

Laboratory Exercise 2

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

You should write a MATLAB program that solves the transportation problem and then test on some problems.

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 vigenere crypto analysis.

The transportation problem

PREPARATORY EXERCISE. Write a MATLAB program with the feature

```
function [x,cost]=transport(s,d,c);
% [x,cost]=transport(s,d,c)
% Input:
%   s = supplies      (m*1)
%   d = demands      (n*1)
%   c = costs        (m*n)
% Output
%   x = optimal solution (m*n)
%   cost = minimal transport cost
...

```

Use the following subroutines, which you can find in your subdirectory lab2.

```
function [x,b]=northwest(s,d)
% [x,b]=northwest(s,d)
% x: shipments using nw-rule (m*n)
% b: 1 for each basic variables 0 for nonbasic (m*n)
% s: supplies (m*1)
% d: demands (n*1)
if (sum(s)~=sum(d)),
    disp('ERROR: The total supply is not equal to the total demand. ');
    return;
end
m=length(s);
n=length(d);

```

```

i=1;
j=1;
x=zeros(m,n);
b=zeros(m,n);
while ((i<=m) & (j<=n))
    if s(i)<d(j)
        x(i,j)=s(i);
        b(i,j)=1;
        d(j)=d(j)-s(i);
        i=i+1;
    else
        x(i,j)=d(j);
        b(i,j)=1;
        s(i)=s(i)-d(j);
        j=j+1;
    end
end
end

```

```

function [u,v]=multipliers(x,c,b)
% [u,v]=multipliers(x,c,b)
% x: current solution (m*n)
% b: 1 for each basic variables 0 for nonbasic (m*n)
% c: costs (m*n)
% u: lagrange multipliers for rows (m*1)
% v: lagrange multipliers for columns (n*1)
[m,n]=size(x);
if sum(sum(b))< m+n-1
    disp('Error in multipliers')
    break
else
    u=Inf*ones(m,1);
    v=Inf*ones(n,1);
    u(1)=0; % choose an arbitrary multiplier = 0
    nr=1;
    while nr<m+n % until all multipliers are assigned
        for row=1:m
            for col=1:n
                if b(row,col)>0
                    if (u(row)~=Inf) & (v(col)==Inf)
                        v(col)=c(row,col)-u(row);
                    end
                end
            end
        end
        nr=nr+1;
    end
end

```



```

else % column search
    i=1;
    while ~rowsearch
        if (b(i,loop(1,2))~=0) & (i~=loop(1,1))
            loop=[i loop(1,2) ; loop];
            rowsearch=1;
        elseif i==m
            b(loop(1,1),loop(1,2))=0;
            loop=loop(2:length(loop),:);
            rowsearch=1;
        else
            i=i+1;
        end
    end
end
end
end
end
% compute maximal loop shipment
l=length(loop);
theta=Inf;
minindex=Inf;
for i=2:2:l
    if x(loop(i,1),loop(i,2))<theta,
        theta=x(loop(i,1),loop(i,2));
        minindex=i;
    end;
end
% compute new transport matrix
y(row,col)=theta;
for i=2:l-1
    y(loop(i,1),loop(i,2))=y(loop(i,1),loop(i,2))+(-1)^(i-1)*theta;
end
bout(row,col)=1;
bout(loop(minindex,1),loop(minindex,2))=0;

```

The maximal flow problem

You should run an available MATLAB program and find the maximal flow in the network in Figure 1 *from node 10 to node 9*. When you have found a new break-

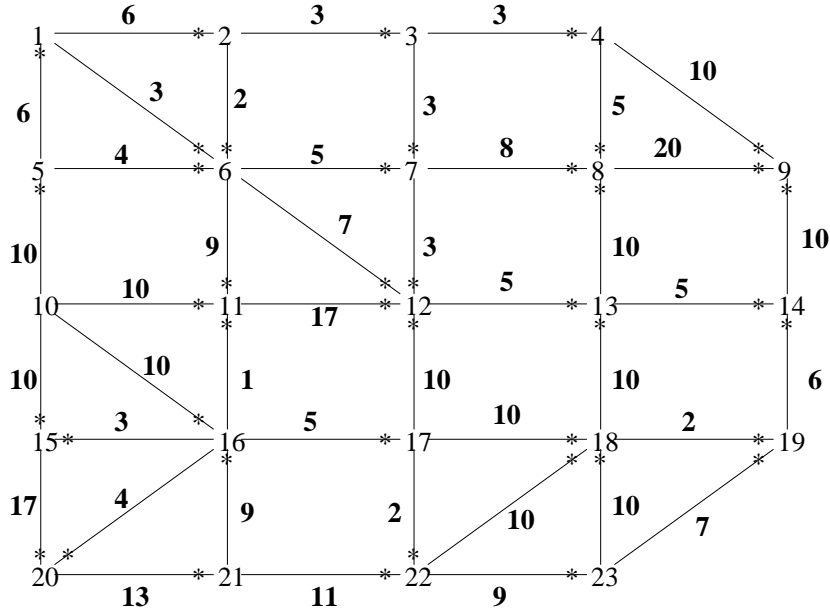


Figure 1: MATLAB's graph of the network. The capacities of the arcs at the start are shown with bold figures and the asterisks symbolizes arrows, e.g. 6 units can be shipped from node 1 to node 2.

through for the flow and input it, the graph will be updated automatically and the asteriks (=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 node according to the max flow-min cut theorem.

Local search and Branch-and-bound

Study the local search (Steepest descent) method and the branch and bound method on two problems. In the directory `lab2` are two subdirectories `@vigcrypto` and `@tsp`. These contain methods for two new objects: `vigcryptot` and `tsp` objects. One can create new objects of these types using

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

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

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

To each of these object a number of methods are 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 goal function f at the point x in the domain of the combinatorial optimization problem.

```
xlist = getneighbours(problem,x);
```

generates a list $xlist$ of all neighbours to the point x in the domain of the 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.

```
[listofsubsets,sizes]=branch(problem,S)
```

generates a list of representatives of subsets to the set S .

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

calculates upper fu and lower fl bounds on the optimal value of the function f in the subset. More information can (hopefully) 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
```

At the computer

1. Download the matlab files needed for the laboration at

```
http://www.maths.lth.se/matematiklth/personal/kalle/kombopt/lab2.tar.gz
```

Copy this file to your home directory, decompress and unpack it.

```
gzip -d lab2.tar.gz
tar xvf lab2.tar
```

Now there is a subdirectory `lab2`. Go to the subdirectory `cd lab2`, start matlab and make sure matlab search path includes the directory `lab2`. This can be done in matlab using the `path` command.

```
>> path(path,pwd);
```

Information about the routines in the directory can be found in the text file `Contents.m`.

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

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

Compare the result with your own calculations by hand! Observe that the matrices s , d and c for Examples 1, 2, 3 of Section 5.1 in the book is saved in the m-files `example511.m`, `example512.m` and `example513.m`.

3. Use Emacs to write your program `transport.m` and test with data from the examples and also with data from exercises 9 and 11 of Section 5.1.1.
4. Solve the maximal flow problem by running `maxflow`. Mark the solution in Figure 1. Divide the nodes by a minimal cut into two groups M och M' such that the source belongs to M and the sink to M' . (Make this division by hand with help of the result from the run).
5. Copy the code in `steepdescstep.pre.m` to `steepdescstep.m`. Modify the code so that the function returns the neighbour with the lowest value on the goal function and a boolean `lokmin` which indicates whether `xin` is a local minimum to the problem.

```

function [xout,lokmin]=steepdescstep(problem,xin);
% function [xout,lokmin]=steepdescstep(problem,xin);
% One step in local search
% Input:
%   problem - The optimization problem.
%   xin     - a point in the domain of problem.
% Output:
%   xout    - the neighbour to xin with the lowest
%             goal function.
%   lokmin  - 1 if xin is a local minimum, 0 otherwise

% Generate all neighbours to xin.
neighbours=getneighbours(problem,xin);
lokmin=1;
f=evaluate(problem,xin);
xout=xin;
for ii=1:size(neighbours,1);
    y=neighbours(ii,:);
    fnew=evaluate(problem,y);
    ...
    add appropriate code here.
    ...
end;

```

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);

```

Generate starting points for steepest descent with

```

xin=randomindomain(problem);

```

Try with different random starting points. Do the routine end up in different local minima or most often in the same?

```

function [xout,steps,locmin]=steepdesc(problem,xin);
% function [xout,steps,locmin]=steepdesc(problem,xin);

```

```
if nargin < 2,
    xin = randomindomain(problem);
end;

steps=0;
locmin=0;
xout=xin;
while ~locmin,
    [xout,locmin]=steepdescstep(problem,xout);
    steps=steps+1;
    % f = evaluate(problem,xout)
end;
```

6. Type

```
>> type 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 minimum?