# Assignment 2

## Linear Programs

### Solver

We are using the `lp_solve` solver for modeling linear, integer linear and mixed integer linear programs. The solver can be obtained from

`http://lpsolve.sourceforge.net/`

and pre-compiled versions for various operating systems (e.g., Linux, OSX, Windows) are available, in addition to the sources. Please use the latest version (5.5.2.0).

### LP Input Format

The input format to be used is described at

`http://lpsolve.sourceforge.net/5.5/lp-format.htm`.

Note that this page also includes many examples and alternative formulations. In order to not introduce errors in the description we refer to that page as the reference and do not include examples here. However, we do provide example code (for Java and Python) for modeling INDEPENDENT SET. See below and the assignment package for details.

**NOTE1:** Here are some words of warning. The page that describes the input format says:

> The relational operator can be any of the following: "$<$" "$\leq$" "$=$" "$>$" "$\geq$". There is no semantic difference between "$<$" and "$\leq$" nor between "$>$" and "$\geq$" (even for integer variables!).

Note that this means that "$<$" and "$\leq$" actually both mean "$\leq$", which is somewhat confusing. If you intend to model a *strictly smaller* relation you may want to add/subtract a small constant to one side.

Another issue that was reported is that declaring variables via "bin" as binary variables may cause issues. If you have problems, instead declare them as integers and add the corresponding bounds.

**NOTE2:** More words of warning: It seems that if one wishes to have integer variables that can take negative values, one has to include constraints that take the form of lower bounds with negative constant. For example: say $n1$ is a variable that is supposed to take any value from $\{-1, 0, 1\}$. Then it seems necessary to have a constraint $n1 >= -1$; in the model, since `lp_solve` otherwise implicitly assumes a bound $n1 >= 0$;.

## Graphs

### Library

The problem instances of the second assignment are represented in the form of an edge-weighted graph. The file format used for graphs is an extension to the DIMACS graph format that is described at

`http://dimacs.rutgers.edu/Challenges/`.

In the assignment package we provide a Java library for reading and processing graphs in this format (as well as a Python class), together with Javadoc documentation that describes the library. The description of the standard DIMACS format and the extended format is for your convenience. Further below we discuss a brief example that uses the graph library / class to generate an `lp-solve` model for INDEPENDENT SET. Note that using the provided Java library / Python class is voluntary.

### Input Format

The input format is text based and follows the following rules.

- If the very first (top) line is of the form `p edge 10 19`, then the input file is expected to follow the *normal* DIMACS format and has 10 nodes and 19 edges but **no attributes**.
- If the very first (top) line is of the form `c f 10 19 " " "edgeformat"` means that the graph is of extended format and has 10 nodes and 19 edges. Note the first pair of quotes `" "`. The field `edgeformat` contains attributes that are given in edge lines. For example `id source dest weight` means that edge lines have format giving the label, source and destination nodes, and weight.
- All other lines than the first one starting with `c` are comment lines.
- A line starting with `e` describes an edge. With standard DIMACS format, the line is of the form `e n1 n2` where `n1` and `n2` are the endpoints of the edge (note: node ids are starting at 1). With extended format it has the attributes defined in the edgeformat.

See the two examples below. In the assignment package you find files called `PrintSpinGlass.java` and `PrintSpinGlass.py` that demonstrate how to access the attributes (such as edge weights) from within the Java library / Python class.

**Example DIMACS file**

```
p edge 4 5
e 1 2
e 2 3
e 3 4
e 4 1
e 1 3
```

**Example extended file**

```
c f 4 5 " " "id source dest weight"
e 1 2 4.3
e 2 3 0.2
e 3 4 2.5
e 4 1 5.0
e 1 3 2.1
```

## Example: INDEPENDENT SET problem

In assignment package you will find the following files.

- A Java jar file that contains the graph library: `graph.jar`
- A Python file that contains the graph class: `graph.py`
- An example program that outputs a linear programming encoding for the INDEPENDENT SET problem: `IndependentSetLP.java` and `IndependentSetLP.py`
- A testgraph in standard DIMACS format: `indepinst_graph.g`

The files `IndependentSetLP.java` and `IndependentSetLP.py` contain a linear program encoder for the INDEPENDENT SET problem using Java and Python, respectively. The linear program encoder creates a binary variable for each node in the graph. The variable decides whether the node belongs to the independent set. For each edge `ek(ni,nj)` the encoder creates a constraint `ni + nj <= 1`. The objective function to maximize is the sum of all node variables `ni`.

### Compiling and running the example

Note that the following instructions were tested on Linux and OSX systems. For other OS the steps would be identical or at least very similar.

- Download the files to the same directory that contains the `lp_solve` binary.
- Compile the program if using Java: `javac -classpath graph.jar:. IndependentSetLP.java`
- Test LP encoder with the test instance:

  ```
  java -classpath graph.jar:. IndependentSetLP indepinst_graph.g | ./lp_solve
  ```

  OR

  ```
  python IndependentSetLP.py indepinst_graph.g | ./lp_solve
  ```

You should see output identical to

```
Value of objective function: 5.00000000

Actual values of the variables:
n1                              1
n10                             1
n2                              0
n3                              0
n4                              0
n5                              1
n6                              1
n7                              0
n8                              1
n9                              0
```