Fundamentos de Programación Entera
B. Sistema de modelado algebraico - GLPK

Carlos Testuri – Fernando Islas

Departamento de Investigación Operativa – Instituto de Computación
Facultad de Ingeniería – Universidad de la República

2012-2023

Contenido

Sistemas de modelado algebraico

En general disponen de un *lenguaje de modelado algebraico* que permite trabajar en alto nivel de abstracción, utilizando una notación similar a la matemática.

Algunos tienen un entorno de desarrollo integrado (IDE) y/o un interprete de línea de comandos incorporados.

Todos proveen mecanismos para vincularlos con *solvers* y así poder resolver los modelos. En algunos casos el *solver* viene incluido en el sistema de modelado.

Ejemplos:

- *GAMS* – General Algebraic Modeling System
- *Xpress* – Modelado matemático y optimización
- *AMPL* – A Modeling Language for Mathematical Programming
- ***GLPK* – GNU Linear Programming Kit**

GLPK

El *GNU Linear Programming Kit* (GLPK) es un sistema de modelado algebraico con soporte para problemas de programación entera mixta. Es software libre con licencia GPL.

Como lenguaje de modelado algebraico utiliza *GNU MathProg* que es una versión libre pero reducida del lenguaje de modelado algebraico AMPL, considerado un estándar de echo para problemas de optimización matemática

Incluye un *solver* para problemas de programación lineal o entera mixta, que puede ser invocado por línea de comandos mediante el comando `glpsol` para resolver modelos escritos en MathProg.

Instalación en Ubuntu y macOS

GLPK está disponible en una gama amplia de sistemas operativos.

Sitio oficial (código fuente, listas de correo):
`http://www.gnu.org/software/glpk/glpk.html`

Ubuntu:
```
sudo apt install glpk-utils
```

macOS:

- Utilizando *homebrew*: `brew install glpk`
- Utilizando *macports*: `sudo port install glpk`

Instalación en Windows

GLPK for Windows:
`http://winglpk.sourceforge.net/`

El proyecto *GLPK for Windows* provee ejecutables pre-compilados para Windows.

El zip de la distribución contiene los ejecutables para 32 y 64 bits en las carpetas *w32* y *w64* respectivamente. Se recomienda agregar la carpeta correspondiente al *PATH* del sistema para poder invocar al *solver* desde cualquier carpeta.

GUSEK (GLPK Under Scite Extended Kit):
`http://gusek.sourceforge.net/gusek.html`

GUSEK provee un IDE básico para GLPK. Incluye un editor de texto con resaltado de syntaxis para los archivos de modelo y datos, consola de errores integrada y la posibilidad ejecutar el solver desde la interfaz gráfica.

Verificación

Para verificar instalación, ejecutar desde línea de comandos:

```
glpsol --version
```

Salida esperada:

```
GLPSOL: GLPK LP/MIP Solver, v4.55

Copyright (C) 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008,
2009, 2010, 2011, 2013, 2014 Andrew Makhorin, Department for Applied
Informatics, Moscow Aviation Institute, Moscow, Russia. All rights
reserved. E-mail: <mao@gnu.org>.

This program has ABSOLUTELY NO WARRANTY.

This program is free software; you may re-distribute it under the terms
of the GNU General Public License version 3 or later.
```

Documentación GNU MathProg

Las distintas distribuciones tienen incluida la documentación de GLPK.

En particular, el manual de referencia de *MathProg* es el archivo `gmpl.pdf`.

También se puede encontrar *online*, por ejemplo en:
`http://gusek.sourceforge.net/gmpl.pdf`

Formulación básica (vectorial)

$$\begin{array}{ll} \max & c^\tau x \\ \text{s.a} & Ax \leq b \\ & x \geq 0, \text{entero.} \end{array}$$

Formulación básica (extendida)

$$\begin{array}{ll} \max & c^\tau x \\ \text{s.a} & Ax \leq b \\ & x \geq 0, \text{entero.} \end{array}$$

$$\begin{array}{ll} \max & \sum_{j \in J} c_j x_j \\ \text{s.a} & \sum_{j \in J} A_{ij} x_j \leq b_i, \forall i \in I \\ & x_j \geq 0, \text{entero}, \forall j \in J. \end{array}$$

Formulación básica (MathProg)

$$
\begin{array}{ll}
\max & \sum_{j \in J} c_j x_j \\
\text{s.a} & \sum_{j \in J} A_{ij} x_j \leq b_i, \forall i \in I \\
& x_j \geq 0, \text{entero}, \forall j \in J.
\end{array}
$$

```
maximize obj:
   sum{j in J} c[j] * x[j];

s.t. ctr{i in I}:
   sum{j in J} A[i,j] * x[j] <= b[i];
```

Formulación básica (MathProg modelo completo)

Conjuntos

I

J

Parámetros

c_j

b_i

A_{ij}

Variable de decisión

x_j

$$\begin{array}{ll} \max & \sum_{j \in J} c_j x_j \\ \text{s.a} & \sum_{j \in J} A_{ij} x_j \leq b_i, \forall i \in I \\ & x_j \geq 0, \text{entero}, \forall j \in J. \end{array}$$

```
set I;
set J;

param c{J};
param b{I};
param A{I,J};

var x{J} >= 0, integer;

maximize obj:
  sum{j in J} c[j] * x[j];

s.t. ctr{i in I}:
  sum{j in J} A[i,j] * x[j]
    <= b[i];

end;
```

Formulación básica (modelo y datos)

basico.mod

```
set I;
set J;

param c{J};
param b{I};
param A{I,J};

var x{J} >= 0, integer;

maximize obj:
  sum{j in J} c[j] * x[j];

s.t. ctr{i in I}:
  sum{j in J} A[i,j] * x[j] <= b[i];

end;
```

basico.dat

```
set I := 1, 2, 3;
set J := 1, 2, 3, 4;

param c := 1 11,
           2 22,
           3 33,
           4 44;

param b :=
  1 100, 2 200, 3 300;

param A :
     1   2   3   4 :=
 1   5   6   7   8
 2   9  10  11  12
 3  13  14  15  15 ;

end;
```

Resolución

Ejecutar desde línea de comandos:

```
glpsol --model basico.mod --data basico.dat --output basico.out
```

- `--model` indica el archivo con el modelo
- `--data` indica el archivo con los datos
- `--output` indica el archivo para guardar la solución

Salida (basico.out)

Resumen con tamaño del problema y *status* final:

```
Problem:    basico
Rows:       4
Columns:    4 (4 integer, 0 binary)
Non-zeros:  16
Status:     INTEGER OPTIMAL
Objective:  obj = 528 (MAXimum)
```

Salida (basico.out)

Actividad en las filas:

No.	Row name	Activity	Lower bound	Upper bound
1	obj	528		
2	ctr[1]	100		100
3	ctr[2]	152		200
4	ctr[3]	195		300

Actividad en las columnas:

No.	Column name		Activity	Lower bound	Upper bound
1	x[1]	*	0	0	
2	x[2]	*	0	0	
3	x[3]	*	4	0	
4	x[4]	*	9	0	

Modelo y datos en un mismo archivo

todo.mod

```
set I;
set J;

maximize obj: ...

data;

set I := 1, 2, 3;
set J := 1, 2, 3, 4;

...

end;
```

En este caso no hace falta especificar `--data` al resolver:

```
glpsol --model todo.mod --output todo.out
```

Ejemplo: Mochilero

mochilero.mod

```
param n;
set I := 1 .. n; # objetos

param c{I}; # valor
param a{I}; # tamaño
param b;    # capacidad

var x{I} binary; # llevar si/no

maximize valor:
  sum{i in I} c[i] * x[i];

s.t. capacidad:
  sum{i in I} a[i] * x[i] <= b;

end;
```

mochilero.dat

```
param n := 8;

param c :=
  1   1, 2 10, 3 15,
  4  40, 5 60, 6 90,
  7 100, 8 15;

param a :=
  1 10, 2 60, 3 30,
  4 40, 5 30, 6 20,
  7 20, 8  2;

param b := 101;

end;
```

Ejemplo: Asignación

asignacion.mod

```
param n;
set I := 1 .. n; # personas
set J := 1 .. n; # tareas

param c{I,J}; # costo

var x{I,J} binary; # asignación i a j

minimize costo:
  sum{i in I, j in J} c[i,j] * x[i,j];

s.t. tareas_personas{i in I}:
  sum{j in J} x[i,j] = 1;

s.t. personas_tareas{j in J}:
  sum{i in I} x[i,j] = 1;

end;
```

asignacion.dat

```
param n := 4;

param c :
       1   2   3   4 :=
  1   54  54  51  53
  2   51  57  52  52
  3   50  53  54  56
  4   56  54  55  53 ;

end;
```

Ejemplo: Conjuntos indizados y parámetros de más de dos dimensiones

prod.mod

```
set P;           # productos
set A{P};        # áreas de mercado
param h;         # número semanas
set T := 1..h; # semanas

# límite de producto por semana
param market{p in P, A[p], T}
  >= 0;

# producto vendido
var Sell{
  p in P, a in A[p], t in T
} >= 0, <= market[p,a,t];

...

end;
```

prod.dat

```
param h := 3;
set P := b c;
set A[b] := east north;
set A[c] := east west export;

param market :=
[b,*,*]:    1     2     3 :=
east     2000  2000  1500
north    4000  4000  2500

[c,*,*]:    1     2     3 :=
east     1000   800  1000
west     2000  1200  2000
export   1000   500   500 ;

...

end;
```

Ejemplo: Producto cartesiano en vez de conjunto indizado

prod2.mod

```
set P;            # productos
set A;            # áreas de mercado
set PA within P cross A; # <-
param h;          # número semanas
set T := 1..h; # semanas

# límite de producto por semana
param market{PA, T} >= 0;

# producto vendido
var Sell{(p, a) in PA, t in T}
  >= 0, <= market[p,a,t];

...

end;
```

prod2.dat

```
param h := 3;
set P := b c;
set A :=
 east west north export;
set PA :=
 (b,east) (b,north)
 (c,east) (c,west) (c,export);

param market :=
[b,*,*]:    1     2     3 :=
east     2000  2000  1500
north    4000  4000  2500

[c,*,*]:    1     2     3 :=
east     1000   800  1000
west     2000  1200  2000
export   1000   500   500 ;
```

Impresión de resultados

MathProg incluye comandos para imprimir el resultado de la evaluación de expresiones, tanto en la salida estándar como en archivo.

Los comandos disponibles son:

- `display`
- `printf`

A la hora de imprimir los resultados nos va a resultar útil iterar sobre los datos. El comando disponible para iteración en MathProg es `for`.

Impresión de resultados

Esto es particularmente útil para ver en forma más amigable el resultado de la resolución del modelo. Para esto es requerido que previamente se invoque la resolución del modelo con el comando `solve`.

modelo.mod

```
set I;
set J;

maximize obj: ...

solve;

... display, printf, for ...

end;
```

display

Modelo

```
param n;
set I := 1 .. n; # objetos

param c{I}; # valor
param a{I}; # tamaño
param b;    # capacidad

var x{I} binary; # llevar si/no

... modelo mochilero ...

solve;

display{i in I}: i, x[i];
display:
  sum{i in I} a[i] * x[i], b;

end;
```

Salida

```
Display statement at line 18
i = 1
x[1].val = 1
i = 2
x[2].val = 0
i = 3
x[3].val = 0
i = 4
x[4].val = 0
i = 5
x[5].val = 1
i = 6
x[6].val = 1
i = 7
x[7].val = 1
i = 8
x[8].val = 1
Display statement at line 19
82
b = 101
```

printf

Modelo

```
param n;
set I := 1 .. n; # objetos

param c{I}; # valor
param a{I}; # tamaño
param b;    # capacidad

... modelo mochilero ...

solve;

printf{i in I}:
  "objeto␣%s,␣decisión␣%d\n", i, x[i];
printf:
  "volumen␣%.2f,␣\ncapacidad␣%d\n",
  sum{i in I} a[i] * x[i], b;

end;
```

Salida

```
objeto 1, decisión 1
objeto 2, decisión 0
objeto 3, decisión 0
objeto 4, decisión 0
objeto 5, decisión 1
objeto 6, decisión 1
objeto 7, decisión 1
objeto 8, decisión 1
volumen 82.00,
capacidad 101
```

for

Modelo

```
param n;
set I := 1 .. n; # objetos

... modelo mochilero ...

solve;

for{i in I : x[i] == 1} {
  printf: "El␣objeto␣%s␣es␣llevado\n",
  i;
}

for{i in I : x[i] == 0} {
  printf: "El␣objeto␣%s", i;
  printf: "␣NO␣es␣llevado\n";
}

end;
```

Salida

```
El objeto 1 es llevado
El objeto 5 es llevado
El objeto 6 es llevado
El objeto 7 es llevado
El objeto 8 es llevado
El objeto 2 NO es llevado
El objeto 3 NO es llevado
El objeto 4 NO es llevado
```

Ejecución

Por omisión los mensajes se van a mostrar en la consola:

```
glpsol --model prob.mod --data prob.dat
```

Para guardar la salida personalizada en un archivo, se debe utilizar la opción `--display`:

```
glpsol --model prob.mod --data prob.dat --display prob.dsp
```