



Modelamiento e Implementación de COPs

Taller 2

Juan Sebastián Estupiñán Cifuentes (201924846)

Juan David Soto Carmona (201958813)

Programación por Restricciones

Juan Francisco Díaz, Ph.D

Escuela de Ingeniería de Sistemas y Computación

Facultad de Ingeniería

Universidad del Valle

30 de abril de 2024

Tabla de contenidos

1. Movimiento de material.....	3
1.1. Modelo.....	3
1.1.1. Parámetros.....	3
1.1.2. Variables.....	3
1.1.3. Restricciones.....	3
1.1.4. Función objetivo.....	4
1.1.5. Anotaciones de búsqueda.....	5
1.2. Árboles de búsqueda.....	5
1.3. Pruebas.....	6
1.3.1. Prueba 1.....	6
1.3.2. Prueba 2.....	6
1.3.3. Prueba 3.....	7
1.4. Conclusiones.....	7
2. Subasta del mercado eléctrico.....	8
2.1. Modelo.....	8
2.1.1. Parámetros.....	8
2.1.2. Variables.....	8
2.1.3. Restricciones.....	8
2.1.4. Función objetivo.....	10
2.1.5. Anotaciones de búsqueda.....	10
2.2. Árboles de búsqueda.....	11
Con condiciones redundantes:.....	12
2.3. Pruebas.....	14
2.3.1. Prueba 1.....	14
2.3.2. Prueba 2.....	14
2.3.3. Prueba 3.....	15
2.4. Conclusiones.....	16

1. Movimiento de material

1.1. Modelo

1.1.1. Parámetros

n : indica el número de bodegas.

$$n \in \mathbb{Z}^+$$

M : unidades de material distribuidas en las n bodegas.

$$M \in \mathbb{Z}^+$$

m : arreglo que representa cómo están distribuidas las M unidades de material en las n bodegas.

$$m_i: \forall i \in \{1, \dots, n\}, m_i \in \{0, \dots, M\}$$

o : arreglo que representa cómo deberían quedar distribuidas las M unidades de material en las n bodegas.

$$o_i: \forall i \in \{1, \dots, n\}, o_i \in \{0, \dots, M\}$$

a : número real mayor o igual a uno, se utiliza en la fórmula para calcular costos.

$$a \in \mathbb{R}^+ \wedge a \geq 1$$

b : número real mayor o igual a uno, se utiliza en la fórmula para calcular costos.

$$b \in \mathbb{R}^+ \wedge b \geq 1$$

*Nota: intentamos representar a y b como tipo *float* pero encontramos conflictos de tipos al aplicar la fórmula de costo y no hallamos la forma de solucionarlos. Por este motivo los dejamos representados como tipo *int*.*

1.1.2. Variables

mov : es una matriz que representa la cantidad de material a mover de la bodega i a la bodega j con el objetivo de que el material quede distribuido como se indica en el arreglo o .

$$mov_{ij}: \forall i \in \{1, \dots, n\} \wedge \forall j \in \{1, \dots, n\}, mov_{ij} \in \{0, \dots, M\}.$$

1.1.3. Restricciones

- **Restricciones sobre los parámetros:**

$$a \geq 1 \wedge b \geq 1 \wedge n \geq 2 \wedge M \geq 0$$

$$\sum_{i=1}^n m_i = M$$

$$\sum_{i=1}^n o_i = M$$

- **No se puede mover material de una bodega a sí misma:**

$$\forall i \in \{1, \dots, n\}, \text{mov}_{i,i} = 0$$

- **Máximo de unidades a sacar de una bodega:** se calculan las unidades de material que hay en una bodega para establecerlo como número máximo que se puede extraer de la misma.

$$\forall i, j \in \{1, \dots, n\}, \text{mov}_{i,j} \leq m_i + \sum_{x=1}^n (\text{mov}_{x,i} - \text{mov}_{i,x})$$

- **Configuración ideal:** asegura que cada bodega contenga exactamente los elementos que se indican en la distribución final o

$$\forall i \in \{1, \dots, n\}, (m_i + \sum_{x=1}^n (\text{mov}_{x,i} - \text{mov}_{i,x})) = o_i$$

- **Restricción para romper simetrías:** la siguiente restricción evita que una vez se pase cualquier cantidad de material de la bodega i a la bodega j , no se puedan hacer movimientos de la bodega j a la bodega i . No tiene sentido devolver el material (Ver **Figuras 3 y 4**).

$$\forall i, j \in \{1, \dots, n\}, \text{mov}_{i,j} > 0 \rightarrow \text{mov}_{j,i} = 0$$

1.1.4. Función objetivo

Minimizar el costo: minimizar el costo total de mover cada unidad de material entre bodegas. Se calcula mediante la siguiente fórmula:

$$\text{minimize } \sum_{i=1}^n \left(\sum_{j=1}^n (\text{mov}_{i,j})^a \star |j - i|^b \right)$$

1.1.5. Anotaciones de búsqueda

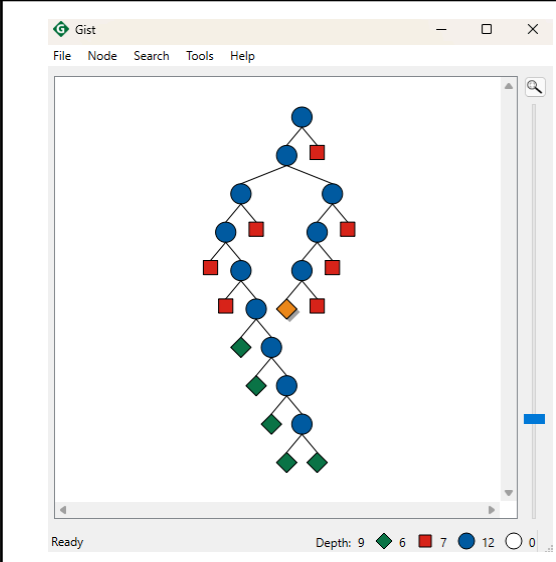
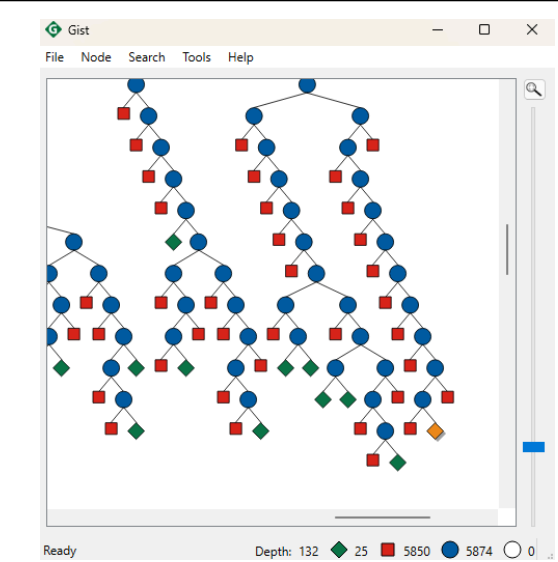
solve :: int_search(mov, dom_w_deg, indomain_min) minimize costo;

Asigna primero los valores más pequeños del dominio resultando en la mejor forma de buscar teniendo en cuenta que queremos obtener la solución con menor costo (Ver **Figura 1**).

solve :: int_search(mov, dom_w_deg, indomain_max) minimize costo;

Al usar *indomain_max*, asignamos primero los valores más grandes del dominio, es decir que vamos a empezar a buscar soluciones con el mayor costo por lo que el árbol de búsqueda se expandirá exponencialmente antes de encontrar la solución con costo mínimo (Ver **Figura 2**).

1.2. Árboles de búsqueda

<i>indomain_min</i>	<i>indomain_max</i>
	
Figura 1: Árbol de búsqueda de la prueba 1 - movimiento de material.	Figura 2: Árbol de búsqueda de la prueba 1 - movimiento de material.

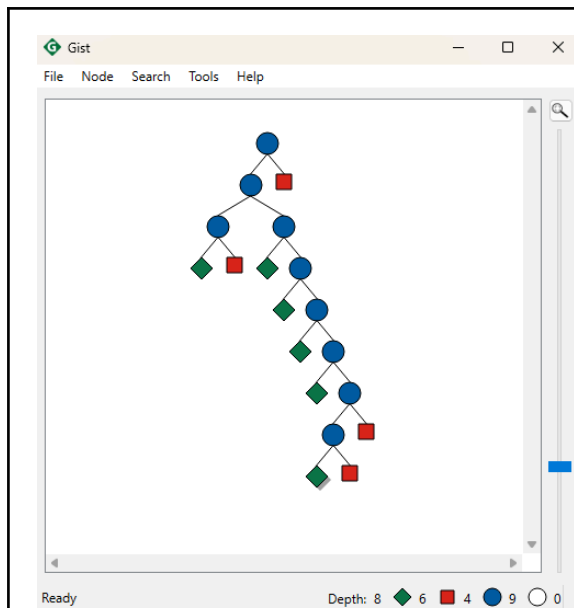


Figura 3: Árbol de soluciones de la prueba 2 - movimiento de material. Rompiendo simetrías

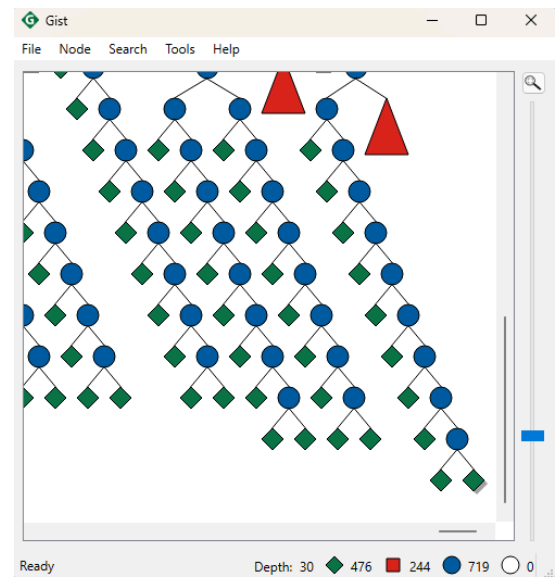


Figura 4: Árbol de soluciones de la prueba 2 - movimiento de material. Sin romper simetrías

1.3. Pruebas

1.3.1. Prueba 1

Entrada:

```
n=4;
M=31;
m=[10,11,5,5];
o=[10,5,11,5];
a=2;
b=2;
```

Salida:

```
mov[1,3] = 1
mov[2,1] = 1
mov[2,3] = 4
mov[2,4] = 1
mov[4,3] = 1

Costo = 26
```

1.3.2. Prueba 2

Entrada:

```
n=3;
M=25;
m=[10,10,5];
o=[5,10,10];
a=2;
b=2;
```

Salida:

```
mov[1,2] = 3
mov[1,3] = 2
mov[2,3] = 3

Costo = 34
```

1.3.3. Prueba 3

Entrada:	Salida:
<pre>n=6; M=40; m=[10,8,2,7,12,1]; o=[6,9,3,7,5,10]; a=2; b=1;</pre>	<pre>mov[1,2] = 1 mov[1,3] = 1 mov[1,4] = 1 mov[1,6] = 1 mov[2,6] = 1 mov[3,6] = 1 mov[4,6] = 2 mov[5,2] = 1 mov[5,3] = 1 mov[5,4] = 1 mov[5,6] = 4 Costo = 48</pre>

1.4. Conclusiones

En este problema de optimización se pueden observar claramente:

- La importancia de la estrategia de búsqueda; elegir la mejor estrategia puede hacer que el programa reduzca su tiempo de ejecución considerablemente.
- Establecer restricciones para romper simetrías también ayuda a podar el árbol de soluciones en gran medida. En este caso, la restricción prohíbe devolver material de una bodega a otra después de ser transferido reduciendo el espacio de búsqueda y mejorando la eficiencia del programa.

2. Subasta del mercado eléctrico

2.1. Modelo

2.1.1. Parámetros

N : cantidad de generadores en el mercado eléctrico.

M : número de unidades de tiempo en que está dividido el día.

D : arreglo que representa la demanda de energía en KW de los consumidores en cada hora j .

$D_j: j \in \{0, \dots, M - 1\}$

P : precio máximo de compra por KW.

C : matriz que representa la capacidad de producción de energía en KW de cada generador i en cada hora j .

$C_{j,i}: j \in \{0, \dots, M - 1\}, i \in \{1, \dots, N\}$

PV : arreglo que representa el precio de venta de energía por KW de cada generador i .

$PV_i: i \in \{1, \dots, N\}$

2.1.2. Variables

PC : precio de compra de la energía.

$PC \in \mathbb{Z}^+$

$PV_{\text{Seleccionado}}$: arreglo que representa el precio de venta de energía por KW de cada generador i seleccionado.

$PV_{\text{Seleccionado}_i}: \forall i \in \{1, \dots, N\}, PV_{\text{Seleccionado}_i} \in \mathbb{Z}^+$

V : matriz que representa la cantidad de KW que debe comprarse a cada generador i en cada hora j .

$V_{j,i}: \forall j \in \{0, \dots, M - 1\}, \forall i \in \{1, \dots, N\}, V_{j,i} \in \mathbb{Z}^+$

2.1.3. Restricciones

- **No negatividad:**

$$N \geq 0, M \geq 1, P \geq 0, PC \geq 0$$

$$\forall j \in \{0, \dots, M - 1\}, D_j \geq 0$$

$$\forall j \in \{0, \dots, M - 1\}, \forall i \in \{1, \dots, N\}, C_{j,i} \geq 0$$

$$\forall i \in \{1, \dots, N\}, PV_i \geq 0$$

$$\forall i \in \{1, \dots, N\}, PV_{\text{Seleccionado}_i} \geq 0$$

$$\forall j \in \{0, \dots, M - 1\}, \forall i \in \{1, \dots, N\}, V_{j,i} \geq 0$$

- **Se satisface toda la demanda en cada unidad de tiempo:**

$$\forall j \in \{0, \dots, M - 1\}, \sum_{i=1}^N V_{j,i} = D_j$$

- **El precio al que se compra la energía a cada generador no supera el precio máximo de compra establecido:**

$$\forall i \in \{1, \dots, N\}, PV_{\text{Seleccionado}_i} \leq P$$

- **La compra de energía a cada generador no supera su capacidad:**

$$\forall j \in \{0, \dots, M - 1\}, \forall i \in \{1, \dots, N\}, V_{j,i} \leq C_{j,i}$$

- **El precio de compra es el mayor PV_i ofertado por los generadores i .**

$$\text{si } \forall i \in \{1, \dots, N\}, \sum_{j=0}^{M-1} V_{j,i} > 0 \quad \text{entonces } PV_{\text{Seleccionado}_i} = PV_i$$

$$\text{si no } PV_{\text{Seleccionado}_i} = 0$$

$$\forall i \in \{1, \dots, N\}, PV_{\text{Seleccionado}_i} \leq PC$$

- **Restricciones redundantes:**

Una restricción redundante podría ser $PC = \max(PV_{\text{Seleccionado}_i})$. Sería redundante porque la restricción $\forall i \in \{1, \dots, N\}, PV_{\text{Seleccionado}_i} \leq PC$ indica que PC debe ser mayor o igual que todos los precios de venta de los generadores seleccionados y, al buscar minimizar PC , el solucionador buscará el mínimo valor para PC , así que es evidente que no seleccionará un valor mayor al máximo de los precios de venta seleccionados porque no sería

el mínimo valor posible, por lo que PC debe ser entonces el máximo de esos precios.

Por lo tanto,

$$PC = \max(PV_{\text{Seleccionado}_i}) \equiv \forall i \in \{1, \dots, N\}, PV_{\text{Seleccionado}_i} \leq PC, \text{ minimize } PC$$

las dos restricciones son equivalentes cuando se está minimizando PC.

2.1.4. Función objetivo

Minimizar el precio de compra: minimizar el mayor precio de venta entre los precios de venta de los generadores seleccionados.

$$\text{minimize } PC$$

2.1.5. Anotaciones de búsqueda

Usando la función *int_search()* se usará el parámetro *indomain_min* para definir el orden de asignación de los valores, ya que este parámetro asigna a las variables su valor de dominio más pequeño, lo que en este caso es adecuado porque se busca minimizar.

Para el parámetro que define la estrategia de búsqueda se presentan dos opciones *anti_first_fail* y *first_fail*:

solve :: int_search(V, anti_first_fail, indomain_min) minimize PC;

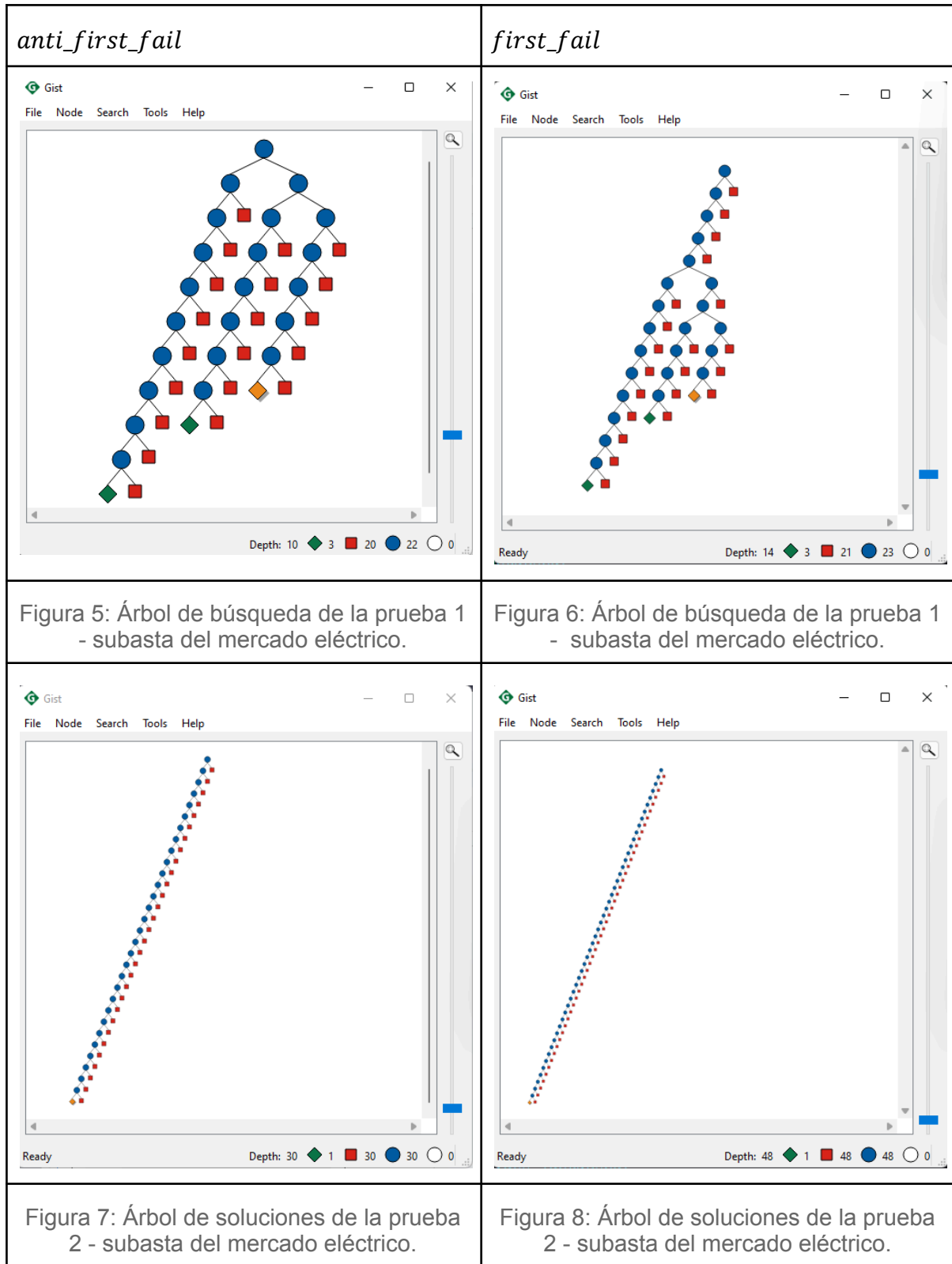
Esta configuración selecciona la variable que restringe la menor cantidad de valores en su dominio. Puede ser útil para buscar soluciones de manera más uniforme.

(Ver **Figura 5**).

solve :: int_search(V, first_fail, indomain_min) minimize PC;

Al usar esta configuración asignamos primero valores a las variables con menor tamaño de dominio, es decir, las variables más restringidas, lo que ayuda a reducir el árbol de búsqueda en anchura pero aumenta en profundidad (Ver **Figura 6**).

2.2. Árboles de búsqueda



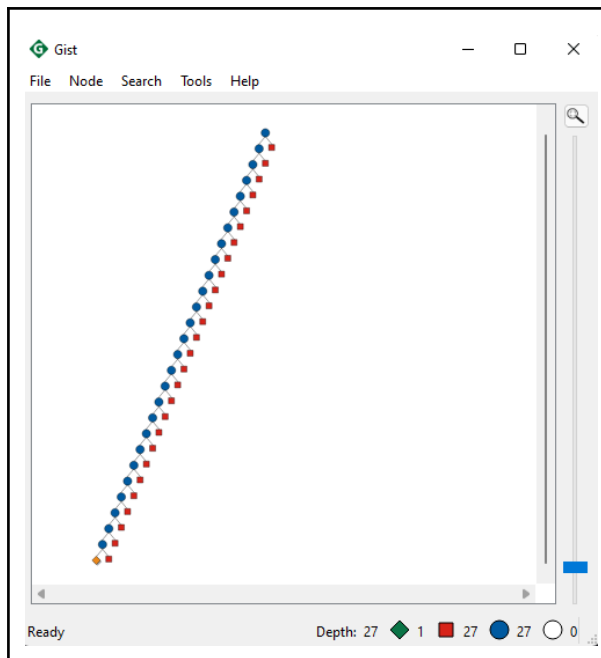


Figura 9: Árbol de soluciones de la prueba 3 - subasta del mercado eléctrico.

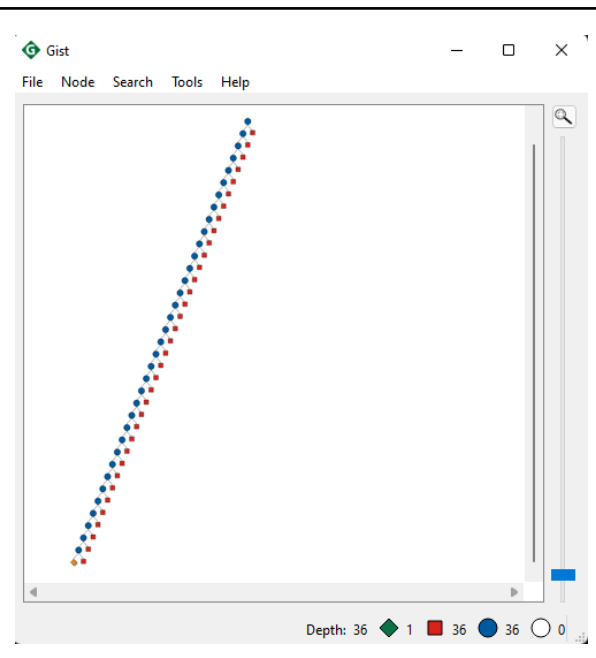


Figura 10: Árbol de soluciones de la prueba 3 - subasta del mercado eléctrico.

Con condiciones redundantes:

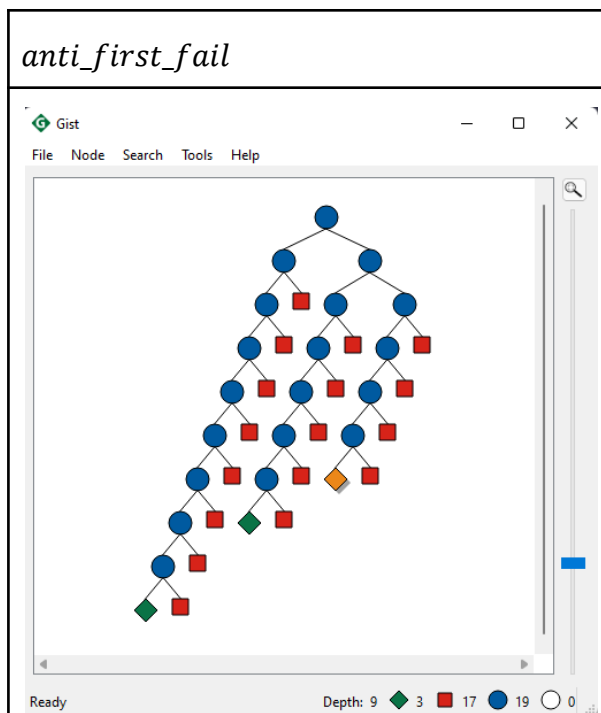


Figura 11: Árbol de búsqueda de la prueba 1 - subasta del mercado eléctrico.

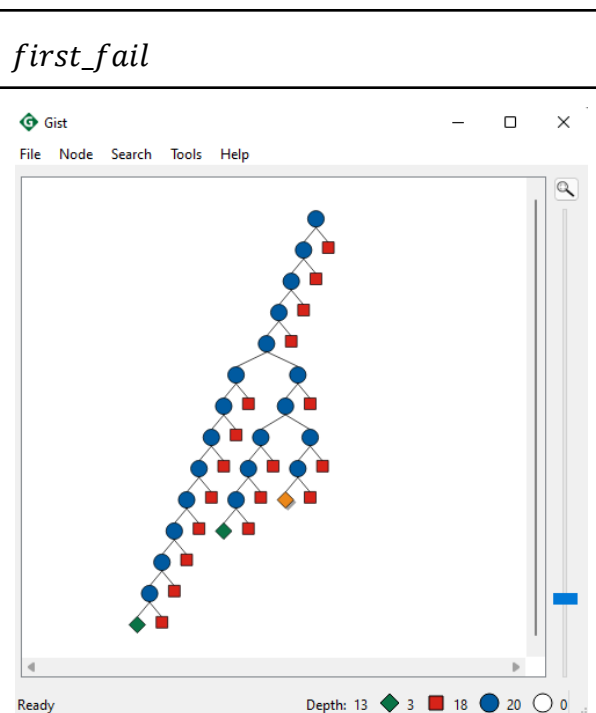


Figura 12: Árbol de búsqueda de la prueba 1 - subasta del mercado eléctrico.

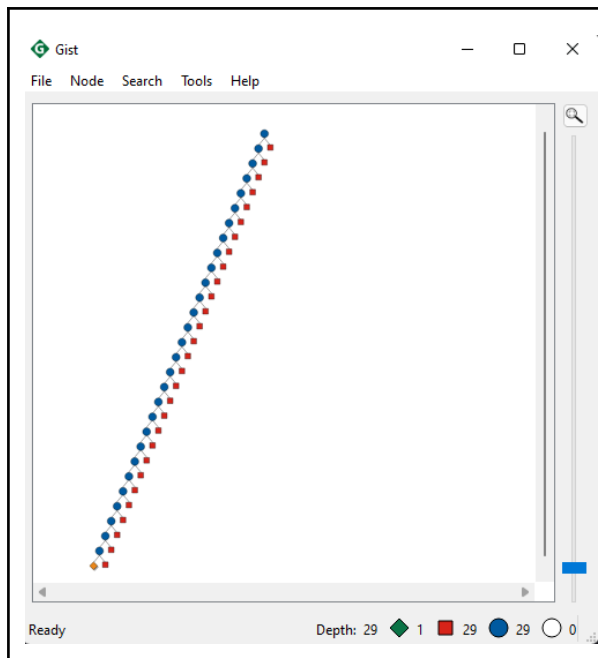


Figura 13: Árbol de soluciones de la prueba 2 - subasta del mercado eléctrico.

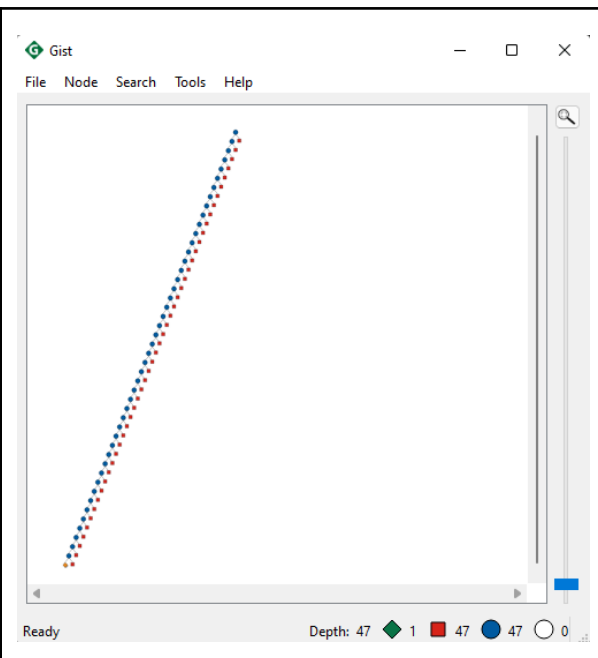


Figura 14: Árbol de soluciones de la prueba 2 - subasta del mercado eléctrico.

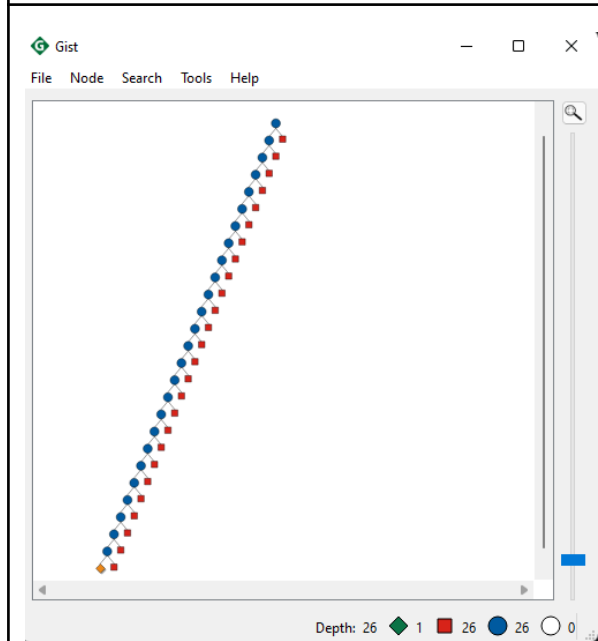


Figura 15: Árbol de soluciones de la prueba 3 - subasta del mercado eléctrico.

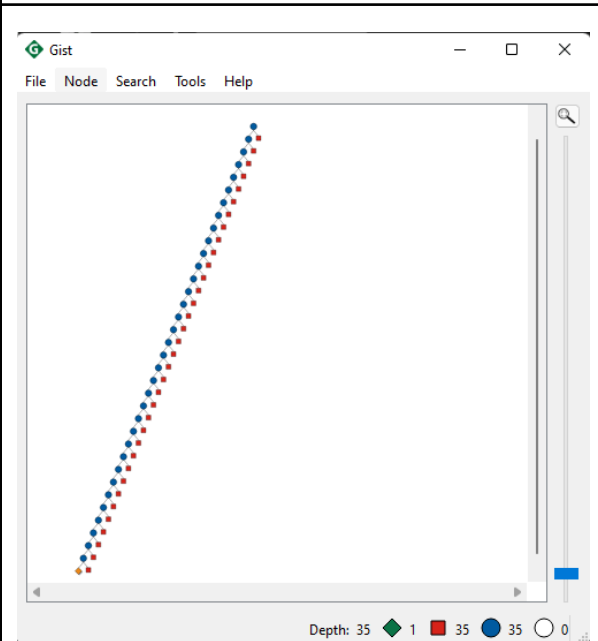


Figura 16: Árbol de soluciones de la prueba 3 - subasta del mercado eléctrico.

2.3. Pruebas

2.3.1. Prueba 1

Entrada:	Salida:
<pre>N = 5; M = 4; D = array1d(0..M-1, [100, 40, 20, 200]); P = 35000; C = array2d(0..M-1, 1..N, [20,50,40,15,40, 20,20,20,20,20, 10,30,80,10,40, 100,10,40,60,90]); PV = [1000,35000,200,1200,500];</pre>	<pre>V = Hora Gen 1 Gen 2 Gen 3 Gen 4 Gen 5 0 [20, 0, 40, 0, 40] 1 [0, 0, 20, 0, 20] 2 [0, 0, 0, 0, 20] 3 [70, 0, 40, 0, 90] Precio de compra = 1000</pre>

2.3.2. Prueba 2

Entrada:	Salida:
<pre>N = 4; M = 24; D = array1d(0..M-1, [100, 40, 20, 200, 500, 400, 500, 800, 1000, 400, 500, 400, 200, 250, 500, 600, 700, 20, 95, 150, 350, 340, 500, 10]); P = 8000; C = array2d(0..M-1, 1..N, [10,50,140,260, 40,20,10,30, 5,15,20,20, 30,5,20,200, 50,100,40,600, 100,150,20,400,</pre>	<pre>V = Hora Gen 1 Gen 2 Gen 3 Gen 4 0 [0, 0, 0, 100] 1 [40, 0, 0, 0] 2 [0, 0, 0, 20] 3 [0, 0, 0, 200] 4 [0, 0, 0, 500] 5 [0, 0, 0, 400] 6 [100, 100, 0, 300] 7 [600, 200, 0, 0] 8 [200, 450, 0, 350] 9 [250, 10, 0, 140] 10 [0, 0, 0, 500] 11 [200, 200, 0, 0] 12 [200, 0, 0, 0] 13 [250, 0, 0, 0] 14 [500, 0, 0, 0] 15 [0, 0, 0, 600]</pre>

<pre> 100,100,100,400, 600,250,44,240, 200,450,550,650, 250,10,300,300, 120,36,500,500, 200,350,250,20, 400,100,410,201, 400,55,200,20, 500,150,20,20, 100,100,800,2000, 100,100,800,2000, 100,10,20,20, 405,45,200,2, 150,15,20,20, 500,450,200,200, 220,45,200,200, 770,48,280,210, 10,10,10,10)); PV = [8000,7000,9000,7500]; </pre>	<pre> 16 [0, 0, 0, 700] 17 [0, 0, 0, 20] 18 [95, 0, 0, 0] 19 [150, 0, 0, 0] 20 [350, 0, 0, 0] 21 [220, 45, 0, 75] 22 [500, 0, 0, 0] 23 [0, 0, 0, 10] Precio de compra = 8000 </pre>
--	---

2.3.3. Prueba 3

<p>Entrada:</p> <pre> N = 4; M = 12; D = array1d(0..M-1, [40, 20, 530, 34, 800, 55, 40, 200, 500, 620, 95, 150]); P = 10000; C = array2d(0..M-1, 1..N, [40,20,10,30, 30,5,20,200, 50,100,40,600, 100,150,20,400, 400,400,400,400, 42,100,410,201, 30,55,200,20, 500,150,20,20, 555,100,800,300, 100,100,800,2000, 50,48,280,210, 12,400,33,44]); </pre>	<p>Salida:</p> <pre> V = Hora Gen 1 Gen 2 Gen 3 Gen 4 0 [0, 0, 10, 30] 1 [0, 0, 20, 0] 2 [0, 0, 40, 490] 3 [0, 14, 20, 0] 4 [400, 0, 400, 0] 5 [0, 0, 55, 0] 6 [0, 0, 40, 0] 7 [180, 0, 20, 0] 8 [0, 0, 500, 0] 9 [0, 0, 620, 0] 10 [0, 0, 95, 0] 11 [0, 117, 33, 0] Precio de compra = 10000 </pre>
--	--

PV = [5500,4500,8700,10000];	
------------------------------	--

2.4. Conclusiones

- Encontrar la mejor configuración para la anotación de búsqueda es clave para reducir el espacio de búsqueda y encontrar soluciones en menor tiempo. En problemas de minimización, comenzar con valores pequeños puede ayudar a encontrar soluciones óptimas más rápidamente, ya que tienden a conducir a soluciones de menor costo.
- La inclusión de condiciones redundantes permite disminuir el espacio de búsqueda, al añadir restricciones adicionales que ayudan al solucionador a encontrar valores correctos más rápido.