ISyE6669 Deterministic Optimization
Computational Project
Due November 30, 2018 (On Campus)
Due December 7, 2018 (DL)

Instructions

1. Form a group of **two** or **three** people. Email the names of your group members to the TA Wanrong Zhang (wanrongz@gatech.edu), and cc'd me (andy.sun@isye.gatech.edu) as soon as possible.

2. Divide your responsibility within the group sensibly, and be collaborative and helpful to each other.

3. Collaboration between groups is not allowed.

4. A list of provided files:

   (a) Problem 1: **diet.colgen.partial.mos**, **diet.xls**

   (b) Problem 2: **cs.Kantorovich.partial.mos**, **cs.colgen.partial.mos**, **kant1.dat**, **cs1.dat**, **cs2.dat**, **cs3.dat**

   (c) Problem 3: **TSP-DFJ-partial.mos**, **US48.dat**, **US48.input**, **US48TourPlot.m**.

5. In the final report, you should clearly state your answers to all the questions, and print out the numerical results required by each problem. The report should also comment on the contributions of each member to the project.

6. Each group submits all their codes and the final report on t-square **before 5pm EST on Nov. 30, 2018** for on-campus students. There is no resubmission allowed on t-square. No physical copy is needed.

7. Some hints:

   - Sometimes, it's useful to first make the overall framework work, then divide into each subtask.
   - Divide and conquer: When one problem consists of different subproblems, work on each subproblem as a stand alone problem. Write separate code for the subproblems, and make sure they work before putting things together.
   - It is a good practice to have clear comments in your code.
   - After this project, you will get a pretty decent working knowledge of column generation and constraint generation. The project requires a considerable amount of effort in understanding the algorithms as well as in coding. To be successful, all members in the group need to work diligently and collaboratively. Also start as early as you can.
   - Have fun!

Problem 1: Column Generation for the Diet Problem

In this problem, we will solve a large diet problem. There are 7146 kinds of food listed, and 30 nutrients. We want to find a diet that has the minimum level of cholesterol intake and at the same time satisfies all the nutritional requirement.

For each nutrient i, let m_i be the minimum daily intake of i and let M_i be the maximum daily intake of i. For each food j, let a_{ij} be the amount of nutrient i in food j. Let $chol_j$ be the amount of cholesterol in food j, and define variables x_j to be the amount of food j in the daily diet. Then, the formulation of the diet problem is:

$$\min \quad \sum_{j=1}^{7146} chol_j \cdot x_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{7146} a_{ij} x_j \leq M_i, \quad \forall i = 1, \ldots, 27 \tag{2}$$

$$\sum_{j=1}^{7146} a_{ij} x_j \geq m_i, \quad \forall i = 1, \ldots, 27 \tag{3}$$

$$x_j \geq 0, \forall j = 1, \ldots, 7146 \tag{4}$$

Suppose you wanted to solve it using the student (free) version of Xpress, which cannot handle 7146 variables. Apparently, you need to solve the problem in a more clever way. Column generation is an ideal approach. Attached file (**diet.colgen.partial.mos**) gives the general framework of column generation, but it is missing the crucial part.

Questions:

1. Complete the construction of the column generation code and submit the mos file online and also submit a physical copy with the final report.
2. Print out the final solution from the code, and submit it in the final report.
3. Now, you want to find an optimal diet with the total calorie intake between 1800 cal and 2000 cal. Modify the data file diet.xls (the minimum and maximum levels in the calorie column), and resolve the above problem. Report in the final report the composition of optimal diet and the total amount of each nutrient provided by the diet.

HINT:

1. getdual(name) is the Xpress function to retrieve the optimal dual variable (also called shadow price) of a constraint called "name". To store that shadow price, you need to save that value in the array given to you.
2. A general way to add a new column to a constraint or the objective function is: name += XXX, where name is the name of the constraint or the objective function, and XXX is the part of the constraint that comes with the new variable.

Problem 2: Solution Strategies for the Cutting Stock Problem

The cutting stock problem formulation that we learned in class is due to P. C. Gilmore and Ralph E. Gomory. They published this formulation in their 1961 paper "A linear programming approach to the cutting-stock problem", *Operations Research*, 8 (1961), 849-859. We call this formulation the Gilmore-Gomory formulation. A little history here: Ralph E. Gomory is a renowned mathematician and a key figure in the development of both theoretical understandings and computational methods for integer programming. He has also been very influential in bringing operations research to the real world. He first worked at IBM as a research mathematician and later became IBM's Senior Vice President for Science and Technology. He helped IBM attain some of the best minds for OR for over 20 years. He still maintains a blog in the Huffington Post on interesting topics such as technology development and industry research[1] at an advanced age.

An alternative optimization formulation for the cutting stock problem was proposed by the Soviet mathematician and economist Leonid V. Kantorovich in 1939. His paper was published in English in 1960 ("Mathematical Methods of Planning and Organising Production" *Management Science*, 6 (1960), 366-422. You can get the paper online through Tech's library.) Later, Kantorovich won the Nobel Prize in Economics in 1975, shared with another pioneer of operations research Tjalling Koopmans, "for their contributions to the theory of optimal allocation of resources."

Kantorovich's formulation for the cutting stock problem is very different from the Gilmore-Gomory formulation. In this problem, we lead you through steps to formulate the Kantorovich formulation and implement in Xpress. Then, we ask you to implement Column Generation on the Gilmore-Gomory formulation. The purpose is to compare these two different formulations and solution strategies for solving the same cutting stock problem. Through this exercise, you will learn that the compact formulation of an optimization problem might not always be computationally easier to solve than a larger formulation. You will also see the power of column generation as a solution strategy to solve large-scale linear programs.

Problem 2.1: The Kantorovich Formulation: A Modelling Exercise

The Kantorovich formulation uses the following data and decision variables:

1. Data:
   - w_i, $i = 1, \ldots, m$: the width of small roll item i.
   - b_i, $i = 1, \ldots, m$: the demand for item i.
   - W: the width of the large rolls.
   - K: the total number of large rolls. (Note that this data is not required in the Gilmore-Gomory formulation.)
2. Decision variables:
   - y^k: if large roll k is cut then $y^k = 1$, otherwise $y^k = 0$, for $k = 1, \ldots, K$.
   - x_i^k: number of times that item i of width w_i is cut on the large roll k.

Now we lead you through the steps to formulate the Kantorovich model.

- The objective is given as to minimize the number of large rolls cut to satisfy demand:

$$\min \quad \sum_{k=1}^{K} y^k$$

[1] `http://www.huffingtonpost.com/ralph-gomory/`

- Constraint 1: Demand must be satisfied with equality for each item $i = 1, \ldots, m$.

- Constraint 2: The width of a large roll can not be exceeded for each large roll $k = 1, \ldots, K$. Here you should consider that Constraint 2 is only needed when a large roll is cut, which involves the y^k variable.

- Constraint 3: Variable bounds and integrality constraints.

Questions:

1. How many variables and constraints are there in this formulation?

2. Use the files **cs.Kantorovich.partial.mos** as a starting point. Finish the Xpress code and solve the problem using the data file **kant1.dat**. Submit the mos file online and also submit a physical copy with the final report. Print out the optimal objective value and the optimal solution in the report. Since there are many variables, to save space, only print out the non-zero optimal variables for both x_i^k's and y^k's.

3. Next change K to 600 and change m to 10 and use the data file **cs1.dat**. What do you observe? Is it easy or hard to solve the problem using the Kantorovich formulation? You can keep the solver running as long as you want, when it terminates (or when you run out of patience and stops it manually, click on the red cross button), answer the following questions:

   - How many branch-and-bound (BB) nodes are searched by the BB algorithm?
   - What is the objective value of the best lower bound and the objective value of the best integer solution found?
   - What is the optimality gap given by the above two numbers?
   - How many integer solutions are found by the BB algorithm?
   - How many seconds did the algorithm take?

   For these questions, click on the "Stats" tab on the right-hand side panel in Xpress. Look at the results after "Current node", "Best Bound", "Best solution", "Gap", "Status", and "Time". Lastly, relax the integrality constraints. Write your answers in the report.

Problem 2.2: The Gilmore-Gomory Formulation: Use Column Generation

You should encounter some difficulty with the Kantorovich formulation for the larger data set. In this part, we want to solve the Gilmore-Gomory formulation (exactly the same formulation we discussed in class) using column generation. The master problem is given below.

$$\begin{aligned} \min \quad & \sum_{j=1}^{N} x_j \\ \text{s.t.} \quad & \sum_{j=1}^{N} a_{ij} x_j = b_i, \qquad \forall i = 1, \ldots, m \quad \text{(Demand Constraints)} \\ & x_j \geq 0, \qquad \forall j = 1, \ldots, N. \end{aligned}$$

Attached file (**cs.colgen.partial.mos**) gives the general framework of the column generation algorithm. It also has detailed instructions on how to complete the code. The programming experience you gained in solving Problem 1, the diet problem, will be useful here. You need to do the following.

Questions:

1. Complete the column generation code. Submit the completed mos file online and also submit a physical copy with the final report.

2. Print out solution summary for each test instance (**cs1.dat**, **cs2.dat**, **cs3.dat**), following the instructions in the **cs.colgen.partial.mos** file.

3. In your report, discuss the differences between the LP solutions and the integer solutions obtained in the three test instances, also the differences between the integer solutions of the two approaches (rounding and resolving). Which approach produces better solution?

4. Now, change the demand constraint from $=$ to $\geq$, that is, $\sum_{j=1}^{N} a_{ij} x_j \geq b_i$. Solve the LP relaxation of the resulting cutting stock problem and obtain the integer solutions using the same procedure as in the above question for **kant1.dat**, **cs1.dat**, **cs2.dat**, **cs3.dat**. Compare the results to the ones of the equality demand constraint. If the two formulations give different solutions, elaborate how the solutions are different for each test case.

HINT:

1. The Xpress code uses dynamic array to implement an array whose size can change dynamically. To create a new variable, use create(variable name).

2. You can use a similar way to add new columns to constraints and the objective function as hinted in problem 1.

Problem 2.3: An Alternative Objective – Minimize Total Waste

Now let us consider an alternative objective of minimizing the total wasted paper for satisfying demand. More specifically, the total wasted paper of a cutting stock solution is the total amount of paper that is cut but not used to satisfy the demand. For instance, if a large roll of width 100 is cut into 3 copies of width 30 and the remaining $10 = 100 - 3 \times 30$ units of width is thrown away, then that 10 units of paper is counted as wasted paper of this roll. The total amount of wasted paper is the sum of wasted paper from each large roll that is cut to satisfy demand. We call the new cutting stock problem, CS-min-waste.

Answer the following questions.

Questions:

1. Modify the Gilmore-Gomory formulation (the LP version) to incorporate the objective of minimizing the total wasted paper. The demand constraint is still an equality constraint. Write down the new formulation.

2. Use column generation to solve CS-min-waste. Write down the reduced master problem (RMP) and the pricing problem. In particular, the pricing problem should be written first as an enumerate problem and then be transformed into an optimization problem.

3. Write a new .mos file (name it cs.colgen.minwaste.mod) to implement the column generation for CS-min-waste. Your code should also include the capability to get an integer solution through resolving. Hand in your code on t-square.

4. Run your code cs.colgen.minwaste.mod on all four test cases **kant1.dat**, **cs1.dat**, **cs2.dat**, **cs3.dat**. Compare CS-min-waste with the original cutting-stock formulation in Problem 2.2. For each test case, does the LP solution of CS-min-waste have the same optimal objective with that of the original cutting-stock problem? What about the optimal LP solution? Does the integer solution of CS-min-waste have the same objective with the original problem? Compare the total amount of wasted paper of the integer solution of the cs-min-waste model with that of the original model? How much different are they?

5. After you run the test cases, does your results suggest some interesting relation between the LP solutions of the two cutting stock models? Prove that these two models (both the LP relaxation and the integer version of the CS-min-waste and the original cutting-stock problems) are in fact equivalent, thus always give the same optimal objective value. Or argue that your experimental results disprove the above statement.

6. Now change the demand constraint to $\geq$ as in Problem 2.2 Question 4. Modify your cs.colgen.minwaste.mod code and rerun all the four test cases. Answer Problem 2.3 Question 4 again with the new results. Can you explain why the new results with demand constraint of $\geq$ are different or the same as the ones with demand constraints of $=$ sign you obtained in Problem 2.3 Question 4?

Problem 3: TSP using Dantzig-Fulkerson-Johnson Formulation

In this problem, we study the classic formulation for the TSP problem, namely the Dantzig-Fulkerson-Johnson (DFJ) formulation, and implement a constraint generation algorithm to solve it.

DFJ Formulation

The DFJ formulation was first proposed by G. Dantzig, D. Fulkerson, and S. Johnson. All three of them made significant contributions to operations research and optimization. The original paper where they first proposed the formulation has become a classic "Solutions of large scale travelling salesman problem", *Operations Research*, 2:393-410, 1954.

$$
\begin{aligned}
(DFJ) \quad \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} && && (5)\\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 && \text{for all cities } i = 1, 2 \ldots, n && (6)\\
& \sum_{j=1}^{n} x_{ji} = 1 && \text{for all cities } i = 1, 2 \ldots, n && (7)\\
& \sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1 && \text{for all subset } S \subset N \text{ and } S \neq N && (8)\\
& x_{ij} \in \{0, 1\} && \text{for all cities } i, j = 1, 2, \ldots, n
\end{aligned}
$$

The decision variables $x_{ij} = 1$ if the tour goes directly from city i to city j, $x_{ij} = 0$ otherwise. The objective is to minimize the total travel distance of the tour $\sum_{j=1}^{n} d_{ij}x_{ij}$, where d_{ij} is the distance between cities i and j. Constraint (6) ensures that the tour goes from city i to exactly one other city, and Constraint (7) ensures that the tour goes into city i exactly from one other city. These two types of constraints together are called the *assignment constraints*. However, as we know, they are not enough to characterize a correct tour, because the solution to the assignment constraints may contain subtours of smaller cycles around subsets of cities. We need constraints (8) to eliminate all the subtours. Here $N = \{1, 2, \ldots, n\}$ is the set of all n cities, and S is any strict subset of N with $|S|$ number of cities.

The DFJ formulation has n^2 binary variables. However, the number of constraints is astronomical! Indeed, we need a subtour elimination constraint (8) for each non-empty subset S of N, excluding N itself (why?). There are $2^n - 2$ such constraints. That's why we say the number of constraints is exponentially many in the number of variables. For a problem of 48 cities, there will be $2^{48} - 2 = 281,474,976,710,654$ (281 trillion) subtour elimination constraints. It will be completely nonsensical to enter such a model into Xpress!

Constraint Generation Algorithm

In the following, we outline a **constraint generation** procedure that can be implemented quite easily in Xpress and can find an optimal (yah!) solution fairly fast. The strategy is as follows:

1. First form the restricted master problem with a subset of constraints (8). In the following we will explore different options for the initial set of constraints.

2. Solve the restricted master problem.

3. Check if any subtour elimination constraint is violated by the current solution. There are many ways to do this. We choose the following method: Find a tour starting at City 1 (which is Atlanta). Such a tour always exists and is unique (Why?). Then, if this tour contains less than 48 cities, then this is a subtour and we can add the corresponding subtour elimination constraint (8) to the restricted master

problem. Go to Step 2 and repeat. Otherwise if the tour has all 48 cities, then it is a legitimate TSP tour, and in fact optimal, then we are done.

Steps to Implement the Algorithm

The above outlined framework of constraint generation is partially coded in TSP-partial.mos. The data file US48.dat contains coordinates of the 48 state capitals. You need to fill in the parts that are marked "!!!!!!!!! fill in your code here !!!!!!!!!!". In particular, you need to do the following to complete the code:

1. Formulate the objective function, all the assignment constraints.
2. Write small subtour elimination constraints for subsets of 1 city, 2 cities, 3 cities, and 4 cities.
3. Finish the constraint generation part:
   (a) Write code to find a subtour starting at Atlanta (City 1). Hint: One way to find such a tour is to start at Atlanta and see which city comes next, then see which city comes after that, etc, until you trace back to Atlanta. Mark all the cities on the subtour.
   (b) Write code to generate the corresponding subtour elimination constraints and add it to the restricted problem.

Computational Experiments

1. Begin with all of the small subtour elimination constraints commented out (So, these will be generated by your constraint generation code if necessary.) Use your Xpress code to solve the problem. If it runs for more than 20 minutes, stop Xpress in the middle — report "Reach time limit of 20 min". If it does finish quickly, record how long it takes and how many constraints were generated. You can find the running time in the "Stats" tab in Xpress.
2. Now un-comment the 1-city subtour elimination constraints, so that the formulation begins with those constraints already in it. Use your Xpress code to solve the problem. If it runs for more than 20 minitues stop Xpress in the middle and report "Reach time limit of 20 min". If it does finish quickly, record how long it takes and how many constraints were generated.
3. Now start the formulation with both 1-city and 2-city subtour elimination constraints. Use your Xpress code to solve the problem. If it runs for more than 20 minitues stop Xpress and report "Reach time limit of 20 min". If it does finish quickly, record how long it takes and how many constraints were generated.
4. Now start the formulation with 1-city, 2-city, and 3-city subtour elimination constraints. Use your Xpress code to solve the problem. If it runs for more than 20 minitues stop Xpress and report "Reach time limit of 20 min". If it does finish quickly, record how long it takes and how many constraints were generated.
5. Now start the formulation with 1-city, 2-city, 3-city, and 4-city subtour elimination constraints. Use your Xpress code to solve the problem. If it runs for more than 20 minitues stop Xpress and report "Reach time limit of 20 min". If it does finish quickly, record how long it takes and how many constraints were generated.
6. For each of (i)-(v), how many subtour elimination constraints did the formulation begin with?
7. Plot the optimal TSP tour you found, using the Matlab code US48TourPlot.m.
8. Figure out each city's name (City 1 is Atlanta). Write the TSP tour in the names of the state capitals.

9. Select 26 cities that you want to visit. Create a new data file US26.dat. You will need to use this data set for the next part as well. Run your Xpress code to find an optimal TSP tour. Report the optimal tour. Modify the Matlab code and generate a similar plot of the tour over these 26 cities. You need to hand-in the new data file for this part.

Deliverables

1. Complete and well-commented Xpress codes (both .mod and US26.dat). Include all of the 1-city, 2-city, 3-city, and 4-city subtour constraints, but comment out the 3-city and 4-city subtour constraints. Data initialization part uses the 48-city data.

2. Matlab code for reading data and generating the plot for 26-city tour.

3. In the final report, answer all the questions in steps 1-6 of Section , and print out the optimal 48-city TSP tour and the optimal 26-city TSP tour, their total distances, and the plots generated by the Matlab code. For the 26-city tour, you need to clearly state the indices and the names of the 26 cities you select, and attach the data file you used in Xpress. Attach the complete Xpress and Matlab code in the end.