IOE 610
Homework 11, problem 2: Column Generation Implementation, Due Tuesday, 12/10/13

Reading and reference: here and in problem 3 we will use AMPL scripting capabilities to implement delayed column/constraint generation methods we are currently discussing in class. For a tutorial on these scripting capabilities, refer the following sections and chapters of the second edition of the AMPL book available online:

- For explanations on how to use scripting to implement column generation in a cutting stock problem: Sections 14.4. Section 14.5 contains a more general discussion of using *named problems* — a concept that Section 14.4 illustrated.
- For basic scripting and looping instructions and syntax: Chapter 13
- Many of the model/data files mentioned in the above documents as examples can be found at `http://ampl.com/BOOK/EXAMPLES/EXAMPLES2/index_files.html` and `http://ampl.com/EXAMPLES/index.html`.

Also, descriptions of commands that allow you to interact with the CPLEX solver from AMPL are given in
`http://www.ampl.com/BOOKLETS/amplcplex122userguide.pdf`. This includes settings of options that control what AMPL and CPLEX print out on the screen while working on the problem.

(a) **Cutting stock problem — Kantorovich'es approach** Download the model file `kanto.mod` from CTools. This file contains the formulation of Kantorovich'es model of the cutting stock problem. Explain the calculation of the upper bound on the number of large rolls needed used in this model, or, if you think it would not provide a valid upper bound, explain why and provide your own alternative. Prepare (and submit) a companion data file `cut.dat` to be used with the above model file to solve the following instance of the problem: $W = 5600$ and

w_i	n_i
1380	22
1520	25
1560	12
1710	14
1820	18
1880	18
1930	20
2000	10
2050	12
2100	14
2140	16
2150	18
2200	20

(data instance courtesy of Wikipedia). Attempt solving this problem by using Kantorovich'es model. If you find an optimal solution, report its value; if the run does not terminate in reasonable time, abort the solver, report how much CPU time you let it use, and the value of the best integer solution found in that time.

Note: when I assigned the problem with the same data instance exactly a year ago, machines available on CAEN could not solve it. This year, however, I did get an optimal solution after a reasonable time, so I expect you to be able to find it as well. Outside of this homework, however, try the instance from the lecture notes to experience first hand how hard cutting stock problems can be!

(b) **Cutting stock problem — column generation approach.** This is a warm-up exercise to provide you with an example script. See above for reference reading.

Download files `cut.mod` and `cut.run`. The former contains two models: one for the restricted master problem for the pattern-based formulation, and for the corresponding pricing subproblem, and the other — the implementation of the column generation algorithm to solve the LP relaxation of this formulation, which uses the above model file and data file `cut.dat` you created in part (a) (double-check to make sure your data file is compatible!).

(i) Add comment lines to `cut.run` to explain what each block of code represents/does. Make sure you highlight any parts of the code that differ from the basic cutting stock algorithm we discussed in class, or perform some additional steps. Submit the resulting AMPL file as part of your solutions.

(ii) Run the column-generation algorithm (by typing `include cut.run` at the AMPL prompt), and report the solution to the cutting stock problem found this way. How many loops of the column generation algorithm were performed to obtain the solution? (Feel free to edit the input/output options in the run file to make output more compact, or to make it display information you want to see.)

(ii) Use the solution to LP relaxation to obtain a feasible solution to the cutting stock problem via the rounding procedure. How do the objective values compare?

(c) **Constrained cutting stock problem.** Imagine now that we have to handle the following manufacturing restriction: we can use at most 3 knives simultaneously, which implies that any cutting pattern we use can create at most 3 items (that's 3 items total, not 3 different types of items).

(i) Formulate a Kantorovich-type model of this problem. Implement it by modifying the file `kanto.mod` (For simplicity, "hard-code" the limit of 3 into the model fine, i.e., don't treat it as a parameter.) Report your findings as in part (a).

(ii) Now let's try a column generation approach. Explain (first in writing) what changes you need to make to the algorithm to incorporate this additional restriction.

(iii) Implement the changes you described in (c.ii) by modifying files `cut.mod` and the commented version of `cut.run` (modify the comments to the latter, and add comments to the former, as necessary to explain the changes you are making). Again, you can "hard-code" the limit of 3 into the files, to keep things simple. Execute the algorithm and report your findings, as in part (b). In testing your implementation, make sure that all patterns you create have at most 3 items!