Introduction to glpkAPI

Louis Luangkesorn *

November 13, 2026

1 Introduction

This document introduces the use of the glpkAPI package[1] for R. The GNU Linear Programming Package (GLPK) is intended for solving linear programming (LP) and mixed integer programming (MIP) and other related problems. In addition, it includes facilities for converting problem information between the GNU MathProg language (a subset of the AMPL mathematical programming language), free and fixed MPS, and the CPLEX LP formats.[2] The GLPK package is an interface into the C Application Programming Interface (API) to the GLPK solver.

This document will introduce the use of the GLPK package through the use of the cannery problem from Dantzig[3] which is used in the GNU MathProg documentation.[4] The model file describing the cannery problem can be found in Appendix A.

2 Entering the model

To use `glpk`, first load the package.

```
> library(glpkAPI)
```

Next read in the model and data.

There are several ways of entering the model. `glpk` can read the model and data in a GNU MathProg Language (GMPL) model file. Alternatively, the model and data can be entered using the GLPK API.

*lugerpitt@gmail.com. Thanks to Leo Lopes for his comments and suggestions.

[1] Package glpkAPI maintained by Gabriel Gelius-Dietrich

[2] GNU Linear Programming Kit: Reference Manual, Version 4.54 Draft, March 2014.

[3] The demand data here is from the GLPK documentation, which differs slightly from Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, NJ, 1963. The documentation demand values are used here for consistancy.

[4] GNU Linear Programming Kit: Modeling Language GNU MathProg, Version 4.50 Draft, May 2013.

2.1 Reading a GNU MathProg Language model

To use a GNU MathProg model requires several steps.

1. Allocating the workspace using `initProbGLPK()`. The problem can then be given an name using `setProbNameGLPK()`.
2. Reading model section using `mplAllocWkspGLPK()` and `mplReadModelGLPK()`.
3. Reading data section(s) using `mplReadDataGLPK()`.
4. Generating the model using `mplGenerateGLPK()`.
5. Building the problem object using `result <- mplBuildProbGLPK()`.
6. Solving the problem using `solveSimplexGLPK()`.
7. Postsolving the model using `mplPostsolveGLPK()`.
8. Freeing the workspace using `mplFreeWkspGLPK()` and `delProbGLPK()`

```
> mip <- initProbGLPK()
> setProbNameGLPK(mip, "transport")
> trans <- mplAllocWkspGLPK()
> result <- mplReadModelGLPK(trans,
+         system.file("extdata", "transport.mod", package = "glpkAPI"), skip=0)
> result <- mplGenerateGLPK(trans)
> result <- mplBuildProbGLPK(trans, mip)
```

If the data was in a separate file, it would need to be read in using

```
mplReadDataGLPK(trans, "transport.mod")
```

Then examine the problem size within R.

The rows represent the objective function as well as the supply and demand constraints.

```
> numrows <- getNumRowsGLPK(mip)
> numrows

[1] 6

> for (i in 1:numrows){
+         print(getRowNameGLPK(mip, i))
+ }

[1] "cost"
[1] "supply[Seattle]"
[1] "supply[San-Diego]"
[1] "demand[New-York]"
[1] "demand[Chicago]"
[1] "demand[Topeka]"
```

The columns represent the decision variables, which are the units sent over the cannary-market links.

```
> numcols <- getNumColsGLPK(mip)
> numcols

[1] 6

> for (j in 1:numcols){
+         print(getColNameGLPK(mip, j))
+ }

[1] "x[Seattle,New-York]"
[1] "x[Seattle,Chicago]"
[1] "x[Seattle,Topeka]"
[1] "x[San-Diego,New-York]"
[1] "x[San-Diego,Chicago]"
[1] "x[San-Diego,Topeka]"

> print(getNumNnzGLPK(mip))

[1] 18
```

After the model and data are entered, the model can then be solved using any one of many algorithms and the output would go to the specified output file. For the Simplex method, the `solveSimplexGLPK()` takes the problem name and solves it using the Simplex method.

```
> return <- solveSimplexGLPK(mip)
> return <- mplPostsolveGLPK(trans, mip, GLP_MIP);
```

We can then look at the solution in terms of the objective and constraints

```
> for (i in 1:numrows){
+ print(getRowNameGLPK(mip, i))
+ print(getRowPrimGLPK(mip, i))
+ }

[1] "cost"
[1] 153.675
[1] "supply[Seattle]"
[1] 350
[1] "supply[San-Diego]"
[1] 550
[1] "demand[New-York]"
[1] 325
[1] "demand[Chicago]"
[1] 300
[1] "demand[Topeka]"
[1] 275
```

as well as the decision variables.

```
> for (j in 1:numcols){
+ print(getColNameGLPK(mip, j))
+ print(getColPrimGLPK(mip, j))
+ }
```

```
[1] "x[Seattle,New-York]"
[1] 50
[1] "x[Seattle,Chicago]"
[1] 300
[1] "x[Seattle,Topeka]"
[1] 0
[1] "x[San-Diego,New-York]"
[1] 275
[1] "x[San-Diego,Chicago]"
[1] 0
[1] "x[San-Diego,Topeka]"
[1] 275
```

Finally, clean up the workspace.

```
> mplFreeWkspGLPK(trans)
> delProbGLPK(mip)
```

2.2 Using the API

If the problem data already in *R*, such as pulled from a database or the result of previous analysis, the model and the data can be specified using the API.

First create R data objects to hold the various model parameters.

```
> print ("USING API")
```

```
[1] "USING API"
```

```
> canneries <- c("Seattle", "San-Diego")
> capacity <- c(350, 600)
> markets <- c("New-York", "Chicago", "Topeka")
> demand <- c(325, 300, 275)
> distance <- c(2.5, 2.5, 1.7, 1.8, 1.8, 1.4)
> dim(distance) <- c(2, 3)
> freight <- 90
```

To use the API, define a problem instance and indicate that the objective is to minimize cost.

```
> lpi <- initProbGLPK()
> setProbNameGLPK(lpi, "cannery API")
> setObjNameGLPK(lpi, "Total Cost")
> setObjDirGLPK(lpi, GLP_MIN)
```

There are 6 columns, corresponding to the six potential cannery-market pairs whose transport the model solving for, each of which has a lower bound of zero.

```
> numlinks <- length(distance)
> nummarkets <- length(markets)
> numcanneries <- length(canneries)
> addColsGLPK(lpi, numlinks)

[1] 1

> for (i in 1:numcanneries){
+         cannerystartrow <- (i-1) * nummarkets
+         for (j in 1:nummarkets){
+                 colname <-toString(c(canneries[i], markets[j]))
+                 transcost <- distance[i, j]*freight/1000
+                 setColNameGLPK(lpi, cannerystartrow+j, colname)
+                 setColBndGLPK(lpi, cannerystartrow+j, GLP_LO, 0.0, 0.0)
+                 setObjCoefsGLPK(lpi, cannerystartrow+j, transcost)
+         }
+ }
```

Next, we will add constraints. There are 5 constraints, two supply constraints relating to the canneries and three demand constraints relating to the markets. In addition, we will make the first row correspond to the objective function. The objective row will be free, and does not have upper or lower bounds.

```
> numcanneries <- length(canneries)
> nummarkets <- length(markets)
> addRowsGLPK(lpi, numcanneries+nummarkets+1)

[1] 1

> setRowsNamesGLPK(lpi, 1, getObjNameGLPK(lpi))
> for (i in 1:numcanneries){
+         setRowsNamesGLPK(lpi, i+1, toString(c("Supply", canneries[i])))
+         setRowBndGLPK(lpi, i+1, GLP_UP, 0, capacity[i])
+ }
> for (j in 1:nummarkets){
+         setRowsNamesGLPK(lpi, numcanneries+j+1, toString(c("Demand", markets[j])))
+         setRowBndGLPK(lpi, numcanneries+j+1, GLP_LO, demand[j], 0)
+ }
```

Now, load the constraint matrix which represents the objective function and the constraints. The non-zero values of the matrix are entered as three vectors, each with one element for each non-zero value. A vector to indicate the row, a vector to indicate the column, and a vector which contains the matrix element value. Last, we call `loadMatrixGLPK(lpi)` to finish.

```
> # create variables to hold the constraint information
> ia <- numeric()
> ja <- numeric()
> ar <- numeric()
> # add in objective coefficients
>
> for (i in 1:numcols){
+         ia[i] <- 1
+         ja[i] <- i
+         ar[i] <- getObjCoefGLPK(lpi, i)
+ }
> for (i in 1:numcanneries){
+         #supply constraints
+         cannerysupplyrow = numcols + (i-1)*nummarkets
+         for (j in 1:nummarkets){
+                 ia[cannerysupplyrow+j] <- (i+1)
+                 ja[cannerysupplyrow+j] <- (i-1)+numcanneries *(j-1)+1
+                 ar[cannerysupplyrow+j] <- 1
+         }
+         #demand constraints
+         marketdemandrow = numcols+numcanneries * nummarkets
+         for (j in 1:nummarkets){
+                 colnum <- (i-1)*nummarkets+j
+                 ia[marketdemandrow + colnum] <- numcanneries+j+1
+                 ja[marketdemandrow + colnum] <- colnum
+                 ar[marketdemandrow + colnum] <- 1
+         }
+ }
> loadMatrixGLPK(lpi, length(ia), ia, ja, ar)
```

Then, examine the problem entered in the API.

```
> numrows <- getNumRowsGLPK(lpi)
> numrows

[1] 6

> numcols <- getNumColsGLPK(lpi)
> numcols

[1] 6

> for (i in 1:numrows){
+         print(getRowNameGLPK(lpi, i))
+ }

[1] "Total Cost"
[1] "Supply, Seattle"
```

```
[1] "Supply, San-Diego"
[1] "Demand, New-York"
[1] "Demand, Chicago"
[1] "Demand, Topeka"
```

```
> for (j in 1:numcols){
+         print(getColNameGLPK(lpi, j))
+ }
```

```
[1] "Seattle, New-York"
[1] "Seattle, Chicago"
[1] "Seattle, Topeka"
[1] "San-Diego, New-York"
[1] "San-Diego, Chicago"
[1] "San-Diego, Topeka"
```

```
> print(getNumNnzGLPK(lpi))
```

```
[1] 18
```

Finally solve using the simplex method and look at the solution.

```
> solveSimplexGLPK(lpi)
```

```
[1] 0
```

```
> for (i in 1:numrows){
+ print(getRowNameGLPK(lpi, i))
+ print(getRowPrimGLPK(lpi, i))
+ }
```

```
[1] "Total Cost"
[1] 153.675
[1] "Supply, Seattle"
[1] 325
[1] "Supply, San-Diego"
[1] 575
[1] "Demand, New-York"
[1] 325
[1] "Demand, Chicago"
[1] 300
[1] "Demand, Topeka"
[1] 275
```

```
> for (j in 1:numcols){
+ print(getColNameGLPK(lpi, j))
+ print(getColPrimGLPK(lpi, j))
+ }
```

```
[1] "Seattle, New-York"
[1] 325
[1] "Seattle, Chicago"
[1] 300
[1] "Seattle, Topeka"
[1] 0
[1] "San-Diego, New-York"
[1] 0
[1] "San-Diego, Chicago"
[1] 0
[1] "San-Diego, Topeka"
[1] 275
```

And save the results to a file.

```
> printSolGLPK(lpi, "transout.api")

[1] 0
```

2.3 Using API to modify the model

Now, we will solve the version of the problem that is found in Dantzig. The demand at New York and Topeka are both 300 instead of 325 and 275. This next section will use the API to modify the problem as read through the MathProg file.

In order to examine an individual row, we need to index the rows and columns. This is done through the use of `createIndexGLPK()`. Then we can use the `findRowGLPK()` and `findColGLPK()`

```
> cindex <- createIndexGLPK(lpi)
> new_york_row = findRowGLPK(lpi, "Demand, New-York")
> topeka_row = findRowGLPK(lpi, "Demand, Topeka")
> new_york_row

[1] 4

> topeka_row

[1] 6

> setRowBndGLPK(lpi, new_york_row, GLP_LO, 300, 0)
> setRowBndGLPK(lpi, topeka_row, GLP_LO, 300, 0)
```

We can solve this modified problem and look at the results.

```
> solveSimplexGLPK(lpi)

[1] 0
```

```
> for (i in 1:numrows){
+ print(getRowNameGLPK(lpi, i))
+ print(getRowPrimGLPK(lpi, i))
+ print(getRowDualGLPK(lpi, i))
+ }

[1] "Total Cost"
[1] 151.2
[1] 0
[1] "Supply, Seattle"
[1] 300
[1] 0
[1] "Supply, San-Diego"
[1] 600
[1] 0
[1] "Demand, New-York"
[1] 300
[1] 0.225
[1] "Demand, Chicago"
[1] 300
[1] 0.153
[1] "Demand, Topeka"
[1] 300
[1] 0.126

> for (j in 1:numcols){
+ print(getColNameGLPK(lpi, j))
+ print(getColPrimGLPK(lpi, j))
+ print(getColDualGLPK(lpi,j))
+ print(getObjCoefGLPK(lpi, j))
+ }

[1] "Seattle, New-York"
[1] 300
[1] 0
[1] 0.225
[1] "Seattle, Chicago"
[1] 300
[1] 0
[1] 0.153
[1] "Seattle, Topeka"
[1] 0
[1] 0.036
[1] 0.162
[1] "San-Diego, New-York"
[1] 0
[1] 0
```

```
[1] 0.225
[1] "San-Diego, Chicago"
[1] 0
[1] 0.009
[1] 0.162
[1] "San-Diego, Topeka"
[1] 300
[1] 0
[1] 0.126
```

Finally, clean up the workspace.

```
> delProbGLPK(lpi)
```

A Model file

TRANSPORT.MOD

```
# A TRANSPORTATION PROBLEM
#
# This problem finds a least cost shipping schedule that meets
# requirements at markets and supplies at factories.
#
#  References:
#              Dantzig, G B., Linear Programming and Extensions
#              Princeton University Press, Princeton, New Jersey, 1963,
#              Chapter 3-3.

set I;
/* canning plants */

set J;
/* markets */

param a{i in I};
/* capacity of plant i in cases */

param b{j in J};
/* demand at market j in cases */

param d{i in I, j in J};
/* distance in thousands of miles */

param f;
/* freight in dollars per case per thousand miles */
```

```
param c{i in I, j in J} := f * d[i,j] / 1000;
/* transport cost in thousands of dollars per case */

var x{i in I, j in J} >= 0;
/* shipment quantities in cases */

minimize cost: sum{i in I, j in J} c[i,j] * x[i,j];
/* total transportation costs in thousands of dollars */

s.t. supply{i in I}: sum{j in J} x[i,j] <= a[i];
/* observe supply limit at plant i */

s.t. demand{j in J}: sum{i in I} x[i,j] >= b[j];
/* satisfy demand at market j */

data;

set I := Seattle San-Diego;

set J := New-York Chicago Topeka;
param a := Seattle     350
           San-Diego   600;

param b := New-York    325
           Chicago     300
           Topeka      275;

param d :              New-York   Chicago   Topeka :=
           Seattle     2.5        1.7       1.8
           San-Diego   2.5        1.8       1.4  ;

param f := 90;

end;
```

B Output

The following is the output of the command: `printSolGLPK(lpi, "transout.api")`

```
Problem:    cannery API
Rows:       6
Columns:    6
Non-zeros:  18
Status:     OPTIMAL
Objective:  Total Cost = 153.675 (MINimum)
```

```
   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 Total Cost   B        153.675
     2 Supply, Seattle
                    B            325                         350
     3 Supply, San-Diego
                    B            575                         600
     4 Demand, New-York
                    NL           325           325                       0.225
     5 Demand, Chicago
                    NL           300           300                       0.153
     6 Demand, Topeka
                    NL           275           275                       0.126

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 Seattle, New-York
                    B            325             0
     2 Seattle, Chicago
                    B            300             0
     3 Seattle, Topeka
                    NL             0             0                       0.036
     4 San-Diego, New-York
                    NL             0             0                       < eps
     5 San-Diego, Chicago
                    NL             0             0                       0.009
     6 San-Diego, Topeka
                    B            275             0

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 2.84e-14 on row 1
        max.rel.err = 9.22e-17 on row 1
        High quality

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

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

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

```
        High quality

End of output
```