Computation and Economics - Fall 2014
Assignment #5(b): Linear and Integer Programming

Professor Sven Seuken
Department of Informatics, University of Zurich

Out Tuesday, November 4, 2014
Due **12:15** sharp: **Tuesday, November 11, 2014**
For submission format, check description below.

[**Total: 60 Points**] This assignment can (but need not) be solved in groups of **up to two students**. Note that if you are working in a group of two, we expect that both students contribute roughly equally to the assignment and both understand all parts of the solution.

Points will be awarded for clarity, correctness and completeness of the answers. Reasoning must be provided with every answer, i.e., please show your work. You get most of the credit for showing the way in which you arrived at the solution, not for the final answer. You are free to discuss the assignment with other students. However, you are not allowed to share (even partial) answers and source code with each other, and **copying will be penalized**.
Every group has to hand in their write-up (PDF) as well as their source code via email to `brero@ifi.uzh.ch` in a zip-file. In contrast previous exercises, where every student had to write his/her own write-up, in this assignment you need to hand in **only 1 write-up per group**. Please send the source code in a zip-file. You can also hand in the write-up physically before the lecture.

1 Notes and Setup

In this assignment, you will implement different linear and integer programs using Java/JOpt, which you have installed for the last exercise sheet.

1. For all problems, we provided a main method with which you can run your code. Please note that your methods also have to work for other test samples (not only the ones in the main method).

2. You can add new classes, methods, etc. However, your code has be executed by calling the method skeleton we provided, without further changes. (We will use automated testings)

3. For the Travelling Salesman Exercise: download additional libraries with Maven. To do so, right-click on the project, and select *Run as → Maven clean.* For this to work, you need to have Maven as well as the Eclipse plugin M2E installed. In Eclipse, click on Windows > Install new Software > Work with: "http://download.eclipse.org/technology/m2e/releases/" > Select and Install "Maven Integration for Eclipse" (Including incubating components).

4. Have fun!

2 Problem Set

1. **[35 Points]** Multicommodity transportation model
   A company has to ship its product from different factories to different customers. Let F be the set of factories (indexed by i) and C be the set of customers (indexed by j).
   Each factory has a_i available units of product and each customer requires at least d_j units of product.
   Assume that each factory cannot send more units than those it has at its disposal and each customer should receive at least the amount of units of product required.
   The product can be sent in fractional quantities from each factory i to each customer j with a cost of $c_{i,j}$ per unit sent. Finally, suppose that each shipment from a factory to a destination should not exceed a certain amount of units of product, this amount being independent from the factory i and the customer j.

   (a) **[10 Points]** Write down the linear programming problem and implement it in Java/JOpt. In the package `mtm`, we provide you with a code skeleton.

   (b) **[10 Points]** Assume now that each origin-destination route i, j has a 'fixed' cost $f_{i,j}$ that the company has to pay (additonal to the normal costs) only if the connection is used (i.e. if some non negative amount of product is sent from i to j). Furthermore, suppose that if connection i, j is used, the company must send at least the amount `limMaxFlow` of units of product through this connection.
   Last, assume that each factory i cannot serve more than a quantity of customers equal to a certain parameter $\texttt{maxServe}_i$. Write down the new linear programming problem and implement it in Java/JOpt. In the package `mtm`, we provide you with a code skeleton.

   (c) **[15 Points]** Consider again the original problem derived in the first point of this exercise.

      i. **[5 Points]** Assume now that the product can be only sent in quantities that are multiple of its reference units, i.e. no fractional units of products can be sent from a factory to a customer. Adjust your program of task (a), such that it solves this changed problem if parameter *transportUnitsAreFractional == false* and the 'regular' problem of (a) if *transportUnitsAreFractional == true.*

      ii. **[6 Points]** Suppose that the number of factories is equal to the number of customers and let $n \geq 2$ be this number. You now should compare the runtimes of the two problems. A simulator is already prepared in the part (c) java file. Complete the simulator, as stated in the comments in the code, and run it for increasing n.

      iii. **[4 Points]** Draw/Print a graph of the runtimes in your writeup-sheet and explain the observed behaviour.

   Hints:

   - The prepared method for the simulation output writes the values of the simulation in coma separated form. This should make it easy to import the file in Excel and plot a graph.
   - Make sure that the values you use for your simulation are realistic and non trivial. E.g. a simulation where supply always exceeds demand by a big factor or a simulation setting where the linear problem often finds an integer solution would not be realistic. Note that trivial settings might not lead to the desired outcome for (iii).

- If your generated simulation values might lead to infeasible solutions, try at least getting the graph for (iii).

2. **[25 Points]** Planning a Roadtrip.

   You are planning to visit the following ten cities by car during the summer holiday:

   - Zurich, Schweiz
   - Genf, Schweiz
   - Freiburg, Deutschland
   - Nijmegen, Niederlande
   - Stockholm, Schweden
   - Turin, Italien
   - Madrid, Spanien
   - St. Petersburg, Russland
   - Venedig, Italien
   - Dubrovnik, Kroatien.

   Starting from Zurich, your goal is to visit each city *exactly once* and then to return to Zurich again, while minimizing the total traveling time.

   (a) **[10 Points]** The above road trip problem can be cast as an instance of the *traveling salesman problem* (TSP). The TSP can be formalized as an integer program. You can find the IP formulation on the web (e.g. on Wikipedia). Research such a formulation and write it down. Be sure to specify the type (binary, integer, or continuous) of each variable and describe the elements of your model (e.g. objective function, constraints).

   (b) **[15 Points]** Implement a TSP solver in Java/JOpt. In the package `travelingsalesperson`, we provide you with a code skeleton. It uses Google Maps to get the traveling times between different cities.

   The cities that you want to visit can be specified in the file `cities.txt` in the folder `src/main/resources` (Note that there is an API limitation to 10 cities). The distance matrix between these cities is then computed automatically (i.e. the code is already implemented). The output of the optimization is written to the file `output.tsp`.

   You need to write code that implements your integer program from (a) and solves it. Do this in the class `TravelingSalesmanProblem.java`.

   What is the result of your optimization (sequence of cities, total traveling time)?

3. **[Bonus Assignment]** Assume the following setting: some students would like to go to a bar. However, before they can do so, they have to solve some exercises for the Economics and Computation class. To save some time, they decide to cheat and distribute the exercises between them: every exercise is solved by only one student and then copied by the others. The students are social and don't leave before all students are ready to go, i.e., before all exercises are solved by at least one student. Obviously, the earlier the last student has finished the exercises which were distributed to him, the sooner the friends can go to the bar and the higher is their utility.
   Student i requires a specific amount of time $p_{i,j}$ to solve exercise j. You can assume that these durations are public knowledge and known long before the students even met to solve the exercises.
   Furthermore, you can assume the following:

   - Students don't need breaks, and they are indifferent between waiting and solving exercises. :)
   - Exercises are non-divisible, i.e., an exercise which is partly solved by one student can not be completed by another student.
   - Copying does not cost any time.

   You now have to distribute the exercises to the students, such that they are solved as early as possible. Note that the number of students as well as the number of exercises are natural numbers.

   (a) Formalize this problem and formulate it as an integer linear problem, i.e. write variables, objective function and constraints on your writeup sheet.

   (b) We prepared some code for you. Complete the method *distributeExercises(int[][] solveTimes)* which has to return the amount of minutes it takes before the students have finished, i.e., the time it take before the last student has finished his exercises.

   (c) Imagine the exercises are divisible, i.e., they can be divided in arbitrary many parts, which then can be solved by different students independent of each other. Which parts would you have to change in your code (you don't actually have to change it). What would you expect to happen with the result? Why?