Taller 1 — Modelado y resolución de CSP en MiniZinc

Estudio de Sudoku, Kakuro, Secuencia Mágica, Acertijo Lógico, Reunión y Rectángulo

John Freddy Belalcazar
Samuel Galindo Cuevas
Nicolas Herrera Marulanda

16 de octubre de 2025

Índice

1. Sudoku **2**
1.1. Modelo . . . 2
1.2. Implementación . . . 2
1.3. Pruebas . . . 3
1.4. Árboles de búsqueda . . . 3
1.5. Análisis y conclusiones . . . 3

2. Kakuro **4**
2.1. Modelo . . . 4
2.2. Implementación . . . 5
2.3. Pruebas . . . 5
2.4. Árboles de búsqueda . . . 5
2.5. Análisis y conclusiones . . . 6

3. Secuencia Mágica **6**
3.1. Modelo . . . 6
3.2. Detalles de implementación . . . 6
3.3. Pruebas . . . 7
3.4. Árboles de búsqueda . . . 7
3.5. Análisis y conclusiones . . . 7

4. Acertijo Lógico **8**
4.1. Modelo . . . 8
4.2. Detalles de implementación . . . 9
4.3. Pruebas . . . 9
4.4. Árboles de búsqueda . . . 9
4.5. Análisis y conclusiones . . . 9

5. Ubicación de personas en una reunión **9**
5.1. Modelo . . . 9
5.2. Detalles de implementación . . . 10
5.3. Pruebas . . . 11
5.4. Árboles de búsqueda . . . 11
5.5. Análisis y conclusiones . . . 11

6. Construcción de un rectángulo **12**
6.1. Modelo . . . 12
6.2. Detalles de implementación . . . 13
6.3. Pruebas . . . 13
6.4. Árboles de búsqueda . . . 14
6.5. Análisis y conclusiones . . . 14

Repositorio del proyecto

Código fuente, instancias, scripts y PDF están disponibles en: `https://github.com/Herreran903/taller-1-restricciones`

1. Sudoku

Puzzle en una grilla 9×9 dividida en nueve cajas 3×3. Se entregan algunas celdas como *pistas* y el objetivo es completar las restantes con dígitos $1-9$ de modo que en cada fila, en cada columna y en cada caja 3×3 no se repita ningún dígito.

1.1. Modelo

Parámetros

P1 — N**:** Tamaño del tablero. En Sudoku clásico, $N = 9$.
P2 — S**:** Índices de filas/columnas: $S = \{1, \dots, N\}$.
P3 — DIG**:** Dígitos válidos: $DIG = \{1, \dots, N\}$.
P4 — G**:** Matriz de pistas $G \in \{0, \dots, N\}^{S \times S}$; $G_{r,c} = 0$ indica vacío y $G_{r,c} \in DIG$ fija la celda.

Variables

V1 — $X_{r,c}$**:** Valor de la celda (r, c): $X_{r,c} \in DIG$, para $r, c \in S$.

Restricciones principales

R1 — **Pistas fijas:** Toda celda con pista dada conserva su valor.

$$\forall (r, c) \in S : \; G_{r,c} > 0 \;\Rightarrow\; X_{r,c} = G_{r,c}.$$

R2 — **Filas sin repetición:** En cada fila, todos los valores son distintos.

$$\forall r \in S : \; \forall c_1, c_2 \in S, \; c_1 \neq c_2 \;\Rightarrow\; X_{r,c_1} \neq X_{r,c_2}.$$

R3 — **Columnas sin repetición:** En cada columna, todos los valores son distintos.

$$\forall c \in S : \; \forall r_1, r_2 \in S, \; r_1 \neq r_2 \;\Rightarrow\; X_{r_1,c} \neq X_{r_2,c}.$$

R4 — **Cajas** 3×3 **sin repetición:** En cada subcuadro 3×3, los valores son distintos entre sí.

$$\forall b_r, b_c \in \{0, 1, 2\} : \; \forall (i_1, j_1), (i_2, j_2) \in \{1, 2, 3\}^2, \; (i_1, j_1) \neq (i_2, j_2) \;\Rightarrow\; X_{3b_r+i_1,\, 3b_c+j_1} \neq X_{3b_r+i_2,\, 3b_c+j_2}.$$

Restricciones redundantes

R5 — **Suma por fila** $= 45$**:** En cada fila, la suma de los valores debe ser igual a 45.

$$\forall r \in S : \; \sum_{c \in S} X_{r,c} = 45.$$

R6 — **Suma por columna** $= 45$**:** En cada columna, la suma de los valores debe ser igual a 45.

$$\forall c \in S : \; \sum_{r \in S} X_{r,c} = 45.$$

R7 — **Suma por caja** $= 45$**:** En cada subcuadro 3×3, la suma de los valores también debe ser igual a 45.

$$\forall b_r, b_c \in \{0, 1, 2\} : \; \sum_{i=1}^{3} \sum_{j=1}^{3} X_{3b_r+i,\, 3b_c+j} = 45.$$

Justificación del modelo

La formulación reproduce con precisión las reglas del Sudoku y conserva corrección y completitud. Las restricciones R1–R4 cubren los principios esenciales: las pistas fijas (R1) respetan la instancia, las filas y columnas sin repetición (R2–R3) garantizan unicidad de dígitos en ambas direcciones, y las cajas 3×3 (R4) extienden la no repetición a las subcuadrículas. El dominio DIG acota los valores a $1-9$ y una única variable por celda simplifica la coherencia entre todas las vistas del tablero. Las redundancias R5–R7, basadas en la suma total de $1-9$, refuerzan la propagación local sin crear soluciones nuevas, por lo que pueden ayudar a detectar inconsistencias con menos exploración.

1.2. Implementación

Modelo

Definimos el conjunto de ramificación $\mathcal{B} = \{X_{r,c} \mid G_{r,c} = 0\}$ y sólo exploramos celdas sin pista. Así evitamos ramificar en valores ya fijados por G y concentramos la búsqueda donde hay incertidumbre.

Restricciones redundantes

Añadimos las sumas a 45 como poda ligera, no cambian el conjunto de soluciones y, en teoría, deberían ayudar a detectar inconsistencias temprano, reduciendo *nodos* y *fallos*.

Ruptura de simetría

Las pistas G fijan la instancia y aplicar simetrías del Sudoku (permutar filas, columnas o bandas, renombrar dígitos, transponer) movería o alteraría G. Para no arriesgar la solución válida, no añadimos rompedores de simetría.

1.3. Pruebas

Las instancias `test_01`, `test_02` y `test_03` siguen una dificultad *aprox.* creciente; no obstante, según solver/heurística `test_02` puede comportarse tan difícil como `test_03`, algo visible en *nodes/fail* y la profundidad del árbol.

Tabla 1: Resultados de pruebas **con** restricciones redundantes.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	2.000e−03	5	4	2
test_01	Chuffed	dom_w_deg	indomain_split	1.000e−03	7	5	3
test_01	Chuffed	input_order	indomain_min	1.000e−03	6	5	2
test_01	Gecode	first_fail	indomain_min	6.086e−03	89	44	7
test_01	Gecode	dom_w_deg	indomain_split	1.436e−03	51	25	8
test_01	Gecode	input_order	indomain_min	1.960e−03	131	65	7
test_02	Chuffed	first_fail	indomain_min	9.000e−03	472	426	13
test_02	Chuffed	dom_w_deg	indomain_split	1.100e−02	555	505	14
test_02	Chuffed	input_order	indomain_min	1.000e−02	574	549	11
test_02	Gecode	first_fail	indomain_min	3.580e−02	5993	2996	17
test_02	Gecode	dom_w_deg	indomain_split	1.446e−02	1449	724	21
test_02	Gecode	input_order	indomain_min	4.586e−02	7505	3752	20
test_03	Chuffed	first_fail	indomain_min	3.000e−03	137	129	7
test_03	Chuffed	dom_w_deg	indomain_split	7.000e−03	354	349	9
test_03	Chuffed	input_order	indomain_min	7.000e−03	370	365	7
test_03	Gecode	first_fail	indomain_min	8.904e−03	933	466	11
test_03	Gecode	dom_w_deg	indomain_split	5.791e−03	489	244	11
test_03	Gecode	input_order	indomain_min	1.396e−02	1653	826	15

Tabla 2: Resultados de pruebas **sin** restricciones redundantes.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	1.000e−03	5	4	2
test_01	Chuffed	dom_w_deg	indomain_split	1.000e−03	7	5	3
test_01	Chuffed	input_order	indomain_min	1.000e−03	6	5	2
test_01	Gecode	first_fail	indomain_min	5.385e−03	89	44	7
test_01	Gecode	dom_w_deg	indomain_split	1.209e−03	49	24	7
test_01	Gecode	input_order	indomain_min	1.651e−03	131	65	7
test_02	Chuffed	first_fail	indomain_min	8.000e−03	471	428	13
test_02	Chuffed	dom_w_deg	indomain_split	9.000e−03	503	468	14
test_02	Chuffed	input_order	indomain_min	1.000e−02	571	537	11
test_02	Gecode	first_fail	indomain_min	3.216e−02	5993	2996	17
test_02	Gecode	dom_w_deg	indomain_split	8.019e−03	1029	514	18
test_02	Gecode	input_order	indomain_min	4.086e−02	7505	3752	20
test_03	Chuffed	first_fail	indomain_min	3.000e−03	137	129	7
test_03	Chuffed	dom_w_deg	indomain_split	7.000e−03	355	349	9
test_03	Chuffed	input_order	indomain_min	7.000e−03	369	364	7
test_03	Gecode	first_fail	indomain_min	7.874e−03	933	466	11
test_03	Gecode	dom_w_deg	indomain_split	5.294e−03	541	270	14
test_03	Gecode	input_order	indomain_min	1.209e−02	1653	826	15

1.4. Árboles de búsqueda

Se capturaron con *Gecode Gist*.

Árboles de búsqueda (Google Drive).

1.5. Análisis y conclusiones

La comparación entre solvers mostró que, en general, Chuffed resolvió el Sudoku en menos tiempo que Gecode. Chuffed combina propagación fuerte con aprendizaje de conflictos, lo que recorta el árbol de búsqueda y acelera cada paso de inferencia, de modo que incluso cuando explora un número de nodos y fallos comparable termina antes por unidad de trabajo más eficaz. En Gecode, el rendimiento depende en mayor medida de la heurística elegida: con estrategias bien informadas puede reducir mucho el árbol y acercarse a los mejores tiempos, pero su velocidad suele ser más sensible a la elección de la búsqueda y, en promedio, queda por detrás de Chuffed. En nuestras pruebas se observa además que Chuffed mantiene un comportamiento más estable entre heurísticas, mientras que Gecode muestra variaciones marcadas según la combinación de selección de variables y política de asignación de valores.

En las pruebas realizadas, se observa en el rendimiento de las heurísticas que `dom_w_deg + indomain_split` destaca en *Gecode* porque prioriza variables con mayor historial de choques y, al dividir el dominio, ingresa temprano en las partes donde se concentra la dificultad. Este patrón se manifiesta en conteos más bajos de *nodes* y *fail* a lo largo de las tres instancias. En *Chuffed* sucede algo distinto. `first_fail + indomain_min` resulta la opción más pareja, ya que cuando muchos dominios quedan cortos tras la propagación, elegir la variable más restringida tiende a cerrar ramas con rapidez, mientras que el *split* añade trabajo sin un beneficio claro en la reducción del árbol. `input_order + indomain_min` queda rezagada en la mayor

parte del conjunto porque no aprovecha señal alguna del estado del problema y a menudo recorre regiones poco informativas del espacio de búsqueda. La única cercanía apreciable aparece en el caso trivial `test_01`, donde se observa un rendimiento próximo al de `dom_w_deg + indomain_split`. También se aprecia que `test_02` puede resultar tan exigente como `test_03` según la combinación elegida, lo que se refleja en la profundidad alcanzada y en los conteos de *nodes* y *fail*.

Finalmente, se observó que añadir las restricciones redundantes de suma no aportó mejoras y, en varios casos, introdujo un ligero sobrecoste. Aunque se entiende que las redundancias pueden ayudar, en nuestro modelo de Sudoku el propagador de *all_different* ya realiza una poda muy fuerte, de modo que las sumas apenas añaden información y sí más trabajo de propagación. En nuestras pruebas, las métricas con redundancias fueron en general similares o algo peores (ligeros aumentos de tiempo y nodos), especialmente con estrategias como `wdeg_split`. Con heurísticas simples tampoco se observó un beneficio claro. En conjunto, el modelo con sumas no redujo el backtracking ni el tiempo de resolución, por lo que se opto por dejarlas desactivadas por defecto.

2. Kakuro

Puzzle en grilla ortogonal con celdas negras que delimitan bloques horizontales y verticales. En cada bloque se colocan dígitos $1-9$ sin repetición cuya suma coincide con la pista, las celdas negras permanecen vacías.

2.1. Modelo

Parámetros

P1 — R**:** Filas del tablero, $R \in \mathbb{N}$.
P2 — C**:** Columnas del tablero, $C \in \mathbb{N}$.
P3 — *white***:** Matriz que indica qué casillas son blancas o negras:

$$\texttt{white} : \text{Cols} \times \text{Rows} \to \{0, 1\},$$

donde $\texttt{white}[c, r] = 1$ significa que la casilla en columna c y fila r es blanca, y $\texttt{white}[c, r] = 0$ indica una casilla negra. El número total de casillas blancas puede obtenerse directamente a partir de `white` mediante $\sum_{c \in \text{Cols}} \sum_{r \in \text{Rows}} \texttt{white}[c, r]$.
P4 — pistas horizontales (A): Número de pistas horizontales `NA`. Para cada $k \in \{1, \ldots, \texttt{NA}\}$ se tiene:

posición de inicio ($a\_col_k \in \{1, \ldots, \texttt{C}\}, a\_row_k \in \{1, \ldots, \texttt{R}\}$), longitud $a\_len_k \in \{1, \ldots, \texttt{R}\}$, suma objetivo $a\_sum_k \in \mathbb{N}$.

La pista horizontal k abarca las celdas

$$A_k = \{(a\_col_k,\ a\_row_k + k) \mid t = 0, \ldots, a\_len_k - 1\},$$

y se exige que $A_k \subseteq white$ (todas sus celdas son blancas).
P5 — pistas verticales (D): Número de pistas verticales `ND`. Para cada $k \in \{1, \ldots, \texttt{ND}\}$ se tiene:

posición de inicio ($d\_col_k \in \{1, \ldots, \texttt{C}\}, d\_row_k \in \{1, \ldots, \texttt{R}\}$), longitud $d\_len_k \in \{1, \ldots, \texttt{C}\}$, suma objetivo $d\_sum_k \in \mathbb{N}$.

La pista vertical k abarca las celdas

$$D_k = \{(d\_col_k + t,\ d\_row_k) \mid t = 0, \ldots, d\_len_k - 1\},$$

y se exige que $D_k \subseteq white$.

Variables

V1 — X_i**:** Valor de la celda blanca i: $X_i \in \{1, \ldots, \texttt{9}\}$ para todo $i \in white$.

Restricciones principales

R1 — Bloques horizontales válidos: En cada bloque horizontal, la suma de los valores debe coincidir con la pista y los dígitos no pueden repetirse.

$$\forall a \in A : \sum_{i \in C_a^A} X_i = a\_sum_a^A, \qquad \forall i_1, i_2 \in C_a^A,\ i_1 \neq i_2 \Rightarrow X_{i_1} \neq X_{i_2}.$$

R2 — Bloques verticales válidos: En cada bloque vertical, la suma de los valores debe coincidir con la pista y los dígitos deben ser distintos entre sí.

$$\forall d \in D : \sum_{i \in C_d^D} X_i = d\_sum_d^D, \qquad \forall i_1, i_2 \in C_d^D,\ i_1 \neq i_2 \Rightarrow X_{i_1} \neq X_{i_2}.$$

R3 — Intersección coherente: Cada celda de cada pista se encuentra dentro de las casillas blancas definidas por white.

$$\forall i \in A :\ X_i \in white \wedge \forall i \in D :\ X_i \in white$$

R4 — Fijar casillas negras: Para toda posición (c, r) tal que $\texttt{white}[c, r] = 0$ se fija la variable de esa casilla al valor reservado (aquí 1):

$$\forall c \in \text{Cols},\ \forall r \in \text{Rows} : \quad \texttt{white}[c, r] = 0 \ \Rightarrow\ x[c, r] = 1.$$

Restricciones redundantes

R4 — Acotación por suma: Si un bloque tiene k celdas y pista s, la suma de sus valores está acotada por las combinaciones posibles de k dígitos distintos.

$$s_{\text{mín}}(k) \leq \sum_{i=1}^{k} X_i \leq s_{\text{máx}}(k), \qquad s_{\text{mín}}(k) = 1 \times k, \quad s_{\text{máx}}(k) = 9 \times -k.$$

Justificación del modelo

La formulación refleja las reglas y mantiene corrección y completitud. R1 y R2 aplican, en el lugar exacto, la suma objetivo y la no repetición para cada bloque horizontal y vertical; R3 asegura coherencia en las celdas al mantener las variables en las celdas permitidas del tablero. La máscara de muros separa posiciones sin decisión de celdas válidas, evita asignaciones fuera del dominio $1-9$. Laa redundanciaa R4 busca fortalecer la poda: las cotas de suma descartan sumas inviables de manera temprana.

2.2. Implementación

Modelo

El modelo usa una máscara binaria *white* para distinguir muros y celdas blancas. Las celdas negras se fijan en uno y las blancas toman dígitos $1-9$. A partir de *white* y de las pistas se extraen los bloques horizontales y verticales como secuencias contiguas. Cada bloque exige suma objetivo y no repetición. La búsqueda solo ramifica sobre celdas blancas, lo que evita posiciones muertas y concentra el esfuerzo donde hay decisión.

Restricciones redundantes

Se añade poda ligera específica por bloque. Para un bloque de tamaño k y pista s se acotan dominios con sumas mínima y máxima posibles sin repetición $\left(s_{\text{mín}}(k) \leq \sum X \leq s_{\text{máx}}(k)\right)$ y se descartan dígitos incompatibles con dichas cotas.

Ruptura de simetría

En el presente modelo no se incorporaron restricciones específicas de ruptura de simetría porque las pistas y la topología del tablero anclan fuertemente las variables: cada pista tiene una posición y una suma fijadas ($a\_row, a\_col, a\_sum, d\_row, d\_col, d\_sum$), y las celdas negras definen una disposición irregular que impide permutaciones no triviales de filas/columnas. Además, las restricciones `all_different` junto con las sumas objetivo rompen en la práctica muchas simetrías de valor (una permutación global de los dígitos no preservaría las sumas fijadas), por lo que el conjunto de simetrías efectivas restantes es reducido. Por último, la detección y eliminación automática de esas pocas simetrías residuales requeriría restricciones adicionales específicas por instancia (y coste de propagación) cuyo beneficio no está garantizado; por tanto, se optó por no aplicarlas y priorizar heurísticas y redundantes verificables empíricamente.

2.3. Pruebas

Se evaluó el modelo con dos instancias, `test_01` y `test_02`.

Tabla 3: Resultados de pruebas **con** restricciones redundantes.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	6.000e−03	5	4	3
test_01	Chuffed	dom_w_deg	indomain_split	6.000e−03	6	3	3
test_01	Gecode	first_fail	indomain_min	5.623e−04	9	4	3
test_01	Gecode	dom_w_deg	indomain_split	5.250e−04	9	4	4
test_02	Chuffed	first_fail	indomain_min	6.000e−03	5	5	1
test_02	Chuffed	dom_w_deg	indomain_split	6.000e−03	5	5	2
test_02	Gecode	first_fail	indomain_min	1.276e−03	9	4	1
test_02	Gecode	dom_w_deg	indomain_split	1.967e−03	9	4	1

Tabla 4: Resultados de pruebas **sin** restricciones redundantes.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	5.000e−03	5	4	3
test_01	Chuffed	dom_w_deg	indomain_split	1.000e−03	6	3	3
test_01	Gecode	first_fail	indomain_min	9.591e−04	9	4	3
test_01	Gecode	dom_w_deg	indomain_split	8.089e−04	9	4	4
test_02	Chuffed	first_fail	indomain_min	6.000e−03	5	5	1
test_02	Chuffed	dom_w_deg	indomain_split	6.000e−03	5	5	2
test_02	Gecode	first_fail	indomain_min	5.317e−04	9	4	1
test_02	Gecode	dom_w_deg	indomain_split	5.488e−04	9	4	1

2.4. Árboles de búsqueda

Se capturaron con *Gecode Gist*.

Árboles de búsqueda (Google Drive).

2.5. Análisis y conclusiones

Las métricas recogidas en las Tablas 3–4 ponen de manifiesto que, en las instancias estudiadas, la activación de las restricciones redundantes no produce una mejora sistemática y significativa en las medidas habituales (nodes, fail y profundidad). Los valores de `nodes` y `fail` se mantienen iguales entre las ejecuciones con y sin redundantes y hay una pequeña reducción en el `time` para las instancias con redundantes, las pequeñas diferencias observadas varían según la combinación solver/heurística, lo que sugiere que están dentro de la variabilidad experimental más que reflejan un efecto claro de las redundantes. Del mismo modo, la elección de heurística (p. ej. `first_fail + indomain_min` frente a `dom_w_deg + indomain_split`) no muestra aquí un ganador absoluto: ambas combinaciones alcanzan contadores de nodos y fallos comparables en las instancias medidas, en cuanto a los solver, `Chuffed` demostró explorar menos `nodes` que `Gecode`, aunque tardó un poco más en finalizar, en el resto de estadísticas se mantienen en resultados muy similares.

En consecuencia, para este modelo concreto se recomienda mantener por defecto la versión sin restricciones redundantes y considerar su activación únicamente como prueba empírica adicional cuando una instancia concreta muestre beneficio neto. Además, dado que no hay una diferencia robusta entre solvers en estas tablas, la selección de solver y heurística debería guiarse por la métrica objetivo (minimizar tiempo frente a minimizar backtracking) y por pruebas previas sobre la clase de instancias de interés, en lugar de aplicar una regla general de activación de redundantes.

3. Secuencia Mágica

Consiste en encontrar una secuencia de longitud n donde cada posición i indica cuántas veces aparece el número i dentro de la misma secuencia. El objetivo del modelo es determinar todas las secuencias posibles que cumplan esta condición para un valor dado de n. El parámetro principal del modelo es n, que define la longitud de la secuencia. Cada elemento de la secuencia puede tomar valores entre 0 y n-1.

3.1. Modelo

Parámetros

P1 — n**:** Longitud de la secuencia mágica. Define el tamaño del arreglo x.

Variables

V1 — $x[i]$**:** Valor en la posición i, con dominio $x[i] \in \{0, 1, \ldots, n-1\}$ para todo $i = 0, 1, \ldots, n-1$.

Restricciones principales

R1 — **Definición de secuencia mágica:** Cada número i aparece exactamente $x[i]$ veces en la secuencia.

$$\forall i \in \{0, \ldots, n-1\} : \quad x[i] = \Big|\{\, j \in \{0, \ldots, n-1\} : x[j] = i \,\}\Big|.$$

Restricciones redundantes

R2 — **Suma total:** La suma de las frecuencias debe ser igual a la longitud total de la secuencia.

$$\sum_{i=0}^{n-1} x[i] = n.$$

R3 — **Equilibrio de valores:** Esta relación expresa el equilibrio entre los índices y las frecuencias, ayudando a reducir el espacio de búsqueda.

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

Justificación del modelo

En este modelo, la restricción $\forall i \in \{0, \ldots, n-1\} : \; x[i] = \Big|\{\, j : x[j] = i \,\}\Big|$ garantiza la *corrección*, pues fuerza a que cada componente $x[i]$ coincida exactamente con el número de apariciones del dígito i (punto fijo del operador "histograma"), y asegura la *completitud*, ya que cualquier secuencia mágica válida satisface por construcción esas igualdades. Los dominios $x[i] \in \{0, \ldots, n-1\}$ son mínimos y consistentes: un conteo no puede ser negativo, no puede superar n, y el valor n es imposible, por lo que $\{0, \ldots, n-1\}$ elimina valores inviables sin perder soluciones.

3.2. Detalles de implementación

Restricciones redundantes

Las restricciones redundantes no alteran el conjunto de soluciones válidas, pero **reducen el espacio de búsqueda** del solucionador al eliminar combinaciones imposibles antes de explorarlas. En el modelo de secuencias mágicas, las restricciones:

$$\sum_{i=0}^{n-1} x[i] = n \quad \text{y} \quad \sum_{i=0}^{n-1} (i-1)\, x[i] = 0$$

actúan como filtros globales.

- La primera asegura que la suma de todas las frecuencias sea igual a la longitud de la secuencia, descartando asignaciones inconsistentes.

- La segunda mantiene el equilibrio entre los índices y sus valores, eliminando ramas que no pueden conducir a una secuencia válida.

Estas condiciones adicionales **mejoran la propagación de restricciones** y acortan el tiempo total de búsqueda.

Ruptura de simetría

En el problema de las secuencias mágicas no existen simetrías relevantes entre las variables, ya que cada posición $x[i]$ representa un índice distinto y tiene un significado propio.

- Permutar las posiciones del arreglo alteraría la interpretación del valor $x[i]$ (que indica cuántas veces aparece el número i).
- Por ello, el modelo es **intrínsecamente asimétrico**, y no se requiere ninguna restricción adicional de rompimiento de simetrías.

3.3. Pruebas

Se usaron 3 pruebas, n = 6, 50, 100 (archivos `test_01`, `test_02` y `test_03`). Cada una se corrió con 2 solvers (*Gecode* y *Chuffed*) y 3 heurísticas de variable de decisión.

Tabla 5: Resultados de pruebas **con** restricciones redundantes (formato compatible).

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Gecode	first_fail	indomain_min	1.63E-03	7	4	3
test_01	Gecode	input_order	indomain_min	5.67E-04	11	6	1
test_01	Gecode	input_order	indomain_split	7.31E-04	7	4	2
test_01	Chuffed	first_fail	indomain_min	3.00E-03	4	4	3
test_01	Chuffed	input_order	indomain_min	2.00E-03	6	6	1
test_01	Chuffed	input_order	indomain_split	3.00E-03	4	4	1
test_02	Gecode	first_fail	indomain_min	1.76E-03	31	5	25
test_02	Gecode	input_order	indomain_min	2.32E-03	94	46	1
test_02	Gecode	input_order	indomain_split	1.49E-03	46	22	5
test_02	Chuffed	first_fail	indomain_min	7.70E-02	78	8	25
test_02	Chuffed	input_order	indomain_min	6.90E-02	48	46	1
test_02	Chuffed	input_order	indomain_split	6.40E-02	25	22	4
test_03	Gecode	first_fail	indomain_min	3.70E-03	56	5	50
test_03	Gecode	input_order	indomain_min	4.33E-03	194	96	1
test_03	Gecode	input_order	indomain_split	4.27E-03	96	47	6
test_03	Chuffed	first_fail	indomain_min	2.42E-01	166	8	50
test_03	Chuffed	input_order	indomain_min	2.37E-01	98	96	1
test_03	Chuffed	input_order	indomain_split	2.62E-01	53	47	5

Tabla 6: Resultados de pruebas **sin** restricciones redundantes (formato compatible).

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Gecode	first_fail	indomain_min	6.55E-04	29	15	4
test_01	Gecode	input_order	indomain_min	9.83E-03	23	12	4
test_01	Gecode	input_order	indomain_split	6.61E-04	19	10	4
test_01	Chuffed	first_fail	indomain_min	9.00E-03	15	15	3
test_01	Chuffed	input_order	indomain_min	1.00E-02	12	12	2
test_01	Chuffed	input_order	indomain_split	1.00E-02	11	10	3
test_02	Gecode	first_fail	indomain_min	1.90E-01	2733	1364	26
test_02	Gecode	input_order	indomain_min	1.10E-01	367	182	4
test_02	Gecode	input_order	indomain_split	3.00E-02	265	129	8
test_02	Chuffed	first_fail	indomain_min	2.70E-01	1412	1406	26
test_02	Chuffed	input_order	indomain_min	2.50E-01	185	182	2
test_02	Chuffed	input_order	indomain_split	1.50E-01	142	130	7
test_03	Gecode	first_fail	indomain_min	4.88	10533	5264	51
test_03	Gecode	input_order	indomain_min	2.80	767	382	4
test_03	Gecode	input_order	indomain_split	4.00E-01	568	280	9
test_03	Chuffed	first_fail	indomain_min	3.58	5362	5356	51
test_03	Chuffed	input_order	indomain_min	2.43	385	382	2
test_03	Chuffed	input_order	indomain_split	1.57	291	277	8

3.4. Árboles de búsqueda

Se capturaron con *Gecode Gist*.
Árboles de búsqueda (Google Drive).

3.5. Análisis y conclusiones

Con las **restricciones redundantes** activadas el rendimiento mejora de forma drástica: los *tiempos* pasan de segundos a *milisegundos*, y el tamaño del árbol (*nodes*/*fail*) cae entre uno y dos órdenes de magnitud, además de reducirse la *profundidad*

(por ejemplo, en `test_03` se evita llegar a profundidades ∼50). En este escenario, **Gecode** es consistentemente más rápido y estable que **Chuffed**. *Sin* redundantes, ambos solvers sufren: aumentan *nodes/fail* y el tiempo crece notablemente (p. ej., `test_02`/`test_03`), aunque Gecode mantiene ventaja relativa. Estos resultados confirman que las redundantes no cambian la solución, pero *podan* significativamente el espacio de búsqueda, mejorando la eficiencia global. Si se busca balance tiempo/nodos, la mejor opción es **Gecode** con `input_order + indomain_split`, reservando `first_fail + indomain_min` como alternativa competitiva.

4. Acertijo Lógico

Este modelo resuelve un acertijo lógico en el que, para cada persona de un conjunto dado, se desea asignar de forma consistente y sin ambigüedades su *apellido*, *edad* y *género musical favorito*, cumpliendo un conjunto de pistas. El enfoque usa valores enteros para representar categorías (p. ej., `GONZALEZ`=1, `GARCIA`=2, `LOPEZ`=3) y "punteros" (variables índice) para referirse a la posición de la persona que cumple un atributo.

4.1. Modelo

Parámetros

P1 — `NOMBRE`**:** Conjunto de personas: {`Juan`, `Oscar`, `Dario`}.
P2 — `AGE`**:** Dominios de edad: $\{24, 25, 26\}$.
P3 — **Códigos de apellidos:** $\{\texttt{GONZALEZ} = 1,\ \texttt{GARCIA} = 2,\ \texttt{LOPEZ} = 3\}$.
P4 — **Códigos de música:** $\{\texttt{CLASICA} = 1,\ \texttt{POP} = 2,\ \texttt{JAZZ} = 3\}$.

Variables

V1 — `apellido`$[n]$**:** Código de apellido de la persona n, con dominio $\{1, 2, 3\}$.
V2 — `musica`$[n]$**:** Género musical de la persona n, con dominio $\{1, 2, 3\}$.
V3 — `edad`$[n]$**:** Edad de la persona n, con dominio $\{24, 25, 26\}$.

Restricciones principales

R1 — **Biyectividad por atributo:** No hay repeticiones dentro de cada atributo:

$$\texttt{alldifferent}\big(\{\texttt{apellido}[n]\}\big),\quad \texttt{alldifferent}\big(\{\texttt{musica}[n]\}\big),\quad \texttt{alldifferent}\big(\{\texttt{edad}[n]\}\big).$$

R2 — **Pistas del acertijo:** Se expresan como implicaciones índice–valor:

$$(\texttt{apellido}[n] = \texttt{GONZALEZ}) \Rightarrow \texttt{edad}[\texttt{Juan}] > \texttt{edad}[n],\ \forall n$$

$$(\texttt{apellido}[n] = \texttt{GONZALEZ}) \Rightarrow \texttt{musica}[n] = \texttt{CLASICA},\ \forall n$$

$$(\texttt{musica}[n] = \texttt{POP}) \Rightarrow \texttt{apellido}[n] \neq \texttt{GARCIA},\ \forall n$$

$$(\texttt{musica}[n] = \texttt{POP}) \Rightarrow \texttt{edad}[n] \neq 24,\ \forall n$$

$$\texttt{apellido}[\texttt{Oscar}] \neq \texttt{LOPEZ},\quad \texttt{edad}[\texttt{Oscar}] = 25,\quad \texttt{musica}[\texttt{Dario}] \neq \texttt{JAZZ}.$$

Justificación del modelo

El modelo es **correcto y completo** respecto al acertijo. Es correcto porque: (i) los dominios codifican exactamente los valores admitidos (apellidos, música y edades), (ii) `alldifferent` garantiza la *biyectividad* persona–valor en cada atributo, como exige el enunciado, y (iii) cada pista se traduce mediante implicaciones índice–valor semánticamente equivalentes (p. ej., `GONZALEZ` ⇒ `CLASICA`, `POP` ⇒ `apellido` ≠ `GARCIA`, `edad`[`Oscar`] = 25), de modo que toda solución del CSP satisface el acertijo. Es completo (*complete*) porque cualquier solución válida del acertijo puede representarse en el modelo: basta mapear las etiquetas a sus códigos enteros y se satisfacen dominios, `alldifferent` y las implicaciones; por lo tanto, ninguna solución real queda fuera. La codificación entera es puramente *representacional* (no añade ni elimina soluciones) y el mapeo inverso asegura salida legible. En conjunto, la combinación de dominios precisos, `alldifferent` y las pistas formalizadas elimina asignaciones espurias y, cuando el enunciado determina una única solución, el modelo también la vuelve única.

Estrategia de búsqueda

Se concatena el vector de decisión `vars` y se usa una heurística reproducible y fuertemente propagante:

`int_search`(`vars`, `dom_w_deg`, `indomain_split`, `complete`).

Justificación de la estrategia

- **Selección de variable (*dom_w_deg*).** Prioriza variables con *dominio pequeño* y *alto grado de conflicto* (ponderado por restricciones que ya han fallado). En este acertijo, las variables de `apellido`, `musica` y `edad` están conectadas por `alldifferent` e implicaciones reificadas; atacar primero las más "tensas" maximiza la poda temprana.
- **Selección de valor (*indomain_split*).** Partir el dominio activa más propagación que probar un solo valor (*min*) porque fuerza una dicotomía global: la mitad inferior frente a la superior. Con dominios pequeños ($\{1, 2, 3\}$ o $\{24, 25, 26\}$), cada corte dispara propagación en `alldifferent` y en las implicaciones (p. ej., `GONZALEZ` ⇒ `CLASICA`).
- **Menos retrocesos, menor profundidad**. La combinación `dom_w_deg` + `split` reduce *nodes/fail* y la *peakDepth* al resolver primero los cuellos de botella y descartar ramas imposibles antes de comprometer valores concretos.

4.2. Detalles de implementación

Restricciones redundantes

Aunque `alldifferent` ya garantiza la unicidad de los valores, se consideraron algunas restricciones redundantes que podrían, en principio, mejorar la propagación.

$$\texttt{count}(\{\texttt{apellido}[n]\}, \texttt{GONZALEZ}) = 1, \qquad \texttt{count}(\{\texttt{musica}[n]\}, \texttt{POP}) = 1,$$

y una suma fija de edades:

$$\sum_n \texttt{edad}[n] = 75 \quad (24 + 25 + 26 = 75).$$

Sin embargo, en las pruebas internas no se observaron mejoras apreciables en tiempo de resolución ni en la cantidad de nodos o fallos. Por ello, en la versión final no se añadieron restricciones redundantes, priorizando la simplicidad y claridad del modelo.

Ruptura de simetría

No hay simetrías relevantes: los valores están etiquetados (GONZALEZ/POP/JAZZ, edades específicas) y las personas (Juan/Oscar/Dario) aparecen en pistas distintas. No necesitas romper simetrías adicionales.

4.3. Pruebas

Para las pruebas se usaron 2 solvers, GeoCode y Chuffed, y 4 heurísticas de variable de decisión (first_fail, input_order, input_order + indomain_split, y dom_wdeg + indomain_split). Una sola prueba debido a que el problema no tiene parametros que varien.

Tabla 7: Resultados de pruebas **sin** restricciones redundantes (formato compatible).

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	4.00E-03	3	1	1
test_01	Chuffed	input_order	indomain_min	2.00E-03	3	1	1
test_01	Chuffed	input_order	indomain_split	2.00E-03	3	1	1
test_01	Chuffed	wdeg_split	indomain_split	3.00E-03	3	1	1
test_01	Gecode	first_fail	indomain_min	2.08E-03	5	2	1
test_01	Gecode	input_order	indomain_min	8.41E-04	5	2	1
test_01	Gecode	input_order	indomain_split	6.51E-04	5	2	1
test_01	Gecode	wdeg_split	indomain_split	7.29E-04	5	2	1

4.4. Árboles de búsqueda

Se capturaron con *Gecode Gist*.
Árboles de búsqueda (Google Drive).

4.5. Análisis y conclusiones

Dado que se trata de un problema **pequeño**, de **única solución** y con **pocas variables**, las diferencias de rendimiento entre solvers y heurísticas son reducidas. Aun así, se observa que **Gecode** es sistemáticamente más rápido que **Chuffed** en todos los casos, si bien la brecha es *mínima* (del orden de los ms).

En cuanto a la heurística, la configuración seleccionada previamente (*p. ej.*, `dom_w_deg + indomain_split`) produce árboles **menos profundos**, lo que la vuelve la opción **más eficiente**.

Finalmente, debido al **tamaño** del problema y a que la estructura ya queda fuertemente determinada por las pistas y la biyectividad, las **restricciones redundantes** no aportan mejoras apreciables en la *poda* del árbol ni en el tiempo total; su impacto es marginal en esta instancia.

5. Ubicación de personas en una reunión

Un grupo de N personas desea tomarse una fotografía en una sola fila. Algunas parejas de personas imponen preferencias de proximidad: *adyacencia*, *no adyacencia* y *cota máxima de distancia*, que limitan cuántas personas pueden quedar entre dos individuos.

5.1. Modelo

Parámetros

P1 — N**:** Número de personas a ubicar. $N \in \mathbb{Z}_{\geq 1}$.

P2 — S**:** Índices válidos para personas. $S = \{1, \ldots, N\}$.

P3 — POS**:** Conjunto de posiciones disponibles en la fila. $POS = \{1, \ldots, N\}$.

P4 — `personas`**:** Vector de nombres. $\texttt{personas} \in \text{String}^S$.

P5 — $K_{\texttt{next}}, K_{\texttt{sep}}, K_{\texttt{dist}}$**:** Cantidad de preferencias de cada tipo. $K_{\texttt{next}},\ K_{\texttt{sep}},\ K_{\texttt{dist}} \in \mathbb{Z}_{\geq 0}$.

P6 — `NEXT`, `SEP`, `DIST`**:** Matrices de preferencias: $\texttt{NEXT} \in S^{K_{\texttt{next}} \times 2}$, $\texttt{SEP} \in S^{K_{\texttt{sep}} \times 2}$, $\texttt{DIST} \in \big(S \times S \times \{0, \ldots, N-2\}\big)^{K_{\texttt{dist}}}$. Cada fila codifica un par de personas y, en `DIST`, una cota M de separación.

Variables

V1 — POS_OF_p**:** Posición que ocupa la persona p. $POS\_OF_p \in POS,\ p \in S$.
V2 — PER_AT_i**:** Persona ubicada en la posición i. $PER\_AT_i \in S,\ i \in POS$.

Restricciones principales

R1 — Biección: La asignación entre personas y posiciones forma una correspondencia uno a uno.

$$\forall p \in S:\ \exists!\, x \in S:\ POS\_OF(p) = x, \qquad \forall x \in S:\ \exists!\, p \in S:\ PER\_AT(x) = p.$$

Además, ambas funciones son inversas entre sí:

$$PER\_AT(POS\_OF(p)) = p, \qquad POS\_OF(PER\_AT(x)) = x.$$

R2 — Preferencias next(A, B): Las personas A y B deben quedar adyacentes.

$$\forall (A, B) \in \texttt{NEXT}:\ |\, POS\_OF(A) - POS\_OF(B)\,| = 1.$$

R3 — Preferencias separate(A, B): Las personas A y B no pueden quedar una junto a la otra.

$$\forall (A, B) \in \texttt{SEP}:\ |\, POS\_OF(A) - POS\_OF(B)\,| \geq 2.$$

R4 — Preferencias distance(A, B, M): Entre A y B puede haber a lo sumo M personas, lo que equivale a una cota superior sobre la distancia de posiciones.

$$\forall (A, B, M) \in \texttt{DIST}:\ |\, POS\_OF(A) - POS\_OF(B)\,| \leq M + 1.$$

Restricciones redundantes

R5 — Límite de apariciones en next: Cada persona puede participar en un máximo de dos relaciones de adyacencia, pues solo puede tener un vecino a cada lado.

$$\forall p \in S:\ \sum_{(A,B)\in \texttt{NEXT}} [p = A \vee p = B] \leq 2.$$

R6 — Consistencia entre next y separate: No puede existir un mismo par de personas simultáneamente en ambas preferencias, ya que se produciría una contradicción.

$$\forall (A, B) \in \texttt{NEXT},\ (C, D) \in \texttt{SEP}:\ \neg\big[(A, B) = (C, D)\ \vee\ (A, B) = (D, C)\big].$$

Restricciones de simetrías

R7 — Rompimiento de simetría izquierda–derecha: Las soluciones reflejadas horizontalmente son equivalentes; para evitar duplicados, se impone un orden sobre las posiciones extremas.

$$PER\_AT(1) < PER\_AT(N).$$

Justificación del modelo

La formulación captura el problema de ubicar N personas en una fila. La biección de R1 garantiza que la asignación sea una permutación válida y mantiene coherencia entre las dos vistas del mismo estado (POS_OF y PER_AT). Las preferencias se modelan de forma directa: R2 impone adyacencia, R3 excluye adyacencia y R4 limita la distancia permitiendo a lo sumo M personas entre A y B. Las redundancias R5–R6 buscan fortalecer la propagación sin alterar soluciones: R5 se basa en el hecho estructural de que cada persona solo puede tener dos vecinos, y R6 elimina inconsistencias lógicas entre next y separate. Finalmente, R7 elimina duplicados por simetría espejo izquierda–derecha, preservando una solución representativa por clase de equivalencia.

5.2. Detalles de implementación

Modelo

Se usan dos vistas de la permutación: `POS_OF` (persona->posición) y `PER_AT` (posición->persona), enlazadas con *inverse*. Esto refuerza la propagación respecto a usar solo una vista con *all_different*, simplifica la salida (recorriendo `PER_AT` en orden) y facilita la ruptura de simetría comparando extremos.

Restricciones redundantes

El modelo base ya ofrece una propagación fuerte gracias a *inverse* y las restricciones principales, por lo que fue difícil encontrar redundancias que aportaran poda real. Se probaron alternativas como imponer *all_different* o forzar la suma de posiciones igual a $N(N+1)/2$, pero no mejoraron el rendimiento. Finalmente, solo se añadieron dos restricciones simples para verificar coherencia de datos: limitar a dos las apariciones de una persona en next y evitar pares repetidos entre next y separate. Estas no afectan la búsqueda, pero permiten detectar errores de entrada antes de ejecutar el modelo.

Ruptura de simetría

Existe simetría de reflexión izquierda–derecha: invertir la fila produce otra solución equivalente. Para evitar duplicados se fija un orden comparando los extremos (`PER_AT[1]` frente a `PER_AT[N]`). Esto reduce la búsqueda sin afectar satisfacibilidad ni óptimos.

5.3. Pruebas

Se usaron cuatro instancias: `test_01` es *UNSAT* por inviabilidad estructural; `test_02` muestra el efecto del rompimiento de simetría; `test_03` es factible y más exigente por restricciones solapadas; y `test_04` valida las redundancias con un caso pequeño e insatisfactible por conflicto entre `next` y `separate`.

Tabla 8: Resultados de pruebas **con** restricciones de simetría.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	chuffed	dom_w_deg	indomain_split	2.200e-02	1055	273	13
test_02	chuffed	dom_w_deg	indomain_split	3.000e-03	83	77	8
test_03	chuffed	dom_w_deg	indomain_split	8.000e-03	531	426	9
test_01	chuffed	first_fail	indomain_min	1.800e-02	180	180	3
test_02	chuffed	first_fail	indomain_min	3.000e-03	93	84	3
test_03	chuffed	first_fail	indomain_min	3.000e-03	147	145	6
test_01	gecode	dom_w_deg	indomain_split	1.634e-03	355	178	8
test_02	gecode	dom_w_deg	indomain_split	4.356e-03	1019	506	12
test_03	gecode	dom_w_deg	indomain_split	1.332e-03	237	98	11
test_01	gecode	first_fail	indomain_min	7.061e-02	30981	15491	6
test_02	gecode	first_fail	indomain_min	1.864e-03	1019	506	7
test_03	gecode	first_fail	indomain_min	1.596e-03	395	177	7

Tabla 9: Resultados de pruebas **sin** restricciones de simetría.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	chuffed	dom_w_deg	indomain_split	2.400e-02	1055	273	13
test_02	chuffed	dom_w_deg	indomain_split	3.000e-03	112	105	8
test_03	chuffed	dom_w_deg	indomain_split	8.000e-03	645	526	9
test_01	chuffed	first_fail	indomain_min	1.800e-02	180	180	3
test_02	chuffed	first_fail	indomain_min	3.000e-03	157	157	3
test_03	chuffed	first_fail	indomain_min	4.000e-03	189	188	7
test_01	gecode	dom_w_deg	indomain_split	1.438e-03	355	178	8
test_02	gecode	dom_w_deg	indomain_split	2.777e-03	1103	544	12
test_03	gecode	dom_w_deg	indomain_split	1.774e-03	263	90	11
test_01	gecode	first_fail	indomain_min	7.473e-02	31331	15666	6
test_02	gecode	first_fail	indomain_min	3.032e-03	1103	544	7
test_03	gecode	first_fail	indomain_min	3.170e-03	429	173	8

Tabla 10: Resultados de pruebas **con** y **sin** restricciones redundantes.

Archivo	Solver	Estrategia	time	nodes	fail	depth	Modo
test_01	chuffed	wdeg_split	2.100e-02	1055	273	13	sin red.
test_04	chuffed	wdeg_split	0.000e+00	11	7	2	sin red.
test_01	chuffed	ff_min	1.800e-02	180	180	3	sin red.
test_04	chuffed	ff_min	0.000e+00	7	7	1	sin red.
test_01	gecode	wdeg_split	2.057e-03	355	178	8	sin red.
test_04	gecode	wdeg_split	1.480e-03	11	6	3	sin red.
test_01	gecode	ff_min	6.857e-02	30981	15491	6	sin red.
test_04	gecode	ff_min	2.768e-04	13	7	1	sin red.
test_01	chuffed	wdeg_split	0.000e+00	0	1	0	con red.
test_04	chuffed	wdeg_split	0.000e+00	0	1	0	con red.
test_01	chuffed	ff_min	0.000e+00	0	1	0	con red.
test_04	chuffed	ff_min	0.000e+00	0	1	0	con red.
test_01	gecode	wdeg_split	2.897e-03	0	1	0	con red.
test_04	gecode	wdeg_split	9.996e-05	0	1	0	con red.
test_01	gecode	ff_min	9.808e-05	0	1	0	con red.
test_04	gecode	ff_min	1.051e-04	0	1	0	con red.

5.4. Árboles de búsqueda

Se capturaron con *Gecode Gist*.

Árboles de búsqueda (Google Drive).

5.5. Análisis y conclusiones

La comparación entre solvers mostró diferencias consistentes frente al problema de ubicación en una reunión. `Chuffed`, gracias a su aprendizaje de conflictos, mantuvo un equilibrio eficiente entre propagación y exploración, recorriendo menos nodos

y controlando mejor el espacio de búsqueda. Aunque no siempre alcanzó el menor tiempo absoluto, su relación entre nodos y fallos fue la más estable. `Gecode`, sin mecanismos de aprendizaje, depende más de la heurística elegida: con `dom_w_deg` obtuvo un rendimiento competitivo, pero en general requirió más nodos para concluir la factibilidad. Estas diferencias se acentúan en instancias más exigentes.

En cuanto a las estrategias de búsqueda, el desempeño depende del solver. En Gecode, `dom_w_deg + indomain_split` suele dar los menores *nodes/fail*, mientras que en Chuffed la opción más consistente es `first_fail + indomain_min`. Esto encaja con la forma en que cada motor explota la información: el conteo de conflictos de `dom/wdeg` guía bien la selección de variables cuando la propagación no concentra de inmediato las fallas, algo más afín a Gecode; en Chuffed, el aprendizaje de conflictos y una propagación más agresiva ya focalizan los dominios relevantes, de modo que `first_fail` acierta antes y el *split* tiende a añadir sobrecosto sin más poda.

El rompimiento de simetría redujo la exploración en `test_02` y `test_03`. Se observaron caídas en *nodes/fail* para ambos solvers en la mayoría de combinaciones. En Chuffed con `first_fail` sobre `test_02`, los conteos pasaron de 157 y 157 sin simetría a 93 y 84 con simetría. En Chuffed con `wdeg_split` sobre `test_3`, la exploración bajó de 645 y 526 sin simetría a 531 y 426 con simetría. La magnitud de la mejora varía según la pareja solver–heurística, pero la tendencia general es a árboles más compactos y búsqueda más dirigida cuando se activa la ruptura de simetría. El efecto se aprecia especialmente en que el número de soluciones se reduce a la mitad al eliminar configuraciones espejo, como se puede observar en los árboles de `Gecode Gist`.

Respecto a las redundancias, se incorporaron únicamente aquellas orientadas a verificar la coherencia lógica de la instancia. Estas actúan como "sanity checks" que permiten detectar contradicciones de entrada de forma inmediata —como en `test_04`—, sin alterar la propagación ni el comportamiento de búsqueda. Otras redundancias exploradas, como restricciones sobre sumatorias o relaciones *all_different*, no aportaron mejoras medibles en tiempo ni poda, ya que el modelo base, reforzado por el *inverse*, ya era suficientemente fuerte.

6. Construcción de un rectángulo

Se busca ubicar n cuadrados de lados $s[i]$ dentro de un rectángulo $W \times H$ sin solapamientos. El modelo decide las coordenadas $(x[i], y[i])$ (esquina superior izquierda) de cada cuadrado, garantizando que queden dentro del contenedor y que no se intersecten. Los parámetros son n, el vector s, y las dimensiones W, H; las variables son $x[i], y[i]$. Se incluye rompimiento de simetría para cuadrados iguales (orden lexicográfico) y una heurística informada por conflictos para acelerar la búsqueda.

6.1. Modelo

Parámetros

P1 — n**:** Número de cuadrados a ubicar.
P2 — $s[i]$**:** Lado del cuadrado i (vector de tamaños).
P3 — W**:** Ancho del rectángulo contenedor.
P4 — H**:** Alto del rectángulo contenedor.

Variables

V1 — $x[i]$**:** Coordenada x de la esquina superior izquierda del cuadrado i, con dominio $x[i] \in \{0, \ldots, W\}$.
V2 — $y[i]$**:** Coordenada y de la esquina superior izquierda del cuadrado i, con dominio $y[i] \in \{0, \ldots, H\}$.

Restricciones principales

R1 — Dentro del contenedor: Cada cuadrado debe quedar completamente dentro de $W \times H$:

$$\forall i \in \{1, \ldots, n\}: \quad x[i] + s[i] \le W, \quad y[i] + s[i] \le H.$$

MiniZinc:

```
constraint forall(i in 1..n)(
  x[i] + s[i] <= W /\ y[i] + s[i] <= H
);
```

R2 — No solapamiento: Los cuadrados no pueden intersectarse:

$$\forall i, j \in \{1, \ldots, n\},\ i \neq j: \quad (x_i + s_i \le x_j) \ \vee \ (x_j + s_j \le x_i) \ \vee \ (y_i + s_i \le y_j) \ \vee \ (y_j + s_j \le y_i)$$

MiniZinc (uso de la global ***diffn****):*

```
constraint diffn(x, y, s, s);
```

Restricciones redundantes (opcionales)

R3 — Filtro de área:

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

Justificación: si el área total de los cuadrados excede el área del contenedor, no existe solución; actúa como poda temprana.

```
% constraint sum(i in 1..n)(s[i]*s[i]) <= W*H;
```

R4 — Rompimiento de simetría (tamaños iguales): Para cuadrados con igual lado, imponer orden lexicográfico en posiciones para evitar permutaciones equivalentes:

$$\forall i < j: \ s[i] = s[j] \ \Rightarrow \ \langle x[i], y[i] \rangle \ \le_{\text{lex}} \ \langle x[j], y[j] \rangle.$$

MiniZinc:

```
constraint forall(i, j in 1..n where i<j /\ s[i]=s[j])(
  lex_lesseq([x[i], y[i]], [x[j], y[j]])
);
```

Justificación del modelo

El modelo es **correcto** porque sus restricciones capturan exactamente la factibilidad geométrica del empaquetado: (i) las desigualdades $x[i] + s[i] \leq W$ y $y[i] + s[i] \leq H$ garantizan que cada cuadrado quede *completamente contenido* en el rectángulo $W \times H$; (ii) la global `diffn(x,y,s,s)` impide *solapamientos* entre pares de cuadrados al imponer separaciones en x o en y; y (iii) los dominios $x[i] \in \{0, \ldots, W\}$, $y[i] \in \{0, \ldots, H\}$ son consistentes con esas cotas, dejando al propagador recortar valores imposibles cuando se activa $x[i] + s[i] \leq W$ y $y[i] + s[i] \leq H$. El modelo es **completo** porque cualquier configuración válida de los cuadrados dentro del rectángulo satisface las restricciones: si un conjunto de posiciones $(x[i], y[i])$ cumple las cotas y no hay solapamientos, entonces se cumple el CSP. La global `diffn` es *representacional* (no añade ni elimina soluciones) y el mapeo inverso asegura salida legible. Cabe aclarar que el modelo aunque incluye restricciones de ruptura de simetría que excluyen resultados de permutaciones de cuadrados del mismo tamaño,no incluye restricciones que excluyan soluciones espejo, por lo que configuraciones equivalentes bajo transformaciones de espejo o rotación (en caso de que $W = H$) se consideran soluciones distintas. En conjunto, la combinación de dominios precisos, las cotas y `diffn` elimina asignaciones espurias y asegura que todas las configuraciones geométricamente válidas se representen, aun cuando existan soluciones espejo equivalentes.

6.2. Detalles de implementación

Restricciones redundantes

Las restricciones redundantes no cambian el conjunto de soluciones, pero **reducen el espacio de búsqueda** al descartar configuraciones inviables antes de explorar ramas profundas. En el modelo de empaquetado de cuadrados dentro de un rectángulo, son útiles:

$$\sum_{i=1}^{n} s[i]^2 \;\leq\; W \cdot H \quad \text{(filtro de área)}$$

- **Filtro de área**: si el área total de los cuadrados supera el área del contenedor, no hay solución; imponerlo evita búsquedas inútiles.

Esta condición **evita la búsqueda** en casos donde desde el inicio no existe una solución.

Simetrías

En este problema sí existen simetrías relevantes:

- **Indistinguibilidad de cuadrados iguales**: si $s[i] = s[j]$, permutar sus coordenadas genera soluciones equivalentes. Se rompe esta simetría imponiendo orden lexicográfico:

$$s[i] = s[j],\ i < j \;\Rightarrow\; \langle x[i], y[i] \rangle \leq_{\text{lex}} \langle x[j], y[j] \rangle.$$

Con estas medidas, el modelo se vuelve **más asimétrico** y el solver evita explorar permutaciones equivalentes, mejorando la eficiencia sin excluir soluciones válidas.

6.3. Pruebas

Se evaluó el modelo con tres instancias, `test_01`, `test_02` y `test_03`.

Tabla 11: Resultados de pruebas **sin** redundancia y **sin** simetrías.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	7.000e−03	144	96	7
test_01	Chuffed	dom_w_deg	indomain_split	8.000e−03	91	74	8
test_01	Gecode	first_fail	indomain_min	9.583e−04	335	120	11
test_01	Gecode	dom_w_deg	indomain_split	9.648e−04	255	80	9
test_02	Chuffed	first_fail	indomain_min	7.000e−03	55	47	5
test_02	Chuffed	dom_w_deg	indomain_split	7.000e−03	65	45	10
test_02	Gecode	first_fail	indomain_min	5.404e−04	147	58	8
test_02	Gecode	dom_w_deg	indomain_split	5.244e−04	115	42	8
test_03	Chuffed	first_fail	indomain_min	8.000e−03	99	65	6
test_03	Chuffed	dom_w_deg	indomain_split	8.000e−03	91	51	12
test_03	Gecode	first_fail	indomain_min	5.025e−04	227	114	11
test_03	Gecode	dom_w_deg	indomain_split	4.990e−04	147	74	9

Tabla 12: Resultados de pruebas **sin** redundancia y **con** simetrías.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	8.000e−03	24	23	6
test_01	Chuffed	dom_w_deg	indomain_split	8.000e−03	27	19	8
test_01	Gecode	first_fail	indomain_min	4.920e−04	85	39	10
test_01	Gecode	dom_w_deg	indomain_split	4.935e−04	83	38	8
test_02	Chuffed	first_fail	indomain_min	7.000e−03	24	24	4
test_02	Chuffed	dom_w_deg	indomain_split	8.000e−03	40	27	10
test_02	Gecode	first_fail	indomain_min	4.101e−04	55	24	8
test_02	Gecode	dom_w_deg	indomain_split	4.325e−04	49	21	6
test_03	Chuffed	first_fail	indomain_min	7.000e−03	48	34	5
test_03	Chuffed	dom_w_deg	indomain_split	8.000e−03	60	33	12
test_03	Gecode	first_fail	indomain_min	4.365e−04	75	38	9
test_03	Gecode	dom_w_deg	indomain_split	4.118e−04	57	29	7

Tabla 13: Resultados de pruebas **con** redundancia y **sin** simetrías.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	9.000e−03	144	96	7
test_01	Chuffed	dom_w_deg	indomain_split	8.000e−03	91	74	8
test_01	Gecode	first_fail	indomain_min	9.488e−04	335	120	11
test_01	Gecode	dom_w_deg	indomain_split	9.280e−04	255	80	9
test_02	Chuffed	first_fail	indomain_min	7.000e−03	55	47	5
test_02	Chuffed	dom_w_deg	indomain_split	7.000e−03	65	45	10
test_02	Gecode	first_fail	indomain_min	5.443e−04	147	58	8
test_02	Gecode	dom_w_deg	indomain_split	5.182e−04	115	42	8
test_03	Chuffed	first_fail	indomain_min	8.000e−03	99	65	6
test_03	Chuffed	dom_w_deg	indomain_split	7.000e−03	91	51	12
test_03	Gecode	first_fail	indomain_min	4.994e−04	227	114	11
test_03	Gecode	dom_w_deg	indomain_split	4.630e−04	147	74	9

Tabla 14: Resultados de pruebas **con** redundancia y **con** simetrías.

Archivo	Solver	Var heur	Val heur	time	nodes	fail	depth
test_01	Chuffed	first_fail	indomain_min	8.000e−03	24	23	6
test_01	Chuffed	dom_w_deg	indomain_split	9.000e−03	27	19	8
test_01	Gecode	first_fail	indomain_min	4.884e−04	85	39	10
test_01	Gecode	dom_w_deg	indomain_split	4.906e−04	83	38	8
test_02	Chuffed	first_fail	indomain_min	8.000e−03	24	24	4
test_02	Chuffed	dom_w_deg	indomain_split	8.000e−03	40	27	10
test_02	Gecode	first_fail	indomain_min	4.073e−04	55	24	8
test_02	Gecode	dom_w_deg	indomain_split	4.163e−04	49	21	6
test_03	Chuffed	first_fail	indomain_min	—	—	—	—
test_03	Chuffed	dom_w_deg	indomain_split	—	—	—	—
test_03	Gecode	first_fail	indomain_min	1.543e−04	0	1	0
test_03	Gecode	dom_w_deg	indomain_split	5.599e−03	0	1	0

6.4. Árboles de búsqueda

Se capturaron con *Gecode Gist*.

Árboles de búsqueda (Google Drive).

6.5. Análisis y conclusiones

Las métricas recogidas en las Tablas 11–14 muestran una tendencia clara: la eliminación de simetrías reduce de forma sistémica el tamaño del árbol de búsqueda (nodes y fail) y, en la mayoría de combinaciones, también el tiempo de resolución. Este efecto es especialmente notable en instancias más exigentes, donde la poda del espacio de búsqueda evita exploraciones redundantes y disminuye la profundidad alcanzada. En cuanto a los solvers, Chuffed presenta un comportamiento más estable entre heurísticas y consigue tiempos competitivos cuando se usa la combinación `first_fail + indomain_min`, mientras que Gecode suele beneficiarse más de heurísticas informadas como `dom_w_deg + indomain_split`, que reducen marcadamente nodes y fail al priorizar variables con historial de conflictos.

La inclusión de restricciones redundantes no muestra un impacto en la cantidad de nodos explorados o profundidad del arbol, aunque se observa una pequeña mejora en el tiempo en la mayoría de los casos a excepción de cuando no se puede obtener una solución desde el inicio donde se evita la búsqueda. Por tanto, la recomendación práctica es activar el rompimiento de simetrías por defecto, dado su impacto consistente y positivo; mantener las restricciones redundantes como opción a evaluar caso por caso (activarlas cuando las pruebas muestren reducción neta de nodes/t iempo) y elegir el solver/heurística según la métrica

objetivo: Chuffed con `first_fail + indomain_min` para tiempos estables y Gecode con `dom_w_deg + indomain_split` cuando se busque minimizar el retroceso y el número de nodos.