Análisis y Diseño de Algoritmos II

Jesús Alexander Aranda Ph.D   Robinson Duque, Ph.D
Juan Francisco Díaz, Ph. D

Universidad del Valle

jesus.aranda@correounivalle.edu.co
robinson.duque@correounivalle.edu.co
juanfco.diaz@correounivalle.edu.co
Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Overview

1. Modelamiento de Problemas I
   - Introducción
   - Generalización de un modelo
   - Arreglos y Conjuntos
2. Modelamiento Básico
   - Problema de coloreado de mapas
   - Ejercicio: Problema de la Dieta
   - Ejercicio: Problema de agricultura

Modelamiento de Problemas I

En esta sección del curso trabajaremos en el modelado e implementación de algunos problemas de **optimización** y **satisfacción de restricciones**.

Problema de la Mochila

Un caminante desea llenar su mochila de capacidad W=6 de forma que se maximice la utilidad de los objetos que empaque. Formule este problema como un MIP teniendo en cuenta la siguiente tabla:

		utility	weight
A	a picture	12	0.2
B	a bottle	7	2
C	another bottle	3	2
D	a pullover	4	0.8
E	chocolate bars	5	1
F	dried fruit	8	1

El objetivo es determinar qué objetos deben tomarse para maximizar la utilidad sin exceder la capacidad de la mochila.

Problema de la Mochila

Modelo de la instancia:

```
maximize      12 * x1 + 7 * x2 + 3 * x3 + 4 * x4 + 5 * x5 + 8 * x6
subject to    x1 * 0,2 + x2 * 2 + x3 * 2 + x4 * 0,8 + x5 + x6 <= 6

              x1, x2, x3, x4, x5, x6 ∈ {0, 1}
```

$$\begin{array}{ll} \texttt{maximize} & 12 * x_1 + 7 * x_2 + 3 * x_3 + 4 * x_4 + 5 * x_5 + 8 * x_6 \\ \texttt{subject to} & x_1 * 0{,}2 + x_2 * 2 + x_3 * 2 + x_4 * 0{,}8 + x_5 + x6 \leq 6 \\ & \\ & x_1, x_2, x_3, x_4, x_5, x_6 \in \{0, 1\} \end{array}$$

Problema de la Mochila

Implementación en MiniZinc

```
var int: utilidad;
var bool: x1;
var bool: x2;
var bool: x3;
var bool: x4;
var bool: x5;
var bool: x6;

constraint x1*0.2 + x2*2 + x3*2 + x4*0.8+ x5*1 +
    x6*1 <= 6;
constraint utilidad = 12*x1 + 7*x2 + 3*x3 + 4*x4 +
    5*x5 + 8*x6;
solve maximize utilidad;
```

Solución: utilidad = 36, x1 = true, x2 = true, x3 = false, x4 = true, x5 = true, x6 = true.

Problema de la Mochila

- Note que la implementación propuesta depende completamente de la instancia que se propone en el problema.
- Si quisieramos resolver otra instancia (e.g., con datos distintos y/o diferentes objetos en la mochila), tendríamos que modificar nuestro modelo.
- Es necesario realizar una generalización de nuestro modelo y de la implementación asociada para resolver cualquier instancia dada.

Problema de la Mochila

Necesitamos entonces distinguir nuestros datos de entrada o parámetros:

Problema de la Mochila

Necesitamos entonces distinguir nuestros datos de entrada o parámetros:

- n : número de objetos;
- w_j: el peso del objeto j $(\forall j \in \{1, .., n\})$;
- c_j: la utilidad del objeto j $(\forall j \in \{1, .., n\})$;
- W: la capacidad de la mochila;

Problema de la Mochila

Necesitamos ahora denotar nuestras variables:

Problema de la Mochila

Necesitamos ahora denotar nuestras variables:

$$x_j = \begin{cases} 1, & \text{si el objeto j debe estar dentro de la mochila} \\ 0, & \text{de lo contrario} \end{cases}$$

$$(\forall j \in \{1, .., n\})$$

Problema de la Mochila

Necesitamos ahora denotar nuestras restricciones:

Problema de la Mochila

Necesitamos ahora denotar nuestras restricciones:

- La capacidad de la mochila no se excede:

$$\sum_{j=1}^{n} w_j * x_j \leq W$$

Problema de la Mochila

Necesitamos ahora denotar nuestra función objetivo:

Problema de la Mochila

Necesitamos ahora denotar nuestra función objetivo:

- Maximizar la utilidad:

$$\text{maximize: } \sum_{j=1}^{n} c_j * x_j$$

Problema de la Mochila

A nivel de implementación podemos dividir nuestro problema en dos:

- Datos de entrada (archivo *.dzn)
- El modelo generalizado (archivo *.mzn)

```
% Archivo de datos *.dzn
n=6;
W=6;
w=[0.2,2,2,0.8,1,1];
c=[12,7,3,4,5,8]
```

Problema de la Mochila

```
% Modelo genérico *.mzn

% Lectura de datos:
int: n;
array[1..n] of float: w;
array[1..n] of float: c;
int: W;
% Variables:
array[1..n] of var bool: x;
% Restricciones:
constraint sum(j in 1..n)( w[j]* x[j] ) <= W;
% Objetivo:
var int: utilidad;
constraint utilidad = sum(j in 1..n)( c[j]* x[j] );
solve maximize utilidad;
```

Arreglos y Conjuntos

Casi siempre estamos interesados en construir modelos en los que el número de restricciones y variables dependa de los datos de entrada. Para ello normalmente utilizaremos arreglos, matrices y conjuntos.

Arreglos y Conjuntos

Los conjuntos se declaran de la forma:

```
set of <type-inst> : <var-name> = { <expr-1>, ... ,
    <expr-n> };
```

donde se permiten conjuntos de enteros, enumeraciones, flotantes o booleanos.

```
set of int : A = { 1,5,7,9};
set of float : B = { 1.4,3.6,7.9,8.5};
set of int : C = 20..30;
```

Los literales establecidos son de la forma:

```
{ <expr-1>, ... , <expr-n> }
```

o son expresiones de rango sobre enteros, enumeraciones o flotantes de la forma:

```
<expr-1> .. <expr-2>
```

Arreglos y Conjuntos

Las variables de conjunto (i.e., Set variables) se declaran de la forma:

```
var set of <type-inst> : <var-name> ;
```

El único tipo permitido son los conjuntos de variables de enteros o enumeraciones (las variables de conjunto están por fuera del alcance de este curso).

Arreglos y Conjuntos

MiniZinc proporciona **matrices unidimensionales y multidimensionales** que se declaran utilizando el tipo:

```
array [<index-set-1 >,...,<index-set-n >] of
<type-inst>
```

- MiniZinc requiere que la declaración de matrices contenga el conjunto de índices de cada dimensión y que el conjunto de índices sea un rango de enteros, una variable de conjunto iniciada en un rango de enteros o un tipo de enumeración.
- Las matrices pueden contener cualquiera de los tipos básicos: enteros, enumeraciones, valores booleanos, flotantes o cadenas. Las matrices también pueden contener conjuntos, pero no pueden contener matrices.

Arreglos y Conjuntos

Los Arreglos unidimensionales son de la forma:

```
[ <expr−1 >,  . . . ,  <expr−n>  ]
```

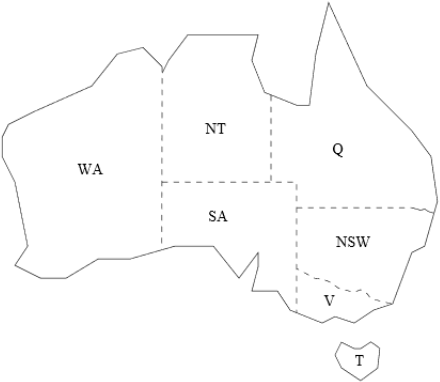
Los Arreglos bidimensionales son de la forma:

```
[|  <expr−1−1 >,  . . . ,  <expr−1−n>  |
    . . .                              |
    <expr−m−1 >,  . . . ,  <expr−m−n>  |]
```

Donde el arreglo tiene *m* filas y *n* columnas.

Problema de coloreado de mapas

Imagine que deseamos colorear un mapa de Australia como se muestra en la Figura. Se compone de siete estados y territorios diferentes, cada uno de los cuales debe tener un color para que las regiones adyacentes tengan colores diferentes.



Problema de coloreado de mapas

```
% Colouring Australia using nc colours
int: nc = 3;
var 1..nc: wa; var 1..nc: nt;  var 1..nc: sa;
var 1..nc: q;  var 1..nc: nsw; var 1..nc: v;
var 1..nc: t;
constraint wa != nt;    constraint wa != sa;
constraint nt != sa;    constraint nt != q;
constraint sa != q;     constraint sa != nsw;
constraint sa != v;     constraint q != nsw;
constraint nsw != v;

solve satisfy;
output ["wa=\(wa)\t nt=\(nt)\t sa=\(sa)\n","q=\(q)\
    t nsw=\(nsw)\t v=\(v)\n", "t=",show(t),"\n"];
```

Problema de coloreado de mapas

- ¿Cómo podemos generalizar el problema de coloreado de mapas?

Problema de coloreado de mapas

- ¿Cómo podemos generalizar el problema de coloreado de mapas?
- **Datos de entrada (Parámetros):**

Problema de coloreado de mapas

- ¿Cómo podemos generalizar el problema de coloreado de mapas?
- **Datos de entrada (Parámetros):**
  - nc: número de colores a utilizar;
  - nt: número de territorios a colorear (e.g., ciudades, pueblos, etc.);
  - $vecinos_{ij}$: matriz que representa m "parejas" de valores de territorios vecinos $(vecinos_{i1}, vecinos_{i2})$; $i \in \{1, .., m\}; j \in \{1, 2\}$

Problema de coloreado de mapas

- **Variables:**

Problema de coloreado de mapas

- **Variables:**
  - $Coloreado_i$: matriz con nt valores a colorear.
    $Coloreado[i] \in \{1 ... nc\}$;
- **Restricciones:**

Problema de coloreado de mapas

- **Variables:**
  - $Coloreado_i$: matriz con nt valores a colorear.
    $Coloreado[i] \in \{1 ... nc\}$;
- **Restricciones:**
  - $Coloreado_i$: matriz con nt valores a colorear.
    $\forall v_{i1}, v_{i2} \in vecinos_{ij}, Coloreado[v_{i1}] \neq Coloreado[v_{i2}]$;
- **Función objetivo:**

Problema de coloreado de mapas

- **Variables:**
  - $Coloreado_i$: matriz con nt valores a colorear.
    $Coloreado[i] \in \{1...nc\}$;
- **Restricciones:**
  - $Coloreado_i$: matriz con nt valores a colorear.
    $\forall v_{i1}, v_{i2} \in vecinos_{ij}, Coloreado[v_{i1}] \neq Coloreado[v_{i2}]$;
- **Función objetivo:**
  - Este es un problema de satisfacción de restricciones, no requiere una función a optimizar;

Problema de coloreado de mapas

```
nc=3;
nt=7;
vecinos=[|1,2|1,3|2,4|2,3|3,4|3,5|3,6|4,5|5,6|];
```

```
%Lectura de Datos
int: nc; %numero de colores
int: nt; %numero de territorios
array[int, int] of int: vecinos;  %Datos Restric.
int: numFilas = (length(vecinos) div 2); %Cálculo
    del número de filas de matriz Vecinos
%Variables
array[1..nt] of var 1..nc: Coloreado;
%Restricciones
constraint forall(i in 1..numFilas) (Coloreado[
    vecinos[i,1] ] != Coloreado[ vecinos[i,2] ]);
solve satisfy;
output["Mapa Coloreado = \(Coloreado)"];
```

Ejercicio: Problema de la Dieta

En una clase anterior resolvimos este problema...
Los datos de contenido nutricional de un grupo de alimentos y la necesidad semanal de un adulto se presentan en la tabla que se muestra a continuación. **Determine el costo semanal más bajo para cumplir con los requerimientos mínimos semanales (i.e., 550g de proteina, 600g de grasa, 2000g de carbohidratos).**

	Food	Proteins	Fats	Carbohydrates	Cost $ per 100g
1	Bread	8%	1%	55%	0.25
2	Butter	–	90%	–	0.5
3	Cheese	25%	36%	–	1.2
4	Cereal	12%	3%	75%	0.6
5	Diet Bar	8%	–	50%	1.5
	Weekly requirement (g)	550	600	2000	

Ejercicio: Problema de la Dieta

Modelo final:

$$
\begin{array}{ll}
\texttt{minimize} & f = 0{,}25x_1 + 0{,}5x_2 + 1{,}2x_3 + 0{,}6x_4 + 1{,}5x_5 \\
\texttt{subject to} & 0{,}08x_1 + 0{,}25x_3 + 0{,}12x_4 + 0{,}08x_5 \geq 550 \\
& 0{,}01x_1 + 0{,}9x_2 + 0{,}36x_3 + 0{,}03x_4 \geq 600 \\
& 0{,}55x_1 + 0{,}75x_4 + 0{,}5x_5 \geq 2000 \\
& x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0
\end{array}
$$

Actividad en Clase

Proponga un modelo genérico para el problema de la dieta. Utilice constantes, arreglos o matrices para representar los datos de entrada, variables, restricciones y función objetivo (Proteinas, grasa, carbohidratos, etc). Escriba su modelo y luego implementelo en MiniZinc.

Ejercicio: Problema de agricultura

En una clase anterior resolvimos este problema...
Un agricultor de vegetales tiene la opción de producir tomates, pimientos verdes o pepinos en su granja de 200 acres. Un total de 500 días-hombre de trabajo están disponibles. En la tabla se muestran los rendimientos y los días-hombre de trabajo por acre:

	Yield \$/ acre	Labor man-days/acre
Tomatoes	450	6
Green Peppers	360	7
Cucumbers	400	5

Suponiendo que los costos de los fertilizantes son los mismos para cada producto, determine la combinación óptima de cultivos.

Ejercicio: Problema de agricultura

Sean x_1, x_2 y x_3 los acres de tierra para tomates, pimientos verdes y pepinos respectivamente. El problema del LP puede ser declarado como:

$$
\begin{array}{ll}
\texttt{maximize} & f = 450x_1 + 360x_2 + 400x_3 \\
\texttt{subject to} & x1 + x2 + x3 \leq 200 \\
 & 6x1 + 7x2 + 5x3 \leq 500 \\
 & x1 \geq 0, x2 \geq 0, x3 \geq 0
\end{array}
$$

Actividad en clase

Proponga un modelo genérico para el problema de agricultura. Utilice constantes, arreglos o matrices para representar los datos de entrada, variables, restricciones y función objetivo. Asuma primero un modelo que soporte solamente tomates, pimientos verdes y pepinos. Luego extiéndalo para soportar cualquier número de productos.

Fin de la Presentación

Lecturas recomendadas:

Modelamiento básico en MiniZinc:
https://www.minizinc.org/doc-2.2.3/en/modelling.html

Modelos más complejos:
https://www.minizinc.org/doc-2.2.3/en/modelling2.html

Fin de la Presentación

¿Preguntas?