

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

14 de octubre de 2025

## Índice

<b>1. Sudoku</b>	<b>2</b>
1.1. Modelo . . . . .	2
1.2. Implementación . . . . .	3
1.3. Pruebas . . . . .	3
1.4. Árboles de búsqueda . . . . .	4
1.5. Análisis y conclusiones . . . . .	5
<b>2. Kakuro</b>	<b>6</b>
2.1. Modelo . . . . .	6
2.2. Detalles de implementación . . . . .	6
2.3. Pruebas . . . . .	6
2.4. Árboles de búsqueda . . . . .	6
2.5. Análisis y conclusiones . . . . .	6
<b>3. Secuencia Mágica</b>	<b>6</b>
3.1. Modelo . . . . .	7
3.2. Detalles de implementación . . . . .	7
3.3. Pruebas . . . . .	7
3.4. Árboles de búsqueda . . . . .	7
3.5. Análisis y conclusiones . . . . .	7
<b>4. Acertijo Lógico</b>	<b>7</b>
4.1. Modelo . . . . .	7
4.2. Detalles de implementación . . . . .	8
4.3. Pruebas . . . . .	8
4.4. Árboles de búsqueda . . . . .	8

4.5. Análisis y conclusiones . . . . .	8
<b>5. Ubicación de personas en una reunión</b>	<b>8</b>
5.1. Modelo . . . . .	8
5.2. Detalles de implementación . . . . .	10
5.3. Pruebas . . . . .	10
5.4. Árboles de búsqueda . . . . .	12
5.5. Análisis y conclusiones . . . . .	12
<b>6. Construcción de un rectángulo</b>	<b>13</b>
6.1. Modelo . . . . .	13
6.2. Detalles de implementación . . . . .	13
6.3. Pruebas . . . . .	13
6.4. Árboles de búsqueda . . . . .	13
6.5. Análisis y conclusiones . . . . .	14

## 1. Sudoku

Puzzle en una grilla  $9 \times 9$  dividida en nueve cajas  $3 \times 3$ . 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  $3 \times 3$  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**: Si hay pista, se respeta:  $(G_{r,c} > 0) \Rightarrow X_{r,c} = G_{r,c}$  para todo  $r, c \in S$ .

**R2** — **Filas sin repetición**:  $\forall r \in S : all\_different([X_{r,c} \mid c \in S])$ .

**R3 — Columnas sin repetición:**  $\forall c \in S : all\_different([X_{r,c} \mid r \in S])$ .

**R4 — Cajas  $3 \times 3$  sin repetición:**  $\forall b_r, b_c \in \{0, 1, 2\} : all\_different([X_{3b_r+i, 3b_c+j} \mid i, j \in \{1, 2, 3\}])$ .

#### Restricciones redundantes (opcionales)

**R5 — Suma por fila = 45:**  $\forall r \in S : \sum_{c \in S} X_{r,c} = 45$ . Aporta poda lineal cuando faltan pocas celdas en la fila.

**R6 — Suma por columna = 45:**  $\forall c \in S : \sum_{r \in S} X_{r,c} = 45$ . Refuerza la propagación vertical.

**R7 — Suma por caja = 45:**  $\forall b_r, b_c \in \{0, 1, 2\} : \sum_{i=1}^3 \sum_{j=1}^3 X_{3b_r+i, 3b_c+j} = 45$ . Útil para cerrar subcuadrículas casi completas.

### 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, reduciendo el árbol sin afectar la corrección.

#### 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*. Mostramos **solo** corridas *con* redundancias, pues los árboles *con/sin* ellos se ven practicamente indenticos.

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 —o en ocasiones mayor— 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 corridas 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 cuanto a las estrategias, el desempeño depende del solver. En Gecode, `wdeg_split` dio sistemáticamente los menores *nodes/fail* en las tres instancias, superando a `ff_min` y con ventaja clara sobre `inorder_min`. En Chuffed, en cambio, `ff_min` fue la más consistente (menos retrocesos en los tres tests), mientras que `wdeg_split` no aportó ganancias y llegó a empeorar. Esto encaja con la forma en que cada motor explota la información: el conteo de conflictos de `dom/wdeg` guía bien la elección de variables cuando la propagación no “aplana” demasiado los dominios —como suele pasar en Gecode—, pero en Chuffed el aprendizaje de conflictos y una propagación más agresiva concentran rápidamente los fallos en variables de dominio pequeño, de modo que `first_fail` suele acertar antes y el *split* introduce sobrecoste sin reducir más el árbol. En términos prácticos, `inorder_min` es generalmente la menos eficaz, con la salvedad del caso trivial `test_01` en Chuffed donde queda muy cerca de `wdeg_split`.

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 optó por dejarlas desactivadas por defecto.

## **2. Kakuro**

Introducción al problema y alcance del modelado. Supuestos y parámetros clave.a

### **2.1. Modelo**

Definición de variables, dominios y restricciones principales. Justificación del modelo.

### **2.2. Detalles de implementación**

Restricciones redundantes, rompimiento de simetrías y decisiones técnicas.

### **2.3. Pruebas**

Casos de prueba, entradas, métricas y tablas o figuras de apoyo.

### **2.4. Árboles de búsqueda**

Nodos explorados, fallos, tiempos y efecto de estrategias de distribución.

### **2.5. Análisis y conclusiones**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur at dui sed justo viverra ultrices. Integer a nisl id enim ornare dictum. Mauris non lectus vel turpis posuere tincidunt. In hac habitasse platea dictumst. Donec et urna non velit tempus vulputate.

Suspendisse potenti. Phasellus lacinia, arcu et gravida pharetra, tortor nisl iaculis augue, eget porta libero sapien in odio. Sed imperdiet, turpis at facilisis varius, arcu velit aliquet justo, vitae convallis lorem ipsum id urna. Cras ut sem vel ex sagittis bibendum.

Praesent euismod, sapien a cursus molestie, risus metus feugiat lorem, vitae gravida enim felis id magna. Aliquam erat volutpat. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

## **3. Secuencia Mágica**

Introducción al problema y alcance del modelado. Supuestos y parámetros clave.

### **3.1. Modelo**

Definición de variables, dominios y restricciones principales. Justificación del modelo.

### **3.2. Detalles de implementación**

Restricciones redundantes, rompimiento de simetrías y decisiones técnicas.

### **3.3. Pruebas**

Casos de prueba, entradas, métricas y tablas o figuras de apoyo.

### **3.4. Árboles de búsqueda**

Nodos explorados, fallos, tiempos y efecto de estrategias de distribución.

### **3.5. Análisis y conclusiones**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur at dui sed justo viverra ultrices. Integer a nisl id enim ornare dictum. Mauris non lectus vel turpis posuere tincidunt. In hac habitasse platea dictumst. Donec et urna non velit tempus vulputate.

Suspendisse potenti. Phasellus lacinia, arcu et gravida pharetra, tortor nisl iaculis augue, eget porta libero sapien in odio. Sed imperdiet, turpis at facilisis varius, arcu velit aliquet justo, vitae convallis lorem ipsum id urna. Cras ut sem vel ex sagittis bibendum.

Praesent euismod, sapien a cursus molestie, risus metus feugiat lorem, vitae gravida enim felis id magna. Aliquam erat volutpat. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

## **4. Acertijo Lógico**

Introducción al problema y alcance del modelado. Supuestos y parámetros clave.

### **4.1. Modelo**

Definición de variables, dominios y restricciones principales. Justificación del modelo.

## 4.2. Detalles de implementación

Restricciones redundantes, rompimiento de simetrías y decisiones técnicas.

## 4.3. Pruebas

Casos de prueba, entradas, métricas y tablas o figuras de apoyo.

## 4.4. Árboles de búsqueda

Nodos explorados, fallos, tiempos y efecto de estrategias de distribución.

## 4.5. Análisis y conclusiones

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur at dui sed justo viverra ultrices. Integer a nisl id enim ornare dictum. Mauris non lectus vel turpis posuere tincidunt. In hac habitasse platea dictumst. Donec et urna non velit tempus vulputate.

Suspendisse potenti. Phasellus lacinia, arcu et gravida pharetra, tortor nisl iaculis augue, eget porta libero sapien in odio. Sed imperdiet, turpis at facilisis varius, arcu velit aliquet justo, vitae convallis lorem ipsum id urna. Cras ut sem vel ex sagittis bibendum.

Praesent euismod, sapien a cursus molestie, risus metus feugiat lorem, vitae gravida enim felis id magna. Aliquam erat volutpat. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

# 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* (**next**), *no adyacencia* (**separate**) y *cota máxima de distancia* (**distance**), 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.  $S = \{1, \dots, N\}$ .

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

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



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

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

### Variables

**V1** —  $\text{POS\_OF}_p$ : Posición que ocupa la persona  $p$ .  $\text{POS\_OF}_p \in \text{POS}$ ,  $p \in S$ .

**V2** —  $\text{PER\_AT}_i$ : Persona ubicada en la posición  $i$ .  $\text{PER\_AT}_i \in S$ ,  $i \in \text{POS}$ .

### Restricciones principales

**R1** — **Biección (canalización)**: La asignación es una permutación válida: cada persona ocupa exactamente una posición y cada posición contiene exactamente una persona.  $\text{inverse}(\text{POS\_OF}, \text{PER\_AT})$ .

**R2** — **Preferencias next**( $A, B$ ):  $A$  y  $B$  deben quedar adyacentes.  $\forall (A, B) \in \text{NEXT} : |\text{POS\_OF}_A - \text{POS\_OF}_B| = 1$ .

**R3** — **Preferencias separate**( $A, B$ ):  $A$  y  $B$  no pueden quedar adyacentes.  $\forall (A, B) \in \text{SEP} : |\text{POS\_OF}_A - \text{POS\_OF}_B| \geq 2$ .

**R4** — **Preferencias distance**( $A, B, M$ ): A lo sumo  $M$  personas entre  $A$  y  $B$ , equivalente a cota sobre distancia de posiciones.  $\forall (A, B, M) \in \text{DIST} : |\text{POS\_OF}_A - \text{POS\_OF}_B| \leq M + 1$ .

### Restricciones redundantes

**R5** — **Límite de apariciones en next**: Cada persona puede participar en a lo sumo dos relaciones de adyacencia, ya que en una fila solo puede tener un vecino a cada lado.  $\forall p \in S : \sum_i [p = \text{NEXT}[i, 1] \vee p = \text{NEXT}[i, 2]] \leq 2$ .

**R6** — **Consistencia entre next y separate**: Se evita que un mismo par de personas aparezca simultáneamente en ambas preferencias, pues sería una contradicción directa.  $\forall (A, B) \in \text{NEXT}, (C, D) \in \text{SEP} : \neg[(A, B) = (C, D) \vee (A, B) = (D, C)]$ .

### Restricciones de simetrías

**R7** — **Rompimiento de simetría izquierda–derecha**: Las soluciones reflejadas son equivalentes; para evitar duplicados, se fija  $\text{PER\_AT}_1 < \text{PER\_AT}_N$ .

## 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*. Esta canalización 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 la canalización *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 canónico comparando los extremos (PER\_AT[1] frente a PER\_AT[N]). Esto reduce la búsqueda sin afectar satisfacibilidad ni óptimos, siempre que no haya reglas que distingan explícitamente los extremos.

## 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 3:** 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 4:** 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 5:** 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, donde la propagación SAT-like de **Chuffed** evita retrocesos innecesarios y mejora la estabilidad del proceso.

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 sobre costo sin más poda.

El rompimiento de simetría redujo la exploración en `test_02` y `test_03`. Se observaron caídas claras 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 la canalización *inverse*, ya era suficientemente fuerte.

## 6. Construcción de un rectángulo

Introducción al problema y alcance del modelado. Supuestos y parámetros clave.

### 6.1. Modelo

Definición de variables, dominios y restricciones principales. Justificación del modelo.

### 6.2. Detalles de implementación

Restricciones redundantes, rompimiento de simetrías y decisiones técnicas.

### 6.3. Pruebas

Casos de prueba, entradas, métricas y tablas o figuras de apoyo.

### 6.4. Árboles de búsqueda

Nodos explorados, fallos, tiempos y efecto de estrategias de distribución.

## 6.5. Análisis y conclusiones

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur at dui sed justo viverra ultrices. Integer a nisl id enim ornare dictum. Mauris non lectus vel turpis posuere tincidunt. In hac habitasse platea dictumst. Donec et urna non velit tempus vulputate.

Suspendisse potenti. Phasellus lacinia, arcu et gravida pharetra, tortor nisl iaculis augue, eget porta libero sapien in odio. Sed imperdiet, turpis at facilisis varius, arcu velit aliquet justo, vitae convallis lorem ipsum id urna. Cras ut sem vel ex sagittis bibendum.

Praesent euismod, sapien a cursus molestie, risus metus feugiat lorem, vitae gravida enim felis id magna. Aliquam erat volutpat. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

## Referencias

- [1] John Freddy Belalcazar. Notas sobre modelamiento de csp. *Preprint*, 2024.
- [2] Samuel Galindo Cuevas. Estrategias de búsqueda en minimización. *Preprint*, 2024.
- [3] Nicolas Herrera Marulanda. Restricciones redundantes y rompimiento de simetrías. *Preprint*, 2024.