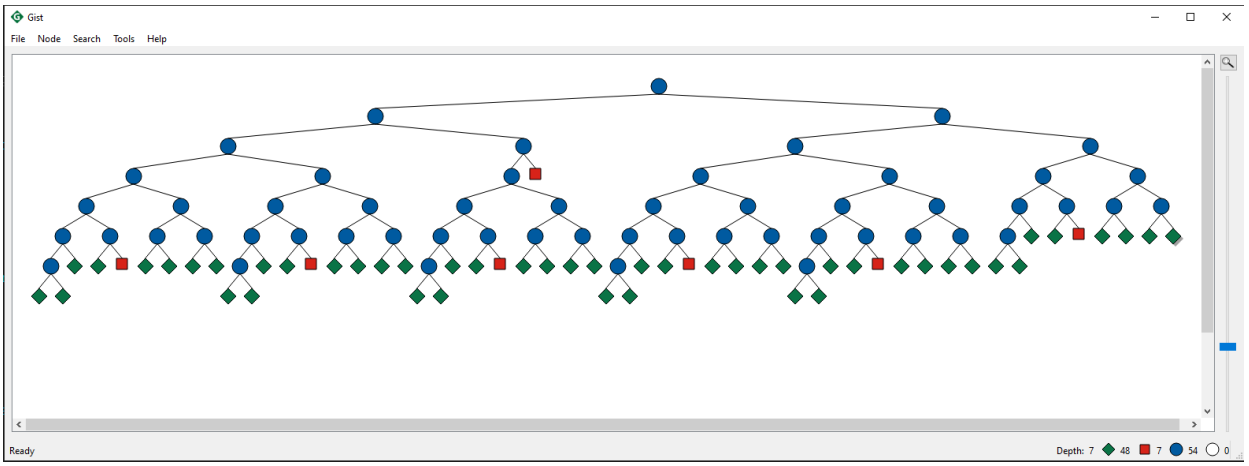
Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	totalTime	solutions	nodes	failures	peakDepth
ejemplo1.dzn	Gecode	first_fail	indomain_split	0,00066	1	1	0	0
ejemplo1.dzn	Chuffed	satisfy	satisfy	0,002	1	1	1	0
ejemplo2.dzn	Gecode	first_fail	indomain_split	0,003931	48	109	7	7
ejemplo2.dzn	Chuffed	first_fail	indomain_min	0,002	48	50	50	2
ejemplo3.dzn	Gecode	satisfy	satisfy	0,0029426	1	7	3	3
ejemplo3.dzn	Chuffed	satisfy	satisfy	0,003	1	4	4	1
ejemplo4.dzn	Gecode	input_order	indomain_min	0,001599	8	17	1	1
ejemplo4.dzn	Chuffed	input_order	indomain_min	0,003	8	9	9	1
ejemplo5.dzn	Gecode	input_order	indomain_min	0,000779	1	3	1	1
ejemplo5.dzn	Chuffed	satisfy	satisfy	0,003	1	2	2	1

ÁRBOLES GENERADOS

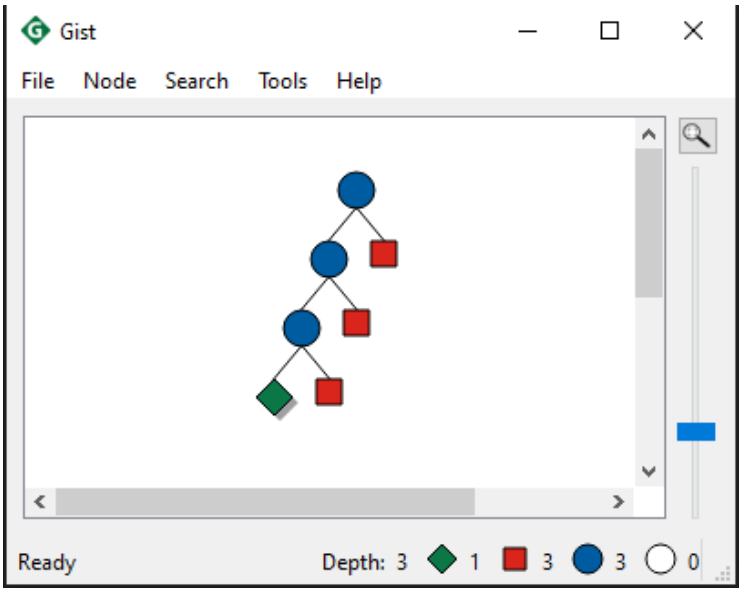
ejemplo1.dzn



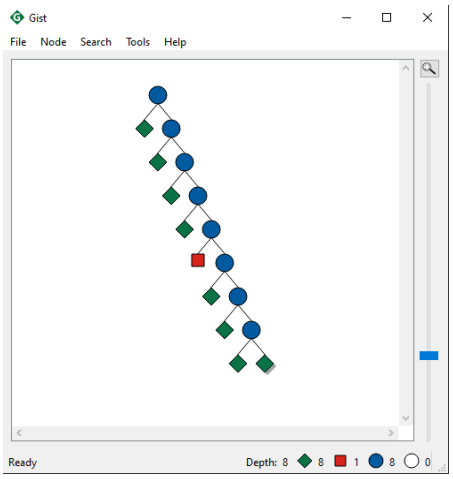
ejemplo2.dzn



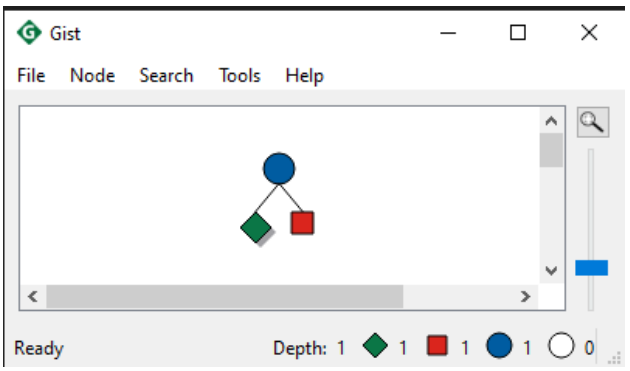
ejemplo3.dzn



ejemplo4.dzn



ejemplo5.dzn



ANÁLISIS

- Aunque en este conjunto de pruebas se evidenció un mayor uso de estrategias explícitas de búsqueda, como first_fail o indomain_min, la mayoría de los casos evaluados (10 en total) siguen utilizando **satisfy** como estrategia principal, por lo que se mantiene como la escogida para este modelo.

- Aunque no se observó una reducción en los parámetros globales del árbol (como la profundidad o la cantidad total de nodos) a pesar de tener las restricciones de redundancia, sí se evidenció que al usar estrategias de búsqueda alternativas se modificó la estructura interna del árbol de búsqueda, reduciendo efectivamente el dominio explorado.

- El solver Chuffed no arrojó árboles más pequeño y aún así conservó las mismas soluciones.

- **Restricciones de Redundancia:** Se incluyeron estas restricciones pero pensamos que quizá las restricciones de redundancia no funcionaron porque el modelo ya está fuertemente restringido por la definición de dominios y otras restricciones básicas, de modo que la adición de redundancias no aporta mejoras significativas. También consideramos que, a pesar de haberse probado más opciones de restricciones de redundancia sin efecto alguno, habría sido necesario hallar otra restricción mejor para lograr reducir efectivamente el dominio explorado. Estas restricciones trataban de acotar los valores posibles de las sumas según la longitud de cada bloque, con el objetivo de reducir el dominio de búsqueda.

- **Restricciones de Simetría:** En el modelo de Kakuro no se aplicaron restricciones para romper asimetrías, ya que las propias reglas del juego (como la suma exacta y la no repetición de dígitos en cada grupo) generan una estructura naturalmente asimétrica. Estas restricciones ya limitan suficientemente las posibles soluciones, por lo que imponer condiciones adicionales de simetría no resulta útil ni mejora el rendimiento del modelo.

SECUENCIA MÁGICA

DESCRIPCIÓN DEL PROBLEMA

El problema consiste en generar todas las secuencias mágicas de longitud n, donde cada número i aparece exactamente x_i veces en la secuencia. Se deben cumplir dos restricciones adicionales para reducir el espacio de búsqueda: la suma de los elementos debe ser n y una ecuación de equilibrio lineal debe mantenerse.

DEFINICIÓN FORMAL DEL MODELO

PARÁMETROS

$n \in Z^+$ Tamaño del conjunto de variables, que define la longitud de la secuencia mágica.

VARIABLES

x : arreglo de tamaño n , donde x_i representa cuántas veces aparece el número i en la secuencia. Con dominio de 0 a n-1. $x \in [0,\ n-1]$

RESTRICCIONES

- **Frecuencia exacta**: Cada número i aparece exactamente $x[i]$ veces en la secuencia.

$$\forall i \in [0, n-1],\ count(x, i) = x[i]$$

RESTRICCIONES REDUNDANTES

- **Suma total**: La suma de los elementos debe ser igual a n.

$$\sum_{i=1}^{n} x[i] = n$$

- **Ecuación de balance**: Expresa una restricción donde la suma de los términos de una secuencia, ponderados por un coeficiente lineal, es igual a cero.

$$\sum_{i=1}^{n} (i-2) * x[i-1] = 0$$

ESTRATEGIAS DE BÚSQUEDA

Para las pruebas sobre el modelo de Secuencia Mágica se utilizaron distintas estrategias de heurísticas de selección de variables

- *first_fail,*
- *dom_w_deg*
- *input_order*
- *satisfy (que el solver decida por sí mismo)*

Combinadas con heurísticas de selección de valores:

- *indomain_min*
- *indomain_split*
- *indomain_median*
- *satisfy (que el solver decida por sí mismo)*

Estas combinaciones se evaluaron utilizando dos solvers diferentes: **Gecode** y **Chuffed**, con el fin de analizar si el uso de un solver distinto generaba alguna diferencia significativa en el rendimiento del modelo, tanto en tiempo de resolución como en la cantidad de nodos explorados, fallos, profundidad del árbol de búsqueda y soluciones.

Después se dejó sólo la estrategia de búsqueda **satisfy** debido a los resultados y se ahondó en la eficiencia del modelo al usar **restricciones de redundancia.**

PRUEBAS

Se realizaron un total de **60** pruebas, resultado de combinar 4 heurísticas de selección de variables con 4 heurísticas de selección de valores, aplicadas a dos solvers distintos: Gecode y Chuffed. Para cada combinación de estrategias y solver se ejecutaron 5 ejemplos de pruebas, lo que permitió observar el comportamiento del modelo bajo diversas configuraciones.

Durante las ejecuciones se recopilaron métricas relevantes como el tiempo de inicialización, tiempo de resolución, tiempo total de ejecución, así como el número de nodos explorados, fallos registrados y la profundidad máxima del árbol de búsqueda. Esta información sirvió de base para un análisis detallado del impacto de las estrategias sobre el rendimiento del modelo.

A partir de este conjunto de pruebas, se seleccionaron 8 combinaciones representativas de heurísticas en cada solver para ser analizadas con mayor detalle, 8 ya que para el valor de $n = 3$ **no había solución**. Cada combinación fue ejecutada con ambos solvers, con el fin de comparar directamente su desempeño bajo las mismas condiciones. Los resultados se resumen en la siguiente tabla, en la que se destacan las diferencias en tiempo, nodos, fallos y profundidad alcanzada,

evidenciando cómo influye el uso del solver y la estrategia de búsqueda en la eficiencia de resolución.

Total de pruebas Secuencia Mágica: Resultados S. Magica.xlsx

PRUEBAS CON REDUNDANCIA.

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	totalTime	solutions	nodes	failures	peakDepth
222.dzn	Gecode	satisfy	satisfy	0,0274911	1	223	111	111
222.dzn	Chuffed	satisfy	satisfy	2,977	1	114	114	2
31.dzn	Gecode	satisfy	satisfy	0,0021263	1	31	15	15
31.dzn	Chuffed	satisfy	satisfy	0,086	1	19	19	2
4.dzn	Gecode	satisfy	satisfy	0,0011794	2	5	1	2
4.dzn	Chuffed	first_fail	indomain_min	0,016	2	3	3	2
97.dzn	Gecode	satisfy	satisfy	0,0147408	1	97	48	48
97.dzn	Chuffed	satisfy	satisfy	0,478	1	52	52	2

Al validar que la mayoría de las estrategias de búsqueda óptimas fueron *satisfy* (es decir, seleccionadas por el *solver*), decidimos comparar las diferencias tanto con las restricciones de redundancia como sin ellas, con el fin de evaluar sus efectos.

PRUEBAS CON REDUNDANCIA

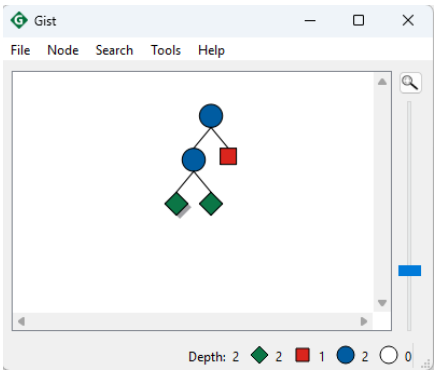
Archivo	initTime	solveTime	totalTime	solutions	nodes	failures	peakDepth
222-redun.dzn	0,009324	0,006705	0,016029	1	223	111	111
222.dzn	Timeout	-	-	-	-	-	-
31-redun.dzn	0,000806	0,000235	0,001041	1	31	15	15
31.dzn	0,000589	0,011176	0,011765	1	1125	562	23
4-redun.dzn	0,000379	8,31E-05	0,000462	2	5	1	2
4.dzn	0,000328	8,67E-05	0,000415	2	17	7	3
97-redun.dzn	0,002064	0,001183	0,003247	1	97	48	48
97.dzn	0,001729	2,39804	2,399769	1	2275	1137	56

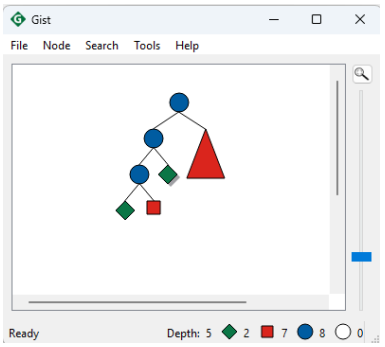
En los resultados de las pruebas con y sin restricciones de redundancia, se observa una reducción significativa en la profundidad máxima de los árboles de búsqueda, así como en el número total de nodos explorados y fallos generados. Esto demuestra que las restricciones adicionales fueron efectivas para limitar el espacio de búsqueda.

ÁRBOLES GENERADOS

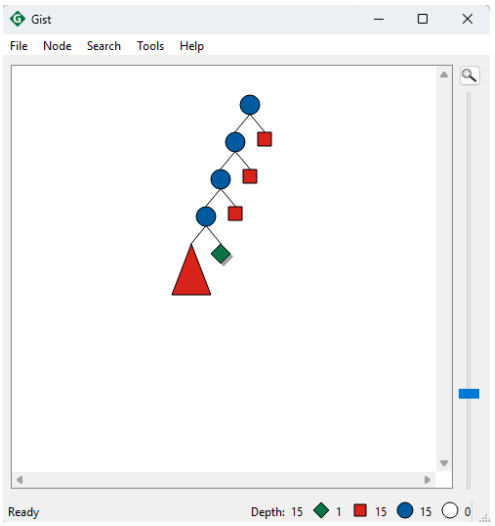
Con restricciones de redundancia y sin redundancia respectivamente:

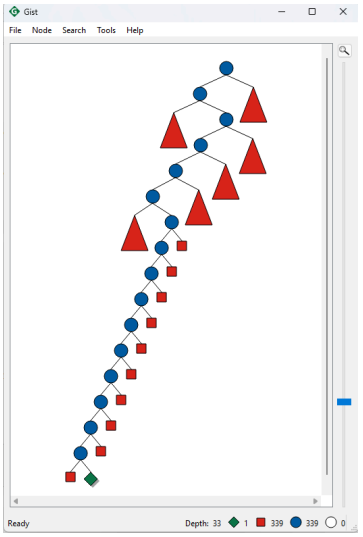
N=4



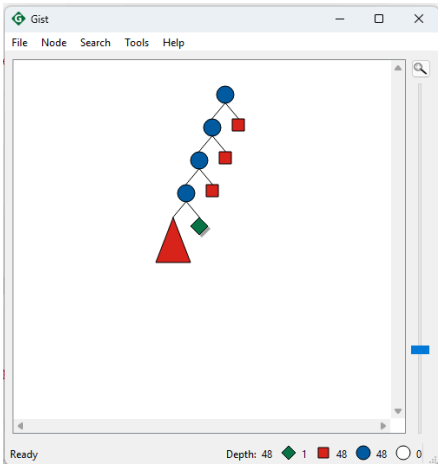


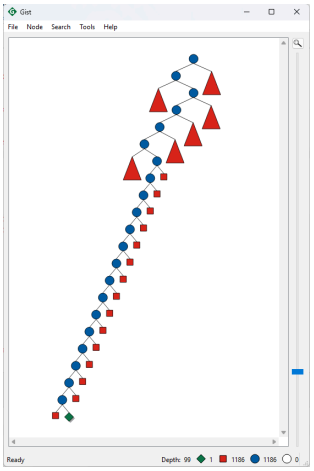
N=31



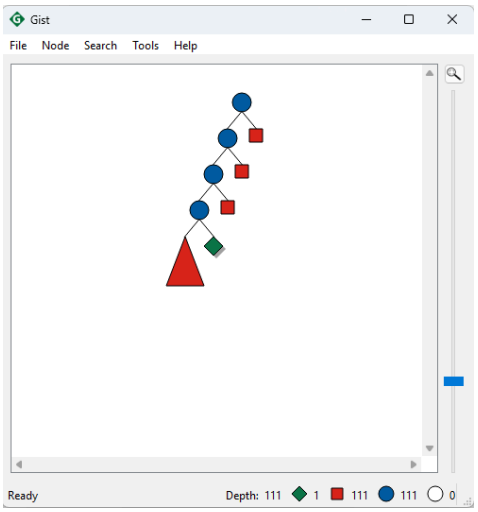


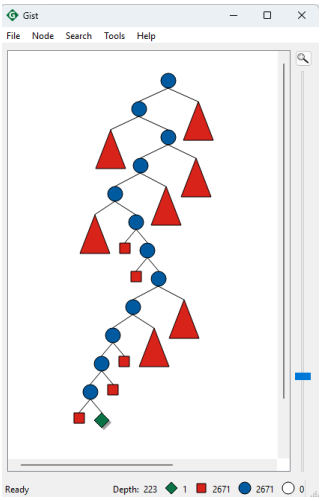
N=97





N=222





ANÁLISIS

- Los resultados muestran un desempeño significativamente mejor en las pruebas con restricciones de redundancia en comparación con aquellas sin redundancia. En términos de tiempos de resolución (*solveTime* y *totalTime*), cantidad de nodos explorados y fallos, las pruebas con redundancia lograron reducir drásticamente los valores en todos los casos.
- La mejor estrategia de búsqueda en este caso fue **satisfy**
- En particular, en el caso del archivo **222.dzn**, la prueba sin redundancia no pudo completarse dentro del tiempo límite (*Timeout*), mientras que con redundancia (**222-redun.dzn**) el problema se resolvió en apenas **0,016 segundos**, evidenciando una optimización sustancial en el rendimiento.
- **Restricciones de Redundancia:** Se incluyeron restricciones de redundancia con el objetivo de mejorar la eficiencia del proceso de búsqueda. La primera redundancia impone que la suma total de los valores de la secuencia sea igual a n, lo cual es una consecuencia directa de la definición del problema (cada número aparece una cantidad específica de veces). La segunda es una ecuación de balance que refuerza la relación entre los índices y la cantidad de veces que aparecen, para tener filtro adicional que descarta combinaciones que no podrían formar una secuencia válida. Claramente estas restricciones no cambian el conjunto de soluciones posibles, sí reducen significativamente el espacio de búsqueda y mejoran el tiempo de resolución, especialmente en instancias con valores de n más altos.
- Para problemas con dominios amplios, no aplicar restricciones de redundancia podría hacer que la resolución sea impráctica debido al crecimiento exponencial del espacio de búsqueda. **Para valores muy grandes va a ser ineficiente sin usar redundancia**

- Dado que los árboles generados en estas pruebas son equivalentes en estructura (binarios y explorados por profundidad), la diferencia en rendimiento radica exclusivamente en la reducción del espacio de búsqueda gracias a la eliminación de valores redundantes.
- **Restricción de Simetría:** No se aplicaron restricciones de simetría, ya que el problema de la secuencia mágica tiene una estructura en la que cada valor depende de sí mismo: el número que está en cada posición indica cuántas veces debe aparecer ese mismo número en toda la secuencia. Esta relación interna hace que no se puedan aplicar transformaciones simétricas (como invertir la secuencia o cambiar valores de lugar), ya que eso cambiaría completamente el significado y rompería la lógica del problema. Por esta razón, no hay simetrías útiles que se puedan aprovechar o romper sin dañar la validez del modelo.

ACERTIJO LÓGICO

DESCRIPCIÓN DEL PROBLEMA

Tres amigos —Juan, Óscar y Darío— tienen cada uno un apellido diferente (González, López o García), una edad distinta entre 24 y 26 años, y gustos musicales distintos (clásica, pop o jazz). A partir de cinco pistas que relacionan estos atributos, se debe deducir qué combinación de nombre, apellido, edad y gusto musical corresponde a cada persona. Las pistas incluyen comparaciones de edad, exclusiones y relaciones directas entre atributos, y el objetivo es encontrar una asignación única y coherente para los tres amigos.

DEFINICIÓN FORMAL DEL MODELO

CONSTANTES

Este modelo no utiliza parámetros porque resuelve una **instancia específica** del problema, con un conjunto fijo de personas, apellidos, edades y gustos musicales. Para usar parámetros sería necesario formular una versión **generalizada**, donde dichos elementos pudieran variar y ser definidos externamente. En este caso, la naturaleza cerrada del acertijo hace innecesaria esa generalización.

$Nombres:$ Enumerador de nombres de los tres amigos {Juan, Óscar, Darío}.
$Apellidos:$ Enumerador de apellidos posibles {González, García, López}.
$Generos\ :$ Enumerador de géneros musicales {Clásica, Pop, Jazz}.

VARIABLES

$Edad$: Variable entera que representa la edad de cada persona.

$$Edad[n] : n \in Nombres \rightarrow \{24, 25, 26\}$$

$Apellido$: Variable simbólica que representa el apellido asignado a cada persona.

$$Apellido[n] : n \in Nombres \rightarrow Apellidos$$

$Genero$: Variable simbólica que representa el género musical preferido de cada persona.

$$Genero[n] : n \in Nombres \rightarrow Generos$$

RESTRICCIONES

- Las edades asignadas a las personas deben ser todas distintas.

$$alldifferent([Edad[n] \mid n \in Nombres])$$

- Cada persona debe tener un apellido diferente.

$$alldifferent([Apellido[n] \mid n \in Nombres])$$

- A cada persona le gusta un género musical distinto.

$$alldifferent([Genero[n] \mid n \in Nombres])$$

- Juan es mayor que la persona de apellido González, a quien le gusta la música clásica.

$$\forall n \in Nombres: Apellido[n] = Gonzalez \Rightarrow (Edad[Juan] > Edad[n] \wedge Genero[n] = Clasica)$$

- El fan del pop no se apellida García ni tiene 24 años.

$$\exists n \in Nombres: Genero[n] = Pop \wedge Apellido[n] \neq Garcia \wedge Edad[n] \neq 24$$

- Óscar tiene 25 años.

$$Edad[Oscar] = 25$$

- Óscar no se apellida López.

$$Apellido[Oscar] \neq Lopez$$

- A Darío no le gusta el jazz.

$$Genero[Dario] \neq Jazz$$

RESTRICCIONES REDUNDANTES

Con el objetivo de mejorar la propagación y eficiencia del modelo, se planteó la inclusión de restricciones redundantes que refuerzan propiedades ya garantizadas por las restricciones principales (alldifferent). Las restricciones redundantes añadidas fueron las siguientes:

- Cada apellido debe ser asignado a una persona.

$$\forall a \in Apellidos : \exists n \in Nombres : Apellido[n] = a$$

- Cada género musical debe estar presente en la solución.

$$\forall g \in Generos : \exists n \in Nombres : Genero[n] = g$$

- Cada edad debe estar asignada a una persona.

$$\forall e \in \{24, 25, 26\} : \exists n \in Nombres : Edad[n] = e$$

Sin embargo, tras realizar pruebas comparativas con y sin dichas restricciones, se observó que:

- La solución final del modelo se mantiene igual.
- La cantidad de nodos explorados y fallos permanece sin cambios.
- La cantidad de propagaciones se incrementa significativamente

Finalmente se decide no implementar estas restricciones en la versión final del modelo.

ESTRATEGIAS DE BÚSQUEDA

Para las pruebas sobre el modelo de Acertijo Lógico se utilizaron distintas estrategias de heurísticas de selección de variables

- *first_fail,*
- *dom_w_deg*
- *input_order*
- *satisfy (que el solver decida por sí mismo)*

Combinadas con heurísticas de selección de valores:

- *indomain_min*
- *indomain_split*
- *indomain_median*
- *satisfy (que el solver decida por sí mismo)*

Estas combinaciones se evaluaron utilizando dos solvers diferentes: **Gecode** y **Chuffed**, con el fin de analizar si el uso de un solver distinto generaba alguna diferencia significativa en el rendimiento del modelo, tanto en tiempo de resolución

como en la cantidad de nodos explorados, fallos, profundidad del árbol de búsqueda y soluciones.

PRUEBAS

Se realizaron 12 pruebas para evaluar el rendimiento del modelo, resultantes de combinar dos solvers (Gecode y Chuffed) con seis estrategias de búsqueda, formadas por diferentes combinaciones heurísticas de selección de variables y valores.

Durante las pruebas se midieron métricas como el tiempo de resolución, la cantidad de nodos explorados, los fallos y la cantidad de soluciones encontradas.

En todos los casos, el modelo encontró una única solución correcta, y el comportamiento fue estable entre estrategias, con pequeñas diferencias en tiempo. Se observó que Gecode realiza más propagaciones que Chuffed, aunque ambos resolvieron eficientemente el problema.

Estas pruebas permitieron verificar la correcta ejecución del modelo y analizar el impacto de distintas configuraciones sobre su rendimiento.

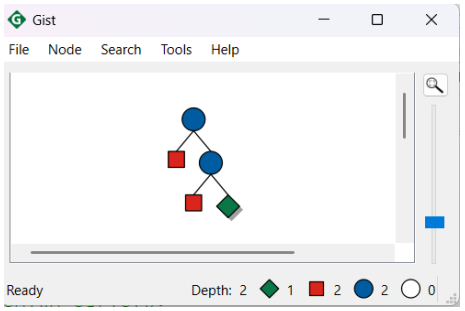
Se puede observar que en las pruebas realizadas el mejor tiempo fue para el solver Gecode, con la estrategia *dom_w_deg/indomain_min.*

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	totalTime	solutions	nodes	failures	peakDepth
acertijo.mzn	Gecode	satisfy	satisfy	0,0005922	1	5	2	1
acertijo.mzn	Gecode	input_order	indomain_min	0,0006714	1	5	2	1
acertijo.mzn	Gecode	first_fail	indomain_min	0,0005938	1	5	2	1
acertijo.mzn	Gecode	first_fail	indomain_median	0,000846	1	5	2	1
acertijo.mzn	Gecode	first_fail	indomain_split	0,0006877	1	5	2	1
acertijo.mzn	Gecode	dom_w_deg	indomain_min	0,0005843	1	5	2	1
acertijo.mzn	Chuffed	satisfy	satisfy	0,002	1	4	3	2
acertijo.mzn	Chuffed	input_order	indomain_min	0,002	1	4	3	2
acertijo.mzn	Chuffed	first_fail	indomain_min	0,002	1	4	3	2
acertijo.mzn	Chuffed	first_fail	indomain_median	0,002	1	4	3	2
acertijo.mzn	Chuffed	first_fail	indomain_split	0,002	1	4	3	2
acertijo.mzn	Chuffed	dom_w_deg	indomain_min	0,002	1	4	3	2

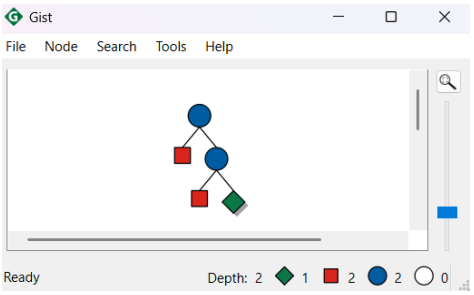
ÁRBOLES GENERADOS

Al visualizar los árboles de búsqueda generados por el solver Gecode con diferentes estrategias (mediante Gist), se observó que la estructura del árbol fue prácticamente idéntica en todos los casos. Esto se debe a que el modelo tiene un espacio de búsqueda pequeño y fuertemente restringido, lo cual permite que la propagación reduzca drásticamente la necesidad de decisiones, haciendo que distintas estrategias recorran efectivamente el mismo camino hacia la solución.

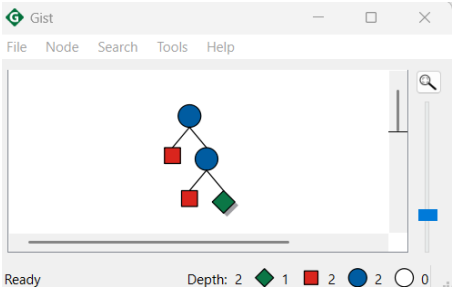
satisfy/satisfy



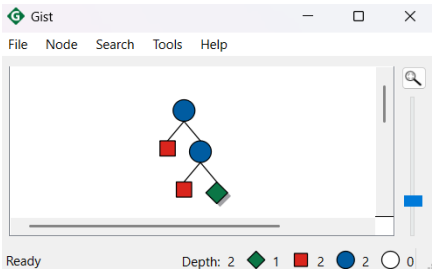
input_order/indomain_min



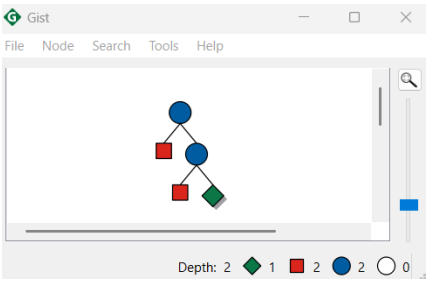
first_fail/indomain_min



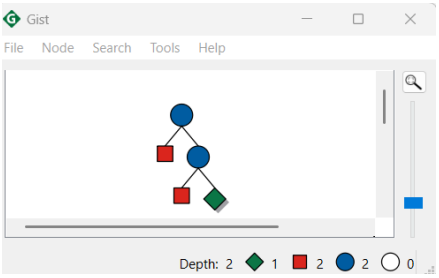
first_fail/indomain_median



first_fail/indomain_split



first_fail/indomain_split



ANÁLISIS

El ejercicio permitió evaluar el comportamiento del modelo bajo distintas estrategias y configuraciones. Se comprobó que la solución es única y estable en todos los casos, y que las estrategias de búsqueda no generaron diferencias significativas en cuanto a nodos o fallos. Aunque se esperaban mejoras con restricciones redundantes, estas solo incrementaron la carga de propagación sin beneficios en eficiencia, por lo que se descartaron. En general, el modelo mostró un buen desempeño y robustez, validando su formulación adecuada para este tipo de problemas.

Se pudo observar que el solver Chuffed presentó un desempeño superior en comparación con Gecode, especialmente en términos de eficiencia en la

propagación de restricciones, así como en la reducción de la cantidad de nodos explorados y fallos durante la búsqueda.

UBICACIÓN DE PERSONAS EN UNA REUNIÓN

DESCRIPCIÓN DEL PROBLEMA

Un grupo de personas se va a tomar una foto organizándose en una fila. Algunas personas expresan preferencias sobre su posición relativa con respecto a otras, como querer estar justo al lado de alguien, no estar juntas, o estar separadas por un número máximo de personas. A partir de estas preferencias, se debe encontrar un orden válido en la fila que cumpla todas las condiciones especificadas. Las entradas incluyen la lista de personas y tres tipos de restricciones: estar juntos, estar separados y mantener una distancia máxima.

DEFINICIÓN FORMAL DEL MODELO

PARÁMETROS

- n: Cantidad total de personas que asistirán a la reunión.
- $personas$: Lista de nombres que identifica a cada persona.
- $next$: La persona A debe estar junto a la persona B en la fotografía.

$$next(A, B)\ ,\quad A, B\ \in\ [1, n]$$

- $separate$: Pares de personas que deben estar separadas.

$$separate(A, B),\quad A, B\ \in\ [1, n]$$

- $distance$: Conjunto de triples que especifican una distancia m entre dos personas.

$$distance(A, B, M),\quad A, B, M\ \in\ [1, n]$$

VARIABLE

- $posicion$: Representa la ubicación de cada persona en la fila para la fotografía. Está modelada como un arreglo de tamaño n, donde cada entrada $posicion[i]$ indica la posición asignada a la persona i. Es una variable de decisión cuyo dominio está definido en el intervalo de 1 a n, es decir:

$$\forall i\ \in\ [1, n],\ posicion_{i}\ \in\ [1, n]$$

RESTRICCIONES

- **Posiciones distintas para cada persona:** Todas las personas deben ocupar una posición única en la fila.

$$alldifferent([posicion_i \mid i \in [1,\ n]])$$

- **Restricción de adyacencia ($next$):** Para cada par de personas (A, B) que deben estar juntas, su diferencia de posición debe ser 1.

$$\forall (A, B) \in [1, n],\ |posicion_A - posicion_B| = 1$$

- **Restricción de separación ($separate$):** Para cada par de personas (A, B) que deben estar separadas, su diferencia de posición debe ser mayor que 1.

$$\forall (A, B) \in [1, n],\ |posicion_A - posicion_B| > 1$$

- **Restricción de distancia máxima ($distance$):** Para cada triple $(A, B,\ M)$, la distancia entre A y B no debe superar el valor M.

$$\forall (A, B, M) \in [1, n],\ |posicion_A - posicion_B| \leq M$$

RESTRICCIONES REDUNDANTES

Con el objetivo de mejorar la eficiencia del modelo, se añadieron restricciones redundantes que permiten acotar el espacio de búsqueda al limitar el rango de asignaciones posibles. Estas restricciones imponen condiciones adicionales sobre las posiciones relativas de las personas, lo que facilita una propagación más efectiva de los dominios y reduce la cantidad de configuraciones exploradas por el solver.

- **Redundancia para la restricción $next$ en los bordes:** Cuando una persona del par $next$ se ubica en uno de los extremos de la fila (posición 1 o n), se impone que la otra persona se ubique en la única posición adyacente disponible.

$$\forall A, B \in [1, n],\ (posicion_A = 1 \rightarrow posicion_B = 2) \wedge (posicion_A = n \rightarrow posicion_B = n - 1)$$

- **Redundancia para la restricción $separate$:** Se refuerza explícitamente que las personas que deben estar separadas tengan una distancia estrictamente mayor a uno, evaluando ambas direcciones.

$$\forall (A, B) \in [1, n],\ (posicion_A - posicion_B > 1) \vee (posicion_B - posicion_A > 1)$$

RESTRICCIÓN PARA ROMPER SIMETRÍA

Para evitar que el modelo explore soluciones que sean reflejos entre sí, se agregó una restricción que ordena parcialmente las posiciones cuando las relaciones en $next$ son simétricas. Es decir, si no existen pares en $next$ con direcciones distintas (es decir, todos los pares son reflejos entre sí), se impone que la persona 1 debe estar antes que la persona n en la fila.

$$\forall (A, B) \in [1, n]\ si\ \exists (B, A) \in [1, n] \rightarrow posicion_1 < posicion_n$$

ESTRATEGIAS DE BÚSQUEDA

Para las pruebas sobre el modelo de Sudoku se utilizaron distintas estrategias de heurísticas de selección de variables

- *first_fail,*
- *dom_w_deg*
- *input_order*
- *satisfy (que el solver decida por sí mismo)*

Combinadas con heurísticas de selección de valores:

- *indomain_min*
- *indomain_split*
- *indomain_median*
- *satisfy (que el solver decida por sí mismo)*

Estas combinaciones se evaluaron utilizando dos solvers diferentes: **Gecode** y **Chuffed**, con el fin de analizar si el uso de un solver distinto generaba alguna diferencia significativa en el rendimiento del modelo, tanto en tiempo de resolución como en la cantidad de nodos explorados, fallos, profundidad del árbol de búsqueda y soluciones

PRUEBAS

Se realizaron un total de 216 pruebas de las cuales 108 se hicieron ambas con simetría y con redundancia y 108 con simetría y sin redundancia, resultado de combinar 4 heurísticas de selección de variables con 4 heurísticas de selección de valores, para cada solver: Gecode y Chuffed. Para cada combinación de estrategias

y solver se ejecutaron 6 pruebas, lo que permitió observar el comportamiento del modelo bajo diversas configuraciones.

Durante las ejecuciones se recopilaron métricas relevantes como el tiempo de inicialización, tiempo de resolución, tiempo total de ejecución, así como el número de nodos explorados, fallos registrados y la profundidad máxima del árbol de búsqueda. Esta información sirvió de base para un análisis detallado del impacto de las estrategias sobre el rendimiento del modelo.

A partir de este conjunto de pruebas, se seleccionaron 18 combinaciones representativas de heurísticas en cada solver para ser analizadas con mayor detalle. Cada combinación fue ejecutada con ambos solvers, con el fin de comparar directamente su desempeño bajo las mismas condiciones. Los resultados se resumen en la siguiente tabla, en la que se destacan las diferencias en tiempo, nodos, fallos y profundidad alcanzada, evidenciando cómo influye el uso del solver y la estrategia de búsqueda en la eficiencia de resolución.

Total de pruebas reunión: Resultados Reunion.xlsx

PRUEBAS CON ROMPIMIENTO DE SIMETRÍA Y RESTRICCIONES DE REDUNDANCIA

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	totalTime	solutions	nodes	failures	peakDepth
example.dzn	Gecode	satisfy	satisfy	0,2483656	3168	6335	0	10
example.dzn	Chuffed	satisfy	satisfy	0,252	3168	3787	3202	9
ejemplo2.dzn	Gecode	dom_w_deg	indomain_min	0,0008177	12	31	4	5
ejemplo2.dzn	Chuffed	satisfy	satisfy	0,008	12	30	27	6
example3.dzn	Gecode	satisfy	satisfy	0,028466	580	1159	0	9
example3.dzn	Chuffed	input_order	indomain_min	0,041	580	704	580	7
example4.dzn	Gecode	satisfy	satisfy	1,411809	14976	31939	994	14
example4.dzn	Chuffed	first_fail	indomain_median	1,443	14976	17018	15186	9
example5.	Gecode	satisfy	satisfy	0,683465	6336	12691	10	13

dzn				2				
example5.dzn	Chuffed	satisfy	satisfy	0,696	6336	7157	6528	11
example6.dzn	Gecode	input_order	indomain_min	0,0008847	8	27	6	3
example6.dzn	Chuffed	first_fail	indomain_median	0,008	8	16	16	2
example7.dzn	Gecode	satisfy	satisfy	0,0013355	64	127	0	6
example7.dzn	Chuffed	first_fail	indomain_median	0,008	64	102	90	5
example8.dzn	Gecode	dom_w_deg	indomain_min	0,0012154	56	115	2	5
example8.dzn	Chuffed	first_fail	indomain_median	0,009	56	102	88	6
example9.dzn	Gecode	dom_w_deg	indomain_min	0,0013196	76	151	0	5
example9.dzn	Chuffed	satisfy	satisfy	0,009	76	106	89	6

PRUEBAS CON ROMPIMIENTO DE SIMETRÍA Y SIN RESTRICCIONES DE REDUNDANCIA

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	totalTime	solutions	nodes	failures	peakDepth
example.dzn	Gecode	dom_w_deg	indomain_min	0,2437343	3168	6335	0	10
example.dzn	Chuffed	satisfy	satisfy	0,247	3168	3213	3205	8
ejemplo2.dzn	Gecode	satisfy	satisfy	0,0006618	12	33	5	4
ejemplo2.dzn	Chuffed	first_fail	indomain_median	0,002	12	20	20	4
example3.dzn	Gecode	input_order	indomain_min	0,0276105	580	1159	0	9
example3.dzn	Chuffed	dom_w_deg	indomain_min	0,036	580	592	592	6
example4.dzn	Gecode	satisfy	satisfy	1,5286373	14976	31155	602	14
example4.dzn	Chuffed	first_fail	indomain_median	1,453	14976	15194	15191	8

example5.dzn	Gecode	satisfy	satisfy	0,7042392	6336	12691	10	13
example5.dzn	Chuffed	satisfy	satisfy	0,694	6336	6553	6514	10
example6.dzn	Gecode	satisfy	satisfy	0,0006732	8	21	3	3
example6.dzn	Chuffed	first_fail	indomain_median	0,002	8	21	19	4
example7.dzn	Gecode	satisfy	satisfy	0,0013726	64	129	1	6
example7.dzn	Chuffed	satisfy	satisfy	0,002	64	108	105	7
example8.dzn	Gecode	satisfy	satisfy	0,0010554	56	117	3	6
example8.dzn	Chuffed	satisfy	satisfy	0,002	56	77	72	6
example9.dzn	Gecode	satisfy	satisfy	0,0011813	76	151	0	6
example9.dzn	Chuffed	satisfy	satisfy	0,002	76	114	108	6

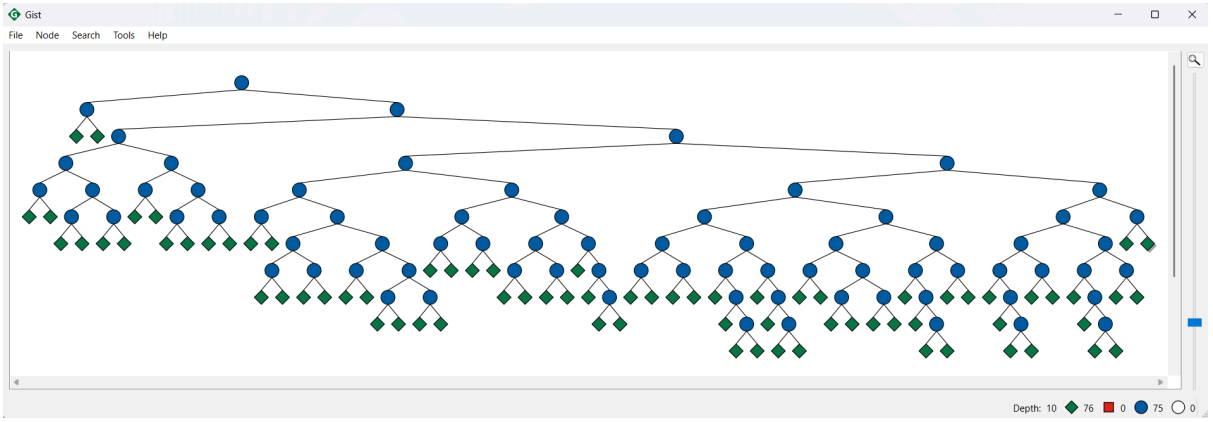
ÁRBOLES GENERADOS

A partir de los ejemplos previamente seleccionados en la tabla, se generan los árboles de búsqueda utilizando Gecode Gist. **Sin restricciones de redundancia y con restricciones de redundancia respectivamente, ambas con simetría.**

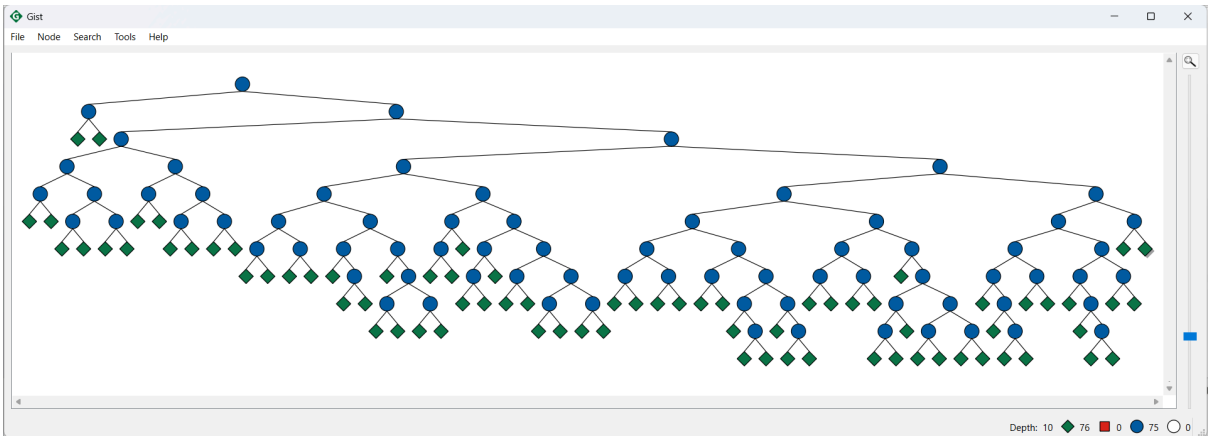
Se toman las pruebas con n = 6 para facilitar la visualización.

N = 6 Pares next = 1 Pares separate = 1 Triples distance = 1

Sin redundancia

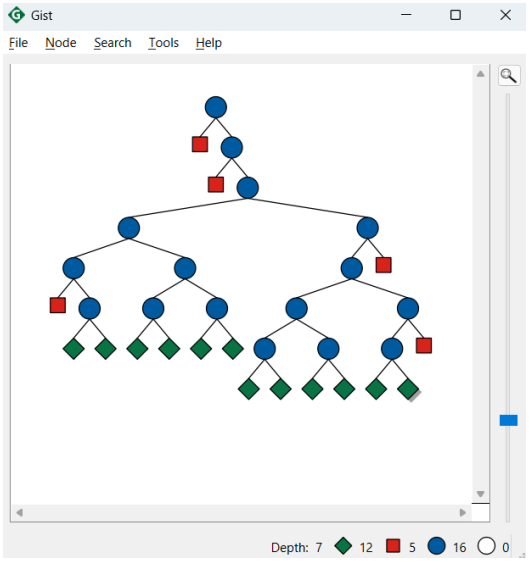


Con redundancia

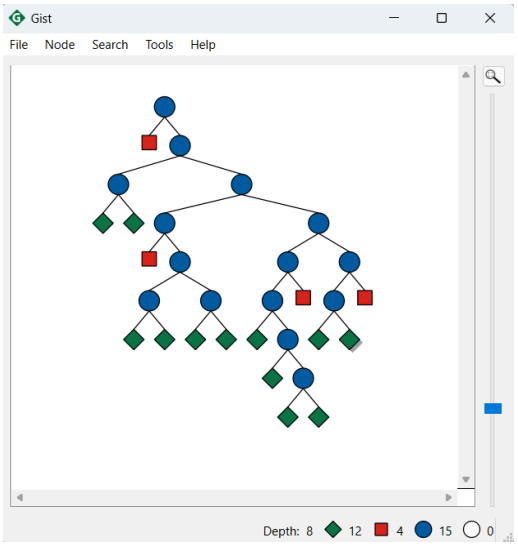


N = 6 Pares next = 2 Pares separate = 1 Triples distance = 1

Sin redundancia

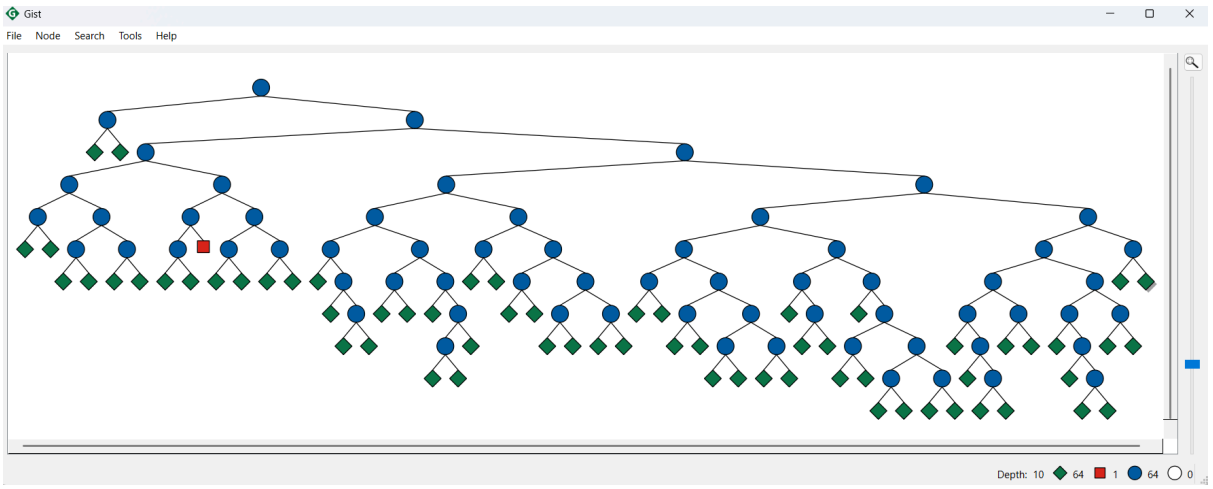


Con redundancia

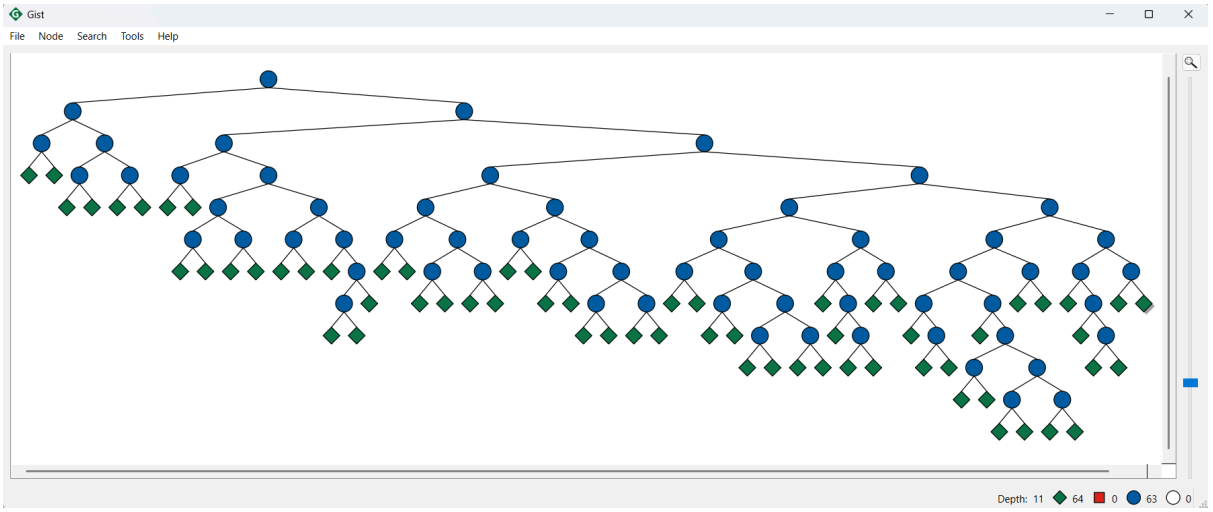


N = 6 Pares next = 1 Pares separate = 2 Triples distance = 1

Sin redundancia

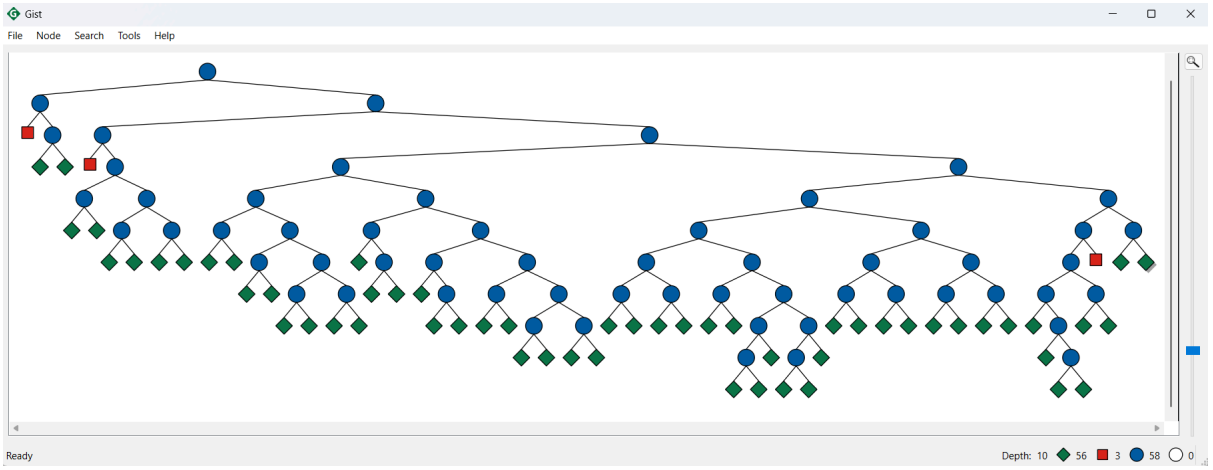


Con redundancia

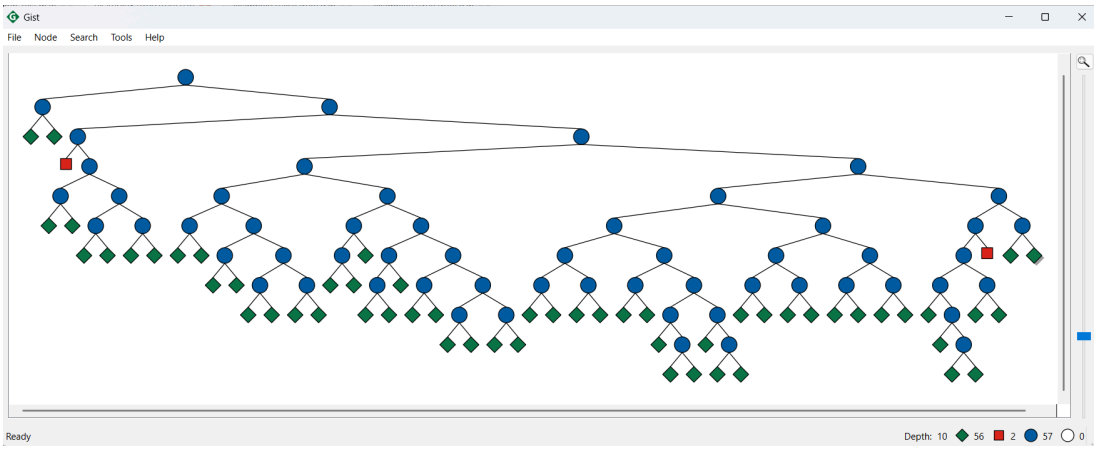


N = 6 Pares next = 1 Pares separate = 1 Triples distance = 2

Sin redundancia

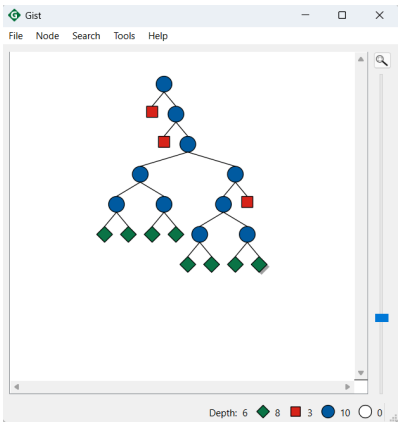


Con redundancia

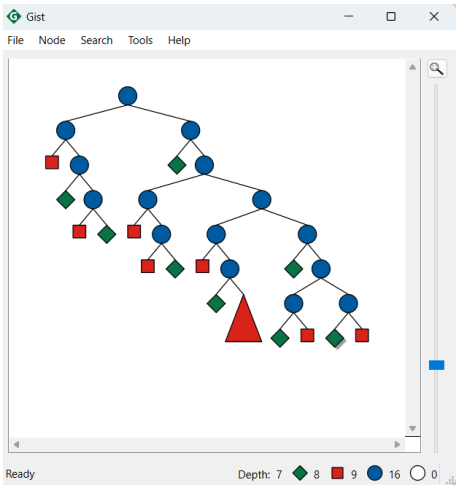


N = 6 Pares next = 2 Pares separate = 2 Triples distance = 2

Sin redundancia



Con redundancia

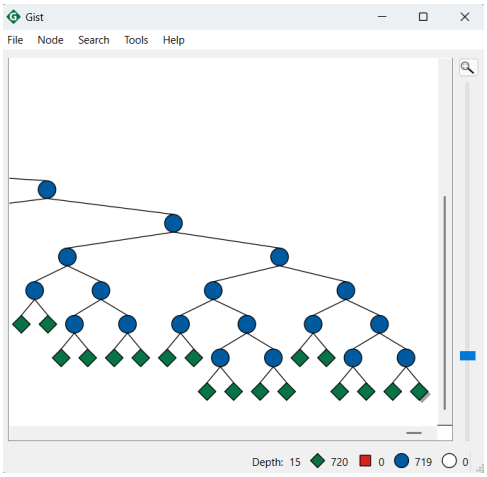


ANÁLISIS

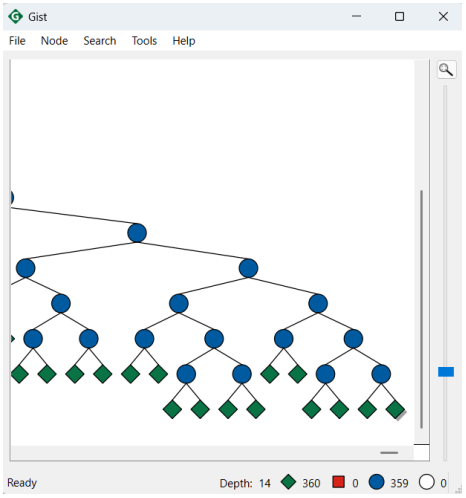
- El solver Chuffed demuestra un rendimiento superior en comparación con Gecode, dado por una reducción en la cantidad de nodos desplegados durante el proceso de resolución. En varios casos, Chuffed requiere explorar más o menos la mitad de nodos que Gecode para alcanzar soluciones, lo que representa una mejora.
- Los resultados anteriores muestran que la estrategia "satisfy / satisfy" destaca por su eficacia tanto en Gecode como en Chuffed, logrando resolver múltiples instancias con una exploración mínima de nodos y pocos fallos.
- **Restricciones de redundancia:** La incorporación de restricciones redundantes logró disminuir el número de nodos explorados en varias pruebas, se identificaron situaciones donde ocurrió lo contrario, aumentando la cantidad de nodos generados. Esto evidencia que el impacto de las restricciones redundantes puede variar según la combinación de heurísticas y el solver utilizado, y que su efectividad no siempre es garantizada.
- **Restricción de simetría:** El objetivo de la restricción de simetría es disminuir el espacio de búsqueda del modelo y sus soluciones, impidiendo que se produzcan soluciones que sean reflejos o configuraciones que se asemejen entre sí. Esto ocurre cuando las relaciones de proximidad entre individuos son totalmente iguales, es decir, cuando para cada par también existe su opuesto. En tales situaciones, se establece una ordenación parcial en la fila, determinando que un individuo específico se ubique antes que otro, con el objetivo de romper esa simetría y descartar soluciones espejo que no añaden valor extra al conjunto de respuestas. Esta técnica no altera la validez del modelo y puede mejorar significativamente la eficiencia del solver, especialmente en instancias donde las relaciones no dependen del orden.

Para comprobar de que la restricción de simetría funciona se hace con un ejemplo sin esta y sin condiciones next, separate y distance, es decir, de que las personas se organicen tal como llegan, esto causará una combinatoria de 6!. Ahora se hace una prueba igual que la anterior pero ya con la restricción de simetría, este se reduce a la mitad, evidenciando de que se eliminan las organizaciones o soluciones espejo

Sin simetría.



Con simetría.



Para así corroborar, que muestra rompimiento para eliminar espejos funciona.

CONSTRUCCIÓN DE UN RECTÁNGULO

DESCRIPCIÓN DEL PROBLEMA

Se cuenta con un conjunto de cuadrados de diferentes tamaños (algunos iguales entre sí), y el objetivo es organizarlos, sin superposición, para formar un rectángulo de dimensiones específicas (ancho y alto). Cada cuadrado debe colocarse completamente dentro del rectángulo, con sus lados paralelos a los ejes X y Y, y todos deben ser utilizados.

El reto consiste en encontrar una disposición válida para todos los cuadrados que permita formar el rectángulo sin dejar espacios vacíos ni sobreponer figuras.

DEFINICIÓN FORMAL DEL MODELO

PARÁMETROS

n : Número total de cuadrados a ubicar dentro del rectángulo.

W: Ancho del rectángulo que se debe construir. Define el número total de columnas disponibles para ubicar los cuadrados.

H: Altura del rectángulo. Define el número total de filas disponibles para ubicar los cuadrados.

$sizes[]$: Arreglo de tamaño n, donde cada elemento $sizes[i]$ representa el lado del cuadrado número i.

VARIABLES

$x[i]$: Coordenada horizontal (eje X) de la esquina inferior izquierda del cuadrado i, con dominio {0, ..., W - 1}.

$y[i]$: Coordenada vertical (eje Y) de la esquina inferior izquierda del cuadrado i, con dominio {0, ..., H - 1}.

RESTRICCIONES

- Cada cuadrado debe estar completamente contenido dentro de los límites del rectángulo:

$$\forall i \in [1,n]:\ x[i] + sizes[i] \leq W \wedge y[i] + sizes[i] \leq H$$

- Ningún par de cuadrados puede solaparse entre sí. Para cada par de cuadrados $i \neq j$, se impone:

  $$x[i] + sizes[i] \leq x[j] \vee x[j] + sizes[j] \leq x[i] \vee y[i] + sizes[i] \leq y[j] \vee y[j] + sizes[j] \leq y[i]$$

- Esta restricción garantiza que el área total del rectángulo sea igual a la suma de las áreas de todos los cuadrados, asegurando así que el conjunto de cuadrados proporcionado pueda cubrir completamente el rectángulo sin dejar espacios vacíos.

$$W \cdot H = \sum_{i=1}^{n} sizes[i]^2$$

RESTRICCIONES ROMPIMIENTO DE SIMETRÍA

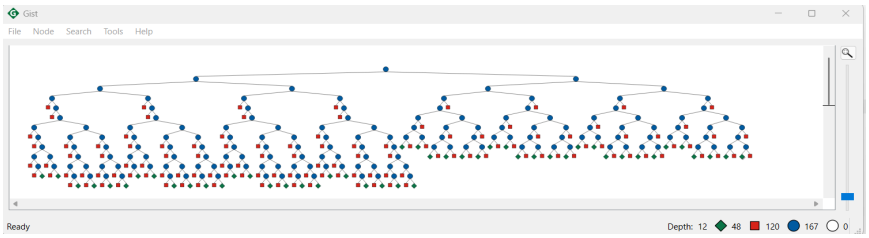
- Para reducir la cantidad de soluciones equivalentes producidas por permutaciones de cuadrados iguales, se impone un orden lexicográfico entre sus coordenadas $(x[i],\ y[i])$

  Formalmente, si dos cuadrados i y j tienen el mismo tamaño, se debe cumplir:

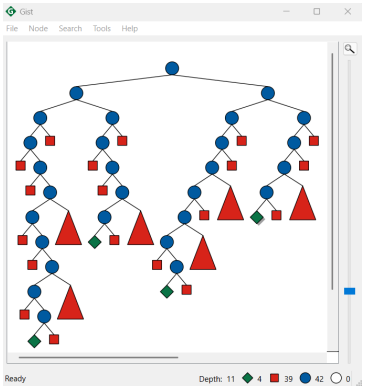
$$lex\_lesseq([x[i],\ y[i]],\ [x[j],\ y[j]]) \textbf{ si } sizes[i] = sizes[j] \ \wedge \ i < j$$

PRUEBAS RESTRICCIÓN DE ROMPIMIENTO DE SIMETRÍA Y SIN RESTRICCIÓN DE ROMPIMIENTO DE SIMETRÍA

ejemplo1.dzn:
Sin restricción de ROMPIMIENTO DE SIMETRÍA:

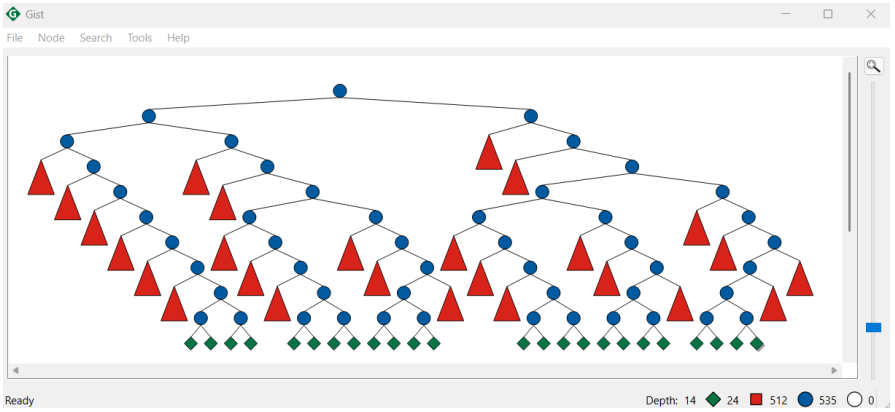


Con restricción de simetría:

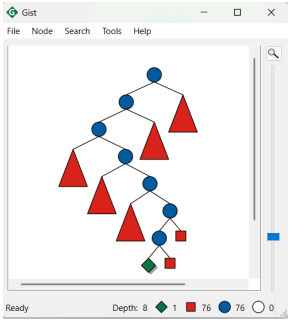


ejemplo2.dzn:

Sin restricción de simetría:

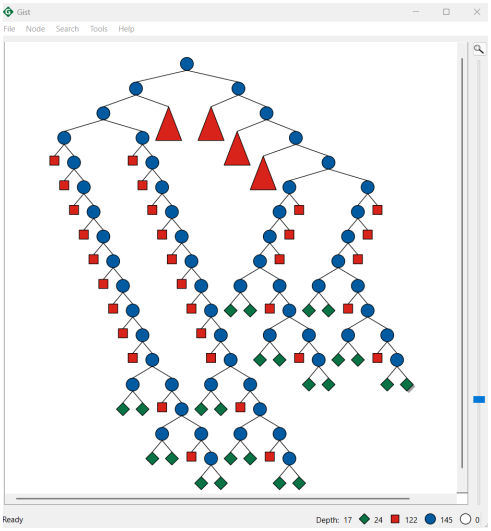


Con restricción de simetría:

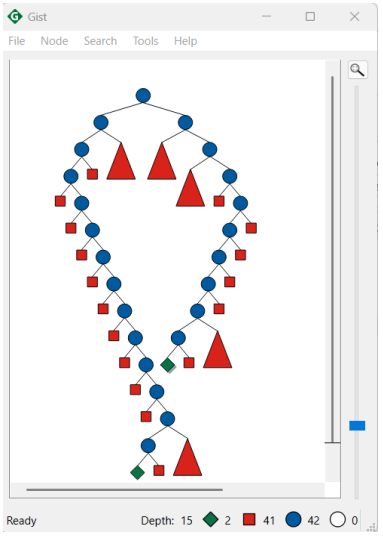


ejemplo3.dzn:

Sin restricción de simetría:

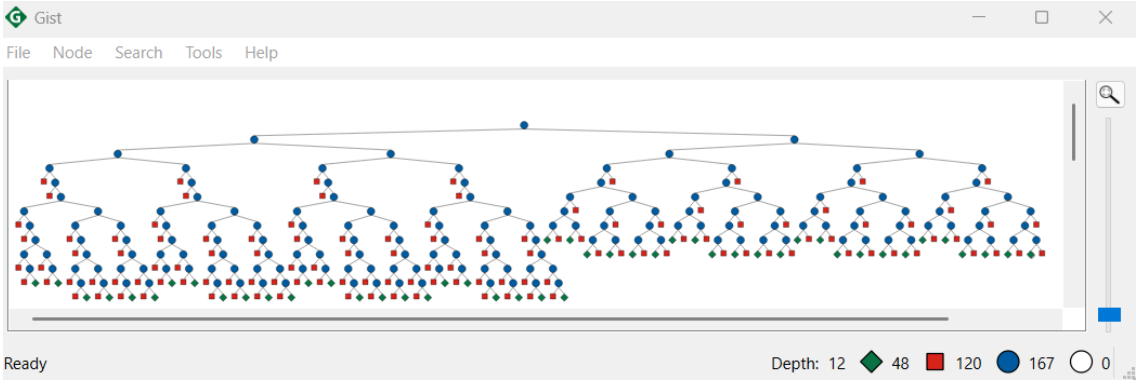


Con restricción de simetría:

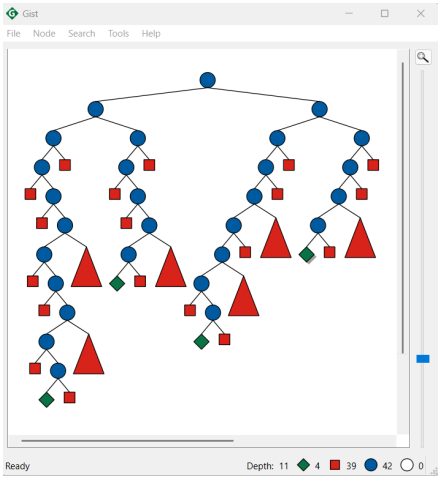


ejemplo5.dzn:

Sin restricción de simetría:



Con restricción de simetría:



PRUEBAS

Se realizaron cinco pruebas distintas con el modelo rectangulo.mzn, evaluando su rendimiento bajo diferentes combinaciones de solver y estrategia. Se utilizaron los solvers Gecode y Chuffed, junto con varias heurísticas de selección de variables y valores, lo que resultó en un total de 44 ejecuciones.

Durante las pruebas se registraron métricas como el tiempo de resolución, la cantidad de nodos explorados, los fallos y la profundidad del árbol de búsqueda. En todos los casos, el modelo encontró soluciones válidas, excepto para la instancia ejemplo4.dzn, que fue correctamente identificada como insatisfactible.

Aunque todas las estrategias generaron la misma cantidad de soluciones por instancia, se observaron diferencias importantes en el rendimiento. En particular, el solver Chuffed con la estrategia input_order / indomain_min obtuvo el mejor tiempo de resolución. La estrategia satisfy, sin heurísticas específicas, también mostró buen desempeño general.

Estas pruebas confirmaron la correcta ejecución del modelo y permitieron identificar el impacto real de cada configuración sobre su eficiencia.

Además, se generó una segunda tabla de pruebas aplicando la restricción de rompimiento de simetría mediante comparación lexicográfica entre las posiciones de los cuadrados del mismo tamaño. Esto permitió comparar de forma directa el impacto de dicha restricción sobre el espacio de búsqueda.

En general, se observó que al incluir el rompimiento de simetría:

- Se redujo considerablemente la cantidad de soluciones reportadas, eliminando configuraciones equivalentes por permutación.

- Disminuyeron significativamente el número de nodos explorados, fallos y la profundidad del árbol de búsqueda.

- El tiempo de resolución también mejoró en varios casos, especialmente con estrategias heurísticas.

Estas pruebas confirmaron la correcta ejecución del modelo y permitieron identificar de forma clara el beneficio de aplicar técnicas de optimización declarativa, como el rompimiento de simetría, para mejorar la eficiencia del proceso de resolución sin afectar la validez de los resultados.

Tabla de pruebas completa: Resultados Rectangulo

PRUEBAS SIN RESTRICCIÓN DE SIMETRÍA

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	solutions	nodes	failures	peakDepth
ejemplo1	Gecode	satisfy	satisfy	48	195	50	11
ejemplo1	Chuffed	input_order	indomain_min	48	97	79	6
ejemplo2	Gecode	satisfy	satisfy	24	143	48	10
ejemplo2	Chuffed	satisfy	satisfy	24	58	52	7
ejemplo3	Gecode	satisfy	satisfy	24	193	73	8
ejemplo3	Chuffed	input_order	indomain_min	24	87	81	5
ejemplo5	Gecode	satisfy	satisfy	48	195	50	11
ejemplo5	Chuffed	input_order	indomain_min	48	97	79	6

PRUEBAS CON RESTRICCIÓN DE SIMETRÍA

Archivo	Solver	Estrategia (var_heur)	Estrategia (val_heur)	solutions	nodes	failures	peakDepth
ejemplo1	Gecode	satisfy	satisfy	4	63	28	8
ejemplo1	Chuffed	first_fail	indomain_split	4	22	21	6
ejemplo2	Gecode	satisfy	satisfy	1	23	11	6
ejemplo2	Chuffed	satisfy	satisfy	1	12	9	5
ejemplo3	Gecode	dom_w_deg	indomain_min	2	41	19	6
ejemplo3	Chuffed	first_fail	indomain_min	2	17	13	5
ejemplo5	Gecode	satisfy	satisfy	4	63	28	8
ejemplo5	Chuffed	first_fail	indomain_split	4	22	21	6

ANÁLISIS

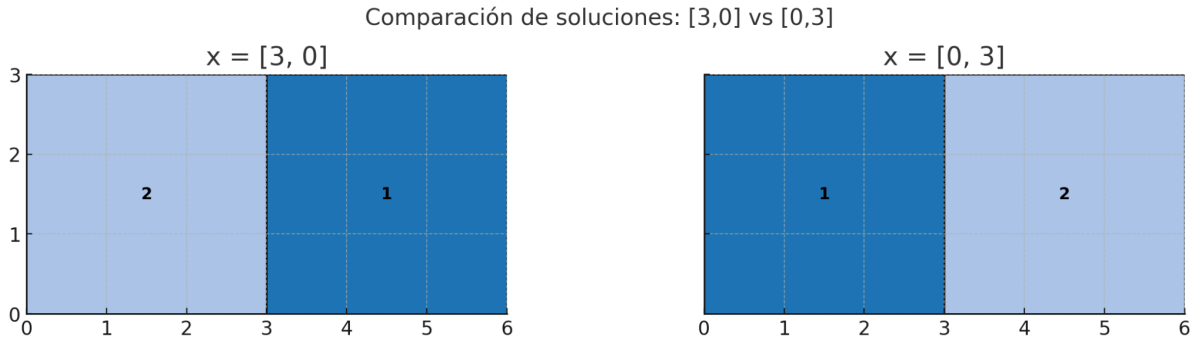
El ejercicio permitió analizar el comportamiento del modelo bajo distintas configuraciones de solver, estrategia de búsqueda y presencia o ausencia de restricciones de rompimiento de simetría. Se comprobó que el modelo es correcto y estable, ya que en todas las instancias válidas encontró soluciones consistentes. Además, permitió comparar el impacto real del uso de restricciones adicionales sobre el rendimiento computacional del modelo.

- En una primera fase se realizaron pruebas sin aplicar rompimiento de simetría. En este escenario, aunque se encontraron múltiples soluciones, muchas de ellas eran equivalentes desde el punto de vista estructural, lo que incrementó el número de nodos explorados, fallos y la profundidad del árbol.

- Posteriormente, se aplicó una restricción de rompimiento de simetría basada en el orden lexicográfico entre cuadrados del mismo tamaño, con el objetivo de evitar configuraciones redundantes. Al realizar esta segunda batería de pruebas, se observó una disminución general en el número de soluciones, nodos, fallos y profundidad, lo cual evidencia que dicha restricción tuvo un impacto positivo en la eficiencia del modelo.

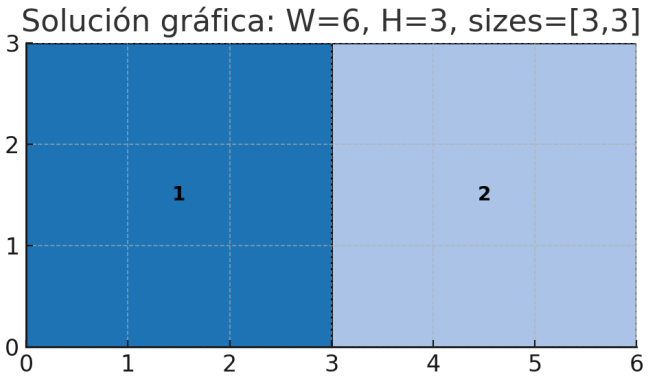
- Para ilustrar el efecto del rompimiento de simetría de forma clara y controlada, se utilizó una instancia mínima con dos cuadrados de tamaño 3×3 dentro de un rectángulo de dimensiones 6×3. Esta configuración permite ubicar los cuadrados exactamente uno al lado del otro, en posiciones opuestas pero simétricas.

Al ejecutar el modelo sin la restricción de simetría, se observaron dos soluciones distintas:

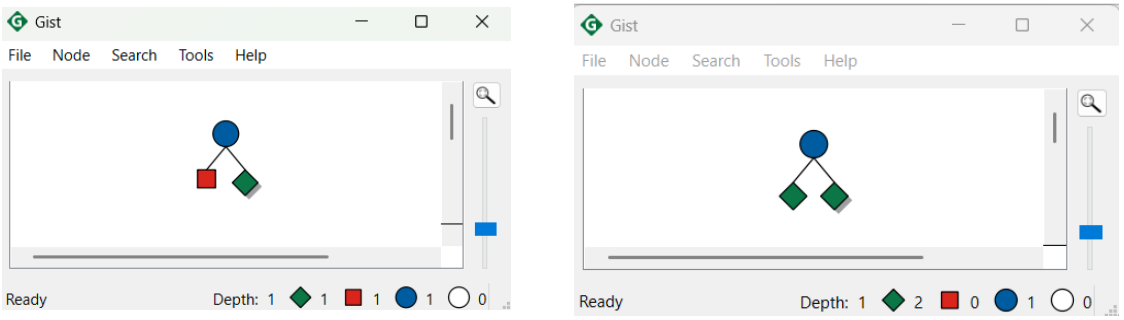


Desde el punto de vista del solver, son dos configuraciones diferentes. Sin embargo, geométricamente son equivalentes: una es el reflejo horizontal de la otra dentro del rectángulo. Este tipo de simetría no aporta valor adicional a la solución, pero sí duplica el trabajo de búsqueda.

Al activar la restricción de rompimiento de simetría, el modelo filtró automáticamente las configuraciones redundantes. Como resultado, se obtuvo solo una de las dos soluciones posibles



Lo anterior también se puede evidenciar en los árboles generados por gecode gist



- En este modelo se prefirió evitar el uso de restricciones redundantes, con el fin de no generar propagaciones adicionales que, en este caso, no aportaban

mejoras en el rendimiento. Esto se debe posiblemente a que la región factible ya se encuentra lo suficientemente acotada por las restricciones principales del modelo. Durante las pruebas preliminares se comprobó que dichas restricciones no reducían la cantidad de nodos explorados ni mejoraban los tiempos de resolución. Por esta razón, se optó por centrarse únicamente en evaluar el impacto de las estrategias de búsqueda y del rompimiento de simetría.

- Con base en el análisis de las tablas generadas, se pudo observar que el uso del solver Chuffed, en combinación con estrategias como *input_order / indomain_min* o *first_fail / indomain_split*, ofreció la menor carga de exploración del árbol en términos de nodos.

CONCLUSIONES

El desarrollo del presente taller nos permitió afianzar los conceptos vistos durante el curso sobre el paradigma de programación con restricciones, mediante el uso del IDE MiniZinc para resolver problemas combinatorios. A través de los modelos desarrollados, pudimos evidenciar el potencial del paradigma y de la herramienta.

Durante el taller se evaluaron distintas estrategias de búsqueda y heurísticas aplicadas en dos modelos, y fue evidente que su impacto varía significativamente según el problema específico que se esté resolviendo. Se observó con claridad que la combinación entre el modelo, el solver y la estrategia aplicada influye directamente en el comportamiento del árbol de búsqueda, afectando tanto la propagación como el número de nodos explorados, fallos y profundidad. Esto demuestra que una estrategia efectiva para un modelo puede no serlo para otro, y que no existe una configuración óptima universal. Por lo tanto, la elección adecuada del solver y de las heurísticas debe hacerse en función del tipo de problema, ya que puede mejorar de forma considerable el rendimiento del modelo y reducir significativamente los recursos computacionales requeridos.

Se pudo observar que el uso de restricciones redundantes no siempre genera beneficios en cuanto al rendimiento del modelo. En algunos casos, estas restricciones pueden contribuir a mejorar la propagación y reducir el esfuerzo de búsqueda, pero en otros, su inclusión no aporta mejoras significativas **e incluso** puede incluso aumentar el tiempo de resolución, la cantidad de nodos explorados, fallas y propagaciones, lo que se traduce en un mayor gasto computacional. Por ello, su uso debe evaluarse cuidadosamente según el comportamiento observado en cada modelo.

En la comparación entre los solvers implementados, aunque solo fue posible visualizar los árboles generados mediante gráficos en Gecode Gist, se observó que Chuffed generó árboles de búsqueda más reducidos en comparación con Gecode. Además, Chuffed aprovechó mejor las heurísticas de búsqueda definidas, mientras que Gecode, en general, utilizó su estrategia por defecto (*satisfy*), lo que limitó el impacto de las heurísticas personalizadas.

En cuanto a las restricciones de simetría, se tuvo cuidado al aplicarlas, ya que no eran adecuadas para todos los problemas. Solo se implementaron en los casos del problema de los rectángulos y el problema de la reunión. En el de la reunión, se podían eliminar simetrías mediante reflexiones o rotaciones de 180°, reduciendo el espacio de búsqueda a la mitad. En el caso de los rectángulos, algunas soluciones podrían representarse de formas equivalentes, por lo que se eliminaron hasta 1/4 de las soluciones repetidas (reflexiones horizontales, verticales y rotaciones de 180° ya

que eran 2 dimensiones). Sin embargo, se procuró no eliminar soluciones válidas, manteniendo un orden específico en lugar de fijar valores concretos.

La propagación y la búsqueda variaron según el solver y la estrategia utilizada, lo cual se evidenció en la construcción de los árboles de búsqueda. En Gecode, el rendimiento fue mejor cuando el solver utilizó su estrategia por defecto, generando árboles más optimizados. Esto indica que maneja internamente una buena combinación entre propagación y búsqueda. Por otro lado, Chuffed respetó más las estrategias definidas externamente, pero aun así logró árboles más reducidos, lo que demuestra una propagación más eficiente en varios casos.