1 Aproximačné algoritmy

Predpokladáme, že čitateľ pozná techniky návrhu efektívnych algoritmov a stretol sa s analýzou ich časovej zložitosti. Je tiež užitočné, ak má predstavu o základoch teórie zložitosti, aj keď nateraz nám bude stačiť, že za efektívne riešiteľné považujeme tie problémy, pre ktoré existuje polynomiálny algoritmus (t.j. algoritmus, ktorý pre každý vstup vráti správny výstup a jeho čas behu je ohraničený polynómom od veľkosti vstupu) a že existuje veľa problémov, pre ktoré žiaden takýto algoritmus nepoznáme.
V nasledujúcom texte pokračujeme z tohoto východiska. Budeme sa, až na zopár výnimiek, zaoberať problémami, pre ktoré nepoznáme polynomiálny algoritmus, a napriek tomu by sme ich chceli nejako efektívne riešiť. Prijmeme tézu, že efektívne riešenie musí byť v polynomiálnom čase, a preto v ďalšom texte budeme, okrem prípadov, keď vyslovene povieme inak, za *algoritmus* považovať *algoritmus pracujúci v polynomiálnom čase*. Budeme skúmať jeden z možných prístupov: ak už nevieme spraviť algoritmus, ktorý vždy vráti správne riešenie, možno by sme sa dokázali uspokojiť s algoritmom, ktorý vždy nájde "takmer správne" riešenie. Toto, samozrejme, závisí od typu problému. Napr. pre rozhodovacie problémy (t.j. také, kde odpoveď je *áno* alebo *nie*) je každá odpoveď "takmer správna". Veľa zaujímavých problémov sú ale *optimalizačné problémy*. Neformálne, pre každý vstup optimalizačného problému existuje množina tzv. *prípustných riešení*. Navyše, každé prípustné riešenie má istú *mieru* (cenu alebo zisk). Cieľom je navrhnúť algoritmus, ktorý pre každý vstup nájde prípustné riešenie s optimálnou (minimálnou alebo maximálnou) mierou. Príkladom optimalizačného problému je problém batoha:

Definícia 1.1. Na vstupe je daných n vecí s cenami $c_1, \ldots, c_n$ a objemami $v_1, \ldots, v_n$ (pričom ceny aj objemy sú prirodzené čísla). Takisto je dané prirodzené číslo B, ktoré reprezentuje veľkosť batoha. Cieľom prolému MAX-KNAPSACK je vybrať množinu vecí, ktoré sa zmestia do batoha a ich cena je maximálna, t.j. nájsť množinu $\mathcal{I} \subseteq \{1, 2, \ldots, n\}$ takú, že $\sum_{i \in \mathcal{I}} v_i \leq B$ a maximalizuje sa $\sum_{i \in \mathcal{I}} c_i$ spomedzi všetkých množín $\mathcal{I}$.

Prípustné riešenia sú všetky výbery, ktoré sa zmestia do batoha, miera (zisk) riešenia je súčet cien vybratých vecí a cieľom je nájsť prípustné riešenie, ktoré maximalizuje zisk. MAX-KNAPSACK je príkladom problému, pre ktorý nepoznáme polynomiálny algoritmus.
Prvým pokusom v nami naznačenom smere by mohlo byť navrhnúť algoritmus, ktorý vždy nájde prípustné riešenie s cenou najviac o konštantu menšou ako optimálne riešenie, t.j. chceme mať algoritmus `A` a konštantu c tak, že pre každý vstup x, `A` nájde riešenie s cenou aspoň $OPT(x) - c$, kde $OPT(x)$ je cena optimálneho riešenia. O taktomto algoritme povieme, že má *absolútnu chybu* c. Po krátkej úvahe ale zistíme, že takto postavený cieľ je rovnako ťažký ako pôvodný.

Veta 1.2. *Ak existuje polynomiálny algoritmus pre* MAX-KNAPSACK *s absolútnou chybou c, potom existuje aj polynomiálny exakntý algoritmus.*

Dôkaz: Nech existuje taký algoritmus `A` s absolútnou chybou c. Zostrojíme algoritmus `A'`, ktorý modifikuje daný vstup x tak, že objemy vecí aj veľkosť batoha ponechá, ale ceny prenásobí koeficientom $c + 1$. Na takto modifikovaný vstup x' potom použije algoritmus `A`. Ten vráti riešenie, ktoré je prípustné pre x' a má cenu aspoň $OPT(x') - c$. Lenže v x' sú ceny všetkých vecí násobky $c + 1$, a teda jediná možná cena riešenia v intervale $\langle OPT(x') - c, OPT(x') \rangle$ je $OPT(x')$. Teda `A` musel nájsť optimálne riešenie pre vstup x'. Vidno ale, že prípustné riešenia problémov x a x' si jednoznačne zodpovedajú, preto máme aj optimálne riešenie pre vstup x. □

Z predchádzajúceho dôkazu vidno, v čom je problém absolútnej chyby: ak máme úlohu, v ktorej sa cena všetkých prípustných riešení dá rovnomerne "nafúknuť", vieme algoritmus s konštantnou

absolútonu chybou prinútiť, aby našiel optimum. Túto "nafukovaciu" vlastnosť má veľa problémov, ktoré nás budú zaujímať, a preto až na niekoľko výnimiek nebudeme mať algoritmy s ohraničenou konštantou chybou.

Trieda APX

Druhý prístup k meraniu chyby, ktorý je v praxi častý, je merať *relatívnu* chybu, t.j. o akú pomernú časť sme sa pomýlili. Formálne to docielime tak, že absolútnu chybu preškálujeme tak, aby sme dostali číslo z intervalu $\langle 0, 1)$

Definícia 1.3. Majme vstup x a prípustné riešenie y s cenou $c(y)$. Relatívna chyba riešenia y je

$$\mathcal{E}(x, y) = \frac{\mid OPT(x) - c(y) \mid}{\max\{\ OPT(x),\ c(y)\ \}}$$

Predchádzajúca definícia je zapísaná tak, aby zahŕňala maximalizačné aj minimalizačné problémy, ale pre názornosť je lepšie si ju rozpísať zvlášť. Pre maximalizačné problémy je relatívna chyba

$$\mathcal{E}_{\max}(x, y) = \frac{OPT(x) - c(y)}{OPT(x)} = 1 - \frac{c(y)}{OPT(x)}$$

a pre minimalizačné problémy

$$\mathcal{E}_{\min}(x, y) = \frac{c(y) - OPT(x)}{c(y)} = 1 - \frac{OPT(x)}{c(y)}$$

V oboch prípadoch vidno, že optimálny algoritmus vždy vráti riešenie s cenou $OPT(x)$, a preto relatívna chyba je 0. Zároveň, pretože budeme uvažovať iba problémy s nezápornými cenami, je relatívna chyba vždy menšia ako 1. Pre maximalizačné problémy má relatívnu chybu 1 triviálny algoritmus, ktorý vráti riešenie s hodnotou 0 (horšie to pri nezáporných cenách nemôže byť); pre minimalizačné problémy sa relatívna chyba blíži k 1 pre algoritmus, ktorého cena riešenia sa blíži k ∞. Aby sme predišli technickým problémom, budeme predpokladať, že $\max\{\ OPT(x),\ c(y)\ \} > 0$. Ďalší patoogický prípad môže nastať, ak máme minimalizačný problém, ktorého optimum je 0, vtedy totiž každý algoritmus vracajúci kladné riešenie má relatívnu chybu 1. Namiesto toho, aby sme v definícii takéto prípady ošetrili, ostaneme pri tejto jednoduchšej verzii a sľúbime si, že také problémy nebudeme študovať. Za "*aproximovateľné*" budeme považovať problémy, pre ktoré vieme navrhnúť algoritmus s ohraničenou relatívnou chybou:

Definícia 1.4. Triedu APX tvoria optimalizačné problémy, pre ktoré existuje polynomiálny algoritmus `A` a konštanta ε, $0 < \varepsilon < 1$ taká, že na každom vstupe je relatívna chyba riešenia algoritmu `A` najviac ε.

V niektorých prípadoch je pracovať s relatívnou chybou trochu ťažkopádne, a preto sa používa ekvivalentný pojem *aproximačného pomeru*:

$$\mathcal{R}(x, y) = \frac{1}{1 - \mathcal{E}(x, y)}$$

Rozpísané zvlášť pre maximalizačné a minimalizačné problémy:

$$\mathcal{R}_{\max}(x, y) = \frac{OPT(x)}{c(y)} \qquad\qquad \mathcal{R}_{\min}(x, y) = \frac{c(y)}{OPT(x)}$$

Vidíme, že aproximačný pomer je vždy aspoň 1 a optimálny algoritmus má aproximačný pomer 1. Algoritmus, ktorého aproximačný pomer je najviac r budeme volať *r-aproximačný*. Pre minimalizačné problémy to znamená, že vždy vráti riešenie, ktorého cena je najviac r-násobkom optima,

pre maximalizačné problémy vždy vráti riešenie, ktorého cena je aspoň $1/r$-násobok optima. Trieda APX je teda tvorená problémami, pre ktoré existuje r-aproximačný algoritmus s konštantným r.

Pozrime sa terz, či sa nám podarí navrhnúť aproximačný algoritmus z triedy APX pre problém Max-Knapsack. Máme teda dve úlohy: navrhnúť algoritmus a dokázať, že má konštantný aproximačný pomer. Druhá úloha je väčšinou ťažšia, lebo potrebujeme porovnať riešenie nášho algoritmu s optimálnym riešením, ktoré nepoznáme. Ako ešte veľakrát v tomto texte uvidíme, jadrom dôkazu je nájsť prefíkaný spôsob, ako odhadnúť optimálne riešenie. V prípade problému batoha si môžme zobrať modifikovaný problém, v ktorom povolíme veci krájať: z i-tej veci môžeme zobrať časť α_i. Dostávame nasledovný problém

Definícia 1.5. Na vstupe je daných n vecí s cenami $c_1, \ldots, c_n$ a objemami $v_1, \ldots, v_n$ (pričom ceny aj objemy sú prirodzené čísla). Takisto je dané prirodzené číslo B, ktoré reprezentuje veľkosť batoha. Cieľom prolému Max-Q-Knapsack je nájsť racionálne čísla $\alpha_1, \ldots, \alpha_n$ tak, aby pre všetky i platilo $0 \leq \alpha_i \leq 1$ a

1. $\sum_i \alpha_i v_i \leq B$
2. $\sum_i \alpha_i c_i$ je maximálne možné

Riešenia problému Max-Knapsack sú aj riešeniami problému Max-Q-Knapsack; tým, že veci môžme krájať sme množinu prípustných riešení rozšírili a optimálne riešenie sme možno zväčšili. Budeme hovoriť, že Max-Q-Knapsack je zvoľnením (*relaxáciou*) Max-Knapsack. Zároveň je problém Max-Q-Knapsack ľahko riešiteľný greedy algoritmom $\mathtt{A_g}$, ktorý utriedi veci podľa jednotkovej ceny c_i/v_i od najdrahšej po najlacnejšiu a ukladá ich do batoha (celé, t.j. s $\alpha_i = 1$) kým sa zmestia. Z prvej veci, ktorá sa nezmestí, zoberie takú časť, aby bol batoh celkom zaplnený (samozrejme, ak sa do batoha zmestia všetky veci, tak vezme všetky). Je jednoduchým cvičením presvedčiť sa, že $\mathtt{A_g}$ je optimálny: k ľubovoľnému inému prípustnému riešeniu vieme nájsť lepšie tak, že v batohu vymeníme kúsok nejakej veci za rovnako veľký kúsok drahšej nezobratej veci.
Rovnaký greedy algoritmus sa dá použiť aj na problém Max-Knapsack (s tým, že do batoha ukladá veci, kým sa zmestia, a potom skončí), ale, žiaľ, nielen že nenájde optimum, ale ani dobrú aproximáciu: uvažujme napr. veci s cenami $1, 1, \ldots, 1, 2^n - 1$ a objemami $1, 1, \ldots, 1, 2^n$, pričom batoh má veľkosť $B = 2^n$. Greedy algoritmus zoberie všetky jednotkové veci (majú jednotkovú cenu 1, kým posledná vec má jednotkovú cenu $1 - 1/2^n$) a skončí s riešením s cenou $n - 1$, pretože posledná vec sa do batoha nezmestí. Optimálne riešenie je ale zobrať poslednú vec, ktorá zaplní celý batoh, a zarobiť $2^n - 1$. Aproximačný pomer greedy algoritmu je teda aspoň $(2^n - 1)/(n - 1)$, čo v žiadnom prípade nie je konštanta pre rastúce n.
Urobme ešte zúfalý pokus na záchranu situácie: uvažujme algoritmus $\mathtt{A}$, ktorý porovná riešenie získané greedy algoritmom a riešenie, ktoré zoberie jedinú vec s najväčšou cenou $c_{\max}$ (bez ujmy na všeobecnosti predpokladáme, že každá vec sa sama osebe do batoha zmestí, ináč ju hneď v predspracovaní vyhodíme) a vráti väčšie z tých dvoch.
V nasledujúcom dôkaze sa ukáže idea, ktorá sa neskôr bude v rôznych obmenách opakovať: optimálne riešenie relaxovaného problému (ktoré poznáme) použijeme ako horný odhad optima (ktoré nepoznáme).

Veta 1.6. *Algoritmus* $\mathtt{A}$ *je 2-aproximačný.*

Dôkaz. Potrebujeme ukázať, že pre každý vstup je aproximačný pomer algoritmu $\mathtt{A}$ najviac 2, t.j. že vždy vráti riešenie s cenou aspoň $OPT/2$, kde OPT je hodnota optimálneho riešenia. Majme veci utriedené podľa jednotkovej ceny a nech j je index prvej veci, ktorá sa nezmestí celá do batoha, t.j. optimálne riešenie Max-Q-Knapsack má hodnotu $\sum_{i=1}^{j-1} c_i + \alpha c_j$ pre nejaké α, $0 < \alpha \leq 1$. Označme $\overline{c}_j = \sum_{i=1}^{j-1} c_i$. Pretože Max-Q-Knapsack je relaxáciou Max-Knapsack, je $OPT \leq \overline{c}_j + \alpha c_j \leq \overline{c}_j + c_j$. Rozlíšime dva prípady:

- Ak $\overline{c}_j \leq c_j \leq c_{max}$, máme $OPT \leq 2c_{max}$

- Ak $\overline{c}_j > c_j$, máme $OPT < 2\overline{c}_j$

Pretože riešenie, ktoré nájde algoritmus A má cenu $\max\{\overline{c}_j, c_{\max}\}$, je dôkaz hotový.

□

Lepšie ako APX: $PTAS$ a $FPTAS$

V predchádzajúcej časti sme predstavili triedu APX ako triedu "aproximovateľných" problémov. Ako je to s praktickým využitím takýchto algoritmov? Problémom tu je slovíčko "existuje konštanta" z Definície 1.4. Podobne, ako polynomiálny algoritmus, ktorého čas je ohraničený polynómom n^{234}, aj aproximačný algoritmus, ktorý vždy vráti 856-násobok optima, nie je v praxi použiteľný. Podobne, ako pri stupni polynómu, aj pri aproximačnom pomere je ťažké (a nepraktické) stanoviť konštantu, nad ktorou už príslušný aproximačný pomer nie je zaujímavý. Občas sa nám ale podarí tento problém obísť elegantne a navrhnúť tzv. *aproximačnú schému* (PTAS, polynomial-time approximation scheme):

Definícia 1.7. Triedu $PTAS$ tvoria optimalizačné problémy, pre ktoré pre každú fixnú konštantu $0 < \varepsilon < 1$ existuje polynomiálny algoritmus $\mathtt{A}_\varepsilon$, ktorý na každom vstupe vráti riešenie s relatívnou chybou najviac ε.

Ako uvidíme o chvíľu, v tejto definícii je kľúčové, že ε je fixná konštanta, t.j. čas algoritmu $\mathtt{A}_\varepsilon$ musí byť polynomiálny od veľkosti vstupu, ale môže (aj nepolynomiálne) závisieť od ε. Uvažujme problém Min-Partition. V rozhodovacej verzii je cieľom zistiť, či sa prirodzené čísla zo vstupu dajú rozdeliť na dve časti s rovnakým súčtom. V optimalizačnej verzii sa budeme snažiť nájsť "čo najpodobnejšie" časti, pričom je niekoľko spôsobov, ako definovať "čo najpodobnejšie". Keby sme napríklad povedali, že chceme minimalizovať rozdiel väčšej a menšej časti, dostali by sme minimalizačný problém, ktorého optimum môže byť 0, a také sme si sľúbili neuvažovať. Zvoľme preto inú definíciu, v ktorej sa snažíme minimalizovať veľkosť väčšej časti:

Definícia 1.8. Na vstupe je daných n prirodzených čísel $a_1, \ldots, a_n$. Úlohou problému Min-Partition je nájsť rozklad množiny $\{1, \ldots, n\}$ na dve dizjunktné množiny Y_1, Y_2 tak, aby cena

$$\max\left\{\sum_{i\in Y_1} a_i, \sum_{i\in Y_2} a_i\right\}$$

bola najmenšia možná.

Podobne ako v predchádzajúcom príklade, lepšie sa nám bude pracovať s aproximačným pomerom namiesto relatívnej chyby. Naším cieľom bude pre ľubovoľné fixné $r > 1$ navrhnúť r-aproximačný algoritmus. Pretože každé riešenie má veľkosť aspoň $1/2\sum_{i=1}^{n} a_i$, pre $r \geq 2$ stačí vrátiť rozdelenie $Y_1 = \emptyset$, $Y_2 = \{1, \ldots, n\}$. Predpokladajme preto, že $r < 2$. Pozrime sa najprv na jednoduchý "greedy" algoritmus, ktorý utriedi čísla od najväčšieho po najmenšie, a potom postupne vždy pridáva do tej časti, ktorá je práve menšia. Ľahko vidno, že takýto algoritmus nemôže mať aproximačný pomer lepší ako 7/6: pre vstup $3, 3, 2, 2, 2$, ktorého optimum je 6 $(3+3,\ 2+2+2)$, greedy algoritmus vráti rozdelenie $3+2,\ 3+2+2$ s hodnotou 7. Budeme vychádzať z intuície, že najdôležitejšie je správne rozdeliť veľké čísla, lebo pri ich nesprávnom uložení vznikne väčšia chyba. Pri malých číslach, naopak, ani greedy algoritmus veci príliš nepokazí. Navrhnime preto nasledovný algoritmus A:

1: vstup: čísla $a_1 \geq a_2 \geq \cdots \geq a_n$, a r, $1 < r < 2$
2: $k :=$ 🔑
3: nájdi optimálne riešenie Y_1, Y_2 pre $a_1, \ldots, a_k$
4: **for** $i = k+1$ to n **do**
5: pridaj i do množiny Y_ℓ s menšou hodnotou $\sum_{j\in Y_\ell} a_j$
6: **end for**

Algoritmus `A` prvých k čísel rozloží optimálne (napr. preskúšaním všetkých 2^k možností) a zvyšok dorobí greedy algoritmom. Kľúčom k riešeniu je vhodne zvoliť parameter k. Pozrime sa, aký aproximačný pomer `A` dosahuje pri zvolenom parametri k. Zaveďme najprv niekoľko označení. Nech $W_1 = \sum_{j \in Y_1} a_j$ a $W_2 = \sum_{j \in Y_2} a_j$. Bez ujmy na všeobecnosti predpokladajme $W_1 > W_2$, takže `A` nájde riešenie s hodnotou W_1. Ďalej nech $L = (W_1 + W_2)/2$, t.j. L je dolné ohraničenie optimálneho riešenia. Nech h je maximálny index prvku v Y_1.
Ak $h \leq k$, tak všetky prvky sa do Y_1 dostali na riadku 3 a v cykle na riadku 5 sa nepridá žiaden prvok.

TODO: $FPTAS$ Knapsack, $O(\log n)$ aproximácia Set Cover