Linköping Institute of Technology
Department of Mathematics/Division of Optimization
Kaj Holmberg

2018–09–10

Lab Information

1 Information about GLPK/glpsol

1.1 Introduction to GLPK

The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP) and mixed integer programming (MIP). GLPK supports GMPL (GNU MathProg Language), which is a modeling language (almost identical to AMPL).

The GLPK package containts the following main components:
* Revised simplex method.
* Primal-dual interior point method.
* Branch-and-bound method.
* Translator for GNU MathProg.
* API (Application program interface)
* Stand-alone LP/MIP solver, glpsol.

The main idea of glpsol and the GNU MathProg language is to ease modeling of optimization problems. The model is written on a file, and data for the problem is written on another file.

1.2 Solving models with the solver `glpsol`

The glpsol code is run in a terminal window, and solves a problem written in the modeling language GMPL. The problem file and the data file are given by command line parameters.

Example: Read model file `myprob.mod`, data file `myprob.dat`, solve the problem and write the solution to file `myprob.sol`.

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

Examples of parameters which could be used:

-m `filename`	Read model (and possibly data) from file `filename`.
-d `filename`	Read data from file `filename`.
-o `filename`	Write solution to file `filename` as standard text.
--ranges `filename`	Write result of sensibility analysis to file `filename`.
--wglp `filename`	Write problem to `filename` in GLPK format.
--log `filename`	Write a copy of log information on the screen to file `filename`.
--interior	Use the interior-point method, not the simplex method.
--nomip	Consider all integer variables as continuous (solve LP relaxation).
--cuts	Generate cuts (require --intopt).
--help	Get help information.

1.3 A small example

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

Model file `typex2.mod`: (See the documentation about 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;
```

Write in the terminal window:

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

The following output appears on the screen, when solving the problem.

```
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'...
```

Hint: Always check whether is says "OPTIMAL SOLUTION FOUND", and not "PROBLEM HAS NO PRIMAL FEASIBLE SOLUTION", which means that the problem has no feasible solution, and the output is rather meaningless.

Afterwards, the file `typex2.sol` will contain the following.

```
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
```

Here we see that x_1 and x_3 are basic variables at the optimum, and that the solution is $x_1 = 10$, $x_2 = 0$, $x_3 = 12.5$, $x_4 = 0$, with the objective function value equal to 75. The shadow prices are 0, 7 and -3 for the three constraints.

When one solves a MIP problem, the output from `glpsol` contains
- number of iterations by the simplex method;
- value of the objective function for the best known feasible integer solution, which is an upper (minimization) or lower (maximization) - global bound for the optimal value of the objective function;
the best local bound for active nodes, which is a lower (minimization) or upper (maximization) global bound for the optimal value of the objective function;
- the relative MIP gap, in percentage;
- number of the open (active) subproblems;
- number of explored subproblems.

```
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)
```

4 simplex iterations were needed for solving the first LP relaxation in this example (the previous example but with integer variables). The objective function value of this LP

relaxation is 75. It takes 9 simplex iterations in total to solve the integer programming problem. The optimal objective function value is 68. It requires 7 nodes in the search tree to find the optimal solution (and validate optimality).

2 The GMPL/AMPL modeling language

Introduction

GMPL is a modeling language which allows for simplifying modeling an optimization problem with help of sets, sums etc. Here is a short and simplified description of the language. (Model and data files described below can be used both for GMPL and AMPL.)

Each variable has a name (x1, x2 etc), the objective function has a name (z) and each constraint has a name (con1, con2 etc). It is free to choose names, but they must be different. To write comments to the model, use first the sign `#`. Each row should end with semicolon (`;`).

For solving the same model with several different data, it is good to separate model and data. One can define parameters and sets used in sums etc. Consider the following general formulation.

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

Then model file for GMPL is created as follows. We use introduced parameters, although they do not have numerical values yet.

```
param n;                    # number of variables
param m;                    # number of constraints
param c{1..n};              # Definition of objective function coefficients
param a{1..m,1..m};         # Definition of constraint matrix
param b{1..m};              # Definition of right-hand-sides

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

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;
```

Here `n, m, c, a` and `b` are parameters (`param`) and `x` are variables (`var`). `n` is the number of variables and `m` is the number of constraints. Then `1..n` is the set of indices of all variables and `1..m` is the set of indices of all constraints. `sum {j in 1..n}` is a sum over the set of indices of all variables.

The file with numerical data will be as follows. Here the values for all parameters which were declared in the model file must be assigned.

```
param n := 4;      #number of variables
param m := 3;      #number of constraints

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;
```

Observe that it is required to provide indices for vectors (`c` and `b`), and both row indices and column indices for matrices (`a`).

2.1 Syntax

Below we describe some of the most common commands in GMPL. Note that GMPL distinguishes between uppercase and lowercase letters, and GMPL does not break rows, therefore each command or declaration must end with ; (semicolon).

param

Constants are declared with *param*. Assignment of values for each parameter is done as follows.

```
param n := 7;
param : vec := 1 50  2 75  3 100;
param : vec2 := 1 34  2 105  3 65 4 120;
param matr : 1   2   3 : =
         1  80   9  77
         2  11 120  13;
```

The parameter `n` gets value 7, the vector `vek` is assigned values `vec[1]=50`, `vec[2]=75` and `vec[3]=100` (observe that every second number is an index) and the matrix `matr` is assigned values `matr[1,1]=80`, `matr[1,2]=9`, `matr[1,3]=77`, `matr[2,1]=11` etc.

set

A set is declared by *set*. It may contain both numerical and symbolic values.

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

var

All variables must be declared with *var*. You can choose any names for the variables.

```
var x{1..n} >=0;          # Vector of non-negative variables x[i], i=1..n
var x{1..8,1..20};        # Variable matrix x[i,j]
var y{CARS} binary;       # Vector of binary variables y[i] for each i from the set CARS
var y{1..n} >=0,<=4;      # Vector of variables y[i], where 0<=y[i]<=4, for i=1..n
var w{1..m} >=0, integer; # Vector of non-negative integer variables
```

sum

A sum over several variables in objective function and constraints is declared by *sum*.

```
sum{i in Cars} Weight[i]          # Sum Weight over all elements in the set Cars
sum{i in 1..20} (x[i] - y[i])     # Summ x[i]-y[i] for i=1 to 20
sum{i in ORIG, j in DEST} z[i,j]  # Sum over two indices
```

maximize / minimize

Specifies the objective function (which must be given a name). The name is followed by : (colon) and then the objective function.

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

subject to

Declares one or a set of constraints. The constraints must have different names. The name is followed by : (colon) and then the constraint.

```
subject to constraint1: x + y = 7;

subject to storage{i in Produkter}:
    produced[i] + left[i] - sold[i] <= storagecap[i];

subject to con3{i in 1..n}:
    sum{j in 1..i} x[j] >= d[i];
```

solve

Declares that the problem should be solved.

display

Displays various information, e.g. the solution

```
display x;               # Display vector x on the screen
display x > file.res;    # Write vector x to the file 'file.res'
display cons1.dual;      # Display dual variables for contraint 'cons1'
display cons1.slack;     # Display slack variables for constraint 'cons1'
display x.rc;            # Display reduced  costs for variables x
```

printf

More controlled output, which allows e.g. to avoid displaying zero variables.

```
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

GMPL requires that both model and data files end with *end*. (AMPL does not require this, but allows it.)