Programación con Restricciones

Resumen

Daniel San Martín

Departamento de Informática
Universidad Técnica Federico Santa María

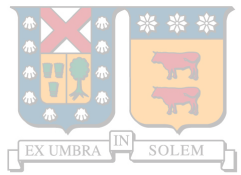
15 de noviembre de 2017

Ejercicio 1

Enunciado

¿Qué valores entre 0 y 9 todos distintos satisfacen la ecuación?

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline & M & O & N & E & Y \end{array} \quad (1)$$

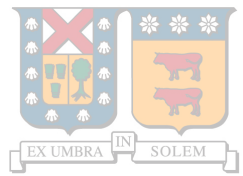


Ejercicio 1

El ejercicio define para $S, E, N, D, M, O, R, Y \in [0, ..., 9]$, $S, M \neq 0$ y se puede restringir con dos modelos.

- Utilizando las unidades, decenas, centenas, etc.:

$$\begin{array}{rrrrrr} & & 1000 \cdot S & 100 \cdot E & 10 \cdot N & 1 \cdot D \\ + & & 1000 \cdot M & 100 \cdot O & 10 \cdot R & 1 \cdot E \\ \hline & 10000 \cdot M & 1000 \cdot O & 100 \cdot N & 10 \cdot E & 1 \cdot Y \end{array} \quad (2)$$

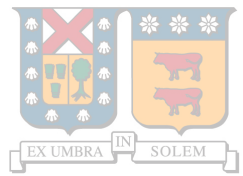


Ejercicio 1

- Utilizando el acarreo de la suma:

$$\begin{aligned} D + E &= Y + 10 \cdot R_0 \\ R_0 + N + R &= E + 10 \cdot R_1 \\ R_1 + E + O &= N + 10 \cdot R_2 \\ R_2 + S + M &= O + 10 \cdot R_3 \\ R_3 &= M \end{aligned} \tag{3}$$

con $R_i \in [0, 1]$ para $i = 0, ..., 3$.



Ejercicio 1: Desarrollo

Para analizar el ejercicio se disponen las siguientes opciones:

- Utilizar la ecuación 2 o 3.
- Utilizar la restricción de desigualdad para cada variable (R1) o utilizar la restricción *alldifferent* (R2).
- Enumerar primero a la variable de menor (E1) o mayor dominio (E2).
- Asignar el menor (A1) o mayor (A2) valor a la primera variable seleccionada.

Ejercicio 1: Resultados

		R1-E1-A1	R1-E1-A2	R1-E2-A1	R1-E2-A2
Initial	Propagators:	57	57	57	57
	Branchers:	1	1	1	1
Summary	Runtime:	0.299 ms	0.27 ms	0.413 ms	0.274 ms
	Solutions:	1	1	1	1
	Propagations:	103	100	545	81
	Nodes:	7	7	67	7
	Failures:	3	3	33	3
	Restarts:	0	0	0	0
	No-goods:	0	0	0	0
	Peak depth:	1	1	9	2

Cuadro 1: Estadísticas utilizando ecuación 2 (parte 1)

Ejercicio 1: Resultados

		R2-E1-A1	R2-E1-A2	R2-E2-A1	R2-E2-A2
Initial	Propagators:	2	2	2	3
	Branchers:	1	1	1	1
Summary	Runtime:	0.237 ms	0.256 ms	0.307 ms	0.286 s
	Solutions:	1	1	1	1
	Propagations:	14	14	100	17
	Nodes:	7	7	67	7
	Failures:	3	3	33	3
	Restarts:	0	0	0	0
	No-goods:	0	0	0	0
	Peak depth:	1	1	9	2

Cuadro 2: Estadísticas utilizando ecuación 2 (parte 2)

Ejercicio 1: Resultados

		R1-E1-A1	R1-E1-A2	R1-E2-A1	R1-E2-A2
Initial	Propagators:	61	61	61	61
	Branchers:	1	1	1	1
Summary	Runtime:	0.313 ms	0.323 ms	0.393 ms	0.388 ms
	Solutions:	1	1	1	1
	Propagations:	175	167	429	424
	Nodes:	11	11	35	43
	Failures:	5	5	17	21
	Restarts:	0	0	0	0
	No-goods:	0	0	0	0
	Peak depth:	1	1	5	5

Cuadro 3: Estadísticas utilizando ecuación 3 (parte 1)

Ejercicio 1: Resultados

		R2-E1-A1	R2-E1-A2	R2-E2-A1	R2-E2-A2
Initial	Propagators:	6	6	6	6
	Branchers:	1	1	1	1
Summary	Runtime:	0.304 ms	0.298 ms	0.327 ms	0.286 s
	Solutions:	1	1	1	1
	Propagations:	68	70	164	170
	Nodes:	11	11	35	43
	Failures:	5	5	17	21
	Restarts:	0	0	0	0
	No-goods:	0	0	0	0
	Peak depth:	1	1	5	5

Cuadro 4: Estadísticas utilizando ecuación 3 (parte 2)

Ejercicio 2

Enunciado

¿Cuál es el máximo número de reinas que pueden ser puestas en un tablero de ajedrez de $n \times n$ sin que se ataquen?

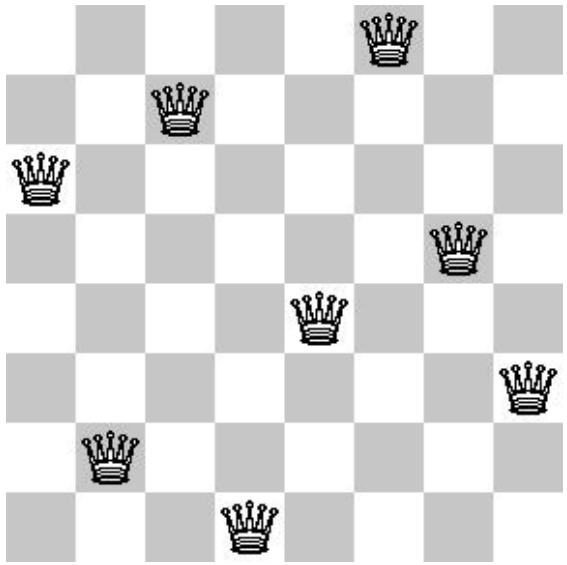
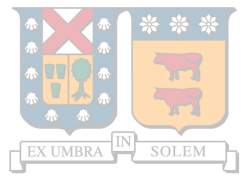


Figura 1: Problema n-reinas

Ejercicio 2: Desarrollo

Las restricciones utilizadas en este problema son las siguientes. Sea $q_i, q_j,\ i, j = 1, .., n$ la posición de dos reinas en el tablero de $n \times n$, para que estas no se ataquen se debe cumplir:

1. $q_i \neq q_j$
2. $q_i - i \neq q_j - j$
3. $q_i + i \neq q_j + j$



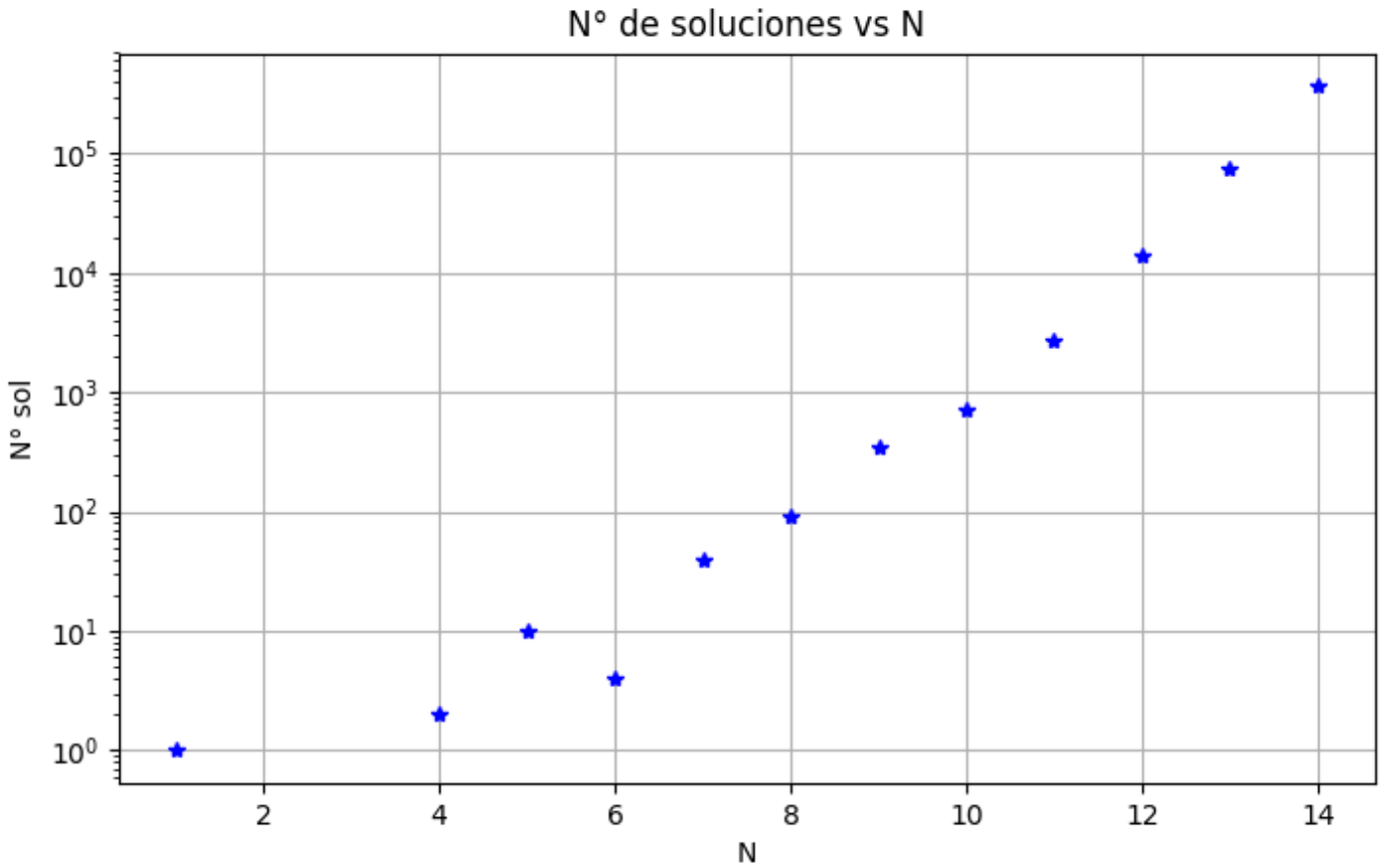
Ejercicio 2: Desarrollo

- Estrategia 1:
  - Enumerar primero la variable con menor dominio.
  - Asignar el menor valor a la primera variable seleccionada.
- Estrategia 2:
  - Enumerar primero la variable con menor dominio.
  - Asignar el mayor valor a la primera variable seleccionada.

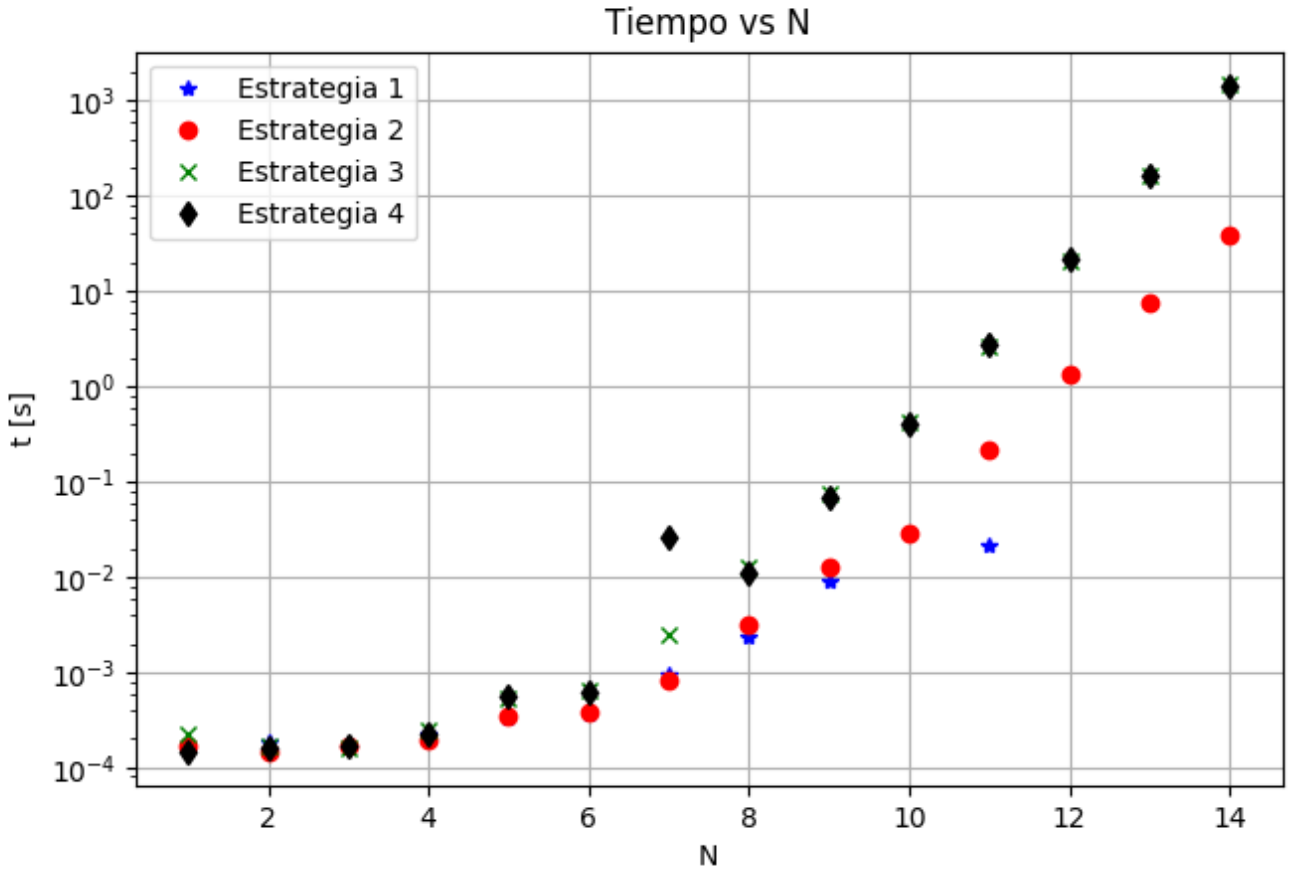
Ejercicio 2: Desarrollo

- Estrategia 3:
  - Enumerar primero la variable con mayor dominio.
  - Asignar el menor valor a la primera variable seleccionada.
- Estrategia 4:
  - Enumerar primero a la variable con mayor dominio.
  - Asignar el mayor valor a la primera variable seleccionada.

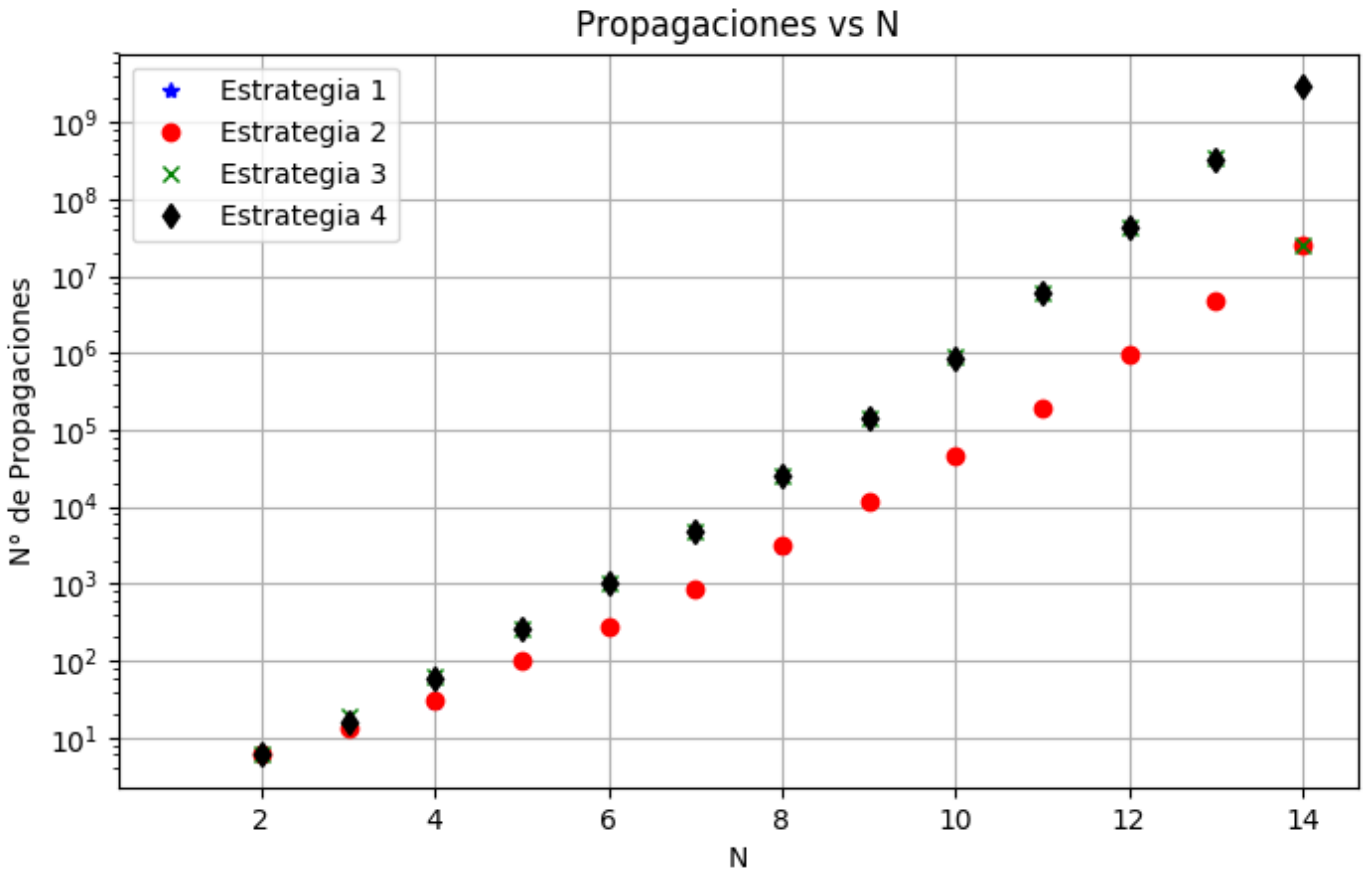
Ejercicio 2: Desarrollo



Ejercicio 2: Desarrollo



Ejercicio 2: Desarrollo



El problema Open Shop

Un conjunto finito de operaciones tiene que ser procesado en un conjunto dado de máquinas. Cada operación tiene un tiempo de procesamiento específico durante el cual no se puede interrumpir. Las operaciones se agrupan en trabajos, de modo que cada operación pertenece exactamente a un trabajo. Además, cada operación requiere exactamente una máquina para su procesamiento.

El problema Opne Shop

Objetivo

El objetivo del problema es programar todas las operaciones, es decir, determinar su hora de inicio, para minimizar el tiempo máximo de finalización (*makespan*).

El problema Open Shop

Restricciones

Las operaciones que pertenecen al mismo trabajo y las operaciones que utilizan la misma máquina no se pueden procesar simultáneamente.

Ejemplo

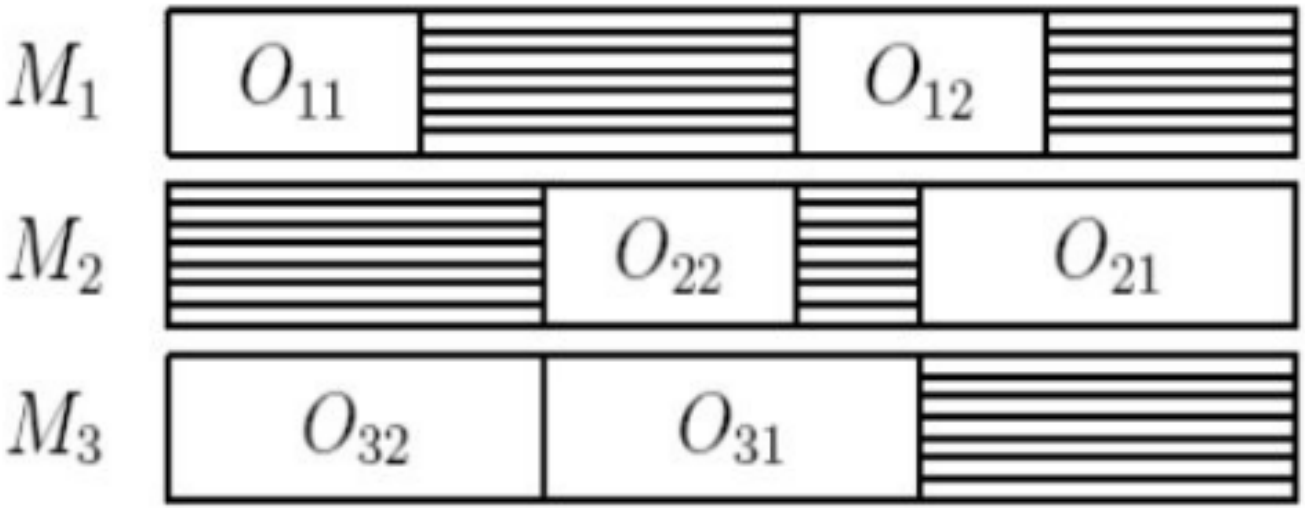


Figura 5: Ejemplo

Desarrollo

Para probar el código de *Gecode* se utilizaron 3 ejemplos con 3 máquinas y 3 tareas. Además, se realizaron 4 estrategias de *branching*.

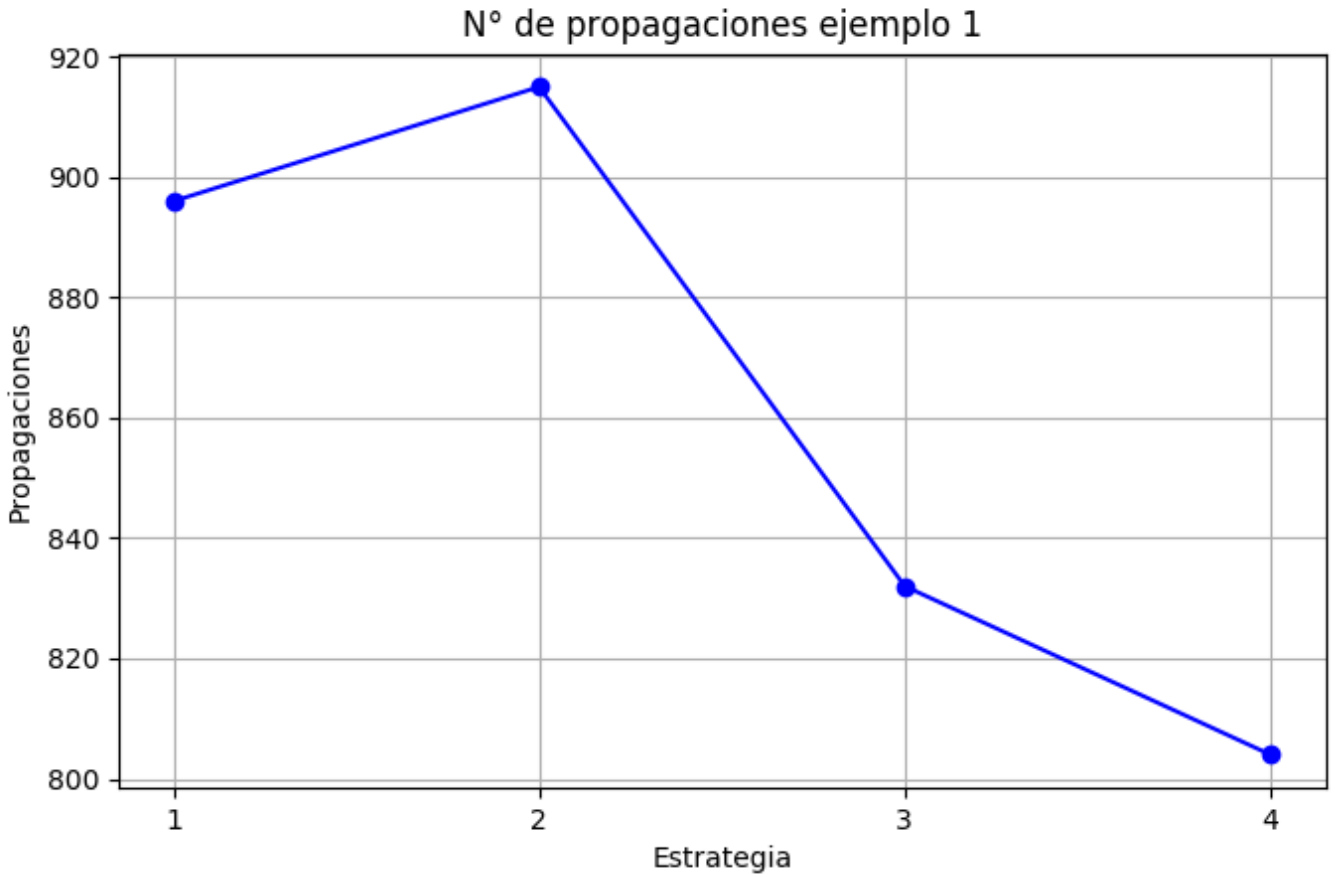
1. Selección de variable: Mayor recuento de fallos acumulados. Selección de mayor valor.
2. Selección de variable: Mayor recuento de fallos acumulados. Selección de menor valor.
3. Selección de variable: Menor recuento de fallos acumulados. Selección de mayor valor.
4. Selección de variable: Menor recuento de fallos acumulados. Selección de menor valor.

Ejemplo 1

	Job 0	**Job 1**	**Job 2**
Task 0	661	6	333
Task 1	168	489	343
Task 2	171	505	324

Cuadro 5: Ejemplo 1

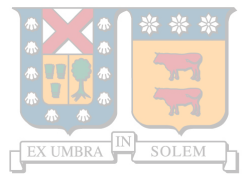
Resultados ejemplo 1



Ejemplo 2

	Job 0	**Job 1**	**Job 2**
Task 0	1000	603	123
Task 1	674	349	223
Task 2	271	467	874

Cuadro 6: Ejemplo 2

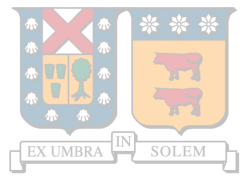


Resultados ejemplo 2

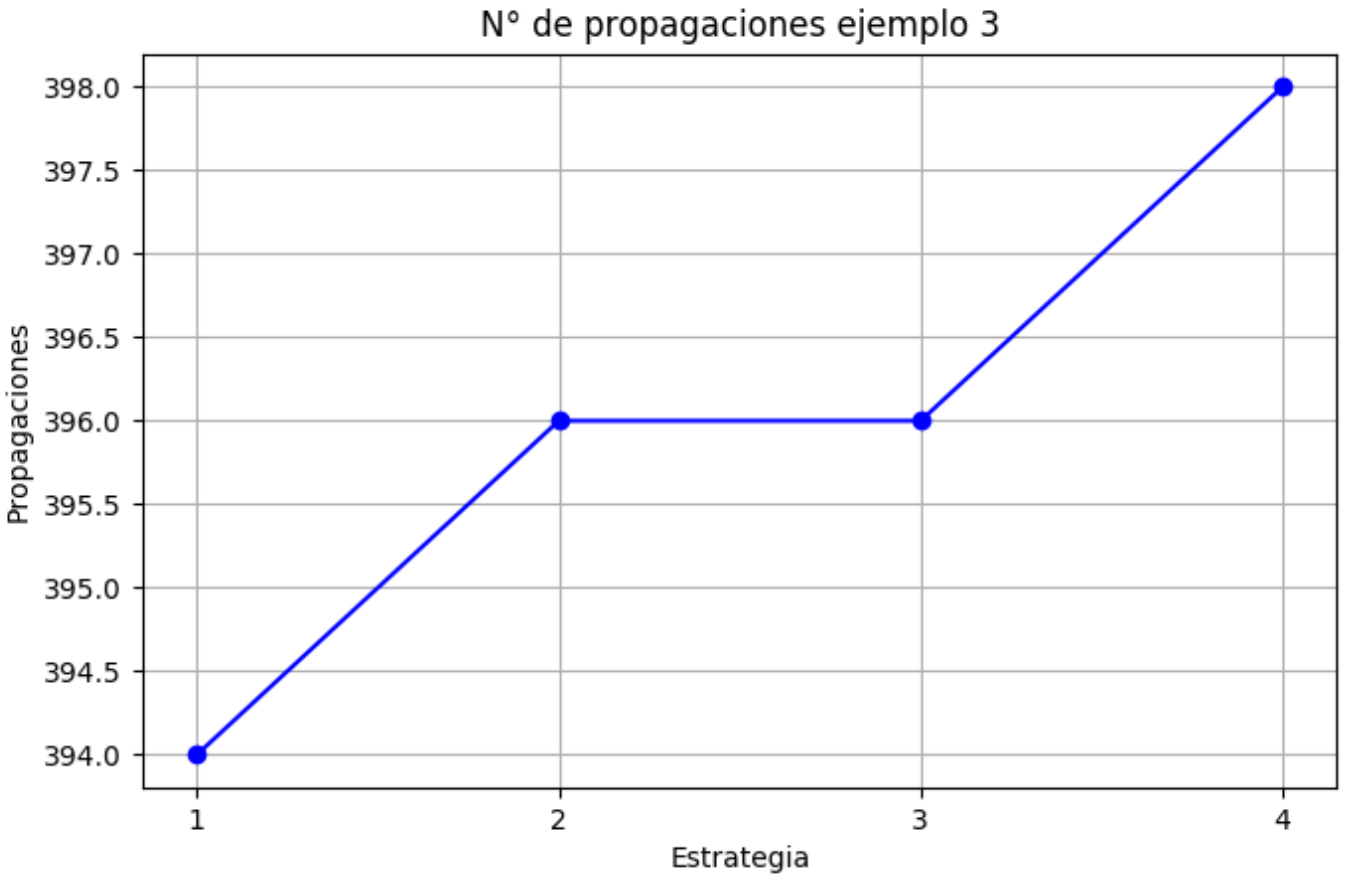
Ejemplo 3

	Job 0	**Job 1**	**Job 2**
Task 0	61	6	33
Task 1	18	89	243
Task 2	201	105	134

Cuadro 7: Ejemplo 3



Resultados ejemplo 3



El problema Set Covering

Dado un conjunto de elementos $\{1, 2, ..., m\}$ (llamado universo) y n conjuntos cuya unión comprende el universo, el *set cover problem* consiste en identificar el menor número de conjuntos cuya unión aun contiene todos los elementos del universo.

El problema Set Covering

Definición Formal

Sea el universo $\mathcal{U}$ y la familia $\mathcal{S}$ de subconjuntos de $\mathcal{U}$, una cobertura es una subfamilia $\mathcal{C} \subseteq \mathcal{S}$ de conjuntos cuya unión es $\mathcal{U}$.

El problema Set Covering

Este problema puede ser formulado como un problema de programación lineal entera.

$$\text{Minimizar } Z = \sum_{S \in \mathcal{S}} x_S$$

$$\text{sujeto a } \sum_{S:\, e \in S} x_S \leq 1,\ \forall e \in \mathcal{U}$$

$$x_S \in \{0, 1\},\ \forall S \in \mathcal{S}$$

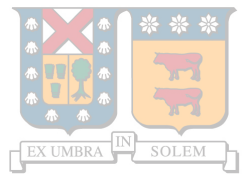
Ejemplo: Ubicación de estaciones de bomberos

Hay seis ciudades en el condado de Kilroy. El condado debe determinar dónde construir las estaciones de bomberos. El condado quiere construir el número mínimo de estaciones de bomberos necesarias para asegurar que al menos una estación de bomberos se encuentre dentro de los 15 minutos (tiempo de conducción) de cada ciudad (Winston y Goldberg, 2004).

Ejemplo: Ubicación de estaciones de bomberos

	Hasta					
Desde	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5	Ciudad 6
Ciudad 1	0	10	20	30	30	20
Ciudad 2	10	0	25	35	20	10
Ciudad 3	20	25	0	15	30	20
Ciudad 4	30	35	15	0	15	25
Ciudad 5	30	20	30	15	0	14
Ciudad 6	20	10	20	25	14	0

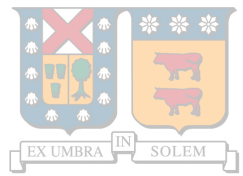
Cuadro 8: Tiempos entre ciudades



Ejemplo: Solución

Para cada ciudad donde se debe construir una estación se definen variables binarias $x_i,\ i = 1, ..., 6$, de forma que:

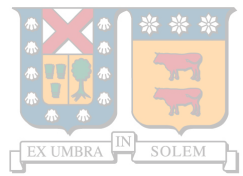
$$x_i = \begin{cases} 1 & \text{si hay que construir una estación en la ciudad } i \\ 0 & \text{en otro caso} \end{cases}$$



Ejemplo: Solución

De esta forma el número total de estaciones que se deben construir queda definido por $\sum_{i=1}^{6} x_i$, es decir, la función a minimizar es

$$z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6$$

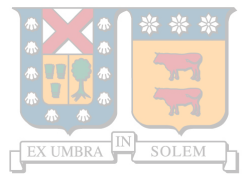


Ejemplo: Solución

Para construir las restricciones necesitamos conocer las ciudades que están dentro de 15 minutos de distancia. Esto se puede ver en el Cuadro 9

Ciudad	Dentro de 15 minutos
1	1, 2
2	1, 2, 6
3	3, 4
4	3, 4, 5
5	4, 5, 6
6	2, 5, 6

Cuadro 9: Ciudades dentro de los 15 minutos

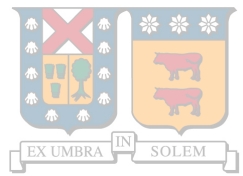


Ejemplo: Solución

Con la información anterior, es posible modelar el problema de la siguiente forma:

$$\text{mín } z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6$$

$$\begin{aligned} \text{s. a} \quad & x_1 + x_2 \geq 1 \\ & x_1 + x_2 + x_6 \geq 1 \\ & x_3 + x_4 \geq 1 \\ & x_3 + x_4 + x_5 \geq 1 \\ & x_4 + x_5 + x_6 \geq 1 \\ & x_2 + x_5 + x_6 \geq 1 \end{aligned}$$



Desarrollo

Para solucionar el problema, se realizaron 4 estrategias de *branching*:

1. Selección de variable de mayor dominio. Selección de valores mayores que el promedio del menor y mayor valor.
2. Selección de variable de mayor dominio. Selección de valores menores que el promedio del menor y mayor valor.
3. Selección de variable de menor dominio. Selección de valores mayores que el promedio del menor y mayor valor.
4. Selección de variable de menor dominio. Selección de valores menores que el promedio del menor y mayor valor.

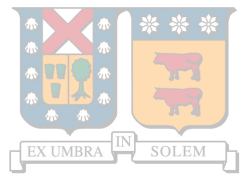
Desarrollo: Resultados

Tanto para la segunda y cuarta (primera y tercera) estrategias se obtuvieron los mismos resultados. La solución del problema es $z = 2$, $x_2 = x_4 = 1$, $x_1 = x_3 = x_5 = x_5 = x_6 = 0$, esto significa ubicar las estaciones de bomberos en las ciudades 2 y 4. Las estadísticas de *Gecode* se muestran en el Cuadro 11

Desarrollo: Estadísticas *Gecode*

Estrategia		**2-4**	**1-3**
Initial	Propagators:	7	7
	Branchers:	2	2
Summary	Runtime:	∼0.23 ms	∼0.33 ms
	Solutions:	1	5
	Propagations:	37	85
	Nodes:	7	17
	Failures:	3	4
	Restarts:	0	0
	No-goods:	0	0
	Peak depth:	4	6

Cuadro 10: Resumen resultados



El problema del Vendedor Viajero

Dada una lista de ciudades y las distancias entre cada par de ellas, el problema del vendedor viajero o (TSP por sus sigas en inglés) busca encontrar la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen.

El problema del Vendedor Viajero

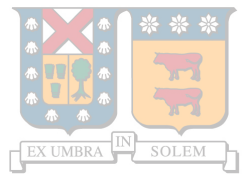
Complejidad

Dentro de la optimización combinatoria se considera un problema *NP-completo*, muy importante en la investigación de operaciones y en la ciencia de la computación.

El problema del Vendedor Viajero

Este problema puede ser formulado como un problema de programación lineal entera. Etiquetando las ciudades con $1, ..., n$ y definiendo

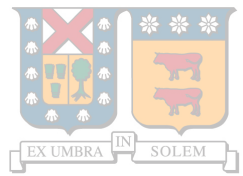
$$x_{ij} = \begin{cases} 1 & \text{si existe un camino para ir de la ciudad } i \text{ a la ciudad } j \\ 0 & \text{en otro caso} \end{cases}$$



El problema del Vendedor Viajero

Para $i = 1, ..., n$, sea u_i una variable artificial, y sea c_{ij} la distancia desde la ciudad i a la ciudad j. El modelo queda definido por

$$\text{Minimizar} \quad Z = \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$



El problema del Vendedor Viajero

Sujeto a

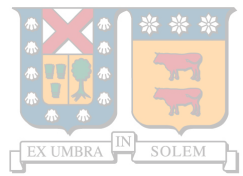
$$0 \leq x_{ij} \leq 1 \qquad i,j = 1,...,n$$

$$u_i \in \mathbb{Z} \qquad i = 1,...,n$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1,...,n$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1,...,n$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq 2$$



Ejercicios

Se probaron 3 de los 4 ejercicios que vienen implementados en el código de ejemplo de *Gecode*.

1. Problema de 7×7.
2. Problema de 10×10.
3. Problema de 17×17

Desarrollo: Estadísticas *Gecode* ejercicio 1

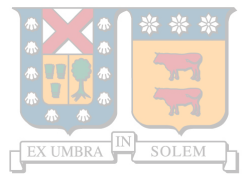
Estrategia		**Ejercicio 1**
Initial	Propagators:	11
	Branchers:	2
Summary	Runtime:	∼0.618 ms
	Solutions:	3
	Propagations:	395
	Nodes:	27
	Failures:	11
	Restarts:	0
	No-goods:	0
	Peak depth:	5

Cuadro 11: Resumen resultados

Desarrollo: Estadísticas *Gecode* ejercicio 2

Estrategia		**Ejercicio 2**
Initial	Propagators:	14
	Branchers:	2
Summary	Runtime:	∼0.814 ms
	Solutions:	5
	Propagations:	617
	Nodes:	53
	Failures:	19
	Restarts:	0
	No-goods:	0
	Peak depth:	12

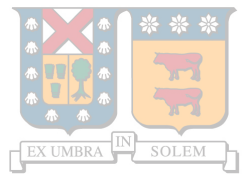
Cuadro 12: Resumen resultados



Desarrollo: Estadísticas *Gecode* ejercicio 3

Estrategia		**Ejercicio 3**
Initial	Propagators:	21
	Branchers:	2
Summary	Runtime:	∼3.632 ms
	Solutions:	3
	Propagations:	4044
	Nodes:	393
	Failures:	185
	Restarts:	0
	No-goods:	0
	Peak depth:	25

Cuadro 13: Resumen resultados



Conclusiones

Las principales conclusiones del curso.

- Los problemas de satisfacción de restricciones tienen una amplia gama de aplicaciones, por ejemplo en logística y asignación de recursos.
- Generalmente muestran gran complejidad, requiriendo de métodos heurísticos y búsqueda combinatoria para ser resueltos en un tiempo razonable.

Conclusiones

- Es importante escoger de forma correcta la estrategia de enumeración y asignación de variables. Esto dado que se disminuyen considerablemente los tiempos, propagaciones y fallos en la búsqueda de las soluciones.
- Gecode es una herramienta potente para resolver problemas de programación con restricciones. Posee una serie de directivas que permiten mejorar el rendimiento de restricciones recurrentes.

Referencias

Gonzalez, T., y Sahni, S. (1976). Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, *23*(4), 665–679.

Schulte, C., Lagerkvist, M., y Tack, G. (2017). Gecode. *Software download and online material at the website: http://www.gecode.org*.

Winston, W. L., y Goldberg, J. B. (2004). *Operations research: applications and algorithms* (Vol. 3). Duxbury press Boston.