UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH

APROXIMAČNÉ A PRAVDEPODOBNOSTNÉ ALGORITMY

Ondrej Krídlo a Gabriel Semanišin

Prírodovedecká fakulta

Košice 2018

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
Prírodovedecká fakulta
Ústav informatiky

Aproximačné a pravdepodobnostné algoritmy

Vysokoškolské učebné texty

Ondrej Krídlo a Gabriel Semanišin

Košice 2018

Aproximačné a pravdepodobnostné algoritmy
Vysokoškolský učebný text

Autori:
RNDr. Ondrej Krídlo, PhD.
doc. RNDr. Gabriel Semanišin, PhD.
Ústav informatiky, Prírodovedecká fakulta, UPJŠ v Košiciach

Vedecký redaktor:
Prof. RNDr. Viliam Geffert, DrSc.
Ústav informatiky, Prírodovedecká fakulta, UPJŠ v Košiciach

Recenzenti:
RNDr. Zuzana Bednárová, PhD.
Ústav informatiky, Prírodovedecká fakulta, UPJŠ v Košiciach
RNDr. Ján Katrenič, PhD.

Vydavateľ: Univerzita Pavla Jozefa Šafárika v Košiciach
Dostupné od: 05.10.2018

ISBN 978-80-8152-622-0

Obsah

Predhovor **5**

1 Základy teórie pravdepodobnosti **7**
1.1 Základné pojmy 7
1.2 Podmienená pravdepodobnosť 9
1.3 Nezávislé udalosti 11
1.4 Náhodná premenná 12
1.5 Simulácia hracej kocky pomocou mince 16
1.6 Simulácia klasickej mince pomocou falošnej 17

2 Pravdepodobnostné algoritmy **19**
2.1 Problém verifikácie identity obsahu 19
2.2 Modely pravdepodobnostných algoritmov 23
2.2.1 Prvý model pravdepodobnostných algoritmov 24
2.2.2 Problém maximálnej splniteľnosti boolovskej formuly . 29
2.2.3 Druhý model pravdepodobnostných algoritmov 31
2.2.4 Problém triedenia 31
2.2.5 Problém nájdenia k-tého najmenšieho prvku 34
2.3 Klasifikácia pravdepodobnostných algoritmov 34
2.4 Algoritmy typu Las Vegas 35
2.5 Algoritmy typu Monte Carlo 40
2.5.1 Algoritmy typu Monte Carlo s jednostrannou chybou . 40
2.5.2 Algoritmy typu Monte Carlo s ohraničenou chybou . . 41
2.5.3 Algoritmy typu Monte Carlo s neohraničenou chybou 44

3 Aproximačné algoritmy **47**
3.1 Optimalizačný problém 47
3.2 Aproximačné riešenie 49
3.2.1 Rozvrhovací problém 50

3.3 Príklady ďalších algoritmov 52
3.3.1 Problém minimálneho vrcholového pokrytia 52
3.3.2 Problém množinového pokrytia 54
3.3.3 Problém maximálneho rezu 58
3.4 Klasifikácia optimalizačných problémov 59
3.5 Polynomiálne aproximačné schémy 61
3.5.1 Problém batohu 61
3.5.2 Problém obchodného cestujúceho 67
3.6 Neaproximovateľnosť 71
3.7 Randomizované aproximačné algoritmy 74

4 k-cestné vrcholové pokrytie 75
4.1 Motivácia 75
4.2 Základné pojmy a vlastnosti 75
4.3 Randomizovaný prístup 79
4.4 2-aproximácia pre 3PVCP 83
4.5 Variácie problému 83

5 Vzorové série úloh 85
5.1 Náhodné výbery a náhodné generátory 85
5.2 Algoritmy typu Las Vegas a Monte Carlo 86
5.3 Aproximačné algoritmy 87

Literatúra 89

Predhovor

Dnešný život je poznamenaný v minulosti neobvyklým prienikom výpočtových prostriedkov do bežného života. Kým v minulosti boli počítače skryté v súkromí výpočtových stredísk a prístup k nim mala len obmedzená skupina ľudí, dnes prakticky každý disponuje mobilným zariadením, ktoré sa svojimi schopnosťami, výpočtovým výkonom a úložným priestorom bez problémov vyrovná veľkým výpočtovým komplexom z minulosti.

To by mohlo navodzovať dojem, že v súčasnosti už máme dostatočnú výpočtovú kapacitu, aby sme dokázali vyriešiť akýkoľvek problém z teoretickej oblasti alebo aplikačnej praxe. Avšak nie je tomu tak. Stále existuje obrovská množina problémov, ktoré dokážeme riešiť iba pomocou exponenciálnych algoritmov. Ľahko môžeme nahliadnuť, že pri využívaní takýchto algoritmov rýchlo narazíme na neprekonateľné problémy. Ak totiž zhora odhadneme dĺžku jedného ľudského života (predpokladajme, že sa dožijeme 100 rokov, rok má 1 000 dní, deň 100 hodín, hodina 100 minút a minúta 100 sekúnd), tak sa dopracujeme k číslu 10^{11} sekúnd. Ak by náš výpočet potreboval vykonať iba 70! inštrukcií (tým, ktorí za týmto výrazom nevidia exponenciálnu funkciu odporúčame použiť Stirlingovu formulu), tak to predstavuje približne 10^{100} inštrukcií. Najnovší (v čase písania učebných textov) superpočítač IBM má výkon 200 petaflopov (jeden petaflop je 10^{15} operácií za sekundu), čo je 2.10^{28} operácií za 1 nami uvažovaný ľudský život. Evidentne je teda obrovský nepomer medzi tým, čo by sme potrebovali realizovať a čo počas nášho života reálne stihneme.

Pre informatikov je preto neustálou výzvou hľadať efektívne algoritmy, prípadne rôzne technické a implementačné možnosti vylepšenia výpočtovej zložitosti v súčasnosti známych algoritmov. Čiastočne riešenia ponúkajú napr. gridové prístupy a paralelizácia výpočtu. Veľkým, ale zatiaľ nenaplneným prísľubom, sú kvantové počítače. Z teoretického hľadiska veľmi dobré riešenia ponúkajú pravdepodobnostné a aproximačné prístupy. A práve nimi sa chceme podrobnejšie zaoberať.

Tieto elektronické vysokoškolské učebné texty sú doplnkovým učebným

textom k predmetu Aproximačné a pravdepodobnostné algoritmy. Učebné texty vznikli z podkladov autorov a prof. RNDr. Viliama Gefferta, DrSc., ktorému autori ďakujú za poskytnutie rukopisu. Rovnako poďakovanie patrí aj RNDr. Jánovi Katreničovi, PhD., ktorý zostavil prvú sériu úloh na cvičenia k danému predmetu, a Žanete Semanišinovej, ktorá prečítala veľkú časť textu a prispela svojimi návrhmi k jeho vylepšeniu.

Pre prácu s učebným textom sa očakáva, že čitateľ má základné vedomosti z teórie čísel, programovania, matematickej analýzy, teórie grafov a výpočtovej zložitosti.

V prvej kapitole sumarizujeme nevyhnutné vedomosti z teórie pravdepodobnosti, ktoré využívame v kapitole 4. Druhá kapitola je venovaná aproximačným algoritmom. Vychádza predovšetkým z renomovanej monografie J. Hromkoviča [2]. Obsahom tretej kapitoly sú pravdepodobnostné algoritmy. Táto kapitola vznikla predovšetkým na základe ďalšej monografie J. Hromkoviča [3]. Vo štvrtej kapitole predstavujeme problém k-cestného vrcholového pokrytia, ktorý má pôvod a je skúmaný aj na Ústave informatiky PF UPJŠ. Ide o zovšeobecnenie problému minimálneho vrcholového pokrytia a ukážeme, ako je možné ho riešiť pomocou aproximačných a pravdepodobnostných algoritmov. V piatej kapitole sú ukážkové série určené na samostatné precvičovanie učiva.

Autori

Kapitola 1

Základy teórie pravdepodobnosti

1.1 Základné pojmy

Hlavným cieľom tejto kapitoly je uviesť základné pojmy z teórie pravdepodobnosti, ktoré budú využívané v nasledujúcom texte. Ukážeme tiež niekoľko jednoduchých procedúr, ktoré budú využívané pri tvorbe pravdepodobnostných algoritmov.

Pokiaľ sa výsledok nejakého javu (udalosti, experimentu, ...) nedá predpovedať, hovoríme, že sa jedná o **náhodný jav** (*angl.* random event). Snáď najbežnejší príklad náhodného javu je hod mincou alebo hod kockou.

Všetky možné výsledky nejakého náhodného javu nazývame **elementárne náhodné udalosti**. Množinu všetkých elementárnych udalostí nazývame **výberový priestor** (*angl.* sample space - budeme ho označovať S). Uvažujme nasledovné príklady výberových priestorov:

- hod mincou $S = \{0, 1\}$ resp. {rub, líc}, $\{true, false\}$,
- hod kockou $S = \{1, 2, 3, 4, 5, 6\}$.

Náhodná udalosť (*angl.* random event, budeme označovať E) je ľubovoľná podmnožina výberového priestoru $E \subseteq S$, napríklad:

- na minci padne rub $E = \{0\}$,
- na kocke padne párne číslo $E = \{2, 4, 6\} \subseteq \{1, 2, 3, 4, 5, 6\}$,
- na kocke padne prvočíslo $E = \{2, 3, 5\} \subseteq \{1, 2, 3, 4, 5, 6\}$.

Distribúciou pravdepodobnosti nazývame každé zobrazenie ohodnocujúce náhodné udalosti $\mathrm{Prob} : \mathcal{P}(S) \to \langle 0, 1\rangle$, spĺňajúce nasledujúce podmienky:

(i) $\mathrm{Prob}(\{x\}) \geq 0$ pre všetky $x \in S$,

(ii) $\mathrm{Prob}(S) = 1$,

(iii) $\mathrm{Prob}(A \cup B) = \mathrm{Prob}(A) + \mathrm{Prob}(B)$ pre všetky $A, B \subseteq S$ také, že $A \cap B = \emptyset$,

Hodnotu $\mathrm{Prob}(A)$ nazývame **pravdepodobnosťou náhodnej udalosti** A. Dvojicu (S, Prob), kde S je výberový priestor a Prob je distribúcia pravdepodobnosti na S, nazývame **pravdepodobnostným priestorom**. Hovoríme, že distribúcia pravdepodobnosti Prob je **rovnomerná** na S ak $\mathrm{Prob}(\{x\}) = \frac{1}{|S|}$ pre všetky $x \in S$.

Lema 1.1.1 (Základné vlastnosti)**.** *Nech S je ľubovoľný priestor elementárnych náhodných udalostí. Ak Prob je distribúcia pravdepodobnosti na S, tak platia nasledovné vlastnosti:*

(i) $\mathrm{Prob}(\emptyset) = 0$,

(ii) $\mathrm{Prob}(S \setminus A) = 1 - \mathrm{Prob}(A)$,

(iii) $\mathrm{Prob}(A) \leq \mathrm{Prob}(B)$ *pre všetky* $A \subseteq B \subseteq S$,

(iv) $\mathrm{Prob}(A) = \sum_{x \in A} \mathrm{Prob}(\{x\})$.

Príklad 1.1.2. *Predstavme si zopár jednoduchých príkladov pravdepodobnostných priestorov s rovnomernou distribúciou pravdepodobnosti:*

- *klasická kocka* $S = \{1, 2, 3, 4, 5, 6\}$, $\mathrm{Prob}(\{x\}) = \frac{1}{6}$,
- *klasická minca (rovnomerne náhodný bit)* $S = \{0, 1\}$, $\mathrm{Prob}(\{x\}) = \frac{1}{2}$,
- *falošná minca* $S = \{0, 1\}$, $\mathrm{Prob}(\{0\}) = p$, $\mathrm{Prob}(\{1\}) = 1 - p$,
- *tri hody klasickou mincou* $S = \{000, 001, 010, 011, 100, 101, 110, 111\}$, $\mathrm{Prob}(\{x\}) = \frac{1}{8}$.

Príklad 1.1.3. *Nech n, k, $n > k$ sú dve prirodzené čísla. Uvažujme pravdepodobnostný priestor (S_n, Prob), kde $S_n = \{(x_1, \ldots, x_n), x_i \in \{rub, líc\}\}$ a* Prob *je rovnomerné rozdelenie pravdepodobnosti na S_n*

1. *Aká je pravdepodobnosť, že rub (líc) padne práve k-krát?*
2. *Aká je pravdepodobnosť, že rub (líc) padne aspoň (nanajvýš) k-krát?*

1.2 Podmienená pravdepodobnosť

Definícia 1.2.1. *Nech (S, Prob) je pravdepodobnostný priestor a $A, B \subseteq S$ sú ľubovoľné udalosti také, že $\mathrm{Prob}(B) \neq 0$. **Podmienená pravdepodobnosť** udalosti A za predpokladu, že nastala udalosť B, je*

$$\mathrm{Prob}(A|B) = \frac{\mathrm{Prob}(A \cap B)}{\mathrm{Prob}(B)}.$$

Príklad 1.2.2. *(Klasická kocka s rovnomerným rozdelením pravdepodobnosti) Uvažujme nasledovné udalosti:*

- $A = \{4, 5, 6\}$ *... padne číslo* ≥ 4,
- $B = \{2, 4, 6\}$ *... padne párne číslo.*

Potom $Prob(A|B)$ je pravdepodobnosť, že padne číslo ≥ 4 za predpokladu, že padlo číslo párne. Pre jednotlivé pravdepodobnosti platí:

- $\mathrm{Prob}(A) = \frac{1}{2}$ *a* $\mathrm{Prob}(B) = \frac{1}{2}$,
- $\mathrm{Prob}(A \cap B) = \frac{|\{4,6\}|}{|\{1,2,3,4,5,6\}|} = \frac{1}{3}$,
- $\mathrm{Prob}(A|B) = \frac{\mathrm{Prob}(A \cap B)}{\mathrm{Prob}(B)} = \frac{\frac{1}{3}}{\frac{1}{2}} = \frac{2}{3}$.

Daný výsledok zodpovedá aj situácii, že ak padlo 2, 4 alebo 6 tak pravdepodobnosť, že padlo číslo väčšie alebo rovné 4, je $\frac{2}{3}$.

Veta 1.2.3. *Nech (S, Prob) je pravdepodobnostný priestor. Nech $B \subseteq S$ je ľubovoľná taká udalosť, že $\mathrm{Prob}(B) > 0$. Potom aj (B, Prob') je tiež pravdepodobnostný priestor za predpokladu, že $\mathrm{Prob}'(A) = \mathrm{Prob}(A|B)$ pre ľubovoľnú $A \subseteq B$.*

Dôkaz. Overme vlastnosti priestoru (B, Prob').

(i) $\mathrm{Prob}'(\{x\}) = \mathrm{Prob}(\{x\}|B) = \frac{\mathrm{Prob}(\{x\}\cap B)}{\mathrm{Prob}(B)} \geq 0$ (lebo $\mathrm{Prob}(B) > 0$ je predpoklad vety a (S, Prob) je pravdepodobnostný priestor),

(ii) $\mathrm{Prob}'(B) = \mathrm{Prob}(B|B) = \frac{\mathrm{Prob}(B\cap B)}{\mathrm{Prob}(B)} = 1$,

(iii) $\mathrm{Prob}'(A_1 \cup A_2) = \frac{\mathrm{Prob}((A_1\cup A_2)\cap B)}{\mathrm{Prob}(B)} = \frac{\mathrm{Prob}((A_1\cap B)\cup(A_2\cap B))}{\mathrm{Prob}(B)} =$
$= \frac{\mathrm{Prob}(A_1\cap B)+\mathrm{Prob}(A_2\cap B)}{\mathrm{Prob}(B)} = \cdots = \mathrm{Prob}'(A_1) + \mathrm{Prob}'(A_2)$.

Teda (B, Prob') spĺňa definíciu pravdepodobnostného priestoru. □

Príklad 1.2.4. *Podmienenú pravdepodobnosť môžeme vnímať ako zúženie pravdepodobnostného priestoru. Príkladom je hádzanie klasickou kockou, kde výberový priestor* $S = \{1, 2, 3, 4, 5, 6\}$ *s rovnomerným rozdelením pravdepodobnosti* $\mathrm{Prob}(\{x\}) = \frac{1}{6}$ *zúžime iba na párne čísla* $S' = \{2, 4, 6\}$ *a dostávame* $\mathrm{Prob}(\{2\}) = \mathrm{Prob}(\{4\}) = \mathrm{Prob}(\{6\}) = \frac{1}{3}$.

Úvahu z predchádzajúceho príkladu môžeme rozšíriť nasledovne:

Veta 1.2.5. *Ak* (S_1, Prob_1) *je pravdepodobnostný priestor a* f *je funkcia taká, že* $f : S_1 \to S_2$*, tak aj* (S_2, Prob_2) *je pravdepodobnostný priestor, pričom*

$$\mathrm{Prob}_2(\{y\}) = \sum_{f(x)=y} \mathrm{Prob}_1(\{x\}).$$

Príklad 1.2.6. *Uvažujme klasickú kocku* $S_1 = \{1, 2, 3, 4, 5, 6\}$ *s rovnomerným rozdelením pravdepodobnosti ako pravdepodobnostný priestor. Uvažujme zobrazenie* $f : \{1, 2, 3, 4, 5, 6\} \to \{0, 1\}$*, ktoré je definované nasledovne:*

- $f(1) = f(2) = f(3) = f(4) = 1$,
- $f(5) = f(6) = 0$.

Funkcia f *nám v takom prípade pretvorí klasickú kocku na falošnú mincu s rozdelením pravdepodobnosti* $\mathrm{Prob}(\{1\}) = \frac{4}{6} = \frac{2}{3}$ *a* $\mathrm{Prob}(\{0\}) = \frac{2}{6} = \frac{1}{3}$.

Nasledujúca lema popisuje dôležité vlastnosti podmienenej pravdepodobnosti.

Lema 1.2.7. *Nech* A, B *a* $A_i, i \in \{1, 2, \ldots, n\}$ *sú náhodné udalosti. Potom*

(i) $\mathrm{Prob}(A|B) = Prob(B|A) \cdot \frac{\mathrm{Prob}(A)}{\mathrm{Prob}(B)}$ *(*Bayesov vzorec*),*

(ii) $\mathrm{Prob}(\bigcap_{i=1}^{n} A_i) = \mathrm{Prob}(A_1) \cdot \mathrm{Prob}(A_2|A_1) \cdot \mathrm{Prob}(A_3|A_1 \cap A_2) \cdot \cdots \cdot \mathrm{Prob}(A_n|\bigcap_{i=1}^{n-1} A_i)$ *za predpokladu, že* $\mathrm{Prob}(\bigcap_{i=1}^{k} A_i) \neq 0$ *pre všetky* $k \in \{1, \ldots, n-1\}$.

1.3 Nezávislé udalosti

Definícia distribúcie pravdepodobnosti zahŕňa základný fakt, že pravdepodobnosť niekoľkých navzájom disjunktných udalostí zodpovedá súčtu jednotlivých udalostí. Poďme sa teraz pozrieť, čomu zodpovedá súčin pravdepodobností.

Definícia 1.3.1. *Dve udalosti $A, B \subseteq S$ nazývame **nezávislými** ak platí, že*

$$\mathrm{Prob}(A \cap B) = \mathrm{Prob}(A) \cdot \mathrm{Prob}(B).$$

Príklad 1.3.2. *Uvažujme tri hody klasickou mincou, t.j. $S_3 = \{0,1\}^3$, a nasledovné dve udalosti*

- $A = 1\{0,1\}^2$ *... v prvom hode padne rub,*
- $B = \{0,1\}0\{0,1\}$ *... v druhom hode padne líc.*

Potom

- $\mathrm{Prob}(A) = \frac{|\{110,101,110,111\}|}{2^3} = \frac{4}{8} = \frac{1}{2}$,
- $\mathrm{Prob}(B) = \frac{|\{000,001,100,101\}|}{2^3} = \frac{4}{8} = \frac{1}{2}$,
- $\mathrm{Prob}(A \cap B) = \frac{|\{100,101\}|}{2^3} = \frac{2}{8} = \frac{1}{4} = \mathrm{Prob}(A) \cdot \mathrm{Prob}(B)$,

čo znamená, že udalosti sú nezávislé.

Príklad 1.3.3. *Uvažujme opäť hod klasickou kockou a udalosti*

- $A = \{2,4,6\}$ *... pri hode padne párne číslo, $Prob(A) = \frac{1}{2}$,*
- $B = \{2,3,5\}$ *... pri hode padne prvočíslo, $Prob(B) = \frac{1}{2}$.*

Potom $\mathrm{Prob}(A \cap B) = \frac{|\{2\}|}{6} = \frac{1}{6} \neq \mathrm{Prob}(A) \cdot Prob(B)$ *a z toho vyplýva, že dané udalosti nie sú nezávislé.*

Nasledujúca lema popisuje nutnú a postačujúcu podmienku pre nezávislé udalosti.

Lema 1.3.4. *Nech (S, Prob) je pravdepodobnostný priestor, $A, B \subseteq S$, $\mathrm{Prob}(B) \neq 0$. Potom A, B sú nezávislé udalosti práve vtedy, keď platí $\mathrm{Prob}(A|B) = \mathrm{Prob}(A)$.*

Dôkaz. Predpokladajme najprv, že A a B sú nezávislé. Potom

$$\text{Prob}(A|B) = \frac{\text{Prob}(A \cap B)}{\text{Prob}(B)} = \frac{\text{Prob}(A) \cdot \text{Prob}(B)}{\text{Prob}(B)} = \text{Prob}(A).$$

Ak $\text{Prob}(A|B) = \text{Prob}(A)$, tak platí

$$\text{Prob}(A) \cdot \text{Prob}(B) = \text{Prob}(A|B) \cdot \text{Prob}(B) = \\ = \frac{\text{Prob}(A \cap B)}{\text{Prob}(B)} \cdot \text{Prob}(B) = \text{Prob}(A \cap B)$$

a z toho vyplýva, že A a B sú nezávislé.

□

1.4 Náhodná premenná

Ľubovoľnú funkciu nad výberovým priestorom $X : S \to \mathbb{R}$ nazývame **náhodná premenná** na S (V skutočnosti funkcia X nemôže byť úplne ľubovoľná, ale pre naše potreby môžeme technické detaily vynechať). Ak je množina udalostí konečná (spočítateľná), tak

$$\mathbb{R}_X = \{x \in \mathbb{R}; x = X(s), s \in S\}$$

bude tiež konečná (spočítateľná).

Pre danú náhodnú premennú $X : S \to \mathbb{R}$ a dané $x \in \mathbb{R}$ definujeme

$$\text{Event}(X = x) = \{s \in S; X(s) = x\}$$

je množina tých udalostí priestoru S, pre ktoré $X(s)$ nadobúda hodnotu x.

Potom $f_X(x) = \text{Prob}(\text{Event}(X = x))$ je pravdepodobnosť, že náhodná premenná X nadobúda hodnotu x (zapisujeme aj stručnejšie $\text{Prob}(X = x)$).

Distribučná funkcia pre náhodnú premennú $X : S \to \mathbb{R}$ je funkcia $Dis_X : \mathbb{R} \to \langle 0, 1\rangle$ definovaná ako

$$Dis_X(x) = \text{Prob}(X \leq x) = \sum_{\hat{x} \in \mathbb{R}_X : \hat{x} \leq x} \text{Prob}(\text{Event}(X = \hat{x})).$$

Príklad 1.4.1. *Uvažujme tri hody klasickou kockou* $S = \{1,2,3,4,5,6\}^3$, $s = [s_1, s_2, s_3]$. *Vezmime* $X(s) = X([s_1, s_2, s_3]) = s_1 + s_2 + s_3$ *náhodnú premennú reprezentujúcu súčet čísel hodených v troch hodoch za sebou. V takom prípade* $\mathrm{Prob}(s) = \frac{1}{6^3} = \frac{1}{216}$ *pre všetky* $s \in S$.

$$\mathrm{Prob}(X = 5) = \sum_{[s_1,s_2,s_3]\in S; s_1+s_2+s_3=5} \mathrm{Prob}(\{[s_1, s_2, s_3]\}) = 6 \cdot \frac{1}{216} = \frac{1}{36},$$

pričom množina vyhovujúcich trojíc je

$$\{[1,1,3],[1,3,1],[3,1,1],[1,2,2],[2,1,2],[2,2,1]\}.$$

Uvažujme teraz inú náhodnú premennú $Y([s_1, s_2, s_3]) = \max\{s_1, s_2, s_3\}$ *(z troch hodov kockou vyberieme najvyšší hod). Potom*

$$\mathrm{Prob}(Y = 3) = \frac{1}{216} \cdot (9 + 6 + 4) = \frac{19}{216},$$

pričom vyhovujúca podmnožina výberového priestoru je

$$\{[3,\{1,2,3\},\{1,2,3\}],[\{1,2\},3,\{1,2,3\}],[\{1,2\},\{1,2\},3]\}.$$

Ak máme dve náhodné premenné $X, Y : S \to \mathbb{R}$ tak definujme udalosť

$$\begin{aligned}\mathrm{Event}(X = x \text{ and } Y = y) &= \mathrm{Event}(X = x) \cap \mathrm{Event}(Y = y) = \\ &= \{s \in S; X(s) = x \text{ and } Y(s) = y\}.\end{aligned}$$

Dve náhodné premenné X, Y nazveme **nezávislé**, ak pre všetky $x \in \mathbb{R}_X$ a $y \in \mathbb{R}_Y$ platí

$$\mathrm{Prob}(X = x \text{ and } Y = y) = \mathrm{Prob}(X = x) \cdot \mathrm{Prob}(Y = y).$$

Priemerná hodnota (*angl.* expected value) náhodnej premennej X je definovaná ako

$$\mathrm{E}[X] = \sum_{x\in\mathbb{R}_X} x \cdot \mathrm{Prob}(X = x).$$

Pri definícii sa požaduje buď konečnosť sumy alebo absolútna konvergencia.

Príklad 1.4.2. *Nech daný výberový priestor je* $S = \{0,1\}^4$, *t.j. uvažujeme štyri hody mincou a* $X(i) = X([i_1, i_2, i_3, i_4]) = i_1 + i_2 + i_3 + i_4$, *čo zodpovedá počtu jednotiek v štyroch hodoch, napr.* $X([1,0,0,1]) = 2$. *Potom*

- $\text{Prob}(i) = \frac{1}{2^4} = \frac{1}{16}$,
- $\text{Prob}(X = 0) = \binom{4}{0} \cdot \frac{1}{16} = \frac{1}{16}$ ... (0000),
- $\text{Prob}(X = 1) = \binom{4}{1} \cdot \frac{1}{16} = \frac{4}{16} = \frac{1}{4}$ ... (0001, 0010, 0100, 1000),
- $\text{Prob}(X = 2) = \binom{4}{2} \cdot \frac{1}{16} = \frac{3}{8}$ ... (0011, 0101, 1001, 0110, 1010, 1100),
- $\text{Prob}(X = 3) = \binom{4}{3} \cdot \frac{1}{16} = \frac{4}{16} = \frac{1}{4}$ ... (1110, 1101, 1011, 0111),
- $\text{Prob}(X = 4) = \binom{4}{4} \cdot \frac{1}{16} = \frac{1}{16}$ ... (1111).

Priemerný počet jednotiek je teda

$$\text{E}(X) = 0 \cdot \frac{1}{16} + 1 \cdot \frac{1}{4} + 2 \cdot \frac{3}{8} + 3 \cdot \frac{1}{4} + 4 \cdot \frac{1}{16} = \frac{32}{16} = 2.$$

Vezmime teraz náhodnú premennú $Y([i_1, i_2, i_3, i_4]) = i_1 + i_2 + i_3$, čiže počet jednotiek v prvých troch hodoch. V takom prípade máme:

- $\text{Prob}(Y = 0) = \binom{3}{0} \cdot 2 \cdot \frac{1}{16} = \frac{2}{16}$ ... (0001, 0000),
- $\text{Prob}(Y = 1) = \binom{3}{1} \cdot 2 \cdot \frac{1}{16} = \frac{6}{16}$ ... (001*, 010*, 100*), *kde* $* \in \{0, 1\}$,
- $\text{Prob}(Y = 2) = \binom{3}{2} \cdot 2 \cdot \frac{1}{16} = \frac{6}{16}$ ... (110*, 101*, 011*), *kde* $* \in \{0, 1\}$,
- $\text{Prob}(Y = 3) = \binom{3}{3} \cdot 2 \cdot \frac{1}{16} = \frac{2}{16}$ ... (1110, 1111),
- $\text{Prob}(Y = 4) = 0$ ... *v prvých troch hodoch nepadne* 4.

Potom

$$\text{E}(Y) = 0 \cdot \frac{2}{16} + 1 \cdot \frac{6}{16} + 2 \cdot \frac{6}{16} + 3 \cdot \frac{2}{16} = \frac{3}{2},$$

čo znamená, že priemerný počet jednotiek v prvých troch hodoch zo štyroch je 1,5.

Ďalšia uvažovaná náhodná premenná bude počet jednotiek v štvrtom hode. Pre ňu platí:

$$\text{E}(Z) = 0 \cdot \frac{8}{16} + 1 \cdot \frac{8}{16} = \frac{1}{2} = \frac{1}{2}.$$

Môžeme pozorovať zaujímavý efekt: $X(s) = Y(s) + Z(s)$, čo je počet jednotiek v štyroch hodoch, je rovný súčtu počtov jednotiek v troch hodoch a počtu jednotiek v štvrtom hode. Pre priemerný počet jednotiek teda dostávame:

$$\text{E}(X) = 2 = \frac{3}{2} + \frac{1}{2} = \text{E}(Y) + \text{E}(Z) = \text{E}(Y + Z).$$

Skôr ako uvedieme ďalší ilustračný príklad, pripomenieme dve dôležité identity.

Lema 1.4.3. $\sum_{i=0}^{\infty} p^i = \frac{1}{1-p}$ *pre* $p \in (0,1)$.

Dôkaz.

$$\begin{aligned} S_n &= 1 + p + p^2 + p^3 + \cdots + p^{n-1} + p^n \\ p \cdot S_n &= p + p^2 + p^3 + p^4 + \cdots + p^n + p^{n+1} \\ S_n - p \cdot S_n &= 1 - p^{n+1} \\ S_n &= \frac{1 - p^{n+1}}{1-p}. \end{aligned}$$

Úhrnom máme

$$\sum_{i=0}^{\infty} p^i = \lim_{n\to\infty} \sum_{i=0}^{n} p^i = \lim_{n\to\infty} \frac{1-p^{n+1}}{1-p} = \frac{1}{1-p},$$

pretože z predpokladu lemy $p \in (0,1)$ dostávame okamžite, že $p^n \to 0$ pre $n \to \infty$. □

Lema 1.4.4. $\sum_{i=1}^{\infty} i \cdot p^{i-1} = \frac{1}{(1-p)^2}$ *pre* $p \in \langle 0,1\rangle$.

Dôkaz.

$$\begin{aligned} \sum_{i=1}^{\infty} i \cdot p^{i-1} = 1 \cdot p^0 + 2 \cdot p^1 + 3 \cdot p^2 + 4 \cdot p^3 + 5 \cdot p^4 \cdots = \\ = p^0 + p^1 + p^2 + p^3 + p^4 + \cdots + \\ + p^1 + p^2 + p^3 + p^4 + \cdots + \\ + p^2 + p^3 + p^4 + \cdots + \\ + p^3 + p^4 + \cdots + \\ + p^4 + \cdots + \\ \cdots = \\ = \frac{1}{1-p} + p \cdot \frac{1}{1-p} + p^2 \cdot \frac{1}{1-p} + p^3 \cdot \frac{1}{1-p} + p^4 \cdot \frac{1}{1-p} + \cdots = \\ = \frac{1}{1-p} \cdot \left(1 + p + p^2 + p^3 + p^4 + \cdots\right) = \\ = \frac{1}{1-p} \cdot \frac{1}{1-p} = \frac{1}{(1-p)^2}. \end{aligned}$$

□

Príklad 1.4.5 (Tvrdohlavý hazardér - dôležitý príklad, ktorý sa uplatňuje v mnohých aplikáciach)**.** *Majme falošnú mincu s rozdelením pravdepodobnosti*

- *1 padá s pravdepodobnosťou* $1-p$*,*
- *0 padá s pravdepodobnosťou* p*.*

Tvrdohlavý hazardér hádže mincou dovtedy, kým nepadne 1 (kľudne aj do nekonečna). Zaujíma nás, aký je priemerný počet hodov.

Výberový priestor je v tomto prípade $S = \{1, 2, 3, 4, \dots\}$*, t.j. počet hodov. Potom* $Prob(\{i\}) = p^{i-1} \cdot (1-p)$*, pretože najprv* $(i-1)$*-krát padne 0 a potom padne 1.*

Náhodná premenná $X(i) = i$ *zodpovedá počtu pokusov kým nepadne 1. Priemerný počet pokusov v takom prípade bude:*

$$\begin{aligned} \mathrm{E}(X) &= 1.(1-p) + 2.p.(1-p) + 3.p^2.(1-p) + \cdots = \\ &= (1-p).\sum_{i=1}^{\infty} i.p^{i-1} = (1-p).\frac{1}{(1-p)^2} = \frac{1}{1-p}. \end{aligned}$$

1.5 Simulácia hracej kocky pomocou mince

Máme k dispozícii klasickú mincu. Chceme zostrojiť procedúru, ktorá simuluje náhodný hod klasickou kockou. Požadované riešenie dostaneme tak, že hádžeme mincou trikrát ($4 = 2^2 < 6 < 2^3 = 8$, teda hádzať iba dvakrát by nestačilo) a použijeme nasledovný postup:

- $000 \to 1$,
- $001 \to 2$,
- $010 \to 3$,
- $011 \to 4$,
- $100 \to 5$,
- $101 \to 6$,
- $11 \to$ opakuj celý postup znova $\dots$,
- $11000 \to 1$,
- $11001 \to 2$,

- ...

Teoreticky môže celý postup pokračovať do nekonečna.

Pravdepodobnosť že padne $i \in \{1,2,3,4,5,6\}$ je rovná súčtu nekonečného radu pravdepodobností, že i padne v prvom kole, druhom kole, treťom kole, ...

$$\frac{1}{2^3} + \frac{1}{2^2}\cdot\frac{1}{2^3} + \frac{1}{2^2}\cdot\frac{1}{2^2}\cdot\frac{1}{2^3} + \frac{1}{2^2}\cdot\frac{1}{2^2}\cdot\frac{1}{2^2}\cdot\frac{1}{2^3} + \cdots = \frac{1}{8}\cdot\sum_{i=0}^{\infty}\left(\frac{1}{4}\right)^i = \\ = \frac{1}{8}\cdot\frac{1}{1-\frac{1}{4}} = \frac{1}{8\cdot\frac{3}{4}} = \frac{1}{6}.$$

Teda je vidieť, že každé číslo padá rovnomerne s pravdepodobnosťou $\frac{1}{6}$ a nami vytvorená procedúra perfektne simuluje klasickú kocku. Poďme sa ale pozrieť na priemerný čas výpočtu nejakého čísla (v najhoršom prípade je čas $= \infty$). Priemerný čas výpočtu vyrátame ako priemernú hodnotu náhodnej premennej, ktorá každému prvku výberového priestoru typu $(11)^{i-1}x$, kde $x \in \{000, 001, 010, 011, 100, 101\}$ priradíme hodnotu i, teda $X((11)^{i-1}x) = i$, kde i symbolizuje počet kôl pri generovaní čísla kocky. Dostávame:

$$1\cdot\frac{6}{8} + 2\cdot\frac{1}{4}\cdot\frac{6}{8} + 3\cdot\frac{1}{4}\cdot\frac{1}{4}\cdot\frac{6}{8} + \cdots = \frac{3}{4}\sum_{i=1}^{\infty} i\cdot\left(\frac{1}{4}\right)^{i-1} = \frac{3}{4}\cdot\frac{1}{\left(1-\frac{1}{4}\right)^2} = \frac{4}{3},$$

čiže priemerný počet kôl pre vygenerovanie čísla kocky je $\frac{4}{3}$. V každom kole sa hádže dvakrát mincou okrem posledného kola, kedy hádžeme trikrát. Teda priemerný počet hodov mincou je $1 + 2\cdot\frac{4}{3} = \frac{11}{3} \approx 3,66$.

1.6 Simulácia klasickej mince pomocou falošnej

Máme k dispozícii falošnú mincu s $Prob(1) = 1 - p$ a $Prob(0) = p$. Chceme vyrobiť procedúru, ktorá použitím tejto falošnej mince simuluje generátor náhodných bitov s rovnomerným rozdelením $Prob'(x) = \frac{1}{2}$ pre $x \in \{0,1\}$. Hodnota p nie je známa, ale v nasledujúcej úvahe ukážeme, že to nie je prekážkou.

Riešenie nášho problému je nasledovné:

- $01 \to 1$,
- $10 \to 0$,

- $00, 11 \rightarrow$ opakuj odznova (teoreticky aj do nekonečna).

Označme $q = 2p(1-p)$ a analyzujme teraz situáciu týkajúcu sa jedného kola:

- $\overline{\text{Prob}'}(0) = (1-p)p = \frac{1}{2}q$,
- $\overline{\text{Prob}'}(1) = p(1-p) = \frac{1}{2}q$,
- $\overline{\text{Prob}'}(opakovanie) = p.p + (1-p).(1-p) = 1-q$.

Následne stanovíme pravdepodobnosť, že padne 0 alebo 1:

$$\begin{aligned}\text{Prob}'(0) &= \frac{1}{2}\cdot q + (1-q)\cdot\frac{1}{2}\cdot q + (1-q)^2\cdot\frac{1}{2}\cdot q + \cdots = \\ &= \frac{1}{2}\cdot q\cdot\sum_{i=0}^{\infty}(1-q)^i = \frac{1}{2}\cdot q\cdot\frac{1}{1-(1-q)} = \frac{1}{2}.\end{aligned}$$

Hodnotu $\text{Prob}'(1) = \frac{1}{2}$ získame rovnakým spôsobom. Popísaná procedúra teda evidentne simuluje klasickú mincu s rovnomerným rozdelením.

Nakoniec analyzujme priemerný čas simulácie. Podobne ako v predchádzajúcom príklade:

$$\begin{aligned}&1\cdot q + 2\cdot(1-q)\cdot q + 3\cdot(1-q)\cdot(1-q)\cdot q + \cdots = \\ &= q\cdot\sum_{i=1}^{\infty} i\cdot(1-q)^{i-1} = q\cdot\frac{1}{(1-(1-q))^2} = q\cdot\frac{1}{q^2} = \frac{1}{2\cdot p\cdot(1-p)}.\end{aligned}$$

Priemerný počet hodov mincou je dvojnásobok priemerného počtu kôl, čo činí $\frac{1}{p\cdot(1-p)}$. Funkcia $\frac{1}{p\cdot(1-p)}$ nadobúda minimum pre $p = \frac{1}{2}$. Znamená to, že čím je minca falošnejšia, tým ťažšie bude simulovať klasickú mincu.

Kapitola 2

Pravdepodobnostné algoritmy

V tejto časti sa budeme venovať algoritmom, ktoré sa nesprávajú deterministicky, ale využívajú istý prvok náhodnosti. Ukážeme, že pomocou takéhoto prístupu vieme v praxi dosahovať veľmi zaujímavé výsledky.

2.1 Problém verifikácie identity obsahu

V tejto časti sa budeme zaoberať problémom verifikácie identity obsahu, ktorý spočíva v porovnaní obsahu dvoch vzdialených databáz R_1 a R_2, pričom každá obsahuje rovnaký počet bitov.

$$R_1 : x_1 x_2 \dots x_n \leftrightarrow R_2 : y_1 y_2 \dots y_n$$

Rádovo je veľkosť databáz $n \approx 10^{16}$. Pri priamočiarom deterministickom riešení R_1 pošle R_2 celý obsah, teda všetkých n bitov, čo je časovo náročný proces. Ešte náročnejšie by bolo zaručiť korektnosť prenosu všetkých bitov z jedného miesta na druhé.

Navrhneme preto pravdepodobnostné riešenie. K tomu pre reťazec $x = x_1 x_2 \dots x_n$, $x_i \in \{0, 1\}$ definujeme $Number(x) = \sum_{i=1}^{n} 2^{n-i} x_i$, teda dekadické číslo, ktorého binárnym zápisom je reťazec x. Porovnanie databáz realizujeme algoritmom `DBComp`.

Analyzujme teraz komunikačnú zložitosť algoritmu:

- na binárny zápis (akéhokoľvek) čísla $m \in \mathbb{N}$ potrebujeme $1 + \lfloor \log_2 m \rfloor$ bitov,

Algoritmus 1 `DBComp`

Require: reťazec $x_1x_2 \dots x_n$ lokalizovaný na R_1 a $y_1y_2 \dots y_n$ na R_2

1: R_1 zvolí náhodne prvočíslo $p \in \{2, \dots, n^2\}$ ($\star$ všetky prvočísla z danej množiny majú rovnakú pravdepodobnosť výberu $\star$)
2: R_1 zráta $s = Number(x) \mod p$ a pošle binárne reprezentácie čísel p a s databáze R_2 ($\star$ $s \leq p < n^2$ a teda každé z týchto čísel bude reprezentované $\lceil \log_2 n^2 \rceil$ bitmi $\star$)
3: R_2 po prečítaní s a p zráta $q = Number(y) \mod p$ a porovná či $s = q$
4: **if** $s \neq q$ **then**
5: **return** $x \neq y$
6: **else**
7: **return** $x = y$
8: **end if**

- prenášame dve čísla $s < p < n^2$, teda stačí nám $2(1 + \lfloor \log_2 n^2 \rfloor) \leq 4\lceil \log_2 n \rceil \leq O(\log_2 n)$

Pre databázu veľkosti $n = 10^{16}$ stačí teda preniesť správu o dĺžke

$$4\lceil \log_2 10^{16} \rceil \leq 4\lceil 16 \log_2 10 \rceil \leq 256$$

bitov. Takáto krátka správa sa dá vždy bezpečne preniesť (použitím vhodných kontrolných mechanizmov).

Avšak uvedený algoritmus nemusí dať vždy správny výsledok. Napríklad v situácii:

- $x = 01111$; $Number(x) = 15$,
- $y = 10110$; $Number(y) = 22$

Ak vyberieme náhodne z prvočísel $\{2, 3, 5, 7, 11, 13, 17, 19\}$ číslo 7, tak dostaneme $s = 15 \mod 7 = 2$ a $q = 22 \mod 7 = 2$ a algoritmus vráti odpoveď $x = y$, čo nie je pravda.

Vidíme, že takýto pravdepodobnostný prístup dáva na výstup aj chybné výsledky. Stanovme teraz pravdepodobnosť takejto chyby. Označme ako $P(m)$ množinu $\{p : p \leq m$ a p je prvočíslo$\}$. Pre každú dvojicu (x, y) budeme uvažovať množinu

$$BadP(x, y) = \{p \in P(n) : \text{ algoritmus vráti zlú odpoveď pre } (x, y)\}.$$

Pravdepodobnosť chyby bude potom:

$$\text{Prob}_{error}(x, y) = \frac{|BadP(x, y)|}{|P(n^2)|}$$

Rozlíšime dva prípady:

Prípad 1. Ak $x = y$, tak $N(x) \mod p = N(y) \mod p$ pre všetky prvočísla p, teda popísaný algoritmus dá odpoveď $x = y$ pre všetky $p \leq n^2$. Čiže $BadP(x, y) = \emptyset$ a teda $\text{Prob}_{error}(x, y) = 0$. Pravdepodobnosť chyby je 0. V prípade, že $x = y$ algoritmus vráti vždy správnu odpoveď.

Prípad 2. Ak $x \neq y$, potom potrebujeme podrobnejšiu analýzu:
Prime Number Theorem (veta o prvočíslach) hovorí, že

$$\lim_{m\to\infty} \frac{|P(m)|}{\frac{m}{\ln m}} = 1$$

a takisto bolo dokázané, že pre všetky $m \geq 67$ platí:

$$|P(m)| > \frac{m}{\ln m}.$$

Ďalej pre všetky $n \geq 9$ platí:

$$|P(n^2)| > \frac{n^2}{2 \ln n}. \tag{2.1}$$

Algoritmus pre $x \neq y$ dá zlú odpoveď iba ak (predpokladajme bez ujmy na všeobecnosti $x > y$) nastane situácia $N(x) \mod p = N(y) \mod p$ t.j.:

$$w = (N(x) - N(y)) \mod p = 0.$$

To znamená, že p je deliteľom w. Pritom x, y sú binárne n-ciferné čísla a teda $w < 2^n$ a nedá sa deliť viac ako $n - 1$ rôznymi prvočíslami. V opačnom prípade sme mohli w faktorizovať nasledovne:

$$\begin{aligned} w &= p_{i_1}^{\alpha_1}.p_{i_2}^{\alpha_2} \dots p_{i_n}^{\alpha_n} \\ &\geq p_{i_1}.p_{i_2} \dots p_{i_n} \\ &\geq p_1.p_2 \dots p_n \quad (\star \text{ predpokladáme, že } p_{i_1} < \dots < p_{i_n} \star) \\ &\geq 2.2 \dots 2 = 2^n, \end{aligned}$$

čo je spor s tým, že $w < 2^n$. Počet prvočísel, ktoré delia w je maximálne $n - 1$, a teda zrejme

$$|BadP(x, y)| \leq n - 1. \tag{2.2}$$

Zo vzťahov 2.1 a 2.2 vyplýva, že:

$$\text{Prob}_{error}(x, y) \leq \frac{n-1}{\frac{n^2}{2 \ln n}} \leq \frac{2 \ln n}{n}$$

pre $n \geq 9$. V našom prípade je $n = 10^{16}$ a

$$\text{Prob}_{error}(x, y) \leq 0,369.10^{-14}.$$

Ak máme pravdepodobnosť chyby $\approx 10^{-14}$, tak riziko získania nesprávneho výsledku je veľmi malé. Pre zníženie pravdepodobnosti je možné navyše algoritmus viackrát zopakovať.

Algoritmus 2 `DBComp10`

Require: reťazec $x_1x_2 \ldots x_n$ lokalizovaný na R_1 a $y_1y_2 \ldots y_n$ na R_2

1: R_1 zvolí náhodne 10 prvočísel $p_1, p_2, \ldots, p_{10}$ z množiny $\{2, \ldots, n^2\}$
2: R_1 zráta $s_i = Number(x) \mod p_i$ pre $i \in \{1, 2, \ldots, 10\}$ a pošle k R_2 hodnoty $p_1, p_2, \ldots, p_{10}, s_1, s_2, \ldots, s_{10}$
3: R_2 po prijatí $p_1, p_2, \ldots, p_{10}$ a $s_1, s_2, \ldots, s_{10}$ zráta $q_i = Number(y) \mod p_i$ pre $i \in \{1, 2, \ldots, 10\}$ a otestuje či $s_i = q_i$ pre všetky $i \in \{1, 2, \ldots, 10\}$.
4: **if** všade nastane zhoda **then**
5: **return** $x = y$
6: **else**
7: **return** $x \neq y$
8: **end if**

Evidentne počet krokov bude 10-krát väčší ako v prípade `DBComp`, t.j. $20(1 + \lfloor \log_2 n^2 \rfloor) \leq O(\log_2 n)$, čo ale pre $n = 10^{16}$ vyžaduje preniesť 2 560 bitov čo je stále z praktického hľadiska veľmi málo.

Ak $x = y$, tak zrejme platí $Proberror'(x, y) = 0$. Ak $x \neq y$, tak algoritmus vráti zlú odpoveď iba ak všetkých 10 prvočísel bolo zvolených tak, že $s_i = q_i$, čo nastane (všetkých 10 pokusov bolo na sebe nezávislých a náhodných) s pravdepodobnosťou

$$\text{Prob}'_{error}(x, y) = \text{Prob}_{error}(x, y)^{10} \leq \left(\frac{2 \log_2 n}{n}\right)^{10} = \frac{2^{10}(\log_2 n)^{10}}{n^{10}}.$$

Pre $n = 10^{16}$ je to hodnota

$$\text{Prob}'_{error}(x, y) \leq (0,369.10^{-14})^{10} \leq 0,472.10^{-141}.$$

Ak vezmeme do úvahy, že:

- počet mikrosekúnd od Big Bang-u je 24-ciferné číslo (teda rádovo 10^{23}),

- alebo počet protónov vo viditeľnej časti vesmíru je 79-ciferné číslo (teda rádovo 10^{79}),

tak pravdepodobnosť chyby nášho algoritmu by sa voľne dala prirovnať k pravdepodobnosti zázraku.

Pri prenose 10^{16} bitov bezchybne deterministicky by určite s oveľa väčšou pravdepodobnosťou nastalo preklopenie niektorého bitu pri prenose a teda náš algoritmus je v tomto prípade oveľa spoľahlivejší. Treba si totiž uvedomiť, že algoritmus, ktorý zráta výsledok za 10 sekúnd s pravdepodobnosťou chyby 10^{-30} je z praktického hľadiska spoľahlivejší ako deterministický algoritmus, ktorý je teoreticky správny, ale výsledok ráta týždeň.

2.2 Modely pravdepodobnostných algoritmov

Cieľom tejto časti je predstaviť modelovanie pravdepodobnostných algoritmov v zmysle pravdepodobnostného priestoru. Ukážeme si aj ako môže byť koncept náhodnej premennej nápomocný pri analýze pravdepodobnostných algoritmov.

Pravdepodobnostné algoritmy sú špeciálnym typom stochastických algoritmov. Stochastické algoritmy si môžeme predstaviť ako algoritmy, ktoré sú čiastočne ovplyvňované náhodným procesom. Inými slovami, stochastický algoritmus pripúšťa rozhodovanie „hodom mince“ kedykoľvek potrebuje, pričom výsledok hádzania mince je používaný pre rozhodovanie akým sa bude výpočet uberať ďalej. Kvalita stochastického algoritmu je bežne posudzovaná časom jeho behu (efektívnosť) a stupňom spoľahlivosti (korektnosť). Zmysel pojmu „stupeň spoľahlivosti“ môžeme interpretovať rôzne. Napríklad môžeme uvažovať distribúciu pravdepodobnosti nad všetkými možnými vstupmi, a následne merať stupeň spoľahlivosti ako pravdepodobnosť spočítania korektného výsledku pre náhodne zvolený vstup. To znamená, že môžeme akceptovať stochastický algoritmus, ktorý sa „zle“ správa (beží dlho, resp. nekorektne) pre niektoré vstupné inštancie pokiaľ je algoritmus „dobrý“ (efektívny a s vysokou pravdepodobnosťou korektný) pre väčšinu vstupov. Pravdepodobnostné algoritmy sú špeciálnym prípadom stochastických algoritmov, ktoré sa nesprávajú „zle“ pre žiadnu vstupnú inštanciu.

Vzhľadom na to, že požadujeme možnosť efektívneho počítania korektného výstupu s rozumnou pravdepodobnosťou pre každý vstup, je nutné preskúmať správanie pravdepodobnostného algoritmu pre akýkoľvek možný vstup. Teda nemá zmysel uvažovať rozdelenie pravdepodobnosti nad množinou vstupov. Jediným zdrojom náhody, ktorý stojí za pozornosť, je náhodou

ovládaný samotný algoritmus. V nasledujúcom si predstavíme dva modely pravdepodobnostných algoritmov.

2.2.1 Prvý model pravdepodobnostných algoritmov

Prvý model bude jednoduchším spomedzi dvoch prezentovaných modelov. Uvažujme pravdepodobnostný algoritmus A ako rozdelenie pravdepodobnosti nad konečnou kolekciou stratégií v podobe deterministických algoritmov $A_1, \ldots, A_n$.

To znamená, že A pre akýkoľvek vstup w náhodne vyberie A_i a spustí jeho výpočet pre vstup w. Takýmto spôsobom dostaneme n výpočtov algoritmu A pre každý algoritmus z množiny $\{A_1, \ldots, A_n\}$ a pre vstup w. Inými slovami dostaneme n rôznych behov algoritmu A pre vstup w. Činnosť algoritmu A pre vstup w môžeme modelovať pravdepodobnostným priestorom

$$(S_{A,w}, \text{Prob}),$$

kde $S_{A,w} = \{A_1, A_2, \ldots, A_n\}$ a Prob je rovnomerná distribúcia pravdepodobnosti nad $S_{A,w}$.

Vo väčšine prípadov budeme uvažovať

$$S_{A,w} = \{C_1, C_2, \ldots, C_n\}$$

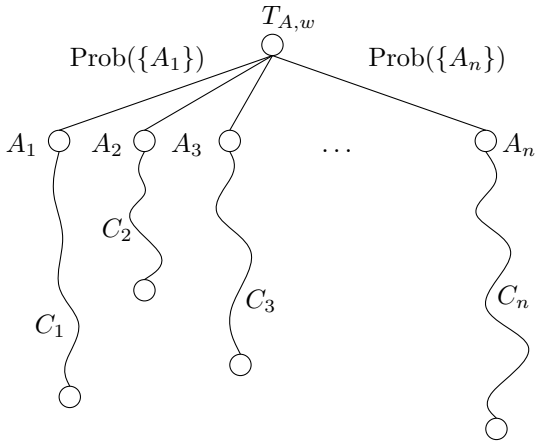
ako množinu všetkých možných výpočtov (behov) algoritmu A pre vstup w. Každý výpočet C_i je určený stratégiou A_i a vstupom w. Teda formálne na tom nezáleží, či uvažujeme stratégie výpočtov, či samotné výpočty.

Na obrázku 2.1 je názorne zobrazená štruktúra algoritmu A, ktorý v počiatku rovnomerne náhodne vyberie deterministickú stratégiu výpočtu. Teda po prvotnom náhodnom výbere je už ďalší proces deterministický.

Tak ako je to znázornené na obrázku, jednotlivé stratégie algoritmu A môžu mať pre vstup w rôznu dĺžku výpočtu. Nech $\text{Time}(C_i)$ označuje dĺžku (časovú zložitosť) výpočtu C_i. Pokiaľ je potrebné skúmať efektívnosť algoritmu A pre vstup w, je možné tak urobiť definovaním náhodnej premennej $Z : S_{A,w} \to \mathbb{N}$ definovanej ako

$$Z(C_i) = \text{Time}(C_i) \text{ pre } i \in \{1, 2, \ldots, n\}.$$

S touto náhodnou premennou je možné definovať efektívnosť práce algoritmu A pre vstup w ako **očakávanú časovú zložitosť** A **pre** w definovanú



Obr. 2.1: Grafické znázornenie prvého modelu pravdepodobnostných algoritmov

ako

$$\begin{aligned}\text{Exp-Time}_A(w) = \mathrm{E}[Z] &= \sum_{i=1}^{n} \text{Prob}(\{C_i\}).Z(C_i) \\ &= \sum_{i=1}^{n} \text{Prob}(\{C_i\}).\text{Time}(C_i).\end{aligned}$$

Keďže sa snažíme stanoviť hornú hranicu očakávanej časovej zložitosti pre akýkoľvek vstup dĺžky n, definujeme očakávanú časovú zložitosť A pre najhorší možný prípad.

Očakávaná časová zložitosť algoritmu A je funkcia

$$\text{Exp-Time}_A : \mathbb{N} \to \mathbb{N},$$

definovaná pre každé $n \in \mathbb{N}$ ako

$$\text{Exp-Time}_A(n) = \max\{\text{Exp-Time}_A(w) | \text{dĺžka } w \text{ je } n\}.$$

Taktiež je možné definovať **časovú zložitosť pravdepodobnostného algoritmu** A ako

$$\text{Time}_A(n) = \max\{\text{Time}(C) | C \text{ je beh } A \text{ pre vstup dĺžky } n\}.$$

Pokiaľ nás zaujíma spoľahlivosť pravdepodobnostného algoritmu A pre vstup w, môžeme definovať indikačnú náhodnú premennú $X : S_{A,w} \to \{0,1\}$ definovanú ako

$$X(C_i) = \begin{cases} 1 & \text{pokiaľ } C_i \text{ vypočíta správny výsledok na } w \\ 0 & \text{pokiaľ } C_i \text{ vypočíta nesprávny výsledok na } w \end{cases}$$

pre všetky $i \in \{1, 2, \ldots, n\}$. Potom

$$\begin{aligned} \mathrm{E}[X] &= \sum_{i=1}^{n} X(C_i).\mathrm{Prob}(\{C_i\}) \\ &= \sum_{X(C_i)=1} 1.\mathrm{Prob}(\{C_i\}) + \sum_{X(C_i)=0} 0.\mathrm{Prob}(\{C_i\}) \\ &= \mathrm{Prob}(\mathrm{Event}(X = 1)). \end{aligned}$$

Je to teda pravdepodobnosť, že A vyráta správny výsledok.

Hodnota $\mathrm{E}[X]$ sa nazýva **pravdepodobnosť úspechu** A **pre** w. Hodnota $1 - \mathrm{E}[X]$ sa nazýva **pravdepodobnosť chyby** A **pre** w a označujeme ju ako $\mathrm{Error}_A(w)$.

Pravdepodobnosť chyby algoritmu A je definovaná ako (najhorší možný prípad)

$$\mathrm{Error}_A(n) = \max\{\mathrm{Error}_A(w) | \text{ dĺžka vstupu } w \text{ je } n\}.$$

Týmto spôsobom Error_A poskytuje garanciu, že pravdepodobnosť chyby je nanajvýš $\mathrm{Error}_A(n)$ pre každý vstup dĺžky n.

Všimnime si, že naša definícia časovej zložitosti predpokladá, že všetky behy algoritmu A sú konečné. Vo všeobecnosti, pravdepodobnostné algoritmy sú schopné mať nekonečné výpočty. V takom prípade meriame očakávanú časovú zložitosť iba cez konečné výpočty A. Nekonečné výpočty sú považované za chybné a pravdepodobnosť ich výskytu sa priraďuje k pravdepodobnosti chyby algoritmu A.

Na nasledujúcich príkladoch si ukážeme ako možno pomocou predošlých formalizmov modelovať a analyzovať pravdepodobnostné algoritmy.

Príklad 2.2.1. *Uvažujme pravdepodobnostný protokol z motivačného príkladu z podkapitoly 2.1 na porovnávanie dvoch reťazcov x a y dĺžky n. Na začiatku protokol R náhodne vyberie prvočíslo p z množiny* $\mathrm{PRIM}(n^2)$ *s rovnomerným rozdelením pravdepodobnosti. Ak C_p predstavuje beh protokolu R daný prvočíslom p pre vstup (x, y), tak je možné danú situáciu modelovať pravdepodobnostným priestorom* $(S_{R,(x,y)}, \mathrm{Prob})$ *kde*

- $S_{R,(x,y)} = \{C_p | p \in \mathrm{PRIM}(n^2)\}$, *a*
- Prob *je rovnomerná distribúcia pravdepodobnosti na* $S_{R,(x,y)}$.

Keďže každý beh C_p *je jednoznačne určený prvočíslom* $p \in \mathrm{PRIM}(n^2)$, *tak je možné náš pravdepodobnostný priestor, ktorým modelujeme činnosť protokolu* R *so vstupom* (x, y) *vnímať ako dvojicu* $(\mathrm{PRIM}(n^2), \mathrm{Prob})$.

Všetky behy protokolu R *majú rovnakú komunikačnú zložitosť* $4\lceil \log_2 n \rceil$.

Zjavne sa nemusíme zaoberať prípadom keď $x = y$, *pretože pravdepodobnosť chyby je v tomto prípade rovná 0. Pravdepodobnosť chyby v prípade keď* $x \neq y$ *určíme pomocou indikačnej náhodnej premennej* X *definovanej ako*

$$X(C_p) = \begin{cases} 1 & \text{ak } p \text{ je dobré pre } (x,y) \\ 0 & \text{ak } p \text{ je zlé pre } (x,y). \end{cases}$$

Pretože

$$\mathrm{Prob}(\{C_p\}) = \frac{1}{|\mathrm{PRIM}(n^2)|} \text{for all } p \in \mathrm{PRIM}(n^2)$$

a v predchádzajúcom sme dokázali, že počet zlých prvočísel pre vstup (x, y) *spĺňajúci* $x \neq y$ *nie je viac ako* $n - 1$, *dostávame*

$$\begin{aligned} \mathrm{E}[X] &= \sum_{p \in \mathrm{PRIM}(n^2)} X(C_p).\mathrm{Prob}(\{C_p\}) \\ &= \sum_{p \in \mathrm{PRIM}(n^2)} X(C_p) \cdot \frac{1}{|\mathrm{PRIM}(n^2)|} \\ &= \frac{1}{|\mathrm{PRIM}(n^2)|} \cdot \sum_{p \text{ je dobré}} 1 \\ &\geq \frac{1}{|\mathrm{PRIM}(n^2)|} \cdot (|\mathrm{PRIM}(n^2)| - (n-1)) \\ &= 1 - \frac{n-1}{|\mathrm{PRIM}(n^2)|}. \end{aligned}$$

Týmto spôsobom vyjadríme pravdepodobnosť chyby protokolu R *pre* (x, y) *ako*

$$\begin{aligned} \mathrm{Error}_R((x,y)) = 1 - \mathrm{E}[X] \leq \\ \leq \frac{n-1}{|\mathrm{PRIM}(n^2)|} \leq \frac{1.\ln n}{n}, \end{aligned}$$

pre každé $n \geq 9$, *pretože v takom prípade je* $|\mathrm{PRIM}(n^2)| \geq \frac{n^2}{\ln n^2}$.

Uvažujme teraz modifikovaný protokol R_2 taký, že na začiatku vyberá náhodne dve prvočísla p a q, ktoré akceptujú vstup iba ak

$$(x \mod p = y \mod p) \text{ a zároveň } (x \mod q = y \mod q).$$

Výpočet $C_{p,q}$ protokolu R_2 pozostáva v posielaní sady $\langle p, q, (x \mod p), (x \mod q)\rangle$ od R_{I} k R_{II} a porovnávaním zvyškov $\mod p$ a $\mod q$. Na analýzu pravdepodobnosti chyby protokolu R_2 definujme indikačnú náhodnú premennú

$$Y(C_{p,q}) = \begin{cases} 1 & p \text{ alebo } q \text{ sú dobré pre vstup } (x,y) \\ 0 & p \text{ a zároveň } q \text{ sú zlé pre vstup } (x,y) \end{cases}$$

pre všetky $C_{p,q} \in S_{R_2,(x,y)} = \{C_{r,s} | r, s \in \mathrm{PRIM}(n^2)\}$. Analýzou v pravdepodobnostnom priestore $(S_{R_2,(x,y)}, Prob_2)$ dostávame

$$\begin{aligned}
\mathrm{E}[Y] &= \sum_{C_{p,q} \in S_{R_2,(x,y)}} Y(C_{p,q}).\mathrm{Prob}(\{C_{p,q}\}) \\
&= \sum_{p,q \in \mathrm{PRIM}(n^2)} Y(C_{p,q}) \cdot \frac{1}{|\mathrm{PRIM}(n^2)|^2} \\
&= \mathrm{Prob}(\mathrm{Event}(Y = 1)) \\
&= 1 - \mathrm{Prob}(\mathrm{Event}(Y = 0)) \\
&= 1 - \mathrm{Prob}(\mathrm{Event}(p \text{ a } q \text{ sú zlé pre } (x,y))) \\
&= 1 - \mathrm{Prob}(\mathrm{Event}(p \text{ je zlé pre } (x,y))).\mathrm{Prob}(\mathrm{Event}(q \text{ je zlé pre } (x,y))) \\
&\{p \text{ aj } q \text{ boli vyberané nezávisle na sebe}\} \\
&\geq 1 - \left(\frac{n-1}{|\mathrm{PRIM}(n^2)|}\right)^2 \\
&\geq 1 - \frac{4 \ln^2 n}{n^2}
\end{aligned}$$

pre všetky dostatočne veľké n. Pokiaľ $\mathrm{E}[Y]$ predstavuje pravdepodobnosť úspechu výpočtu protokolu R_2 na vstupe (x,y) pre $x \neq y$ (inými slovami, že udalosť $\mathrm{Event}(Y = 1)$ predstavuje skutočnosť, že R_2 vráti správnu odpoveď), potom pravdepodobnosť chyby protokolu R_2 pre vstup (x,y) je

$$\mathrm{Error}_{R_2}((x,y)) = 1 - \mathrm{E}[Y] \leq \frac{4 \ln^2 n}{n^2}.$$

2.2.2 Problém maximálnej splniteľnosti boolovskej formuly

Uvažujme optimalizačný problém MAX-SAT, ktorý je sformulovaný nasledovne: Nech $X = \{x_1, x_2, \ldots, x_n\}$ je množina premenných typu Boolean. Množina všetkých literálov nad X je množina $Lit_X = \{x, \overline{x} | x \in X\}$, kde $\overline{x}$ je negácia x. Klauzula je akákoľvek konečná disjunkcia literálov (napr. $x_1 \vee x_2 \vee \overline{x_4} \vee x_7$). Booleovská formula F je v konjunktívnom normálnom tvare (CNF) ak Φ je konečná konjunkcia klauzúl. Formula Φ je v k-konjunktívnom normálnom tvare (kCNF) pre kladné prirodzené číslo k ak Φ je v CNF a navyše každá klauzula pozostáva z najviac k literálov. Tento problém sa anglicky nazýva Maximum Satisfability Problem (MAX-SAT) a ide o problém nájdenia ohodnotenia premenných formuly v CNF tak, aby počet splnených formúl bol maximálny.

Algoritmus 3 `RSAM`

Require: formula $\Phi = F_1 \wedge F_2 \wedge \cdots \wedge F_m$ v CNF nad množinou premenných $\{x_1, x_2, \ldots, x_n\}$ typu Boolean

1: vyber náhodne ohodnotenie premenných $(\alpha_1, \alpha_2, \ldots, \alpha_n) \in \{0,1\}^n$, pričom $\text{Prob}(\alpha_i = 1) = \text{Prob}(\alpha_i = 0) = \frac{1}{2}$ pre všetky $i \in \{1, 2, \ldots, n\}$

2: **return** $(\alpha_1, \alpha_2, \ldots, \alpha_n)$

Tento problém je NP-ťažký. Zoslabíme požiadavku nájdenia optimálneho riešenia na nájdenie dostatočne dobrého riešenia s vysokou pravdepodobnosťou. To nám už umožňuje navrhnúť nasledovný jednoduchý a efektívny algoritmus náhodného ohodnocovania (*angl.* Random Sampling - `RSAM`).

Je vidieť, že algoritmus `RSAM` jednoducho náhodne generuje priradenia premenným formuly Φ a vráti dané ohodnotenie ako výstup. Keďže každý algoritmus pre MAX-SAT musí produkovať vhodné riešenie danej dĺžky, asymptoticky rýchlejší algoritmus neexistuje.

Každý výstup algoritmu `RSAM` je ohodnotenie premenných formuly Φ a teda je to vhodné riešenie pre Φ ako inštanciu problému. V tomto prípade teda nemá zmysel uvažovať nad pravdepodobnosťou chyby.

Čo si budeme všímať a vyhodnocovať je tzv. dokonalosť daného algoritmu ako podiel počtu splnených klauzúl a počtu všetkých. Na to budeme potrebovať pre vstup $\Phi = F_1 \wedge F_2 \wedge \cdots \wedge F_m$, m indikačných náhodných premenných $Z_1, Z_2, \ldots, Z_m$ takých, že

$$Z_i(\alpha) = \begin{cases} 1 & \text{ak klauzula } F_i \text{ je splnená ohodnotením } \alpha \\ 0 & \text{ak klauzula } F_i \text{ nie je splnená ohodnotením } \alpha \end{cases}$$

pre každé $\alpha \in \{0,1\}^n$. V pravdepodobnostnom priestore

$$(\{0,1\}^n \text{Prob})$$

uvažujme náhodnú premennú

$$Z = \sum_{i=1}^{m} Z_i,$$

ktorá určuje počet splnených klauzúl. Zase nás bude zaujímať stredná hodnota $\mathrm{E}[Z]$. Z linearity strednej hodnoty dostávame

$$\mathrm{E}[Z] = \mathrm{E}\left[\sum_{i=1}^{m} Z_i\right] = \sum_{i=1}^{m} \mathrm{E}[Z_i].$$

Na dokončenie analýzy budeme potrebovať odhad $\mathrm{E}[Z_i]$, t.j. pravdepodobnosti, že $F_i = l_{i1} \vee l_{i2} \vee \cdots \vee l_{ik}$ bude splnená. Zrejme klauzula F_i nebude splnená iba ak všetkých k literálov nebude splnených. Pravdepodobnosť, že literál nie je splnený je $\frac{1}{2}$. Pokiaľ sú premenné ohodnocované náhodne a nezávisle, tak pravdepodobnosť, že ani jeden z literálov klauzuly F_i nebude splnený je $\left(\frac{1}{2}\right)^k$. Teda pravdepodobnosť splnenia klauzuly F_i je

$$\mathrm{E}[Z_i] = 1 - \frac{1}{2^k}.$$

Pokiaľ ale každá klauzula má nanajvýš jeden literál, tak pre všetky $i \in \{1, 2, \ldots, m\}$ dostávame dolný odhad

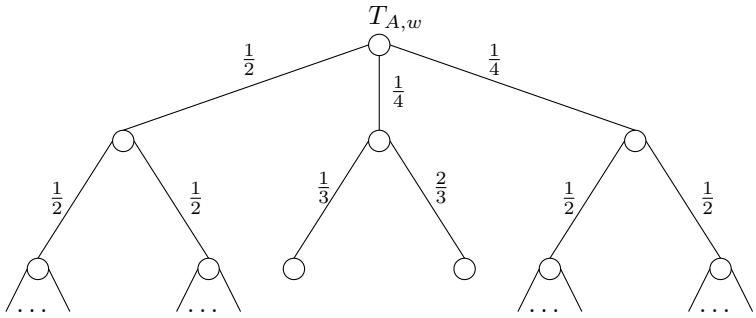
$$E[Z_i] \geq \frac{1}{2}.$$

Teda

$$\mathrm{E}[Z] = \sum_{i=1}^{m} \mathrm{E}[Z_i] \geq \sum_{i=1}^{m} \frac{1}{2} = \frac{m}{2},$$

čo znamená, že pri náhodnom ohodnotení premenných formuly Φ dostaneme aspoň polovicu splnených klauzúl.

Tento odhad je ale príliš hrubý. Mnoho z klauzúl má viac ako jeden literál. Pokiaľ predpokladáme, že každá klauzula má aspoň tri literály, tak dostávame $\mathrm{E}[Z_i] \geq \frac{7}{8}$ pre všetky $i \in \{1, 2, \ldots, m\}$. Z toho teda môžeme vyvodiť záver, že sedem osmín všetkých klauzúl formuly bude splnených.



Obr. 2.2: Grafické znázornenie druhého modelu pravdepodobnostných algoritmov

2.2.3 Druhý model pravdepodobnostných algoritmov

Niekedy je prirodzenejšie predstaviť si pravdepodobnostný algoritmus ako nedeterministický algoritmus s distribúciou pravdepodobnosti pri každom nedeterministickom výbere (kroku) algoritmu. Všetky možné výpočty takého algoritmu si môžeme prestaviť ako výpočtový strom $T_{A,w}$ algoritmu A na vstupe w (obr. 2.2). Každá cesta v tomto strome od koreňa k listom predstavuje jeden konkrétny výpočet A na w. Každá hrana je označená číslom z $[0, 1]$, čo predstavuje pravdepodobnosť výberu daného kroku algoritmu, ktorý táto hrana predstavuje. Ak vezmeme výberový priestor $S_{A,w}$ ako množinu všetkých možných výpočtov A na w, tak môžeme uvažovať pravdepodobnostný priestor $(S_{A,w}, \text{Prob})$, kde $\text{Prob}(C)$ ľubovoľného výpočtu $C \in S_{A,w}$ predstavuje pravdepodobnosť javu, že algoritmus A pri vstupe w vyprodukuje práve výpočet C. $\text{Prob}(C)$ je potom možné vypočítať ako súčin všetkých pravdepodobností pri jednotlivých nedeterministických krokoch, ktorými sa algoritmus počas daného výpočtu vybral. Inými slovami, je to súčin všetkých hodnôt na hranách v strome na konkrétnej ceste, ktorá daný výpočet predstavuje.

2.2.4 Problém triedenia

Uvažujme teraz problém triedenia lineárne usporiadanej množiny Aa predstavme si pravdepodobnostnú verziu dobre známeho triediaceho algoritmu Quicksort (`RQS`).

Algoritmus `RQS` skončí prácu vždy s usporiadanou postupnosťou prvkov množiny A. Teda pravdepodobnosť chyby pre každý vstup je vždy rovná 0.

Na druhej strane môžeme sledovať rôzne zložitosti rôznych výpočtov

Algoritmus 4 `RQS`

Require: množina A
1: **if** $A = \{b\}$ **then**
2: **return** b
3: **else**
4: vyber náhodne $a \in A$
5: **end if**
6: $A_{\leq a} := \{b \in A | b \leq a\}$
7: $A_{>a} := \{b \in A | b > a\}$
8: **return** $\mathrm{RQS}(A_{\leq a}), a, \mathrm{RQS}(A_{>a})$

v závislosti od vykonaných porovnaní medzi prvkami vstupnej množiny. V kroku 6 a kroku 7 algoritmus `RQS` vykoná $|A| - 1$ porovnaní, pretože každý prvok množiny $A \setminus \{a\}$ je porovnaný s pivotom a. Všimnime si, že `RQS` má exponenciálne veľa rôznych výpočtov. Ak `RQS` vyberá za pivot stále najmenší alebo najväčší prvok vstupnej množiny, tak každý výsledný výpočet bude pozostávať z $|A| - 1$ rekurzívnych volaní a počet porovnaní bude

$$\sum_{i=0}^{n} i = \frac{n.(n-1)}{2} \in O(n^2).$$

Pokiaľ bude `RQS` stále vyberať za pivota medián vstupnej množiny, tak klasickou analýzou známej stratégie zvanej „rozdeľuj a panuj“ dostávame rekurentný vzťah

$$\mathrm{Time}_{\texttt{RQS}}(n) \leq 2.\mathrm{Time}_{\texttt{RQS}}\left(\frac{n}{2}\right) + n - 1,$$

ktorého riešením je $\mathrm{Time}_{\texttt{RQS}}(n) \in O(n \log n)$. Takýto výpočet má iba $\log_2 n$ rekurzívnych volaní.

V nasledovnom budeme analyzovať očakávanú zložitosť `RQS`. Hĺbka výpočtového stromu sa pohybuje v rozmedzí od $\log n$ po $n-1$ a s tým súvisiace pravdepodobnosti jednotlivých výpočtov sa veľmi rôznia. Ak uvažujeme pravdepodobnostný priestor

$$(S_{\texttt{RQS}(A)}, \mathrm{Prob})$$

a zvolíme indikačnú náhodnú premennú

$$X(C) = \text{ počet porovnaní počas chodu výpočtu } C$$

pre akýkoľvek výpočet $C \in S_{\texttt{RQS}(A)}$, tak je možné odhadnúť očakávanú zložitosť `RQS` ako E$[X]$.

Nech $s_1, s_2, \ldots, s_n$ je utriedená postupnosť - výstup `RQS`. Teda s_i je i-ty najmenší prvok množiny A. Definujme indikačnú náhodnú premennú

$$X_{ij}(C) = \begin{cases} 1 & \text{ak počas behu výpočtu } C \text{ boli } s_i \text{ a } s_j \text{ porovnané} \\ 0 & \textit{opak.} \end{cases}$$

Pre každé $i, j \in \{1, 2, \ldots, n\}$, $i < j$ je potom náhodná premenná X definovaná ako

$$X(C) = \sum_{i=1}^{n} \sum_{j>i} X_{ij}(C).$$

Táto premenná zodpovedá celkovému počtu porovnaní a teda

$$\mathrm{E}[X] = \text{Exp-Time}_{\texttt{RQS}}(A).$$

Kvôli linearite strednej hodnoty náhodnej premennej máme

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n} \sum_{j>i} X_{ij}\right] = \sum_{i=1}^{n} \sum_{j>i} \mathrm{E}\left[X_{ij}\right].$$

Pretože chceme odhadnúť hodnotu E$[X]$, musíme odhadnúť E$[X_{ij}]$ pre všetky $i, j \in \{1, 2, \ldots, n\}$, $i < j$. Nech p_{ij} označuje pravdepodobnosť javu, že s_i a s_j boli porovnané počas vykonávania algoritmu `RQS`. Potom platí

$$\mathrm{E}[X_{ij}] = p_{ij}.1 + (1 - p_{ij}).0 = p_{ij},$$

čo znamená, že očakávaná hodnota náhodnej premennej E$[X_{ij}]$ je rovná hodnote p_{ij}. Zostáva nám ešte odhadnúť hodnotu p_{ij}.

Uvedomme si v akom prípade sú prvky s_i a s_j porovnávané. Na to aby boli prvky s_i a s_j porovnané je nutné, aby prvok s_i alebo s_j bol vybraný za pivota.

Pokiaľ je za pivota vybraný akýkoľvek prvok z množiny $\{s_1, \ldots, s_{i-1}\}$ alebo $\{s_{j+1}, \ldots, s_n\}$, potom oba prvky s_i a s_j padnú spolu do množiny $A_>$ resp. $A_<$ a teda na to, či spomínané prvku budú neskôr porovnané alebo nie, to nemá žiaden vplyv.

Ak bude za pivota vybraný akýkoľvek prvok z množiny $\{s_{i+1}, \ldots, s_{j-1}\}$, tak v tomto prípade by prvok s_i padol do množiny $A_<$ a prvok s_j do množiny $A_>$ a už nikdy by dané prvky nemohli byť porovnávané. Teda pre výber

pivota je možné uvažovať iba množinu $\{s_i, s_j\}$ oproti množine $\{s_i, \ldots, s_j\}$. Z toho plynie, že pravdepodobnosť porovnania prvkov s_i a s_j je

$$p_{ij} = \frac{|\{s_i, s_j\}|}{|\{s_i, \ldots, s_j\}|} = \frac{2}{j-i+1}.$$

Pre strednú hodnotu potom máme

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n} \sum_{j>i} p_{ij} = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} \leq \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \\
&\quad \text{substitúciou } k = j-i+1 \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{1}{k} = 2 \sum_{i=1}^{n} \mathrm{Har}(n) = 2n.\mathrm{Har}(n) \\
&= 2n \ln n + \Theta(n).
\end{aligned}$$

Ukázali sme, že

$$\text{Exp-Time}_{\texttt{RQS}}(n) \in O(n \log n)$$

a teda, že `RQS` je efektívny triediaci algoritmus.

2.2.5 Problém nájdenia k-tého najmenšieho prvku

Uvažujme teraz nasledovný problém: Je daná množina A rôznych prvkov s lineárnym usporiadaním a prirodzené číslo $k \leq |A|$. Našou úlohou je nájsť k-tý najmenší prvok množiny A. Tento problém je možné riešiť usporiadaním a potom nájdením k-teho prvku. Vieme ale, že časová zložitosť triedenia je $\Theta(n \log n)$. Nie je ťažké vidieť,že pravdepodobnostným algoritmom `RSEL` môžeme dosiahnuť lepšiu časovú zložitosť.

Je zrejmé, že algoritmus `RSEL` skončí vždy s korektnou odpoveďou pri každom vstupe. Nie je ťažké odvodiť, že $\text{Exp-Time}_{\texttt{RSEL}}(A, k) \in O(n)$.

2.3 Klasifikácia pravdepodobnostných algoritmov

Náplňou tejto kapitoly bude predstaviť základnú a všeobecne známu klasifikáciu pravdepodobnostných algoritmov. Táto klasifikácia je založená na hodnote pravdepodobnosti chyby a nemá nič do činenia s návrhom algoritmu. Pravdepodobnosť chyby teda môžeme vnímať ako mieru použiteľnosti navrhovaného algoritmu a uvidíme, že hlavným dôvodom klasifikácie nie je iba meranie absolútnej pravdepodobnosti chyby, ale hlavne vnímanie

Algoritmus 5 `RSEL`

Require: množina $A = \{a_1, a_2, ..., a_n\}, n \in \mathbb{N} \setminus \{0\}$ a $k \in [1, n]$

if $A = \{a_1\}$ **then**
  return a_1
else
  vyber náhodne $i \in \{1, 2, ..., n\}$
end if
$A_< := \{b \in A | b < a_i\}$
$A_> := \{b \in A | b > a_i\}$
if $|A_<| \geq k$ **then**
  return RSEL$(A_<, k)$
end if
if $|A_<| = k - 1$ **then**
  return a_i
else
  return RSEL$(A_>, k - (|A_<| + 1))$
end if

počtu opakovaní nezávislých behov algoritmu pre ten istý vstup za účelom dostatočného zníženia pravdepodobnosti zlého výstupu. Inými slovami, klasifikácia poukazuje na rýchlosť zníženia pravdepodobnosti chyby s počtom opakovaní.

Klasifikácia pravdepodobnostných algoritmov nie je rovnaká pre všetky výpočtové úlohy. Význam slov „nesprávny výstup“ sa môže za rôznych okolností líšiť. Pri riešení rozhodovacej úlohy alebo pri počítaní nejakej funkcie sa jedná práve o pravdepodobnosť nesprávneho výstupu. Pri optimalizačných úlohách nepovažujeme neoptimálne riešenie za nechcený výsledok resp. dokonca za chybu. Možný výsledok, ktorého kvalita sa veľmi nevzďaľuje od optima môže byť veľmi cenný. Takýmito prípadmi sa budeme zaoberať v nasledovnej kapitole.

2.4 Algoritmy typu Las Vegas

Algoritmy typu Las Vegas sú pravdepodobnostné algoritmy, ktoré garantujú vždy správny výsledok. To znamená, že nesprávne výsledky (výstupy) sú v tomto prípade neprípustné. V literatúre je možné nájsť rôzne modely Las Vegas algoritmov líšiacich sa v tom či pripúšťajú odpoveď „neviem“ (to znamená, že algoritmus nevie vypočítať správny výsledok).

Najskôr vezmime v úvahu iba prípad kedy „neviem“ ako odpoveď nie

je prípustná. Označením $A(x)$ budeme označovať výstup algoritmu A pre vstup x.

Definícia 2.4.1 (LV1)**.** *Pravdepodobnostný algoritmus A nazývame* ***Las Vegas algoritmus*** *počítajúci funkciu F ak pre každý vstup x platí*

$$\text{Prob}(A(x) = F(x)) = 1.$$

Pre takéto Las Vegas algoritmy vyšetrujeme vždy očakávanú zložitosť (napríklad očakávanú časovú zložitosť Exp-Time$_A(n)$). Je potrebné predpokladať, že výpočty algoritmu pre nejaký vstup sú rôznych dĺžok. Ak by neboli, tak by sme vzali akýkoľvek z nich a prehlásili ho za deterministické riešenie, ktoré by bolo navyše efektívne (keďže všetky by v tom prípade boli efektívne).

Ilustratívne príklady Las Vegas algoritmov sú algoritmy RQS a RSEL. V oboch prípadoch sú značné rozdiely medzi zložitosťou najhoršieho možného prípadu a očakávanou zložitosťou. Zásadným faktom ale ostáva veľká blízkosť očakávanej zložitosti a najefektívnejšieho výpočtu daného pravdepodobnostného algoritmu.

Prejdime teraz k druhej definícii, pričom symbolom „?“ budeme označovať výstup „neviem“.

Definícia 2.4.2 (LV?)**.** *Nech A je pravdepodobnostný algoritmus, ktorý pripúšťa odpoveď „neviem“. Vravíme, že A je* ***Las Vegas algoritmus*** *pre funkciu F, ak pre každý vstup x platí*

(i) $\text{Prob}(A(x) = F(x)) \geq \frac{1}{2}$

(ii) $\text{Prob}(A(x) = ?) = 1 - \text{Prob}(A(x) = F(x)) \leq \frac{1}{2}$.

Z druhej podmienky vidíme, že pravdepodobnosť nesprávneho výsledku sa v definícii vôbec nevyskytuje, buď dostaneme správny výsledok, alebo odpoveď „neviem“. V prvej podmienke stojí, že pravdepodobnosť správneho výsledku je aspoň jedna polovica. Spomínaná konštanta nie je dôležitá. Definícia by fungovala s akýmkoľvek $0 < \epsilon < 1$. Dôvodom tohto tvrdenia je fakt, že pre akékoľvek $\epsilon \leq \frac{1}{2}$ je možné zvýšiť pravdepodobnosť úspechu nad hranicu $\frac{1}{2}$ a to opakovaním algoritmu pre ten istý vstup konštantne veľa krát. (Ak budeme algoritmus A opakovať k-krát, tak výstup „neviem“ prehlásime, iba ak všetkých k behov skončí s odpoveďou „neviem“. Pokiaľ aspoň raz algoritmus vráti korektnú odpoveď, tak ju dáme na výstup.)

V nasledujúcom si ukážeme ekvivalenciu medzi obomi definíciami algoritmov Las Vegas.

Lema 2.4.3. LV? $\rightarrow$ LV1

Dôkaz. Nech A je Las Vegas algoritmus typu LV?, kde odpoveď „neviem“ dostávame s ohraničenou pravdepodobnosťou. Modifikujeme algoritmus A na algoritmus A' tým, že ho budeme spúšťať znova pre ten istý vstup vždy, keď pôvodný algoritmus A skončí s výsledkom „neviem“. Ak $T_{A,w}$ je výpočtový strom algoritmu A pre vstup w, potom výsledkom spomínanej konštrukcie je strom $T_{A',w}$ všetkých výpočtov algoritmu A' pre w, kde na všetkých listoch s odpoveďou „?“ je zavesený strom $T_{A,w}$. Výsledkom je teda nový algoritmus A', ktorý teoreticky pripúšťa aj nekonečný výpočet. Na druhej strane, ale ak sa výpočet dopracuje k výsledku, tak máme istotu, že je správny.

Aká je teda pravdepodobnosť, že A' zastaví? Táto pravdepodobnosť je vysoká, pretože sa približuje k 1 rastúcou dĺžkou výpočtu. Ako je to možné? Nech $\mathrm{Time}_A(w)$ je najhoršia časová zložitosť A, teda hĺbka stromu $T_{A,w}$. To znamená, že pravdepodobnosť, že A' zastaví v čase $\mathrm{Time}_A(w)$ je aspoň $1/2$. Pravdepodobnosť, že A' zastaví v čase $2.\mathrm{Time}_A(w)$ je aspoň $3/4$, pretože A' začína nové behy A pre vstup w pre každý list s odpoveďou „?“. To znamená, že po čase $k.\mathrm{Time}_A(w)$ algoritmus A' zráta správny výsledok s pravdepodobnosťou aspoň

$$1 - \frac{1}{2^k},$$

pretože 2^{-k} je horná hranica pravdepodobnosti vrátenia výsledku „?“ v k nezávislých behoch algoritmu A pre w.

Aká veľká je očakávaná hodnota časovej zložitosti $\text{Exp-Time}_{A'}(n)$ pre A'? V nasledovnom si ukážeme, že

$$\text{Exp-Time}_{A'}(n) \in O(\mathrm{Time}_A(n)).$$

Bez ujmy na všeobecnosti môžeme predpokladať, že všetky výpočty A, ktoré končia „?“ majú maximálnu dĺžku $\mathrm{Time}_A(w)$, a teda všetky takéto výpočty majú rovnakú dĺžku. Nech pre všetky $i \in \mathbb{N} - \{0\}$ je

$$Set_i = \{C \in S_{A',w} | (i-1).\mathrm{Time}_A(w) < \mathrm{Time}(C) \leq i.t\mathrm{Time}_A(w)\}$$

množina všetkých výpočtov, ktoré končia práve v i-tom behu A. Potom je $S_{A',w} = \bigcup_{i=0}^{\infty} Set_i$ a $Set_r \cap Set_s = \emptyset$ pre $r \neq s$. V predchádzajúcom sme pozorovali, že

$$\sum_{C \in \bigcup_{i=1}^{k} Set_i} \mathrm{Prob}(\{C\}) \geq 1 - \frac{1}{2^k}$$

pre každé $k \in \mathbb{N} - \{0\}$. Priamym dôsledkom tohoto faktu je, že

$$\sum_{C \in Set_i} \text{Prob}(\{C\}) \leq \frac{1}{2^{i-1}}$$

pre všetky $i \geq i$.

Na základe toho dostávame

$$\begin{aligned}
\text{Exp-Time}_{A'}(n) &= \sum_{C \in S_{A',w}} \text{Time}_{A'}(C).\text{Prob}(\{C\}) \\
&= \sum_{i=1}^{\infty} \sum_{C \in Set_i} \text{Time}_{A'}(C).\text{Prob}(\{C\}) \\
&\leq \sum_{i=1}^{\infty} \sum_{C \in Set_i} i.\text{Time}_A(w).\text{Prob}(\{C\}) \\
&\quad \{\text{pokiaľ Time}_{A'}(C) \leq i.\text{Time}_A(w) \text{ pre}\} \\
&\quad \text{každý výpočet } C \in Set_i\} \\
&= \sum_{i=1}^{\infty} i.\text{Time}_A(w). \sum_{C \in Set_i} \text{Prob}(\{C\}) \\
&\leq \sum_{i=1}^{\infty} i.\text{Time}_A(w) \cdot \frac{1}{2^{i-1}} \\
&= \text{Time}_A(w). \sum_{i=1}^{\infty} \frac{i}{2^{i-1}} \\
&= \text{Time}_A(w).2 \sum_{i=1}^{\infty} \frac{i}{2^i} \\
&= 2.\text{Time}_A(w). \left(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \sum_{i=5}^{\infty} \frac{i}{2^i} \right) \\
&< 6.\text{Time}_A(w)
\end{aligned}$$

pretože $\sum_{i=5}^{\infty} \frac{i}{2^i} < 1$. □

Dokážeme teraz aj opačnú implikáciu.

Lema 2.4.4. LV1 → LV?

Dôkaz. Teraz je našou snahou modifikovať algoritmus typu Las Vegas, ktorý dáva stále správnu odpoveď na typ, ktorý pripúšťa i odpoveď „neviem“. Uvažujme výpočtový strom $T_{A,w}$, ktorý obsahuje veľa krátkych (efektívnych), ale aj zopár dlhých (možno aj nekonečných) výpočtov. Napríklad krátke výpočty môžu prebiehať v lineárnom čase, pričom dlhé výpočty v kubickom čase. V prípade, že výpočet prebieha príliš dlho je možné ho stopnúť a vyhlásiť „neviem“ ako odpoveď. Otázka je akú hranicu pre zastavenie výpočtu zvoliť. V nasledujúcom si ukážeme, že hranica

$$2.\text{Exp-Time}_A(w)$$

je vhodná horná hranica pre zastavenie výpočtu. Dokážeme to sporom. Predpokladajme, že

$$\text{Prob}(B(w) =?) > \frac{1}{2}.$$

Označme

$$S_{A,w}(?) = \{C \in S_{A,w} | \text{Time}(C) > 2.\text{Exp-Time}_A(w)\} \subset S_{A,w}.$$

Potom je možné predpoklad preformulovať na

$$\sum_{C \in S_{A,w}(?)} \text{Prob}(C) > \frac{1}{2}.$$

Platí teda, že

$$\text{Time}(C) \geq 2.\text{Exp-Time}_A(w) + 1 \text{ pre všetky } C \in S_{A,w}(?).$$

Označme taktiež množinu krátkych výpočtov

$$S_{A,w}(F(w)) = S_{A,w} - S_{A,w}(?).$$

Z toho plynie, že

$$\begin{aligned} \text{Exp-Time}_A(w) &= \sum_{C \in S_{A,w}} \text{Time}_A(C).\text{Prob}(\{C\}) \\ &= \sum_{C \in S_{A,w}(?)} \text{Time}_A(C).\text{Prob}(\{C\}) \\ &+ \sum_{C \in S_{A,w}(F(w))} \text{Time}_A(C).\text{Prob}(\{C\}) \end{aligned}$$

$$
\begin{aligned}
&> \sum_{C \in S_{A,w}(?)} (2.\text{Exp-Time}_A(w) + 1).\text{Prob}(\{C\}) + 0 \\
&= (2.\text{Exp-Time}_A(w) + 1) \sum_{C \in S_{A,w}(?)} \text{Prob}(\{C\}) \\
&> (2.\text{Exp-Time}_A(w) + 1) \cdot \frac{1}{2} \\
&= \text{Exp-Time}_A(w) + \frac{1}{2}.
\end{aligned}
$$

Dostali sme, že $\text{Exp-Time}_A(w) > \text{Exp-Time}_A(w) + \frac{1}{2}$, čo je spor a teda platí negácia predpokladu. Platí teda, že $\text{Prob}(B(w) =?) < \frac{1}{2}$, čo znamená, že náš nový algoritmus je typu LV?.

□

2.5 Algoritmy typu Monte Carlo

Na rozdiel od algoritmov typu Las Vegas, u algoritmov typu Monte Carlo pripúšťame aj chybnú odpoveď. Prirodzenou požiadavkou však je, aby sme veľkosť chyby mali pod istou kontrolou. Takisto existujú situácie, keď isté typy chýb chceme vylúčiť alebo aspoň obmedziť. Ilustrovať sa to dá na príklade chybného stanovenia diagnózy lekárom. Z hľadiska dopadu na pacienta je väčšinou horšie, ak lekár nesprávne diagnostikuje chorého pacienta v porovnaní s prípadom, že zdravého pacienta nesprávne označí za chorého. Z toho dôvodu sa rozlišuje viacero druhov algoritmov typu Monte Carlo.

2.5.1 Algoritmy typu Monte Carlo s jednostrannou chybou

Tento typ pravdepodobnostných algoritmov sa používa na tzv. rozhodovacie problémy.

Majme daný rozhodovací problém dvojicou (Σ, L), kde Σ je ľubovoľná abeceda a $L \subseteq \Sigma$ jazyk nad abecedou Σ. Potom pre akýkoľvek reťazec $x \in \Sigma^*$ je treba rozhodnúť, či x je z jazyka L. To znamená, pre každé $x \in \Sigma^* - L$ vyžadujeme, aby jednostranne chybový algoritmus poskytol správnu odpoveď „0“, pokiaľ $x \notin L$ s istotou a pochybiť môže pre vstupy z jazyka L len s obmedzenou pravdepodobnosťou. Tento koncept môžeme sformalizovať.

Definícia 2.5.1. *Nech algoritmus A je pravdepodobnostný algoritmus a nech (Σ, L) je rozhodovací problém. Hovoríme, že algoritmus A je* ***typu Monte Carlo s jednostrannou chybou*** *pre L (ozn. 1MC), ak platí:*

(i) pre každé $x \in L : \text{Prob}(A(x) = 1) \geq \frac{1}{2}$,

(ii) pre každé $x \notin L : \text{Prob}(A(x) = 0) = 1$.

Príkladom je pravdepodobnostný R– protokol. Pokiaľ $\Sigma \in \{0,1\}$ a $L_{\neq} = \{(x,y)|(x,y) \in \{0,1\}^n : x \neq y, n \in \mathbb{N}\}$, navrhnutý R– protokol akceptuje jazyk $L_{\neq}$. Potom ak $x = y$ (to znamená $(x,y) \notin L_{\neq}$), tak R– protokol vráti „$x = y$“, resp. „rovný“ s určitosťou. Ak $x \neq y$ (to zn. $(x,y) \in L_{\neq}$), tak R– protokol vráti „$x \neq y$“, resp. „nerovný“ s pravdepodobnosťou aspoň $1 - \frac{2 \ln n}{n}$.

2.5.2 Algoritmy typu Monte Carlo s ohraničenou chybou

Definícia 2.5.2. *Nech F je funkcia. Hovoríme, že pravdepodobnostný algoritmus A je* ***typu Monte Carlo s ohraničenou chybou*** *(ozn. 2MC), ak existuje reálne číslo $\varepsilon, 0 < \varepsilon \leq \frac{1}{2}$ také, že pre každý vstup x funkcie F platí* $\text{Prob}(A(x) = F(x)) \geq \frac{1}{2} + \varepsilon$.

Táto trieda pravdepodobnostných algoritmov je definovaná takým spôsobom, že

- existencia čísla ε a jeho nezávislosť od vstupov x nám umožňuje znížiť pravdepodobnosť chyby pod vopred stanovenú hranicu δ konštantno–násobným opakovaním nezávislých výpočtov,

- akékoľvek uvoľnenie podmienok definície 2MC vedie k situácií, pri ktorej ani polynomiálne násobným opakovaním výpočtov nedostaneme pravdepodobnosť chyby pod vopred stanovenú hranicu δ.

Ak nejaký algoritmus A je 1MC algoritmus (alebo LV), potom algoritmus A_2 (2– zopakovaný algoritmus A) zráta korektný výsledok s pravdepodobnosťou aspoň $\frac{3}{4}$ a teda spĺňa podmienku $\text{Prob}(A(x) = F(x)) \geq \frac{1}{2} + \varepsilon$.

Pravdepodobnostný R– protokol je 1MC algoritmus pre definovaný jazyk $L_{\neq} = \{(x,y)|(x,y) \in \{0,1\}^n : x \neq y, n \in \mathbb{N}\}$, ale nie je 1MC algoritmus pre jazyk $L_{=} = \{(x,y)|(x,y) \in \{0,1\}^n : x \neq y, n \in \mathbb{N}\}$. Naopak, tento protokol je pre jazyk $L_{=}$ typu 2MC.

Pre každý 2MC algoritmus A a ľubovoľné kladné číslo t, ukážeme rýchlosť zníženia pravdepodobnosti chýb v 2MC algoritmoch vzhľadom na počet opakovaní výpočtov na rovnakom vstupe.

Veta 2.5.3. *Algoritmus* $\mathtt{A_t}$ *patrí do 2MC.*

Algoritmus 6 $\mathtt{A_t}$

Require: x, 2MC algoritmus $\mathtt{A}$
  for $i = 1$ **to** t **do**
    vykonaj algoritmus $\mathtt{A}$ pre vstup x a zaznamenaj výsledok do α_i
  end for
  if v postupnosti $\langle \alpha_1, \alpha_2, \ldots, \alpha_t \rangle$ existuje α, ktorá sa opakuje aspoň $\lceil \frac{t}{2} \rceil$-krát **then**
    return α
  else
    return „neviem“
  end if

Dôkaz. Nech $\varepsilon > 0$ je taká konštanta, že pre všetky vstupy algoritmu x platí $\mathrm{Prob}(A(x) = F(x)) \geq \frac{1}{2} + \varepsilon$. Pre každý vstup x položme $p = p(x) = \mathrm{Prob}(A(x) = F(x)) \geq \frac{1}{2} + \varepsilon_x, \varepsilon_x \geq \varepsilon$.

Algoritmus $\mathtt{A_t}$ vráti „neviem“, ak správny výsledok nie je vyrátaný aspoň $\lceil \frac{t}{2} \rceil$– krát v t nezávislých opakovaniach algoritmu A pre vstup x.

Pre všetky $i < \lceil \frac{t}{2} \rceil$ označme pr_i pravdepodobnosť, že $\mathtt{A_t}$ zráta správny výsledok v práve i opakovaniach. V snahe ohraničiť pravdepodobnosť chyby algoritmu $\mathtt{A_t}$ nás zaujíma horná hranica pr_i pre všetky $i < \lceil \frac{t}{2} \rceil$, pričom vychádzame z binomického rozdelenia pravdepodobnosti. Teda máme

$$
\begin{aligned}
pr_i(x) &= \binom{t}{i} p^i (1-p)^{t-i} = \binom{t}{i} (p(1-p))^i (1-p)^{t-2i} \\
&\{\text{kým } t \geq 2i, \text{ máme } (1-p)^{t-i} = (1-p)^i (1-p)^{t-2i} \text{ a vtedy}\} \\
&= \binom{t}{i} \left(\left(\frac{1}{2} + \varepsilon_x \right) \left(\frac{1}{2} - \varepsilon_x \right) \right)^i \left(\frac{1}{2} - \varepsilon_x \right)^{2\left(\frac{t}{2} - i\right)} \\
&\{\text{lebo } p = \frac{1}{2} + \varepsilon_x \Leftrightarrow 1 - p = \frac{1}{2} - \varepsilon_x\} \\
&= \binom{t}{i} \left(\frac{1}{4} - \varepsilon_x^2 \right)^i \left(\left(\frac{1}{2} - \varepsilon_x \right)^2 \right)^{\frac{t}{2} - i}
\end{aligned}
$$

$$
\begin{aligned}
&< \binom{t}{i}\left(\frac{1}{4}-\varepsilon_x^2\right)^i\left(\left(\frac{1}{2}-\varepsilon_x\right)\left(\frac{1}{2}+\varepsilon_x\right)\right)^{\frac{t}{2}-i}\\
&\{\text{pretože } \frac{1}{2}-\varepsilon_x < \frac{1}{2}+\varepsilon_x\}\\
&= \binom{t}{i}\left(\frac{1}{4}-\varepsilon_x^2\right)^i\left(\frac{1}{4}-\varepsilon_x^2\right)^{\frac{t}{2}-i}\\
&= \binom{t}{i}\left(\frac{1}{4}-\varepsilon_x^2\right)^{\frac{t}{2}}\\
&\leq \binom{t}{i}\left(\frac{1}{4}-\varepsilon^2\right)^{\frac{t}{2}}\\
&\{\text{keďže } \varepsilon_x \geq \varepsilon, \text{ pre každý vstup } x\}.
\end{aligned}
$$

Vzhľadom k tomu, že $\mathtt{A_t}$ vypočíta správny výsledok F pre vstup x práve vtedy, keď aspoň $\lceil \frac{t}{2} \rceil$ opakovaní A vypočíta správny výsledok $F(x)$, dolná hranica pravdepodobnosti úspechu bude:

$$
\begin{aligned}
\text{Prob}(A_t(x) = F(x)) &= 1 - \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} pr_i(x)\\
&> 1 - \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} \binom{t}{i}\left(\frac{1}{4}-\varepsilon_x^2\right)^{\frac{t}{2}}\\
&= 1 - \left(\frac{1}{4}-\varepsilon_x^2\right)^{\frac{t}{2}} \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} \binom{t}{i}\\
&> 1 - \left(\frac{1}{4}-\varepsilon_x^2\right)^{\frac{t}{2}} 2^t\\
&\geq 1 - \left(1-4\varepsilon_x^2\right)^{\frac{t}{2}}\\
&\geq 1 - \left(1-4\varepsilon^2\right)^{\frac{t}{2}}.
\end{aligned}
$$

Odtiaľ potom dostávame hornú hranicu $(1-4\varepsilon^2)^{\frac{t}{2}}$ pre pravdepodobnosť chyby algoritmu $\mathtt{A_t}$, ktorá sa blíži k 0 s rastúcim t. Potom za hľadanú konštantu k (konštanta pre opakovanie algoritmu), pre ktorú chceme, aby platilo $\text{Prob}(A_k(x) = F(x)) > 1-\delta$, stačí zobrať $k \geq \frac{2\ln\delta}{\ln(1-4\varepsilon^2)}$.

Preto, ak sú δ a ε sú považované za pevné konštanty, potom počet k opakovaní je tiež konštantný (vzhľadom na vstup). Z tohto dôvodu môžeme konštatovať, že

$$\mathrm{Time}_{A_k}(n) \in O(\mathrm{Time}_A(n)).$$

□

2.5.3 Algoritmy typu Monte Carlo s neohraničenou chybou

Pravdepodobnostný algoritmus nemôže byť použitý pre výpočet funkcie F, pokiaľ je pravdepodobnosť chyby menšia ako $\frac{1}{2}$. Inak by sme dostávali nesprávne výsledky s rovnakou pravdepodobnosťou ako správne výsledky. Dokonca, ani opakovaním niekoľkých nezávislých výpočtov na rovnakom vstupe nedokážeme rozoznať, ktorý výsledok je správny. V dôsledku tohto, pravdepodobnostné algoritmy vyžadujú, aby pravdepodobnosť chyby bola menšia ako $\frac{1}{2}$.

Definícia 2.5.4. *Nech F je funkcia. Hovoríme, že pravdepodobnostný algoritmus A je* ***Monte Carlo algoritmus s neohraničenou chybou*** *(ozn. MC), ak pre každý vstup x funkcie F platí* $\mathrm{Prob}(A(x) = F(x)) > \frac{1}{2}$.

V prípade MC algoritmov sa môže vzdialenosť medzi pravdepodobnosťou chyby a $\frac{1}{2}$ blížiť k 0 s rastúcou veľkosťou vstupu $|x|$. Napr. algoritmus A rovnomerne vyberá z $2^{|x|}$ deterministických stratégií pre vstup x a pre väčšinu z nich (aspoň pre $2^{|x|-1}+1$) dáva korektný výsledok. Potom A je MC algoritmus a platí $\mathrm{Prob}(A(x) = F(x)) = \frac{1}{2} + \frac{1}{2^{|x|}} > \frac{1}{2}$. Vzdialenosť medzi pravdepodobnosťou chyby a $\frac{1}{2}$ je rovná $\varepsilon_x = \frac{1}{2^{|x|}}$ a s rastúcim $|x|$ sa blíži k hodnote 0.

Ďalším problémom, ktorý nás zaujíma je koľkokrát potrebujeme nezávisle opakovať algoritmus A pre vstup x, ak chceme aby pre fixné δ platilo $\mathrm{Prob}(A_k(x) = F(x)) > 1 - \delta$?

Z predchádzajúcej analýzy 2MC algoritmu $\mathtt{A_t}$ vzhľadom na ε_x, bez zmeny pre MC algoritmy máme $\mathrm{Prob}(A(x) = F(x)) \geq 1 - (1 - 4\varepsilon_x^2)^{\frac{t}{2}}$. Dolnú hranicu počtu nezávislých opakovaní algoritmu A je potom možné získať z požadovanej podmienky $\mathrm{Prob}(A_k(x) = F(x)) \geq 1 - \delta$ nasledovne:

$$k = k(|x|) \geq \frac{2\ln\delta}{\ln(1-4\varepsilon_x^2)} = \frac{2\ln\delta}{\ln(1-4.2^{-2|x|})} = (-2\ln\delta).2^{2|x|},$$

pretože platí $\ln(1-y) \leq -y$, pre $0 < y < 1$. Týmto spôsobom dostávame

$$\mathrm{Time}_{A_k}(x) \geq (-2\ln\delta).2^{2|x|}.\mathrm{Time}_A(x).$$

Uvažujme teraz nasledovný protokol.

Algoritmus 7 Protokol `UMC`

Require: R_I obsahujúca n bitov $x = x_1x_2 \dots x_n$, R_{II} obsahujúca tiež n bitov $y = y_1y_2 \dots y_n$, $n \in \mathbb{N} - \{0\}$;
R_I rovnomerne náhodne vyberie $j \in \{1, 2, \dots, n\}$ a pošle R_{II} informáciu (j, x_j);
if $x_j \neq y_j$ **then**
  return akceptuj vstup (x, y)
else
  return akceptuj vstup (x, y) s pravdepodobnosťou $\frac{1}{2} - \frac{1}{2n}$, odmietni vstup (x, y) s pravdepodobnosťou $\frac{1}{2} + \frac{1}{2n}$
end if

Zrejme komunikačná zložitosť protokolu `UMC` je $\lceil log_2(n + 1) \rceil + 1$.

Veta 2.5.5. *Protokol* `UMC` *je typu MC.*

Dôkaz. Aby sme ukázali, že ide o MC protokol rozoberme dva prípady.

Prípad 1. Predpokladajme, že $(x, y) \notin L_{\neq}$, t.j. $x = y$.
Uvažujme $2n$ výpočtov C_{il}, kde $i \in \{1, 2, \dots, n\}, l \in \{0, 1\}$, pričom C_{il} je výpočet, v ktorom R_I rovnomerne náhodne vyberie index i a R_{II} vráti na výstup l ($l = 1$ ak akceptuje a $l = 0$ ak odmietne).

Dostávame pravdepodobnostný priestor $(S_{UMC,(x,y)}, \text{Prob})$, kde

$$S_{UMC,(x,y)} = \{C_{il} | 1 \leq i \leq n, l \in \{0, 1\}\}$$

a platí

$$\text{Prob}(\{C_{i0}\}) = \frac{1}{n} \cdot \left(\frac{1}{2} + \frac{1}{2n}\right)$$

a tiež

$$\text{Prob}(\{C_{i1}\}) = \frac{1}{n} \cdot \left(\frac{1}{2} - \frac{1}{2n}\right)$$

pre všetky $i \in \{1, 2, \dots, n\}$. Označme $A_0 = \{C_{i0} | 1 \leq i \leq n\}$. Potom

$$\text{Prob}(A_0) = \text{Prob}(UMC \text{ odmietne } (x, y)) = \sum_{i=1}^{n} \text{Prob}(\{C_{i0}\}) =$$

$$= \sum_{i=1}^{n} \frac{1}{n} \cdot \left(\frac{1}{2} + \frac{1}{2n}\right) = n \cdot \frac{1}{n}\left(\frac{1}{2} + \frac{1}{2n}\right) = \frac{1}{2} + \frac{1}{2n} > \frac{1}{2}.$$

Prípad 2. Predpokladajme teraz, že $(x, y) \in L_{=}$, t.j. $x \neq y$.
Zaujíma nás pravdepodobnosť toho, že UMC akceptuje (x, y), to znamená,

že existuje $j \in \{1, 2, \ldots, n\}$ také, že $x_j \neq y_j$. Bez ujmy na všeobecnosti uvažujme najhorší prípad, keď existuje len jediná taká možnosť. Teda existuje len jediný bit, v ktorom sa x a y líšia. Potom máme výpočet C_j, ktorý akceptuje vstup (x, y), a pre ktorý $\text{Prob}(\{C_j\}) = \frac{1}{n}$ a práve $2(n-1)$ výpočtov C_{il}, kde $i \in \{1, 2, \ldots, n\} \setminus \{j\}, l \in \{0, 1\}$. Potom C_{il} vyberie i pri vykonávaní R_I a l pri vykonávaní R_{II}.

Náš pravdepodobnostný priestor je $(S_{UMC,(x,y)}, \text{Prob})$, kde

$$(S_{UMC,(x,y)}, \text{Prob}) = \{C_j\} \cup \{C_{il} | 1 \leq i \leq n, i \neq j, l \in \{0, 1\}\}$$

a platí

$$\text{Prob}(\{C_j\}) = \frac{1}{n},$$

$$\text{Prob}(\{C_{i0}\}) = \frac{1}{n}\left(\frac{1}{2} + \frac{1}{2n}\right),$$

a

$$\text{Prob}(\{C_{i1}\}) = \frac{1}{n}\left(\frac{1}{2} - \frac{1}{2n}\right)$$

pre všetky $i \in \{1, 2, \ldots, n\}$. Označme $A_1 = \{C_j\} \cup \{C_{i1} | 1 \leq i \leq n, i \neq j\}$, potom

$$\begin{aligned}
\text{Prob}(A_1) &= \text{Prob}(UMC \text{ akceptuje } (x, y)) = \\
&= \text{Prob}(\{C_j\}) + \sum_{i=1, i \neq j}^{n} \text{Prob}(\{C_{i1}\}) = \\
&= \frac{1}{n} + \sum_{i=1, i \neq j}^{n} \frac{1}{n}\left(\frac{1}{2} - \frac{1}{2n}\right) \geq \\
&\geq \frac{1}{2n} + \frac{1}{2n} + \sum_{i=1}^{n-1}\left(\frac{1}{2n} - \frac{1}{2n^2}\right) = \\
&= \frac{1}{2n} + \sum_{i=1}^{n} \frac{1}{2n} - \sum_{i=1}^{n-1} \frac{1}{2n^2} = \\
&= \frac{1}{2n} + \frac{1}{2} - \frac{n-1}{2n^2} = \frac{1}{2} + \frac{1}{2n^2} > \frac{1}{2}.
\end{aligned}$$

Protokol teda spĺňa požadované podmienky a je typu MC. □

Kapitola 3

Aproximačné algoritmy

Aj keď s rozvojom výpočtových prostriedkov a ich intenzívnym prenikaním do nášho každodenného života by sa mohlo zdať, že súčasné technológie nám umožňujú zvládnuť všetky výpočtové problémy každodenného života, nie je tomu tak. Obrovské množstvo známych a ľahko formulovateľných problémov je NP-ťažkých a dosiaľ nie sú známe algoritmy, ktoré by ich umožňovali riešiť efektívne. Preto je potrebné na ich vyriešenie využívať iné prístupy. Jeden z nich predstavujú aproximačné algoritmy. Tie je možné využiť vtedy, ak sa dokážeme uspokojiť s približným riešením, ktoré vieme získať v pomerne krátkom čase. Takéto riešenie je pre praktický život častokrát postačujúce a umožňuje nám fungovať v každodennom živote. Veľmi podstatné však je, či vieme exaktne stanoviť ako veľmi sa získané riešenie líši od optimálneho riešenia. Týmto aspektom sa budeme venovať v tejto kapitole.

3.1 Optimalizačný problém

V matematike a informatike sa často narába s pojmom problém. Väčšinou si vystačíme s neformálnou definíciou. Napríklad problémom môžeme rozumieť triedu úloh rovnakého typu. Na analýzu algoritmov však potrebujeme exaktné vymedzenie tohoto pojmu.

Definícia 3.1.1. ***Optimalizačným problémom*** *nazývame usporiadanú sedmicu* $U = \langle \Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal \rangle$, *kde*

(i) Σ_I *je vstupná abeceda problému* U,

(ii) Σ_O *je výstupná abeceda problému* U,

(iii) $L \subseteq \Sigma_I^*$ *je jazyk prípustných inštancií problému (t.j. tie reťazce nad* Σ_I*, ktoré kódujú zmysluplné zadania problému),*

(iv) $L_I \subseteq L$ *je jazyk (aktuálnych) inštancií problému* U*,*

(v) $\mathcal{M} : L \to \mathcal{P}(\Sigma_O^*)$*, kde pre každé* $x \in L$ *je* $\mathcal{M}(x)$ *množina prípustných riešení pre vstup* x*,*

(vi) cost je účelová funkcia taká, že pre každú dvojicu (u, x)*, kde* $u \in \mathcal{M}(x)$ *a* $x \in L$ *je* $cost(u, x)$ *kladné reálne číslo,*

(vii) $goal \in \{\max, \min\}$*.*

Pre každé $x \in L_I$*, prípustné riešenie* $y \in \mathcal{M}(x)$ *sa nazýva* ***optimálne riešenie pre*** x ***a*** U *ak* $cost(y, x) = goal\{cost(z, x) | z \in \mathcal{M}(x)\}$*. Optimálnu hodnotu účelovej funkcie problému* U *pre vstupný reťazec* x *budeme označovať* $Opt_U(x) = cost(y, x)$*.*

U sa vzhľadom na hodnotu goal nazýva ***maximalizačný*** *resp.* ***minimalizačný*** *problém.*

Algoritmus A *je konzistentný pre* U*, ak pre každé* $x \in L_I$ *je jeho výstup* $A(x) \in \mathcal{M}(x)$*. Hovoríme, že algoritmus* A *rieši optimalizačný problém* U*, ak je konzistentný pre* U *a pre všetky* $x \in L_I$ *je* $A(x)$ *optimálne riešenie pre* x *a* U*.*

Nech $U_i = \langle \Sigma_I, \Sigma_O, L, L_{I,i}, \mathcal{M}, cost, goal \rangle$ *pre* $i \in \{1, 2\}$ *sú dva optimalizačné problémy. Hovoríme, že* U_1 *je* ***podproblémom*** U_2 *ak* $L_{I,1} \subseteq L_{I,2}$*.*

V praxi sa veľmi často využíva zjednodušený prístup a optimalizačné problémy sa často definujú nasledovným spôsobom:

- nezaoberáme sa špecifikáciou Σ_I a Σ_O,
- špecifikuje sa iba L_I,
- špecifikujú sa obmedzenia pre vstupné inštancie a tiež $\mathcal{M}(x)$ pre každé $x \in L_I$,
- špecifikuje sa účelová (hodnotiaca) funkcia,
- špecifikuje sa cieľ (*goal*).

Ilustrujme si uvedenú definíciu na príklade Problému obchodného cestujúceho (*angl.* Traveling Salesman Problem - TSP). Pripomeňme si, že v danom prípade je našim cieľom nájsť najlacnejšiu hamiltonovskú kružnicu v ohodnotenom grafe.

PROBLÉM OBCHODNÉHO CESTUJÚCEHO (TSP)
VSTUP: Ohodnotený úplný graf (G, c), kde $G = (V, E)$ a $c : E \to \mathbb{N}$, $V = \{v_1, \ldots, v_n\}$, $n \in \mathbb{N} \setminus \{0\}$.
OBMEDZENIA: Pre každý vstup (inštanciu problému) (G, c) je $\mathcal{M}(G, c) = \{(v_{i_1}, \ldots, v_{i_n}) | \{i_1, \ldots, i_n\}$ je permutácia množiny $\{1, \ldots, n\}\}$, t.j. je to množina hamiltonovských kružníc v G.
ÚČELOVÁ FUNKCIA: pre všetky $(v_{i_1}, \ldots, v_{i_n}) \in \mathcal{M}(G, c)$ je

$$cost((v_{i_1}, \ldots, v_{i_n}), (G, c)) = \sum_{j=1}^{n} c(\{v_{i_j}, v_{i_k}\}), \text{ kde } k = (j + 1 \mod n).$$

CIEĽ: minimum.

Vstupom môže byť incidenčná matica modifikovaná tak, že na príslušných miestach budú v nej uvedené váhy hrán.

Metrický TSP ($\triangle$-TSP) je taký podproblémom TSP, že každá jeho inštancia (G, c) spĺňa trojuholníkovú nerovnosť

$$c(\{u, v\}) \leq c(\{u, w\}) + c(\{w, v\})$$

pre všetky $u, v, w \in V$.

Geometrický TSP (nazývaný aj euklidovský TSP) je taký podproblém TSP, že každá inštancia (G, c) má vrcholovú množinu, ktorá sa dá vnoriť do dvojrozmerného euklidovského priestoru takým spôsobom, že $c(\{u, v\})$ je euklidovská vzdialenosť medzi bodmi prislúchajúcim vrcholom u a v.

3.2 Aproximačné riešenie

Už sme spomínali, že ak optimalizačný problém neumožňuje, aby sme efektívne vypočítali optimálne riešenie, v praktickom živote je veľmi často užitočná aj aproximácia optimálneho riešenia. Je fascinujúce, ako nám malá zmena v zadaní úlohy (pripustíme ϵ-ovú chybu) umožní namiesto exponenciálneho algoritmu použiť polynomiálny a úlohu uspokojivo vyriešiť.

Definícia 3.2.1. *Nech $U = \langle \Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal \rangle$ je optimalizačný problém a A je konzistentný algoritmus pre U. Pre každé $x \in L_I$ je* ***relatívna chyba*** *algoritmu A pre vstup x definovaná ako*

$$\epsilon_A(x) = \frac{|cost(A(x)) - Opt_U(x)|}{Opt_U(x)}.$$

Pre všetky $n \in \mathbb{N}$ *definujeme* ***relatívnu chybu*** *algoritmu* A

$$\epsilon_A(n) = \max\{\epsilon_A(x) | x \in L_I \wedge \Sigma_I^n\}.$$

Pre každé $x \in L_i$ *definujeme* ***aproximačný pomer*** *algoritmu* A *pre vstup* x

$$R_A(x) = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\} = 1 + \epsilon_A(x).$$

Pre každé $n \in \mathbb{N}$ *definujeme* ***aproximačný pomer*** *algoritmu* A *ako*

$$R_A(n) = \max\{R_A(x) | x \in L_I \wedge \Sigma_I^n\}.$$

Potom pre každé reálne $\delta > 1$ *hovoríme, že* A *je* δ***-aproximačný algoritmus*** *pre* U*, ak* $R_A(x) \leq \delta$ *pre všetky* $x \in L_I$*.*
Pre každú funkciu $f : \mathbb{N} \to \mathbb{R}^+$ *hovoríme, že* A *je* $f(n)$***-aproximačný algoritmus*** *pre* U *ak* $R_A(n) \leq f(n)$ *pre každé* $n \in \mathbb{N}$*.*

3.2.1 Rozvrhovací problém

Rozoberme si najprv jednu z verzií Rozvrhovacieho problému, ktorá je v anglickej literatúre známa pod názvom Makespan Scheduling Problem (MSP). Na vstupe je $n \in \mathbb{N}$ úloh s časom vykonania $p_1, \ldots, p_n$ a $m \geq 2$ rovnakých strojov. Cieľom je minimalizovať čas za ktorý bude všetkých n úloh zrealizovaných. Ak máme napríklad úlohy s časom spracovania 2,3,4,1,3,6,4, tak minimálny čas pre štyri stroje je 6. Stačí na prvom stroji spracovať úlohy s trvaním 2 a 4, na druhom s trvaním 3 a 3, na treťom 6 a ma štvrtom zostávajúce dve úlohy. V tomto prípade je to zjavne optimálne riešenie, pretože v zozname je úloha, ktorá sa za čas kratší ako 6 spracovať nedá. Formálna definícia študovaného problému je nasledovná:

Rozvrhovací problém (MSP)
VSTUP: kladné celé čísla $p_1, p_2, \ldots, p_n$, $n \in \mathbb{N} \setminus \{0\}$ a $m \in \mathbb{N}, m \geq 2$
OBMEDZENIA: pre vstup $(p_1, p_2, \ldots, p_n, m)$ je $\mathcal{M}(p_1, p_2, \ldots, p_n, m) = \{S_1, S_2, \ldots, S_m : S_i \subseteq \{1, 2, \ldots, n\}$ pre $i = 1, 2, \ldots, n,\ S_i \cap S_j = \emptyset$ pre $i \neq j,\ \bigcup_{i=1}^m S_i = \{1, 2, \ldots, n\}\}$
ÚČELOVÁ FUNKCIA: pre každé prípustné riešenie je

$$cost((S_1, S_2, \ldots, S_m), (p_1, p_2, \ldots, p_n, m)) = \max\{\sum_{j \in S_i} p_j : i = 1, 2, \ldots, m\}$$

CIEĽ: minimum.

Algoritmus 8 `GMS`

Require: $(p_1, \ldots, p_n, m) \in \mathbb{N}^{n+1}$, $m \geq 2$
  for $i = 1$ **to** m **do**
    $T_i := \{i\}$
    $\mathrm{Time}(T_i) := p_i$
  end for
  for $i = m + 1$ **to** n **do**
    vyrátaj k také, že $\mathrm{Time}(T_k) = \min\{\mathrm{Time}(T_j) : 1 \leq j \leq m\}$
    $T_k := T_k \cup \{i\}$
    $\mathrm{Time}(T_k) := \mathrm{Time}(T_k) + p_i$
  end for
  return $(T_1, \ldots, T_m)$

Ukážeme si teraz pažravý (*angl.* greedy) algoritmus, ktorý je 2-aproximačný. Hlavná idea algoritmu spočíva v tom, že usporiadame časy trvania úloh nerastúco a doteraz nepriradenú úlohu sa vždy snažíme priradiť na prvý čo najmenej vyťažený stroj.

Analyzujme teraz algoritmus `GMS` (MS označuje triedu problému). Zrejme

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq \cdots \geq p_n.$$

Navyše

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m},$$

čo je priemerný čas práce každého stroja. Ďalej pre každé k máme

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k}, \text{ lebo } p_k \leq p_{k-1} \leq \cdots \leq p_1.$$

Rozlíšime teraz dva prípady:

$n \leq m$: Pretože vždy platí $Opt_{\mathrm{MS}}(I) \geq p_1$ a v tomto prípade je optimálne obsadenie strojov $(\{1\}, \{2\}, \ldots, \{n\}, \emptyset, \ldots, \emptyset)$, `GMS` nájde optimum a jeho aproximačný pomer je rovný 1.

$n > m$: Nech T_k je také, že $cost(T_k) = \sum_{r \in T_k} p_r = cost(\mathtt{GMS}(I))$ (na tomto stroji sa nadobúda maximum). Nech q je najväčší index v T_k.

Ak $q \leq m$, tak $|T_k| = 1$ (tomuto stroju bola zatiaľ priradená iba jedna úloha) a teda $Opt_{\mathrm{MS}}(I) = p_1 = p_q$ a $\mathtt{GMS}(I)$ dáva optimálny výsledok.

Nech teraz $m < q$. Ak v tomto prípade z T_k odoberieme p_q tak sa z tohto rozvrhu stane aktuálne najkratší a to znamená, že

$$Opt_{\mathrm{MS}}(I) \geq cost(\mathtt{GMS}(I)) - p_q.$$

Keďže

$$p_q \leq \frac{\sum_{i=1}^{q} p_i}{q},$$

dostávame

$$cost(\texttt{GMS}(I)) - Opt_{\mathrm{MS}}(I) \leq p_q \leq \frac{\sum_{i=1}^{q} p_i}{q}.$$

Pripomeňme tiež, že

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m}.$$

Potom

$$\frac{cost(\texttt{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \leq \frac{\frac{\sum_{i=1}^{q} p_i}{q}}{\frac{\sum_{i=1}^{n} p_i}{m}} \leq \frac{m}{q} < 1.$$

Teda aproximačný pomer algoritmu GMS, t.j. $R_{\texttt{GMS}}(x)$, je menší ako 2 pre každé $x \in L_I$. GMS je preto 2-aproximačný algoritmus pre problém MSP.

3.3 Príklady ďalších algoritmov

V nasledujúcom sa budeme venovať niekoľkým príkladom, ktoré nám umožnia ilustrovať rozdielnu mieru aproximovateľnosti reálnych problémov.

3.3.1 Problém minimálneho vrcholového pokrytia

Jedným z typických problémov študovaných v rámci teórie výpočtovej zložitosti je Problém minimálneho vrcholového pokrytia (*angl.* Minimum Vertex Cover Problem - MinVCP).

MINIMÁLNE VRCHOLOVÉ POKRYTIE (MinVCP)
VSTUP: Graf (G, E)
OBMEDZENIA: $\mathcal{M}(G) = \{S \subseteq V :$ každá hrana z E je incidentná s aspoň jedným vrcholom z $S\}$
ÚČELOVÁ FUNKCIA: pre všetky $S \in \mathcal{M}(G)$ je $cost(S, G) = |S|$
CIEĽ: minimum.

Algoritmus VCP ponúka efektívne a prirodzené riešenie.

Príklad 3.3.1. *Ukážeme si ako algoritmus* VCP *funguje na grafe* $G^* = (V, E)$ *s vrcholovou množinou* $V = \{a, b, c, d, e, f, g, h\}$ *a množinou hrán* $E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\}, \{d, f\}, \{d, g\}, \{d, h\}, \{g, h\}, \{e, f\}\}$ *(graf je znázornený na obr. 3.1).*

Algoritmus 9 `VCP`

Require: Graf $G = (V, E)$
  $C := \emptyset$ ($C \subseteq V$, na konci bude C obsahovať vrcholové pokrytie)
  $A := \emptyset$ ($A \subseteq E$ je spárenie, na konci algoritmu A bude maximálne spárenie)
  $E' := E$ ($E' \subseteq E$, E' obsahuje práve hrany, ktoré nie sú pokryté aktuálnym spárením, na konci $E' = \emptyset$)
  while $E' \neq \emptyset$ **do**
    vyber ľubovoľnú hranu $\{u, v\} \in E'$
    $C := C \cup \{u, v\}$
    $A := A \cup \{\{u, v\}\}$
    $E' := E' \setminus \{$ všetky hrany incidentné s u alebo $v\}$
  end while
  return C

Optimálne vrcholové pokrytie je $\{b, e, d, g\}$. *Toto optimum sa nedá dosiahnuť nijakým výberom hrán v algoritme. Jedna z možností postupností výberu hrán je nasledovná:*

1. $\{b, c\} \to A \qquad C := \{b, c\}$

$E' = \{\{e, d\}, \{e, f\}, \{d, f\}, \{d, g\}, \{d, h\}, \{h, g\}\}$

2. $\{e, f\} \to A \qquad C := \{b, c, e, f\} \qquad E' = \{\{d, g\}, \{d, h\}, \{h, g\}\}$

3. $\{d, g\} \to A \qquad C := \{b, c, e, f, d, g\} \qquad E' = \emptyset$

Lema 3.3.2. *Algoritmus pracuje v čase* $O(|E|) = O(|V|^2)$.

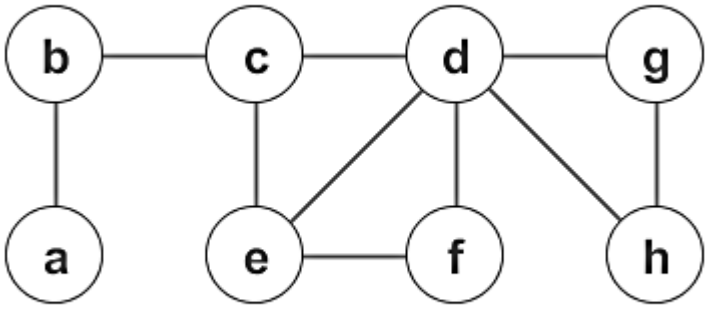
Dôkaz. Ak zvolíme vhodnú dátovú štruktúru, tak zrejme s každou hranou narábame práve raz.

□

Lema 3.3.3. *Algoritmus vždy nájde vrcholové pokrytie s aproximačným pomerom nanajvýš 2.*

Dôkaz. Zrejme algoritmus nájde vrcholové pokrytie, lebo zastaví, keď $E' = \emptyset$ (t.j. všetky hrany sú pokryté). Evidentne $|C| = 2 \cdot |A|$. Na pokrytie všetkých vrcholov v spárení treba aspoň $|A|$ vrcholov. Pretože $A \subseteq E$, mohutnosť každého vrcholového spárenia je aspoň $|A|$, t.j. $Opt_{\text{MinVCP}}(G) \geq |A|$. Teda $cost_A(G) = |C| = 2 \cdot |A| \leq 2 \cdot Opt_{\text{MinVCP}}(G)$ a $R_A(G) \leq 2$ pre každý graf G. Prezentovaný algoritmus je 2-aproximačný.

□



Obr. 3.1: Graf G^*

3.3.2 Problém množinového pokrytia

Ďalším zaujímavým problémom je Problém množinového pokrytia (*angl.* Set Cover Problem - SCP).

PROBLÉM POKRYTIA MNOŽÍN (SCP)
VSTUP: $(X, \mathcal{F}$, kde X je konečná množina, $\mathcal{F} \subseteq \mathcal{P}(X)$ taká, že $X = \bigcup_{S \in \mathcal{F}} S\}$
OBMEDZENIA: pre všetky $(X, \mathcal{F})$ je $\mathcal{M}(X, \mathcal{F}) = \{C \subseteq \mathcal{F} : X = \bigcup_{S \in C} S\}$
ÚČELOVÁ FUNKCIA: pre všetky $C \in \mathcal{M}(X, \mathcal{F})$ je $cost(C, (X, \mathcal{F})) = |C|$
CIEĽ: minimum.

Uvedený problém budeme riešiť nasledovným algoritmom.

Na analýzu kvality algoritmu budeme potrebovať pojem harmonického čísla.

Definícia 3.3.4. *Pre ľubovoľné prirodzené číslo n je n-té* ***harmonické*** *číslo definované ako*

$$\mathrm{Har}(n) = \sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

Algoritmus 10 `SCP`

Require: $(X, \mathcal{F})$ X je konečná množina, $\mathcal{F} \subseteq \mathcal{P}(X) = 2^X$, $X = \bigcup_{Q \in F} Q$
  $\mathcal{C} := \emptyset$ ($\mathcal{C} \subseteq \mathcal{F}$ a na konci je $\mathcal{C}$ množinové pokrytie $(X, \mathcal{F})$)
  $U := X$ ($U \subseteq X$, $U = X \setminus \bigcup_{Q \in \mathcal{C}} Q$ pre aktuálnu $\mathcal{C}$, na konci $U = \emptyset$)
  while $U \neq \emptyset$ **do**
    vyber $S \in \mathcal{F}$ takú, že $|S \cap U|$ je maximálne
    $U := U \setminus S$
    $\mathcal{C} := \mathcal{C} \cup \{S\}$
  end while
  return $\mathcal{C}$

Zrejme $\lim_{n \to \infty} \mathrm{Har}(n) = \infty$ harmonický rad diverguje lebo

$$\mathrm{Har}(n) = (1) + \left(\frac{1}{2} + \frac{1}{3}\right) + \left(\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}\right) + \cdots + \\ + \left(\frac{1}{2^{k-1}} + \cdots + \frac{1}{2^k - 1}\right) + \cdots + \frac{1}{n}$$

a každý prvok k-tej skupiny je väčší ako $\frac{1}{2^k}$, teda súčet všetkých 2^{k-1} prvkov k-tej skupiny je medzi $\frac{2^{k-1}}{2^k} = \frac{1}{2}$ a $\frac{2^{k-1}}{2^{k-1}} = 1$.

Zgrupovacia procedúra teda ukazuje, že ak n je v k-tej v skupine, tak

$$\frac{k}{2} < \mathrm{Har}(n) \leq k.$$

Teda

$$\frac{\lfloor \log_2 n \rfloor}{2} + \frac{1}{2} < \mathrm{Har}(n) \leq \lfloor \log_2 n \rfloor + 1.$$

Lema 3.3.5. *Algoritmus* `SCP` *je* $\mathrm{Har}(\max\{|S| : S \in \mathcal{F}\})$*-aproximačný algoritmus pre problém SCP.*

Dôkaz. Nech $\mathcal{C} = \{S_1, S_2, \ldots, S_n\}$ je výstup algoritmu, kde S_i je i-tá množina vybraná algoritmom. Pretože $U = \emptyset$, zrejme $\mathcal{C}$ je množinové pokrytie $(X, \mathcal{F})$.

Pre každé $x \in X$ definujeme

$$w(x) = \frac{1}{|S_i \setminus \bigcup_{j=1}^{i-1} S_j|},$$

za predpokladu, že $x \in S_i \setminus \bigcup_{j=1}^{i-1} S_j$. Takže $w(x)$ je definovaná akonáhle je x prvýkrát pokryté $\mathcal{C}$. Ak položíme $w(\mathcal{C}) = \sum_{x \in X} w(x)$ tak dostávame

$$w(\mathcal{C}) = \sum_{x \in X} w(x) = \sum_{i=1}^{n} \frac{|S_i \setminus \bigcup_{j=1}^{i-1} S_j|}{|S_i \setminus \bigcup_{j=1}^{i-1} S_j|} = \sum_{i=1}^{n} 1 = n = |\mathcal{C}| = cost(\mathcal{C}).$$

Nech $\mathcal{C}_{opt}$ je optimálne riešenie s $cost(\mathcal{C}_{opt}) = Opt_{\mathrm{SCP}}(X, \mathcal{F})$. Chceme ukázať, že rozdiel medzi $w(\mathcal{C})$ a $|\mathcal{C}_{opt}|$ nie je príliš veľký.

Ukážeme, že pre každú $S \in \mathcal{F}$ platí $\sum_{x \in S} w(x) \leq \mathrm{Har}(|S|)$. Pre každú $S \in \mathcal{F}$ a pre každé $i \in \{0, 1, \ldots, |\mathcal{C}|\}$ položíme

$$cov_i(S) = \left| S \setminus \bigcup_{j=1}^{i} S_i \right|.$$

Zrejme pre každú $S \in \mathcal{F}$ je

- $cov_0(S) = S$,
- $cov_{|\mathcal{C}|}(S) = 0$,
- $cov_{i-1}(S) \geq cov_i(S)$ pre ľubovolné $i \in \{1, 2, \ldots, |C|\}$.

Nech k je najmenšie číslo také, že $cov_k(S) = 0$. Pretože $cov_{i-1}(S) - cov_i(S)$ prvkov z S je pokrytých v i-tom kroku algoritmu, tak

$$\sum_{x \in S} w(x) = \sum_{i=1}^{k} (cov_{i-1}(S) - cov_i(S)) \cdot \frac{1}{|S_i \setminus \bigcup_{j=1}^{i-1} S_j|}.$$

Všimnime si, že

$$\left| S_i \setminus \bigcup_{j=1}^{i-1} S_j \right| \geq \left| S \setminus \bigcup_{j=1}^{i-1} S_j \right| = cov_{i-1}(S),$$

lebo ináč by v danom kroku algoritmu bola vybraná množina S namiesto S_i. Z posledných dvoch vzťahov dostávame

$$\sum_{x \in S} w(x) \leq \sum_{i=1}^{k} (cov_{i-1}(S) - cov_i(S)) \cdot \frac{1}{cov_{i-1}(S)}.$$

Pretože

$$\mathrm{Har}(b) - \mathrm{Har}(a) = \sum_{i=a+1}^{b} \frac{1}{i} \geq (b - a) \cdot \frac{1}{b},$$

dostávame pre ľubovoľné $i \in \{1, \ldots, k\}$ vzťah

$$(cov_{i-1}(S) - cov_i(S)) \cdot \frac{1}{cov_{i-1}(S)} \leq \mathrm{Har}(cov_{i-1}(S)) - \mathrm{Har}(cov_i(S)).$$

Ak skombinujeme posledné dva výrazy, tak dostávame

$$\begin{aligned}\sum_{x \in S} w(x) &\leq \sum_{i=1}^{k} \mathrm{Har}(cov_{i-1}(S)) - \mathrm{Har}(cov_i(S)) = \\ &= \mathrm{Har}(cov_0(S)) - \mathrm{Har}(cov_k(S)) = \\ &= \mathrm{Har}(|S|) - \mathrm{Har}(0) = \\ &= \mathrm{Har}(|S|).\end{aligned}$$

Pretože $\mathcal{C}_{opt}$ je tiež pokrytie X, dostávame

$$|\mathcal{C}| = \sum_{x \in X} w(x) \leq \sum_{S \in \mathcal{C}_{opt}} \sum_{x \in S} w(x).$$

Keďže sme ukázali, že pre každé $S \in \mathcal{F}$ je $\sum_{x \in S} w(x) \leq \mathrm{Har}(|S|)$, tak dostávame

$$\begin{aligned}|\mathcal{C}| &\leq \sum_{S \in \mathcal{C}_{opt}} \sum_{x \in S} w(x) \\ &\leq \mathrm{Har}(|S|) \\ &\leq |\mathcal{C}_{opt}| \cdot \mathrm{Har}(\max\{|S| : S \in \mathcal{F}\}).\end{aligned}$$

Teda

$$\frac{\mathcal{C}}{\mathcal{C}_{opt}} \leq \frac{|\mathcal{C}_{opt}| \cdot \mathrm{Har}(\max\{|S| : S \in \mathcal{F}\})}{|\mathcal{C}_{opt}|} = \mathrm{Har}(\max\{|S| : S \in \mathcal{F}\}).$$

□

Lema 3.3.6. *Algoritmus* `SCP` *je* $\ln(|X|)$*-aproximačný algoritmus pre problém SCP.*

Dôkaz. Vieme, že $\max\{|S| : S \in \mathcal{F}\} \leq |X|$ a $\mathrm{Har}(n) \leq \ln n$ pre každé $n \in (N)$.

□

Lema 3.3.7. *Časová zložitosť algoritmu* `SCP` *je* $O(n^{\frac{3}{2}})$.

Dôkaz. Zakódujme vstup tak, že $|X| \cdot |\mathcal{F}|$ je veľkosť inštancie $(X, \mathcal{F})$ (stačí pre každú $S \in \mathcal{F}$ definovať charakteristickú funkciu). Jeden krok cyklu algoritmu trvá $O(|X| \cdot |\mathcal{F}|)$ krokov. Počet cyklov je pritom limitovaný číslom $\min\{|X|, |\mathcal{F}|\} \leq (|X| \cdot |\mathcal{F}|)^{\frac{1}{2}}$. Teda časová zložitosť celého algoritmu je $O((|X| \cdot |\mathcal{F}|)^{\frac{1}{2}})$.

□

Z predchádzajúcich dvoch tvrdení okamžite plynie nasledujúca veta.

Veta 3.3.8. *Prezentovaný algoritmus je polynomiálny* $\ln n$*-aproximačný algoritmus pre SCP.*

3.3.3 Problém maximálneho rezu

Ďalej sa budeme zaoberať Problémom maximálneho rezu v grafe (*angl.* Maximum Cut Problem MAX-CUT).

Definícia 3.3.9. ***Rez v grafe*** $G = (V, E)$ *je rozklad množiny* V *na množiny* (V_1, V_2)*, že* $V = V_1 \cup V_2$ *a* $V_1 \cap V_2 = \emptyset$*.* ***Cena rezu*** *je počet hrán spájajúcich obe množiny* $|E \cap \{(u, v) \in S \times (V \setminus S)\}|$*.*

PROBLÉM MAXIMÁLNEHO REZU (MAX-CUT)
VSTUP: $G = (V, E)$
OBMEDZENIA: pre všetky G je $\mathcal{M}(G) = \{(V_1, V_2) : V_1, V_2 \subseteq V, V_1 \cap V_2 = \emptyset, V_1 \cup V_2 = V\}$
ÚČELOVÁ FUNKCIA: pre všetky $(V_1, V_2) \in \mathcal{M}(G)$ je $cost((V_1, V_2), G)$ rovný cene rezu (V_1, V_2)
CIEĽ: maximum.

Na riešenie tejto úlohy použijeme nasledovný algoritmus.

Algoritmus 11 `CUT`

Require: Graf $G = (V, E)$
  $S := \emptyset$ rez je tvaru $(S, V \setminus S)$
  while existuje taký vrchol $u \in V$, že jeho presun z jednej časti rezu $(S, V \setminus S)$ do druhej zvýši cenu rezu **do**
    presuň takýto vrchol z jednej časti do druhej
  end while
  return $(S, V \setminus S)$.

Lema 3.3.10. *Čas behu algoritmu* `CUT` *je* $O(|E| \cdot |V|)$.

Dôkaz. Prehľadanie vrcholovej množiny sa udeje v čase $O(|V|)$, pričom pri každom vrchole je potrebné zistiť cenu rezu, čo je možné v čase $O(|E|)$. □

Veta 3.3.11. *Algoritmus* `CUT` *je polynomiálny 2-aproximačný.*

Dôkaz. Algoritmus poskytuje prípustné riešenia a podľa predchádzajúcej lemy je čas jeho behu polynomiálny.

Nech (Y_1, Y_2) je výstup nášho algoritmu. Každý vrchol v Y_1 má aspoň toľko susedov v Y_2 ako susedov v Y_1. Inak by malo zmysel vrchol premiestniť. Teda aspoň polovica hrán grafu je v reze (Y_1, Y_2), teda $cost((S, V \setminus S)) \geq \frac{|E|}{2}$. Pretože $Opt_{\text{MAX-CUT}}(G) \leq |E|$, dostávame, že algoritmus je 2-aproximačný. □

3.4 Klasifikácia optimalizačných problémov

V praktickom živote je samozrejme zaujímavé, či vieme ľubovoľne znižovať hodnotu relatívnej chyby algoritmu pre vybraný problém. Nie vždy je tomu tak, a preto zavádzame nasledovné dve triedy algoritmov.

Definícia 3.4.1. *Nech* $U = \langle \Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal \rangle$ *je optimalizačný problém a* $\text{Time}_A(x, \epsilon^{-1})$ *je časová zložitosť algoritmu* A *so vstupom* (x, ϵ). *Algoritmus* A *sa nazýva* ***polynomiálna aproximačná schéma*** (PTAS) ***pre*** U *ak pre každú dvojicu* $(x, \epsilon) \in L_I \times \mathbb{R}^+$ *algoritmus* A *počíta prípustné riešenie* $A(x)$ *s relatívnou chybou najviac* ϵ *a funkcia* $\text{Time}_A(x, \epsilon^{-1})$ *sa dá ohraničiť funkciou, ktorá je polynomiálna vzhľadom na dĺžku vstupu* x. *Ak* $\text{Time}_A(x, \epsilon^{-1})$ *sa dá ohraničiť funkciou, ktorá je polynomiálna aj vzhľadom na* $|x|$ *aj na* ϵ^{-1}, *tak hovoríme, že* A *je* ***plne polynomiálnou aproximačnou schémou*** (FPTAS) ***pre*** U.

Z definície vidieť, že ak zvyšujeme požiadavky na kvalitu, t.j. ($\epsilon \to 0$), tak to môže negatívne ovplyvňovať výpočtovú zložitosť. Ak je tomu tak, algoritmus patrí do triedy PTAS. Ak však zložitosť nezávisí od požadovanej kvality, tak algoritmus patrí do triedy FPTAS. FPTAS je pravdepodobne to najlepšie čo vieme dosiahnuť pre NP-úplné problémy.

Nasledujúce definície zavádzajú kompletnú klasifikáciu problémov z hľadiska ich aproximovateľnosti.

Definícia 3.4.2. *Problém* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *patrí do triedy optimalizačných problémov* NPO *ak sú splnené nasledovné podmienky:*

(i) $L_I \in P$,

(ii) *existuje polynóm* p_U *taký, že*

(a) *pre každé* $x \in L_I$ *a každé* $y \in \mathcal{M}(x)$ *platí* $|y| \leq p_U(|x|)$ *a zároveň*

(b) *existuje polynomiálny algoritmus taký, že pre všetky* $y \in \Sigma_O^*$ *a každé* $x \in L_I$*, pre ktoré* $|y| \leq p_U(|x|)$*, algoritmus rozhodne, či* $y \in \mathcal{M}(x)$,

(iii) *účelová funkcia sa dá spočítať v polynomiálnom čase.*

Neformálne, problém U je v triede NPO ak

- je možné efektívne verifikovať, že nejaký reťazec je inštanciou problému U,
- veľkosť prípustných riešení je polynomiálna vzhľadom na veľkosť vstupu inštancie a v polynomiálnom čase je možné overiť, že y je riešením problému U pre vstup x,
- cena každého riešenia sa dá efektívne vypočítať.

Definícia 3.4.3. *Problém* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *patrí do triedy optimalizačných problémov* PO *ak*

(i) $U \in$ NPO,

(ii) *existuje polynomiálny algoritmus, ktorý pre každé* $x \in L_I$ *vypočíta optimálne riešenie pre* x.

Pomocou zavedených definícií týkajúcich sa aproximovateľnosti, môžeme NPO rozdeliť na päť podtried:

NPO(I): Obsahuje všetky optimalizačné problémy z NPO, pre ktoré existuje FPTAS.

NPO(II): Obsahuje všetky optimalizačné problémy z NPO, pre ktoré existuje PTAS.

NPO(III): Obsahuje všetky optimalizačné problémy $U \in$ NPO také, že

(a) existuje polynomiálny δ-aproximačný algoritmus pre nejaké $\delta > 1$,

(b) neexistuje polynomiálny d-aproximačný algoritmus pre U pre nejaké $d < \delta$ (za nejakého rozumného predpokladu napr. P $\neq$ NP), t.j. neexistuje PTAS pre U. (pozn. v PTAS sa pripúšťa ľubovoľné $\epsilon > 0$).

NPO(IV): Obsahuje všetky $U \in$ NPO také, že

(a) existuje polynomiálny $f(n)$-aproximačný algoritmus pre U pre nejakú $f : \mathbb{N} \to \mathbb{R}^+$, kde f je ohraničená nejakou polylogaritmickou funkciou,

(b) za nejakých rozumných predpokladov (ako napr. P $\neq$ NP) neexistuje polynomiálny δ-aproximačný algoritmus pre U pre nejaké $\delta \in \mathbb{R}^+$.

NPO(V): Obsahuje všetky $U \in$ NPO také, že ak existuje polynomiálny $f(n)$-aproximačný algoritmus pre U (za nejakých rozumných predpokladov ako NP $\neq$ P) $f(n)$ nie je ohraničená nijakou polylogaritmickou funkciou.

Za predpokladu, že P $\neq$ NP ani jedna z tried NPO nie je prázdna. Niektoré algoritmy z NPO(I) majú veľký exponent, a teda sú menej praktické ako $\log n$-aproximačné algoritmy pre NPO(II). MSP patrí do NPO(I). SCP patrí do NPO(IV). MinVCP je z NPO(III).

3.5 Polynomiálne aproximačné schémy

V tejto časti sa budeme venovať problémom, ktoré sa dajú veľmi dobre aproximovať.

3.5.1 Problém batohu

Prvým takýmto problémom je Problém batohu (*angl.* Knapsack Problem - KP). Jeho formálna definícia je nasledovná:

PROBLÉM BATOHU (KP)
VSTUP: $b \in \mathbb{Z}^+$, $w_1, \ldots, w_n, c_1, \ldots, c_n \in \mathbb{Z}^+$, $n \in \mathbb{N} \setminus \{0\}$
OBMEDZENIA: prípustné riešenia patria do $\mathcal{M}(b, w_1, \ldots, w_n, c_1, \ldots, c_n) = \{T \subseteq \{1, 2, \ldots, n\} : \Sigma_{i \in T} w_i \leq b\}$
ÚČELOVÁ FUNKCIA: pre všetky $T \in \mathcal{M}(b, w_1, w_2, \ldots, w_n, c_1, \ldots, c_n))$

$$cost(T, b, w_1, \ldots, w_n, c_1, \ldots, c_n) = \Sigma_{i \in T} c_i$$

CIEĽ: maximum.

Problém batohu je jeden z najjednoduchších NP-ťažkých optimalizačných problémov vzhľadom k polynomiálno-časovej aproximácii, a teda je reprezentatívnym príkladom z triedy NPO(I). V nasledovnom použijeme tento problém na predstavenie základných konceptov návrhu aproximačných algoritmov. Začnime zjednodušenou verziou SKP - v anglickej terminológii Simple Knapsack Problem.

JEDNODUCHÝ PROBLÉM BATOHU (SKP)
VSTUP: $b \in \mathbb{Z}^+$, $w_1, \ldots, w_n \in \mathbb{Z}^+$, $n \in \mathbb{N} \setminus \{0\}$
OBMEDZENIA: $\mathcal{M}(b, w_1, \ldots, w_n) = \{T \subseteq \{1, 2, \ldots, n\} : \Sigma_{i \in T} w_i \leq b\}$
ÚČELOVÁ FUNKCIA: pre všetky prípustné $T \in \mathcal{M}(b, w_1, w_2, \ldots, w_n))$ je $cost(T, b, w_1, \ldots, w_n) = \Sigma_{i \in T} w_i$
CIEĽ: maximum.

Algoritmus 12 `GreedySKP`

Require: kladné celé čísla $(w_1, \ldots, w_n, b) \in \mathbb{N}^{n+1}$
  utrieď $w_1, .., w_n$ (zrejme môžeme predpokladať. že $w_1 \geq \cdots \geq w_n$);
  polož $T := \emptyset$ a $cost(T) := 0$
  for $i = 1$ **to** n **do**
    if $cost(T) + w_i \leq b$ **then**
      $T := T \cup \{i\}$ a $cost(T) := cost(T) + w_i$
    end if
  end for
  return T

Algoritmus `GreedySKP` pracuje v čase $O(n \log n)$, pretože toľko potrebujeme na zotriedenie váh. V ďalšom kroku potrebujeme čas $O(1)$ a následný cyklus pracuje v čase $O(n)$.

Lema 3.5.1. *Algoritmus* `GreedySKP` *je 2-aproximačný algoritmus pre SKP.*

Dôkaz. Hlavnou ideou dôkazu bude ukázať, že buď $cost(T) \geq \frac{b}{2}$ alebo T je optimálne riešenie. Predpokladajme, že $b \geq w_1 \geq w_2 \geq \cdots \geq w_n$. Nech $j + 1$ je najmenší index nevyskytujúci sa v T. Poznamenajme, že $T \neq \emptyset$ lebo $w_1 \leq b$. Ak $j = 1$, tak $w_1 + w_2 > b$. Pretože $w_1 \geq w_2$, tak dostávame $cost(T) = w_1 > \frac{b}{2}$.

Predpokladajme teraz, že $j \geq 2$. Vo všeobecnosti

$$cost(T) + w_{j+1} > b \geq Opt_{\mathrm{SKP}}(w_1, \ldots, w_n, b).$$

Pretože platí $w_1 \geq \cdots \geq w_n$, tak máme

$$w_{j+1} \leq w_j \leq \frac{w_1 + w_2 + \cdots + w_j}{j} \leq \frac{b}{j}.$$

Z toho $cost(T) > b - w_{j+1} \geq b - \frac{b}{j} \geq \frac{b}{2}$ pre každé $j \geq 2$. □

Algoritmus 13 `PTAS-SKP`

Require: kladné celé čísla $(w_1, \ldots, w_n, b) \in \mathbb{N}^{n+1}$ a $\epsilon \in \langle 0, 1\rangle$
utrieď $w_1, .., w_n$ (zrejme môžeme predpokladať. že $w_1 \geq \cdots \geq w_n$);
polož $k := \lceil \frac{1}{\epsilon} \rceil$;
pre každú $S \subseteq \{1, 2, \ldots, n\}$ takú, že $|S| \leq k$ a $\sum_{i \in S} w_i \leq b$, nájdi jej rozšírenie S^* použitím pažravého prístupu popísaného v algoritme `GreedySKP`
return S^* s maximálnou cenou $cost(S^*)$

Veta 3.5.2. *Algoritmus* `PTAS-SKP` *je* PTAS *pre SKP.*

Dôkaz. Stanovme najprv časovú zložitosť Time(n) pre inštanciu problému $(w_1, \ldots, w_n, b, \epsilon)$. Usporiadanie váh môže byť vykonané v čase $O(n \log n)$ a nasledujúci krok v čase $O(1)$. Počet množín $S \subseteq \{1 \ldots, n\}$ mohutnosti $|S| \leq k$ je

$$\sum_{0 \leq i \leq k} \binom{n}{i} \leq \sum_{0 \leq i \leq k} n^i = \frac{n^{k+1} - 1}{n - 1} = O(n^k).$$

Keďže tieto množiny môžu byť vytvárané v lexikografickom poradí, pričom je možné vytvoriť ďalšiu z predošlej v čase $O(1)$ a na rozšírenie S na S^* je potrebný čas $O(n)$, tak potom

$$\text{Time}(n) \leq \left[\sum_{0 \leq i \leq k} \binom{n}{i} \right] \cdot O(n) = O(n^{k+1}) = O\left(n^{\lceil 1/\epsilon \rceil + 1}\right).$$

V ďalšom ukážeme, že aproximačný pomer pre ľubovoľný vstup SKP je vždy zhora ohraničený

$$R_{\texttt{PTAS-SKP}}(w_1, \ldots, w_n, b, \epsilon) \leq 1 + \frac{1}{k} \leq 1 + \epsilon.$$

Nech $M = \{i_1, i_2, \ldots, i_p\}, i_1 < \cdots < i_p$ je optimálne riešenie. $cost(M) = Opt_{\text{SKP}}(w_1, \ldots, w_n, b)$. Rozlíšime dva prípady:

Nech $p \leq k$, potom analyzovaný algoritmus nájde M deterministicky v treťom kroku a teda S^* je optimum a relatívna chyba je rovná 0.

Nech $p > k$. Algoritmus rozšíri množinu $P = \{i_1, \ldots, i_k\}$ obsahujúcu k najťažších váh obsiahnutých v cene $cost(M) = w_{i_1} + \cdots + w_{i_p}$. Keďže $P^* = M$, tak je opäť chyba rovná 0. Ak $P^* \neq M$ potom musí existovať $i_q \in M \setminus P^*$, že $i_q > i_k \geq k$ a

$$cost(P^*) + w_{i_q} > b \geq cost(M).$$

Pokiaľ ale

$$w_{i_q} \leq \frac{w_{i_1} + \cdots + w_{i_k} + w_{i_q}}{k+1} \leq \frac{cost(M)}{k+1},$$

tak dostávame

$$\begin{aligned} R(w_1, \ldots, w_n, b, \epsilon) = \frac{cost(M)}{cost(S^*)} \leq \frac{cost(M)}{cost(P^*)} \leq \frac{cost(M)}{cost(M) - w_{i_q}} \leq \\ \leq \frac{cost(M)}{cost(M) - \frac{cost(M)}{k+1}} = \frac{1}{1 - \frac{1}{k+1}} = \frac{k+1}{k} = \\ = 1 + \frac{1}{k} \leq 1 + \epsilon. \end{aligned}$$

□

Algoritmus 14 DPKP

Require: $(w_1, \ldots, w_n, c_1, \ldots, c_n, b) \in \mathbb{N}^{2n+1}$
  polož $Triple(1) := \{(0, 0, \emptyset)\} \cup \{(c_1, w_1, \{1\})\}$
  for $i = 1$ **to** $n - 1$ **do**
    $Set(i) := Triple(i)$
    for $(k, w, T) \in Triple(i)$ **do**
      if $w + w_{i+1} \leq b$ **then**
        $Set(i+1) := Set(i+1) \cup \{(k + c_{i+1}, w + w_{i+1}, T \cup \{i+1\})\}$
      end if
    end for
    polož $Triple(i{+}1)$ rovnú množine $Set(i{+}1)$ obsahujúcej práve jednu trojicu (m, w', T') pre každý dosiahnuteľný profit m s minimálnou váhou w'
  end for
  spočítaj $c := \max\{k | (k, w, T) \in Triple(n)\}$
  return indexová množina T_c, že $(c, w_c, T_c) \in Triple(n)$

Teraz prejdime späť k všeobecnej verzii Problému batohu, kde okrem váh budeme uvažovať aj ceny. Najprv uvedieme algoritmus dynamického programovania pre KP, ktorý budeme potrebovať neskôr pri FPTAS algoritme pre KP.

Algoritmus 15 `FPTAS-KP`

Require: $(w_1, \ldots, w_n, c_1, \ldots, c_n, b) \in \mathbb{N}^{2n+1}$ a $\epsilon \in \langle 0, 1\rangle$
  Stanovme $c_{\max} = \max\{c_1, \ldots, c_n\}$ a $t := \left\lfloor \log_2 \frac{\epsilon \cdot c_{\max}}{(1+\epsilon)n} \right\rfloor$
  for $i = 1$ **to** n **do**
    $c'_i := \lfloor c_i . 2^{-t} \rfloor$
  end for
  $T' :=$ `DPKP`$(w_1, \ldots, w_n, c'_1 \ldots, c'_n, b)$
  return T'

Definícia 3.5.3. *Nech U je optimalizačný problém s celočíselnými vstupmi. Nech A je algoritmus, ktorý rieši U. Hovoríme, že A je* ***pseudopolynomiálny*** *algoritmus pre U ak existuje polynóm p dvoch premenných taký, že*

$$\mathrm{Time}_A(x) = O(p(|x|, \mathrm{MaxInt}(x))),$$

kde $x = x_1 \# x_2 \# \ldots \# x_n$ je vstup problému, zakódovaná n-tica kladných celých čísel, $x_i \in \{0,1\}^*$ a

$$\mathrm{MaxInt}(x_i) = \max\{Number(x_i) | i \in \{1, \ldots, n\}\}$$

pre všetky $i \in \{1, \ldots, n\}$.

Využitím zavedených pojmov môžeme sformulovať nasledujúcu vetu.

Veta 3.5.4. *Pre ľubovoľný vstup $I = (w_1, \ldots, w_n, c_1, \ldots, c_n, b)$ problému KP platí*

$$\mathrm{Time}_{\texttt{DPKP}}(I) \in O(|I|^2 \cdot \mathrm{MaxInt}(I)),$$

teda `DPKP` *je pseudopolynomiálny algoritmus pre KP.*

Dôkaz. Krok 1 je vykonateľný v čase $O(1)$. Pre inštanciu problému KP $I = (w_1, \ldots, w_n, c_1, \ldots, c_n, b)$ máme spočítať $n-1$ množín $Triple(i)$ v kroku 2. Na výpočet $Triple(i+1)$ z $Triple(i)$ potrebujeme $O(|Triple(i)|)$ krokov. Keďže

$$|Triple(i)| \leq \sum_{i=1}^{n} c_i \leq n.\mathrm{MaxInt}(I)$$

pre všetky $i \in \{1, \ldots, n\}$. Zložitosť druhého kroku algoritmu je $O(n^2 \cdot \text{MaxInt}(I))$. Zložitosť tretieho kroku je $O(n \cdot \text{MaxInt}(I))$ pretože treba nájsť iba maximum medzi $|Triple(n)|$ prvkami.

Pretože $n \leq |I|$, časová zložitosť `DPKP` na I je $O(|I|^2 \cdot \text{MaxInt}(I))$. □

Veta 3.5.5. *Algoritmus* `FPTAS-KP` *je* FPTAS *pre KP.*

Dôkaz. Najprv ukážeme, že algoritmus je aproximačná schéma. Pretože výstup T' (optimálne riešenie pre I') je prípustné riešenie pre I' a I sa nelíši od I' vo váhach, T' je prípustné riešenie pre I tiež. Nech T je optimálne riešenie pre pôvodný vstup I. Chceme ukázať, že

$$R(I) = \frac{cost(T, I)}{cost(T', I)} \leq 1 + \epsilon.$$

$$\begin{aligned}
cost(T, I) &= \sum_{j \in T} c_j \geq \\
&\geq \sum_{j \in T'} c_j = cost(T', I) \geq && \text{lebo } T' \text{ je prípustné riešenie} \\
&\geq 2^t \sum_{j \in T'} c'_j \geq && \text{lebo } c'_j = \lfloor c_j . 2^{-t} \rfloor \\
&\geq 2^t \sum_{j \in T} c'_j = && \text{lebo } T' \text{ je optimálne pre } I' \\
&= \sum_{j \in T} 2^t \lfloor c_j \cdot 2^{-t} \rfloor \geq \\
&\geq \sum_{j \in T} 2^t (c_j \cdot 2^{-t} - 1) \geq \\
&\geq \left(\sum_{j \in T} c_j \right) - n \cdot 2^t = \\
&= cost(T, I) - n \cdot 2^t.
\end{aligned}$$

Takže

$$\begin{aligned}
&cost(T, I) \geq cost(T', I) \geq cost(T, I) - n \cdot 2^t \text{ a} \\
&0 \leq cost(T, I) - cost(T', I) \leq n \cdot 2^t \leq n \cdot \frac{\epsilon \cdot c_{\max}}{(1 + \epsilon) \cdot n} = \epsilon \cdot \frac{c_{\max}}{1 + \epsilon}.
\end{aligned}$$

Pretože $w_i \leq b$ pre všetky $i \in 1, \ldots, n$, tak môžeme predpokladať, že $cost(T, I) \geq c_{\max}$ a dostávame $cost(T', I) \geq c_{\max} - \epsilon \cdot \frac{c_{\max}}{1+\epsilon}$.

$$\begin{aligned} R(I) = \frac{cost(T,I)}{cost(T',I)} &= \frac{cost(T',I) + cost(T,I) - cost(T',I)}{cost(T',I)} \leq \\ &\leq 1 + \frac{\epsilon \cdot \frac{c_{\max}}{1+\epsilon}}{cost(T',I)} \leq 1 + \frac{\epsilon \cdot \frac{c_{\max}}{1+\epsilon}}{c_{\max} - \epsilon \cdot \frac{c_{\max}}{1+\epsilon}} = \\ &= 1 + \frac{\epsilon}{1+\epsilon} \cdot \frac{c_{\max}}{c_{\max} - \epsilon \cdot \frac{c_{\max}}{1+\epsilon}} = \\ &= 1 + \frac{\epsilon}{1+\epsilon} \cdot \frac{1}{1 - \frac{\epsilon}{1+\epsilon}} = 1 + \frac{\epsilon}{1+\epsilon-\epsilon} = 1 + \epsilon. \end{aligned}$$

Ešte musíme ukázať, že zložitosť algoritmu je polynomiálna vzhľadom na n aj ϵ^{-1}. Prvý a druhý krok algoritmu sa dá vyrátať v čase $O(n)$. Tretí krok spočíva vo vykonaní algoritmu `DPKP` a teda pre vstup I' potrebujeme $O(n \cdot Opt_{\mathrm{KP}})$ krokov.

Ohraničíme $Opt_{\mathrm{KP}}(I')$ nasledovne:

$$\begin{aligned} Opt_{\mathrm{KP}}(I') \leq \sum_{i=1}^{n} c'_i &= \sum_{i=1}^{n} \left\lfloor c_i \cdot 2^{-\log_2 \frac{\epsilon \cdot c_{\max}}{(1+\epsilon)\cdot n}} \right\rfloor \leq \\ &\leq \sum_{i=1}^{n} \left(c_i \cdot 2 \cdot \frac{(1+\epsilon)\cdot n}{\epsilon \cdot c_{\max}} \right) = \\ &= 2 \cdot (1+\epsilon) \cdot \epsilon^{-1} \cdot \frac{n}{c_{\max}} \cdot \sum_{i=1}^{n} c_i \leq \\ &\leq 2 \cdot (1+\epsilon) \cdot \epsilon^{-1} \cdot \frac{n}{c_{\max}} \cdot n \cdot c_{\max} = \\ &= 2 \cdot (1+\epsilon) \cdot \epsilon^{-1} \cdot n^2 \in O(\epsilon^{-1} \cdot n^3). \end{aligned}$$

□

3.5.2 Problém obchodného cestujúceho

V tejto kapitole predstavíme dva aproximačné algoritmy pre špeciálne verziu (niekedy sa používa aj termín *relaxácia*) Problému obchodného cestujúceho $\triangle$-TSP (pozri časť 3.1). V algoritmoch budeme podstatným spôsobom využívať to, že vzdialenosti v grafe spĺňajú trojuholníkovú nerovnosť.

Na vstupe predchádzajúceho algoritmu sa požaduje úplný graf, čo sa môže zdať obmedzujúcou požiadavkou. V skutočnosti to tak nie je, pretože

Algoritmus 16 `MinSpan`$\triangle$

Require: Úplný graf $G(V, E)$, váhová funkcia $c : E \to \mathbb{N}^+$ spĺňajúca $\triangle$-nerovnosť
Zostroj minimálnu kostru T grafu G vzhľadom k funkcii c
Vyber ľubovoľný vrchol $v \in V$. Pomocou prehľadávania T do hĺbky, začínajúc vo v usporiadaj vrcholy do postupnosti $\overline{H}$.
return hamiltonovský okruh $\overline{H}$

chýbajúce hrany môžeme nahradiť buď hranami s ohodnotením ∞ (dobre nám poslúži akákoľvek veľká konštanta) alebo v reálnej situácií vieme vždy prepočítať vzdialenosti medzi ľubovoľnými dvomi vrcholmi v súvislom grafe.

Nasledujúca veta hovorí o tom, aký má algoritmus aproximačný pomer.

Veta 3.5.6. *Algoritmus* `MinSpan`$\triangle$ *je polynomiálny 2-aproximačný algoritmus pre Triangle-TSP.*

Dôkaz. Prvý aj druhý krok algoritmu sa dajú zrealizovať v polynomiálnom čase $O(|E|)$. Teda $\text{Time}_a(n) \in O(m) \subseteq O(n^2)$.

Nech teraz H_{OPT} je optimálny hamiltonovský okruh ceny $cost(H_{Opt}) = Opt_{\triangle-\text{TSP}}(I)$ pre vstup $I = ((V, E), c)$. Nech $\overline{H}$ je výstup nášho algoritmu. Zrejme $\overline{H}$ je prípustné riešenie problému, pretože je to permutácia vrcholovej množiny. Všimnime si, že

$$cost(T) = \sum_{e \in E(T)} c(e) \leq cost(H_{Opt}),$$

pretože odstránením jednej hrany z H_{Opt} dostávame cestu, čo je špeciálny prípad kostry a T je minimálna kostra.

Nech W je cesta, ktorá je výsledkom prehľadávania grafu do hĺbky (*angl.* Depth First Search - DFS). Potom

$$cost(W) = 2.cost(T),$$

lebo cesta W navštívi každú hranu T dvakrát. Teda

$$cost(W) = 2cost(T) \leq 2cost(H_{Opt}).$$

$\overline{H}$ sa získa z W výmenou niektorých hrán za kratšie cesty (skratky), ktoré vyplývajú z $\triangle$-nerovnosti. Platí teda:

$$cost(\overline{H}) \leq cost(W).$$

Úhrnom dostávame vzťah

$$cost(\overline{H}) \leq cost(W) \leq 2.cost(H_{opt}),$$

z ktorého priamo vyplýva aproximačný pomer algoritmu.

□

V ďalšom ukážeme, že daný algoritmus je ešte možné zlepšiť.

Algoritmus 17 `Christoph`$\triangle$

Require: Úplný graf $G(V, E)$, váhová funkcia $c : E \to \mathbb{N}^+$ spĺňajúca $\triangle$-nerovnosť
Zostroj najlacnejšiu kostru T grafu G vzhľadom k funkcii c
$S := \{v \in V : \deg_T(v)$ je nepárny$\}$
Nájdi perfektné spárenie M na množine S minimálnej ceny
Zostroj multigraf $G' = (V, E(T) \cup M)$ a zostroj eulerovský okruh ω v G'
Zostroj hamiltonovský okruh H grafu G odstránením opakujúcich sa vrcholov v ω
return hamiltonovský okruh H

Veta 3.5.7. *Christophidesov algoritmus* `Christoph`$\triangle$ *je polynomiálny 1,5-aproximačný algoritmus pre* $\triangle$*-TSP.*

Dôkaz. Analyzujme najprv časovú zložitosť $\text{Time}_{Ch}(n)$ Christophidesovho algoritmu. Najlacnejšiu kostru v grafe je možné nájsť v čase $O(m) = O(n^2)$. Identifikácia vrcholov nepárneho stupňa sa dá urobiť v takom istom čase. Najlacnejšie perfektné spárenie (dôležitým faktom je, že počet vrcholov nepárneho stupňa v grafe je vždy párny) sa dá nájsť v čase $O(n^2m)$. Multigraf G' je možné zostrojiť v čase $O(n^2)$. Eulerovský ťah v G' sa dá vytvoriť v čase $O(|V(T)| + |M|) = O(n)$. A nakoniec hamiltonovský okruh sa získa z eulerovského ťahu tiež v čase $O(|V(T)| + |M|) = O(n)$. Úhrnom teda získavame celkovú časovú zložitosť $\text{Time}_{Ch}(n) = O(n^2m) = O(n^4)$.

Zamerajme sa teraz na aproximačný faktor. Nech H_{Opt} je optimálny hamiltonovský okruh s cenou $cost(H_{Opt}) = Opt_{\triangle-TSP}(I)$ pre vstup $I = (G, c)$ pre $\triangle$-TSP. Nech H je výstup Christophidesovho algoritmu a ω je skonštruovaný eulerovský ťah. Zrejme platí:

$$cost(\omega) = cost(G') = \sum_{e \in E(T) \cup M} c(e) = cost(T) + cost(M).$$

Z $\triangle$-nerovnosti máme

$$cost(H) \leq cost(\omega) \leq cost(T) + cost(M).$$

Pretože odstránenie ľubovoľnej hrany z H_{Opt} rezultuje do kostry, evidentne platí: $cost(T) \leq cost(H_{Opt})$.

Nech teraz $S = \{v_1, v_2, \ldots, v_{2m}\}$ je množina vrcholov nepárneho stupňa v T. Bez ujmy na všeobecnosti môžeme predpokladať, že H_{Opt} má tvar

$$v_1, \alpha_1, v_2, \alpha_2, \ldots, \alpha_{2m-1}, v_{2m}, \alpha_{2m}, v_1,$$

kde α_i je cesta, potenciálne aj prázdna, medzi v_i a v_{i+1} resp. v_m a v_1. Uvážme dve spárenia v H_{Opt}:

$$M_1 = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{2m-1}, v_{2m}\}$$
$$M_2 = \{\{v_2, v_3\}, \{v_3, v_4\}, \ldots, \{v_{2m}, v_1\}\}.$$

Z $\triangle$-nerovnosti plynie, že

$$cost(v_i, \alpha_i, v_{i+1}) \geq c(\{v_i, v_{i+1})$$

pre všetky $i = 1, 2, \ldots, 2m$ (pri zohľadnení cyklického číslovania). Z toho plynie:

$$cost(H_{Opt}) \geq \sum_{i=1}^{2m} c(\{v_i, v_{i+1}\}) = cost(M_1) + cost(M_2).$$

Na druhej strane, M_1 a M_2 sú dve ľubovoľné spárenia v S a M je optimálne spárenie v S. Preto $cost(M_1) \geq cost(M)$ a $cost(M_2) \geq cost(M)$. Z toho vyplýva, že

$$cost(M) \leq \min\{cost(M_1), cost(M_2)\} \leq \frac{1}{2} cost(H_{Opt}).$$

Úhrnom teda dostávame

$$cost(H) \leq cost(T) + cost(M) \leq cost(H_{Opt}) + \frac{1}{2} cost(H_{Opt}) = \frac{3}{2} cost(H_{Opt}),$$

z čoho okamžite vieme stanoviť aproximačný pomer Christophidesovho algoritmu.

□

3.6 Neaproximovateľnosť

Skúsime teraz zodpovedať otázku, či pre všetky problémy je možné nájsť nejaký aproximačný algoritmus.

Vo všeobecnosti sa používajú tri metódy na dôkaz dolného ohraničenia pre polynomiálnu aproximovateľnosť.

(a) Redukcia NP-ťažkého rozhodovacieho problému - NP-ťažký problém sa redukuje na problém nájdenia prípustného riešenia s d-aproximačným pomerom.

(b) Redukcia zachovávajúca aproximáciu - ak už máme optimalizačný problém, ktorý neumožňuje konštantnú aproximáciu, tak stačí náš problém transformovať na známy problém pri zachovaní aproximačného pomeru.

(c) Aplikácia tzv. PCP-vety - ide o aplikáciu hlbokého teoretického výsledku.

Definícia 3.6.1. *Nech $U = \langle \Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal \rangle$ je optimalizačný problém. Pre každé $c \in \mathbb{R}^{>1}$ definujeme* ***c-aproximačný problém asociovaný s*** *U, skrátene $c-App(U)$, ako problém nájdenia prípustného riešenia $S(x) \in \mathcal{M}(x), x \in L_I$ takého, že*

$$\max \left\{ \frac{cost(S(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(S(x))} \right\} \leq c.$$

Pre každú funkciu $f : \mathbb{N} \to \mathbb{N}$ definujeme ***$f(n)$-aproximačný problém asociovaný s*** *U, skrátene $f(n)-App(U)$, ako problém nájdenia prípustného riešenia $S(x) \in \mathcal{M}(x), x \in L_I$ takého, že*

$$\max \left\{ \frac{cost(S(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(S(x))} \right\} \leq f(|x|).$$

Zrejme existencia c-aproximačného riešenia pre U implikuje, že problém $c - App(U)$ sa dá riešiť polynomiálne.

Všeobecná schéma klasickej redukcie je nasledovná: Nech $L \in \Sigma^*$ je NP-ťažký problém a nech $c-APP(U)$ je aproximačný problém minimalizačného problému $U = \langle \Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal \rangle$ z triedy NPO, $c \in \mathbb{R}^{>1}$. Máme nájsť polynomiálnu transformáciu F z Σ^* do L_I takú, že existuje efektívne vypočítateľná funkcia $f_F : \mathbb{N} \to \mathbb{N}$ s nasledujúcimi vlastnosťami:

(i) pre každé $x \in L$ množina $\mathcal{M}(F(x))$ obsahuje riešenie y s $cost(y) \leq f_F(|x|)$,

(ii) pre každé $x \in \Sigma^*$ s pre každé prípustné riešenie $z \in \mathcal{M}(F(x))$ je $cost(z) > c.f_F(|x|)$.

Transformácia F sa dá teraz použiť na posúdenie, či dané $x \in \Sigma^*$ patrí do L ak existuje polynomiálny c-aproximačný algoritmus A pre U. Dá sa totiž transformovať x na $F(x)$ a študovať $A(F(x))$ a $cost(A(F(x))$:

- ak $cost(A(F(x)) \leq c.f_F(|x|)$, tak $x \in L$,
- ak $cost(A(F(x)) > c.f_F(|x|)$, tak $x \notin L$.

Predchádzajúce vzťahy vyplývajú z toho, že $x \in L$ implikuje

$$Opt_U(F(x)) \leq f_F(|x|) < \frac{1}{c} cost(A(F(x))$$

a posledný vzťah odporuje faktu, že existuje c-aproximačný algoritmus pre U.

Vyššie uvedené úvahy teraz využijeme na analýzu problému TSP.

Lema 3.6.2. *Pre každé kladné celé číslo d je problém nájdenia hamiltonovskej kružnice (skrátene HC) možné polynomiálne redukovať na $d - App$(TSP).*

Dôkaz. Nech $G = (V, E)$ je vstup pre problém HC. Použijeme transformáciu F, ktorá na základe grafu G vytvorí úplný graf $K_{|V|} = (V, E')$ a účelovú funkciu $c : E' \to \mathbb{N}^+$ nasledovným spôsobom:

$$c(e) = \begin{cases} 1 & \text{ak } e \in E, \\ (d-1)|V| + 2 & \text{ak } e \notin E. \end{cases}$$

Položíme $f_F(|G|) = |V|$. Rozlíšime dva prípady.

Prípad 1. Ak G obsahuje hamiltonovskú kružnicu, tak $K_{|V|}$ obsahuje hamiltonovský okruh s cenou $|V|$. V takom prípade je

$$Opt_{TSP}(K_{|V|}, c) = |V| = f_F(|G|).$$

Prípad 2. Ak G neobsahuje hamiltonovskú kružnicu, tak každý hamiltonovský okruh v $K_{|V|}$ obsahuje aspoň jednu hranu z $E' \setminus E$. Potom cena takéhoto okruhu je aspoň

$$|V| - 1 + (d-1)|V| + 2 = d|V| + 1 > d|V| = d.f_F(|G|).$$

Evidentne ak by sa problém d-App(TSP) dal vyriešiť v polynomiálnom čase, tak aj problém HC by musel byť riešiteľný v polynomiálnom čase.

□

Predchádzajúca lema okamžite implikuje nasledovný výsledok.

Dôsledok 3.6.3. *Ak* P≠NP*, tak neexistuje polynomiálny d-aproximačný algoritmus aproximujúci problém TSP pre nejakú konštantu d.*

Nasledujú veta ukazuje, že problém TSP nie je možné aproximovať ani pomocou polylogaritmickej funkcie.

Veta 3.6.4. *Ak* P≠NP*, tak neexistuje polynomiálny $p(n)$-aproximačný algoritmus pre nijaký polynóm p.*

Dôkaz. Uvažujme redukciu problému HC na TSP podobne ako v predchádzajúcom dôkaze. Avšak účelovú funkciu definujeme nasledovne:

$$c(e) = \begin{cases} 1 & \text{ak } e \in E, \\ |V|.2^{|V|} + 1 & \text{ak } e \notin E. \end{cases}$$

Transformácia F je polynomiálne vypočítateľná, pretože každé číslo veľkosti $|V|.2^{|V|}$ sa dá uložiť do priestoru $|V| + \lceil \log_2 |V| \rceil$ a teda reprezentácia grafu G=$(K_{|V|}, C)$ je veľkosti najviac $O(|V|^3)$.

Ak G obsahuje hamiltonovskú kružnicu, tak $Opt_{TSP}(F(G)) = |V| \leq f_F(|G|)$.

Ak graf G neobsahuje hamiltonovskú kružnicu, tak každý hamiltonovský okruh má cenu aspoň

$$|V| - 1 + 2^{|V|}.|V| + 1 > 2^{|V|}.|V| = 2^{|V|}.f_F(|G|).$$

Teda ak $L_{HC} \notin$P, tak neexistuje nijaký polynomiálny $p(n)$-aproximačný algoritmus pre TSP.

□

Vzhľadom na vyššie uvedené výsledky má zmysel zaviesť označenie pre tie problémy, ktoré je možné aproximovať pomocou konštantného aproximačného faktora.

Definícia 3.6.5.

$APX = \{U \in \text{NPO} :$ *existuje polynomiálny c-aproximačný algoritmus pre U a $c \in \mathbb{R}^{>1}$*$\}$.

3.7 Randomizované aproximačné algoritmy

V tejto kapitole ilustrujeme ako je možné skombinovať pravdepodobnostný prístup a aproximačné algoritmy. Využijeme na to Problém maximálnej splniteľnosti (*angl.* Maximum Satisfability Problem), ktorým sme sa už zaoberali v časti 2.2.2. Jeho formálna definícia je nasledovná:

PROBLÉM MAXIMÁLNEJ SPLNITEĽNOSTI (MAX-SAT)
VSTUP: Formula $\Phi = F_1 \wedge F_2 \wedge \cdots \wedge F_m$ nad $X = \{x_1, x_2, \dots\}$, kde $m, n \in \mathbb{N}$.
OBMEDZENIA: Pre každú formulu Φ nad $\{x_1, x_2, \dots, x_n\} \subseteq X$ je $\mathcal{M}(\Phi) = \{0,1\}^n$ (t.j. každé ohodnotenie premenných je prípustným riešením)
ÚČELOVÁ FUNKCIA: Pre všetky formuly Φ v konjunktívnej normálnej forme a pre každé ohodnotenie $\alpha \in \mathcal{M}(\Phi)$ je $cost(\alpha, \Phi$ rovné počtu splnených klauzúl.
CIEĽ: maximum.

Na riešenie problému použijeme už prezentovaný algoritmus náhodného ohodnocovania premenných `RSAM`.

Lema 3.7.1. *Algoritmus* `RSAM` *je randomizovaný 2-očakávaný aproximačný algoritmus.*

Dôkaz priamo vyplýva z úvah v časti 2.2.2, pretože stredná hodnota počtu splnených formúl je $m/2$, čo je polovica maximálneho možného počtu splnených formúl. Táto horná hranica je zároveň horným ohraničením aj pre optimálne riešenie problému.

Kapitola 4

k-cestné vrcholové pokrytie

V tejto kapitole predstavíme problematiku, ktorá je aktuálna a vznikla na pôde Ústavu informatiky PF UPJŠ v rámci slovensko-slovinského projektu. Ide o problém k-cestného vrcholového pokrytia grafu.

4.1 Motivácia

Problém je motivovaný dizajnom komunikačných a bezpečnostných protokolov v bezdrôtových senzorových sieťach. Jeden takýto algoritmus navrhol M. Novotný v [6] a v angličtine jeho názov je **Generalised Canvas Protocol**. Pre tento algoritmus je dôležité, aby každá správa bola minimálne raz za k komunikačných kôl skontrolovaná v jednom z uzlov siete. Toto je možné dosiahnuť tak, že do senzorovej siete vložíme kvalitnejšie senzory, ktoré budú schopné kontrolu vykonávať.

Problém teda spočíva v stanovení minimálneho počtu senzorov, ktoré je potrebné vložiť do senzorovej siete tak, aby sa na každej ceste rádu k nachádzal aspoň jeden takýto senzor. Tento problém dostal názov **Problém k-cestného vrcholového pokrytia** (*angl.* k-Path Vertex Cover Problem - skrátene kPVCP) a po prvýkrát bol sformulovaný v [1].

4.2 Základné pojmy a vlastnosti

Formálne môžeme problém zaviesť nasledovne:

Definícia 4.2.1. *Množina S, podmnožina vrcholovej množiny grafu G, sa nazýva k-**cestné vrcholové pokrytie** ak S obsahuje minimálne jeden vrchol z každej cesty P_k, $k \geq 1$, grafu G. Minimálna veľkosť takejto množiny sa označuje $\psi_k(G)$.*

Problém k-cestného vrcholového pokrytia je špeciálnym prípadom problému MinVCP. Ďalším špeciálnym prípadom tohoto problému je intenzívne študovaný problém, ktorý je známy pod anglickým názvom **Feedback Set Problem**. Zatiaľ čo v prípade kPVCP chceme istým spôsobom kontrolovať všetky cesty rádu k, v druhom prípade sú kontrolovanou štruktúrou kružnice v grafe. Pozorný čitateľ okamžite spozoruje, že takýmto spôsobom je možné generovať celú triedu problémov. Stačí iba zmeniť zakázanú resp. kontrolovanú subštruktúru. Nie všetky takto sformulované problémy sú však rovnako zaujímavé z praktického alebo teoretického hľadiska.

Pozrime sa teraz bližšie na kPVCP. Pre $k = 1$ je kPVCP identický s problémom určenia počtu vrcholov grafu. Pre $k = 2$ dostávame problém MinVCP, pre ktorý sme študovali v kapitole 3. V [1] je ukázané, že pre $k \geq 3$ patrí kPVCP medzi NP-úplné problémy, podobne ako je to v prípade MinVCP.

Podobná situácia nastáva aj pokiaľ ide o aproximáciu. Ako bolo skôr ukázané, Algoritmus 9 nájde v polynomiálnom čase 2-aproximačné riešenie problému MinVCP. Tento algoritmus je možné zovšeobecniť nasledovným spôsobom.

Algoritmus 18 `kApprox`

Require: graf G
  $S := \emptyset$;
  while G obsahuje nejakú cestu P_k **do**
    vyber ľubovoľnú cestu P rádu k;
    $S := S \cup V(P)$;
    $G := G \setminus V(P)$
  end while
  return S

Lema 4.2.2. *Algoritmus* `kApprox` *nájde v polynomiálnom čase k-aproximáciu minimálneho k-cestného vrcholového pokrytia v grafe G.*

Dôkaz. Algoritmus `kApprox` evidentne nájde k-cestné vrcholové pokrytie S grafu G, pretože končí vtedy, keď v redukovanom grafe nezostane nijaká cesta P_k. Tie sú postupne eliminované z pôvodného grafu pričom z každej takejto cesty je aspoň jeden vrchol vložený do pokrytia S.

Ďalej je zrejmé, že z každej cesty P_k grafu G musí byť minimálne jeden vrchol v minimálnom k-cestnom vrcholovom pokrytí. Keďže algoritmus `kApprox` vezme v najhoršom prípade z každej takejto cesty nanajvýš k vrcholov, tak získané riešenie je nanajvýš k-krát horšie ako optimálne. □

V prípade MinVCP je otvoreným problémom, či je možné nájsť algoritmus na riešenie problému, ktorý bude mať konštantný aproximačný pomer lepší ako 2.

Ukážeme, že vo všeobecnejšom prípade je situácia trochu rozdielna. Prvou ukážkou je výsledok, ktorý bol publikovaný v [7] pre triedu kubických grafov. Ide o triedu grafov, ktorých všetky vrcholy majú stupeň 3. Na dôkaz správnosti algoritmu $\texttt{VCP3} - \texttt{CG}$ budeme potrebovať dva štrukturálne výsledky, ktoré je možné nájsť v [1] a [7].

Lema 4.2.3. *Nech n je prirodzené číslo. Potom*

1. $\psi_3(P_n) = \lfloor \frac{n}{3} \rfloor \leq \frac{n}{3}$;
2. $\psi_3(C_n) = \lceil \frac{n}{3} \rceil \leq \frac{n+2}{3}$.

Veta 4.2.4. *Nech G je kubický graf rádu n. Potom*

$$\frac{2}{5}n \leq \psi_3(G) \leq \frac{n}{2}$$

a obe hranice sa nedajú zlepšiť.

Algoritmus 19 $\texttt{VCP3} - \texttt{CG}$

Require: kubický graf G

1: $F := \emptyset$; $G^* = G$; $V^* = V$;
2: **if** v grafe G^* nie sú vrcholy stupňa 3 **then**
3: pokračuj krokom 6;
4: **end if**
5: vyber ľubovoľný vrchol u stupňa 3 a polož $F := F \cup \{u\}$; $V^* := V^* \setminus \{u\}$; $G^* := G^* \setminus \{u\}$ a pokračuj krokom 2;
6: nájdi minimálne k-cestné vrcholové pokrytie F' grafu G^* a polož $F := F \cup F'$;
7: **return** F

Nasledujúca veta popisuje vlastnosti algoritmu $\texttt{VCP3} - \texttt{CG}$.

Veta 4.2.5. *Algoritmus $\texttt{VCP3} - \texttt{CG}$ je 1,57-aproximačný algoritmus pre problém 3PCP v triede kubických grafov.*

Dôkaz. Uvedomme si najprv, že v kroku 6 algoritmu je každý komponent grafu G^* buď cesta, alebo kružnica, alebo izolovaný vrchol, a preto nájdenie minimálneho 3-cestného vrcholového pokrytia F' je v tomto prípade jednoduché. Množina F je teda 3-cestným vrcholovým pokrytím a to dokazuje korektnosť algoritmu. Čas na vykonanie kroku 2 je konštantný, pretože stačí verifikovať zmeny v množine vrcholov stupňa 3. Podobne je potrebný konštantný čas na vykonanie kroku 5, kde manipulujeme s tromi množinami. Navyše počet opakovaní tohoto kroku je ohraničený počtom vrcholov grafu. A pretože pokrytie v 6.kroku je možné nájsť v lineárnom čase, tak celková časová zložitosť algoritmu je $O(n)$.

Stanovme teraz aproximačný faktor algoritmu. Keď algoritmus opustí cyklus v kroku 5, tak F je nezávislá množina a G^* pozostáva z ciest, kružníc a izolovaných vrcholov. Označme mohutnosť množiny F po ukončení kroku 5 ako m a nech a, b, c_1 a c_2 postupne označujú počet komponentov G^*, ktoré sú izomorfné izolovaným vrcholom, cestám, trojuholníkom a kružniciam rádu aspoň 4. Potom musí platiť:

$$n = m + a + b + c_1 + c_2 \quad \text{a} \quad |V^*| = a + b + c_1 + c_2,$$

kde n označuje rád grafu G. Uvažujme množinu hrán $E(F, V^*)$ medzi dvomi časťami pôvodného grafu. Pretože F je nezávislá množina, tak platí $3m = |E(F, V^*)| \leq 3a + 2b + c_1 + c_2$. Potom $m \leq a + 2b/3 + c_1/3 + c_2/3$. Vzhľadom na naše predpoklady musí byť $c_2 \geq 4$ alebo $c_2 = 0$. Ak $c_2 \geq 4$, tak v kroku 4 dostaneme pokrytie mohutnosti $|F'| \leq b/3 + c_1/3 + (c_2 + 2)/3$. V takom prípade je kardinalita pokrytia F rovná $m + |F'|$ a dostávame

$$\frac{|F|}{n} = \frac{m + |F'|}{m + a + b + c_1 + c_2} \leq \frac{m + b/3 + c_1/3 + (c_2 + 2)/3}{m + a + b + c_1 + c_2}.$$

Uvažujme teraz funkciu $f(m) = \frac{m+b/3+c_1/3+(c+2)/3}{m+a+b+c_1+c_2}$. Pretože $f(m) \geq 0$ a $m \leq a + 2b/3 + c_1/3 + c_2/3$ dostávame

$$\frac{|F|}{n} \leq \frac{a + b + 2c_1/3 + 2(c_2 + 1)/3}{2a + 5b/3 + 4c_1/3 + 4c_2/3} \leq \max\left\{\frac{1}{2}, \frac{3}{5}, \frac{c_2 + 1}{c_2}\right\}.$$

Keďže $c_2 \geq 4$, tak okamžite máme $\frac{c_2+1}{2c_2} \leq \frac{5}{8}$ a následne $|F| \leq \frac{5n}{8}$.

Ak $c_2 = 0$, tak máme $|F'| \leq \frac{b}{3} + \frac{c_1}{3}$. Mohutnosť pokrytia F v kroku 4 je $m + |F'|$ a

$$\frac{|F|}{n} = \frac{m + |F'|}{m + a + b + c_1 + c_2} \leq \frac{a + b + 2c_1/3}{2a + 5b/3 + 4c_1/3} \leq \frac{3}{5} \leq \frac{5}{8}.$$

Z toho plynie, že $|F| \leq \frac{5n}{8}$.

Na druhej strane, ak OPT je minimálne 3-cestné vrcholové pokrytie grafu G, tak podľa vety 4.2.4 platí: $\frac{|F|}{|OPT|} \leq \frac{5n/8}{2n/5} = \frac{25}{16} \leq 1,57$.

Teda aproximačný pomer algoritmu je 1,57. □

4.3 Randomizovaný prístup

Keďže kPVCP je NP-úplný pre každé $k \geq 2$, jednou z otvorených otázok je, ako presne je možné aproximovať optimálne riešenie tohoto problému.

Aj keď dnes sú už známe aj lepšie výsledky, ukážeme zaujímavý výsledok využívajúci pravdepodobnostný prístup. Tento výsledok bol publikovaný v [4]. Najprv uvedieme jeden štrukturálny výsledok z [1].

Lema 4.3.1. *Nech G je graf s maximálnym stupňom Δ. Potom*

$$\psi_3(G) \leq \frac{\lceil \frac{\Delta-1}{2} \rceil}{\lceil \frac{\Delta+1}{2} \rceil} |V(G)|.$$

Navyše toto pokrytie sa dá vypočítať v čase $O(|E(G)|\Delta(G))$.

Algoritmus, ktorý nájde 3-cestné vrcholové pokrytie podla Lemy 4.3.1 označíme $\texttt{LDeg}(G)$.

Algoritmus 20 $\texttt{D3PVC}(G)$

Require: graf G;

odstráň z G všetky komponenty neobsahujúce P_3
if $V(G) = \emptyset$ **then**
  $S := \emptyset$ a vráť S;
end if
if G obsahuje cestu (u, v, w), deg(u) = 1 **then**
  vráť $\{v, w\} \cup \texttt{D3PVC}(G \setminus \{u, v, w\})$;
end if
if G obsahuje cestu (u, v, w), deg(v) = 2 **then**
  vráť $\{u, w\} \cup \texttt{D3PVC}(G \setminus \{u, v, w\})$
end if
for $k := 2$ to $\Delta(G) - 1$ **do**
  $V_k := \{u; u \in V(G), k < deg(u)\}$;
end for
$S := \min_{2 \leq k \leq \Delta(G)} \texttt{D3PVC}(G \setminus V_k) \cup V_k$;
$S := \min\{S, \texttt{LDeg}(G)\}$
return S

Nasledovný klasický výsledok je dokázaný v [5].

Veta 4.3.2. *Nech G je graf s maximálnym stupňom $k+l+1$, $k,l \geq 0$. Potom existuje taký rozklad V_1, V_2 vrcholovej množiny grafu G, že maximálny stupeň podgrafu indukovaného V_1 je maximálne k a maximálny stupeň podgrafu indukovaného V_2 je nanajvýš l.*

Teraz môžeme pristúpiť k formulácii výsledku ohľadom skúmaného algoritmu.

Veta 4.3.3. *Pre graf G s n vrcholmi a maximálnym stupňom Δ algoritmus* `D3PVC` *je* $\max\left\{2, \frac{5}{2}\frac{\lceil\frac{\Delta-1}{2}\rceil}{\lceil\frac{\Delta+1}{2}\rceil}\right\}$*-aproximačným algoritmom pre problém 3PVCP a jeho časová zložitosť je $O(2^{\Delta}n^{O(1)})$ a priestorová zložitosť je $O(n^{O(1)})$.*

Dôkaz. Budeme sa najprv zaoberať časovou zložitosťou algoritmu. Nech $T(n,\Delta)$ je najhorší možný čas behu algoritmu `D3PVC` na grafe s najviac n vrcholmi a maximálnym stupňom Δ.

Evidentne čas vykonávania vrchnej vrstvy rekurzívneho algoritmu na grafe G, odhliadnuc od rekurzívnych volaní, sa dá ohraničiť pomocou $O(n^3)$.

Ak je splnená podmienka v kroku 5 alebo kroku 8, tak

$$T(n,\Delta) \leq T(n-3,\Delta) + O(n^3).$$

Ak tieto podmienky splnené nie sú, tak krok 14 zabezpečí redukciu problému na najviac $\Delta - 3$ podproblémov, kde podproblém k má najviac n vrcholov a maximálny stupeň nanajvýš k. Nakoniec, implementácia `LDeg` je možná v čase $O(\Delta n^2)$, a preto kroky 14 a 15 vedú k nasledovnej rekurzii:

$$T(n,\Delta) \leq \sum_{2\leq k<\Delta} T(n,k) + O(n^3).$$

Riešením oboch uvedených rekurzií dostávame výslednú zložitosť $T(n,\Delta)$, a to $O(2^{\Delta}n^{O(1)})$.

Poďme teraz stanoviť aproximačný pomer. Uvažujme preto riešenie, ktoré vypočíta algoritmus. Aplikujúc kroky 2, 5 a 8 je automaticky garantovaná 2-aproximácia. Máme teda $\delta \geq 3$. Nech $V(G) = A \cup B$, kde A je optimálne riešenie, t.j. $\psi_3(G) = |A|$. Pre $3 \leq i \leq \Delta$ nech a_i a b_i označuje počet vrcholov stupňa i v A a B.

Ak $\sum_{i=k+1}^{\Delta} a_i \geq \sum_{i=k+1}^{\Delta} b_i$ pre nejaké k, pre ktoré platí $2 \leq k \leq \Delta - 1$, tak `D3PVC`$(G_k \setminus V_k) \cup V_k$ s využitím matematickej indukcie dáva $\max(2, \frac{5}{2} \cdot$

$\frac{\lceil \frac{k-1}{2} \rceil}{\lceil \frac{k+1}{2} \rceil}$)-aproximáciu, čo ďalej môžeme transformovať na

$$\frac{5}{2} \cdot \frac{\lceil \frac{k-1}{2} \rceil}{\lceil \frac{k+1}{2} \rceil} \leq \frac{5}{2} \cdot \frac{\lceil \frac{\Delta-1}{2} \rceil}{\lceil \frac{\Delta+1}{2} \rceil}.$$

V opačnom prípade pre $2 \leq k \leq \Delta - 1$ platí

$$\sum_{i=k+1}^{\Delta} a_i < \sum_{i=k+1}^{\Delta} b_i. \tag{4.1}$$

Ak $\sum_{i=3}^{\Delta} a_i \geq \frac{2}{5}n$, tak rozklad pomocou vety 4.3.2 dáva $\left(\frac{5}{2} \cdot \frac{\lceil \frac{\Delta-1}{2} \rceil}{\lceil \frac{\Delta+1}{2} \rceil} \right)$-aproximáciu, pretože

$$\psi_3(G) \leq \frac{\lceil \frac{\Delta-1}{2} \rceil}{\lceil \frac{\Delta+1}{2} \rceil} n.$$

V opačnom prípade platí

$$\sum_{i=3}^{\Delta} a_i < \frac{2}{5}n = \frac{2}{5} \sum_{i=3}^{\Delta} (a_i + b_i),$$

čo je ekvivalentné s

$$3 \sum_{i=3}^{\Delta} a_i < 2 \sum_{i=3}^{\Delta} b_i. \tag{4.2}$$

Pre množinu $E_{A,B}$, všetkých hrán medzi A a B, máme vzťah

$$\sum_{i=3}^{\Delta} (i-1) b_i \leq |E_{A,B}| \leq \sum_{i=3}^{\Delta} i a_i. \tag{4.3}$$

Ak sčítame nerovnosť (4.1) pre $2 \leq k \leq \Delta - 1$, pričítame (4.2) a využijeme nerovnosť (4.3), tak dostávame

$$\sum_{i=3}^{\Delta} i a_i < \sum_{i=3}^{\Delta} (i-1) b_i \leq \sum_{i=3}^{\Delta} i a_i,$$

čo je spor. □

Tento výsledok je využitý pri konštrukcii nasledovného randomizovaného algoritmu.

Algoritmus 21 `A3PVC`

Require: graf G
1: **if** $\Delta(G) \leq 11$ **then**
2: **return** `D3PVC`(G)
3: **end if**
4: vyber ľubovoľný vrchol u, $d(u) \geq 12$
5: $S := \emptyset$
6: **for each** $v \in N(u)$ **do**
7: vlož v do S s pravdepodobnosťou $\frac{1}{|N(u)|-1}$
8: **end for**
9: **return** $S \cup \{u\} \cup$ `A3PVC`$(G \setminus (S \cup \{u\}))$

Veta 4.3.4. *Algoritmus* `A3PVC` *je randomizovaný polynomiálny algoritmus, ktorý nájde aproximačné riešenie 3PVCP s očakávaným aproximačným pomerom* $\frac{23}{11} = 2.\overline{09}$.

Dôkaz. Nech $A(n,t)$ je veľkosť pokrytia, ktoré vráti algoritmus `A3PVC` za predpokladu, že graf G na vstupe má n vrcholov a $\psi_3(G) \leq t$. Indukciou vzhľadom na n a t dokážeme, že $E[A(n,t)] \leq (2 + \frac{1}{11})t$.

Nech n označuje počet vrcholov grafu G a nech F je optimálne riešenie pre G a $\psi_3(G) = |F|$.

Ak $\Delta(G) \leq 11$, tak z vety 4.3.3 dostávame, že `D3PVC`(G) je $(2 + \frac{1}{12})$-aproximačný algoritmus, t.j. $E[A(n,t)] \leq (2+\frac{1}{12})t$. Časová zložitosť v tomto prípade je $O(2^{11}n^O(1))$.

V opačnom prípade pokračuje algoritmus krokmi 4–7. Nech $a = |S \cap F|$ a $b = |S \setminus F|$ po ukončení kroku 6 algoritmu. V kroku 9 sa pridáva $a+b+1$ vrcholov do výsledného riešenia a problém sa redukuje na menší podproblém. Uvažujme dva prípady.

Prípad 1. Ak $u \in F$ tak $\psi_3(G \setminus (S \cup \{u\})) \leq t - a - 1$. Preto

$$A(n,t) \leq a+b+1+A(n-a-b-1, t-1-a) \leq a+b+1+A(n-a-b-1, t-1).$$

Pretože $E[a+b] = \frac{\deg(u)}{\deg(u)-1}$ a podľa indukčného predpokladu

$$E[A(n-a-b-1, t-1)] \leq (2 + \frac{1}{11})(t-1),$$

tak dostávame

$$E[A(n,t)] \leq \frac{\deg(u)}{\deg(u)-1} + 1 + (2 + \frac{1}{11})(t-1) \leq (2 + \frac{1}{11})t.$$

Prípad 2. Ak $u \notin F$, tak $\psi_3(G \setminus (S \cup \{u\})) \leq t - a$. Preto dostávame

$$A(n,t) \leq a + b + 1 + A(n - a - b - 1, t - a).$$

Pretože $E[a+b] = \frac{\deg(u)}{\deg(u)-1}$ a podľa indukčného predpokladu je

$$E[A(n-a-b-1, t-a)] \leq (2 + \frac{1}{11})(t-a),$$

tak dostávame

$$E[A(n,t)] \leq \frac{deg(u)}{deg(u)-1} + 1 + (2 + \frac{1}{11})(t-a) \leq (2 + \frac{1}{11}) + (2 + \frac{1}{11})(t-a).$$

Keďže aspoň jeden sused u nie je v $|F|$, tak $E[a] \geq 1$ a $E[A(n,t)] \leq (2 + \frac{1}{11})t$. □

4.4 2-aproximácia pre 3PVCP

Lemma 4.2.2 ukazuje, že je relatívne jednoduché nájsť k-aproximáciu problému kPVCP pre každé $k \geq 1$. Na druhej strane pre problém MinVCP (2PVCP) sa predpokladá, že pre neho neexistuje $(2-\epsilon)$-aproximačný algoritmus pre nijaké $\epsilon > 0$.

Algoritmus `2approxVCP3` ukazuje, že pre $k = 3$ je tomu inak. Analýzu algoritmu a dôkaz nasledujúcej vety je možné nájsť v [8].

Veta 4.4.1. *Algoritmus* `2approxVCP3` *nájde v polynomiálnom čase 2-aproximačné riešenie problému 3PVCP.*

4.5 Variácie problému

Problém kPVCP je intenzívne študovaný vo viacerých variáciách. Viacero výsledkov bolo získaných pre špeciálne triedy grafov, napr. stromy, outerplanárne grafy, kaktusy, grafy s ohraničeným stupňom, ...

Zásadne iný problém vzniká vtedy keď ho sformulujeme pre vrcholovo-ohodnotené grafy. Vtedy nás nezaujíma mohutnosť k-cestného vrcholového pokrytia, ale jeho váha. Tento problém bol zatiaľ vyriešený pre stromy a úplné bipartitné grafy.

Ďalší variant problému môžeme získať tak, že od k-cestného vrcholového pokrytia budeme vyžadovať nejakú doplnkovú vlastnosť, napr, že generuje súvislý graf. To opäť zásadným spôsobom není charakter problému a vyžaduje si iné prístupy k riešeniu. Pre tento typ problému boli doteraz získané výsledky len pre veľmi špecifické triedy grafov.

Algoritmus 22 `2approxVCP3`

Require: graf G
  ($\star$ Dekompozičná fáza $\star$)
  $H := G$; $w' := w$; i:=0
  while H obsahuje nejakú cestu na 3 vrcholoch **do**
    $c := \min_{u \in V(H)} \left\{ \frac{w'(u)}{d_H(u)} \right\}$
    $G_i := H$, $V_i := V(G_i)$
    for all $v \in V(G_i)$ **do**
      $t_i(v) := c.d_{G_i}(v)$
      $w'(v) := w'(v) - t_i(v)$
    end for
    $H :=$ podgraf G_i indukovaný vrcholmi u s $w'(u) > 0$
    $i := i + 1$
  end while
  $k := i$, $G_k := H$, $V_k := V(G_k)$
  ($\star$ Rozširovacia fáza $\star$)
  $F_k := \emptyset$
  for $i = k, \ldots, 1$ **do**
    rozšír F_i na pokrytie F_{i-1} grafu G_{i-1} pridaním minimálnej množiny vrcholov z $V_{i-1} \setminus V_i$
  end for
  return F_0

Kapitola 5

Vzorové série úloh

V tejto kapitole úvadzame vybrané vzorové série určené na samostatné precvičovanie. Ďalšie úlohy je možné nájsť napr. v [2] a [3].

5.1 Náhodné výbery a náhodné generátory

1. Vytvorte procedúru, ktorá pomocou nespravodlivej mince (znak padá s neznámou pravdepodobnosťou p) simuluje hod spravodlivej mince.

2. Vytvorte procedúru, ktorá pomocou spravodlivej mince simuluje hádzanie nespravodlivou mincou, ktorej znak padá s pravdepodobnosťou p.

3. Navrhnite postup generovania náhodného prirodzeného čísla z intervalu $1, 2, \ldots, n$ pomocou spravodlivej mince.

4. Navrhnite postup, ako pomocou spravodlivej mince vygenerovať náhodnú permutáciu čísel $1, 2, \ldots, n$. Hodnota n pritom môže byť rádovo niekoľko tisíc.

5. Mincou, ktorej znak padá s pravdepodobnosťou $p \in (0, 1)$, hádžeme, kým nepadne znak. Aký je očakávaný počet hodov?

6. Máme k dispozícii klasickú mincu a predpokladajme, že máme deterministický polynomiálny algoritmus na testovanie prvočíselnosti. Vygenerujte prvočíslo z intervalu $1, 2 \ldots, n$. Aký je očakávaný počet testov prvočíselnosti potrebných pre výber jedného prvočísla?

5.2 Algoritmy typu Las Vegas a Monte Carlo

1. Modifikujte 2MC algoritmus $\mathtt{A_t}$ na pravdepodobnostný algoritmus $\mathtt{A'_t}$ takým spôsobom, aby $\mathtt{A'_t}$ počítal ako výstup najviac frekventovaný výsledok. Aká je pravdepodobnosť chyby modifikovaného algoritmu $\mathtt{A'_t}$?

2. Pomocou metódy Monte Carlo určte obsah plochy nad parabolou $y = x^2$, mimo kružnice so stredom $[2, 4]$ a polomerom $3/2$, ale vo vnútri obdĺžnika určeného bodmi $[0, 0]; [5, 0]; [5, 25]; [0, 25]$ s presnosťou na aspoň 3 desatinné miesta.

3. Dvaja hráči striedavo hádžu kockou. Po každom hode si hráč pripočíta práve hodené číslo (obaja začínajú so sumou 0). Hru vyhráva hráč, ktorý prvý dosiahne alebo presiahne hodnotu 100.

   (a) Aká je pravdepodobnosť, že vyhrá prvý hráč?

   (b) Určte výsledok s presnosťou na 3 desatinné miesta.

   (c) Určte výsledok s presnosťou na 8 desatinných miest.

   (d) Určte výsledok v tvare zlomku.

4. V predchádzajúcich kapitolách sme odvodili, že očakávaný počet porovnaní algoritmu pre k-tý najmenší element je maximálne $8n$. Uvažujme algoritmus, kedy namiesto výberu jediného pivota vyberieme náhodne 3 prvky a pivotom prehlásime medián z týchto troch prvkov. Ukážte, že touto technikou získame nižší očakávaný počet porovnaní.

5. Pre jazyky L a L^C sa nám podarilo zostrojiť algoritmus typu Monte Carlo s jednostrannou chybou. Ukážte, že pre jazyk L je tiež možné zostrojiť algoritmus typu Las Vegas.

6. Podarilo sa nám zostrojiť Monte Carlo algoritmus A s očakávanou časovou zložitosťou $\mathrm{T(n)}$ s vlastnosťou $\mathrm{Prob(A(x) = F(x))} = \gamma(n)$. Naviac sa nám podarilo zostrojiť deterministický algoritmus B, ktorý pre zadané $\mathrm{x, y}$ v čase $\mathrm{t(n)}$ rozhodne, či $\mathrm{y = F(x)}$. Ukážte, že vieme zostrojiť algoritmus typu Las Vegas s očakávanou časovou zložitosťou

$$\frac{\mathrm{t(n) + T(n)}}{\gamma(n)}.$$

7. Máme algoritmus A, ktorý vracia jednu z piatich odpovedí s vlastnosťou $\text{Prob}(\text{A}(\text{x}) = \text{F}(\text{x})) \geq \frac{1}{3}$ a $\text{Prob}(\text{A}(\text{x}) = \text{y}) \leq \frac{1}{4}$ pre každú nesprávnu odpoveď y. Ukážte, ako pre problém F zostrojiť Monte Carlo algoritmus s obojstrannou chybou.

8. Daný je graf G a číslo k. Uvažujme algoritmus, ktorý náhodne zoradí vrcholy grafu do postupnosti $v_1, v_2, \ldots, v_n$ a následne nájde najdlhšiu cestu rešpektujúcu túto postupnosť vrcholov. Pomocou tejto techniky ukážte, ako zostrojiť Monte Carlo algoritmus na nájdenie nejakej cesty dĺžky k s časovou zložitosťou $\text{O}(\text{k!n}^2)$.

5.3 Aproximačné algoritmy

1. Uvažujme Problém vyrovnávania záťaže (*angl.* Load Balancing Problem) pre dva stroje. V tomto prípade je cieľom rozdeliť n úloh na stroje tak, aby rozdiel výsledných časov vykonávania úloh na jednotlivých strojoch bol čo najmenší. Jednotlivé úlohy majú čas realizácie $t_1, t_2, \ldots, t_n$. Profesor Smart navrhol pre tento problém aproximačný algoritmus s názvom `Alg` a tvrdí, že jeho aproximačný pomer je 1,05. Spustíme algoritmus pre vstup s celkovou dobou vykonávania všetkých úloh 200 a `Alg` vráti riešenie, ktorého hodnota účelovej funkcie je 120.

   (a) Predpokladajme, že vieme, že čas vykonávania každej z úloh je nanajvýš 100. Môžeme z toho urobiť záver, že profesor Smart urobil chybu? Odôvodnite svoje tvrdenie.

   (b) Riešte rovnakú otázku za predpokladu, že čas trvania každej úlohy je nanajvýš 10.

2. Daný je bipartitný graf $G = (U, V, E)$. Ukážte ako zostrojiť 2-aproximačný algoritmus pre problém maximálneho párovania v bipartitnom grafe s lineárnou časovou zložitosťou.

3. Máme niekoľko závaží s hmotnosťami $a_1 \leq a_2 \leq \cdots \leq a_n \leq 1$. Potrebujeme umiestniť tieto závažia do prepraviek, aby súčet závaží v každej prepravke bol maximálne 1.

   (a) Koľko prepraviek je nutné použiť?

   (b) Nájdite 2-aproximačný algoritmus. (Návod: Skúste začať ľubovoľným pažravým algoritmom.)

(c) Ukážte, že zostrojiť lepší ako $1,5$-aproximačný algoritmus je vo všeobecnom prípade NP-ťažký problém. (Návod: Treba vhodným spôsobom redukovať problém vyvažovania.)

4. Nájdite 3-aproximačný algoritmus pre Problém najväčšej nezávislej množiny (*angl.* Maximum Independent Set - MIS) v triede grafov s maximálnym stupňom 3. (Návod: Skúste vhodným spôsobom postupne odoberať z grafu vrcholy alebo skupiny vrcholov.)

5. Daných je niekoľko podmnožín množiny A a číslo k. Chceme vybrať k množín zo vstupu tak, aby sme nimi pokryli čo najviac prvkov množiny A. Vytvorte pažravý algoritmus a dokážte jeho aproximačný pomer $\frac{e}{e-1}$.

6. Máme m rovnakých procesorov a n úloh na realizáciu. Jednotlivé úlohy majú časy na realizáciu $p_1, p_2, \ldots, p_n$. Chceme realizovať každú z úloh na jednom z procesorov tak, aby výpočet všetkých úloh skončil čo najskôr.

   (a) Ukážte, že pažravá stratégia, ktorá spracúva úlohy v závislosti od časov zoradených do nerastúcej postupnosti, dáva aproximačný pomer $\frac{3}{2}$.

   (b) Dokážte, že v prechádzajúcej úlohe sa dá dosiahnuť aproximačný pomer $\frac{4}{3}$.

Literatúra

[1] B. Brešar, F. Kardoš, J. Katrenič, G. Semanišin, *Minimum k-path vertex cover*, Discrete Appl. Math. 159 (12) (2011) 1189–1195.

[2] J. Hromkovič, *Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*, Texts in Theoretical Computer Science. An EATCS Series, Springer 2004, ISBN 978-3-642-07909-2.

[3] J. Hromkovič, *Design and Analysis of Randomized Algorithms - Introduction to Design Paradigms*. Texts in Theoretical Computer Science. An EATCS Series, Springer 2005, ISBN 978-3-540-23949-9.

[4] F. Kardoš, J. Katrenič, I. Schiermeyer, *On computing the minimum 3-path vertex cover and dissociation number of graphs*, Theor. Comp. Science 412 (2011) 7009–7017.

[5] L. Lovász. *On decompositions of graphs*, Studia Sci. Math Hungar. 1 (1966) 237–238.

[6] M. Novotný, *Design and Analysis of a Generalized Canvas Protocol*. In: Samarati P., Tunstall M., Posegga J., Markantonakis K., Sauveron D. (eds) Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices. WISTP 2010. Lecture Notes in Computer Science, vol 6033. Springer, Berlin, Heidelberg.

[7] J. Tu, F. Yang, *The vertex cover P_3 problem in cubic graphs*, Information Processing Letters 113 (2013) 481–485.

[8] J. Tu, W. Zhou, *A factor 2-approximation algorithm for the vertex cover P_3 problem*, Information Processing Letters 111 (14) (2011) 683–686.

Aproximačné a pravdepodobnostné algoritmy
Vysokoškolský učebný text

Autori: **RNDr. Ondrej Krídlo, PhD.**
doc. RNDr. Gabriel Semanišin, PhD.
Vydavateľ: Univerzita Pavla Jozefa Šafárika v Košiciach
Umiestnenie: http://www.unibook.upjs.sk
Vydanie: prvé
Rok: 2018
Počet strán: 90
Rozsah: 4,5 AH

ISBN 978-80-8152-622-0