Aproximační a online algoritmy

2. ledna 2013

Obsah

1 Minimální vrcholové pokrytí

Přikládám vždy celou hranu, tu utrhnu – 2-aproximace.

1.1 Bipartitní graf

V něm je optimum rovno velikosti maximálního toku.

1.2 Vážená verze

Řeší se přes lineární programování. Proměnná na vrchol, buď 0 nebo 1. Chceme, aby pro každou hranu byl součet alespoň 1. Vyjde něco, co má s celočíselným řešení něco společného. Dá nám dolní odhad na optimum. Vezmu všechny vrcholy, které jsou alespoň 0.5, ty nastavím na 1. Velikost je nejvýše 2× větší, než hodnota lineárního programu.

Věta 1 (PCP) *Je NP-těžké rozlišit splnitelné formule od formulí, kde lze splnit nejvíce* 99% *klauzulí.*

Toto je až upravená verze.

Z toho například plyne, že je NP-těžké α-aproximovat vrcholové pokrytí pro $\alpha < \frac{7}{6}$.

Existuje hypotéza, ze které plyne, že asi neexistuje lepší, než 2-aproximace (protože by to mělo podobné následky, jako kdyby se ukázalo, že $P = NP$).

Když hledáme maximální nezávislou množinu v grafu, tak je to opačné k vrcholovému pokrytí. Z toho, mimo jiné, plyne, že nelze odlišit optimum od nezávislých množit, které jsou „hodně dobré“.

Hledání největší kliky je také problém nějak duální k nezávislé množině (klika v doplňku je totéž jako nezávislá množina originálu).

2 Obchodní cestující

Pro obecnou nejde najít žádnou aproximaci (uměli bychom řešení hamiltonky).

Jsou-li vzdálenosti podle trojúhelníkové nerovnosti, můžu pomocí kostry hledat 2-aproximaci (určitě to nebude víc jak 2∗ horší, než kostra, kostra je nejvýše tak velká, jako hamiltonka).

Vylepšená verze – najdeme minimální kostru, potom všechny vrcholy, kde je stupeň v kostře lichý. Potom najdeme minimální perfektní párování na

těchto vrcholech. Potom uděláme eulerovský tah na kostře sjednocené s párováním (poznámka: tohle může být multigraf). Tohle potom zkrátím (vyházu duplicity).

Tohle je $\frac{3}{2}$-aproximace. Párování je nejvýše polovina optima. To se sečte, takže max. $\frac{3}{2}$ optima.

To, že to párování je nejvýše polovina je vidět z toho, že hamiltonka lze složit ze dvou párování (a jedné hrany navíc).

3 Rozvrhování

Budeme mít m počítačů. Vstup je posloupnost čísel $p_1, \ldots, p_n \in \mathbb{R}^+$, což budou úlohy. Výstup má být zobrazení $S : 1 \ldots n \to 1 \ldots m$. Cíl bude minimalizovat délka rozvrhu, tedy $\max_{i \in 1 \ldots n} \sum_{j \in 1 \ldots m; S(j)=i} p_j$ (tedy, jen rozdělení na počítače, ne jak za sebou).

Opačně máme deadline, chceme co nejmíň počítačů. To se nazývá bin-packing.

3.1 Hladový algoritmus

Vždy vrhne úlohu na nejméně zatížený počítač. Je hloupý, ale je online.

Toto je $2 - \frac{1}{m}$ aproximační algoritmus. Nechť T je doba našeho rozvrhu a t je startovní čas úlohy, která končí v T. $T - t < p_j \leq opt$. $t \leq opt$ (protože každý pracuje alespoň do t).

Dokonce, $T + (m-2)t \leq m \cdot opt$, tedy $m \cdot T \leq (2m-1) \cdot opt$.

Řekneme, že algoritmus je ***R-kompetitivní***, pokud je online a $\exists c \forall$ vstupy dá nejhůř $R \cdot opt + c$. Ví se, že existuje nějaký pro $R < 2$, ale neví se, jak přesně ani jaký to je.

Pokud napřed seřadíme, tak je $\frac{4}{3}$ aproximační.

Tvrzení 1 *Je to NP-těžké.*

Důkaz:
Převedu na to loupežníky, vezmu $m = 2$.

☻

Ze stejného důvodu i bin-packing.

Věta 2 *Neexistuje $\frac{3}{2} - \epsilon$ aproximační algoritmus pro bin-packing pokud $P \neq NP$ (tedy, potřebuju rozlišit 2 a 3 počítače).*

3.2 Rozvrhování se závislostmi

Máme podmínky tak, že něco musí skončit dřív, než něco jiného začne.

Uděláme hladový algoritmus. Bereme čas postupně, v čase t, pokud existuje volný počítač a úloha se splněnými závislosti, tak ji tam hodíme.

Tvrzení 2 *Toto je* $(2 - \frac{1}{m})$ *aproximace.*

Důkaz:
Toto je zobecnění minulého hladového algoritmu. Zkonstruujeme cestu v grafu G takovou, že kdykoliv v rozvrhu je volný počítač, tak běží jedna z úloh na této cestě. Toto je dolní odhad na optimum.

Konstruuju od konce, vezmu poslední, pak se mrknu na poslední volno, pokud existuje. V tom případě se v té době na něco muselo čekat (jinak jsme to mohli pustit).

Délka tohoto je dolní odhad optima, musí jít po sobě.

☻

3.3 Hladový bin-packing

Používá „first-fit“, nacpe do první, kam se vejde. Když nevejde, koupí další počítač.

Tvrzení 3 *Tohle je* 2*-aproximace.*

Důkaz:
Všechny kromě jednoho jsou alespoň z poloviny zaplněné. Nová úloha buď menší než půl, potom když začínám, tak zbytek musí být víc než napůl. Když dostanu velkou, můžu začít nové, a zase bude do půl plné.

Tedy, na konci máme určitě $\frac{m-1}{2}$ množství práce. Z toho vyjde, že $m < 2 \cdot opt + 1$.

☻

Ve skutečnosti je to $1.7 \cdot opt + const$.

Řekneme, že opt. problém má poly-čas aproximační schéma, pokud $\forall\epsilon\exists$ algoritmus běžící v čase polynomiálním ve velikosti instance a je $1+\epsilon$ aproximace.

Plně-polynomiální aproximační schéma vyžaduje polynomialitu i v $\frac{1}{\epsilon}$.

Je-li problém silně NP-těžký a má FPTAS, pak $P = NP$.

4 Množinové pokrytí

Máme množinu U prvků a množinový systém $\mathcal{S} \subseteq 2^U$, chceme $\bigcup \mathcal{C} = U$, chceme minimalizovat váhu množin.

Můžeme zapsat jako lineární program.

Chceme $x_s \geq 0$, chceme, aby prvek z univerza měl aspoň 1. Minimalizujeme $\sum w_s x_s$. Duální program je $f = \max_i$ počet množin, ve kterých se vyskytuje daný prvek. Dá se získat f-aproximační algoritmus, jednoduše vezmeme všechny, které tam jsou alespoň jednou f-tinou.

Když vezmu duál, tak můžu každou dát každou množinu do výsledku s pravděpodobností x_s^*. Tohle nemusí dát správný výsledek, ale když to zopakujeme řádově $\log n$ krát.

Pravděpodobnost, že někdo je nepokrytý je menší, než $\left(1 - \frac{1}{e}\right)^{\alpha \cdot \log n} < \frac{1}{2}n$. Pravděpodobnost, že něco je nepokryté je půl.

Cena je $\alpha \cdot \log n \cdot opt_{lp}$.

4.1 Primar-dual pro set-cover

Na začátku nastavím $y := 0$ a $\mathcal{C} := \emptyset$. Vezmu nějaké nepokryté i. Zvyšujeme y_i, dokud některá duální podmínka není těsná. Přidáme všechny S takové, které mají w_s rovny y.

4.2 Hladový algoritmus

Přidávám vždy tu s nejlepším poměrem ceny a „nových“ prvků (ty, co ještě nemám).

Hladový algoritmus je H_k je aproximační, kde k je max. velikost množiny, H je k-té harmonické číslo.

4.3 Semidefinitní programování

Matice A je ***pozitivně semidefinitní***, pokud:

- $\forall x; x^T A x \geq 0$
- $\forall \lambda; \lambda \geq 0$
- $A = W^T W$
- Všechny hlavní podmatice mají determinant ≥ 0

Program nazvu ***pozitivně semidefinitní***, pokud má proměnné v matici a chceme maximalizovat nějakou lineární funkci $C \cdot X$ za podmínek, že X je pozitivě semidefinitní, $A_k \cdot X = b_k$. Toto je zobecnění lineárního programu (můžu si udělat proměnné jen po diagonále, můžu mít nerovnosti obvyklým podvodem).

Vektorové programy, kde proměnné jsou vektory. Taktéž chceme maximalizovat, tentokrát $\sum c_{i,j}(v_i \cdot v_j)$ a podmínky jsou, že $\forall k \sum a_{i,j,k} \cdot (v_i \cdot v_j)$.

Věta 3 *Semidefinitní programy a vektorové programy jsou ekvivalentní.*

Důkaz:
Do matice W napíšu vektory jako sloupce, utvořím matici $X = W^T \cdot W$, pak je to matice, co má prvky jako skalární součiny.

☻

4.4 Maximální řez

Chceme rozdělit graf na 2 části tak, aby součet vah hran mezi nimi byl maximální.

Můžeme vzít každou hranu s pravděpodobností polovina, pak máme v průměru $\frac{1}{2}$ aproximaci. Nebo to můžu zkusit hladově a házet do jedné nebo do druhé, podle toho, kde dostane víc.

Zkusíme to takto. Vezmeme proměnné, což budou ± 1. Snažím se maximalizovat $\sum_{e \in E} w_e \cdot (1 - v_i \cdot v_j)$.

Místo toho vezmeme reálné vektory v $\mathbb{R}^n$, $v_i \cdot v_i = 1$. To už je vektorový program, to řešit lze.

4.5 Vektorový program

Bude jen $1 + \epsilon$ aproximace, protože mohou vycházet iracionální čísla.

Vezmu náhodný vektor r na sféře. Všechny, kde $v_i \cdot r \geq 0$ dám do jedné, ostatní do druhé.

5 Online algoritmy

Máme problém prohledávání přímky (cow-path problem, bridge problem). Máme přímku, kráva hledá díru v přímce.

Algoritmus je ***R-kompetetivní***, pokud je online algoritmus a pro všechny instance projde maximálně $R \cdot opt(I) + c$.

Věta 4 *Algoritmus „chodím v* $2*$ *vzrůstajících intervalech po přímce“ je* 9*-kompetitivní.*

Pravděpodobnostní algoritmus je R-kompetitivní, že průměrný výsledek je R-kompetitivní. Dvě možnosti – oblivious adversary je takový, že nereaguje na průběžné výsledky programu, adaptive adversary si může vymýšlet podle toho, co dělá algoritmus.

5.1 Pagin/caching

Máme k stránek rychlé paměti. Vstup je posloupnost požadavků na jednotlivé stránky z nějakých n stránek, které existují celkem (pomalých). Cena je za načítání stránek.

Líné algoritmy – v i-tém kroku čte maximálně jednu stránku.

$k = 1$ – každý algoritmus je optimální.

Potom je fifo, clock, statistiky za poslední dobu, počty požadavků, stárnutí.

LRU je že vyberu tu, která se nejdéle nepožadovala. FWF (Flush When Full) – když musím vyhazovat, vyhodím všechno.

FIFO, LRU, FWF jsou k-kompetitivní pro deterministický algoritmus.

Značkovací algoritmus je takový, který si pamatuje příznak, ten nastaví při požadavku na stránku ho nastaví, při page-fault příznaky ruší (např. když žádná bez značky není) a vyhazuje něco neoznačeného.

LRU a FWF jsou značkovací.

Každý značkovací algoritmus je k-kompetitivní. Posloupnost požadavků rozdělím na fáze. Každý první požadavek nové fáze je ten, kdy mažu značky. Algoritmus má maximálně $k \cdot M$ vyhazování (M je počet fází). Optimum musí být alespoň $M - 1$. Ve dvou po sobě jdoucích fází je alespoň $k + 1$ různých stránek, je tedy potřeba něco vyhodit.

FIFO není značkovací. Ale když vyházeme požadavky, kde není page-fault, na tom se to chová stejně. A na tom se to chová jako LRO.

Věta 5 *Každý pravděpodobnostní* R*-kompetitivní algoritmus pro paging má* $R \geq H_k$.

6 k-server problém

Máme danou metriku M a parametr k. Vstupem je posloupnost bodů v M (to jsou požadavky). Chceme rozposílat k agentů tak, aby každý požadavek byl navštívený tak, aby nacestovali co nejméně.

Cachování je speciální případ pro uniformní metrika.

6.1 Potenciál

Pro každý online R kompetitivní algoritums lze najít potenciál, který říká, kolik peněz může stát libovolné další pokračování.

Potom platí, že $0 \leq \Phi(S_N, A_N) \leq \Phi(S_0, A_0) + R \cdot OPT - ALG$.

7 Vážené stránkování

Podobně, jako stránkování, ale každá stránka má nějakou váhu, tedy cenu, kolik ji stojí nabrat do cache.

Toto má k-kompetetivní deterministický algoritmus. Je to ekvivalentní k-serveru na metrice „hvězdička“.

Existuje řádově $\log k$ kompetitivní pravděpodobnostní algoritmus.

8 Stránkování souborů

Každá stránka má dva parametry – velikost a cena.

Existuje k-kompetitivní deterministický a $O(\log^2 k)$ kompetitivní pravděpodobnostní pro cenu 1 a rovnou velikosti.

9 Problém pohybu

Máme robota ve skladišti, kde jsou krabice, má se dostat k jedné (dané) straně. Když narazí na něco, tak to detekuje.

Na toto může existovat vejlépe $\sqrt{n}$ kompetitivní algoritmus. Obrázek – překrývající-se „cihličky“. Vezmu ty, které potká, musím obcházet a to k-krát.