Discrete Optimization
School of Computation, Information and Technology
Technical University of Munich

Problem Sheet 3

Discrete Optimization (CIT413041)

Some of the upcoming exercise classes will feature computational exercises. You will learn how to model and solve integer programming problems using the programming language Python and the integer linear programming solver SCIP (using its Python interface PySCIPOpt).

For the exercise classes, we expect you to have a working installation of Python and PySCIPOpt (and all other required packages) on your device, and that you are familiar with a suitable editor so you can actually work on the problems on this sheet. Please make sure to read the following instructions carefully in preparation for the next exercise class.

Need help? If you encounter any problems or have any questions when setting up Python / PySCIPOpt, please ask for help on our moodle forum. In case you already have experience with Python, feel free to help your fellow students by answering their questions or posting advice that you found useful.

Downloading Python/PySCIPOpt and helpful resources If you are new to Python, we recommend the tutorials on getting started with Python and the example on building and solving a model from the official PySCIPOpt website. Python itself may be installed from the official Python website or using the project management tool uv (see below).

If you already have Python installed, please make sure you are on a somewhat recent version (at least 3.10 is recommended). For the very first coding exercises on this sheet, you will need to be able to understand and write some basic Python code. Any knowledge of modelling and solving integer programs with PySCIPOpt is helpful but not strictly necessary; the exercise will guide you through the first steps.

The template and data files required for this sheet are available as a ZIP file on the moodle course page, please also download those and copy them to your computer. Optionally, you may also download those files by cloning a GitHub repository (with the advantage that future updates will be available in the same repository), a link is on the moodle course page.

Editor To edit your code, Visual Studio Code is a good choice unless you already have a preferred editor. It will be helpful to install the Python extension as explained in the link.

Solver and Dependencies For working with integer linear programs we will need a "solver" and the corresponding Python interface. Some other packages are useful for certain aspects of the problems on this sheet, too. It may be helpful to use a project manager like uv for that (available at the uv website), but you may also just install the required packages directly.

To start your coding project, unzip the file downloaded from the moodle page into a new directory on your computer. Then either use uv or pip (the latter comes with most python installations) to install the required packages. Open the directory in a terminal (or in VS Code, a terminal will then be

available below the file editor). Change into the directory that contains the files for this problem sheet and install all required packages with the command `pip install -r requirements.txt` or `uv sync`. If you prefer to install packages through your editor, you should install the following packages:

- `pyscipopt` for modelling and solving integer linear programs
- `matplotlib` for creating pictures of graphs
- `networkx` for representing graphs
- `numpy` for easily working with arrays (specifically matrices)

If you use uv remember to run your code with a command like `uv run gcd.py` in the directory `SHEET03`. This will automatically use the packages you have installed before.

Homework Problems

Exercise H 3.1 *(Greatest Common Divisor)*

a) Implement the integer program for computing the greatest common divisor (gcd) of two integers a and b that you have seen in the lecture. You may use the template `gcd.py`.

b) Use your model from part a) to compute the gcd of the following pairs (a, b) and verify your solution:

a	13	8265	918	2021	36	150	−20
b	2	24	2781	−31	81	35	−96
gcd(a, b)	1	3	27	1	9	5	4

Exercise H 3.2 *(Minimum Vertex Cover)*

a) Implement the integer programming formulation for the minimum vertex cover problem that you have seen in the lecture. We recommend using the template `vertexcover.py`. Test your implementation on the test instance, which is comprised of two separate `.txt` files (see the template) and models 29 cities in Bavaria.

b) What happens to your solution when you drop the integrality constraints on all variables in your model? (Make sure to keep the variable bounds and only remove the integrality constraints.)

Problems for the Exercise Class

Exercise T 3.3 *(Bipartite Matching)*

A matching in a graph $G = (V, E)$ is a subset $M \subset E$ of edges such that no two edges in M intersect in a common vertex. A matching in G of maximum cardinality is called a maximum matching.

a) We want to find a maximum matching in the bipartite graph shown in figure 1. Formulate this task as an integer program. Implement and solve your model, starting from the template `matching\_example.py`. How many edges does your maximum matching have?

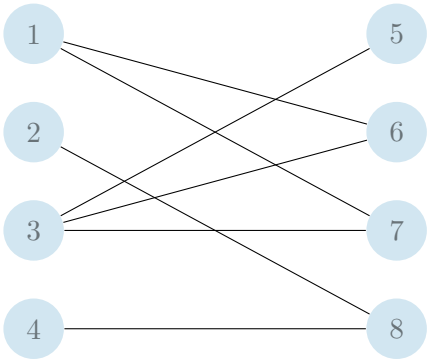


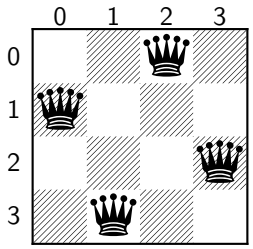
Figure 1: A bipartite graph on 8 nodes.

b) Generalize your integer programming model from part a) to arbitrary graphs G. You may find the template `matching.py` useful. The template uses the package NetworkX, which provides a convenient data structure for constructing and manipulating graphs. The template contains code for constructing a graph from its adjacency matrix, given as a file `adj-*.txt`.

   Please note one peculiarity: For an undirected graph `graph = Graph()` one can access the set of edges through `graph.edges`. However, edges are represented as tuples of the form `(u,v)` in that list (more precisely, this is an iterator), but NetworkX does not store the order of u and v in that tuple internally and thus that order might change in subsequent calls to `graph.edges`. To circumvent this and obtain a predictable result every time, one can resort to using `tuple(sorted(e))` for an edge `e` provided the node names can be sorted (e. g., when the node names are numbers).

c) (Bonus) Now drop the integrality constraints on the variables in your model and solve the instances from part b) once again. What do you observe? (Make sure to keep the variable bounds and only remove the integrality constraints.)

Exercise T 3.4 *(The n-Queens Problem)*
Can you place $n \in \mathbb{N}$ chess queens on an $n \times n$ chessboard such that no two queens threaten each other (i. e., no queen can directly move to a square with another queen on it)? Recall that a queen can move any number of squares horizontally, vertically or diagonally. Here is an example for $n = 4$:



Formulate this problem as an integer program and implement your model. For given n, you should either report that no solution exists, or output an $n \times n$ matrix with a 0/1 pattern, where a 1 entry in row i and column j means placing a queen on position (i, j) of the chessboard.

The template file for this exercise is `queens.py`. It contains the code for printing the solution in a nicely formatted way.