
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

Esta variación del JSSP incorpora un número limitado $k$ de operarios especializados. Cada operación requiere ejecutarse en una máquina específica $y$ la supervisión de un operario, quien no puede atender múltiples operaciones simultáneamente. Esto limita el paralelismo del sistema aún con máquinas disponibles.

**Elementos adicionales al JSSP clásico:**

- $k$ operarios disponibles ($k <$ potencial de tareas simultáneas)
- Asignación obligatoria de operario por operación
- Restricción de exclusividad temporal del operario
- **Objetivo dual:** Minimizar makespan y balancear carga entre operarios

## 3.2. Modelamiento como CSP

### 3.2.1. Parámetros y Variables

**Parámetros:** `jobs`, `tasks`, `k` (operarios), `d[JOB,TASK]` (duraciones).
**Variables principales:**

- `s[JOB,TASK]`: Tiempos de inicio de cada operación
- `o[JOB,TASK]`: Asignación de operario a cada operación
- `end`: Makespan (tiempo total)
- `carga[OP]`: Carga de trabajo por operario (para balanceo)
- `used[OP]`: Indicador de operarios utilizados (ruptura de simetrías)

### 3.2.2. Restricciones Principales

**Job Shop clásico:**

1. Precedencia: $s[i,j] + d[i,j] \leq s[i,j+1]$ (secuencialidad dentro de cada trabajo)
2. Capacidad de máquinas: `disjunctive` por cada tarea (no solapamiento)
3. Makespan: $s[i, \text{tasks}] + d[i, \text{tasks}] \leq \text{end}$

**Operarios limitados:**

4. No solapamiento: Si $o[i_1, j_1] = o[i_2, j_2]$ entonces intervalos temporales disjuntos

5. Carga: $\text{carga}[p] = \sum d[i,j] \cdot \mathbb{1}_{o[i,j]=p}$

**Ruptura de simetrías:**

6. Uso consecutivo: $\text{used}[p] \implies \text{used}[p-1]$
7. Anclaje: $o[1,1] = 1$
8. Ordenamiento: $\text{carga}[p] \geq \text{carga}[p+1]$

**Redundante (poda eficiente):**

9. `cumulative`$(S, D, [1, \ldots, 1], k)$: A lo sumo $k$ tareas simultáneas. Redundante lógicamente pero mejora propagación drásticamente.

## 3.3. Detalles de Implementación

**Aspectos clave:**

- Uso de restricciones globales (`disjunctive`, `cumulative`) para propagación eficiente
- Formulación explícita para no-solapamiento de operarios (condicional a asignación)
- Variables derivadas (`maxload`, `minload`) para objetivo de balanceo
- Función objetivo: $W \cdot \text{end} + (\text{maxload} - \text{minload})$ con $W = \text{total} + 1$

## 3.4. Estrategias de Búsqueda Implementadas

Se implementaron tres estrategias para evaluar su impacto en calidad de solución y eficiencia:

1. **Estrategia 1 – Búsqueda Libre:** Sin anotaciones explícitas, usa heurísticas por defecto del solver. Probada con Gecode, Chuffed y HiGHS para comparar comportamiento entre solvers.
2. **Estrategia 2 – `dom_w_deg` para tiempos:**

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

   Decide tiempos con heurística adaptativa (`dom_w_deg`), luego operarios con `first_fail`. Probada solo con Gecode.

3. **Estrategia 3 – Operarios primero:** Decide operarios antes que tiempos, ambos con `first_fail`. Hipótesis: agrupar por operario facilita scheduling. Probada solo con Gecode.

## 3.5. Configuración de Pruebas

Las pruebas fueron ejecutadas mediante el script `run_jobshop_tests.sh` bajo las siguientes condiciones:

**Parámetros del script:**

- **Límite de tiempo:** 60,000 ms (60 segundos) por instancia
- **Semilla aleatoria:** 1 (para reproducibilidad)
- **Modo de solución:** Mejor solución encontrada (no todas las intermedias)
- **Estadísticas habilitadas:** `-s` (para capturar nodos, propagaciones, fallos)
- **Instancias:** 11 archivos (data00.dzn a data10.dzn)
- **Modelos:** 3 estrategias (jobshop_search_1.mzn, jobshop_search_2.mzn, jobshop_search_3.mzn)

**Solvers utilizados:**

- Estrategia 1: Gecode 6.3.0, Chuffed, HiGHS (comparación multi-solver)
- Estrategias 2–3: Gecode 6.3.0 únicamente (`seq_search` no soportado por otros solvers)

| Inst. | Jobs | Tasks | k | Ops. | Carga |
|---|---|---|---|---|---|
| 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: Instancias (complejidad creciente, $k$/tasks $\in [0{,}4, 0{,}8]$)

### 3.5.1. Resultados Experimentales

**Estrategia 1 – Comparación Multi-Solver**

- **Gecode:** 100 % triviales, **todas las instancias alcanzaron el límite de 60s** sin encontrar soluciones no triviales
- **Chuffed:** 27 % críticos, **todas las instancias alcanzaron timeout** pero con mejores soluciones intermedias

| Inst. | Gecode Mks | Gecode Desb | Chuffed Mks | Chuffed Desb | HiGHS Mks | HiGHS Desb |
|---|---|---|---|---|---|---|
| data00 | 86 | 86 | 32 | 1 | 32 | 1 |
| data01 | 108 | 108 | 58 | 2 | 55 | 0 |
| data02 | 112 | 112 | 51 | 1 | 44 | 1 |
| data03 | 120 | 120 | 45 | 16 | 45 | 0 |
| data04 | 118 | 118 | 110 | 4 | 64 | 4 |
| data05 | 120 | 120 | 61 | 7 | 43 | 0 |
| data06 | 138 | 138 | 73 | 38 | 50 | 1 |
| data07 | 162 | 162 | 140 | 0 | 88 | 0 |
| data08 | 155 | 155 | 152 | 78 | 64 | 2 |
| data09 | 175 | 175 | 174 | 98 | 70 | 1 |
| data10 | 187 | 187 | 187 | 1 | 69 | 14 |
| **Prom.** | 134.6 | 134.6 | 98.5 | 22.4 | 56.7 | 2.2 |

Cuadro 2: Estrategia 1 (búsqueda libre) con 3 solvers

- **HiGHS:** 0 % problemáticos, 45 % perfectos, **ninguna instancia alcanzó timeout** (resolución completa)

**HiGHS claramente superior.**

| Inst. | Est. 1 (libre) Mks | Dsb | Nod(M) | Prop(M) | Est. 2 (dom_w_deg) Mks | Dsb | Nod(M) | Prop(M) | Est. 3 (op primero) Mks | Dsb | Nod(M) | Prop(M) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| data00 | 86 | 86 | 8.7 | 929 | 39 | 1 | 11.0 | 1179 | 86 | 86 | 10.7 | 1147 |
| data01 | 108 | 108 | 9.4 | 935 | 57 | 4 | 4.9 | 348 | 108 | 108 | 15.2 | 1026 |
| data02 | 112 | 112 | 9.1 | 911 | 54 | 1 | 10.0 | 1205 | 112 | 112 | 11.3 | 1086 |
| data03 | 120 | 120 | 9.0 | 802 | 45 | 0 | 0.004 | 1.4 | 120 | 120 | 12.8 | 1098 |
| data04 | 118 | 118 | 8.2 | 842 | 62 | 0 | 2.9 | 487 | 118 | 118 | 13.6 | 996 |
| data05 | 120 | 120 | 8.5 | 858 | 43 | 0 | 0.4 | 52 | 120 | 120 | 16.7 | 1153 |
| data06 | 138 | 138 | 8.7 | 784 | 65 | 1 | 8.6 | 952 | 138 | 138 | 15.5 | 1067 |
| data07 | 162 | 162 | 7.9 | 664 | 86 | 6 | 2.5 | 363 | 162 | 162 | 9.8 | 942 |
| data08 | 155 | 155 | 8.5 | 794 | 66 | 1 | 6.5 | 1050 | 155 | 155 | 8.7 | 958 |
| data09 | 175 | 175 | 7.6 | 710 | 71 | 1 | 5.9 | 996 | 175 | 175 | 8.1 | 818 |
| data10 | 187 | 187 | 7.5 | 751 | 84 | 1 | 6.2 | 858 | 187 | 187 | 8.7 | 622 |
| **Prom.** | 134.6 | 134.6 | 8.4 | 816 | 61.1 | 1.5 | 5.4 | 772 | 134.6 | 134.6 | 11.9 | 992 |

Cuadro 3: Comparación de estrategias en Gecode 6.3.0

**Estrategias 2 y 3 – Solo Gecode**

- **Est.1 vs Est.2:** `dom_w_deg` reduce makespan 55 % (134.6 → 61.1) y desbalance 99 % (134.6 → 1.5). **Est.1 y Est.3 alcanzaron timeout en todas las instancias**, mientras que **Est.2 también alcanzó timeout universal** pero encontrando soluciones drásticamente mejores.
- **Est.1 vs Est.3:** Calidad idéntica (ambas triviales por timeout), pero Est.3 usa 42 % más nodos y 22 % más propagaciones antes de agotar tiempo.
- **Est.2 eficiencia:** Mejor calidad con menos nodos promedio que Est.1 (5.4M vs 8.4M), pero alta variabilidad. A pesar del timeout, la heurística adaptativa explora mejor el espacio de búsqueda.
- **data03 notable:** Est.2 resuelve con solo 4K nodos (balanceo perfecto), vs 9M en Est.1. Única instancia sin timeout en Est.2.

## 3.6. Análisis Comparativo

### 3.6.1. Hallazgo 1: El Solver Importa Tanto Como la Estrategia

Comparando resultados de Estrategia 1 (libre) entre solvers:

- **HiGHS:** Makespan 56.7, desbalance 2.2, 2.5K nodos
- **Gecode Est.2 (con anotaciones):** Makespan 61.1, desbalance 1.5, 5.4M nodos
- **Chuffed:** Makespan 98.5, desbalance 22.4, 1.5M nodos
- **Gecode Est.1:** Makespan 134.6, desbalance 134.6, 8.4M nodos

**Conclusión:** HiGHS sin anotaciones supera a Gecode con `dom_w_deg`, demostrando que heurísticas internas bien diseñadas eliminan necesidad de anotaciones explícitas. Cabe destacar, que los resultados triviales de Gecode Est.1 se deben al agotamiento del límite de tiempo (60s) sin encontrar soluciones mejores, mientras que HiGHS resuelve óptimamente sin alcanzar el timeout.

### 3.6.2. Hallazgo 2: Impacto de Estrategias en Gecode

Comparando las 3 estrategias implementadas (ver Tabla 3):

- **Est.1:** Makespan prom. 134.6, desbalance 134.6 (100 % triviales), 8.4M nodos, **timeout en todas las instancias**
- **Est.2:** Makespan prom. 61.1 (55 % mejor), desbalance 1.5 (99 % mejor), 5.4M nodos, **timeout en 10/11 instancias** (solo data03 terminó antes)
- **Est.3:** Makespan prom. 134.6 (idéntico a Est.1), 11.9M nodos (42 % peor eficiencia), **timeout en todas las instancias**
- **Est.2 eficiencia variable:** data03 con 4K nodos (única sin timeout) vs data00 con 11M nodos (alta dependencia de estructura)
- **Timeout universal en Gecode:** Las 3 estrategias agotan el límite de 60s en la mayoría de instancias, pero Est.2 encuentra mejores soluciones intermedias gracias a `dom_w_deg`

## 3.7. Conclusiones

1. **HiGHS es el mejor solver para este problema:** Sin anotaciones de búsqueda, logra makespan promedio 56.7 (58 % mejor que Gecode sin anotaciones, 7 % mejor que Gecode con `dom_w_deg`), desbalance máximo 14, y solo 2.5K nodos. **Ninguna instancia alcanzó el límite de 60s,** indicando resolución completa. Sus heurísticas internas son superiores.

2. **Las heurísticas por defecto varían drásticamente:** Gecode produce soluciones triviales (100 % casos) **agotando el timeout en todas las instancias**, Chuffed tiene 27 % casos críticos **también con timeout universal**, HiGHS 0 % casos problemáticos **sin alcanzar timeout**. La elección del solver es tan importante como la estrategia de búsqueda.

3. **Anotaciones explícitas necesarias en Gecode:** `dom_w_deg` mejora Gecode 55 % (makespan 134.6 → 61.1), pero aún queda 8 % por debajo de HiGHS sin anotaciones. **A pesar de alcanzar timeout en 10/11 instancias,** las heurísticas adaptativas encuentran mejores soluciones intermedias, aunque no compensan totalmente deficiencias del solver base.

4. **Decidir operarios primero es contraproducente:** Estrategia 3 produce resultados idénticos a Estrategia 1 pero con 42 % más nodos antes de agotar tiempo. El orden temporal debe decidirse antes que asignaciones de recursos.

5. **Limitación de portabilidad:** `seq_search` solo funciona en Gecode, imposibilitando aplicar Estrategia 2 a HiGHS para verificar si mejora aún más sus ya excelentes resultados.

**Recomendación:** Usar HiGHS sin anotaciones para este tipo de problemas. Si se requiere Gecode, aplicar Estrategia 2 (`dom_w_deg`).

# 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.