FACULTAD DE INGENIERÍA DE SISTEMAS
Y COMPUTACIÓN

Informe Taller 2

Extensiones del Job Shop Scheduling Problem

Programación por Restricciones

Autores:
[Código Estudiante 1]
[Código Estudiante 2]

Profesor: Robinson Duque

Noviembre 2025

Índice

1. Introducción

Este informe presenta el desarrollo e implementación de extensiones del problema clásico de Job Shop Scheduling (JSSP) utilizando MiniZinc. El JSSP es un problema de optimización combinatoria donde se deben planificar operaciones de múltiples trabajos en diferentes máquinas, respetando restricciones de precedencia y capacidad, con el objetivo de minimizar el tiempo total de finalización (makespan).

En este taller se abordan dos variaciones del problema que incorporan restricciones adicionales que reflejan contextos industriales y logísticos más realistas:

1. **Job Shop con Mantenimiento Programado:** Cada máquina debe detenerse periódicamente para mantenimiento preventivo, durante el cual no puede procesar operaciones.

2. **Job Shop con Operarios Limitados:** Algunas máquinas requieren operarios especializados, pero hay menos operarios disponibles que máquinas, lo que añade restricciones adicionales de asignación y concurrencia.

Para cada problema se presentan las variables de decisión, dominios, restricciones, detalles de implementación, restricciones redundantes, estrategias de ruptura de simetrías, estrategias de búsqueda, pruebas exhaustivas y análisis de resultados. Además, se exploran diferentes estrategias de búsqueda para evaluar su impacto en el rendimiento del solver.

El objetivo principal es demostrar la capacidad de extender modelos clásicos de programación por restricciones para incorporar condiciones reales, analizar el impacto de diferentes estrategias de búsqueda y optimización, y evaluar el comportamiento de los solvers en instancias de complejidad variable.

2. Problema 1.1: Job Shop con Mantenimiento Programado

2.1. Descripción del Problema

En esta variación del Job Shop Scheduling Problem, cada máquina debe detenerse periódicamente para realizar mantenimiento preventivo. Durante estos períodos de mantenimiento, la máquina no está disponible para procesar ninguna operación, lo que añade restricciones temporales adicionales al problema clásico.

Contexto industrial: En entornos de producción real, el mantenimiento preventivo es esencial para evitar fallas catastróficas, prolongar la vida útil de las máquinas y garantizar la calidad del producto. Sin embargo, estos períodos de inactividad deben planificarse cuidadosamente para minimizar su impacto en el tiempo total de producción.

Elementos del problema:

- Un conjunto de *jobs* (trabajos), cada uno con una secuencia ordenada de *tasks* (operaciones)
- Cada operación tiene una duración fija y debe ejecutarse en una máquina específica
- Las operaciones de un mismo trabajo deben ejecutarse en orden (restricciones de precedencia)
- Cada máquina puede ejecutar solo una operación a la vez
- **Nuevo:** Cada máquina tiene intervalos de mantenimiento $[a_m, b_m]$ donde no está disponible
- **Objetivo:** Minimizar el makespan (tiempo de finalización del último trabajo)

2.2. Modelamiento como CSP

2.2.1. Parámetros del Modelo

[Por completar con la descripción de parámetros]

2.2.2. Variables de Decisión

[Por completar con la descripción de variables]

2.2.3. Dominios

[Por completar con la descripción de dominios]

2.2.4. Restricciones

[Por completar con la descripción de restricciones principales]

2.3. Detalles de Implementación en MiniZinc

2.3.1. Aspectos Relevantes

[Por completar con aspectos técnicos de la implementación]

2.3.2. Restricciones Redundantes

[Por completar con restricciones redundantes y su justificación]

2.3.3. Ruptura de Simetrías

[Por completar con estrategias de ruptura de simetrías]

2.4. Estrategias de Búsqueda

[Por completar con descripción de estrategias exploradas]

2.5. Pruebas Realizadas

[Por completar con casos de prueba y resultados]

2.6. Análisis de Resultados

[Por completar con análisis comparativo]

2.7. Conclusiones del Problema 1.1

[Por completar con conclusiones específicas]

3. Problema 1.2a: Job Shop con Operarios Limitados

3.1. Descripción del Problema

En esta variación del Job Shop Scheduling Problem, se introduce la restricción de que hay un número limitado de operarios especializados disponibles. Aunque cada operación requiere ejecutarse en una máquina específica, también requiere la presencia de un operario para supervisarla u operarla. Como hay menos operarios que máquinas, no todas las máquinas pueden funcionar simultáneamente, lo que añade una nueva dimensión de complejidad al problema.

Contexto industrial: En entornos de producción modernos, aunque las máquinas puedan ser numerosas y especializadas, el personal calificado para operarlas es un recurso limitado y costoso. La planificación eficiente debe considerar tanto la disponibilidad de máquinas como de operarios, buscando además balancear la carga de trabajo para evitar cuellos de botella y fatiga.

Elementos del problema:

- Un conjunto de *jobs* con sus respectivas secuencias de *tasks*
- Cada operación tiene duración fija y máquina asignada
- Restricciones de precedencia dentro de cada trabajo
- Cada máquina procesa una operación a la vez
- **Nuevo:** Solo hay k operarios disponibles (donde $k <$ número de tareas concurrentes potenciales)
- **Nuevo:** Cada operación debe ser asignada a exactamente un operario
- **Nuevo:** Un operario no puede estar en dos operaciones simultáneas
- **Objetivos duales:**
  1. Minimizar el makespan (tiempo total)
  2. Balancear la carga de trabajo entre operarios

3.2. Modelamiento como CSP

3.2.1. Parámetros del Modelo

Los parámetros definen la instancia del problema:

- **jobs** (`int`): Número de trabajos a planificar
- **tasks** (`int`): Número de operaciones por trabajo (asumiendo estructura uniforme)
- **k** (`int`): Número de operarios disponibles

- **d** (`array[JOB, TASK] of int`): Matriz de duraciones, donde $d[i, j]$ es la duración de la tarea j del trabajo i

Adicionalmente se definen conjuntos para facilitar la iteración:

- **JOB** $= \{1, \ldots, \text{jobs}\}$
- **TASK** $= \{1, \ldots, \text{tasks}\}$
- **OP** $= \{1, \ldots, k\}$: Conjunto de operarios

Justificación: Esta estructura permite modelar problemas de tamaño variable manteniendo la claridad del modelo. La suposición de estructura uniforme (mismo número de tareas por trabajo) simplifica el modelo sin perder generalidad significativa.

3.2.2. Variables de Decisión

El modelo define las siguientes variables de decisión:

1. **s** (`array[JOB, TASK] of var 0..total`):
   - $s[i, j]$ representa el tiempo de inicio de la tarea j del trabajo i
   - Dominio: $[0, \text{total}]$ donde $\text{total} = \sum_{i,j} d[i, j]$ (cota superior trivial)
   - Esta es la variable principal que determina el cronograma
2. **o** (`array[JOB, TASK] of var OP`):
   - $o[i, j]$ representa qué operario es asignado a la tarea j del trabajo i
   - Dominio: $\{1, \ldots, k\}$ (identificadores de operarios)
   - Esta variable conecta las restricciones de capacidad de operarios con el cronograma
3. **end** (`var 0..total`):
   - Representa el tiempo de finalización del último trabajo (makespan)
   - Variable objetivo principal
4. **used** (`array[OP] of var bool`):
   - used$[p]$ es verdadero si y solo si el operario p es asignado a al menos una tarea
   - Variable auxiliar para romper simetrías
5. **carga** (`array[OP] of var 0..total`):
   - carga$[p]$ representa la carga total de trabajo asignada al operario p
   - Suma de duraciones de todas las tareas asignadas a ese operario
   - Variable derivada para el objetivo de balanceo

Justificación del modelamiento: La elección de representar tanto tiempos de inicio como asignaciones de operarios como variables permite al solver explorar ambas dimensiones simultáneamente, facilitando la propagación de restricciones. Las variables auxiliares (`used`, `carga`) son técnicamente redundantes pero esenciales para expresar restricciones de simetría y objetivos de balanceo de forma clara y eficiente.

3.2.3. Restricciones

Restricciones del Job Shop Clásico **1. Precedencia dentro de cada trabajo:**

$$\forall i \in \text{JOB}, \forall j \in \{1, \ldots, \text{tasks} - 1\} : s[i,j] + d[i,j] \leq s[i,j+1] \tag{1}$$

Interpretación: La tarea $j+1$ del trabajo i no puede comenzar hasta que la tarea j haya terminado. Esto captura la dependencia secuencial de operaciones dentro de un trabajo.

2. Capacidad de máquinas (una operación por tarea a la vez):

$$\forall j \in \text{TASK} : \text{disjunctive}([s[i,j] \mid i \in \text{JOB}], [d[i,j] \mid i \in \text{JOB}]) \tag{2}$$

Interpretación: Para cada índice de tarea j (que corresponde a una máquina en el modelo clásico), todas las operaciones que usan esa máquina deben ejecutarse sin solapamiento temporal. La restricción global `disjunctive` asegura que los intervalos $[s[i,j], s[i,j] + d[i,j])$ sean disjuntos.

3. Definición del makespan:

$$\forall i \in \text{JOB} : s[i, \text{tasks}] + d[i, \text{tasks}] \leq \text{end} \tag{3}$$

Interpretación: El makespan debe ser al menos el tiempo de finalización del último trabajo completado.

Restricciones de Operarios **4. No solapamiento de operarios:**

$$\begin{aligned} &\forall (i_1, j_1), (i_2, j_2) \in \text{JOB} \times \text{TASK} \text{ con } (i_1, j_1) < (i_2, j_2) : \\ &\qquad o[i_1, j_1] = o[i_2, j_2] \implies \\ &(s[i_1, j_1] + d[i_1, j_1] \leq s[i_2, j_2] \vee s[i_2, j_2] + d[i_2, j_2] \leq s[i_1, j_1]) \end{aligned} \tag{4}$$

Interpretación: Si dos tareas son asignadas al mismo operario, sus intervalos temporales no pueden solaparse. Esta es la restricción fundamental que limita el paralelismo en el sistema.

Nota técnica: La condición $(i_1, j_1) < (i_2, j_2)$ (orden lexicográfico) evita comparar cada par dos veces y la auto-comparación.

5. Definición de carga por operario:

$$\forall p \in \text{OP} : \text{carga}[p] = \sum_{i \in \text{JOB}, j \in \text{TASK}} d[i,j] \cdot \mathbb{1}_{o[i,j]=p} \tag{5}$$

donde $\mathbb{1}_{o[i,j]=p}$ es 1 si $o[i,j] = p$, y 0 en caso contrario.

Interpretación: La carga de cada operario es la suma de duraciones de todas las tareas que se le asignan.

Ruptura de Simetrías 6. Uso consecutivo de operarios:

$$\begin{aligned} &\forall p \in \text{OP} : \text{used}[p] \iff \exists (i,j) \in \text{JOB} \times \text{TASK} : o[i,j] = p \\ &\forall p \in \{2, \ldots, k\} : \text{used}[p] \implies \text{used}[p-1] \end{aligned} \tag{6}$$

Interpretación: Forzamos que los operarios se usen consecutivamente (1, 2, 3, ...), eliminando simetrías por permutación de operarios idénticos.

7. Anclaje del primer operario:

$$o[1,1] = 1 \tag{7}$$

Interpretación: La primera tarea del primer trabajo siempre usa el operario 1, eliminando rotaciones del conjunto de operarios.

8. Ordenamiento de cargas:

$$\forall p \in \{1, \ldots, k-1\} : \text{carga}[p] \geq \text{carga}[p+1] \tag{8}$$

Interpretación: Las cargas se ordenan de mayor a menor, eliminando permutaciones de operarios con cargas idénticas. Esta restricción es compatible con el objetivo de balanceo.

Restricción Redundante 9. Restricción cumulative (poda de dominios):

$$\text{cumulative}(S_{\text{all}}, D_{\text{all}}, R_{\text{all}}, k) \tag{9}$$

donde:

- $S_{\text{all}} = [s[i,j] \mid i \in \text{JOB}, j \in \text{TASK}]$: vector de tiempos de inicio
- $D_{\text{all}} = [d[i,j] \mid i \in \text{JOB}, j \in \text{TASK}]$: vector de duraciones
- $R_{\text{all}} = [1 \mid i \in \text{JOB}, j \in \text{TASK}]$: cada tarea requiere 1 recurso
- k: capacidad máxima del recurso

Interpretación: En cualquier momento, a lo sumo k tareas pueden estar ejecutándose simultáneamente (porque solo hay k operarios). Esta restricción es **redundante** porque se deriva lógicamente de la restricción 4, pero la restricción global `cumulative` implementa algoritmos de propagación especializados que podan el espacio de búsqueda más eficientemente.

¿Por qué es útil esta redundancia?

- La restricción 4 propaga cuando dos tareas comparten operario
- La restricción `cumulative` propaga globalmente sobre todas las tareas simultáneas
- En la práctica, reduce drásticamente los nodos explorados

3.3. Detalles de Implementación en MiniZinc

3.3.1. Aspectos Relevantes de la Implementación

1. Uso de restricciones globales:

```
constraint forall(j in TASK) (
  disjunctive([s[i,j] | i in JOB], [d[i,j] | i in JOB])
);
```

La restricción global `disjunctive` es mucho más eficiente que descomponer manualmente en disyunciones. Implementa algoritmos especializados de propagación para scheduling.

2. Restricciones de no-solapamiento de operarios:

Se optó por una formulación explícita con implicación en lugar de usar `disjunctive` para operarios porque:

- La asignación de operarios es una variable de decisión
- El no-solapamiento es condicional (solo si comparten operario)
- Permite mayor control sobre la propagación

3. Variables auxiliares para balanceo:

Las variables `maxload` y `minload` se definen como:

```
var 0..total: maxload = max(p in OP)(carga[p]);
var 0..total: minload = min(p in OP)(carga[p]);
```

Estas son variables derivadas pero el solver las trata eficientemente gracias a las optimizaciones de MiniZinc.

4. Cota superior opcional:

El modelo incluye un parámetro `ub_end` que permite especificar una cota superior conocida para el makespan, facilitando la prueba de instancias específicas o la comparación con soluciones conocidas.

3.4. Estrategias de Búsqueda

Se implementaron tres estrategias de búsqueda diferentes para evaluar su impacto en el rendimiento:

3.4.1. Estrategia 1: Búsqueda Libre

```
solve minimize W * end + (maxload - minload);
```

Características:

- No se especifica una anotación de búsqueda explícita
- El solver (Gecode) usa sus heurísticas por defecto

- Típicamente explora variables en orden de declaración con seleccción de valor por defecto

Función objetivo: $W \cdot \text{end} + (\text{maxload} - \text{minload})$
donde $W = \text{total}+1$ es un peso grande que prioriza makespan sobre balanceo.
Ventajas esperadas: Simplicidad, sin necesidad de ajustar parámetros.
Desventajas esperadas: Puede explorar variables en orden subóptimo, llevando a mayor número de nodos.

3.4.2. Estrategia 2: Búsqueda Secuencial con dom_w_deg

```
solve
:: seq_search([
     int_search([s[i,j] | i in JOB, j in TASK],
                dom_w_deg, indomain_min),
     int_search([o[i,j] | i in JOB, j in TASK],
                first_fail, indomain_min)
   ])
minimize W * end + (maxload - minload);
```

Características:

- **Fase 1:** Decide tiempos de inicio (`s`) usando `dom_w_deg`
  - `dom_w_deg`: Selecciona la variable con mayor ratio de peso-de-restricciones / tamaño-de-dominio
  - "Peso" se incrementa cuando una restricción falla
  - Prioriza variables que han causado conflictos recientemente
  - `indomain_min`: Prueba el valor mínimo del dominio primero
- **Fase 2:** Decide asignaciones de operarios (`o`) usando `first_fail`
  - `first_fail`: Selecciona la variable con el dominio más pequeño
  - Ataca primero las decisiones más restringidas

Justificación:

- Los tiempos de inicio son más críticos para el makespan
- `dom_w_deg` se adapta dinámicamente a la estructura del problema
- Las asignaciones de operarios pueden propagarse después de fijar tiempos

Ventajas esperadas: Mejor adaptación a conflictos, exploración más guiada.
Desventajas esperadas: Mayor overhead computacional por heurística dinámica.

3.4.3. Estrategia 3: Operarios Primero

```
solve
:: seq_search([
     int_search([o[i,j] | i in JOB, j in TASK],
                first_fail, indomain_min),
     int_search([s[i,j] | i in JOB, j in TASK],
                first_fail, indomain_min)
   ])
minimize W * end + (maxload - minload);
```

Características:

- **Fase 1:** Decide primero todas las asignaciones de operarios
- **Fase 2:** Luego decide tiempos de inicio
- Ambas fases usan `first_fail`

Justificación:

- Agrupar tareas por operario antes de planificar tiempos
- Puede facilitar la detección temprana de infactibilidades relacionadas con operarios
- Una vez fijados los operarios, el problema se descompone en subproblemas de scheduling

Ventajas esperadas: Buena descomposición del problema, posible poda temprana.

Desventajas esperadas: Puede fijar asignaciones de operarios prematuramente sin considerar consecuencias temporales.

3.5. Pruebas Realizadas

3.5.1. Configuración de las Pruebas

Las pruebas se realizaron utilizando el script `run_jobshop_tests.sh` con los siguientes parámetros:

- **Solver:** Gecode 6.3.0
- **Límite de tiempo:** 60,000 ms (60 segundos) por instancia
- **Semilla aleatoria:** 1 (para reproducibilidad)
- **Modelos evaluados:** 3 estrategias de búsqueda
- **Instancias:** 11 archivos de datos (data00.dzn a data10.dzn)

Justificación del límite de tiempo: Las instancias de mayor tamaño presentaban tiempos de ejecución excesivamente largos (superiores a 8 minutos con múltiples hilos) sin convergencia. El límite de 60 segundos permite evaluar el rendimiento relativo de las estrategias en un tiempo razonable, priorizando la exploración inicial del espacio de búsqueda.

3.5.2. Descripción de las Instancias

Todas las instancias siguen el formato:

```
jobs = <numero>;
tasks = <numero>;
k = <numero>;
d = [| <matriz de duraciones> |];
```

Características de las instancias:

Instancia	Jobs	Tasks	Operarios (k)	Operaciones Totales	Carga Total
data00	4	4	3	16	86
data01	5	5	2	25	108
data02	5	5	3	25	112
data03	5	5	4	25	120
data04	6	4	2	24	118
data05	6	4	3	24	120
data06	6	6	4	36	138
data07	7	5	2	35	162
data08	7	5	3	35	155
data09	8	5	3	40	175
data10	8	6	4	48	187

Cuadro 1: Características de las instancias de prueba

Observaciones:

- Las instancias aumentan progresivamente en complejidad
- La relación operarios/tareas varía (desde $k/\text{tasks} = 0{,}4$ hasta 0,8)
- La carga total aumenta de 86 a 187

3.5.3. Resultados Experimentales

Estrategia 1: Búsqueda Libre Observaciones críticas:

- **Desbalance máximo:** En todas las instancias, el desbalance es igual al makespan
- **Interpretación:** Esto indica que **todos los operarios excepto uno tienen carga cero**

Instancia	Makespan	Desbalance	Nodos	Fallos	Propagaciones
data00	86	86	8,656,470	4,328,217	928,732,226
data01	108	108	9,380,158	4,690,062	934,887,110
data02	112	112	9,141,499	4,570,731	910,658,768
data03	120	120	8,977,618	4,488,791	801,706,349
data04	118	118	8,163,317	4,081,638	841,537,470
data05	120	120	8,483,181	4,241,571	858,491,031
data06	138	138	8,708,307	4,354,133	783,575,748
data07	162	162	7,897,032	3,948,492	663,627,990
data08	155	155	8,461,777	4,230,865	793,876,685
data09	175	175	7,587,331	3,793,638	710,109,798
data10	187	187	7,500,955	3,750,452	750,849,628

Cuadro 2: Resultados de Estrategia 1 (Búsqueda Libre)

- En la columna de carga (no mostrada) se observa: $[\text{carga\_total}, 0, 0, \ldots]$
- **Causa:** La estrategia por defecto no considera el objetivo de balanceo efectivamente
- **Número de nodos:** Muy alto (8-9 millones en instancias pequeñas)
- **Propagaciones:** Entre 600 millones y 900 millones

Conclusión: Esta estrategia encuentra soluciones triviales donde un solo operario hace todo el trabajo. Aunque técnicamente válidas, estas soluciones no son prácticas ni deseables.

Instancia	Makespan	Desbalance	Nodos	Fallos	Propagaciones
data00	39	1	11,048,555	5,524,261	1,178,703,001
data01	57	4	4,946,586	2,473,281	347,713,759
data02	54	1	9,969,532	4,984,745	1,204,675,845
data03	45	0	4,061	1,955	1,410,504
data04	62	0	2,868,894	1,434,410	486,862,749
data05	43	0	396,851	198,299	52,210,307
data06	65	1	8,615,337	4,307,617	952,185,260
data07	86	6	2,513,049	1,256,462	363,079,829
data08	66	1	6,460,261	3,230,108	1,050,313,215
data09	71	1	5,859,083	2,929,506	996,370,298
data10	84	1	6,196,054	3,097,958	857,916,673

Cuadro 3: Resultados de Estrategia 2 (dom_w_deg para tiempos)

Estrategia 2: Búsqueda Secuencial con dom_w_deg Observaciones importantes:

- **Makespan dramáticamente reducido:** Comparado con Estrategia 1

- data00: 86 → 39 (54.7 % reducción)
- data01: 108 → 57 (47.2 % reducción)
- data03: 120 → 45 (62.5 % reducción)

- **Balanceo significativo:** Desbalance entre 0 y 6
  - data03, data04, data05: Desbalance = 0 (balanceo perfecto)
  - Mayoría: Desbalance ≤ 1 (casi perfecto)
- **Nodos explorados:** Variable, pero con casos de eficiencia extrema
  - data03: Solo 4,061 nodos (vs 8.9M en Estrategia 1)
  - data05: 396,851 nodos (vs 8.4M)
- **Propagaciones:** Aún muy altas, pero eficiencia por nodo mejorada

Conclusión: Esta estrategia es **significativamente superior**, logrando soluciones de alta calidad con balanceo efectivo y makespans mínimos. La heurística `dom_w_deg` demuestra su valor adaptativo.

Instancia	Makespan	Desbalance	Nodos	Fallos	Propagaciones
data00	86	86	10,717,053	5,358,504	1,146,528,186
data01	108	108	15,223,333	7,611,644	1,026,296,393
data02	112	112	11,312,284	5,656,120	1,085,806,822
data03	120	120	12,804,492	6,402,224	1,098,461,046
data04	118	118	13,626,036	6,812,990	995,898,055
data05	120	120	16,699,548	8,349,745	1,152,613,520
data06	138	138	15,456,023	7,727,984	1,066,657,338
data07	162	162	9,797,646	4,898,789	942,471,122
data08	155	155	8,666,460	4,333,197	958,498,638
data09	175	175	8,088,673	4,044,299	817,757,449
data10	187	187	8,660,372	4,330,147	621,775,327

Cuadro 4: Resultados de Estrategia 3 (Operarios Primero)

Estrategia 3: Operarios Primero Observaciones:

- **Resultados idénticos a Estrategia 1:** Mismo makespan, mismo desbalance extremo
- **Más nodos explorados:** Peor que Estrategia 1 en casi todas las instancias
  - data01: 15.2M nodos (vs 9.4M en Estrategia 1)
  - data05: 16.7M nodos (vs 8.5M en Estrategia 1)
- **Más propagaciones:** Overhead adicional sin beneficio

Conclusión: Decidir operarios primero sin información temporal resulta en decisiones pobres que luego no pueden revertirse. Esta estrategia es **estrictamente inferior** a las otras dos.

3.6. Análisis Comparativo

3.6.1. Comparación de Makespan

Instancia	Estrategia 1	Estrategia 2	Estrategia 3
data00	86	**39**	86
data01	108	**57**	108
data02	112	**54**	112
data03	120	**45**	120
data04	118	**62**	118
data05	120	**43**	120
data06	138	**65**	138
data07	162	**86**	162
data08	155	**66**	155
data09	175	**71**	175
data10	187	**84**	187
Mejora media	—	**54.3 %**	—

Figura 1: Comparación de makespan entre estrategias

Conclusiones:

- Estrategia 2 **siempre** encuentra mejores soluciones
- Mejora promedio: 54.3 % reducción de makespan
- Mejora máxima: 62.5 % (data03)
- Mejora mínima: 47.2 % (data01)

3.6.2. Comparación de Balanceo

El desbalance mide la diferencia entre la carga máxima y mínima de los operarios:

- **Estrategias 1 y 3:** Desbalance = Makespan (un operario hace todo)
- **Estrategia 2:** Desbalance ≤ 6 en todas las instancias
  - 3 instancias con balanceo perfecto (desbalance = 0)
  - 6 instancias con desbalance = 1 (casi perfecto)

Ejemplo concreto (data03):

- Carga total: 120
- Operarios: 4
- Estrategia 2: carga = [30, 30, 30, 30] → desbalance = 0
- Estrategias 1/3: carga = [120, 0, 0, 0] → desbalance = 120

3.6.3. Comparación de Eficiencia Computacional

Métrica	**Estrategia 1**	**Estrategia 2**	**Estrategia 3**
Nodos promedio	8,359,786	5,720,751	11,914,174
Propagaciones promedio	816,191,255	771,920,976	992,342,155
Mejor instancia (nodos)	7,500,955	4,061	8,088,673

Cuadro 5: Eficiencia computacional promedio

Observaciones:

- Estrategia 2 tiene el **mejor promedio de nodos**
- Estrategia 3 es la **menos eficiente** (42 % más nodos que Estrategia 1)
- La varianza en Estrategia 2 es alta: desde 4K hasta 11M nodos
- Estrategias 1 y 3 muestran comportamiento consistentemente malo

3.6.4. Análisis de Escalabilidad

Observando la tendencia de nodos explorados con el tamaño del problema:

- **Estrategias 1 y 3:** Nodos decrece ligeramente con instancias grandes
  - Razón probable: Alcanzan límite de tiempo antes, reportan soluciones triviales tempranas
- **Estrategia 2:** Comportamiento no monótono
  - data03, data05: Muy eficiente (estructura favorable)
  - data00, data02, data06: Menos eficiente (estructura compleja)

Interpretación: La estructura específica de cada instancia (distribución de duraciones, número de operarios) afecta significativamente la dificultad del problema para Estrategia 2, mientras que las otras estrategias fallan uniformemente.

3.7. Conclusiones del Problema 1.2a

1. **Importancia crítica de la estrategia de búsqueda:** Los resultados demuestran que la elección de estrategia puede significar la diferencia entre soluciones triviales inaceptables y soluciones de alta calidad. En este problema, la diferencia no es marginal sino **cualitativa**.

2. **Superioridad de dom_w_deg para problemas de scheduling multi-recurso:** La heurística dinámica `dom_w_deg` demostró adaptarse efectivamente a la estructura del problema, logrando:

   - Reducción promedio del makespan del 54.3 %
   - Balanceo casi perfecto de cargas (desbalance ≤ 6)
   - Eficiencia computacional superior en promedio

3. **El orden de las decisiones importa:** Decidir asignaciones de operarios antes que tiempos (Estrategia 3) resulta en decisiones prematuras sin suficiente información, llevando a soluciones pobres. El enfoque correcto es decidir tiempos primero (donde la información de duraciones y precedencias guía la búsqueda) y luego asignar operarios.

4. **Efectividad de restricciones redundantes:** La restricción `cumulative`, aunque redundante lógicamente, proporciona poda adicional crucial. Sin embargo, los resultados muestran que **por sí sola no es suficiente**: Estrategias 1 y 3 incluyen esta restricción pero fallan igualmente.

5. **Objetivos múltiples requieren pesos cuidadosos:** La función $W \cdot \text{end} + \text{desbalance}$ con $W = \text{total} + 1$ efectívamente prioriza makespan sin ignorar completamente el balanceo, pero **solo cuando la estrategia de búsqueda explora soluciones diversas**. Las estrategias por defecto convergen prematuramente a soluciones con un solo operario.

6. **Ruptura de simetrías esencial:** Las restricciones de ordenamiento de operarios (uso consecutivo, ordenamiento de cargas) son fundamentales para reducir el espacio de búsqueda. Sin embargo, deben combinarse con estrategias de búsqueda apropiadas para ser efectivas.

7. **Variabilidad estructural:** Diferentes instancias con tamaños similares pueden tener dificultades muy distintas. La relación operarios/tareas, distribución de duraciones y estructura de precedencias interactúan de formas complejas que afectan la eficiencia del solver.

8. **Límites prácticos de tiempo:** El límite de 60 segundos fue necesario para completar todas las pruebas, pero probablemente interrumpió la búsqueda en algunas instancias. Estrategia 2 podría encontrar soluciones aún mejores con más tiempo, mientras que Estrategias 1 y 3 probablemente no mejorarían significativamente.

Recomendación final: Para problemas de Job Shop con operarios limitados, se recomienda enfáticamente usar estrategias de búsqueda secuenciales que decidan tiempos de inicio con heurísticas dinámicas (`dom_w_deg` o similar) antes de asignar operarios con `first_fail`. Las estrategias por defecto o que priorizan asignaciones de recursos antes que decisiones temporales deben evitarse.

4. Problema 1.2b: Job Shop con Operarios Limitados y Habilidades

4.1. Descripción del Problema

Esta es una extensión del problema de operarios limitados donde además se consideran habilidades específicas: cada operación requiere ciertas habilidades y cada operario posee un conjunto de habilidades. Una operación solo puede ser asignada a un operario que posea todas las habilidades necesarias.

Contexto industrial: En entornos de producción reales, no todos los operarios están calificados para todas las tareas. Las certificaciones, experiencia y especializaciones determinan quién puede operar cada máquina o realizar cada operación.

Elementos adicionales:

- Cada operación requiere un conjunto de habilidades específicas
- Cada operario posee un conjunto de habilidades
- Un operario solo puede ejecutar operaciones para las cuales tiene todas las habilidades requeridas
- Sigue habiendo limitación en el número de operarios (k ¡número de máquinas)
- **Objetivo dual:** Minimizar makespan y balancear la carga de trabajo

4.2. Modelamiento como CSP

4.2.1. Parámetros del Modelo

[Por completar con la descripción de parámetros]

4.2.2. Variables de Decisión

[Por completar con la descripción de variables]

4.2.3. Dominios

[Por completar con la descripción de dominios]

4.2.4. Restricciones

[Por completar con la descripción de restricciones principales]

4.3. Detalles de Implementación en MiniZinc

4.3.1. Aspectos Relevantes

[Por completar con aspectos técnicos de la implementación]

4.3.2. Restricciones Redundantes

[Por completar con restricciones redundantes y su justificación]

4.3.3. Ruptura de Simetrías

[Por completar con estrategias de ruptura de simetrías]

4.4. Estrategias de Búsqueda

[Por completar con descripción de estrategias exploradas]

4.5. Pruebas Realizadas

[Por completar con casos de prueba y resultados]

4.6. Análisis de Resultados

[Por completar con análisis comparativo]

4.7. Conclusiones del Problema 1.2b

[Por completar con conclusiones específicas]

5. Conclusiones Generales

El desarrollo de este taller ha permitido aplicar técnicas avanzadas de modelamiento de problemas de optimización combinatoria mediante programación por restricciones. La extensión del Job Shop Scheduling Problem clásico con restricciones adicionales ha demostrado la flexibilidad y potencia de MiniZinc para modelar problemas industriales complejos.

Los aspectos más relevantes aprendidos incluyen:

1. **Modelamiento de restricciones complejas:** La incorporación de restricciones adicionales como mantenimiento programado y disponibilidad limitada de operarios requiere un análisis cuidadoso de las interacciones entre diferentes recursos y restricciones temporales.

2. **Uso estratégico de restricciones redundantes:** La inclusión de restricciones redundantes como `cumulative` puede reducir significativamente el espacio de búsqueda y mejorar los tiempos de resolución, incluso cuando no agregan información semántica nueva al modelo.

3. **Ruptura de simetrías:** La identificación y eliminación de simetrías, especialmente en la asignación de operarios, es crucial para evitar explorar soluciones equivalentes y reducir el árbol de búsqueda.

4. **Impacto de estrategias de búsqueda:** Las diferentes estrategias de búsqueda (`first_fail`, `dom_w_deg`, `seq_search`) tienen impactos significativamente diferentes en el rendimiento, dependiendo de la estructura del problema y la instancia específica.

5. **Trade-offs en optimización multi-objetivo:** Balancear múltiples objetivos como minimizar makespan y balancear carga de trabajo requiere técnicas de agregación cuidadosas y análisis de sensibilidad de pesos.

6. **Escalabilidad y límites computacionales:** Las instancias más grandes demuestran los límites prácticos de los solvers de propósito general, motivando el uso de técnicas como límites de tiempo, cotas superiores y estrategias de búsqueda especializadas.

6. Referencias

- MiniZinc Documentation: `https://www.minizinc.org/doc-latest/en/`
- MiniZinc Handbook, Peter J. Stuckey et al.
- Taller 2 – Programación por Restricciones, Universidad del Valle, 2025
- Brucker, P. (2007). Scheduling Algorithms. Springer.
- Baptiste, P., Le Pape, C., & Nuijten, W. (2012). Constraint-Based Scheduling. Springer.