Jumpstart GLPK
Bo Waggoner
Updated: 2014-02-16

Abstract

The GNU Linear Programming Kit (GLPK) solves LPs and mixed integer programs either as a standalone solver or as a library callable from Python, C, and other languages. Here we focus on the standalone solver, **glpsol**, and the associated modeling language, which is very similar to AMPL. I assume familiarity with Linux command line or ability to translate into your own operating system.

1 Installation

You can install from the GLPK homepage: `http://www.gnu.org/software/glpk/`.
Or on Linux, install using

```
$ apt-get install glpk
```

(substituting the appropriate package manager for apt-get).
Note: the documentation is included with the download; on Linux, find it at /usr/share/doc/glpk-doc/gmpl.pdf.

2 Example 0

Name this file example0.mod:

```
# example0.mod
/* this is a multi-
   line comment */

param n := 10;
set N := 1..n;  # N is a set consisting of numbers 1,2,...,n
var x{i in N};  # x is a choice variable indexed by elements of N

maximize name_of_objective: sum{i in N} x[i];

s.t.
  name_of_constraint_1:          sum{i in N} x[i] <= 7;
  name_of_constraint_2{i in N}: x[i] <= 0.5;    # constraint for all i in N

data;  # this program has no data

end;
```

To solve this LP with glpsol, we use the -m (or –model) flag to specify that the input language is the GNU MathProg language and the -o flag to specify an output file into which we write results. From the command line, run

```
$ glpsol -m example0.mod -o out.txt
```

3 Example 1

Name this file example1.mod:

```
# example1.mod
# shortest path on unweighted graph

set NODES;
param adj_matrix{u in NODES, v in NODES};  # 2-d array
param s, symbolic, in NODES;  # start node. symbolic means not necessarily a number
param t, symbolic, in NODES;  # end node
var use_edge{u in NODES, v in NODES}, binary;
/* could also have declared:
   use_edge{u in NODES, v in NODES}, >= 0, <= 1; */

minimize path_length: sum{u in NODES, v in NODES} use_edge[u,v];

s.t. start_at_s: sum{v in NODES} use_edge[s,v] = 1 + sum{u in NODES} use_edge[u,s];
s.t. end_at_t:   sum{u in NODES} use_edge[u,t] = 1 + sum{v in NODES} use_edge[t,v];
s.t. through_others{i in NODES: i != s and i != t}:
                 sum{u in NODES} use_edge[u,i] = sum{v in NODES} use_edge[i,v];
s.t. legal_paths{u in NODES,v in NODES}: use_edge[u,v] <= adj_matrix[u,v];

data;

set NODES := A,B,C,D,E;
param adj_matrix :
   A B C D E :=
 A 0 1 0 1 0
 B 1 0 1 1 0
 C 1 1 0 0 1
 D 0 1 0 1 0
 E 0 1 1 0 0;

param s := A;
param t := E;

end;
```

4 The Output File

Running from the command line:

```
 $ glpsol -m example1.mod -o out.txt
```

produces an output file that looks like:

```
Problem:    example1
Rows:       31
Columns:    25 (25 integer, 25 binary)
Non-zeros:  90
Status:     INTEGER OPTIMAL
```

```
Objective:  path_length = 3 (MINimum)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 path_length                3
     2 start_at_s                 1             1             =
     3 end_at_t                   1             1             =
     4 through_others[B]
                                  0            -0             =
     5 through_others[C]
                                  0            -0             =
     6 through_others[D]
                                  0            -0             =
     7 legal_paths[A,A]
                                  0                          -0
[...]

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 use_edge[A,A]
                     *            0             0             1
     2 use_edge[A,B]
                     *            1             0             1
[...]
     8 use_edge[B,C]
                     *            1             0             1
[...]
    15 use_edge[C,E]
                     *            1             0             1
[...]

Integer feasibility conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality
```

Note that I've omitted some lines describing boring constraints and variables. The key information is the value of the objective: (6th line), and the settings of the variables. In particular, in our example we see that use_edge[A,B], use_edge[B,C], and use_edge[C,E] are one and all other instances of use_edge are zero, giving us our shortest path.