


GLPK Formato LP Formato MathProg

Programação Linear Inteira UTILIZANDO O GLPK

Haroldo Gambini Santos
Universidade Federal de Ouro Preto

30 de agosto de 2011



Programação Linear Inteira, UTILIZANDO O GLPK 1 / 30

Notas

GLPK Formato LP Formato MathProg

Conteúdo

- 1 GLPK
- 2 Formato LP
- 3 Formato MathProg




Programação Linear Inteira, UTILIZANDO O GLPK 2 / 30

Notas

GLPK Formato LP Formato MathProg

GLPK

- *GNU Linear Programming Kit*
- resolvidor de problemas lineares, incluindo problemas de programação inteira
- código aberto, disponível gratuitamente
- farta documentação



Programação Linear Inteira, UTILIZANDO O GLPK 3 / 30

Notas

GLPK Formato LP Formato MathProg

GLPK

Instalação

- Windows:
 - <http://glplabw.sourceforge.net/>
- Linux (debian/ubuntu e derivadas):
 - `sudo apt-get install glpk`



Programação Linear Inteira, UTILIZANDO o GLPK 4 / 30


Notas

GLPK Formato LP Formato MathProg

GLPK

Obtendo Ajuda

- Pacote disponível em: <http://www.gnu.org/software/glpk/>
 - Manuais em PDF (diretório `doc`)
 - Exemplos (diretório `examples`)
- Lista de Discussão: help-glpk@gnu.org



Programação Linear Inteira, UTILIZANDO o GLPK 5 / 30


Notas

GLPK Formato LP Formato MathProg

Introdução

Formatos conhecidos para entrada de programas lineares:

- MPS *Mathematical Programming System* :
 - padrão da indústria
 - pouco intuitivo, confuso e com limitações
- LP *CPLEX LP file format* :
 - padrão criado para uso com o resolvidor CPLEX
 - mais fácil e prático do que o formato MPS
 - aceito nos principais resolvidores modernos
 - arquivos podem ser convertidos para MPS



Programação Linear Inteira, UTILIZANDO o GLPK 6 / 30

Notas

Formato LP

Componentes

- função objetivo
- restrições
- informações de variáveis
 - limites
 - variáveis inteiras genéricas
 - variáveis binárias



Notas

Problema da Dieta

DIETA.LP

Minimize

$$\text{obj: } 3 x_1 + 2.5 x_2$$

Subject to

$$\text{vitaminas: } 6 x_1 + 4 x_2 \geq 32$$

$$\text{proteinas: } 5 x_1 + 6 x_2 \geq 36$$

End



Notas

Resolvendo

Comando

```
glpsol --cpxlp dieta.lp -o sol.txt
```

sol.txt

```

Problem:
Rows:    2
Columns: 2
Non-zeros: 4
Status:  OPTIMAL
Objective: obj = 17.75 (MINIMUM)

```

No.	Row name	St	Activity	Lower bound	Upper bound
1	vitaminas	NL	32	32	
2	proteinas	NL	36	36	

No.	Column name	St	Activity	Lower bound	Upper bound
1	x1	B	3	0	
2	x2	B	3.5	0	



Notas

Limites e Integralidade

- Podem ser especificadas restrições específicas sobre algumas variáveis
- Essas restrições são informadas após a seção das restrições normais e podem ser dos seguintes tipos:
 - Restrição de limites - Seção Bounds:

Exemplo Limites

```
Subject to
...
Bounds
0 <= x1 <= 40
2 <= x4 <= 3
```



Notas

Limites e Integralidade (cont.)

- Para variáveis para as quais não são permitidos valores fracionários temos as seções:
 - **General**: variáveis inteiras de modo geral
 - **Binary**: variáveis binárias que somente podem assumir valor zero ou um

Exemplo Integralidade

```
Subject to
...
Binary
x3
General
x1
x2
```



Notas

Problema da Dieta

DIETAINT.LP

Minimize

obj: 3 x1 + 2.5 x2

Subject to

vitaminas: 6 x1 + 4 x2 \geq 32

proteinas: 5 x1 + 6 x2 \geq 36

General

x1

x2

End



Notas

GLPK Formato LP Formato MathProg

Nova Solução

sol.txt

Problem:
 Rows: 2
 Columns: 2 (2 integer, 0 binary)
 Non-zeros: 4
 Status: INTEGER OPTIMAL
 Objective: obj = 18.5 (MINimum)

No.	Row name	Activity	Lower bound
1	vitaminas	32	32
2	proteinas	40	36

No.	Column name	Activity	Lower bound
1	x1	*	2
2	x2	*	5

Programação Linear Inteira, UTILIZANDO o GLPK

13 / 30

Notas

GLPK Formato LP Formato MathProg

Formato *MathProg*

- Formato de mais alto nível, incluindo abstrações:
 - Comandos
 - Conjuntos
 - Parâmetros
- Facilita a geração de problemas grandes e complexos
- Permite a separação entre o modelo e os dados

Programação Linear Inteira, UTILIZANDO o GLPK

14 / 30

Notas

GLPK Formato LP Formato MathProg

Resolvendo a Dieta, versão *MathProg*

DIETA.MOD

```

set N; /* nutrientes */
set A; /* alimentos */
param r{N}; /* requerimento diario por nutriente */
param v{A,N}; /* valor que um alimento oferece
              de um nutriente */
param c{A}; /* custo unit. de cada alimento */

var x{a in A} >= 0;
/* reais do alimento a que devem ser comprados */

s.t. nutrir{n in N}: sum{a in A} v[a,n] * x[a] >= r[n];
/* satisfazer necessidade de nutriente n */

minimize custo: sum{a in A} c[a] * x[a]; /* conta */
    
```

Programação Linear Inteira, UTILIZANDO o GLPK

15 / 30

Notas

Resolvendo a Dieta, versão *MathProg*

DIETA.DAT (parte 1/2)

```

data;
param : N : r :=
    Calorias      3 /* milhares */
    Proteinas     70 /* gramas */
    Calcio        24 /* gramas */
    Ferro         33 /* miligramas */ ;
set A := Trigo Queijo Fgado Salmao Espinafre;
param v default 0
:   Calorias  Proteinas  Calcio  Ferro :=
Trigo    44.7    30        2.0    20
Queijo   7.4     44        16.4   19
Fgado    2.2     33         .2     80
Salmao   5.8     70        6.8    45
Espinafre 1.1     10        9.0    138 ;

```

(continua)



Notas

Resolvendo a Dieta, versão *MathProg*

DIETA.DAT (parte 2/2)

```

param : c :=
    Trigo      0.9
    Queijo     1.5
    Fgado      2.1
    Salmao     2.4
    Espinafre  2.2 ;

```



Notas

Resolvendo a Dieta, versão *MathProg*

Comando

```
glpsol -m dieta.mod -d dieta.dat -o dietaSol.txt
```



Notas

Comando set (1/2)

```

set I; /* declara conjunto que sera informado depois */
set I := 1..10; /* declara um conjunto
               preenchendo com num. de 1 a 10 */
set A := Trigo Queijo Figado Salmao Espinafre;
/* declara e informa elementos de um conjunto */
set Cidades;
set Capitais within Cidades;
/* conjunto capitais fica restrito a ser um
   subconjunto de cidades */

```



Notas

Comando set (2/2)

```

set C{1..m};
/* conjunto de m conjuntos */
set C[1] := 1 14 3;
set C[2] := 5 9 25 18;
...
/* preenchimento de C */

```



Notas

Comando param (1/2)

```

param n;
/* parametro que sera informado depois */
param c, > 0;
/* parametro que sera informado depois e
   deve ser positivo */
param n, integer, > 0;
/* parametro que sera informado depois e
   deve ser positivo e inteiro */

```



Notas

Comando param (2/2)

```
param c{A};
/* parametro que deve ser informado para cada
   elemento do conjunto A */

param :   c :=
         elementoDeA1  valorDeCparaA1
         elementoDeA2  valorDeCparaA2
         ...           ;
/* valor do parametro c para cada element de A */
```



Notas

Comando var

```
var x >= 0;
/* uma variavel positiva */

var x{I,J}, integer, >= 0;
/* cria uma variavel x para cada par (i,j) de I e J,
   variavel identificada por x[i,j] */

var z{i in I, j in J} >= i+j;
/* cria uma variavel z para cada par (i,j) de I e J,
   variavel identificada por z[i,j],
   cada variavel pode assumir valor minimo i+j */
```



Notas

Comando de Adição de Restrições: s.t.

```
s.t. r: x + y + z, >= 0, <= 1;
/* restricao com nome r, restringindo o somatorio
   de x + y + z para o intervalo [0,1] */

s.t. capacidade: sum{i in I} p[i]*x[i] <= c;
/* uma restricao gerada sobre o somatorio da variavel
   x[i] multiplicando uma constante p[i] */

s.t. estoqueFinal{p in P}:
   estoque[12,p] + compras[12,p] - vendas[12,p] >= 500;
/* uma restricao sera gerada para cada produto p */
```



Notas

Exemplo: Problema da Mochila - Modelo

```

param n, integer, > 0; /* numero de itens */
param c; /* capacidade */
set I := 1..n; /* itens */
param l{I}; /* lucro */
param p{I}; /* peso */
var x{I} binary;
/* 1 se incluir, 0 caso contrario */
maximize lucro: sum{i in I} l[i]*x[i];
s.t. capacidade: sum{i in I} p[i]*x[i] <= c;

```



Notas

Exemplo: Problema da Mochila - Dados

```

data;
param n:= 7;
param c:= 10;
param l := 1 7, 2 7, 3 2, 4 1,
           5 5, 6 5, 7 8;
param p := 1 3, 2 4, 3 1, 4 1,
           5 3, 6 3, 7 6;

```



Notas

O Problema da Alocação Generalizada

Definição

Temos um conjunto de **tarefas** a realizar e um conjunto de **agentes** que podem realizá-las.

Cada agente tem uma determinada habilidade para cada tarefa, influenciando no tempo que o mesmo leva para concluir a tarefa e também determinando o custo do mesmo realizar aquela tarefa. Além disso, os agentes tem uma capacidade limitada.

Deve-se determinar quais agentes farão cada tarefa respeitando suas capacidades e minimizando o custo total.



Notas

Problema da Alocação Generalizada - Modelo (1/2)

```

param n, integer, > 0; /* n tarefas */
param m, integer, > 0; /* m agentes */
set J := 1..n; /* conjunto de tarefas */
set I := 1..m; /* conjunto de agentes */
param a{i in I, j in J}, >= 0; /* recursos que o ag. i
                             gasta para tarefa j */
param b{i in I}, >= 0; /* capacidade em
                       recursos do ag. i */
param c{i in I, j in J}, >= 0; /* custo do agente i
                             fazer tarefa j */

```



Notas

Problema da Alocação Generalizada - Modelo (2/2)

```

var x{i in I, j in J}, binary;
/* x[i,j] = 1 : agente i faz tarefa j */

s.t. processar{j in J}: sum{i in I} x[i,j] = 1;
/* cada tarefa deve ser processada por um agente */

s.t. lim{i in I}: sum{j in J} a[i,j] * x[i,j] <= b[i];
/* capacidade do agente */

minimize obj: sum{i in I, j in J} c[i,j] * x[i,j];
/* achar alocacao mais barata */

```



Notas

Problema da Alocação Generalizada - Dados

```

data;
param m := 5;
param n := 15;
param a :
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 :=
1  8 15 14 23  8 16  8 25  9 17 25 15 10  8 24
2  15  7 23 22 11 11 12 10 17 16  7 16 10 18 22
3  21 20  6 22 24 10 24  9 21 14 11 14 11 19 16
4  20 11  8 14  9  5  6 19 19  7  6  6 13  9 18
5  8 13 13 13 10 20 25 16 16 17 10 10  5 12 23 ;
param b := 1 36, 2 34, 3 38, 4 27, 5 33;
param c :
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 :=
1  17 21 22 18 24 15 20 18 19 18 16 22 24 24 16
2  23 16 21 16 17 16 19 25 18 21 17 15 25 17 24
3  16 20 16 25 24 16 17 19 19 18 20 16 17 21 24
4  19 19 22 22 20 16 19 17 21 19 25 23 25 25 25
5  18 19 15 15 21 25 16 16 23 15 22 17 19 22 24 ;
end;

```



Notas
