3. Bipartitní párování a globální k-souvislost

V předešlých kapitolách jsme se zabývali aplikacemi toků na hledání maximálního párování a minimálního *st*-řezu. Nyní si předvedeme dva algoritmy pro podobné problémy, které se obejdou bez toků.

Maximální párování v regulárním bipartitním grafu [1]

Nejprve si nadefinujme operaci *štěpení grafu,* která zadaný graf $G = (V, E)$ se všemi vrcholy sudého stupně a sudým počtem hran v každé komponentě souvislosti rozloží na disjunktní podgrafy $G_1 = (V, E_1)$ a $G_2 = (V, E_2)$, v nichž bude pro každý vrchol v platit $\deg_{G_1}(v) = \deg_{G_2}(v) = \deg_G(v)/2$. Tuto operaci můžeme snadno provést v lineárním čase tak, že si graf rozdělíme na komponenty, v každé nalezneme eulerovský tah a jeho hrany budeme přidávat střídavě do G_1 a do G_2.

Štěpení nám pomůže ke snadnému algoritmu pro nalezení maximálního párování ve 2^t-regulárním bipartitním grafu.⟨1⟩ Komponenty takového grafu mají určitě sudý počet hran, takže jej můžeme rozštěpit na dva 2^{t-1}-regulární grafy. Libovolný jeden z nich pak opět rozštěpíme atd., až dostaneme 1-regulární graf, který je perfektním párováním v G. To vše jsme schopni stihnout v lineárním čase, jelikož velikosti grafů, které štěpíme, exponenciálně klesají. Také bychom mohli rekurzivně zpracovávat obě části a tak se v čase $\mathcal{O}(m \log n)$ dobrat ke kompletní 1-faktorizaci zadaného grafu.⟨2⟩

Pokud zadaný graf nebude 2^t-regulární, pomůžeme si tím, že ho novými hranami doplníme na 2^t-regulární a pak si při štěpeních budeme vybírat ten podgraf, do kterého padlo méně nových hran, a ukážeme, že nakonec všechny zmizí. Abychom graf příliš nezvětšili, budeme se snažit místo přidávání úplně nových hran pouze zvyšovat násobnost hran existujících. Pro každou hranu e si tedy budeme pamatovat její násobnost $n(e)$.

Štěpení grafu s násobnostmi pak budeme provádět následovně: hranu e s násobností $n(e)$ umístíme do G_1 i do G_2 s násobností $\lfloor n(e)/2 \rfloor$ a pokud bylo $n(e)$ liché, přidáme hranu do pomocného grafu G'. Všimněte si, že G' bude graf bez násobností, v němž mají všechny vrcholy sudý stupeň, takže na něj můžeme aplikovat původní štěpící algoritmus a G'_i přiřadit ke G_i. To vše zvládneme v čase $\mathcal{O}(m)$.

Mějme nyní k-regulární bipartitní graf. Obě jeho partity jsou stejně velké, označme si počet vrcholů v každé z nich n. Zvolme t tak aby $2^t \geq kn$. Zvolme dále parametry $\alpha := \lfloor 2^t/k \rfloor$ a $\beta := 2^t \bmod k$. Každé původní hraně nastavíme násobnost α a přidáme triviální párování F (i-tý vrchol vlevo se spojí s i-tým vrcholem vpravo) s násobností β. Všimněte si, že $\beta < k$, a proto hran v F (včetně násobností) bude méně než 2^t.

⟨1⟩ Všimněte si, že takové párování bude vždy perfektní (viz Hallova věta).

⟨2⟩ To je rozklad hran grafu na disjunktní perfektní párování a má ho každý regulární bipartitní graf.

Takto získáme 2^t-regulární graf, jehož reprezentace bude lineárně velká. Na tento graf budeme aplikovat operaci štěpení a budeme si vybírat vždy tu polovinu, kde bude méně hran z F. Po t iteracích dospějeme k párování a jelikož se v každém kroku zbavíme alespoň poloviny hran z F, nebude toto párování obsahovat žádnou takovou hranu a navíc nebude obsahovat ani násobné hrany, takže bude podgrafem zadaného grafu, jak potřebujeme.

Časová složitost algoritmu je $\mathcal{O}(m \log n)$, jelikož provádíme inicializaci v čase $\mathcal{O}(m)$ a celkem $\log_2 kn = \mathcal{O}(\log n)$ iterací po $\mathcal{O}(m)$.

Stupeň souvislosti grafu

Problém zjištění *stupně hranové souvislosti* grafu lze převést na problém hledání minimálního řezu, který již pro zadanou dvojici vrcholů umíme řešit pomocí Dinicova algoritmu v čase $\mathcal{O}(n^{2/3}m)$. Pokud chceme najít minimum ze všech řezů v grafu, můžeme vyzkoušet všechny dvojice (s, t). To však lze snadno zrychlit, pokud si uvědomíme, že jeden z vrcholů (třeba s) můžeme zvolit pevně: vezmeme-li libovolný řez C, pak jistě najdeme alespoň jedno t, které padne do jiné komponenty než pevně zvolené s, takže minimální st-řez bude nejvýše tak velký jako C. V orientovaném grafu musíme projít jak řezy pro $s \to t$, tak i $t \to s$. Algoritmus bude mít složitost $\mathcal{O}(n^{5/3}m)$.

U *vrcholové k-souvislosti* to ovšem tak snadno nepůjde. Pokud by totiž fixovaný vrchol byl součástí nějakého minimálního separátoru, algoritmus může selhat. Přesto ale nemusíme procházet všechny dvojice vrcholů. Stačí jako s postupně zvolit více vrcholů, než je velikost minimálního separátoru. Algoritmus si tedy bude pamatovat, kolik vrcholů už prošel a nejmenší zatím nalezený st-separátor, a jakmile počet vrcholů překročí velikost separátoru, prohlásí separátor za minimální. To zvládne v čase $\mathcal{O}(\kappa(G) \cdot n^{3/2}m)$, kde $\kappa(G)$ je nalezený stupeň souvislosti G.

Pro minimální řezy v neorientovaných grafech ovšem existuje následující rychlejší algoritmus:

Globálně minimální řez (Nagamochi, Ibaraki [2])

Buď G neorientovaný multigraf s nezáporným ohodnocením hran. Označíme si:

Značení:

- $r(u, v)$ buď kapacita minimálního uv-řezu,
- $d(P, Q)$ buď kapacita hran vedoucích mezi množinami $P, Q \subseteq V$,
- $d(P) = d(P, \overline{P})$ buď kapacita hran vedoucích mezi $P \subseteq V$ a zbytkem grafu,
- $d(v) = d(\{v\})$ buď kapacita hran vedoucích z v (tedy pro neohodnocené grafy stupeň v),
- analogicky zavedeme $d(v, w)$ a $d(v, P)$.

Definice: *Legálním uspořádáním vrcholů* (LU) budeme nazývat lineární uspořádání vrcholů $v_1 \dots v_n$ takové, že platí $d(\{v_1 \dots v_{i-1}\}, v_i) \geq d(\{v_1 \dots v_{i-1}\}, v_j)$ pro každé $1 \leq i < j \leq n$.

Lemma: Je-li $v_1 \dots v_n$ LU na G, pak $r(v_{n-1}, v_n) = d(v_n)$.

Důkaz: Buď C libovolný řez oddělující v_{n-1} a v_n. Dokážeme, že jeho velikost je alespoň $d(v_n)$. Utvořme posloupnost vrcholů u_i takto:

1. $u_0 := v_1$
2. $u_i := v_j$ tak, že $j > i$, v_i a v_j jsou odděleny řezem C a j je minimální takové. [Tedy v_j je nejbližší vrchol na druhé straně řezu.]

Každé u_{i-1} je tedy buď rovno u_i, pokud jsou v_i a v_{i-1} na stejné straně řezu, nebo rovno v_i, pokud jsou v_i a v_{i-1} na stranách opačných. Z toho dostáváme, že $d(\{v_1 \dots v_{i-1}\}, u_i) \leq d(\{v_1 \dots v_{i-1}\}, u_{i-1})$, protože buďto $u_{i-1} = u_i$, a pak je nerovnost splněna jako rovnost, nebo je $u_{i-1} = v_i$ a nerovnost plyne z legálnosti uspořádání.

Chceme ukázat, že velikost našeho řezu C je alespoň taková, jako velikost řezu kolem vrcholu v_n. Všimneme si, že $|C| \geq \sum_{i=1}^{n-1} d(v_i, u_i)$. Hrany mezi v_i a u_i jsou totiž navzájem různé a každá z nich je součástí řezu C. Ukážeme, že pravá strana je alespoň $d(v_n)$:

$$\begin{aligned}
\sum_{i=1}^{n-1} d(v_i, u_i) &= \sum_{i=1}^{n-1} d(\{v_1 \dots v_i\}, u_i) - d(\{v_1 \dots v_{i-1}\}, u_i) \geq \\
&\geq \sum_{i=1}^{n-1} d(\{v_1 \dots v_i\}, u_i) - d(\{v_1 \dots v_{i-1}\}, u_{i-1}) = \\
&= d(\{v_1 \dots v_{n-1}\}, u_{n-1}) - d(\{v_1 \dots v_0\}, u_0) = \\
&= d(\{v_1 \dots v_{n-1}\}, v_n) - 0 = d(v_n).
\end{aligned}$$

♡

Dokázali jsme, že libovolný řez separující v_{n-1} a v_n je alespoň tak velký jako jednoduchý řez skládající se jen z hran kolem v_n. Když tedy sestavíme nějakou LU posloupnost vrcholů, budeme mít k dispozici jednoduchý minimální řez v_{n-1} a v_n. Následně vytvoříme graf G', v němž v_{n-1} a v_n kontrahujeme. Rekurzivně najdeme minimální řez v G' (sestrojíme nové LU atd.). Hledaný minimální řez poté buďto odděluje vrcholy v_n a v_{n-1}, a potom je řez kolem vrcholu v_n minimální, nebo vrcholy v_n a v_{n-1} neodděluje, a v takovém případě jej najdeme rekurzivně. Hledaný řez je tedy menší z rekurzivně nalezeného řezu a řezu kolem v_n.

Zbývá ukázat, jak konstruovat LU. Postačí hladově: Pamatujeme si $\forall v \neq v_1 \dots v_{i-1}$ hodnotu $d(\{v_1 \dots v_{i-1}\}, v)$, označme ji z_v. V každém kroku vybereme vrchol v s maximální hodnotou z_v, prohlásíme ho za v_i a přepočítáme z_v.

Zde se hodí datová struktura, která dokáže rychle hledat maxima a zvyšovat hodnoty prvků, například Fibonacciho halda. Ta zvládne *DeleteMax* v čase $\mathcal{O}(\log n)$ a *Increase* v $\mathcal{O}(1)$ amortizovaně. Celkem pak náš algoritmus bude mít složitost $\mathcal{O}(n(m + n \log n))$ pro obecné kapacity.

Pokud jsou kapacity malá celá čísla, můžeme využít přihrádkové struktury. Budeme si udržovat obousměrný seznam zatím použitých hodnot z_v, každý prvek

takového seznamu bude obsahovat všechny vrcholy se společnou hodnotou z_v. Když budeme mít seznam seřazený, vybrání minimálního prvku bude znamenat pouze podívat se na první prvek seznamu a z něj odebrat jeden vrchol, případně celý prvek ze seznamu odstranit. Operace *Increase* poté bude reprezentovat pouze přesunutí vrcholu o malý počet přihrádek, případně založení nové přihrádky na správném místě. *DeleteMax* proto bude mít složitost $\mathcal{O}(1)$, všechny *Increase* dohromady $\mathcal{O}(m)$, jelikož za každou hranu přeskakujeme maximálně jednu přihrádku, a celý algoritmus $\mathcal{O}(mn)$.

Literatura

[1] N. Alon. A simple algorithm for edge-coloring bipartite multigraphs. *Inf. Process. Lett.*, 85(6):301–302, 2003.

[2] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discret. Math.*, 5(1):54–66, 1992.