GUSEK Tutorial

SA405: Fall 2018

CDR Jay Foraker

Adapted from Professor Phillip’s SA305 GUSEK Tutorial

GUSEK is a standalone executable that combines the SCIntilla based Text Editor (SciTE) editor plus the linear/integer programming solver GNU Linear Programming Kit (GLPK). GUSEK can be downloaded from the following URL: <http://gusek.sourceforge.net/gusek.html>. An *editor* is a program that lets you write, modify, and save text files. Both SciTE and GLPK are open source, and are free. The solver GLPK can read a *math programming language* called mathprog, which is a subset of AMPL. A math programming language is intended to be used for optimization models such as linear and integer programs. This document will describe the basics of how to write and solve linear programs in mathprog using GUSEK. More detailed descriptions of available commands and syntax are provided in the file gmpl.pdf, which is included as part of the GUSEK installation package.

**Basic commands**

# a pound sign makes the whole line a comment

/\* can also be used to insert a comment \*/

var varname [>= 0];

maximize objname: (formula);

[subject to] constraint\_name: (equation/inequality);

Note that things in brackets are optional; i.e. the [>= 0] is optional. If you include it, the variable is constrained to be nonnegative. Note that you must WRITE OUT maximize and minimize. Finally, subject to may be replaced with s.t. or be omitted entirely.

**Output commands**

GUSEK can write the results of your formulation to an output file. It can also display the results in the main program window. In order to display the results you should use something like the following:

printf{1..56} "="; printf "\n";

printf "Minimum total cost is %g", objname;

The printf{1..56} "="; command writes 56 equals signs to the screen. The printf "\n"; starts a new output line. The command printf "Minimum total cost is %g\n\n", objname; writes the words “Minimum total cost is” to the screen. The %g symbol is a placeholder for the value of the objective function given by objname.

Here is a sample formulation of Example 1.1 from page 5 in your book. Farmer Jones decides to sell two types of cakes, chocolate and vanilla. Each chocolate cake sold gives a profit of $3, and the profit on each vanilla cake sold is $5. Each chocolate cake requires 20 minutes of baking time and uses 4 eggs and 4 pounds of flour, while each vanilla cake requires 40 minutes of baking time and uses 2 eggs and 5 pounds of flour. If Farmer Jones has available only 260 minutes of baking time, 32 eggs, and 40 pounds of flour, how many of each type of cake should be baked in order to maximize Farmer Jones’ profit?

var C, >= 0;

/\* number of chocolate cakes to bake \*/

var V, >= 0;

/\* number of vanilla cakes to bake \*/

maximize obj: 3 \* C + 5 \* V;

s.t. number\_of\_eggs:

4 \* C + 2 \* V <= 32;

amount\_of\_flour:

4 \* C + 5 \* V <= 40;

baking\_time:

20 \* C + 40 \* V <= 260;

solve;

printf{1..56} "="; printf "\n";

printf "Maximum profit is %g\n\n", obj;

printf "Number of chocolate cakes to make: %g\n", C;

printf "Number of vanilla cakes to make: %g\n", V;

end;

Note that GUSEK needs us to separate the formulation from the output statements by using the solve; statement. GUSEK also likes it when we include an end; statement at the end of the file. The end; statement is not strictly required, but GUSEK will complain about it if you omit it. The final two printf commands in the above example give the values of the decision variables, C and V, at the optimal solution.

**Solving your program**

In order to solve your model, you should first save your file with a .mod extension. You can then select the menu item Tools-Go to solve the model. A helpful shortcut is the function 5 (F5) key, which is a shortcut key for the Tools-Go menu command.

**Some coding tips**

A semicolon MUST end every non-comment line. Names for variables, objectives, and constraints are **required**, **cannot have spaces**, and **cannot repeat**.

**A More Complicated Example**

Here is a more advanced example formulation for Example 1.3 on page 16 in your book. Senior design students are trying to determine which person will take primary responsibility for the remaining tasks the team must complete. Below is a table of the amount of time each student would need to complete the corresponding task, in hours.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 |
| Student 1 | 12 | 5 | 8 | 9 | 6 | 11 |
| Student 2 | 14 | 8 | 7 | 11 | 10 | 5 |
| Student 3 | 10 | 9 | 9 | 8 | 7 | 8 |
| Student 4 | 11 | 8 | 10 | 10 | 9 | 10 |

If each student must do at least one but no more than two tasks, how should the tasks be divided so as to minimize the total amount of time required to finish all the tasks?

set I:={1..4};

/\* set of students \*/

set J:={1..6};

/\* set of tasks \*/

param time{i in I, j in J};

/\* time student i would need to complete task j in hours \*/

var x{i in I, j in J} binary;

/\* 1 if student i is assigned to task j, 0 otherwise \*/

minimize obj: sum{i in I, j in J} time[i,j] \* x[i,j];

/\* objective is to minimize the total amount of time required to complete all tasks \*/

s.t. each\_student\_at\_least\_one\_task{i in I}:

sum{j in J} x[i,j] >= 1;

each\_student\_at\_most\_two\_tasks{i in I}:

sum{j in J} x[i,j] <= 2;

at\_least\_one\_student\_to\_each\_task{j in J}:

sum{i in I} x[i,j] >= 1;

solve;

printf{1..56} "="; printf "\n";

printf "Minimum total time is %g\n\n", obj;

printf "Student Task\n";

printf "------- ----\n";

printf{i in I, j in J: x[i,j] != 0}: "%7s %4s\n", i, j;

printf{1..56} "="; printf "\n";

data;

/\* The data could also be provided in a separate dat file \*/

param time : 1 2 3 4 5 6 :=

1 12 5 8 9 6 11

2 14 8 7 11 10 5

3 10 9 9 8 7 8

4 11 8 10 10 9 10;

end;

The command printf{i in I, j in J: x[i,j] != 0}: "%7s %4s\n", i, j; is used to give the values of i and j ONLY when x[i,j] is NOT EQUAL to zero in the optimal solution. The command data; is used to separate the output commands from the parameter data. The parameter data could be specified in a separate .dat file, which we will do later in the course. It is simpler at this point to include the data in the .mod file.

**Getting Help**

I encourage you to contact me if you are having difficulty getting your code to run. Please try to resolve the bug yourself for a few minutes, but DO NOT spend more than 10 minutes looking for a syntax error. Instead, email me your code. I will respond as quickly as possible with a bug fix.

**Don’t Get Discouraged**

If you have limited coding experience, don’t be afraid to give it a try. You really can’t “break” anything. If you are having significant difficulty, contact me and we can set up EI to get you up to speed.