Computer Laboratory 2
LINEAR AND COMBINATORIAL OPTIMIZATION

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF MATHEMATICS

February 12, 2018

Preparation for the labs

Read through the whole manual and do the preparation exercises before the lab. Download the Matlab files needed for the exercises (lab1.zip and lab2.zip) from the course webpage.

Laboratory exercise 2

Read this before you come to the lab session

This lab deals with the transportation problem, the maximal flow problem, the local search method and the branch and bound method.

At the lab you will write a Matlab program that solves the transportation problem and then you should test your program. By means of an available Matlab program, you should solve a maximal flow problem for a network with 23 nodes. You should also verify that "max flow"="min cut". Finally, you should test the local search algorithm and the branch and bound algorithm on two test problems: the travelling salesman problem and the Vigenère cypher.

Before coming to the lab session, do the following: Read and make sure that you understand Examples 1, 2 and 3 of Section 5.1 of Kolman–Beck, and that you have checked the computations there by hand. Read the section about the Vigenère cipher in the lecture notes for Lecture 9. Read through the rest of the instructions for the lab, so that you know what you will do in the lab session.

At the lab session

Download the Matlab files needed for the session from the course home page. Copy this file to your home directory, decompress and unpack it. Now you have a subdirectory `lab2`. Start Matlab, and go to this directory. Information about the routines in the directory can be found in the text file `Contents.m`.

The transportation problem

Run the program `transportmovie.m` on Example 1 of Section 5.1 in the book with the following commands:

```
>> example511;
>> transportmovie(s,d,c);
```

Then try the same thing on example512 and example513.
Using the subroutines `northwest`, `multipliers` and `cycle` that you can find in the subdirectory lab2, write a Matlab program with the feature

```
function [x,cost]=transport(s,d,c);
% [x,cost]=transport(s,d,c)
% Input:
%    s  =  supplies           (mx1)
%    d  =  demands            (nx1)
%    c  =  costs              (mxn)
% Output:
%     x  =  optimal solution  (mxn)
%  cost  =  minimal transport cost
```

Test your codes using the files `example511`, `example512` and `example513`, whose solutions are given by

511:

$$\mathbf{x} = \begin{bmatrix} 100 & 0 & 20 & 0 \\ 0 & 60 & 60 & 20 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

with optimal cost 1900,

512:

$$\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 30 & 70 \\ 20 & 60 & 80 & 0 & 0 \\ 70 & 0 & 0 & 70 & 0 \end{bmatrix}$$

with optimal cost 1930, and

513:

$$\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 30 & 70 \\ 20 & 60 & 80 & 0 & 0 \\ 30 & 0 & 0 & 70 & 0 \end{bmatrix}$$

with optimal cost 1730.

The maximal flow problem

In this session you will find the maximal flow in the network in Figure 1, from node 10 to node 9. Run the program `maxflow.m`. When you have found a new breakthrough for the flow and input it, the graph will be updated automatically and the arrows will show in which direction of the arcs more flow can be shipped. When you think you have found the optimal solution you should divide the nodes according to the max flow-min cut theorem. Mark the solution in the figure. Divide the nodes by a minimal cut into two groups M and N such that the source node belongs to M and the sink node to N. Make this division by hand with help of the result from the run. Then input the two sets in matlab and check if you have found a minimum cut.

Local search and branch-and-bound

The purpose of this part of the lab is to study the local search (steepest descent) method and the branch-and-bound method on two problems. In the directory `lab2`, there are two subdirectories `@vigcrypto` and `@tsp`. These contain methods for two new objects: `vigcrypto` and `tsp` objects. You can create new objects of these types using

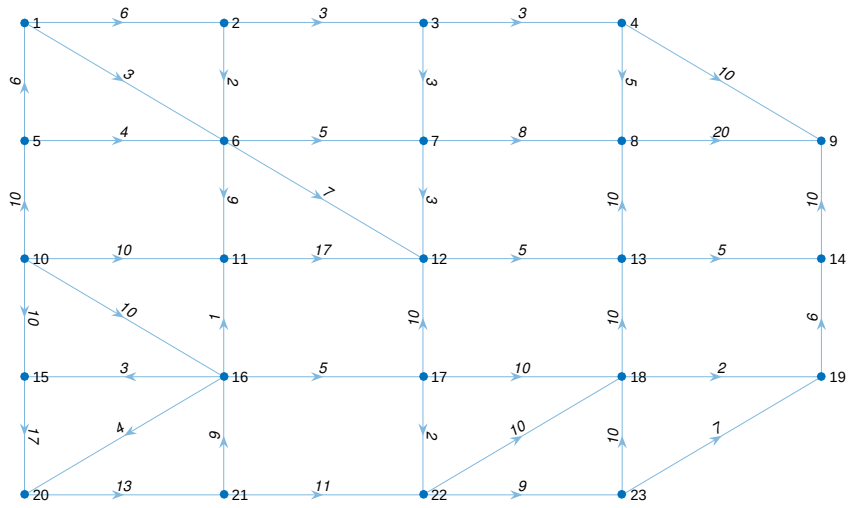

Figure 1: Matlab's graph of the original network

```
>> problem = demoproblem(tsp);
>> problem = demoproblem(vigcrypto);
```

You can also construct other instances of the problem classes above using the constructors

```
>> problem = tsp(relevantdata);
>> problem = vigcrypto(relevantdata);
```

To each of these objects, a number of methods is given. For example

```
>> x=randomindomain(problem);
```

generates a representative `x` for a point in the domain of the combinatorial optimization problem. In general, the points `x` are represented as row matrices.

```
>> f=evaluate(problem,x);
```

evaluates the objective function `f` at the point `x` in the domain of the combinatorial optimization problem. Each row of the matrix `xlist` is a representative of a point (a neighbour) close to `x`.

```
>> D=getdomain(problem);
```

generates a representative `D` of the whole domain of the problem. Subsets are also represented as a row matrix.

```
>> [fl,f,fu]=bound(problem,subset);
```

calculates upper `fu` and lower `fl` bounds on the optimal value of the function `f` in `subset`. More information can be found in `Contents.m` and in the comments in each file. Try for example

```
>> help lab2
>> help tsp
>> methods tsp
>> help tsp/evaluate
>> help branchandbound
```

For the vigcrypto, there is also a function `describe.m`, which can be used for viewing your deciphered text. You can change the text to be enciphered/deciphered by changing `vigtext1` to one of `vigtext2`, ..., `vigtext6` in the file `demoproblem.m`.
Copy the code in `steepdescstep.pre.m` to `steepdescstep.m`. Modify the code so that the function returns the neighbour with the lowest value of the objective function and a boolean `lokmin` which indicates whteher `xin` is a local minimum to the problem.

Now try the routine `steepdesc` (which uses your `steepdescstep` routine) on both a travelling salesman problem and the vigcrypto problem. Generate problem objects with

```
>> problem = demoproblem(tsp);
>> problem = demoproblem(vigcrypto)
```

Try with different random starting points. Does the routine lead to different local minima or does it most often end up in the same one?
Type

```
>> edit branchandbound
```

to see the code for the branch and bound algorithm. Try the algorithm

```
>> [dmin,fumin,res]=branchandbound(problem);
```

on both a travelling salesman problem and on the vigcrypto problem. Did the local search find the global minimizer? To view the decyphered (optimal) text for the vigcrypto, write

```
>> f=describe(problem,dmin)
```

Experiment with using different passwords (and different lengths) and different texts for encryption.