Optimalizační problémy a aproximační algoritmy

PETR OSIČKA

Přírodovědecká fakulta UP, Olomouc

Mnoho lidí již řešilo problém nalezení nejkratší cesty mezi dvěma místy. Velmi často, když navštívíme nějaké nové místo, a chceme se dostat k nějaké jeho zajímavé části, například nějaké památce, spustíme na mobilním telefonu aplikaci s mapou a necháme ji najít nejkratší cestu z místa, kde stojíme, do cílového místa. V článku si ukážeme abstraktní problém hledání nejkratších cest v grafu, který zachycuje podstatu problému řešeného mobilní aplikací hledající cestu na mapě. Uvidíme, že tento problém má jasně definovanou strukturu: žádá se v něm, abychom našli objekt s nejmenší cenou z přesně definované množiny objektů. Takovou strukturu má velká skupina algoritmických problémů, takzvané *optimalizační problémy.*

Budeme se věnovat také algoritmům pro optimalizační problémy. Uvidíme, že existují optimalizační problémy, které jsou snadné v tom smyslu, že existuje efektivní algoritmus, který vždy najde optimální řešení. Příkladem takového problému je právě problém nalezení nejkratší cesty v grafu. Existují ovšem i optimalizační problémy, které snadné nejsou. U takových problémů se musíme smířit s tím, že efektivní algoritmus nemůže vždy nalézt optimální řešení. Můžeme se ovšem snažit se optimálnímu řešení co nejvíce přiblížit. Takovému efektivnímu, ale přitom neoptimálnímu algoritmu říkáme *aproximační algoritmus.*

1. Optimalizační problémy

V abstraktní verzi problému vyhledávání nejkratší cesty na mapě hledáme nejkratší cestu mezi dvěma uzly v grafu. *Graf* je matematický objekt, který je tvořen množinou *vrcholů* V a množinou *hran* H, přitom každá z hran je dvojice vrcholů z V. Můžeme jej jednoduše zobrazit tak, že vrcholy namalujeme jako malá kolečka a hrany jako úsečky spojující vždy dvojici koleček, která dané hraně odpovídají. Například graf s vr-

choly $V = \{a, b, c, d, e, f\}$ a hranami[1)]

$$H = \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, d\}, \{b, e\}, \{c, e\}, \{d, e\}, \{e, f\}, \{d, f\}\} \quad (1)$$

je na obr. 1 (a).

Cesta z vrcholu u do vrcholu v v grafu je posloupnost vzájemně různých vrcholů

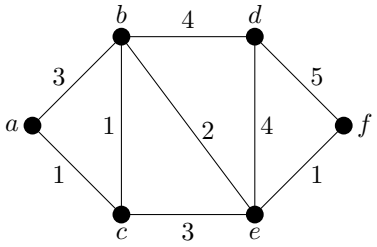
$$u = u_1, u_2, \ldots, u_k = v$$

takových, že pro $i = 1, 2, \ldots, k - 1$ jsou vždy vrcholy u_i a u_{i+1} spojeny hranou. O takové hraně pak řekneme, že na cestě leží. V našem příkladě je tedy posloupnost a, b, c, e, f cestou z vrcholu a do vrcholu f, ale posloupnost a, c, d, f cestou není, protože mezi vrcholy c a d není hrana. Pro danou dvojici u, v může být v grafu cest z u do v více, případně nemusí existovat žádná. Ve zbytku textu budeme předpokládat, že pro každou dvojici vrcholů existuje v grafu existuje alespoň jedna cesta. Cest z vrcholu a do vrcholu f je v grafu na obr. 1 (a) celkem 13. Můžeme si je prohlédnout na obr. 1 (b).

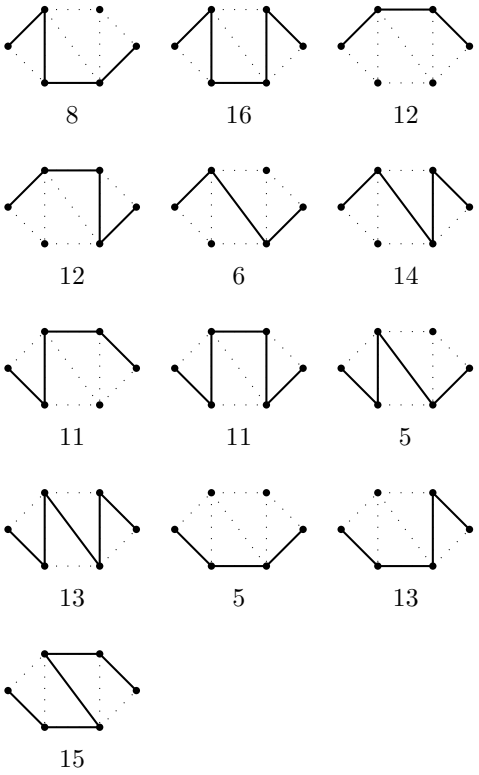
Jaký je vztah mapy města a grafu? Vrcholy grafu můžeme považovat za místa na mapě města. Hrana mezi dvěma vrcholy grafu pak říká, že mezi místy, která spojuje, můžeme přejít bez projití některým z ostatních míst. Vhodnými kandidáty pro vrcholy pak jsou například křižovatky ulic, významná místa ve městě a podobně, hrany v takovém případě mohou představovat ulice nebo chodníčky v parku. Každé cestě v grafu tedy odpovídá jedna cesta městem. Z mapy lze obvykle vyčíst vzdálenosti mezi jednotlivými místy na mapě. Do grafu tyto vzdálenosti přirozeně zařadíme tak, že tuto vzdálenost přiřadíme k hranám. Uděláme to tak, že zavedeme funkci δ, budeme jí říkat *ohodnocení hran*, která hraně přiřadí kladné racionální číslo. Délku cesty $u_1, u_2, \ldots, u_k$ pak spočítáme jako součet ohodnocení hran, které na této cestě leží. Nejkratší cesta na mapě tak odpovídá nejkratší cestě v grafu.

Problém nalezení nejkratší cesty je svou formou podobný velké množině jiných problémů, kterým souhrně říkáme *optimalizační problémy.* Jsou charakteristické tím, že každý z nich můžeme úplně popsat pomocí následujících čtyř položek.

[1)] Formálně je pro nás hrana dvojprvková množina vrcholů. Hrana $\{a, b\}$ je tedy stejná jako hrana $\{b, a\}$. Hovoříme-li tedy o hraně z vrcholu a do vrcholu b, myslíme tím i hranu jdoucí z vrcholu b do vrcholu a. Hrany tedy nemají směr (orientaci). Grafy, kde na orientaci hran záleží, existují, nebudeme si jimi ale zabývat.



(a)



(b)

Obr. 1 (a) Graf s ohodnocením hran. (b) Všechny cesty z vrcholu a do vrcholu f v grafu (a) spolu s jejich délkami. Hrany ležící na cestě jsou vyznačeny tučně.

- **Množina vstupních instancí**[2]. Popíšeme, jaké objekty považujeme za platné instance problému.
- **Přípustná řešení**. Každé konkrétní instanci přiřadíme množinu tzv. přípustných řešení. Přípustné řešení považujeme za korektní výsledek.
- **Cena přípustného řešení**. Určíme, jak ocenit jednotlivá přípustná řešení. Cenou je vždy racionální číslo.
- **Cíl.** Určíme, zda-li chceme nalézt přípustné řešení s maximální cenou nebo to s minimální cenou. O takových řešeních říkáme, že jsou *optimální*.

Pro problém nalezení nejkratší cesty v grafu je vstupní instance tvořena grafem s ohodnocením hran a počátečním a cílovým vrcholem (například grafem na obrázku 1 (a) a vrcholy a a f tohoto grafu). Množina přípustných řešení je pro konkrétní takovou instanci tvořena všemi cestami z počátečního do cílového vrcholu v daném grafu. Za cenu jedné cesty považujeme její délku. (Množinu přípustných řešení a jejich cen pro náš příklad vidíme na obrázku 1 (b).) Zajímá nás nejkratší cesta, tudíž cílem je najít přípustné řešení s minimální cenou.

Nabízí se následující naivní algoritmus. Pro vstup tvořený grafem G, počátečním vrcholem u a cílovým vrcholem v, nejdříve vygenerujeme všechny cesty v G, které vedou z u do v. Potom spočítáme jejich délky a vybereme tu nejkratší. Například nejkratší cestu z a do f v grafu na obr. 1 (a) nalezneme tak, že namalujeme obr. 1 (b) a na něm najdeme cestu s minimální cenou. Po krátkém zamyšlení ovšem zjistíme, že počet cest mezi dvěma uzly v grafu může s počtem uzlů grafu růst více než polynomiálně a naivní algoritmus má proto příliš velkou časovou složitost, a to i přesto, že velikost jedné cesty (tj. počet vrcholů na cestě) je vzhledem k velikosti grafu (tj. počtu vrcholů grafu) malá.

Podobná úvaha platí pro mnoho praktických a přirozených optimalizačních problémů kombinatorické povahy. Na takové problémy se ve zbytku textu omezíme. Uveďme si proto ještě jednou explicitně, že se zajímáme o problémy, u kterých

- přípustná řešení mají vzhledem k velikosti vstupní instance nejvýše polynomickou velikost,

[2] Slovem instance označujeme konkrétní vstup algoritmického problému.

- umíme v polynomickém čase (opět vzhledem k velikosti vstupní instance) spočítat cenu přípustného řešení,
- samotných přípustných řešení může být mnoho a v tom spočívá obtížnost daného problému.

2. Dijkstrův algoritmus

V této kapitole si ukážeme algoritmus, který v polynomickém čase nalezne nejkratší cestu v grafu z počátečního do cílového vrcholu. Vstupem algoritmu je graf G s množinou vrcholů V, množinou hran H, ohodnocení hran δ, počáteční vrchol s a cílový vrchol t.

Algoritmus postupně hledá nejkratší cesty z s k ostatním vrcholům v grafu, a tedy speciálně i k vrcholu t. Začíná s prázdnou množinou vrcholů, ke kterým je známa nejkratší cesta, a v každém kroku do této množiny jeden vrchol přidá. Ke každému vrcholu u si algoritmus uchovává dvě položky. Jsou to

- $l(u)$: délka doposud nejkratší nalezené cesty z s do u. Pokud algoritmus ještě žádnou takovou cestu nenalezl, je hodnota rovna ∞. Připomeňme, že $x+\infty = \infty$ pro každé reálné číslo x a že $\infty+\infty = \infty$.
- $p(u)$: vrchol, který je na doposud nejkratší nalezené cestě z s do u těsně před vrcholem u. Pokud takový vrchol neexistuje (protože algoritmus dosud žádnou cestu nenašel, nebo protože je u prvním vrcholem na cestě), je $p(u)$ rovna $-$.

Množinu vrcholů, ke kterým již algoritmus našel nejkratší cestu, označíme Q a vrchol, ke kterému algoritmus našel nejkratší cestu v aktuálním kroku označíme v. Běh algoritmu na grafu z obrázku 1 (a) je znázorněn na obrázku 2.

Dijkstrův algoritmus
Vstupem je graf G s ohodnocením hran δ, vrcholy s a t.
Výstupem je nejkratší cesta v G z s do t

---

1. Nastav $l(s) \leftarrow 0$ a $p(s) \leftarrow -$. Pro všechny vrcholy u nerovnající se s nastav $l(u) \leftarrow \infty$ a $p(u) \leftarrow -$.
2. Nastav $Q \leftarrow \emptyset$ a $v \leftarrow s$.

3. Pokud je v vrcholem t, pak pomocí položek $p()$ vrcholů sestav nejkratší cestu z s do v a tu vrať jako výsledek. Vrácená cesta vznikne obrácením posloupnosti

$$t, p(t), p(p(t)), p(p(p(t))), \ldots, s.$$

4. Pokud v není vrcholem t, pak pro každý vrchol $u \notin Q$, do kterého vede hrana z vrcholu v, zkontroluj jestli

$$l(v) + \delta(v, u) < l(u).$$

Pokud tomu tak je, nastav

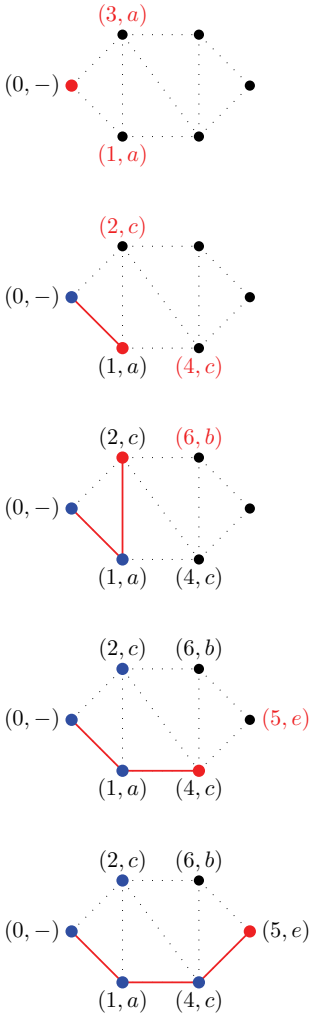
$$l(u) \leftarrow l(v) + \delta(v, u),$$
$$p(u) \leftarrow v.$$

5. Přidej vrchol v do množiny Q.
6. Nastav v na vrchol, který nepatří do Q a který má mezi vrcholy nepatřícími do Q nejmenší položku $l()$.
7. Přejdi k bodu 3.

O Dijkstrově algoritmu můžeme ukázat, že má polynomickou složitost vzhledem k počtu vrcholů grafu, ten označme n. Klíčem je pozorování, že po každém provedení kroku 5 se množina Q zvětší o 1 uzel. Tím vidíme, že kroky 3 až 6 se provedou nejvýše n-krát. Dále si všimneme, že každý z kroků 3 až 6 zabere konstantní násobek n operací, protože v nich procházíme všechny uzly a pro každý z nich provádíme pevný počet operací. Celkově je tedy složitost algoritmu přibližně n^2.

Lze také dokázat, že Dijkstrův algoritmus vždy vrátí cestu s nejmenší cenou, tedy tu nejkratší. Důkaz je pro účely tohoto textu příliš komplikovaný a proto ho vynecháme. Pro zájemce ale uveďme alespoň jeho nástin. Klíčovým pozorováním je to, že kdykoliv přiřadíme proměnné v nějaký vrchol, nalezli jsme nejkratší cestu z s do tohoto vrcholu. Toto pozorování lze dokázat matematickou indukcí přes velikost množiny Q, využije se při tom faktu, že ohodnocení hran grafu jsou nezáporná.

Dijkstrův algoritmus je tedy *korektní*, tj. vždy vrací přípustné řešení, a *optimální*, tj. vždy vrací optimální řešení. Pojem korektního a optimálního algoritmu je obecný a můžeme jej přirozeně používat i pro jiné algoritmy a jiné optimalizační problémy.



Obr. 2 Průběh Dijkstrova algoritmu na grafu z obrázku 1 (a). U vrcholů jsou zachyceny položky $l(u)$ a $p(u)$, položky s hodnotou $l(u)$ rovnou ∞ jsou vynechány. Graf je zachycen vždy po provedení bodu 4 algoritmu. Aktuální vrchol v, položky, které se během tohoto kroku změnily, a nejkratší cesta z počátečního vrcholu do v jsou zvýrazněny červeně. Vrcholy patřící do množiny Q jsou zvýrazněny modře. Pro jednoduchost jsou vynechána ohodnocení hran.

3. Obtížné problémy a aproximační algoritmy

V předchozí kapitole jsme viděli, že pro problém nalezení nejkratší cesty v grafu existuje algoritmus s polynomickou složitostí, který najde optimální řešení. Problém nalezení nejkratší cesty v grafu je z tohoto pohledu snadný. V této kapitole si ukážeme, že existují i optimalizační problémy, pro které takový algoritmus zatím neznáme a dokonce věříme, že neexistuje. Takové problémy považujeme za výpočetně obtížné. Nejdříve definujme rozhodovací problém přidružený optimalizačnímu problému.

Definice 1. Rozhodovací problém Π_{dec} přidružený minimalizačnímu problému[3)] Π je problém, jehož vstupem je instance I problému Π a racionální číslo k. Úkolem je rozhodnout, jestli existuje přípustné řešení instance I s cenou menší nebo rovnou k.

Například rozhodovací verze problému hledání nejkratší cesty je následující problém: Pro graf G, vrcholu s a t, a číslo k rozhodni, zda-li v G existuje cesta z s do t s cenou nejvýše k.

Algoritmus $\mathcal{A}$, který v polynomickém čase najde optimální řešení optimalizačního problému Π, můžeme použít k vyřešení jeho přidruženého rozhodovacího problému Π_{dec}. Pro instanci (I, k) nejdříve spočítáme pomocí algoritmu $\mathcal{A}$ optimální řešení instance I (připomeňme, že to je instance optimalizačního problému Π), poté spočítáme jeho cenu, porovnáme ji s k a na základě toho odpovíme. Každý z předchozích kroků můžeme provést v polynomickém čase, a proto celý postup můžeme považovat za algoritmus pro Π_{dec} pracující v polynomickém čase. Pokud například chceme vyřešit rozhodovací verzi problému hledání nejkratší cesty v grafu, stačí pomocí Dijkstrova algoritmu nalézt nejkratší cestu, spočítat její délku d a porovnat ji s k. Pokud je $d \leq k$, odpovíme *ano*, v opačném případě odpovíme *ne*. Vidíme tedy, že platí následující věta.

Věta 1. *Pokud pro optimalizační problém* Π *existuje optimální algoritmus s polynomickou složitostí, pak existuje algoritmus s polynomickou složitostí i pro problém* Π_{dec}.

Důležitým důsledkem předchozí věty je následující tvrzení.

Důsledek 1. *Pokud pro problém* Π_{dec} *neexistuje algoritmus s polynomickou složitostí, neexistuje ani optimální algoritmus s polynomickou složitostí pro problém* Π.

[3)] Pro maximalizační problém je definice analogická.

Pro mnoho přirozených optimalizačních problémů platí, že jejich rozhodovací verze je NP-úplná. Pokud přijmeme za pravdivou domněnku P $\neq$ NP, pak můžeme usoudit, že pro takové problémy neexistuje polynomický algoritmus, který vrací optimální řešení pro každý svůj vstup. Nemůžeme ovšem vyloučit existenci algoritmu s polynomickou složitostí, který sice pro některé vstupy optimální řešení nevrátí, ale cena jím vráceného řešení je ceně optimálního řešení rozumně blízko. Jak uvidíme níže, takové algoritmy skutečně existují. Říkáme jim *aproximační algoritmy*[4].

Jako příklad obtížného optimalizačního problému si uveďme problém nalezení *minimálního vrcholového pokrytí grafu*, jehož název budeme zkracovat jako Min-VC.[5] Vstupem problému je neorientovaný graf G s množinou vrcholů V a množinou hran H. Pro množinu vrcholů $C \subseteq V$ a hranu $h \in H$ řekneme, že h je *pokrytá* C (případně že C pokrývá h), pokud alespoň jeden z vrcholů h (připomeňme, že hranu chápeme jako množinu dvou vrcholů) patří do C. O množině C řekneme, že je vrcholovým pokrytím grafu, pokud pokrývá každou hranu v H. Množinou přípustných řešení grafu G je množina všech jeho vrcholových pokrytí. Cenou pokrytí C je počet vrcholů, které obsahuje. Cílem je, jak napovídá název problému, nalézt pokrytí s nejmenší cenou (tedy s nejmenším počtem vrcholů). Příklad grafu, několika jeho vrcholových pokrytí a několika množin vrcholů, která vrcholovým pokrytím nejsou, znázorňuje obr. 3.

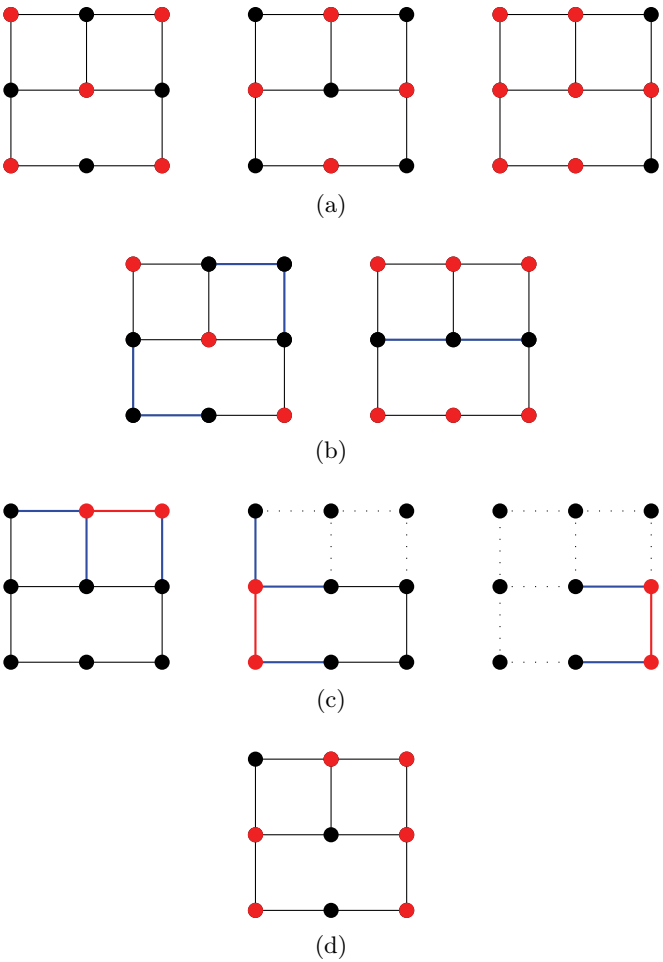
Problém Min-VC má mnoho praktických aplikací v různých oblastech (např. výpočetní biologie, chemie, metalurgické inženýrství). Abychom viděli jednoduchý příklad, představme si, že chceme pokrýt vnitřní prostory budovy kamerami tak, abychom mohli sledovat všechny chodby na budově. Přitom chceme použít kamer co nejméně. Přirozeným místem pro kamery jsou křížení chodeb. Pokud tedy v grafu, kde vrcholy odpovídají křížení chodeb a hrany chodbám samotným, najdeme optimální vrcholové pokrytí, nalezli jsme optimální rozmístění kamer.

Uveďme aproximační algoritmus pro Min-VC. Příklad běhu tohoto algoritmu je znázorněn na obrázku 3 (c).

MIN-VC-APPROX
Vstupem je graf $G = (V, H)$.
Výstupem je vrcholové pokrytí G.

---

[4] Jméno je převzato z anglického výrazu approximation algorithms.
[5] Zkratka anglického *minimal vertex cover*.



Obr. 3 (a) Vrcholová pokrytí grafu. Vrcholy patřící do pokrytí jsou zvýrazněny červeně. Obrázek nezachycuje všechna vrcholová pokrytí grafu, pouze některá. (b) Množiny vrcholů, které netvoří vrcholová pokrytí. Nepokryté hrany (tj. hrany, které nemají žádný červený vrchol) jsou zvýrazněny modře. (c) Průběh algoritmu Min-VC-approx. Červeně je zobrazena hrana h a její uzly vybrané v kroku 2. Modře jsou zvýrazněny hrany, které jsou v kroku 4 odstraněny spolu s hranou h. (d) Vrcholové pokrytí, které výsledkem.

1. Nastav $C \leftarrow \emptyset$ a $E \leftarrow H$.
2. Pokud je množina E prázdná, vrať C jako výsledek algoritmu a skonči.
3. Vyber hranu $h \in E$ a přidej její vrcholy do C.
4. Z množiny E odstraň hranu h a všechny hrany, které s h sdílejí vrchol.
5. Přejdi na krok 2.

Algoritmus během svého běhu udržuje množinu C vrcholů, o kterých už víme, že patří do pokrytí (na začátku je tato množina prázdná), a množinu E hran, které nejsou pomocí C pokryty (na začátku to jsou všechny hrany). V kroku 3 vždy vybere nějakou dosud nepokrytou hranu h. Tu pokryje tak, že do množiny C přidá její vrcholy. Tím pádem C pokrývá jak h, tak všechny hrany, které s h sdílejí vrchol. Proto takové hrany v kroku 4 odstraníme z E. Opakování kroků 2 až 4 končí v momentě, kdy je množina nepokrytých hran prázdná (to testujeme v kroku 2), a C je tak vrcholovým pokrytím. Vidíme také, že kroky 2 až 4 se zopakují nejvýše tolikrát, kolik je v grafu hran. Protože počet hran v grafu s n uzly je menší než n^2 a protože jedno opakování kroků 2 až 4 lze provést v čase polynomickém[6)] vzhledem k n, je složitost tohoto algoritmu polynomická. Vidíme tedy, že Min-VC-approx je korektní algoritmus pro problém Min-VC s polynomickou složitostí.

Vraťme se ještě ke kroku 3 algoritmu. Není v něm přesně určeno, kterou hranu z množiny E máme vybrat. Mohlo by se zdát, že zde není algoritmus popsaný dostatečně přesně. Je tomu skutečně tak. Pro jednoduchost jsme vynechali to, jak konkrétně v algoritmu reprezentujeme množiny E a C, a díky tomu jsme v kroku 3 výběr hrany neupřesnili. Pro konkrétní implementaci to ovšem udělat musíme. Pokud například množinu E uchováváme jako lineární seznam, zvolíme jako h v kroku 3 první prvek tohoto seznamu. V dalších úvahách o algoritmu se s touto nepřesností vypořádáme tak, že pro výběr h v kroku 3 připustíme jakoukoliv možnost[7)].

Na obr. 3 je vidět, že algoritmus nevrátil optimální řešení. Můžeme se zabývat velmi přirozenou otázkou: Jak moc se cena řešení vráceného

[6)] Přesná složitost záleží na konkrétní implementaci množin E a H a algoritmech pro přidávání a odebírání prvků těchto množin.

[7)] Můžeme se velmi rychle přesvědčit o tom, že pokud v kroku 3 zvolíme jiné hrany, než které jsme zvolili na obrázku 3 (c), může algoritmus vrátit zcela jiné vrcholové pokrytí.

Min-VC-approx může lišit od ceny optimálního řešení? Odpověď na tuto otázku dává následující věta.

Věta 2. *Vrcholové pokrytí, které vrátí algoritmus* Min-VC-approx*, obsahuje nejvýše dvakrát více vrcholů než optimální pokrytí.*

Důkaz. Označme A množinu všech hran, které jsme v kroku 3 algoritmu přiřadili do h. Protože algoritmus dává do pokrytí vždy oba vrcholy hrany h, platí $|C| \leq 2 \cdot |A|$. Současně ale díky kroku 4 nemůže množina A obsahovat dvě hrany, které spolu sdílí nějaký vrchol. Platí tedy

$$|C| = 2 \cdot |A|. \tag{2}$$

Libovolné vrcholové pokrytí, a speciálně i to optimální vrcholové pokrytí, musí určitě pokrýt hrany z množiny A, každou z nich alespoň jedním vrcholem. Odtud

$$|A| \leq \mathrm{OPT}, \tag{3}$$

kde OPT je počet vrcholů v optimálním vrcholovém pokrytí. Pokud dosadíme nerovnici (3) do rovnice (2) dostaneme

$$|C| \leq 2 \cdot \mathrm{OPT}$$

a vidíme, že věta platí. □

Předchozí věta ukazuje na algoritmu Min-VC-approx obecnou vlastnost aproximačních algoritmů, které říkáme *aproximační faktor*. Zaveďme si tento pojem přesněji. Předpokládejme, že máme algoritmus $\mathcal{A}$ pro minimalizační[8)] problém Π. Aproximační faktor algoritmu $\mathcal{A}$ pro instanci I označíme $f_{\mathcal{A}}(I)$ a definujeme jako

$$f_{\mathcal{A}}(I) = \frac{\mathrm{cost}\,(\mathcal{A}(I))}{\mathrm{OPT}(I)},$$

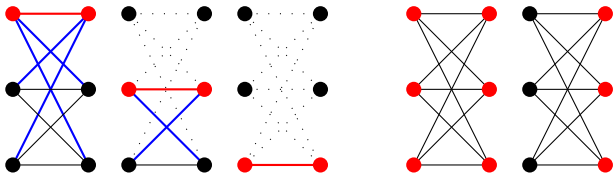
kde $\mathrm{cost}\,(\mathcal{A}(I))$ je cena řešení, které vrátí algoritmus $\mathcal{A}$ pro instanci I, a $\mathrm{OPT}(I)$ je cena optimálního řešení pro instanci I.

Definice 2. *Aproximační faktor algoritmu* $\mathcal{A}$ je funkce $R_{\mathcal{A}} : \mathbb{N} \to \mathbb{R}^+$ definovaná $R_{\mathcal{A}}(n) = \max\{f_{\mathcal{A}}(I) \mid |I| = n\}$.

[8)] Pro maximalizační problémy je definice analogická.

Abychom ukázali, že algoritmus Min-VC-approx má aproximační faktor $R(n) = 2$, musíme najít instanci, pro kterou algoritmus Min-VC-approx vrátí pokrytí s dvakrát více vrcholy než pokrytí optimální. Z věty 2 totiž její existence neplyne. Naštěstí není pro Min-VC-approx obtížné takové instance najít. Jsou to úplné bipartitní grafy.

Úplný bipartitní graf s $2n$ vrcholy ($n = 1, 2, \dots$), označme ho K_n, je takový graf, jehož uzly můžeme rozdělit do dvou disjunktních množin, V_1 a V_2, tak, že každá z nich má n vrcholů, každý vrchol z množiny V_1 je spojen hranou s každým vrcholem z množiny V_2 a graf žádné jiné hrany neobsahuje (graf K_3 vidíme na obr. 4).



Obr. 4 (a) Průběh algoritmu Min-VC-approx pro graf K_3. Aktuálně vybraná hrana je zobrazena červeně, hrany, které s ní sdílejí uzel modře. (b) Vlevo je vrcholové pokrytí spočtené algoritmem Min-VC-approx, vpravo optimální pokrytí. Vrcholy zařazené do pokrytí jsou zobrazeny červeně.

Prozkoumáme-li běh algoritmu pro K_n, zjistíme, že po prvním provedení kroků 3 a 4 nám nutně zůstane, uvážíme-li hrany z množiny E a vrcholy, které nejsou v C, graf K_{n-1}. Po dalším provedení těchto kroků graf K_{n-2}, a tak dále. Celkem tedy kroky 3 a 4 provedeme n-krát. Přitom vždy přidáme do C dva vrcholy, algoritmus proto vrátí pokrytí s $2n$ vrcholy, tedy množinu všech vrcholů v grafu. Optimální vrcholové pokrytí přitom má pouze n vrcholů, tvoří jej totiž libovolná z množin V_1 a V_2. Po krátké úvaze zjistíme, že pokrytí s menším počtem vrcholů v grafu neexistuje. Pro graf K_3 situaci ilustruje obr. 4.

Literatura

[1] *Vazirani, V. V.*: Approximation algorithms. Springer Verlag, 2003.

[2] *Villiamson, D. P., Shmoys, D. B.*: The design of approximation algorithms. Cambridge University Press, Cambridge, 2011. Dostupné na: http://www.designofapproxalgs.com/