GLPK - Gnu Linear Programming Kit

Solver GLPK / Linguagem C

Rosklin Juliano Chagas

Pesquisa Operacional - Prof. Alexandre Salles da Cunha
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

13 de Novembro de 2012

GLPK - GNU Linear Programming Kit

- Pacote para resolução de problemas de programação linear e programação linear inteira mista.
- Fornece API e o solver glpsol.
- Livre sob licença GNU.

Instalação GLPK - Ubuntu 12.04

- *http://www.gnu.org/software/glpk/*.
- *http://www.cs.unb.ca/˜bremner/docs/glpk/glpk.pdf* (documentação)
- *http://ftp.gnu.org/gnu/glpk/* (download *glpk-4.47.tar.gz*)
- Comandos (download feito em .../glpk_install/):
  - *tar -xzvf arquivo.tar.gz*
  - *cd glpk-4.47/*
  - *./configure*
  - *make*
  - *sudo make install*
  - *export LD_LIBRARY_PATH="/usr/local/lib"* (sempre que abrir terminal)
  - *gcc -o cutting_stock.o cutting_stock.c -I/home/user_name/prog/glpk_install/glpk-4.47/src/ -lglpk*

Modelo de Programação Linear

$$
\begin{aligned}
\max\ z &= 10x^1 + 6x^2 + 4x^3 \\
\text{suj. a}\ & x^1 + x^2 + x^3 \leq 100 \\
& 10x^1 + 4x^2 + 5x^3 \leq 600 \\
& 2x^1 + 2x^2 + 6x^3 \leq 300 \\
& x \geq 0.
\end{aligned}
$$

Modelo de Programação Linear

$$\begin{aligned} \max\ z &= 10x^1 + 6x^2 + 4x^3 \\ \text{suj. a } p &= x^1 + x^2 + x^3 \\ q &= 10x^1 + 4x^2 + 5x^3 \\ r &= 2x^1 + 2x^2 + 6x^3 \end{aligned}$$

$$\begin{aligned} -\infty < p \le 100 \qquad\qquad & 0 \le x^1 < \infty \\ -\infty < q \le 600 \qquad\qquad & 0 \le x^2 < \infty \\ -\infty < r \le 300 \qquad\qquad & 0 \le x^3 < \infty \end{aligned}$$

Código-fonte em C

```
#include <glpk.h>
glp_prob *lp;
lp = glp_create_prob();
glp_set_prob_name(lp, "sample");
glp_set_obj_dir(lp, GLP_MAX);

glp_add_rows(lp, 3);
glp_set_row_name(lp, 1, "p");
glp_set_row_bnds(lp, 1, GLP_UP, 0.0, 100.0);

glp_add_cols(lp, 3);
glp_set_col_name(lp, 1, "x1");
glp_set_col_bnds(lp, 1, GLP_LO, 0.0, 0.0);
glp_set_obj_coef(lp, 1, 10.0);
```

Código-fonte em C

Matriz A (basta preencher as posicoes $A[i,j] \neq 0$):

```
ia[1] = 1, ja[1] = 1, ar[1] = 1.0; // a[1,1] = 1
ia[2] = 1, ja[2] = 2, ar[2] = 1.0; // a[1,2] = 1
...
ia[9] = 3, ja[9] = 3, ar[9] = 6.0; // a[3,3] = 6

glp_load_matrix(lp, 9, ia, ja, ar);
```

Código-fonte em C - Variáveis contínuas

```
// Resolve o problema continuo
glp_simplex(lp, NULL);
printf("Z_otimo = %5.2f.\n", glp_get_obj_val(lp));

// Mostra a solucao
total_colunas = glp_get_num_cols(lp);
for(i=1; i <= total_colunas; i++) {
   printf("%5.2f\n", glp_get_col_prim(lp, i));
}

// Coleta os valores das variaveis duais
total_linhas = glp_get_num_rows(lp);
for(i=1; i <= total_linhas; i++) {
   dual[i] = glp_get_row_dual(lp, i);
}
```

Código-fonte em C - Variáveis inteiras

Antes de resolver o problema:

```
// Variaveis binarias x_j
for (j=1;j<=total_colunas;j++) {
   glp_set_col_kind(lp, j, GLP_BV);
}

// Variaveis inteiras x_j
for (j=1;j<=total_colunas;j++) {
    glp_set_col_kind(lp, j, GLP_IV);
}
```

Código-fonte em C - Variáveis inteiras

```
// Resolve o problema
glp_simplex(lp, NULL);
glp_intopt(lp,NULL);


// Exibe os resultados
z = glp_mip_obj_val(lp);

xj = glp_mip_col_val(lp, j);    // j=1,...,n.
```

Geração de Colunas

$$\text{Incluir a coluna } A_{j=n+1} = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 5 \\ 0 \end{bmatrix}$$

```
glp_add_cols(lp, 1);
j = glp_get_num_cols(lp);
glp_set_col_bnds(lp, j, GLP_LO, 0, 0);
glp_set_obj_coef(lp, j, 1.0);

indice[1]=2; valor[1]=3;
indice[2]=4; valor[2]=5;

glp_set_mat_col(lp, j, 2, indice, valor);
```

Adição de Restrições

Construa uma cópia do problema *lp*:

```
glp_prob * lp2;
lp2 = glp_create_prob();
glp_copy_prob(lp2, lp, GLP_OFF);
```

Adicionar a restrição $x_j \geq \lceil x_j \rceil$:

```
glp_add_rows(lp2, 1);
i = glp_get_num_rows(lp2);
xj = glp_get_col_prim(lp2, j);
glp_set_row_bnds(lp2, i, GLP_LO, ceil(xj),0);

indice[1] = j;   valor[1] = 1;
glp_set_mat_row(lp2, i , 1, indice, valor);
// indice indica a coluna j.
```

Corte de Barras - Definição do Problema

Objetivo

Comprar uma quantidade mínima de barras de aço, todas com o mesmo comprimento W.

Restrição

Produzir b_i barras de comprimento w_i distintos. Assuma $w_i \leq W$ e w_i inteiro para cada $i = \{1, 2, ..., m\}$.

Versões do problema:

- Não inteira:
  - Geração de Colunas - versão não inteira.
- Inteira:
  - Modelo de Kantorovich
  - Branch-and-Price

Corte de Barras - Modelo de Kantorovich

$$\min \sum_{k=1}^{K} x_0^k \tag{1}$$

$$\sum_{k=1}^{K} x_i^k \geq b_i \qquad i = 1, ..., m. \tag{2}$$

$$\sum_{i=1}^{m} w_i x_i^k \leq W x_0^k, \qquad k = 1, ..., K. \tag{3}$$

$$x_0^k = 0 \text{ ou } 1, \qquad k = 1, ..., K. \tag{4}$$

$$x_i^k \geq 0 \text{ e inteiro}, \qquad i = 1, ..., m. \quad k = 1, ..., K. \tag{5}$$

onde:

- K - é um limite superior sobre o número de barras necessárias.
- x_0^k - é igual a 1 se a barra k é usada e, 0 caso contrário.
- x_i^k - é o número de vezes que o item i aparece no padrão de corte da barra k.

Corte de Barras - Geração de Colunas

Problema Original:

$$\min \sum_{j=1}^{n} x_j$$

$$\text{suj.} \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, ..., m,$$

$$x_j \geq 0 \quad j = 1, ..., n.$$

Coluna A_j com min $\overline{c}_j = 1 - p' A_j$:

$$\max \sum_{i=1}^{m} p_i a_i$$

$$\text{suj. a} \sum_{i=1}^{m} w_i a_i \leq W$$

$$a_i \geq 0 \text{ e inteiro.} \quad i = 1, ..., m.$$

Corte de Barras - Geração de Colunas

1. Construa uma versão restrita do problema com apenas m restrições e m variáveis.
2. Gere colunas até que a coluna de menor custo reduzido tenha $\overline{c}_j \geq 0$.
   1. Resolva o problema restrito (GLPK).
   2. Colete os valores das variáveis duais no problema mestre.
   3. Gere uma nova coluna j resolvendo o problema da mochila inteira (não é mochila 0/1).
   4. Se o custo reduzido $\overline{c}_j < 0$, adicione o novo padrão de corte Aj na versão restrita do problema.
3. Exiba a solução do problema.

Corte de Barras - Branch-and-Price

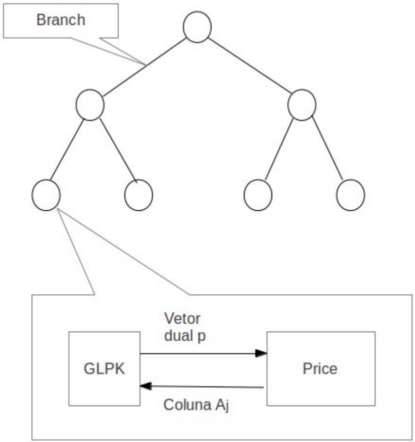

Figura 1 - Algoritmo Branch-and-Price.

Corte de Barras - Branch-and-Price

Algoritmo com uma única estratégia de poda:

1. Construa uma solução básica inicial viável.
2. Resolva o problema restrito utilizando geração de colunas.
3. Adicione o problema na lista L na posição $L[1]$.
4. Faça $atual = 1$ e $L.tam = 1$.
5. Enquanto $atual \leq L.tam$ faça:
    1. Se o problema $L[atual]$ é viável, realize os passos (2-3) a seguir: [1]
    2. Resolva o problema $L[atual]$ utilizando geração de colunas.
    3. Realize o *branch* para cada variável x_j fracionária:
        - ★ Adicione em $L[tam + 1]$ o modelo $L[atual] \cup \{x_j \leq \lfloor x_j \rfloor\}$
        - ★ Adicione em $L[tam + 2]$ o modelo $L[atual] \cup \{x_j \geq \lceil x_j \rceil\}$
        - ★ $tam = tam + 2$
    4. $atual = atual + 1$
6. Mostra a solução ótima inteira.

[1] $(glp\_get\_status(L[atual]) \neq GLP\_NOFEAS)$

Referências

- *http://www.ampl.com/BOOK/download.html*
- *http://www.gnu.org/software/glpk/*
- O problema da dieta pode ser encontrado em *http://www.ampl.com/BOOK/CHAPTERS/05-tut2.pdf*
- *http://www.cs.unb.ca/~bremner/docs/glpk/glpk.pdf* (documentação C/GLPK)
- Introdução aos Pacotes de Otimização. Dilson Lucas Pereira. DCC-UFMG.