ISyE/CS 728 – Integer Optimization
Spring 2021
Assignment #2

Due Date: February 25, 1pm.

The assignment should be submitted electronically in pdf format (except for code files) in Canvas. Late submission policy: 20% of total points will be deducted per hour.

Students are strongly encouraged to work in groups of two on homework assignments. Only one set of solutions should be submitted for both group members. In order to submit the assignment for your group please follow these steps in Canvas. Step 1: Click on the "People" tab, then on "Assignments Groups", and join one of the available groups; Step 2: When also your partner has joined the same group, one of the two can submit the assignment by clicking on the "Assignments" tab, then on the assignment to be submitted, and finally on "Submit assignment". The submission will count for everyone in your group. If you prefer to be in no group, then just skip Step 1.

Groups must work independently of each other, may not share answers with each other, and solutions must not be copied from the internet or other sources. If improper collaboration is detected, *all groups* involved will automatically receive a 0. Students must properly give credit to any outside resources they use (such as books, papers, etc.). In doing these exercises, you must justify all of your answers and cite every result that you use. You are not allowed to share any content of this assignment.

1 Degree-constrained minimum spanning tree

Let $G = (V, E)$ be a graph with $n = |V|$ nodes. A *spanning tree* in G is a set of edges $T \subseteq E$ with $|T| = n - 1$ and which contains no cycles. (You can convince yourself that the requirement that it have $n - 1$ eddges ensures that T is *spanning* in that it connects the entire graph.) Given a graph $G = (V, E)$ and weights $w : E \to \mathbb{R}_+$, the minimum spanning tree (MST) problem is to find a spanning tree in G that minimizes the weight of the edges in the tree. The MST problem can be solved very efficiently with a greedy algorithm (Kruskal's algorithm). For the purpose of these problems, you do not need to know this algorithm, just that any MST problem can be solved very efficiently.

Given a graph $G = (V, E)$, edge weights $w : E \to \mathbb{R}_+$, and node degree limits $k : V \to \mathbb{Z}_+$, the degree-constrained minimum spanning tree (DCMST) problem is the problem of finding a minimum weight spanning tree T in G that has at most k_i edges incident to each node $i \in V$. (In other words, $|\delta(i) \cap T| \leq k_i$ for all $i \in V$, where $\delta(i) = \{e \in E : i \in e\}$ is the set of edges that are adjacent to node i.)

1.1 Problem (2 points)

Formulate the DCMST problem as a binary integer program, using (only) variables x_e for $e \in E$, where $x_e = 1$ if and only if edge e is in the selected tree. You should explain each of the constraints in your formulation, but a formal verification that your formulation is correct is not required for this problem.

1.2 Problem (1 points)

Given a DCMST problem, prove that the optimal value of the MST problem obtained by ignoring the degree constraints provides a lower bound on the optimal value of the DCMST problem.

1.3 Problem (3 points)

Design a branch-and-bound algorithm for solving the MDCST problem, where the lower bound at each node is obtained by solving a MST. In other words, the subproblem to be solved at each node of the branch-and-bound tree should be an MST problem in some graph. Clearly define the branching rule in your branch-and-bound algorithm, and describe how the MST problem is defined at each child node given your branching rule.

Hints: (1) The MST solution may violate the degree constraint for some node: in this case the optimal degree-constrained spanning tree cannot possibly contain all the edges adjacent to that node in the MST solution, so in any feasible solution, at least one of those edges must not be selected. (2) Deleting edges from the graph just gives another graph, a subgraph of the original; the subproblem to be solved at each node of the branch-and-bound tree should be an MST problem in some subgraph. (3) The branching rule is allowed to create more than two subproblems.

2 Solving the asymmetric traveling salesman problem

2.1 Problem (3 points)

Implement the "compact" formulation (based on position variables and constraints) for eliminating subtours in the asymmetric traveling salesman problem we saw in class. You may use any software you like for your implementation. (My example solution will use Gurobi in Python.) Turn in a summary of the results of the tests in the bullets below, your code, and a file containing the output of the code when it runs. *You must summarize your results in your submission.* Significant credit will be taken off for submissions that only include printouts of the output of your code as it runs. Two test instances, both with complete directed graphs, are posted on the course web site. The first number in each file represents the number of nodes n in the graph. The remaining numbers represent all $n \times n$ costs between pairs of nodes in the graph (the cost may be different in different directions). For convenience, the cost matrix includes a "cost" for a node to travel to itself, even though this is not allowed. I have also posted a starter python code that reads the cost matrix, in case you choose to use Python for your implementation.

1. Set a time limit of 5 minutes, and attempt to solve the two instances posted on the web site: ftv47.atsp and si175.atsp. (In Gurobi, the time limit is set with command 'm.Params.timelimit = 300.0'.) For each instance, report in your summary the optimal objective value, the solution time, the number of nodes explored in the branch-and-bound tree, and the root relaxation objective value. If either of the instances does not solve in the time limit, report the ending optimality gap obtained when the time limit is reached.

2. Set a time limit of 3 minutes for these experiments. For instance ftv47.atsp, change the following settings (one by one) and re-solve the instance and report the same results you reported in part (a).

    - Change the branching strategy to most fractional (params.varbranch=2 in Gurobi).
    - Change the branching strategy to strong branching (params.varbranch=3 in Gurobi).
    - Turn off all cuts (params.cuts=0 in Gurobi).

2.2 Problem (3 points)

In this exercise, you will build on your work from the previous problem, but instead implement an algorithm for the asymmetric TSP that uses the subtour elimination constraints for eliminating subtours. Implement an algorithm for solving the asymmetric TSP by solving a sequence of integer programs that have only the x variables, and adding violated subtour elimination constraints until no violated constraints are found. Run this algorithm on both of the test instances of the previous problem. For each instance, report the total time and total branch and bound nodes (over all integer programming solves) required to solve it. (Notice how this compares to the performance in the previous question.) Submit a summary of the results, your code, and a file containing the output of the code when it runs.

Hint: For finding violated subtour elimination constraints, you can use a breadth-first search on the graph defined by the arcs selected in the solution. (E.g., Wikipedia provides a psuedocode: `https://en.wikipedia.org/wiki/Breadth-first\_search`) Starting from any node in the graph, if the breadth-first search algorithm visits a set of nodes that is not the full set of nodes, then the visited nodes define a subtour and adding the corresponding subtour elimination constraint will eliminate that infeasible solution. Otherwise, the selected set of arcs is a valid Hamiltonian cycle. In the case that you find a subtour, you can find more subtours by repeating the graph search starting from a node that wasn't visited in the first subtour found, and you can add a subtour elimination constraint for each identified subtour. Adding multiple subtour elimination constraints in each iteration might speed up convergence.