6. Rychlejší algoritmy na minimální kostry

V této kapitole popíšeme několik pokročilejších algoritmů pro problém minimální kostry. Vesměs to budou různá vylepšení klasických algoritmů z minulé kapitoly.

Upravená verze Borůvkova algoritmu pro rovinné grafy

Vyjdeme z myšlenky, že můžeme po každém kroku původního Borůvkova algoritmu vzniklé komponenty souvislosti grafu kontrahovat do jednoho vrcholu a tím získat menší graf, který můžeme znovu rekurzivně zmenšovat. To funguje obecně, ale ukážeme, že pro rovinné grafy tak dosáhneme lineární časové složitosti.

Pozorování: Pokud $F \subseteq \mathrm{MST}(G)$ (kde $\mathrm{MST}(G)$ je minimální kostra grafu G), G' je graf vzniklý z G kontrakcí podél hran z F, pak kostra grafu G, která vznikne z $\mathrm{MST}(G')$ zpětným expandováním kontrahovaných vrcholů, je $\mathrm{MST}(G)$. Pokud kontrakcí vzniknou smyčky, můžeme je ihned odstraňovat; pokud paralelní hrany, ponecháme z nich vždy tu nejlehčí. To nás vede k následujícímu algoritmu:

Algoritmus: MST v rovinných grafech [8]

1. Ke každému vrcholu najdeme nejlevnější incidentní hranu – dostaneme množinu hran $F \subseteq E$.
2. Graf kontrahujeme podle F následovně:
3. Prohledáme do šířky graf (V, F) a přiřadíme každému vrcholu číslo komponenty, v níž se nachází.
4. Přečíslujeme hrany v G podle čísel komponent.
5. Odstraníme násobné hrany:
6. Setřídíme hrany lexikograficky přihrádkovým tříděním (násobné hrany jsou nyní pospolu).
7. Projdeme posloupnost hran a z každého úseku multihran odstraníme všechny až na nejlevnější hranu. Také odstraníme smyčky.
8. Pokud stále máme netriviální graf, opakujeme předchozí kroky.
9. Vrátíme jako MST všechny hrany, které se v průběhu algoritmu dostaly do F.

Časová složitost: Označme si n_i a m_i počet vrcholů a hran na počátku i-té iterace. Každý z kroků 1–7 trvá $\mathcal{O}(m_i)$, proto i celý cyklus algoritmu trvá $\mathcal{O}(m_i)$. Počet vrcholů grafu klesá s každým cyklem exponenciálně: $n_i \leq n/2^i$. Na začátku každého cyklu je graf rovinný (kontrakcí hrany v rovinném grafu se rovinnost zachovává) a není to multigraf, takže počet jeho hran je lineární v počtu vrcholů: $m_i < 3n_i$. Celkovou časovou složitost dostaneme jako součet dob trvání všech cyklů: $\mathcal{O}(\sum_i m_i) = \mathcal{O}(\sum_i n_i) = \mathcal{O}(n)$.

Minorově uzavřené třídy

Předchozí algoritmus ve skutečnosti funguje v lineárním čase i pro obecnější třídy grafů, než jsou grafy rovinné. Tím správným universem jsou minorově uzavřené třídy:

Definice: Graf H je *minorem* grafu G (značíme $H \preceq G$), pokud lze H získat z G mazáním vrcholů či hran a kontrahováním hran (s odstraněním smyček a násobných hran).

Pozorování: Relace $\preceq$ je částečné uspořádání na třídě všech grafů (reflexivita a tranzitivita plynou přímo z definice, pro antisymetrii si stačí všimnout, že jak mazáním, tak kontrakcemi klesá součet počtu vrcholů s počtem hran).

Pozorování: Pokud je graf H podgrafem grafu G, pak je také jeho minorem. Opačně to neplatí: například C_3 je minorem libovolné delší kružnice, ale určitě ne jejím podgrafem.

Definice: Třída grafů $\mathcal{C}$ je *minorově uzavřená,* pokud kdykoliv $G \in \mathcal{C}$ a $H \preceq G$, platí také $H \in \mathcal{C}$.

Příklady: Třída všech grafů a prázdná třída jsou jistě minorově uzavřené. Těm říkáme *triviální* třídy. Mezi ty netriviální patří třeba lesy, rovinné grafy, grafy nakreslitelné na další plochy, grafy nakreslitelné do $\mathbb{R}^3$ tak, že žádná kružnice netvoří uzel.

Definice: Pro třídu grafů $\mathcal{C}$ definujeme $\mathrm{Forb}(\mathcal{C})$ jako třídu všech grafů, jejichž minorem není žádný graf z $\mathcal{C}$. Pro zjednodušení značení budeme pro konečné třídy psát $\mathrm{Forb}(G_1, \ldots, G_k)$ namísto $\mathrm{Forb}(\{G_1, \ldots, G_k\})$.

Příklady: $\mathrm{Forb}(K_2)$ je třída všech grafů, které neobsahují žádnou hranu. $\mathrm{Forb}(C_3)$ je třída všech lesů. $\mathrm{Forb}(K_5, K_{3,3})$ jsou všechny rovinné grafy – to je minorová analogie Kuratowského věty (pokud graf G obsahuje dělení grafu H, pak je $H \preceq G$; opačně to obecně není pravda, ale zrovna pro dvojici K_5 a $K_{3,3}$ to funguje; zkuste si sami).

Lze každou minorově uzavřenou třídu $\mathcal{C}$ zapsat jako $\mathrm{Forb}(\mathcal{F})$ pro nějakou třídu $\mathcal{F}$ zakázaných minorů? Zajisté ano, stačí například zvolit $\mathcal{F}$ jako doplněk třídy $\mathcal{C}$. Slavná Robertsonova-Seymourova věta [11] dokonce zaručuje existenci *konečné* třídy zakázaných minorů. To není samo sebou, dokazuje se to dosti obtížně (a je to jedna z nejslavnějších kombinatorických vět za posledních mnoho let), ale plyne z toho spousta zajímavých důsledků. My si vystačíme s daleko jednoduššími prostředky a zájemce o hlubší studium minorové teorie odkážeme na Diestelovu knihu [2].

Ve zbytku této sekce dokážeme následující větu, která je zobecněním klasického tvrzení o hustotě rovinných grafů na minorově uzavřené třídy:

Definice: *Hustotou* neprázdného grafu G nazveme $\varrho(G) = |E(G)|/|V(G)|$. Hustotou třídy $\varrho(\mathcal{C})$ pak nazveme supremum z hustot všech neprázdných grafů ležících v této třídě.

Věta (o hustotě minorově uzavřených tříd): Pokud je třída grafů $\mathcal{C}$ minorově uzavřená a netriviální (alespoň jeden graf v ní leží a alespoň jeden neleží), pak má konečnou hustotu.

Důsledek: Pokud používáme kontrahující Borůvkův algoritmus na grafy ležící v nějaké netriviální minorově uzavřené třídě, pak všechny grafy, které algoritmus v průběhu výpočtu sestrojí, leží také v této třídě, takže na odhad jejich hustoty můžeme použít předchozí větu. Opět vyjde, že časová složitost algoritmu je lineární.

Důkaz věty: Ukážeme nejprve, že pro každou třídu $\mathcal{C}$ existuje nějaké k takové, že $\mathcal{C} \subseteq \mathrm{Forb}(K_k)$.

Už víme, že $\mathcal{C}$ lze zapsat jako $\mathrm{Forb}(\mathcal{F})$ pro nějakou třídu $\mathcal{F}$. Označme F graf z $\mathcal{F}$ s nejmenším počtem vrcholů; pokud existuje více takových, vybereme libovolný. Hledané k zvolíme jako počet vrcholů tohoto grafu.

Inkluze tvaru $\mathcal{A} \subseteq \mathcal{B}$ je ekvivalentní tomu, že kdykoliv nějaký graf G neleží v $\mathcal{B}$, pak neleží ani v $\mathcal{A}$. Mějme tedy nějaký graf $G \notin \mathrm{Forb}(K_k)$. Proto $K_k \preceq G$. Ovšem triviálně platí $F \preceq K_k$ a relace „být minorem“ je tranzitivní, takže $F \preceq G$, a proto $G \notin \mathcal{C}$.

Víme tedy, že $\mathcal{C} \subseteq \mathrm{Forb}(K_k)$. Proto musí platit $\varrho(\mathcal{C}) \leq \varrho(\mathrm{Forb}(K_k))$. Takže postačuje omezit hustotu tříd s jedním zakázaným minorem, který je úplným grafem, a to plyne z následující Maderovy věty. ♡

Věta (Maderova): Pro každé $k \geq 2$ existuje $c(k)$ takové, že kdykoliv má graf hustotu alespoň $c(k)$, obsahuje jako podgraf nějaké dělení grafu K_k.

Důkaz: Viz Diestel [2].

Jarníkův algoritmus s Fibonacciho haldou

Původní Jarníkův algoritmus s haldou má díky ní složitost $\mathcal{O}(m \log n)$, to zlepšíme použitím Fibonacciho haldy H, do které si pro každý vrchol sousedící se zatím vybudovaným stromem T uložíme nejlevnější z hran vedoucích mezi tímto vrcholem a stromem T. Tyto hrany bude halda udržovat uspořádané podle vah.

Algoritmus: Jarníkův algoritmus #2 (Fredman, Tarjan [4])

1. Začneme libovolným vrcholem v_0, $T \leftarrow \{v_0\}$.
2. Do haldy H umístíme všechny hrany vedoucí z v_0.
3. Opakujeme, dokud $H \neq \emptyset$:
4. $\quad vw \leftarrow \mathit{ExtractMin}(H)$, přičemž $v \notin T, w \in T$.
5. $\quad T \leftarrow T \cup \{vw\}$
6. $\quad$ Pro všechny sousedy u vrcholu v, kteří dosud nejsou v T, upravíme haldu:
7. $\quad\quad$ Pokud ještě v H není hrana incidentní s u, přidáme hranu uv.
8. $\quad\quad$ Pokud už tam nějaká taková hrana je a je-li těžší než uv, nahradíme ji hranou uv a provedeme *DecreaseKey*.

Správnost algoritmu přímo plyne ze správnosti Jarníkova algoritmu.

Časová složitost: Složitost tohoto algoritmu bude $\mathcal{O}(m+n \log n)$, neboť vnější cyklus se provede nanejvýš n-krát, za *ExtractMin* v něm tedy zaplatíme celkem $\mathcal{O}(n \log n)$, za přidávání vrcholů do H a nalézání nejlevnějších hran zaplatíme celkem $\mathcal{O}(m)$ (na každou hranu takto sáhneme nanejvýš dvakrát), za snižování vah vrcholů v haldě rovněž pouze $\mathcal{O}(m)$ (nanejvýš m-krát provedu porovnání vah a *DecreaseKey* v kroku 8 za $\mathcal{O}(1)$).

Toto zlepšení je důležitější, než by se mohlo zdát, protože nám pro grafy s mnoha hranami (konkrétně pro grafy s $m = \Omega(n \log n)$) dává lineární algoritmus.

Kombinace Jarníkova a Borůvkova algoritmu

K dalšímu zlepšení dojde, když před předchozím algoritmem spustíme $\log\log n$ cyklů Borůvkova algoritmu s kontrahováním vrcholů, čímž graf zahustíme.

Algoritmus: Jarníkův algoritmus #3 (původ neznámý)

1. Provedeme $\log\log n$ cyklů upraveného Borůvkova algoritmu s kontrahováním hran popsaného výše.
2. Pokračujeme Jarníkovým algoritmem #2.

Časová složitost: Složitost první části je $\mathcal{O}(m \log\log n)$. Počet vrcholů se po první části algoritmu sníží na $n' \leq n/\log n$ a složitost druhé části bude tedy nanejvýš $\mathcal{O}(m + n' \log n') = \mathcal{O}(m)$.

Jarníkův algoritmus s omezením velikosti haldy

Ještě většího zrychlení dosáhneme, omezíme-li Jarníkovu algoritmu #2 vhodně velikost haldy, takže nám nalezne jednotlivé podkostřičky zastavené v růstu přetečením haldy. Podle těchto podkostřiček graf následně skontrahujeme a budeme pokračovat s mnohem menším grafem.

Algoritmus: Jarníkův algoritmus #4 (Fredman, Tarjan [4])

1. Opakujeme, dokud máme netriviální G (s alespoň jednou hranou):
2. $t \leftarrow |V(G)|$.
3. Zvolíme $k \leftarrow 2^{2m/t}$ (velikost haldy).
4. $T \leftarrow \emptyset$.
5. Opakujeme, dokud existují vrcholy mimo T:
6. Najdeme vrchol v_0 mimo T.
7. Spustíme Jarníkův alg. #2 pro celý graf od v_0. Zastavíme ho, pokud:
8. $|H| \geq k$ (byla překročena velikost haldy) nebo
9. $H = \emptyset$ (došli sousedé) nebo
10. do T jsme přidali hranu oboustranně incidentní s hranami v T (připojili jsme novou podkostru k nějaké už nalezené).
11. Kontrahujeme G podle podkoster nalezených v T.

Pozorování: Pokud algoritmus ještě neskončil, je každá z nalezených podkoster v T incidentní s alespoň k hranami (do toho počítáme i vnitřní hrany vedoucí mezi vrcholy podkostry). Jak to vypadá pro jednotlivá ukončení:

8. $|H| \geq k$ – všechny hrany v haldě jsou incidentní s T a navzájem různé, takže incidentních je dost.
9. $H = \emptyset$ – nemůže nastat, algoritmus by skončil.
10. Připojím se k už existující podkostře – jen ji zvětším.

Časová složitost: Důsledkem předchozího pozorování je, že počet podkoster v jednom průchodu je nanejvýš $2m/k$. Pro t' a k' v následujícím kroku potom platí $t' \leq 2m/k$

a $k' = 2^{2m/t'} \geq 2^k$. Průchodů bude tedy nanejvýš $\log^* n$[1], protože průchod s $k > n$ bude už určitě poslední. Přitom jeden vnější průchod trvá $\mathcal{O}(m + t \log k)$, což je pro $k = 2^{2m/t}$ rovno $\mathcal{O}(m)$. Celkově tedy algoritmus poběží v čase $\mathcal{O}(m \log^* n)$.

I odhad $\log^* n$ je ale příliš hrubý, protože nezačínáme s haldou velikosti 1, nýbrž $2^{2m/n}$. Můžeme tedy počet průchodů přesněji omezit funkcí $\beta(m, n) = \min\{i : \log^{(i)} n < m/n\}$ a časovou složitost odhadnout jako $\mathcal{O}(m\beta(m, n))$. To nám dává lineární algoritmus pro grafy s $m \geq n \log^{(k)} n$ pro libovolnou konstantu k, jelikož $\beta(m, n)$ tehdy vyjde omezená konstantou.

Další výsledky

- $\mathcal{O}(m\alpha(m, n))$, kde $\alpha(m, n)$ je obdoba inverzní Ackermannovy funkce definovaná podobně, jako je $\beta(m, n)$ obdobou $\log^*$. [1], [9]
- $\mathcal{O}(\mathcal{T}(m, n))$, kde $\mathcal{T}(m, n)$ je hloubka optimálního rozhodovacího stromu pro nalezení minimální kostry v grafech s patřičným počtem hran a vrcholů [10]. Jelikož každý deterministický algoritmus založený na porovnávání vah lze popsat rozhodovacím stromem, je tento algoritmus zaručeně optimální. Jen bohužel nevíme, jak optimální stromy vypadají, takže je stále otevřeno, zda lze MST nalézt v lineárním čase. Nicméně tento algoritmus pracuje i na Pointer Machine, pročež víme, že pokud je lineární složitosti možné dosáhnout, není k tomu potřeba výpočetní síla RAMu.[2]
- $\mathcal{O}(m)$ pro grafy s celočíselnými vahami (na RAMu) [3] – ukážeme v jedné z následujících kapitol.
- $\mathcal{O}(m)$, pokud už máme hrany setříděné podle vah: jelikož víme, že záleží jen na uspořádání, můžeme váhy přečíslovat na $1 \ldots m$ a použít předchozí algoritmus.
- $\mathcal{O}(m)$ průměrně: randomizovaný algoritmus, který pro libovolný vstupní graf doběhne v očekávaném lineárním čase [5].
- Na zjištění, zda je zadaná kostra minimální, stačí $\mathcal{O}(m)$ porovnání [7] a dokonce lze v lineárním čase zjistit, která to jsou [6]. Z toho ostatně vychází předchozí randomizovaný algoritmus.

Literatura

[1] B. Chazelle. A Minimum Spanning Tree Algorithm with Inverse-Ackermann Type Complexity. *J. ACM*, 47:1028–1047, 2000.

[2] R. Diestel. *Graph Theory*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2005.

[3] M. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. In *Proceedings of FOCS'90*, pages 719–725, 1990.

[1] $\log^* n$ je inverzní funkce k „věži z mocnin“, čili $\min\{i : \log^{(i)} n < 1\}$, kde $\log^{(i)} n$ je i-krát iterovaný logaritmus.

[2] O výpočetních modelech viz příští kapitola.

[4] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[5] D. R. Karger, P. N. Klein, and R. E. Tarjan. Linear expected-time algorithms for connectivity problems. *J. ACM*, 42:321–328, 1995.

[6] V. King. A simpler minimum spanning tree verification algorithm. In *Workshop on Algorithms and Data Structures*, pages 440–448, 1995.

[7] J. Komlós. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985.

[8] M. Mareš. Two linear time algorithms for MST on minor closed graph classes. *Archivum Mathematicum*, 40:315–320, 2004.

[9] S. Pettie. Finding minimum spanning trees in $O(m\alpha(m, n))$ time. Tech Report TR99-23, Univ. of Texas at Austin, 1999.

[10] S. Pettie and V. Ramachandran. An Optimal Minimum Spanning Tree Algorithm. In *Proceedings of ICALP'2000*, pages 49–60. Springer Verlag, 2000.

[11] N. Robertson and P. D. Seymour. Graph minors: XX. Wagner's Conjecture. *Journal of Combinatorial Theory Series B*, 92(2):325–357, 2004.