Kombinatorická optimalizace
cvičení č.1
Seznámení s experimentálním prostředím

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

16. února 2011

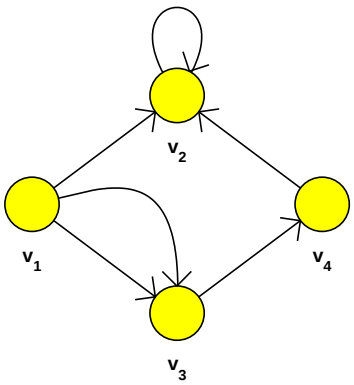
1 TORSCHE

TORSCHE je toolbox pro Matlab a je určený především pro vývoj rozvrhovacích a grafových algoritmů. V našem předmětu využijeme především možností grafové části tohoto nástroje. Připravené objekty toolboxu umožňují přehlednější a systematičtější práci s grafy. Navíc jsou zde připraveny podpůrné utility, které například umožňují vytvářet a upravovat grafy na grafické úrovni.

1.1 Reprezentace grafu

Definice 1.1 *Orientovaný graf je trojice $G = (V, E, \epsilon)$ tvořená neprázdnou konečnou množinou V, jejíž prvky nazýváme vrcholy, konečnou množinou E, jejíž prvky nazýváme orientovanými hranami, a zobrazením $\epsilon : E \to V^2$, které nazýváme vztahem incidence. Toto zobrazení přiřazuje každé hraně $e \in E$* ***uspořádanou*** *dvojici vrcholů (x, y).*

Někdy je orientace hran v grafu nepodstatná. Pro takový případ zavádíme pojem *neorientovaný graf*. Ten se liší od orientovaného grafu tím, že zobrazení $\epsilon : E \to V^2$ přiřazuje každé hraně $e \in E$ **neuspořádanou** dvojici vrcholů $\{x, y\}$ ("zapomeneme" na pořadí vrcholů x a y) [1].



Obrázek 1: Příklad grafu G.

Grafy lze popsat pomocí několika typů matic. První je takzvaná *matice sousednosti* (nebo také *adjacenční*), která se označuje M_G^+. Pro její prvky m_{ij}^+ platí $m_{ij}^+ = m^+(v_i, v_j)$; m^+ je počet hran z vrcholu v_i do vrcholu v_j. Matice sousednosti odpovídající grafu z obrázku 1 vypadá následovně

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

Pokud neuvažujeme orientaci hran, matice sousednosti vypadá následovně

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

Dalším způsobem reprezentace grafu je *incidenční matice*. Pro orientovaný graf G bez smyček zvolíme nejen pořadí vrcholů $v_1, ..., v_n$, ale i pořadí hran $e_1, ..., e_m$. Potom můžeme reprezentovat graf incidenční maticí B_G typu (n, m) předpisem

$$b_{i,j} = \left\{ \begin{array}{ll} 1 & \text{jestliže } v_i \text{ je počátečním vrcholem hrany } e_j, \\ -1 & \text{jestliže } v_i \text{ je koncovým vrcholem hrany } e_j, \\ 0 & \text{v ostatních případech.} \end{array} \right) \tag{3}$$

Incidenční matice grafu G z obrázku 1 je

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

Pro neorientované grafy bez smyček se incidenční matice definuje předpisem

$$b_{i,j} = \left\{ \begin{array}{ll} 1 & \text{jestliže } v_i \text{ je incidentní s hranou } e_j \\ 0 & \text{v ostatních případech} \end{array} \right. \tag{5}$$

Incidenční matice grafu G z obrázku 1 pokud neuvažujeme orientace hran je

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

Velmi často jsou hrany v grafu ováženy nějakou číselnou hodnotou. Potom je nejčastěji takový graf reprezentován maticí vah W, kde na pozici $w_{i,j}$ je váha hrany z vrcholu v_i do vrcholu v_j. Předpokladem je, že graf je prostý a tudíž nemá paralelní hrany.

1.2 Grafy a TORSCHE

TORSCHE toolbox si stáhněte ze stránek předmětu KO, kde je k dispozici testovací verze, kterou budeme na cvičeních používat. Stabilní oficiální verzi naleznete na adrese `http://rtime.felk.cvut.cz/scheduling-toolbox`, v sekci download vyplníte požadované údaje a obdržíte email s odkazem na stažení toolboxu. Po rozbalení staženého archivu je v Matlabu potřeba nastavit cestu do adresáře *scheduling*.

V toolboxu TORSCHE lze grafy vytvářet jak z matic sousednosti a incidenčních matic, tak z matic vah. Pro vytvoření grafu slouží funkce `graph`. Z matice sousednosti lze graf vytvořit pomocí následujícího příkazu.

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

Z incidenční matice

```
>> 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)
```

Pokud je graf zadán maticí vah, je možno graf vytvořit pomocí příkazu

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

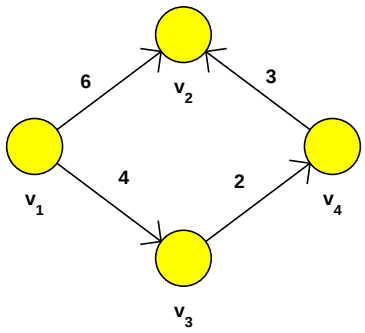
Chceme-li se podívat na základní parametry grafu, použijeme příkaz `get`, k jednotlivým položkám pak lze přistupovat přes tečku.

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

Graf je možné vytvářet i graficky pomocí editoru grafů, který se spouští příkazem `graphedit`. Pokud v něm chceme otevřít existující graf, stačí zadat příkaz

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

Po drobných úpravách dříve vytvořeného grafu, bude graf vypadat tak, jak je ukázáno na obrázku 2. Detailní popis *grapheditu* najdete v dokumentaci k TORSCHE.



Obrázek 2: Příklad grafu g.

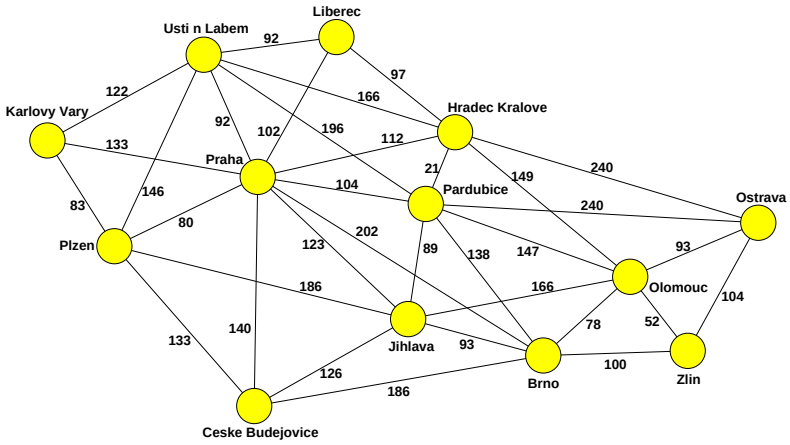
K váhám hran lze přistupovat pomocí funkcí `edges2matrixparam` a `matrixparam2edges`. Váhy hran grafu `g` lze získat příkazem

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

Na vytvořené grafy lze zavolat grafové algoritmy, které jsou součástí TORSCHE, nebo je možné přidat svůj vlastní grafový algoritmus. V následující kapitole si ukážeme příklad aplikace, v níž je na vytvořený graf použit algoritmus hledající hamiltonovskou kružnici v grafu.

2 Příklad na hamiltonovskou kružnici

Máme dáno několik měst České republiky a spojnice mezi nimi s udanou vzdáleností. Naším úkolem je naplánovat uzavřenou cestu skrze všechna města tak, abychom každé navštívili právě jednou. K vyřešení této úlohy použijeme teorii grafů a samozřejmě TORSCHE toolbox. Města si můžeme představit jako

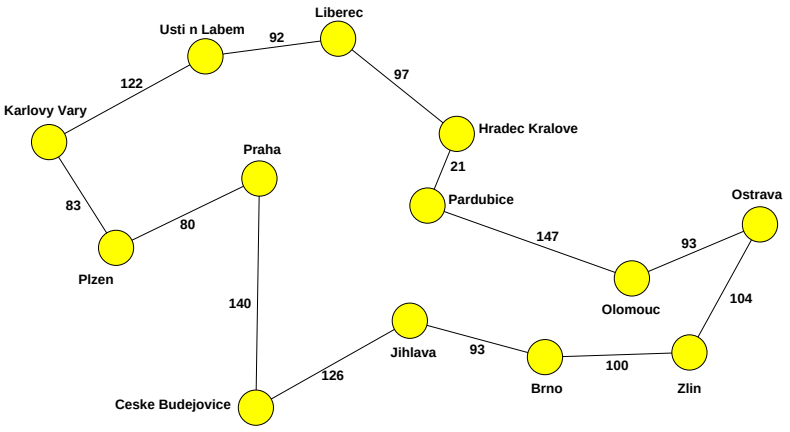


Obrázek 3: Graf měst

vrcholy grafu a spojnice mezi nimi jako hrany grafu s váhou odpovídající dané vzdálenosti. S použitím Grapheidtu se pokusíme vytvořit graf `g` jako je na obrázku 3, pro zadání jmen měst a vzdáleností ke hranám slouží Property Editor.

Tato úloha se dá řešit pomocí hledání hamiltonovské kružnice v grafu, tedy uzavřené cesty, procházející každým vrcholem právě jednou. Cílem je najít takovou cestu, kde součet vah hran bude minimalní. V toolboxu je implementována funkce `hamiltoncircuit`, jejímž prvním vstupním argumentem je graf a druhý, volitelný, udává, zda se má brát v potaz orientace hran či nikoliv. V našem případě nás orientace nezajímá a tedy použijeme následující příkaz.

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



Obrázek 4: Hamiltonovská kružnice

Tím jsme úlohu úspěšně vyřešili (viz obrázek 4), ale zajímalo by nás nyní pořadí měst, v jakém se mají navštívit, dále jejich souřadnice na mapě a také celková vzdálenost. Z grafu s hamiltonovskou kružnicí nejprve získáme seznam všech hran použitých v cestě a převedeme ho na matici.

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

Nyní ve smyčce projdeme všechny hrany tak, aby na sebe navazovaly a vypíšeme přitom názvy měst a jejich souřadnice.

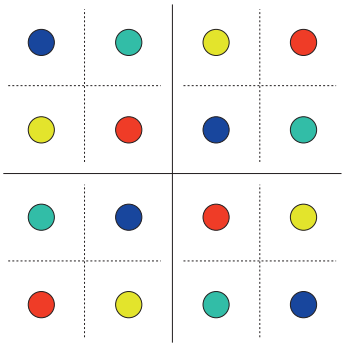
```
>> actEdge = 1;
>> for i = 1:size(edges, 1)
>>  actCity = edges(actEdge, 1);
>>  cityName = g.N(actCity).Name;
>>  cityName(size(cityName, 2)+1:18) = ' ';
>>  x = g.N(actCity).graphicParam(1).x;
>>  y = g.N(actCity).graphicParam(1).y;
>>  str = sprintf('City: %sCoordinates(x,y): %g, %g', cityName,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

Nakonec sečteme celkovou vzdálenost, tedy váhy všech použitých hran.

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

3 Příklad na využití barvení grafu

Úkol: Vytvořte grafový model sudoku s čísly od $1 \ldots 4$ a tabulkou velikosti 4×4. Každý vrchol grafu bude odpovídat jednomu políčku v tabulce a jeho barva bude odpovídat číslu, které je mu přiřazeno. Model musí splňovat klasická omezení sudoku - v každém řádku, každém sloupci a každé buňce velikosti 2×2 musí být každé číslo právě jednou. K vyřešení úlohy použijte algoritmus na barvení grafu a jednotlivé vrcholy rozmístěte do řádků a sloupců viz obrázek 5, kde každá barva odpovídá jednomu číslu. Vytvoření grafu a veškeré jeho úpravy provádějte z příkazové řádky, Graphedit použijte jen k vykreslení výsledku.



Obrázek 5: Sudoku

Postup řešení: Vytvořte si pomocí adjacenční matice graf o 16 vrcholech a spojte hranou každé dva vrcholy, ve kterých nesmí být stejné číslo. Poté stačí zavolat algoritmus `graphcoloring` a uzpůsobit grafickou podobu tak, aby pozice vrcholů odpovídala rozmístění sudoku. Na závěr ještě odstraňte názvy vrcholů a vykreslete graf pomoci Grapheditu.

Reference

[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.