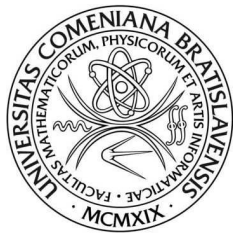
U N I V E R Z I T A K O M E N S K É H O

Fakulta matematiky, fyziky a informatiky

Katedra informatiky



Vybrané kapitoly z teoretickej informatiky-II

Riešenie ťažkých problémov

(Pomocné texty k prednáške 2–AIN–205)

(verzia 10. decembra 2012)

doc. RNDr. Dana Pardubská, CSc.

Obsah

Kapitola 1

Úvod

Pod ťažkými problémami nemáme na mysli len NP-ťažké, ale aj tie, pre ktoré nepoznáme polynomiálne algoritmy rozumného stupňa. Aké máme možnosti, keď chceme riešiť ťažké problémy deterministicky? Odpoveď záleží aj od odpovede na otázku, v čom spočíva obtiažnosť toho problému. Je problém, resp. algoritmus na jeho rešenie, zlý preto, že má niektoré zlé vstupy, alebo je zlý "vo svojej podstate"?

1. V prvom prípade máme šancu, že naše vstupy budú patriť do kategórie "vhodných" a pre ne môže existovať dobrý algoritmus. A tak sa tam, kde sa to dá, popri snahe o znižovanie zložitosti najhoršieho a priemerného prípadu snažíme (aj) o konštrukciu takých algoritmov, ktoré umožňujú vyšpecifikovať veľké/čo najväčšie množiny vstupov, na ktorých sú dokázateľné rozumné.

2. Akceptujeme exponenciálnu zložitosť najhoršieho prípadu, keď na rozumných vstupoch (dajú sa v praktických situáciách očakávať) je rozumnej zložitosti.

3. Konštruujeme exponenciálne algoritmy, dokonca aj v očakávanom prípade, ale snažíme sa znižovať tú exponencialitu (od 2^n ideme napr. k $(1.2)^n$)

4. Upustíme od toho, že náš algoritmus má na všetkých vstupoch vždy vyriešiť náš problém korektne. Dostávame dve významné triedy algoritmov. Ak v rozhodovacích problémoch pripustíme, že odpoveď je správne len s veľkou pravdepodobnosťou, môžme používať tzv. pravdepodobnostné (náhodou riadené,randomized) algoritmy. V prípade optimalizačných úloh vedie upustenie od presnosti k tzv. aproximačným algoritmom. Vyžadujeme, aby algoritmus bol rýchly a získané riešenie blízko k optimálnemu riešeniu. Často sa takto získané riešenia berú ako dobrý štart pre ďalšie metódy.

Kvôli úplnosti pridávame text o NP-úplnosti, ktorý je čiastočne opakovaním konca minulého semestra.

Kapitola 2

NP-úplnosť

O význame triedy P ako triede prakticky riešiteľných problémov sme už hovorili. Videli sme, že polynomiálny čas je taká vlastnosť problémov, ktorá je nezávislá od výberu výpočtového modelu, pokiaľ sa hýbeme v prvej počítačovej triede. Triedu P rozhodne môžme považovať za triedu prakticky riešiteľných úloh. (Neskôr došlo k posunu – za prakticky riešiteľné považujeme aj pravdepodobnostné a aproximačné algoritmy polynomiálnej zložitosti.)

Je veľa takých úloh, pre ktoré nepoznáme deterministcké polynomiálne riešenie; všetky známe deterministické algoritmy sú exponenciálnej zložitosti, často založené na preberaní všetkých možností. Mnohé z nich sú však také, že ak *poznáme riešenie*, jeho správnosť vieme overiť v polynomiálnom čase. Konštrukcia nedeterministického stroja na riešenie tohto problému metódou "uhádni a over" tak v prípade, že riešenie je polynomiálne od veľkosti vstupu, vedie k nedeterministickému polynomiálnemu času. Dostali sme sa tak trochu inak k triede NP - sú to tie problémy, ktorých riešenie vieme overiť v čase polynomiálnom od veľkosti vstupu[1]. Z tohto pohľadu je otázka vzťahu P?NP vlastne otázkou— je ľahšie nájsť riešenie alebo overiť jeho správnosť?

Nedeterminizmus priniesol do riešenia problémov novú kvalitu. Hoci pochopenie tohto konceptu nie je jednoduché, nedeterministické riešenia—algoritmy—sú často zrozumiteľnejšie, elegantnejšie. Vzhľadom na simulácie z predchádzajucej časti vieme, že prechod od nedeterminizmu k determinizmu nás môže stáť až exponenciálny nárast času. Je to nutné? Sú problémy riešiteľné v nedeterministicom polynomiálnom čase tak ťažké, že sa nedajú riešiť v polynomiálnom deterministickom čase? Otázka vzťahu tried P a NP patrí k tým najvýznamnejším. Väčšina informatikov predpokladá, že $P \neq NP$ a namiesto riešenia vzťahu týchto dvoch tried sa tak sústreďujeme skôr na to, ako riešiť tie problémy, ktoré sú ťažké.

Kedy hovoríme, že problém je ťažký? Napr. vtedy, ak nepatrí do P. Dokázať však o konkrétnom probléme, že nepatrí do triedy P, je nesmierne obtiažne, čo vlastne znamená, že takáto "definícia" pojmu ťažký je nám zbytočná. Uvidíme však, že existuje "nástroj", pomocou ktorého pre mnohé problémy pomerne ľahko ukážeme, že sa za predpokladu $P \neq NP$ v polynomiálnom deterministickom čase riešiť nedajú. Týmto nástrojom sú NP-úplné problémy. No a to, v situácii, keď akceptujeme $P \neq N$, poskytuje dobrú formuláciu toho, čo je ťažké.

K čomu je to dobré? Ak ukážeme, že problém je NP-úplný, je ťažký a teda máme dobrý dôvod sa domnievať, že sa nedá riešiť v P. No a keď to nejde deterministicky polynomiálne, treba z niečoho, konkrétne nejakých požiadaviek na riešenie, zľaviť.

[1] Uvedomme si, že tým implicitne hovoríme, že riešenie je polynomiálne vzhľadom na veľkosť vstupu.

Dostávame sa tak k *aproximačným* algoritmom (v prípade optimalizačných úloh upúšťame od požiadavky na optimálne riešenie a uspokojíme sa s približným, ktoré je rýchlo a vieme ohraničiť, ako ďaleko od optimálneho), *pravdepodobnostným* algoritmom (upúšťame od toho, že riešenie je vždy správne, vyžadujeme však, aby bolo rýchlo a aby pravdepodobnosť korektnej odpovede bola vysoká; väčšinou vyššia ako 0,5), *heuristickým algoritmom* (algoritmus sleduje nejakú rozumnú stratégiu, ktorá však negarantuje, že dostaneme korektné riešenie; môže sa stať, že riešenie vôbec nedostaneme napriek tomu, že existuje.).

2.1 Redukcie, NP-úplnosť

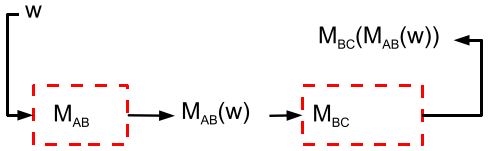
Aby sme mohli hovoriť o ťažších a ľahších problémoch, potrebujeme mať možnosť ich porovnávať. V snahe o vymedzenie najťažších problémov v triede NP začneme definíciami redukcií, ktoré umožňujú obtiažnosť problémov porovnávať.

polynomiálna redukcia $L \rightsquigarrow L'$

Definícia 2.1 *Jazyk L sa nazýva* polynomiálne redukovateľný *na jazyk L', ak existuje polynóm p a $p(n)$-časovo ohraničený DTS M taký, že M prepíše každý vstupný reťazec $w \in \Sigma_L^*$ na reťazec $w' \in \Sigma_{L'}^*$, tak, že $w \in L \Leftrightarrow M(w) = w' \in L'$.*

tranzitivita $\rightsquigarrow$

Ľahko vidno, že relácia polynomiálnej redukcie je tranzitívna: Nech $A \rightsquigarrow B$ a $B \rightsquigarrow C$, M_{AB}, M_{BC} sú DTS počítajúce príslušné redukcie a p_{AB}, p_{BC} sú polynómy ohraničujúce ich časovú zložitosť. Nech α, β je vstup do M_{AB}, resp. M_{BC}. Potom $|M_{AB}(\alpha)| \leq p_{AB}(\alpha)$; analogicky $|M_{BC}(\beta)| \leq p_{BC}(\beta)$.
Redukciu $A \rightsquigarrow C$ ľahko realizujeme napr. zreťazením strojov M_{AB}, M_{BC}: $M_{AC}(w) = M_{BC}(M_A B(w))$.



Pre čas $t(w)$ výpočtu redukcie $M_{AC}(w)$ zrejme platí

$$t(w) \leq p_{AB}|w| + p_{BC}(M_{AB}(w)) \leq p_{AB}|w| + p_{BC}(|p_{AB}(|w|)|)$$

Keďže skladaním polynómu dostávame opäť polynóm, existuje polynóm p_{AC} taký, že

$$t(w) \leq p_{AC}(|w|)$$

A teraz už môžme definovať NP-úplnosť.

NP-ťažký, NP-úplný jazyk

Definícia 2.2 *Jazyk L_0 sa nazýva* NP-ťažký, *ak $\forall L \in NP$: L je polynomiálne redukovateľný na L_0. Ak navyše $L_0 \in NP$ tak je jazyk* NP-úplný.
Triedu NP-úplných problémov budeme označovať NPU.

O význame NP-úplných problémov hovorí nasledujúca veta, najmä podmienka 2. Jej dôkaz prenechávame čitateľovi.

Veta 2.1

1. *Ak $NPU \cap P \neq \emptyset$, tak $P = NP$*
2. *Ak $P \neq NP$, tak $NPU \subset NP \setminus P$*

Zo znenia vety vidno, že ak chceme ukázať rovnosť tried $P = NP$, stačí pre jediný NP-úplný problém nájsť deterministický polynomiálny algoritmus.

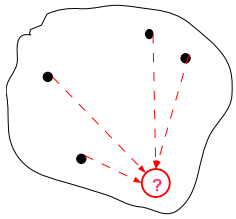
Naopak – ak prijmeme hypotézu , že $P \neq NP$, je dôkaz NP-úplnosti dôkazom toho, že problém je z $NP \setminus P$. Takto nám pojem NP-úplnosti poskytuje možnosť alternatívneho "dôkazu" toho, že problém nie je prakticky riešiteľný.

Existuje aj iná definícia NP-úplnosti, s ktorou sa stretávame v prípade problémov, ktoré nie sú rozhodovacie.

Definícia 2.3 *Problém L_0 z NP je NP-úplný, ak z existencie algoritmu polynomiálnej zložitosti $T(n)$ riešiaceho L_0 vyplýva existencia algoritmu zložitosti $p_L(T(n))$ pre každý problém $L \in NP$, pričom p_L je polynóm, ktorý závisí od L.*

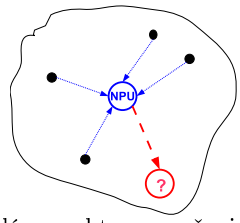
Pri dokazovaní NP-úplnosti robí problém najmä dôkaz toho, že problém je NP-ťažký. Tu využijeme tranzitivitu relácie "byť polynomiálne redukovateľný".

NP-úplnosť redukciou



Nech problém $L_0 \in NP$ je NP-úplný (na obr. NPU). Potom ho môžme využiť pri dôkaze NP-úplnosti problému L (na obr. ?) nasledovne:

1. ukážeme, že $L \in NP$
2. ukážeme, že L_0 je polynomiálne transformovateľný na L; toto dokazuje polynomiálnu redukovateľnosť ľubovoľného L' na L (prečo?)



K použitiu tohto prístupu však potrebujeme nejaký problém, o ktorom už vieme, že je NP-úplný. NP-úplnosť prvého problému musíme dokázať podľa definície.

2.2 NP-úplnosť problému splniteľnosti BF

Hlavným výsledkom tejto časti je Cookova veta, ktorá o probléme splniteľnosti boolovskej formuly dokáže, že je NP-úplný.

Definícia 2.4 Boolovskou formulou*(BF) nazveme výrazy*

- $x, \neg x$
- *ak p, q sú boolovské formuly, tak aj $(p \vee q), (p \wedge q), \neg p$ sú boolovské formuly*
- *iné boolovské formuly nie sú*

Nech BF je Boolovská formula n premenných $x_1, \ldots, x_n$; označujeme $BF(x_1, \ldots, x_n)$. Hovoríme, že BF je splniteľná*, ak existuje také priradenie hodnôt $0, 1$ premenným $x_1, \ldots, x_n$, pri ktorom BF nadobúda hodnotu 1*

Problém splniteľnosti BF

Vstup: Boolovská formula BF, resp. reťazec, ktorý ju kóduje
Výstup: 1, ak je formula splniteľná
0 inak

Podľa potreby a kontextu bude SAT označovať alebo problém splniteľnosti BF alebo triedu splniteľných BF.

Veta 2.2 (Cook) *Problém splniteľnosti BF je NP-úplný*

Dôkaz: Príslušnosť problému splniteľnosti do NP ukážeme konštrukciou NTS, ktorý ho rieši.

- nech vstupom je $BF(x_1, \ldots, x_n)$; nech m označuje dĺžku vstupu
- DTS uhádne priradenie hodnôt $\alpha_1, \ldots, \alpha_n$ premenným $x_1, \ldots, x_n$ (prechádzaním po vstupnej páske si na pracovnú pásku zapisuje premenné a im uhádnuté hodnoty; časová zložitosť $O(m^2)$)
- dosadí uhádnuté priradenie hodnôt $\alpha_1, \ldots, \alpha_n$ do BF (prechádzaním po vstupnej páske opisuje na druhú pracovnú pásku BF zo vstupu, pričom namiesto jednotlivých premenných píše hodnoty; časová zložitosť $O(m^2)$)
- vyhodnotením BF s dosadenými hodnotami overí, či $BF(\alpha_1, \ldots, \alpha_n) = 1$; časová zložitosť $O(m^2)$

$L \in NP \Rightarrow$ L **je polynomiálne redukovateľný na SAT** K dôkazu tejto časti treba skonštruovať DTS polynomiálnej zložitosti, ktorý pri vstupe w v čase polynomiálnom skonštruuje BF tak, že $w \in L \Leftrightarrow BF \in SAT$. Ak pri konštrukcii DTS využijeme len tie vlastnosti/znalosti o jazyku L, ktoré sú spoločné všetkým jazykom z NP, bude táto konštrukcia platná pre všetky jazyky z NP.

Ak $L \in NP$, potom existuje NTS M rozpoznávajúci L; $L = L(M)$. O NTS M môžme bez ujmy na všeobecnosti predpokladať:

- $NTSM$ je jednopáskový
- vstupná abeceda $\Sigma = \{0, 1\}$
- abeceda symbolov $\Gamma = \{1, 2, \ldots, s\}$; pritom predpokladáme, že (napr) 1 označuje B (blank)
- množina stavov je $K = \{1, 2, \ldots, q\}$; nech 1 je počiatočný stav, q koncový akceptujúci
- časová zložitosť stroja M je zhora ohraničená polynómom $p(n)$

O výpočte $C = C_1, C_2, \ldots, C_T$ NTS M môžme bez ujmy na všeobecnosti predpokladať, že

- je dĺžky *presne* $p(n)$
- každá konfigurácia je veľkosti $p(n)$ - uvažujeme teda aj prípadne "blanky"

Booleovská formula BF_w vytvorená DTS k vstupnému slovu w (pri znalosti popisu NTS M) bude používať tri skupiny premených. Ak zafixujeme nejaký konkrétny výpočet NTS M $C = C_1, C_2, \ldots, C_{p(n)}$, možno sa na tieto premenné pozerať ako na predikáty.

$C(i,j,t)$,	$1 \le i \le p(n), 1 \le j \le s,$ $0 \le t \le p(n)$	?je pravda, že vo výpočte C je v čase t na i-tom políčku symbol j?
$S(k,t)$,	$1 \le k \le q, \le t \le p(n)$	?je pravda, že vo výpočte C je v čase t stroj v stave k?
$H(i,t)$,	$1 \le i \le p(n), 0 \le t \le p(n)$	?je pravda, že vo výpočte C je v čase t hlava na i-tom políčku?

Máme tak $O(p^2(n))$ premenných, na zápis každej z nich stačí priestor $O(\log n)$.

Nech $Q_0, Q_1, \ldots, Q_{p(n)}$ je popis postupnosti konfigurácií. Tento popis je korektným *popisom akceptujúceho výpočtu*, ak platí nasledujúcich 7 podmienok:

1. Q_0 je počiatočná konfigurácia
2. V každom čase je hlava na práve jednom políčku
3. V každom čase je stroj práve v jednom stave
4. V každom čase je na každom políčku pásky práve jeden symbol
5. Mení sa len to políčko, na ktorom je nastavená hlava
6. Zmena sa deje podľa prechodovej funkcie
7. Posledná konfigurácia je akceptujúca

Teraz popíšeme výslednú formulu BF_w. Táto vznikne ako logický súčin formúl ABCDEFG. Pri konštrukcii týchto podformulí využijeme predikát

$$U(y_1, y_2, \ldots, y_n) = (y_1 \vee y_2 \vee \ldots \vee y_r) \cdot \prod_{i \neq j} (\neg y_i \vee \neg y_j)$$

ktorý nadobúda hodnotu 1 práve vtedy, keď *práve jedna* z premenných $y_1, \ldots, y_r$ nadobúda hodnotu 1. Uvedomme si, že zápis $U(y_1, \ldots, y_r)$ je len označenie formuly, ktorá používa $O(r^2)$ premenných a symbolov.

K 1. Q_0 je počiatočná konfigurácia

$$A = S(1,0) \wedge H(1,0) \wedge \prod_{1 \leq i \leq n} C(i, w_i, 0) \wedge \prod_{n < i \leq p(n)} C(i, 1, 0)$$

K 2. V každom čase je hlava na práve jednom políčku Zafixujme čas t. Potom

$$B_t = U(H(1,t), \ldots, H(p(n), t))$$

Keďže podmienka má platiť v každom časovom okamihu,

$$B = \prod_{0 \leq t \leq p(n)} B_t$$

K 3. V každom čase je stroj práve v jednom stave

$$C = \prod_{0 \leq t \leq p(n)} U(S(1,t), \ldots, S(q,t))$$

K 4. V každom čase je na každom políčku pásky práve jeden symbol Zafixujme najprv čas t a políčko i

$$D_{i,t} = U(C(i,1,t), \ldots, C(i,s,t))$$

Podmienka má platiť v každom časovom okamihu a na každom políčku, preto

$$D = \prod_{\substack{1 \leq p(n) \\ 0 \leq t \leq p(n)}} D_{i,t}$$

K 5. Mení sa len to políčko, na ktorom je nastavená hlava
To znamená, že alebo je obsah políčka v dvoch po sebe idúcich časových okamihoch rovnaký, alebo na ňom bola nastavená hlava. Zafixujme najprv čas, políčko aj symbol

$$E_{i,j,t} = (C(i,j,t) \equiv C(i,j,t+1)) \vee H(i,t)$$

Podmienka má platiť v každom čase (pre každé políčko a symbol)

$$E = \prod_{\substack{1 \leq i \leq p(n) \\ 0 \leq t \leq p(n) \\ 1 \leq j \leq s}} E_{i,j,t}$$

K 6. Zmena sa deje podľa prechodovej funkcie
Zafixujme čas t, políčko i, symbol j a stav k. Tieto štyri hodnoty spolu alebo súvisia – v čase t je stroj v stave k, hlavu má na políčku i a na tomto políčku je symbol j alebo spolu nesúvisia. Nech

$$\delta(j,k) = \{(j_1, k_1, pos_1), \ldots (j_{m(j,k)}, k_{m(j,k)}, pos_{m(j,k)})\}$$

Potom

$$F_{i,j,k,t} = \neg C(i,j,t) \vee \neg S(k,t) \vee \neg H(i,t) \vee \sum_{1 \leq l \leq m(j,k)} C(i, j_l, t+1) \wedge H(i+pos_l, t+1) \wedge S(k_l, t+1)$$

Podmienka platí pre všetky hodnoty i, j, k, t

$$F = \prod_{\substack{1 \leq i \leq p(n) \\ 0 \leq t \leq p(n) \\ 1 \leq j \leq s,\ 1 \leq k \leq q}} F_{i,j,k,t}$$

K 7. Posledná konfigurácia je akceptujúca
Pri platnosti predchádzajúcich podmienok to znamená, že psledný stav má byť akceptujúci

$$G = S(q, p(n))$$

Výstupom DTS je teda výsledná formula $BF_w = A \wedge B \wedge C \wedge D \wedge E \wedge F \wedge G$. Vzhľadom k tvaru podformúl A, B, C, D, E, F, G je zrejmé, že časová zložitosť DTS je polynomiálna vzhľadom na veľkosť výslednej BF_w. Sústredíme sa preto len na veľkosť BF_w.

A	$O(p(n) \log n)$
B	$O(p^3(n) \log n)$
C	$O(q^2 p(n) \log n)$
D	$O(s^2 p(n) \log n)$
E	$O(p^2(n) s \cdot \log n)$
F	$O(p^2(n) s \cdot q \cdot M \cdot \log n)$, kde $M = \max\{m(i,j)\}$
G	$O(\log n)$

Výsledná veľkosť BF_w je $O(p^3(n) \log n)$. Ešte ostáva ukázať, že $w \in L \Leftrightarrow BF_w \in SAT$.

$$w \in L$$

$$\Updownarrow$$

$$\exists \text{ akceptujúci výpočet } Q = Q_0, Q_1, \ldots, Q_{p(n)}$$

$$\Updownarrow$$

priradenie hodnôt premenným $C(i,j,t), S(k,t), H(i,t)$, podľa predpisu

$C(i,j,t) = 1 \Leftrightarrow$ je pravda, že vo výpočte Q je na i-tom políčku v čase t symbol j

$S(k,t) = 1 \Leftrightarrow$ je pravda, že vo výpočte Q je v čase t stroj v stave q

$H(i,t) = 1 \Leftrightarrow$ je pravda, že vo výpočte Q je v čase t hlava na i-tom políčku

zabezpečuje, že $BF_w(C(i,j,t), S(k,t), H(i,t)) = 1$

□

Keď že vytvorenú formulu BF_w vieme v polynomiálnom čase previesť na formulu v tvare KNF, dostávame nasledujúci dôsledok.

Dôsledok 2.3 *Problém splniteľnosti formuly v tvare KNF je z NP.*

2.3 Niektoré NP-úplné problémy

Keď už máme NP-úplný problém, metódou redukcie ukážeme NP-úplnosť aj o niektorých ďalších problémoch. Význam tejto časti je na jednej strane v prezentácii redukcií ako takých, na strane druhej v jej tvrdeniach; metódami na riešenie NP-ťažkých problémov sa budeme zaoberať neskôr, pričom ich budeme vysvetľovať aj na niektorých z tu prezentovaných problémov.

2.3.1 Problém 3-spniteľnosti je NP-úplný.

Začnime definíciou problému.

Problém 3 splniteľnosti

Vstup: formula $3BF$ v tvare $3KNF = F_1 \wedge F_2 \wedge \ldots \wedge F_q$, $F_i = x_{i_1}^{\sigma_{i_1}} \vee x_{i_2}^{\sigma_{i_2}} \vee x_{i_3}^{\sigma_{i_3}}$
Výstup: $1 \Leftrightarrow 3BF \in SAT$

NP-úplnosť problému 3-splniteľnoasti dokazujeme redukciou z problému splniteľnosti formuly v tvare KNF. Časť, že **3-Splniteľnosť** $\in NP$ dokazovať netreba, pretože všebecný problém splniteľnosti je z NP.

Polynomiálna redukcia KNF splniteľnosti na 3 splniteľnosť: Nech KNF je formula v tvare KNF. Bez ujmy na všeobecnosti môžme predpokladať, že je to formula tvaru $(x_1 \vee \ldots \vee x_n)$. Uvažujme formulu

$$3BF = (x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee x_3 \vee y_2) \wedge \ldots \wedge (\overline{y_{i-2}} \vee x_i \vee y_{i-1}) \wedge \ldots \wedge (\overline{y_{n-3}} \vee x_{n-1} \vee x_n)$$

kde y_j sú nové premenné. Túto formulu zrejme na TS vytvoríme v čase polynomiálnom od veľkosti pôvodnej formuly BF.

BF ∈ SAT Nech $\alpha = (\alpha_1, \ldots, \alpha_n)$ je spĺňajúci vektor hodnôt, na ktorom $BF(\alpha) = 1$. Potom existuje také i, že v α je $x_i = 1$. Definujme vektor hodnôt β premenných $y_1, \ldots, y_{n-3}$ $BF \longrightarrow 3BF$

$$\begin{cases} y_j = 1 & \text{pre } j \leq i-2 \\ \\ y_j = 0 & \text{pre } i-2 < j \leq n-3, \end{cases}$$

Potom $3BF(\alpha, \beta) = 1$.

$3BF \in SAT$ Nech $\alpha = (\alpha_1, \ldots, \alpha_n)$ je vektor hodnôt, na ktorom $3BF(\alpha) = 1$. Stačí ukázať, že existuje i také, že pre $\alpha = \alpha_1, \ldots, \alpha_n,\ i = 1$. Postupujme sporom. Nech by $x_i = 0$ pre každé i. Keďže $3BF(\alpha) = 1$, musí platiť: *3BF⟶BF*

$$y_{n-3} = 0 \Rightarrow y_{n-2} = 0 \Rightarrow \ldots \Rightarrow y_1 = 0$$

To ale znamená, že $(x_1 \vee x_2 \vee y_1) = 0$, čo je spor s predpokladom, že $3BF(\alpha) = 1$.

všeobecný prípad V prípade, keď formula F nie je $(x_1 \vee \ldots \vee x_n)$ ale $(x_1^{\sigma_1} \vee \ldots \vee x_n^{\sigma_n})$, zameníme v konštruovanej 3BF výskyt x_i za $x_i^{\sigma_i}$.
Ak F je konjunkcia viacerých elementárnych disjunkcií, každá konjunkcia pracuje s novými pomocnými premennými.

dorob

□

2.3.2 Problém k-klika je NP-úplný

Začnime definíciou problému.

Problém k-kliky

Vstup: $G = (V, E),\ k \in \mathbb{N}$
Výstup: $1 \Leftrightarrow G$ obsahuje úplný podgraf o k vrcholoch

NP-úplnosť problému k-kliky dokazujeme redukciou z problému splniteľnosti formuly v tvare KNF. Časť **problém k-kliky je z NP** prenechávame čitateľovi.

Redukcia spočíva v dvoch krokoch - k vstupnej formule $F = F_1 \wedge F_2 \wedge \ldots \wedge F_q$ zostrojíme v polynomiálnom čase graf $G = (V, E)$, o ktorom dokážeme, že obsahuje q-kliku práve vtedy, keď formula F je splniteľná.

Konštrukcia grafu G Nech $F = F_1 \wedge F_2 \wedge \ldots \wedge F_q,\ F_i = x_{i1} \vee x_{i2} \vee \ldots \vee x_{ik(i)}$.

Množinu vrcholov vytvoríme tak, že ku každému výskytu literálu vo formule F vytvoríme jeden vrchol. Hranu medzi dva vrcholy pridáme vtedy, ak príslušné literály *môžu* mať v priradení hodnôt rovnakú hodnotu. Inými slovami nepridáme hranu medzi také vrcholy, ktorých literály *nemôžu* mať rovnakú hodnotu (x a $\neg x$ nemôžu mať rovnakú hodnotu).

Formálnejšie: množinu vrcholov V tvoria dvojice

$$V = \{[i, j] \mid 1 \leq i \leq q, 1 \leq j \leq k(i)\}$$

Prvá zložka vrchola $[i, j]$ ukazuje na i-tu elementárnu disjunkciu formuly F, druhá zložka na literál na j-tej pozícii v nej.

Množina hrán E je tvorená množinou dvojíc vrcholov

$$E = \{([i, j], [k, l]) \mid i \neq k, x_{ij} \neq \neg x_{kl}\}$$

Ak literály odpovedajúce hranou spojeným vrcholom obsahujú rovnakú premennú, potom sú tieto literály rovnaké:

$$(([i, j], [k, l]) \in E, x_{ij} = x_m^{\sigma_m}, x_{kl} = x_t^{\sigma_t} \wedge m = t) \Rightarrow (\sigma_m = \sigma_t)$$

Je zrejmé, že tento popis vieme na DTS vytvoriť v polynomiálnom čase.

$F \longrightarrow G$

F je splniteľná Nech formula F je splniteľná. Potom existuje súbor hodnôt α taký, že $F(\alpha) = 1$. Nech x_{i,m_i} je ten literál elementárnej disjunkcie F_i, ktorý sa vyhodnocuje na 1. Tvrdíme, že množina vrcholov $\{[i, m_i], 1 \leq i \leq q\}$ tvorí kliku v grafe G. K dôkazu stačí overiť, že $\forall i \neq j$ tvorí dvojica vrcholov $[i, m_i], [j, m_j]$ hranu v grafe G. Nech by existovali také $i \neq j$, že $([i, m_i], [j, m_j]) \notin E$. Keďže $i \neq j$, znamená to, že $x_{i,m_i} = \neg x_{j,m_j}$. To je ale v spore s faktom, že $x_{i,m_i} = x_{j,m_j} = 1$

$G \longrightarrow F$

G obsahuje q-kliku Nech q-kliku tvorí množina vrcholov $\{[i, m_i] 1 \neq i \neq q\}$. Vytvorme dve množiny premenných
$S1 = \{y \mid y = x_{i,m_i}, y \text{ je premenná}, 1 \leq i \leq q\}$
$S0 = \{y \mid y = \neg x_{i,m_i}, y \text{ je premenná}, 1 \leq i \leq q\}$
Sporom ukážeme, že $S1 \cap S0 = \emptyset$. Ak by $\exists i \neq j$ také, že $x_{i,m_i} = y = \neg x_{j,m_j}$, potom by platilo, že $([i, m_i], [j, m_j]) \notin E$.
Potom pre vektor hodnôt α, ktorý vznikne tak, že $y = h$ ak $y \in Sh,\ h = 0, 1$ platí: $F(\alpha) = 1$

□

Cvičenie 2.1 *Uvažujme* ***Problém k-pokrytia***

Vstup: $G = (V, E),\ k \in \mathbb{N}$
Výstup: $1 \Leftrightarrow$ *ak existuje množina vrcholov* $S,\ S \subseteq V$, *mohutnosti* k *tak, že* $\forall (u, v) \in E : (u \in S$ *alebo* $v \in S)$

Redukciou z problému k-kliky dokážte, že problém k-pokrytia je NPÚ.

2.3.3 Problém k-zafarbiteľnosti je NP-úplný

Začnime definíciou problému.

Problém k-zafarbiteľnosti

Vstup: $G = (V, E),\ k \in \mathbb{N}$
Výstup: 1 ak množinu vrcholov V vieme zafarbiť k farbami tak, že žiadna hrana nemá oba konce rovnakej farby.

NP-úplnosť problému k-zafarbiteľnosti dokazujeme redukciou z problému 3splniteľnosti. To, že problém k-zafarbiteľnosti patrí do NP prenechávame čitateľovi.

Redukciu spravíme v dvoch krokoch - k danej vstupnej formule $F = F_1 \wedge \ldots \wedge F_t$ nad premennými $x_1, \ldots, x_n$ zostrojíme graf $G = (V, E)$, o ktorom ukážeme, že F je splniteľná práve vtedy keď G je $(n+1)$-zafarbiteľný.

konštrukcia G

Graf G budeme konštruovať tak, aby sme vynútili použitie n farieb (úplný graf na n vrcholoch) a aby stačila jedna ďalšia farba práve vtedy, ak je formula splniteľná. Farbenie touto novou farbou bude odpovedať priradeniu hodnoty 0 príslušnej premennej.

$V = \{x_i, \neg x_i, v_i, 1 \leq i \leq n\} \cup \{F_i, 1 \leq i \leq t\}$

$E =$	$\{(v_i, v_j)\ i \neq j\} \cup$	vynútili sme si n farieb
	$\{(v_i, x_j), (v_i, \neg x_j)\ i \neq j\} \cup$	x_i môže byť rovnakej farby ako v_i stačí n farieb
	$\{(x_i, \neg x_i)\} \cup$	x_i a $\neg x_i$ musia byť rôznej farby,

	potrebujeme novú farbu, označme ju s
$\{(x_i, F_j) \mid x_i$ nie je termom $F_j\} \cup$	pre $n \geq 4$ nemôžme na farbenie
$\{(\neg x_i, F_j) \mid \neg x_i$ nie je termom $F_j\}$	F_j použiť farbu s

Všimnime si, že vrcholy $v_1, \ldots, v_n$ tvoria úplný podgraf, preto na ich zafarbenie potrebujeme n farieb. Nech teda v_i je zafarbené i-tou farbou.
Keďže oba vrcholy $x_i, \neg x_i$ sú spojené s každým $v_j, j \neq i$, môžme pri farbení vrcholov $x_i, \neg x_i$ používať i-tu farbu. Navyše je ale každý vrchol x_i spojený s vrcholom $\neg x_i$, preto potrebujeme novú farbu – nazvime ju s. Takže z dvojice $x_i, \neg x_i$ je jeden vrchol zafarbený i-tou farbou a druhý farbou s. Uvedomme si, že ak chceme graf farbiť minimálnym počtom farieb, je popísané farbenie "jednoznačné".
Ostáva farbenie vrcholov/termov. Čo vieme o F_j? F_j je elementárna disjunkcia troch termov. Ak teda $n \geq 4$, pre každé j existuje $i(j)$ také, že F_j je spojený aj s $x_{i(j)}$ aj s $\neg x_{i(j)}$. To vylučuje použitie farby s na farbenie vrchola F_j.

Ukážeme, že **pri farbení vrcholov F_j nepotrebujeme ďalšiu farbu práve vtedy, ak F je splniteľná formula.**
Nech F_j obsahuje literál y, pričom $\neg y$ je zafarbený farbou s. Potom F_j nie je spojený so žiadnym iným vrcholom, ktorý má rovnakú farbu ako y (y má i-tu farbu - rovnako ako v_i, ktoré je spojené s každým ďalším x_r, resp. $\neg x_r$. Preto žiaden z vrcholov, s ktorými je spojený F_j, nemôže mať farbu ako v_i). V takomto prípade môžeme F_j zafarbiť rovnakou farbou ako y. Ak budeme zafarbenie farbou s považovať za priradenie hodnoty 0 a zafarbenie farbou $1, 2, \ldots, n$ za priradenie hodnoty 1, dostávame

F_j môže byť zafarbené bez použitia ďalšej farby

$\Updownarrow$

$\exists$ priradenie $(n+1)$ farieb literálom tak, že $\forall$ elementárna disjunkcia obsahuje literál y taký, že $\neg y$ má priradenú farbu s

$\Updownarrow$

ak možno priradiť hodnoty premenným tak, že $\forall$ elementárna disjunkcia obsahuje $y = 1$ ($\neg y$ má farbu s, $\neg y = 0$)

$\Updownarrow$

F je splniteľná

□

2.3.4 problém Hamiltonovskej kružnice je NP-úplný.

Začnime definíciou problému.

Problém HK

Vstup: $G = (V, E)$
Výstup: $1 \Leftrightarrow G$ obsahuje hamiltonovskú kružnicu
(jednoduchý cyklus prechádzajúci cez každý vrchol grafu)

NP-úplnosť problému HK budeme opäť dokazovať redukciou - tentokrát z problému vrcholového pokrytia. Príslušnosť problému HK do NP nechávame čitateľovi ako cvičenie.

Redukciu problému spravíme v dvoch krokoch – k vstupnému neorientovanému grafu $G = (V, E)$ a prirodzenému číslu k zostrojíme orientovaný graf $G_D = (V_D, E_D)$, o ktorom ukážeme, že G_D má HK práve vtedy, keď G má vrcholové pokrytie mohutnosti k.

konštrukcia G_D

$\mathbf{V_D} = \{a_1, a_2, \ldots, a_k\} \cup \{[v, e, b] \mid v \in V, e \in E, b \in \{0, 1\}, v \text{ incidentný s } e\}$

$\mathbf{E_D}$: Ku každému vrcholu $v_i \in V$ môžme priradiť usporiadaný zoznam hrán $e_{i1}, \ldots, e_{ip(i)}$ s ním incidentných; pre vrchol v_i nech $p(i)$ označuje počet hrán incidentných s v_i. Hranu medzi dvomi vrcholmi v_i, v_j, w.l.o.g. $i < j$, môžme označiť/identifikovať viacerými spôsobmi; je to hrana (i, j), (v_i, v_j) ale tiež $e_{ik(j)}$, resp. $e_{jk(i)}$, pretože je $k(j)$ ta v zozname pre vrhol i, resp. $k(i)$ta v zozname pre vrchol j.

Potom do E_D pridávame hrany

$$\begin{array}{lll} \forall j, i & a_j \to [v_i, e_{i1}, 0], & e_{i1} \text{ je prvá na zozname pre } v_i \\ & [v_i, e_{ip(i)}, 1] \to a_j, & e_{ip(i)} \text{ je posledná na zozname pre } v_i \end{array}$$

Každá hrana $(v_i, v_j) \in E$, $i < j$, spôsobí vznik podgrafu o 4 vrcholoch

$$\begin{array}{ccc} [v_i, (i,j), 0] & \rightleftarrows & [v_j, (i,j), 0] \\ \downarrow & & \downarrow \\ [v_i, (i,j), 1] & \rightleftarrows & [v_j, (i,j), 1] \end{array} \qquad \begin{array}{ccc} \text{A} & \rightleftarrows & \text{B} \\ \downarrow & & \downarrow \\ \text{D} & \rightleftarrows & \text{C} \end{array}$$

pokrytie $\longrightarrow$ HK

Nech $v_1, \ldots, v_k$ tvoria pokrytie v G. Ukážeme, že v grafe G_D existuje HK. Uvažujme najprv kružnicu

$$\begin{array}{lll} K = & a_1, & [v_1, e_{11}, 0], [v_1, e_{11}, 1], \ldots, [v_1, e_{1p(1)}, 0], [v_1, e_{1p(1)}, 1], \\ & a_2, & [v_2, e_{21}, 0], \ldots, , [v_2, e_{2p(2)}, 1], \\ & \ldots & \\ & a_k, & [v_k, e_{k1}, 0], \ldots, [v_k, e_{kp(k)}, 1], a_1 \end{array}$$

pre $i < j$ označme (i,j) hranu medzi vrcholmi v_i, v_j

Ak ostali nejaké nezaradené vrcholy $[v_j, (i,j), 0], [v_j, (i,j), 1]$, tak vrchol v_i je z pokrytia (prečo?). Vtedy modifikujeme K tak, že úsek

$$[v_i, (i,j), 0], [v_i, (i,j), 1]$$

nahradíme

$$[v_i, (i,j), 0], [v_j, (i,j), 0], [v_j, (i,j), 1], [v_i, (i,j), 1]$$

Týmto spôsobom zaradíme všetky doteraz nezaradené vrcholy a na záver bude K tvoriť HK.

HK $\longrightarrow$ pokrytie

Nech v G_D existuje HK K. Vrcholy $a_1, \ldots, a_k$ rozdeľujú K na k úsekov (kvôli jednoduchosti predpokladajme, že $a_1, \ldots, a_k$ je to poradie, v ktorom sa tieto vrcholy vyskytujú na K). Vzhľadom k tomu, že ak vojdeme do vrchola A (obr. vyššie) je prechod štvoricou vrcholov A, B, C, D v HK možný jedine dvomi spôsobmi - $ABCD$ alebo AD, definuje každý úsek medzi a_i a a_{i+1} vlastne jeden vrchol z pokrytia; každému takémuto úseku možno priradiť vrchol $v_t \in V$ taký, že každý vrchol z tohto úseku je tvaru $[v_t, (t,j), 0], [v_t, (t,j), 1]$ alebo $[v_j, (t,j), 0], [v_j, (t,j), 1]$ alebo $[v_t, (j,t), 0], [v_t, (j,t), 1]$ alebo $[v_j, (j,t), 0]$ alebo $[v_j, (j,t), 1]$. Týmto spôsobom HK K definuje v grafe G vrcholové pokrytie mohutnosti nanajvýš k.

□

2.3.5 Problém plnenia batohu je NP-úplný

Posledným problémom, pre ktorý ukážene NP-úplnosť, je problém plnenia batohu. Existuje viacero modifikácií problému, či už rozhodovacie alebo optimalizačné. Začnime teda definíciou problému a jeho verzií.

rozhodovacia verzia 1

subset-sum 0/1-batoh

Vstup: Konečná množina objektov $U = \{u_1, \ldots, u_n\}$ spolu s váhami: $w(u) \in N$ a kapacita batoha $b \in N$
Otázka: Existuje podmnožina $U' \subseteq U$ tak, že $\sum_{u_i \in U'} w(u_i) = b$?

Iná, rovnako ťažká verzia, uvažuje objekty nielen s váhami ale aj s profitmi.

rozhodovacia verzia 2

rozhodovací 0/1-batoh

Vstup: Konečná množina objektov $U = \{u_1, \ldots, u_n\}$ spolu s váhami: $w(u) \in N$, cenami: $c(u) \in N$, kapacitou batoha $b \in N$ a želaným profitom P.
Otázka: Existuje podmnožina $U' \subseteq U$ tak, že $\sum_{u_i \in U'} w(u_i) \leq b$ a $\sum_{u_i \in U'} p(u_i) \geq P$?

optimalizačná verzia

optimalizačný 0/1-batoh

Vstup: Konečná množina objektov $U = \{u_1, \ldots, u_n\}$ spolu s váhami: $w(u) \in N$, cenami: $c(u) \in N$ a kapacitou batoha $b \in N$.
Cieľ: maximalizovať $\sum_{u_i \in U'} p(u_i)$, pričom $\sum_{u_i \in U'} w(u_i) \leq b$

Cvičenie 2.2 *Vysvetlite, ako by ste pomocou algoritmu na riešenie rozhodovacieho 0/1-batohu riešili subset-sum 0/1-batoh*

Cvičenie 2.3 *Vysvetlite, ako by ste pomocou algoritmu na riešenie subset-sum 0/1-batoh riešili rozhodovací 0/1-batoh.*

Cvičenie 2.4 *Vysvetlite, ako by ste pomocou algoritmu na riešenie optimalizačného 0/1-batohu riešili subset-sum 0/1-batoh.*

Keďže sú všetky tri verzie rovnako ťažké, budeme sa zaoberať zložitosťou len jednej z nich.

Veta 2.4 *Problém subset-sum 0/1-batoh je NP-úplný.*

Dôkaz: Dôkaz NP-ťažkosti problému spravíme redukciou z problému 3SAT-splniteľnosti. Príslušnosť problému do triedy NP nechávame ako cvičenie.

Nech $F(x_1, \ldots, x_n) = F_1 \wedge F_2 \wedge \ldots \wedge F_q$, $F_i = x_{i_1}^{\sigma i_1} \vee x_{i_2}^{\sigma_{i_2}} \vee x_{i_3}^{\sigma_{i_3}}$ je formula v tvare KNF. K nej zostrojíme prípad (subset-sum-fill)0/1-knapsack tak, že tento bude mať riešenie práve vtedy, keď F bude splniteľná.

Objekty, ktoré budeme klásť do batohu, budú dvoch typov: objekty typu h budeme používať na priradenie hodnôt jednotlivým premenným a objekty fypu f budú "vyvažovať" váhu jednotlivých klauzúl. Váha batoha bude určená tak, aby si obsahom batoha vynucovala spĺňajúce priradenie.

Formálne:

objekty Pre každú premennú x_i, $1 \leq i \leq n$ pridáme do množiny objektov dva prvky: $h1_i$ a $h0_i$.

Pre každú klauzulu F_j, $1 \leq j \leq q$, pridáme do množiny objektov dva prvky $f1_i$ a $f2_i$.

váhy objektov Váhy objektov budú (decimálne/ternárne) čísla dĺžky presne $n+q$: prvých n cifier popisuje/vynucuje korektné priradenie hodnôt premenným, posledných q cifier súvisí so spniteľnosťou.

$$w(h1_i) = 0^{i-1}10^{n-i}c_{i1}\dots c_{iq}, \text{ kde } c_{ij} = \begin{cases} 1, & x_i \text{ sa vyskytuje v klauzule } F_j \\ 0, & \text{inak} \end{cases}$$

$$w(h0_i) = 0^{i-1}10^{n-i}c_{i1}\dots c_{iq}, \text{ kde } c_{ij} = \begin{cases} 1, & \neg x_i \text{ sa vyskytuje v klauzule } F_j \\ 0, & \text{inak} \end{cases}$$

$$w(f1_i) = w(f2_i) = 0^{n+i-1}10^{q-i}$$

váha batoha $W = 1^n3^q$

Vzťah medzi splniteľnosťou a riešením nášho (subset-sum-fill)0/1-knapsack problému je priamočiary:

Nech $\alpha_1, \dots, \alpha_n$ je spĺňajúce priradenie hodnôt premenným $x_1, \dots, x_n$. Potom do batoha vložíme objekty nasledovne: *3SAT↝batoh*

- z dvojice $h1_i, h0_i$ zoberieme $\begin{cases} h1_i, & \text{ak } \alpha_i = 1 \\ h0_i, & \text{ak } \alpha_i = 0 \end{cases}$
- Nech $\sum_{i-1}^{n}(h1_i + h0_i) = 1^n c_1 \dots c_q$. Uvedomme si, že hodnota c_i vyjadruje počet literálov v klauzule F_i, ktoré majú hodnotu 1.

  Potom z dvojice $f1_i, f2_i$ zoberieme $\begin{cases} f1_i, & \text{ak } c_i = 2 \\ f1_i \text{ aj } f2_i, & \text{ak } c_i = 1 \\ \text{ani jeden z nich}, & \text{ak } c_i = 3 \end{cases}$

Naopak, ak existuje riešenie pre problém batoha, potom toto riešenie prvkami typu h v batohu jednoznačne určuje spĺňajúce priradenie $\alpha_1, \dots, \alpha_n$ premenných $x_1, \dots, x_n$. Ako? *3SAT↝batoh*

□

Príklad 2.1 *Aplikujeme popísanú redukciu pre vstup*

$F = (x_1 \vee \neg x_2 \vee x_4)(x_2 \vee \neg x_3 \vee \neg x_5)(x_3 \vee x_4 \vee x_5)(\neg x_1 \vee x_2 \vee \neg x_5),\ n = 5,\ q = 4$

	x_1	x_2	x_3	x_4	x_5	c_1	c_2	c_3	c_4
$h0_1$	1	0	0	0	0	0	0	0	1
$h1_1$	1	0	0	0	0	0	0	0	0
$h0_2$	0	1	0	0	0	1	0	0	0
$h1_2$	0	1	0	0	0	0	1	0	1
$h0_3$	0	0	1	0	0	0	1	0	1
$h1_3$	0	0	1	0	0	0	0	1	0
$h0_4$	0	0	0	1	0	0	0	0	0
$h1_4$	0	0	0	1	0	1	0	1	0
$h0_5$	0	0	0	0	1	0	1	0	1
$h1_5$	0	0	0	0	1	0	0	1	0
$f1_1$	0	0	0	0	0	1	0	0	0
$f2_1$	0	0	0	0	0	1	0	0	0
$f1_2$	0	0	0	0	0	0	1	0	0
$f2_2$	0	0	0	0	0	0	1	0	0
$f1_3$	0	0	0	0	0	0	0	1	0
$f2_3$	0	0	0	0	0	0	0	1	0
$f1_4$	0	0	0	0	0	0	0	0	1
$f2_4$	0	0	0	0	0	0	0	0	1
W	1	1	1	1	1	3	3	3	3

$F(1,1,0,1,0) = 1$

do batohu preto dáme prvky

- $h1_1, h1_2, h0_3, h1_4, h0_5$
  kvôli spĺňajúcim hodnotám
  x_1*=1,* x_2*=1,* x_3*=0,* x_4*=1,* x_5*=0*
- $f1_1$*, pretože v* $F_1 = (x_1 \vee \neg x_2 \vee x_4)$
  majú 2 literály hodnotu 1
- *ani* $f1_2$ *ani* $f2_2$
  pretože v $F_2 = (x_2 \vee \neg x_3 \vee \neg x_5)$
  majú hodnotu 1 všetky literály
- $f2_3, f1_4$ *pretože ...*

Kapitola 3

Deterministické metódy na riešenie ťažkých problémov

V tejto časti sa budeme venovať deterministickým metódam na riešenie ťažkých problémov. S výnimkou heuristík pôjde o riešenie korektné na celej množine definičného oboru. Jednotiacim aspektom metód z častí 3.1, 3.2 je spoločná snaha o vyjadrenie zložitosti algoritmu vzhľadom na kvalitu/charakter/vlastnosti vstupu, nielen jeho veľkosť. V častiach 3.3.3, 3.4, 3.5 sa venujeme rôznym prístupom k zmenšovaniu prehľadávanej časti stavového priestoru; zmenšenie nevieme dokázať, máme však racionálne dôvody sa nazdávať, že k nemu dôjde. Časť 3.3 sa venuje znižovaniu najhoršieho prípadu exponenciálnych algoritmov a napokon v časti 3.6 sa venujeme aplikácii lineárneho programovania pri riešení rôznych problémov.

3.1 Algoritmy pseudopolynomiálnej časovej zložitosti

Začnime problémom plnenia batoha(knapsack problem). Vieme, že jeho 0/1 verzia je NP-ťažká, čo za predpokladu $\mathsf{P} \neq \mathsf{NP}$ znamená, že preň neexistuje efektívny polynomiálny algoritmus. Ukážeme, že využitie metódy dynamického programovania vedie k algoritmu, ktorého zložitosť podstatne závisí nielen od veľkosti vstupu, ale aj od konkrétnych hodnôt na vstupe. Dostaneme sa tak k pojmu pseudopolynomiálneho algoritmu. V závere zadefinujeme pojem silno NP-ťažkého problému, vysvetlíme jeho význam a ukážeme/vysvetlíme, že pre silno NP-ťažké problémy pseudopolynomiálne algoritmy neexistujú.

Plnenie batohu

Vstup: $\{u_1, \ldots, u_n \mid u_i = (w_i, c_i)\},\ b,\ n \in \mathbb{N}/\{0\}$
b je kapacita batoha
w_i sú váhy
c_i sú profity.
Cieľ: $\max \sum_i x_i c_i$ pri zachovaní $\sum_i x_i w_i \leq b$ a $x_i \in \{0, 1\}$

Triviálne riešenie exponenciálnej zložitosti je založené na *metóde hrubej sily* – vygenerujeme všetky potenciálne podmnožiny objektov a vyberieme tú, ktorá prináša maximálny zisk.

Vylepšenie dosiahneme, ak sa nám podarí zredukovať počet tých množín, ktoré budeme generovať. Nebudeme generovať všetky podmnožiny, ale len tie, ktoré sú

minimálnej váhy pre profit, ktorý prinášajú. Ako? Aplikujeme metódu dynamického programovania, pri ktorej budeme postupne predlžovať vstup (počet prvkov, z ktorých vyberáme do batohu) a budeme počítať, aké rôzne profity pri takomto vstupe môžme dosiahnuť. Pre každý potenciálny dosiahnuteľný profit si budeme pamätať tú konkrétnu podmnožinu, ktorá ho dosahuje s minimálnou váhou.

Uvedomme si, že zatiaľ čo všetkých možných riešení (z hľadiska obsahu batohu) je 2^n, z hľadiska možného zisku je to $n \cdot \max\{c_1, \ldots, c_n\}$. Označme preto

$\mathbf{C} = \sum_{i=1}^{n} c_i$

$\mathbf{S(i,c)}, i \in \{1, \ldots, n\},\ c \in \{0, \ldots, nC\}$ podmnožinu objektov $\{1, \ldots, i\}$, ktorých celkový zisk je c pri *minimálnej váhe*

W(i,c) súčet váh objektov z $S(i, c)$; kladieme $W(i, c) = \infty$ v prípade, že množina $S(i, c)$ neexistuje

Nech $S(i + 1, c)$ je tá podmnožina $\{u_1, \ldots, u_{i+1}\}$, ktorá je minimálnej váhy a dosahuje profit c. V tejto množine sa prvok u_{i+1} alebo nachádza alebo nenachádza, preto pre $W(i + 1, c)$ platí:

$$W(i+1,c) = \begin{cases} \min\{W(i,c), w_{i+1} + W(i, c - c_{i+1})\} & c_{i+1} \leq c \\ W(i,c) & \text{inak} \end{cases}$$

Uvedený vzťah je základom pre použitie *dynamického programovania*. Optimálne riešenie má cenu B, kde $B = \max\{c \mid W(n, c) \leq b\}$.

Algoritmus 1 DPKP - Knapsack

```
1: W(1,0) ← 0
2: for c:=1 to C do
3:     if c=c_1 then W(1,c) ← v_1; S(1,c) ← {1}
4:     else W(1,c) ← ∞; S(1,c) ← ∅
5: for i=1 to n-1 do
6:     W(i+1,0) ← 0
7:     for c=1 to C do
8:         if c_{i+1} ≤ c then W(i+1,c) ← min{W(i,c), w_{i+1} + W(i,c − c_{i+1})}
9:         else W(i+1,c) ← W(i,c)
10:        if W(i+1,c) = W(i,c) then S(i+1,c) ← S(i,c)
11:        else S(i+1,c) ← S(i,c) ∪ {i+1}
12: B ← max{c | W(n,c) ≤ b}
13: S ← S(n,c), pre ktorú W(n,c) = B
14: return (B,S)
```

Zložitosť algoritmu je $O(n \cdot C)$. Ak označíme $MaxInt(x)$ maximálnu hodnotu zo vstupu x, tak $n \leq |x|$ a $C \leq n \cdot MaxInt(x) \leq |x| MaxInt$. Pre zložitosť $Time_{DPKP}()$ algoritmu DPKP teda platí:

$$Time_{DPKP}(x) = O(|x|^2 \cdot MaxInt(x))$$

□

Zložitosť algoritmu sme vyjadrili ako funkciu veľkosti vstupu a hodnôt, ktoré vstupné parametre môžu nadobúdať; dostali sme sa tak k pojmu *pseudopolynomiálneho* algoritmu.

Ak by sme *veľkosť hodnoty* vstupných parametrov uvažovali ako *premennú*, vzhľadom na ktorú vyjadrujeme zložitosť algoritmu, dostaneme (často, nielen teraz) algoritmus polynomiálnej zložitosti. Keďže vo všeobecnosti hodnoty premenných môžu byť až exponenciálne vzhľadom na veľkosť vstupu, dostávame tak algoritmy v najhoršom prípade exponenciálnej zložitosti. Napriek tomu majú algoritmy, ktoré sú polynomiálne nielen vzh2adom na ve2kos5 vstupu ale aj vzhľadom na veľkosť hodnoty vstupných premenných veľkých význam – sú totiž polynomiálne pre veľkú množinu vstupov.

Pri formálnejšom vysvetlení pojmu pseudopolynomiálneho algoritmu sa sústredíme na problémy, ktorých vstupy možno chápať ako celočíselné. Budeme ich kódovať binárne, oddeľovačom bude #.

Nech $x = x_1\#x_2\#\ldots\#x_n$, $x_i \in \{0,1\}^+$ je reťazec, $y \in \{0,1\}^*$ binárny reťazec. Potom

Num(**y**) označuje číslo s binárnym zápisom y
Int(**x**) $= (Num(x_1), \ldots, Num(x_n))$
MaxInt(**x**) $= \max\{Num(x_i) \mid i = 1, 2, \ldots, n\}$

Majme teda problém, ktorý je definovaný na veľkej množine vstupov. Stretli sme sa s tým, že pre rôzne podmnožiny množiny vstupov sa z hľadiska zložitosti ten-ktorý algoritmus správa rôzne, niekedy dokonca podstatne rôzne[1]. Niekedy je dokonca problém na rôznych množinách vstupov rôzne obtiažny, napr. problém plnenia batoha je v množine racionálnych čísel greedy metódou riešiteľný v polynomiálnom čase, zatiaľ čo jeho riešenie v množine $\{0,1\}$ je NP-ťažké. Nás teraz bude zaujímať obmedzenie množiny vstupov podľa ich hodnoty. Ukážeme, že pre niektoré problémy je zložitosť problému spôsobená hodnotami vstupov, zatiaľ čo pre iné problémy hodnoty vstupných premenných zložitosť podstatne neovplyvňujú. Je zrejmé, že problémy prvého typu sú pre nás priaznivejšie, lebo vytvárajú možnosť existencie efektívneho algoritmu pre dostatočne veľkú množinu vstupov.

Definícia 3.1 *Hovoríme, že algoritmus A je* pseudopolynomiálnej zložitosti*, ak pre jeho zložitosť $T_A()$ platí* *pseudopolynomiálny algoritmus*

$$T_A(x) = O(p(|x|, MaxInt(x)))$$

Inými slovami povedané je algoritmus polynomiálny pre celočíselné vstupy, ktoré sú zadávané unárne. Z definície je zrejmé, že pseudopolynomiálny algoritmus je efektívny, keď maximálna hodnota na vstupe je rozumná, ohraničená rozumnou funkciou. Toto ohraničenie vstupov formálne zachytáva pojem zúženia problému.

Definícia 3.2 *Nech U je problém s celočíselnými vstupmi, h neklesajúca funkcia z $\mathbb{N}$ do $\mathbb{N}$. Potom $h-$*zúžením problému *U je problém, ktorý vznikne z U tak, že množinu vstupov zredukujeme na tie, pre ktoré $MaxInt(x) \leq h(|x|)$.* *h-zúženie U*

Význam tejto definície je sformulovaný v nasledujúcej vete.

Veta 3.1 *Nech U je problém s celočíselnými vstupmi, A pseudopolynomiálny algoritmus, ktorý ho rieši. Potom pre každý* polynóm *h existuje polynomiálny algoritmus na riešenie $h-$zúženia U.*

Dôkaz: Keďže A je pseudopolynomiálny algoritmus, existuje polynóm p dvoch premenných $|x|, MaxInt(x)$ taký, že $Time_A(x) = O(p(|x|, MaxInt(x)))$ pre každý vstup x do U. Keďže $MaxInt(x) \in O(|x|^c)$ pre vstupy z $h-$zúženia U, teda s $h(n) = O(n^c)$, je A polynomiálny algoritmus pre h-zúženie U.

□

Limity aplikovateľnosti

Pseudopolynomiálne algoritmy nie sú všeliekom na riešenie ťažkých problémov. Rozumné riešenie môžu poskytovať pre problémy s celočíselnými vstupmi, ktorých zložitosť podstatne závisí od *hodnôt* vstupov. Jednou z tried problémov, pre ktoré pseudopolynomiálne algoritmy rozumné riešenie neposkytujú, sú tzv. silno NP-ťažké problémy.

silno NP-ťažký problém

Definícia 3.3 *Hovoríme, že problém U je* silno NP-ťažký*, ak existuje polynóm p taký, že $p-$zúženie U je NP-ťažký problém.*

Silno NP-ťažké problémy patria v triede ťažkých problémov k tým ťažším. Ich obtiažnosť je v samotnom probléme, nie vo veľkých hodnotách čísel, s ktorými sa pracuje.

Veta 3.2 *Nech $P \neq NP$, U je silno NP-ťažký problém s celočíselnými vstupmi. Potom* neexistuje *algoritmus pseudopolynomiálnej zložitosti riešiaci U.*

Dôkaz: Tvrdenie triviálne vyplýva z Vety 3.1. Pri dôkaze postupujeme sporom. Nech A je pseudopolynomiálny algoritmus, ktorý rieši silno NP-ťažký problém U v zložitosti $O(p(|x|, MaxInt(x)))$ pre nejaký polynóm p. Nech h je polynóm. Potom A rieši aj h-zúženie U, pričom pre zložitosť $T_h()$ platí

$$T_h(x) = O(p(|x|, MaxInt(x))) = O(p(|x|, h(|x|)))$$

Zložením polynómov dostaneme polynóm, čo znamená, že h-zúženie $U \in P$ a to je spor s predpokladom, že U je silno NP-ťažký problém. □

Ak teda chceme o nejakom celočíselnom probléme U ukázať, že je veľmi ťažký, stačí vyargumentovať, že preň *neexistuje* pseudopolynomiálny algoritmus. Na základe Vety 3.2 teda stačí ukázať, že pre nejaký polynóm h je $h-$zúženie U NP-ťažký problém. Dôkaz robíme tak, že naň zredukujeme nejaký NPU alebo NP-ťažký problém U'. Táto redukcia potom implikuje, že existencia pseudopolynomiálneho algoritmu pre pôvodný problém U by znamenala existenciu polynomiálneho algoritmu pre tento NP-úplný/ NP-ťažký problém U'.

Príkladom je problém obchodného cestujúceho(TSP)[2]. Označme $TSP(K_n, c)$ problém obchodného cestujúceho, ktorého vstupným grafom je úplný graf K_n na n vrcholoch, a ktorého hrany sú ohodotené cenovou funkciou $c : E \longrightarrow \mathbb{N}$.

Veta 3.3 *TSP je silno NP-ťažký.*

Dôkaz: Prevedieme redukciu problému HK na TSP. Ku grafu $G = (V, E)$ spravíme ohodnotený graf tak, že

$$c(e) = \begin{cases} 1, & e \in E \\ 2, & e \notin E \end{cases}$$

Ľahko vidno, že graf G obsahuje HK práve vtedy keď pre hodnotu optimálneho riešenia $OPT_{TSP}(K_n, c)$ platí $OPT_{TSP}(K_n, c) = n$.

□

[1] Rozdiel medzi zložitosťou najlepšieho a najhoršieho prípadu,...

[2] Pre daný orientovaný graf s hranami ohodnotenými nezápornými číslami chceme nájsť takú HK, ktorej cena°(teda súčet hrán)°je minimálna.

3.2 Parametrizovaná zložitosť

Hlavnou ideou parametrizovanej zložitosti je precíznejšia analýza vstupu a na jej základe vyjadrenie zložitosti ako funkcie viacerých parametrov. Pomocou týchto parametrov sa snažíme vymedziť skupinu vstupov, ktoré sú daným algoritmom zvládnuteľné. Rozdelíme teda vstupy do skupín podľa hodnoty nejakého parametra tak, že algoritmus bude polynomiálny od veľkosti vstupu, ale možno nie od tohto zvoleného parametra. Napr. pre vstup s parametrami (n, k) dostaneme zložitosť $2^k n$. Pre malé hodnoty k je $2^k n$ akceptovateľná zložitosť.

parametrizácia

Definícia 3.4 *Nech U je výpočtový problém, I množina všetkých jeho (prípustných) vstupov.* Parametrizáciou U *nazývame funkciu $Par : I \longrightarrow \mathbb{N}$ takú, že*

1. *Par je vypočitateľná v polynomiálnom čase*
2. *pre nekonečne veľa $k \in \mathbb{N}$ je $Set_U(k) = \{x \in I \mid Par(x) = k\}$* nekonečná.

Uvedomme si, že hodnota parametra "nezávisí" od veľkosti vstupu v tom zmysle, že rovnakú hodnotu tohto parametra môžu mať vstupy ľubovoľných veľkostí – napr. stupeň grafu, maximálny profit pri probléme plnenia batoha, ...

Parametrizáciu Par využijeme pri vyjadrení zložtosti algoritmu:

Par-parametrizovaný polynomiálny algoritmus

Definícia 3.5 *Hovoríme, že A je Par* -parametrizovaný polynomiálny algoritmus *pre U, ak*

1. *A rieši U*
2. *existuje polynóm p a funkcia $f : \mathbb{N} \longrightarrow \mathbb{N}$ taká, že $\forall x \in I : TimeA(x) \leq f(Par(x))p(|x|)$*

Pri konštrukcii algoritmov sa snažíme o nízky stupeň polynómu p a pripúšťame superpolynomialitu funkcie f. Vidíme, že p hovorí o zložitosti algoritmu pri fixovaných parametroch, f určuje, pre aké hodnoty parametra je problém zvládnuteľný(tractable).

vrcholové pokrytie

Príklad 3.1 *Uvažujme problém vrcholové pokrytia (VC[3]), o ktorom vieme, že je to NP-úplný problém. Ukážeme dva rôzne prístupy k riešeniu tohto problému.*

prístup I

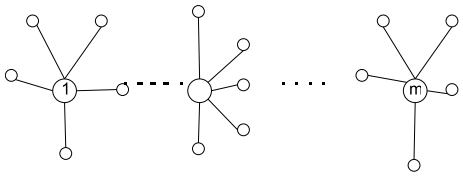
Nech vstupom je graf $G = (V, E)$ a hodnota $k \in \mathbb{N}$. Za parameter (parametrizáciu) vezmime mohutnosť vrcholového pokrytia k. Algoritmus je založený na dvoch jednoduchých pozorovaniach/faktoch.

Fakt 3.4 *Ak má graf $G = (V, E)$ vrcholové pokrytie S mohutnosti k, tak S obsahuje všetky vrcholy stupňa aspoň $k + 1$.*

Fakt 3.5 *Ak G má vrcholové pokrytie mohutnosti m a stupeň grafu je zhora ohraničený k, tak G má nanajvýš $m(k + 1)$ vrcholov.*

Využijeme tieto fakty v nasledujúcom algoritme – najprv do S dáme všetky vrcholy, ktoré majú príslušne veľký stupeň, potom "nahrubo" (backtrackom) doprezeráme zvyšok. Veľkosť zvyšku vieme ohraničiť na základe faktu 3.5
Pozrime sa na zložitosť tohto algoritmu.
1 $O(n)$
4 $O(1)$

[3]VC tiež označuje množinu takých vstupov (G, k), pre ktoré G obsahuje vrcholovém pokrytie mohutnosti nanajvýš k



Obrázok 3.1: ilustrácia faktu 2.5.

Algoritmus 2 VC-I

//vstupom je graf G a prirodzené číslo k

1: $S \leftarrow \{v \in V, k \leq deg(v)\}$
2: **if** $|S| > k$ **then** reject
3: **else** $m \leftarrow k - |S|; G' \leftarrow G_{V/S}$
4: **if** $|V - S| > m(k+1)$ **then** reject
5: backtrackom na G' hľadaj VC mohutnosti nanajvýš m (uvedom si, že stupeň grafu G' je zhora ohraničený k)
6: **if** existuje **then** accept
7: **else** reject

5 Hľadáme vrcholové pokrytie mohutnosti m v grafe, ktorý má maximálne $m(k+1) \leq k(k+1)$ vrcholov (fakt 3.5). Počet operácií je O(maximálny počet hrán × počet možností pre to pokrytie)

$$O\left(k \cdot m(k+1)\binom{m(k+1)}{m}\right) \subseteq O\left(k^3\binom{k(k+1)}{k}\right) \subseteq O(k^{2k})$$

Zhrnutím dostávame

Veta 3.6 *Algoritmus VC-I je Par-parametrizovaný polynomiálny algoritmus pre problém vrcholového pokrytia.*

prístup II

Iným prístupom k riešeniu problému vrcholového pokrytia je využitie metódy rozdeľuj a panuj v kombinácii s nasledujúcim faktom.

Fakt 3.7 *Nech $G = (V, E)$ je graf, $e = (u, v) \in E$. Potom každé vrcholové pokrytie obsahuje aspoň jeden z vrcholov u, v.*

Idea je veľmi jednoduchá - nech $G = (V, E), k \in \mathbb{N}$ je vstup, $(v_1, v_2) \in E$ ľubovoľná hrana. Označme $G(i)$ ten graf, ktorý vznikne z grafu G vynechaním vrchola v_i a s ním incidentných hrán. Zrejme:

$$(G, k) \in VC \iff [(G(1), k-1) \in VC \vee (G(2), k-1)VC]$$

Takže rekurzívne dostávame zložitosť $O(2^k n)$.

3.3 Znižovanie zložitosti najhoršieho prípadu(aj exponenciálne algoritmy)

Namiesto "naivného" úplného prehľadávania používame niečo rozumnejšie, čo nám zabezpečí *zníženie zložitosti aj v najhoršom prípade*. Snažíme sa o znižovanie exponentu, resp. základu. Vplyv týchto zmien vidíme v tabuľke.

	n=10	n=50	n=100
2^n	1024	16 cifier	91 cifier
$2^{n/2}$	32	$33{\cdot}10^6$	46 cifier
$(1.2)^n$	7	9100	24 cifier

3.3.1 Rozdeľuj a panuj a 3SAT

Vráťme sa opäť k problému[4] 3SAT, keď namiesto triviálneho prehľadávania hrubou silou $O(2^n)$ dostaneme algoritmus zložitosti $O(|F|1,84^n)$. Vylepšenie spočíva v zakomponovaní informácie o znalosti hodnoty niektorej premennej– v niektorých prípadoch nám informácia o hodnote nejakej premennej umožní zjednodušenie formuly F.

3SAT

$\boldsymbol{F(x_1 = a_1, ..., x_k = a_k)}$ označuje formulu, ktorá vznikla z formuly F dosadením hodnoty a_i za premennú $x_i, 1 \leq i \leq k$. V našom prípade je F formula v tvare 3KNF. Ako v tomto prípade môžme využiť/realizovať znalosť hodnoty premennej?

$\boldsymbol{F(x = 1)}$ vznikne z F aplikovaním nasledujúcich pravidiel

- odstránenie všetkých klauzúl, ktoré obsahujú x
- ak v klauzule je okrem $\neg x$ aj iný literál, tak $\neg x$ z klauzuly odstránime
- ak klauzula obsahuje len $\neg x$, tak je formula nesplniteľná

$\boldsymbol{F(x = 0)}$ vznikne z F aplikovaním nasledujúcich pravidiel

- odstránenie všetkých klauzúl, ktoré obsahujú $\neg x$
- ak v klauzule je okrem x aj iný literál, tak x z klauzuly odstránime
- ak klauzula obsahuje len x, tak je formula nesplniteľná

Vidíme, že uvedené "dosadenie" realizujeme v lineárnom čase.

Nech $\mathcal{F}$ je 3KNF formula, $(x \vee y \vee z)$ klauza v nej. Vždy, keď je splnená formula F, je splnená každá jej klauzula, teda aj $(x \vee y \vee z)$. To ale znamená, že $\mathcal{F}$ je splniteľná práve vtedy ak je splniteľná aspoň jedna z formúl $\mathcal{F}(x{=}1)$, $\mathcal{F}(x{=}0, y{=}1)$ alebo $\mathcal{F}(x{=}0, y{=}0, z{=}1)$. Všimni e si, že realizácia "dosadení" $\mathcal{F}(x{=}1)$, $\mathcal{F}(x{=}0, y{=}1)$, $\mathcal{F}(x{=}0, y{=}0, z{=}1)$ vyššie uvedeným spôsobom potom znižuje počet klauzúl aj počet premenných vo formule. Označme

$\boldsymbol{3KNF(n, r)}$ = $\{F \mid F$ je formula v tvare 3KNF nad n premennými, ktorá obsahuje nanajvýš r klauzúl$\}$.

$$\mathcal{F} \in 3KNF(n,r) \implies \begin{cases} \mathcal{F}(x=1) & \in & 3KNF(n-1, r-1) \\ \mathcal{F}(x=0, y=1) & \in & 3KNF(n-2, r-1) \\ \mathcal{F}(x=0, y=0, z=1) & \in & 3KNF(n-3, r-1) \end{cases}$$

Všetky tieto úvahy vedú k použitiu metódy rozdeľuj-a-panuj, ktorá je realizovaná v Algoritme 3 –DC-3SAT(F).

- Korektnosť algoritmu vyplýva z predchádzajúcich úvah.
- Kvôli analýze času označme $T(n, r)$ čas algoritmu pri vstupnej formule z $3KNF(n, r)$.

[4] 3SAT – pre vstupnú boolovskú formulu $F(x = x_1, \ldots, x_n)$ v tvare 3KNF(konjunkcia elementárnych disjunkcií dĺžky maximálne 3) máme určiť, či existuje také priradenie $\alpha_1, \ldots, \alpha_n \in \{0, 1\}$ premenným $x_1, \ldots, x_n$, že $F(\alpha_1, \ldots, \alpha_n) = 1$

Algoritmus 3 DC-3SAT(F)

1: ak $F \in 3KNF(3,k)$ alebo $F \in 3KNF(m,2)$, tak dosadením všetkých hodnôt zisti odpoveď
2: nech H je jedna z najkratších klauzúl v F
3: ak $H = (x)$ tak return DC-3SAT(F(x=1))
4: ak $H = (x \vee y)$ tak return DC-3SAT(F(x=1))$\vee$ DC-3SAT(F(x=0, y=1))
5: ak $H = (x \vee y \vee z)$ tak return DC-3SAT(F(x=1))$\vee$ DC-3SAT(F(x=0, y=1))$\vee$ DC-3SAT(F(x=0, y=0,z=1))

Triviálne platí, že $|F|/3 \leq r \leq |F|$. Preto

$$T(3,r) \leq 24r \quad T(2,r) \leq 12r \quad T(1,r) \leq 3r$$

Navyše, dosadenie $F(l = a)$ vieme realizovať v čase $\leq 9r$. Zložitosť algoritmu teda môžme vyjadriť vzťahom

$$T(n,r) \leq \begin{cases} 24r & \text{ak } n \leq 3 \text{ alebo } r \leq 2 \\ 54r + T(n-1,r-1) + T(n-2,r-1) + T(n-3,r-1) \end{cases}$$

Indukciou overíme, že $T(n,r) \leq 27r \cdot (1.84^n - 1)$. Triviálne prípady preskočíme;

$$\begin{aligned} T(n,r) \leq \quad & 54r + 27(r-1)(1.84^{n-1} - 1) + 27(r-1)(1.84^{n-2} - 1) \\ & +27(r-1)(1.84^{n-3} - 1) \\ & \leq 54r + 27r(1.84^{n-1} + 1.84^{n-2} + 1.84^{n-3} - 3) \\ & \leq 27r \cdot 1.84^n \left(\tfrac{1}{1.84} + \tfrac{1}{1.84^2} + \tfrac{1}{1.84^3}\right) + \underbrace{54r - 3 \cdot 27r}_{27r} \\ & = 27r \left[\left(\tfrac{1.84^2+1.84+1}{1.84^3}\right) \cdot 1.84^n - 1\right] \\ & \leq 27r(1.84^n - 1) \end{aligned}$$

Ukázali sme, že pre zložitosť algoritmu DC-3SAT(F) so vstupom $F \in 3KNF(n,r)$ platí $O(r \times 1,84^n)$.

□

3.3.2 Rozumné prehľadávanie s orezávaním

Metóda LC-B&B bola snahou o minimalizáciu prehľadávanej časti stavového priestoru, keď sme do úvahy brali hodnotu ceny priradenej príslušnému vrcholu v generovanom strome. Čím bližšie je hodnota ceny vrchola k hodnote optimálneho riešenia v príslušnom podstrome, tým väčšiu šancu, *nie istotu*, máme, že nás bude dobre navigovať v stavovom priestore a veľkú časť stavového priestoru neprezrieme. Sústreďme sa preto na orezávanie stromu prehľadávania s cieľom minimalizovať čas najhoršieho prípadu úplného prezretia stromu v najhoršom prípade.

3SAT

Algoritmus DC-3SAT(F)sa možno pozerať ako na orezávanie, keď pri rozvíjaní vrcholu výberom klauzuly $(x \vee y \vee z)$ vynechávame generovanie vetvy pre možnosť $x = y = z = 0$. O zložitosti platí

$$T(n) \leq T(n-1) + T(n-2) + T(n-3) + O(n+m)$$

$T(n)$ je polynóm od α^n, kde α je najväčší reálny koreň polynómu $\alpha^3 = \alpha^2 + \alpha + 1$. Keďže $\alpha \approx 1.8393$, riešením je $T(n) = O^*(1.8393^n)$. Notácia O^* znamená až na multiplikatívny polynóm, presnejšie:

$$O(T(m(x)) \cdot p(|x|)) = O^*(T(m(x)))$$

Zakomponovaním voľby klauzuly, ktorá nevyžaduje branching, resp. takej, ktorá vedie na formuly s $k-1$ literálmi dostávame zložitosť[5] $O*(1.6181^n)$ pre $k=3$

Ďalšie vylepšenia súvisia s lepšou analýzou: analýza počtu 2-klauzúl viedla k $O^*(1.5783^n)$, presnejšia analýza k $O^*(1.4963^n)$.

Príklad 3.2 *Majme graf $G=(V,E)$, $|V|=n$. Podmnožina $S \subseteq V$ tvorí* nezávislú množinu vrcholov*, ak neexistujú hrany medzi vrcholmi z S ($\forall u,v \in S$ $(u,v) \notin V$). Úlohou je pre vstupný graf G a $k \in S$ zistiť, či v G existuje nezávislá množina vrcholov veľkosti k.*
Označme IS množinu vstupov (G,k) takých, že v G existuje nezávislá množina vrcholov mohutnosti k.

nezávislá množina

Pri riešení hrubou silou prezrieme všetky k-prvkové podmnožiny množiny vrcholov a overíme, či tá-ktorá množina tvorí nezávislú množinu vrcholov. (Aká je zložitosť tohto prístupu?)

hrubá sila

Orezávanie vlastne znamená, že v priebehu výpočtu identifikujeme vetvy/potenciálne riešenia, ktoré prípustným riešením nie sú. Použijeme teda dynamický strom prehľadávania, pričom vetvenie budeme robiť podľa maximálneho stupňa vrchola.

strom prehľadávania

Triviálnym prípadom takého prístupu je zrejme graf, v ktorom je maximálny stupeň nanajvýš dva. Takýto graf pozostáva len z cyklov a ciest (a izolovaných vrcholov). Riešenie v tomto prípade je evidentne polynomiálnej zložitosti.(Ako by ste to robili?)

triviálny prípad

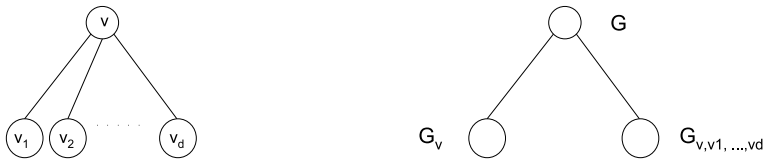
Majme teda graf, v ktorom existuje aspoň jeden vrchol stupňa aspoň 3. Nech v je vrchol maximálneho stupňa, $v_1,\ldots,v_d, d \geq 3$ jeho susedia. Ak S je nezávislá množina vrcholov, potom sú z pohľadu vrchola v dve možnosti:

vetvenie

$$\text{vrchol } v \text{ alebo patrí alebo nepatrí do množiny } S$$

Ak v nepatrí do S, tak môžme z grafu G odstrániť vrchol v a všetky s ním susediace hrany a množina S sa nezmení. Nazvime takto modifikovaný graf G_v.

$v \notin S$



Ak v patrí do nezávislej množiny, zaradíme ho tam, a z grafu odstránime nielen vrchol v a s ním susediace hrany, ale aj vrcholy $v_1,\ldots,v_d$ a s nimi susediace hrany. Takto modifikovaný graf označíme $G_{v,v1,\ldots,vd}$. Je zrejmé, že

$v \in S$

$$(G,k) \in \mathsf{IS} \Leftrightarrow (G_v,k) \in \mathsf{IS} \vee (G_{v,v1,\ldots,vd}, k-1) \in \mathsf{IS}$$

Rekurzívny algoritmus vedie k zložitosti (prečo?)

$$T(n) \leq T(n-1) + T(n-4) + O(n+m)$$

Riešením tejto rekurentnej rovnice je $T(n) = O^*(\gamma^n)$, kde $\gamma \approx 1.3803$ je maximálny reálny koreň $\gamma^4 = \gamma^3 + 1$.

$O^*(1.3803^n)$

Príklad 3.3 *Majme graf $G = (V, E)$. Lineárnym usporiadaním vrcholov rozumieme ich očíslovanie číslami z množiny $\{1, \ldots, n\}$. Nech f je bijekcia, ktorá realizuje toto usporiadanie $f : V \to \{1, \ldots, n\}$. Ak vnímame očíslovanie ako umiestnenie na čiaru, hrany pôvodného grafu dostávajú prirodzenú dĺžku, tzv.* stretch*; dĺžku najdlhšej z nich voláme* bandwidth *bandwidth*

stretch hrany $[u, v] \in E$ je vzdialenosť $|f(u) - f(v)|$

bandwidth grafu je $\max_{(u,v)\in E} |f(u) - f(v)|$

Hľadáme také usporiadanie f, ktoré minimalizuje bandwidth.

Úloha súvisí s vnáraním jednej štruktúry do druhej. Ak bol sieťový algoritmus A dizajnovaný pre topológiu G_1, ale reálna sieť je topológie G_2, potom využitie A pre topológiu G_2 je spojené s vnorením G_1 do G_2: vrcholy grafu G_1 umiestnime do vrcholov grafu G_2 a komunikáciu po hrane (u, v) v G_1 nahradíme/realizujeme komunikáciou po ceste, ktorá spája obrazy vrcholov u, v. Dĺžka tejto ceny spôsobuje spomalenie pôvodného algoritmu, preto sa snažíme o také vnorenie G_1 do G_2, ktoré minimalizuje dĺžku cesty v G_2 odpovedajúcu hrane v G_1.

Riešenie hrubou silou vedie k zložitosti $O^*(n!)$, my sa budeme venovať riešeniu Feige& Kilian zložitosti $O^*(20^n)$.

Šetrenie docielime postupným budovaním dynamického prehľadávacieho stromu, ktorým bude v dvoch etapách generovať potenciálne prípustné riešenia; overovanie vlastností vznikajúceho riešenia počas tvorby riešenia umožňuje orezávanie.

n, b sú mocniny dvojky

Namiesto toho, aby sme vygenerovali všetky lineárne usporiadania a našli medzi nimi to, ktoré má minimálnu hodnotu bandwith, budeme pre $b := 1, \ldots, n$ postupne overovať, či konkrétna hodnota b je bandwidth. Overenie spočíva v hľadaní/generovaní lineárneho usporiadania s aktuálne testovanou hodnotou bandwith b. Prvé lineárne usporiadanie, ktoré korektne dogenerujeme, je to,ktoré hľadáme. (Prečo?) Kvôli jednoduchosti predpokladáme, že b, n sú mocniny dvojky.

Algoritmus pracuje v dvoch etapách:

1. vytvoríme množinu čiastočných riešení, v ktorých je každý vrchol "umiestnený" do jedného z intervalov $I_i = \{(i-1)b/2 + 1, \ldots, ib\}, 1 \leq i \leq \frac{2n}{b}$
2. generujeme všetky lineárne usporiadania s aktuálnym bandwidth, ktoré z daného čiastočného riešenia môžu vzniknúť usporiadaním prvkov v rámci jedného intervalu

vytváranie čiastočných riešení

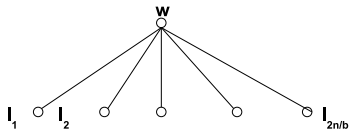
Cieľom prvej etapy je umiestniť vrcholy grafu do $\frac{2n}{b}$ intervalov veľkosti $b/2$ tak, aby sme nevytvárali/eliminovali hrany dlhšie ako b. Predstavme si, že intervaly (dĺžky b/2) sú poukladané za seba $I_1, \ldots, I_{2n/b}$, pričom vrcholy u, v sú umiestené tak, že $u \in I_{i(u)}, v \in I_{i(v)}$. V korektnom usporiadaní musí platiť:

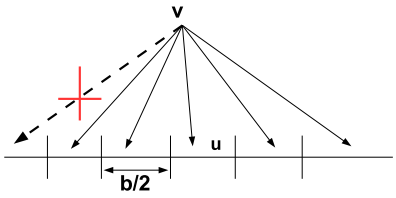
$$|i(u) - i(v)| \cdot \frac{b}{2} < |f(u) - f(v)| < \frac{b}{2} \cdot |i(u) - i(v) + 1|$$

Ak sa teda chceme vyhýbať hranám, ktoré sú *určite* dlhšie ako b, vrcholy, spojené v grafe hranou, môžu ležať len v takých intervaloch, medzi ktorými je nanajvýš jeden ďalší interval. Pri generovaní potenciálne korektných usporiadaní vychádzame z nasledujúcich pozorovaní/princípov:

[5] $O^*(\beta^n)$, kde β je maximálny reálny koreň $\beta = 2 - \frac{1}{\beta^{k-1}}$

1. prvý vrchol $w \in V$ môžme umiestniť do každého z $2n/b$ intervalov; vznikne tým $2n/b$ vrcholov v strome podproblémov
2. každého suseda u už umiestneného vrchola v môžme umiestniť do nanajvýš 5 možných intervalov $I_{i(v)-2}, I_{i(v)-1}, I_{i(v)}, I_{i(v)+1}, I_{i(v)+2}$. Ak je interval, do ktorého chceme umiestniť u, už plný, vrchol doň nekladieme; ak ho nemáme kam umiestniť, lebo všetkých 5 povolených intervalov je plných, túto vetvu výpočtu ukončíme s neúspechom
3. umiestňovali sme $n-1$ vrcholov/susedov, skončíme preto s nanajvýš $\frac{2n}{b} \cdot 5^{n-1}$ vrcholmi v strome podproblémov, ktoré odpovedajú "rozhádzaniam" vrcholov pôvodného grafu do intervalov/čiastočným riešeniam.

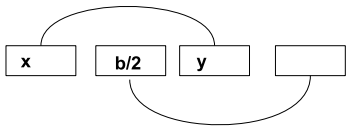




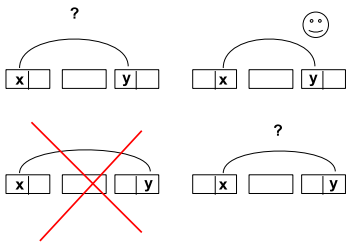
overovanie

Snažíme sa o usporiadanie vrcholov v boxoch tak, aby sme získali lineárne usporiadanie s bandwidth b.

- Overíme, že nexistujú hrany určite dlhšie ako b (ako? prečo?)
- Odstránime hrany vrámci jednotlivých boxov (prečo?)
- Odstránime hrany medzi susednými boxami (prečo?)
- Ostali hrany medzi párnymi, resp. nepárnymi intervalmi, čo vedie k riešeniu dvoch podproblémov - hľadanie usporiadania pre vrcholy v párnych, resp. nepárnych boxoch.



- V rámci podproblému môže byť vrchol uložený do ľavej aj pravej polovice toho boxu, v ktorom sa momentálne nachádza. Vytvorili sme teda 2^n nových čiastočných riešení so zmenšenou veľkosťou boxu. V nich opäť odstránime hrany, ktoré môžme a rekurzívne pokračujeme ďalej.



Nech $T(k)$ označuje čas na vyriešenie existencie korektného preusporiadania čiastočného riešenia pre k vrcholov

$$T(k) \leq 2^k(T(k/2) + T(k/2)) = 2^k \cdot 2 \cdot T(k/2) = 2^{k+1}2^{k/2+1}T(k/4) = \cdots = O^*(4^k)$$

Riešime $O^*(5^n)$ podproblémov, pre každý z nich stačí čas $O^*(4^n)$, preto je celková zložitosť $O^*(20^n)$. Lepšia analýza vedie k zložitosti $O^*(10^n)$.

□

3.3.3 (LC-)Branch-and-Bound

Veľmi časté využitie metódy Branch-and-bound (LC-B&B, resp/ B&B) nachádzame pri riešení takých optimalizačných kombinatorických problémov, keď prehľa-

danie priestoru potenciálnych riešení síce garantuje optimáne riešenie ale veľkosť prehľadávaného priestoru robí problém prakticky neriešiteľným. Snahou je "zmenšiť" prehľadávaný priestor identifikovaním tých jeho častí, v ktorých sa hľadané riešenie určite nenachádza; hovoríme tomu *orezávanie*.

MAxSAT

Príklad 3.4 *Majme problém MaxSAT: vstupom je formula F v KNF*[6]*, cieľom také priradenie α, ktoré maximalizuje počet splnených klauzúl; niekedy nás zaujíma len to číslo.*
Nech vstupom je formula F

$$\begin{aligned} F = \quad & (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2) \\ & \wedge (x_1 \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_3} \vee x_4) \\ & \wedge x_3 \wedge (x_1 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge x_1 \end{aligned}$$

Aplikujme B&B pri rôznych stratégiách generovania vrcholov (bez orezávania, do hĺbky najprv ľavý syn, do hĺbky najprv pravý syn, ...) Všimnime si, ako sa zvolená stratégia prejavila na počte vygenerovaných vrcholov.

Identifikovať oblasti, v ktorých sa optimálne riešenie nenachádza, nie je jednoduché. Neraz však je charakter úlohy taký, že analýzou čiastočného riešenia dokážeme usúdiť, že kvalita každého riešenia, ktoré z neho dostaneme, bude horšia ako nejaké už známe riešenie. To je jeden z dôvodov, prečo sa často sústredíme práve na fázu predvýpočtu, v ktorej sa snažíme o získanie čo najlepšieho *odhadu* optimálneho riešenia. Na základe tohto odhadu potom môžme veľa vetiev v strome priestoru riešení orezať.

Aké metódy sa používajú na získavanie týchto dobrých odhadov?
- Aproximačné algoritmy
- Relaxácia redukciou na lineárne programovanie
- Random sampling
- Lokálne prehľadávanie
- Heuristiky (simulované žíhanie, genetické algoritmy,..)

Metóda LC-B&B postupne generuje nasledovníkov v priestore (čiastočných)riešení (branch), na základe analýzy prípustnosti a kvality najlepšieho dosiahnuteľného riešenia oreže vetvy, ktoré daným kritériom neprešli (bound), potom prežité prvky ohodnotí a presunie sa do najlepšieho z nich (LC=least cost). [7] V priebehu výpočtu samozrejme ako kritérium orezávania používame to lepšie z {odhad, doteraz-najlepšie-riešenie}. Uvedomme si, že doteraz-najlepšie-riešenie môžme na orezanie použiť len vtedy, ak v priebehu celého výpočtu platí, že cena vrchola x nie je väčšia ako najlepšie riešenie v podstrome s koreňom v x. (Prečo?)

Spomeňme si aj na riešenie problému plnenia batoha touto metódou. Vzhľadom k tomu, že ide o maximalizačnú úlohu, vyrobili sme prenásobením účelovej funkcie mínus jednotkou z problému úlohu minimalizačnú. Mohli by sme však rovnako dobre použiť namiesto minimalizácie maximalizáciu.

Keďže sme použili statický strom, na úrovni j sme rozhodovali o zaradení/nezaradení prvku (w_i, p_i). Ako dolný odhad na korektnú hodnotu $c(x)$ by sme prirodzene použili doteraz zozbieraný profit $\sum_{1 \leq i < j} p_i x_i$. Pre horný odhad použijeme na neznámu časť hodnotu optimálneho riešenia pri racionálnych koeficientoch. Riešenie pri celočíselných koeficientoch lepšie byť nemôže.

[6] KNF-konjunktívna normálna forma/konjunkcia elementárnych disjunkcií

[7] Príkladom použitia tejto metódy bolo riešenie problému obchodného cestujúceho minulý semester.

Je zrejmé, že ak príjmeme predpoklad $P \neq NP$, tak nemožno očakávať, že by uvedená metóda garantovala polynomiálny čas. Ziskom nie je garantovane menšia zložitosť v každom prípade, ale znižovanie zložitosti konkrétnych inštancií.

3.3.4 Predspracovanie

Zníženiu zložitosti riešenia môže pomôcť predspracovanie vstupu. Začnime príkladmi na zahriatie, kde predspracovanie pomôže k zníženiu zložitosti pôvodne polynomiálneho algoritmu.

Príklad 3.5 *Na vstupe máme dve postupnosti čísel a jednu hodnotu*

$x_i + y_j \stackrel{?}{=} S$

$$x_1, \ldots, x_k;\ y_1, \ldots, y_k,\ S$$

Pýtame sa, či existujú také indexy i, j, aby $x_i + y_j = S$.

Triviálne riešenie "hrubou silou" preverí všetky potenciálne dvojice a je teda zložitosti $O(k^2)$.

Postavme otázku trochu inak. Pre každé y_j sa pýtame na exstenciu x_i tak, aby $x_i + y_j = S$, resp. $x_i = S - y_j$. Pri takejto formulácii vidíme, že zložitosť závisí od efektivity vyhľadávania. Ak $x_1, \ldots x_k$ utriedime, môžme x_i hľadať binárnym vyhľadávaním, čo dáva celkovú zložitosť $O(k \log k)$

Príklad 3.6 *Upravme mierne zadanie. Na vstupe máme postupnosť bodov a čísel*

$x_{i(j)} + z_j \leq W,$
$y_{i(j)} \rightarrow \max$

$$(x_1, y_1), \ldots, (x_k, y_k); z_1 \ldots, z_k, W$$

Ku každému z_j hľadáme maximálne také $y_{i(j)}$, že $x_i(j) + z_j \leq W$

Triviálne riešenie prehľadaním všetkých možností je opäť $O(k^2)$.

Predspracovanie spočíva v eliminácii takých bodov, ktoré výsledok neovplyvnia. *predspracovanie*

- Majme dva body $(x_i, y_i), (x_j, y_j)$ také, že $x_i \leq x_j$ & $y_i \geq y_j$. Vtedy hovoríme, že (x_i, y_i) *dominuje* nad (x_j, y_j), pretože y_j nemôže byť maximálne pre žiadne z_k.
- Zo vstupu teda môžme odstrániť tie body, ktorým iné dominujú.
- Hľadanie dominujúcich bodov je štandardná metóda výpočtovej geometrie: utrieď podľa x a na jeden prechod odstráň tie body, ktorým iný dominuje.

Hľadaním v utriedenom poli nájdeme najväčšie y_i, pre ktoré $x_i + z_j \leq W$. Zložitosť sme tak znížili na $O(k \log k)$.

◇

Príklad 3.7 *Uvažujme problém čiastočných súčtov:* *subset-sum*

vstup: *postupnosť kladných čísel $a_1, \ldots, a_n$ a hodnota S*
výstup: *1 ak existuje množina indexov $I \subseteq \{1, \ldots, n\}$ tak, že $\sum_{i \in I} a_i = S$*

Riešenie hrubou silou je zložitosti $O(2^n)$.

Ak existuje riešenie tohto problému, potom príslušnú množinu indexov I môžme rozdeliť na dve dizjunktné podmnožiny I_1 a I_2 tak, že $S = S_1 + S_2$, kde $S_1 = \sum_{i \in I_1} a_i$, $S_2 = \sum_{i \in I_2} a_i$.

To vedie k nasledovnému riešeniu: *predspracovanie*

- Rozdelíme množinu $\{1, \ldots, n\}$ na dve podmnožiny $I = \{1, \ldots, \lfloor n/2 \rfloor\}$ a $J = \{\lfloor n/2 \rfloor + 1, \ldots, n\}$
- Vytvoríme množinu X všetkých čiastočných súčtov prvkov s indexami z I, analogicky Y sú všetky čiasotčné súčty prvkov s indexami z J

  $O(2^{n/2})$
- Utriedime X $O(2^{n/2} \log 2^{n/2})$
- Ku každému prvku $y_i \in Y$ hľadáme v X prvok $S - y_i$

  $O(2^{n/2} \cdot \log 2^{n/2})$

Celková zložitosť je

$$\mathbf{O(2^{n/2})} + \mathbf{O(2^{n/2} \log 2^{n/2})} + \mathbf{O(2^{n/2} \cdot \log 2^{n/2})} = O(n \cdot 2^{n/2}) = O^*(2^{n/2})$$

◇

Úloha: Využite predspracovanie na zníženie zložitosti pre problém plnenia batoha s binárnymi koeficientami. Akú zložitosť má Vaše riešenie?

3.3.5 Predspracovanie a textové algoritmy

V tejto časti sa budeme zaoberať problémami, ktoré súvisia so spracovaním textov. Konkrétne pôjde o vyhľadávanie vzorky v texte pri viacerých spôsoboch zadania vzorky či textu
Napriek tomu, že problémy sa ľahko riešia polynomiálnymi algoritmami, existuje viacero dôvodov, prečo sa týmito problémami budeme zaoberať.

- problémy samotné sú zaujímavé a dôležité (editory,...)
- riešenie týchto problémov je ukážkou rôznych prístupov k predspracovaniu toho istého problému
- pri riešení budeme vidieť prepojenie s problematikou datových štruktúr a konečných automatov[8]

Pre začiatok budú vzorka aj text zadané ako reťazce znakov. Budeme sa teda venovať riešeniu nasledovného problému:

Text: $a_1 a_2 \ldots a_n$
Vzorka: $b_1 b_2 \ldots b_m$

Výstup: index pozície, kde sa vzorka začína (končí)

Naivným algoritmom na riešenie tohto problému je postupné prikladanie vzorky k textu so zmenou začiatku priloženia na prvú možnú pozíciu. Tento prístup vedie k zložitosti $O(mn)$
Ukážeme, že *predspracovaním vzorky* ale aj *predspracovaním textu* môžme zložitosť algoritmu znížiť.

Prespracovanie vzorky zadanej reťazcom znakov

predspracovanie vzorky

Predstavme si, že priložíme vzorku k textu tak, že začína na pozícii p. Nech sa zhoduje s textom na prefixe dĺžky i. Ak sa nasledujúci $(i+1)$-ý symbol vzorky líši od odpovedajúceho symbolu textu, musíme vzorku v texte posunúť. Vieme povedať,

[8] Viac o problematike konečných automatov v prednáške Formálne jazyky a automaty

o koľko symbolov? Položme túto otázku inak – koľko už prečítaných symbolov textu môžme považovať za začiatok vzorky? Predpokladajme, že odpoveďou na túto otázku je hodnota funkcie f:
Pre fixovanú vzorku $b_1b_2\ldots b_m$ nech $\mathbf{f(i)}$ je maximálna dĺžka *vlastného* sufixu reťazca $b_1b_2\ldots b_m$, ktorý sa rovná prefixu tohto reťazca

$$f(i) = \max\{j \mid b_1b_2\ldots b_j = b_{i-j+1}\ldots b_i\}$$

$$\begin{array}{l} a_1a_2\ \ldots\ a_pa_{p+1}\ldots\ a_{p+i-1}\ldots a_n \\ \quad\ b_1b_2\quad \ldots\ b_i \\ \quad\quad\ b_1\ \ldots\ b_{f(i)} \end{array}$$

Predstavme si, že pri čítaní textu postupne zľava doprava máme v premennej i index pozície čítaného symbolu a premenná $dlzka$ obsahuje maximálnu dĺžku sufixu doteraz prečítanej časti textu, ktorú môžme považovať za začiatok vzorky. Vedeli by sme definovať funkciu *update*, ktorá by na základe i (resp. a_i) a $dlzka$ vypočítala novú hodnotu $dlzka$ tak, aby odpovedala prečítaniu symbola a_i zo vstupu?

Ak by sme poznali funkciu f, tak pomerne ľahko. Ak sa posledných $dlzka$ symbolov prečítaného textu rovná $b_1\ldots b_{dlzka}$ a ak prečítaný symbol je $b_{dlzka+1}$, tak novou hodnotou $dlzka$ bude $dlzka+1$. Inak je nová hodnota $dlzka$ určená na základe toho, ako by bola zmenená, keby jej hodnota bola $f(dlzka)$.

Algoritmus 4 Výpočet *update* pri znalosti chybovej funkcie f

1: $update(x, 0) \leftarrow 0$ pre $x \neq b_1$
2: $update(b_1, 0) \leftarrow 1$
3: **for** dlzka := 1 to m **do**
4: $\quad update(b_{dlzka+1}, dlzka) \leftarrow dlzka + 1$
5: $\quad update(b, dlzka) \leftarrow update(b, f(dlzka))$ pre $b \neq b_{dlzka+1}$

Ostáva vypočítať chybovú funkciu f. Pri jej výpočte si treba uvedomiť, že ak $f(i+1) = s$, potom platí

$$b_1b_2\ldots b_s = b_{i-s+2}\ldots b_{i+1} \qquad \text{ale aj} \qquad b_1b_2\ldots b_{s-1} = b_{i-s+2}\ldots b_i$$

Preto $\mathbf{s \leq f(i) + 1}$

Algoritmus 5 Výpočet chybovej funkcie f

1: $f(1) \leftarrow 0$
2: **for** i:=2 to m **do**
3: $\quad l \leftarrow f(i-1)$
4: $\quad$ **while** $(b_{l+1} \neq b_i) \wedge (l > 0)$ **do** $l \leftarrow f(l)$
5: $\quad$ **if** $(b_{l+1} \neq b_i) \wedge (l = 0)$ **then** $f(i) \leftarrow 0$
6: $\quad$ **else** $f(i) := l + 1$

Korektnosť riešenia je zrejmá z predchádzajúcej diskusie. **Zložitosť** získame na základe analýzy zložitosti algoritmov 44 a 45.

- zložitosť výpočtu tabuľky hodnôt $f(n)$ súvisí s počtom modifikácií globálnej premennej l. Keďže táto štartuje s počiatočnou hodnotou 0 a v priebehu výpočtu je buď zmenšovaná (príkazom $l \leftarrow f(l)$ v riadku 4) alebo zvýšená o 1 nanajvýš raz v cykle pre i(v riadku 6), je zložitosť výpočtu chybovej funkcie $O(m)$.

- zložitosťou výpočtu tabuľky pre funkciu *update* je úmerná veľkosti "tabuľky", ktorou je *update* zadaná, a preto je $O(m)$.
- Zložitosť samotného výpočtu je zrejme $O(n)$.

Fakt 3.8 *Celková zložitosť algoritmu vyhľadávania vzorky v texte s chybovou funkciou je* $\mathbf{O(m+n)}$.

Úloha: Použite analogický prístup na vytvorenie algoritmu, ktorý v texte vyhľadáva výskyt jedného zo zadaných reťazcov

$$b_{11}b_{12}\dots b_{1m_1}$$
$$\vdots$$
$$b_{k1}b_{k2}\dots b_{km_k}$$

Algoritmus má pracovať so zložitosťou $O(n+\sum m_i)$.

Predspracovanie textu - pozičné stromy

Postupujme inak – predspracujme text. Pomocou tohto prístupu môžeme riešiť viaceré problémy

- výskyt vzorky v texte
- najdlhší opakujúci sa podreťazec v texte
- najdlhší spoločný podreťazec dvoch reťazcov

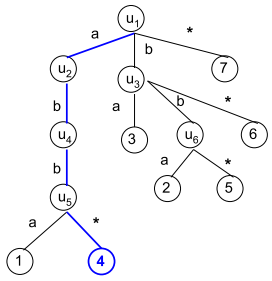
Majme reťazec $x = a_1a_2\dots a_n$. Hovoríme, že symbol a_i je *na pozícii* i. Podreťazec u **určuje pozíciu** i **v slove** x ak je to najkratší taký podreťazec, ktorého jediný výskyt v slove x začína na pozícii i. Reťazcu u budeme hovoriť **identifikátor pozície i**. Formálnejšie:

$$x = yuz,\ |y| = i-1 \quad (x = y'uz' \Rightarrow y = y')$$

Napr. V reťazci $abbabb*$ určuje pozíciu 2 reťazec bba, ale pozícia 2 nie je určená reťazcom bb, pretože $a.bb.abb* = abba.bb.*$.
Všimnime si, že pridanie nového znaku "*" na koniec slova spôsobilo, že máme pre každú pozíciu zaručenú existenciu reťazca, ktorý túto pozíciu určuje.
Nech $S(i)$ označuje identifikátor pozície i. Vytvorme *lexikografický strom*[9] reprezentujúci množinu identifikátorov $\{S(1), S(2), \dots, S(n+1)\}$. Do listu l tohto lexikografického stromu pridajme číslo tej pozície, ktorej identifikátor je uložený na hranách z koreňa do tohto listu l. Takýto strom budeme volať **pozičný strom**.

text=$abbabb*$

$$\begin{aligned} S(1) &= abba \\ S(2) &= bba \\ S(3) &= ba \\ S(4) &= abb* \\ S(5) &= bb* \\ S(6) &= b* \\ S(7) &= * \end{aligned}$$



Fakt 3.9 $|S(i)| = j \Rightarrow |S(i-1)| \leq j+1$.

[9] Lexikografixký strom pre množinu reťazcov má na hranách symboly a každý reťazec, čítaný po ceste z koreňa do listu, je reťazcom z reprezentovanej množiny a naopak.

Dôkaz: Nech $t = |S(i-1)| > |S(i)|+1$. Potom existuje pozícia, povedzme m, $m > i$ niekde v reťazci taká, že sa na nej nachádza podreťazec dĺžky *väčšej* ako $|S(i)|+1$ a

$$S(i-1) = a_{i-1}a_i \dots a_{i+t-2} = a_m a_{m+1} \dots a_{m+t-1}$$

To ale znamená, že

$$a_i \dots a_{i+t-2} = a_{m+1} \dots a_{m+t-1}$$

čo je v spore s tým, že $|S(i)| = t$

Fakt 3.10 *$S(i)$ nie je vlastným prefixom $S(j)$.*

Pozičné stromy môžme využiť na riešenie viacerých problémov spracovania textov.

základný pattern matching – pýtame sa, či je reťazec $y = b_1 b_2 \dots b_p$ podreťazcom reťazca $x = a_1 a_2 \dots a_n$

Nech T je pozičný strom pre reťazec $x = a_1 a_2 \dots a_n *$. Aby sme zistili, či sa reťazec y vyskytuje ako podreťazec x, pohybujme sa v strome T podľa symbolov, tak ako ich čítame zo vzorky y. Nech $b_1 b_2 \dots b_j$ je maximálna cesta, ktorú sme v T prešli. Táto cesta končí vo vrchole v stromu T.Môžu nastať 3(4) prípady

$j < p$ a v *nie je list*, potom odpoveď je NIE

$j \leq p$ a v *je list* označený i. V tejto situácii je jediným možným/poytenciálnym začiatkom vzorky v texte pozícia i. Treba skontrolovať, či je tomu tak[10]. Ak áno, odpoveď je ÁNO, resp. i, ak nie, odpoveď je NIE.

$j = p$ a v je vnútorný vrchol. V tomto prípade sa jedná o viacnásobný výskyt vzorky v texte. Pozície začiatku výskytov sú uložené v listoch podstromu s koreňom v.

Najdlhší opakujúci sa podreťazec odpovedá ceste do vrchola s maximálnou hĺbkou

Najdlhší podreťazec reťazcov x a y - na DÚ.

Konštrukcia pozičného stromu

konštrukcia pozičného stromu

Ukážeme konštrukciu pozičného stromu, ktorá je lineárna vzhľadom na počet vrcholov tohto stromu. Hoci v niektorých prípadoch môže byť počet vrcholov skonštruovaného stromu až $O(n^2)$, dá sa ukázať, že pre náhodný reťazec je to $O(n)$.
Nech $S_i(j)$ označuje identifikátor pozície j v reťazci $x_i = a_i a_{i+1} \dots a_n *$. Pozor, číslo pozície počítame podľa pôvodného reťazca x.

Ako sa líšia $S_i(j)$ a $S_{i+1}(j)$? Odpoveď na túto otázku je základom efektívneuj konštrukcie pozičného stromu. Tým, že sme predĺžili reťazec x_{i+1} na reťazec x_i sa mohlo stať, že niektorý identifikátor $S_{i+1}(k)$ sa stal prefixom identifikátora $S_i(i)$. Preto v tomto prípade bude treba predĺžiť $S_{i+1}(k)$ na $S_i(k)$. Napr.

$$S_2(4) = a \text{ ale } S_1(4) = abb*$$

Dôležité je, že môže byť len *jediné*[11] také k.

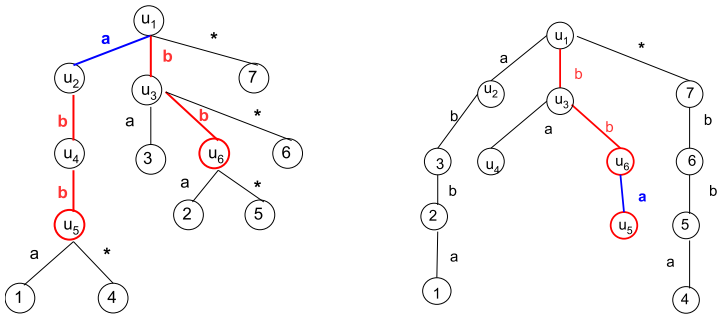
Ak chceme vedieť, či pridanie a_i znamená, že niektorý z identifikátorov bude treba predlžovať, potrebovali by sme (zišlo by sa) vedieť, či sa niekde v reťazci x_{i+1} vyskytuje podreťazec $a_i y$, pričom y je prefix $S_{i+1}(i+1)$. Ak takýto reťazec existuje, treba sa vedieť čo najefektívnejšie dostať k identifikátoru, ktorý je treba predlžovať. Napr.

[10] Ako?
[11] Inak by $S_{i+1}(k1)$ aj $S_{i+1}(k2)$ boli prefixom $S_i(i)$ a teda boli jeden vlastným prefixom druhého.

$\underbrace{a}_{a_1}\underbrace{bb}_{y}$ sa vyskytuje v reťazci $x_2 = bbabb*$

Preto budeme používať nasledujúce dátové štruktúry:

- $\mathbf{T_i}$ označuje pozičný strom skonštruovaný pre reťazec x_i
- do každého vrchola v pridáme bitový vektor $\mathbf{B_v[a]} = 1 \Leftrightarrow aY(v)$ je podreťazec x_i, pričom $Y(v)$ označuje reťazec čítaný na ceste z koreňa do vrchola v.
- do každého vrchola pridáme **informáciu o jeho hĺbke**
- $\mathbf{A_i}$ označuje pomocný strom, ktorý je konštruovaný nad tou istou množinou vrcholov ako pozičný strom T_i, ale má inú množinu hrán. Konštruujeme ho preto, aby sme urýchlili hľadanie identifikátora, ktorý treba predlžovať. Preto sa konštruuje nasledovne:
  w je a-synom[12] v v A_i , ak ay je reťazec odpovedajúci $Y(w)$ v pozičnom strome T_i, a y je reťazec odpovedajúci $Y(v)$ v pozičnom strome T_i



Obrázok 3.2: pozičný strom a pomocný strom

Fakt 3.11 *Každý pozičný strom má pomocný strom.*

krok iterácie

Konštrukcia T_i z T_{i+1}.
Začni z listu $i+1$ a postupuj smerom do koreňa dovtedy, kým nenájdeš vrchol u taký, že je prvý na tej ceste s $B_u[a_i] = 1$, resp. sa dostaneš do koreňa a $u \neq$ koreň. Potom pre každý vrchol v na ceste z $i+1$ do syna vrchola u, resp. koreňa, nastav $B_v[a_i] = 1$

1. **u neexistuje**, čo znamená, že doteraz sa symbol a_i nepoužil. Preto dostane koreň nového a_i-syna identifikujúceho pozíciu i a $B_i[a] \leftarrow 0$ pre všetky a.
2. **u existuje a nemá a_i -syna v A_{i+1}**. Toto je situácia, keď sa reťazec $a_iY(u)$ nevyskytuje celý ako (doterajší) identifikátor pozície, ale len jeho prefix. Preto postupuj ďalej do koreňa až kým nenájdeš prvý taký vrchol u_1, ktorý už má a_i -syna v A_{i+1}. Týmto synom je list v_1 identifikujúci pozíciu k.

   V tejto situácii si konštrukcia T_i vyžaduje väčšiu modifikáciu T_{i+1} :

   - Nech $u_1, u_2, \ldots, u_q, u$ je postupnosť vrcholov na ceste z u_1 do u. Nech e_i je meno hrany vychádzajúcej z vrchola u_i.
   - Vytvor q nových vrcholov $v_2, \ldots, v_q, v$. Pritom v_i je e_{i-1}-synom vrchola v_{i-1} a v je e_q-synom vrchola v_q v pozičnom strome T_i.

[12] a-synom vrchola v nazývame toho syna vrchola v, do ktorého sa dostaneme po hrane označenej symbolom a

– Nech j je i+ hĺbka vrchola u (v T_{i+1})
  Nech m je k+ hĺbka vrchola v (v T_{i+1})
  Pridaj dva nové listy identifikujúce pozíciu i, resp. k:
    i je a_{j+1}- syn vrchola v
    k je a_{m+1}- syn vrchola v
– $\forall a \in I\ B_v[a] \leftarrow B_{v1}[a], v \in \{v_2, \ldots, v_q, v, k\}$
– $B_i[a] \leftarrow 0$

Analogicky treba modifikovať aj A_{i+1}:

– v je a_i -synom u
– u_1 je a_{j+1}-syn u, u_2 je a_{m+1}-syn u
  i je a_i-syn u_1, k je a_i-syn u_2
– v_t je a_i-syn u_t

3. **u existuje a má v A_{i+1} a_i-syna v** - v tejto situácii vieme, že v T_{i+1} sa reťazec $a_iY(u)$ vyskytuje celý ako prefix identifikátora nejakej pozície

   - v je list označený k To znamená, že
     [-] $S_i(i) = a_i a_{i+1} \ldots a_j a_{j+1}$

     [ ]$S_{i+1}(k) = a_k a_{k+1} \ldots a_m$, pričom $a_k a_{k+1} \ldots a_m = a_i a_{i+1} \ldots a_j$
     → odstráň označenie k z vrchola v
     → v dostane dvoch nových synov: a_{j+1}-syna i, a a_{m+1}-syna k
     → Ďalej potrebné úpravy $B[a]$ a A_{i+1}.
   - v je vnútorný vrchol
     → j je i+hĺbka u
     → nový vrchol i je a_{j+1}-syn v
     → $B_i[a] \leftarrow 0$.
     → Nech u_1 je a_{j+1}-syn u v T_{i+1}. Potom nový vrchol i sa stane a_i-syn u_1.

Zložitosť konštrukcie pozičného stromu.

- nájdenie listu $i+1$ realizujeme v konštantnom čase (je to posledne pridávaný list–identifikátor pozície). Celkovo teda $O(n)$
- pridanie i je úmerné počtu pridávaných vrcholov pri prechode od T_{i+1} k T_i. Pridanie všetkých identifikátorov pozícií je preto úmerné veľkosti stromu T_1.
  $O(|T_1|)$
- súčet dĺžok ciest do jednotlivých u

Nech $d_2, \ldots, d_{n+1}$ je vzdialenosť $2, \ldots, n+1$ do príslušného u, nech e_i je hĺbka i v T_i

$$e_i \leq e_{i+1} - d_{i+1} + 2$$

$$\sum_{i=2}^{n+1} d_i \leq \sum_{i=2}^{n+1} (e_i - e_{i-1} + 2) = e_2 - e_1 + e_3 - e_2 + \cdots + e_{n+1} - e_n + 2n = e_{n+1} - e_1 + 2n = O(n)$$

Celkový čas konštrukcie pozičného stromu je $O(n) + O(|T_1|) = O(|T_1|)$. Dá sa ukázať, že v priemernom prípade to vychádza $O(n)$.

Úloha: Zamyslite sa nad charakterizáciou reťazcov z pohľadu veľkosti výslednej reprezentácie pozičným stromom.

3.4 Lokálne prehľadávanie

Metóda lokálneho prehľadávania sa používa pri riešení optimalizačných úloh. Priestor riešení zorganizujeme tak, aby pre každé riešenie bola určená množinu jeho susedov. Potom sa hýbeme v tomto priestore cez množiny susedov tak, aby sme v každom kroku riešenie vylepšili. Pri takomto prehľadávaní skončíme zrejme v lokálnom optime. Na čo sa pri uvedenej metóde treba sústrediť:

- Ako získame **štartovací bod**? Často použijeme nejakú rýchlu heuristiku. Niekedy sa využíva aj viacnásobné spustenie LSS z náhodne vygenerovaných štartových bodov - potom hovoríme o *multistart local search*
- Ako definujeme **množinu susedov**? Uvedomme si, že vypočítať suseda nesmie byť veľmi zložité. Tiež by tá množina susedov nemala byť príliš veľká, keďže ju pri výbere nasledujúceho kroku chceme prezerať
- Ako určíme **nové riešenie**? Zvyknú sa používať dva základné prístupy
  - prvé lepšie (*first improvement*)
  - najlepšie spomedzi susedov (*best improvemenet*)

susednosť

Formálnejšie. Nech x je vstup do optimalizačného problému U, $M(x)$ označuje množinu prípustných riešení. Definujeme *susednosť* ako funkciu f_x z $M(x)$ do $Pot(M(x))$, kde $Pot(A)$ označuje vštky podmnožiny množiny A. Vyžadujeme, aby

1. $\alpha \in f_x(\alpha) \quad \forall \alpha \in M(x)$
2. $\beta \in f_x(\alpha) \Longrightarrow \alpha \in f_x(\beta)$ //symetria
   //od tejto podmienky niekedy upúšťame
3. $\forall \alpha, \beta \in M(x)\ \exists k, \gamma_1, \ldots, \gamma_k \in M(x)$ tak, že
   $\gamma_1 \in f_x(\alpha), \gamma_{i+1} = f_x(\gamma_i), \beta = f_x(\gamma_k)$ //dosiahnuteľnosť

Susednosť f_x prirodzene definuje graf susednosti

$$G_{M(x),f_x} = (M(x), \{(\alpha, \beta) | \alpha \in f_x(\beta), \alpha \neq \beta, \alpha, \beta \in M(x)\})$$

Má tiež zmysel hovoriť o vzdialenosti dvoch riešení vzhľadom k susednosti f_x; vzdialenosťou dvoch riešení rozumieme dĺžku cesty, ktorá ich v grafe susednosti spája. Túto vzdialenosť označíme *dist*

Schému lokálneho prehľadávania podľa susednosti f_x možno načrtnúť takto:

Algoritmus 6 LSS_{f_x}

```
1: nájdi prípustné riešenie α ∈ M(x)
2: while α nie je lokálne optimum do
3:     nájdi β ∈ f_x(α) s lepšou cenou
4:     α ← β
5: return α
```

lokálne optimum vzhľadom k f_x

Čo myslíme lokálnym optimom? Nech U je optimalizačný problém s účelovou funkciou *cena*, *ciel* $\in \{\max, \min\}$ určuje, či ju chceme minimalizovať alebo maximalizovať. Potom hovoríme, že prípustné riešenie $\alpha \in M(x)$ je *lokálne optimum vzhľadom k f_x* ak

$$cena(\alpha) = ciel\{cena(\beta) \mid \beta \in f_x(\alpha)\}$$

použitie LSS

Ľahko vidno, že k popisu algoritmu LSS pre riešenie nejakého problému väčšinou stačí popísať susednosť $f_x(\alpha)$

Fakt 3.12 *Každý algoritmus lokálneho prehľadávania, ktorý je založený na LSS, dá na výstup prípustné riešenie, ktoré je lokálne optimálne vzhľadom k použitej susednosti f_x.*

Čo vieme povedať o zložitosti riešenia problému použitím Algoritmu LSS? Zložitosť zrejme ovplyvňuje

- zložitosť výpočtu lokálneho optima //presnosť vs. čas
- počet iterácii while cyklu

  //Ak riešime problém s celočíselnou účelovou funkciou,

  // znamená každé opakovanie vylepšenie aspoň o 1.

Príklad 3.8 *Uvažujme problém obchodného cestujúceho a dve riešenia založené na dvoch rôzne definovaných susednostiach* *TSP*

2-exchange *z existujúcej HK(prípustného riešenia TSP) odstránime dve hrany (a,b), (c,d) také, že $|\{a,b,c,d\}| = 4$. Tieto dve hrany nahradíme hranami (a,c), (b,d).*

3-exchange *z existujúcej HK odstránime 3 hrany a nahradíme ich inými (nie nutne odlišnými) tromi hranami*

Príklad 3.9 *Pri hľadaní spĺňajúceho priradenia pre boolovskú formulu $F(x_1,\ldots,x_n)$ je susedom vektora hodnôt $\alpha = (\alpha_1,\ldots,\alpha_n)$ taký vektor hodnôt, ktorý vznikne* zmenou práve jedného bitu [13] *v riešení α.* *SAT*

Úzkym miestom v metóde LSS je to, že sa hýbeme len v okolí momentálneho riešenia, pričom trváme na tom, aby sa riešenie stále vylepšovalo. Uvažovala sa taká modifikácia, že v jednej iterácii prezrieme okolie riešenia do nejakej vzdialenosti/hĺbky — *prehľadávanie s variabilnou hĺbkou.* Predpokladáme taký typ optimalizačného problému, pri ktorom riešenie vieme popísať zoznamom špecifikácií $(p_1,\ldots,p_n)$. Nech pre vstup x je $M(x)$ množina prípustných riešení. Potom *Kernighan-Lin prehľadávanie s variabilnou hĺbkou*

$$f_x^k(\alpha) = \{\beta \in M(x) \mid dist_{f_x}(\alpha,\beta) \leq k\}$$

je množina riešení, ktoré z riešenia α vieme získať postupným aplikovaním niekoľkých, nanajvýš však k, lokálnych transformácií na α.
Nech α, β sú prípustné riešenia, *cena* účelová funkcia. Definujme

$$zisk(\alpha,\beta) \stackrel{def.}{=} cena(\alpha) - cena(\beta)$$

Potom v jednom "kroku" aplikujeme nanajvýš n lokálnych transformácií, pričom

- Ak začneme s prípustným riešením $\alpha = (p_1,\ldots,p_n)$, tak pre výsledné prípustné riešenie $\gamma = (q_1,\ldots,q_n)$ platí, že $q_i \neq p_i$
- Ak $\alpha_0, \alpha_1, \ldots, \alpha_m$ je vytváraná postupnosť prípustných riešení, tak $\alpha_0{=}\alpha$, $\alpha_m{=}\gamma$ a
  1. $zisk(\alpha_i, \alpha_{i+1}) = \max\{zisk(\alpha_i,\delta) \mid \delta \in f_x(\alpha_i)\}$
  2. ak sa v i-tej iterácii zmenil parameter p_j, tak p_j sa v žiadnej ďalšej iterácii nezmení
- Po vytvorení postupnosti $\alpha_0, \alpha_1, \ldots, \alpha_m$ sa α nahradí tým α_i, ktoré optimalizuje $zisk(\alpha,\alpha_j)$. Ak je to zmena k lepšiemu, pokračujeme ďalšou iteráciou. Ak nie, výstupom je α.

Algoritmus 7 KL-LSS (s variabilnou hĺbkou)

1: nájdi prípustné riešenie $\alpha = (p_1, \ldots, p_n) \in M(x)$
2: *zlepsenie* $\leftarrow$ true
3: *exchange* $\leftarrow \{1, \ldots, n\}; j \leftarrow 0; \alpha_j \leftarrow \alpha$
4: **while** *zlepsenie* **do**
5: **while** *exchange* $\neq \emptyset$ **do**
6: $j \leftarrow j + 1$
7: $\alpha_j \leftarrow$ riešenie z $f_x(\alpha)$, ktoré optimalizuje $zisk(\alpha_{j-1}, \alpha_j)$ a od α_{j-1} sa líši len v paramteroch z *exchange*
8: *exchange* $\leftarrow$ *exchange*/parametre, v kt. sa líšia α_{j-1}, α_j
9: vypočítaj $zisk(\alpha, \alpha_i)$ pre $i = 1, \ldots, j$
10: vypočítaj $\ell \in \{1, \ldots, j\}$ také, že $zisk(\alpha, \alpha_\ell) = \max\{zisk(\alpha, \alpha_i) | i = 1, \ldots, j\}$
11: **if** $zisk(\alpha, \alpha_\ell) > 0$ **then**
12: $\alpha \leftarrow \alpha_\ell$
13: *exchange* $\leftarrow \{1, \ldots, n\}$
14: **else** *zlepsenie* $\leftarrow$false
15: return α

Niekoľko krokov nesprávnym smerom môže byť eliminovaných jedným krokom správnym smerom. Na hľadanie najlepšieho riešenia v priestore $f_x^k(\alpha)$ používame greedy prístup.
Uvedomme si, že tento KL-LSS algoritmus sa drží greedy prístupu, čo zabraňuje tomu, aby sme pri hľadaní suseda pre α mali čas exponenciálny od $|\alpha|$. Takže zložitosť jedného vylepšenia je $O(n \cdot t(|\alpha|)|f_x(\alpha)|)$, pričom funkcia f hovorí o zložitosti realizácie jednej transformácie.

Ukážeme aplikáciu prehľadávania s variabilnou hĺkou na dvoch problémoch. Začneme **problémom minimálneho vyváženého rezu**:

Vstup: graf s ohodnotenými hranami $G = (V, E, c)$
Výstup: rozdelenie množiny vrcholov na disjunktné rovnako veľké podmnožiny V_1, V_2, ktoré minimalizujú cenu hranového rezu

KL-LSS a minimálny vyvážený rez

- Nech V_1, V_2 je prípustné riešenie. Lokálna transformácia spočíva vo výbere dvoch vrcholov $a \in V_1, b \in V_2$ a ich vzájomnej výmene
- Aplikáciou výmeny $a \longleftrightarrow b$ sa zníži/zmení cena rezu o hodnotu $\Delta(a, b)$

$$\Delta(a,b) = \sum_{\substack{\{a,v\}\in E \\ v\in V_2\setminus\{b\}}} c(a,v) - \sum_{\substack{\{a,v\}\in E \\ v\in V_1}} c(a,v) + \sum_{\substack{\{b,v\}\in E \\ v\in V_1\setminus\{a\}}} c(b,v) - \sum_{\substack{\{b,v\}\in E \\ v\in V_2}} c(b,v)$$

- nové prípustné riešenie je najlepšie spomedzi tých, ktoré získame postupnosťou n zámen

[13] Hamingova vzdialenosť vektorov je 1

Algoritmus 8 iterácia Min-Balanced-Cut

//predpokladámne, že nejaké rozdelenie V_1, V_2 už máme
$U_1 \leftarrow V_1; U_2 \leftarrow V_2$
$X \leftarrow V$
for i=1 to n **do**
    vyber vrcholy $a_i \in U_1 \bigcap X, b_i \in U_2 \bigcap X$ pre kt. je $\Delta(a,b)$ maximálna
    $U_1 \leftarrow (U_1 \setminus \{a_i\}) \cup \{b_i\}$
    $U_2 \leftarrow (U_2 \setminus \{b_i\}) \cup \{a_i\}$
    $X \leftarrow X \setminus \{a_i, b_i\}$
vyber k maximalizujúce $\sum_{i=1}^{k} \Delta(a_i, b_i)$
aplikovaním postupnosti zmien $a_i \leftrightarrow b_i, \ldots, a_k \leftrightarrow b_k$ na V_1, V_2 získaj V_1', V_2'
return (V_1', V_2')

KL-LSS a TSP

Druhým príkladom použitia metódy prehľadávania s variabilnou hĺbkou je použitie pre TSP[14]. Vychádzame z existujúcej hamiltonovskej kružnice α. V iterácii sa snažíme nájsť dve rovnako dlhé postupnosti hrán také, že jedna z nich obsahuje len hrany z α a druhá zas len hrany mino α, pričom ich vzájomná výmena zachová HK ($\alpha \rightsquigarrow \alpha'$) a zníži jej cenu. Hľadáme teda $C = (p_1, q_1), \ldots, (p_k, q_k)$, $C' = (s_1, t_1), \ldots, (s_k, t_k)$ tak, že

1. p_i, q_i sú v α susedné, $1 \leq i \leq k$
2. s_i, t_i nie sú v α susedné, $1 \leq i \leq k$
3. $q_i = s_i$, $1 \leq i \leq k$
4. $t_i = p_{i+1}$, $1 \leq i \leq k-1$
5. $t_k = p_1$
6. $\alpha \setminus C \cup C'$ vytvorí novú HK α' kratšej dĺžky

Algoritmus 9 iterácia TSP

$\Delta = 0, k = 1$, (p_1, q_1) je dvojica vrcholov susedných v α
$C \leftarrow \{(p_1, q_1)\}$; $s_1 \leftarrow q_1$, $i \leftarrow 1$
loop nájdi vrcholy x, y susedné v α také, že
    nepatria do C
    ak $i+1$ hrán (p_1, q_1), ..., (p_i, q_i), (x, y) nahradíme hranami (s_1, t_1), ..., (s_i, x), (y, p_1) tak z α vznikne HK α'
    $\Delta + c(p_i, q_i) + c(x, y) - c(p_1, y) - c(s_i, x) > 0$
  if také x, y existujú **then**
    $t_i = p_{i+1} = x$, $q_{i+1} = s_{i+1} = y$, $k \leftarrow i+1$
    $i \leftarrow i+1$
  if $k > 1$ **then** nahraď v α k hrán $(p_1, q_1), \ldots, (p_k, q_k)$ za $(s_1, t_1), \ldots, (s_k, t_k)$
return(α)

◇

kvalita vs. zložitosť

Ak máme NP ťažké problémy, nemôžme očakávať, že by táto metóda viedla k polynomiálnemu riešeniu. Jej zložitosť môžme vyjadriť ako

(čas hľadania v množine susedov) × (počet vylepšení)

Mohli by sme sa pýtať

> *Pre ktoré NP ťažké problémy môžme nájsť susednosť f_x polynomiálnej veľkosti tak, aby LSS_{f_x} dávalo vždy optimálne riešenie?*
> (pozor, nehovoríme o polynomialite celého riešenia!)

presná susednosť(okolie)

Definícia 3.6 *Nech $U = (I, Sol, cena, ciel)$ je optimalizačný problém, f určuje susednosť/okolie na U. Hovoríme, že f je* presná susednosť/okolie, *ak $\forall x \in I$ je každé lokálne optimum pre x podľa f_x optimálnym riešením pre x.*

polynomiálna prehľadávanie

Susednosť f je polynomiálne prehľadávanie *ak existuje polynomiálny algoritmus, ktorý $\forall x \in I$ a $\forall \alpha \in Sol(x)$ nájde jedno z najlepších prípustných riešení v $f_x(\alpha)$.*

Uvedomme si, že polynomiálne prehľadávanie nezaručuje polynomiálnu veľkosť množiny susedov!

Ďalšou zmysluplnou otázkou je otázka existencie presnej susednosti pre U, ktorá by bola aj polynomiálnym prehľadávaním. Kladná odpoveď by hovorila, že všetko záleží od počtu iterácií. Negatívna odpoveď zas znamená, že nevieme polynomiálnym prehľadávaním garantovať optimálne riešenie.

Existujú metódy, ako dokázať, že problém je z hľadiska lokálneho prehľadávania ťažký. Prvá využíva pojem ohraničenej ceny.

Definícia 3.7 *Nech $U = (I, Sol, cena, ciel)$ je celočíselný optimalizačný problém. Hovoríme, že U má* ohraničenú cenu, *ak $\forall I \in I$ $Int(I) = (i_1, \ldots, i_n), i_j \in N$*

$$cena(\alpha) \leq \sum_{j=1}^{n} i_j \text{ pre } \alpha \in Sol(I)$$

Uvedená podmienka je celkom prirodzená, väčšinou je ohraničujúca funkcia súčtom nejakej podmnožiny hodnôt zo vstupu.

Analógiou triedy NP pre optimalizačné problémy je trieda NPO.

NPO

Definícia 3.8 *Nech $U = (I, Sol, cena, ciel)$ je optimalizačný problém. $U \in$ NPO ak platia nasledovné podmienky:*

1. *$I \in$ P*
2. *existuje polynóm p taký, že*
   - *$\forall x \in I, y \in M(x)$, $|y| \leq p(|x|)$*
   - *existuje polynomiálny algoritmus, ktorý $\forall x \in I$ a $y \in \Sigma_O^*$ dĺžky $|y| \leq p(|x|)$ rozhodne, či $y \in M(x)$*
3. *účelová funkcia cena sa počíta v polynomiálnom čase*

PO

Ak pre NPO problém existuje deterministický polynomiálny algoritmus, ktorý počíta optimálne riešenie, potom problém patrí do PO.

Veta 3.13 *Nech $U \in$ NPO je celočíselný optimalizačný problém s ohraničenou cenou taký, že existuje polynomiálny algoritmus $\mathcal{B}$, ktorý každému vstupu vypočíta nejaké prípustné riešenie. Ak P $\neq$ NP a U je silno NP-ťažký, tak pre U neexistuje presná susednosť, ktorá je polynomiálnym prehľadávaním.*

Dôkaz: Kvôli sporu predpokladajme, že existuje presné okolie, ktoré je polynomiálnym prehľadávaním. Podľa definície existenciu polynomiálneho prehľadávania potvrdzuje existencia polynomiálneho algoritmu $\mathcal{A}$, ktorý prípustnému riešeniu α v polynomiálnom čase vypočíta nezhoršené prípustné riešeni β. Využijeme algoritmy $\mathcal{A}, \mathcal{B}$ pri konštrukcii pseudopolynomiálneho algoritmu pre silno NP-ťažký (minimalizačný) problém U. Vieme, že existencia pseudopolynomiálneho algoritmu pre silno NP-ťažký problém potom vedie k sporu s predpokladom P $\neq$ NP.

[14] problém obchodného cestujúceho

Algoritmus 10 A_U- pseudopolynomiálny algoritmus pre silno NP-ťažký problém

```
1: pomocou B nájdi prípustné riešenie α ← B(x)
2: β ← A(α)
3: while α > β do
4:     α ← β
5:     β ← A(α)
6: return α
```

- Keďže $U \in NPO$, je dĺžka $|\alpha|$ polynomiálna od dĺžky vstupu $|x|$.
- Algoritmy $\mathcal{A}, \mathcal{B}$ sú polynomiálne, preto sú polynomiálne aj od $|x|$
- Nech $Int(x) = (i_1, \ldots, i_n)$. Keďže U je celočíselný problém s ohraničenou cenou, cena prípustného riešenia $\in \{1, 2, \ldots \sum_{j=1}^{n} i_j\} = \{1, 2, \ldots n \times MaxInt(x)\}$. Každé vylepšenie znamená vylepšenie aspoň o 1, preto je počet iterácií nanajvýš $|x| \times MaxInt(x)$

□

Dôsledok 3.14 *Ak $P \neq NP$, tak neexistuje presná susednosť, ktorá je polynomiálnym prehľadávaním, pre žiaden z problémov $TSP, \Delta - TSP$*

Definícia 3.9 *Nech $U = (I, Sol, cena, ciel)$ je optimalizačný problém z NPO. Pre U definujeme* suboptimálny rozhodovací problém *suboptimálny rozhodovací problém*

$$SUBOPT_U = \{(x, \alpha) \mid x \in I, \alpha \in Sol(x) \text{ a } \alpha \text{ nie je optimálne}\}$$

Veta 3.15 *Nech $U \in NPO$. Ak $P \neq NP$ a $SUBOPT_U$ je NP-ťažký, tak pre U neexistuje presná susednosť, ktorá je polynomiálnym prehľadávaním.*

3.5 Heuristiky založené na lokálnom prehľadávaní

Metódu lokálneho prehľadávania využívame najmä pri riešení optimalizačných úloh. Opakovanie jednotlivých iterácií umožňuje prechod od jedného prípustného riešenia α k druhému β vtedy, ak tento prechod znamená zlepšenie. Tento princíp je mierne porušený pri prehľadávaní s variabilnou hĺbkou, keď niekoľko pokusov v rámci jednej iterácie umožňuje uvažovať aj dočasné zhoršenie. Princíp dočasného zhoršenia využívajú aj *heuristika simulovaného žíhania*, *genetické algoritmy* a *tabu search*.

3.5.1 Simulované žíhanie

V najjednoduchšej verzii tejto metódy je definovaná (potenciálne nekonečná) postupnosť $t_1, t_2, \ldots, t_i > 0$, *prahových hodnôt*, ktoré určujú podmienky na prechod riešenia α k riešeniu β. V k-tom kroku algoritmu lokálneho vyhľadávania (predpokladáme, že máme problém minimalizácie) je tento prechod umožnený, ak $cena(\beta) - cena(\alpha) < t_k$. Pozitívna hodnota t_k umožňuje akceptovať aj zhoršenie, nulový prah zas definuje úlohu lokálneho prehľadávania.

pravidlo ukončenia

Vo všeobecnosti využívame postupnosti, v ktorých prahové hodnoty monotónne klesajú. Navyše, pre každé $\epsilon > 0$ existuje také i, že $t_i \leq \epsilon$. Výpočet končí po k krokoch, ak pre aktuálne prípustné riešenie α a všetky s ním susediace prípustné riešenia β je $cena(\beta) - cena(\alpha) \geq t_{k+1}$.

Ak prechod od lepšieho riešenia k horšiemu akceptujeme s pravdepodobnosťou inverzne proporcionálnou rozdielu $cena(\beta) - cena(\alpha)$, resp. s pravdepodobnosťou t_i, hovoríme o simulovanom žíhaní. Názov metódy súvisí s podobnosťou s fyzikálnym procesom žíhania(kalenia).

parametre

Metóda má tri voľné parametre, ktorých vhodné nastavenie ovplyvňuje úspešnosť. Nastavujú sa väčšinou experimentálne v procese ladenia metódy. Okrem toho určujeme aj podmienku ukončenia:

t-teplota
r-rýchlosť znižovania teploty
ℓ-počet testovaných prvkov v okolí
Δ-kritérium ukončenia

Algoritmus 11 simulované žíhanie

```
                                //predpokladáme vstupnú inštanciu x minimalizačného problému U
1:  τ ← t; s ← počiatočné prípustné riešenie
2:  repeat
3:      for ℓ krát do
4:          vyber nenavštívené s' ∈ M(x)
5:          δ ← cena(s') − cena(s)
6:          if δ < 0 then
7:              s ← s'
8:          else s ← s' s pravdepodobnosťou e^(−δ/π)
9:      τ ← r · τ
10: until Δ
11: return s
```

3.5.2 Genetické algoritmy

Motiváciou pre tento typ heuristiky je biológia. Genetický algoritmus udržiava *populáciu* riešení, ktorá sa vyvíja cez *generácie*. V každom kroku sa nová poplulácia

danej veľkosti maximálne N generuje z pôvodnej, pričom susednosť sa definuje náhodným aplikovaním pravidiel motivovaných genetickými zmenami, ako mutácia, kríženie,... Do ďalšej generácie prechádzajú najsilnejší jedinci. Táto metóda je vhodná aj na paralelné spracovanie.

Typicky sa nová generácia počíta v troch fázach:
Kvalita riešenia v populácii sa vyhodnotí vhodne definovanou funkciou *Vyhodnotenie kvality*

Riešenie prechádza do novej generácie s pravdepodobnosťou úmernou jeho kvalite *Výber*

Ak do novo vygenerovanej generácie prešlo menej ako určený počet N jedincov/riešení, tak sa generácia doplní o nové riešenia, ktoré vzniknú rekombináciou (crossover) resp.modifikáciou (mutation) niektorých riešení. *Generovanie nových riešení*

Riešenia udržiavame väčšinou ako binárne reťazce. Operácie sa potom realizujú nasledovne

crossover nová dvojica reťazcov β_1, β_2 vznikla z riešení α_1, α_2 zámenou prefixu reťazca α_1 dĺžky i za suffix reťazca α_2 dĺžky i pre náhodne vygenerované i, $1 \leq i \leq |\alpha_1| = |\alpha_2|$

mutation bit po bite sa každý bit s danou pravdepodobnosťou preklopí

Rez je určený rozkladom (V_1, V_2) množiny vrcholov $V = \{v_1, \ldots, v_{2n}\}$. Riešenie udržiavame ako binárny vektor α dĺžky $2n$, kde $\alpha_i = 1$ znamená príslušnosť v_i do V_1, $\alpha_i = 0$ zas indikuje príslušnosť v_i do V_2. *minimálny vyvážený rez*

Kvalita riešenia je daná funkciou Ψ

$$\Psi(\alpha) = C - \sum_{i=1}^{2n}\sum_{j=1}^{2n} c_{ij}\alpha_i(1-\alpha_j) + K|n - \sum_{i=1}^{2n}\alpha_i|$$

kde c_{ij} je cena hrany (v_i, v_j), K je dostatočne veľká konštanta, ktorej úlohou je znížiť kvalitu neprípustných riešení a C je konštanta, ktorá umožňuje kvalitu maximalizovať.

3.5.3 Tabu search

Podobne ako pri simulovanom žíhaní sa pri prehľadávaní okolia môžme náhodne rozhodnúť o akceptovaní horšieho riešenia, sofistikovanejšie je prehľadávanie.

1. pre riešenie s označuje $\Delta(s)$ množinu transformácií, ktoré sa naň môžu aplikovať. Funkcia σ^* definovaná na podmnožinách $S \subseteq \Delta(s)$ vráti najlepšiu transformáciu $\sigma^*(S) \in \Delta(s)$
2. Počas výpočtu si udržiavame množinu $\mathcal{T}$ zakázaných transformácií. Veľkosť tejto množiny určuje parameter t.
3. V každom kroku poznáme najlepšie doteraz nájdené riešenie. Nájdeme najlepšie riešenie, ktoré vieme dostať z aktuálneho riešenia aplikovaním povolenej, teda non-tabu operácie. $\mathcal{T}$ aktualizujeme.
4. Výpočet končí, keď $\Delta(s) = \mathcal{T}$ alebo sme v predchádzajúcich k krokoch nenašli zlepšujúce riešenie.

Príklad 3.10 *Uvažujme problém* k-min-tree*: V neorientovanom ohodnotenom grafe $G = (V, E, c)$, kde c je funkcia, ktorá každej hrane určuje cenu, máme identifikovať podstrom minimálnej ceny s k hranami.*

1. greedy Metódou nájdeme nejaké prípustné riešenie: začneme s hranou minimálnej ceny a pridávame postupne hrany s monotónne rastúcou cenou tak, že strom narastá. Hodnotu aj strom si pamätáme ako najlepšie riešenie $(T^*, c(T^*))$.
2. Treba určiť susednosť, resp.množinu povolených operácií. Vychádzame z operácie *swap*, ktorá robí výmenu hrany zo stromu za hranu nestromovú, pričom do operácie vstupujú len "netabuizované" hrany. Myslíme tým hrany, ktorých stav nie je Tabu-aktívny, resp. sú mimo množiny TABU. (Mohli by sme uvažovať aj iné kritérium, keď by swap bol zakázaný len vtedy, keď sú obe hrany tabu-aktívne.)

   Uvažuje sa statický swap-keď sa nemení množina vrcholov stromu a dynamický, keď množinu vrcholov meniť môžme.
3. Ak najlepší swap vedie k zníženiu ceny, nové riešenie akceptujeme a ak je jeho cena lepšia ako doteraz najlepšia, zaktualizujeme $(T^*, c(T^*))$. Ak nedošlo k vylepšeniu momentálneho riešenia, nastavíme hrany, ktoré sa swapu zúčastnili, ako tabu-aktívne s vhodnouy dĺžkou trvania ℓ. Pre hranu, ktorú sme odstraňovali, hodnota ℓ určuje, koľko nasledujúcich kôl ju nemôžme vložiť, pre tú, ktorú sme vložili, ℓ označuje počet kôl, ktoré má v uvažovanom strome zotrvať. Tieto hodnoty môžu byť rôzne[15]
4. Vhodne treba nastaviť kritérium ukončenia

3.5.4 Problém prideľovanie úloh: využitie branch-and-bound a simulovaného žíhania

Uvažujme Job-shop-scheduling:

vstup

m strojov $M_1, \ldots, M_m$

n zadaní(jobov) $J_1, \ldots, J_n$; predpokladáme, že job musí prejsť cez príslušné stroje v presne určenom poradí

p(i,j) - trvanie spracovania úlohy j na stroji i. Predpokladáme, že hodnoty $p(i,j)$ poznáme dopredu.

úloha

minimalizovať dĺžku C_{max} celkového spracovania

Problém formulujeme grafom $G(N, X, Y)$, kde

N sú operácie, ktoré treba vykonať $\{(1,2),(1,4),\ldots\}$

X sú hrany popisujúce poradie realizácie operácií pre jednotlivé joby

$(i,j) \rightarrow (k,j) \in X$ hovorí, že v jobe j spracovanie na stroji i predchádza spracovanie na stroji k

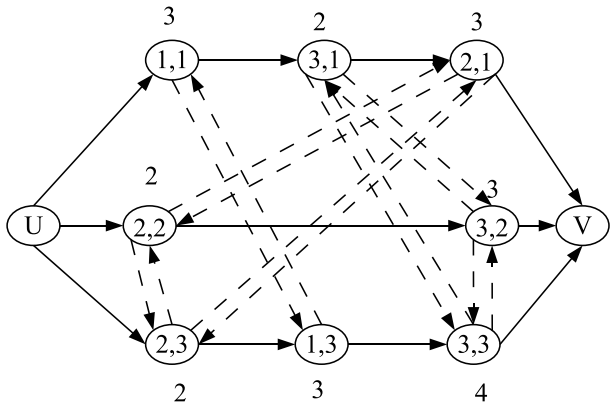
Pridávame hrany $U \longrightarrow$ do prvej operácie každého jobu, z poslednej operácie každého jobu $\longrightarrow V$

Y sú hrany, ktoré popisujú konflikty jednotlivých operácií na strojoch (nemôžu ísť dve operácie na jednom stroji naraz)

$(i,j) \dashrightarrow (i,\ell), (i,\ell) \dashrightarrow (i,j) \in Y$, hovoria, že úlohy j, ℓ nemôžu ísť na stroji i naraz

[15] Hodnote ℓ sa hovorí tabu tenure

job	postupnosť strojov	čas spracovania
1	$1, 3, 2$	$p(1,1) = 3, p(3,1) = 2, p(2,1) = 3$
2	$2, 3$	$p(2,2) = 2, p(3,2) = 3$
3	$2, 1, 3$	$p(2,3) = 2, p(1,3) = 3, p(3,3) = 4$



Výberom rozumieme taký podgraf $D \subseteq Y$, v ktorom je z každej dvojice (i,j) $\dashrightarrow$ $(i,\ell), (i,\ell) \dashrightarrow (i,j)$ práve jedna hrana. Ak D spolu s hranami X je acyklický, tak máme *prípustný výber*, presnejšie jeho grafickú reprezentáciu.

Z každého prípustného výberu vieme určiť rozvrh, a teda prípustné riešenie.[16] Keď máme nejaký prípustný výber, tak najdlhšia $U-V$ cesta určuje makespan a hovoríme jej *kritická*. Pod dĺžkou pritom rozumieme súčet časov spracovania vo vrcholoch na nej.

Jedným prístupom je generovanie všetkých *aktívnych rozvrhov* a z nich potom vyberieme najlepšie. Aktívny rozvrh je taký rozvrh, ktorý nemôžme vylepšiť—žiadna operácia nemôže byť dopočítaná skôr bez toho, aby inú zdržala—alternáciou operácií na strojoch. Všetky aktívne rozvrhy môžme generovať algoritmom Aktívny rozvrh: *aktívny rozvrh*

Rozvetvenie v kroku 3 vyberá priradenie prípustnej operácie na stroj i^*. Počet vetiev je rovný počtu prípustných operácií v Ω'; konkrétny výber vlastne fixuje niektoré výberové Y-hrany, ktoré pribúdajú do D'.

Ako získavame dolný odhad? Označme $G(D')$ ten kombinovaný graf pre aktuálny výber D. Jednoduchý dolný odhad LB je cena kritickej cesty.
Vieme získať aj vyšší dolný odhad. Uvažujme dopočítanie úlohy z pohľadu jedného stroja, keď ostatným "dovolíme", aby počítali viac úloh naraz. Odpovedá to situácii, že existujú dvojice operácií, pre ktoré sme nezafixovali/nevybrali jednu z dvojice odpovedajúcich Y-hrán. Uvedomme si, že tento predpoklad hodnotu dolného (aj horného) odhadu vlastne znižuje. Zaujíma nas maximálne spomalenie L_{max}. K hodnote jednoduchého odhadu LB potom pripočítame (maximalizované cez stroje) cenu dokončenia na izolovanom stroji.

1. vypočítaj dolný odhad LB ako cenu kritickej cesty v $G(D')$
2. pre všetky operácie (i,j) na stroji i vypočítaj $r_{i,j}$, čo je ekvivalentné najdlhšej $U \longrightarrow (i,j)$ ceste v $G(D')$

[16] Napíšte program, ktorý to spraví.

Algoritmus 12 Aktívny rozvrh

Ω množina operácií, ktorých predchodcovia už sú zaradení; $\Omega' \subseteq \Omega$
$r_{i,j}$ najskorší možný začiatok pre operáciu (i,j)
D' graf, ktorý odpovedá priebežnému rozvrhu

1. inicializácia:
   Ω :=prvá operácia pre každý job
   $r_{i,j} := 0 \forall (i,j) \in \Omega$
   D' obsahuje všetky X-hrany a žiadne Y-hrany.
2. Výber stroja:
   vypočítaj $t(\Omega)$ aktuálneho rozvrhu
   $$t(\Omega) = \min\{r_{i,j} + p(i,j) | (i,j) \in \Omega\}$$
   a index stroja i^*, ktorý tú hodnotu minimalizuje
3. branching
   $\Omega' = \{(i^*, j) | r_{i^*,j} < t(\Omega)\}$
   - $\forall (i^*, j) \in \Omega'$ rozšír čiastočný rozvrh o zaradenie (i^*, j) na stroj i^* a súčasne odstráň (i^*, j) z Ω.
   - zaraď nasledovníka (i^+, j) práve pridelenej operácie (i^*, j) do Ω a vypočítaj $r_{i^+,j}$; túto hodnotu získame ako hodnotu najdlhšej $U - (i^+, j)$ cesty v aktuálnom $G(D')$
   - goto 2

3. vypočítaj minimálny čas $\Delta_{i,j}$ medzi začiatkom realizácie operácie (i,j) a koncom rozvrhu; je to najdlhšia $(i,j) \longrightarrow V$ cesta v $G(D')$. Nutný čas $d_{i,j}$ je potom

$$d_{i,j} = LB - \Delta_{i,j} + p(i,j)$$

4. pre každý stroj zvlášť vypočítaj minimalizované oneskorenie L_j a potom

$$LB^{new} = LB + \max L_j$$

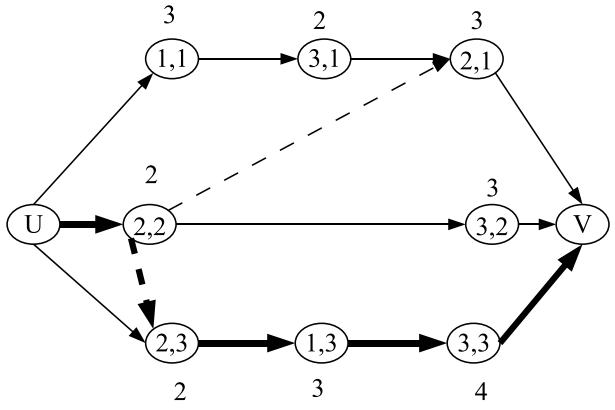
Tento dolný odhad používame ako usmernenie, kam máme pokračovať.

Riešenie:

úroveň 0 $\Omega = \{(1,1), (2,2), (2,3)\}$
$t(\Omega) = 2$; $i^* = 2$
$\Omega' = \{(2,2), (2,3)\}$

Koreň bude mať 2 synov-operácie (2,2) s (2,3). Každému synovi vypočítame odpovedajúci graf D', ktorý vznikne z grafu tvoreného X-hranami pridaním odpovedajúcich Y-hrán.

úroveň 1 vrchol (2,2)– na stroj $M2$ sme zaradili operáciu $(2,2)$. Pridáme preto do D' hrany $(2,2)$ --→ $(2,1)$ a $(2,2)$ --→ $(2,3)$.



Dolný odhad LB=l(U,(2,2),(2,3),(1,3),(3,3),V)=11.

Doriešime problém pre stroj M_i zvlášť, aby sme vylepšili dolný odhad. Príslušné údaje pre rozhodovanie

$job1$	$job3$
$r_{11}=0$	$r_{13}=4$
$\Delta_{11}=8$	$\Delta_{13}=7$
$d_{11}=6$	$d_{13}=7$

Rozvrh na M_1 bude takýto: (1,1) začne v čase 0, skončí 3; (1,3) začne v čase 4, skončí 7. V oboch prípadoch je to skôr ako príslušný nutný čas, preto $L_1=0$.

$job1$	$job2$	$job3$
$r_{21}=5$	$r_{22}=0$	$r_{23}=2$
$\Delta_{21}=3$	$\Delta_{22}=11$	$\Delta_{23}=9$
$d_{21}=11$	$d_{22}=2$	$d_{23}=4$

Rozvrh na M_2 bude takýto: (2,1) začne v čase 5, skončí 8; (2,2) začne v čase 0, skončí 2, (2,3) začne v 2, skončí v 4. Všetky operácie skončia skôr ako príslušný nutný čas, preto $L_2=0$.

$job1$	$job2$	$job3$
$r_{31}=5$	$r_{32}=2$	$r_{33}=7$
$\Delta_{31}=5$	$\Delta_{32}=3$	$\Delta_{33}=4$
$d_{31}=8$	$d_{32}=11$	$d_{33}=11$

Rozvrh na M_3 bude takýto: (3,1) začne v čase 3, skončí 5; (3,2) začne v čase 5, skončí 7, (3,3) začne v 7, skončí v 11. Všetky operácie skončia skôr ako príslušný nutný čas, preto $L_3=0$.

Nový dolný odhad teda je

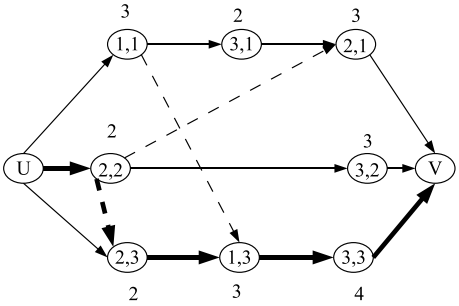
$$LB^{new}=LB+\max L_i=11$$

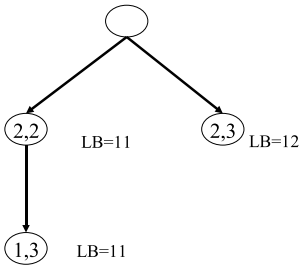
Uvedomme si, že sme dopočítali operáciám začiatky $r_{i,j}$, ktoré platia pre výpočet ďalšej úrovne v tejto vetve. Pokračujeme teda vo výpočte tejto vetvy. Odstránime operáciu $(2,2)$ z Ω a zaradíme do Ω jej X-nasledovníka $(3,2)$.

$\Omega=\{(1,1),(3,2),(2,3)\}$
$t(\Omega)=3$; $i^*=1$
$\Omega'=\{(1,1)\}$

Vrchol (2,2) má jedného syna-(1,1).

úroveň 2 Zaradenie operácie (1,1) na stroj M_1 si vynucuje pridanie Y-hrany $(1,1)\dashrightarrow(1,3)$ do D'





Dolný odhad LB=l(U,(2,2),(2,3),(1,3),(3,3),V)=11.....

simulované žíhanie

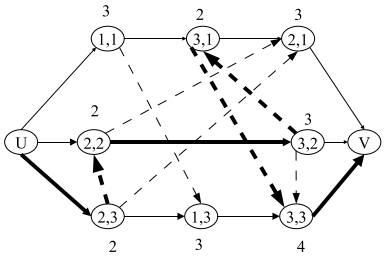
Úloha: Presvedčte sa, že hodnota LB pre vrchol (2,3) na úrovni 1 je naozaj 12

Riešme ten istý problém simulovaným žíhaním. Nastavíme počiatočnú teplotu $T = 100$, parameter α, ktorý modifikuje teplotu, nastavíme $\alpha = 0.95$ a hranicu pravdepodobnosti na akceptovanie $\epsilon = 0.98$.

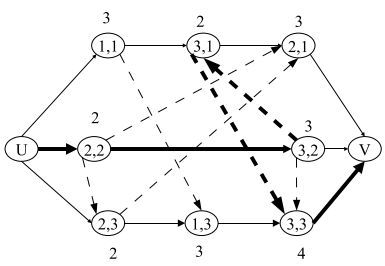
Algoritmus 13 Simulované žíhanie

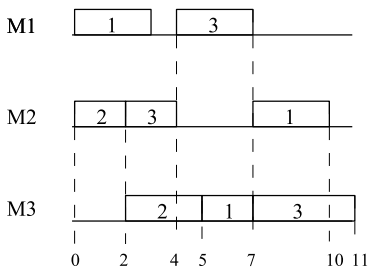
1. zvoľ dostatočne veľkú teplotu $T = T_0$
2. zvoľ vhodný parameter redukcie α, $0 < \alpha < 1$
3. nastav v grafe D prípustné riešenie
4. nastav počiatočnú energiu $E(i)$ako cenu kritickej cesty
5. kým je T v rozumnom rozsahu opakuj
   - zmeň orientáciu niektorej Y-hrany na kritickej ceste
   - vypočítaj novú energiu $E(j)$ a odpovedajúcu zmenu ΔE
   - ak $\Delta E < 0$, akceptuj nový stav
   - ak $\Delta E > 0$, akceptuj nový stav ak $e^{-\Delta E/T} > \epsilon$
   - $T \leftarrow T \cdot \alpha$

i=0 Nastavíme prípustné riešenie a v ňom nájdeme kritickú cestu. Jej cena určuje energiu $E(0) = 13$

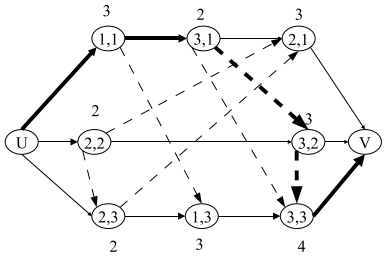


i=1 nové riešenie vznikne otočením orientácie jednej z Y-hrán na kritickej ceste; napr $(2,3) \dashrightarrow (2,2)$ zmeníme na $(2,2) \dashrightarrow (2,3)$. Kritická cesta sa zmení, jej cena určuje novú energiu $E(1) = 11$. $T \leftarrow \alpha \cdot T = 95$. Keďže $\Delta E = 11 - 13 = -2 < 0$, túto zmenu akceptujeme:

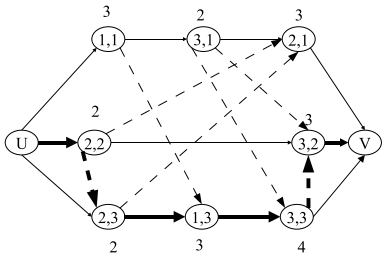




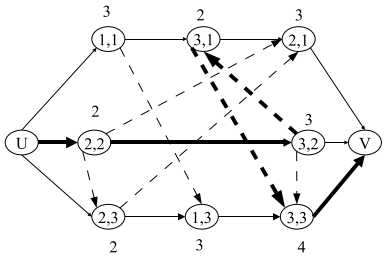
i=2 nové riešenie vznikne otočením orientácie jednej z Y-hrán na kritickej ceste; napr $(3,2)$ --→ $(3,1)$ zmeníme na $(3,1)$ --→ $(3,2)$. Kritická cesta sa zmení, jej cena určuje novú energiu $E(2) = 12$. $T \leftarrow \alpha \cdot T = 90.25$ Keďže $\Delta E = 12 - 11 = 1 > 0$, o akceptovaní zmeny rozhoduje hodnota $Pr(akc) = e^{-\Delta E/T} = e^{-1/90.25} = 0.9890 > \epsilon$ túto zmenu akceptujeme.



i=3 nové riešenie vznikne otočením orientácie jednej z Y-hrán na kritickej ceste; napr $(3,2)$ --→ $(3,3)$ zmeníme na $(3,3)$ --→ $(3,2)$. Kritická cesta sa zmení, jej cena určuje novú energiu $E(2) = 14$. $T \leftarrow \alpha \cdot T = 85.7375$ Keďže $\Delta E = 14 - 12 = 2 > 0$, o akceptovaní zmeny rozhoduje hodnota $Pr(akc) = e^{-\Delta E/T} = e^{-2/85.7375} = 0.9769 < \epsilon$ túto zmenu *neakceptujeme*.



Vrátime sa k situácii na konci kroku pre $i = 2$ a skúsime otočiť inú Y-hranu. Keďže to nejde, je koniec. Riešenie s minimálnou energiou je v úrovni 1



3.6 Relaxácia na úlohu lineárneho programovania

Všeobecná úloha lineárneho programovania je riešiteľná v polynomiálnom čase, zatiaľ čo 0-1, resp. celočíselná úloha LP sú NP-ťažké. Všetky typy týchto úloh majú ohraničenia $A \cdot X = b$, kde X je vektor premenných, matica A a vektor b sú koeficienty ohraničení; úlohou je minimalizovať funkciu $X \cdot c^T$. Uvedené tri typy sa líšia v definičnom obore vektora X. Základná úloha, riešiteľná v polynomiálnom čase, sa rieši nad reálnymi číslami. 0/1-LP sa rieši nad $\{0, 1\}$ a I-LP nad celými číslami, obe verzie sú NP-ťažké.

Významnosť úlohy LP spočíva aj v tom, že mnoho ťažkých optimalizačných úloh sa dá formulovať ako úloha LP. Dokonca aj v prípade, že získame 0/1-LP alebo I-LP, môžme využiť relaxáciu na reálne čísla a vhodným zaokrúhlením môžme získať kvalitné riešenie[17]
Metóda relaxácie na LP teda pozostáva z troch krokov:

Redukcia Vyjadríme vstup X do optimalizačného problému U ako vstup $I(x)$ do 0/1-LP alebo I-LP.

Relaxácia Relaxujeme od podmienok celočíselnosti a vypočítame optimálne riešenie α všeobecnej úlohy LP so vstupom $I(x)$

Riešenie pôvodného problému Použijeme α na výpočet optimálneho riešenia pôvodnej úlohy, resp. dostatočne kvalitného prípustného riešenia.

Pre danú postupnosť reálnych čísel $a_1, \ldots, a_n$ a postupnosť premenných $x_1, \ldots, x_n$ je *lineárna funkcia* f definovaná predpisom

$$f(x_1, \ldots, x_n) = a_1 x_1 + \cdots + a_n x_n = \sum_{i=1}^{n} a_i x_i$$

Ak b je reálne číslo a f je lineárna funkcia, tak rovnica

$$f(x_1, \ldots, x_n) = b$$

sa nazýva *lineárnou rovnosťou* a nerovnice

$$f(x_1, \ldots, x_n) \leq b, \quad f(x_1, \ldots, x_n) \geq b$$

sa nazývajú *lineárne nerovnosti.* Termínom lineárne oraničenie označujeme lineárne rovnosti a nerovnosti.

LP

Definícia 3.10 *Vstupom* úlohy lineárneho programovania *je lineárna funkcia (účelová funkcia) a sústava lineárnych ohraničení nad premennými $x_1, \ldots, x_n$. Cieľom je maximalizovať alebo minimalizovať účelovú funkciu pri splnení daných lineárnych ohraničení.*

I-LP

V úlohe celočíselného lineárneho programovania *naviac požadujeme, aby premenné nadobúdali len celočíselné hodnoty.*

0/1-LP

V úlohe 0/1-lineárneho programovania *nadobúdajú premenné len hodnoty 0 a 1.*

Každé riešenie úlohy LP, ktoré spĺňa všetky lineárne ohraničenia, sa nazýva *prípustným riešením* úlohy LP. Prípustné riešenie, ktoré optimalizuje hodnotu účelovej funkcie, sa nazýva *optimálnym riešením* úlohy LP.

Pri riešení úlohy LP sa používajú viaceré algoritmy:

[17] Viac o tomto prístupe v časti o aproximačných algoritmoch.

- simplexový algoritmu, ktorý má síce v najhoršom prípade exponenciálnu zložitosť, v praxi sa však chová veľmi dobre
- elipsoidový algoritmus, ktorý má síce polynomiálnu zložitosť, v praxi je však kvôli vysokým konštantám pomalý
- metódy vnútorných bodov, ktoré sú pre veľké vstupné inštancie rovnako rýchle, resp. aj rýchlejšie, ako simplexový algoritmus

3.6.1 Príklady redukcie

Problém najkratšej cesty

Vstup: orientovaný graf $G = (V, E)$ s ohodnotením hrán w;
vrcholy $s, t \in V$
Cieľ: dĺžka najkratšej $s - t$ cesty

najkratšia cesta ako LP

Zavedieme pre každý vrchol $v \in V$ premennú x_v a formulujeme úlohu LP pre problém najkratšej cesty ako

$$\min x_t$$

pri ohraničeniach

$$x_u \leq x_v + w(u, v) \forall (u, v) \in E, \quad x_s = 0$$

poriadne dorobtoky

Problém batohu Každému objektu $i = 1, \ldots, n$ priradíme premennú x_i, pričom hodnota 1 odpovedá situácii, že príslušný objekt je obsiahnutý v riešení. Formulujme úlohu ako úlohu 1/0-LP:

batoh ako 0/1-LP

maximalizovať	$\sum_{i=1}^{n} p_i x_i$
pri ohraničeniach	$\sum_{i=1}^{n} w_i x_i \leq b$

Minimálne vrcholové pokrytie Zopakujme si: vstupom problému je ohodnotený graf $G = (V, E, c), c : V \to \mathbb{N} - 0$. Cieľom je nájsť vrcholové pokrytie $S \subset V$: $\forall (u, v) \in E[u \in S \vee v \in S]$, ktorého cena $\sum_{v \in S} c(v)$ je minimálna.

MVC ako 0/1-LP

Pri formulácii úlohy ako úlohy LP využijeme reprezentáciu množiny S charakteristickým vektorom $X_S = (x_1, \ldots, x_n) \in \{0, 1\}^n$, pričom

$$x_i = 1 \iff v_i \in S$$

minimalizovať	$\sum_{i=1}^{n} c(v_i) x_i$
pri ohraničeniach	$x_i + x_j \geq 1 \quad (v_i, v_j) \in E$
	$x_i \leq 1, x_i \geq 0$

3.6.2 Vlastnosti úlohy lineáreho programovania

Hovoríme, že úloha je v *štandardnom tvare*, ak všetky lineárne ohraničenia majú tvar *rovností* a hodnoty premenných sú nezáporné: úlohou je minimalizovať

$$X \cdot c^T = \sum_{i=1}^{n} c_i x_i$$

pri ohraničeniach

$$\begin{array}{rl} A \cdot X = b & \sum_{i=1}^{n} a_{ji} x_i = b_j, j = 1, \ldots, m \\ x_i \geq 0 & i = 1, \ldots, n \end{array}$$

Hovoríme, že úloha je v *kanonickom tvare*, ak všetky lineárne ohraničenia majú tvar *nerovností* a hodnoty premenných sú nezáporné

$$\begin{array}{rl} A \cdot X \geq b & \sum_{i=1}^{n} a_{ji} x_i \geq b_j, j = 1, \ldots, m \\ x_i \geq 0 & i = 1, \ldots, n \end{array}$$

Každá úloha sa dá previesť z kanonického do štandardného tvaru a naopak. Pri prechode od kanonického tvaru je základom nahradenie nerovností rovnosťami za použitia dodatočnej premennej[18].

$$\sum_{i=1}^{n} a_{ji} x_i \geq b_j \quad \rightsquigarrow \quad \sum_{i=1}^{n} a_{ji} x_i - s_j = b_j, \; s_j \geq 0$$

Príklad 3.11 *Uvažujme úlohu LP*

$$\begin{array}{lrcl} \textit{minimalizovať} & 7x_1 + x_2 + 5x_3 & & \\ \textit{pri ohraničeniach} & x_1 - x_2 + 3x_3 & \geq & 10 \\ & 5x_1 + 2x_2 - x_3 & \geq & 6 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Označme z^* minimálnu hodnotu účelovej funkcie. Čo vieme povedať o hodnote z^*?

horný odhad na z^*

Keďže máme minimalizačnú úlohu, každé prípustné riešenie poskytuje horný odhad. Ak máme nejakú hodnotu α a chceme vedieť, či $z^* \leq \alpha$, tak existencia vhodného prípustného riešenia je certifikátom, ktorý potvrdzuje kladnú odpoveď. [19]

Dolný odhad na optimálnu hodnotu účelovej funkcie z^* môžme získať z lineárnych nerovností. Keďže hodnoty premenných sú nezáporné, možme na základe porovnania koeficientov pri jednotlivých premenných písať

$$7x_1 + x_2 + 5x_3 \geq x_1 - x_2 + 3x_3 \geq 10, \text{ resp.}$$
$$7x_1 + x_2 + 5x_3 \geq (5x_1 + 2x_2 - x_3) + (x_1 - x_2 + 3x_3) \geq 16$$

Všeobecným "návodom" na hľadanie dolného odhadu hodnoty účelovej funkcie je brať nezáporné násobky jednotlivých nerovností tak, aby koeficienty jednotlivých premenných v získanom súčte neprevyšovali odpovedajúce koeficienty v účelovej funkcii. Súčet pravých strán je potom dolným odhadom. Snažíme sa pritom, aby získaný dolný odhad bol čo najvyšší. Dostali sme sa tak k maximalizačnej úlohe LP:

$$\begin{array}{lrcl} \text{maximalizovať} & 10y_1 + 6y_2 & & \\ \text{pri ohraničeniach} & y_1 + 5y_2 & \leq & 7 \\ & -y_1 + 2y_2 & \leq & 1 \\ & 3y_1 - y_2 & \leq & 5 \\ & y_1, y_2 & \geq & 0 \end{array}$$

primárna a duálna úloha LP

Pôvodnú úlohu voláme *primárnou úlohou* LP, novoskonštruovanú úlohu nazývame *duálnou úlohou* LP. Analogicky môžme postupovať, keď pôvodná úloha je maximalizačná. Vyskúšajte si.

Formálnejšie:

[18] surplus

[19] nájdite certifikát pre potvrdenie hypotézy, že $z^* \leq 30$ v príklade 3.11

Primárna		
minimalizovať	$\sum_{j=1}^{n} c_j x_j$	
pri ohraničeniach	$\sum_{j=1}^{n} a_{ij} x_j \geq b_i$	$i = 1, \ldots, m$
	$x_j \geq 0$	$j = 1, \ldots, n$

Duálna		
maximalizovať	$\sum_{i=1}^{m} b_i y_i$	
pri ohraničeniach	$\sum_{i=1}^{m} a_{ij} y_i \leq c_j$	$j = 1, \ldots, n$
$y_i \geq 0$	$i = 1, \ldots, m$	

O vzťahu primánej a duálnej úlohy hovorí nasledovná veta.

Veta 3.16 (Slabá veta o dualite) *Ak* $\mathbf{x} = (x_1, \ldots, x_n)$ *a* $\mathbf{y} = (y_1, \ldots, y_m)$ *sú prípustné riešenia primárnej a duálnej úlohy, tak*

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$$

Dôkaz: Dôkaz je veľmi jednoduchý. Keďže $\mathbf{y}$ je prípustným riešením a hodnoty x_j sú nezáporné, môžme využiť ohraničenia pre c_j z duálnej úlohy

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j$$

Podobne pre prípustné riešenie $\mathbf{x}$ a nezáporné hodnoty y_i

$$\sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i \geq \sum_{i=1}^{m} b_i y_i$$

Tvrdenie vyplýva z pozorovania

$$\sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} x_j \right) y_i$$

□

Z dôkazu slabej vety o dualite môžme odvodiť komplementárne podmienky.

Veta 3.17 (Komplementárne podmienky) *Nech* $\mathbf{x} = (x_1, \ldots, x_n)$ *a* $\mathbf{y} = (y_1, \ldots, y_m)$ *sú prípustné riešenia primárnej a duálnej úlohy LP. Potom* $\mathbf{x}$ *a* $\mathbf{y}$ *sú optimálne práve vtedy, keď sú splnené nasledujúce podmienky:*

Primárne komplementárne podmienky *Pre každé* $1 \leq j \leq n$: *buď* $x_j = 0$ *alebo* $\sum_{i=1}^{m} a_{ij} y_i = c_j$

Duálne komplementárne podmienky *Pre každé* $1 \leq i \leq m$: *buď* $y_i = 0$ *alebo* $\sum_{j=1}^{n} a_{ij} x_j = b_i$

Veta 3.18 (Silná veta o dualite) *Primárna úloha LP má optimálne riešenie práve vtedy, ak k nej duálna úloha má optimálne riešenie. Navyše, ak* $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$ *a* $\mathbf{y}^* = (y_1^*, \ldots, y_m^*)$ *sú optimálne riešenia primárnej a duálnej úlohy, tak*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

Všimnime si, že toto je naozaj min-max vzťah, keďže jeden program je minimalizačný problém a druhý maximalizačný problém. Prípustné riešenie primárnej(duálnej) úlohy poskytuje YES-certifikát(No-certifikát) na otázku Je optimálna hodnota menšia alebo rovná ako α? Vlastne to ukazuje, že $LP \in NP \bigcap co - NP$.

Kapitola 4

Optimalizačné problémy, aproximačné algoritmy a triedy zložitosti

V tejto časti sa budeme venovať riešeniu optimalizačných úloh. Väčšinou spôjde o také optimalizačné problémy, ktorých rozhodovacie verzie sú NP-úplné. To znamená, že nemožno očakávať existenciu efektívnych algoritmov na ich riešenie. V tejto situácii sa upustenie od požiadavky na optimalitu pri zachovaní požiadavky na riešiteľnosť v polynomiálnom čase javí byť rozumným východiskom.

Najskôr si všimneme vzťah rozhodovacej verzie a konštrukčnej verzie optimalizačného problému, potom si vysvetlíme prístupy k hodnoteniu kvality aproximačných algoritmov. Pod kvalitou sa tu myslí tzv. aproximačný pomer, teda pomer optimálneho a algoritmom garantovaného riešenia. Zadefinujeme triedy zložitosti optimalizačných problémov APX, PTAS, FPTAS, kde sledovanou mierou zložitosti je kvalita získaného algoritmu a ukážeme príklady problémov z jednotlivých tried. V nasledujúcej kapitole sa potom budeme venovať návrhu aproximačných algoritmov pre viacero konkrétnych optimalizačných problémov.

Zopakujme definíciu pojmu optimalizačný problém a triedy NPO, ktorá je analógiou NP.

Definícia 4.1 Optimalizačný problém $\mathcal{P}$ *je štvorica* $(I, Sol, m, ciel)$*, kde*

I	*je množina vstupných inštancií*
Sol	*je funkcia, ktorá každej vstupnej inštancii* $x \in I$ *priradí množinu* $Sol(x)$ prípustných *riešení*
m	*je* účelová *funkcia/miera/cena, ktorá* $\forall x \in I, y \in Sol(x)$ *vráti cenu* $m(x, y)$[1]
$ciel$	*je alebo* min *alebo* max

Optimálna hodnota $opt(x) = ciel\{m(x, y) \mid y \in Sol(x)\}$*. Budeme ju označovať aj* $m^*(x)$*,* $m(x, y^*(x))$*,* $m^*(x, y)$*.* $Sol^*(x)$ *bude označovať množinu optimálnych riešení k vstupu* x*.*

Definícia 4.2 *Triedu* NPO *tvoria optimalizačné problémy* $\mathcal{P}$=$(I, Sol, m, ciel)$ *pre ktoré:*

- $\exists$ *polynóm* p *taký, že* $\forall x \in I, y \in Sol(x) : \ |y| \leq p(|x|)$
- *problém príslušnosti* $y \in Sol(x) \in$ P
- *pre dané* x, y *trvá výpočet* $m(x, y)$ *polynomiálny čas*

4.1 Optimalizačné vs. rozhodovacie problémy

Riešiť optimalizačný problém znamená nájsť také prípustné riešenie, ktoré optimalizuje účelovú funkciu. V tejto súvislosti celkom prirodzene vznikajú viaceré otázky: je optimálna hodnota menšia/väčšia/rovná nejakej konkrétnej hodnote? Ak viem, že m^* je optimálna hodnota, ako nájdem to riešenie α, ktoré ju dosahuje? Aký je vzťah týchto otázok/problémov navzájom?

Nech $\mathcal{P}$ je optimalizačný problém.

riešenie

Konštrukčný problém $\mathcal{P}_C$ je problém, keď k vstupnej inštancii $x \in I$ chceme vypočítať optimálne riešenie $y^*(x) \in Sol(x)$ aj jeho hodnotu $m^*(x)$

hodnota

Hodnotový problém $\mathcal{P}_E$ je taká verzia, keď nás pre vstupnú inštanciu $x \in I$ zaujíma len hodnota optimálneho riešenia $m^*(x)$

$m(x,y)\ ?\ k$

Rozhodovací problém/underlying language $\mathcal{P}_D$ je problém, keď pre vstupnú inštanciu $x \in I$ a hodnotu $k \in \mathbb{N}$ máme rozhodnúť, či

$$\exists y \in Sol(x) \begin{cases} m(x,y) \leq k, & \text{pre minimalizačný problém} \\ \\ m(x,y) \geq k, & \text{pre maximalizačný problém} \end{cases}$$

Začnime jednoduchým pozorovaním.

Fakt 4.1 *Ak optimalizačný problém $\mathcal{P} \in$ NPO tak $\mathcal{P}_D \in$ NP*

Nedeterministický TS so vstupom (x,k) uhádne y, overí, že $y \in Sol(x)$, vypočíta $m(x,y)$ a porovná $k?m(x,y)$. ◇

Kedy hovoríme, že je optimalizačný problém ťažký? Pri definícii využijeme Turingovskú redukovateľnosť[2] (budeme značiť $\leq_T^p$) a pojem Turingovho stroja s pomocou. Turingov stroj s pomocou–značíme T^g–je TS, ktorý má jednu špeciálnu pásku a dva špeciálne stavy $q_?, q_!$. Keď je v stave $q_?$ a na špeciálnej(query) páske má napísané slovo x, TS T^g v jednom kroku prejde do stavu $q_!$, pričom slovo x na query páske sa nahradí výsledkom $g(x)$.

Definícia 4.3 *Nech $\mathcal{P}_i=(I_i, Sol_i, m_i, ciel_i), i = 1,2$, sú dva optimalizačné problémy. Nech $g(x) \in Sol_2^*(x)$. Hovoríme, že problém $\mathcal{P}_1$ je výpočtovo redukovateľný na optimalizačný problém $\mathcal{P}_2$, značíme $\mathcal{P}_1 \leq_T^p \mathcal{P}_2$, ak existuje deterministický TS s pomocou g, ktorý v polynomiálnom čase rieši $\mathcal{P}_1$.*
Ak $\mathcal{P}_1 \leq_T^p \mathcal{P}_2$ a $\mathcal{P}_2 \leq_T^p \mathcal{P}_1$, potom sú problémy $\mathcal{P}_1$ a $\mathcal{P}_2$ výpočtovo ekvivalentné, čo označujeme $\mathcal{P}_1 \equiv_T^p \mathcal{P}_2$

Inými slovami, problém $\mathcal{P}_1$ by sme vedeli optimálne vypočítať v polynomiálnom čase, ak by sme vedeli v konštantnom čase získať optimálne riešenie optimalizačného problému $\mathcal{P}_2$. Polynomiálny čas teda spotrebovávame na transformáciu vstupu x_1 pre problém $\mathcal{P}_1$ na vstup x_2 pre problém $\mathcal{P}_2$ a na transformáciu optimálneho riešenia y_2^* problému $\mathcal{P}_2(x_2)$ na optimálne riešenie y_1^* problému $\mathcal{P}_1(x_1)$.

Pojem výpočtovej redukovateľnosti hrá pri definícii NPO-ťažkého problému analogickú úlohu ako pojem polynomiálnej redukovateľnosti pri definícii NP-ťažkého problému.

[2] Turingovská redukovateľnosť: Nech A je problém, ktorý počíta funkciu $g : I_A \to Sol_A$. Problém A $\leq_T^p$ B, ak existuje polynomiálny algoritmus na riešenie A s orákulom B (pre vstup $x \in I_B$ vráti $f(x) \in Sol_B$)

Definícia 4.4 *Optimalizačný problém $\mathcal{P} = (I, Sol, m, ciel)$ je* NPO*- ťažký, ak pre každý* NPO *optimalizačný problém $\mathcal{P}'$ platí $\mathcal{P}' \leq_T^p \mathcal{P}$*

Definícia 4.5 *Hovoríme, že optimalizačný problém $\mathcal{P}$ je* NP-ťažký*, ak pre každý rozhodovací problém $\mathcal{P}' \in NP$ je $\mathcal{P}' \leq_T^p \mathcal{P}$.*

Ľahko vidno:

Veta 4.2 *Nech $\mathcal{P} \in$ NPO. Ak rozhodovacia verzia $\mathcal{P}_\mathcal{D} \in NPU$, tak $\mathcal{P}$ je* NP*-ťažký.*

Veta 4.3 $P \neq NP \Rightarrow PO \neq NPO$

O vzťahu rozhodovacej, hodnotovej a konštrukčnej verzie optimalizačného problému hovorí nasledujúca veta. Prirodzene je konštrukčná verzia najťažšia - pomocou skonštuovaného riešenia vieme zodpovedať rozhodovaciu aj hodnotovú verziu problému. Ukazuje sa tiež, že rozhodovacia a hodnotová verzia sú rovnako ťažké.

Veta 4.4 $\forall \mathcal{P} \in NPO\ \mathcal{P}_D \equiv_T^p \mathcal{P}_E \leq_T^p \mathcal{P}_C$

Dôkaz:

Chceme riešiť rozhodovaciu verziu problému pomocou hodnotovej; pre vstup (x, k) nás zaujíma, či $m^*(x)$ je menšie alebo väčšie ako k, pričom pri riešení môžme využívať informáciu o optimálnej hodnote. Potrebný Turingov stroj T s pomocou preto môžme skonštruovať nasledovne: $\mathcal{P}_\mathbf{D} \leq_\mathbf{T}^\mathbf{p} \mathcal{P}_\mathbf{E}$

- T používa funkciu g takú, že pre vstup x vráti optimálnu hodnotu $m^*(x)$
- pre vstup (x, k) napíše T na query pásku x a prejde do stavu $q_?$, v ktorom komunikuje s orákulom. Následne sa na páske objaví hodnota $g(x)$, teda hodnota optimálneho riešenia $m^*(x)$.
- T porovná $g(x) = m^*(x) \overset{?}{\leq} k$ a správne odpovie.

Čas výpočtu stroja T je polynomiálny.

Ak máme prístup k optimálnej hodnote aj optimálnemu riešeniu, optimálna hodnota nie je problém. Stroj s pomocou dostane vstup x, napíše na query pásku dotaz x, orákulum vráti $(m^*(x), y^*(x))$, z čoho T oddelí $m^*(x)$. Čas je triviálne polynomiálny. $\mathcal{P}_\mathbf{E} \leq_\mathbf{T}^\mathbf{p} \mathcal{P}_\mathbf{C}$

Keďže náš problém je z NPO, vieme dĺžku zápisu hodnoty $m(x, y)$ zhora ohraničiť časom, čiže hodnotou $p(|x|)$. To ale znamená, že optimálna hodnota $m^*(x)$ je zhora ohraničená hodnotou $2^{p(|x|)}$. Keď poznáme interval, v ktorom sa hodnota nachádza, môžme ju využitím dotazov na orákulum rozhodujúce $k \leq m^*(x)$ získať binárnym vyhľadávaním. Robíme teda nanajvýš $\log 2^{p(|x|)} = O(p(|x|))$ dotazov na orákulum, preto je celkový čas polynomiálny. $\mathcal{P}_\mathbf{E} \leq_\mathbf{T}^\mathbf{p} \mathcal{P}_\mathbf{D}$

□

Veta 4.5 *Nech $\mathcal{P} \in NPO$ a nech jeho rozhodovacia verzia $\mathcal{P}_D \in$ NPU. Potom $\mathcal{P}_C \leq_T^p \mathcal{P}_D$*

Dôkaz: W.l.o.g. predpokladajme, že pôvodný problém bol maximalizačný. Konštrukciu spravíme v dvoch krokoch

- Najprv ukážeme, že existuje NP problém B_D taký, že $\mathcal{P}_C \leq_T^p B_D$
- Potom využijeme NP-úplnosť $\mathcal{P}_D$, z čoho dostaneme $B_D \leq_T^p \mathcal{P}_D$

Problém B sa od pôvodného problému $\mathcal{P}$ líši v cene. Zvolíme ju tak, aby sme z riešenia problému B vedeli ľahko získať riešenie problému $\mathcal{P}$. Presnejšie: nech p je polynóm, ktorý ohraničuje dĺžku prípustného riešenia problému $\mathcal{P}$, nech $\lambda(y)$ označuje poradie prípustného riešenia y v lexikografickom usporiadaní. Potom cenu prípustného riešenia problému B definujeme

$$m_B(x,y) = 2^{p(|x|)+1} m_{\mathcal{P}}(x,y) + \lambda(y) \quad \leftrightsquigarrow \quad \boxed{m_{\mathcal{P}}(x,y)}\underbrace{\boxed{0^*\lambda(y)}}_{p|x|}$$

Je zrejmé, že $|\lambda(y)| \leq p(|x|)$. Preto

$$\forall x \in I_B \forall y_1, y_2 \in Sol_B(x), y_1 \neq y_2 : \; m_B(x,y_1) \neq m_B(x,y_2)$$

To ale znamená, že vieme nájsť jednoznačné optimálne riešenie $y_B^*(x)$.

$\mathcal{P}_C \leq_T^p B_D$ *cez* $\mathcal{P}_C \leq_T^p B_E$

Keďže $B_D \equiv_T^p B_E$, budeme dokazovať $\mathcal{P}_C \leq_T^p B_E$. Stroj T s pomocou , ktorý konštruuje optimálne riešenie pre problém $\mathcal{P}$, používa ako orákulum funkciu, ktorá počíta hodnotu optimálneho riešenia problému B: $m_B(x,y^*) = 2^{p(|x|)+1} m_{\mathcal{P}}(x,y^*) + \lambda(y)$. Pracuje nasledovne:

- Napíše vstupné slovo $x \in I_{\mathcal{P}} = I_B$ na query pásku a dotazom na orákulum zistí hodnotu optimálneho riešenia $m_B(x,y^*) = 2^{p(|x|)+1} m_{\mathcal{P}}(x,y^*) + \lambda(y)$ problému B.
- Z hodnoty $m_B(x,y^*) = 2^{p(|x|)+1} m_{\mathcal{P}}(x,y^*) + \lambda(y^*)$ vypočíta $\lambda(y^*)$. Následne postupným generovaním $y \in SOL_B(x)$ nájde y^*, pre ktoré $\lambda(y^*) = \lambda(y)$.
- Pomocou B_D vieme simulovať B_E v polynomiálnom čase, preto $m_{\mathcal{P}}^*(x)$ vieme získať v polynomiálnom čase pomocou orákula pre B_D.

$B_D \leq_T^p \mathcal{P}_D$

Stačí simulovať orákulum B_D. Keďže $B_D \in \mathsf{NP}$ a $\mathcal{P}_D \in \mathsf{NPU}$, existuje redukcia $\mathcal{R}$ problému B_D na $\mathcal{P}_D$; $B_D \rightsquigarrow \mathcal{P}_D$. TS na riešenie B_D realizuje redukciu $\mathcal{R}$, napíše na dotazovaciu pásku $\mathcal{R}(x)$, využitím orákula, ktoré počíta $\mathcal{P}_D$, získa odpoveď $B_D(\mathcal{R}(x))$, z ktorej vie odpoveď na pôvodnú otázku.

□

Či existuje $\mathcal{P} \in NPO$, pre ktorý je $\mathcal{P}_C$ ťažší ako $\mathcal{P}_E$ sa nevie.

4.2 Aproximačné algoritmy a klasifikácia optimalizačných problémov

Ako sme už povedali, aproximačné algoritmy používame väčšinou pri optimalizačných úlohách, kde získanie optimálneho riešenia je neúmerne ťažké. Vtedy *upúšťame od presnosti* a snažíme sa o približné riešenie, ktoré ale *dostaneme rýchlo*. Začnime definíciami základných pojmov.

Definícia 4.6 Aproximačný algoritmus A *pre optimalizačný problém U je algoritmus* polynomiálnej *časovej zložitosti, ktorý pre každý prípad x problému U vypočíta nejaké prípustné riešenie* $A(x)$; $A(x) \in Sol(x)$

absolútna chyba $D(x,y) := |m^*(x) - m(x,y)|$

relatívna chyba

$$E_A(x,y) := \frac{D(x,y)}{\max\{m^*(x), m(x,y)\}} = \begin{cases} 1 - \frac{m^*(x)}{m(x,y)}, & \textit{pre minimalizačný problém;} \\ 1 - \frac{m(x,y)}{m^*(x)}, & \textit{pre maximalizačný problém.} \end{cases}$$

$$E_A(n) = \max\{E_A(x,y) | x \in L \cap (\Sigma_I)^n\}$$

aproximačný pomer

$$R_A(x,y) := \max\left\{\frac{m(x,A(x))}{(m^*(x)}, \frac{(m^*(x)}{m(x,A(x))}\right\} = \begin{cases} \frac{m(x,y)}{m^*(x)}, & \textit{pre minimalizačný problém;} \\ \frac{m^*(x)}{m(x,y)}, & \textit{pre maximalizačný problém.} \end{cases}$$

$$R_A(n) := \max\{R_A(x,y) | x \in L \cap (\Sigma_I)^n\}$$

Zrejme

$$R_A(x,y) = \frac{1}{1 - E_A(x,y)}$$

Čím bližšie je hodnota $R_A(x,y)$ k 1, resp. $E_A(x,y)$ k 0, tým kvalitnejšie riešenie(bližšie k optimálnemu) algoritmus vypočíta.

k-absolútny, ϵ-aproximačný a r-aproximačný algoritmus

Nech U je optimalizačný problém a nech A je algoritmus pre U, ktorý je polynomiálnej časovej zložitosti vzhľadom k dĺžke vstupnej inštancie $x \in L$. Ak kvalitu algoritmu A vyjadrujeme absolútnou chybou, hovoríme o k-absolútnom algoritme, ak relatívnou chybou, hovoríme o ϵ-aproximačnom algoritme a ak kvalitu vyjadrujeme aproximačným pomerom, hovoríme o r-aproximačnom algoritme:

Hovoríme, že algoritmus A je *k-absolútny* pre U ak existuje konštanta k taká, že $D_A(x, A(x)) \leq k$ pre každé $x \in L$.

Hovoríme, že algoritmus A je *ϵ-aproximačný* pre U ak pre nejaké reálne číslo $0 \leq \epsilon \leq 1$ platí $E_A(x, A(x)) \leq \epsilon$ pre každé $x \in L$.

Hovoríme, že algoritmus A je *r-aproximačný* pre U ak pre nejaké reálne číslo $r > 1$ platí $R_A(x, A(x)) \leq r$ pre každé $x \in L$.

Hovoríme, že algoritmus A je *$f(n)$-aproximačný* pre U ak pre funkciu $f : \mathbb{N} \to \mathbb{R}^+$ a pre každé $n \in \mathbb{N}$ je $R_A(n) \leq f(n)$.

APX

Definícia 4.7 *APX označuje triedu optimalizačných problémov, pre ktoré existuje polynomiálny r-aproximačný algoritmus (pre nejaké r)*

Môžme presnosť/veľkosť chyby považovať za ďalší z parametrov hľadaného algoritmu? Dokážeme vždy skonštruovať aproximačný algoritmus s požadovanou presnosťou? Uvedomme si, že samotný pojem aproximačného algoritmu v sebe zahŕňa požiadavku na polynomiálny čas!

Definícia 4.8 *Nech $U = (I, Sol, m, ciel)$ je optimalizačný problém. Hovoríme, že algoritmus A je* polynomiálna aproximačná schéma *(označujeme* PTAS*) pre U, ak $\forall(x, \epsilon) \in I \times R^+$ algoritmus A vypočíta prípustné riešenie s maximálnou relatívnou chybou ϵ, pričom $Time_A(x, \epsilon^{-1})$ môže byť ohraničený funkciou, ktorá je polynomiálna od $|x|$.* PTAS

Ak navyše $Time_A(x, \epsilon^{-1})$ môžme ohraničiť funkciou, ktorá je polynomiálna od oboch *parametrov $|x|, \epsilon^{-1}$, tak A je* úplná polynomiálna aproximačná schéma *(označujeme* FPTAS*) pre U.* FPTAS

Je zrejmé, že FPTAS je "to najlepšie", čo pre ťažké optimalizačné problémy môžme dostať. A teraz už spomínaná klasifikácia.

NPO(I) tvoria problémy, pre ktoré existuje FPTAS

NPO(II) tvoria problémy, pre ktoré existuje PTAS

NPO(III) tvoria problémy, pre ktoré existuje δ-aproximačný algoritmus pre nejaké $\delta > 1$, ale neexistuje d-aproximačný algoritmus pre $d < \delta$. (teda neexistuje PTAS)

NPO(IV) tvoria problémy, pre ktoré existuje $f(n)$-aproximačný algoritmus pre nejakú funkciu $f : \mathbb{N} \to \mathbb{R}$, ktorá sa dá zhora ohraničiť polylogaritmickou funkciou, ale pre ktoré neexistuje δ-aproximačný algoritmus pre žiadnu konštantu $\delta \in \mathbb{R}$

NPO(V) tvoria ostatné problémy z NPO

PTAS, resp FPTAS označujú aj triedu optimalizačných problémov, pre ktoré existuje PTAS, resp. FPTAS. Ľahko vidno, že

$$\mathsf{PO} \subseteq \text{FPTAS} \subseteq \text{PTAS} \subseteq \text{APX} \subseteq \mathsf{NPO}$$

Ak by $P = NP$, tak by všetky tieto triedy boli rovnaké. Naopak, platnosť $P \neq NP$ znamená, že sú navzájom rôzne. Dá sa totiž ukázať, že

- Knapsack $\in FPTAS$, ale rozhodovacia verzia je NPÚ, takže Knapsack $\notin PO$
- rozhodovacia verzia LargePacking je NP-úplná v silnom zmysle, takže existencia FPTAS by znamenala P=NP.
- LargePacking má PTAS
- Max3Sat patrí do APX, ale nie do $PTAS$
- existencia polynomiálneho algoritmu s konštantným performance ratio pre TravelingSalesmanProblem by znamenala existenciu polynomiálneho algoritmu pre HamiltonianCircuit

4.2.1 Ilustračné príklady-Absolútna chyba

Absolútnou chybou rozumieme absolútnu hodnotu rozdielu medzi získaným a optimálnym riešením

$$D(x, y) = |m(x, y) - m^*(x)|$$

Príkladov problémov s absolútnou chybou nie je veľa. Väčšinou sa jedná o problémy, kde sa hodnota účelovej funkcie pohybuje v malej množine hodnôt.

farbenie hrán vieme, že sa to dá alebo $\triangle$ alebo $\triangle + 1$. Čo z toho platí pre daný vstup je NP-ťažké...

farbenie planárnych grafov vieme, že planárne grafy sa dajú zafarbiť 4 farbami, existuje polynomiálny algoritmus, ktorý použije 5 farieb.

Určite neprekvapí, že existujú príklady ťažkých problémov, pre ktoré aproximačný algoritmus s absolútnou chybou neexistuje.

Negatívny výsledok

Veta 4.6 *Ak $P \neq NP$, tak pre problém* KnapSack*(KS) neexistuje aproximačný algoritmus (polynomiálnej zložitosti) s absolútnou chybou.*

Dôkaz: Pri dôkaze postupujeme štandardne - ak by sme taký algoritmus A mali, vedeli by sme pomocou neho vyriešiť nejaký NP-úplný problém v polynomiálnom čase; resp. NPO-úplný v polynomiálnom čase presne.

Ako? Nech A je algoritmus, ktorý rieši problém batohu s absolútnou chybou k.

- najprv prepočítame *profity*: $p'_i = (k+1)p_i$; uvedomme si, že pri takto prepočítaných hodnotách je hodnota každého (prípustného aj optimálneho) riešenia násobkom $(k+1)$
- vyriešime takto modifikovaný problém algoritmom A, ktorý sa dopustil absolútnej chyby nanajvýš k
- získané riešenie vlastne identifikuje optimálne riešenie pôvodného problému, pretože

$$|m(x', y) - m^*(x')| \leq k$$

$$|(k+1)m(x,y) - (k+1)m^*(x)| = (k+1)|m(x,y) - m^*(x)| \leq k$$

$$|m(x,y) - m^*(x)| \leq \frac{k}{k+1} < 1$$

Keďže sa hýbeme v prirodzených číslach, z $|m(x,y) - m^*(x)| < 1$ vyplýva, že $m(x,y) - m^*(x) = 0$, a teda algoritmus vypočítal optimálnu hodnotu $m^*(x) = m(x,y)$. □

4.2.2 Ilustračné príklady-Relatívna chyba, aproximačný pomer

$$E(x,y) = \frac{|m^*(x) - m(x,y)|}{max\{m^*(x), m(x,y)\}} \qquad R(x,y) = \max\left\{\frac{m^*(x)}{m(x,y)}, \frac{m(x,y)}{m^*(x)}\right\}$$

Príklad 4.1 *Uvažujme* KnapSack *a greedy metódu, ktorá vyberá prvky podľa maximálneho relatívneho profitu.*

Vieme, že ak pripustíme riešenie v racionálnych číslach, uvedený greedy prístup garantuje optimalitu získaného riešenia (v polynomiálnom čase). Pri riešení v celých číslach, resp. 0/1-riešení, je však problém ťažký, a preto greedy algoritmus optimalitu negarantuje.

Čo vieme povedať o kvalite takto získaného riešenia? Vieme aproximačný pomer ohraničiť? Uvažujme takýto vstup: *kvalita*

$w_i = 1$ pre $i = 1, \ldots, n-1$, $w_n = kn$ — váhy

$p_i = 1$ pre $i = 1, \ldots, n-1$, $p_n = b - 1$ — profity

$b = kn$ veľkosť batoha

Keďže pre $i = 1, \ldots, n-1$ je relatívny profit $p_i/w_i = 1 > p_n/w_n = (b-1)/b$, dosiahne greedy prístup profit $\mathbf{m_G(x, y) = n - 1}$. Pritom výber posledného objektu vedie k optimálnej hodnote $\mathbf{m^*(x) = b - 1 = kn - 1}$

$$R(x,y) = \frac{m^*(x)}{m_G(x,y)} = \frac{b-1}{n-1} = \frac{kn-1}{n-1} > k$$

Uvedený príklad poskytuje návod, ako k ľubovoľne veľkej hodnote $k \in \mathbb{N}$ skonštruovať vstup, pre ktorý algoritmus poskytuje riešenie s aproximačným pomerom *väčším* ako k. Preto *aproximačný pomer greedy algoritmu môže dosahovať ľubovoľne veľké hodnoty.*

Modifikácia greedy

Ľahko vidno, že optimalitu pokazil vlastne posledný objekt, ktorého relatívny profit bol nižší, ako relatívny profit ostatných, celkový zisk za jeho umiestnenie do batoha však prevýšil možný zisk za umiestnenie všetkých ostatných objektov súčasne. Toto pozorovanie môžme zakomponovať do greedy riešenia, čím získame algoritmus H.

Algoritmus 14 ModifGreedy H

▷ predpokladáme, že $w_1, \ldots, w_n \leq b$, $p_{max} = \max\{p_i,\ 1 \leq i \leq n\}$

1: utrieď nerastúco podľa relatívneho profitu $\frac{p_1}{w_1} \geq \cdots \geq \frac{p_n}{w_n}$
2: greedy algoritmom vypočítaj $m_G(x, y)$
3: $m_H(x, y) \leftarrow \max\{p_{max}, m_G(x, y)\}$

kvalita

Čo vieme povedať o kvalite, resp. chybe algorimu **H**? Nech j je index prvého prvku, ktorý sme *nezaradili* do greedy riešenia. Potom pre doterajšiu váhu $W(j)$ a profit $P(j)$ platí (prečo?)

$$W(j) = \sum_{i=1}^{j-1} w_i \leq b; \qquad b - W(j) < w_j$$
$$P(j) = \sum_{i=1}^{j-1} p_i \leq m_G(x,y)$$

Vieme, že hodnota optimálneho racionálneho riešenia je horným odhadom na cenu optimálneho celočíselného riešenia a že vektor optimálneho racionálneho riešenia by bol $(x_1, \ldots, x_n)$, kde $x_1 = \cdots = x_{j-1} = 1$ a $x_j = \frac{(b - W(j))}{w_j} < 1$. Príspevok profitu j-teho objektu v optimálnom riešení *racionálnej* verzie by teda bol

$$x_j p_j = (b - W(j)) \cdot \frac{p_j}{w_j} < \frac{w_j p_j}{w_j} = p_j$$

Spojením dostávame:

$$\boxed{m^*(x)} \leq m_G(x,y) = P(j) + x_j p_j \boxed{< P(j) + p_j}$$

Analýzu spravíme podľa toho, ako v algoritme H dopadlo priradenie na riadku 3; výsledok závisí od vzťahu $P(j)$ a p_{max}. Ak je index prvku s maximálnym profitom p_{max} nanajvýš $j-1$, algoritmus vráti $P(j)$. V opačnom prípade vráti p_{max}, pričom $p_j \leq p_{max}$. Uvažujme nasledujúce dva prípady:

$\mathbf{p_j \leq P(j)}$

$$m^*(x) \leq P(j) + p_j \leq 2P(j) = 2m_G(x) = 2m_H(x)$$
$$\Rightarrow \quad \frac{m^*(x)}{m_H(x)} \leq 2$$

$\mathbf{p_j > P(j)}$

$$m^*(x) \leq P(j) + p_j < 2p_j \leq 2p_{max} = 2m_H(x)$$
$$\Rightarrow \quad \frac{m^*(x)}{m_H(x)} < 2$$

Ukázali sme, že aproximačný pomer algoritmu ModifGreedy je 2; garantujeme teda, že pre každý vstup algoritmus nájde riešenie, ktorého profit je aspoň polovičný v porovnaní s optimálnym.

Fakt, že pre NP-úplný problém KnapSack existuje polynomiálny aproximačný algoritmus s aproximačným pomerom 2, neznamená, že všetky NP-úplné problémy sú takto dobre aproximovateľné. Príkladom za rozumných predpokladov neaproximovateľného problému je TSP[3].

Veta 4.7 *Ak $P \neq NP$ tak pre TSP neexistuje ϵ-aproximačný algoritmus (polynomiálnej zložitosti).* *Negatívny výsledok*

Dôkaz: Budeme postupovať sporom a ukážeme, že pomocou ϵ-aproximačného algoritmu TSP_ϵ pre problém obchodného cestujúceho by sme vedeli (presne) vyriešiť problém HK v deterministickom polynomiálnom čase. Základom je redukcia na problém TSP, ktorý vznikne vhodným dodefinovaním váh v úplnom grafe s rovnakým počtom vrcholov:

Algoritmus 15 TSP-HK

1: G je vstup pre problém HK
2: prerobíme G na vstup H pre problém TSP tak, že pridáme nenulové ceny pre všetky – teda aj neexistujúce – hrany. Pritom ceny hrán budú také, aby nám pomohli identifikovať, či nájdená Hamiltonovská kružnica prechádza aj po neexistujúcich[4] hranách.

$$c(i,j) = \begin{cases} 1, & (i,j) \in E; \\ \frac{|V|}{1-\epsilon}, & (i,j) \notin E \end{cases}$$

3: Vyriešme TSP pomocou algoritmu TSP_ϵ s relatívnou chybou ϵ

Nech $m(x,y)$ je riešenie získané algoritmom TSP-HK. Čo by sme vedeli povedať o cene získaného, resp. optimálneho riešenia pre H, ak vieme, že HK v pôvodom grafe existovala/neexistovala?

Ak pôvodný graf G obsahoval HK, potom optimálne riešenie pre H obsahuje len existujúce hrany, ktoré majú cenu nižšiu ako je $\frac{|V|}{1-\epsilon}$ (cena neexistujúcich hrán). Preto v tomto prípade $m^*(x) = |V|$. Keďže algoritmus TSP_ϵ je ϵ-aproximačný, *v $G \exists$ HK*

$$\frac{m(x,y) - m^*(x)}{m(x,y)} = \frac{m(x,y) - |V|}{m(x,y)} \leq \epsilon, \text{ čo po úprave dáva } \boxed{m(x,y) \leq \frac{|V|}{1-\epsilon}}$$

Ak pôvodný graf HK nemá, tak riešenie vypočítané algoritmom TSP-HK obsahuje aspoň jednu neexistujúcu hranu a preto $m^*(x) \geq |V| - 1 + \frac{|V|}{1-\epsilon} > \frac{|V|}{1-\epsilon}$. V tejto situácii teda $\boxed{m(x,y) \geq m^*(G) > \frac{|V|}{1-\epsilon}}$ *v $G \nexists$ HK*

Vidíme, že

$$G \text{ obsahuje HK} \iff m(x,y) \leq \frac{|V|}{1-\epsilon}$$

Existenciu alebo neexistenciu HK v G teda určíme na základe hodnoty $m(x,y)$, ktorú vypočítal algoritmus TSP-HK pre problém obchodného cestujúceho v grafe H s dopočítanými cenami.

□

[3] TSP=Traveling salesperson problem-problém obchodného cestujúceho

4.2.3 Ilustračné príklady-PTAS pre MinPartition

MinPartition

Na vstupe máme množinu X prirodzených čísel $\{x_1, ..., x_n\}$, ktorú chceme rozdeliť na dve množiny Y_1, Y_2 tak, aby sme minimalizovali ich rozdiel, resp. jednotlivé súčty.

vstup: $X = \{x_1, ..., x_n\}$, pričom $S = \sum x_i$
výstup: rozklad $Y = (Y_1, Y_2)$ množiny X na disjunktné množiny, ktorý minimalizuje $\max\{\sum_{x_i \in Y_1} x_i, \sum_{x_i \in Y_2} x_i, \} \rightsquigarrow \min$

Triviálnym prípustným riešením problému je vrátiť rozklad $Y = (\emptyset, X)$, ktorého hodnota je $m(x, y) = S$. Keďže $m^*(x) \geq S/2$, pre aproximačný pomer dostávame

$$\frac{m(x,y)}{m^*(x)} \leq \frac{m(x,y)}{(S/2)} \leq \frac{S}{S/2} = 2$$

Preto stačí pre aproximačný pomer r uvažovať prípad $r \leq 2$.

Algoritmus vychádza z predpokladu/pozorovania, že väčšie čísla sú pre kvalitu riešenia dôležitejšie. Pracuje preto v dvoch fázach - najprv najväčšie čísla spracuje optimálne, potom zvyšok dokončí greedy metódou. Počet prvkov, ktoré v prvej fáze usporiadame optimálne, určíme na základe zadanej/požadovanej hodnoty r tak, aby sme dosiahli požadovanú chybu.

Algoritmus 16 MinPartition

```
1: utriedime prvky nerastúco podľa hodnoty x_1 ≥ ... ≥ x_n
2: vypočítame hodnotu k(r), ktorá určuje počet prvkov spracovávaných presne
                                   ////vzťah pre výpočet k(r) odvodíme neskôr
3: nájdeme optimálne riešenie pre vstup x_1, ..., x_k(r)
4: for j=k(r)+1 to n do                                        //doriešime greedy
5:     if Σ_{x_i∈Y_1} x_i ≤ Σ_{x_i∈Y_2} x_i then
6:         Y_1 := Y_1 ∪ {x_j}
7:     else Y_2 := Y_2 ∪ {x_j}
```

čas

1 triedenie $O(n \log n)$

3 hľadanie optimálneho riešenia závisí od počtu možností prenásobeného zložitosťou kontroly; ak použije metódu hrubej sily, je počet možností exponenciálny od $k(r)$.
Zložitosť $O(2^{k(r)} \cdot n)$ tejto časti je v prípade, že považujeme r za konštantu, polynomiálna.

4 pri rozumnej implementácii realizujeme tento cyklus v čase $O(n)$

Získali sme algoritmus, ktorý je polynomiálny od veľkosti pôvodného vstupu (a exponenciálny od hodnoty $k(r)$).

kvalita

Kvôli jednoduchosti predpokladajme, že

$$x_1 \geq \ldots \geq x_n$$

Nech výsledkom sú množiny Y_1, Y_2, pričom

$$W(Y_1) = \sum_{x_i \in Y_1} x_i, \; W(Y_2) = \sum_{x_i \in Y_2} x_i, \; W(Y_1) \geq W(Y_2)$$

$$L = \frac{W(Y_1) + W(Y_2)}{2}$$

Hodnota L je dolným odhadom na hodnotu optimálneho riešenia (prečo?).

$h \leq k$

Nech h je index posledného prvku, ktorý sme počas realizácie algoritmu pridali do Y_1. Hodnota $h \leq k$ hovorí, že v Y_1 sú len prvky, ktoré sme tam umiestnili v prvej fáze. To ale znamená, že *optimálny výsledok* sme dostali už v prvej časti algoritmu.

Predpokladajme teda, že $h > k$, x_h je prvok, ktorý sme do Y_1 pridali v druhej fáze. Keďže x_h sme pridali do Y_1, tak $W(Y_1) - x_h \leq W(Y_2)$, resp. $W(Y_1) \leq x_h + W(Y_2)$. $h \geq k$
Pripočítaním $W(Y_1)$ k obom stranám

$$2W(Y_1) \leq x_h + W(Y_2) + W(Y_1) = x_h + W(X) = x_h + 2L$$

$$W(Y_1) \leq L + \frac{x_h}{2}$$

Vieme, že $\forall j < k(r)$ je $x_h \leq x_j$, preto $2L = W(Y_1) + W(Y_2) \geq x_h(k(r)+1)$

$$W(Y_1) \geq L \geq W(Y_2) \quad \& \quad m^*(x) \geq L$$

$$\frac{W(Y_1)}{m^*(x)} \leq \frac{W(Y_1)}{L} \leq \frac{L + \frac{x_h}{2}}{L} = 1 + \frac{x_h}{2L} \leq 1 + \frac{x_h}{x_h(k(r)+1)} = 1 + \frac{1}{k(r)+1}$$

Ak chceme, aby $1 + \frac{1}{k(r)+1} \leq r$, musí platiť(prečo?) $\frac{2-r}{1-r} \leq k(r)$. Preto v algoritme nastavíme *určenie $k(r)$*

$$\mathbf{k(r)} = \left\lceil \frac{\mathbf{(2-r)}}{\mathbf{(r-1)}} \right\rceil$$

$k(r)$ je konštanta závislá od k, ktorá garantuje požadovaný aproximačný pomer

$$\frac{W(Y_1)}{m^*(x)} \leq r$$

◇

4.2.4 Ilustračné príklady-FPTAS pre KnapSack

O probléme KnapSack vieme, že greedy metóda dáva optimálne riešenie v obore racionálnych čísel a že dynamickým programovaním získame pseudopolynomiálny algoritmus na presné riešenie KS v obore celých čísel v čase $O(n \cdot \sum_{i=1}^{n} c_i) = O(n^2 c_{max})$, kde $c_{max} = \max\{c_1, \ldots, c_n\}$. Ak by sme z pseudopolynomiálneho algoritmu chceli dostať polynomiálny, stačilo by vhodne (ako?) znížiť hodnotu c_{max}. Presne takto budeme postupovať. *Naškálujme vstup tak, aby sme nepracovali s hodnotami exponenciálnymi vzhľadom na veľkosť vstupu. Tým stratíme presnosť.*

Algoritmus FPTAS-KnapSack

Algoritmus 17 FPTAS-KnapSack

1: vypočítame hodnotu $\mathbf{T} \leftarrow \lfloor \frac{\mathbf{c_{max}(r-1)}}{\mathbf{rn}} \rfloor$ //voľbu T vysvetlíme neskôr
2: naškálujeme vstup $c'_i \leftarrow \lfloor \frac{c_i}{T} \rfloor$
3: pre naškálovaný vstup nájdeme *presné* riešenie $m^*(x', y')$ dynamickým programovaním; y' je množina indexov prvkov, ktoré sú umiestnené do batoha
4: $m(x, y')$ je cena riešenia y' s *pôvodnými* cenami

kvalita

Keby o pôvodnej cene každého prvku platilo $c_i = T \cdot c'_i$, bolo by vypočítané riešenie y' s cenou $m(x, y') = T \cdot m^*$ optimálne. To však nie je pravda, preto pri analyzovaní chyby $\Delta = m^*(x) - m(x, y)$, ktorej sme sa dopustili, musíme zvážiť, že pre $d_i \leftarrow T \cdot c'_i$ platí

$$d_i \leq c_i \quad \text{pričom} \quad |c_i - d_i| < T$$

Každý prvok zaradený do riešenia môže priniesť chybu max. T, preto

$$m^*(x) - m(x,y) \leq n \cdot T$$

Súčasne $n \cdot c_{max} \geq m^*(x) \geq c_{max}$, čiže môžme písať

$$\frac{m^*(x) - m(x,y)}{m^*(x)} \leq \frac{n \cdot T}{c_{max}}$$

$$m^*(x)(c_{max} - n \cdot T) \leq c_{max} m(x,y)$$

$$\frac{\mathbf{c_{max} - n \cdot T}}{\mathbf{c_{max}}} \leq \frac{\mathbf{m(x,y)}}{\mathbf{m^*(x)}}$$

Ak chceme dosiahnuť aproximačný pomer r, musí platiť

$$\frac{m^*(x)}{m(x,y)} \leq r, \text{ resp. ekvivalentne } \frac{\mathbf{m(x,y)}}{\mathbf{m^*(x)}} \geq \frac{\mathbf{1}}{\mathbf{r}}$$

Hľadáme teda T tak, aby platilo

$$\frac{\mathbf{1}}{\mathbf{r}} \leq \frac{\mathbf{c_{max} - n \cdot T}}{\mathbf{c_{max}}}$$

Úpravou dostávame

$$\mathbf{T} \leq \frac{\mathbf{c_{max}(r-1)}}{\mathbf{rn}}$$

Voľbou $T \leftarrow \lfloor \frac{c_{max}(r-1)}{rn} \rfloor$ na riadku 1 v algoritme 17 teda garantujeme požadovaný aproximačný pomer r.

čas

Preškálovanie vstupu prináša dodatočnú zložitosť $O(n)$, preto celková zložitosť je $O(n^2 c'_{max})$(prečo?). Keďže $T \leftarrow \lfloor \frac{c_{max}(r-1)}{rn} \rfloor$, platí aj $T \geq \frac{c_{max}(r-1)}{rn} - 1$, a preto

$$\mathbf{c_{max}} \leq \frac{(T+1)rn}{(r-1)} \leq \frac{\mathbf{2Trn}}{\mathbf{(r-1)}}$$

$$n^2 c'_{max} \leq n^2 \cdot \frac{c_{max}}{T} = \frac{n^2}{T} \cdot \mathbf{c_{max}} \leq \frac{n^2}{T} \cdot \frac{\mathbf{2Trn}}{\mathbf{(r-1)}} = O\left(\frac{n^3 r}{r-1}\right)$$

Celkový čas algoritmu je teda $O(\frac{n^3 r}{r-1})$

V čase, ktorý je polynomiálny od vstupu aj r a $1/r$ máme aproximatívny algoritmus s aproximačným pomerom $r \Rightarrow$ **FPTAS**.

◇

Úloha: Vyjadrite relatívnu chybu algoritmu 17 a zložitosť vzhľadom na veľkosť vstupu a relatívnu chybu.

4.2.5 Ilustračné príklady-SetCoverProblem a nekonštantný aproximačný pomer

S výnimkou základného greedy algoritmu na riešenie 0-1 batohu mali všetky doteraz prezentované aproximačné algoritmy pre ťažké optimalizačné problémy aproximačný pomer ohraničený konštantou. Patrili teda do triedy APX. Mohlo by sa zdať, že všetky optimalizačné problémy patria do tejto triedy. Nie je tomu ale tak. Ukážeme príklad problému, kde horný odhad na aproximačný pomer je neohraničene rastúca funkcia.

Začnime definíciou problému SCP-minimálneho pokrytia množinami

problém SCP

Vstup $(X, \mathcal{F})$, kde X je konečná množina,
$\mathcal{F} = \{F_1, \ldots, F_m\}$ je systém množín, $\mathcal{F} \subseteq 2^X$, taký, že
$\bigcup_{F_i \in F} F_i = X$

Výstup $\{F_{i_1}, \ldots, F_{i_k}\}$ také, že $X = \bigcup_{j=1}^{k} F_{i_j}$, k minimálne možné

Prirodzeným prístupom je aplikovanie greedy metódy, keď vyberáme množinu—kandidáta na zaradenie—ako tú, ktorá prinesie maximálny počet doteraz nepokrytých prvkov. Nech

C je množina tých prvkov z $\mathcal{F}$, ktoré už boli zaradené do pokrytia a teda určujú množinu už pokrytých prvkov (C - cover)

U je množina ešte nepokrytých prvkov (U - uncovered)

Algoritmus 18 greedy - SCP

(pri výbere maximalizujeme počet prvkov z ešte nepokrytej časti množiny X)

1: $C \leftarrow \emptyset$ $C \subseteq F$ a na záver C bude riešenie
2: $U \leftarrow X$ $U \subseteq X, U = X - \bigcup_{Q \in C} Q$ pre aktuánu hodnotu C a na záver $U = \emptyset$
3: **while** $U \neq \emptyset$ **do**
4: nech $S \in F$ je také, že $|S \cap U|$ je maximálne spomedzi $S \in F$
5: $U \leftarrow U - S; C \leftarrow C \cup \{S\}$
6: return C

Veta 4.8 *Algoritmus greedy-SCP je*[5] $Har(\max\{|S|, S \in F\})$*-aproximačný algoritmus pre SCP.*

Dôkaz: Nech v nájdenom riešení $C = \{S_1, \ldots, S_r\}$ indexy odpovedajú poradiu, v akom boli jednotlivé množiny algoritmom greedy-SCP pridávané do C, C^* označuje optimálne riešenie. Každému prvku $x \in X$ priradíme jeho *váhu* $w_C(x)$ *vzhľadom k* C ako $1/$(počet nových prvkov, ktoré spolu s ním prišli do C) *definícia* $w_C(x)$

$$w_C(x) = \frac{1}{|S_i - (S_1 \cup \ldots \cup S_{i-1})|}, \qquad x \in S_i - \bigcup_{j=1}^{i-1} S_j$$

Všimnime si, že súčet váh prvkov, ktoré sa do C dostali v jednej iterácii, je 1. Ľahko vidno, že ak označíme $w_C(X) = \sum_{x \in X} w_C(x)$, potom $\mathbf{w_C(X) = \sum_{x \in X} w_C(x) = |C|}$.

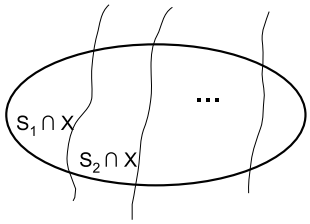
Dôkaz budeme robiť v dvoch krokoch. Najprv budeme predpokladať, že

$$\forall S \in \mathcal{F} : \sum_{x \in S} w_C(x) \quad \leq \quad Har(|S|) \tag{4.1}$$

a za tohto predpokladu určíme aproximačný pomer greedy algoritmom vypočítaného výsledku. Až potom dokážeme platnosť (3.1)

Nech teda $\forall \mathbf{S} \in \mathcal{F} : \sum_{\mathbf{x} \in \mathbf{S}} \mathbf{w_C(x)} \leq \mathbf{Har(|S|)}$. Potom pre veľkosť $|C|$ platí

$$\begin{aligned} \mathbf{|C| = \sum_{x \in X} w_C(x)} &\leq \sum_{S \in C^*} \sum_{\mathbf{x} \in \mathbf{S}} \mathbf{w_C(x)} \\ &\leq \sum_{S \in C^*} \mathbf{Har(|S|)} \\ &\leq |C^*| Har(max\{|S|\}) \end{aligned}$$



A teraz prejdime k dôkazu faktu (4.1). Pre každú množinu S nech $\mathbf{cov_i(S)}$ označuje počet prvkov množiny S, ktoré po i iteráciách ostali nepokryté a treba ich ešte pokryť. *dôkaz (4.1)*

$$cov_i(S) = |S - (S_1 \cup \ldots \cup S_i)| \qquad \forall i = 1, \ldots |C|$$

Ľahko vidno, že

$$cov_0(S) = |S| \qquad cov_{|C|}(S) = 0 \qquad cov_i(S) \leq cov_{i-1}(S)$$

Uvedomme si, že v i-tej iterácii vyberáme takú množinu S_i, ktorá maximalizuje $|S_i - (S_1 \cup \ldots \cup S_{i-1})|$

$$\begin{aligned}
\sum_{\mathbf{x}\in\mathbf{S}} \mathbf{w_C(x)} &= \sum_{i=1}^{|C|}(cov_{i-1}(S) - cov_i(S)) \cdot \frac{1}{|S_i - (S_1 \cup \ldots \cup S_{i-1})|} \\
&\leq \sum_{i=1}^{|C|}(cov_{i-1}(S) - cov_i(S)) \cdot \frac{1}{|S - (S_1 \cup \ldots \cup S_{i-1})|} \\
&= \sum_{i=1}^{|C|}(cov_{i-1}(S) - cov_i(S)) \cdot \frac{1}{cov_{i-1}(S)} \\
&\overset{(Lema\ 4.9)}{\leq} \sum_{i=1}^{|C|}(Har(cov_{i-1}(S)) - Har(cov_i(S))) \\
&= Har(cov_0(S) - Har(cov_{|C|}(S))) = Har(|S|) - Har(0) \\
&= \mathbf{Har(|S|)}
\end{aligned}$$

Ukázali sme, že algoritmus SCP má aproximačný pomer $Har(max\{|S|\})$, a je teda príkladom *f(n)-aproximačného* algoritmu □

Lema 4.9 *Platí* $Har(b) - Har(a) = \sum_{i=a+1}^{b} \frac{1}{i} \geq (b-a) \cdot \frac{1}{b}$

Dôkaz: Vlastnosť harmonických čísel dokážeme matematickou indukciou. Pre $a = b$ tvrdenie triviálne platí. Nech tvrdenie platí pre a, b, ukážeme, že platí aj pre $a, b+1$.

$$\sum_{i=a+1}^{b+1} \frac{1}{i} = \frac{1}{b+1} + \sum_{i=a+1}^{b} \frac{1}{i} \overset{IP}{\geq} \frac{1}{b+1} + \frac{b-a}{b} = \frac{b + (b+1)(b-a)}{b(b+1)} \overset{\text{prečo?}}{\geq} \frac{b+1-a}{b+1}$$

□

Dôsledok 4.10 *Algoritmus greedy-SCP je polynomiálny* $ln(|X|)-$*aproximačný algoritmus pre SCP.*

Dôkaz: Zakódujme vstup tak, aby sme mohli $|X||F|$ považovať za veľkosť vstupu (Ako?). Jedna iterácia kroku 3 trvá $O(|X||F|)$, počet opakovaní tohto kroku je zhora ohraničený (prečo?) $\min\{|X|, |F|\} \leq (|X||F|)^{1/2}$. Preto je zložitosť $O(n^{3/2})$.

□

[5] $Har(n) = 1 + 1/2 + \cdots + 1/n$

Kapitola 5

Návrh aproximačných algoritmov

5.1 Ďalšie príklady

Všimnime si využitie metód konštrukcie algoritmov pri návrhu aproximačných algoritmov. Už sme videli využitie greedy metódy pri probléme SetCover, spolu s naškálovaním sme greedy metódu využili na získanie FPTAS pre KnapSack.

MinVCP a greedy Túto metódu môžme použiť aj pri riešení problému MinVCP, kde sa snažíme pokryť množinu hrán minimálnym počtom vrcholov. Videli sme, že pri presnom riešení metódou rozdeľuj-a-panuj je zložitosť exponenciálna. Greedy metóda poskytne polynomiálny aproximačný algoritmus. Idea je jednoduchá - budeme konštruovať množinu nezávislých hrán A a do množiny C, ktorá obsahuje vrcholy z výsledného pokrytia, budeme zaraďovať oba konce nezávislých hrán.

Algoritmus 19 greedy - MinVCP

(náhodne vyberieme hranu (u, v), oba konce zaradíme do pokrytia a z grafu odstránime všetky hrany, incidentné s u, v)

1: $C \leftarrow \emptyset$ //$C \subseteq V$ obsahuje vrcholy z pokrytia
2: $A \leftarrow \emptyset$ //$A \subseteq E$ sú hrany párovania[1]
3: $E' \leftarrow E$ //$E' \subseteq E$ je množina ešte nepokrytých hrán
4: **while** $E' \neq \emptyset$ **do**
5: náhodne zvoľ hranu $(u, v) \in E'$
6: $C \leftarrow C \cup \{u, v\}$
7: $A \leftarrow A \cup \{(u, v)\}$
8: $E' \leftarrow E'$-$\forall$ hrany susedné s u, v // $E' \leftarrow E' \setminus \{(x, y) | (x, y) \in E'; x \in \{u, v\}\}$
9: return C

Počet opakovaní while-cyklu je zhora ohraničený počtom hrán, pri rozumnej implementácii je celkový príspevok while cyklu $O(|E|)$.(Naprogramujte) *čas*

Z popisu je zrejmé, že vypočítané riešenie je prípustným riešením. Ľahko tiež vidno, že po skončení každého while-cyklu v algoritme platí *kvalita*

$$2|A| = |C|$$

Keďže nezávislé hrany nemajú spoločné vrcholy, treba na ich pokrytie toľko vrcholov, koľko je hrán. Preto dolným odhadom na veľkosť pokrytia je maximálny počet nezávislých hrán.

Nech $m^*(G)$ označuje cenu optimálneho riešenia, $m(G)$ cenu riešenia, ktoré vypočítal algoritmus 19; $m(G) = |C|$. Zrejme

$$m^*(G) \geq |A| = \frac{|C|}{2} \quad \rightsquigarrow |C| \leq 2m^*(G) \quad \rightsquigarrow \frac{m(G)}{m^*(G)} \leq 2$$

Jednoduchý greedy prístup s náhodným výberom nezávislých hrán poskytuje aproximačný algoritmus s garantovaným aproximčným pomerom 2. □

W-MinVCP a LP Zmeňme trochu zadanie. Pridáme vrcholom cenu $c : V \to \mathbb{R}^+$ a budeme hľadať také pokrytie C, ktoré minimalizuje $\sum_{v \in C} c(v)$. V tejto situácii vyššie popísaná greedy prístup nepomôže.

Úloha: Nech $r > 1$. Nájdite taký vstup do Algoritmu 19, aby pomer $\frac{m(G)}{m^*(G)}$ bol väčší ako r.

Použitie lineárneho programovania je v tomto prípade efektívnejšie.

Algoritmus 20 LP-W-MinVCP

1: vyjadríme ako úlohu 0/1-lineárneho programovania
2: zrelaxujeme na LP
3: vyriešime zrelaxovanú LP $\rightsquigarrow X = (x_1, \ldots, x_n) \in (\mathbb{R}^+)^n$
4: $S_X = \{v_i \mid x_i \geq 1/2\}$
5: return S_X

kvalita

Čo vieme povedať o kvalite získaného riešenia? S_X je prípustné riešenie, pretože ohraničenie $x_i + x_j \geq 1$ pre každú hranu $(v_i, v_j) \in E$ garantuje, že aspoň jedno z $x_i, x_j \geq 1/2$ a teda aspoň jeden z vrcholov v_i, v_j zaradíme do pokrytia.

Riešenie získané relaxáciou 0/1-LP na LP nemôže byť horšie ako riešenie pôvodného 0/1-LP, preto $m^*_{0/1-LP}(G) \geq m^*_{LP}(G)$ a následne

$$cena(S_X) = \sum_{v \in S_X} c(v) = \sum_{\substack{i: \\ x_i \geq 1/2}} c(v_i) \overset{\text{prečo?}}{\leq} \sum_{\substack{i: \\ x_i \geq 1/2}} 2x_i c(v_i) = 2m^*_{LP}(G) \leq 2m^*_{0/1-LP}(G)$$

□

MaxCut

Maximálny hranový rez a lokálne prehľadávanie Ďalšou z metód, ktorú sme spomínali, je lokálne prehľadávanie. Použijeme túto metódu na riešenie problému maximálneho hranového rezu.

Vstup: $G = (V, E)$
Výstup: rozklad (V_1, V_2) taký, že $V_1 \cup V_2 = V,\ V_1 \cap V_2 = \emptyset$
maximalizujeme: počet hrán rezu: $|E \cap \{(u, v) \mid u \in V_1, v \in V_2\}|$

Pri aplikovaní metódy lokálneho prehľadávania začneme z nejakého prípustného riešenie, ktoré vylepšujeme prechodom do ďalšieho prípustného riešenia v jeho okolí. Začneme z triviálneho riešenie $(\emptyset, V)$. Okolím riešenia (X, Y) je také riešenie (X', Y'), ktoré z pôvodného vzniklo presunutím jedného vrchola z jednej množiny do druhej

Algoritmus 21 Search-MaxCut

1: $S \leftarrow \emptyset$ (rez budeme uvažovať ako $(S, V - S)$)
2: **while** $\exists v \in V$ taký, že presun v z jednej strany na druhú zvýšši cenu, **do**
3: realizuj tento presun
4: return $(S, V - S)$

zložitosť

Zamyslime sa nad zložitosťou algoritmu. Rrealizácia jedného opakovania while-cyklu trvá $O(|V|)$. Počet opakovaní je zhora ohraničený $|E|$ - každé opakovanie zvýši hodnotu aspoň o 1. Preto $O(|V||E|)$.

kvalita

Nech výstupom Algoritmu Search-MaxCut je (Y_1, Y_2). Ofarbime hrany modrou (v rámci jednotlivých Y_1, Y_2) a červenou (medzi Y_1 a Y_2). Keďže má každý vrchol $v \in Y_1$ aspoň toľko susedov v Y_2 ako v Y_1(prečo?), je modrých hrán nanajvýš toľko ako červených. Preto je v reze aspoň $\frac{|E|}{2}$ hrán. Keďže horný odhad na hodnotu optimálneho rezu je $|E|$

$$\frac{|E|}{2} \leq m(Y_1, Y_2) \leq m^* \leq |E| \leq 2m(Y_1, Y_2)$$

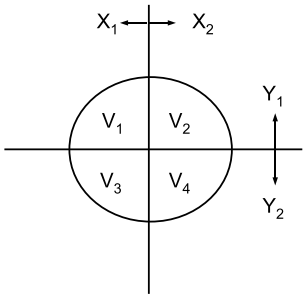
$$\frac{m^*}{m(Y_1, Y_2)} \leq \frac{2m(Y_1, Y_2)}{m(Y_1, Y_2)} = 2$$

Ľahko sme ukázali, že Algoritmus 21 má aproximačný pomer 2. Vieme spraviť aj presnejšiu analýzu tohto algoritmu?

Nech (Y_1, Y_2) je riešenie získané algoritmom a (X_1, X_2) optimálne riešenie.

Označme

$$V_1 = Y_1 \cap X_1$$
$$V_2 = Y_1 \cap X_2$$
$$V_3 = Y_2 \cap X_1$$
$$V_4 = Y_2 \cap X_2$$



e_{ij} počet hrán $|E \cap \{(x, y) \mid x \in V_i, y \in V_j\}|$

Potom

$$cena(X_1, X_2) = e_{12} + e_{14} + e_{23} + e_{34} \quad cena(Y_1, Y_2) = e_{13} + e_{14} + e_{23} + e_{24}$$

Uvažujme vrchol $x \in V_1 \subseteq Y_1$. Keďže (Y_1, Y_2) je výsledok prezentovaného algoritmu, je počet hrán medzi x a vrcholmi z $V_1 \cup V_2$ nanajvýš taký, ako medzi x a vrcholmi z $V_3 \cup V_4$. Preto

$$2e_{11} + e_{12} \leq e_{13} + e_{14}$$

Analogicky pre vrcholy z V_2, V_3, V_4. Preto

$$2e_{11} + e_{12} \leq e_{13} + e_{14}$$
$$2e_{22} + e_{12} \leq e_{23} + e_{24}$$
$$2e_{33} + e_{34} \leq e_{23} + e_{13}$$
$$2e_{44} + e_{34} \leq e_{14} + e_{24}$$

Sčítaním a predelením 2 dostaneme

$$\begin{aligned} \textstyle\sum_{i=1}^{4} e_{ii} + e_{12} + e_{34} &\leq e_{23} + e_{24} + e_{13} + e_{14} \qquad // + e_{14} + e_{23} \\ \textstyle\sum_{i=1}^{4} e_{ii} + e_{12} + e_{34} + e_{14} + e_{23} &\leq 2(e_{23} + e_{14}) + e_{24} + e_{13} \end{aligned}$$

$$\begin{aligned} cena(X_1, X_2) &= e_{12} + e_{34} + e_{14} + e_{23} \leq \sum_{i=1}^{4} e_{ii} + e_{12} + e_{34} + e_{14} + e_{23} \\ &\leq 2e_{23} + 2e_{14} + e_{24} + e_{13} \\ &= 2(e_{23} + e_{14} + e_{24} + e_{13}) - e_{24} + e_{13} = 2cena(Y_1, Y_2) - (e_{24} + e_{13}) \end{aligned}$$

Ak si uvedomíme, aké úpravy sme robili, dostávame

$$cena(X_1, X_2) + \sum_{i=1}^{4} e_{ii} \leq 2cena(Y_1, Y_2) - (e_{24} + e_{13})$$

resp.

$$cena(X_1, X_2) \leq 2cena(Y_1, Y_2) - \underbrace{\left(\sum_{i=1}^{4} e_{ii} + e_{24} + e_{13}\right)}_{\geq 0}$$

Kvalita získaného riešenia je teda pre veľa vstupov oveľa lepšia.

□

5.2 ΔTSP

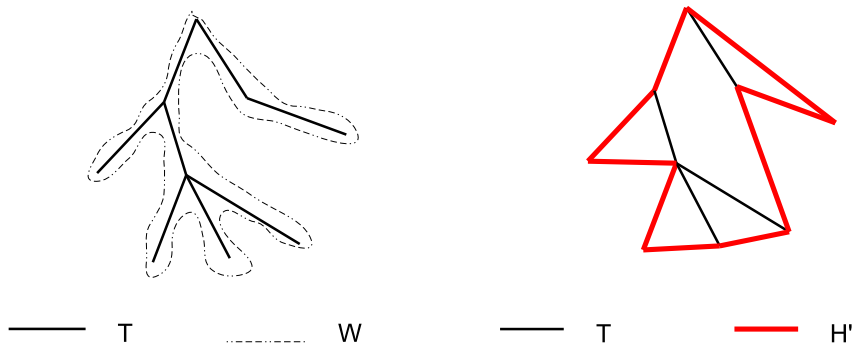
V tejto časti sa budeme venovať problému obchodného cestujúceho. Vieme, že tento problém sa nedá aproximovať s konštantnýn aproximačným pomerom (prečo?). Ukážeme, že napriek tomu pre veľkú množinu vstupov algoritmus s konštantným aproximačným pomerom existuje. Uvažujme také vstupy, v ktorých pre veľkosti hrán platí trojuholníková nerovnosť; v takom prípade hovoríme o probléme ΔTSP.

Δ TSP

Základom rozumného aproximačného algoritmu je minimálna kostra, ktorú vieme zostrojiť vpolynomiálnom čase (ako?). Nech teda T je minimálna kostra v grafe G. Ak by sme "obišli" hrany T, dostali by sme uzavretú cestu W, ktorej cena by bola rovná dvojnásobku ceny T. Ako z tejto uzavretej cesty získame kružnicu? Stačí začať kráčať po W a vynechávať tie vrcholy, v ktorých sme už boli. Jedinou výnimkou je vrchol, v ktorom sme začínali a v ňom aj skončíme.

Algoritmus 22 ΔTSP

1: nájdi minimálnu kostru T
2: vyber vrchol kostry v a začni prehľadávanie do hĺbky, pri ktorom vytváraj postupnosť H vrcholov tak, ako ich objavuješ
3: return $H' = H, v$



čas

Minimálnu kostru zostrojíme v čase $O(|E|)$, z nej požadovanú kružnicu H' v čase $O(n)$. Algoritmus je teda polynomiálny.

kvalita

Čo vieme povedať o kvalite? Nech H^* označuje HK s optimálnou cenou

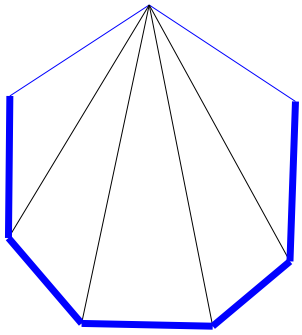
1. Odobratím hrany z hamiltonovskej kružnice získame kostru, preto $\sum_{e \in T} c(e) = cena(T) \leq cena(H^*)$
2. $cena(W) = 2cena(T) \leq 2cena(H^*)$
3. H' vznikla z W nahradením cesty hranou; z platnosti trojuholníkovej nerovnosti vyplýva, že táto hrana je kratšia ako cesta. Preto

$$cena(H') \leq cena(W) \leq 2cena(H^*)$$

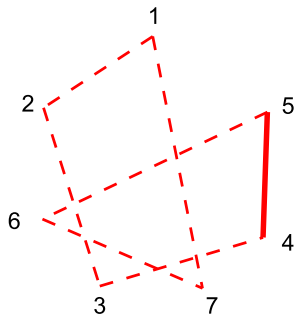
□

Ako ukazuje nasledujúci príklad, analýza algoritmu sa vylepšiť nedá.

Príklad 5.1 *Uvažujme úplný graf o n vrcholoch, v ktorom sú všetky hrany až na $n-2$ ceny 1, tých $n-2$ má cenu 2. Na obrázku sú hrany s cenou 2 vyznačené hrubšou čiarou*



minimálna kostra a kružnica vypočítaná algoritmom



optimálna kružnica

*minimálna kostra má cenu $n-1$, cena H' je $2(n-2)+2 =$ **$2n-2$**, pritom optimálna kružnica má cenu $n+1$*

$$\frac{\mathbf{2n-2}}{n+1} \longrightarrow 2$$

◇

Aproximačný pomer 2 sme dosiahli preto, že sme HK vytvárali z kružnice, ktorá bola dvojnásobnej dĺžky ako minimálna kostra. Skúsme minimálnu kostru T doplniť minimálnym počtom hrán tak, aby bol vzniknutý graf Eulerovský–aby v ňom existoval uzavretý ťah prechádzajúci po každej hrane práve raz. Keďže:

– graf je Eulerovský práve vtedy ak je každý vrchol párneho stupňa

– v grafe je počet vrcholov nepárneho stupňa párny

získame Eulerovský graf z minimálnej kostry T pridaním úplného párovania medzi vrcholmi nepárneho stupňa. Toto je idea Christofidesovho algoritmu.

Algoritmus 23 Christofides-ΔTSP

1: nájdi minimálnu kostru T
2: $S = \{v \in V \mid deg_T(v)$je nepárny$\}$
3: párovanie M minimálnej ceny na S (hranami v G)
4: $G' = (V, E(T) \cup M)$, v ňom eulerovský ťah ω
5: hamiltonovská kružnica H v G vznikne z ω tak, že vynechávame vrcholy, ktoré už boli navštívené (s výnimkou prvého vrchola, do ktorého sa vrátime)
6: return H

čas

Začnime analýzou času.

- minimálnu kostru spravíme v $O(|E|)$
- minimálne párovanie sa dá spraviť v $O(n^2|E|)$
- eulerovský ťah nájdeme v $O(n)$
- HK vyrobíme v $O(n)$

kvalita

H označuje výstup algoritmu, H^* je optimálna HK kružnica, ω je eulerovský ťah. Zrejme platí:

$$cena(H) \leq cena(\omega) = cena(T) + cena(M)$$

Keďže H^* je HK, vynechaním hrany z nej získame kostru, ktorá nemôže mať lepšiu cena ako minimálna kostra

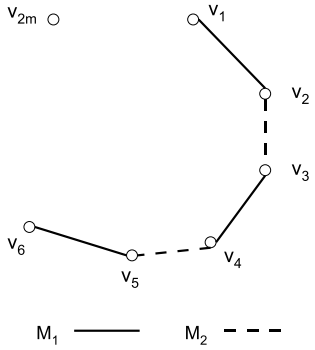
$$cena(T) \leq cena(H^*)$$

Sústredíme sa teda na odhad ceny párovania M. Nech $S = \{v_1, \ldots, v_{2m}\}$ sú vrcholy kostry T nepárneho stupňa v poradí, v akom sa vyskytujú v optimálnej KH H^*;

$$H^* = v_1, \alpha_1, v_2\alpha_2, \ldots, v_{2m}, \alpha_{2m}, v_1$$

Uvažujme 2 *konkrétne* párovania M_1, M_2, ktoré spoločne vytvárajú kružnicu prechádzajúcu všetkými vrcholmi z S:

$$\begin{array}{ll} M_1: & (v_1, v_2), (v_3, v_4), \ldots, (v_{2m-1}, v_{2m}) \\ M_2: & (v_2, v_3), (v_4, v_5), \ldots, (v_{2m}, v_1) \end{array}$$



Vzhľadom k trojuholníkovej nerovnosti

$$cena(v_1, \alpha_i, v_{i+1}) \geq cena(v_i, v_{i+1})$$

a preto

$$\begin{array}{rcl} cena(H^*) & \geq & \sum_{i=1}^{2m} cena(v_i, v_{i+1}) \\ & = & cena(M_1) + cena(M_2) \end{array}$$

Vieme, že M_1, M_2, M sú všetko úplné párovania(perfect matching), pritom M s minimálnou cenou, a teda

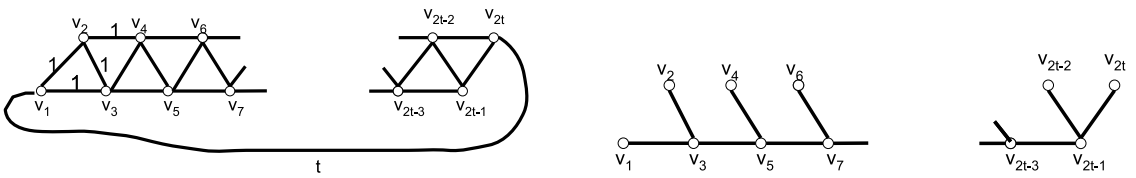
$$cena(M) \leq \min\{cena(M_1), cena(M_2)\} \leq \frac{cena(M_1) + cena(M_2)}{2} \leq \frac{cena(H^*)}{2}$$

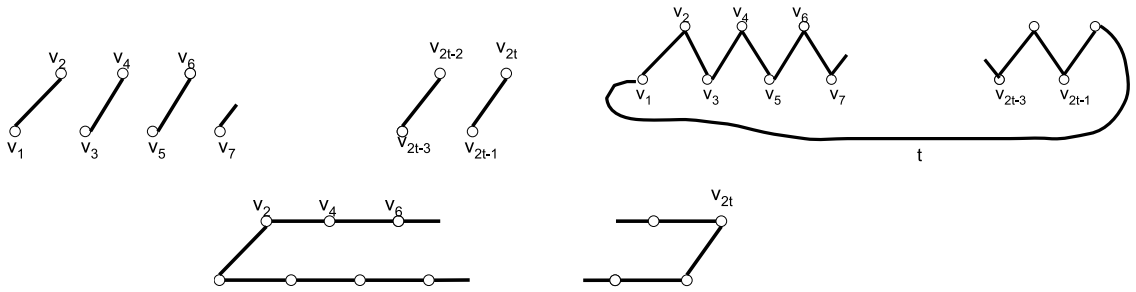
Spojením dostávame

$$cena(H) \leq cena(T) + cena(M) \leq cena(H^*) + \frac{cena(H^*)}{2} = \frac{3}{2} cena(H^*)$$

□

Opäť existuje príklad vstupu, pri ktorom sa "dosahuje" hranica aproximačného pomeru 3/2.





5.3 Stabilizácia

Už vieme, že vo všeobecnosti sa nie všetky optimalizačné problémy dajú rozumne aproximovať. Naproti tomu existujú také problémy, že pre niektoré rozumné podmnožiny ich vstupov vieme rozumnú aproximáciu napriek tomu garantovať. Je preto prirodzenou snahou hľadanie hranice zvládnuteľné $\longleftrightarrow$ nezvládnuteľné.

FPTAS, PTAS - väčšinou dobré, ale geometrický TSP má PTAS $n^{40\cdot\epsilon^{-1}}$

$\log n$ **aproximácia -** môže byť praktická

NPO(V) - sú vlastne *nezvládnuteľné*. Vieme totiž, že ak existuje polynomiálny $f(n)-$aproximačný algoritmus, tak za rozumných predpokladov (napr. $P \neq NP$) $f(n)$) nie je ohraničená polylogaritmickou funkciou. Napríklad problém TSP

Budeme pozítivnia zameriame sa na hľadanie toho, kde *je* to zvládnuteľné. Majme optimalizačný problém s dvoma rôznymi množinami vstupov I_1 a I_2, $I_1 \subsetneq I_2$. Navyše,

- pre I_1 *existuje* polynomiálny Δ-aproximačný algoritmus $\mathcal{A}$, $\Delta > 1$
- pre I_2 *neexistuje* žiaden polynomiálny $\gamma-$ aproximačný algoritmus, $\gamma > 1$ (ak $P \neq NP$).

Je situácia naozaj tak zlá, že algoritmus $\mathcal{A}$ možno rozumne použiť len pre vstupy z I_1? Nedokážeme použitím algoritmu $\mathcal{A}$ na inej množine vstupov garantovať pre tieto vstupy žiaden aproximačný pomer?

Nasledujúce pojmy sú snahou o zachytenie vlastností, ktoré v niektorých prípadoch pomôžu túto otázku zodpovedať. Snažíme sa zachytiť vlastnosť vstupu, ktorá nejakým spôsobom určuje/garantuje aproximačný pomer aj v prípade, že "dobrý" aproximačný algoritmus nebol použitý na "ideálnych" vstupoch.

Zadefinujme vzdialenosť d pre $x \in I_2 - I_1$. Ideálne by bolo, keby bol algoritmus taký, že dokážeme vyjadriť jeho kvalitu v závislosti od vzdialenosti príslušného vstupu od "dobrých" vstupov. Chceli by sme, aby $\forall k > 0$, $\forall x : d(x) \leq k$ $\mathcal{A}$ vypočítal $\gamma(k, \Delta)-$ aproximáciu $m^*(x)$. Ak to pre algoritmus $\mathcal{A}$ bude pravda povieme, že algoritmus $\mathcal{A}$ je *stabilný vzhľadom k d.*

Definícia 5.1 *$U = (I, Sol, m, ciel)$, $U' = (I', Sol, m, ciel)$, $I \subsetneq I'$ Hovoríme, že* ***vzdialenosť pre U′ vzhľadom k I*** *je funkcia $h_{I'} : I' \to \mathbb{R}^{\geq 0}$*

vzdialenosť, stabilita, kvázistabilita

- $h_{I'}(x) = 0$ $\forall x \in I$
- *$h_{I'}(x)$ je vypočitateľná v polynomiálnom čase*

Nech h je vzdialenosť pre U' vzhľadom k I. Potom

$$\mathbf{Ball_{r,h}(I)} = \{w \in I' \mid h(w) \leq r\}$$

Nech A je $\epsilon-$aproximačný algoritmus, $p \in R^{>0}$. Hovoríme, že algoritmus **A *je***

p-stabilný podľa h *ak $\forall 0 < r \leq p$ existuje konštanta $\delta_{r,\epsilon} \in R^{>0}$ tak, že $\mathcal{A}$ je $\delta_{r,\epsilon}-$ aproximačný pre $U_r = (Ball_{r,h}(I), Sol, m, ciel)$*

je stabilný podľa h *ak je p-stabilný podľa h $\forall p \in R^+$*

je nestabilný podľa h *ak neexistuje $p \in R^{>0}$, aby bol p-stabilný podľa h*

$(\mathbf{r}, \mathbf{f_r(n)})-$ **kvázistabilný podľa h** *ak existujú $r \in Z^+, f_r : \mathbb{N} \to R^{>1}$ tak, že $\mathcal{A}$ je $f_r(n)-$aproximačný pre $U_r = (Ball_{r,h}(I), Sol, m, ciel)$*

Vieme, že ΔTSP je "dobrým" prípadom všeobecného problému TSP. Budeme sa snažiť využiť rozumný algoritmus pre tento dobrý prípad aj na vstupoch, ktoré nie sú "dobré"; konkrétne si budeme všímať jeho kvalitu, keď ho aplikujeme na vstupy v nejakej vzdialenosti od dobrého prípadu. Začneme definovaním viacerých vzdialeností. Všímame si pritom pomer dĺžky jednej hrany k dĺžke cesty, ktorá spája príslušné koncové vrcholy, keď táto cesta má alebo nemá obmedzenie na počet hrán. Toto je dôležité v prípade, keď vstupy nespĺňajú trojuholníkovú nerovnosť.

Nech $I-$ označuje všetky grafy , $I_\Delta-$ iba grafy s trojuholníkovou nerovnosťou.

dist$(\mathbf{G}, \mathbf{c})$ umožňuje odhadnúť zhoršenie v prípade, keď nahradíme hranou cestu, ktorá je tvorená dvomi hranami

$$dist(G,c)=\max\left\{0, \max\left\{\frac{c(u,v)}{c(u,p)+c(p,v)} - 1 | u,v,p \in V(G), p \neq u \neq v \neq p\right\}\right\}$$

dist$_\mathbf{k}$ $(\mathbf{G}, \mathbf{c})$ umožňuje odhadnúť zhoršenie v prípade, keď nahradíme hranou cestu, ktorá je tvorená nanajvýš k hranami

$$dist_k(G,c)=\max\left\{0, \max\left\{\frac{c(u,v)}{\sum_{i=1}^{m} c(p_i,p_{i+1})} - 1 \mid u = p_1, p_2, \ldots, p_m = v; m-1 \leq k\right\}\right\}$$

distance$(\mathbf{G}, \mathbf{c})$ umožňuje odhadnúť zhoršenie v prípade, keď nahradíme hranou cestu ľubovoľnej dĺžky

$$distance(G,c) = \max\{dist_k(G,c) \mid 2 \leq k \leq |V(G)| - 1\}$$

V ďalšom bude $\mathbf{Ball_{r,dist}(I_\Delta)}$ označovať množinu tých grafov, v ktorých $\forall u \neq v$ platí $c(u,v) \leq (1+r)(c(u,p) + c(p,v))$.

Fakt 5.1 *Ak $dist_k(G,c) \leq r$ potom každé priame spojenie medzi u, v je nanajvýš $(r+1)$krát cena ľubovoľnej cesty medzi u, v, ktorá obsahuje maximálne k hrán.*

Analogicky k pojmu stabilného algoritmu môžme definovať pojem stabilnej, resp. super-stabilnej schémy.

Definícia 5.2 *$U = (I, Sol, m, ciel)$, $U' = (I', Sol, m, ciel)$, $I \subsetneq I'$. Nech h je vzdialenosť pre* **U′** *vzhľadom k I, a nech $A = \{A_\epsilon\}_{\epsilon>0}$ je PTAS pre U.*

Uvažujme rozšírenie U na $U_r = (Ball_{r,h}(I), Sol, m, ciel)$. Ak $\forall r > 0, \epsilon > 0$ A_ϵ je $\delta_{r,\epsilon}$-aproximačný pre U_r, tak ***PTAS* A *je stabilná podľa h.***

Ak $\delta_{r,\epsilon} \leq f(\epsilon) \cdot g(r)$, kde

- $f, g : R^{\geq 0} \to R^{>0}$
- $\lim_{\epsilon \to \infty} f(\epsilon) = 0$

Potom ***PTAS* A *je superstabilná podľa* h.**

Ako vyplýva z nasledujúceho faktu, h super-stabilná PTAS poskytuje PTAS na väčšej množine vstupov, ktorá je definovaná pomocou vzdialenosti h.

Fakt 5.2 *Ak A je superstabilná PTAS pre problém* $(I, Sol, m, ciel)$*, tak A je PTAS pre* $(Ball_{r,h}(I), Sol, m, ciel)$

Úloha: Dokážte Fakt 5.2.

stabilita Christofidesa

Vráťme sa k Christofidesovmu algoritmu a skúmajme, či je alebo nie je stabilným vzhľadom na nejakú mieru vzdialenosti. Začnime vzdialenosťou *distance*.

Christofides a distance

Lema 5.3 *Christofides je stabilný vzhľadom k distance*

Dôkaz: Majme vstup $I = (G, c) \in Ball_{r,distance}(I), r \in \mathbb{R}^{>0}$. Pri realizácii algoritmu postupne vytvárame

- minimálnu kostru T , perfect matching minimálnej ceny M (pre potreby analýzy uvažujeme konkrétne matchingy M_1, M_2)
- Eulerovskú cestu $\omega = v_1, \alpha_1, v_2, \alpha_2, \ldots, v_n, \alpha_n, v_1$ v grafe $T \cup M$
- HK $H = v_1, v_2, \ldots, v_n, v_{n+1} = v_1$, ktorá vznikla z ω postupným vynechávaním už navštívených vrcholov

Keďže vstupy sú z $Ball_{r,distance}(I)$, platí

$$cena(v_i, v_{i+1}) \leq (1 + r) cena(v_i, \alpha_i, v_{i+1})$$

a potom dosadením

$$cena(H) \leq (1 + r) \cdot cena(\omega)$$

$$cena(\omega) = cena(T) + cena(M)$$

$$cena(\omega) = cena(T) + cena(M) < cena(H^*) + (r + 1) cena(H^*) = (r + 2) cena(H^*)$$

$$cena(H) \leq (1 + r) cena(\omega) < (1 + r)(2 + r) cena(H^*)$$

□

Dôsledok 5.4 $\forall r > 0$ *je Christofides* $(1 + r)(2 + r)-$ *aproximačný algoritmus polynomiálnej zložitosti pre* $(Ball_{r,distance}(I_\Delta), Sol, cena, \min)$

Keď budeme uvažovať zúženie množiny vstupov vzdialenosťou *dist*, dostaneme iný výsledok; analýza ukazuje, že Christofidesov algoritmus je vtedy kvázistabilný.

Christofides a dist

Lema 5.5 $\forall r \in \mathbb{R}^{>0}$ *je Christofides* $(r, O(n^{\log_2(1+r)^2}))-$*kvázistabilný pre* dist

Dôkaz: Majme vstup $I = (G, c) \in Ball_{r,dist}(I_\Delta)$, T, M, ω, H sú minimálna kostra, minimálny perefect matching, eulerovská cesta a HK podľa Christofidesovho algoritmu.

Všimnime si, čo vzhľadom k definovanej vzdialenosti [2] $dist$ môžme povedať o cenách jednotlivých častí po nahradení cesty v, α, u hranou (v, u). $v, \alpha, u \longrightarrow (v, u)$

1. pre každú trojicu vrcholov $p, s, t \in V(G)$ platí $c(p, t) \leq (r + 1)c(p, s, t)$
2. nech cesta $P(u, \alpha, v)$ mala pôvodne m hrán
   - skrátenie[3] $m \to \lceil m/2 \rceil$ prináša multiplikatívny nárast $\times(r + 1)$
   - po $\log_2 m$ skracovaniach máme cestu dĺžky 1
   - výsledkom skracovania cesty u, α, v je teda hrana (u, v) o cene ktorej platí $cena(u, v) = c(u, v) \leq (1 + r)^{\lceil \log_2 m \rceil} \cdot cena(u, \alpha, v)$
3. vzhľadom k (1) pre vzťahy dĺžok $cena(H^*), cena(M)$ a $cena(H), cena(\omega)$ platí

$$cena(H) \leq (1 + r)^{\lceil \log_2 n \rceil} \cdot cena(\omega)$$

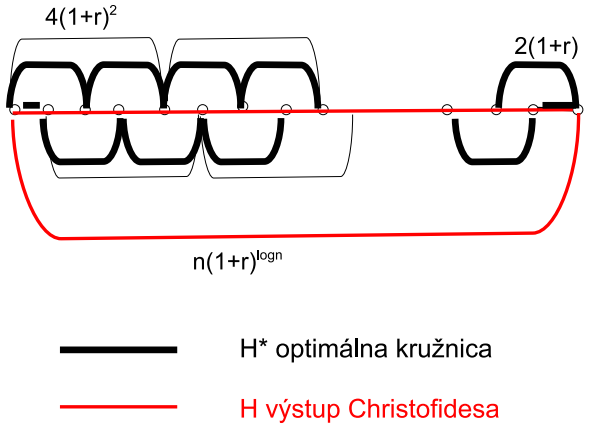
$$cena(M) \leq \frac{1}{2}(1 + r)^{\lceil \log_2 n \rceil} \cdot cena(H^*)$$

Spojením dostávame:

$$\begin{aligned} cena(H) &\leq (1 + r)^{\lceil \log_2 n \rceil} cena(\omega) = (1 + r)^{\lceil \log_2 n \rceil} (cena(T) + cena(M)) \\ &\leq (1 + r)^{\lceil \log_2 n \rceil} [cena(H^*) + \tfrac{1}{2}(1 + r)^{\lceil \log_2 n \rceil} cena(H^*)] \\ &= (1 + r)^{\lceil \log_2 n \rceil} \left(1 + \tfrac{1}{2}(r + 1)^{\log_2 n}\right) cena(H_{opt}) \\ &= O\left(n^{\log_2 (1+r)^2} \cdot cena(H^*)\right) \end{aligned}$$

□

Na obrázku je príklad grafu-vstupu do Christofidesovho algoritmu. Hrubou(čiernou) čiarou je naznačené optimálne riešenie, zatiaľ čo strednou(červenou) farbou je zvýraznené riešenie vypočítané Christofidesovým algoritmom.



Pre aproximačný pomer platí:

$$\begin{aligned} \frac{cena(D)}{cena(H_{opt})} = \frac{n + n(1 + r)^{\log n}}{2 + 2(1 + r)(n - 2)} &\geq \frac{n(1 + r)^{\log n}}{2 + 2(1 + r)(n - 2)} \\ &\geq \frac{n(1 + r)^{\log n}}{2(1 + r)n} = \frac{n^{\log_2 (1+r)}}{2(1 + r)} \end{aligned}$$

[2] $dist(G, c) = \max\left\{0, \max\left\{\frac{c(u,v)}{c(u,p)+c(p,v)} - 1 | u, v, p \in V(G), p \neq u \neq v \neq p\right\}\right\}$

[3] skrátením rozumieme nahradenie cesty p, s, t hranou pt postupne pre všetky trojice na ceste u, α, v

Dokázali sme tak nasledujúcu lemu.

Lema 5.6 *$\forall r \in \mathbb{R}^+$, ak Christofides je $(r, f_r(n))-$ kvázistabilný pre* dist*, tak*

$$f_r(n) \geq \frac{n^{\log_2(1+r)}}{2(1+r)}$$

Môžme úpravou dostať stabilitu pre dist? Zamyslime sa, čo spôsobilo, že Christofidesov algoritmus nie je stabilný vzhľadom k *dist*. Bolo to preto, že pri vytváraní kružnice H z eulerovskej cesty ω sme nahrádzali hranou vzhľadom na počet hrán potenciálne dlhé cesty. Ak by sa nám podarilo vytvoriť takú cestu, z ktorej by k redukcii na HK stačilo odstraňovať cesty vzhľadom na počet hrán *dokázateľne* krátke, mohlo by to pomôcť.

Algoritmus Sekanina zabezpečil odstraňovanie dlhých ciest z eulerovského ťahu. Postupne vysvetlíme ideu algoritmu a tiež dôkaz jeho korektnosti:

1. Ku každému stromu $T = (V, E)$ môžme vytvoriť graf T^k, ktorý vznikne doplnením hrán, spájajúcich–po hranách T– vrcholy vo vzdialenosti k. Ukážeme, že k minimálnej kostre T zostrojený graf $T^3 = (V, \{(x, y) | \exists \text{ cesta } x, P, y \text{ dĺžky max.3}\})$ obsahuje HK. V tomto bode pod dĺžkou cesty myslíme počet hrán.

2. Odhadneme dĺžku cesty $P_H(U) = u_1, P_{u_1,u_2}, u_2, \ldots, u_n, P_{u_n.u_1}$ v T vzhľadom k HK $(u_1, u_2, \ldots, u_n)$ v T^3. Pozor, teraz užuvažujeme váhované/ocenené dĺžky hrán.

3. Ukážeme, že sa to dá spraviť tak, že každá hrana z T sa v $P(H)$ objaví dvakrát.

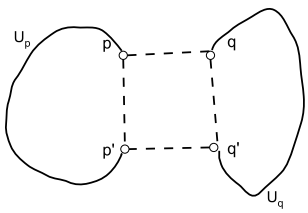
Definícia 5.3 *Nech T je strom, $\forall (u, v) \in E(T)$ nech $P_{u,v}$ je jediná cesta v strome T medzi vrcholmi u, v. Nech $U = u_1, \ldots, u_m$ je jednoduchá cesta v T^k. Potom U-cesta v T je* $P_T(U)$

$$P_T(U) = u_1, P_{u_1,u_2}, u_2, \ldots, P_{u_{m-1},u_m}, u_m$$

A teraz už kľúčová lema pre algoritmus Sekanina.

Lema 5.7 *$T, n \geq 3, (p, q) \in E(T)$ Potom T^3 obsahuje Hamiltonovskú cestu U : $p{=}v_1, v_2, \ldots, v_n{=}q$ takú, že každá hrana z $E(T)$ sa v $P_T(H)$ vyskytuje presne dvakrát, pričom $H = U, p$ je HK v T^3*

Dôkaz: (Náčrt)Spravíme indukciu vzhľadom na počet vrcholov v T. (Bázu indukcie ako cvičenie)
Vezmime hranu $(p, q) \in E(T)$ a odstráňme ju zo stromu T. Dostaneme dva podstromy T_p, T_q s koreňom v p, resp. q; $T = T_p \cup T_q \cup (p, q)$.



Nech p' je sused p v T_p, ak existuje. Analogicky q'.

Skonštruujeme hamiltonovské cesty U_p v T_p^3 a U_q v T_q^3. Pri konštrukcii rozlišujeme 3 prípady.

1. Ak $|V(T_p)| = 1$, potom $U_p = p = p'$

2. Ak $|V(T_p)| = 2$, potom $U_p = (p, p')$

3. Ak $3 \leq |V(T_p)| \leq n - 1$, potom využije IP. Nech U_p je hamiltonovská cesta z p do p' v T_p^3. Pritom $P(U_p, p)$ obsahuje každú hranu z $E(T_p)$ práve trikrát. Analogicky U_q, $P(U_q, q)$.

Keďže p', p, q, q' je cesta v T, hrana $(p', q') \in E(T^3)$. Preto U_p, q', U_q^R je hamiltonovská cesta v T^3 a $H = U_p, q', U_q^R, p$ je hamiltonovská kružnica v T^3.

Ostáva ukázať, že každá hrana z T je v H práve dvakrát.
Opäť IP. U_p, p obsahuje každú hranu z $E(T_p)$ práve 2krát, preto U_p obsahuje každú hranu z $E(T_p \setminus (p, p'))$ dvakrát a hranu (p, p') práve raz. Analogicky pre hrany cesty U_q. Spolu teda máme:

- $\hookrightarrow$ až na hrany $(p, p'), (q, q'), (p, q)$ sú všetky hrany práve 2krát
- $\hookrightarrow$ hrany (p, p') a (q, q') máme raz
- $\hookrightarrow$ hrana (p, q) sa v $U_p \cup U_q$ nevyskytuje
- $\hookrightarrow$ Prepojenie hamiltonovských ciest U_p a U_q^R do kružnice prináša v T existujúcu hranu (p, q) a v T neexistujúcu hranu (p', q'), ktorá reprezentuje v T jedinú cestu medzi vrcholmi p' a q': $p' \rightsquigarrow p \rightsquigarrow q \rightsquigarrow q'$. To prinieslo hrany $(p', p), (p, q), (q, q')$.

□

Algoritmus 24 Sekanina

vstup: $G = (V, E), c : E \to N^+$

1: konštrukcia minimálnej kostry T
2: konštrukcia T^3
3: HK v T^3 taká, že $P_T(H)$ obsahuje každú hranu z $E(T)$ dvakrát
4: return(H)

Veta 5.8 *Algoritmus Sekanina je 2-aproximačný pre ΔTSP*

korektnosť

Dôkaz: Korektnosť algoritmu vyplýva z predchádzajúcich úvah a tvrdení.

čas

Konštrukcia minimálnej kostry T a následne aj grafu T^3 sa realizuje v čase $O(n^2)$ (naprogramujte). Konštrukcia H je podľa lemy 5.7 v čase $O(n)$

cena

1. T je minimálna kostra, preto $cena(T) \leq cena(H^*)$
2. H vznikla ako skracovanie cesty $P_T(H)$, v ktorej sa každá hrana z T objavuje dvakrát. Preto
$$cena(P_T(H)) = 2cena(T) \leq 2cena(H^*)$$
3. H vzniklo nahrádzaním $u_i P_i u_{i+1}$ hranou $u_i u_{i+1}$. Keďže platí trojuholníková nerovnosť,
$$cena(H) \leq cena(P_T(H))$$

Spojením dostávame

$$cena(H) \leq cena(P_T(H)) \leq 2cena(H^*)$$

□

Označme $\Delta TSP_r = (Ball_{r,dist}(L_\Delta), Sol, cena, min)$. Ľahko sa ukáže, že algoritmus 24 je $2(1+r)^2-$aproximačný pre ΔTSP_r, a teda stabilný vzhľadom k $dist$.

stabilita pre dist

Veta 5.9 *Algoritmus 24 je pre ΔTSP_r, $r \in \mathbb{R}^+$, $2(1+r)^2-$aproximačný*

Dôkaz: Majme vstup $I = (G, c) \in \Delta TSP_r$. Keďže $(G, c) \in Ball_{r,dist}(L_\Delta)$, tak

$$\begin{aligned} c(v_1, v_4) &\leq (1+r)^2 \cdot cena(v_1, v_2, v_3, v_4) \\ c(u_1, u_3) &\leq (1+r) \cdot cena(u_1, u_2, u_3) \end{aligned}$$

Nahrádzame cestu $u_i P_i u_{i+1}$ hranou, preto

$$cena(H) \leq (1+r)^2 cena(P_T(H)) \leq 2(1+r)^2 cena(H^*)$$

□

Dôsledok 5.10 *Algoritmus Sekanina je stabilný pre dist.*

zosilnená trojuholníková nerovnosť

Definícia 5.4 *Nech (G, c) je vstup do TSP, $1 > p \geq 1/2$. Hovoríme, že (G, c) spĺňa podmienku p-zosilnenej trojuholníkovej nerovnosti (str(p)), ak*

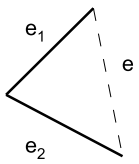
$$\forall u, v, w \quad c(u, v) \leq p \cdot [c(u, w) + c(w, v)]$$

Uvedomme si, že ak $p = 1/2$ tak všetky hrany, a teda aj všetky HK sú rovnako dlhé. Našou snahou je ukázať, že algoritmus Christofides je stabilný pre vstupy, na ktorých platí $str(p)$.

Označme teda
- $\Delta TSP_{p-1} = (I_{str(p)}, Sol, cena, min)$ verziu TSP na vstupoch, kde platí $str(p)$
- $c_{min}(G) = \min\{c(e) \mid e \in E\}$
- $c_{max}(G) = \max\{c(e) \mid e \in E\}$

Lema 5.11 *Nech $1/2 \leq p < 1$ a $(G, c) \in \Delta TSP_{p-1}$. Potom*

1. *$\forall$ susediace e_1, e_2 $cena(e_1) \leq \frac{p}{1-p} cena(e_2)$*
2. *$c_{max}(G) \leq \frac{2p^2}{1-p} \cdot c_{min}(G)$*



Dôkaz:

1. K dôkazu prvej časti lemy uvažujme aj dodatočnú(možno neexistujúcu) hranu e. Keďže platí zosilnená trojuholníková nerovnosť, môžme písať:

$$\begin{aligned} c(e_1) &\leq p[c(e_2) + c(e)] \leq p[c(e_2) + p(c(e_1) + c(e_2))] \\ &= pc(e_2) + p^2 c(e_1) + p^2 c(e_2) \\ c(e_1)(1-p^2) &\leq c(e_2)(p+p^2) \\ c(e_1) &\leq c(e_2)\tfrac{p(1+p)}{(1-p)(1-p)} = c(e_2)\tfrac{p}{1-p} \end{aligned}$$

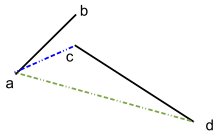
2. Nech $(a, b), (c, d)$ sú tie hrany, pre ktoré $c(a, b) = c_{min}(G), c(c, d) = c_{max}(G)$. Rozlíšime dva prípady podľa toho, či hrany $(a, b), (c, d)$ majú spoločný bod alebo nie:

- ak spoločný bod majú, tak

$$c_{max}(G) \overset{lema(1)}{\leq} \frac{p}{1-p} c_{min}(G) \overset{1/2 \leq p < 1}{\leq} \frac{2p^2}{1-p} c_{min}(G)$$

- ak spoločný bod nemajú, tak využitím už dokázanej časti 1.

$$\begin{aligned} c(a,c) &\leq \tfrac{p}{1-p}c(a,b) = \tfrac{p}{1-p}c_{min}(G) \\ c(a,d) &\leq \tfrac{p}{1-p}c(a,b) = \tfrac{p}{1-p}c_{min}(G) \\ c(c,d) &\leq p\left((c(a,c)+c(a,d))\right) \leq p\cdot 2\cdot \tfrac{p}{1-p}c_{min}(G) \end{aligned}$$



□

Dôsledok 5.12 *Každá HK H v ΔTSP_{p-1} spĺňa*

$$\frac{cena(H)}{opt_{TSP}(G,c)} \leq \frac{2p^2}{1-p}$$

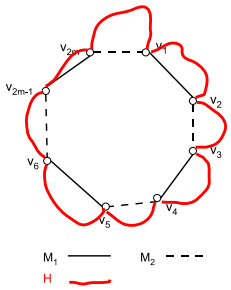
Veta 5.13 *$\forall 1/2 \leq p < 1$ je Christofidesov algoritmus $\left(1+\frac{2p-1}{3p^2-2p+1}\right)$ − aproximačný pre ΔTSP_{p-1}*

Dôkaz: Pri konštrukcii HK H algoritmom Christofides možno identifikovať dve skracovania

- jedno skracovanie je pri extrakcii HK H z eulerovského ťahu ω
- uvedomme si, že aj párovanie M možno chápať ako skracovanie HK

Ak má M málo hrán, šetríme pri párovaní, ak je $|M|$ veľká, šetríme pri skracovaní eulerovského ťahu. Poďme odhadnúť, koľko. Zopakujme si označenia z Christofidesovho algoritmu.

H	výstup CHA pri vstupe $(G,c) \in \Delta TSP_{p-1}$
ω	Eulerovský ťah
$\mathbf{H^*}$	optimálne riešenie
T	minimálna kostra
$v_1, v_2, \ldots, v_k, k = 2m$	vrcholy nepárneho stupňa v T s indexami v takom poradí ako sa vyskytujú v H^*
$\mathbf{M_1}$	$v_1v_2, v_3v_4, \ldots$
$\mathbf{M_2}$	$v_2v_3, v_4v_5, \ldots$



$$\begin{aligned} cena(H^*) &\geq cena(M_1) + cena(M_2) + (n-k)(1-p)2c_{min}(G) \\ cena(M) &\leq \tfrac{1}{2}(cena(M_1) + cena(M_2)) \\ &\leq \tfrac{1}{2}cena(H^*) - (n-k)(1-p)c_{min}(G) \\ cena(T) &\leq cena(H^*) - c_{min}(G) \leq \\ &\leq cena(H^*) - (1-p)\cdot 2\cdot c_{min}(G) \qquad //2(1-p) \geq 1 \\ cena(\omega) &= cena(T) + cena(M) \\ &\leq cena(H^*) - 2(1-p)c_{min}(G) + \tfrac{1}{2}cena(H^*) - (n-k)(1-p)c_{min}(G) \\ &= \tfrac{3}{2}cena(H^*) - (1-p)c_{min}(G)(2+n-k) \end{aligned}$$

$$\begin{aligned} cena(H) &\leq cena(\omega) - \left(\tfrac{k}{2}-1\right)(1-p)\cdot 2c_{min}(G) \qquad //\omega \text{ má } n-1+k/2 \text{ hrán} \\ &\leq \tfrac{3}{2}cena(H^*) - (1-p)c_{min}(G)(2+n-k) - (1-p)\cdot 2c_{min}(G)\left(\tfrac{k}{2}-1\right) \\ &= \tfrac{3}{2}cena(H^*) - (1-p)c_{min}(G)[2+n-k+k-2] \\ &= \tfrac{3}{2}cena(H^*) - (1-p)c_{min}(G)\cdot n \qquad //c_{max} \leq \tfrac{2p^2}{1-p}c_{min}(G) \\ &\leq \tfrac{3}{2}cena(H^*) - \tfrac{(1-p)^2}{2p^2}c_{max}(G)\cdot n \end{aligned}$$

Nech $\Gamma = \{\gamma \geq 1 \mid cena(H) \leq \gamma \cdot cena(H^*) \leq n \cdot c_{max}\}$
$\forall \gamma \in \Gamma$: $cena(H) \leq \frac{3}{2} cena(H^*) - \frac{(1-p)^2}{2p^2} \cdot \gamma \cdot cena(H^*) = \left(\frac{3}{2} - \gamma \cdot \frac{(1-p)^2}{2p^2}\right) cena(H^*)$

$cena(H) \leq \min\{\min\{\gamma, \frac{3}{2} - \gamma \cdot \frac{(1-p)^2}{2p^2}\} \mid \gamma \in \Gamma\} cena(H^*)$
Minimalizáciou dostaneme

$$\gamma = \frac{3p^2}{3p^2 - 2p + 1} = 1 + \frac{2p - 1}{3p^2 - 2p + 1} = 1 + \delta(\mathbf{p})$$

□

5.4 dual-PTAS pre minimum MakeSpan

Niekedy sú podmienky kladené na výstup príliš komplikované, resp. nie celkom špecifikované,.. Vtedy má zmysel uvažovať také "riešenia"/takmer-riešenia/... ťažkých problémov, ktoré majú síce dobrú kvalitu, ale *nie sú* prípustným riešením. Tieto kvalitné takmer-riešenia sa niekedy dajú využiť na nájdenie rozumného prípustného riešenia.

Analogicky ako sme uvažovali o vzdialenosti vstupu od "dobrých" vstupov, môžme uvažovať o vzdialenosti "riešenia" od množiny prípustných riešení. Dostávame satak k pojmu h-dual-ϵ-aproximačného algoritmu.

$Sol_h^\epsilon(x)$

Nech h je funkcia, ktorá priradí potenciálnemu riešeniu $\tilde{y}$ jeho vzdialenosť od množiny prípustných riešení, pričom:
- $h(x,\tilde{y}) = 0$ pre každé $S \in Sol(x)$ // hovoríme o vzdialenosti $\tilde{y}$ a $Sol(x)$
- $h(x,\tilde{y}) > 0$ pre $S \notin Sol(x)$
- $h(x,\tilde{y})$ vypočitateľná v polynomiálnom čase

Označme $Sol_h^\epsilon(x) = \{\tilde{y} \mid h(x,\tilde{y}) \leq \epsilon\}$

h-dual-ϵ-aproximačný algoritmus

Definícia 5.5 *Majme vzdialenosť*[4] *h. Hovoríme, že algoritmus A je h-dual-ϵ aproximačný algoritmus pre problém U, ak*
- $A(x) \in Sol_h^\epsilon(x)$
- $cena(A(x))$ *je* nie je horši*a ako cena optimálneho riešenia; presnejšie*

$$cena(A(x)) \begin{cases} \geq opt_U(x) & \textit{pre maximalizačný problém} \\ \leq opt_U(x), & \textit{pre minimalizačný problém.} \end{cases}$$

Analogicky *h-dual*-PTAS, *h-dual*-FPTAS. Napíšte definície.

BinP

Príklad 5.2 Uvažujme nasledujúcu verziu problému BinPacking, keď chceme umiestniť objekty do minimálneho počtu košov jednotkovej ceny.

vstup: $(r_1,\ldots,r_n); r_i$ sú racionálne čísla z $(0,1)$
výstup: $S = \{Y_i \in \{0,1\}^n \mid \sum_{i=i}^n Y_i r_i \leq 1,\ \sum Y_i = (1,\ldots,1)\}$
cieľ: minimalizácia $|S|$

Ukážeme, ako algoritmus, ktorý pre špeciálne vstupy dáva v polynomiálnom čase presné riešenie, môžme využiť na konštrukciu h-duálnej schémy pre všeobecné riešenie a tú následne pre zostrojenie PTAS pre jednu z verzií problému rozvrhovania. Pri konštrukcii vysvetlíme aj princíp duálnej úlohy.

MS

Uvažujme takú verziu problému rozvrhovania, keď máme n úloh s rôznym časom spracovania p_i vyriešiť pomocou m počítačov tak, aby sme minimalizovali celkový čas. Inými slovami chceme n úloh (resp. ich indexov) rozdeliť na m podmnožín $S_1,\ldots.S_m$ tak, aby sme minimalizovali (maximálny) čas potrebný na spracovanie úloh v jednotlivých podmnožinách:

vstup: $I = (p_1,\ldots,p_n,m),\ m \geq 2,\ n \in \mathbb{N} - \{0\}$
výstup: $S = \{S_1,\ldots,S_m \mid S_i \subseteq \{1,\ldots,n\}, \bigcup_k S_k = \{1,\ldots,n\},\ S_i \cap S_j = \emptyset\}$
cena: $cena(S) = cena(S_1,\ldots,S_m) := \max_{S_i} \left\{ \sum_{j \in S_i} p_j \right\}$
cieľ: minimalizácia $cena(S)$

BinP↔ MS

V akom zmysle sú oba problémy duálne? Ak vieme vypočítať úlohy s trvaním $p_1, \ldots, p_n$ na m počítačoch v čase d, tak vieme objekty o váhe $p_1, \ldots, p_n$ umiestniť do m košov veľkosti d a naopak. Keďže v našom zadaní sú koše veľkosti 1, musíme veľkosti objektov patrične zmenšiť.

$$\boxed{\textit{opt-BinP}(\tfrac{p_1}{d}, \ldots, \tfrac{p_n}{d}) \leq m \Leftrightarrow \textit{opt-MS}(p_1, \ldots, p_n, m) \leq d}$$

5.4.1 h-dual PTAS pre BinP

A teraz k riešeniu BinP. h-PTAS budeme konštruovať v troch krokoch:

1. navrhneme algoritmus, ktorý dynamickým programovaním *presne* rieši problém na takých vstupoch, v ktorých sa vyskytuje *konštantne veľa rôznych hodnôt*; problém označíme $BinP_s$, algoritmus A_{BinP_s}
2. pre zadanú konštantu ϵ vyriešime pomocou A_{BinP_s} problém na množine takých vstupov, kde $p_i \geq \epsilon$; algoritmus nazveme A_{BinP_ϵ}.
3. napokon využijeme A_{BinP_ϵ} na konštrukciu h-duálnej PTAS pre všeobecný BinP.

presné riešenie $BinP_s$

1. Ak sa vo vstupe nachádza s rôznych hodnôt, môžme vstup vyjadriť aj takto: $I = (q_1, \ldots, q_s; n_1, \ldots, n_s)$, kde q_i je hodnota a n_i počet prvkov na vstupe s hodnotou q_i; $0 < q_i \leq 1$ a $\sum n_i = n$.
Pre fixované hodnoty $q_1, \ldots, q_s$ označíme $BinP_s(m_1, \ldots, m_s), 0 \leq m_i \leq n_i$, hodnotu optimálneho riešenia pre $BinP_s(q_1, \ldots, q_s, m_1, \ldots, m_s)$. Prirodzeným rozsahom problému je $m = \sum_i m_i$.
Riešenie pozostáva z prvkov umiestnených do jednotlivých košov; jedného a zvyšných. Na optimálne riešenie sa teda možno pozrieť ako na správne naplnenie prvého koša a optimálne doriešenie pre zmenšený rozsah problému. Prirodzené je preto dynamické programovanie, ktoré správne naplnenie prvého koša rieši zvažovaním všetkých potenciálnych možností:

Algoritmus 25 A_{BinP_s} - dynamické programovanie pre $BinP_s$

//toto je len vysvetlenie princípu , nie algoritmus samotný

1: $BinP_s(0, \ldots, 0) = 0$

2: $BinP_s(x_1, \ldots, x_s) = 1$ $\forall (x_1, \ldots, x_s) \in \{0, \ldots, n_1\} \times \ldots \times \{0, \ldots, n_s\}$ také, že $\sum_{i=1}^{s} x_i q_i \leq 1$, $\sum_{i=1}^{s} x_i \geq 1$

//vyberáme prvky, ktoré sa zmestia do jedného koša

3: $BinP_s(m_1, \ldots, m_s) =$

$$1 + \min_{(x_1,\ldots,x_s) \in \{0,\ldots,n_1\} \times \ldots \times \{0,\ldots,n_s\}} \{BinP_s(m_1 - x_1, \ldots, m_s - x_s) \mid \sum_{i=1}^{s} x_i q_i \leq 1$$

Úloha: Upravte algoritmus A_{BinP_s}tak, aby vypísal aj obsahy jednotlivých košov optimálneho riešenia.

čas A_{BinP_s}

Počítame $\Delta = n_1 \times \ldots \times n_s$ hodnôt $BinP_s(m_1, \ldots, m_s)$. Výpočet každej z nich trvá $O(\Delta)$. Keďže

$$\Delta = n_1 \times \ldots \times n_s \leq \left(\frac{\sum_{i=1}^{s} n_i}{s} \right)^s = \left(\frac{n}{s} \right)^s$$

časová zložitosť algoritmu je $O\left(\left(\frac{n}{s} \right)^{2s} \right)$

bez malých vstupov

2. Ako druhý uvažujme taký prípad, keď vstup neobsahuje prvky veľkosti menšej ako ϵ: $I = (r_1, \ldots, r_n)$, pričom $r_i \geq \epsilon$, $0 < \epsilon < 1$. "Zaokrúhlením" prerobíme tento vstup na taký, ktorý bude mať len s rôznych hodnôt a bude sa naň teda dať aplikovať algoritmus A_{BinP_s}: zoberieme postupnosť $\ell_1, \ldots, \ell_s$ a hodnotu z intervalu $[\ell_i, \ell_{i+1})$ "zaokrúhlime" na ℓ_i. Hodnoty s, ℓ_i volíme tak, aby:

$$\ell_1 = \epsilon, \quad \ell_{s+1} = 1, \quad \ell_i = \ell_{i-1}(1+\epsilon), \text{ čo pre dané } \epsilon \text{ dáva } s = \left\lceil \frac{\log_2(1/\epsilon)}{\epsilon} \right\rceil$$

Potom aplikujeme dynamické programovanie.

Algoritmus 26 A_{BinP_ϵ} - dynamické programovanie pre vstup bez malých prvkov

1: $(r_1, \ldots, r_n)$, $\epsilon < r_1 \leq r_2 \leq \ldots \leq r_n \leq 1$
2: $s \leftarrow \left\lceil \frac{\log_2(1/\epsilon)}{\epsilon} \right\rceil$
3: $\ell_1 = \epsilon$
4: **for** i=2 to s **do**
5: $\quad \ell_i \leftarrow \ell_{i-1}(1+\epsilon)$
6: $\ell_{s+1} \leftarrow 1$
7: **for** i=1 to s **do**
8: $\quad L_i \leftarrow \{r_1, \ldots, r_n\} \cap [\ell_i, \ell_{i+1})$
9: $\quad n_i \leftarrow |L_i|$
10: $\quad$ **for all** $r \in L_i$ **do** $r \leftarrow \ell_i$
11: $BinP_s(\ell_1, \ldots, \ell_s; n_1, \ldots, n_s)$

čas

Keďže $s \leq n$, kroky 1 až 10 trvajú $O(n^2)$. Dynamické programovanie na riadku 11 trvá

$$O\left(\left(\frac{n}{s}\right)^{2s}\right) = \left(\left(\frac{n\epsilon}{\log(1/\epsilon)}\right)^{\frac{2\log_2(1/\epsilon)}{\epsilon}}\right)$$

kvalita

Aby sme mali h-dual PTAS potrebujeme ukázať, že

1. $cena(A_{BinP_\epsilon}(I)) \leq opt(I)$; toto je zrejmé, pretože hodnotu r_i sme "zaokrúhľovaním" mohli len zmenšiť.
2. Odhadnime veľkosť jednotlivých košov, keď do nich budeme ukladať objekty *pôvodnej* veľkosti. Uvedomme si, že vzhľadom k veľkosti r_i platí, že počet kusov v koši je nanajvýš $1/\epsilon$. Pre jednotlivé koše preto

$$\sum_{i=1}^{s} x_i \leq \left\lfloor \frac{1}{\epsilon} \right\rfloor \tag{5.1}$$

Nech $(x_1, \ldots, x_s)$ charakterizuje obsah koša S podľa algoritmu 26. Potom

$$\begin{aligned} \sum_{\mathbf{j \in S}} \mathbf{r_j} &\leq \sum_{i=1}^{s} x_i \ell_{\mathbf{i+1}} \\ &= \sum_{i=1}^{s} x_i \ell_i (1+\epsilon) \\ &= (1+\epsilon) \sum_{i=1}^{s} x_i \ell_i \\ &\leq \mathbf{1} + \epsilon \end{aligned}$$

◇

A teraz už h-dual PTAS pre všeobecný prípad dostaneme jednoducho - najprv malé prvky odstránime, použijeme algoritmus A_{BinP_ϵ} a potom malé prvky greedy prístupom pridáme k priebežnému výsledku.

[4] tzv. constraint distance

Algoritmus 27 h-PTAS$_\epsilon$ pre BinP

vstup: $(I, \epsilon), I = (r_1, \ldots, r_n), 0 \leq r_1 \leq \ldots \leq r_n \leq 1$

1: Nájdi i také, že $0 < r_1 \leq \ldots \leq r_i \leq \epsilon \leq r_{i+1} \leq \ldots \leq r_n \leq 1$

2: A_{BinP_ϵ} na vstupe $(r_{i+1}, \ldots, r_n)$; výstupom sú množiny indexov $S_1, \ldots, S_m$

3: greedy prístupom dosyp $r_1, \ldots, r_i$ do existujúcich košov s obsahom menším ako 1 a prípadne ďalších

časová zložitosť je zrejme polynomiálna od n *čas*

Potrebujeme vyargumentovať, že *kvalita*

1. $cena(h\text{-PTAS}_\epsilon(I)) \leq cena(opt(I))$. Vieme, že $cena(opt(r_{i+1}, \ldots, r_n)) \geq cena(A_{BinP_\epsilon})$. Ak sme na pridanie $r_1, \ldots, r_i$ potrebovali pridať kôš veľkosti $1+\epsilon$, bolo treba kôš aj v optimálnom prípade, pretože mali všetky obsah väčší ako 1.
2. veľkosť koša je nanajvýš $1 + \epsilon$, čo sme v priebehu algoritmu dodržiavali

5.4.2 PTAS pre MakeSpan

Nech $I = (p_1, \ldots, p_n)$. Ako sme už povedali,

$$opt\text{-}BinP(\tfrac{p_1}{d}, \ldots, \tfrac{p_n}{d}) \leq m \Leftrightarrow opt - MS(I, m) \leq d$$

Ak teda vieme, že optimálne riešenie pre $MS(I)$ je d^*, potom vieme, že

$$cena(A_{BinP_\epsilon}(\tfrac{p_1}{d^*}, \ldots, \tfrac{p_n}{d^*})) \leq opt\text{-}BinP(\tfrac{p_1}{d^*}, \ldots, \tfrac{p_n}{d^*}) \leq m$$

Keďže kôš v riešení algoritmu h-PTAS$_\epsilon$ pre BinP má veľkosť $1 + \epsilon$, toto riešenie implikuje existenciu riešenia pre MS s hodnotou $(1 + \epsilon)d^*$

Hodnotu d^* ale nepoznáme, preto ju musíme nejako odhadnúť. Budeme ju hľadať binárnym vyhľadávaním. Na začiatku určíme hodnotu dolného a horného odhadu na d^*, DO, HO a potom budeme postupne zmenšovať veľkosť intervalu, pričom to, či posunieme DO alebo HO určíme na základe kvality riešenia BinP s príslušnou hodnotou. Skončíme, keď interval, v ktorom sa nachádza optimálna hodnota, bude dostatočne malý. *ALE*

Je zrejmé, že najmenší možný čas pre MS je určený maximálnou, resp. priemernou hodnotou p_i: *DO, HO*

$$DO \leftarrow \max\left\{\frac{\sum_{i=1}^{n} p_i}{m}, \max\{p_1, \ldots, p_n\}\right\}$$

Horný odhad určíme podľa analýzy greedy batohu

$$opt_{MS}(I, m) \leq \frac{1}{m}\sum_{i=1}^{n} p_i + \max\{p_1, \ldots, p_n\} \leq HO$$

Vzhľadom k predchádzajúcim úvahám je zrejmé, že časová zložitosť algoritmu 28 PTAS pre MS polynomiálna od n. *čas*

Čo vieme povedať o kvalite získaného riešenia? *kvalita*
• Počas výpočtu algoritmu držíme hodnotu d^* v intervale $[DO, HO]$. Zabezpečujeme to testom na riadku 7

Algoritmus 28 PTAS pre MS

vstup: $(I, m), \epsilon, I = (p_1, \ldots, p_n)$

1: $DO \leftarrow \max\left\{\frac{\sum_{i=1}^{n} p_i}{m}, \max\{p_1, \ldots, p_n\}\right\}$
2: $HO \leftarrow 2DO$
3: $k \leftarrow \lceil \log_2(4/\epsilon) \rceil$ ▷ vhodnosť voľby uvidíme neskôr
4: **for** i=1 to k **do**
5: $\quad d \leftarrow \frac{DO+HO}{2}$
6: $\quad c \leftarrow cena(h\text{-PTAS}_\epsilon(\frac{p_1}{d}, \ldots, \frac{p_n}{d}))$
7: $\quad$ **if** $c > m$ **then** $DO \leftarrow d$
8: $\quad$ **else** $HO \leftarrow d$
9: $d^* \leftarrow HO$
10: $h\text{-PTAS}_\epsilon(\frac{p_1}{d^*}, \ldots, \frac{p_n}{d^*})$

• Z riešenia algoritmu $h\text{-}PTAS_{\epsilon/2}(\frac{p_1}{d^*}, \ldots, \frac{p_n}{d^*})$ vieme skonštruovať prípustné riešenie problému MS, o kvalite ktorého platí, že je nanajvýš $\left(1 + \frac{\epsilon}{2}\right) d^*$. (ako?)

• V každom behu cyklu sme veľkosť intervalu zmenšili na polovicu. Preto po k krokoch bude o veľkosti intervalu platiť:

$$HO - DO = \frac{DO}{2^k} \quad \rightsquigarrow d^* = HO = DO(1 + 1/2^k) \leq \left(1 + \frac{1}{2^k}\right) opt_{MS}(I, m)$$

• Úpravami dostávame:

$$\begin{array}{rcl} \frac{d^* - opt_{MS}(I,m)}{opt_{MS}(I,m)} & \leq & \frac{1}{2^k} \leq \frac{\epsilon}{4} \\ & \downarrow & \\ d^* & \leq & opt_{MS}\left(1 + \frac{\epsilon}{4}\right) \end{array}$$

$$\begin{array}{rcl} cena(h\text{-}PTAS_{\epsilon/2}(\frac{p_1}{d^*}, \ldots, \frac{p_n}{d^*})) & & \\ & \leq & \left(1 + \frac{\epsilon}{2}\right) d^* \leq \left(1 + \frac{\epsilon}{2}\right)\left(1 + \frac{\epsilon}{4}\right) opt_{MS}(I, m) \\ & = & (1 + \frac{\epsilon}{2} + \frac{\epsilon}{4} + \frac{\epsilon^2}{8}) opt_{MS}(I, m) \\ & \leq & (1 + \epsilon) opt_{MS}(I, m) \end{array}$$

Výsledkom algoritmu 28 je riešenie, ktoré je nanajvýš $(1+\epsilon)$-násobkom optimálneho. Jeho čas je polynomiálny od $|I|$, preto je to PTAS pre problém MakeSpan.

□

5.5 Neaproximovateľnosť*

Podobne ako pri dôkaze nerozhodnuteľnosti či NP-úplnosti, aj v prípade neaproximovateľnosti problémov sa výsledky získavajú redukciami. Spomenieme dve techniky:

- redukcia na NP-ťažké problémy

  keby sme mali rozumný aproximačný algoritmus, vedeli by sme v polynomiálnom čase riešiť NP-ťažký problém

- aproximáciu zachovávajúce redukcie

  výsledok o neaproximovateľnosti prenesieme pomocou takej redukcie, ktorá umožňuje hovoriť o aproximácii

K optimalizačnému problému U môžme priradiť problém c-$app(U)$, $f(n)$-$app(U)$, keď pre vstupné x chceme také prípustné riešenie y problému U, $y \in Sol(x)$, ktorého aproximačný pomer je nanajvýš c, resp. $f(|y|)$.

redukcia na NP-ťažký problém

Pri použití metódy redukcie na NP-ťažký problém máme dva problémy: optimalizačný problém U a NP-ťažký problém L. Pre konštantu $c \in \mathbb{R}^{>1}$ nájdeme transformáciu F takú, že existuje funkcia $f_F : N \to N$ tak, že

- $\forall x \in L$ obsahuje $Sol(F(x))$ riešenie y s cenou $cena(y) \leq f_F(|x|)$
- $\forall x \in \Sigma^* - L$ má každé riešenie $y \in Sol(F(x))$ cenu $cena(y) > cf_F(|x|)$

Hodnota $cena(y)$ tak poskytuje informáciu o tom, či x patrí alebo nepatrí do L. Metódu dôkazu neaproximovateľnosti redukciou na NP-ťažký problém sme už použili, keď sme dokazovali, že TSP nemá aproximačný algoritmus s konštantným aproximačným pomerom redukciou na problém HK. Nech $G = (E, V)$ je graf, u ktorého riešime existenciu HK. Pre ľubovoľnú konštantu d, potenciálny aproximačný pomer d-aproximačného algoritmu riešiaceho TSP, vytvoríme inštanciu TSP

$$c(e) = \begin{cases} 1, & \text{pre } e \in E \\ \\ (d-1)|V| + 2, & \text{pre } e \notin E \end{cases}$$

Ľahko vidno, že ak A je d-aproximačný algoritmus, tak

ak G má HK	$\Rightarrow$	optimálne riešenie má cenu $\|V\|$, $cena((A(x))) \leq d\|V\|$
ak G nemá HK	$\Rightarrow$	optimálne riešenie obsahuje aspoň jednu hran s cenou $(d-1)\|V\| + 2$
		$cena(A(x)) \leq (d-1)\|V\| + 2 + \|V\| - 1 = \|V\|d + 1$

Na základe hodnoty, vypočítanej algoritmom A teda vieme určiť, či graf G obsahuje HK alebo nie.

Môžme to potiahnuť ešte ďalej

$$c(e) = \begin{cases} 1, & \text{pre } e \in E \\ \\ |V| \cdot 2^{|V|} + 1, & \text{pre } e \notin E \end{cases}$$

Hodnotu $|V| \cdot 2^{|V|}$ uložíme do $|V| + \log |V|$ bitov, takže transformácia bude polynomiálna, vytvorený graf je veľkosti $O(|V|^3)$. Pritom v prípade, že G neobsahuje HK, pre cenu riešenia platí

$$cena \geq |V| \cdot 2^{|V|} + 1 + |V| - 1 = |V| \left(2^{|V|} + 1\right) > 2^{|V|}|V|$$

Ak teda $\mathsf{P} \neq \mathsf{NP}$, tak pre problém TSP neexistuje polynomiálny algoritmus s aproximačným pomerom $2^{\Omega(\sqrt[3]{n})}$

□

AP redukcia

Uvažujme triedu APX optimalizačných problémov, pre ktoré existujú polynomiálne algoritmy s aproximačným pomerom ohraničeným konštantou. Pre túto triedu hrá existencia PTAS podobnú úlohu ako NP-úplnosť pre triedu NP.

$$\text{APX} = \{U \in NPO \mid \exists \text{ polynomiálny } c\text{-}app \text{ algoritmus pre } U\}$$

Potrebujeme tak v polynomiálnom čase vypočitateľnú redukciu R

$$U_1 \overset{R}{\rightsquigarrow} U_2,\ U_1 \text{ nemá PTAS } \Rightarrow\ U_2 \text{ nemá PTAS}$$

AP redukcia

$U_1 \leq_{AP} U_2$ práve vtedy, ak existujú v polynomiálnom čase vypočitateľné funkcie F, H a konštanta $\alpha > 0$

$F : \Sigma^*_{I_1} \times Q^+ \to \Sigma^*_{I_2}$,
$H : \Sigma^*_{I_1} \times Q^+ \times \Sigma^*_{O_2} \longrightarrow \Sigma^*_{O_1}$

1. $\forall x_1 \in I_1\ Sol(x_1) \neq \emptyset\ x_2 = F(x_1, \epsilon) \in I_2$
2. $\forall x_1 \in I_1, \epsilon \in Q^+, y_2 \in Sol(F(x_1), \epsilon)$ platí $y_1 = H(x_1, \epsilon, y_2) \in Sol(x_1)$
3. F, H sú vypočitateľné v čase polynomiálnom od $|x|, |y|$
4. časová zložitosť pre F, H je nerastúca od ϵ pri fixovanej veľkosti $|x_1|, |y_2|$
5. $\forall x_1 \in I_1,\ \epsilon \in Q^+,\ y_2 \in Sol_2(F(x_1, \epsilon))$ platí:

$$\max\left\{\frac{opt_{U_2}(F(x_1,\epsilon))}{cena_2(y_2)}, \frac{cena_2(y_2)}{opt_{U_2}(F(x_1,\epsilon))}\right\} \leq 1 + \epsilon$$

$$\max\left\{\frac{cena_1(H(x_1,\epsilon,y_2))}{opt_1(x_1)}, \frac{opt_1(x_1)}{cena_1(H(x_1,\epsilon,y_2))}\right\} \leq 1 + \alpha\epsilon$$

$$\begin{array}{ccc} x_1 & \xrightarrow{F} & x_2 \\ \vdots & & \downarrow \\ y_1 & \xleftarrow{H} & y_2 \end{array}$$

Nasledujúca jednoduchá lema je základom pre používanie AP redukcie.

Lema 5.14 *Ak existuje PTAS pre U_2, $U_1 \leq_{AP} U_2$, potom existuje PTAS pre U_1*

Dôkaz: Ak algoritmus (PTAS) A pracuje s chybou $1+\epsilon$, potom chyba po výpočte $x_1 \xrightarrow{F} x_2 \xrightarrow{A} y_2 \xrightarrow{H} \rightsquigarrow y_1$ je $1+\alpha\epsilon$. Ak teda chceme dostať PTAS s chybou δ, musíme/stačí od PTAS pre U_2 žiadať chybu $\epsilon = \delta \setminus \alpha$.

□

Definícia 5.6 *Hovoríme, že problém U je APX-úplný, ak*
– $U \in APX$
– $\forall\ W \in APX,\ W \leq_{AP} U$

Príkladom APX-redukcie je redukcia MaxSAT na kliku. Stačí spraviť polynomiálnu redukciu F na G, použitú pri dôkaze NP-úplnosti problému kliky. Po redukciu dostaneme graf G, pre ktorý platí

k klauzúl v F je splniteľných práve vtedy, ak graf G obsahuje kliku o veľkosti k

Konštanta α je teda v tomtom prípade rovná 1.

Časť o aproximačných algoritmoch ukončíme GP-redukciou.

$GAP_{c,s}(U)$

Definícia 5.7 *Nech $0 \leq s < c \leq 1$, U optimalizačný problém. $GAP_{s,c}(U)$ je nasledujúci rozhodovací problém:*

vstup: $x \in I$: $opt_U(x)/|x| \geq c$ *alebo* $opt_U(x)/|x| < s$
výstup: *"ano"* $\iff opt_U(x)/|x| \geq c$
"nie" $\iff opt_U(x)/|x| < s$

Ľahko sa dá dokázať nasledujúca lema(dokážte)

Lema 5.15 *$U \in \mathsf{NPO}, 0 \leq s \leq c \leq 1$. Ak $GAP_{c,s}(U)$ je NP-ťažký, potom za predpokladu, že $\mathsf{P} \neq \mathsf{NP}$ neexistuje c/s-aproximačný algoritmus pre U.*

No a teraz tá redukcia.

GP-redukcia

Definícia 5.8 *Nech U_1, U_2 sú optimalizačné problémy. Hovoríme, že transformácia $\xrightarrow{GP}$ je GP-redukcia s parametrami $s, c, s', c'; 0 \leq s \leq c \leq 1,\ 0 \leq s' \leq c' \leq 1$, ak*

- $x \in I_1, A(x) \in I_2$
- $\frac{opt_1}{|x|} \geq c \Longrightarrow \frac{opt_2(A(x))}{A(x)} \geq c'$
- $\frac{opt_1}{|x|} < s \Longrightarrow \frac{opt_2(A(x))}{A(x)} < s'$

Ako sa táto redukcia využíva? Majme dva optimalizačné problémy U_1, U_2, $U_1 \xrightarrow{GP} U_2$. Ak $GAP_{c,s}(U_1)$ je NP-ťažký, potom $GAP_{c,s}(U_2)$ je tiež NP-ťažký.

Kapitola 6

Pravdepodobnostné algoritmy

Pokračujeme v riešení problémov, ktoré sú ťažké. Treba rozlišovať medzi *neviem* a *nedá sa*

- To, že *nevieme* nejaký problém riešiť lepšie znamená, že momentálne nemáme lepší algoritmus. Nehovorí to nič o tom, či lepší algoritmus v princípe môže alebo nemôže existovať.
- Naproti tomu tvrdenie, že sa niečo *nedá* lepšie riešiť v sebe zahŕňa existenciu dôkazu, že to lepšie nejde.

Formálne sú problémy v NPÚ také, ktoré *nevieme* riešiť deterministickými polynomiálnymi algoritmami. Napriek tomu to v praxi znamená, že k nim pristupujeme tak, akoby sa riešiť lepšie *nedali*.

Randomizácia sa stala štandardným prístupom v návrhu algoritmov. Takéto algoritmy sú zaujímavé hlavne kvôli svojej efektívnosti a jednoduchosti. A to aj napriek strate stopercentnej spoľahlivosti – riešenie s malou pravdepodobnosťou nie je korektné, resp. ho nedostaneme.

Kľúčovým pojmom pre tvorbu a pochopenie pravdepodobnostných (randomized) algoritmov je pojem náhody. Pri definovaní/popisovaní pojmu náhody máme väčšinou tendenciu povedať, že náhoda/náhodná je taká udalosť, ktorej výsledok nevieme predpovedať.

6.1 Príklady

Začnime niekoľkými príkladmi pravdepodobnostných algoritmov.

6.1.1 Rovnosť databáz

Majme dva vzdialené počítače, pričom každý z nich udržiava databázu údajov. Pre jednoduchosť databázou údajov rozumieme binárny reťazec; databázu jednotlivých počítačov reprezentujeme reťazcami $X = x_1x_2 \dots x_n$, resp. $Y = y_1y_2 \dots y_n$. Komunikáciou medzi počítačmi potrebujeme zistiť, či sú tieto "databázy" totožné. Keďže prirodzene vyžadujeme korektný prenos správ, snažíme sa o minimalizáciu komunikácie medzi počítačmi; chceme protokol garantujúci čo najmenší počet prenesených bitov.

jednosmerná komunikácia

Kvôli jednoduchosti predpokladáme len jedno kolo komunikácie - prvý počítač pošle správu druhému a ten rozhodne, či sú databázy rovnaké. Dá sa dokázať, že pri deterministickom prístupe nič lepšie ako poslanie celej informácie – teda celého bi-

nárneho reťazca X dĺžky n – jedného počítača tomu druhému v princípe nenájdeme. Vedeli by ste to dokázať?

Ak však pripustíme, že nie každá odpoveď musí byť na 100% korektná, môžme použiť jednoduchý pravdepodobnostný algoritmus 29; R_1 (R_2) označuje protokol prvého (druhého) počítača. Idea algoritmu je jednoduchá - binárne reťazce budeme považovať za binárne čísla a budeme ich porovnávať na základe zvyšku po delení náhodným prvočíslom[1]: rôzny zvyšok po delení garantuje, že porovnávané čísla (reťazce) sú rôzne, rovnaký zvyšok po delení nám však 100% istotu nedáva. (prečo?) Ukážeme, že rozumná voľba prvočísla, teda množiny, z ktorej prvočíslo náhodne vyberáme, poskytuje malú pravdepodobnosť chybnej odpovede.

Algoritmus 29 základný RP

//počítač R_x má reťazec X, počítač R_y reťazec Y

protokol pre začínajú R_x

1: náhodne vyber prvočíslo $p \in \{0, \ldots, n^2\}$
2: nech $num(X)$ označuje číslo s binárnym zápisom X;
3: vypočítaj číslo $RX \leftarrow num(X) \mod p$
4: pošli $\langle RX, p\rangle$ druhému počítaču

protokol pre prijímajúce R_y

1: po prijatí $\langle RX, p\rangle$ vypočítaj $RY \leftarrow num(Y) \mod p$
2: **if** $RX = RY$ **then** return "$X = Y$"
3: **else** return "$X \neq Y$"

počet bitov

Analýzu algoritmu začneme analýzou počtu prekomunikovaných bitov. Počas spoločného výpočtu podľa základného algoritmu RP prekomunikovali tieto dva počítače toľko bitov, koľko bolo treba na prenos čísel RX, p. Vzhľadom k tomu, že $RX \leq p < n^2$, je dĺžka posielanej správy zrejme

$$2 \cdot \lceil \log_2 n^2 \rceil \leq 4 \cdot \lceil \log_2 n \rceil$$

Pri hodnote $n = 10^{16}$ je dĺžka prekomunikovanej správy nanajvýš $4 \cdot 16 \cdot \lceil \log_2 10 \rceil = 256$. Správu takejto veľkosti dokážeme preniesť bezpečne.

korektnosť

Ľahko vidno, že negatívna odpoveď $X \neq Y$ je vždy pravdivá; rovnaké čísla nemôžu mať rôzny zvyšok po delení rovnakým číslom. V prípade pozitívnej odpovede $X = Y$ to však neplatí. Môže sa stať, že aj keď sú reťazce X, Y, a teda čísla $num(X), num(Y)$, rôzne, zvyšky po delení prvočíslom p majú rovnaké.

$$num(X) \bmod p = num(Y) \bmod p \iff |num(X) - num(Y)| \bmod p \equiv 0$$

Aby sme určili pravdepodobnosť, s akou sme sa dopustili chyby v prípade pozitívnej odpovede, spočítame počet tzv. "zlých" prvočísel; zlým prvočíslom máme na mysli to, ktoré bezo zvyšku delí rozdiel $|num(X) - num(Y)|$. Potom hľadaná pravdepodobnosť je

$$\frac{\text{počet zlých prvočísel}}{\text{počet všetkých možných prvočísel}} \tag{6.1}$$

zlé prvočísla

Začnime ohraničením počtu zlých prvočísel. Prvočíslo p je zlé vzhľadom na X, Y ak $|num(X) - num(Y)|$ je deliteľné p. Keďže oba binárne reťazce X, Y sú dĺžky n, môžme odhadnúť veľkosť rozdielu $|num(X) - num(Y)|$ nasledovne

$$W = |num(X) - num(Y)| < 2^n$$

[1] prvočíslo vyberáme náhodne spomedzi prvočísel menších ako n^2

Každé zlé prvočíslo delí W, preto horný odhad na počet zlých prvočísel môžme odhadnúť na základe faktorizácie; p_i označuje i-te prvočíslo:

$$W = p_1^{i_1} \cdot p_2^{i_2} \cdot \ldots \cdot p_k^{i_k}, 2 \leq p_1 < p_2 < \ldots < p_k$$

Uvedomme si, že maximálna možná hodnota k je horným odhadom na počet zlých prvočísel. Spojením

$$2^n > W = p_1^{i_1} \cdot \ldots \cdot p_k^{i_k} \geq p_1 \cdot \ldots \cdot p_k \geq 2^k$$

pre počet k zlých prvočísel dostávame

$$\boxed{k \leq n-1}$$

odhad počtu prvočísel

Označme $PRIM(m)$ množinu prvočísel nanajvýš veľkosti m a $Prim(m)$ jej veľkosť; $Prim(m) = |PRIM(m)|$. Predstavu o hodnote $Prim(m)$ dáva Lema 6.1, ktorú uvádzame bez dôkazu.

Prim(m)

Lema 6.1

$$\lim_{m \to \infty} \frac{Prim(m)}{m/\ln m} = 1$$

$$Prim(m) > \frac{m}{\ln m}, \; m > 67$$

Dosadením odhadu pre $Prim(n^2)$ do (6.1) dostávame

$$\frac{\text{počet zlých prvočísel}}{\text{počet všetkých možných prvočísel}} = \frac{n-1}{Prim(n^2)} < \frac{n-1}{n^2/\ln n^2} \leq \frac{\ln n^2}{n} = \frac{2\ln n}{n}$$

Pravdepodobnosť chybnej odpovede pre $X \neq Y$ je teda nanajvýš $\frac{2\ln n}{n}$, čo pre $n = 10^{16}$ je nanajvýš $0.36892 \cdot 10^{-14}$.

Pravdepodobnosť nesprávnej odpovede môžme znížiť niekoľkonásobným *nezávislým* zopakovaním výberu prvočísla v základnom algoritme RP. Pre počet opakovaní 10 dostaneme protokol RP_{10}.

Algoritmus 30 RP_{10}

1: R_1
2: náhodne vyberie 10 prvočísel $p_1, \ldots p_{10}$ v intervale $0..n^2$
3: nech $num(X)$ označuje číslo s binárnym zápisom X;
4: vypočíta 10 čísel $RX_i \leftarrow num(X) \bmod p_i, \; 1 \leq i \leq 10$
5: pošle $RX_1, \ldots, RX_{10}, p_1, \ldots, p_{10}$ druhému počítaču
6: R_2
7: po prijatí $RX_1, \ldots, RX_{10}, p_1, \ldots, p_{10}$ druhý počítač vypočíta
8: $RY_i \leftarrow num(X) \bmod p_i, \; 1 \leq i \leq 10$
9: **if** $\forall i, \; RX_i = RY_i$ **then** return "$x = y$"
10: **else** return "$x \neq y$"

Počet prenesených bitov/komunikačná zložitnosť narástla desaťnásobne. Čo vieme povedať o pravdepodobnosti chyby? Nesprávnu odpoveď dáva algoritmus RP_{10} iba v prípade, ak všetkých 10 porovnaní $RX_i \stackrel{?}{=} RY_i$ dopadne pozitívne, hoci $X \neq Y$. Keďže 10 prvočísel sme vyberali *nezávisle*, pre pravdepodobnosť chyby platí

$$\left(\frac{n-1}{Prim(n^2)}\right)^{10} \leq \left(\frac{\ln n^2}{n}\right)^{10} = \frac{2^{10} \cdot (\ln n)^{10}}{n^{10}}$$

Pre $n = 10^{16}$ je pravdepodobnosť nesprávnej odpovede nanajvýš $0.4717 \cdot 10^{-141}$.

Cvičenie 6.1 *Napíšte program, ktorý pre vstupnú hodnotu dĺžky databáz n a hodnotu chyby ϵ určí*
(a.) počet opakovaní k tak, aby pravdepodobnosť chyby pri použití algoritmu RP_k bola nanajvýš ϵ

(b.) hodnotu m tak, aby pri použití základného algoritmu RP, v ktorom $PRIM(n^2)$ nahradíme $PRIM(m)$, bola pravdepodobnosť chyby nanajvýš ϵ

6.1.2 Existuje také j, že $x_j = y_j$?

Tentokrát si naše vzdialené počítače udržiavajú postupnosť, povedzme 10, reťazcov

$$\begin{array}{ll} \mathsf{RI} & x_1, \ldots, x_{10} \quad x_i \in \{0,1\}^n \\ \mathsf{RII} & y_1, \ldots, y_{10} \quad y_i \in \{0,1\}^n \end{array}$$

Zaujíma nás, či existuje taký index j, pre ktorý $x_j = y_j$. Opäť sa dá dokázať, že deterministický protokol vyžaduje prekomunikovanie informácie veľkosti $10n$. Pritom pravdepodobnostnému algoritmu, opäť využívajúcemu zvyšky po delení prvočíslom, stačí prekomunikovať menej.

"neviem"

Namiesto klamania dovolíme algoritmu, aby v situácii, keď si svojou odpoveďou nie je istý, povedal "neviem". Vyžadujeme však, aby to nerobil príliš často. Takémuto typu algoritmov hovoríme Las Vegas.

Algoritmus 31 — Existuje j, $x_j = y_j$?

RI:

1: náhodne zvoľ 10 prvočísel $p_1, \ldots, p_{10}$ v intervale $0..n^2$
2: $s_i \leftarrow x_i \bmod p_i$
3: pošli $p_1, \ldots, p_{10}, s_1, \ldots, s_{10}$ počítaču RII

RII:

1: $s'_i \leftarrow y_i \bmod p_i$
2: **if** $s_i \neq s'_i$ pre všetky i **then** return "nie"
3: **else** nech j je minimálne také, že $s_j = s'_j$
4: pošli počítaču RI j, y_j

RI:

1: **if** $x_j = y_j$ **then** odpoveď je "áno"
2: **else** odpoveď je "neviem"

Ľahko vidno, že počítače prekomunikovali $20(2\lceil \log n \rceil) + \log 10 + n$ bitov.

analýza chyby

Analýzu chyby algoritmu rozdelíme na dva prípady podľa toho, či hľadaný index j existuje alebo nie.

$\boxed{\forall \mathbf{i}\ \mathbf{x_i \neq y_i}}$

Začnime prípadom, keď korektná odpoveď algoritmu má byť "nie". Už vieme, že napriek tomu, že čísla X, Y dĺžky n sú rôzne, s pravdepodobnosťou $\frac{\ln n^2}{n} = \frac{2 \ln n}{n}$ sa môže stať, že pre náhodne zvolené prvočíslo $p \leq n^2$ sa ich zvyšky po delení p rovnajú: $X \mod p \equiv Y \mod p$. My volíme náhodne 10 prvočísel, preto s pravdepodobnosťou $\left(1 - \frac{2 \ln n}{n}\right)^{10}$ sa to *nestane* pre žiadne z nich; vtedy RII bez ďalšej komunikácie s RI korektne odpovie "nie". Pre pravdepodobnosť Pr korektnej odpovede (v prípade $x_i \neq y_i\ \forall i$) preto platí[2]

[2] Podľa binomickej vety

$$
\begin{aligned}
Pr \quad &\geq \left(1-\tfrac{2\ln n}{n}\right)^{10} = \textstyle\sum_{i=0}^{10}\binom{10}{i}\left(\tfrac{2\ln n}{n}\right)^{i}(-1)^{10-i} \\
&= 1+\textstyle\sum_{i=1}^{5}\binom{10}{2i}\left(\tfrac{2\ln n}{n}\right)^{2i} - \sum_{i=0}^{4}\binom{10}{2i+1}\left(\tfrac{2\ln n}{n}\right)^{2i+1} \\
&\geq 1-\tbinom{10}{1}\tfrac{2\ln n}{n}+ \\
&\underbrace{\binom{10}{10}\left(\frac{2\ln n}{n}\right)^{10}+\sum_{i=1}^{4}\left[\binom{10}{2i}\left(\frac{2\ln n}{n}\right)^{2i}-\binom{10}{2i+1}\left(\frac{2\ln n}{n}\right)^{2i+1}\right]}_{\geq 0} \\
&\geq 1-\tfrac{20\ln n}{n} \quad \geq \quad 1/2
\end{aligned}
$$

V prípade keď existuje nejaké i také, že $x_i = y_i$, je množina indexov, na ktorých sa X, Y zhodujú, neprázdna. Nech j je najmenší z týchto indexov: $x_j = y_j$. Označme E_j udalosť, keď j je najmenšie také, že $x_j \mod p_j = y_j \mod p_j$; inými slovami, RII korektne pošle RI index j a reťazec y_j.

$\exists \mathbf{i}\ \mathbf{x_i} = \mathbf{y_i}$

- ak $j = 1$, RII pošle do RI index $1 = j$; RI úspešne overí, že $x_1 = y_1$ a korektne odpovie
- ak $j > 1$, je nenulová pravdepodobnosť, že RII pošle do RI index j pre ktorý $x_j \neq y_j$. Pravdepodobnosť korektnej odpovede je rovná pravdepodobnosti udalosti E_j, čo je pravdepodobnosť toho, že pre $\forall i < j$ je $x_i \mod p_i \neq y_i \mod p_i$

$$\left(1-\frac{2\ln n}{n}\right)^{j-1} \geq 1-\frac{2(j-1)\ln n}{n} \geq 1-\frac{18\ln n}{n} \geq 1/2, \text{ pre } n \geq 189$$

Nezávisle od toho, či hľadaný index existuje alebo nie, dá algoritmus s pravdepodobnosťou aspoň $1/2$ korektnú odpoveď, v ostatných prípadoch povie "neviem".

6.1.3 $AB \stackrel{?}{=} C$

Uvažujme tri štvorcové matice A, B, C typu $n \times n$. Zaujíma nás, či platí rovnosť

$$AB \stackrel{?}{=} C$$

Deterministický algoritmus založený na Strassenovom násobení matíc je zložitosti $O(n^{\log_2 7})$. Ukážeme, že pravdepodobnostný algoritmus môže byť efektívnejší. Založený je na uvedomení si jednoduchého faktu: ak $AB = C$ tak pre ľubovoľný vektor x platí $ABx = Cx$. Alebo aj: ak existuje vektor x, pre ktorý $ABx \neq Cx$, tak $AB \neq C$.

Algoritmus 32 — Test AB=C

1: náhodne vyber $x \in \{0,1\}^n$
2: **if** $A(Bx) \neq Cx$ **then** return "určite $AB \neq C$"
3: **else** return "asi $AB = C$"

Časová zložitosť – vďaka asociativite násobenia (pri existencii dobrého generátora náhodných čísel) sme zložitosť znížili na $O(n^2)$ aritmetických operácií.

Chyba – chyby sa dopustíme v tom prípade, ak $AB \neq C$ a pritom pre náhodne zvolený vektor x platí $ABx = Cx$. Odhadnime pravdepodobnosť toho, že sa tak stane.

Lema 6.2 *Nech $AB \neq C$, $x \in \{0,1\}^n$. Potom $Pr[ABx = Cx] \leq 1/2$*

Dôkaz: Ak $AB \neq C$, potom $AB - C \neq 0$. Existujú teda indexy i, j tak, že $(AB - C)[i, j] \neq 0$. Nech $(d_1, \ldots, d_n)$ označuje $i-$ty riadok matice $D = AB - C$. Potom

$$\begin{array}{rcl} Pr[(AB - C)x = 0] & \leq & Pr\left[\sum_{k=1}^{n} d_k x_k = 0\right] \\ & = & Pr[x_j = \frac{-1}{d_j} \sum_{k \neq j} d_k x_k] \quad = \quad 1/2 \end{array}$$

Využili sme, že $d_j \neq 0$ a $Pr[x_i = 0] = Pr[x_i = 1] = 1/2$

Ak algoritmus skončí s odpoveďou "asi", môžme ho zopakovať. Pri $k-$násobnom zopakovaní sa časová zložitosť zvýši o multiplikatívnych k. Pritom $k-$ násobné zopakovanie algoritmu — $k-$násobné získanie odpovede "asi" — znamená chybu nanajvýš $1/2^k$.

□

Uvedený algoritmus je rýchly, jedným smerom — $AB \neq C$ — hovorí korektne, chyby sa môže dopustiť len pri odpovedi "asi $AB = C$". Takýto typ algoritmu sa volá Monte Carlo.

6.2 Klasifikácia náhodou riadených algoritmov

Pri klasifikácii pravdepodobnostných algoritmov si všímame dva aspekty. Jeden z možných pohľadov na klasifikáciu pravdepodobnostných algoritmov je ten, keď si všímame **umiestnenie pravdepodobnosti** v nich.

umiestnenie pravdepodobnosti

I. modelom pravdepodobnostného algoritmu je pravdepodobnostné rozdelenie nad množinou deterministických stratégií

II. pravdepodobnostný algoritmus modelujeme nedeterministickým algoritmom s pravdepodobnostným rozdelením nad nedeterministickými voľbami

Druhým aspektom je chyba algoritmu. Pri uvažovaní o **type a veľkosti chyby** sa vlastne snažíme charakterizovať praktickosť algoritmov. Rozlišujeme 2 základné modely chýb

typ chyby

Monte Carlo algoritmy dajú odpoveď vždy, s rozumnou pravdepodobnosťou však nemusí byť korektná

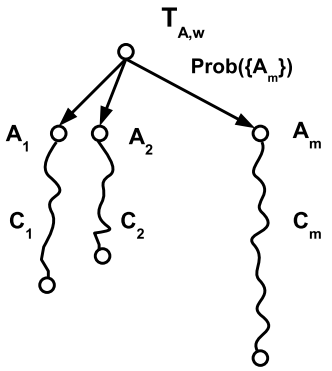
Las Vegas algoritmy sú také, keď každá odpoveď je správna, pripúšťame však alebo odpoveď "neviem" v rozumnom čase, alebo potenciálne nekonečné výpočty.

Zaujímavé tiež je, akým počtom opakovaní vieme—a či vôbec—dosiahnuť požadovanú spoľahlivosť.

6.2.1 Umiestnenie pravdepodobnosti — I. model

Pravdepodobnostný algoritmus vnímame ako pravdepodobnostné rozdelenie $Prob$ nad konečnou množinou deterministických stratégií $A_1, \ldots, A_m$.

- pre vstup w náhodne zvolíme i a realizujeme výpočet $C_i = A_i(w)$, ktorého časová zložitosť je $Time(C_i)$. Náhodný výber i odpovedá pravdepodobnostnému rozdeleniu $Prob$
- hýbeme sa v pravdepodobnostnom priestore $(S_{A,w}, Prob)$, kde $S_{A,w} = \{A_1, \ldots, A_m\}$



Čas je náhodná premenná Z, ktorá jednotlivým behom/výpočtom priraďuje dĺžku ich trvania. Pri skúmaní efektívnosti/časovej zložitosti si preto všímame hodnoty tejto premennej.

$\mathbf{Z}: \quad S_{A,w} \to N \qquad Z(C_i) = Time(C_i)$

Na jednom vstupe existuje viacero výpočtov, ktoré sa môžu líšiť dĺžkou trvania. Ako potom hovoríme o zložitosti algoritmu na konkrétnom vstupe? Pribúda pojem *očakávanej zložitosti/očakávaného času* $ExpTime_A(w)$, ktorý je váženým priemerom/strednou hodnotou potenciálnych behov:

očakávaný prípad

$$\mathbf{ExpTime_A(w)} = \sum_{i=1}^{m} Prob[C_i] \cdot Z(C_i) = \sum_{i=1}^{m} Prob[C_i] \cdot Time(C_i)$$

$$\mathbf{ExpTime_A(n)} = \max_{|w|=n}\{ExpTime_A(w)\}$$

$\mathbf{Time_A(n)} = \max_{i,|w|=n}\{Time(A_i(w))\}$

korektnosť

Pri skúmaní pravdepodobnosti korektnej odpovede si pomáhame indikačnou premennou X, ktorej hodnota závisí od správnosti príslušného výpočtu.

$$\mathbf{X(C_i)} = \begin{cases} 1, & \text{ak výpočet } C_i \text{ dáva korektnú odpoveď;} \\ 0, & \text{inak} \end{cases}$$

Pravdepodobnosť korektnej odpovede je potom strednou hodnotou indikačnej premennej X:

$$\begin{aligned} \mathbf{E[X]} &= \textstyle\sum_{i=1}^{m} X(C_i) \cdot Prob[C_i] \\ &= \textstyle\sum_{X(C_i)=1} 1 \cdot Prob[C_i] + \sum_{X(C_i)=0} 0 \cdot Prob[C_i] \\ &= Prob[X = 1] = Prob[\text{A počíta korektný výstup}] \end{aligned}$$

Pravdepodobnosť chyby

$$Error_A(w) = 1 - E[X]$$

$$Error_A(n) = \max_{|w|=n}\{Error_A(w)\}$$

Príklad 6.1 *Analyzujme z tohto pohľadu algoritmus na porovnanie "databáz"*

- *pravdepodobnostný priestor je tvorený množinou behov, pričom jednotlivé behy sú určené voľbou prvočísla* $S_{R,(x,y)} = \{C_p \mid p \in Prim(n^2)\}$
- *uvažujeme rovnomerné rozdelenie, preto* $Prob[C_p] = \frac{1}{Prim(n^2)}$
- *pri analýze chyby si pomôžeme indikačnou premennou* $X(C_p)$*; keďže v prípade* $X = Y$ *k žiadnej chybe nedochádza, význam indikačnejpremennej sa prejaví v prípade* $X \neq Y$*, kedy* $X(C_p) = \begin{cases} 1, & p \text{ je "dobré"} \\ 0, & p \text{ je "zlé"} \end{cases}$
- $\begin{aligned} E[X] &= \textstyle\sum_{p\in Prim(n^2)} X(C_p) \cdot Prob[C_p] \\ &= \textstyle\sum_{p\in Prim(n^2)} X(C_p) \cdot \frac{1}{Prim()n^2} = \frac{1}{Prim(n^2)} \sum_{p \text{ je dobre}} X(C_p) \\ &\geq \frac{1}{Prim(n^2)}(Prim(n^2) - (n-1)) = 1 - \frac{n-1}{Prim(n^2)} \end{aligned}$
- $Error_R((x,y)) = 1 - E[X] \leq \frac{n-1}{Prim(n^2)} \leq \frac{2\ln n}{n}$

Príklad 6.2 (Max-Sat) *Pre vstupnú formulu* $\Phi(x_1, \ldots, x_n)$ *v KNF chceme vypočítať také priradenie vstupnch hodnôt* $\alpha \in \{0,1\}^n$*, ktoré spĺňa maximálny počet klauzúl.*

Ukážeme, že náhodný vektor α *spĺňa s veľkou pravdepodobnosťou "dosť" klauzúl. Preto nasledujúci algoritmus RSAM má zmysel.*

Algoritmus 33 RSAM

vstup: $\Phi = F_1 \wedge F_2 \wedge \ldots \wedge F_m,\ F_i = (l_{i,1} \vee l_{i,2} \vee \ldots \vee l_{i,k})$

1: náhodne $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ ▷ $Pr(\alpha_j = 1) = Pr(\alpha_j = 0) = 1/2$

2: return(α)

Pre vstup α *priradíme každej elementárnej disjunkcii* F_i *indikačnú premenné* $Z_i(\alpha)$*, a celej formule priradíme premennú* $Z(\alpha)$*, ktorá počíta počet splnených klauzúl; keď*

je α zrejmé z kontextu, budeme niekedy písať len Z_i, Z. Zaujíma nás kvalita, resp. hodnota $E[Z]$.

$$Z_i(\alpha) = \begin{cases} 1, & F_i(\alpha) = 1 \\ 0, & F_i(\alpha) = 0 \end{cases} \qquad Z = \sum_{i=1}^{m} Z_i$$

$$E[Z] = E\left[\sum_{i=1}^{m} Z_i\right] = \sum_{i=1}^{m} E[Z_i]$$

Pre výpočet $E[Z]$ potrebujeme odhad na $E[Z_i]$.

- $F_i(\alpha) = 0 \iff \forall j \; l_{i,j} = 0$

  $Pr[F_i(\alpha) = 0] = \left(\frac{1}{2}\right)^k = \frac{1}{2^k}$, *kde k je počet literálov v $F_i(\alpha)$*

- *$E[Z_i]$ je vlastne pravdepodobnosť toho, že sa elementárna disjunkcia $F_i(\alpha)$ vyhodnotí na 1:* $E[Z_i] = 1 - \frac{1}{2^k}$. *Navyše, každá F_i má aspoň jeden literál, $k \geq 1$, preto $E[Z_i] \geq 1/2$*

- $E[Z] = \sum_{i=1}^{m} E[Z_i] \geq \sum_{i=1}^{m} 1/2 = \frac{m}{2}$

Keďže vieme, že očakávaný počet klauzúl, ktoré náhodný vektor spĺňa, je aspoň $m/2$, môžme to využiť na jednoduché hľadanie vektora, ktorý spĺňa aspoň $m/2$ klauzúl.

Algoritmus 34 modif-RSAM

1: náhodne $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$
2: $r \leftarrow |\{i \mid F_i(\alpha) = 1\}|$
3: **if** $r \geq m/2$ **then** return(α)
4: **else** goto 1

Čo platí o časovej zložitosti tohto algoritmu?

6.2.2 Umiestnenie pravdepodobnosti — II. model

Pri tomto type priradíme pravdepodobnosť jednotlivým nedeterministickým voľbám. Príkladom je pravdepodobnostný Quicksort RQS:

Algoritmus 35 RQS(A)

vstup: $A = \{a_1, \ldots, a_n \mid a_i \in (S, <)\}$
1: **if** $A = \{a\}$ **then** return a
2: **else** $b \leftarrow random(A)$
3: $A_< = \{a \in A \mid a < b\}$
4: $A_> = \{a \in A \mid a > b\}$
5: return $RQS(A_<), b, RQS(A_>)$

Čas algoritmu meriame počtom porovnaní, jednotlivé prípady závisia od voľby pivota b v jednotlivých volaniach

- vždy extrém, potom $O(n^2)$
- vždy medián, potom $O(n \log n)$
- riešenie $T(n) \leq T(n/8) + T(7n/8) + n - 1$ je tiež $O(n \log n)$; takéto rozdelenie je dosť pravdepodobné

Nech výstupom je $s_1 < s_2 < \ldots < s_n$. Pre $i < j$ označme

$$X_{i,j}(C) = \begin{cases} 1, & s_i \text{ sa v priebehu C porovnávalo s } s_j \\ 0, & s_i, s_j \text{ sa v priebehu C neporovnávajú} \end{cases}$$

Čas konkréteho výpočtu je počet realizovaných porovnaní: $T(C) = \sum_{i=1}^{n} \sum_{j>i} X_{i,j}(C)$

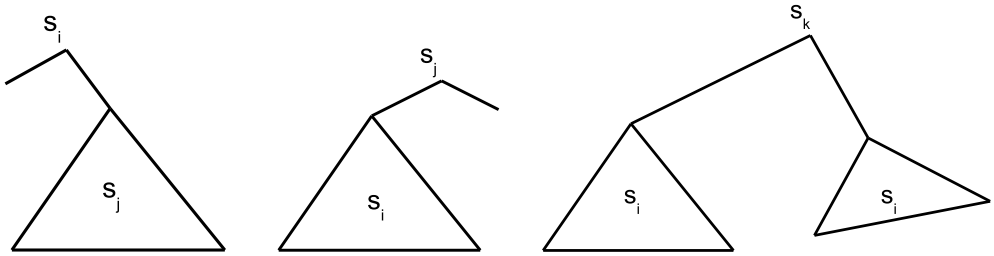
$E[T] = E\left[\sum_{i=1}^{n} \sum_{j>i} X_{i,j}\right] = \sum_{i=1}^{n} \sum_{j>i} E[X_{i,j}]$

$E[X_{i,j}]$

Nech $p_{i,j}$ označuje pravdepodobnosť, že sa s_i, s_j porovnávajú; zrejme $E[X_{i,j}] = p_{i,j}$. Dva prvky sa porovnávajú len vtedy, ak je jeden nich pivotom pri delení množiny, v ktorej sa nachádza aj ten druhý prvok.

$$\underbrace{s_1 \quad \ldots \quad s_{i-1}}_{\mathsf{L}}\, s_i\, \underbrace{\ldots}_{\mathsf{M}}\, s_j\, \underbrace{s_{j+1} \quad \ldots \quad s_n}_{R}$$

s_i sa porovnáva s s_j v takých výpočtoch, keď jeden z x_i, x_j je ako pivot zvolený skôr, ako ktorýkoľvek z prvkov v M



$$p_{i,j} = \frac{|\{s_i, s_j\}|}{|\mathsf{M} \cup \{s_i, s_j\}|} = \frac{2}{j-i-1+2} = \frac{2}{j-i+1}$$

$$\begin{aligned} E[T] = \sum_{i=1}^{n} \sum_{j>i} p_{i,j} &= \sum_{i=1}^{n} \sum_{j>i} \tfrac{2}{j-i+1} \\ &\leq \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \tfrac{2}{k} \\ &\leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \tfrac{1}{k} \\ &= 2 \sum_{i=1}^{n} H(n) = \mathbf{O(n \ln n)} \end{aligned}$$

$$H(n) = \underbrace{\frac{1}{1}}_{\leq 1} + \underbrace{\frac{1}{2} + \frac{1}{3}}_{<1} + \underbrace{\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}}_{<1} + \cdots + \tfrac{1}{n}$$

$\ln n$ skupín so súčtom ≤ 1

6.2.3 Las Vegas

V súvislosti s Las Vegas algoritmami sa možno stretnúť s dvomi definíciami-jedna pripúšťa nekonečné výpočty, druhá zas odpoveď "neviem". Označme $A(x)$ odpoveď algoritmu A a $F(x)$ korektnú odpoveď na vstupe w.

D1

Definícia 6.1 *Pravdepodobnostný algoritmus A je typu Las Vegas (LV) keď*

$$Pr[A(x) = F(x)] = 1$$

Každá odpoveď je korektná, kedy ju dostaneme, nevieme. Zaujímavá je preto očakávaná zložitosť ExpTime.

D2

Definícia 6.2 *Pravdepodobnostný algoritmus A je typu Las Vegas (LV) keď* $\forall x$

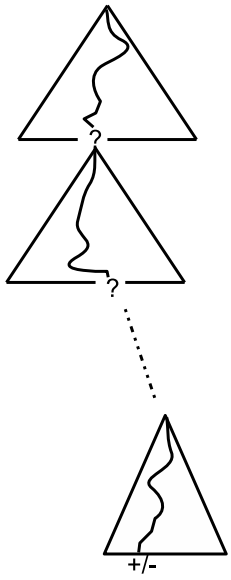
- $Pr[A(x) = F(x)] \geq 1/2$
- $Pr[A(x) = \text{"?"}] = 1 - Pr[A(x) = F(x)] < 1/2$

Konštanta $1/2$ nie je podstatná, vyhovuje ľubovoľné $O < \varepsilon < 1$. Zamyslime sa, aký je vzťah medzi týmito dvomi definíciami. Ukážeme, že sú ekvivalentné.

Lema 6.3 *Ku každému LV algoritmu $A_?$ v zmysle definície 6.2 existuje ekvivalentný algoritmus A, ktorý je LV v zmysle definície 6.1. Navyše, čas narastie len o multiplikatívnu konštantu: $ExpTime_A(n) = O(ExpTime_{A_?}(n))$*

Dôkaz: Samotná konštrukcia ekvivalentného algoritmu A je priamočiara—budeme opakovať výpočet algoritmu $A_?$ dovtedy, kým nedostaneme korektnú odpoveď.

pravdepodobnosť korektnej odpovede
Čo vieme povedať o pravdepodobnosti získania korektnej odpovede A, resp. dosiahnutia odpovede "?" algoritmu $A_?$? Nech p je pravdepodobnosť korektnej odpovede algoritmu $A_?$, $p \geq 1/2$. Potom pravdepodobnosť toho, že po k opakovaniach behu algoritmu $A_?$ ešte stále nemáme korektnú odpoveď, je nanajvýš $1/2^k$. Preto $Prob[A(x) = F(x)] = 1$



čas
Je zrejmé, že A môže realizovať nekonečné výpočty. Ukážeme, že v očakávanom prípade je situácia oveľa lepšia. Uspokojíme sa s horným odhadom na očakávaný prípad, preto budeme bez ujmy na všeobecnosti predpokladať, že každý výpočet algoritmu $A_?$, ktorý končí odpoveďou ?, dosahuje zložitosť najhoršieho prípadu.

Všimnime si výpočty, ktoré *skončia* v i-tom kole.

- Nech $Set_i = \{C \in S_{A,w} \mid (i-1)Time_{A_?} < Time_A(C) \leq iTime_{A_?}(w)\}$

  Zrejme $S_{A,w} = \cup S_{i=0}^{\infty} Set_i,\ \forall i \neq j\ Set_i \cap Set_j = \emptyset$

- Aby výpočet C mohol skončiť v i-tom kole, musí $(i-1)$ kôl skončiť odpoveďou "?". Pravdepodobnosť odpovede "?" je nanajvýš $1/2$, preto pravdepodobnosť, že výpočet neskončí po $i-1$ kolách je nanajvýš $\left(\frac{1}{2}\right)^{i-1}$. Z toho dostávame

$$\sum_{C \in Set_i} Prob[C] \leq \frac{1}{2^{i-1}}$$

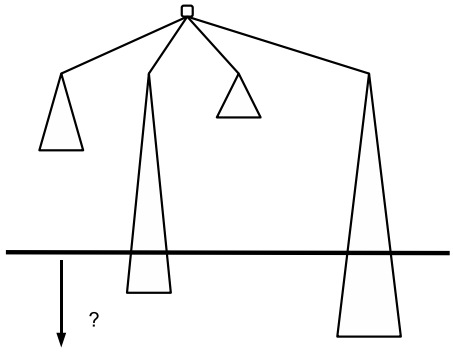
$$\begin{aligned}
ExpTime_A(w) &= \sum_{C \in S_{A,w}} Time_A(C) \cdot Prob[C] \\
&= \sum_{i=1}^{\infty} \sum_{C \in Set_i} Time_A(C) \cdot Prob[C] \\
&\leq \sum_{i=1}^{\infty} \sum_{C \in Set_i} iTime_{A_?(w)} \cdot Prob[C] \\
&= \sum_{i=1}^{\infty} iTime_{A_?}(w) \cdot \sum_{C \in Set_i} Prob[C] \\
&\leq \sum_{i=1}^{\infty} iTime_{A_?}(w) \cdot \frac{1}{2^{i-1}} \\
&= Time_{A_?}(w) \cdot \sum_{i=1}^{\infty} \frac{i}{2^{i-1}} \\
&= Time_{A_?}(w) \cdot 2 \cdot \sum_{i=1}^{\infty} \frac{i}{2^i} = 4 \cdot Time_{A_?}(w)
\end{aligned}$$

□

Lema 6.4 *Ku každému LV algoritmu A v zmysle definície 6.1 existuje ekvivalentný algoritmus $A_?$, ktorý je LV v zmysle definície 6.2.*

Dôkaz:

Ak do korektného algoritmu ideme vniesť nejednoznačnosť/odpoveď neviem, tak je prirodzené vyžadovať, aby ten nový algoritmus bol aspoň rýchly. Takto budeme konštruovať požadovaný algoritmus $A_?$. Zvolíme rozumnú hranicu a všetky výpočty, ktoré dovtedy neskončili, "ustrihneme" s odpoveďou "?".



Pri voľbe hranice pre odpoveď "?" treba mať na pamäti, že podľa definície sa pod touto hranicou môže ocitnúť nanajvýš polovica všetkých výpočtov. Keďže *najviac polovica čísel má hodnotu väčšiu ako dvojnásobok priemeru*, bude rozumným ohraničením

$$T = 2ExpTime_A(w)$$

Kvôli sporu predpokladajme, že $Prob[A_?(w) = \text{"?"}] > 1/2$. Označme

- $S_{A,w}(?)$ množinu výpočtov s odpoveďou "?"
  $S_{A,w}(?) = \{C \in S_{A,w} \mid Time(C) > 2ExpTime_A(w)\}$
- $S_{A,w}(F(w))$ množinu výpočtov s korektnou odpoveďou $F(w)$
  $S_{A,w}(F(w)) = S_{A,w} - S_{A,w}(?)$

Platí:

- $Prob[A_?(w) = \text{"?"}] = \sum_{C \in S_{A,w}(?)} Prob[C] > \frac{1}{2}$
- Všetky výpočty z $S_{A,w}(?)$ sú dlhé: $Time(C) \geq 2ExpTime_A(w) + 1$.

Preto

$$\begin{aligned}
\mathbf{ExpTime_A(w)} &= \sum_{C\in S_{A,w}} \overbrace{Time_A(C)Prob[C]}^{*} \\
&= \sum_{C\in S_{A,w}(?)} * + \sum_{C\in S_{A,w}(F(w))} * \\
&\geq \sum_{C\in S_{A,w}(?)} (2ExpTime_A(w)+1)\cdot Prob[C] + 0 \\
&= (2ExpTime_A(w)+1) \underbrace{\sum_{C\in S_{A,w}(?)} Prob[C]}_{>1/2} \\
&> \frac{(2ExpTime_A(w)+1)}{2} = \mathbf{ExpTime_A(w) + 1/2}
\end{aligned}$$

□

Trieda LV algoritmov, pre ktoré $Prob[A(x) = F(x)] = 1$, sa niekedy označuje Las Vegas*, resp. LV^*; hviezdička zrejme naznačuje možnosť nekonečných výpočtov. *trieda LV^**

6.2.4 Monte Carlo s jednosmernou chybou

O tejto triede algoritmov[3] hovoríme v súvislosti s rozhodovacími problémami[4]. Neformálne ide o také algoritmy, ktoré majú povolené sa mýliť/klamať len jedným smerom.

Definícia 6.3 *Pravdepodobnostný algoritmus A pre rozhodovací problém* (Σ, L) *je* Monte Carlo s jednosmernou chybou/jednosmerný Monte Carlo*(1MC, one-sided MC), ak* *1MC*

(a) $\forall x \in L \quad Prob[A(x) = 1] \geq 1/2$
(b) $\forall x \notin L \quad Prob[A(x) = 0] = 1$

Keď podmienku (a) *nahradíme podmienkou* *1MC**

(a*) $\forall x \in L \quad Prob[A(x) = 1] \to 1$ *s rastúcou dĺžkou* $|x|$

dostaneme tzv. 1MC algoritmy*

Príkladom 1MC pravdepodobnostného algoritmu je algoritmus 29 na porovnanie databáz, resp. algoritmus 32 na pravdepodobnostné porovnanie matíc $?AB = C?$.

Ako sme videli, pravdepodobnosť chyby klesá exponenciálne s počtom *nezávislých* opakovaní. Ak je teda 1MC algoritmus efektívny, ľahko z neho získame algoritmus rádovo rovnakej zložitosti s exponenciálne menšou chybou. *znižovanie chyby*

- nech $\alpha_1, \ldots, \alpha_k \in \{0,1\}^k$ je k výsledkov nezávislých behov 1MC algoritmu A
- ak $\exists k : \alpha_k = 1$, môžme so 100% istotou akceptovať
- ak $\alpha_1 = \ldots = \alpha_k = 0$, tak pre $x \in L$ je $Prob[A(x) = 0] \leq 1/2$ a teda $(Prob[A(x) = 0])^k \leq 2^{-k}$.

[3] One-sided Monte Carlo
[4] Rozhodovací problém je dvojica (Σ, L), pričom sa pre $x \in \Sigma^*$ pýtame, či $?x \in L?$

6.2.5 Monte Carlo s ohraničenou chybou

V prípade takého problému, ktorý nie je rozhodovací, ale je to problém počítania funkcie $F(x)$, pod korektnou odpoveďou prirodzene rozumieme, že algoritmus vypočítal presne hodnotu $F(x)$; nekorektnou odpoveďou je nepresný výsledok. Pri Monte Carlo algoritoch s ohraničenou chybou[5] pripustíme, aby algoritmus dával nesprávnu odpoveď, budeme ale vyžadovať, aby bol "dostatočný" rozdiel medzi odpoveďou korektnou a nekorektnou. Formálne:

2MC

Definícia 6.4 *Pravdepodobnostný algoritmus A je Monte Carlo s ohraničenou chybou(2MC, bounded-error MC), ak*

$$\exists 0 < \varepsilon \leq 1/2 \quad \forall x \quad Prob[A(x) = F(x)] \geq 1/2 + \varepsilon$$

znižovanie chyby

V prípade 1MC algoritmov sme pravdepodobnosť chyby jednoducho znížili nezávislým opakovaním algoritmu. Ako nám pomôže nezávislé opakované spúšťanie 2MC algoritmu?

Uvažujme t nezávislých opakovaní 2MC algoritmu A a odpovedzme len vtedy, ak sa na odpovedi zhodne aspoň $t/2$ behov (algoritmus 36). Zaujíma nás, aká je pravdepodobnosť toho, že *nedostaneme* korektnú odpoveď.

Algoritmus 36 A_t

1: nech $\alpha_1, \ldots, \alpha_t$ je výsledok t nezávislých behov 2MC algoritmu A
2: ak existuje taký výsledok α, ktorý sa vyskytol aspoň $t/2$ krát, výsledkom je "α", inak je výsledkom "?"

Keďže A je 2MC, existuje $0 < \varepsilon \leq 1/2$ $Prob[A(x) = F(x)] \geq 1/2 + \varepsilon$. Nech x je vstup. Označme

- $p = p(x) = Prob[A(x) = F(x)] = 1/2 + \varepsilon_x$, $\varepsilon_x \geq \varepsilon$
- $pr_i(x)$ pravdepodobnosť toho, že spomedzi k opakovaní $\mathsf{A}(x)$ je presne i odpovedí korektných; je presne $\binom{k}{i}$ možností takýchto výpočtov

Chybu algoritmu 36 získame odhadom $pr_i(x)$.

$$\begin{aligned}
pr_i(x) &= \binom{k}{i} p^i (1-p)^{k-i} = \binom{k}{i} [p(1-p)]^i (1-p)^{k-2i} \\
&= \binom{k}{i} \left[\left(\tfrac{1}{2} + \varepsilon_x\right)\left(\tfrac{1}{2} - \varepsilon_x\right)\right]^i \left(\tfrac{1}{2} - \varepsilon_x\right)^{k-2i} \\
&= \binom{k}{i} \left(\tfrac{1}{4} - \varepsilon_x^2\right)^i \left[\left(\tfrac{1}{2} - \varepsilon_x\right)^2\right]^{k/2-i} \\
&< \binom{k}{i} \left(\tfrac{1}{4} - \varepsilon_x^2\right)^i \left[\left(\tfrac{1}{2} + \varepsilon_x\right)\left(\tfrac{1}{2} - \varepsilon_x\right)\right]^{k/2-i} \\
&= \binom{k}{i} \left(\tfrac{1}{4} - \varepsilon_x^2\right)^i \left(\tfrac{1}{4} - \varepsilon_x^2\right)^{k/2-i} = \binom{k}{i} \left(\tfrac{1}{4} - \varepsilon_x^2\right)^{k/2} \\
&\leq \binom{k}{i} \left(\tfrac{1}{4} - \varepsilon^2\right)^{k/2}
\end{aligned}$$

A_k odpovie korektne, ak aspoň $\lceil k/2 \rceil$ jednotlivých behov odpovie korektne.

[5] Bounded-Error Monte Carlo

$$\begin{aligned}
Prob[A_k(x) = F(x)] &= \sum_{i=\lceil k/2 \rceil}^{k} pr_i(x) = 1 - \sum_{i=0}^{\lfloor k/2 \rfloor} pr_i(x) \\
&> 1 - \sum_{i=0}^{\lfloor k/2 \rfloor} \binom{k}{i} \left(\frac{1}{4} - \varepsilon_x^2\right)^{k/2} \\
&= 1 - \left(\frac{1}{4} - \varepsilon_x^2\right)^{k/2} \sum_{i=0}^{\lfloor k/2 \rfloor} \binom{k}{i} \\
&> 1 - \left(\frac{1}{4} - \varepsilon_x^2\right)^{k/2} \cdot 2^k \\
&\geq 1 - (1 - 4\varepsilon_x^2)^{k/2} \geq 1 - (1 - 4\varepsilon^2)^{k/2} \qquad [**]
\end{aligned}$$

Ak chceme, aby $Prob[A_k(x) = F(x)] \geq 1 - \delta$, stačí zvoliť

$$k = \frac{2 \ln \delta}{\ln(1 - 4\varepsilon^2)}$$

Uvedomme si, že keďže ε aj δ sú konštanty, je konštantou aj k. Preto za zníženie pravdepodobnosti chyby na δ "platíme" len konštantným nárastom zložitosti

$$T_{A_k}(n) = O(kT_A(n)) = O(T_A(n))$$

□

6.2.6 Monte Carlo s neohraničenou chybou

Konštanta ε v 2MC algoritmoch vytvárala akúsi "bariéru", ktorá umožňovala efektívne odlíšiť, s veľkou pravdepodobnosťou, korektné odpovede od nekorektných. Ak upustíme od tejto požiadavky, dostaneme triedu MC, pri ktorej to so znižovaním pravdepodobnosti chyby nebude také optimistické.

Definícia 6.5 *Pravdepodobnostný algoritmus A je typu Monte Carlo s neohraničenou chybou (MC, unbounded-error MC), ak* MC

$$Prob[A(x) = F(x)] > 1/2$$

Príklad 6.3 *Predstavme si, že máme taký pravdepodobnostný algoritmus A, ktorý má pre vstupné slovo x až $2^{|x|}$ možných behov, pričom korektných je $2^{|x|-1}+1$, teda "tesná väčšina". Takýto algoritmus je zrejme MC*

$$Prob[A(x) = F(x)] = \frac{2^{|x|-1} + 1}{2^{|x|}} = \frac{1}{2} + \frac{1}{2^{|x|}} > \frac{1}{2}$$

Pritom hodnota $\frac{1}{2^{|x|}}$ tu hrá úlohu ε_x z 2MC algoritmov.

Zaujíma nás, koľko opakovaní tohto algoritmu potrebujeme, aby sme pri hlasovaní nadpolovičnou väčšinou dosiahli pravdepodobnosť chyby nanajvýš δ. znižovanie chyby
Využijeme, čo už vieme–ak chceme $Prob[A_k(x) = F(x)] \geq 1 - (1 - 4\varepsilon^2)^{k/2} > 1 - \delta$, stačí zvoliť

$$\begin{aligned}
k = k(x) &\geq \frac{2 \ln \delta}{\ln(1 - 4\varepsilon_x^2)} = \frac{2 \ln \delta}{\ln\left(1 - 4 \cdot 2^{-2|x|}\right)} \\
&\geq \frac{2 \ln \delta}{-2^{-2|x|}} \qquad //0 < y < 1 \quad \Rightarrow \quad \ln(1 - y) \leq -y \\
&= (-2 \ln \delta) 2^{2|x|}
\end{aligned}$$

To ale znamená, že $\boxed{Time_{A_k}(x) = (-2 \ln \delta) 2^{2|x|} \cdot Time_A(x)}$

Uvedený príklad je z akéhosi pohľadu "worst-case". Ak by sme mali $\varepsilon_x = \frac{1}{\log|x|}$, nebolo by to s časom $Time_{A_k}(x)$ také zlé.

Na záver tejto časti ešte jeden konkrétny príklad MC algoritmu.

Príklad 6.4 *Uvažujme opäť porovnávanie databáz. Tentokrát budeme vstup akceptovať vtedy, ak sú databázy rôzne. Náhodný výber využijeme na "uhádnutie" miesta, kde sa databázy líšia. Výsledkom je Algoritmus UMC.*

Algoritmus 37 UMC

vstup: RI: $X = x_1 \dots x_n$
RII:$Y = y_1 \dots y_n$

výstup: 1 ak $X \neq Y$

RI
1: náhodne zvoľ $j \in \{1, \dots, n\}$
2: pošli j, x_j do RII

RII
1: **if** $x_j \neq y_j$ **then** "accept" ▷ 100% istota
2: **else**
3: "accept" s pravdepodobnosťou $1/2 - \frac{1}{2n}$
4: "reject" s pravdepodobnosťou $1/2 + \frac{1}{2n}$

počet bitov Celá komunikácia spočíva v prenesení j, x_j, preto prenášame $\lceil \log(n+1) \rceil + 1$ bitov.

UMC je MC Analýzu chyby rozdelíme na dve časti podľa korektnej odpovede. Najprv označenia.

$C_{j,l}$ Označme $C_{j,l}$ takú udalosť, že RI zvolilo (v prvej fáze) index j a RII v druhej fáze povedalo l (1 je accept, 0 reject).

$$Prob[C_{j,0}] = \frac{1}{n}\left(\frac{1}{2} + \frac{1}{2n}\right)$$

$$Prob[C_{j,1}] = \frac{1}{n}\left(\frac{1}{2} - \frac{1}{2n}\right)$$

A_l Označme A_l množinu tých behov, ktoré dajú odpoveď l

$X = Y$ Ak $X = Y$, tak neexistuje možnosť, aby $x_j \neq y_j$. V tejto situácii RII rozhoduje pravdepodobnostne. Zaujíma nás pravdepodobnosť korektnej odpovede:

$$\mathbf{Prob[A_0]} = \sum_{i-1}^{n} Prob[C_{i,0}] = \sum_{i=1}^{n} \frac{1}{n}(\frac{1}{2} + \frac{1}{2n}) = \frac{1}{2} + \frac{1}{2n} > \mathbf{1/2}$$

$X \neq Y$ Ak $X \neq Y$, potom existuje taký index j, že $x_j \neq y_j$. Budeme predpokladať, že je jediný(čo nezvýši pravdepodobnosť korektnej odpovede).
Zaujíma nás pravdepodobnosť $A_1 = C_j \cup \{C_{i,1} \mid 1 \leq i \leq n,\ i \neq j\}$, pričom C_j označuje udalosť, keď RI zvolilo jediný index, na ktorom sa X, Y líšia.

$$\begin{aligned}
\mathbf{Prob[A_1]} &= Prob[C_j] + \sum_{i\neq j} Prob[C_{i,1}] \\
&= \frac{1}{n} + \sum_{i\neq j} \frac{1}{n}(\frac{1}{2} - \frac{1}{2n}) \\
&= \frac{1}{2n} + \frac{1}{2n} + \sum_{i\neq j}(\frac{1}{2n} - \frac{1}{2n^2}) \\
&= \frac{1}{2n} + \sum_{i=1}^{n} \frac{1}{2n} - \sum_{i\neq j} \frac{1}{2n^2} \\
&= \frac{1}{2n} + \frac{1}{2} - \frac{n-1}{2n^2} = \frac{1}{2} + \frac{1}{2n^2} > \mathbf{\frac{1}{2}}
\end{aligned}$$

□

6.2.7 Pravdepodobnostné algoritmy pre optimalizačné úlohy

V prípade optimalizačných úloh zrejme nezávislé opakovanie behu algoritmu použijeme na získanie riešenia lepšej kvality. Zopakujme si definíciu optimalizačného problému, ktorá sa formálne mierne odlišuje od definície 4.1.

Definícia 6.6 *Optimalizačný problém je* $U = (\Sigma_I, \Sigma_O, L, L_I, Sol, cena, ciel)$, *kde* *optimalizačný problém*

Σ_I	*je vstupná abeceda*
Σ_O	*je výstupná abeceda*
$L \subseteq \Sigma_I^*$	*je jazyk prípustných vstupov*
$L_I \subseteq L$	*je jazyk aktuálnych vstupov*
Sol	$Sol : L \to Pot(\Sigma_O^*) \forall x \in L$ *je* $Sol(x)$ *množina prípustných riešení*
$cena$	*je účelová funcia;* $cena(x, Sol(x)) \in \mathbb{R}$
$ciel$	$ciel \in \{min, max\}$

Prípustné riešenie $y \in Sol(x)$ *nazveme* optimálnym riešením *pre* x, *ak pre účelovú funkciu platí* $cena(x, y) = ciel\{(x, z), z \in Sol(x)\}$. *Ak* y *je opimálne riešenie pre* x *a* U, *tak označíme* *optimálne riešenie*

$$cena(x, y) = OPT_U(x)$$

Hovoríme, že algoritmus A je *konzistentný* pre U, ak $\forall x \in L_I A(x) \in Sol(x)$. Hovoríme, že algoritmus B *rieši* optimalizačný problém U, ak *konzistentný algoritmus*

- B je konzistentný pre U
- $\forall x \in L_I\ B(x) = OPT_U(x)$

Poznámka Uvedená definícia zachytáva všetky aspekty súvisiace s definíciou optimalizačnej úlohy: vstup je reťazec nad abecedou Σ_I, ale korektný vstup je z jazyka L_I. Analogicky pre výstup. Často budeme používať zjednodušenú formuláciu, keď problém identifikujeme množinou vstupov I, zobrazením Sol, ktoré vstupu priraďuje množinu prípustných riešení, ohodnocovacou funkciou (m, $cena$,..) a cieľom (max, min).

Kvalita riešenia sa posudzuje prostredníctvom chyby. Zaujíma nás aproximačný pomer $Ratio_A(x) = \max\{\frac{OPT(x)}{A(x)}, \frac{A(x)}{OPT(x)}\}$. Možné sú dva prístupy – zaujímame sa o očakávanú hodnotu $Ratio_A(x)$, alebo nás zaujíma pravdepodobnosť toho, že bude tento pomer v rozumných hraniciach.

Definícia 6.7 *Algoritmus* A *je* pravdepodobnostný $E[\delta]$-aproximačný, *ak* $E[\delta]$-*aproximácia*

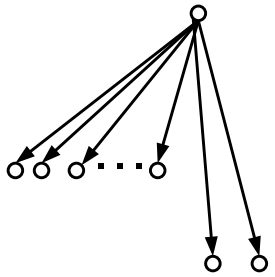
- $Prob[A(x) \in Sol(x)] = 1$

- $E[Ratio_A(x)] \leq \delta \ \forall x \in L$

Definícia 6.8 *Algoritmus A je* pravdepodobnostný δ-aproximačný, *ak* *δ-aproximácia*

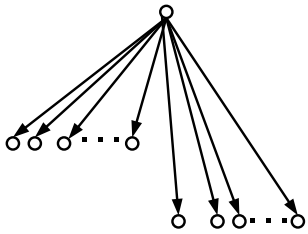
- $Prob[A(x) \in Sol(x)] = 1$
- $Prob[Ratio_A(x) \leq \delta] \geq 1/2 \ \forall x \in L$

Definície sú podobné, nie však totožné.



$$Ratio_{A,x}(C_i) = 2 \quad \text{pre} \quad i = 1, \ldots, 10$$
$$Ratio_{A,x}(C_i) = 50 \quad \text{pre} \quad i = 11, 12$$
$$E[Ratio_A(x)] = 10$$
$$Prob[Ratio_A(x) \leq 2] = 5/6 \geq 1/2$$

Veľa "dobrých", málo "zlých" spôsobí, že hoci je pravdepodobnosť dobrej odpovede slušná, stredná hodnota je zlá.



$$Ratio_A(x) = 11 \quad \text{pre} \quad 1000 \text{ behov}$$
$$Ratio_A(x) = 1 \quad \text{pre} \quad 999 \text{ behov}$$
$$E[Ratio_A(x)] = \frac{11999}{1999} = 6.0025$$
$$Prob[Ratio_A(x) \leq 10] = \frac{999}{1999} = 0.499.. < 1/2$$

Keď je "dobrých" aj "zlých" skoro rovnako, pravdepodobnosť dobrej odpovede môže byť $< 1/2$, ale stredná hodnota je aj tak dobrá.

Nasledujúci jednoduchý fakt sa dá rozumne využiť.

Fakt 6.5 *Pravdepodobnosť udalosti, že náhodná premenná x má hodnotu $\leq 2E[x]$, je aspoň $1/2$.*

Lema 6.6 *Nech $\delta > 0$, U optimalizačný problém. Ak algoritmus B je pravdepodobnostný $E[\delta]-$aproximačný, potom B je $2\delta-$aproximačný.*

Príklad 6.5 *Uvažujme problém Max-kSAT, kde vstupom je formula F v konjunktívnej normálnej forme(KNF), pričom každá klauzula má* presne *k literálov. Chceme vektor, ktorý spĺňa čo najviac klauzúl. Vráťme sa k Algoritmu 33, ktorý vráti náhodne zvolený vektor x. Pre $Z_i(\alpha) = 1 \Leftrightarrow F_i(\alpha) = 1$, $Z_F = \sum_{i=1}^{m} Z_i$ sme ukázali, že $E[Z_F] = \sum E[Z_i] = m(1 - \frac{1}{2^k})$. Keďže $OPT \leq m$,*

$$E[Ratio(F)] = \frac{OPT(F)}{E[Z_F]} \leq \frac{m}{m(1 - \frac{1}{2^k})} = \frac{2^k}{2^k - 1}$$

Ukážeme, že Algoritmus 33 je tiež $\frac{2^{k-1}}{2^{k-1}-1}$-aproximačný.

Keďže
priemerný počet splnených klauzúl $E[Z_F] = m(1 - \frac{1}{2^k})$
$m(1 - \frac{1}{2^k})$ je priemer z $m, m(1 - \frac{1}{2^{k-1}})$
aspoň polovica behov musí spĺňať aspoň $m(1 - \frac{1}{2^{k-1}})$ klauzúl. ◇

MaxCut

Príklad 6.6

vstup: *neohodnotený graf* $G = (V, E)$
výstup: *rozklad* (V_1, V_2) *množiny vrcholov* V
cena: $cost((V_1, V_2)) = |E \cap (V_1 \times V_2)|$
cieľ: *max*

Ukážeme, že náhodná voľba je E[2]-aproximačný algoritmus

Algoritmus 38 RC

$V_1 = V_2 = \emptyset$
$\forall v \in V$ náhodne zvoľ i a $V_i \leftarrow V_i \cup v$
return (V_1, V_2)

Indikačná premenná bude počítať počet hrán rezu. Nech $e = (u, v)$

$$X_e = \begin{cases} 0, & \textit{vrcholy } u, v \textit{ nepatria do rezu, ale sú spoločne v jednej z množín } V_1, V_2 \\ 1, & \textit{hrana } (u, v) \textit{ patrí do rezu} \end{cases}$$

$E[X_e] = Prob[x_e = 1]$
$Prob[x_e = 1] = Prob[u \in V_1 \& v \in V_2] + Prob[u \in V_2 \& v \in V_1] = \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$
Takže
$E[x_e] = 1/2$
$X = \sum_{e \in E} x_e$, $E[X] = |E|/2$
$E[Ratio] = \frac{OPT}{E[X]} \leq \frac{|E|}{E[X]} = 2$

◇

Kapitola 7

Metódy tvorby pravdepodobnostných algoritmov

Analýzou existujúcich pravdepodobnostných algoritmov možno "vytiahnuť" niekoľko metód, ktoré sa pri ich návrhu využívajú. Často ide o kombináciu, niekedy možno algoritmus vnímať optikou viacerých metód. Začneme krátkym súhrnom metód, ktoré potom podrobnejšie rozvinieme a ilustrujeme na príkladoch v nasledujúcich podkapitolách.

Eliminácia protihráča

Eliminácia protihráča je v podstate metódou vyhýbania sa najhorším prípadom. Keď máme deterministický algoritmus, vstup pre najhorší prípad pre tento konkrétny algoritmus sa väčšinou konštruuje pomerne jednoducho. Pravdepodobnostný algoritmus je svojím spôsobom množinou deterministých algoritmov (pridaním konkrétnej postupnosti náhodných bitov k vstupu sa z pravdepodobnostného algoritmu stane deterministický). Aby bol konkrétny vstup zlým pre takýto pravdepodobnostný algoritmus, mal by byť zlým pre dostatočne veľa postupnoťou náhodných bitov určených deterministických algoritmov. Dá sa očakávať, že toto nie je jednoduché a že náhodný výber "deterministického" algoritmu v *očakávanom* prípade eliminuje zložitosť nahoršieho prípadu; uvedomme si, že zložitosti najhoršieho prípadu sme sa tým nevyhli.

S príkladom použitia metódy eliminácie protihráča sme sa už stretli.

R_I, R_{II} $x \stackrel{?}{=} y$

V prípade zisťovania rovnosti dvoch vzdialených databáz (reprezentovaných binárnymi reťazcami) bola množina deterministických algoritmov určená množinou prvočísel: porovnanie $x \stackrel{?}{=} y$ sme nahradili porovnaním $x \mod p \stackrel{?}{=} y \mod p$, kde $p \in PRIM(n^2)$. Vieme, že aspoň $\frac{Prim(n^2)-(n-1)}{Prim(n^2)} = 1 - \frac{2\ln n}{n}$-tina odpovedí je korektná.
$\rightsquigarrow$ náhodná voľba prvočísla teda vedie k veľkej pravdepodobnosti korektnej odpovede; algoritmus je korektný na takmer každom vstupe

QUICKSORT

Algoritmus QUICKSORT je príkladom, keď sa viaceré deterministické stratégie líšia voľbou pivota, pričom
- každá dáva korektnú odpoveď
- každá je efektívna na väčšine vstupov
  $T(1) = 0, \quad T(n) = n - 1 + T(A_<) + T(A_>)$

Videli sme, že náhodná voľba pivota viedla v očakávanom prípade k zložitosti rádovo rovnej dolnému odhadu, ktorý je $\Omega(n \log n)$

Metóda svedkov

Metóda svedkov je vhodná najmä pre také rozhodovacie problémy, keď nás zaujíma, či daný vstup má alebo nemá nejakú vlastnosť V, napr. či je dané číslo n prvočíslo. Testovanie prvočíselnosti je ťažký problém, ak však má číslo n deliteľa, prvočíslom byť nemôže. Keď nám niekto povie, že 3 je *kandidát* na delenie čísla n, ľahko overíme, či je to pravda. Ak áno, 3 sa stalo *svedkom* toho, že n nie je prvočíslom; ak nie, riešeniu nášho problému to nepomohlo. Uvedomme si, že pomocou takéhoto svedka–3 ako deliteľa n–nemôžme potvrdiť, že n je prvočíslo, môžme to len vyvrátiť.

Meóda svedkov teda spočíva v náhodnom generovaní kandidátov a overovaní, či sú alebo nie svedkami danej vlastnosti. K jej použitiu je potrebné, aby

- existovala vhodná definíciu toho, čo je to svedok. Je to nejaká dodatočná informácia W, ktorá pomáha určiť, či skúmaná vlastnosť platí/neplatí

  $V \rightsquigarrow$ prvočíselnosť $W \rightsquigarrow$ delitele
- existovala dostatočne veľká efektívne konštruovateľná množina kandidátov, o ktorých vieme efektívne zistiť, či sú svedkami
- v prípade, že existuje svedok, množina kandidátov obsahovala dostatočne veľa svedkov

Pri prvočíselnosti sú kandidátmi $p \in PRIM(n^2)$; je ich aspoň $Prim(n^2)$ - $(n-1)$. Pravdepodobnosť, že náhodne zvolený kandidát je svedok je

$$\frac{\frac{n^2}{\ln n^2} - (n-1)}{\frac{n^2}{\ln n^2}} \geq 1 - \frac{\ln n^2}{n}$$

Nemáme algoritmus, ktorý efektívne hľadá svedkov. Zoberme prehľadávanie od najmenšieho prvočísla. Keďže "zlých" prvočísel je nanajvýš $n-1$, po n pokusoch určite skončíme. Zložitosť takéhoto prístupu je však $4n \log n$, čo je veľa. Prečo nevieme garantovať nič lepšie? Lebo kvôli istote potrebujeme $k+1$ pokusov pre vstup (x, y) taký, že $Num(x) - Num(y) = p_1 \cdot p_2 \cdot \ldots \cdot p_k$

Freivaldsova metóda

Metóda odtlačkov môže byť považovaná za špeciálny prípad metódy svedkov. Používa sa vtedy, keď nás zaujíma rovnosť/ekvivalencia nejakých komplexných objektov O_1, O_2, pričom porovnanie týchto objektov je náročné. Postupujeme nasledovne:

1. vezmeme množinu M vhodných zobrazení úplných reprezentácií do čiastočných reprezentácií (napr. $N \rightarrow PRIM(n^2)$)
   $h \leftarrow random(M)$

2. vypočítame redukované reprezentácie $h(O_1)$, $h(O_2)$; tomu sa hovorí odtlačok/stopa/fingerprint

3. porovnáme redukované reprezentácie
   if $h(O_1) = h(O_2)$ **then** return("ekvivalentné")
   else return ("nie sú ekvivalentné")

Je zrejmé, že môžu existovať dva rôzne objekty s rovnakou redukovanou reprezentáciou a teda odpoveď *ekvivalentné* nemusí byť pravdivá.
K úspešnému použitiu metódy potrebujeme

- $h(O)$ efektívne počítateľné

- porovnanie $h(O_1) \stackrel{?}{=} h(O_2)$ efektívne vypočitateľné
- existenciu kandidátov/svedkov v M, ktoré to v prípade $O_1 \neq O_2$, potvrdzujú; vedú teda k $h(O_1) \neq h(O_2)$

Neprekvapí, že máme tradeoff medzi komplexnosťou $h(O)$ a pravdepodobnosťou korektnej odpovede.

náhodná vzorka

Náhodná vzorka/Random sampling využívame v situácii, keď nevieme hľadať objekty daných vlastností ale vieme, že ich je veľa. Potom náhodne vybraná množina kandidátov by s veľkou pravdepodobnosťou mala hľadaný objekt obsahovať.

Táto pravdepodobnostná metóda je založená na dvoch jednoduchých faktoch:

Fakt 7.1 *Ku každej náhodnej premennej X existuje hodnota, ktorá nie je menšia (väčšia) ako $E[X]$*

Fakt 7.2 *Ak pre náhodný objekt je pravdepodobnosť, že má nejakú vlastnosť, nenulová, potom existuje objekt, ktorý túto vlastnosť má.*

opakovanie

Zvyšovanie úspešnosti opakovaním je realizované znižovaním pravdepodobnosti chyby nezávislým opakovaním behov, resp. len ich častí, na tom istom vstupe. Už sme niekoľko teoretických výsledkov o tomto prístupe videli v prechádzajúcich častiach.

zaokrúhľovanie

Optimizácia a náhodné zaokrúhľovanie Niekedy je optimalizačný problém ťažký na množine diskrétnych vstupov, ale jeho riešenie na $\mathbb{R}$ je efektívne (napr. lineárne programovanie). V tejto situácii problém často riešime tak, že relaxujeme od diskrétnych hodnôt, vyriešime problém v reálnych číslach a získané riešenie vhodne zaokrúhlime.

Kapitolou samou o sebe je derandomizácia—prehľadný a efektívny pravdepodobnostný algoritmus "derandomizujeme" na determnistický; väčšinou simulovaním všetkých možných behov pravdepodobnostného algoritmu.

7.1 Eliminácia protivníka

V tejto časti sa budeme venovať metóde eliminácie protihráča. Sústredíme sa na konštrukciu triedy deterministických algoritmov pre ktoré platí, že *pre každý vstup je väčšina z nich efektívna.* Aplikujeme na dva príklady.

hašovanie

Pre problém hašovanie ukážeme, že

- ku každej hašovacej funkcii existuje zlý vstup
- existuje trieda hašovacích funkcií H taká, že pre každý vstup S a náhodnú hašovaciu funkciu $h \in H$, je $h(S)$ "dobre" rozložené

on line plánovanie

Pre problém plánovania ukážeme, že pravdepodobnostný algoritmus dáva kvalitatívne lepšie riešenie ako ľubovoľný deterministický algoritmus.

7.1.1 Hašovanie

Uvažujme problém manažmentu dát, keď v štruktúre T vyhľadávame prvky podľa kľúča k. Operácie, ktoré potrebujeme efektívne realizovať, sú—Search(T,k), Insert(T,k), Delete(T,k). Ide o implemntáciu tzv. slovníka, keď na implementáciu množiny objektov použijeme tabuľku spolu s vhodným hašovaním.

Predpokladáme, že

T je *tabuľka* s priamym prístupom, $T = \{0, 1, \ldots, m-1\}$
$U = \{0, 1, \ldots, d\}$; $|U| \ggg |T|$ je *univerzum*

Zaujíma nás taká *hašovacia funkcia*, resp. zobrazenie $h : U \to T$, ktoré každú množinu $S \subseteq U, |S| = n$ "dobre rozloží v T": v priemere $\frac{|S|}{|T|} = \frac{n}{m}$ prvkov zahašovaných do jednej bunky/hodnoty. Táto požiadavka je prirodzená, keďže počet prvkov zahašovaných do jednej bunky má vplyv na zložitosť realizácie spomínaných slovníkových inštrukcií. Ak b-ta bunka osahuje zoznam ℓ prvkov zahašovaných na hodnotu b, tak zložitosť realizácie slovníkových inštrukcií je v najhoršom prípade ℓ, v očakávanom prípade $\ell/2$.

Od hašovacej funkcie h preto vyžadujeme nasledujúce vlastnosti

– dá sa efektívne vypočítať
– pre *väčšinu* $S \subseteq U$ prvky "rovnomerne rozhadzuje"; inými slovami

$$T(i) = \{a \in S \mid h(a) = i\} \quad \Rightarrow \quad |T_i| = O\left(\frac{|S|}{|T|}\right)$$

Uvedomme si, že nič lepšie žiadať nemôžeme, nakoľko pre $S = U_{h,i} = \{a \in U \mid h(a) = i\}$, sa celá množina S zobrazí do jednej bunky...

Lema 7.3 *Nech* $U = \mathbb{N}$, $T = \{0, 1, \ldots, m-1\}$, $n \in \mathbb{N}^+$, $h : U \to T$ *spĺňa:* $Pr[h(x) = i] = \frac{1}{m}$. *Potom*

a. očakávaný počet prvkov v jednej bunke je $\frac{n}{m} + 1$
b. ak $n = m$, *potom* $Pr[$*viac ako jeden prvok* $h(x) = i]) < 1/2$

Dôkaz: Nech $S = \{s_1, \ldots, s_n\}$ je množina prvkov náhodne vybratých z U. Pri počítaní kolízií si pomôžeme náhodnými premennými:

$$X^l_{i,j}(S) = \begin{cases} 1, & h(s_i) = h(s_j) = l \\ 0, & \text{inak} \end{cases}$$

$$E[X^l_{i,j}] = Pr[h(s_i) = l] Pr[h(s_j) = l] = \frac{1}{m^2}$$

Ohraničíme strednú hodnotu počtu kolízií v l-tej bunke:

$$E[X^l] = \sum_{1 \le i < j \le n} E[X^l_{i,j}] = \sum_{1 \le i < j \le n} \frac{1}{m^2} = \frac{n(n-1)}{2} \frac{1}{m^2} < \frac{n^2}{2m^2}$$

$$n = m \quad \Rightarrow \quad E[X^l] < 1/2$$

Ak sa do l-tej bunky zahašuje v priemere k prvkov, spôsobí to v priemere $\binom{k}{2}$ kolízií

$$\frac{n^2}{2m^2} > E[X^l] = \frac{k(k-1)}{2} > \frac{(k-1)^2}{2} \quad \Rightarrow \quad \frac{n}{m} + 1 > k$$

□

Ak je teda $n \sim m$, tak očakávaná zložitosť je $O(1)$. Reálne data ale nie sú veľmi rovnomerne rozložené...

univerzálna trieda hašovacích funkcií

Definícia 7.1 *Nech* H *je konečná množina hašovacích funkcií:* $U \to T = \{0, 1, , \ldots, m-1\}$. *Hovoríme, že* H *je* univerzálna *trieda funkcií, ak* $\forall x, y \in U, x \neq y$

$$|\{h \in H \mid h(x) = h(y)\}| \le \frac{|H|}{m}$$

Veta 7.4 *Nech $S \subseteq U$, $|S| = n$, H je univerzálna trieda hašovacích funkcií. Potom pre každé $x \in S$ a náhodnú hašovaciu funkciu $h \in H$ pre očakávanú veľkosť množiny $S_x(h) = \{a \in S \mid a \neq x,\ h(a) = h(x)\}$ platí*

$$E[|S_x(h)|] \leq \frac{|S|}{|T|} = \frac{n}{m}$$

Dôkaz: Uvažujme náhodnú premennú $Z_{x,y}(h)$, ktorá pre hašovaciu funciu h identifikuje kolíziu medzi x, y a premennú $Z_x(h)$, ktorá spočíta počet kolízií s prvkom x. Stredná hodnota $Z_x(h)$ určuje očakávaný počet kolízií.

$$Z_{x,y}(h) = \begin{cases} 1, & h(x) = h(y) \\ 0, & h(x) \neq h(y) \end{cases} \qquad E[Z_{x,y}(h)] = Pr[h(x) = h(y)] \leq 1/m$$

$$Z_x(h) = \sum_{y \in S, x \neq y} Z_{x,y}(h) \quad E[Z_x(h)] = \sum_{y \in S, x \neq y} E[Z_{x,y}(h)] \leq \frac{|S| - 1}{m} < \frac{n}{m}$$

□

Pojem univerzálnej triedy hašovacích funkcií je dobre definovaný univerzálna trieda hašovacích funkcií existuje. Ľahko sa dá ukázať, že stačí zobrať všetky funkcie $U \to T$.

Lema 7.5 *Trieda $H_{U,T} = \{h \mid h : U \to T\}$ je univerzálna trieda hašovacích funkcií.*

Dôkaz: Nech $|T| = m$

- $|H_{U,T}| = m^{|U|}$
- $|H(x, y, i)| = |\{h \mid h(x) = h(y) = i\}| = m^{|U|-2}$
- $|H(x, y)| = \{h \mid h(x) = h(y)\} = \sum_{i=0}^{m-1} |H(x, y, i)| = m^{|U|-1} = \frac{|H_{U,T}|}{m}$

□

Trieda $H_{U,T}$ však pre praktické účely nie je vhodná, pretože
– je príliš veľká
– väčšina funkcií nemá efektívnu reprezentáciu

Existujú ale aj také univerzálne triedy hašovacích funkcií, ktoré praktické sú. Zoberme ako univerzum $U = \{0, 1, \ldots, p-1\}$, hašovacia tabuľka T je veľkosti m, pričom p, m sú prvočísla. Ukážeme, že vhodnou univerzálnou triedou hašovacích funkcií je napr. nižšie definovaná trieda hašovacích funkcií H^p_{lin}: H^p_{lin}

$$h_{a,b}(x) = ((ax + b) \mod p) \mod m$$

$$H^p_{lin} = \{h_{a,b} \mid a \in \{1, 2, \ldots, p-1\};\ b \in \{0, 1, \ldots, p-1\}\}$$

Veta 7.6 *Pre každé prvočíslo p (a číslo m) je trieda H^p_{lin} univerzálnou triedou hašovacích funkcií z $U = \{0, 1, \ldots, p-1\}$ do $T = \{0, 1, \ldots, m-1\}$.*

Dôkaz: Označme

$$H^p_{lin}(x, y) = \{h_{a,b} \in H^p_{lin},\ h_{a,b}(x) = h_{a,b}(y)\}$$
$$M(x, y) = \{(r, s) \in U \times U \mid r \neq s \text{ a } r \equiv s \mod m\}$$

Zrejme $|H^p_{lin}(x, y)| = |M(x, y)|$. Potrebujeme ukázať, že $\forall x, y : |H^p_{lin}(x, y)| \leq \frac{|H^p_{lin}|}{m}$. Uvažujme najprv zjednodušenú verziu $h'_{a,b}(x) = (ax + b) \mod p$.

1. Ukážeme, že $f_{x,y}(a,b) = (h'_{a,b}(x), h'_{a,b}(y))$ je bijekcia; $a, b \overset{h'_{a,b}}{\leftrightsquigarrow} r, s$
2. Spočítame, koľko je k danému r takých s, že $r \neq s \mod p$ ale $r = s \mod m$

1. Poďme argumentovať, že $f_{x,y}$ je bijekcia. Nech $r = h'_{a,b}(x) = (ax+b) \mod p$, $s = h'_{a,b}(x) = (ay+b) \mod p$.

$x \neq y \Rightarrow r \neq s$

Kvôli sporu predpokladajme, že $x \neq y$ a $r = s$. Potom

$$0 = r - s \equiv a(x-y) \mod p$$

Keďže p je prvočíslo, znamenalo by to $a \equiv 0 \mod p$ alebo $x \equiv y \mod p$, čo vzhľadom k voľbe a, x, y nie je možné. Preto $x \neq y \Rightarrow r \neq s$. To ale znamená, že $f_{x,y}$ je zobrazenie z $\underbrace{\{U - \{0\}\}}_{ls} \rightarrow \underbrace{\{(r,s)|r,s \in U, r \neq s\}}_{rs}$. Keďže $|ls| = p(p-1) = |rs|$, stačí ukázať, že jedna z $f_{x,y}, f_{x,y}^{-1}$ je injektívna.

$r \neq s \Rightarrow a \neq b$

Nech $r \neq s$. Keďže p je prvočíslo, pre $x \neq y, 0 \leq x, y \leq p-1$ inverzný prvok k $(x-y)$ existuje :

$$\begin{array}{rcl} r - s = a(x-y) \mod p & \Rightarrow & a = (r-s)(x-y)^{-1} \mod p \\ & \Rightarrow & b = (r - ax) \mod p \end{array}$$

2. Pre x, y voľba náhodne zvolenej $h_{a,b}$ jednoznačne určuje dvojicu (r,s) takú, že $(r,s) = (h'_{a,b}(x), h'_{a,b}(y))$, preto $|H^p_{lin}(x,y)| = |M(x,y)|$. K výpočtu $|H^p_{lin}(x,y)|$ preto stačí spočítať, koľko je takých $r \neq s$, že $r \equiv s \mod m$.
K jednému r je $\lceil \frac{p}{m} \rceil \leq \frac{p+m-1}{m} = \frac{p-1}{m} + 1$ takých s, že $r \equiv s \mod m$. Preto

$$|H^p_{lin}(x,y)| = |M(x,y)| \leq \frac{p(p-1)}{m} = \frac{|H^p_{lin}|}{m}$$

□

Vec

Ďalším príkladom je trieda hašovacích funkcií, ktorú konštruujeme k univerzu $U(r,m)$, kde m je prvočíslo, $r \in \mathbb{N}$

$$U(m,r) = \{x = (x_0, \ldots, x_r);\ 0 \leq x_i \leq m-1, i = 0, \ldots, r\} = T^{r+1}$$

Nech $\alpha = (\alpha_0, \ldots, \alpha_r) \in T^{r+1}$; definujme

$$\mathsf{Vec} = \{h_\alpha; \alpha \in T^{r+1}\},\ \text{kde}\ h_\alpha(x_0, \ldots, x_r) = \left(\sum_{i=0}^{r} \alpha_i x_i \right) \mod m$$

Lema 7.7 *Vec je univerzálna množina hašovacích funkcií*

Dôkaz: Ukážeme, že $\forall x \neq y$ je $|H_\alpha(x,y)| = |\{h_\alpha \in \mathsf{Vec} \mid h_\alpha(x) = h_\alpha(y)\}| \leq \frac{|\mathsf{Vec}|}{m} = m^r$
Ak $x \neq y$, tak sa tieto vektory líšia aspoň v jednej zložke; pre jednoduchosť nech $x_0 \neq y_0$.

$$\begin{array}{rcl} h_\alpha(x) = h_\alpha(y) & \Leftrightarrow & \sum_{i=0}^{r} \alpha_i x_i \equiv \sum_{i=0}^{r} \alpha_i y_i \mod m \\ & & \alpha_0(x_0 - y_0) \equiv \sum_{i=0}^{r} \alpha_i (y_i - x_i) \mod m \end{array}$$

Keď m je prvočíslo, existuje pre $x_0 \neq y_0$ inverzný prvok $(x_0 - y_0)^{-1}$. Preto

$$\alpha_0 \equiv \sum_{i=1}^{r} \alpha_i (y_i - x_i)(x_0 - y_0)^{-1} \mod m$$

Hodnota α_0 je teda jednoznačne určená hodnotami $x, y, \alpha_1, \ldots, \alpha_r$, čo znamená, že $|H_\alpha(x,y)| \leq m^r$. □

Pozitívne na množine Vec je to, že
- vieme efektívne generovať náhodné h_α
- h_α sa efektívne počíta.

7.1.2 On line algoritmy

On line problém je taký, keď postupnosť vstupov prichádza postupne a rozhodnutia treba robiť priebežne. Otázka je, nakoľko dobrý môže byť algoritmus, ktorý nepozná dopredu budúcnosť v porovnaní s tým, ktorý celú budúcnosť pozná dopredu. Čo považujeme za kvalitu? *kvalita on-line algoritmov*

- kvalita riešenia
- zložitosť

dispečing obmedzený počet sanitiek/taxíkov/... obsluhuje požiadavky naň. Dispečér priebežne rozhoduje, koho kam pošle. Kritéria optimalizácie *motivačné príklady*

- celkový počet najazdených km
- čas čakania pacientov/pasažierov -celkový resp. maximalizovaný na jedného

rozvrhovanie/scheduling analogicky - máme niekoľko strojov rôznej kvality a požiadavky na ne. Prideľujeme úlohy strojom, pričom optimalizujeme

- celkový čas
- priemerné zaťaženie
- čas čakania

Majme optimalizačný problém $U = (I, Sol, m, ciel)$. Algoritmus A je online pre U, ak pre každý vstup $x = x_1, \ldots, x_n \in I$ *on-line algoritmus*

- $x_1, \ldots, x_i$ je prípustný vstup $\forall 1 \leq i \leq n$
- $\mathsf{A}(x) \in Sol(x)$
- $\mathsf{A}(x_1, \ldots, x_i)$ je časť $\mathsf{A}(x)$ $\forall 1 \leq i \leq n$

  vstup $x_1, \ldots, x_n \quad \rightsquigarrow \quad$ výstup $y_1, \ldots, y_n$, $y_i = f_i(x_1, \ldots, x_i)$

Kvalitu algoritmu A meriame vzhľadom k optimálnemu off-line algoritmu, resp. jeho cene. $Comp_{\mathsf{A}}(x)$

$$Comp_{\mathsf{A}}(x) = \max\left\{\frac{OPT_U(x)}{m_{\mathsf{A}}(x)}, \frac{m_{\mathsf{A}}(x)}{OPT_U(x)}\right\}$$

V prípade aproximačných algoritmov definujeme chybu a aproximačný prah ako hranicu "najlepšej" kvality, v prípade on-line algoritmov máme analogické pojmy.

Nech $\delta \geq 1$ je konštanta. Hovoríme, že on-line algoritmus A je δ-*competitive*, ak $\forall x\ Comp_{\mathsf{A}}(x) \leq \delta$. On-line problém je δ-ťažký, ak neexistuje d-competitive algoritmus na jeho riešenie pre žiadne $d < \delta$. *δ-ťažký problém*

Príklad 7.1 *Uvažujme nasledovný problém stránkovania. Máme cash/buffer veľkosti k, ostatné stránky sú v pomalej pamäti. Zo vstupu prichádzajú požiadavky na stránky. Ak požadovaná stránka s nie je v buffri, musíme niektorú zo stránok v buffri odstrániť/presunúť a vložiť/presunúť tam požadovanú stránku s; v takomto prípade realizujeme presun(resp. fault). Úlohou je zostrojiť on-line algoritmus, ktorý minimalizuje počet presunov medzi cash a hlavnou pamäťou.* *stránkovanie*

začiatok: $s_1, \ldots, s_k$ *v Cash*
$s_{k+1}, \ldots, s_n$ *v Main;* $n \ggg k$

inštancia: *postupnosť indexov* $i_1, i_2, \ldots, i_m$; $1 \leq i_j \leq n$

cieľ: *minimalizovať počet presunov medzi Cash a Main*

Ukážeme, že problém Stránkovania je k-ťažký. Dôkaz spočíva v popise stratégie protihráča, ktorý k ľubovoľnému algoritmu A skonštruuje "zlý" vstup x_A *protihráč*

1. *pri požiadavke na stránku s_{k+1} musí algoritmus A presunúť niektorú stránku z buffra-povedzme stránku s indexom j_1*
2. *po presunutí stránky s_{j_1} potom prichádza od protihráča požiadavka j_1; algoritmus A premiestňuje j_2*

   ⋮

Je zrejmé, že nech by sme mali akýkoľvek algoritmus, takto konštruovaný vstup $k+1, j_1, \ldots, j_{k-1}$ vyžaduje k presunov. Pritom optimálne—offline—riešenie v prvom kroku presunie $i \in \{1, \ldots, k\} - \{j_1, \ldots, j_{k-1}\}$, preto mu na spracovanie rovnakej postupnosti stačí len jeden presun.

$$Comp_A(x_A) = \frac{costA(x_A)}{OPT(x_A)} = \frac{k}{1} = k$$

◇

Definícia 7.2 *Pravdepodobnostný algoritmus A je* pravdepodobnostný online algoritmus *pre problém U ak $\forall x_1 \ldots x_n \in L$ a $\forall 1 \leq i \leq n-1$*

- *výstup každého behu $A(x_1, \ldots, x_i)$ je prípustné riešenie*
- *pre každý vsup $C(x_1, \ldots, x_i), x_{i+1}$, kde $C(x_1, \ldots, x_i) \in M(x_1, \ldots, x_i)$, všetky behy A dopočítajú prípustné riešenie pre $x_1, \ldots, x_{i+1}$*

$$\begin{array}{lll} \textit{vstup } x_1 \ldots x_i x_{i+1} & \rightsquigarrow & \textit{výstup } y_1, \ldots, y_{i+1} \\ & & y_{i+1} = f_{i+1}(x_1, \ldots, x_{i+1}) = g_{i+1}(f_i(x_1, \ldots, x_i), x_{i+1}) \end{array}$$

Pre inštanciu vstupu X máme
$\mathbf{S_{A,X}}$ je množina výpočtov A na X
$\mathbf{Prob_{A,X}}$ je odpovedajúce pravdepodobnostné rozdelenie na $S_{A,X}$
$\mathbf{Z_X(C)} = Comp_C(X)$ je náhodná premenná v $(S_{A,X}, Prob_{A,X})$

Kvalitu meriame očakávanou hodnotou
$\mathbf{Exp - Comp_A(X)} = E[Z_X]$
Nech $\delta \in R$. Hovoríme, že A je **E[δ]-competitive**, ak $Exp - Comp_A(X) = E[Z_X] \leq \delta$
Nech $h : \mathbb{N} \to R^{\geq 1}$. Hovoríme, že A je **Exp[h]-competitive**, ak $Exp - Comp_A(X) = E[Z_X] \leq h(|x|)$

Cieľom tejto časti je na probléme rozvrhovania ukázať, že existujú problémy, pre ktoré pravdepodobnostný algoritmus dosahuje lepšiu kvalitu ako najlepší možný deterministický algoritmus.

problém

Problém Rozvrhovanie predpokladá

- máme m rôznych (typov) strojov $M_1, \ldots, M_m$, pričom z každého typu stroja je len jeden kus
- *job* je m úloh $A_1, \ldots, A_m$ reprezentovaných permutáciou indexov $i_1, \ldots, i_m$; job $(i_1, \ldots, i_m)$ znamená, že
  - úlohy treba počítať v poradí $A_1, \ldots, A_m$; najskôr jednu úlohu dokončiť, potom druhú začať
  - A_j sa musí riešiť na stroji M_{i_j}
  - idealizovane predpokladáme, že výpočet na každom stroji trvá presne jednotku času

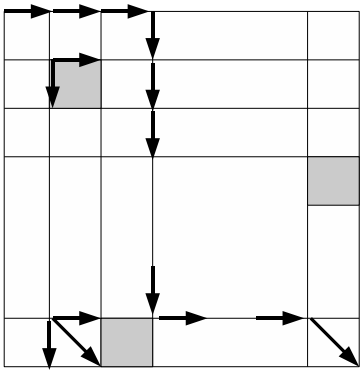
Unit(m,d), resp. skrátene $U(m, d)$

vstup: m typov strojov, d jobov(m-tíc úloh)
výstup: pridelenie úloh jednotlivým strojom v diskrétnom čase tak, aby každý robil vždy len na jednej úlohe a aby príslušná úloha mala k dispozícii všetky potrebné výsledky (zachovanie poradia)

Budeme uvažovať najjednoduchšiu verziu úlohy $U(m,2)$. Vstupom sú dva vektory $\alpha = (i_1, \ldots, i_m)$ a $\beta = (j_1, \ldots, j_m)$. Predstavme si ich ako $m \times m$ mriežku $G(\alpha, \beta)[j,k]$, pričom riadky odpovedajú β, stĺpce α. **Kolízia** nastáva, ak $G[k,\ell] = j_k = i_\ell$

Ľahko vidno, že ak úlohy $j_1, \ldots, j_{k-1}$ a $i_1, \ldots, j_{\ell-1}$ skončili naraz a $j_k \neq i_\ell$, potom možno úlohy j_k a i_ℓ počítať paralelne; inak jedna z nich musí počkať.

Do mriežky na miesta kolízií dáme značku. Hýbať sa možno smerom doprava a dole a to po riadkoch, stĺpcoch a diagonálach, ak tam nie je značka. Priradenie je cesta z ľavého horného do pravého spodného rohu.



Pohyb po horizontálnej šípke aj pohyb po vertikálnej šípke odpovedajú tomu, že jedna úloha stojí. Keďže máme štvorcovú mriežku, je počet vertikálnych a horizontálnych šípok na každej ceste rovnaký.

Majme teda cestu S
$del_\alpha(S)$, počet vertikálnych hrán na ceste S
$del_\beta(S)$, počet horizontálnych hrán na ceste S
$\Big\}\ delay(S) = del_\alpha(S) = del_\beta(S)$

$$cost(S) = m + delay(S) = m + \frac{del_\alpha(S) + del_\beta(S)}{2}$$

Fakt 7.8 *Pre $m \in \mathbb{N}$ má každá inštancia (α, β) pre $U(m,2)$ presne m konfliktov, pričom v každom riadku a každom stĺpci je práve jeden konflikt.*

Najprv ukážeme, že vieme konštruovať zlé vstupy.

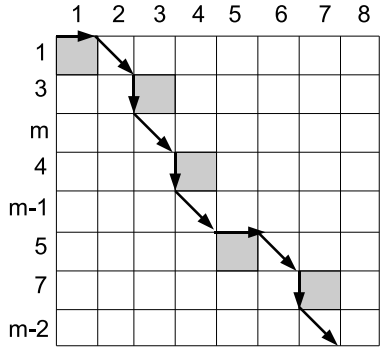
Lema 7.9 *Nech $m = 0 \mod 8$. Pre každý online algoritmus A riešiaci $U(m,2)$ existuje taká inštancia $I = ((1,2,\ldots,m), \beta)$, že*

$$cost(\mathsf{A}(I)) \geq m + \frac{m}{8}$$

Dôkaz: Popíšeme stratégiu protivníka, ktorý bude ukladať konflikty tak, že časť cesty, ktorá prejde za $m/2$ ty riadok a $m/2$-ty stĺpec, použije aspoň polovicu hrán, ktoré nie sú diagonálne. To bude znamenať, že nabrala zdržanie aspoň polovicu z $\frac{1}{2}\frac{m}{2}$, čo je požadovaných $m/8$.
Konštruujeme $\beta = j_1, \ldots, j_m$

- $j_1 = 1$ spôsobí, že máme konflikt. Algoritmus môže značku obísť po vertikálnej alebo horizontálnej hrane, oboje spôsobí prírastok 1 do príslušnej $del_\alpha(S)$ alebo $del_\beta(S)$
- Ak algoritmus obišiel značku po horizontálnej hrane, α má riešiť úlohu 2, β úlohu 1–toto sa dá riešiť paralelne. Sme v pozícii, keď α ide riešiť úlohu 3 a my (protiháč) chceme, aby aj β riešila úlohu 3–dáme teda $j_2 = 3$

- Ak algoritmus obišiel značku po vertikálnej hrane, α má riešiť úlohu 1, β novú úlohu j_2. Aby sa to dalo riešiť paralelne, položíme $j_2 = m$ a potom voľba $j_3 = 2$ spôsobí opäť konflikt.



Takto postupujeme ďalej. Ak algoritmus obchádza prekážku po horizontálnej hrane, doplníme na príslušné miesto v β prvok z množiny $\{1, 2, \ldots, m/2\}$. Ak ju obíde po vertikálnej hrane, budeme vkladať prvok z množiny $\{m/2 + 1, \ldots, m\}$ □

Teraz odhadneme zložitosť najlepšieho offline deterministického algoritmu. Vezmeme množinu konkrétnych deterministických stratégií, spočítame strednú hodnotu pri ich použití. Potom optimálny algoritmus nemôže byť horší ako táto stredná hodnota.

Lema 7.10 *Nech $m \in \mathbb{N}$. Pre každý vstup I do $U(m, 2)$ platí*

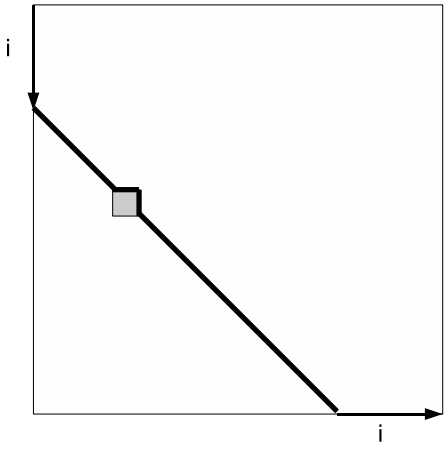
$$Opt(I) \leq m + \sqrt{m}$$

Dôkaz: Kvôli zjednodušeniu uvažujme $m = k^2$.

Pre $i = 0, 1, \ldots, k$ označme
D_i diagonálu, ktorá ide z $(0, i)$ do $((m-i, m))$
D_{-i} diagonálu, ktorá ide z $(i, 0)$ do $((m, m-i))$

Stratégia $A(i), i \in \{-\sqrt{m}, \ldots, 0, \ldots, \sqrt{m}\}$, príde po obvode k diagonále D_i, potom sa snaží ísť po diagonále D_i a následne po obvode najkratšou cestou do pravého spodného rohu. Ak sú na diagonále prekážky, obchádza ich jednou horizontálnou a jednou vertikálnou hranou.

Analogicky stratégia $A(-i)$ (pozri obr.)



Pre cenu stratégie $A(i)$ potom platí
$cost(A(i)) = |i| + (m - |i|) + |i| +$ počet prekážok $= m + \underbrace{|i| + \text{počet prekážok}}_{\text{zdržanie}}$

Keďže je počet všetkých prekážok (v mriežke) m, pre súčet diagonálnych stratégií dostávame

$$m + \sum_{i=-\sqrt{m}}^{\sqrt{m}} |i| = m + 2\sum_{i=1}^{\sqrt{m}} |i| = m + 2\frac{\sqrt{m}(\sqrt{m}+1)}{2} = 2m + \sqrt{m}$$

Priemerný počet zdržaní je potom

$$\frac{2m + \sqrt{m}}{2\sqrt{m} + 1} < \frac{2m + \sqrt{m}}{2\sqrt{m}} = \sqrt{m} + 1/2$$

Existuje teda algoritmus so zdržaním nanajvýš $\sqrt{m}$ a optimálny od neho nemôže byť horší. Preto $Opt(I) \leq m + \sqrt{m}$. □

Na základe lemy 7.9 a lemy 7.10 máme nasledujúce tvrdenie.

Veta 7.11 *Problém $U(m,2)$ je $(9/8 - \varepsilon)$-ťažký.*

Dôkaz: Z lemy 7.9 vieme, že ku každému algoritmu A existuje inštancia I, na ktorej je $cost_{\mathsf{A}}(I) \geq m + \frac{m}{8}$. Lema 7.10 zas ohraničuje cenu optimálneho algoritmu. Preto

$$Comp_{\mathsf{A}}(I) = \frac{cost_{\mathsf{A}}(I)}{Opt(I)} \geq \frac{\frac{9}{8}m}{m+\sqrt{m}} = \frac{9}{8}\left(1 - \underbrace{\frac{1}{\sqrt{m}+1}}_{\varepsilon}\right)$$

□

A teraz konečne ten pravdepodobnostný algoritmus

Algoritmus 39 DIAG

1: náhodne zvoľ $i \in \{-\sqrt{m}, \ldots, \sqrt{m}\}$
2: rieš stratégiou $A(i)$

Veta 7.12 *Algoritmus DIAG je pravdepodobnostný online, pričom pre každú inštanciu vstupu I do $U(m,2)$ platí*

$$ExpComp_{DIAG}(I) \leq 1 + \frac{1}{\sqrt{m}} + \frac{1}{2m}$$

Dôkaz: Na základe predchádzajúcich úvah

- očakávaný počet zdržaní pre stratégiu $A(i)$ je nanajvýš $\lceil\sqrt{m}\rceil + 1/2$.
- očakávaný čas je nanajvýš $m + \lceil\sqrt{m}\rceil + 1/2$
- $Opt \geq m$

$$ExpComp_{DIAG}(i) = \frac{\text{očakávaná cena}}{Opt(I)} \leq \frac{\text{očakávaná cena}}{m} \leq \frac{m + \lceil\sqrt{m}\rceil + 1/2}{m} = 1 + \frac{1}{\sqrt{m}} + \frac{1}{2m}$$

□

O probléme $U(m,2)$ sme ukázali *záverom*
- je $(9/8 - \varepsilon)$-ťažký
- existuje pravdepodobnostný algoritmus s ExpComp nanajvýš $1 + \frac{1}{\sqrt{m}} + \frac{1}{2m}$

7.2 Metóda odtlačkov

Ako sme už povedali, metóda odtlačkov sa používa, keď nás zaujíma rovnosť/ekvivalencia nejakých komplexných objektov.

1. Vezmeme množinu M vhodných zobrazení úplných reprezentácií do čiastočných reprezentácií (napr. $N \to PRIM(n^2)$)
   $h \leftarrow random(M)$
2. vypočítame redukované reprezentácie $h(O_1)$, $h(O_2)$; tomu sa hovorí odtlačok/stopa/fingerprint
3. porovnáme skrátené reprezentácie
   if $h(O_1) = h(O_2)$ **then return** ("ekvivalentné")
   else return ("nie sú ekvivalentné")

K úspešnému použitiu potrebujeme

- $h(O)$ efektívne počítateľné
- porovnanie $h(O_1) \stackrel{?}{=} h(O_2)$ efektívne vypočitateľné
- existenciu dostatočne veľkej množiny kandidátov/svedkov v M, ktorí v prípade $O_1 \neq O_2$ potvrdzujú, že $O_1 \neq O_2$

Dôraz je na konštrukcii množiny kandidátov M. Jej kardinalita je určená dĺžkou/komplexnosťou popisu jednotlivých objektov a tak proti sebe idú snaha o dostatočné množstvo kandidátov a snaha o krátky popis.

7.2.1 Porovnávanie databáz

Aplikáciou tejto metódy je Algoritmus 29 na porovnanie dvoch vzdialených databáz. Analogickým spôsobom môžme riešiť viaceré podobné príklady

Príklad 7.2 *Majme vzdialené databázy. X, U_i, V_j označujú podľa kontextu reťazec ale aj číslo s binárnym zápisom X, resp. U_i, V_j*

R_I: $X = x_1 \dots x_n$
R_{II}: $U = \{U_1, \dots, U_k\}$; $U_i \in \{0,1\}^n$

Zaujíma nás, či $X \in U$.
Množina zobrazení/vytváraní odtlačkov vzniká využitím prvočísel – pre prvočíslo p označuje h_p funkciu, ktorá ako odtlačok berie $\bmod p$

$$M = \{h_p \mid p \in PRIM(n^2)\}$$

Algoritmus je jasný

Algoritmus 40 — Test X ∈U

RI:

1: náhodne vyber $h_p \in M$
2: $s \leftarrow X \mod p$
3: pošli s, p druhému počítaču RII

RII:

1: vypočítaj $q_i \leftarrow U_i \mod p$
2: **if** $s \in \{q_1, \dots, q_k\}$ **then** return "$X \in U$"
3: **else** return "X∉ U"

chyba

Je zrejmé, že keď $X \in U$, ľubovoľná voľba h_p (výpočet C(p)) vedie ku korektnej odpovedi. Chyby sa môžme dopustiť len vtedy, ak $X \notin U$ ale existuje U_i tak, že $X \equiv U_i \mod p$. Vieme, že pre konkrétne i je počet takýchto "zlých" prvočísel nanajvýš $n-1$, pričom n je dĺžka zápisu X. Ak teda označíme

$$A_i = \{p \mid p \in PRIM(n^2) \ \& \ s = q_i v \text{ behu } C(p)\}$$

$$A = \bigcup_{i=1}^{k} A_i$$

tak

$$Error(X,U) = Prob(A) = \sum_{i=1}^{k} Prob(A_i) \leq \sum_{i=1}^{k} \frac{2\ln n}{n} = k \cdot \frac{2\ln n}{n}$$

Pre $k \leq \frac{n}{4\ln n}$ je pravdepodobnosť chyby nanajvýš 1/2. Inými slovami, ak mohutnosť množiny U je nanajvýš $\frac{n}{4\ln n}$, je Algoritmus 7.2 Monte Carlo algoritmus s jednosmernou chybou.

Analogicky riešime problém neprázdneho prieniku dvoch databáz.

Príklad 7.3 *Majme vzdialené databázy*
R_I: $V = \{V_1 \dots V_\ell\}; V_i \in \{0,1\}^n$
R_{II}: $U = \{U_1, \dots, U_k\};\ U_i \in \{0,1\}^n$
Zaujíma nás $V \bigcap U \stackrel{?}{=} \emptyset$.

Algoritmus 41 — Test V∩U

RI:

1: náhodne vyber $h_p \in M$
2: $s_i \leftarrow V_i \mod p$
3: pošli $s_1, \dots, s_\ell, p$ druhému počítaču RII

RII:

1: vypočítaj $q_i \leftarrow U_i \mod p$
2: **if** $\{s_1, \dots, s_\ell\} \cap \{q_1, \dots, q_k\} = \emptyset$ **then** return "$V \cap U = \emptyset$"
3: **else** return "$V \cap U \neq \emptyset$"

Algoritmus prekomunikuje $(\ell+1)2\lceil \log n \rceil$ bitov. *komunikácia*

Je zrejmé, že keď $V \cap U = \emptyset$, ľubovoľná voľba h_p (výpočet C(p)) vedie ku korektnej odpovedi. Chyby sa môžme dopustiť len vtedy, ak $V \cap U \neq \emptyset$, ale existujú U_i, V_j tak, že $U_i \equiv V_j \mod p$. Ak teda označíme B_i udalosť, keď $s_i \in \{q_1, \dots, q_k\}$, tak *chyba*

$$Error(V,U) = Prob(\bigcup_{i=1}^{\ell} B_i) \leq \sum_{i=1}^{\ell} k \cdot \frac{2\ln n}{n} = \ell k \cdot \frac{2\ln n}{n}$$

Pre $\ell k = o(\frac{n}{\ln n})$ máme Monte Carlo algoritmus s jednosmernou chybou.

Uvedomme si, že ak zväčšíme množinu M tak, že budeme uvažovať prvočísla $p \in PRIM(n^d)$, počet "zlých" prvočísel sa nezmení, ale zmenší sa pravdepodobnosť chyby. Pre $X = x_1 \dots x_n, Y = y_1 \dots y_n$ totiž

$$Pr[X \equiv Y \mod p \mid X \neq Y] \leq \frac{n-1}{Prim(n^d)} \leq \frac{d\ln n}{n^{d-1}}$$

Vo všetkých uvedených príkladoch sme objekty porovnávali na základe "odtlačkov", ktorými boli zvyšky po delení prvočíslom. Iným prístupom je Freivaldsova metóda, ktorá ako odtlačky používa hodnotu polynómu/matice/funkcie v bodoch.

7.2.2 Freivaldsova metóda - ekvivalencia polynómov

Príkladom využitia Freivaldsovej metódy bol Algoritmus 32 na porovnanie $AB = C$ pre matice A,B,C. Zvolili sme náhodný vektor α a odpoveď na otázku $AB \stackrel{?}{=} C$ sme spravili na základe výsledku porovnania $AB\alpha \stackrel{?}{=} C\alpha$. Analogický postup môžme využiť pre porovnávanie polynómov (a iných štruktúr, ktoré sa na porovnanie polynómov dajú transformovať).

Vstupom sú polynómy $P_1(X), P_2(X)$ stupňa nanajvýš n. Zaujíma nás, či $P_1(X) = P_2(X)$. Ak chceme deterministický algoritmus na porovnanie polynómov, využijeme ich "normálny tvar"

$$P(X) = \sum_{i=0}^{n} c_i X^i$$

resp.

$$Q(x_1, \ldots, x_n) = \sum_{i_1=0}^{d} \cdots \sum_{i_n=0}^{d} c_{i_1} \cdots c_{i_n} x_1^{i_1 \cdots x_n^{i_n}}$$

pre polynóm $Q(x_1, \ldots, x_n)$ n premenných, z ktorých každá môže byť stupňa nanajvýš d. Toto deterministické porovnanie má tú nevýhodu, že "normálny" tvar môže byť až exponenciálne dlhý vzhľadom k zadanému tvaru polynómu; napr. $(x_1 + x_2)(x_1 + x_3) \cdots (x_1 + x_n)$

Uvažujme tento jednoduchý algoritmus

Algoritmus 42 — Test $P_1(X) = P_2(X)$

Nech polynómy P_1, P_2 stupňa n majú premenné z poľa F, nech $S \subseteq F$, $|S| \geq n+1$

1: náhodne vyber $r \in S$
2: $p_1 \leftarrow P_1(r),\ p_2 \leftarrow P_2(r)$
3: **if** $p_1 = p_2$ **then** return "$P_1(X) = P_2(X)$"
4: **else** return "$P_1(X) \neq P_2(X)$"

chyba

Potrebujeme odhadnúť pravdepodobnosť chybnej odpovede. Ak sú polynómy rovnaké, chyby sa evidentne nedopustíme. Chyba nastane vtedy, ak pre polynóm $Q(X) = P_1(X) - P_2(X)$, ktorý nie je identicky rovný 0, zvolíme r tak, že $Q(r) = 0$.

Fakt 7.13 *Polynóm stupňa d má nanajvýš d rôznych koreňov.*

Na základe faktu 7.13 je pravdepodobnosť chyby nanajvýš $\frac{n}{|S|}$. Ak chceme chybu zmenšiť, môžme zväčšiť množinu S, resp. opakovať niekoľko behov algoritmu. Uvedomme si, že ak $Q(X) \neq 0$, potom $n + 1$ behov s *rôznymi* hodnotami r to musí potvrdiť.

"ponaučenie"

Zhrnutím dostávame, že test na rovnosť polynómov $P_1(X) \stackrel{?}{=} P_2(X)$ je rozumné transformovať na test $P_1(X) - P_2(X) \stackrel{?}{=} 0$

Pri prechode k polynómom viacerých premenných využijeme nasledujúcu vetu.

Veta 7.14 (Schwartz-Zippel) *Nech $Q(x_1, \ldots, x_n) \in F[x_1, \ldots, x_n]$ je polynóm viacerých premenných celkového stupňa d. Nech $S \subseteq F$ a $r_1, \ldots, r_n$ sú náhodne vybraté z S. Potom*

$$Pr[Q(r_1, \ldots, r_n) = 0 \mid Q(x_1, \ldots, x_n) \neq 0] \leq \frac{d}{|S|}$$

Dôkaz: Postupujeme indukciou vzhľadom na počet premenných n.

n=1 Polynóm stupňa d má nanajvýš d rôznych koreňov. ✓

n>1 Nech $k \leq d$ je maximálny stupeň exponentu premennej, povedzme že je to x_1. Potom

$$Q(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2, \ldots, x_n)$$

Vzhľadom na výber vieme, že

- koeficient $Q_k(x_2, \ldots, x_n)$ pri x_1^k nie je nula
- stupeň polynómu $Q_k(x_2, \ldots, x_n)$ je nanajvýš $d - k$
- pravdepodobnosť, že $Q_k(r_2, \ldots, r_n) = 0$ je nanajvýš $\frac{d-k}{|S|}$

Nech teda $Q_k(r_2, \ldots, r_n) \neq 0$. Uvažujme polynóm

$$q(x_1) = Q(x_1, r_2, \ldots, r_n) = \sum_{i=0}^{k} x_1^i Q_i(r_2, \ldots, r_n)$$

Stupeň polynómu $q(x_1)$ je nanajvýš k, nie je identicky rovný 0, preto

$$Pr[q(r_1) = 0 \mid q(x_1) \not\equiv 0] \leq \frac{k}{|S|}$$

Využijúc $Pr[A] \leq Pr[A|\overline{B}] + Pr[B]$ dostávame

$$Pr[Q(r_1, \ldots, r_n) = 0 \mid Q(x_1, \ldots, x_n) \neq 0] \leq \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}$$

□

Ak počítame $\mod p$, kde p je prvočíslo, chybu môžme odhadnúť pomocou nasledujúcej lemy.

Lema 7.15 *Nech p je prvočíslo, $Q(x_1, \ldots, x_n) \neq 0$ polynóm nad Z_p, $d \in \mathbb{N}$ je maximálny stupeň premenných. Potom Q má nanajvýš $n \cdot d \cdot p^{n-1}$ koreňov.*

Dôkaz: Analogicky ako v predchádzajúcom dôkaze-indukcia cez počet premenných.

n=1 Počet koreňov je nanajvýš $d = ndp^{n-1}$

n>1 $Q(x_1, \ldots, x_n) = \sum_{i=0}^{d} x_1^i Q_i(x_2, \ldots, x_n)$. Ak $Q(\alpha_1, \ldots, \alpha_n) \equiv 0 \mod p$, tak alebo

(a) $Q_i(\alpha_2, \ldots, \alpha_n) \equiv 0 \mod p$ pre každé i, alebo

(b) $Q_j(\alpha_2, \ldots, \alpha_n) \not\equiv 0 \mod p$, pričom α_1 je koreňom polynómu

$$q(x_1) = Q_0(\alpha_2, \ldots, \alpha_n) + x_1 Q_1(\alpha_2, \ldots, \alpha_n) + \cdots + x_1^d Q_d(\alpha_2, \ldots, \alpha_n)$$

Spočítame obe možnosti zvlášť. Keďže $Q(x_1, \ldots, x_n) \not\equiv 0$, musí existovať taký index k, že polynóm $Q_k(x_2, \ldots, x_n) \not\equiv 0$. Ten má podľa IP nanajvýš $(n-1)dp^{n-2}$ koreňov takých, že $Q_i(\alpha_2, \ldots, \alpha_n) \equiv 0 \mod p$. Keď dovolíme α_1 ľubovoľne, je počet koreňov $(\alpha_1, \ldots, \alpha_n)$ v prípade (a) nanajvýš $(n-1)dp^{n-1}$. *(a)*

Polynóm $q(x_1)$ je polynóm jednej premennej stupňa d, preto má nanajvýš d koreňov. V tomto prípade máme teda nanajvýš dp^{n-1} koreňov. *(b)*

Celkovo teda je koreňov nanajvýš

$$(n-1)dp^{n-1} + dp^{n-1} = ndp^{n-1}$$

□

Algoritmus 42 ľahko upravíme na Monte Carlo algoritmus s jednosmernou chybou, ktorý rieši porovnanie polynómov viacerých premenných.

Algoritmus 43 AQP

Polynómy P_1, P_2 n premenných nad Z_p, maximálny stupeň jednotlivých premenných d, prvočíslo p

1: náhodne vyber $\alpha \in \{0,1\}^n$
2: $h_\alpha(P_1) \leftarrow P_1(\alpha) \mod p$
3: $h_\alpha(P_2) \leftarrow P_2(\alpha) \mod p$
4: **if** $h_\alpha(P_1) = h_\alpha(P_2)$ **then** return "$P_1 \equiv P_2$"
5: **else** return "$P_1 \not\equiv P_2$"

Pravdepodobnosť, že v prípade $Q() = P_1() - P_2() \not\equiv 0$ zvolíme náhodne α tak, že $Q(\alpha) \not\equiv 0$, je (prečo?) aspoň $\left(1 - \frac{ndp^{n-1}}{p^n}\right) = \left(1 - \frac{nd}{p}\right)$.

Uvedený algoritmus môžme využiť na pravdepodobnostné zodpovedanie otázky, či má daný bipartitný graf $G(U,V,E), E \subseteq U \times V$ úplné párovanie (perfect matching)[1].

úplné párovanie

Úplné párovanie je zrejme možné len vtedy, keď $|U| = |V|$. Predpokladáme teda, že $U = \{u_1, \ldots, u_n\}, V = \{v_1, \ldots, v_n\}$. Úplné párovanie je vtedy dané permutáciou π, ktorá hovorí, že v párovaní sú hrany $(u_i, v_{\pi(i)})$. Využijeme Edmondsonovu vetu

Veta 7.16 (Edmondson) *Nech A je matica typu $n \times n$, ktorú k bipartitnému grafu $G(U,V,E)$ skonštruujeme nasledovne*

$$A[i,j] = \begin{cases} x_{ij}, & (u_i, v_j) \in E \\ 0, & (u_i, v_j) \notin E \end{cases}$$

Definujme polynóm $Q(x_{11}, \ldots, x_{nn}) = det(A)$.[2] Potom G má úplné párovanie práve vtedy ak $Q \not\equiv 0$.

Dôkaz: $det(A) = \sum_{\pi \in S(n)} -1^{sgn(\pi)} A[1,\pi(1)]A[2,\pi(2)] \ldots A[n,\pi(n)]$; tu $S(n)$ je symetrická grupa permutácií a $sgn(\pi)$ je parita počtu výmen, ktorými z identickej permutácie dostaneme permutáciu π. Keďže každé x_{ij} je v matici nanajvýš raz, sumáciou sa nemôžu "vynulovať". Preto determinant nie je identicky rovný 0 práve vtedy, keď existuje permutácia, pre ktorú je príslušný člen nenulový. To je ale ekvivalentné tomu, že

$$\left.\begin{array}{ll} A[1,\pi(1)] \neq 0 \Rightarrow & (1,\pi(1)) \in E \\ A[2,\pi(2)] \neq 0 \Rightarrow & (2,\pi(2)) \in E \\ & \ldots \\ A[n,\pi(n)] \neq 0 \Rightarrow & (n,\pi(n)) \in E \end{array}\right\} \text{úplné párovanie}$$

□

Determinant je polynóm, preto testovanie determinantu je testom polynómu na nulu a na to máme pravdepodobnostný test. Využitie tohto prístupu—súvis párovania a determinantu matice/polynómu—je skôr v konštrukcii paralelného algoritmu na hľadanie úplného párovania.

[1] Párovanie je taká množina hrán, v ktorej je každý vrchol obsiahnutý nanajvýš raz. Párovanie je úplné, ak je v ňom každý vrchol obsiahnutý práve raz.

[2] $det(A)$ označuje determinant matice A.

7.2.3 Porovnávanie reťazcov

Venujme sa teraz problému porovnávania/vyhľadávania textov (pattern matching).

vstup: $X = x_1 \dots x_n \; x_i \in \Sigma$
$Y = y_1 \dots y_m \;\; y_j \in \Sigma$
výstup k pozícia výskytu Y v X: $x_k \dots x_{k+m-1} = Y$
$k = n - m + 2$ znamená, že Y sa v X nevyskytuje

"Klasické" deterministické prístupy sú dva

1. $O(n \cdot m)$ "brute force" algoritmus založený na prikladaní Y postupne na pozície $1, 2$ až $n - m + 1$
2. $O(n + m)$ algoritmus využívajúci predspracovanie Y

Predspracovanie vzorky zadanej reťazcom znakov

vsuvka - predspracovanie vzorky

Predstavme si, že priložíme vzorku k textu tak, že začína na pozícii p. Nech sa zhoduje s textom na prefixe dĺžky i. Ak sa nasledujúci $(i+1)$-ý symbol vzorky líši od odpovedajúceho symbolu textu, musíme vzorku v texte posunúť. Vieme povedať, o koľko symbolov? Položme túto otázku inak – koľko už prečítaných symbolov textu môžme považovať za začiatok vzorky? Predpokladajme, že odpoveďou na túto otázku je hodnota funkcie f:
Pre fixovanú vzorku $b_1 b_2 \dots b_m$ nech $\mathbf{f(i)}$ je maximálna dĺžka *vlastného* sufixu reťazca $b_1 b_2 \dots b_m$, ktorý sa rovná prefixu tohto reťazca

$$f(i) = \max\{j \mid b_1 b_2 \dots b_j = b_{i-j+1} \dots b_i\}$$

$$\begin{array}{l} a_1 a_2 \;\; \dots \; a_p a_{p+1} \dots \; a_{p+i-1} \dots a_n \\ \qquad\quad b_1 b_2 \quad \dots \; b_i \\ \qquad\qquad\quad b_1 \; \dots \; b_{f(i)} \end{array}$$

Predstavme si, že pri čítaní textu postupne zľava doprava máme v premennej i index pozície čítaného symbolu a premenná *dlzka* obsahuje maximálnu dĺžku sufixu doteraz prečítanej časti textu, ktorú môžme považovať za začiatok vzorky. Vedeli by sme definovať funkciu *update*, ktorá by na základe i (resp. a_i) a *dlzka* vypočítala novú hodnotu *dlzka* tak, aby odpovedala prečítaniu symbola a_i zo vstupu?

Ak by sme poznali funkciu f, tak pomerne ľahko. Ak sa posledných *dlzka* symbolov prečítaného textu rovná $b_1 \dots b_{dlzka}$ a ak prečítaný symbol je $b_{dlzka+1}$, tak novou hodnotou *dlzka* bude $dlzka + 1$. Inak je nová hodnota *dlzka* určená na základe toho, ako by bola zmenená, keby jej hodnota bola $f(dlzka)$.

Algoritmus 44 Výpočet *update* pri znalosti chybovej funkcie f

1: $update(x, 0) \leftarrow 0$ pre $x \neq b_1$
2: $update(b_1, 0) \leftarrow 1$
3: **for** dlzka := 1 to m **do**
4: $\quad update(b_{dlzka+1}, dlzka) \leftarrow dlzka + 1$
5: $\quad update(b, dlzka) \leftarrow update(b, f(dlzka))$ pre $b \neq b_{dlzka+1}$

Osatáva vypočítať chybovú funkciu f. Pri jej výpočte si treba uvedomiť, že ak $f(i + 1) = s$, potom platí

$$b_1 b_2 \dots b_s = b_{i-s+2} \dots b_{i+1} \qquad \text{ale aj} \qquad b_1 b_2 \dots b_{s-1} = b_{i-s+2} \dots b_i$$

Algoritmus 45 Výpočet chybovej funkcie f

```
1: f(1) ← 0
2: for i:=2 to m do
3:     l ← f(i − 1)
4:     while (b_{l+1} ≠ b_i) ∧ (l > 0) do l ← f(l)
5:     if (b_{l+1} ≠ b_i) ∧ (l = 0) then f(i) ← 0
6:     elsef(i) := l + 1
```

Preto $\mathbf{s \leq f(i) + 1}$

Korektnosť riešenia je zrejmá z predchádzajúcej diskusie. **Zložitosť** získame na základe analýzy zložitosti algoritmov 44 a 45.

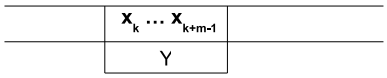
- zložitosť výpočtu tabuľky hodnôt $f(n)$ súvisí s počtom modifikácií globálnej premennej l. Keďže táto štartuje s počiatočnou hodnotou 0 a v priebehu výpočtu je buď zmenšovaná (príkazom $l \leftarrow f(l)$ v riadku 4) alebo zvýšená o 1 nanajvýš raz v cykle pre i(v riadku 6), je zložitosť výpočtu chybovej funkcie $O(m)$.
- zložitosťou výpočtu tabuľky pre funkciu *update* je úmerná veľkosti "tabuľky", ktorou je *update* zadaná, a preto je $O(m)$.
- Zložitosť samotného výpočtu je zrejme $O(n)$.

Fakt 7.17 *Celková zložitosť algoritmu vyhľadávania vzorky v texte s chybovou funkciou je* $\mathbf{O(m+n)}$.

---

vzorka pomocou odtlačkov

Metódu odtlačkov vieme využiť v (asymptoticky) neefektívnejšej verzii deterministického algoritmu– vzorku Y budeme postupne prikladať na jednotlivé pozície v X, porovnanie príslušných podreťazcov však spravíme cez ich odtlačky získané funkciou O_p. Opäť využijeme počítanie modulo prvočíslo p.



$$x_k \ldots x_{k+m-1} \stackrel{?}{=} Y \quad \Longrightarrow \quad O_p(x_k \ldots x_{k+m-1}) \stackrel{?}{=} O_p(Y)$$

Pre jednoduchosť predpokladáme, že $\Sigma = \{0, 1\}$. Potom reťazec dĺžky m môžme vnímať ako binárny zápis čísla a jeho odtlačkom bude zvyšok po delení prvočíslom p. Nech $X(j)$ označuje *číslo*, ktorého binárny zápis je podreťazec dĺžky m reťazca X, ktorý začína na pozícii j; analogicky Y.

Vezmime prvočíslo $p \leq \tau$ náhodne, $O_p(X) = X \mod p$. Ak by sme rovnosť odtlačkov považovali za definitívne určenie pozície, na ktorej sa Y v X nachádza, chyba algoritmu by bola

$$\sum_{j=1}^{n-m+1} Pr[O_p(Y) = O_p(X(j))]$$

Keďže pre binárne číslo dĺžky m máme nanajvýš $m-1$ "zlých" prvočísel, môžme písať

$$\sum_{j=1}^{n-m+1} Pr[O_p(Y) = O_p(X(j)) \mid Y \neq X(j)] < \frac{nm}{Prim(\tau)} = O\left(\frac{nm \log \tau}{\tau}\right)$$

To pri voľbe $\tau = f(n,m) = n^2 m \log n^2 m$ dáva pravdepodobnosť chyby MC algoritmu s jednosmernou chybou

$$\frac{nm\log(n^2 m\log n^2 m)}{n^2 m\log n^2 m} \leq \frac{\log(n^2 m)^2}{\log n^2 mn} = \frac{2\log n^2 m}{n\log n^2 m} = \frac{2}{n}$$

resp. $O(1/n)$.

Algoritmus 46 String(f(n,m))

▷ f určuje veľkosť použitých prvočísel

1: náhodne vyber $p \in PRIM(f(n,m))$
2: $O_p(Y) \leftarrow Y \mod p$
3: $O_p(X(0)) \leftarrow Num(x_1 \dots x_{m-1}) \mod p$
4: $k \leftarrow 1$
5: **while** k<n-m+1 **do**
6: $O_p(X(k)) \leftarrow (2[O_p(X(k-1)) - x_{k-1}2^{m-1} \mod p] + x_{k+m-1}) \mod p$
▷ $O_p(X) \leftarrow X(k) \mod p$; $x_0 = 0$
7: **if** $O_p(Y) = O_p(X(k))$ **then**
8: **if** $Y = X(k)$ **then** return "Y sa vyskytuje od pozície k"
▷ z pravdepodobnostného algoritmu robíme deterministický
9: **else** $k \leftarrow k+1$
10: return "Y sa v X nevyskytuje"

Ak však rovnosť odtlačkov použijeme len ako indikáciu *potenciálneho* výskytu Y v X a skutočnosť overíme "priložením" Y na príslušnú pozíciu, dostaneme algoritmus Las Vegas, ktorého každá odpoveď je korektná. V tomto prípade má zmysel sa pýtať, aká je jeho očakávaná časová zložitosť.

Pri realizácii algoritmu využívame efektívne počítanie $X(j) \mod p$, ktoré vychádza z rovnosti

$$X(j+1) = 2[X(j) - 2^{m-1}x_j] + x_{j+m}$$

Analyzujme Las Vegas verziu (Algoritmus 46). Očakávaná zložitosť je *čas*

$$O\left(\underbrace{(n+m)\left(1-\frac{1}{n}\right)}_{\text{žiaden potenciálny výskyt}} + \underbrace{mn\left(\frac{1}{n}\right)}_{\text{overovanie na každej pozícii}}\right) = O(n+m)$$

Upravme algoritmus tak, že po falošnej zhode vygenerujeme prvočíslo p náhodne. Potom pravdepodobnosť toho, že spravíme viac ako t reštartov, je nanajvýš $\frac{1}{n^t}$. *modifikácia Las Vegas*

Úloha: Porovnajte čas Las Vegas verzie a deterministického algoritmu s chybovou funkciou.

7.2.4 Interaktívne dôkazy*

Poslednou (našou) aplikáciou metódy odtlačkov sú tzv. *interaktívne dôkazy.* Predpokladajme, že niekto tvrdí, že pozná dôkaz. Našou úlohou je sa presvedčiť, že ho pozná bez toho, aby nám ho celý ukázal. Môžme vidieť iba "odtlačok" tohto dôkazu, pričom odtlačkom sú odpovede na otázky, ktoré budeme klásť. Kvalita dôkazov, ktoré takýmto spôsobom dokážeme overiť, zrejme závisí od kvantity odpovedí a kvality otázok. Ukážeme jeden ilustračný príklad—problém izomorfizmu a neizomorfizmu grafov.[3]

[3] Grafy $G_1 = (V, E_1)$, $G_2 = (V, E_2)$ sú izomorfné, ak existuje permutácia π taká, že $(i,j) \in E_1 \Leftrightarrow (\pi(i), \pi(j)) \in E_2$

Problém izomorfizmu grafov(GI): dvojicu vstupných grafov (G_1, G_2) akceptujeme, ak sú izomorfné. Ľahko vidno, že tento problém patrí do NP: uhádneme izomorfizmus τ a overíme rovnosť $\tau(G_1) = G_2$.

Problém neizomorfizmu grafov(GNI) dvojicu vstupných grafov (G_1, G_2) akceptujeme, ak nie sú izomorfné. Tento problém je z $co - NP$. Nemáme krátke dôkazy, asi je to ťažšie....

Na dokazovanie použijeme systém dvoch hráčov/algoritmov.

IP interaktívny dôkaz

V-verifier je pravdepodobnostný polynomiálny algoritmus, ktorý sa snaží overiť, že grafy G_1, G_2 nie sú izomorfné. Môže pritom klásť otázky dôkazu P

P-proover je algoritmus, ktorému neohraničujeme výpočtovú silu. Zakážeme mu jediné a to prístup k náhodným bitom algoritmu V

výpočet je komunikácia medzi nimi, pričom posledné slovo má V. Vyžadujeme

- ak G_1, G_2 nie sú izomorfné, potom P má šancu presvedčiť V
- ak G_1, G_2 sú izomorfné, potom akákoľvek snaha P vedie k akceptovaniu na strane V s pravdepodobnosťou nanajvýš 1/2

Takémuto komunikačnému algoritmu hovoríme interaktívny dôkaz. Konštrukcia komunikačného protokolu (IP) pre V a P na riešenie problému neizomorfizmu grafov GNI je jednoduchá.

IP pre GNI

V pracuje takto

- uhádne $i \in \{1, 2\}$ a permutáciu τ
- vypočíta $H = \tau(G_i)$
- pošle H do P a spýta sa na index i

P pošle V index j

V porovná si indexy i, j. Ak $i = j$, akceptuje, že G_1, G_2 sú neizomorfné; inak zamieta

Veta 7.18 *Ak G_1, G_2 nie sú izomorfné, čestný P presvedčí V. Inak je pravdepodobnosť toho, že (hoci aj nečestný) P presvedčí V, nanajvýš 1/2.*

Dôkaz: Analyzujeme podľa reálnej situácie

$G_1 \not\sim G_2$ Ak grafy nie sú izomorfné, výpočtovo neohraničene silný P dokáže nájsť aj korektný index i aj permutáciu τ, ktorou vzniklo $H = \tau(G_i)$. V tomto prípade V odpovedá korektne.

$G_1 \sim G_2$ Ak sú grafy izomorfné, potom $G_1 \sim H \sim G_2$. Akokoľvek silný P bez prístupu k náhodným bitom V nedokáže zistiť index, ktorý po ňom V chce. P si preto tipne, čo vedie k pravdepodobnosti chyby nanajvýš 1/2. □

Len pre zaujímavosť. Označme IP vyššie popísaný systém (V, P). Dá sa (nie jednoducho) ukázať, že IP=PSPACE.

7.3 Zvyšovanie úspešnosti opakovaním a náhodná vzorka

Tieto dve metódy sú natoľko podobné, že niekedy ťažko rozlíšiť, o ktorú z nich ide. Navyše, ich vhodná kombinácia dáva dobré výsledky.

opakovanie

Doteraz sme k znižovaniu chyby pod želanú hranicu využívali nezávislé opakovanie celých výpočtov. Metódu teraz potiahneme ďalej

- opakovanie nie celých výpočtov, ale len vybraných častí
- opakovanie rôznych častí výpočtu rôzny počet krát: častejšie budeme opakovať tie časti, v ktorých je pravdepodobnosť výskytu chyby vyššia.

náhodná vzorka

Pri hľadaní objektu danej vlastnosti postupujeme tak, že náhodne vyberáme kandidátske objekty o ktorých následne overujeme, či spĺňajú alebo nespĺňajú požadované vlastnosti. Ak je pravdepodobnosť toho, že objektov s danou vlatnosťou je dosť, veľká, poskytuje táto metóda "rozumné" riešenie pre viaceré problémy, pre ktoré rozumné/efektívne deterministické algoritmy nepoznáme.

7.3.1 Zlepšovanie úspešnosti opakovaním kritických častí

Vplyv premysleného opakovania pravdepodobnostného algoritmu na čas a kvalitu získaného algoritmu budeme prezentovať na probléme minimálneho rezu - Min-Cut:

vstup: multigraf[4] $G = (V, E, c)$ $c : E \to \mathbb{N} - \{0\}$ určuje násobnosť hrán
Prípustné riešenia: $M(G) = \{(V_1, V_2) \mid V_1 \cup V_2 = V,\ V_1 \cap V_2 = \emptyset\}$
je to množina hranových rezov
cena: $cena((V_1, V_2), G) = \sum_{\ell \in S(V_1,V_2)} c(\ell)$
$S(V_1, V_2) = \{(x, y) \in E \mid x \in V_1, y \in V_2\}$
cieľ: minimalizácia ceny

Najlepší známy deterministický algoritmus je zložitosti $O(|V| \cdot |E| \cdot \log \frac{|V|^2}{|E|})$, čo je v najhoršom prípade $O(n^3)$.

kontrakcia hrán

Pravdepodobnostný algoritmus je založený na *kontrakcii hrán*. Neformálne je kontrakcia hrany vlastne realizáciou rozhodnutia, že dva susediace (pseudo) vrcholy x, y budú v budúcnosti *oba* patriť do rovnakej časti rozkladu vrcholov V_1 alebo V_2. Dôsledky tohto rozhodnutia sa prejavia úpravou grafu: hranu nahradí pseudovrchol a hrany pôvodne smerujúce do x, resp. y budú smerovať do tohto vzniknutého pseudovrchola.

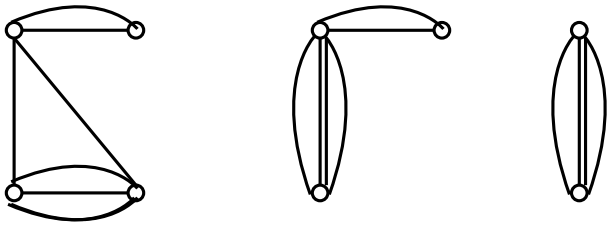
Kvôli jednoduchosti budeme v prípade potreby každý vrchol $v \in V$ považovať za jednoprvkovú *množinu*. Nech (x, y) je hrana; jej kontrakciou rozumieme

contract(G,(x,y)):

- zlúčenie vrcholov x, y do jedného (pseudo) vrchola: formálne vznikne vrchol $x \cup y$
- odstránenie vrcholov x, y a ich nahradenie pseudovrcholom $x \cup y$;
  $V \leftarrow V - \{x, y\} + x \cup y$
- odstránenie hrany (x, y) a hrán smerujúcich do jedného z (pseudo) vrcholov x, y a ich presmerovanie do pseudovrchola $x \cup y$:
  $E \leftarrow E - \{(u, v) \in E \mid u \in \{x, y\}, v \in V\} + \{(u, x \cup y) \mid (u, x) \in E \vee (u, y) \in E\}$

Takto vzniknutý graf označíme $G/(x, y)$. Nech $F = \{(x_1, y_1), \ldots, (x_k, y_k)\} \subseteq E^+$. Graf, ktorý vznikne postupným kontrahovaním hrán $(x_1, y_1), \ldots, (x_k, y_k)$ označíme

G/F. Všimnime si, že na poradí odstraňovania hrán nezáleží.



Algoritmus na výpočet minimálneho rezu je jednoduchý - náhodne zvolíme hranu, spravíme kontrakciu a opakujeme, kým graf neobsahuje dva (pseudo)vrcholy. Ako výsledok vezmeme množinu hrán medzi týmito vrcholmi (Algoritmus 47)

Algoritmus 47 Contraction

1: $label(v) \leftarrow v$
2: **while** G má viac ako 2 vrcholy **do**
3: náhodne vyber hranu $e = (x, y) \in E(G)$
4: $G \leftarrow contract(G, e)$
5: $label(z) \leftarrow label(x) \cup label(y)$, kde z je kontrakciou novovzniknutý vrchol
6: nech $G = (\{u, v\}, E(G))$: return$((label(u), label(v)), |E(G)|)$

Veta 7.19 *Algoritmus 47 je polytime pravdepodobnostný algoritmus, ktorý optimálne rieši problém Min-Cut s pravdepodobnosťou aspoň* $\frac{2}{n(n-1)}$*, kde* n *je počet vrcholov vstupného grafu* G.

Dôkaz: Začneme analýzou časovej zložitosti algoritmu 47.

čas

- *jedna kontrakcia* je úmerná prezretiu hrán prislúchajúcich k dvom vrcholom; keďže násobné hrany reprezentujeme pomocou funkcie c, stojí nás táto úprava čas $O(n)$
- *počet opakovaní* kontrahovania hrán je daný počtom vrcholov pôvodného grafu - s kontrakciami skončíme, keď kontrahovaný graf má 2 vrcholy. Počet iterácií je teda $n - 2$

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{O(n\cdot(n-2))=O(n^2)}$

korektnosť

Analýzu úspešnosti/chyby spravíme spočítaním pravdepodobnosti, že hrany z *fixovaného* minimálneho rezu C_{min} ostanú neskontrahované. Uvedomme si, že sústredenie sa na jeden konkrétny minimálny rez pravdepodobnosť chyby neznižuje. Nech $cena(C_{min}) = k$. Pri argumentácii využijeme nasledujúce-zjavné-fakty

- Keďže každá hrana má dva konce, počet hrán v grafe je rovný polovici súčtu stupňov jednotlivých vrcholov.
- Ak k je cena minimálneho hranového rezu, potom každý vrchol v G má stupeň aspoň k. Preto je počet hrán v grafe aspoň $\frac{nk}{2}$
- Algoritmus 47 počíta $C_{min} \iff$ žiadna hrana z $E(C_{min})$ nebola kontrahovaná

Označme

$\mathbf{E_i}$ = {výpočty, ktoré v i-tej iterácii nevybrali na kontrakciu hranu z $E(C_{min})$}

Zaujíma nás pravdepodobnosť udalosti $\bigcap_{\mathbf{i=1}}^{\mathbf{n-2}} \mathbf{E_i}$

$$Pr\left[\bigcap_{i=1}^{n-2} E_i\right] = Pr\left[E_1\right] \cdot Pr\left[E_2|E_1\right] \cdot Pr\left[E_3|E_1 \cap E_2\right] \cdots Pr\left[E_{n-2}|\bigcap_{i=1}^{n-3} E_i\right]$$

Postupne:

$$Pr\left[E_1\right] = \frac{|E| - |E(C_{min})|}{|E|} = 1 - \frac{k}{|E|} \geq 1 - \frac{k}{\frac{kn}{2}} = 1 - \frac{2}{n}$$

Po $i-1$ iteráciách—náhodných kontrakciách—máme graf G/F_{i-1} s $(n-i+1)$ vrcholmi, pričom každý z nich stále musí mať stupeň aspoň k. Ostalo nám teda aspoň $\frac{k(n-i+1)}{2}$ hrán, z ktorých vyberáme, pričom $|E(C_{min})|$ z nich je "zlých"

$$\mathbf{Pr}\left[\mathbf{E_i}|\bigcap_{\mathbf{j=1}}^{\mathbf{i-1}} \mathbf{E_j}\right] \geq \frac{|E(G/F_{i-1}) - E(C_{min})|}{|E(G/F_{i-1}))|} \geq 1 - \frac{k}{|E(G/F_{i-1})|} \geq 1 - \frac{k}{\frac{k(n-i+1)}{2}} = \mathbf{1 - \frac{2}{n-i+1}}$$

$$\mathbf{Pr}\left[\bigcap_{\mathbf{i=1}}^{\mathbf{n-2}} \mathbf{E_i}\right] \geq \prod_{i=1}^{n-2}\left(\mathbf{1 - \frac{2}{n-i+1}}\right) = \frac{n-2}{\mathbf{n}} \cdot \frac{n-3}{\mathbf{n-1}} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{\mathbf{2}}{4} \cdot \frac{\mathbf{1}}{3} = \prod_{\ell=n}^{3} \frac{\ell-2}{\ell} = \frac{2}{n(n-1)} = \mathbf{\frac{1}{\binom{n}{2}}}$$

□

Máme teda v $O(n^2)$ garantovanú úspešnosť $\frac{2}{n(n-1)} > \frac{2}{n^2}$, resp. pravdepodobnosť chyby nanajvýš $1 - \frac{2}{n^2}$. To znamená, že ak zopakujeme $n^2/2$ nezávislých behov tohto algoritmu a ako výsledok vezmeme najlepšie z riešení, chyba bude s pravdepodobnosťou nanajvýš[5]

$$\left(1 - \frac{2}{n^2}\right)^{n^2/2} < \frac{1}{e}$$

Lenže čas je $O(n^4)$....

Uvedený prístup je "naivný" v tom, že sme opakovali celé výpočty. Vidíme pritom, že pravdepodobnosť chyby so zväčšujúcim sa počtom iterácií rastie, kedže počet hrán, z ktorých vyberáme, klesá. Zdá sa preto rozumné proces kontrahovania pri vhodnej veľkosti ukončiť a minimálny rez dopočítať (deterministicky) presne. Zamyslime sa nad tým, dokedy má zmysel robiť kontrakciu, kedy je už graf dostatočne malý. Uvažujme funkciu $\ell(n)$, ktorá určuje ukončenie procesu kontrakcie—počet vrcholov grafu, pri ktorom zvyšok dopočítame deterministicky. Vyžadujeme, aby $1 < \ell(n) < n$, ℓ bola monotónna

počet iterácií riadený $\ell(n)$

Algoritmus 48 DetRan(ℓ)

1: aplikuj Algoritmus Contraction(47) na G dovtedy, kým nemá $\ell(n)$ vrcholov; výsledkom je graf G/F
2: aplikuj na G/F s $\ell(n)$ vrcholmi najlepší deterministický algoritmus D
3: return (D(G/F))

Analyzujme časovú zložitosť Algoritmu 48.

čas

$$\left.\begin{array}{ll} O((n-\ell(n)) \cdot n) = O(n^2), & \text{príspevok Algoritmu 47} \\ O((\ell(n))^3), & \text{príspevok algoritmu D} \end{array}\right\} O(n^2 + (\ell(n))^3)$$

úspešnosť

[5] $\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x = e$, $\lim_{x\to\infty}\left(1-\frac{1}{x}\right)^x = e^{-1}$

Chyba Algoritmu 48 závisí len od prvej časti, v ktorej realizujeme kontrakcie, potom už je algoritmus deterministický.

$$\begin{aligned}\mathbf{Pr}\left[\bigcap_{\mathbf{i=1}}^{\mathbf{n-\ell(n)}} \mathbf{E_i}\right] &\geq \prod_{i=1}^{n-\ell(n)}\left(1-\tfrac{2}{n-i+1}\right) = \\ &= \frac{\prod_{i=1}^{n-2}\left(1-\frac{2}{n-i+1}\right)}{\prod_{j=n-\ell(n)+1}^{n-2}\left(1-\frac{2}{n-j+1}\right)} = \frac{\frac{1}{\binom{n}{2}}}{\frac{1}{\binom{\ell(n)}{2}}} = \frac{\binom{\ell(\mathbf{n})}{\mathbf{2}}}{\binom{\mathbf{n}}{\mathbf{2}}}\end{aligned}$$

Dokázali sme nasledujúce tvrdenie

Lema 7.20 *Ak $1 < \ell(n) < n$ je monotónna funkcia, tak Algoritmus DetRan 48 je polynomiálny pravdepodobnostný algoritmus, ktorý nájde minimálny rez s pravdepodobnosťou aspoň* $\frac{\binom{\ell(n)}{2}}{\binom{n}{2}}$

voľba $\ell(n)$

Ostáva vhodne zvoliť $\ell(n)$. Podľa predchádzajúcej lemy 7.20 je pravdepodobnosť úspechu algoritmu DetRan aspoň

$$\frac{\binom{\ell(n)}{2}}{\binom{n}{2}} \leq \frac{\ell^2(n)}{n^2}$$

Ak teda spravíme $\frac{n^2}{\ell^2(n)}$ nezávislých opakovaní algoritmu DetRan(ℓ), dosiahneme v čase

čas

$$O\left((n^2+\ell^3(n))\cdot\frac{n^2}{\ell^2(n)}\right) = O\left(\frac{n^4}{\ell^2(n)} + n^2\ell(n)\right)$$

pravdepodobnosť *úspechu* aspoň

úspešnosť

$$1-\left(1-\frac{\binom{\ell(n)}{2}}{\binom{n}{2}}\right)^{\frac{n^2}{\ell^2(n)}} \geq 1-\left(1-\frac{\ell^2(n)}{n^2}\right)^{\frac{n^2}{\ell^2(n)}} = 1-\frac{1}{e}$$

Z hľadiska časovej zložitosti je najlepšia voľba pre funkciu $\ell(n)$ taká[6], keď

$$\frac{n^4}{\ell^2(n)} = n^2\ell(n) \quad\Longrightarrow\quad \ell(n) = \left\lfloor n^{2/3}\right\rfloor$$

Zhrnutím týchto úvah máme

Veta 7.21 *Algoritmus Detran($n^{2/3}$) opakovaný $\frac{n^2}{\ell^2(n)} = n^{4/3}$ krát vypočíta v čase $O\left(\frac{n^4}{\ell^2(n)}\right) = O(n^{8/3})$ minimálny rez s pravdepodobnosťou úspechu aspoň $1-\frac{1}{e}$.*

Získaný algoritmus[7] je lepší ako najlepší známy deterministický. Skúsme vytvoriť ešte lepší algoritmus.

Vráťme sa k základnému algoritmu 47 a všimnime si, ako sa pri postupnosti kontrakcií pravdepodobnosť chyby mení.

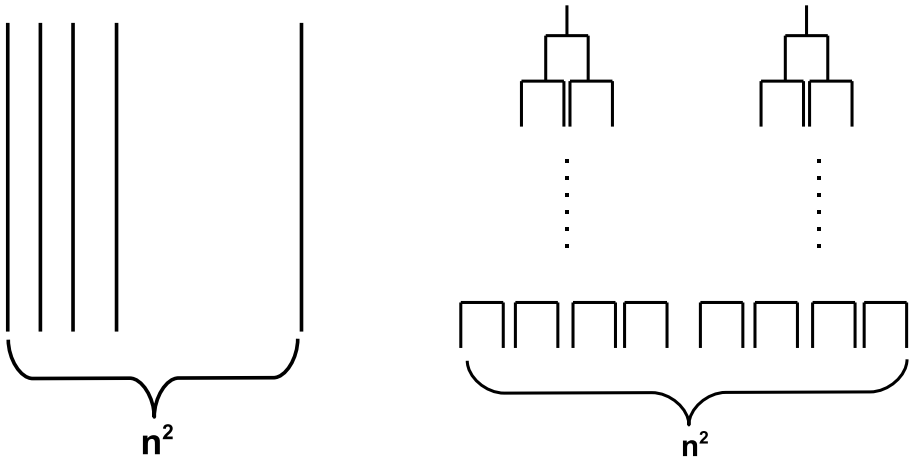
$$\frac{2}{n}, \frac{2}{n-1}, \frac{2}{n-2}, \ldots, \frac{2}{3}$$

Pravdepodobnosť chyby v priebehu opakovania kontrakcií rastie, preto budeme algoritmus častejšie opakovať ku koncu. Predstavme si, že jednotlivé opakované behy

[6] $O(a+b) = O(\max\{a,b\})$

[7] Kde je *random sampling*?

Algoritmu 47 robíme paralelne (obr. 7.1 vľavo). Vtedy si čas možno predstaviť ako plochu. Ak by sme behy vytvárali postupne (obr. 7.1 vpravo)—začneme s dvomi, po nejakom čase výpočtu každý proces pri náhodnej voľbe rozdelíme na dva,..., bude čas odpovedať súčtu dĺžok ciest z koreňa do listov. Aj "opticky" to vyzerá lepšie. Ukážeme, že pri vhodnej voľbe deliacich bodov tomu tak naozaj je.



Obrázok 7.1: Opakovania Algoritmu Contract: vľavo n^2 kompletných behov, vpravo $O(n^2)$ behov, ktoré vznikajú postupným "delením"

Počet behov zdvojnásobíme, keď sa počet vrcholov kontrakciami zredukoval na $1/\sqrt{2}$-tinu. Keďže počet kontrakcií je $n-2$, je počet behov, ktoré realizujeme, $2^{\log_{\sqrt{2}} n-2} = O(n^2)$.

Algoritmus 49 RepTree(G)

1: ak $n \leq 6$ vypočítaj minimálny rez deterministicky
2: $h \leftarrow \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$
3: realizuj paralelne výpočet na dvoch behoch dovtedy, kým kontrakciami nevzniknú grafy G/F_1, G/F_2 veľkosti h
4: RepTree(G/F_1)
5: RepTree(G/F_2)
6: return lepší minimálny rez

Veta 7.22 *Časová zložitosť Algoritmu RepTree je $O(n^2 \log n)$, pravdepodobnosť úspechu aspoň $\frac{1}{\Omega(\log n)}$.*

Dôkaz: Začnime analýzou času.

čas

- Veľkosť multigrafu klesá o (multiplikatívny) faktor $1/\sqrt{2}$, preto je hĺbka vnorenia rekurzie nanajvýš $\log_{\sqrt{2}} n = O(\log n)$.
- Časová zložitosť Algoritmu Contract, keď štartoval na n vrcholoch, je $O(n^2)$. Prechod od m vrcholov k $\left\lceil 1 + \frac{m}{\sqrt{2}} \right\rceil$ teda môžme zhora ohraničiť $O(m^2)$.
- Časovú zložitosť možno vyjadriť rekurentnou nerovnicou

$$T(n) \leq \begin{cases} O(1), & n \leq 6 \\ 2T(\left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil) + O(n^2) & \text{inak} \end{cases}$$

ktorej riešením je $O(n^2 \log n)$

Pripomeňme si, že máme fixovaný minimálny rez C_{min} s cenou $|E(C_{min})| = k$ *úspešnosť*

- označme p_ℓ pravdepodobnosť toho, že graf G/F_i veľkosti $\left\lceil 1+\frac{n}{\sqrt{2}} \right\rceil$ obsahuje C_{min}, ak ho obsahoval graf G veľkosti ℓ pred rozdelením na $G/F_1,\ G/F_2$.

$$p_\ell \geq \frac{\binom{\left\lceil 1+\frac{\ell}{\sqrt{2}} \right\rceil}{2}}{\binom{\ell}{2}} = \frac{\left\lceil 1+\frac{\ell}{\sqrt{2}} \right\rceil \left(\left\lceil 1+\frac{\ell}{\sqrt{2}} \right\rceil - 1 \right)}{\ell(\ell-1)} \geq \frac{(\sqrt{2}+\ell)\ell}{2\ell(\ell-1)} \geq \frac{1}{2}$$

- označme $Pr(n)$ pravdepodobnosť toho, že algoritmus RepTree nájde minimálny rez C_{min}. Potom $p_\ell Pr\left(\left\lceil 1+\frac{\ell}{\sqrt{2}} \right\rceil\right)$ je dolným odhadom na pravdepodobnosť toho, že z G/F vypočítame C_{min} redukciou na G/F_i, ak G/F ho obsahovalo
- pravdepodobnosť toho, že rekurziou *nevypočítame* C_{min} je nanajvýš

$$\left(1 - p_\ell Pr\left(\left\lceil 1+\frac{\ell}{\sqrt{2}} \right\rceil\right)\right)^2$$

- $Pr(2) = 1$

$$Pr(\ell) \geq 1 - \left(1 - p_\ell Pr\left(\left\lceil 1+\frac{\ell}{\sqrt{2}} \right\rceil\right)\right)^2 \geq 1 - \left(1 - 1/2 Pr\left(\left\lceil 1+\frac{\ell}{\sqrt{2}} \right\rceil\right)\right)^2$$

Teraz ukážeme, že riešením je $\Theta(\frac{1}{\log l})$. Nech $k = \Theta(\log l)$ je hĺbka rekurzie. Potom dolný odhad na pravdepodobnosť úspechu $P(k) = Pr(2^k)$ získame riešením rekurentnej rovnice

$$P(k) = \begin{cases} 1, & \text{k=0} \\ P(k-1) - \frac{P(k-1)^2}{4}, & k > 0 \end{cases}$$

Označme $q(k) = 4/P(k) - 1$. Potom $P(k) = \frac{4}{q(k)+1}$

$$\begin{array}{rcl} \frac{4}{q(k+1)+1} & = & \frac{4}{q(k)+1} - \left(\frac{4}{q(k)+1}\right)^2/4 \\ \frac{1}{q(k+1)+1} & = & \frac{q(k)}{(q(k)+1)^2} \end{array}$$

Úpravou dostávame $q(k+1) = q(k) + 1 + \frac{1}{q(k)}$ a následne:

$$\begin{array}{rcl} q(k) & = & q(k+1) + 1 + \frac{1}{q(k-1)} \\ & = & q(k-2) + 1 + \frac{1}{q(k-2)} + 1 + \frac{1}{q(k-1)} \\ & = & q(0) + k + \sum_{i=0}^{k-1} \frac{1}{q(i)} = {}^8 3 + k + \sum_{i=0}^{k-1} \frac{1}{q(i)} \end{array}$$

$$k < q(k) < k + 3 + \sum_{i=0}^{k-1} \frac{1}{i} = k + 3 + H_{k-1}$$

Teda $q(k) = k + \Theta\log k$, $P(k) = \frac{4}{q(k)+1} = \frac{4}{k+\Theta\log k} = \Theta(\frac{1}{k})$ a teda $\boxed{Pr(l) = \Theta(\frac{1}{\log l})}$

□

Na základe analýzy algoritmu RepTree je jasné, že $\log n$ nezávislých opakovaní tohto algoritmu stačí, aby sme s konštantnou pravdepodobnosťou dosiahli optimálne riešenie problému Min-Cut. Pri $O(\log^2 n)$ opakovaniach ide pravdepodobnosť neúspechu rýchlo k nule. Pritom čas v tomto prípade je $O(n^2 \log^3 n)$.

7.3.2 Opakovaná náhodná vzorka a 3-splniteľnosť

V tejto časti ukážeme, ako vhodné skombinovanie viacerých metód

- opakovanie
- náhodná vzorka
- lokálne prehľadávanie

vedie k "rozumnému" riešeniu NPÚ problému 3SAT. Doteraz nie je známy polytime pravdepodobnostný algoritmus na riešenie NPÚ problému s ohraničenou chybou, skôr sa predpokladá, že NP-ťažké sú ťažké aj pre polytime pravdepodobnostné. Čo ale môžme dosiahnuť je rozumný exponenciálny čas.

f(n)	10	50	100	300
n!	$\approx 3.6 \cdot 10^6$	65 digits	158 digits	625 digits
2^n	1024	16 digits	31 digits	91 digits
$2^{n/2}$	32	$\approx 33 \cdot 10^6$	16 digits	46 digits
$(1.2)^n$	$\approx$6.19	9100	$\approx 8.2 \cdot 10^7$	24 digits
$10 \cdot 2^{\sqrt{n}}$	$\approx$30	$\approx$1345	10240	$\approx 1.64 \cdot 10^6$
$n^2 \cdot 2^{\sqrt{n}}$	895	≈ 336158	$1.024 \cdot 10^7$	$\approx 1.48 \cdot 10^{11}$
n^6	10^6	$1.54 \cdot 10^{10}$	10^{12}	$\approx 7.29 \cdot 10^{14}$

Smerujeme k 1MC algoritmu pre 3KNF-splniteľnosť, ktorého časová zložitosť bude $O(|F| \cdot n^{3/2} \cdot (\frac{4}{3})^n)$. Základná idea spočíva v niekoľkonásobnom prehľadaní vhodného okolia náhodne vybraného vektora: cieľom je nájsť vektor, ktorý spĺňa vstupnú formulu.

Algoritmus pracuje v kolách, ich počet je určený konštantou S. Jej voľba bude zrejmá po analýze pravdepodobnosti chyby jedného kola. Počet kôl je volený tak, aby sme sa s pravdepodobnosťou chyby dostali pod konštantu e^{-1}. V jednom kole vygeneruje náhodný vektor hodnôt α. Ak $F(\alpha) \neq 1$, spraví nanajvýš $M = 3n$ náhodných krokov/pokusov v okolí vektora α. Krok spočíva vo vygenerovaní nového testovacieho vektora α, ktorý vznikne flipom jednej z premenných; túto premennú určíme tak, že náhodne zvolíme nesplnenú klauzulu v $F(\alpha)$ a premennú v nej.

Algoritmus 50 Schöning

```
1:  N ← 0                                          ▷ počet prezeraných vzoriek
2:  S ← ⌈2 · √(3πn) (4/3)^n⌉          ▷ odhad počtu v tejto iterácii prezretých vzoriek
3:  Found ← False
4:  while N < S and not Found do
5:      N ← N + 1
6:      náhodne zvoľ α ∈ {0,1}^n
7:      if F(α) = 1 then Found ← True
8:      M ← 0
9:      while M < 3n and not Found do
10:         M ← M + 1
11:         nájdi v F(α) nesplnenú klauzulu C a náhodne zmeň niektorý z bitov
12:         v tejto klauzule, čím vznikne nová hodnota α
13:         if F(α) = 1 then Found ← True
14: if Found = True then return (splniteľná, α)
15: else return (nesplniteľná)
```

Veta 7.23 *Algoritmus Schöning je 1MC pre problém splniteľnosti 3KNF formuly. Jeho časová zložitosť je* $O\left(|F| \cdot n^{3/2} \cdot (4/3)^n\right)$

Dôkaz: Začneme časom.

čas

- pri použití opakovaného umocňovania vypočítame hodnotu S na riadku 2 v čase $O(n^2 \log n)$
- celkovo robíme S krát náhodnú vzorku s lokálnym vyhľadávaním

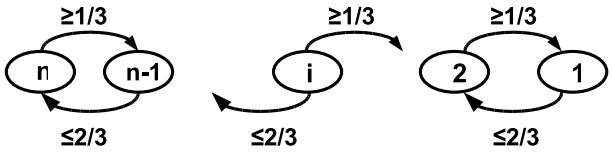
$$O(|F| \cdot \underbrace{n^{1/2} \cdot (4/3)^n}_{S} \cdot \underbrace{n}_{M}) = O(|F| \cdot n^{3/2} \cdot (4/3)^n)$$

korektnosť

Ak vstupná formula nie je splniteľná, algoritmus korektne odpovie, že je nesplniteľná. Chybu môže algoritmus spraviť len vtedy, ak sa mu nepodarí nájsť spĺňajúce priradenie, hoci formula splniteľná je. Spočítame, aká je pravdepodobnosť, že v prípade splniteľnej formuly nájdeme jedno *fixované* priradenie α^*.
Označme

- p pravdepodobnosť toho, že jednou vzorkou a lokálnym prehľadávaním toto priradenie α^* nájdeme(riadky 6-13).
- $Dist(\alpha, \beta)$ - počet bitov, v ktorých sa vektory α, β líšia
- $Class(j) = \{\beta \in \{0,1\}^n \mid Dist(\alpha^*, \beta) = j\}$; zrejme
  $Class(0) = \{\alpha^*\}$
  $|Class(j)| = \binom{n}{j}$

Pozrime sa na lokálne prehľadávanie ako na putovanie medzi jednotlivými triedami (obr.7.2)



Obrázok 7.2: Proces lokálneho prehľadávania možno vnímať ako putovanie po grafe. Cieľom je sa z náhodného vrchola dostať do vrchola 0.

- Pravdepodobnosť toho, že sa z j posunieme v jednom lokálnom kroku/jedným flipom do $(j-1)$ je aspoň $1/3$. Analogicky, presun z j do $(j+1)$ je s pravdepodobnosťou nanajvýš $2/3$.

- Označme $q_{j,i}$ pravdepodobnosť, že keď začneme v triede $Class(j)$, skončíme v α^*, pričom prejdeme *presne* $(j+i)$ krokov smerom ku α^* a i smerom od α^*. Zrejme $i, j \leq n$

- Popis lokálneho prehľadávania možno znázorniť reťazcom $\{+,-\}^{j+2i}$, pričom $j+2i \leq 3n$, $+$ znázorňuje posun smerom ku α^*, znak - zas posun od α^*. Nie každý takýto reťazec je korektným popisom lokálneho prehľadávania. Ten korektný má v každom suffixe aspoň toľko znakov $+$ ako -. Dá sa ukázať, že takýchto reťazcov je

$$\binom{j+2i-1}{i} - \binom{j+2i-1}{i-1} = \binom{j+2i}{i} \frac{1}{j+2i} \tag{7.1}$$

Označme množinu týchto reťazcov $\mathcal{B}$.

- každé $w \in \mathcal{B}$ definuje množinu výpočtov $Event(w)$

$$Pr[Event(w)] \geq \left(\frac{1}{3}\right)^{j+i}\left(\frac{2}{3}\right)^{i}$$

Preto

$$q_{j,i} \geq \underbrace{\frac{1}{j+2i}\binom{j+2i}{i}}_{\#w}\underbrace{\left(\frac{1}{3}\right)^{j+i}\left(\frac{2}{3}\right)^{i}}_{\text{že je to podľa w}}$$

- q_j označuje pravdepodobnosť, že sa z $Class(j)$ dostaneme v rámci 3n krokov lokálneho prehľadávania do α^*

$$q_j \geq \sum_{i=0}^{j} q_{j,i} \quad 0 \leq i \leq j \leq n,\ j+2i \leq 3n$$

$q_o = 1$

$$\begin{array}{lll} q_j & \geq & \sum_{i=0}^{j}\left[\frac{1}{j+2i}\binom{j+2i}{i}\left(\frac{1}{3}\right)^{j+i}\left(\frac{2}{3}\right)^{i}\right] \\ & > & j\cdot\frac{1}{3j}\binom{3j}{j}\left(\frac{1}{3}\right)^{j+i}\left(\frac{2}{3}\right)^{i} = \frac{1}{3}\binom{3j}{j}\left(\frac{1}{3}\right)^{j+i}\left(\frac{2}{3}\right)^{i} \qquad //i=j \text{ je dolný odhad} \\ & = & \frac{1}{3}\cdot\frac{(3j)!}{j!(2j)!}\cdot\left(\frac{1}{3}\right)^{2j}\left(\frac{2}{3}\right)^{j} \approx \frac{1}{3}\frac{\sqrt{2\pi 3j}(3j/e)^{3j}}{\sqrt{2\pi j}(j/e)^{j}\sqrt{2\pi 2j}(2j/e)^{2j}}\left(\frac{1}{3}\right)^{2j}\left(\frac{2}{3}\right)^{j} \qquad //r! \approx \sqrt{2\pi r}\left(\frac{r}{e}\right)^{r} \\ & = & \frac{1}{2\sqrt{3\pi j}}\cdot\left(\frac{1}{2}\right)^{j} \end{array}$$

Na začiatku sme zvolili vektor náhodne, preto pravdepodobnosť p_j, že začneme v $Class(j)$, je

$$p(j) = \frac{\binom{n}{j}}{2^n}$$

A teraz už $p = Pr[\text{Schöning nájde po max 3n krokoch}] \geq \sum_{j=0}^{n} p_j q_j$

$$p > \sum_{j=0}^{n}\left[\left(\frac{1}{2}\right)^{n}\cdot\binom{n}{j}\cdot\frac{1}{2\sqrt{3\pi j}}\cdot\left(\frac{1}{2}\right)^{j}\right]\cdot 1^{n-j} = \sum_{j=0}^{n}\frac{1}{2\sqrt{3\pi j}}\left(\frac{1}{2}\right)^{n}\left(1+\frac{1}{2}\right)^{n} \geq$$

$$\geq \frac{1}{2\sqrt{3\pi n}}\left(\frac{1}{2}\right)^{n}\left(1+\frac{1}{2}\right)^{n} = \frac{1}{2\sqrt{3\pi n}}\left(\frac{3}{4}\right)^{n} = \widetilde{p}$$

Chyba jedného kola je nanajvýš $(1-\widetilde{p})$, preto pri $t = 1/\widetilde{p}$ opakovaniach bude pravdepodobnosť chyby nanajvýš

$$(1-\widetilde{p})^t \leq e^{-\widetilde{p}t} = e^{-1}$$

Ak zrealizujeme $10t$ kôl, bude $Erorr(F) \leq e^{-10} \leq 10^{-5}$. Teraz by už malo byť zrejmé, prečo sme volili

$$S = \lceil 2\cdot\sqrt{3\pi n}\left(\frac{4}{3}\right)^{n}\rceil = 10/\widetilde{p}$$

pár slov k (7.1)

Ostáva ešte ukázať platnosť (7.1). Nech $W(i,j)$ označuje počet "korektných" reťazcov dĺžky $i+2j$, v ktorých je i núl(smerom od α^*) a $j+i$ jednotiek(smerom ku α^*). Indukciou argumentujeme, že

$$W(i,j) = \binom{j+2i}{i}\frac{j}{j+2i}$$

Pre $i = 0$ je $W(0, j) = 1$, pre $n = 1, 2$ je $i = 0$ a pre $n = 3$ je alebo $i = j = 0$ alebo $i = 0, j = 3$. ✓ $n \leq 3$

Pre $r + 2l < n$ IP platí. Pre $i + 2j = n$ rozlíšime situáciu podľa prvého symbolu $n > 3$

$$\text{prvý symbol je } \begin{cases} 1, & W(i, j-1) = \binom{j-1+2i}{i} \frac{j-1}{j-1+2i} \\ 0, & W(i-1, j+1) = \binom{j-1+2i}{i-1} \frac{j+1}{j-1+2i} \end{cases}$$

a dopočítame zvlášť pre $j = 1,\ j > 1$

□

7.4 Metóda svedkov

Pre použitie metódy svedkov je kľúčovým momentom dostatok svedkov. To úzko súvisí aj s odpoveďou na otázku, či pravdepodobnostné algoritmy môžu byť efektívnejšie ako deterministické. Ak máme efektívny pravdepodobnostný algoritmus založený na metóde svedkov a svedkov vieme efektívne generovať, máme vlastne aj efektívny deterministický algoritmus. Ak sú však svedkovia rozmiestnení náhodne,...

Pár slov na úvod

- Pre použitie metódy svedkov potrebujeme *dostatok svedkov*, čím väčšinou rozumieme to, že pomer kandidátov a svedkov je konštanta.
- Samotné aplikovanie metódy potom spočíva v náhodnom výbere kandidáta a overení, či je alebo nie je tento kandidát svedkom. Ak je v množine kandidátov dostatočne veľa svedkov, je pravdepodobnosť toho, že vybratý kandidát je svedok, dostatočná.

V tejto súvislosti sa budeme venovať prvočíselnosti, a to konkrétne
– testovaniu prvočíselnosti
– generovaniu náhodných prvočísel

Začneme opakovaním znenia [9] niektorých známych viet, ktoré budeme v argumentácii využívať.

Veta 7.24 (Malá Fermátová) $p \in PRIM$, $a \in Z_p^* = \{d \in Z_p | gcd(a,p) = 1\}$. *Potom*

$$a^{(p-1)} \mod p \equiv 1$$

Veta 7.25 (O čínskych zvyškoch, II. verzia) *Nech* $n = p \times q, gcd(p,q) = 1$. *Nech* $\odot_{p,q}, \oplus_{p,q}$ *sú operácie nad* $Z_p \times Z_q$.

$$\begin{aligned}(a_1,a_2) \oplus_{p,q} (b_1,b_2) &= ((a_1 \oplus_{p,q} b_1) \mod p, (a_2 \oplus_{p,q} b_2) \mod q)\\ (a_1,a_2) \odot_{p,q} (b_1,b_2) &= ((a_1 \odot_{p,q} b_1) \mod p, (a_2 \odot_{p,q} b_2) \mod q)\end{aligned}$$

Potom $(Z_n, \oplus_{\mod p}, \odot_{\mod q})$ *a* $(Z_p \times Z_q, \oplus_{p,q}, \odot_{p,q})$ *sú izomorfné.*

Veta 7.26 (O čínskych zvyškoch, I. verzia) *Nech* $m = m_1 \times m_2 \times \ldots \times m_k$, *kde* $k \in N - \{0\}, gcd(m_i, m_j) = 1$ *pre* $i \neq j$. *Potom pre každú postupnosť* $r_1 \in Z_{m_1}, \ldots, r_k \in Z_{m_k}$ *existuje* jediné *číslo* $r \in Z_m : r \equiv r_i (\mod m_1)$

Veta 7.27 (Lagrange) *Pre každú podgrupu* $(H, \circ)$ *konečnej grupy* $(A, \circ)$ *platí*

$$|A| = Index_H(A) \cdot |H|$$

Tu
- $H \circ b = \{h \circ b | h \in H\}$
- $Index_H(A) = |\{H \circ b | b \in H\}|$

7.4.1 Hľadanie svedkov pre test prvočíselnosti

V tejto časti sa budeme venovať testu prvočíselnosti: pre vstupné n chceme vedieť, či je to prvočíslo alebo zložené číslo. Zaujíma nás pravdepodobnostný algoritmus, ktorého časová zložitosť je $O((\log n)^c)$ pre nízku konštantu c. Veľkosť konštanty c je dôležitá, kedže pre potreby kódovania sú prvočísla dlhé stovky cifier.

Jednotlivé algoritmy vychádzajú z vhodnej definície *svedka*.

naivný prístup

Za **naivný prístup** možno považovať test, ktorý vychádza zo základnej definície prvočísla ako čísla, ktoré je deliteľné iba jednotkou a samým sebou. Algoritmus je jednoduchý—testujeme postupne deliteľnosť n kandidátom a od 1 po $n-1$, resp. $\sqrt{n}$. Zložitosť je $O(2^{(\log n}/2)$, čo je vzhľadom na veľkosť vstupu $\log n$ exponenciálne.

požiadavky na svedka

i. prvok a je svedkom pre zložené číslo, ak poskytuje efektívne overenie tohto faktu
ii. ľahko rozlíšime, či kandidát je svedok elebo nie
iii. množina kandidátov obsahuje dostatočne veľa svedkov

deliteľ

Pre identifikáciu zloženého čísla vieme triviálne využiť deliteľnosť: prvok a je svedok, že $n \notin \mathsf{PRIM}^{10} \Leftrightarrow a$ delí n
Takáto definícia spĺňa vlastnosti (i,ii), problém však môže byť s vlastnosťou (iii), ktorá nie je vždy splnená—v prípade $n = pq$, kde p, q sú prvočísla, je pravdepodobnosť, že náhodne zvolené číslo a je svedkom pre n iba $2/(n-2)$.

Fermátova veta

Na základe malej Fermátovej vety máme nasledujúce kritérium — číslo a je svedok[11] pre $n \notin \mathsf{PRIM} \Leftrightarrow a^{\frac{n-1}{2}} \mod n \neq 1$.
(ii) Efektívny výpočet $a^{p-1} \mod p$ je založený na iterovanom umocňovaní

- $a^2 \mod p = a \cdot a \mod p$
  $a^{2^k} \mod p = [(a^{2^{k-1}} \mod p) \cdot (a^{2^{k-1}} \mod p)] \mod p$
- nech $b = \sum_{i=1}^{k} b_i 2^{i-1}$; potom

$$a^b = a^{b_1 2^0} \cdot a^{b_2 2^1} \cdot \ldots \cdot a^{b_k^{k-1}}$$

$a^b \mod p$ vypočítame:
– $a_i = a^{2^{i-1}} \mod p$
– $\prod_{i=1,b_i=1}^{k} a_i \mod p$

Výpočet realizujeme $O(\log n)$ násobeniami nad Z_p, čo znamená $O((\log n)^3)$ bitových operácií. Takýto výpočet je teda efektívny, existuje však nekonečná množina tzv. Carmichaelových čísel, pre ktoré je tento test nevhodný. *Carmichael číslo* je také, že $\forall a \in \{1, \ldots, n-1\}$ platí $a^{(n-1)} \mod n = 1$, ale n NIE JE prvočíslo.

Ďalší test vychádza z novej "definície" prvočísla

alternatívna definícia

Veta 7.28 *Nech $p > 2$ je nepárne. Potom*

$$p \text{ je prvočíslo } \Leftrightarrow a^{\frac{(p-1)}{2}} \mod p \in \{1, p-1\} \quad \forall a \in Z_p - \{0\}$$

⇒

Dôkaz: Nech p je nepárne prvočíslo, $p = 2p' + 1$. Z Malej Fermatovej vety dostávame

$$\left.\begin{array}{l} a^{p-1} \equiv 1(\mod p) \forall a \in Z_p - \{0\} \\ a^{p-1} = a^{2p'} = (a^{p'} - 1)(a^{p'} + 1) + 1 \end{array}\right\} (a^{p'} - 1)(a^{p'} + 1) \equiv 0(\mod p)$$

Preto $a^{\frac{(p-1)}{2}} \mod p \in \{1, p-1\}$

⇐

Nech $a^{\frac{(p-1)}{2}} \in \{1, p-1\} \forall a \in Z_p - \{0\}$. Kvôli sporu predpokladajme, že $p = ab$.

$$\left.\begin{array}{ll} a^{\frac{p-1}{2}} \mod p & \in \{1, p-1\} \\ a^{\frac{p-1}{2}} \mod p & \in \{1, p-1\} \end{array}\right\} (ab)^{\frac{p-1}{2}} \mod p = a^{\frac{p-1}{2}} \cdot b^{\frac{p-1}{2}} \mod p \in \{1, -1\}$$

Keďže $ab = p$,

$0 = p \mod p = p^{\frac{p-1}{2}} \mod p = (ab)^{\frac{p-1}{2}} \mod p \in \{1, -1\}$ SPOR □

MVF

Na základe tejto vety môžme definovať nasledujúce kritérium pre svedka. Nech $n \geq 3$ je nepárne. Číslo $a \in \{1, \ldots, n-1\}$ je svedok, že $n \notin \mathsf{PRIM} \Leftrightarrow a^{\frac{n-1}{2}} \notin \{1, n-1\}$. Kritérium je vhodné pre nepárne n také, že $n \equiv 3(\mod 4)$

Veta 7.29 *Pre každé* $n, n \equiv 3(\mod 4)$*, platí*
(a.) ak n *je prvočíslo, potom* $a^{(n-1)/2} \mod n \in \{1, n-1\} \quad \forall a \in \{1, \ldots, n-1\}$
(b.) ak n *je zložené, potom* $a^{(n-1)/2} \mod n \notin \{1, n-1\}$ *pre aspoň polovicu prvkov z* $\underbrace{\{1, \ldots, n-1\}}_{A(n-1)}$

Dôkaz: Potrebujeme argumentovať (b). Vezmime teda *zložené* $n, n \equiv 3(\mod 4)$. Rozdeľme $A(n-1)$ na množinu svedkov a nesvedkov:

$Wit_n = \{a \in A(n-1) \mid a^{(n-1)/2} \mod n \notin \{1, n-1\}\}$
$Euler = \{a \in A(n-1) \mid a^{(n-1)/2} \mod n \in \{1, n-1\}\}$

Ukážeme, že existuje *injektívne* zobrazenie z Euler do Wit $_n$, čo implikuje $|\mathsf{Euler}| \leq |\mathsf{Wit}_n|$.

definícia h_b

Predpokladajme, že máme $b \in \mathsf{Wit}_n$, pričom b^{-1} existuje (určíme neskôr). Pomocou b definujme spomínané zobrazenie

$$h_b(a) = ab \mod n$$

Ukážeme, že h_b je zobrazenie $\mathsf{Euler} \longrightarrow \mathsf{Wit}_n$, je injektívne a napokon určíme b a b^{-1}.

$h_b(a) \in$ Wit$_n$

Nech $a \in \mathsf{Euler}$, $h_b(a) = ab \mod n$. Potom
$(a \cdot b)^{(n-1)/2} \mod n = (a^{(n-1)/2} \mod n) \cdot (a^{(n-1)/2} \mod n) = \pm b^{(n-1)/2} \mod n$
Keďže $b \in \mathsf{Wit}_n$, $b^{(n-1)/2} \mod n \notin \{1, n-1\}$ a teda $h_b(a) \in \mathsf{Wit}_n$.

injekcia

Nech $a_1, a_2 \in \mathsf{Euler}$, $a_1 \neq a_2$ a nech $h_b(a_1) = h_b(a_2)$, resp. $a_1 b \equiv a_2 b \ (\mod n)$. Potom

$$a_1 \equiv a_1 b b^{-1} \mod n \equiv a_2 b b^{-1} (\mod n) = a_2$$

existencia b, b^{-1}

Využime, že n je zložené[12], $n = pq, gcd(p, q) = 1$. Potom $(a \mod p, a \mod q)$ jednoznačne reprezentuje $a \in \{1, \ldots, n-1\}$. Číslo a určíme jeho reprezentáciou. Kvôli tomu si všimnime, čo platí o reprezentácii čísel z Euler. Podľa definície pre $a \in \mathsf{Euler}$ je $a^{(n-1)/2} \mod pq \in \{1, n-1\}$

- ak $a^{(n-1)/2} = kpx + 1$, potom $a^{(n-1)/2} \mod p \equiv a^{(n-1)/2} \mod q \equiv 1$; reprezentácia $a^{(n-1)/2}$ je v tomto prípade $(1, 1)$
- ak $a^{(n-1)/2} = kpx + n - 1$, potom
  $a^{(n-1)/2} \mod p = (n-1) \mod p = (pq-1) \mod p = p - 1$
  $a^{(n-1)/2} \mod q = (n-1) \mod q = (pq-1) \mod q = q - 1$

V tomto prípade je $a^{(n-1)/2}$ reprezentované $(p-1, q-1)$

$b \in Wit_n$

Hodnotu b určíme reprezentáciou $(1, q-1)$. Overme, že $b \in Wit_n$:
$(b^{(n-1)/2} \mod p, b^{(n-1)/2} \mod q) = (1^{(n-1)/2} \mod p, (-1)^{(n-1)/2} \mod q) = (1, -1)$.
Číslo $b \notin \mathsf{Euler}$, preto $b \in Wit_n$.

$b^{-1} = b$

Inverzný prvok k b je b:
$b \cdot b = (1, q-1) \odot (1, q-1) = (1 \mod p, (q-1)(q-1) \mod q) = (1, 1)$ □

Na základe predchádzajúcich úvah dostávame, že pre zložené n je pravdepodobnosť toho, že výstupom algoritmu SSSA je korektná odpoveď $n \notin \mathsf{PRIM}$, aspoň polovica.

[9] V tejto prednáške sú vety nástrojom, dôkazy preto vynechávame.
[10] PRIM označuje množinu prvočísel
[12] Doriešte pre $n = p^i$

Algoritmus 51 SSSA - zjednodušený Solovay-Strassen

1: zvoľ náhodne $a \in \{1, \ldots, n-1\}$
2: $A \leftarrow a^{(n-1)/2} \mod n$
3: **if** $A \in \{1, -1\}$ **then** return($n \in$ PRIM)
4: **else** return($n \notin$ PRIM)

Veta 7.30 *Pre $n = 4k + 3$ je algoritmus SSSA polytime 1MC pre zložené čísla.*

7.4.2 Algoritmus Solovay-Strassen pre testovanie prvočíselnosti

V tejto časti sa zemeriame na odstránenie podmienky $n \equiv 3(\mod 4)$. Pridáme ďalšie kritérium, ktoré pomáha identifikovať zložené čísla – $gcd \neq 1$.

Definícia 7.3 *$a \in \{1, \ldots, n-1\}$ je svedok, že nepárne $n \notin$ PRIM, ak*
– *$gcd(a, n) > 1$ alebo*
– *$gcd(a, n) = 1$ a $a^{(n-1)/2} \mod n \notin \{1, -1\}$*

Kritérium vieme efektívne overiť, keďže $gcd(a, n)$ vieme efektívne počítať pomocou Euclidovho algoritmu.

Algoritmus 52 Euclid(a,b)

1: **if** $b = 0$ **then** return(a)
2: **else** return(Euclid(b, a $\mod$ b))

Rekurzívne volanie znižuje aspoň jeden z parametrov na polovicu, preto je hĺbka rekurzie $O(\log b)$, pri "školskom" algoritme delenia je celková zložitosť $O((\log a + b)^3)$.
Hoci je podmienka z definície 7.3 vhodná pre $n \neq 3(\mod 4)$, pre Carmichaelove čísla vhodná nie je. Platí ale

Fakt 7.31 *Nech n je zložené, nie Carmichaelovo. Potom aspoň polovica zo $Z_n - \{0\}$ potvrdzuje $n \notin$ PRIM podľa podmienky z definície 7.3*

Pokračujeme vo vylepšovaní definície svedka.

Definícia 7.4 *Nech $p > 2$ je prvočíslo, a kladné, $gcd(a, p) = 1$. Legendrov symbol pre a a p je*

$$Leg\left[\frac{a}{p}\right] \begin{cases} 1, & a \text{ je kvadratický zvyšok modulo } p \\ -1, & a \text{ je kvadratický nezvyšok modulo } p \end{cases}$$

Pre prvočíslo p a a s $gcd(a, p) = 1$ platí

$$Leg\left[\frac{a}{p}\right] = a^{(p-1)/2} \mod p$$

Na základe tohto faktu vieme $Leg\left[\frac{a}{p}\right]$ efektívne počítať.

Definícia 7.5 *Nech $n = p_1^{k_1} p_2^{k_2} \ldots p_\ell^{k_\ell}$ je faktorizácia nepárneho $n \geq 3$, pričom k_j sú kladné. Ak $gcd(a, n) = 1$, potom Jacobiho symbol pre a, n je*

$$Jac\left[\frac{a}{n}\right] = \prod_{i=1}^{\ell} \left(Leg\left[\frac{a}{p_i}\right]\right)^{k_i} = \prod_{i-1}^{\ell} \left(a^{(p_i-1)/2} \mod p_i\right)^{k_i}$$

Fakt 7.32 *Pre každé a, n spĺňajúce podmienku definície Jacobiána*

$$Jac\left[\frac{a}{n}\right] \in \{-1, 1\}$$

Jacobián chceme využiť na definíciu svedka, faktorizáciu ale nemáme. Na základe vlastností z nasledujúcej lemy 7.33 ho budeme vedieť efektívne vypočítať.

Lema 7.33 *Majme nepárne $n \geq 3$, $a, b : gcd(a, n) = gcd(b, n) = 1$. Platí*

1. $Jac\left[\frac{ab}{n}\right] = Jac\left[\frac{a}{n}\right] \cdot Jac\left[\frac{b}{n}\right]$
2. $Jac\left[\frac{a}{n}\right] = Jac\left[\frac{b}{n}\right]$ *pre* $a \equiv b(\mod n)$
3. $Jac\left[\frac{a}{n}\right] = -1^{\frac{(a-1)}{2}\frac{(n-1)}{2}} Jac\left[\frac{n}{a}\right]$ *pre n nepárne*
4. $Jac\left[\frac{1}{n}\right] = 1, \quad Jac\left[\frac{n-1}{n}\right] = (-1)^{(n-1)/2}$
5. $Jac\left[\frac{2}{n}\right] = \begin{cases} -1, & \text{pre } n \mod 8 \in \{3,5\} \\ 1, & \text{pre } n \mod n \in \{1,7\} \end{cases}$

Dôkaz: Ukážeme prvé dve vlastnostnosti, ostatné ako cvičenie.

1

$$\begin{aligned} Jac\left[\frac{ab}{n}\right] &= \prod_{i=1}^{\ell} \left((ab)^{(p_i-1)/2} \mod p_i\right)^{k_i} = \\ &= \prod_{i=1}^{\ell} \left(a^{(p_i-1)/2} \mod p_i\right)^{k_i} \left(b^{(p_i-1)/2} \mod p_i\right)^{k_i} = \\ &= \prod_{i=1}^{\ell} \left(a^{(p_i-1)/2} \mod p_i\right)^{k_i} \prod_{i=1}^{\ell} \left(b^{(p_i-1)/2} \mod p_i\right)^{k_i} \\ &= Jac\left[\frac{a}{n}\right] Jac\left[\frac{b}{n}\right] \end{aligned}$$

2

Stačí, ak ukážeme, že

$(a \equiv b(\mod p)) \& (gcd(a, b) = 1) \quad \Rightarrow Leg\left[\frac{a}{p}\right] = Leg\left[\frac{b}{p}\right]$

$$\begin{aligned} a \equiv b \mod p \quad &\Rightarrow \quad a = pr + z \\ &\qquad\quad b = ps + z \\ Leg\left[\frac{a}{p}\right] &= a^{(p-1)/2} \mod p = (pr+z)^{(p-1)/2} \mod p = \\ &= \sum_{i=0}^{(p-1)/2} \binom{(p-1)/2}{i} (pr)^{\frac{p-1}{2}-i} z^i \mod p = z^{(p-1)/2} \mod p \end{aligned}$$

Analogicky $Leg\left[\frac{b}{p}\right] = z^{(p-1)/2} \mod p$ □

A teraz už spomínaný algoritmus na výpočet Jacobiána. Každé rekurzívne volanie

Algoritmus 53 Jacobian(a,n)

▷ vstupom sú nepárne $n \geq 3$ a a, pričom $gcd(a, n) = 1$

case

$a = 1$	:	$Jac\left[\frac{1}{n}\right] \leftarrow 1$
$a = 2 \& n \mod 8 \in \{1,7\}$	:	$Jac\left[\frac{2}{n}\right] \leftarrow 1$
$a = 2 \& n \mod 8 \in \{3,5\}$	:	$Jac\left[\frac{2}{n}\right] \leftarrow -1$
a nepárne	:	$Jac\left[\frac{a}{n}\right] \leftarrow Jac\left[\frac{2}{n}\right] Jac\left[\frac{a/2}{n}\right]$
$a > n$	:	$Jac\left[\frac{a}{n}\right] \leftarrow Jac\left[\frac{a \mod n}{n}\right]$
inak	:	$Jac\left[\frac{a}{n}\right] \leftarrow (-1)^{\frac{a-1}{2}\frac{n-1}{2}} Jac\left[\frac{n \mod a}{a}\right]$

zníži aspoň jeden z parametrov aspoň na polovicu, Preto je hĺbka rekurzie $O(\log n)$. Manipulácia s parametrami je $O(1)$ aritmetických, resp. $O((\log a + n)^2)$ bitových operácií. Čas výpočtu $Jac\left[\frac{a}{n}\right]$ je $O((\log(a+n))^3)$ binárnych operácií.

A teraz sme už pripravení definovať novú charakterizáciu svedka.

Definícia 7.6 *Nech $n \geq 3$ je nepárne. Hovoríme, že $a \in \{1, \ldots, n-1\}$ je* Jac-svedok *pre $n \notin PRIM$, ak* *Jac-svedok*

- $gcd(a, n) \neq 1$, *alebo*
- $gcd(a, n) = 1$ *a* $Jac\left[\frac{a}{n}\right] \neq a^{(n-1)/2} \mod n$

O korektnosti pravdepodobnostného algoritmu založeného na Jac-svedkoch hovorí nasledujúca veta.

Veta 7.34 *Pre každé nepárne $n \geq 3$ platí*

1. *ak n je* prvočíslo, *potom* $Jac\left[\frac{a}{n}\right] = Leg\left[\frac{a}{n}\right] = a^{(n-1)/2} \mod n$ *pre všetky* $a \in \{1, 2, \ldots, n-1\}$
2. *ak n je* zložené číslo, *potom* $Jac\left[\frac{a}{n}\right] \neq a^{(n-1)/2} \mod n$ *pre aspoň polovicu z tých* $a \in \{1, 2, \ldots, n-1\}$, *pre ktoré* $gcd(a, n) = 1$

Dôkaz: Dokazovať potrebujeme vlastnosť 2. Majme teda $n \geq 3$ nepárne. Množina kandidátov je $Z_n - \{0\}$. Opäť uvažujme dve množiny

$$\begin{array}{rcl} \overline{\mathsf{Wit}_n} & = & \{a \in Z_n^* \mid Jac\left[\frac{a}{n}\right] = a^{(n-1)/2} \mod n\} \\ \mathsf{Wit}_n & = & Z_n^* - \overline{\mathsf{Wit}_n} \end{array}$$

Ukážeme, že $|\overline{\mathsf{Wit}_n}| \leq |Z_n^*|/2$, z čoho potom dostaneme $|\{1, 2, \ldots, n-1\} - \overline{\mathsf{Wit}_n}| \geq |\overline{\mathsf{Wit}_n}|$. Pri argumentácii si pomôže vlastnosťami grúp a podgrúp. Začneme tým, že ukážeme, že $\left(\overline{\mathsf{Wit}_n}, \odot_{modn}\right)$ je vlastná podgrupa $(Z_n^*, \odot_{modn})$—$\overline{\mathsf{Wit}_n}$ je grupa a existuje prvok zo $Z_n - \overline{\mathsf{Wit}_n}$.

grupa

Nech $a, b \in \overline{\mathsf{Wit}_n}$. Potom

$$\left.\begin{array}{rcl} Jac\left[\frac{ab}{n}\right] & = & Jac\left[\frac{a}{n}\right] Jac\left[\frac{b}{n}\right] = \left(a^{\frac{n-1}{2}} \mod n\right)\left(b^{\frac{n-1}{2}} \mod n\right) \\ & = & (ab)^{\frac{n-1}{2}} \mod n \end{array}\right\} \Rightarrow ab \in \overline{\mathsf{Wit}_n}$$

vlastná podgrupa

Potrebujeme ukázať existenciu prvku a zo $Z_n - \overline{\mathsf{Wit}_n}$. Nech $n = \underbrace{p_1^{i_1}}_{q} \underbrace{p_2^{i_2} \ldots p_k^{i_k}}_{m}$, ≥ 1, je faktorizácia n. Prvok a nebudme hľadať priamo v Z_n^*, ale v $Z_q \times Z_m$. Nech g je generátor cyklickej grupy $(Z_q^*, \odot_{modq})$. Prvok a zvolíme tak, že

$$\left.\begin{array}{l} a \equiv g (mod\ \ q) \\ a \equiv 1 (mod\ m) \end{array}\right\} a \stackrel{def}{=} (g, 1) \in Z_q \times Z_m$$

Ak $m = 1$, ako a vezmeme g.

$a \in Z_n^*$

Fakt, že $gcd(a, n) = 1 \Leftrightarrow$ žiadne z prvočísel $p_1, \ldots, p_k$ nedelí a.

Ak by p_1 delilo a, tak $a = 0 \mod p_1$ súčasne s $g \equiv a(\mod p_1)$ je v spore s tým, že g je generátor.

Ak p_r delí a, potom

$$\left.\begin{array}{r} a = p_r b) \\ a = mx + 1 \end{array}\right\} p_r b = mx + 1 = p_r \left(\frac{m}{p_r}\right) x + 1$$

Dostali sme, že p_r delí 1, pričom $p_r > 1$.

$a \notin \overline{Wit_n}$

Dôkaz tejto časti rozdelíme na dve časti podľa hodnoty i_1 $(1, \geq 2)$. Vypočítame $Jac\left[\frac{a}{n}\right]$, $a^{(n-1)/2} \mod n$ a porovnáme.

$i_1 = 1, n = p_1 m, \gcd(p_1, m) = 1$

$$\begin{aligned} Jac\left[\tfrac{a}{n}\right] &= Jac\left[\tfrac{a}{p_1}\right] \prod_{j=2}^{k} \left(Jac\left[\tfrac{a}{p_j}\right]\right)^{i_j} \\ &\overset{a\equiv 1 \bmod m}{=} Jac\left[\tfrac{a}{p_1}\right] \prod_{j=2}^{k} \left(\underbrace{Jac\left[\frac{1}{p_j}\right]}_{=1}\right)^{i_j} \\ &= Jac\left[\tfrac{a}{p_1}\right] = Jac\left[\tfrac{g}{p_1}\right] = Leg\left[\tfrac{g}{p_1}\right] \\ &= -1 \qquad \text{//generátor nemôže byť kvadratický zvyšok} \end{aligned}$$

Teraz hodnota $a^{(n-1)/2} \mod n, n = p_1 m$

$$a^{(n-1)/2} \mod m = (a \mod m)^{(n-1)/2} \mod m = 1^{(n-1)/2} \mod m = 1$$

Preto $a^{(n-1)/2} \mod n = 1$ a $\boxed{-1 = Jac\left[\tfrac{a}{n}\right] \neq a^{(n-1)/2} \mod n \text{ pre } i_1 = 1}$

$i_1 \geq 2$
Nech by $a \in \overline{\mathsf{Wit}_n}$, čo by znamenalo $a^{(n-1)/2} \mod n = Jac\left[\tfrac{a}{n}\right] \in \{1, -1\}$. Preto $a^{(n-1)} \equiv 1(\mod n)$. Keďže $g \equiv a(\mod q)$, $n = p_1^{i_1} m = qm$

$$1 = a^{n-1} \mod n = a^{n-1} \mod q = (a \mod q)^{n-1} \mod q = g^{n-1} \mod q$$

g je generátor cyklickej grupy, jeho rád je $|Z_q^*|$, preto $|Z_q^*|$ delí $n-1$

$Z_q^* = \{x \in Z_q | p_1 \text{ nedelí } x\}$.

$$|Z_q^*| = |Z_q| - \underbrace{\frac{|Z_q|}{p_1}}_{\text{deliteľné } p_1} = p_1(p_1^{i_1-1} - p_1^{i_2-1})$$

Máme teda
p_1 delí $|Z_q^*|$, $|Z_q^*|$ delí $n-1$
p_1 delí $n = p_1^{i_1} m$
$\Big\} \Longrightarrow p_1$ delí $n, n-1$ □

Algoritmus 54 SSA- Solovay-Strassen

▷ vstupom je nepárne $n \geq 3$

1: vygeneruj náhodne $a \in \{1, 2, \ldots, n-1\}$
2: vypočítaj $gcd(a, n)$ ▷ $O((\log n)^3)$
3: **if** $gcd(a, n) \neq 1$ **then** return$(n \notin \mathsf{PRIM})$
4: $J \leftarrow Jac\left[\tfrac{a}{n}\right]$; $A \leftarrow a^{(n-1)/2} \mod n$ ▷ $O((\log n)^3)$
5: **if** $J = A$ **then** return$(n \in \mathsf{PRIM})$
6: **else**return$(n \notin \mathsf{PRIM})$

Veta 7.35 *Algoritmus Solovay-Strassen je polytime 1MC algoritmus pre rozpoznávanie zložených čísel.*

Dôkaz: Korektnosť dôkazu vyplýva z predchádzajúcich úvah.
Máme $a < n$, teda na reprezentáciu a, n stačí $\lfloor \log n \rfloor + 1$ bitov. *čas*

2 na výpočet *gcd* použijeme Euclidov algoritmus, ktorý zbehne v čase $O((\log n)^3)$ (bitové operácie)

4 na výpočet hodnôt J, A stačí $O((\log n)^3)$ bitových operácií

5 porovnanie realizujeme v $O(\log n)$

úspech

$n \in$ PRIM nenájdeme svedka, ktorý by potvrdil, že n je zložené

$n \notin$ PRIM pravdepodobnosť, že svedka nájdeme, je podľa vety 7.34 aspoň 1/2.

□

7.4.3 Generovanie náhodných prvočísel

V tejto časti sa budeme zaoberať generovaním prvočísel; našou úlohu je pre danú dĺžkuℓ vygenerovať náhodné prvočíslo dĺžky ℓ. Stratégia je zrejmá — vygenerujeme náhodné číslo n príslušnej dĺžky a overíme, či je prvočíslo. Pre zvýšenie pravdepodobnosti úspechu spravíme niekoľko ($2\ell^2$) generovaní n a niekoľko (k) testov prvočíselnosti pre príslušné n. Na testovanie prvočíselnosti používame algoritmus Solovay-Strassen.

Algoritmus 55 PrimGen(ℓ, k)

▷ vstupom je dĺžka prvočísla *ell* a počet opakovaní testov k

1: $Founde \leftarrow False$, $I \leftarrow 0$
2: **while** $notFound$ and $I < 2\ell^2$ **do**
3: vygeneruj náhodnú postupnosť $a_1, a_2, \ldots, a_{\ell-2} bitov$
4: $n \leftarrow 2^{\ell-1} + \sum_{i=1}^{\ell-2} a_i 2^i + 1$
5: sprav k nezávislých testov Solovay-Strassenovým algoritmom
6: **if** niektorý dokázal $n \notin$ PRIM **then** $I \leftarrow I + 1$
7: **else**$Found \leftarrow True$, return(n)
8: **if** $I = 2\ell^2$ **then** return(nenašiel)

Veta 7.36 *Algoritmus PrimGen je algoritmus s ohraničenou chybou, ktorý v čase polynomiálnom od dĺžky (výstupu) ℓ vygeneruje prvočíslo.*

Dôkaz: Keďže v prípade úspechu je dĺžka výstupu exponenciálna od dĺžky vstupu, meriame zložitosť algoritmu vzhľadom na veľkosť výstupu.

čas

$2\ell^2$ opakovaní while-cyklu, k opakovaní SSA testu, ktorý trvá $O(\ell^3)$ dáva celkovú zložitosť bitových operácií $O(\ell^5 k)$, resp. pre $k = \ell$ $O(\ell^6)$.

úspešnosť

Algoritmus je neúspešný z dvoch dôvodov

I ak sa mu nepodarilo vygenerovať prvočíslo (resp. číslo, o ktorom by dokázal, že je zložené)

II na výstup dal ako prvočíslo číslo, ktoré je v skutočnosti zložené

I

Aby sme na výstup nedali žiadne číslo, musel každý test, ktorý sme robili, dopadnúť "zle": $n \notin$ PRIM

- pravdepodobnosť, že náhodne zvolené číslo n dĺžky ℓ

  JE prvočíslo, je aspoň $\frac{1}{\ln n} > \frac{1}{2\ell}$

  NIE JE prvočíslo je nanajvýš $(1 - \frac{1}{2\ell})$

- pravdepodobnosť, že ℓ behov Solovay-Strassenovho algoritmu uspeje v dôkaze, že zložené $n \notin \mathsf{PRIM}$ je $w_\ell \geq 1 - 1/2^\ell$
- $Prob[PrimGen(\ell,\ell) = \text{"nenašiel"}] < \left(\left(1 - \frac{1}{2\ell}\right) w_\ell\right)^{2\ell^2} < \left(1 - \frac{1}{2\ell}\right)^{2\ell^2} < e^{-\ell}$

Pre $\ell \geq 100$ je $e^{-\ell} \ll 10^{-40}$

Označme p_i pravdepodobnosť toho, že v i-tom behu, $i \in 1, \ldots, 2\ell^2$, dáme na výstup ako prvočíslo číslo, ktoré je v skutočnosti zložené. To znamená, že $(i-1)$ predchádzajúcich behov vygenerovalo zložené číslo a úspešne to aj overilo, ale v poslednom behu vygenerované číslo sa nepodarilo dokázať. II

- $p_1 < \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell}$
- $p_i \leq \left[\left(1 - \frac{1}{2\ell}\right) w_\ell\right]^{i-1} \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} = \left(1 - \frac{1}{2\ell}\right)^i w_\ell \frac{1}{2\ell}$

$$\begin{aligned}
Pr[chyba(\ell,\ell)] &\leq p_1 + \sum_{j=2}^{2\ell^2} p_j \leq \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} + \sum_{j=2}^{2\ell^2} \left(1 - \frac{1}{2\ell}\right)^i w_\ell \frac{1}{2\ell} \leq \\
&\leq \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} \left[\sum_{j=1}^{2\ell^2-1} \left(1 - \frac{1}{2\ell}\right)^j w_\ell + 1\right] \\
&\leq \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} \left[\sum_{j=1}^{2\ell^2-1} \underbrace{\left(1 - \frac{1}{2\ell}\right)^j + 1}_{<2}\right] \\
&< \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} 2\ell^2 \leq \frac{\ell^2}{2^{\ell-1}}
\end{aligned}$$

Pre $\ell \geq 100$ je $\frac{\ell^2}{2^{\ell-1}} \leq 1.58 \cdot 10^{-26}$

□

7.5 Optimalizačné úlohy a náhodné zaokrúhľovanie

Užitočným nástrojom pri riešení optimalizačných úloh je lineárne programovanie(LP). Zopakujme si — problém lineárneho programovania je definovaný nasledovne:

vstup matica $A = [a_{ij}], i = 1, \ldots, m; j = 1, \ldots, n$
vektor $b \in \mathbb{R}^m$
$c \in \mathbb{R}^n$
ohraničenia $M(A, b, c) = \{X \in (\mathbb{R}^{0,+})^n \mid AX = b\}$
cena $cost(X, (A, b, c)) = c^T X = \sum_{i=1}^{n} c_i x_i$
cieľ minimalizácia

Resp. ohraničenia s rovnosťou nahradíme ohraničeniami s nerovnosťami:
ohraničenia $\sum_{i=1}^{n} a_{ji} x_i = b_j, j \in I_1$
$\sum_{i=1}^{n} a_{ji} x_i \geq b_s, s \in I_2$
$\sum_{i=1}^{n} a_{ji} x_i \leq b_r, r \in \{1, \ldots, m\} - (I_1 \cup I_2)$

O úlohe lineárneho programovania vieme, že efektívnosť jej riešenia závisí od oboru hodnôt riešenia X.

- pre $X \in \mathbb{R}$ existuje algoritmus plynomiálnej zložitosti
- pri $X \in \mathbb{Z}$, tzv. ILP, resp. $X \in \{0, 1\}^n$, tzv. 0-1LP algoritmus polynomiálnej zložitosti nepoznáme

Vzhľadom k tejto rôznej zložitosti využívame LP pri návrhu *aproximačných* algoritmov. Základná schéma využitia je nasledovná:

1. formulácia pôvodného problému ako úlohy ILP, resp. 0-1LP; $ILP(I)$
2. relaxácia na úlohu lineárneho programovania LP; $Rel - LP(I)$
3. transformácia optimálneho riešenia relaxovaného problému na *prípustné* riešenie pôvodného problému. Výsledná kvalita získaného algoritmu zrejme záleží od tejto transformácie. Jednou z vhodných metód je *metóda náhodného zaokrúhľovania*.

Spravme relaxáciu $ILP(I) \rightsquigarrow Rel - LP(I)$. Nech
α je optimálne riešenie pre $Rel - LP(I)$
$Opt_U(I)$ je optimálne riešenie $ILP(I)$.
Potom (prečo?)

$$cost(\alpha) = Opt(Rel - LP(I)) \begin{cases} \leq Opt_U(I), & \text{pre minimalizačný problém} \\ \geq Opt_U(I), & \text{pre maximalizačný problém} \end{cases}$$

Pri návrate od optimálneho α k prípustnému β sa snažíme pokaziť kvalitu α čo najmenej. Toto je tá časť, ktorá súvisí s NP-ťažkosťou pôvodného problému ILP. Na príklade Min-VC ukážeme formuláciu optimalizačných úloh ako úloh LP.

Min-VC

Príklad 7.4 *Uvažujme problém minimálneho vrcholového pokrytia, v ktorom hľadáme minimálnu množinu vrcholov U, ktorá pokrýva všetky hrany. Prípustným riešením je teda vektor $(x_1, \ldots, x_n) \in \{0, 1\}^n$ taký, že $x_i = 1 \Leftrightarrow v_i \in U$. Formulujme ako úlohu LP:*

minimalizovať $\sum_{i=1}^{n} x_i$

podmienky $x_i + x_j \geq 1 \quad \forall (v_i, v_j) \in E$
$x_i \in \{0,1\}$

relaxácia *podmienku* $x_i \in \{0,1\}$ *nahrádzame dvomi podmienkami* $x_i \geq 0$ *a* $x_i \leq 1$

Majme optimálne riešenie $\alpha = (\alpha_1, \ldots, \alpha_n) \in [0,1]^n$ relaxovaného problému. K nemu zostrojíme prípustné riešenie β pôvodného problému zaokrúhlením takto

$$\beta_i = 1 \Leftrightarrow \alpha_i \geq 1/2$$

Ukážeme, že sme získali 2-aproximačný algoritmus.

- V optimálnom riešení relaxovanej úlohy α platí $\alpha_i + \alpha_j \geq 1$ pre každú hranu $(v_i, v_j) \in E$. To ale znamená, že aspoň pre jedno $\alpha_t, t \in \{i, j\}$ platí $\alpha_t \geq 1/2$. Následne do vrcholového pokrytia zaradíme vrchol v_t. Každá hrana je teda pokrytá. *prípustné riešenie*

- Keďže $\beta_i \leq 2\alpha_i$ pre každé i, o cene získaného prípustného riešenia β platí *kvalita*

$$cost(\beta) = \sum_{i=1}^{n} \beta_i \leq 2 \sum_{i=1}^{n} \alpha_i = 2cost(\alpha)$$

$$\text{Aproximačný pomer je } \frac{cost(\beta)}{cost(\alpha)} \leq \frac{2cost(\alpha)}{cost(\alpha)} = 2$$

◇

7.5.1 Náhodné zaokrúhľovanie a problém MaxSAT

Vráťme sa opäť k riešeniu problému MaxSat. V časti 6.2.1 sme ukázali, že náhodne vygenerované pripradenie hodnôt premenným vedie v očakávanom prípade k splneniu polovice klauzúl. V tejto časti ukážeme efektívny pravdepodobnostný algoritmus, ktorý je výsledkom kombinácie metódy relaxácie k LP a náhodného zaokrúhľovania.
Majme teda

$$F(x_1, \ldots, x_n) = F_1 \wedge F_2 \wedge \ldots \wedge F_m, \quad \text{kde } F_i \text{ je elementárna disjunkcia}$$

Označme

$Set^+(F_i)$ množina premenných zo $Set(F_i)$, ktoré sa v F_i vyskytujú bez negácie
$Set^-(F_i)$ množina premenných zo $Set(F_i)$, ktoré sa v F_i vyskytujú s negáciou
$In^+(F_i)$ indexy premenných z $Set^+(F_i)$
$In^-(F_i)$ indexy premenných z $Set^-(F_i)$

Riešime úlohu

formulácia

maximalizovať $\sum_{j=1}^{m} Z_j$

lineárne ohraničenia $\sum_{i \in IN^+(F_j)} y_i + \sum_{i \in IN^-(F_j)} (1 - y_i) \geq Z_j, \; j = 1, \ldots, m$

n+m nelineárnych ohraničení
$y_i \in \{0,1\} \; i = 1, \ldots, n$
$Z_j \in \{0,1\} \; j = 1, \ldots, m$

relaxácia
$1 \geq y_i, y_i \geq 0$
$1 \geq Z_j, Z_j \geq 0$

Algoritmus 56 RRR - random rounding

1: vstupom je úloha 0/1 LP(F)
2: relaxácia 0/1-LP(F) $\rightsquigarrow$ Rel-LP(F)
3: nech $(\alpha(y_1), \ldots, \alpha(y_n), \alpha(Z_1), \ldots, \alpha(Z_m))$ je optimálne riešenie Rel-LP(F)
4: vygeneruj náhodne $\gamma_1, \ldots, \gamma_n \in [0,1]^n$
5: **if** $\gamma_i \in [0, \alpha(y_i)]$ **then** $\beta_i \leftarrow 1$
6: **else** $\beta_i \leftarrow 0$
7: return $(\beta_1, \ldots, \beta_n)$

Označme $\alpha(u), u \in \{y_1, \ldots, y_n, Z_1, \ldots, Z_m\}$ hodnotu u v optimálnom riešení relaxovaného problému. Ľahko vidno, že $\sum_{j=1}^{m} \alpha(Z_j)$ je horný odhad na počet klauzúl, ktoré môžu byť splnené (prečo?). Kľúčovým pre kvalitné prípustné riešenie je zaokrúhľovanie; spravíme ho náhodne:
Uvedomme si, že takto definovaná $\alpha(y_i)$ je vlastne pravdepodobnosť toho, že $\beta_i = 1$[13]

Lema 7.37 *Nech $k \in N, F_j$ je klauzula s k literálmi, $(\alpha(y_1), \ldots, \alpha(y_n), \alpha(Z_1), \ldots, \alpha(Z_m))$ je optimálne riešenie Rel-LP(F). Potom pravdepodobnosť, že $\beta = RRR(F)$ spĺňa F_j, je aspoň*

$$\left(1 - \left(1 - \frac{1}{k}\right)^k\right) \alpha(Z_j)$$

Dôkaz: Keďže sa na ostatné klauzuly formuly nepozeráme, môžme predpokladať, že $F_j = x_1 \vee \ldots \vee x_k$. Z podmienok LP vyplýva, že

$$x_1 + x_2 + \ldots + x_k \geq Z_j$$

Klauzula F_j bude *nesplnená* práve vtedy ak každé $\beta_i \leftarrow 0, i = 1, \ldots, k$. To nastane s pravdepodobnosťou $\prod_{i=1}^{k}(1 - \alpha(y_i))$. *$F_j$ bude* splnená s pravdepodobnosťou

$$1 - \prod_{i=1}^{k}(1 - \alpha(y_i))$$

čo nadobúda minimum pre $\alpha(y_i) = \frac{\alpha(Z_j)}{k}$. Dostávame teda

$$Pr[F_j(\beta) = 1] \geq 1 - \prod_{i=1}^{k}\left(1 - \frac{\alpha(Z_j)}{k}\right)$$

čo je vlastne funkcia jednej premennej—$\alpha(Z_j)$. Využijeme nasledujúci fakt, ktorého platnosť ukončuje dôkaz.

Fakt 7.38 *Nech $k \in N$. Potom*

$$f_k(r) = 1 - \left(1 - \frac{r}{k}\right)^k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) r = g_k(r) \ \forall r \in [0,1]$$

K dôkazu faktu 7.38 si stačí uvedomiť, že

- funkcia g_r je lineárna
- funkcia f_k je konkávna
- $f_k(0) = 0 = g_k(0)$

[13] na rozdiel od 1/2 v prípade náhodného generovania pravdivostných hodnôt

- $f_k(1) = 1 - \left(1 - \frac{1}{k}\right)^k = g_k(1)$

□

Teraz už ľahko zanalyzujeme Algoritmus 56.

Veta 7.39 *Algoritmus RRR je polynomiálny*

1. *pravdepodobnostný* $E\left[\frac{e}{e-1}\right]$*-aproximačný pre problém MaxSAT*
2. *pravdepodobnostný* $E\left[\frac{k^k}{k^k-(k-1)^k}\right]$*-aproximačný pre problém Max-EkSAT*[14]

Dôkaz: Začnime s analýzou času. Redukcia 0/1-LP na Rel-LP je v lineárnom čase, optimálne riešenie Rel-LP vypočítame v polynomiálnom čase a získanie prípustného riešenia pre 0/1-LP z optimálneho pre Rel-LP spravíme v lineárnom čase. *čas*

Zopakuje si – algoritmus A je $E[\Delta]$-aproximačný, ak $E\left[\frac{Opt(U)}{A()}\right] \leq \Delta$, resp. ekvivalentne $E[A] \geq Opt/\Delta$. Stačí teda ukázať, že očakávaný počet splnených klauzúl je aspoň $\frac{e-1}{e}\sum_{j=1}^{m}\alpha(Z_j)$, resp. $\frac{k^k-(k-1)^k}{k^k}\sum_{j=1}^{m}\alpha(Z_j)$. *chyba*

Nech $\delta \in \{0,1\}^n$ je potenciálne riešenie. Potom

$$Prob[\delta = (\delta_1, \ldots, \delta_n)] = \prod_{i=1}^{n} q_i, \quad q_i = \begin{cases} \alpha(y_i), & \text{ak } \delta_i = 1 \\ (1 - \alpha(y_i)), & \text{ak } \delta_i = 0 \end{cases}$$

Uvažujme indikačnú premennú Z_j

$$Z_j(\delta) = \begin{cases} 1, & \text{ak } \delta \text{ spĺňa } F_j \\ 0, & \text{ak } \delta \ F_j \text{ nespĺňa} \end{cases}$$

Potom $E[Z_j]$ je pravdepodobnosť toho, že RRR(F) spĺňa F_j.

$$E[Z_j] \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right)\alpha(Z_j)$$

Nás zaujíma $E[Z]$ pre $Z = \sum_{j=1}^{m} Z_j$

$$E[Z] = \sum_{j=1}^{m} E[Z_j] \geq \sum_{j=1}^{m}\left(1 - \left(1 - \frac{1}{k}\right)^k\right)\alpha(Z_j) = \left(1 - \left(1 - \frac{1}{k}\right)^k\right)\sum_{j=1}^{m}\alpha(Z_j)$$

EkSAT

$$\begin{aligned} E[Ratio - EkSAT] &\leq \frac{Opt_{EkSAT}(F)}{E[Z]} \leq \frac{\sum_{j=1}^{m}\alpha(Z_j)}{\left(1 - \left(1 - \frac{1}{k}\right)^k\right)\sum_{j=1}^{m}\alpha(Z_j)} \\ &= \frac{1}{\left(1 - \frac{(k-1)^k}{k^k}\right)} = \frac{k^k}{k^k - (k-1)^k} \end{aligned}$$

Keďže $\left(1 - \frac{1}{k}\right)^k \leq e^{-1}$, je $1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - e^{-1} = \left(1 - \frac{1}{e}\right)$. Preto *SAT*

$$E[Z] \geq \left(1 - \frac{1}{e}\right)\sum_{j=1}^{m}\alpha(Z_j)$$

[14] Problém Max-EkSAT je MaxSAT obmedzený na vstupy, v ktorých každá klauzula má *presne* k literálov.

$$E[Ratio - SAT] = \frac{OptMaxSAT(F)}{E[Z]} \leq \frac{1}{(1-\frac{1}{e})} = \frac{e}{e-1}$$

Pri vylepšovaní opakovaním opakujeme len časť od vygenerovania náhodného vektora γ. □

7.5.2 Náhodná vzorka s náhodným zaokrúhľovaním

Prezentovali sme dva algoritmy pre riešenie problému MaxSAT — RSAM s pomerom 2 a RRR s pmerom $\frac{e}{e-1}$. Hoci $2 > \frac{e}{e-1}$, $\frac{2^k}{2^k-1} < \frac{k^k}{k^k-(k-1)^k}$ a teda pre dlhé klauzuly je naivný RSAM lepší. V skutočnosti sa dá ukázať (zamyslite sa), že existujú také vstupy I, I', I'' že

- $E[RSAM(I)] > E[RRR(I)]$ //MAxSAT
- $E[RRR(I')] > E[RSAM(I')]$ //MAxSAT
- $E[Ratio - RRR(I'')] < E[Ratio - RSAM(I'')]$ //MAx-EkSAT

Keďže majú rôzne správanie, je rozumné pokúsiť sa ich skombinovať. Spustíme oba algoritmy nezávisle a výsledkom bude lepší zo získaných výsledkov: Poďme analy-

Algoritmus 57 COMB-MAxSAT

1: $\beta \leftarrow RSAM(F)$
2: $\gamma \leftarrow RRR(F)$
3: **if** β spĺňa viac klauzúl ako γ **then** return β
4: **else** return γ

zovať tento algoritmus. Keďže oba algoritmy su polynomiálne, je polynomiálny aj COMB-MaxSAT.

čas

aproximačný pomer

Na vstupe máme formulu $F(x_1, \ldots, x_n) = F_1 \wedge \ldots \wedge F_m$.

$(S_{RSAM}, Prob_S)$	2^n možných výpočtov
$(S_{RRR}, Prob_R)$	priestor určuje optimálne riešenie α pre Rel-LP(F)
$(S_{RSAM} \times S_{RRR}, Prob)$	pričom $Prob[C, D] \stackrel{def.}{=} Prob_S[C] \times Prob_R[D]$

Majme beh algoritmu COMB-MaxSAT, ktorý vypočítal optimálne priradenia β, γ. Označme

$Y[\beta, \gamma]$ počet splnených klauzúl v $F(\beta)$

$Z[\beta, \gamma]$ počet splnených klauzúl v $F(\gamma)$

$U[\beta, \gamma] = \max\{Y[\beta, \gamma], Z[\beta, \gamma]\}$. Keďže maximum nie je menšie ako aritmetický priemer

$$E[U] \geq \frac{E[Y] + E[Z]}{2}$$

Kvôli podrobnejšej analýze si klauzuly rozdelíme do triedy podľa počtu literálov; C_k budú tie klazuly, ktoré majú presne k literálov.

Algoritmus RRR vypočíta priradenie, ktoré spĺňa nanajvýš $\sum_{j=1}^{m} \alpha(Z_j)$ klauzúl.

$$
\begin{aligned}
E[Z] &\geq \sum_{k\geq 1}\sum_{F_j\in C(k)}\left(1-\left(1-\frac{1}{k}\right)^k\right)\alpha(Z_j)\\
E[Y] &= \sum_{k\geq 1}\sum_{F_j\in C(k)}\left(1-\frac{1}{2^k}\right) \geq \sum_{k\geq 1}\sum_{F_j\in C(k)}\left(1-\frac{1}{2^k}\right)\alpha(Z_j)\\
E[U] &\geq \frac{E[Z]+E[Y]}{2} \geq \frac{1}{2}\sum_{k\geq 1}\sum_{F_j\in C(k)}\left[\left(1-\frac{1}{2^k}\right)+\left(1-\left(1-\frac{1}{k}\right)^k\right)\right]\alpha(Z_j)\\
&\geq \frac{1}{2}\cdot\frac{3}{2}\sum_{k\geq 1}\sum_{F_j\in C(k)}\alpha(Z_j) = \frac{3}{4}\sum_{j=1}^{m}\alpha(Z_j)
\end{aligned}
$$

Dokázali sme teda nasledovnú vetu:

Veta 7.40 *Algoritmus COMB-MAxSAT je polynomiálny pravdepodobnostný $E[4/3]$-aproximačný algoritmus pre problém MaxSAT.*