Linköpings Tekniska Högskola
Matematiska institutionen/Optimeringslära
Kaj Holmberg

2020–09–11

Laborationsinformation

1 Information om GLPK/glpsol

1.1 Introduktion till GLPK

GLPK (GNU Linear Programming Kit) är avsett för lösning av stora linjärprogrammeringsproblem (LP) och blandade heltalsprogrammeringsproblem (MIP). GLPK kan använda GMPL (GNU MathProg language), som är ett modelleringsspråk. (GMPL är i det närmaste identiskt med AMPL.)

GLPK innehåller följande huvudkomponenter:
* Reviderad simplexmetod.
* Primal-dual inre-punktsmetod.
* Trädsökning.
* Översättare för GMPL.
* API (Application program interface).
* Fristående LP/MIP-lösare, glpsol.

Huvudtanken bakom glpsol och GMPL är att möjliggöra enklare modellering av optimeringsproblem. Modellen skrives på en fil och problemdata skrives på en annan fil.

1.2 Problemlösning med `glpsol`

Koden glpsol körs från ett terminalfönster, och löser problem som beskrivits i modelleringsspråket GMPL. Man anger problemfil och datafil med kommandoradsparametrar.

Exempel: Läs modellfil `myprob.mod`, datafil `myprob.dat`, lös problemet och skriv lösningen på fil `myprob.sol`.

```
glpsol -m myprob.mod -d myprob.dat -o myprob.sol
```

Exempel på parametrar som kan användas:

-m `filnamn`	Läs modell (och möjligtvis data) från fil `filnamn`.
-d `filnamn`	Läs data från fil `filnamn`.
-o `filnamn`	Skriv lösning på fil `filnamn` som vanlig text.
--ranges `filnamn`	Skriv resultat från känslighetsananlys på fil `filnamn`.
--wglp `filnamn`	Skriv problemet på `filnamn` i GLPK-format.
--log `filnamn`	Skriv en kopia av skärmutskrifterna på fil `filnamn`.
--interior	Använd en inre-punktsmetod, inte simplexmetoden.
--nomip	Betrakta alla heltalsvariabler som kontinuerliga (lös LP-relaxationen).
--cuts	Generera snitt (kräver --intopt).
--help	Ger hjälpinformation.

1.3 Ett litet exempel

$$
\begin{array}{lrcrcrcrcl}
\max & z = & 5x_1 & + & 6x_2 & + & 2x_3 & + & 4x_4 & \\
\text{då} & & 3x_1 & + & 2x_2 & + & x_3 & + & 5x_4 & \leq \ 80 \\
& & 2x_1 & + & x_2 & + & 2x_3 & + & 4x_4 & \leq \ 45 \\
& & 3x_1 & - & 3x_2 & + & 4x_3 & + & 5x_4 & \geq \ 80 \\
& & x_1 & , & x_2 & , & x_3 & , & x_4 & \geq \ 0
\end{array}
$$

Modellfil `typex2.mod`: (Se dokumentationen om GMPL.)

```
var x1 >=0;                                        # Variable definition
var x2 >=0;                                        # Variable definition
var x3 >=0;                                        # Variable definition
var x4 >=0;                                        # Variable definition
maximize z: 5*x1 + 6*x2 + 2*x3 + 4*x4;             # Objective function
subject to con1: 3*x1 + 2*x2 +   x3 + 5*x4 <= 80; # Constraint 1
subject to con2: 2*x1 +   x2 + 2*x3 + 4*x4 <= 45; # Constraint 2
subject to con3: 3*x1 - 3*x2 + 4*x3 + 5*x4 >= 80; # Constraint 3
end;
```

Skriv i terminalen:

```
glpsol -m typex2.mod -o typex2.sol
```

Följande utskrift fås på skärmen, när problemet löses.

```
GLPSOL: GLPK LP/MIP Solver, v4.47
Parameter(s) specified in the command line:
 -m typex2.mod -o typex2.sol
Reading model section from typex2.mod...
9 lines were read
Generating z...
Generating con1...
Generating con2...
Generating con3...
Model has been successfully generated
GLPK Simplex Optimizer, v4.47
4 rows, 4 columns, 16 non-zeros
Preprocessing...
3 rows, 4 columns, 12 non-zeros
Scaling...
 A: min|aij| =  1.000e+00  max|aij| =  5.000e+00  ratio =  5.000e+00
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part = 3
      0: obj =   0.000000000e+00  infeas =  8.000e+01 (0)
*     2: obj =   4.500000000e+01  infeas =  0.000e+00 (0)
*     3: obj =   7.500000000e+01  infeas =  0.000e+00 (0)
OPTIMAL SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (115476 bytes)
Writing basic solution to 'typex2.sol'...
```

Tips: Kolla alltid att det verkligen står "OPTIMAL SOLUTION FOUND", och t.ex. inte "PROBLEM HAS NO PRIMAL FEASIBLE SOLUTION", som betyder att problemet saknar tillåten lösning.

Efter detta kommer filen `typex2.sol` att innehålla bl.a. följande.

```
Problem:    typex2
Rows:       4
Columns:    4
Non-zeros:  16
Status:     OPTIMAL
Objective:  z = 75 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 z            B             75
     2 con1         B           42.5                        80
     3 con2         NU            45                        45             7
     4 con3         NL            80            80                        -3

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 x1           B             10             0
     2 x2           NL             0             0                       -10
     3 x3           B           12.5             0
     4 x4           NL             0             0                        -9
```

Vi ser att x_1 och x_3 är basvariabler i optimum, och att lösningen är $x_1 = 10$, $x_2 = 0$, $x_3 = 12.5$, $x_4 = 0$, med målfunktionsvärde 75. Skuggpriserna är 0, 7 och -3 för de tre bivillkoren.

Utskriften från `glpsol` när man löser ett MIP-problem innehåller
- antal iterationer som gjorts med simplexmetoden;
- målfunktionsvärdet för den bästa kända tillåtna heltalslösningen, vilket är en övre (minimering) eller undre (maximering) global gräns för det optimala målfunktionsvärdet;
- den bästa lokala gränsen för aktiva noder, vilket är en undre (minimering) eller övre (maximering) global gräns för det optimala målfunktionsvärdet;
- det relativa MIP-gapet, i procent;
- antalet öppna (aktiva) subproblem;
- antalet avsökta subproblem.

```
Solving LP relaxation...
GLPK Simplex Optimizer, v4.47
3 rows, 4 columns, 12 non-zeros
      0: obj =   0.000000000e+00  infeas =  8.000e+01 (0)
*     3: obj =   4.500000000e+01  infeas =  0.000e+00 (0)
*     4: obj =   7.500000000e+01  infeas =  0.000e+00 (0)
OPTIMAL SOLUTION FOUND
Integer optimization begins...
+     4: mip =     not found yet <=              +inf        (1; 0)
+     9: >>>>>   6.800000000e+01 <=   6.800000000e+01   0.0% (4; 0)
+     9: mip =   6.800000000e+01 <=     tree is empty   0.0% (0; 7)
INTEGER OPTIMAL SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (127461 bytes)
```

I detta exempel (det tidigare exemplet men med heltalskrav) tar det 4 simplexiterationer för att lösa den första LP-relaxationen. Målfunktionsvärdet av denna LP-lösning är 75. Det tar totalt 9 simplexiterationer att lösa heltalsproblemet. Det optimala målfunktionsvärdet

är 68. Det krävdes undersökning av 7 trädsökningsnoder för att hitta optimallösningen (och bevisa optimalitet).

2 Modelleringsspråket GMPL/AMPL

GMPL är ett modelleringsspråk där man enkelt kan modellera optimeringsproblem med hjälp av mängder, summor mm. Här ges en kort och förenklad beskrivning av språket. (Modell- och datafiler som beskrives här kan användas både av GMPL och AMPL.)

Varje variabel har ett namn (x1, x2 etc), målfunktionen har ett namn (z) och varje villkor har ett namn (con1, con2 osv). Namnen är valfria, men måste vara olika. Kommentarer i modellen görs efter tecknet `#`. Varje rad måste avslutas med semikolon (`;`).

Om man ska lösa samma modell med flera olika uppsättningar data är det bra att skilja på modell och data. Man definierar parametrar och mängder som kan användas i summor etc. Vi utgår från följande generella formulering.

$$\begin{aligned} \max z \quad &= \sum_{j=1}^{n} c_j x_j \\ \text{då} \quad & \sum_{j=1}^{n} a_{ij} x_j \quad \leq \quad b_i \quad i = 1, \ldots, m \\ & x_j \quad \geq \quad 0 \quad j = 1, \ldots, n \end{aligned}$$

Modellfilen för GMPL blir som följer. Man använder införda parametrar, trots att de inte fått numeriska värden än.

```
param n;                    # Antal variabler
param m;                    # Antal villkor
param c{1..n};              # Definition av målfunktionsvektor
param a{1..m,1..n};         # Definition av bivillkorsmatris
param b{1..m};              # Definition av högerledsvektor

var x{1..n} >= 0;           # Variabeldefinition

maximize z: sum{j in 1..n} c[j]*x[j];
subject to con{i in 1..m}: sum{j in 1..n} a[i,j]*x[j] <= b[i];
end;
```

`n, m, c, a` och `b` är parametrar (`param`) och `x` är variabler (`var`). `n` anger antalet variabler och `m` antalet villkor. Då blir `1..n` mängden av alla variabelindex och `1..m` mängden av alla bivillkorsindex. `sum {j in 1..n}` summerar således över alla variabler x_j.

Datafilen för detta problem ser ut på följande sätt. Här anges värden på alla parametrar som har deklarerats i modellfilen.

```
param n := 4;      #antal variabler
param m := 3;      #antal villkor

param : c :=
   1    5
   2    6
```

```
   3    2
   4    4;

param a :   1  2  3  4 :=
    1       3  2  1  5
    2       2  1  2  4
    3       3 -3  4  5;

param : b :=
    1   80
    2   45
    3   80;
end;
```

Observera att man måste ge index för vektorer (`c` och `b`) och både rad- och kolumnindex för matriser (`a`).

Nedan beskrivs några av de viktigare kommandona i GMPL. Notera att GMPL skiljer på stora och små bokstäver, och att GMPL inte bryr sig om radbrytningar, utan varje kommando eller deklaration måste avslutas med ; (semikolon).

param

Konstanter deklareras med *param*. Tilldelning av värde till en parameter görs på följande sätt.

```
param n := 7;
param : vek := 1 50  2 75  3 100;
param : vek2 := 1 34  2 105  3 65 4 120;
param matr : 1   2   3 : =
         1  80   9  77
         2  11 120  13;
```

Parametern `n` får värdet 7, vektorn `vek` tilldelas värden `vek[1]=50`, `vek[2]=75` och `vek[3]=100` (observera att varannan siffra är index) och matrisen `matr` tilldelas värden `matr[1,1]=80`, `matr[1,2]=9`, `matr[1,3]=77`, `matr[2,1]=11` osv.

set

En mängd deklareras med *set*. Den kan innehålla både numeriska värden och symboliska värden (dvs. ord).

```
set CARS := SAAB VOLVO BMW;
set UddaNR := 1 3 5 7 9;
set VECKOR := 1..N;
```

var

Alla variabler måste deklareras med *var*. Variabelnamnet är valfritt.

```
var x{1..n}>=0;      # Ickenegativ variabelvektor x[i], i=1..n
var x{1..8,1..20};   # Variabelmatris x[i,j]
var y{CARS} binary;  # Binär variabelvektor y[i] för alla i ur mängden CARS
var y{1..n}>=0,<=4;  # Variabelvektor y[i], där 0<=y[i]<=4, för i=1..n
var w{1..m} >=0, integer; # Ickenegativ heltalsvariabelvektor
```

sum

En summa i målfunktion och bivillkor över ett antal variabler definieras av *sum*.

```
sum{i in Cars} Weight[i]          # Summera Weigth över i ur mängden Cars
sum{i in 1..20} (x[i] - y[i])     # Summera x[i]-y[i] för i=1 till 20
sum{i in ORIG, j in DEST} z[i,j] # Summera över två index
```

maximize / minimize

Specificerar målfunktionen (som måste ges ett namn). Efter namnet följer ett : (kolon) och därefter målfunktionen.

```
maximize profit: sum{i in Units} c[i] * x[i];
```

subject to

Definitionen *subject to* anger att det som följer på raden är ett bivillkor. Bivillkoren måste ges olika namn. Efter namnet följer : (kolon) och därefter bivillkoret.

```
subject to bivillkor1: x + y = 7;

subject to lager{i in Produkter}:
    Tillverkat[i] + Kvar[i] - Sålt[i] <= Lagerkap[i];

subject to biv3{i in 1..n}:
    sum{j in 1..i} x[j] >= d[i];
```

solve

Anger att problemet ska lösas.

display

Skriver ut saker, t.ex. lösningen.

```
display x;               # Skriv vektor x på skärmen
display x > file.res;    # Skriv vektor x på filen 'file.res'
display cons1.dual;      # Skriv ut dualvariablerna för bivillkoren 'cons1'
display cons1.slack;     # Skriv ut slacket i bivillkoren 'cons1'
display x.rc;            # Skriv ut reducerade kostnader för variablerna x
```

printf

Mer kontrollerad utskrift, t.ex. för att undvika variabler som är noll.

```
printf{i in 1..8, j in 1..8: x[i,j]>0}: "Pos (%d,%d)\n",i, j;
printf{j in SET: x[j]>0} " x(%d)=%.2f\n",j,x[j];
printf "Optvariabler:"; printf{j in 1..n: x[j]>0} " %d",j; printf "\n";
```

end

Här är en skillnad mellan GMPL, som kräver att modelldelen och datadelen avslutas med *end*, och AMPL, som inte bryr sig.