Úvod do aproximačních a pravděpodobnostních algoritmů

Jiří Sgall

18. ledna 2016

1 Úvod

V této přednášce se budeme zabývat dvěma moderními oblastmi teorie algoritmů: algoritmy, které hledají nikoli optimální řešení ale pouze dostatečně dobré řešení, a algoritmy, které používají náhodnost.

1.1 Těžké úlohy a přístupy k jejich řešení

Ze základních přednášek známe klasifikaci úloh na řešitelné v polynomiálním čase – ty považujeme za efektivně řešitelné – a NP-těžké – ty považujeme za těžké.

Mezi polynomiálně řešitelné úlohy patří z kombinatorických problémů hledání nejkratší cesty, hledání minimální kostry grafu, hledání minimálních řezů, toky v síti, párování v grafech a další, ale také lineární programování nebo řešení soustav lineárních rovnic.

Mezi NP-těžké problémy patří splnitelnost a její varianty, problém batohu a řada dalších úloh s dělením vážených objektů do skupin (rozvrhování, bin packing), a opět řada kombinatorických problémů: barevnost grafu a hledání nezávislé množiny, Hamiltonovská cesta a problém obchodního cestujícího, hledání maximálního řezu, možinové pokrytí atd.

Tato klasifikace je důležitá, i když samozřejmě nepostihuje všechny praktické aspekty. Některé polynomiální algoritmy jsou poměrně složité, např. pro toky nebo párování, jiné nejsou úplně uspokojivé z jiných důvodů, např. algoritmy pro lineární programování jsou polynomiální pouze v bitové velikosti vstupu. Naopak některé NP-těžké problémy jsou v praxi docela dobře řešitelné, například problém obchodního cestujícího, problém batohu nebo některé varianty rozvrhování.

Řada obtížných kombinatorických úloh ovšem patří mezi ty, které chceme v praxi řešit, a proto se studuje řada přístupů. Jmenujme některé z nich:

- **Heuristiky:** Studují se algoritmy, které je obtížné teoreticky analyzovat, nefungují nutně pro všechny vstupy, ale v různých situacích se osvědčují, což se prokazuje např. testováním na různých benchmarkových datech. Častou variantou jsou algoritmy, které se snaží chytře prohledat prostor všech řešení, např. algoritmy branch-and-bound a algoritmy lokálního prohledávání. Také sem patří algoritmy založené na metodách umělé inteligence jako simulované žíhání, genetické algoritmy apod.

- **Speciální případy:** Studují se zajímavé podproblémy, např. pro grafové problémy varianty omezené na různé třídy grafů, nebo jinak omezené množiny vstupů u jiných problémů. V jistém smyslu sem patří i studium algoritmů pro vstupy z dané náhodné distribuce.

- **Podrobnější analýza času běhu algoritmu:** V parametrizované složitosti se studuje závislost času běhu nejen na velikosti vstupu ale i na hodnotě dalšího parametru či více parametrů. Je také zajímavé studovat časovou složitost algoritmů, které nejsou polynomiální: algoritmus s časovou složitostí $O(1.5^n)$ může být mnohem užitečnější než prohledávání hrubou silou se složitostí $O(2^n)$, čas $2^{O(n)}$ je velký pokrok proti $2^{O(n \log n)}$ nebo $2^{O(n^2)}$.

- **Přibližné řešení:** U optimalizačních problémů, které měří kvalitu řešení (nejde tedy o rozhodovací problémy), se můžeme spokojit s řešením, které není optimální ale zároveň máme záruku, že se jeho kvalita od optima příliš neliší.

V naší přednášce se budeme zabývat posledním uvedeným přístupem. Budou nás zajímat algoritmy, které fungují pro všechny vstupy, vždy dají řešení, které je skoro optimální, a pracují v polynomiálním čase. Takovým algoritmům říkáme aproximační algoritmy.

Budeme se zabývat několika základními metodami a příklady návrhu aproximačních algoritmů. V první skupině půjde o kombinatorické algoritmy založené na hladových algoritmech a také na lokálním prohledávání (v případech, kdy nejde jen o heuristiku, ale umíme dokázat odhad na kvalitu řešení). Ve druhé skupině půjde o algoritmy založené na lineárním programování a různých metodách zaokrouhlování jejich řešení; v současném algoritmickém výzkumu se jedná o jednu z nejproduktivnějších technik.

1.2 Použití náhodnosti v algoritmech

Náhodnost patří dnes ke standardním technikám návrhu algoritmů, ale má i bohatou historii. Studium stochastických jevů a statistické metody patří mezi klasické matematické obory; při studiu algoritmů se pak aplikují například při analýze rozsáhlých dat pomocí náhodného výběru vzorků, či naopak při již zmíněné analýze algoritmů při výběru vstupu z dané distribuce. Pravděpodobnost a její použití je klíčové ve von Neumannově teorii (maticových) her, kde je nutné uvažovat smíšené, tj. pravděpodobnostní, strategie. V oblasti počítačových her je náhodnost samozřejmá – deterministická hra by většinou prostě nebyla zajímavá.

V naší přednášce se budeme zabývat použitím náhodnosti, které dává zaručené výsledky pro všechny vstupy, a záruka bude mít většinou podobu odhadu na průměrnou kvalitu řešení nebo na průměrný čas běhu algoritmu. Takovým algoritmům říkáme pravděpodobnostní algoritmy.

Použití náhodnosti je z mnoha pohledů atraktivní, ale má samozřejmě svou cenu. Získat zdroj skutečné náhodnosti není snadné, z praktického hlediska skoro nemožné. Běžně se tedy náhodnost nahrazuje pseudonáhodnými generátory, které mají problémy jak praktické tak teoretické. I když pseudonáhodný generátor projde řadou testů, nemůžeme zaručit, že bude pro danou aplikaci fungovat dobře – a z historie známe řadu případů, kdy nastal problém ať už kvůli špatné konstrukci generátoru nebo kvůli jeho špatnému použití. Z teoretického hlediska není situace lepší: použitím generátoru se z pravděpodobnostního algoritmu fakticky stává deterministický a tím pádem do značné míry ztrácíme výhody použití náhodnosti.

V teorii algoritmů je tak častá situace, kdy navrhneme pravděpodobnostní algoritmus, a pak studujeme, zda v něm použití náhodnosti můžeme omezit a ideálně získat deterministický algoritmus. Tomu říkáme derandomizace.

Použití náhodnosti v teorii algoritmů můžeme rozdělit do tří skupin.

- **Efektivnější algoritmy.** Nejpřirozenějším cílem je navrhnout pravděpodobnostní algoritmy pro kombinatorické problémy, které budou rychlejší či jinak měřitelně lepší

než deterministické. To má ovšem základní problém: Dokázat, že dobrý deterministický algoritmus pro daný problém neexistuje, umíme jen ve velmi omezených případech. Realisticky tedy navrhujeme většinou pravděpodobnostní algoritmy, které jsou efektivnější než všechny známé deterministické algoritmy. Některé se později daří derandomizovat, příkladem může být algoritmus na testování prvočíselnosti.

- **Koncepční jednoduchost.** Často je hlavní výhodou pravděpodobnostního algoritmu jeho jednoduchost. To je analogické kombinatorickým důkazům pravděpodobnostní metodou, kdy často studium vlastností náhodně vybraného objektu je podstatně snazší než konstrukce objektu s danou vlastonstí. U algoritmů je pak efektivita spíše vedlejším cílem a často je spíše srovnatelná s nejlepšími deterministickými algoritmy, ne výrazně lepší. Dobrým příkladem je Karger-Steinův algoritmus na hledání minimalního řezu.

- **Situace neřešitelné deterministicky.** Pokud se vzdálíme světu kombinatorických problémů, je řada scénářů, které dokazatelně nemají deterministické řešení, jako už uvedené maticové hry. V algortimickém světě jsou to například tyto:

  – Kryptografie. Moderní kryptografie s veřejným klíčem se bez náhodnosti neobejde.

  – Distribuované protokoly. Bez náhodnosti není možné vybrat jeden počítač mezi mnoha identickými. Také třeba síťové protokoly používají náhodnost.

  – Interaktivní protokoly a důkazy. I mimo oblast kryptografie se studují různé varianty interaktivních protokolů, které se často neobejdou bez náhodnosti. Dobrým příkladem je PCP věta, která říká, že namísto ověřování celého důkazu náležení slova do jazyka v NP je možné přečíst jen konstantně mnoho náhodně zvolených bitů důkazu (pro pečlivě modifikovaný formát důkazů a pečlivě zvolené náhodné bity).

  – Online a streamovací algoritmy. Algoritmy, které dostávají vstup a generují výstup po částech nebo mají jinak omezený přístup ke vstupu, mohou díky náhodnosti zlepšit své vlastnosti.

V naší přednášce uvedeme příklady algoritmů z řady oblastí: aproximační a paralelní algoritmy pro kombinatorické problémy, datové struktury, distribuované protokoly. Zmíníme i jednu techniku derandomizace.

2 Pravděpodobnost – základní pojmy

Definice 2.1. **Pravděpodobnostní prostor** je trojice (Ω, Σ, Pr), kde Ω je množina, jejíž prvky nazýváme **elementární jevy**, $\Sigma \subseteq 2^\Omega$ je množina podmnožin Ω, jejíž prvky nazýváme **jevy**, a $Pr : \Sigma \to [0,1]$ je funkce nazývaná **pravděpodobnost**.

Množina jevů vždy obsahuje Ω a je uzavřená na doplňky a spočetná (a konečná) sjednocení.

Pravděpodobnost splňuje $Pr[\Omega] = 1$ a je aditivní pro spočetné (a konečné) množiny disjunktních jevů, tedy pro posloupnost po dvou disjunktních jevů $E_1, E_2, \ldots$ platí

$$Pr\left[\bigcup_{i=1}^{\infty} E_i\right] = \sum_{i=1}^{\infty} Pr[E_i]\,.$$

Doplňkový jev k A značíme $\bar{A} = \Omega \setminus A$.

Z definice plyne řada dalších intuitivních vztahů: $Pr[\emptyset] = 0$, a obecně $Pr[\bar{A}] = 1 - Pr[A]$, pro jevy $A \subseteq B$ platí $Pr[A] \leq Pr[B]$ a pro obecnou posloupnost jevů $Pr\left[\bigcup_{i=1}^{\infty} E_i\right] \leq \sum_{i=1}^{\infty} Pr[E_i]$.

I když to definice nevyžaduje, my budeme vždy pracovat s prostory, které obsahují všechny elementární jevy jako jevy, tj. Σ obsahuje všechny jednoprvkové podmnožiny Ω. Často nebudeme rozlišovat jednoprvkový jev a jemu odpovídající elementární jev. Většinou je množina jevů zřejmá z kontextu, pak je třeba specifikovat pouze funkci pravděpodobnosti a mluvíme o pravděpodobnostním **rozdělení**.

My se nejčastěji setkáme s diskrétními prostory, kde Ω je konečná nebo spočetná. Pak Σ obsahuje všechny podmnožiny a pravděpodobnost všech jevů je určena pravděpodobností všech elementárních jevů.

Nejběžnějším příkladem konečného pravděpodobnostního prostoru je **rovnoměrné rozdělení** na konečné množině Ω, kde $Pr[A] = |A|/|\Omega|$ pro každý jev A. Příklady jsou hod kostkou nebo mincí (tj. náhodný bit), náhodná podmnožina n-prvkové množiny, náhodná permutace, náhodný graf atd. Má samozřejmě smysl uvažovat i jiná rozdělení; pokud však rozdělení nespecifikujeme, máme u konečné množiny vždy na mysli rovnoměrné.

Na spočetné množině rovnoměrné rozdělení neexistuje. Důležitým příkladem je **geometrické rozdělení** (přesněji jeho speciální případ). Pro $\Omega = \{1, 2, \ldots\}$ definujeme $Pr[i] = 2^{-i}$. Toto rozdělení popisuje např. počet hodů mincí, než poprvé padne hlava.

V klasické teorii pravděpodobnosti se setkáváme především se spojitými rozděleními na $\Omega \subseteq \mathbb{R}$. Jevy jsou pak nikoliv všechny podmnožiny $\mathbb{R}$ ale jen (např. lebesgueovsky) měřitelné množiny a pro formální vybudování pravděpodobnosti potřebujeme teorii míry. Pravděpodobnost elementárních jevů je typicky 0.

Spojitá rozdělení na $\mathbb{R}$ se pak popisují funkcí hustoty pravděpodobnosti $f : \mathbb{R} \to [0, +\infty)$ splňující $\int_{-\infty}^{+\infty} f(t)\,dt = 1$, která je analogií pravděpodobnosti elementárních jevů pro diskrétní pravděpodobnostní prostory. Pro množinu A pak dostáváme $Pr[A] = \int_{-\infty}^{+\infty} \chi_A(t) f(t)\,dt$, kde χ_A je charakteristická funkce množiny A. Používá se také distribuční funkce $F(x) = Pr[(-\infty, x]] = \int_{-\infty}^{x} f(t)\,dt$.

Na nespočetné množině **rovnoměrné rozdělení** může existovat, např. na jednotkovém intervalu $I = [0, 1]$ je dáno funkcí hustoty $f(x) = \chi_I(x)$. Přesněji, rovnoměrné rozdělení existuje na podmnožinách $\mathbb{R}$, které mají konečnou míru; oproti tomu na celém $\mathbb{R}$ nebo na intervalu $[0, +\infty)$ rovnoměrné rozdělení neexistuje.

Náhodná proměnná X je funkce, která každému elementárnímu jevu přiřadí hodnotu; speciálně reálná náhodná proměnná je funkce $X : \Omega \to \mathbb{R}$. (Mluvíme-li o jiných než diskrétních pravděpodobnostních prostorech, je potřeba přidat podmínku, že funkce je měřitelná – a tedy také je-li obor hodnot nespočetný, musíme na něm mít míru.)

Střední hodnota $E[X]$ reálné náhodné proměnné X udává její "průměrnou" hodnotu. Pro diskrétní rozdělení se definuje jako $\sum_{\omega \in \Omega} X(\omega) \cdot Pr[\omega]$ a pro spojité rozdělení na $\mathbb{R}$ s hustotou f jako $\int_{-\infty}^{+\infty} X(t) f(t)\,dt = 1$. Přímo z definice lze odvodit dvě důležité vlastnosti střední hodnoty. Pokud X je nezáporná, platí $Pr[X \geq t \cdot E[X]] \leq 1/t$ (Markovova nerovnost). Střední hodnota je lineární, tj. $E[X + Y] = E[X] + E[Y]$ pro libovolné náhodné proměnné (a bez dalších podmínek např. na nezávislost).

Indikátorová náhodná proměnná X nabývá hodnot 0 a 1; je to vlastně jen jiný pohled na jev $A = \{\omega \mid X(\omega) = 1\}$. Platí $E[X] = Pr[A]$. Spolu s linearitou střední hodnoty jsou indikátorové proměnné dobrým nástrojem pro odhadování počtu splněných jevů v nějakém

systému.

Podmíněná pravděpodobnost jevu A za podmínky B formalizuje situaci, kdy víme, že jev B nastal. Formálně definujeme pro B takové, že $Pr[B] > 0$, podmíněnou pravděpodobnost takto:

$$Pr[A|B] = \frac{Pr[A \cap B]}{Pr[B]}.$$

Pokud $\{B_i\}_{i \in I}$ je rozklad množiny elementárních jevů na jevy nenulové pravděpodobnosti, pak platí

$$Pr[A] = \sum_{i \in I} Pr[A|B_i] Pr[B_i]\,.$$

Jevy A a B jsou nezávislé, jestliže platí $Pr[A \cap B] = Pr[A] \cdot Pr[B]$. To je ekvivalentní jak tomu, že $Pr[A|B] = Pr[A]$, tak tomu, že $Pr[B|A] = Pr[B]$. Odpovídá to tedy intuitivní představě, že pravděpodobnost jednoho z jevů se nezmění, pokud víme, zda druhý jev nastane.

Pro systém více jevů nestačí požadovat, že každé dva jsou nezávislé (obdobně jako lineární nezávislost systému vektorů se nedá odvodit z lineární nezávislosti dvojic vektorů). **Jevy $\{A_i\}_{i \in I}$ jsou nezávislé**, jestliže pro každou podmnožinu $J \subseteq I$ platí

$$Pr\left[\bigcap_{i \in J} A_i\right] = \prod_{i \in J} Pr[A_i]\,.$$

Pokud tato podmínka platí jen pro J s nejvýše k prvky, řekneme, že jevy $\{A_i\}_{i \in I}$ jsou **po k nezávislé**. To je podstatně slabší vlastnost, než nezávislost.

Náhodné proměnné $\{X_i\}_{i \in I}$ na diskrétním pravděpodobnostním prostoru *jsou nezávislé*, jestliže pro každou podmnožinu $J \subseteq I$ a každé hodnoty $y_i, i \in J$ platí

$$Pr\left[(\forall i \in J) X_i = y_i\right] = \prod_{i \in J} Pr[X_i = y_i]\,.$$

Pokud tato podmínka platí jen pro J s nejvýše k prvky, řekneme, že proměnné $\{X_i\}_{i \in I}$ jsou **po k nezávislé**. Pokud pravděpodobnostní prostor není diskrétní, je potřeba namísto jevů $X_i = y_i$ použít jevy $X_i \in Y_i$, kde Y_i je libovolná měřitelná podmnožina oboru hodnot. Pro reálné náhodné proměnné stačí uvažovat jevy $X_i \leq y_i$ pro $y_i \in \mathbb{R}$.

3 Pravděpodobnostní algoritmy – model a příklady

Obvyklým teoretickým modelem pravděpodobnostních výpočtů jsou Turingovy stroje s dodatečnou páskou, která obsahuje spočetně mnoho nezávislých náhodných bitů. Stejně dobře můžeme použít jiný výpočetní model a přidat instrukci s náhodným výběrem jedné ze dvou možností. Je i možné přidat instrukci výběru s jinými pravděpodobnosti, kterou lze v předchozích modelech efektivně simulovat.

Kdy považujeme pravděpodobnostní algoritmus za dobrý? Rozhodně by měl být efektivní.

Pokud požadujeme, aby pravděpodobnostní algoritmus vždy běžel v polynomiálním čase, musíme připustit nějakou chybu; takové algoritmy se někdy nazývají Monte Carlo. U rozhodovacích problémů typicky připouštíme konstantní pravděpodobnost chyby. Buď oboustranou, pak při každém vstupu požadujeme, aby pravděpodobnost správné odpovědi byla alespoň 2/3; třída takto rozpoznatelných jazyků se nazývá BPP. Lepší je, pokud je chyba jednostranná. Např. algoritmus všechny vstupy mimo jazyk vždy zamítne, ale slova z jazyka přijme

s pravděpodobností alespoň 1/2; tato třída se nazývá *RP*. (Uvědomme si, že pokud pro slova jazyka požadujeme pouze nenulovou pravděpodobnost přijetí, dostáváme třídu NP.) Přesné hodnoty konstant jsou tradiční ale nikoli podstatné, protože algoritmy můžeme několikrát opakovat a chybu zmenšit.

Druhá možnost je požadovat, aby algoritmus vždy odpověděl správně. Pak ale musíme oslabit požadavek na efektivitu: požadujeme, aby průměrný čas běhu byl polynomiální. Takové algoritmy se někdy nazývají Las Vegas; třída takto rozpoznatelných jazyků se nazývá ZPP.

3.1 Quicksort

Známý deterministický třídící algoritmus QUICKSORT pracuje na náhodném vstupu v průměrném čase $O(n \log n)$, ale na některých vstupech potřebuje kvadratický čas. My pravděpodobnostní analýzu využijeme k tomu, abychom navrhli pravděpodobnostní variantu algoritmu, která každý jednotlivý vstup setřídí v průměrném čase $O(n \log n)$. Technicky se jedná o naprosto analogický důkaz, ale v jeho interpretaci je podstatný rozdíl.

Pro jednoduchost prezentace předpokládáme, že všechny tříděné prvky jsou různé.

ALGORITMUS QS(S)
- Pokud $|S| \leq 1$, *vystup* S (jako prázdnou nebo jednoprvkovou posloupnost)
- Vyber uniformně náhodně pivot $p \in S$
- $A := \{a \in S \mid a < p\}$; $B := \{b \in S \mid b > p\}$
- *Výstup:* posloupnost $QS(A), p, QS(B)$

Všechny náhodné volby jsou nezávislé.

Věta 3.1. *Pro každý vstup s n prvky je střední doba běhu algoritmu* QUICKSORT *s náhodným výběrem pivotů $O(n \log n)$. Střední počet porovnání je nejvýše $2nH_n$, kde $H_n = \sum_{k=1}^{n} \frac{1}{k}$ je n-té harmonické číslo.*

Důkaz. Zafixujme vstupni posloupnost. Nechť $A_{i,j}$ je jev, který nastane, když v průběhu algoritmu porovnáme i-tý a j-tý prvek ve výsledném pořadí, pro $j > i$.

Pokud jsou v dané úrovni rekurze i-tý a j-tý prvek v S, porovnáme je právě tehdy, pokud jeden z nich zvolíme za pivot, a navíc v takovém případě je porovnáme jen jednou. Pokud pivot leží mezi i-tým a j-tým prvkem, pak tyto dva prvky už nikdy neporovnáme, protože nebudou společně ani v A ani v B. Pokud je pivot menší nebo větší než oba prvky, tak naopak oba postoupí do další úrovně rekurze společně buď v A nebo v B. Máme tedy $j + 1 - i$ voleb pivota tak, že prvky spolu nepostoupí do další úrovně a jen 2 z těchto voleb vedou k porovnání. Uvážíme-li toto pro poslední úroveň rekurze, kam prvky postoupí společně, plyne z toho

$$Pr[A_{i,j}] = \frac{2}{j + 1 - i}\,.$$

Přesněji bychom mohli argumentovat, že pro každý uzel stromu rekurze je toto pravděpodobnost porovnání za podmínky, že toto je poslední rekurzivní volání, kam postoupily oba uvažované prvky. Protože podmínky pro různé uzly tvoří rozklad pravděpodobnostního prostoru, podle součtové formule je stejná i nepodmíněná pravděpodobnost porovnání.

Nechť X je náhodná proměnná, která udává počet porovnání v průběhu algoritmu na daném vstupu. Nechť $X_{i,j}$ je indikátorová náhodná proměnná jevu $A_{i,j}$, tj $X_{i,j} = 1$ pokud porovnáme i-tý a j-tý prvek a $X_{i,j} = 0$ jinak. Pak střední počet porovnání odhadneme jako

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{i,j}] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} Pr[A_{i,j}] \leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \leq 2n \cdot \sum_{k=1}^{n} \frac{1}{k} = 2n \cdot H_n \,,$$

kde $H_n = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}$ se nazývá n-té harmonické číslo a platí $H_n \approx \ln n$.

Doba běhu algoritmu je lineární v počtu porovnání. □

3.2 Konflikty v distribuovaném systému

Předpokládejme, že máme jednoduchý distribuovaný protokol, kde se n stejných procesů snaží o exkluzivní přístup ke sdílenému prostředku (paměti, databázi, ...). Přímá komunikace mezi procesy není možná. Procesy jsou synchronizované, v každém cyklu proces může požadovat přístup. Uspěje ovšem jedině tehdy, když je jediným procesem, který přístup požaduje.

Zajímá nás algoritmus, který co nejdříve uspokojí všechny procesy. Deterministické řešení neexistuje, protože všechny procesy se budou o přístup pokoušet ve stejných cyklech. Předvedeme pravděpodobnostní algoritmus, který s velkou pravděpodobností umožní přístup všem procesům v průměrném čase $O(n \log n)$ pro vhodně zvolený parametr p.

> Algoritmus Přístup (popis jednoho procesu)
> - V každém cyklu se s pravděpodobností p pokus o přístup.
> - Opakuj, dokud není pokus o přístup úspěšný

Náhodné volby všech procesů jsou nezávislé, protože spolu nekomunikují. Náhodné volby jednoho procesu v jednotlivých cyklech volíme jako nezávislé.

Věta 3.2. *Algoritmus* Přístup *s parametrem* $p = 1/n$ *s pravděpodobností alespoň* $1 - 1/n$ *uspěje po* $t = 2en \ln n$ *cyklech.*

Důkaz. Především algoritmus modifikujme tak, že každý proces se pokouší o přístup neustále, i poté, co ho již jednou získal. Tato změna může jedině snížit pravděpodobnost úspěchu, ale podstatně zjednoduší analýzu.

V následujících výpočtech v plné síle využijeme nezávislost náhodných voleb, protože pravděpodobnost počítáme jako součin mnoha nezávislých jevů.

Nechť $A_{i,r}$ je jev označující, že i-tý proces uspěje v r-tém cyklu. Máme

$$Pr[A_{i,r}] = p \cdot (1-p)^{n-1} = \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{en} \,.$$

Druhý krok je dosazení zvoleného parametru p; snadno se ověří, že zvolené p je lokální maximum dané funkce. Poslední odhad $\frac{1}{e} \leq \left(1 - \frac{1}{n}\right)^{n-1}$ je standardní horní odhad na $1/e$ pomocí posloupnosti mocnin. Často se hodí i obdobný dolní odhad $\left(1 - \frac{1}{n}\right)^{n} \leq \frac{1}{e}$, použijeme ho i v příštím výpočtu. Obě posloupnosti k $1/e$ monotónně konvergují.

Nechť $F_{i,t}$ je jev označující, že i-tý proces neuspěje v žádném z t cyklů. Máme

$$Pr[F_{i,t}] = \prod_{r=1}^{t} (1 - A_{i,r}) \leq \left(1 - \frac{1}{en}\right)^{t} = \left(\left(1 - \frac{1}{en}\right)^{en}\right)^{\frac{t}{en}} \leq e^{-\frac{t}{en}} \leq \left(e^{\ln n}\right)^{-2} = n^{-2} \,.$$

Pravděpodobnost, že existuje proces, který neuspěje, odhadneme na rozdíl od předchozích přesných počítání zdánlivě nepřesně; dá se však ukázat, že výsledek je asymptoticky správný.

$$Pr\left[\bigcup_{i=1}^{n} F_{i,t}\right] \leq \sum_{i=1}^{n} Pr[F_{i,t}] \leq n \cdot n^{-2} = n^{-1}.$$

□

4 Aproximační algoritmy – základní definice

Pokud uvažujeme o přibližných řešeních problému, musíme mít definováno, co je pro danou instanci dobré řešení. Jde tedy o podstatně jiné problémy než rozhodovací problémy v klasické složitosti, kde je odpověď binární. Potřebný typ problémů vymezuje následující definice.

Definice 4.1. **Optimalizační problém** je čtveřice $(\mathcal{I}, \mathcal{F}, f, g)$, kde $\mathcal{I}$ je množina vstupů neboli instancí; funkce $\mathcal{F}$ pro každou instanci I udává množinu přípustných řešení $\mathcal{F}(I)$, účelová funkce f pro každou instanci I a přípustné řešení dává hodnotu řešení a g je bit, který říká, zda účelovou funkci chceme maximalizovat nebo minimalizovat.

Takto obecná definice zahrnuje i spojité problémy matematického programování. Nás ale z algoritmického hlediska budou zajímat především kombinatorické problémy, kde instance i přípustná řešení jsou konečné řetězce (a tedy čísla jsou přirozená nebo racionální). Typické kombinatorické optimalizační problémy patří do třídy NP-optimalizačních problémů, pro které požadujeme, aby (i) délka všech přípustných řešení byla polynomiálně omezená v délce instance, (ii) množina všech dvojic instancí a přípustných řešení byla v P, a (iii) účelová funkce byla počitatelná v polynomiálním čase.

Řada takových problémů je NP-těžká, pak studujeme aproximační algoritmy, které jsou efektivní, ale nenajdou vždy optimální řešení. Namísto toho ale požadujeme, aby aproximační algoritmus pro *každou* instanci našel s hodnotou nepříliš dalekou od optima.

Pro danou instanci I označujeme optimální řešení i jeho hodnotu $OPT(I)$, řešení algoritmu A i jeho hodnotu označujeme $A(I)$. V případě pravděpodobnostního algoritmu je $A(I)$ náhodná proměnná, která závisí na náhodných bitech použitých v algoritmu; pak nás zajímají algoritmy, u kterých je průměrná hodnota $A(I)$ blízká optimu.

Definice 4.2. Algoritmus A se nazývá **R-aproximační**, jestliže v polynomiálním čase pro každou instanci I najde přípustné řešení a navíc:

Pro minimializační problém platí $(\forall I)\, A(I) \leq R \cdot OPT(I)$, resp. pro pravděpodobnostní algoritmus $(\forall I)\, E[A(I)] \leq R \cdot OPT(I)$.

Pro maximalizační problém platí $(\forall I)\, A(I) \geq OPT(I)/R$, resp. pro pravděpodobnostní algoritmus $(\forall I)\, E[A(I)] \geq OPT(I)/R$.

Pro některé maximalizační problémy se tradičně uvádí aproximační poměr menší než 1, tj. jako převrácená hodnota; pak tedy požadujeme $A(I) \geq R \cdot OPT(I)$.

5 Hladové algoritmy a lokální prohledávání

Přirozené heuristiky pro kombinatorické problémy se snaží najít řešení tak, aby v jednotlivých krocích vybraly lokálně nejlepší možnost. Hladové algoritmy typicky postupně doplňují řešení až jsou splněny všechny podmínky problému. Lokální prohledávání naopak začne od libovolného řešení a snaží se ho postupně zlepšovat malými úpravami.

5.1 Problém obchodního cestujícího

Problém obchodního cestujícího patří mezi nejstudovanější optimalizační problémy z mnoha teoretických i praktických pohledů. My si předvedeme dva aproximační algoritmy, které jsou v principu hladové, i když nepoužívají postupnou konstrukci řešení. Namísto toho využijí optimální řešení jiných grafových problémů, které lze řešit v polynomiálním čase.

> METRICKÝ PROBLÉM OBCHODNÍHO CESTUJÍCÍHO
> *Vstup:* Metrika d na n vrcholech V. (Tj. nezáporná funkce splňující symetrii $(\forall u, v)\, d(u, v) = d(v, u)$ a tranzitivitu $(\forall u, v, w)\, d(u, v) \leq d(u, w) + d(w, v)$.)
> *Výstup:* Cyklus C daný permutací n vrcholů $v_1, v_2, \ldots, v_n$, tj. obsahující hrany v_1v_2, v_2v_3, $\ldots$, $v_{n-1}v_n$, v_nv_1.
> *Cíl:* minimalizovat délku cyklu, t.j. $d(C) = d(v_n, v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$.

Je užitečné se na vstup dívat jako na úplný graf s váženými hranami. Cílem je pak najít podmnožinu hran, která tvoří Hamiltonovský cyklus. Definici délky můžeme přirozeně rozšířit na množinu orientovaných i neorientovaých hran E tak, že $d(E)$ je součet délek jednotlivých hran. V průběhu algoritmu budeme potřebovat pracovat s multigrafem, kde se hrany mohou opakovat; definice se rozšíří přirozeným způsobem.

Pro naše algoritmy budeme potřebovat následující podprogram

> PODPROGRAM Z (ZKRÁCENÍ)
> *Vstup:* Eulerovský (orientovaný nebo neorientovaný) multigraf (V, E).
> - Najdi (uzavřený) Eulerovský tah s hranami E. Nechť $v_1, v_2, \ldots, v_n$ je permutace vrcholů V v pořadí podle prvního výskytu v Eulerovském tahu (s jakýmkoliv začátkem).
>
> *Výstup:* cyklus C daný touto permutací, tj. obsahující hrany v_1v_2, v_2v_3, $\ldots$, $v_{n-1}v_n$, v_nv_1.

Výstup může obsahovat i hrany, které nejsou v E. Nicméně z trojúhelníkové nerovnosti plyne, že $d(Z(E)) \leq d(E)$.

> KOSTROVÝ ALGORITMUS
> *Vstup:* Metrika d na n vrcholech.
> - Najdi minimální kostru T v úplném grafu s vahami hran danými metrikou d.
> - E je množina $2n-2$ orientovaných hran vzniklých z T nahrazením každé hrany jejími oběma orientacemi.
>
> *Výstup:* $Z(E)$

Lemma 5.1. $d(T) \leq OPT$.

Důkaz. Pokud z cyklu OPT vynecháme jednu hranu, dostaneme minimální kostru a ta nemůže být menší než T. □

Věta 5.2. *Kostrový algoritmus je 2-aproximační pro metrický problém obchodního cestujícího.*

Důkaz. Z předchozích pozorování odhadneme hodnotu výstupu: $d(Z(E)) \leq d(E) = 2 \cdot d(T) \leq 2 \cdot OPT$. □

Christofidesův algoritmus
Vstup: Metrika d na n vrcholech.
- Najdi minimální kostru T v úplném grafu s vahami hran danými metrikou d
- W je množina vrcholů lichého stupně v T
- Najdi M, perfektní párování minimální váhy v grafu indukovaném W

Výstup: $Z(T \dot\cup M)$

Především musíme ověřit, že algoritmus je korektní a efektivní. Hledání minimální kostry, perfektního párování minimální váhy, eulerovského tahu i ostatní kroky lze implementovat v polynomiálním čase (i kdy pro párování to není jednoduché). Množina W má sudý počet vrcholů, perfektní párování tedy existuje. Protože párování je perfektní, $T \dot\cup M$ má všechny vrcholy sudého stupně; zde využíváme toho, že jde o disjunktní sjednocení, tedy pokud hrana je v T i M, tak ji v Eulerovském tahu použijeme dvakrát. Navíc je $(V, T \dot\cup M)$ souvislý (multi)graf, protože obsahuje kostru.

Věta 5.3. *Christofidesův algoritmus je 3/2-aproximační pro metrický problém obchodního cestujícího.*

Důkaz. Ukážeme, že $d(M) \leq OPT/2$. Cyklus OPT zkrátíme na cyklus na W tak, že ostatní vrcholy v pořadí vynecháme; cena se tím díky trojúhelníkové nerovnosti nezvýší. Tento cyklus má sudou délku a tedy tvoří dvě párování. Jedno z nich je dlouhé nejvýš $OPT/2$ a tedy i $d(M) \leq OPT/2$.

Z předchozích pozorování odhadneme hodnotu výstupu: $d(Z(E)) \leq d(E) = d(T)+d(M) \leq OPT + OPT/2$. □

Pro oba předchozí algoritmy lze najít příklady, které ukazují, že aproximační poměr není lepší, než jsme výše dokázali.

Pro metrický problém obchodního cestujícího dodnes není známý lepší než 3/2-aproximační algoritmus. Naopak pro eukleidovské prostory je možné se optimálnímu řešení libovolně přiblížit, pro každé $\varepsilon > 0$ existuje $(1+\varepsilon)$-aproximační algoritmus. Takovému systému algoritmů se říká **polynomiální aproximační schéma**.

5.2 Rozvrhování

V **rozvrhování** máme na vstupu n úloh, které je třeba rozvrhnout na jednom či více strojích. Existuje řada variant, které se liší v parametrech úloh a strojů, v požadavcích na rozvrh, či v účelové funkci.

Téměř vždy je jedním z parametrů úlohy j čas běhu, značený p_j. Většinou je třeba každou úlohu rozvrhnout na jeden z m strojů v jediném intervalu $[S_j, C_j)$ tak, že intervaly různých úloh na stejném stroji jsou po dvou disjunktní. Standardní konvence je, že rozvrh začíná v čase 0, tj. $S_j \geq 0$, a parametry úloh jsou celočíselné nebo alespoň racionální. Pro rozvrhování na identických strojích je $C_j = S_j + p_j$.

Jedna z nejčastějších účelových funkcí je **délka rozvrhu**, definovaná jako $\max_j C_j$ a standardně označovaná $C_{\max}$.

V nejjednodušších případech, jako je následující, stačí určit přiřazení na jednotlivé stroje.

ROZVRHOVÁNÍ NA IDENTICKÝCH STROJÍCH
Vstup: Počet strojů m, časy běhu úloh $p_1, \ldots, p_n \in \mathbb{R}^+$.
Výstup: Rozvrh, tj. rozklad množiny $\{1, \ldots, n\}$ na množiny $I_1, \ldots, I_m$.
Cíl: minimalizovat délku rozvrhu, t.j. $\max_{i \in \{1,\ldots,m\}} \sum_{j \in I_i} p_j$.

Časy zahájení a dokončení jednotlivých úloh lze dopočítat – nejsou ovšem určeny jednoznačně, rozvrhů se stejným přiřazením úloh na počítače je celá řada. Jeden z nich dostaneme tak, že položíme pro každý $j \in I - i$ zvolíme čas zahájení $S_j = \sum_{k \in I_i, k<j} p_j$ a a pochopitelně čas ukončení $C_j = s_j + p_j$. Definujme také **délku rozvrhu stroje** i jako $max_{j \in I_i} C_j$. Snadno ověříme, že intervaly na jednom stroji na sebe navazují, tedy jsou disjunktní, a délka rozvrhu stroje i je rovna $\sum_{j \in I_1} p_j$. Délka rozvrhu $C_{\max}$ je rovna největší délce rozvrhu stroje a je také rovna hodnotě z definice problému v rámečku, navíc splňuje $C_{\max} = \max_j C_j$.

Pro rozvrhování na identických strojích nás zajímá jak uvedená varianta, kdy m je součástí vstupu, a aproximační poměr nezávisí na m, tak i varianta, kdy m je pevná konstanta.

Lokální prohledávání

Lokální prohledávání je oblíbená heuristika s mnoha variantami. V některých případech je její výsledek dokazatelně dobrý. Takový algoritmus si předvedeme pro rozvrhování na identických strojích.

Přirozené pravidlo pro lokální změnu rozvrhu je přesunout jednu úlohu na jiný stroj tak, aby se délka rozvrhu zkrátila. To má dva problémy.

První problém je to, že pokud máme více strojů s maximální délkou rozvrhu stroje, přesunutím jedné úlohy nemůžeme délku rozvrhu snížit a tedy nemůžeme udělat žadný platný krok. Nicméně přesunutím jedné úlohy můžeme snížit počet strojů s maximální délkou rozvrhu a po nejvýše m takových krocích se délka rozvrhu sníží. Budeme tedy používat tyto obecnější kroky, kdy vždy přesuneme úlohu ze stroje s maximální délkou rozvrhu tak, aby nová délka rozvrhu stroje, kam je úloha přesunuta, byla menší. Pak už umíme dokázat, že ve chvíli, kdy není žádný krok možný, je rozvrh dobrou aproximací.

Druhý problém je, že neumíme odhadnout počet kroků do zastavení algoritmu. To bývá často největší problém návrhu algoritmů lokálního prohledávání. V našem případě se tomu vyhneme tak, že upřesníme pravidlo, kterou úlohu a na který stroj přesuneme.

LOKÁLNÍ PROHLEDÁVÁNÍ PRO ROZVRHOVÁNÍ NA IDENTICKÝCH STROJÍCH
Vstup: Počet strojů m, časy běhu úloh $p_1, \ldots, p_n \in \mathbb{R}^+$.
(1) Rozvrhni všechny úlohy libovolně (např. na stroj 1)
(2) Pro libovolný stroj i s maximální délkou rozvrhu stroje odeber poslední úlohu a rozvrhni ji jako poslední na (nějaký) stroj i s minimální délkou rozvrhu stroje, pokud tato operace zmenší čas dokončení dané úlohy.
(3) Pokud v kroku (2) nedojde k přesunu úlohy, vystup aktuální rozvrh. Jinak opakuj krok (2)

Věta 5.4. *Výše uvedený algoritmus lokálního prohledávání pro rozvrhování na m strojích je $2 - 1/m$-aproximační pro pevné m a 2-aproximační je-li m částí vstupu.*

Důkaz. Nejprve ukážeme, že algoritmus skončí v polynomiálním čase. Označme $C_{\min}$ minimální délku rozvrhu stroje. Všimněme si, že hodnota $C_{\min}$ se v kroku (2) nezmenší. Z toho

plyne, že každá úloha j bude přesunuta maximálně jednou: Po prvním přesunu je čas zahájení S_j roven předchozí hodnotě $C_{\min}$, v druhém přesunu by se S_j změnilo na aktuální hodnotu $C_{\min}$, která je alespoň tak velká, a úloha by neskončila dříve. Krok (2) se tedy opakuje nejvýše n-krát a lze ho implementovat efektivně, stejně jako nalezení počátečního rozvrhu.

Uvažujme nyní výsledný rozvrh. Algoritmus skončí tak, že v kroku (2) nepřesune úlohu; označme tuto úlohu j. Z podmínky na přesunutí plyne, že $C_{\max} - C_{\min} \leq p_j \leq OPT$, kde poslední nerovnost plyne z toho, že i optimum rozvrhne úlohu j.

Z definice $C_{\min}$ víme, že do času $C_{\min}$ jsou všechny stroje zaplněné a tedy $C_{\min} \leq OPT$, protože optimální rozvrh musí také rozvrhnout všechnu práci. Z toho okamžitě plyne $C_{\max} = C_{\min} + (C_{\max} - C_{\min}) \leq 2 \cdot OPT$. Ve skutečnosti můžeme pro pevné m odhad o trochu zlepšit, protože práce vykonaná optimem zahrnuje i práci vykonanou na úlohze p_j po čase $C_{\min}$ v cekovém objemu $C_{\max} - C_{\min}$. Dostáváme

$$C_{\min} \leq \frac{\sum_{k=1}^{n} p_k}{m} - \frac{C_{\max} - C_{\min}}{m} \leq \left(1 - \frac{1}{m}\right) OPT\,,$$

a tedy

$$C_{\max} = C_{\min} + (C_{\max} - C_{\min}) \leq \left(1 - \frac{1}{m}\right) OPT + OPT = \left(2 - \frac{1}{m}\right) OPT\,.$$

□

Hladové algoritmy

Hladový algoritmus probírá úlohy postupně, každou přiřadí na stroj s nejmenší délkou rozvrhu pro předchozí úlohy. (Je-li takových strojů více, volíme libovolně.) Hladový algoritmus pro rozvrhování se také nazývá *List Scheduling*. Důkaz aproximačního poměru je obdobný jako u předchozího algoritmu.

Věta 5.5. *Hladový algoritmus pro rozvrhování na m strojích je $(2 - 1/m)$-aproximační pro pevné m a 2-aproximační je-li m částí vstupu.*

Důkaz. Algoritmus zjevně pracuje v polynomiálním čase.

Bez újmy na obecnosti můžeme předpokládat, že poslední dokončená úloha je úloha n, tj. poslední rozvržená. Jinak můžeme vynechat všechny úlohy za poslední dokončenou, tím se náš rozvrh nezkrátí a OPT neprodlouží.

Pak $C_n - S_n = p_n \leq OPT$, protože i optimum rozvrhne úlohu n. Z definice hladového algoritmu víme, že do času S_n jsou všechny stroje zaplněné a tedy $S_n \leq OPT$, protože optimální rozvrh musí také rozvrhnout všechnu práci. Ve skutečnosti můžeme odhad o trochu zlepšit, protože jsme do odhadu na S_n zbytečně započítali p_n, a obdobně jako u lokálního prohledávání dostáváme

$$S_n \leq \frac{\sum_{j=1}^{n} p_j}{m} - \frac{p_n}{m} \leq OPT - \frac{p_n}{m}\,,$$

a tedy

$$C_n \leq OPT + \left(1 - \frac{1}{m}\right) p_n \leq \left(2 - \frac{1}{m}\right) OPT\,.$$

□

Je zřejmé, že problematický vstup pro hladový algoritmus je takový, kdy je poslední úloha velmi dlouhá, srovnatelná se zbytkem rozvrhu. Snadno nalezneme příklad, který ukazuje, že předchozí analýza je těsná.

Pokud úlohy nejprve seřadíme sestupně podle velikosti a poté rozvrhneme hladově, získáme podstatně lepší algoritmus. Nazývá se LPT algoritmus (zkratka z "Largest Processing Time first").

Věta 5.6. LPT *algoritmus pro rozvrhování na m strojích je $(4/3-1/(3m))$-aproximační pro pevné m a $4/3$-aproximační je-li m částí vstupu.*

Důkaz. Bez újmy na obecnosti můžeme předpokládat, že $p_1 \geq p_2 \geq \cdots \geq p_n$ a poslední dokončená úloha je úloha n, tj. poslední rozvržená. Stejně jako u hladového algoritmu jinak můžeme vynechat všechny úlohy za poslední dokončenou, tím se náš rozvrh nezkrátí a OPT neprodlouží.

Rozlišíme dva případy. Pokud je $p_n \leq OPT/3$, tak počítáme analogicky jako u hladového algoritmu: $S_n \leq \sum_{j=1}^{n} p_j/m - p_n/m \leq OPT - p_n/m$, a tedy

$$C_n \leq OPT + \left(1 - \frac{1}{m}\right) p_n \leq \left(\frac{4}{3} - \frac{1}{3m}\right) OPT\,.$$

Pokud $p_n > OPT/3$, tak optimum rozvrhne nanejvýš dvě úlohy na každý stroj. Ukážeme, že LPT vytvoří optimální rozvrh. Uvážíme následující dolní odhady na OPT. Je-li $n \geq m+1$, platí $p_m + p_{m+1} \leq OPT$, protože optimální řešení pro prvních $m+1$ úloh má nutně dvě úlohy na stejném stroji a ty jsou alespoň tak velké jako nejmenší úlohy p_m a p_{m+1}. Je-li $n \geq m+2$, platí $p_{m-1} + p_{m+2} \leq OPT$, protože z prvních $m+2$ úloh má optimum dvě dvojice na stejném stroji, tedy alespoň jedna úloha velikosti alespoň p_{m-1} je ve dvojici. (Použili jsme také podmínku $p_n > OPT/3$, ze které plyne, že optimum nemůže mít na stejném stroji trojici úloh.) Obdobně platí $p_{m-2} + p_{m+3} \leq OPT$ atd. Nyní si všimneme, že LPT vytvoří na jednotlivých strojích právě tyto dvojice. (Použili jsme opět podmínku $p_n > OPT/3$, ze které nyní plyne, že ani LPT nerozvrhne na stejném stroji trojici úloh.) □

Opět je možné zkonstruovat příklad, který ukazuje, že předchozí analýza je těsná.

Přesné řešení rozvrhování na identických počítačích je NP-těžké, existují redukce z problémů PARTITION a 3-PARTITION (nebo z jiné varianty problému batohu). Na druhou stranu, existují i lepší aproximační algoritmy, dokonce polynomiální aproximační schémata.

Online algoritmy

Hladový algoritmus pro rozvrhování patří mezi online algoritmy. Může dostávat vstup po jednotlivých úlohách a hned doplnit rozvrh nové úlohy tak, že máme platný rozvrh pro celou dosavadní instanci. Stejně tak hladové algoritmy pro další problémy jsou často online algoritmy ve vhodném modelu.

Pro online algoritmy mluvíme o **kompetitivním poměru** namísto aproximačního poměru. Záruka kvality řešení je pak stejná jako u aproximačních algoritmů, nepožaduje se však polynomiální čas běhu algoritmu.

Pro rozvrhování na identických strojích je snadné nahlédnout, že neexistuje lepší než $3/2$-kompetitivní algoritmus. Na vstupu jsou nejprve dvě úlohy velikosti 1, které algoritmus musí

rozvrhnout na různé stroje. Poté vstup pokračuje $m-1$ úlohami velikosti 2; algoritmus bude mít délku rozvrhu alespoň 3, ale optimum je 2.

Speciálně pro $m = 2$ z předchozího dolního odhadu plyne, že hladový algoritmus je optimální online algoritmus; stejný výsledek lze dokázat i pro $m = 3$. Naopak pro větší m lepší algoritmy existují. Pro velké m existuje 1.923-kompetitivní algoritmus a nejlepší dolní odhad říká, že neexistuje 1.88-kompetitivní deterministický algoritmus.

5.3 Problém bin packing

Problém bin packing je v jistém smyslu duální k rozvrhování na identických počítačích. Máme danou velikost košů, obvykle normalizovanou na 1, a chceme dané prvky naskládat do co nejmín košů. To odpovídá minimalizaci počtu strojů, je-li délka rozvrhu pevně omezená.

> BIN PACKING
> *Vstup:* prvky $a_1, \ldots, a_n \in \mathbb{R}^+$.
> *Výstup:* rozklad množiny $\{1, \ldots, n\}$ na koše $I_1, \ldots, I_m$, každý s celkovou velikostí nejvýše 1, tj. $(\forall i) \sum_{j \in I_i} a_i \leq 1$.
> *Cíl:* minimalizovat počet košů m.

Algoritmus FIRSTFIT probírá prvky popořadě, každý dá do prvního koše, kam se vejde. Nový koš vytvoří, když se prvek nevejde nikam. Algoritmus BESTFIT postupuje obdobně, ale prvek dá do nejvíce plného koše, kam se vejde. Algoritmus je ANYFIT algoritmus, pokud postupuje obdobně s libovolným výběrem koše pro nový prvek; jediné omezení je, že nový koš se tvoří, když se prvek nevejde do žádného stávajícího. FIRSTFIT a BESTFIT jsou příklady ANYFIT algoritmu.

Věta 5.7. *Každý* ANYFIT *algoritmus je 2-aproximační.*

Důkaz. Podle ANYFIT pravidla platí, že každé dva koše mají dohromady velikost větší než 1. Z toho pro $m \geq 2$ sečtením nerovnosti $a_1 + a_m > 1$ a všech $m-1$ nerovností $a_i + a_{i+1} > 1$ plyne, že celková velikost košů je alespoň $m/2$. Zjevně platí, že $\sum_{i=1}^{n} a_i \leq OPT$. Celkově tedy $m/2 \leq \sum_{i=1}^{n} a_i \leq OPT$. □

Pro algoritmy FIRSTFIT a BESTFIT lze ukázat, že jsou 1.7-aproximační (a ne lepší).

Je NP-těžké rozhodnout, zda $OPT \leq 2$ (z problému PARTITION), a tedy nemůže existovat lepší než 3/2-aproximační algoritmus: takový algoritmus pro všechny instance s $OPT \leq 2$ musí najít optimální řešení a proto by nám umožnil v polynomiálním čase odlišit instance s $OPT \leq 2$ od těch s $OPT \geq 3$.

Naopak existují **asymptotická aproximační schémata**, přesněji algoritmy, které dosáhnou $A(I) \leq 1 + (1+\varepsilon)OPT(I)$.

5.4 Hledání disjunktních cest v grafu

Pokud v grafu hledáme více disjunktních cest z vrcholu s do vrcholu t, jde o problém ekvivalentní hledání maximálního toku, který je dobře řešitelný. Jakmile se ale ptáme, zda existují disjunktní cesty mezi dvojicemi (s_i, t_i), ale nedovolujeme spojit (s_i, t_j) pro $i \neq j$, jde o NP-těžký problém, pro který v této kapitole navrhneme aproximační algoritmus.

Problém disjunktních cest lze formulovat v řadě variant. Nás budou zajímat hranově disjunktní cesty, problém pro vrcholově disjunktní cesty je ovšem velmi podobný. Algoritmy,

které uvedeme, pracují stejně dobře pro orientované i neorientované grafy, ovšem pro neorientované grafy je v některých případech možné výraznější zlepšení.

Kromě základní varianty disjunktních cest nás bude zajímat i varianta, kde každá hrana má kapacitu c pro nějakou konstantu c, tj. každou hranu může použít c cest. Uvidíme, že tato varianta má podstatně lepší algoritmus, který používá zajímavou techniku.

HRANOVĚ DISJUNKTNÍ CESTY V GRAFU S KAPACITOU HRAN
Vstup: Parametr c, neorientovaný nebo orientovaný graf $G = (V, E)$, posloupnost dvojic vrcholů $(s_1, t_1), \ldots, (s_k, t_k)$.
Výstup: cesty P_i pro $I \subseteq \{1, \ldots, k\}$ takové, že cesta P_i spojuje s_i a t_i a každá hrana je použita nejvýše v c cestách.
Cíl: maximalizovat počet cest $|I|$.

Nejprve analyzujeme jednoduchý hladový algoritmus pro verzi s kapacitou $c = 1$, který vždy použije nejkratší cestu ze všech možností.

HLADOVÝ ALGORITMUS PRO HLEDÁNÍ DISJUNKTNÍCH CEST
Vstup: $G = (V, E)$, dvojice vrcholů $(s_1, t_1), \ldots, (s_k, t_k)$
(1) $I := \emptyset$; $m := |E|$;
(2) Najdi nejkratší cestu P z s_i do t_i, přes všechna $i \notin I$. Pokud P neexistuje, jdi na (4).
(3) $I := I \cup \{i\}$; $P_i := P$; $E := E \setminus P$. Jdi na (2).
(4) *Výstup:* cesty P_i, $i \in I$.

Věta 5.8. *Hladový algoritmus pro hledání disjunktních cest (s kapacitou hran 1) je $O(\sqrt{m})$-aproximační.*

Důkaz. Uvažujme instanci, kde $OPT \geq 1$, pak také $|I| \geq 1$. Z počtu hran plyne, že v OPT je nejvýše $\sqrt{m}$ cest delších než $\sqrt{m}$. Uvažme cestu P^* v OPT délky nejvýše $\sqrt{m}$. Pokud P^* není vybrána algoritmem, musí mít společnou hranu s některou cestou P_i délky nejvýše $\sqrt{m}$. Jedna taková cesta P_i však může zablokovat každou hranou nejvýš jednu cestu z P, celkově nejvýše $\sqrt{m}$ cest. Je tedy $OPT \leq \sqrt{m} + \sqrt{m}|I| \leq 2\sqrt{m}|I|$. □

Příklad s poměrem $\sqrt{m}$ lze snadno zkonstruovat: Graf bude strom s jednou cestou $s_1, s_2, \ldots, s_k, t_1$ a z vrcholu s_{i+1}, $i = 2, \ldots, k$ odbočí cesta délky k do vrcholu t_i. Algoritmus použije první cestu, čímž rozpojí všechny ostatní dvojice, optimum naopak spojí všechny ostatní dvojice. Aproximační poměr je $k - 1$, počet vrcholů i hran je $\Theta(k^2)$.

Pro obecnou kapacitu c, kde c je libovolná konstanta, potřebujeme použít trochu složitější postup. Myšlenka je taková, že hrany, jejichž kapacita je částečně vyčerpána, považujeme při hledání nejkratší cesty za dostatečně dlouhé. V analýze se ukáže, že správné zvyšování délek hran je exponenciální v počtu jejich použití, tj. při použití hrany její délku vynásobíme vhodným faktorem. Podobný postup se používá i v řadě dalších algoritmů pro směrování v sítích i jiné podobné problémy.

HLADOVÝ ALGORITMUS PRO HLEDÁNÍ CEST V GRAFU S KAPACITOU c
Vstup: $G = (V, E)$, dvojice vrcholů $(s_1, t_1), \ldots, (s_k, t_k)$
(1) $\beta := m^{1/(c+1)}$; $I := \emptyset$; $m := |E|$; $d(e) := 1$ pro všechny hrany $e \in E$.
(2) Najdi nejkratší cestu P z s_i do t_i vzhledem k délkám d, přes všechna $i \notin I$. Pokud P neexistuje nebo spolu s cestami P_i, $i \in I$, porušuje podmínku kapacity c, jdi na (4).
(3) $I := I \cup \{i\}$; $P_i := P$; pro všechny hrany $e \in P$, $d(e) := \beta \cdot d(e)$. Jdi na (2).
(4) *Výstup:* cesty P_i, $i \in I$.

Věta 5.9. *Hladový algoritmus pro hledání cest v grafu s kapacitou c je $O(m^{1/(c+1)})$-aproximační.*

Důkaz. Pokud $OPT < c$, tak algoritmus najde optimum. Uvažujme instanci, kde $OPT \geq c$, pak také $|I| \geq c$.

Řekneme, že cesta je krátká, pokud její délka je menší než β^c. Dokud algoritmus vybírá krátké cesty, neporuší podmínku kapacity c, protože každá hrana má délku menší než β^c a tedy je použita méně než c cestami.

Odteď budeme pracovat s délkami $\bar{d}$ takovými jako v první iteraci algoritmu, kdy nelze vybrat krátkou cestu.

Pokud OPT spojí (s_i, t_i) cestou P_i^* a algoritmus je nespojí, pak $\bar{d}(P_i^*) \geq \beta^c$, a tedy $\bar{d}(E) \geq \beta^c \cdot (OPT - |I|)/c$.

Nyní omezíme $\bar{d}(E)$ pomocí $|I|$. Na počátku algoritmu je $\bar{d}(E) = m$. V iteraci, kdy přidáme krátkou cestu, zvýšíme její délku nejvýše na $\beta \cdot |P_i| \leq \beta \cdot \beta^c = m$. Je tedy $\bar{d}(E) \leq (|I|+1) \cdot m$. Celkově tedy máme $|I|+1 \geq \bar{d}(E)/m \geq (OPT - |I|)/(\beta c)$. Dostáváme $(\beta c + 1)(|I|+1) \geq OPT$ a z $|I| \geq c$ a toho, že c je konstanta, plyne aproximační poměr $O(\beta)$. □

Pro problémy hledání cest existují i lepší (a složitější) algoritmy, studuje se také jak řada speciálních případů tak i závislost na počtu vrcholů namísto počtu hran. Pro variantu orientovaných grafů bez kapacit je ale možné dokázat, že poměr $\Theta(\sqrt{m})$ je nejlepší možný v závislosti na počtu hran.

6 Algoritmy založené na lineárním programování

6.1 Splnitelnost

MAXSAT
Vstup: Posloupnost klauzulí $C_1, \ldots, C_m$, každá klauzule C_i je disjunkce k_i literálů, každý literál je proměnná $x_1, \ldots, x_n$ nebo její negace $\neg x_1, \ldots, \neg x_n$.
Výstup: Ohodnocení $(a_1, \ldots a_n) \in \{0, 1\}^n$.
Cíl: maximalizovat počet splněných klauzulí.

Pro pořádek můžeme předpokládat, že každá klauzule je neprázdná, a že žádná klauzule neobsahuje některou proměnnou i její negaci jako literál. (Prázdná klauzule je vždy nesplněná, klauzule obsahující $x_i \vee \neg x_i$ vždy splněná. Jejich vynechání nezhorší aproximační poměr.)

Varianty MAXSAT získáme omezením klauzulí na vstupu. Jestliže $k_i \leq k$ pro všechna i, mluvíme o MAX-kSAT, jestliže dokonce $k_i = k$ pro všechna i, pak mluvíme o MAX-EkSAT. Také MAX-2SAT je NP-těžký problém, dokonce je NP-těžké ho aproximovat s konstantním aproximačním poměrem blízkým 1.

RAND-SAT
Vstup: klauzule $C_1, \ldots, C_m$.
Vyber ohodnocení $(a_1, \ldots a_n) \in \{0,1\}^n$ uniformně náhodně.
Výstup: $(a_1, \ldots a_n)$.

Lemma 6.1. *Pro každou klauzuli C s k literály je $Pr[C(\vec{a}) = 1] = 1 - 2^{-k}$.*

BIASED-SAT
Vstup: klauzule $C_1, \ldots, C_m$.
Vyber $a_i \in \{0,1\}$ nezávisle náhodně takto:
Jestliže $\{x_i\}$ se vyskytuje jako jednoprvková klauzule častěji než $\{\neg x_i\}$, pak zvol $a_i = 1$ s pravděpodobností $\phi - 1 = (\sqrt{5}-1)/2 \approx 0.618$ a $a_i = 0$ jinak.
Pro zbývající i zvol $a_i = 0$ s pravděpodobností $\phi - 1 = (\sqrt{5}-1)/2$ a $a_i = 1$ jinak.
Výstup: $(a_1, \ldots a_n)$.

Věta 6.2. *Algoritmus* BIASED-SAT *je $(\phi - 1)$-aproximační algoritmus pro* MAXSAT, *kde* $\phi - 1 = (\sqrt{5}-1)/2 \approx 0.618$.

Důkaz. Z dvojice klauzulí $\{x_i\}$ a $\{\neg x_i\}$ je právě jedna splněná, můžeme tedy každou takovou dvojici vynechat bez zhoršení aproximačního poměru.

Zbývající klauzule délky 1 jsou splněny s pravděpodobností přesně $\phi - 1$. Klauzule délky $k \geq 2$ jsou splněny s pravděpodobností alespoň $1 - (\phi-1)^k \geq 1 - (\phi-1)^2 = \phi - 1$. □

Nechť

$$f(C_j) = \sum_{i: x_i \in C_j} y_i + \sum_{i: \neg x_i \in C_j} (1 - y_i).$$

Potom splňující ohodnocení přesně odpovídají celočíselným řešením následujícího lineárního programu:

$$\begin{aligned} &\text{maximalizuj} && \sum_{j=1}^{m} z_e \\ &\text{pro} && y_1, \ldots, y_n \in [0,1], \quad z_1, \ldots, z_m \in [0,1] \\ &\text{za podmínek} && f(C_t) \geq z_t \qquad \text{pro } i = 1, \ldots, m \end{aligned} \tag{6.1}$$

LP-SAT
Vstup: klauzule $C_1, \ldots, C_m$.
Nechť $y_1^*, \ldots, y_n^*, z_1^*, \ldots, z_m^*$ je optimum lineárního programu (6.1).
Vyber $a_i \in \{0,1\}$ nezávisle náhodně tak, že $a_i = 1$ s pravděpodobností y_i^* a $a_i = 0$ jinak.
Výstup: $(a_1, \ldots a_n)$.

Lemma 6.3. *Pro každou klauzuli C_j s k_j literály je $Pr[C(\vec{a}) = 1] \geq (1 - (1 - 1/k_j)^{k_j}) z_j^*$.*

Důkaz. Ze symetrie volby pravděpodobnosti plyne, že stačí důkaz provést pro $C_j = x_1 \vee x_2 \vee \cdots \vee x_{k_j}$. Pak je $y_1^* + \ldots + y_{k_j}^* \geq z_j^*$ a za pomoci nerovnosti mezi aritmetickým a geometrickým průměrem dostáváme

$$Pr[C_j(\vec{a}) = 0] = \prod_{i=1}^{k_j} (1 - y_i^*) \leq \left(\frac{1}{k_j} \sum_{i=1}^{k_j} (1 - y_i^*) \right)^{k_j} = \left(1 - \frac{1}{k_j} \sum_{i=1}^{k_j} y_i^* \right)^{k_j} \leq \left(1 - \frac{z_j^*}{k_j} \right)^{k_j}.$$

Nyní využijeme konkávnost funkce $f_k(t) = 1-(1-t/k)^k$ a toho, že $f_k(0) = 0$ a $f_k(1) = 1-(1-1/k)^k$. Pro $z_j^* \in [0,1]$ dostáváme

$$Pr[C_j(\vec{a}) = 1] \geq 1 - \left(1 - \frac{z_j^*}{k_j}\right)^{k_j} \geq \left(1 - \left(1 - \frac{1}{k_j}\right)^{k_j}\right) z_j^*.$$

□

Věta 6.4. *Algoritmus* LP-SAT *je* $1-1/e \approx 0.63$*-aproximační algoritmus pro* MaxSAT.

Best-SAT *Vstup:* klauzule $C_1, \ldots, C_m$. S pravděpodobností 1/2 použij algoritmus Rand-SAT a jinak algoritmus LP-SAT.

Věta 6.5. *Algoritmus* Best-SAT *je* 3/4*-aproximační algoritmus pro* MaxSAT.

Důkaz.

k	Rand-SAT	LP-SAT	Best-SAT
1	0.5	1	0.75
2	0.75	0.75	0.75
≥ 3	≥ 0.875	$\geq 1 - 1/e \approx 0.632$	> 0.75

□

6.2 Odstranění náhodnosti metodou podmíněných pravděpodobností

6.3 Množinové pokrytí

6.3.1 Formulace problému a souvisejících lineárních programů

Set Cover *Vstup:* Systém množin $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$, jejich ceny $c_1, \ldots, c_m \geq 0$. *Výstup:* Podsystém $S_i, i \in I$, $I \subseteq \{1, \ldots, m\}$ takový, že $\bigcup_{i \in I} S_i = \{1, \ldots, n\}$. *Cíl:* minimalizovat cenu I, tj. $\sum_{i \in I} c_i$.

Důležité parametry problému: $g = \max_j |S_j|$, $f = \max_e |\{j \mid e \in S_j\}|$.

Pro analýzu i formulaci algoritmů pro množinové pokrytí využijeme lineární programování a dualitu. Primární lineární program je:

$$\begin{aligned} &\text{minimalizuj} && \sum_{i=1}^{m} c_i x_i && \\ &\text{pro} && x_1, \ldots, x_m \geq 0 && \\ &\text{za podmínek} && \sum_{j: e \in S_j} x_j \geq 1 && \text{pro } e = 1, \ldots, n \end{aligned} \tag{6.2}$$

a duální lineární program je:

$$\begin{aligned} &\text{maximalizuj} && \sum_{e=1}^{n} y_e && \\ &\text{pro} && y_1, \ldots, y_n \geq 0 && \\ &\text{za podmínek} && \sum_{e: e \in S_j} y_e \leq c_j && \text{pro } j = 1, \ldots, m \end{aligned} \tag{6.3}$$

Intuitivní význam duality...
Slabá věta o dualitě...
Podmínky komplementarity...

6.3.2 Hladový algoritmus

GREEDY-SC
Vstup: Systém množin $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$, jejich ceny $c_1, \ldots, c_m \geq 0$.
(1) $I := \emptyset$, $E := \emptyset$.
(2) Dokud nejsou pokryté všechny prvky, tj. $E \subsetneq \{1, \ldots, n\}$ opakuj:
Pro $j \in \{1, \ldots, m\}$ s $S_j \not\subseteq E$ polož $p_j := c_j / |S_j \setminus E|$.
Nechť j_0 je takové, že p_{j_0} je definované a minimální.
Polož $q_e := p_{j_0}$ pro všechna $e \in |S_{j_0} \setminus E|$; polož $I := I \cup \{j_0\}$ a $E := E \cup S_{j_0}$.
Výstup: I.

Ukážeme, že hladový algoritmus má aproximační poměr $H_g \leq H_n$. Nejdříve příklad, že nemůže být lepší. Systém má n jednoprvkových množin $S_j = \{j\}$ s cenou $c_j = 1/j$ a jednu n-prvkovou $S_{n+1} = \{1, \ldots, n\}$ s cenou $c_{n+1} = 1 + \varepsilon$ pro malé $\varepsilon > 0$. GREEDY-SC postupně vybere množiny $S_n, \ldots, S_1$ s celkovou cenou H_n, zatímco optimum vybere jedinou množinu S_{n+1}.

Věta 6.6. *Algoritmus* GREEDY-SC *je H_g-aproximační algoritmus pro množinové pokrytí.*

Důkaz. Algoritmus běží v polynomiálním čase.

Uvážíme vektor $\mathbf{q}$ a ukážeme, že je nejvýše H_g-krát větší než přípustné řešení duálního programu. Přesněji, ukážeme, že $\mathbf{q}' = \frac{1}{H_g}\mathbf{q}$ je přípustné řešení duálního programu. Evidentně jsou q_e nezáporná.

Dále potřebujeme pro každé $j = 1, \ldots, m$ ukázat platnost podmínky

$$\sum_{e : e \in S_j} q'_e \leq c_j \,. \tag{6.4}$$

Označme prvky $S_j = \{e_1, \ldots, e_k\}$ tak, že algoritmus pokryje nejprve prvek e_k, pak e_{k-1} atd., až nakonec e_1. (Prvky pokryté ve stejné iteraci uspořádáme libovolně.) Z definice g platí $k \leq g$.

Pro $e_i \in S_j$ platí, že jsme ho v iteraci, kdy bylo pokryto, mohli pokrýt také množinou S_j s cenou $p_j \leq c_j/i$, protože ještě alespoň i prvků $e_1, \ldots, e_i$ z S_j není pokryto. Je tedy $q_{e_i} \leq c_j/i$. Nyní odhadneme levou stranu (6.4) a podmínku dokážeme:

$$\sum_{e : e \in S_j} q'_e = \frac{1}{H_g} \sum_{i=1}^{k} q_{e_i} \leq \frac{1}{H_g} \sum_{i=1}^{k} \frac{c_j}{i} = \frac{1}{H_g} \cdot c_j \cdot H_k \leq c_j \,.$$

Dokázali jsme, že $\mathbf{q}'$ je přípustné řešení duálního programu.

Z konstrukce algoritmu vidíme, že $\sum_{j \in I} c_j = \sum_{e=1}^{n} q_e$. Navíc podle duální účelová funkce přípustného řešení $\mathbf{q}'$ je dolním odhadem optima lineárního programu a tedy i optimálního celočíselného řešení. Dostáváme $\sum_{e=1}^{n} q_e = H_g \sum_{e=1}^{n} q'_e \leq H_g \cdot OPT$, a tedy výsledné řešení je H_g-aproximace. □

6.3.3 Zaokrouhlování LP a primárně-duální algoritmus

LP-SC
Vstup: Systém množin $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$, jejich ceny $c_1, \ldots, c_m \geq 0$.
Nechť $x_1^*, \ldots, x_m^*$ je optimum lineárního programu (6.2).
Výstup: $I = \{j \mid x_j^* \geq 1/f\}$.

Primárně-duální SC
Vstup: Systém množin $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$, jejich ceny $c_1, \ldots, c_m \geq 0$.
(1) $y_1, \ldots, y_n := 0$. $I := \emptyset$. $E := \emptyset$.
(2) Dokud existuje $e \notin E$, pro nějaké takové e proveď:
$\delta := \min_{j:e \in S_j} \left(c_j - \sum_{e \in S_j} y_e\right)$.
$y_e := y_e + \delta$.
Pro všechna j taková, že $e \in S_j$ a $\sum_{e \in S_j} y_e = c_j$ polož $I := I \cup \{j\}$ a $E := E \cup S_j$.
Výstup: I.

6.4 Vrcholové pokrytí

Vertex Cover
Vstup: Graf $G = (V, N)$, ceny vrcholů $c_v \geq 0$, $v \in V$.
Výstup: Podmnožina vrcholů $W \subseteq V$ taková, že $(\forall e \in E) e \cap W \neq \emptyset$.
Cíl: minimalizovat cenu W, tj. $\sum_{v \in W} c_v$.

Vrcholové pokrytí je speciálním případem množinového pokrytí, kde $f = 2$ a $g \leq n$ je maximální stupeň grafu.

Kombinatorický aproximační algoritmus pro neváženou verzi...

Vztah k nezávislé množině, rozdílnost aproximačního poměru...

Převod na množinové pokrytí...

LP a algoritmus pro váženou verzi...

$$\begin{aligned} \text{minimalizuj} \quad & \sum_{v \in V} c_v x_v \\ \text{pro} \quad & x_v \geq 0 \\ \text{za podmínek} \quad & x_u + x_v \geq 1 \qquad \text{pro } \{u, v\} \in E \end{aligned} \tag{6.5}$$

7 Hašování a implementace slovníku

7.1 Hašovací funkce

Definice 7.1. Nechť U a V jsou množiny, $m = |U|$ a $n = |V|$ jejich velikosti a H systém funkcí zobrazujících U do V.

Systém H se nazývá 2-*univerzální systém hašovacích funkcí* jestliže pro každé $x_1, x_2 \in U$, $x_1 \neq x_2$ a pro h uniformně náhodně vybrané z H platí

$$Pr_{h \in H}[h(x_1) = h(x_2)] \leq \frac{1}{n}\,.$$

Systém H se nazývá *silně 2-univerzální systém hašovacích funkcí* jestliže pro každé $x_1, x_2 \in U$, $x_1 \neq x_2$, $y_1, y_2 \in V$ a pro h uniformně náhodně vybrané z H platí

$$Pr_{h \in H}[h(x_1) = y_1 \wedge h(x_2) = y_2] = \frac{1}{n^2}\,.$$

Silně 2-univerzální systém hašovacích funkcí se také někdy nazývají po dvou nezávislé hašovací funkce. To odpovídá tomu, že náhodné proměnné $h(x)$ indexované $x \in U$, kde h je uniformně náhodně vybraná funkce z H, jsou po dvou nezávislé. (Tedy po dvou nezávislé jsou hodnoty h v různých bodech, ne různé funkce h.)

Pozorování 7.2. *Pro každý systém H existují $x_1, x_2 \in U$, $x_1 \neq x_2$ takové, že*

$$Pr_{h \in H}[h(x_1) = h(x_2)] \geq \frac{1}{n} - \frac{1}{m}\,.$$

Lemma 7.3.

7.2 Dynamický slovník

7.3 Statický slovník

8 Paralelní algoritmy pro maximální nezávislou množinu v grafu

Nechť d_v značí aktuální stupeň vrcholu v.

PARA-MIS
Vstup: Graf $G = (V, E)$.
$I := \emptyset$. Nechť d_v značí stupeň vrcholu v. Dokud $V \neq \emptyset$, opakuj následující kroky.
(1) Paralelně pro každý vrchol $v \in V$: Jestliže $d_v = 0$, pak $I := I \cup \{v\}$ a $V := V \setminus \{v\}$.
(2) Paralelně pro každý vrchol $v \in V$: Označ v s pravděpodobností $1/(2d_v)$. Pravděpodobnosti pro různé vrcholy a iterace jsou nezávislé.
(3) Paralelně pro každou hranu $\{u, v\} \in E$: Jestliže oba u a v jsou označené, odeber značku z vrcholu nižšího stupně z nich (libovolnou značku pro $d_u = d_v$).
(4) Nechť S je množina označených vrcholů a $N(S)$ množina jejich sousedů. $I := I \cup S$ a $V := V \setminus (S \cup N(S))$; z E odeber všechny hrany incidentní s vrcholem v $S \cup N(S)$).

Definice 8.1. Nechť je dán graf $G = (V, E)$. Vrchol v se nazývá *dobrý*, jestliže má alespoň $d_v/3$ sousedů stupně nejvýše d_v. Ostatní vrcholy jsou *špatné*. Hrana je dobrá, obsahuje-li dobrý vrchol. Hrana je špatná, jestliže má oba vrcholy špatné.

Lemma 8.2. *Existuje konstanta $\alpha > 0$ taková, že Pro každý dobrý vrchol platí, že v jedné iteraci algoritmu je odstraněn s pravděpodobností alespoň α, kde $\alpha > 0$ je vhodná konstanta.*

Lemma 8.3. *V každém grafu je alespoň polovina hran dobrá.*

Důkaz. Zorientujme všechny hrany směrem ke koncovému vrcholu většího stupně. (Mají-li oba vrcholy stejný stupeň, vyberme libovolně.) Každá špatná hrana vstupuje do špatného vrcholu (podle definice). Každý špatný vrchol má alespoň dvakrát více vystupujících hran než

vstupujících hran (opět z definice). Přiřaďme tedy každé špatné hraně e dvě hrany $f_1(e)$ a $f_2(e)$ vystupující z vrcholu, kam hrana e vstupuje, a to tak, že žádná vycházející hrana není přiřazena dvěma špatným vcházejícím hranám. Každá hrana vychází z jediného vrcholu, tedy i globálně se každá hrana e' vyskytuje nejvýše jednou jako hodnota $f_1(e)$ nebo $f_2(e)$. (Tj. funkce f_1 a f_2 jsou prosté a mají disjunktní obory hodnot.) Tím jsme dokázali, že dvojnásobek počtu špatných hran je nejvýše počet všech hran. Alespoň polovina hran je tedy dobrých. □

Věta 8.4. *Pravděpodobnostní algoritmus* Para-MIS *najde maximální nezávislou množinu v grafu o n vrcholech v průměrném čase $(\log n)^{O(1)}$ na polynomiálně mnoha procesorech.*

8.1 Derandomizace pomocí po dvou nezávislých proměnných

Důkaz Lemmatu 8.2. . □

9 Pravděpodobnostní testování identit s maticemi a polynomy

9.1 Matice

9.2 Polynomy

Nyní se budeme zabývat pravděpodobnostním testováním identit s polynomy, tedy rovnosti polynomů více proměnných. Podobně jako u matic stačí testovat rovnost daného polynomu nulovému polynomu.

Stejně jako u testování identit s maticemi se efektivita algoritmu odlišuje podle zadání polynomu. Pokud by polynom byl dán explicitně jako součet monomů, je jednoduché ověřit, zda součet koeficientů u monomů se stejnými mocninami proměnných je vždy 0. Zajímavější je případ, kdy je polynom dán jinak, třeba nějakou krátkou formulí nebo procedurou, která umožňuje vyhodnocení v libovolném bodě. Pak můžeme krátce zakódovat polynom s exponenciálně mnoha různými monomy, např. $(1+x_1)(1+x_2)\cdots(1+x_n)$ a rozvinutí polynomu a porovnání všech monomů není efektivní. Tomuto problému se říká testování polynomiálních identit, zkráceně PIT (z anglického "polynomial identity testing"). V našich aplikacích budou polynomy často dány ve formě symbolického determinantu.

Pokud pracujeme nad konečnými tělesy, je testování polynomiálních identit jiný problém než testování, zda se polynom rovná nule pro všechny hodnoty proměnných. Například nad dvouprvkovým tělesem se polynom $x^2 - x$ vždy vyhodnotí na 0, i když se o nulový polynom nejedná. (Je-li polynom roven nulovému, pak ovšem i jeho hodnota je vždy 0. Pro polynomy více proměnných je složitost obou problémů podstatně různá. Na problém testování, zda je polynom všude nula, nad dvouprvkovým tělesem se dá zredukovat jazyk výrokových tautologií, a to je coNP-úplný problém. Pro testování polynomiálních identit si předvedeme efektivní pravděpodobnostní algoritmus. Tento rozdíl ovšem existuje jen u konečných těles; je-li těleso nekonečné, pak polynom rovný všude 0 je rovný nulovému polynomu. Jak uvidíme, totéž platí i v konečném tělese, které je dostatečně velké v závislosti na stupni polynomu.

V našem algoritmu budeme ve skutečnosti testovat, zda je polynom rovný nule pro náhodně vybraný bod z dostatečně velké množiny. K tomu je nutné umět odhadnout počet kořenů. U polynomů s jednou proměnnou je to jednoduché: Polynom stupně nejvýše d má nejvýše d kořenů. Obdobný odhad platí i pro polynomy více proměnných. Řekneme, že **celkový stupeň** polynomu je nejvýše d jestliže v každém monomu (v explicitní reprezentaci) je součet stupňů všech proměnných nejvýše d.

Lemma 9.1. *Nechť $P(x_1, \ldots, x_n)$ je polynom v n proměnných celkového stupně d nad tělesem K, který není identicky nulový. Nechť $S \subseteq K$ je konečná neprázdná a nechť $r_1, \ldots, r_n \in S$ jsou nezávislé uniformně náhodné proměnné. Pak*

$$Pr[P(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}$$

Důkaz. Indukcí podle n. Napišme polynom P jako $P(x_1, \ldots, x_n) = x_1^k A(x_2, \ldots, x_n) + B(x_1, \ldots, x_n)$, kde x_1^k je nejvyšší mocnina x_1 v P a polynom B má stupeň v x_1 menší než k (tj. A sdružuje všechny monomy P s x_1^k).

Uvědomme si, že pro každé dva jevy E a F platí $Pr[E] \leq Pr[F] + Pr[E \mid \neg F]$. Použijeme tuto nerovnost pro jevy $E \iff P(r_1, \ldots, r_n) = 0$ a $F \iff A(r_2, \ldots, r_n) = 0$.

Protože stupeň A je nejvýše $d - k$ a $A \not\equiv 0$, z indukčního předpokladu plyne $Pr[F] = Pr[A(r_2, \ldots, r_n) = 0] \leq (d-k)/|S|$.

Pro každou konkrétní hodnotu $r_2, \ldots, r_n$, pro kterou $A(r_2, \ldots, r_n) \neq 0$, je $P(x_1, r_2, \ldots, r_n)$ polynom stupně právě k a tedy má nejvýše k kořenů mezi $|S|$ možnými hodnotami pro r_1. Náhodné r_1 je tedy kořenem s pravděpodobností nejvýše $k/|S|$. Průměrováním přes všechny hodnoty $r_2, \ldots, r_n$ dostaneme $Pr[E \mid \neg F] = Pr[P(r_1, \ldots, r_n) = 0 \mid A(r_2, \ldots, r_n) \neq 0] \leq k/|S|$.

Celkově tedy $Pr[P(r_1, \ldots, r_n) = 0] \leq (d-k)/|S| + k/|S| = d/|S|$. □

Algoritmus je zřejmý...

Pokud by existoval deterministický algoritmus pracující v polynomiálním čase, plynulo by z toho, že buď některé problémy v NEXP nelze počítat polynomiálními booleovskými obvody nebo permanent nelze počítat polynomiálními aritmetickými obvody.

10 Perfektní párování

Nyní algoritmus pro testování identit polynomů použijeme k testování existence perferktního párování a později postup modifikujeme pro nalezení párování.

Testovat budeme ve skutečnosti nenulovost určitých symbolických determinantů. Nyní definujeme potřebné matice a dokážeme jejich základní strukturální vlastnosti.

Edmondsova matice bipartitního grafu $G = (U, V, E)$ na vrcholech $U = \{u_1, \ldots, u_n\}$ a $V = \{v_1, \ldots, v_n\}$ je matice polynomů B rozměru $n \times n$ definována předpisem

$$B_{ij} = \begin{cases} x_{ij} & \text{pro } (u_i, v_j) \in E \\ 0 & \text{jinak.} \end{cases}$$

Věta 10.1. *Nechť B je Edmondsova matice bipartitního grafu $G = (U, V, E)$. Pak $rank(det(B))$ je roven velikosti největšího párování v G.*

Tutteho matice grafu $G = (V, E)$ na n vrcholech $V = \{v_1, \ldots, v_n\}$ je matice polynomů B rozměru $n \times n$ definována předpisem

$$B_{ij} = \begin{cases} x_{\{i,j\}} & \text{pro } \{v_i, v_j\} \in E \text{ a } i < j \\ -x_{\{i,j\}} & \text{pro } \{v_i, v_j\} \in E \text{ a } i > j \\ 0 & \text{jinak.} \end{cases}$$

Definice 10.2. Nechť m je monom polynomu s proměnnými x_e, $e \in E$. Pak $F(m)$ označuje multimnožinu hran z E, kde hrana e se vyskytuje tolikrát, kolik je stupeň proměnné x_e v m.

Lemma 10.3. *Nechť m je monom, který má v determinantu Tutteho matice grafu $G = (V, E)$ nenulový koeficient. Pak multigraf $(V, F(m))$ je disjunktním sjednocením sudých cyklů.*

Věta 10.4. *Nechť B je Tutteho matice grafu $G = (V, E)$. Pak $rank(det(B))$ je roven dvojnásobku velikosti největšího párování v G.*

10.1 Testování existence perfektního párování

Počítat determinanty umíme v polynomiálním čase (nejlepší známé algoritmy potřebují $n^{2.3727}$ aritmetických operací).

10.2 Paralelní algoritmus pro nalezení perfektního párování

Determinant umíme počítat rychle i na paralelních strojích. Determinant matice s k-bitovými čísly umíme počítat v čase $O(\log^2 n)$ na $O(n^{3.5}k)$ procesorech.

Naivní algoritmus, jeho korektnost pokud existuje jediné párování...

Izolující lemma...

Modifikovaná verze Tutteho matice bipartitního a obecného grafu...

PARALELNÍ PERFEKTNÍ PÁROVÁNÍ
Vstup: Graf $G = (V, E)$.
(1) $M := \emptyset$. Pro každou hranu $\{u, v\}$ vyber nezávisle uniformně náhodně váhu $w_{\{u,v\}} \in \{0, \ldots, 2|E| - 1\}$.
(2) Nechť B je Tutteho matice grafu G po substituci $2^{w_{uv}}$ za x_{uv}.
(3) Spočítej $det(B)$ a W takové, že 2^{2W} je nejvyšší mocnina dvojky, která dělí $det(B)$.
(4) Pro každou hranu $\{u, v\} \in E$ paralelně spočítej $d = det(C^{\{u,v\}})$, kde $C^{\{u,v\}}$ je matice, která vznikne z B vynecháním dvou řádků a dvou sloupců odpovídajících u a v. Jestliže $d \cdot 2^{2w_{uv}}/2^{2W}$ je liché celé číslo, přidej $\{u, v\}$ do M.
(5) Paralelně pro každý vrchol zkontroluj, zda jeho stupeň je 1. Pokud ano, vystup M.

Lemma 10.5. *(i) Jestliže 2^{2W} je největší mocnina dvojky, která dělí $det(B)$, pak graf v G existuje perfektní párování M s váhou nejvýše W.*
(ii) Jestliže graf G má jediné perfektní párování M minimální váhy W, pak 2^{2W} je největší mocnina dvojky, která dělí $det(B)$. Navíc hrana $\{u, v\} \in E$ je v M právě když největší mocnina dvojky, která dělí $C^{\{u,v\}}$, je $2^{2W}/2^{2w_{uv}}$.

Důkaz. (i):
(ii): Předpokládejme, že G má jediné perfektní párování M minimálni váhy W a $uv \in E$. Hrana uv může ale nemusí být v M. Nechť G' je podgraf indukovan □

Věta 10.6. *Algoritmus najde perfektní párování s pravděpodobností alespoň $1/2$ a pracuje v čase $O(\log^2 n)$ na $n^{O(1)}$ procesorech.*

Existence deterministického paralelního algoritmu podobné efektivity je otevřená.

11 Rozvrhování na počítačích s nezávislými časy

Nyní budeme studovat následující zobecnění rozvrhování na identických strojích. Vstupem je matice hodnot p_{ij}, kde p_{ij} udává dobu trvání úlohy j na stroji i. Tyto hodnoty jsou na sobě zcela nezávislé. Cílem je opět minimalizovat délku rozvrhu.

> ROZVRHOVÁNÍ NA STROJÍCH S NEZÁVISLÝMI ČASY
> *Vstup:* Počet strojů m, časy běhu úloh $p_{ij} \in \mathbb{R}^+, i = 1, \ldots, m, j = 1, \ldots n$.
> *Výstup:* Rozvrh, tj. rozklad množiny $\{1, \ldots, n\}$ na množiny $I_1, \ldots, I_m$.
> *Cíl:* minimalizovat délku rozvrhu, t.j. $\max_{i \in \{1,\ldots,m\}} \sum_{j \in I_i} p_{ij}$.

Následující lineární program je přirozenou relaxací problému a celočíselná řešení přesně odpovídají rozvrhům.

$$\begin{aligned} \text{minimalizuj} \quad & T \\ \text{pro} \quad & T \geq 0,\ x_{ij} \geq 0, \quad i = 1, \ldots, m,\ j = 1, \ldots n \\ \text{za podmínek} \quad & \sum_{i=1}^{m} x_{ij} = 1 \qquad \text{pro } j = 1, \ldots, n \\ & \sum_{j=1}^{n} p_{ij} x_{ij} \leq T \qquad \text{pro } i = 1, \ldots, m \end{aligned} \tag{11.1}$$

Bohužel ale jeho optimum může být libovolně vzdálené celočíselnému optimu, takže ho pro aproximaci nemůžeme použít. Příklad...

Lépe nám poslouží následující lineární program pro rozhodovací verzi problému s parametrem T udávajícím požadovanou délku rozvrhu. Všechna přípustná celočíselná řešení odpovídají přesně všem rozvrhům délky nejvýše T. Navíc bude mít tu vlastnost, že pokud má přípustné zlomkové řešení, tak existuje celočíselné řešení s délkou rozvrhu nejvýše $2T$.

$$\begin{aligned} \text{minimalizuj} \quad & 0 \\ \text{pro} \quad & x_{ij} \geq 0, \quad i = 1, \ldots, m,\ j = 1, \ldots n \\ \text{za podmínek} \quad & \sum_{i=1}^{m} x_{ij} = 1 \qquad \text{pro } j = 1, \ldots, n \\ & \sum_{j=1}^{n} p_{ij} x_{ij} \leq T \qquad \text{pro } i = 1, \ldots, m \\ & x_{ij} = 0 \qquad \text{pro } p_{ij} > T \end{aligned} \tag{11.2}$$

Pro dané řešení lineárního programu definujme *graf zlomkových proměnných* jako bipartitní graf G, kde vrcholy jedné partity jsou stroje $\{1, \ldots, m\}$, vrcholy druhé partity jsou úlohy $\{1, \ldots, n\}$ a stroj i je spojen hranou s úlohou j právě když x_{ij} je neceločíselná.

Lemma 11.1. *Pokud je lineární program (11.2) přípustný, tak existuje optimální řešení takové, že graf neceločíselných proměnných G je les. Navíc takové řešení lze nalézt v polynomiálním čase.*

Důkaz. Pokud G obsahuje cyklus, ukážeme, že můžeme upravit hodnoty proměnných odpovídajícím hranám cyklu tak, že neceločíselných proměnných bude méně. ... □

LP-UNRELATED
Vstup: Časy $p_{ij} \in \mathbb{R}^+, i = 1, \ldots, m, j = 1, \ldots n$.
(1) Najdi počáteční přípustný rozvrh tak, že každá úloha j se rozvrhne na počítač i s minimálním p_{ij}; nechť jeho délka je τ.
(2) Binárním vyhledáváním T v intervalu $[\tau/m, \tau]$ najdi minimální hodnotu T, pro kterou má lineární program (11.2) přípustné řešení.
(3) Nechť $x_{ij}, i = 1, \ldots, m, j = 1, \ldots n$ je optimum lineárního programu (11.2) takové, že graf neceločíselných proměnných G je les.
(4) Dokud existuje neceločíselná proměnná x_{ij}, proveď:
Nechť i a j jsou takové, že x_{ij} je jediná neceločíselná z $x_{ij'}, j' = 1, \ldots n$. Přiřaď úlohu j celočíselně na stroj i, tj. $x_{ij} := 1$ a $x_{i'j} := -0$ pro $i' \neq i$.
Výstup: Rozklad, kde $I_i := \{j \mid x_{ij} = 1\}$.

Věta 11.2. *Algoritmus* LP-UNRELATED *lze implementovat v polynomiálním čase a výsledek je* 2*-aproximační algoritmus pro rozvrhování s nezávislými časy.*

Důkaz. Jestliže les G je neprázdný, tak obsahuje list. List nemůže být úloha, protože součet proměnných u úlohy je vždy 1. List je tedy stroj a můžeme provést krok (4). Zachová se...

Krok (4) provedeme pro každý stroj i nejvýše jednou a čas úloh přiřazených na stroj i se tím zvýší nejvýše o T. čelková délka rozvrhu je tedy na konci nejvýše $2T$. □

12 Dělení zdrojů (cake cutting)

Budeme studovat situaci, kdy chceme rozdělit nějaký majetek mezi n hráčů. Problém je, že různí hráči různě oceňují různé části majetku. Naopak situaci zjednoduší, že majetek se dá libovolně dělit (a tedy také jeho nedělitelné části mají nulovou hodnotu). Příklad může být třeba dort nebo pozemek.

Příkladem vhodného protokolu je tradiční dělení na dvě části: Jeden hráč rozdělí dort na dva kusy a druhý hráč si z nich jeden vybere. Otázka je, co bude dělat 10 hráčů.

Formálně majetek modelujeme jako interval $C = [0, 1]$ a jeho ohodnocení hráčem i jako spojitou neklesající funkci $F_i : [0, 1] \to [0, 1]$ splňující $F_i(0) = 0$ a $F_i(1) = 1$. Hodnota intervalu $[0, x]$ pro hráče i je $F_i(x)$; normalizovali jsme tedy hodnocení hráčů tak, že hodnota C je pro každého hráče 1. Z jiného pohledu F_i není nic jiného než distrubuce pravděpodobnosti na C. Můžeme tedy stejně dobře mluvit o míře μ_i, která definuje hodnotu každé měřitelné množiny. (Taková míra má některé speciální vlastnosti, např. každý bod má míru 0.)

Výpočetní model je následující. Míry hráčů nejsou známy, ale protokol může kteréhokoliv hráče požádat, aby provedl *řez* dané hodnoty. Na požadavek řezu v bodě α odpoví hráč i bodem x takovým, že $F_i(x) = \alpha$. Navíc se může hráče zeptat, jakou hodnotu má řez provedený jiným hráčem. Na konci protokol jako výstup vydá rozklad C na n částí pro jednotlivé hráče.

Protokol považujeme za *spravedlivý*, jestliže každý hráč má za to, že hodnota jeho části je alespoň $1/n$. Je i řada jiných možností, které také vedou k zajímavým problémům úplně jiného charakteru. (Např. můžeme požadovat, že každý hráč si má cenit své části více než jakékoliv jiné části.)

Složitost protokolu měříme počtem řezů, počet dotazů na hodnotu nepočítáme. (Řez tedy považujeme za složitou operaci, což má své opodstatnění. Koneckonců nechceme, aby dort byl úplně rozdrobený, ale není tak podstatné, jak dlouho si o něm budeme povídat.) Z toho tedy plyne, že se můžeme beztrestně zeptat všech hráčů na hodnotu každého provedeného

řezu. Pak také můžeme požadavky na řezy zadávat relativně k nějakému podintervalu, což nám usnadní formulaci rekurzivních protokolů.

Ve skutečnosti naše protokoly budou v mnoha ohledech jednoduché: Nebudeme potřebovat znát přesnou hodnotu jednotlivých řezů, postačily by otázky typu "Je $F_i(x) > \alpha$?". Také výsledné části jsou vždy intervaly.

12.1 Deterministický protokol

Pro jednoduchost budeme předpokládat, že počet hráčů je mocnina dvojky. Zobecnění pro libovolný počet hráčů je přímočaré. Pro rekurzi je vhodné formulovat algoritmus tak, že dělíme obecný interval.

Idea protokolu je jednoduchá. Hledáme řez takový, že obě vzniklé části bude preferovat polovina hráčů. To provedeme deterministicky snadno tak, že každý hráč provede řez v polovině podle své míry a z těchto řezů vybereme medián. Přesnou formulace trochu znepřehledňuje fakt, že tyto řezy nemusí být různé, což může nechávat určitou volnost pro přiřazení hráčů k polovinám intervalu.

DETERMINISTICKÉ DĚLENÍ INTERVALU
Vstup: Interval I, 2^t hráčů s mírami μ_i.
(1) Pokud $t = 0$, tak jediný hráč dostane celý interval, konec.
(2) Každý hráč provede řez x_i v I s hodnotou $\mu_i(I)/2$. Nechť X je 2^{t-1} nejmenší hodnota řezu (medián).
(3) Rozděl hráče na dvě množiny L a R velikosti 2^{t-1} tak, že $(\forall i \in L)x_i \geq X$ a $(\forall i \in R)x_i \leq X$.
(4) Rekurzivně rozděl interval $I_l = I \cap [0, X]$ mezi hráče L a interval $I_R = I \cap (X, 1]$ mezi hráče R.

Věta 12.1. *Protokol najde spravedlivé rozdělení a provede $O(n \log n)$ řezů.*

Důkaz. Protože X je medián řezů, rozdělení v kroku (3) vždy existuje. Z jeho definice pak plyne, že pro hráče v L má interval I_l hodnotu alespoň poloviny I a pro hráče v R má interval I_r hodnotu alespoň poloviny I. Z toho plyne, že výsledné rozdělení je spravedlivé.

Počet řezů protokolu před rekurzivním voláním je 2^t pro $t \geq 1$. Celkem indukcí sečteme, že počet řezů je přesně $t2^t$. □

Počet řezů $O(n \log n)$ je optimální pro deterministické protokoly; důkaz je elementární ale není jednoduchý.

12.2 Pravděpodobnostní protokol

Pro pravděpodobnostní protokol si uvědomíme, že v kroku (2) není nutné dělat všechny řezy. Pokud provedeme jeden řez, rozdělíme hráče na ty, kteří preferují levou a pravou část intervalu. V té části, kterou preferuje méně hráčů, už další řezy dělat nemusíme, a můžeme se omezit na hráče v druhé polovině. Pokud hráče pro řez vybereme náhodně, průměrný počet zbývajících hráčů se změní o čtvrtinu a medián tedy nalezneme za pomoci $O(logn)$ řezů. Myšlenka je obdobná jako při pravděpodobnostním vyhledávaní mediánu, ovšem nyní měříme počet řezů, nikoliv počet porovnání.

Krok (2) deterministického protokolu nahradíme následujícím podprotokolem. V obecném stavu máme I rozdělený na tři intervaly I_l, J, I_r a hráči jsou rozdělení do tří množin: v L

jsou ti, kteří preferují i samotný interval I_l; v R, kteří preferují i samotný interval I_r; konečně v J jsou ti, u kterých záleží na rozdělení intervalu J.

PRAVDĚPODOBNOSTNÍ HLEDÁNÍ MEDIÁNU
Vstup: Interval I, 2^t hráčů s mírami μ_i.

Vstup: Medián X z hodnot řezů x_i v I s hodnotou $\mu_i(I)/2$.
(1) $L := \emptyset$, $R := \emptyset$, $M := \{1, \ldots, 2^t\}$, $I_l := \emptyset$, $I_r := \emptyset$, $J := I$.
Opakuj body (2)-(5):
(2) Vyber uniformně náhodně hráče $i \in M$ a nech ho provést řez x_i v I s hodnotou $\mu_i(I)/2$. Nechť je L' množina hráčů $j \in M$, pro které má $I \cap [0, X)$ hodnotu alespoň $\mu_j(I)/2$ a R' množina hráčů $j \in M$, pro které má $I \cap [0, X)$ hodnotu nejvýše $\mu_j(I)/2$.
(3) Jestliže $|L \cap L'| \leq 2^{t-1}$ tak $L := L \cap L'$, $M := M \setminus L'$, $I_l := I \cap [0, x_i]$, $J := J \cap (x_i, 1]$.
(4) Jestliže $|R \cap R'| \leq 2^{t-1}$ tak $R := R \cap R'$, $M := M \setminus R'$, $I_r := I \cap [x_i, 1]$, $J := J \cap [0, x_i)$.
(5) Jestliže nenastane žádný z předchozích případů anebo $M := \emptyset$, vrať $X = x_i$.

Věta 12.2. *Pravděpodobnostní protokol najde spravedlivé rozdělení s průměrným počtem řezů $O(n)$.*

Důkaz. Pokud X je medián řezů, korektnost algoritmu plyne stejně jako v deterministickém případě. Během hledání mediánu zachováváme invariant, že v L jsou přesně hráči, jejichž řez by byl v I_l a v R jsou přesně hráči, jejichž řez by byl v I_r, tedy v M jsou přesně hráči, jejichž řez by byl v J. Navíc $|L|, |R| \leq 2^{t-1}$, a tedy medián řezů je vždy v J. Jedině v případě, že medián je přesně poslední řez tak nemůžeme rozdělení předefinovat, ovšem v tom případě skončíme a vrátíme správné X.

Při každém řezu se J zmenší o jednu z množin L' nebo R', které dohromady dávají J. Pokud náhodně vybraný hráč má k-tý řez v pořadí z hráčů v J, pak $|L'| \geq k$ a $|R'| \geq |J| - k$. Z toho plyne, že průměrná velikost J po provedení cyklu je nejvýše $\frac{3}{4}|J|$. Z toho dále plyne, že po $O(t)$ krocích je průměrná velikost J nejvýše 1 a tedy průměrný počet opakování cyklu je $O(t)$. Počet řezů je tedy také $O(t)$.

Celkový počet řezů je $O(2^t)$ ze standardní rekurence. □