Combinatorial optimisation
Seminar No. 1
An introduction to the experimental environment

Roman Čapek, Přemysl Šůcha (capekrom@fel.cvut.cz, suchap@fel.cvut.cz)

February 8, 2013

1 TORSCHE

TORSCHE is a Matlab toolbox which is designed for developing scheduling and graph algorithms. This toolbox is used in our seminar since toolbox functions and objects are very helpful in working with graphs and the toolbox includes an utility for creating and editing graphs.

1.1 Graph representation

Definition 1.1 *Directed graph G consists of directed edges E, nodes V, and incidence projection $\epsilon : E \to V^2$, which assigns the* ***ordered pair*** *of nodes (x, y) to each directed edge $e \in E$. We say that edge (x, y) leaves x and enters y.*

Nevertheless, not all problems have to be formulated by a directed graph since directions of edges are not always necessary. For this purpose, an *undirected graph* is devised. This graph is similar to a directed graph, but the difference is that the projection $\epsilon : E \to V^2$ assigns **unordered pair** of nodes $\{x, y\}$ to each edge $e \in E$ [1].

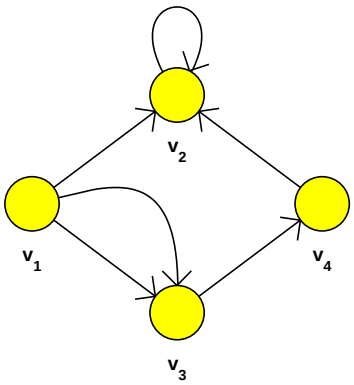

Figure 1: An example of a directed graph G.

Every graph can be stored in a matrix, which describes the graph structure. For example, an *adjacency matrix*, which is denoted as M_G^+, is a matrix where each element m_{ij}^+ is equal to $m_{ij}^+ = m^+(v_i, v_j)$ where m^+ is the number of edges from v_i to v_j. The adjacency matrix of the

graph in Figure 1 is shown below.

$$M_G^+ = \begin{pmatrix} 0 & 1 & 2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{1}$$

If directions are not taken into account, the matrix is changed as follows.

$$M_G = \begin{pmatrix} 0 & 1 & 2 & 0 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{2}$$

Another matrix form of a graph is an *incidence matrix*. Dimension of this matrix is $n \times m$ where n is the number of nodes and m is the number of edges. In case of directed graph G without loops, the incidence matrix B_G has the following form.

$$b_{i,j} = \left\{ \begin{array}{ll} 1 & \text{if edge } e_j = (v_i, v_k) \text{ leaves } v_i, \\ -1 & \text{if edge } e_j = (v_k, v_i) \text{ enters } v_i, \\ 0 & \text{otherwise.} \end{array} \right) \tag{3}$$

As an example, the incidence matrix of the graph in Figure 1 can be seen below.

$$B_G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 \\ 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \tag{4}$$

The incidence matrix can also be formulated in the case of an undirected graph.

$$b_{i,j} = \left\{ \begin{array}{ll} 1 & \text{if } v_i \text{ is incident with edge } e_j \\ 0 & \text{otherwise} \end{array} \right. \tag{5}$$

Ignoring edge directions, the incidence matrix of the graph in Figure 1 is shown below.

$$B_G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{6}$$

In many problems, edges are weighted, i.e. a value (e.g. distance, cost, reliability, ...) is assigned to each edge. A *weighted graph* is a directed or undirected graph where all edges are weighted. A weighted graph can be stored in *matrix of weights* W on condition that the graph is a *simple graph*, i.e. no parallel edges and loops are in the graph. Matrix element $w_{i,j}$ corresponds to a weight of the directed or undirected edge. If there is no such an edge in G, weight $w_{i,j}$ will be set to infinity.

2 A seminar assignment

Software set up

Download and install the unstable version of the TORSCHE toolbox from the homepage KO. The stable version can also be downloaded from `http://rtime.felk.cvut.cz/scheduling-toolbox` but this version is not preferred in seminars. Unzip the downloaded file and add path *scheduling* to the Matlab.

Creating and editing graphs in TORSCHE

In the TORSCHE toolbox, the graph can be created using an adjacency matrix, an incidence matrix, or a matrix of weights. To create a graph the function `graph` can be used. Using an adjacency matrix, a graph can be created by following commands.

```
>> M = [0 1 2 0; 0 1 0 0; 0 0 0 1; 0 1 0 0];
>> g = graph('adj', M)
```

Or you can use an incidence matrix.

```
>> B = [ 1 1 1 0 0; -1 0 0 0 -1; 0  -1 -1 1 0; 0 0 0 -1 1];
>> g = graph('inc', B)
```

In case of a weighted graph, a graph is created in the following way.

```
>> W = [inf 6 4 inf; inf inf inf inf; inf inf inf 2; inf 3 inf inf];
>> g = graph(W)
```

To get a graph structure the command `get` is useful. For example, a node name is retrieved by the following commands.

```
>> get(g)
>> g.N(1).Name;
```

In addition, the TORSCHE toolbox enables us to edit graphs. A graph editor can be executed by `graphedit` command. An existing graph can be opened as it is shown below.

```
>> graphedit(g)
```

Having done fine adjustments, the graph will be looking in a similar way as the graph in Figure 2. For more details see TORSCHE documentation.

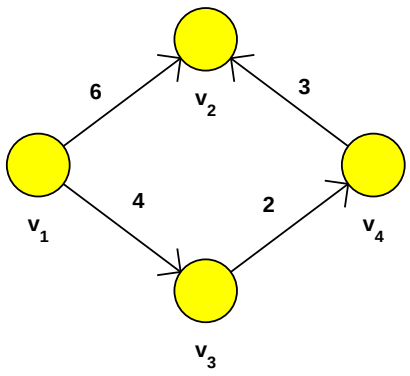

Figure 2: An example of graph g.

To access edge weights the functions `edges2matrixparam` and `matrixparam2edges` can be used. The weights of graph `g` are retrieved by the following command.

```
>> W2 = edges2matrixparam(g)
```

The TORSCHE toolbox provides a lot of graph algorithms which can be applied directly to graphs. In the next task we demonstrate how the TORSCHE toolbox can be used for searching a Hamiltonian circuit.

2.1 Searching a Hamiltonian circuit using TORSCHE

Several cities of the Czech Republic and distances among them are given. The goal is to find the cheapest closed path (Hamiltonian circuit) going through all cities on condition that each city is once visited with the exception of the start position. The cities correspond to the nodes and the edges are connections among cities. All edges are weighted according to the distances among cities, therefore the cheapest path is the shortest one. To solve this issue the theory of graphs and the TORSCHE toolbox are used. Using the graphedit tool, we create the same graph as it is shown in Figure 3. You can use a property editor to input cities names and edge weights (i.e. distances).

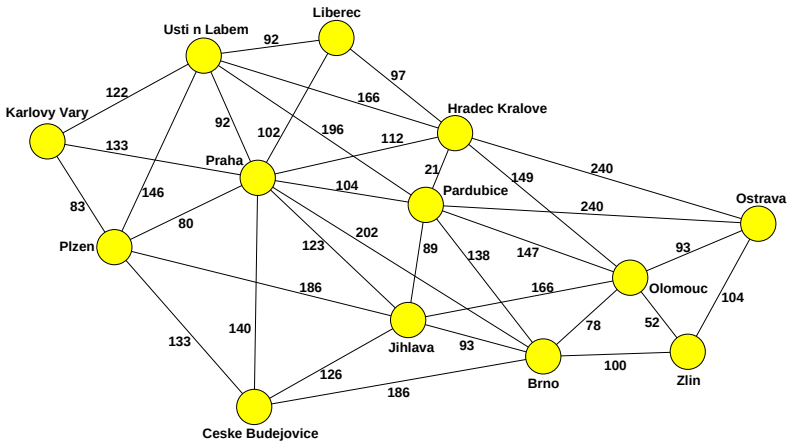

Figure 3: The graph of the cities.

To deal with the problem, the `hamiltoncircuit` function is available in the toolbox. It has two parameters, where the second one is optional. The first parameter is a weighted graph and the second one determines whether directions of edges should be consider. In our case, the directions of the edges are ignored, therefore the following command will be used.

```
>> gHam = hamiltoncircuit(g, 'u');
>> graphedit(gHam, 'position', [1300 50 600 1050], 'fit')
```

This way the Hamiltonian circuit is found (see Figure 4). We would like to get the order of the cities, theirs coordinates, and the total tour distance. Using the graph, we get a list of the circuit edges and transform it into a matrix.

```
>> edges = cell2mat(gHam.edl);
```

After that, we display the cities and theirs coordinates in the correct order.

```
>> actEdge = 1;
>> for i = 1:size(edges, 1)
>>  actCity = edges(actEdge, 1);
>>  cityName = g.N(actCity).Name;
>>  cityName(size(cityName, 2)+1:18) = ' ';
>>  x = g.N(actCity).graphicParam(1).x;
>>  y = g.N(actCity).graphicParam(1).y;
>>  str = sprintf('City: %sCoordinates(x,y): %g, %g', cityName, x, y);
>>  disp(str)
>>  actEdge = find(edges(:, 2) == actCity);
>> end
```

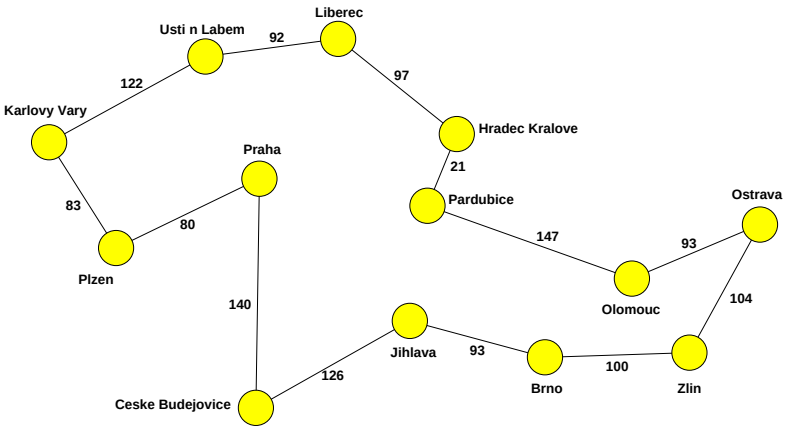

Figure 4: The Hamiltonian circuit.

In the end we compute the total tour distance as the sum of the edge weights.

```
>> distance = sum(edges(:,3));
>> disp(['Total distance: ' num2str(distance) ' km'])
```

A seminar assignment: Use the TORSCHE toolbox to find the Hamiltonian circuit. The resulting circuit should be displayed in a similar way as it is shown in Figure 4. The total tour distance should also be printed to a console.

3 A homework assignment - graph colouring

A homework assignment: Create a graph model of the Sudoku puzzle. The grid size is 4×4, therefore a one of the numbers $1 \ldots 4$ has to be assigned to each box. The goal is to find such a number-placement that all numbers are different in each subregion 2×2, column and row. To solve the puzzle the graph colouring algorithm should be used. Each node corresponds to a box in the grid and a node colour corresponds to a number (see Figure 5). A model must meet the requirements of the puzzle, i.e. some edges have to be added to the graph to ensure correct graph colouring. The graph has to be created and edited using command line. The graphedit tool should only be used for displaying the final graph.

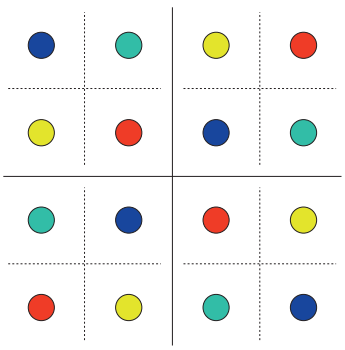

Figure 5: The Sudoku puzzle

Hints: Using an adjacency matrix create the graph with 16 nodes and connect each pair of nodes where the same number is not allowed. Afterwards, the graph colouring algorithm `graphcoloring` can be executed to solve the puzzle. And finally, remove node names, adjust node positions according to the box positions in the puzzle, and display the graph.

References

[1] J. Demel, *Grafy a jejich aplikace*. Academia, second ed., 2002.

[2] B. H. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*. Springer, third ed., 2006.