Lineární programování

Úvod pro informatiky

Jiří Matoušek
KAM MFF UK

Verze 27/VI/2006

1

Co to je a k čemu může být

Lineární programování kupodivu nesouvisí s programováním počítačů. Termín byl zaveden v dobách, kdy počítačů bylo málo a ještě byly většinou utajené. Slovo programování bylo převzato z americké armádní hantýrky, kde se používalo ve významu rozvrh či plán nějaké činnosti, například výcviku. Cílem lineárního programování tehdy bylo stanovit optimální plán. Slovo lineární pak napovídá, že přípustné plány jsou vymezeny lineárními podmínkami pro uvažované veličiny, a také že kvalita plánu (třeba náklady nebo trvání) se poměřuje nějakou lineární funkcí těchto veličin.

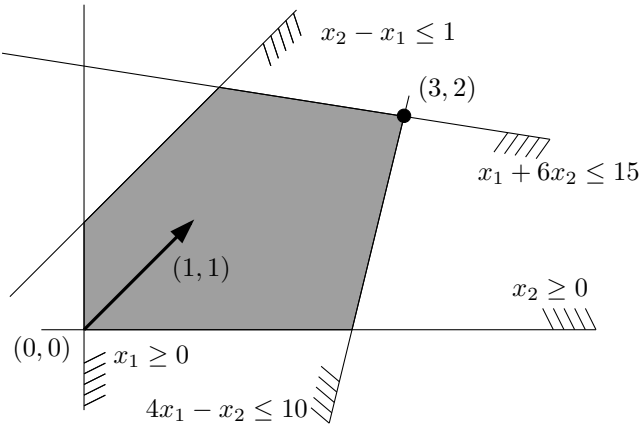
V podobném duchu se lineární programování brzy začalo používat pro plánování všemožných ekonomických aktivit, například přepravy surovin a výrobků mezi továrnami, osívání polí všelijakými plodinami nebo řezání velkých papírových rolí na menší v rozměrech objednaných zákazníky. Sousloví „plánování s lineárními omezujícími podmínkami“ by asi lépe vystihovalo tento původní smysl lineárního programování. Nicméně termín lineární programování se mezitím ustálil, a zároveň se jeho význam podstatně rozšířil: zdaleka už nehraje roli jen v matematické ekonomii, ale objevuje se hojně například i v informatice.

1.1 Úloha lineárního programování

Začneme velmi jednoduchým příkladem úlohy lineárního programování:

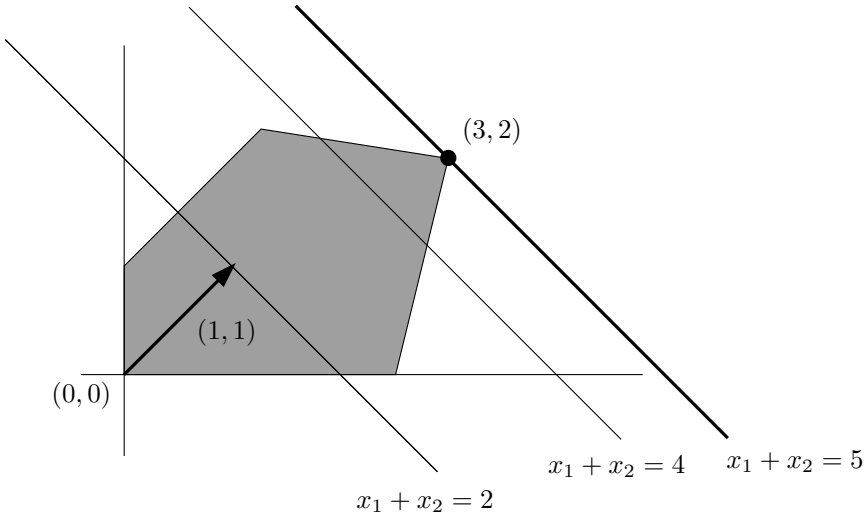
maximalizovat hodnotu	$x_1 + x_2$
mezi všemi vektory $(x_1, x_2) \in \mathbb{R}^2$	
splňujícími podmínky	$x_1 \geq 0$
	$x_2 \geq 0$
	$x_2 - x_1 \leq 1$
	$x_1 + 6x_2 \leq 15$
	$4x_1 - x_2 \leq 10.$

K této úloze snadno nakreslíme obrázek. Množina $\{\mathbf{x} \in \mathbb{R}^2 : x_2 - x_1 \leq 1\}$ je polorovina pod přímkou $x_2 = x_1 + 1$, a podobně každá ze zbývajících čtyř nerovnic definuje polorovinu. Množina všech vektorů splňujících všech pět podmínek zároveň je konvexní mnohoúhelník:



Který bod mnohoúhelníka maximalizuje hodnotu $x_1 + x_2$? Ten, který leží „nejdále ve směru“ vektoru $(1, 1)$, vyznačeného šipkou, čili bod $(3, 2)$.

Obrat „nejdále ve směru“ je v uvozovkách, protože je poněkud nepřesný. Přesněji, uvážíme přímku kolmou k té šipce, rovnoběžně ji posunujeme ve směru šipky a hledáme bod, v němž naposledy protne náš mnohoúhelník. (Funkce $x_1 + x_2$ je totiž na každé přímce kolmé k vektoru $(1, 1)$ konstantní, a když přímku posunujeme ve směru tohoto vektoru, hodnota funkce roste.) Viz obrázek:



V obecné úloze lineárního programování chceme najít vektor $\mathbf{x}^* \in \mathbb{R}^n$ maximalizující (nebo minimalizující) hodnotu dané lineární funkce mezi všemi vektory $\mathbf{x} \in \mathbb{R}^n$ splňujícími danou soustavu lineárních rovnic a nerovnic. Lineární funkce, která se má maximalizovat, případně minimalizovat, se jmenuje **účelová funkce** a je tvaru $\mathbf{c}^T\mathbf{x} = c_1x_1 + \cdots + c_nx_n$, kde $\mathbf{c} \in \mathbb{R}^n$ je daný vektor[1]. Lineárním rovnicím a nerovnicím v úloze se zpravidla říká **omezující podmínky** nebo **omezení**. Počet omezujících podmínek se obvykle značí m.

Úloha lineárního programování se často zapisuje pomocí matic a vektorů, podobně jako soustava lineárních rovnic se v lineární algebře píše $A\mathbf{x} = \mathbf{b}$. Abychom si takový zápis usnadnili, můžeme každou rovnici v úloze nahradit dvěma nerovnicemi. Například místo omezující podmínky $x_1 + 3x_2 = 7$ bychom dali dvě omezení $x_1 + 3x_2 \leq 7$ a $x_1 + 3x_2 \geq 7$. Dále lze směr nerovností obrátit změnou znamének: $x_1 + 3x_2 \geq 7$ je ekvivalentní $-x_1 - 3x_2 \leq -7$, a tudíž můžeme předpokládat, že všechny nerovnosti jsou třeba „$\leq$“. Konečně minimalizace účelové funkce $\mathbf{c}^T\mathbf{x}$ je ekvivalentní maximalizaci $-\mathbf{c}^T\mathbf{x}$, takže můžeme vždy přejít třeba k maximalizační úloze. Po těchto úpravách můžeme každou úlohu lineárního programování zapsat takto:

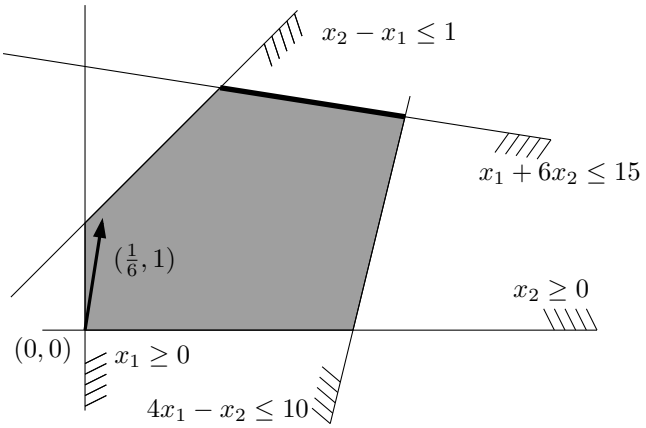
$$\begin{array}{ll} \text{maximalizovat hodnotu} & \mathbf{c}^T\mathbf{x} \\ \text{přes všechny vektory } \mathbf{x} \in \mathbb{R}^n \text{ splňující podmínku} & A\mathbf{x} \leq \mathbf{b}, \end{array}$$

[1] Vektor $\mathbf{c} \in \mathbb{R}^n$ chápeme také jako matici typu $n \times 1$, podobně jako ostatní vektory v tomto textu.

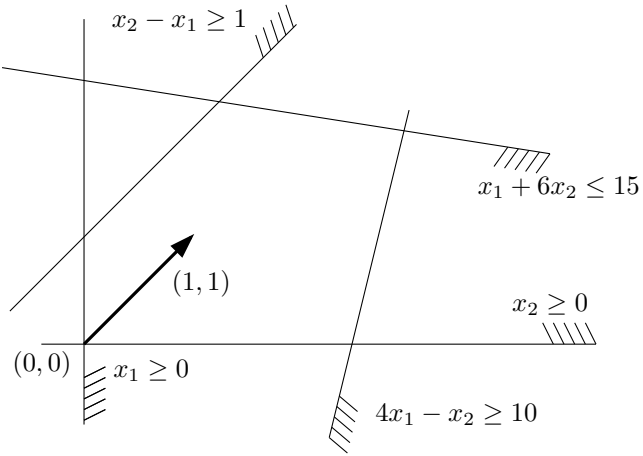
kde A je daná reálná matice typu $m \times n$ a $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ jsou dané vektory. Přitom relace $\leq$ platí pro vektory stejné délky právě tehdy, když platí po složkách.

Vektor $\mathbf{x} \in \mathbb{R}^n$, jenž splňuje všechna omezení dané úlohy, se nazývá **přípustné řešení**. Každému $\mathbf{x}^* \in \mathbb{R}^n$, které dává největší možnou hodnotu $\mathbf{c}^T\mathbf{x}$ mezi všemi přípustnými $\mathbf{x}$, se říká **optimální řešení** nebo krátce **optimum**. V úvodní úloze máme $n = 2$, $m = 5$, a $\mathbf{c} = (1, 1)$. Jediné optimální řešení je vektor $(3, 2)$, kdežto například $(2, \frac{3}{2})$ je přípustné řešení, které není optimální.

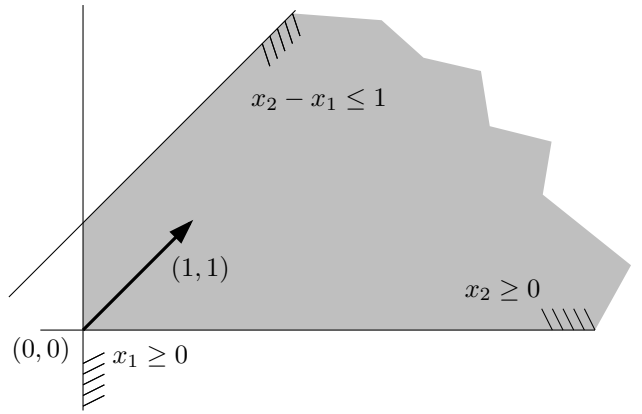
Obecně může úloha lineárního programování mít jediné optimální řešení, nebo nekonečně mnoho optimálních řešení, nebo ani jedno. Situaci s jedním optimálním řešením jsme viděli v úvodní úloze. Změníme-li vektor $\mathbf{c}$ na $(\frac{1}{6}, 1)$, budou optimálními řešeními všechny body na silně vytažené straně pětiúhelníka:



Obrátíme-li v úvodní úloze směr nerovností v omezeních $x_2 - x_1 \leq 1$ a $4x_1 - x_2 \leq 10$, dostaneme úlohu, jež nemá žádné přípustné řešení, a tedy ani řešení optimální:



Konečně optimální řešení existovat nemusí, ani když přípustná řešení existují, totiž tehdy, nabývá-li účelová funkce libovolně velkých hodnot (taková úloha se nazývá **neomezená**). To se stane, třeba když z úvodní úlohy odstraníme omezení $4x_1 - x_2 \leq 10$ a $x_1 + 6x_2 \leq 15$:



V kapitole 4 dokážeme, že žádné podstatně jiné situace nastat nemohou, neboli že každá úloha lineárního programování je buď nepřípustná (nemá žádné přípustné řešení), nebo neomezená, nebo má optimální řešení.

Úvodní úlohu jsme vyřešili graficky. Šlo to dobře, protože jsme měli jenom dvě proměnné. Úlohu se čtyřmi proměnnými ale už obrázkem neznázorníme, natož abychom ji dovedli obrázkově řešit, a přitom pořádná úloha lineárního programování může mít třeba desetitisíce proměnných a podmínek. Grafické znázornění je užitečné pro pochopení pojmů a postupů

lineárního programování, ale jako výpočetní metoda je bezcenné. Někdy může být dokonce i zavádějící, protože objekty ve velké dimenzi se mohou chovat dost jinak, než napovídá intuice získaná v rovině nebo ve třídimenzionálním prostoru.

Jeden z hlavních poznatků, které by si měl člověk o lineárním programování navždy zapamatovat, je:

> Úloha lineárního programování je efektivně řešitelná, jak v teorii tak v praxi.

(Aby poznatek dával smysl, nesmí se samozřejmě ani zapomenout, co úloha lineárního programování je.)

- *V praxi* je na její řešení k dispozici řada softwarových balíků. Zvládnou vstupy s desítkami tisíc proměnných a podmínek. Úlohy se speciální strukturou, jako například s malým počtem nenulových koeficientů v každé omezující podmínce, lze často řešit i při mnohem větších počtech proměnných a omezení.

- *V teorii* byly vyvinuty algoritmy, které dokazatelně vyřeší každou úlohu lineárního programování v čase omezeném jistou polynomiální funkcí velikosti vstupu. Velikost vstupu se přitom měří jako celkový počet bitů, potřebných k zapsání všech koeficientů v účelové funkci a ve všech omezujících podmínkách.

Tato konstatování shrnují výsledky dlouhého a usilovného výzkumu a efektivní metody řešení úlohy lineárního programování nejsou nijak jednoduché.

1.2 Co najdete v tomto spisku

Zbytek kapitoly 1 uvádí lineární programování do souvislosti s lineární algebrou a krátce pojednává o jeho historii a aplikacích.

U naprosté většiny čtenářů se dá očekávat, že setkají-li se kdy s lineárním programováním ve výzkumu či v praxi, budou je používat jako černou skříňku. Z tohoto hlediska je klíčová kapitola 2 popisující řadu výpočetních problémů řešitelných přes lineární programování. V navazující kapitole 3 se mluví o takzvaném celočíselném programování, v němž se také optimalizuje lineární funkce přes množinu určenou lineárními omezeními, ale proměnné navíc musí nabývat celočíselných hodnot. V této souvislosti se ukáže, jak lineární programování může pomoci například přibližnému řešení obtížných výpočetních problémů.

Kapitola 4 přináší základní teoretické poznatky o lineárním programování a o geometrické struktuře množiny všech přípustných řešení. Pojmy jako konvexita a konvexní mnohostěn, s nimiž se tam pracuje, jsou důležité v mnoha dalších odvětvích matematiky a teoretické informatiky.

Další kapitola pokrývá simplexovou metodu, což je základní algoritmus pro řešení úloh lineárního programování. Simplexová metoda se všemi detaily je poměrně komplikovaná, a z dnešního hlediska není pro první seznámení s lineárním programováním nezbytná, i když některé tradiční úvody do lineárního programování se omezují prakticky jenom na ni.

V kapitole 6 zformulujeme a dokážeme větu o dualitě, jeden z hlavních teoretických výsledků lineárního programování a velice užitečný důkazový nástroj.

Kapitola 7 pojednává o dalších dvou důležitých algoritmických přístupech k lineárnímu programování, elipsoidové metodě a metodách vnitřního bodu. Oba jsou poměrně složité, a zde jen naznačíme hlavní myšlenky bez nároku na úplnost a přesnost.

Dvě úrovně textu. Toto pojednání má být především učebním textem pro první ročník MFF UK. Přitom ale mnohé podstatné výsledky o lineárním programování, které by bylo škoda vynechat, jsou složité, případně používají matematické metody, jejichž znalost se v prvním ročníku nedá předpokládat. Proto je text rozdělen do dvou úrovní. Na základní úrovni jsem se snažil o úplnost a dostatečnou podrobnost důkazů.

> Druhá, rozšiřující či „osvětová“ úroveň textu je typograficky vyznačena takto. V takových částech, určených hlavně pro matematicky zralejší čtenáře, řekněme studenty vyšších ročníků, zařazuji nástiny důkazů a ne zcela přesné formulace pokročilejších výsledků. Komu připadají nesrozumitelné, může je prostě ignorovat – základní text by měl dávat smysl i bez nich.

Ani celý základní text se ovšem v přednášce v prvním ročníku zdaleka neprobere, takže čtenář si z něj může vybrat podle sylabu a podle vlastního zájmu.

1.3 Lineární programování a lineární algebra

Na základy lineární algebry můžeme pohlížet jako na teorii řešení soustav lineárních rovnic. V lineární algebře se samozřejmě dělá i mnoho jiných věcí, ale soustavy lineárních rovnic jsou jedním z hlavních témat. Klíčovým algoritmem je Gaussova eliminace, která efektivně najde řešení takové soustavy, a dokonce i popis množiny všech řešení. Geometricky je množina všech řešení afinní podprostor $\mathbb{R}^n$, což je důležitý lineárně algebraický pojem.

Disciplínu lineárního programování můžeme v podobném duchu považovat za teorii řešení soustav lineárních *nerovnic*.

> To je v úloze lineárního programování trochu zatemněno tím, že nehledáme libovolné řešení dané soustavy nerovnic, nýbrž řešení maximalizující danou účelovou funkci. Dá se ale ukázat, že najít jedno (libovolné) přípustné řešení úlohy lineárního programování, pokud existuje, je z výpočetního hlediska problém téměř stejně obtížný jako najít řešení optimální. Postup, jak získat optimální řešení, když umíme počítat přípustná řešení, zde jen naznačíme (elegantnější způsob popíšeme v sekci 6.1). Když například předem víme, že optimální hodnota účelové funkce v dané úloze je mezi 0 a 100, můžeme se nejdřív ptát, jestli existuje přípustné $\mathbf{x} \in \mathbb{R}^n$, pro něž je účelová funkce aspoň 50. To jest, přidáme k omezujícím podmínkám další nerovnici požadující, aby účelová funkce byla aspoň 50, a zjistíme, jestli tato pomocná úloha má přípustné řešení. Pokud ano, budeme se dále stejným trikem ptát, jestli účelová funkce může být aspoň 75, a pokud ne, budeme zjišťovat, jestli může být aspoň 25, atd. Čtenář s informatickými podmíněnými reflexy už nejspíš poznal strategii binárního vyhledávání, jíž můžeme optimální hodnotu účelové funkce poměrně rychle lokalizovat s velkou přesností.

Množina všech řešení soustavy lineárních nerovnic s n proměnnými je geometricky průnikem konečně mnoha poloprostorů v $\mathbb{R}^n$. Takové množině se říká *konvexní mnohostěn*, a známými příklady konvexních mnohostěnů v $\mathbb{R}^3$ jsou krychle, kvádr, čtyřstěn a pravidelný dvanáctistěn. Konvexní mnohostěny jsou objekty matematicky mnohem složitější než vektorové či afinní podprostory. Můžeme být vlastně vděční, že v úloze lineárního programování máme účelovou funkci: jako řešení stačí vypočítat jediný bod $\mathbf{x}^* \in \mathbb{R}^n$ a nemusíme se trápit s celým mnohostěnem.

Roli podobnou té, již má v lineární algebře Gaussova eliminace, hraje v lineárním programování *simplexová metoda*. To je algoritmus, který řeší úlohu lineárního programování, většinou dosti efektivně, a umožňuje též dokazovat výsledky teoretické.

Paralely mezi lineární algebrou a lineárním programováním ještě shrneme v tabulce:

	Základní úloha	Algoritmus	Množina řešení
Lineární algebra	soustava lineárních rovnic	Gaussova eliminace	afinní podprostor
Lineární programování	soustava lineárních nerovnic	simplexová metoda	konvexní mnohostěn

1.4 Význam a historie lineárního programování

Simplexová metoda byla ve zvláštním čísle časopisu *Computing in Science & Engineering* zařazena mezi deset algoritmů, které nejvíce ovlivnily vývoj vědy a techniky ve dvacátém století[2]. I když se jistě leckdo může přít, že simplexová metoda správně patří třeba až na místo čtrnácté, a každé takové hodnocení je nutně subjektivní, důležitost lineárního programování je zpochybnitelná těžko.

Simplexovou metodu vymyslel a rozpracoval v roce 1947 George Dantzig ve službách amerického vojenského letectva. Už dříve, v roce 1939, byl Leonid Vitalijevič Kantorovič v Sovětském Svazu pověřen reorganizací dřevozpracujícího průmyslu a zformuloval přitom jistou speciální třídu úloh lineárního programování a zárodečnou podobu simplexové metody na jejich řešení. Jak už to bývá v takových státech, jeho objevy se neprosadily a nikdo na ně nenavázal. Kantorovič spolu s Tjallingem Koopmansem dostali v roce 1975 Nobelovu cenu v oboru ekonomie za průkopnické práce o optimální alokaci zdrojů. Poněkud ironicky Dantzig, jehož příspěvek k lineárnímu programování je bezpochyby významnější, Nobelovu cenu nedostal – jeho práce byla patrně tehdy shledána na cenu v ekonomii příliš matematickou.

Objev simplexové metody ovlivnil ekonomickou teorii i praxi. I na manažery zvyklé spoléhat na zkušenost a intuici udělalo dojem, když se náklady snížily třeba o 20% pouhou reorganizací podle jakéhosi záhadného výpočtu. Zvlášť, když to dokázal na základě pár čísel někdo, kdo nemohl vědět, jak to v podniku chodí, zdaleka tak dobře jako oni. Najednou už v konkurenčním prostředí nešlo matematické metody beztrestně ignorovat.

Lineární programování se od čtyřicátých let velice rozvinulo, a také se pro něj našly nové typy aplikací, zdaleka ne jenom v matematické ekonomii.

V informatice se z něj stal základní nástroj v konstrukci algoritmů. Pro řadu výpočetních úloh se poprvé podařilo ukázat existenci teoreticky efektivního (polynomiálního) algoritmu přes obecné techniky založené na lineárním programování. Pro obtížné výpočetní úlohy (NP-těžké, pokud tento termín čtenáři něco řekne), které zpravidla není naděje vyřešit přesně, se hledají přibližné algoritmy, a lineární programování je jednou z hlavních částí nejmocnějších známých metod.

Další překvapivé použití lineárního programování je ryze teoretické: *věta o dualitě*, k níž se dostaneme v kapitole 6, se objevuje v důkazech matema-

[2]Zbývajících devět algoritmů je Metropolisova metoda pro Monte Carlo algoritmy, Krylovova metoda řešení soustav lineárních rovnic, dekompozice v maticovém počítání, optimalizující kompilátor Fortranu, QR algoritmus na výpočet vlastních čísel, třídicí algoritmus Quicksort, rychlá Fourierova transformace, algoritmus na hledání celočíselných relací mezi reálnými čísly a rychlá multipolární metoda na výpočet elektrostatických a gravitačních potenciálů.

tických tvrzení, především v kombinatorice, a poskytuje sjednocující abstraktní náhled na mnoho zdánlivě nesouvisejících výsledků. Právě zmíněné metody důkazů a konstrukce algoritmů jsou naneštěstí příliš pokročilé na to, abychom si na ně mohli troufnout v krátkém úvodním textu.

Počítače jsou dnes nesrovnatelně rychlejší než třeba před padesáti lety, takže nikoho neudiví, že dnes lze řešit podstatně větší úlohy lineárního programování. Ale kupodivu větší podíl než zrychlení počítačů má na tomto zvětšení zvládnutelných úloh teoretický pokrok v algoritmech. To dokumentuje, jak se rozrostla i samotná teorie algoritmů lineárního programování. Několik zásadních výsledků nastíníme v kapitole 7.

2

Příklady

Lineární programování je znamenitý nástroj. Aby ho však člověk mohl použít, musí napřed pojmout podezření, že by uvažovaný výpočetní problém mohl jít vyjádřit jako úloha lineárního programování, a potom ho tak doopravdy vyjádřit. Ani jeden z těchto kroků nemusí být jednoduchý. V této kapitole ukážeme aspoň malou část ze širokého spektra úloh, na něž se lineární programování hodí, a budeme demonstrovat několik triků, jak přeformulovat problémy, jež na první pohled na lineární programování nevypadají. Další příklady zmíníme v kapitole 3.

Jakmile se nám podaří uvažovaný problém formulovat jako úlohu lineárního programování, můžeme nasadit některý z obecných algoritmů. Z hlediska programátora, řekněme, je to způsob velmi pohodlný, protože stačí do obecného algoritmu správně zadat účelovou funkci a všechny omezující podmínky.

Nemusí to ale být postup výpočetně nejefektivnější. Pro řadu problémů jsou známy specializované algoritmy, které pracují podstatně rychleji než obecný přístup založený na lineárním programování. Tak například studium toků v sítích (oddíl 2.2) tvoří dnes poměrně rozsáhlý podobor teoretické informatiky, s řadou specializovaných (a někdy dosti složitých) algoritmů. Počítat maximální tok přes lineární programování není pro velké úlohy nejlepší přístup.

Nicméně i u problémů, kde se lineární programování nakonec neukáže jako nejefektivnější možná metoda, má často smysl s ním začít. Můžeme tak rychle získat fungující prototyp algoritmu a na jeho základě se například rozhodnout, jestli se vyplatí vyvíjet pro uvažovanou úlohu specializovaný kód.

2.1 Vaříme zdravě a levně

Potravní inspekce Evropské unie odhalila, že jídla podávaná v restauračním zařízení „U neurvalce“, jako například utopenci, slanečci a guláš se šesti, neodpovídají novým předpisům, a jmenovitě zmínila ve zprávě nedostatek vitamínů A a C a vlákniny. Provozovatel restaurace se snaží nedostatky řešit dodáním zeleninové přílohy, kterou hodlá zkombinovat z bílého zelí, mrkve a zásob nakládaných okurek nalezených ve sklepě. Následující tabulka shrnuje číselné podklady: předepsaná množství jednotlivých vitamínů a minerálů na porci, jejich zastoupení v příslušných potravinách a jednotkové ceny potravin[1].

surovina	mrkev syrová	zelí bílé syrové	okurky nakládané	požadováno na 1 porci
vitamín A [mg/kg]	35	0.5	0.5	0.5 mg
vitamín C [mg/kg]	60	300	10	15 mg
vláknina [g/kg]	30	20	10	4 g
cena [Kč/kg]	15	10	3*	—

*Zůstatková účetní cena zásob, patrně neprodejných.

Za jakou minimální dodatečnou cenu na každou porci lze požadavkům inspekce tímto způsobem vyhovět? To lze vyjádřit následující úlohou lineárního programování:

$$
\begin{array}{ll}
\text{minimalizovat} & 15x_M + 10x_Z + 3x_O \\
\text{za podmínek} & x_M \geq 0 \\
& x_Z \geq 0 \\
& x_O \geq 0 \\
& 35x_M + 0.5x_Z + 0.5x_O \geq 0.5 \\
& 60x_M + 300x_Z + 10x_O \geq 15 \\
& 30x_M + 20x_Z + 10x_O \geq 4.
\end{array}
$$

Proměnná x_M udává množství mrkve přidané k jedné porci, a podobně pro x_Z a x_O. Účelová funkce vyjadřuje cenu této kombinace. Množství mrkve, zelí i okurek jsou vždy nezáporná, což říkají podmínky $x_M \geq 0$, $x_Z \geq 0$, $x_O \geq 0$ (kdybychom je nezahrnuli, optimální řešení by možná mohlo mít množství některé dražší suroviny záporné, čímž by se jakoby ušetřilo). Konečně nerovnice na posledních třech řádcích specifikují dodržení celkového obsahu vitamínů A a C a vlákniny.

Úlohu můžeme vyřešit například simplexovou metodou. Optimální řešení dává cenu 1,40Kč s dávkou 9,5g mrkve, 38g zelí a 290g okurek na porci

[1] Pro ty, kdo se zajímají o zdravou výživu: obsahy vitamínů a další údaje jsou víceméně realistické.

(vše zaokrouhleno na dvě platné číslice). Další inspekcí by nejspíš neprošlo – ve skutečnosti by asi bylo třeba přidat další omezení, například že okurek nesmí být příliš mnoho.

Tento příklad jsem zařadil, abych příliš nevybočoval z řady. Jak se zdá, všechny úvody do lineárního programování začínají různými vyživovacími příklady, nejspíš proto, že první větší optimalizační problém, na kterém se simplexový algoritmus v roce 1947 testoval, bylo složení potravních dávek. Jaké potraviny zkombinovat a v jakém množství, aby se dodržela minimální množství všech důležitých živin a aby denní příděl přitom vyšel co nejlevněji? Tehdy zformulovaná úloha lineárního programování měla 77 proměnných a 9 omezení a její řešení simplexovým algoritmem na ručních kalkulátorech zabralo asi 1000 pracovních hodin.

Později, když už George Dantzig měl přístup k počítači, zkoušel optimalizovat dietu i pro sebe. Optimální řešení první úlohy lineárního programování, již k tomu cíli sestavil, doporučovalo konzumaci několika litrů octa denně. Když z dalšího zadání ocet odstranil, vyšlo jako nejvhodnější základ stravy zhruba 200 polévkových kostek na den.

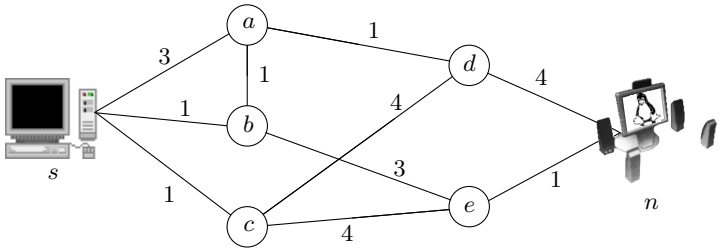
Tato historka, jejíž pravdivost není zcela vyloučena, nijak nesnižuje sílu lineárního programování, ale ilustruje, jak těžké je matematicky zachytit všechny důležité aspekty problému ze života. V oblasti výživy například dodnes není jasné, jaký přesně mají jednotlivé potraviny vliv na lidský organismus (i když samozřejmě spousta věcí jasných je, a naděje, že věda budoucnosti bude doporučovat například tlačenku jako hlavní ingredienci zdravé stravy, budou patrně zklamány). I kdyby to bylo jasné dokonale, málokdo chce a umí přesně zformulovat požadavky, co by od své výživy požadoval – mnohem snáz se to zřejmě dělá pro výživu někoho jiného. Navíc mezi potřebou různých živin jsou různé nelineární závislosti, takže úloha lineárního programování nemůže problém výživy dokonale vystihnout.

Lineární programování se běžně využívá v průmyslu, v zemědělství i ve službách, ale mnohé z takových aplikací jsou z abstraktního hlediska pouhými variantami problému výživy a neobsahují žádné zásadně nové matematické triky. Sestavit v praxi pro takové problémy dobré modely může stále být obtížné, ale obtížnost není rázu matematického. Proto se zde takovými úlohami nebudeme dále zabývat (spousta příkladů je uvedena v Chvátalově knize zmiňované v kapitole 8) a ukážeme si problémy, ve kterých má využití lineárního programování jiný charakter.

2.2 Tok v síti

Správce počítačové sítě přemluvil zaměstnavatele ke koupi nového stroje s kvalitnějšími reproduktory, a chce na něj po síti přenést svoji hudební

sbírku. Síť vypadá takto:



Jaké největší přenosové rychlosti může dosáhnout z počítače s (starý) na počítač n (nový)? Čísla u jednotlivých propojení udávají jejich přenosovou rychlost (třeba v MB/s). Předpokládáme také, že každým propojením se dají přenášet data oběma směry, ale ne zároveň. Například propojením ab se dají *buď* posílat data z a do b jakoukoli rychlostí mezi 0 a 1 MB/s, *nebo* posílat data z b do a jakoukoli rychlostí mezi 0 a 1 MB/s, ale obojí najednou nelze.

Uzly $a, b, \ldots, e$ nejsou uzpůsobeny ke skladování většího množství dat, takže všechna data, která do nich přicházejí, se musí ihned posílat dál. Z toho už je vidět, že nelze využít maximální kapacitu všech propojení, a pro každé propojení je tedy potřeba najít správnou hodnotu datového toku, aby celková přenosová rychlost z s do n byla co největší.

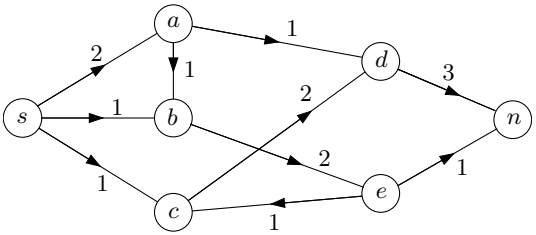
Pro každé propojení zavedeme jednu proměnnou. Například x_{be} udává, jakou rychlostí se data budou přenášet z b do e. Přitom x_{be} může být i záporné, což znamená, že data tečou obráceným směrem, z e do b. (Tudíž další proměnnou x_{eb}, která by udávala rychlost přenosu z e do b, už zavádět nepotřebujeme.) Budeme mít 10 proměnných x_{sa}, x_{sb}, x_{sc}, x_{ab}, x_{ad}, x_{be}, x_{cd}, x_{ce}, x_{dn} a x_{en}. Úloha lineárního programování bude:

$$
\begin{array}{ll}
\text{maximalizovat} & x_{sa} + x_{sb} + x_{sc} \\
\text{za podmínek} & -3 \le x_{sa} \le 3, \quad -1 \le x_{sb} \le 1, \quad -1 \le x_{sc} \le 1 \\
& -1 \le x_{ab} \le 1, \quad -1 \le x_{ad} \le 1, \quad -3 \le x_{be} \le 3 \\
& -4 \le x_{cd} \le 4, \quad -4 \le x_{ce} \le 4, \quad -4 \le x_{dn} \le 4 \\
& -1 \le x_{en} \le 1 \\
& \qquad x_{sa} = x_{ab} + x_{ad} \\
& x_{sb} + x_{ab} = x_{be} \\
& \qquad x_{sc} = x_{cd} + x_{ce} \\
& x_{ad} + x_{cd} = x_{dn} \\
& x_{be} + x_{ce} = x_{en}.
\end{array}
$$

Účelová funkce $x_{sa} + x_{sb} + x_{sc}$ udává, jakou celkovou rychlostí se data odesílají z počítače s. Poněvadž se nikde neukládají ani (doufejme) neztrácejí,

musí se stejnou rychlostí přijímat v n. Dalších 10 podmínek $-3 \leq x_{sa} \leq 3$ až $-1 \leq x_{en} \leq 1$ omezuje přenosové kapacity. Zbývající podmínky pak říkají, že cokoliv do každého z uzlů a až e přichází, musí také odcházet.

Optimální řešení této úlohy vypadá takto:



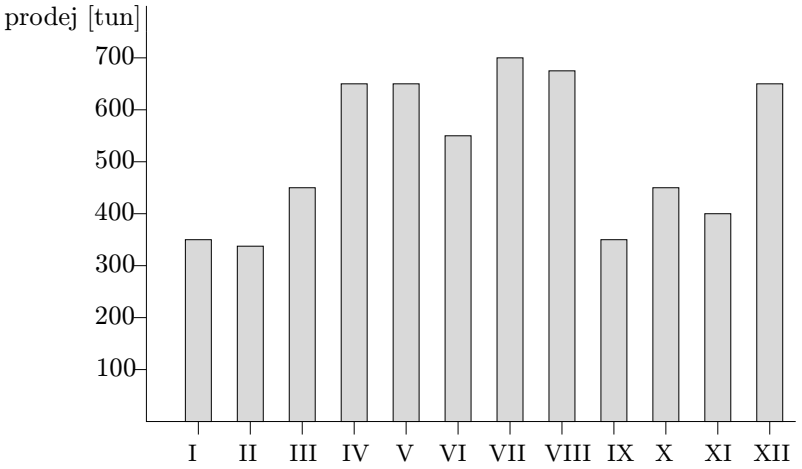
Číslo u každého propojení udává velikost příslušného datového toku, a šipka určuje jeho směr. Všimněte si, že mezi c a e je potřeba posílat data ve směru z e do c, takže $x_{ce} = -1$. Hodnota účelové funkce je 4, což je požadovaná maximální přenosová rychlost.

V tomto příkladu je snadno vidět, že přenosová rychlost nemůže být větší, protože celková kapacita propojení počítačů s a a se zbytkem sítě je 4. To je speciální případ pozoruhodné věty o maximálním toku a minimálním řezu, která se probírá v kombinatorice.

Uvedený příklad datového toku v síti je malý, jednoduchý a ne úplně realistický. V praxi se ovšem uvažují toky ve složitých sítích, dokonce s mnoha zdrojovými a cílovými uzly. Mohou to být například sítě elektrické (teče proud), silniční či železniční (tečou vlaky nebo auta), telefonní (tečou hlasové nebo datové signály), finanční (tečou peníze) a tak dále.

2.3 Zmrzlina po celý rok

Další aplikace lineárního programování souvisí opět s jídlem, což by vzhledem k důležitosti jídla v životě a složitosti optimalizace spánku nebo lásky nemělo být příliš překvapivé. Výrobce zmrzliny potřebuje sestavit plán výroby na další rok. Marketingové oddělení vydalo následující předpověď měsíční spotřeby zmrzliny v příštím roce založenou na datech z předchozích let, rozsáhlém průzkumu trhu a pozorování ptactva:



Jak má vypadat plán výroby, který takovouto poptávku uspokojí?

Jednoduchým řešením by byla produkce „právě včas“, tedy že zmrzlina spotřebovaná v měsíci i se v měsíci i také vyrobí. To ovšem znamená, že se objem výroby bude z měsíce na měsíc velice měnit, a takové změny znamenají nezanedbatelné náklady: bude potřeba najmout nebo propustit sezónní pracovníky, přenastavit stroje a tak dále. Takže by bylo lepší produkovat zmrzlinu rovnoměrněji během celého roku: v měsících s nízkou poptávkou by se nevyužitá kapacita továrny zaměstnala výrobou zmrzliny do zásoby na měsíce s vysokou poptávkou.

Jiným jednoduchým řešením by tedy mohl být naprosto „rovnoměrný“ plán výroby s týmž objemem v každém měsíci. Po chvíli uvažování se zjistí, že takový rozvrh nemusí být možný, pokud chceme na konci roku skončit bez přebytků. Ale i když je takové řešení uskutečnitelné, nemusí být ideální, protože skladováním zmrzliny také vznikají netriviální náklady. Nejvhodnější plán výroby patrně bude někde mezi těmito dvěma extrémy (výrobou sledující poptávku a výrobou neměnnou). Chceme najít kompromis, při kterém jsou celkové náklady na změny výrobní kapacity a na skladování přebytků minimální.

Abychom úlohu mohli zapsat formálně, zavedeme pro poptávku v měsíci i (v tunách) nezápornou proměnnou d_i. Dále zavedeme nezáporné proměnné x_i označující produkci v měsíci i a další nezáporné proměnné s_i pro celkovou skladovou zásobu na konci měsíce i. Na uspokojení poptávky v měsíci i se dá použít zmrzlina vyrobená v měsíci i a přebytky z měsíce $i-1$:

$$x_i + s_{i-1} \geq d_i \quad \text{pro } i = 1, 2, \ldots, 12.$$

Množství $x_i + s_{i-1} - d_i$ je přesně přebytek na konci měsíce i, a tedy

$$x_i + s_{i-1} - s_i = d_i \quad \text{pro } i = 1, 2, \ldots, 12.$$

Předpokládáme, že na začátku ledna nebyla žádná zásoba, a tudíž definujeme $s_0 = 0$. (Pokud bychom vzali v úvahu i minulou výrobu, s_0 by byl přebytek z předchozího roku.) Také dosadíme $s_{12} = 0$, pokud ovšem nechceme pokračovat v plánování i na další rok.

Chceme najít takové nezáporné řešení těchto rovnic a nerovnic, které odpovídá nejnižším celkovým nákladům. Předpokládejme, že změna produkce o 1 tunu mezi měsíci $i-1$ a i stojí 1500 Kč a skladování 1 tuny zmrzliny stojí 600 Kč na měsíc. Potom celkové náklady vyjadřuje funkce

$$1500 \sum_{i=1}^{12} |x_i - x_{i-1}| + 600 \sum_{i=1}^{12} s_i,$$

kde dosadíme $x_0 = 0$ (opět by se dala snadno vzít v úvahu i výroba z minulého roku).

Tato účelová funkce sice bohužel není lineární, ale naštěstí existuje jednoduchý (ale důležitý) trik, který nám ji umožní na lineární převést, a to za cenu zavedení dalších proměnných.

Změna v produkci je buď zvýšení nebo snížení. Zavedeme nezápornou proměnnou y_i pro zvýšení výroby v měsíci i oproti měsíci $i-1$ a nezápornou proměnnou z_i pro snížení. Potom

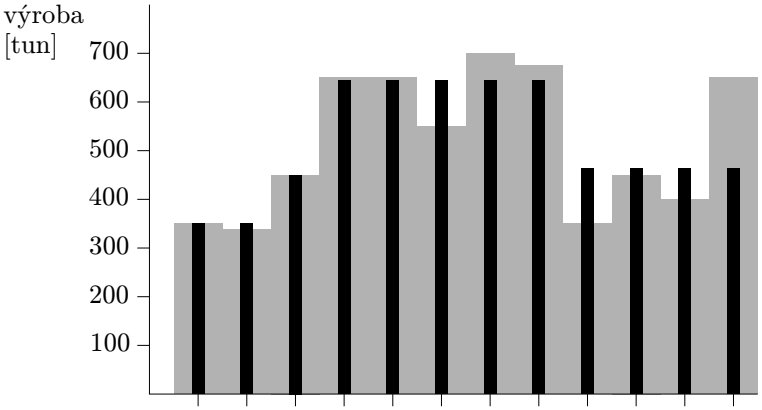
$$x_i - x_{i-1} = y_i - z_i \text{ a } |x_i - x_{i-1}| = y_i + z_i.$$

Plán výroby s minimálními celkovými náklady se tudíž dá najít jako optimální řešení následující úlohy lineárního programování:

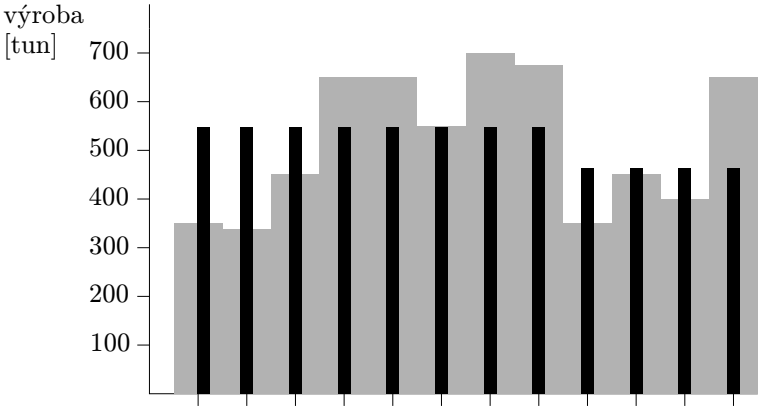
$$\begin{array}{ll}
\text{minimalizovat} & 1500 \sum_{i=1}^{12} y_i + 1500 \sum_{i=1}^{12} z_i + 600 \sum_{i=1}^{12} s_i \\
\text{za podmínek} & x_i + s_{i-1} - s_i = d_i \text{ pro } i = 1, 2, \ldots, 12 \\
& x_i - x_{i-1} = y_i - z_i \text{ pro } i = 1, 2, \ldots, 12 \\
& x_0 = 0 \\
& s_0 = 0 \\
& s_{12} = 0 \\
& x_i, s_i, y_i, z_i \geq 0 \text{ pro } i = 1, 2, \ldots, 12.
\end{array}$$

Abychom ověřili, že optimální řešení $(\mathbf{s}^*, \mathbf{y}^*, \mathbf{z}^*)$ této úlohy lineárního programování skutečně odpovídá plánu výroby, povšimneme si, že pro všechna i musí y_i^* nebo z_i^* být nulové, jinak by šlo obě zmenšit a dostat lepší řešení. To znamená, že $y_i^* + z_i^*$ se skutečně rovná změně produkce mezi měsíci $i-1$ a i.

Když tento lineární program vyřešíme pro výše uvedená zmrzlinářská data, dostaneme následující plán výroby (znázorněný černými sloupky, zatímco šedivý graf odpovídá poptávce).



Další obrázek ukazuje plán výroby, který bychom dostali při nulových nákladech na skladování (tj. po dosazení „0“ místo „600“ ve výše uvedené úloze lineárního programování).



Myšlenka uvedeného řešení je použitelná obecně pro mnoho problémů optimálního řízení. Pěkným příkladem je „Přistání měsíčního modulu“, kdysi populární hra pro programovatelné kalkulačky (nejspíš příliš prostá na to, aby přežila v dnešní konkurenci). Lunární modul s omezenou zásobou paliva klesá kolmo dolů k povrchu Měsíce pod vlivem gravitace a sestup

může řídit brzdnými raketovými motory. Cílem je přistát na povrchu (přibližně) nulovou rychlostí, dřív než dojde palivo. Čtenář může zkusit zformulovat úlohu určit minimálního množství paliva potřebného na měkké přistání pomocí lineárního programování. Přitom je potřeba předpokládat, že tah motorů se mění jen po určitých časových intervalech, řekněme po sekundách (ostatně i ve hře to tak bylo). Jsou-li časové intervaly dostatečně krátké, řešení úlohy se touto diskretizací času téměř nezmění.

Poznamenejme, že v případě přistání lunárního modulu se úloha dá vyřešit přesně pomocí diferenciálního počtu (nebo pomocí matematické teorie optimálního řízení). Ale v situacích jen mírně složitějších je už přesné analytické řešení neschůdné.

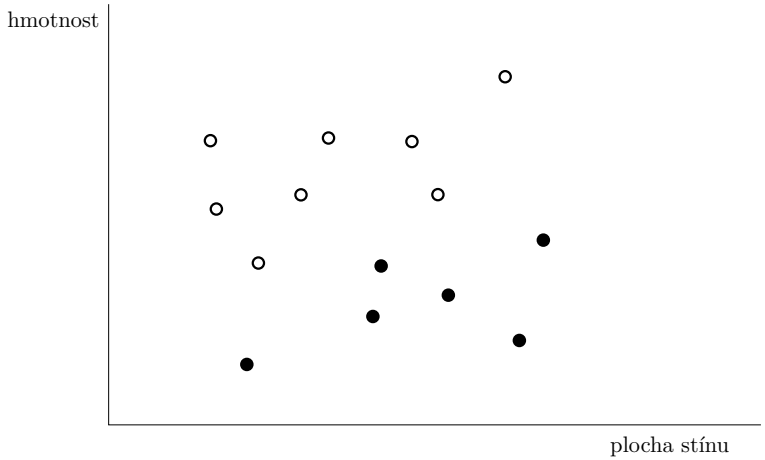
Na závěr zopakujme užitečný trik, který jsme v tomto oddílu předvedli.

> Optimalizační úlohy, které mají v účelové funkci nebo omezujících podmínkách výrazy s absolutními hodnotami, se často dají převést na úlohy lineárního programování vhodným zavedením dalších proměnných.

2.4 Oddělování bodů

Počítačem řízenou králičí past Gromit KP 2.1 chceme naprogramovat tak, aby chytala králíky, ale aby lasičky, které náhodou zabloudí dovnitř, pouštěla zase ven. Past umí chycené zvíře zvážit a také určit plochu jeho stínu.

Na následujícím grafu jsou zobrazena data naměřená pro vzorek králíků a lasiček:



(prázdná kolečka odpovídají králíkům a plná lasičkám).

Zjevně ani hmotnost ani plocha stínu sama o sobě na odlišení králíka od lasičky nestačí. Další jednoduchá možnost odlišení by bylo pomocí lineárního kritéria. Geometricky bychom tedy chtěli oddělit černé body od bílých přímkou. Po překladu do matematické řeči máme dáno m bílých bodů $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ a n černých bodů $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ v rovině a chtěli bychom zjistit, zda existuje přímka, která má všechny bílé body na jedné straně a všechny černé body na druhé straně (na přímce by žádný bod ležet neměl).

Při řešení této otázky rozlišíme tři případy. Nejdřív vyzkoušíme, jestli vyhovuje nějaká svislá přímka. Na to není potřeba ani lineární programování, ani žádná zvláštní chytrost.

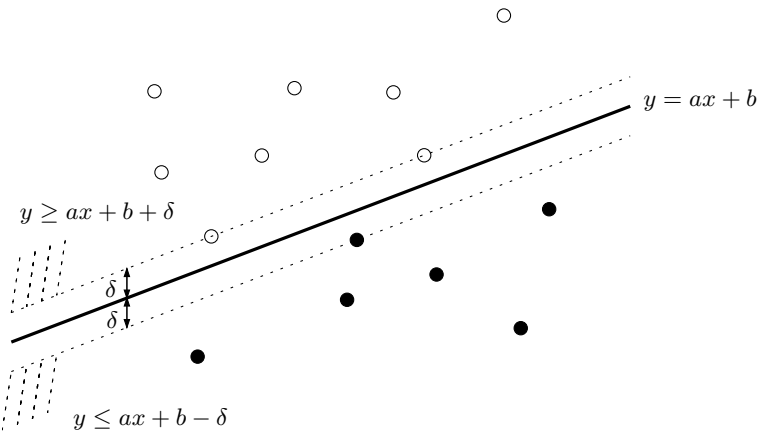
Dále budeme chtít zjistit, zda existuje přímka, která není svislá a má všechny černé body pod sebou a všechny bílé nad sebou. Pišme rovnici takové přímky jako $y = ax + b$, kde a a b jsou zatím neznámá reálná čísla. Bod $\mathbf{r}$ o souřadnicích $x(\mathbf{r})$ a $y(\mathbf{r})$ leží nad touto přímkou pokud $y(\mathbf{r}) > a \cdot x(\mathbf{r}) + b$, a leží pod ní pokud $y(\mathbf{r}) < a \cdot x(\mathbf{r}) + b$. Tudíž vhodná přímka existuje, právě když následující soustava nerovnic pro proměnné a a b má aspoň jedno řešení:

$$\begin{aligned} y(\mathbf{p}_i) &> a \cdot x(\mathbf{p}_i) + b \qquad \text{pro } i = 1, 2, \ldots, m \\ y(\mathbf{q}_j) &< a \cdot x(\mathbf{q}_j) + b \qquad \text{pro } j = 1, 2, \ldots, n. \end{aligned}$$

O ostrých nerovnostech jsme v souvislosti s lineárním programováním zatím nemluvili, a v úloze lineárního programování nejsou povoleny. Ale zde si můžeme pomoci trikem: zavedeme novou pomocnou proměnnou δ,

což bude „mezera“, která zbývá mezi levou a pravou stranou každé ostré nerovnosti. Budeme se pak snažit udělat mezeru co největší:

$$\begin{array}{lll} \text{maximalizovat} & \delta & \\ \text{za podmínek} & y(\mathbf{p}_i) \geq a \cdot x(\mathbf{p}_i) + b + \delta & \text{pro } i = 1, 2, \ldots, m \\ & y(\mathbf{q}_j) \leq a \cdot x(\mathbf{q}_j) + b - \delta & \text{pro } j = 1, 2, \ldots, n. \end{array}$$



Tato úloha lineárního programování má tři proměnné a, b a δ. Optimální δ je kladné, právě když má předchozí soustava nerovnic s ostrými nerovnostmi aspoň jedno řešení, a to nastane, právě když existuje přímka s černými body pod sebou a bílými nad sebou.

Podobně se vyřeší třetí případ, totiž přímka, která není svislá a má všechny černé body nad sebou a všechny bílé pod sebou.

Podobně se dá najít rovina oddělující dvě množiny bodů v $\mathbb{R}^3$ i řešit analogickou úlohu ve vyšších dimenzích. Takže pokud by na odlišení králíka od lasičky nestačila měření dvou veličin, mohli bychom jich zkusit použít víc.

Zde je jiné, možná překvapivější zobecnění. Představme si, že se oddělení králíků od lasiček přímkou ukáže jako nemožné. Potom je můžeme zkusit oddělovat grafem kvadratické funkce (paraboly) tvaru $ax^2 + bx + c$. Máme tedy m bílých bodů $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ a n černých bodů $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ v rovině a ptáme se: Existují koeficienty $a, b, c \in \mathbb{R}$ takové, že všechny bílé body leží nad grafem funkce $f(x) = ax^2 + bx + c$ a všechny černé body leží pod ním? Z toho dostaneme systém nerovnic

$$\begin{array}{ll} y(\mathbf{p}_i) > a(x(\mathbf{p}_i))^2 + bx(\mathbf{p}_i) + c, & \text{kde } i = 1, 2, \ldots, m \\ y(\mathbf{q}_j) < a(x(\mathbf{q}_j))^2 + bx(\mathbf{q}_j) + c, & \text{kde } j = 1, 2, \ldots, n. \end{array}$$

Po zavedení proměnné δ jako výše můžeme sestavit následující úlohu lineárního programování s proměnnými a, b, c a δ:

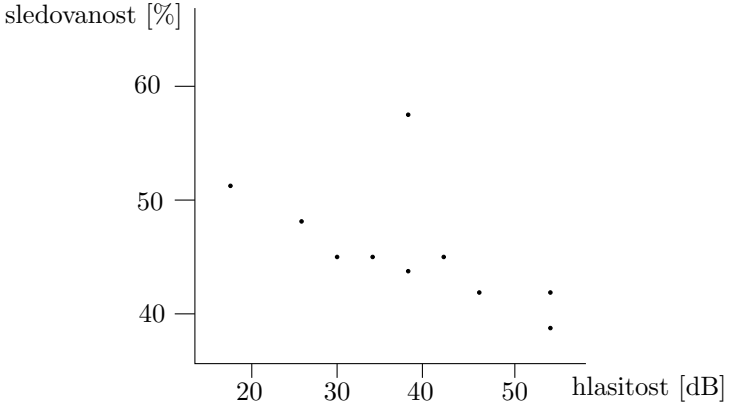
$$\begin{array}{lll} \text{maximalizovat} & \delta & \\ \text{za podmínek} & y(\mathbf{p}_i) \geq ax(\mathbf{p}_i)^2 + bx(\mathbf{p}_i) + c + \delta, & \text{kde } i = 1, 2, \ldots, m \\ & y(\mathbf{q}_j) \leq ax(\mathbf{q}_j)^2 + bx(\mathbf{q}_j) + c - \delta, & \text{kde } j = 1, 2, \ldots, n. \end{array}$$

V této úloze se kvadratické členy vyskytují jen jako koeficienty a nejsou s nimi žádné problémy.

Stejnou metodou můžeme testovat, zda se dvě množiny bodů v rovině nebo ve vyšší dimenzi dají oddělit funkcí tvaru $f(\mathbf{x}) = a_1\varphi_1(\mathbf{x}) + a_2\varphi_2(\mathbf{x}) + \cdots + a_k\varphi_k(\mathbf{x})$, kde $\varphi_1, \ldots, \varphi_k$ jsou dané funkce (mohou být nelineární) a $a_1, a_2, \ldots, a_k$ jsou reálné koeficienty. Oddělování je zde myšleno tak, že pro všechny bílé body $\mathbf{p}_i$ platí $f(\mathbf{p}_i) > 0$ a pro všechny černé body $\mathbf{q}_j$ platí $f(\mathbf{q}_j) < 0$.

2.5 Proložení přímky

Ve městě Bacháni nad Piplavou se večer měřila hlasitost slavičího zpěvu a zjišťovalo se procento lidí, kteří ten den sledovali televizní zprávy. Naměřené hodnoty z řady dnů jsou znázorněny body v rovině:



Nejjednodušší závislosti jsou lineární, a mnoho závislostí lze lineární funkcí dobře aproximovat. Proto chceme zkusit naměřenými body co nejlépe proložit přímku. (Komu nepřipadá uvedený příklad dost realistický, může si vzpomenout na nějaká měření z fyzikálních praktik, kde měřené veličiny skutečně měly záviset přesně lineárně.)

Jak matematicky zformulovat, že přímka je proložena „co nejlépe“? To není vůbec jednoznačné, a v praxi se k prokládání přímek používá řada různých kritérií. Nejpopulárnější je *metoda nejmenších čtverců*, která pro dané

body $(x_1, y_1), \ldots, (x_n, y_n)$ hledá přímku o rovnici $y = ax + b$ minimalizující výraz

$$\sum_{i=1}^{n} (ax_i + b - y_i)^2.$$

Vyjádřeno slovně, pro každý bod vezmeme jeho svislou vzdálenost od té přímky, umocníme ji na druhou, a všechny tyto „čtverce chyb“ sečteme.

Tato metoda nemusí být vždycky nejvhodnější. Je-li například několik málo bodů naměřeno s hodně velkou chybou (nebo když televize zrovna vysílá přímý přenos z lidožroutských obřadů), může to výslednou přímku silně ovlivnit. Chceme-li metodu méně citlivou na malý počet velkých chyb, můžeme místo součtu čtverců chyb minimalizovat součet absolutních hodnot chyb:

$$\sum_{i=1}^{n} |ax_i + b - y_i|.$$

To se kupodivu dá vyjádřit úlohou lineárního programování:

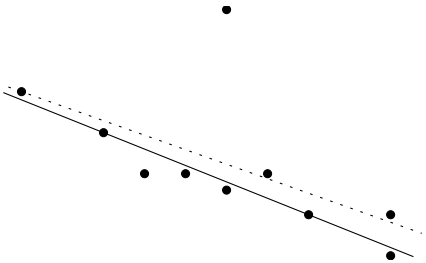
$$\begin{array}{lll} \text{minimalizovat} & e_1 + e_2 + \cdots + e_n & \\ \text{za podmínek} & e_i \geq ax_i + b - y_i & \text{pro } i = 1, 2, \ldots, n \\ & e_i \geq -(ax_i + b - y_i) & \text{pro } i = 1, 2, \ldots, n \end{array}$$

Proměnné jsou a, b a $e_1, e_2, \ldots, e_n$ (kdežto $x_1, \ldots, x_n$ a $y_1, \ldots, y_n$ jsou daná čísla). Každé e_i je pomocná proměnná vyjadřující chybu v i-tém bodě. Podmínky zaručují, že

$$e_i \geq \max\Big(ax_i + b - y_i, -(ax_i + b - y_i)\Big) = |ax_i + b - y_i|.$$

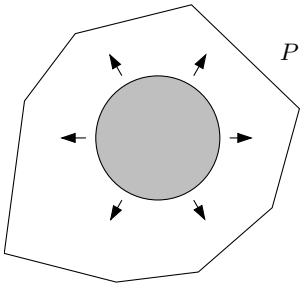
V optimálním řešení bude v této nerovnici pro každé i rovnost, protože jinak bychom mohli příslušná e_i zmenšit.

Obrázek ukazuje přímku proloženou touto metodou (plně) a přímku proloženou metodou nejmenších čtverců (tečkovaně):

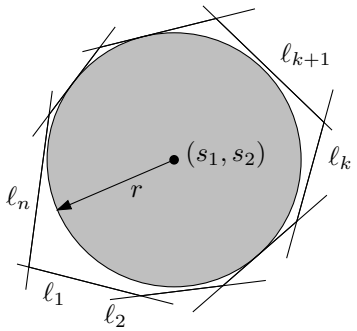


2.6 Kruh v konvexním mnohoúhelníku

Máme dán konvexní n-úhelník P v rovině a chceme najít největší kruh v něm obsažený.



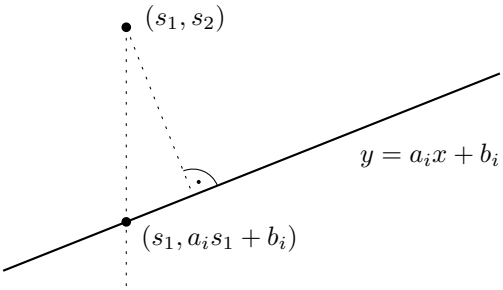
Pro jednoduchost předpokládejme, že žádná ze stran P není svislá. Nechť i-tá strana P leží na přímce ℓ_i o rovnici $y = a_i x + b_i$, $i = 1, 2, \ldots, n$, a zvolme očíslování stran tak, že první, druhá, až k-tá strana je zespodu P, zatímco $(k+1)$-ní až n-tá strana je shora.



Ptejme se teď, kdy kruh se středem $\mathbf{s} = (s_1, s_2)$ a poloměrem r leží celý uvnitř P. Je to právě tehdy, když bod $\mathbf{s}$ má vzdálenost aspoň r od každé z přímek $\ell_1, \ldots, \ell_n$ a leží nad přímkami $\ell_1, \ldots, \ell_k$ a pod přímkami $\ell_{k+1}, \ldots, \ell_n$. Jednoduchým výpočtem pomocí podobnosti trojúhelníků a Pythagorovy věty se zjistí, že absolutní hodnota výrazu

$$\frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}}$$

udává vzdálenost bodu $\mathbf{s}$ od přímky ℓ_i, přičemž výraz je kladný, pokud $\mathbf{s}$ leží nad ℓ_i, a záporný v opačném případě.



Uvažovaný kruh tudíž leží uvnitř P, právě když je splněna každá z nerovnic

$$\begin{aligned} \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}} &\geq r, \qquad i = 1, 2, \ldots, k, \text{ a} \\ \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}} &\leq -r, \quad i = k+1, k+2, \ldots, n. \end{aligned}$$

Chceme tedy najít co největší r takové, že pro něj existují s_1 a s_2 splňující všechny tyto nerovnice. To dává úlohu lineárního programování:

maximalizovat r

za podmínek
$$\begin{aligned} \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}} &\geq r \qquad \text{pro } i = 1, 2, \ldots, k \\ \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}} &\leq -r \qquad \text{pro } i = k+1, k+2, \ldots, n. \end{aligned}$$

(Někdo by se mohl leknout odmocnin, ale ty můžeme vypočítat předem, protože všechna a_i jsou konkrétní daná čísla.) Tato úloha má tři proměnné s_1, s_2, a r. Optimální řešení dává požadovaný největší kruh obsažený v P.

Stejně se dá řešit i analogická úloha ve vyšší dimenzi, například ve třídimenzionálním prostoru: jakou největší kouli můžeme umístit do průniku daných poloprostorů?

Je zajímavé, že podobně vypadající úloha, totiž najít co nejmenší kruh obsahující daný konvexní n-úhelník v rovině, se jako úloha lineárního programování přeformulovat nedá a musí se řešit jinak.

2.7 Příklad z papírenství

Následující typ úlohy z průmyslu se opravdu řešil pomocí lineárního programování, a to zajímavým trikem. Navíc se v něm setkáme s požadavkem celočíselnosti, a to nás přivede k tématu další kapitoly.

Papírna vyrábí papír v rolích standardní šířky 3 m. Odběratelé ale chtějí kupovat role tak široké, jak se hodí jim, a papírna je pro ně musí z 3m rolí nařezat. Jedna 3m role se například může rozříznout na dvě role šíře 93 cm, jednu roli šíře 108 cm, a 6 cm zbytek jde do odpadu.

Uvažme celkovou objednávku

- 97 rolí šíře 135 cm,
- 610 rolí šíře 108 cm,
- 395 rolí šíře 93 cm a
- 211 rolí šíře 42 cm.

Jaký nejmenší počet 3m rolí stačí na tuto objednávku rozřezat, a jak?

K zapojení lineárního programování je potřeba uvažovat velkoryse: vypíšeme všechny délky, které se v objednávkách vyskytují, tedy 42 cm, 93 cm, 108 cm a 135 cm, a pak vyrobíme seznam všech možností, jak 3m roli nařezat na role některých z těchto délek (bereme pouze možnosti, kde odpadní kus je kratší než 42 cm):

M1: 2×135,

M2: $135 + 108 + 42$,

M3: $135 + 93 + 42$,

M4: $135 + 3 \times 42$,

M5: $2 \times 108 + 2 \times 42$,

M6: $108 + 2 \times 93$,

M7: $108 + 93 + 2 \times 42$,

M8: 3×93,

M9: $108 + 4 \times 42$,

M10: $2 \times 93 + 2 \times 42$,

M11: $93 + 4 \times 42$,

M12: 7×42.

Každé možnosti Mj v sestaveném seznamu přiřadíme jednu proměnnou x_j, která reprezentuje počet rolí, nařezaných právě takto. Chceme minimalizovat celkový počet nařezaných rolí, tj. $\sum_{j=1}^{12} x_j$, tak aby každý odběratel byl uspokojen. Například pro splnění objednávky 395 rolí šíře 93 cm je potřeba, aby

$$x_3 + 2x_6 + x_7 + 3x_8 + 2x_{10} + x_{11} \geq 395,$$

a podobnou podmínku dostaneme pro každou z objednaných šířek[2].

Optimální řešení vzniklé úlohy lineárního programování má $x_1 = 48{,}5$, $x_5 = 206{,}25$, $x_6 = 197{,}5$ a ostatní složky 0. Ale abychom mohli nařezat 48,5 rolí podle způsobu M1, museli bychom půlku role rozmotat.

Tady bychom potřebovali vědět o výrobě papíru trochu víc. Je rozmotání poloviny role skutečně technicky a ekonomicky schůdné? Pokud ano, vyřešili jsme problém optimálně. Pokud ne, musíme se snažit dál a nějak se vypořádat tím, že papírnu zajímají jen *celočíselná* řešení uvažovaného lineárního programu. To je obecně nesnadná záležitost a pojednáme o ní v příští kapitole.

[2]Pro skutečnou, asi složitější objednávku by se seznam možností vyráběl nejlépe počítačem, a byli bychom v poměrně typické situaci, kdy by se úloha lineárního programování nezadávala „ručně“, nýbrž generovala nějakým programem. Složitější techniky dokonce generují možnosti postupně, během řešení úlohy lineárního programování, čímž se může silně ušetřit čas a paměť. O tom se pojednává v Chvátalově knize, citované v kapitole 8, odkud je příklad převzat.

3

Celočíselné programování a LP relaxace

3.1 Celočíselné programování

V oddílu 2.7 jsme narazili na situaci, kde byla pro praxi zajímavá jen celočíselná řešení nějaké úlohy lineárního programování. Podobná situace se při pokusech o aplikaci lineárního programování vyskytuje velice často, protože objekty, z nichž se dají snadno oddělovat libovolné zlomky, jsou spíš výjimkou než pravidlem. Při najímání dělníků, nasazování autobusů na linky i řezání papírových rolí se musíme nějak vyrovnat s tím, že dělníci, autobusy i role se vyskytují pouze v celočíselných množstvích.

Někdy stačí složky optimálního řešení prostě zaokrouhlit, a to podle povahy problému buď všechny dolů, nebo všechny nahoru, nebo na nejbližší celá čísla. V příkladu s rolemi papíru z oddílu 2.7 je přirozené zaokrouhlovat nahoru, protože objednávka musí být splněna. Když vyjdeme z optimálního řešení $x_1 = 48{,}5$, $x_5 = 206{,}25$, $x_6 = 197{,}5$, dostaneme zaokrouhlením celočíselné řešení $x_1 = 49$, $x_5 = 207$, $x_6 = 198$ (všechny ostatní složky jsou nulové), což znamená nařezání 454 rolí. Poněvadž známe optimální řešení lineárního programu, víme, že objednávku určitě nemůžeme splnit nařezáním méně než 452,5 rolí. Když trváme na nařezání celočíselného počtu rolí, je nutné nařezat nejméně 453 rolí, a tedy řešení, ke kterému jsme dospěli zaokrouhlením, je poměrně dobré.

Existuje ale řešení poněkud lepší: položíme-li $x_1 = 49$, $x_5 = 207$, $x_6 = 196$, $x_9 = 1$ (a zbylé složky opět nulové), stačí nařezat jen 453 rolí. Jak víme, žádné celočíselné řešení už lepší být nemůže.

Pro jiné úlohy může ale rozdíl mezi zaokrouhleným optimálním řešením lineárního programu a nejlepším možným celočíselným řešením být

mnohem větší. Když třeba optimální řešení lineárního programu říká, že na většinu ze 197 autobusových linek mezi vesnicemi je nejlepší nasadit něco mezi 0,1 a 0,3 autobusu, je jasné, že zaokrouhlení má vliv opravdu radikální.

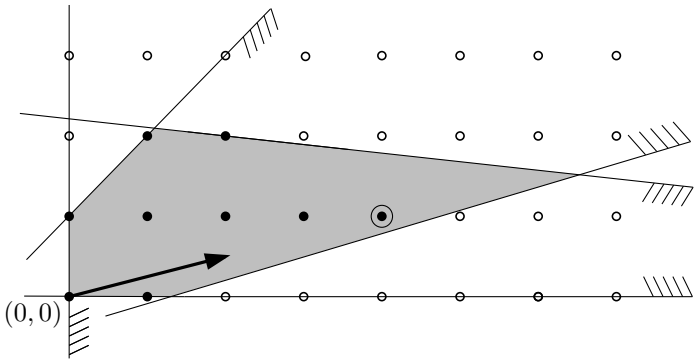
Celočíselné programování. Problém s řezáním rolí ve skutečnosti vede k úloze, v níž máme lineární účelovou funkci a lineární omezující podmínky (rovnice a nerovnice), ale proměnné smějí nabývat pouze celočíselných hodnot. Takové úloze se říká úloha celočíselného programování, a po malé úpravě ji můžeme zapsat podobně, jako jsme v kapitole 1 zapisovali úlohu lineárního programování:

Úloha celočíselného programování:

$$\begin{array}{ll} \text{maximalizovat} & \mathbf{c}^T\mathbf{x} \\ \text{za podmínek} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n. \end{array}$$

Přitom $\mathbb{Z}$ značí množinu všech celých čísel a $\mathbb{Z}^n$ je množina všech n-složkových celočíselných vektorů.

Množina přípustných řešení už není konvexní mnohostěn, jako tomu bylo pro úlohu lineárního programování, ale sestává z jednotlivých celočíselných bodů. Obrázek ilustruje dvoudimenzionální úlohu celočíselného programování s pěti omezeními:



Přípustná řešení jsou znázorněna plnými puntíky a optimální řešení je vyznačeno kroužkem. Všimněte si, že je úplně jinde než optimální řešení úlohy *lineárního* programování s týmiž pěti omezujícími podmínkami a s touž účelovou funkcí.

Je známo, že *obecná úloha celočíselného programování je algoritmicky těžká* (přesně řečeno *NP-úplná*), na rozdíl od úlohy lineárního programování. V praxi se snadno řeší úlohy lineárního programování s desítkami tisíc proměnných a omezení, ale existují např. úlohy celočíselného programování s pouhými 20 proměnnými a 10 omezeními, které jsou nepřekonatelná i pro nejmodernější počítače a software.

> Přidání podmínek celočíselnosti tedy může změnit obtížnost úlohy opravdu drasticky. To přestane vypadat překvapivě, když si všimneme, že celočíselným programováním můžeme modelovat i rozhodnutí ano/ne, protože celočíselná proměnná x_j splňující $0 \leq x_j \leq 1$ má možné hodnoty jen 0 (ne) a 1 (ano). Pro toho, kdo zná základy teorie NP-úplnosti, tak není těžké celočíselným programováním namodelovat problém splnitelnosti logických formulí. V sekci 3.4 uvidíme, jak jím vyjádřit problém největší nezávislé množiny v grafu, což je také jedna ze základních NP-úplných úloh.

Na řešení úloh celočíselného programování byla vyvinuta řada nástrojů. V literatuře se dají najít například pod hesly *odřezávající roviny* (*cutting planes*) a *metoda větví a mezí* (*branch and bound*; to je obecnější strategie, nejen pro celočíselné programování). V nejúspěšnějších strategiích se většinou používá lineární programování jako podprogram pro řešení jistých pomocných úloh. Jak to dělat co nejúčinněji se studuje v odvětví matematiky zvaném *polyedrální kombinatorika*.

Nejrozšířenější aplikací lineárního programování, na níž se spotřebuje i nejvíc strojového času, jsou dnes pravděpodobně pomocné výpočty v úlohách celočíselného programování. Přinejmenším to říkají lidé, kteří by se měli v takových věcech vyznat.

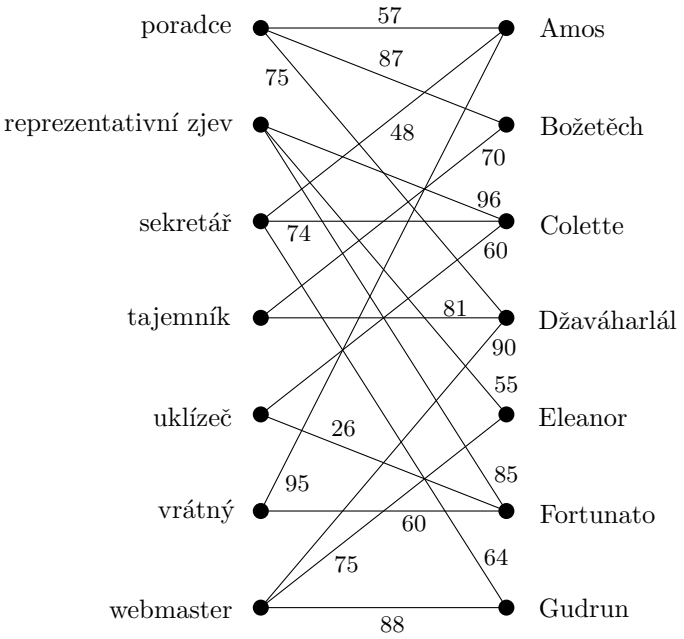
Poznamenejme, že v některých problémech jsou některé z proměnných celočíselné, zatímco jiné mohou nabývat reálných hodnot. Pak se mluví o *smíšeném celočíselném programování*. To je asi nejčastější typ optimalizačních úloh z praxe.

Předvedeme několik důležitých optimalizačních problémů, které se dají snadno formulovat jako úlohy celočíselného programování, a ukážeme, jak se lineární programování dá či nedá použít na jejich řešení. Bude to ale jen malý vzorek z této oblasti, která se v posledních letech velmi rozvinula a používá i docela složité triky.

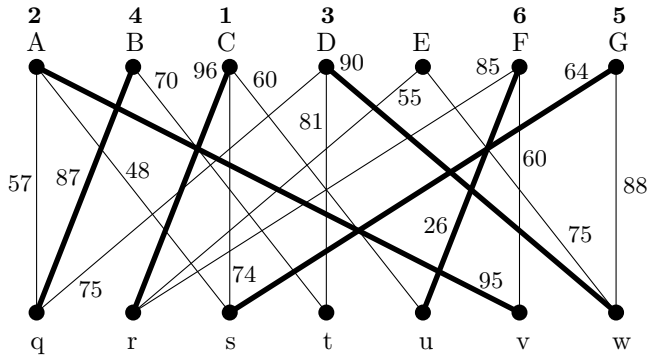
3.2 Párování maximální váhy

Firma při reorganizaci zrušila oddělení kreativního účetnictví se sedmi zaměstnanci, ale pružně pro ně vytvořila sedm nových míst. Personalista dostal na starost místa mezi sedm reorganizovaných co nejlépe rozdělit. Svědomitě s každým pohovořil, dal jim vyplnit složité testy a výsledky shr-

nul do skóre: pro každého vyhodnotil číslem od 0 do 100 jeho vhodnost pro každé z míst, které by byl ochoten vzít. Vše si zakreslil do schématu, v němž znalec pozná bipartitní graf:



Teď by chtěl každému vybrat pozici tak, aby součet všech skóre byl co největší. První, co člověka může napadnout, je dát každému místo, pro něž má největší skóre. To ale nejde, protože například řemeslo webmastera je nejsilnější stránkou Eleanor, Gudrun i Džaváharlála. Zkusíme-li přiřazovat pozice podle „hladového“ algoritmu, kdy v každém kroku přidáme hranu s největším skóre mezi těmi, které spojují dosud neobsazené pozice s ještě nevybranými pracovníky, podaří se přidělit jen 6 míst:



(číslice 1–6 nad vrcholy A–G znamenají pořadí výběru hran v hladovém algoritmu).

V řeči teorie grafů máme bipartitní graf s množinou vrcholů $V = A \cup B$ a množinou hran E. Každá hrana spojuje nějaký vrchol z A s některým vrcholem z B. Navíc platí $|A| = |B|$. Pro každou hranu $e \in E$ je daná nezáporná váha w_e. Chceme najít podmnožinu hran $M \subseteq E$ takovou, že do každého vrcholu z A i z B vchází přesně jedna hrana z M (taková M se v teorii grafů jmenuje **perfektní párování**), a přitom $\sum_{e \in M} w_e$ je co největší.

Abychom problém zapsali jako úlohu celočíselného programování, zavedeme proměnné x_e, jednu pro každou hranu $e \in E$, které smějí nabývat hodnot 0 nebo 1. Budeme jimi kódovat množinu M: $x_e = 1$ znamená $e \in M$ a $x_e = 0$ znamená $e \notin M$. Pak $\sum_{e \in M} w_e$ můžeme přepsat jako $\sum_{e \in E} w_e x_e$, a to bude účelová funkce. To, že do vrcholu $v \in V$ vchází přesně jedna hrana, se vyjádří podmínkou $\sum_{e \in E : v \in e} x_e = 1$. Výsledná úloha celočíselného programování je

$$\begin{array}{ll} \text{maximalizovat} & \sum_{e \in E} w_e x_e \\ \text{za podmínek} & \sum_{e \in E : v \in e} x_e = 1 \text{ pro každý vrchol } v \in V \\ & x_e \in \{0,1\} \text{ pro každou hranu } e \in E. \end{array} \tag{3.1}$$

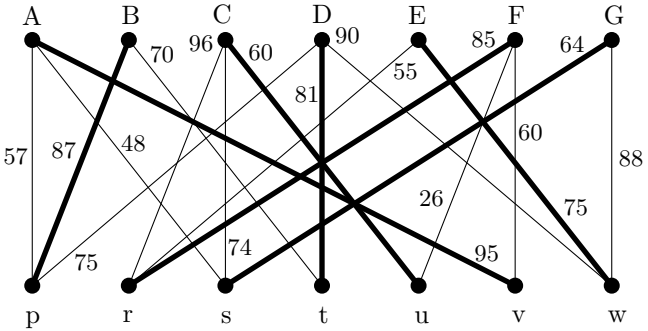
Vynecháme-li v této úloze podmínky celočíselnosti, tj. povolíme-li každému x_e nabývat všech hodnot v intervalu $[0,1]$, dostaneme následující úlohu lineárního programování:

$$\begin{array}{ll} \text{maximalizovat} & \sum_{e \in E} w_e x_e \\ \text{za podmínek} & \sum_{e \in E : v \in e} x_e = 1 \text{ pro každý vrchol } v \in V \\ & 0 \le x_e \le 1 \text{ pro každou hranu } e \in E. \end{array}$$

Té se říká **LP relaxace** úlohy (3.1) – uvolnili jsme, čili relaxovali, požadavky $x_e \in \{0,1\}$ na slabší požadavky $0 \le x_e \le 1$. LP relaxaci můžeme vyřešit třeba simplexovou metodou a získat nějaké optimální řešení $\mathbf{x}^*$.

K čemu může být takové $\mathbf{x}^*$ dobré? Určitě dá *horní odhad* na nejlepší možné řešení původní úlohy (3.1). To proto, že každé přípustné řešení (3.1) je také přípustným řešením LP relaxace, čili v LP relaxaci maximalizujeme přes větší množinu vektorů. Horní odhad může být velmi cenný: Například když se podaří najít nějaké přípustné řešení, pro které je hodnota účelové funkce 98% horního odhadu, můžeme se většinou přestat namáhat, protože víme, že si stejně nemůžeme polepšit o víc než 2% (záleží jistě na tom, čeho 2% – je-li to státní rozpočet, i 2% za trochu námahy ještě stojí).

V naší konkrétní úloze nás čeká příjemné překvapení: z LP relaxace získáme nejen horní odhad, ale přímo optimální řešení původní úlohy! Když totiž simplexovou metodou vyřešíme LP relaxaci, dostaneme optimum $\mathbf{x}^*$, které má všechny složky 0 nebo 1, a určuje tedy perfektní párování. To je nutně optimální (kdyby existovalo lepší perfektní párování, dávalo by lepší přípustné řešení i pro LP relaxaci). Takto nalezené optimální řešení uvažovaného konkrétního problému je na obrázku:



Tak to funguje nejen pro zvolený konkrétní příklad:

3.2.1 Věta. *Buď $G = (V, E)$ libovolný bipartitní graf s reálnými vahami w_e na hranách. Pokud má LP relaxace úlohy (3.1) aspoň jedno přípustné řešení, pak má vždy aspoň jedno celočíselné optimální řešení. To je optimálním řešením i pro úlohu (3.1).*

Pro zájemce větu dokážeme na konci tohoto oddílu.

Věta neříká, že *každé* optimální řešení LP relaxace je nutně celočíselné. Důkaz ale dává recept, jak z libovolného optimálního řešení vyrobit celočíselné. Navíc se ukazuje, že simplexová metoda vždycky vrátí celočíselné optimální řešení (v terminologii kapitoly 4, každé bázické přípustné řešení je celočíselné).

LP relaxaci můžeme uvažovat pro každou úlohu celočíselného programování (požadavek $\mathbf{x} \in \mathbb{Z}^n$ se nahradí $\mathbf{x} \in \mathbb{R}^n$). Případy, kdy z LP relaxace

rovnou dostaneme optimální řešení původní úlohy, jako ve větě 3.2.1, jsou spíš vzácné, ale LP relaxace může být užitečná i jindy.

Maximální hodnota účelové funkce pro LP relaxaci vždycky dává horní odhad na maximum celočíselné úlohy. Ten je někdy dost těsný, ale jindy může být velice špatný – viz sekci 3.4. Těsnost odhadů z LP relaxací se studovala pro mnoho úloh. Někdy se k LP relaxaci přidávají další omezení, která jsou splněna všemi celočíselnými řešeními, ale vylučují některá řešení neceločíselná, a tím se onen horní odhad zlepšuje (to je metoda odřezávajících rovin, *cutting planes*).

Neceločíselné optimální řešení LP relaxace se někdy dá převést na přibližně optimální řešení vhodným zaokrouhlením. Jednoduchý příklad uvidíme v oddílu 3.3.

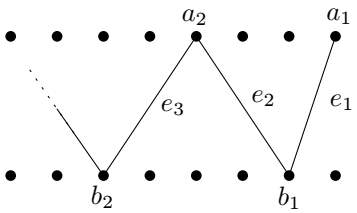
Důkaz věty 3.2.1. Buď $\mathbf{x}^*$ nějaké optimální řešení LP relaxace s hodnotou účelové funkce $w(\mathbf{x}^*) = \sum_{e \in E} w_e x_e^*$. Označme počet neceločíselných složek vektoru $\mathbf{x}^*$ symbolem $k(\mathbf{x}^*)$.

Jestliže $k(\mathbf{x}^*) = 0$, jsme hotovi. Pro $k(\mathbf{x}^*) > 0$ popíšeme postup, jak vyrobit jiné optimální řešení $\tilde{\mathbf{x}}$, pro které $k(\tilde{\mathbf{x}}) < k(\mathbf{x}^*)$. Po konečně mnoha opakováních dospějeme k celočíselnému optimálnímu řešení.

Buď $x^*_{e_1}$ neceločíselná složka vektoru $\mathbf{x}^*$, odpovídající nějaké hraně $e_1 = \{a_1, b_1\}$. Protože $0 < x^*_{e_1} < 1$ a

$$\sum_{e \in E : b_1 \in e} x_e^* = 1,$$

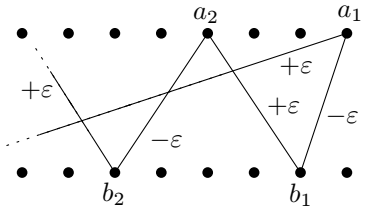
musí existovat jiná hrana $e_2 = \{a_2, b_1\}$, $a_2 \neq a_1$, s neceločíselným $x^*_{e_2}$. Z podobného důvodu najdeme i třetí hranu $e_3 = \{a_2, b_2\}$ s $0 < x^*_{e_3} < 1$. Tak pokračujeme a hledáme neceločíselné složky podél delší a delší cesty $(a_1, b_1, a_2, b_2, \ldots)$:



Protože graf G má konečně mnoho vrcholů, jednou dorazíme do vrcholu, kde jsme už byli. To znamená, že jsme našli *kružnici* C, pro jejíž všechny hrany platí $0 < x_e^* < 1$. Protože graf je bipartitní, délka t kružnice C je sudá. Pro jednoduchost značení předpokládejme, že hrany C jsou $e_1, e_2, \ldots, e_t$, i když ve skutečnosti nalezená kružnice nemusí začínat hned hranou e_1.

Nyní pro malé číslo ε definujeme

$$\tilde{x}_e = \begin{cases} x^*_e - \varepsilon & \text{pro } e \in \{e_1, e_3, \ldots, e_{t-1}\} \\ x^*_e + \varepsilon & \text{pro } e \in \{e_2, e_4, \ldots, e_t\} \\ x^*_e & \text{jinak.} \end{cases}$$



Je vidět, že takto definované $\tilde{\mathbf{x}}$ splňuje všechny podmínky

$$\sum_{e \in E : v \in e} \tilde{x}_e = 1,$$

protože ve vrcholech kružnice C jsme ε jednou přičetli a jednou odečetli, zatímco v ostatních vrcholech se hodnoty pro vcházející hrany nezměnily vůbec. Pro dostatečně malé ε budou splněny také podmínky $0 \leq \tilde{x}_e \leq 1$, protože všechny složky $x^*_{e_i}$ jsou ostře mezi 0 a 1. Takže pro malá ε je $\tilde{\mathbf{x}}$ opět přípustné řešení LP relaxace.

Co se stane s hodnotou účelové funkce? Máme

$$w(\tilde{\mathbf{x}}) = \sum_{e \in E} w_e \tilde{x}_e = w(\mathbf{x}^*) + \varepsilon \sum_{i=1}^{t} (-1)^i w_{e_i}.$$

Poněvadž $\mathbf{x}^*$ je optimální, musí platit $\Delta = \sum_{i=1}^{t} (-1)^i w_{e_i} = 0$, jinak bychom totiž mohli dosáhnout $w(\tilde{\mathbf{x}}) > w(\mathbf{x}^*)$, a to volbou $\varepsilon > 0$ (pro $\Delta > 0$) nebo volbou $\varepsilon < 0$ (pro $\Delta < 0$). To znamená, že $\tilde{\mathbf{x}}$ je optimální řešení pro všechna ta ε, pro něž je přípustné.

Zvolme nyní ε největší takové, že $\tilde{\mathbf{x}}$ je stále přípustným řešením. Potom musí pro některou hranu $e \in \{e_1, e_2, \ldots, e_t\}$ platit $\tilde{x}_e \in \{0, 1\}$, a $\tilde{\mathbf{x}}$ má méně neceločíselných složek než $\mathbf{x}^*$. □

3.3 Minimální vrcholové pokrytí

Ve Svobodné republice Západní Mordor se čile rozvinul Internet a vláda vydala nařízení, že pro vyšší bezpečnost obyvatelstva musí být každá propojovací linka vybavena speciálním zařízením pro sběr statistických dat

o jejím provozu. Provozovatel části sítě musí u každé linky aspoň jeden z počítačů na jejím konci vybavit vládní sledovací krabičkou. Které počítače má okrabičkovat, aby ho to vyšlo co nejlevněji? Řekněme, že za krabičky je jednotný poplatek.

Zase je pohodlné použít názvosloví teorie grafů: Počítače v síti budou vrcholy grafu a jejich propojení budou hrany. Máme tedy graf $G = (V, E)$ a chceme najít množinu vrcholů $S \subseteq V$ takovou, že každá hrana má aspoň jeden koncový vrchol v S (takovému S se říká **vrcholové pokrytí**) a přitom je S co nejmenší.

Tento problém se dá napsat jako úloha celočíselného programování takto:

$$\begin{array}{lll} \text{minimalizovat} & \sum_{v \in V} x_v & \\ \text{za podmínek} & x_u + x_v \geq 1 & \text{pro každou hranu } \{u, v\} \in E \\ & x_v \in \{0, 1\} & \text{pro všechna } v \in V. \end{array} \tag{3.2}$$

Pro každý vrchol v máme jednu proměnnou x_v, která může nabývat hodnot 0 nebo 1. Přitom $x_v = 1$ má význam $v \in S$ a $x_v = 0$ znamená $v \notin S$. Podmínka $x_u + x_v \geq 1$ zaručuje, že hrana $\{u, v\}$ má aspoň jeden vrchol v S. Účelová funkce je velikost S.

Ví se, že přesné řešení problému nejmenšího vrcholového pokrytí je algoritmicky obtížné (NP-úplné). Pomocí lineárního programování sestavíme algoritmus pro přibližné řešení, který vždy najde vrcholové pokrytí nejvýš dvakrát větší, než je nejlepší možné vrcholové pokrytí.

LP relaxace uvedené úlohy celočíselného programování je

$$\begin{array}{lll} \text{minimalizovat} & \sum_{v \in V} x_v & \\ \text{za podmínek} & x_u + x_v \geq 1 & \text{pro každou hranu } \{u, v\} \in E \\ & 0 \leq x_v \leq 1 & \text{pro všechna } v \in V. \end{array} \tag{3.3}$$

První část přibližného algoritmu je vypočítat nějaké optimální řešení $\mathbf{x}^*$ této LP relaxace (nějakým standardním algoritmem pro lineární programování). Složky $\mathbf{x}^*$ jsou reálná čísla mezi 0 a 1. Ve druhém kroku se definuje množina

$$S_{\text{LP}} = \{v \in V : x_v^* \geq \tfrac{1}{2}\}.$$

To je vrcholové pokrytí, protože pro každou hranu $\{u, v\}$ máme $x_u^* + x_v^* \geq 1$, a tedy $x_u^* \geq \frac{1}{2}$ nebo $x_v^* \geq \frac{1}{2}$.

Nechť S_{OPT} je nějaké vrcholové pokrytí nejmenší možné velikosti (to ve skutečnosti neznáme, ale můžeme o něm teoreticky uvažovat). Tvrdíme, že

$$|S_{\text{LP}}| \leq 2 \cdot |S_{\text{OPT}}|.$$

Buď $\bar{\mathbf{x}}$ optimální řešení úlohy celočíselného programování (3.2), které odpovídá množině S_{OPT}, tj. $\bar{x}_v = 1$ pro $v \in S_{\text{OPT}}$ a $\bar{x}_v = 0$ jinak. Toto

$\bar{\mathbf{x}}$ je určitě přípustným řešením LP relaxace (3.3), a tedy nemůže mít lepší hodnotu účelové funkce než řešení optimální:

$$\sum_{v \in V} x_v^* \leq \sum_{v \in V} \bar{x}_v.$$

Na druhé straně $|S_{\mathrm{LP}}| \leq 2 \cdot \sum_{v \in V} x_v^*$, protože $x_v^* \geq \frac{1}{2}$ pro každé $v \in S_{\mathrm{LP}}$. Takže

$$|S_{\mathrm{LP}}| \leq 2 \cdot \sum_{v \in V} x_v^* \leq 2 \cdot \sum_{v \in V} \bar{x}_v = 2 \cdot |S_{\mathrm{OPT}}|.$$

Právě uvedený důkaz ilustruje důležitý aspekt aproximačních algoritmů: Abychom mohli posoudit kvalitu vypočteného řešení, musíme vždy mít nějaký odhad na kvalitu řešení optimálního, ačkoli optimální řešení samo neznáme. LP relaxace takový odhad poskytuje. Někdy je užitečný, jak jsme viděli v této sekci, a někdy, pro jiné výpočetní úlohy, může být k ničemu – to uvidíme v sekci příští.

Poznámky. Přirozený pokus o přibližné řešení diskutované úlohy je *hladový algoritmus*: vybírej vrcholy pokrytí postupně, a vždycky vezmi takový vrchol, který pokryje nejvíc dosud nepokrytých hran. Ale pro tento algoritmus se dají zkonstruovat příklady, kdy dá řešení třeba desetkrát horší, než je optimální (a 10 můžeme nahradit libovolnou jinou konstantou). Čtenář se o takovou konstrukci může pokusit sám, je to docela pěkný oříšek.

Pro minimální vrcholové pokrytí je znám také kombinatorický přibližný algoritmus. V daném grafu najdeme nějaké maximální párování M (maximální ve smyslu inkluze, tj. k M nejde přidat žádná další hrana), a koncové vrcholy všech hran z M použijeme jako vrcholové pokrytí. Toto vrcholové pokrytí je nanejvýš dvakrát větší než pokrytí optimální, podobně jako u algoritmu popsaného výše. Ale algoritmus založený na lineárním programování má tu výhodu, že se snadno zobecní na **vážené vrcholové pokrytí** (krabičky k různým počítačům můžou mít různé ceny). Stejně jako pro ceny jednotkové je vypočtené řešení vždy nejvýš dvakrát dražší než optimální, s prakticky stejným důkazem.

3.4 Největší nezávislá množina

Tady už autora přestalo bavit vymýšlet pseudoreálné problémy ze života, a proto další problém řekneme rovnou v jazyce teorie grafů. Pro graf $G = (V, E)$ se množina vrcholů $I \subseteq V$ nazývá **nezávislá**, pokud žádné dva vrcholy z I nejsou v G spojeny hranou.

Výpočet nezávislé množiny s největším možným počtem vrcholů pro daný graf je jedním z notoricky obtížných grafových problémů. Dá se snadno

formulovat jako úloha celočíselného programování:

$$\begin{array}{lll} \text{maximalizovat} & \sum_{v \in V} x_v & \\ \text{za podmínek} & x_u + x_v \leq 1 & \text{pro každou hranu } \{u, v\} \in E \\ & x_v \in \{0, 1\} & \text{pro všechna } v \in V. \end{array} \tag{3.4}$$

Optimální řešení $\mathbf{x}^*$ odpovídá nezávislé množině I maximální velikosti: $v \in I$, právě když $x_v^* = 1$. Omezující podmínky $x_u + x_v \leq 1$ zajišťují, že kdykoli jsou vrcholy u a v spojeny hranou, do I se může dostat nejvýš jeden z nich.

V LP relaxaci se nahradí podmínka $x_v \in \{0, 1\}$ nerovnicemi $0 \leq x_v \leq 1$. Výsledná úloha lineárního programování má vždycky přípustné řešení, v němž jsou všechna $x_v = \frac{1}{2}$, což dává účelovou funkci rovnou $|V|/2$, a tedy optimální hodnota účelové funkce je aspoň $|V|/2$. Naproti tomu třeba úplný graf (kde každé dva vrcholy jsou spojeny hranou) má největší nezávislou množinu velikosti jen 1. V tomto případě, a to je pointa této sekce, se LP relaxace chová naprosto jinak než původní úloha celočíselného programování.

Úplný graf přitom není ojedinělý příklad, grafy s hodně hranami většinou mívají největší nezávislou množinu mnohem menší, než je polovina počtu vrcholů, a ani pro ně nám tedy optimální řešení LP relaxace mnoho neřekne.

Pro problém největší nezávislé množiny se dokonce ví, že obecně nejde dobře aproximovat vůbec žádným efektivním algoritmem.

4

Teorie lineárního programování: první kroky

4.1 Rovnicový tvar

V úvodní kapitole jsme řekli, jak převést každou úlohu lineárního programování na tvar

$$\begin{array}{ll} \text{maximalizovat} & \mathbf{c}^T\mathbf{x} \\ \text{za podmínek} & A\mathbf{x} \leq \mathbf{b}. \end{array}$$

Simplexová metoda ale vyžaduje tvar jiný. V literatuře se pro něj používá název *standardní tvar*. V tomto textu zavedu méně obvyklý, ale určitější termín *rovnicový tvar*. Vypadá takhle:

Rovnicový tvar úlohy lineárního programování:

$$\begin{array}{ll} \text{maximalizovat} & \mathbf{c}^T\mathbf{x} \\ \text{za podmínek} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{array}$$

Jako obvykle, $\mathbf{x}$ je vektor proměnných, A je daná matice typu $m \times n$ a $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ jsou dané vektory. Omezení tedy jsou jednak rovnice, a jednak nerovnice velice speciálního tvaru $x_j \geq 0$, takzvané **podmínky nezápornosti**. (Takže pozor: ačkoliv tomuto tvaru říkáme rovnicový, jsou v něm nerovnice také, a nesmí se na ně zapomenout!)

Zdůrazněme, že podmínky nezápornosti musí v rovnicovém tvaru splňovat *všechny* proměnné, jež se v úloze vyskytují. V úlohách z praxe vznikají podmínky nezápornosti často automaticky, protože mnohé veličiny, jako třeba množství snědených okurek, záporné být nemohou.

Převod obecné úlohy na rovnicový tvar ukážeme na úloze

$$\begin{array}{ll} \text{maximalizovat} & 3x_1 - 2x_2 \\ \text{za podmínek} & 2x_1 - x_2 \leq 4 \\ & x_1 + 3x_2 \geq 5 \\ & x_2 \geq 0. \end{array}$$

Postupujeme takto:

1. Abychom z nerovnice $2x_1 - x_2 \leq 4$ vyrobili rovnici, zavedeme novou proměnnou x_3, spolu s podmínkou nezápornosti $x_3 \geq 0$, a zmíněnou nerovnici nahradíme rovnicí $2x_1 - x_2 + x_3 = 4$. Pomocná proměnná x_3, jež se nikde jinde v úloze nebude vyskytovat, tedy reprezentuje rozdíl mezi pravou a levou stranou nerovnice.

2. U další nerovnice $x_1 + 3x_2 \geq 5$ napřed změnou znaménka obrátíme směr nerovnosti. Pak použijeme další pomocnou proměnnou x_4, $x_4 \geq 0$, a uvedenou nerovnici nahradíme rovnicí $-x_1 - 3x_2 + x_4 = -5$.

3. Ještě nejsme hotovi: proměnná x_1 v původní úloze má povoleno nabývat i záporných hodnot. Zavedeme dvě nové, již nezáporné proměnné y_1 a z_1, $y_1 \geq 0$, $z_1 \geq 0$, a za x_1 všude v úloze dosadíme $y_1 - z_1$. Samo x_1 tím z úlohy vymizí.

Rovnicový tvar naší úlohy tedy je

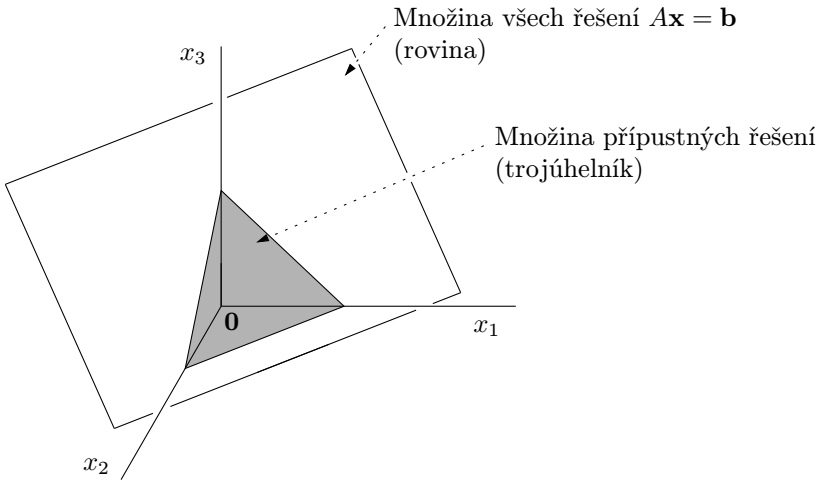
$$\begin{array}{ll} \text{maximalizovat} & 3y_1 - 3z_1 - 2x_2 \\ \text{za podmínek} & 2y_1 - 2z_1 - x_2 + x_3 = 4 \\ & -y_1 + z_1 - 3x_2 + x_4 = -5 \\ & y_1 \geq 0,\ z_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0. \end{array}$$

Abychom úplně vyhověli konvencím rovnicového tvaru, měli bychom ještě přejmenovat proměnné na $x_1, x_2, \ldots, x_5$.

Obecnou úlohu s n proměnnými a m omezujícími podmínkami převedeme právě popsaným způsobem na úlohu v rovnicovém tvaru s nejvýš $m + 2n$ proměnnými a m rovnicemi (a samozřejmě podmínkami nezápornosti pro všechny proměnné).

Geometrie úlohy v rovnicovém tvaru. Z lineární algebry se ví, že množina všech řešení soustavy $A\mathbf{x} = \mathbf{b}$ je afinní podprostor F prostoru $\mathbb{R}^n$ (tj. vektorový podprostor posunutý o nějaký vektor $\mathbf{x}_0$). Geometricky je tedy množina všech přípustných řešení úlohy v rovnicovém tvaru průnikem F s tzv. nezáporným ortantem[1], tj. množinou bodů v $\mathbb{R}^n$ se všemi souřadnicemi nezápornými. Následující obrázek to ilustruje pro $n = 3$ proměnné a $m = 1$ rovnici (konkrétně rovnici $x_1 + x_2 + x_3 = 1$):

[1] V rovině je to *kvadrant*, v $\mathbb{R}^3$ *oktant*, a pro obecnou dimenzi se říká *ortant*.



(V zajímavějších případech máme víc proměnných než 3 a obrázek nakreslit nejde.) Změníme-li soustavu $A\mathbf{x} = \mathbf{b}$ nějakou ekvivalentní úpravou, která zachovává její množinu řešení, neovlivní to přípustná ani optimální řešení příslušné úlohy lineárního programování. Toho se bude hojně využívat v simplexové metodě.

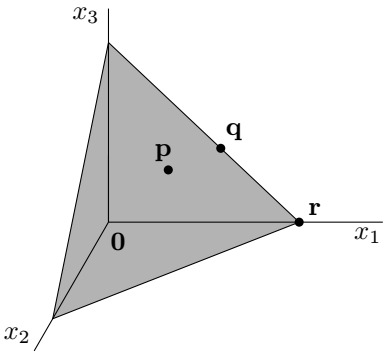
V dalším budeme vždy předpokládat, že pro úlohu v rovnicovém tvaru má soustava rovnic $A\mathbf{x} = \mathbf{b}$ aspoň jedno řešení, a *řádky matice A jsou lineárně nezávislé.*

Jako vysvětlení tohoto předpokladu připomeneme několik věcí z lineární algebry. Zkontrolovat, jestli má soustava $A\mathbf{x} = \mathbf{b}$ aspoň jedno řešení, můžeme třeba Gaussovou eliminací, a pokud řešení nemá, nemusíme se takovou úlohou lineárního programování dál zabývat. (Pozor, to, že $A\mathbf{x} = \mathbf{b}$ má řešení, ještě neznamená, že příslušná úloha lineárního programování v rovnicovém tvaru má přípustné řešení, protože přípustné řešení musí řešit $A\mathbf{x} = \mathbf{b}$ a *navíc* mít všechny složky nezáporné.)

Má-li soustava $A\mathbf{x} = \mathbf{b}$ aspoň jedno řešení a je-li nějaký řádek matice A lineární kombinací ostatních, pak je příslušná rovnice nadbytečná a můžeme ji ze soustavy vynechat, aniž se množina řešení změní. (Nebo obecněji, můžeme najít libovolnou jinou soustavu rovnic se stejnou množinou řešení a s lineárně nezávislými řádky, třeba převodem rozšířené matice soustavy na odstupňovaný tvar.) Takže můžeme předpokládat, že matice A má m lineárně nezávislých řádků a její hodnost je m. Platí $m \leq n$, protože jinak by bylo méně sloupců než řádků a hodnost by nemohla být m.

4.2 Bázická přípustná řešení

Mezi přípustnými řešeními každé úlohy lineárního programování mají výsadní postavení takzvaná bázická přípustná řešení. Zde je budeme zatím uvažovat jen pro úlohy v rovnicovém tvaru. Podívejme se znovu na obrázek množiny přípustných řešení pro úlohu s $n = 3$, $m = 1$:



Z přípustných řešení **p**, **q** a **r** je bázickým přípustným řešením jedině **r**. Geometricky a velmi neformálně vyjádřeno, bázické přípustné řešení je roh (špička) množiny všech přípustných řešení. To přesně zformulujeme později (viz větu 4.4.1).

Definice, již brzy podáme zde, je jiná. Požaduje, velmi zhruba řečeno, aby bázické přípustné řešení mělo dostatečně mnoho nulových složek. Než se k ní dopracujeme, zavedeme nové značení. V tomto oddílu bude A vždy matice s m řádky, $n \geq m$ sloupci a hodností m. Pro podmnožinu $B \subseteq \{1, 2, \ldots, n\}$ značí A_B matici, sestávající z těch sloupců matice A, jejichž indexy jsou v B. Například pro

$$A = \begin{pmatrix} 1 & 5 & 3 & 4 & 6 \\ 0 & 1 & 3 & 5 & 6 \end{pmatrix} \text{ a } B = \{2, 4\} \text{ máme } A_B = \begin{pmatrix} 5 & 4 \\ 1 & 5 \end{pmatrix}.$$

Podobného značení budeme používat pro vektory: například pro $\mathbf{x} = (3, 5, 7, 9, 11)$ a $B = \{2, 4\}$ je $\mathbf{x}_B = (5, 9)$.

Nyní můžeme vyslovit formální definici.

Vektor $\mathbf{x} \in \mathbb{R}^n$ je **bázické přípustné řešení** úlohy

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} = \mathbf{b} \text{ a } \mathbf{x} \geq \mathbf{0},$$

pokud $\mathbf{x}$ je přípustné řešení a existuje m-prvková množina $B \subseteq \{1, 2, \ldots, n\}$ taková, že

- (čtvercová) matice A_B je regulární, tj. sloupce indexované B jsou lineárně nezávislé a
- $x_j = 0$ kdykoli $j \notin B$.

Například $\mathbf{x} = (0, 2, 0, 1, 0)$ je bázické přípustné řešení pro

$$A = \begin{pmatrix} 1 & 5 & 3 & 4 & 6 \\ 0 & 1 & 3 & 5 & 6 \end{pmatrix}, \ \mathbf{b} = (14, 7)$$

s $B = \{2, 4\}$.

Je-li taková B pevně zvolena, proměnným x_j s $j \in B$ se říká **bázické proměnné**, zatímco ostatní proměnné jsou **nebázické**. Krátce tedy můžeme říci, že v bázickém přípustném řešení jsou všechny nebázické proměnné nulové.

Všimněme si, že v definici se vůbec nepracuje s vektorem $\mathbf{c}$ a že bázická přípustná řešení závisejí jen na A a na $\mathbf{b}$.

Pro některé úvahy je pohodlné definici bázického přípustného řešení trochu přeformulovat.

4.2.1 Lemma. *Přípustné řešení* $\mathbf{x}$ *úlohy v rovnicovém tvaru je bázické, právě když sloupce matice* A_K *jsou lineárně nezávislé, kde* $K = \{j \in \{1, 2, \ldots, n\} : x_j > 0\}$.

Důkaz. Jedna implikace je zřejmá: je-li $\mathbf{x}$ bázické přípustné řešení a B je příslušná m-prvková množina jako v definici, pak $K \subseteq B$ a sloupce matice A_K jsou lineárně nezávislé.

Obráceně, nechť $\mathbf{x}$ je takové, že sloupce matice A_K jsou lineárně nezávislé. Jestliže $|K| = m$, můžeme v definici bázického přípustného řešení prostě vzít $B = K$. Jinak doplníme K na m-prvkovou množinu B přidáním dalších $m - |K|$ indexů tak, aby sloupce matice A_B byly lineárně nezávislé. To vždycky jde, například podle Steinitzovy věty z lineární algebry (v našem případě potřebujeme lineárně nezávislou množinu sloupců A_K doplnit dalšími sloupci matice A na bázi sloupcového vektorového prostoru). □

4.2.2 Tvrzení. *Bázické přípustné řešení je jednoznačně určeno množinou B. To znamená, že pro každou m-prvkovou množinu $B \subseteq \{1, 2, \ldots, n\}$, pro niž je A_B regulární, existuje nejvýš jedno přípustné řešení $\mathbf{x} \in \mathbb{R}^n$, ve kterém $x_j = 0$ pro všechna $j \notin B$.*

Radši hned zdůrazněme, že jednomu bázickému přípustnému řešení může odpovídat i mnoho různých množin B.

Důkaz tvrzení 4.2.2. Aby $\mathbf{x}$ bylo přípustné, musí platit $A\mathbf{x} = \mathbf{b}$, a levou stranu můžeme rozepsat $A\mathbf{x} = A_B\mathbf{x}_B + A_N\mathbf{x}_N$, kde $N = \{1, 2, \ldots, n\} \setminus B$. Podle definice bázického přípustného řešení je vektor $\mathbf{x}_N$ nebázických proměnných roven $\mathbf{0}$, takže vektor $\mathbf{x}_B$ bázických proměnných splňuje $A_B\mathbf{x}_B = \mathbf{b}$. A tady se využije podmínky, že A_B je regulární: soustava $A_B\mathbf{x}_B = \mathbf{b}$ má právě jedno řešení $\tilde{\mathbf{x}}_B$. Jestliže $\tilde{\mathbf{x}}_B$ má všechny složky nezáporné, pak máme pro uvažované B přesně jedno bázické přípustné řešení (doplníme $\tilde{\mathbf{x}}_B$ nulami), a jinak žádné. □

Zavedeme takovéto názvosloví: m-prvkové množině $B \subseteq \{1, 2, \ldots, n\}$, pro kterou je matice A_B regulární, budeme říkat **báze**[2]. Jestliže navíc B určuje bázické přípustné řešení, tj. jestliže (jediné) řešení soustavy $A_B\mathbf{x}_B = \mathbf{b}$ je nezáporné, budeme B označovat za **přípustnou bázi**.

Následující základní věta pojednává o existenci optimálního řešení, a navíc ukazuje, že při jeho hledání se můžeme omezit na bázická přípustná řešení.

4.2.3 Věta. *Uvažme úlohu lineárního programování v rovnicovém tvaru*

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} = \mathbf{b},\ x \geq \mathbf{0}.$$

(i) *Pokud existuje aspoň jedno přípustné řešení, a přitom účelová funkce je na množině všech přípustných řešení shora omezená, potom existuje optimální řešení.*

(ii) *Existuje-li optimální řešení, potom i některé z bázických přípustných řešení je optimální.*

Důkaz není pro další čtení nezbytný a odložíme jej na konec tohoto oddílu. Věta plyne také ze správnosti simplexové metody. Tou se budeme zabývat v další kapitole, ale celý její důkaz nedoděláme.

Z věty vyplývá konečný, i když zcela nepraktický algoritmus pro řešení úlohy lineárního programování v rovnicovém tvaru. Stačí probrat všechny

[2]To je zkrácené vyjadřování. Samotná indexová množina B samozřejmě není báze ve smyslu lineární algebry, nýbrž množina sloupců matice A_B je báze sloupcového prostoru matice A.

m-prvkové podmnožiny $B \subseteq \{1, 2, \ldots, n\}$, pro každou zjistit, jestli je to přípustná báze, a spočítat maximum účelové funkce přes všechna takto nalezená bázická přípustná řešení (podle tvrzení 4.2.2 dostaneme pro každou B nejvýš jedno).

Přísně vzato, uvedený algoritmus nefunguje, pokud je úloha neomezená. Vymyšlení modifikace, která pracuje správně i v tomto případě, tj. oznámí, že úloha je neomezená, ponecháváme jako cvičení. Brzy budeme probírat mnohem efektivnější simplexovou metodu, a ukážeme, jak tu věc ošetřit tam.

V uvedeném algoritmu musíme procházet $\binom{n}{m}$ množin B (kde $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ je binomický koeficient). Například pro $n = 2m$ roste funkce $\binom{2m}{m}$ zhruba jako 4^m, čili exponenciálně, a to je příliš mnoho už pro nevelká m.

Simplexová metoda také prochází bázická přípustná řešení, ale chytřeji. Chodí od jednoho ke druhému a přitom stále zlepšuje hodnotu účelové funkce, až najde optimum.

Ještě shrňme nejdůležitější poznatky z tohoto oddílu.

Úloha lineárního programování *v rovnicovém tvaru* má konečně mnoho bázických přípustných řešení, a je-li přípustná a omezená, potom aspoň jedno z bázických přípustných řešení je optimální.

Z toho plyne, že libovolná přípustná a omezená úloha lineárního programování má aspoň jedno optimální řešení.

Důkaz věty 4.2.3. Použijeme některé obraty, které se budou v trochu složitějším provedení opakovat v simplexové metodě, takže tento důkaz je jakási průprava.

Poznamenejme ještě, že část (i) se také dá dokázat z obecného výsledku matematické analýzy (spojitá funkce na kompaktní množině nabývá maxima). Zde se bez něj ale obejdeme.

Dokážeme následující tvrzení:

Je-li účelová funkce úlohy v rovnicovém tvaru shora omezená, potom pro každé přípustné řešení $\mathbf{x}_0$ *existuje bázické přípustné řešení* $\tilde{\mathbf{x}}$ *se stejnou nebo větší hodnotou účelové funkce,* $\mathbf{c}^T\tilde{\mathbf{x}} \geq \mathbf{c}^T\mathbf{x}_0$.

Jak z toho plyne věta? Je-li úloha přípustná a omezená, existuje podle tvrzení pro každé přípustné řešení nějaké bázické přípustné řešení se stejnou nebo větší hodnotou účelové funkce. Protože bázických přípustných řešení je jen konečně mnoho, musí některé z nich dávat největší hodnotu, neboli být optimální. Tak dostáváme naráz části (i) a (ii).

Abychom dokázali tvrzení, uvažme nějaké přípustné řešení $\mathbf{x}_0$. Mezi všemi přípustnými řešeními $\mathbf{x}$ splňujícími $\mathbf{c}^T\mathbf{x} \geq \mathbf{c}^T\mathbf{x}_0$ zvolíme nějaké takové, jež má co nejvíc nulových složek, a nazveme ho $\tilde{\mathbf{x}}$. Definujeme indexovou množinu

$$K = \{j \in \{1, 2, \ldots, n\} : \tilde{x}_j > 0\}.$$

Má-li matice A_K lineárně nezávislé sloupce, je $\tilde{\mathbf{x}}$ bázické přípustné řešení, viz lemma 4.2.1, a jsme hotovi.

Nechť jsou tedy sloupce A_K lineárně závislé, to znamená, že pro nějaký nenulový $|K|$-složkový vektor $\mathbf{v}$ platí $A_K\mathbf{v} = \mathbf{0}$. Doplníme $\mathbf{v}$ nulami v pozicích mimo K na n-složkový vektor $\mathbf{w}$ (tedy $\mathbf{w}_K = \mathbf{v}$ a $A\mathbf{w} = A_K\mathbf{v} = \mathbf{0}$).

Chvíli předpokládejme, že $\mathbf{w}$ splňuje následující dvě podmínky:

(a) $\mathbf{c}^T\mathbf{w} \geq 0$.

(b) Existuje $j \in K$, pro něž $w_j < 0$.

Pro reálné číslo $t \geq 0$ se podívejme na vektor $\mathbf{x}(t) = \tilde{\mathbf{x}} + t\mathbf{w}$. Ukážeme, že pro vhodné $t_1 > 0$ je $\mathbf{x}(t_1)$ přípustné a má více nulových složek než $\tilde{\mathbf{x}}$. Přitom $\mathbf{c}^T\mathbf{x}(t_1) = \mathbf{c}^T\tilde{\mathbf{x}} + t_1\mathbf{c}^T\mathbf{w} \geq \mathbf{c}^T\mathbf{x}_0 + t_1\mathbf{c}^T\mathbf{w} \geq \mathbf{c}^T\mathbf{x}_0$, takže dostaneme spor s tím, že $\tilde{\mathbf{x}}$ mělo mít nejvíc nulových složek.

Pro všechna t platí $A\mathbf{x}(t) = \mathbf{b}$, poněvadž $A\mathbf{x}(t) = A\tilde{\mathbf{x}} + tA\mathbf{w} = A\tilde{\mathbf{x}} = \mathbf{b}$, neboť $\tilde{\mathbf{x}}$ je přípustné. Dále pro $t = 0$ má $\mathbf{x}(0) = \tilde{\mathbf{x}}$ všechny složky z K ostře kladné a všechny ostatní složky nulové. Pro j-tou složku $\mathbf{x}(t)$ máme $x(t)_j = \tilde{x}_j + tw_j$, a je-li $w_j < 0$ jako v podmínce (b), pro dost velké t platí $x(t)_j < 0$. Jestliže začneme s $t = 0$ a necháme t růst, potom ta $x(t)_j$, pro něž $w_j < 0$, se budou zmenšovat, a v jistém okamžiku t_1 první z nich dosáhne hodnoty 0. V této chvíli $\mathbf{x}(t_1)$ má pořád ještě všechny složky nezáporné, čili je přípustné, ale má oproti $\tilde{\mathbf{x}}$ přinejmenším jednu nulovou složku navíc. Z toho, jak už jsme řekli, plyne spor.

Co když ale vektor $\mathbf{w}$ podmínku (a) nebo (b) nesplňuje? Pokud $\mathbf{c}^T\mathbf{w} = 0$, (a) platí a (b) pak můžeme vždycky zachránit případnou změnou znaménka $\mathbf{w}$ (poněvadž $\mathbf{w} \neq \mathbf{0}$). Takže předpokládáme $\mathbf{c}^T\mathbf{w} \neq 0$, a zase případnou změnou znaménka docílíme $\mathbf{c}^T\mathbf{w} > 0$. Neplatí-li teď (b), musí být $\mathbf{w} \geq \mathbf{0}$. To ale znamená, že $\mathbf{x}(t) = \tilde{\mathbf{x}} + t\mathbf{w} \geq \mathbf{0}$ pro všechna $t \geq 0$, a tudíž všechna taková $\mathbf{x}(t)$ jsou přípustná. A hodnota účelové funkce $\mathbf{c}^T\mathbf{x}(t) = \mathbf{c}^T\tilde{\mathbf{x}} + t\mathbf{c}^T\mathbf{w}$ roste nade všechny meze pro $t \to \infty$, takže úloha je neomezená. Tím je důkaz hotov. □

4.3 ABC konvexity a konvexních mnohostěnů

Konvexita je jeden z nejdůležitějších pojmů matematiky a v teorii lineárního programování se s ní člověk setká velmi přirozeně. Tady připomeneme

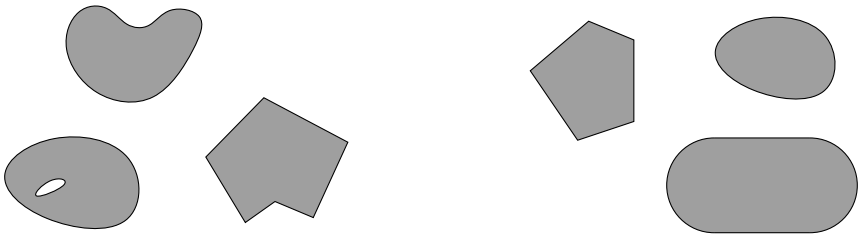
definici a uvedeme několik nejzákladnějších pojmů a výsledků, které přinejmenším pomohou získat o lineárním programování lepší intuitivní představu.

Lineární programování se ovšem dá vykládat i bez nich a v nahuštěnějších kursech na ně není čas. Proto tato a následující sekce jsou pojaty jako rozšiřující materiál, a zbytek textu by měl být srozumitelný i bez nich.

Množina $X \subseteq \mathbb{R}^n$ je **konvexní**, pokud pro každé dva body $\mathbf{x}, \mathbf{y} \in X$ obsahuje X celou úsečku $\mathbf{xy}$. Jinými slovy, pro každé $\mathbf{x}, \mathbf{y} \in X$ a každé $t \in [0,1]$ platí $t\mathbf{x} + (1-t)\mathbf{y} \in X$.

Na vysvětlenou: $t\mathbf{x} + (1-t)\mathbf{y}$ je bod na úsečce $\mathbf{xy}$ ve vzdálenosti t od $\mathbf{y}$ a $1-t$ od $\mathbf{x}$, bereme-li jako jednotku vzdálenosti délku té úsečky.

Tady je několik příkladů konvexních a nekonvexních množin v rovině:



nekonvexní konvexní

Konvexní množina dole na tomto obrázku, stadion, stojí za zapamatování, protože často je protipříkladem na tvrzení o konvexních množinách, která na první pohled mohou vypadat zjevně pravdivá.

V matematické analýze se pracuje hlavně s konvexními *funkcemi*. Oba pojmy spolu úzce souvisí: například reálná funkce $f\colon \mathbb{R} \to \mathbb{R}$ je konvexní, právě když její nadgraf, tj. množina $\{(x,y) \in \mathbb{R}^2 : y \geq f(x)\}$, je konvexní množina v rovině.

Konvexní obal a konvexní kombinace. Je snadno vidět, že průnik libovolného souboru konvexních množin je zase konvexní množina. To umožňuje zavést pojem konvexního obalu.

Buď $X \subset \mathbb{R}^n$ libovolná množina. Její **konvexní obal** definujeme jako průnik všech konvexních množin obsahujících X. Je to tedy nejmenší konvexní množina obsahující X, v tom smyslu, že každá konvexní množina obsahující X obsahuje také konvexní obal X.

Tato definice není příliš konstruktivní. Konvexní obal se dá popsat také pomocí konvexních kombinací, podobně jako lineární obal množiny vektorů se dá popsat pomocí lineárních kombinací. Jsou-li $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ nějaké body v $\mathbb{R}^n$, jejich **konvexní kombinace** je každý bod tvaru

$$t_1\mathbf{x}_1 + t_2\mathbf{x}_2 + \cdots + t_m\mathbf{x}_m, \text{ kde } t_1, t_2, \ldots, t_m \geq 0 \text{ a } \sum_{i=1}^{m} t_i = 1.$$

Konvexní kombinace je tedy speciální typ lineární kombinace, v níž jsou koeficienty nezáporné a sečtou se na 1.

Konvexní kombinace dvou bodů $\mathbf{x}$ a $\mathbf{y}$ jsou tvaru $t\mathbf{x}+(1-t)\mathbf{y}$, $t \in [0,1]$, a jak jsme řekli v souvislosti s definicí konvexní množiny, vyplní přesně úsečku $\mathbf{xy}$. Je snadné, ale poučné si rozmyslet, že všechny konvexní kombinace tří bodů $\mathbf{x}, \mathbf{y}, \mathbf{z}$ vyplní přesně trojúhelník $\mathbf{xyz}$ (pokud body neleží na přímce).

4.3.1 Lemma. *Konvexní obal C množiny $X \subseteq \mathbb{R}^n$ je roven množině*

$$\tilde{C} = \left\{ \sum_{i=1}^{m} t_i\mathbf{x}_i : m \geq 1, \mathbf{x}_1, \ldots, \mathbf{x}_m \in X, t_1, \ldots, t_m \geq 0, \sum_{i=1}^{m} t_i = 1 \right\}$$

všech konvexních kombinací konečně mnoha bodů z X.

Důkaz. Nejdřív dokážeme indukcí podle m, že každá konvexní kombinace musí ležet v C. Pro $m = 1$ je to zřejmé a pro $m = 2$ to plyne přímo z definice konvexity. Buď $m \geq 3$ a nechť $\mathbf{x} = t_1\mathbf{x}_1 + \cdots + t_m\mathbf{x}_m$ je konvexní kombinace bodů z X. Pro $t_m = 1$ máme $\mathbf{x} = \mathbf{x}_m \in C$. Pro $t_m < 1$ položme $t'_i = t_i/(1-t_m)$, $i = 1, 2, \ldots, m-1$. Pak $\mathbf{x}' = t'_1\mathbf{x}_1 + \cdots + t'_{m-1}\mathbf{x}_{m-1}$ je konvexní kombinace bodů $\mathbf{x}_1, \ldots, \mathbf{x}_{m-1}$ (ta t'_i se sečtou na 1) a podle indukčního předpokladu $\mathbf{x}' \in C$. Takže $\mathbf{x} = (1-t_m)\mathbf{x}' + t_m\mathbf{x}_m$ je konvexní kombinací dvou bodů z (konvexní) množiny C a také leží v C.

Tím jsme dokázali $\tilde{C} \subseteq C$. Pro opačnou inkluzi stačí ukázat, že $\tilde{C}$ je konvexní, tj. ověřit, že jsou-li $\mathbf{x}, \mathbf{y} \in \tilde{C}$ dvě konvexní kombinace a $t \in (0,1)$, pak $t\mathbf{x} + (1-t)\mathbf{y}$ je zase konvexní kombinace. To je zcela přímočaré a dovolíme si to vynechat. □

Konvexní množiny, s nimiž se člověk setká v teorii lineárního programování, jsou speciálního typu a jmenují se konvexní mnohostěny.

Nadroviny, poloprostory, mnohostěny. Připomeňme, že **nadrovina** v prostoru $\mathbb{R}^n$ je afinní podprostor dimenze $n-1$. Jinak řečeno, je to množina všech řešení jedné lineární rovnice tvaru

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b,$$

kde $a_1, a_2, \ldots, a_n$ nejsou všechna zároveň rovna 0. V rovině jsou nadrovinami přímky a v $\mathbb{R}^3$ jsou to roviny.

Taková nadrovina rozděluje $\mathbb{R}^n$ na dva poloprostory a je jejich společnou hranicí. Ony dva poloprostory mají analytické vyjádření

$$\left\{\mathbf{x} \in \mathbb{R}^n : a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b\right\}$$

a

$$\left\{\mathbf{x} \in \mathbb{R}^n : a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b\right\}.$$

Přesněji řečeno, jsou to **uzavřené poloprostory**, do nichž patří i jejich hranice.

Průnik konečně mnoha uzavřených poloprostorů v $\mathbb{R}^n$ se nazývá **konvexní mnohostěn**.

Poloprostor je zjevně konvexní, tedy průnik poloprostorů je také konvexní a konvexní mnohostěny se jmenují konvexní právem.

Kruh v rovině je konvexní množina, ale není to konvexní mnohostěn (protože, zhruba řečeno, konvexní mnohostěn musí být „hranatý“... ale zkuste to dokázat pořádně).

Poloprostor je množina všech řešení jedné lineární nerovnice (s aspoň jedním nenulovým koeficientem u nějakého x_j). Množina všech řešení soustavy konečně mnoha lineárních nerovnic, neboli množina všech přípustných řešení úlohy lineárního programování, je geometricky průnik konečně mnoha poloprostorů, neboli konvexní mnohostěn. (Pro pořádek poznamenejme, že nadrovina je průnik dvou poloprostorů, a tedy v soustavě mohou být kromě nerovnic i rovnice.)

Všimněme si, že konvexní mnohostěn může být i neomezený, například jeden poloprostor je též konvexní mnohostěn. V literatuře se někdy konvexním mnohostěnem myslí jen mnohostěn omezený, tj. takový, který se dá umístit do nějaké dostatečně velké koule. V angličtině se rozlišuje *convex polyhedron* (česky: konvexní polyedr), což je konvexní mnohostěn podle naší definice, a *convex polytope*, což je omezený konvexní mnohostěn.

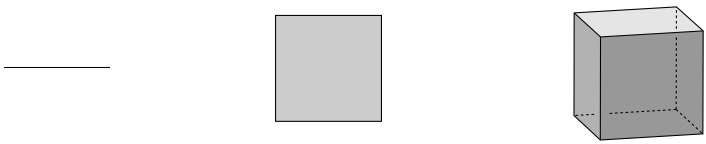
Dimenze konvexního mnohostěnu $P \subseteq \mathbb{R}^n$ je nejmenší dimenze afinního podprostoru, do nějž se dá P umístit. Ekvivalentně, je to největší d,

pro něž P obsahuje nějaké body $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_d$ takové, že d-tice vektorů $(\mathbf{x}_1 - \mathbf{x}_0, \mathbf{x}_2 - \mathbf{x}_0, \ldots, \mathbf{x}_d - \mathbf{x}_0)$ je lineárně nezávislá.

I prázdná množina je konvexním mnohostěnem a její dimenze se obvykle definuje jako -1.

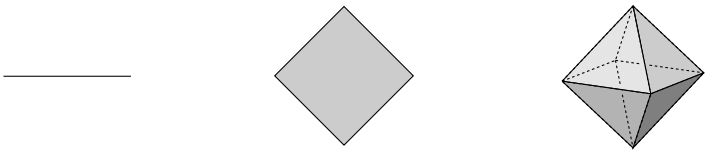
Všechny konvexní mnohoúhelníky v rovině jsou dvoudimenzionální konvexní mnohostěny. Řada typů třídimenzionálních konvexních mnohostěnů se probírá na střední škole a zdobí matematické kabinety, například krychle, kvádry, hranoly, jehlany nebo dokonce pravidelný dvanáctistěn, který může člověk potkat třeba jako méně obvyklý stolní kalendář. Poměrně jednoduché příklady mnohostěnů obecné dimenze n jsou

- n-dimenzionální **krychle** $[-1, 1]^n$, která se dá napsat jako průnik $2n$ poloprostorů (kterých?):



$n = 1$ $n = 2$ $n = 3$

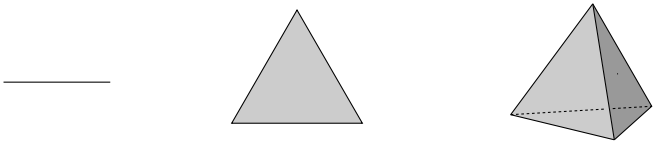
- n-dimenzionální **křížový mnohostěn** $\{\mathbf{x} \in \mathbb{R}^n : |x_1| + |x_2| + \cdots + |x_n| \leq 1\}$:



$n = 1$ $n = 2$ $n = 3$

Pro $n = 3$ dostaneme pravidelný osmistěn. K vyjádření n-dimenzionálního křížového mnohostěnu potřebujeme 2^n poloprostorů (najdete je?).

- Pravidelný n-dimenzionální **simplex**,



$n = 1$ $n = 2$ $n = 3$

který se analyticky nejjednodušeji vyjádří jako podmnožina $\mathbb{R}^{n+1}$:

$$\{\mathbf{x} \in \mathbb{R}^{n+1} : x_1, x_2, \ldots, x_{n+1} \geq 0,\ x_1 + x_2 + \cdots + x_{n+1} = 1\}.$$

Cvičně si všimněme, že to je právě množina přípustných řešení úlohy lineárního programování s jedinou rovnicí $x_1 + x_2 + \cdots + x_{n+1} = 1$ a podmínkami nezápornosti[3], viz obrázek v sekci 4.1. Obecně se simplex říká každému omezenému n-dimenzionálnímu konvexnímu mnohostěnu ohraničenému $n+1$ nadrovinami.

Mnoho zajímavých příkladů konvexních mnohostěnů se dostane právě jako množiny přípustných řešení přirozených úloh lineárního programování. Například LP-relaxace úlohy o perfektním párování maximální váhy (sekce 3.2) vede pro úplný bipartitní graf k takzvanému *Birkhoffovu mnohostěnu*, a pro obecný graf k *Edmondsovu mnohostěnu párování*. Geometrické vlastnosti takovýchto mnohostěnů často zajímavě souvisejí s vlastnostmi kombinatorických objektů a s řešením kombinatorických optimalizačních úloh. Pěkná knížka o konvexních mnohostěnech je

G. M. Ziegler: *Lectures on Polytopes*, Springer-Verlag, Heidelberg, 1994 (opravené druhé vydání 1998).

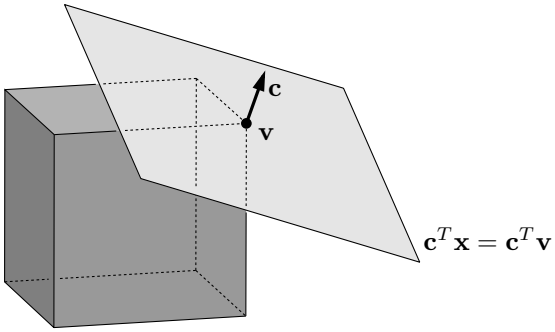
Kapitolu o mnohostěnech obsahují také skripta J. Matoušek: *Introduction to Combinatorial Geometry*, dostupná například na webové stránce autora.

4.4 Vrcholy a bázická přípustná řešení

Vrchol konvexního mnohostěnu si představujeme jako jeho „roh“ nebo „špičku“. Například třídimenzionální krychle má 8 vrcholů a pravidelný osmistěn 6.

Matematicky se vrchol definuje jako bod, v němž nabývá jednoznačného maxima nějaká lineární funkce. Tedy bod $\mathbf{v}$ se nazývá **vrchol** konvexního mnohostěnu $P \subset \mathbb{R}^n$, pokud $\mathbf{v} \in P$ a existuje nenulový vektor $\mathbf{c} \in \mathbb{R}^n$ tak, že $\mathbf{c}^T\mathbf{v} > \mathbf{c}^T\mathbf{y}$ pro každé $\mathbf{y} \in P \setminus \{\mathbf{v}\}$. Geometricky to znamená, že nadrovina $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{v}\}$ se mnohostěnu P dotýká přesně ve $\mathbf{v}$.

[3]Na druhé straně množina přípustných řešení pro úlohu v rovnicovém tvaru rozhodně není vždycky simplex! Simplexová metoda se jmenuje simplexová z poměrně komplikovaného důvodu. Souvisí to s jinou geometrickou představou úlohy v rovnicovém tvaru, než o které jsme mluvili. V ní se m čísel v j-tém *sloupci* matice A spolu s číslem c_j interpretuje jako bod v $\mathbb{R}^{m+1}$. Simplexová metoda se pak dá interpretovat jako pochod po jistých simplexech s vrcholy v těchto bodech. Právě tato představa Dantzigovi dodala důvěru, že simplexová metoda bude dobře fungovat a že má smysl ji zkoumat.



Třídimenzionální mnohostěny mají nejen vrcholy, ale i hrany a stěny. Obecný konvexní mnohostěn $P \subseteq \mathbb{R}^n$ dimenze n může mít vrcholy, hrany, 2-dimenzionální stěny, 3-dimenzionální stěny, až $(n-1)$-dimenzionální stěny. Jsou definovány takto: podmnožina $F \subseteq P$ je ***k*-dimenzionální stěna** konvexního mnohostěnu P, pokud F má dimenzi k a existuje nenulový vektor $\mathbf{c} \in \mathbb{R}^n$ a číslo $z \in \mathbb{R}$ tak, že pro všechna $\mathbf{x} \in F$ platí $\mathbf{c}^T\mathbf{x} = z$ a pro všechna $\mathbf{x} \in P \setminus F$ máme $\mathbf{c}^T\mathbf{x} < z$. Jinými slovy, existuje nadrovina, která se P dotýká přesně v F. Protože taková F je průnikem nadroviny s konvexním mnohostěnem, je sama konvexním mnohostěnem, a tedy má dobře definovanou dimenzi. **Hrana** mnohostěnu je definována jako 1-dimenzionální stěna a vrchol je 0-dimenzionální stěna.

Nyní ukážeme, že vrcholy mnohostěnu a bázická přípustná řešení úlohy lineárního programování jsou totéž.

4.4.1 Věta. *Buď P množina všech přípustných řešení úlohy lineárního programování v rovnicovém tvaru (tedy P je konvexní mnohostěn). Pak následující podmínky pro bod $\mathbf{v} \in P$ jsou ekvivalentní:*

(i) $\mathbf{v}$ *je vrchol mnohostěnu P.*

(ii) $\mathbf{v}$ *je bázické přípustné řešení uvažované úlohy.*

Důkaz. Implikace (i)⇒(ii) plyne ihned z věty 4.2.3 (kde za $\mathbf{c}$ zvolíme vektor definující vrchol $\mathbf{v}$), a zbývá dokázat (ii)⇒(i).

Mějme bázické přípustné řešení $\mathbf{v}$ s přípustnou bází B a definujme vektor $\tilde{\mathbf{c}} \in \mathbb{R}^n$ předpisem $\tilde{c}_j = 0$ pro $j \in B$ a $\tilde{c}_j = -1$ jinak. Máme $\tilde{\mathbf{c}}^T\mathbf{v} = 0$, a přitom $\tilde{\mathbf{c}}^T\mathbf{x} \leq 0$ pro libovolné $\mathbf{x} \geq \mathbf{0}$, tedy $\mathbf{v}$ maximalizuje účelovou funkci $\tilde{\mathbf{c}}^T\mathbf{x}$. Navíc $\tilde{\mathbf{c}}^T\mathbf{x} < 0$ kdykoliv má $\mathbf{x}$ nenulovou složku mimo B. Ale podle tvrzení 4.2.2 je $\mathbf{v}$ jediné přípustné řešení se všemi složkami mimo B nulovými, takže $\mathbf{v}$ je jediný bod z P maximalizující $\tilde{\mathbf{c}}^T\mathbf{x}$. □

Podobná věta platí pro libovolnou úlohu lineárního programování, nejen pro úlohu v rovnicovém tvaru. Dokazovat to zde nebudeme, ale řekneme aspoň definici bázického přípustného řešení pro obecnou úlohu lineárního programování:

Bázické přípustné řešení úlohy lineárního programování s n proměnnými je takové přípustné řešení, pro něž je n lineárně nezávislých omezujících podmínek splněno s rovností.

Omezující podmínka, která je rovnicí, musí být vždycky splněna s rovností, zatímco některé nerovnice mohou být splněny s rovností a jiné s ostrou nerovností. Mezi nerovnice se počítají i podmínky nezápornosti. Lineární nezávislost podmínek znamená, že vektory koeficientů u proměnných jsou lineárně nezávislé, kde například pro $n = 4$ přísluší podmínce $3x_1+5x_3-7x_4 \leq 10$ vektor $(3, 0, 5, -7)$.

Jak víme z lineární algebry, soustava n lineárně nezávislých lineárních rovnic pro n proměnných má právě jedno řešení. Takže pokud $\mathbf{x}$ splňuje s rovností nějakých n lineárně nezávislých podmínek, pak těchto n podmínek splňuje s rovností jedině ono. Geometricky řečeno, podmínky splněné s rovností určují nadroviny, $\mathbf{x}$ leží na nějakých n z nich, a těchto n nadrovin se protíná v jediném bodě.

Definice bázického přípustného řešení pro rovnicový tvar vypadala dost jinak, ale ve skutečnosti je to speciální případ obecnější definice právě uvedené. Pro úlohu v rovnicovém tvaru máme m lineárně nezávislých rovnic splněných vždy, zbývá tedy splnit s rovností $n - m$ podmínek nezápornosti, jež jsou navíc lineárně nezávislé s těmi rovnicemi. Vektor koeficientů podmínky nezápornosti $x_j \geq 0$ je $\mathbf{e}_j$, s jedničkou na místě j a 0 jinde. Má-li $\mathbf{x}$ být bázické přípustné řešení podle nové definice, existuje množina $N \subseteq \{1, 2, \ldots, n\}$ velikosti $n - m$ tak, že $x_j = 0$ pro $j \in N$ a řádky matice A spolu s vektory $(\mathbf{e}_j : j \in N)$ tvoří lineárně nezávislý soubor. To nastane právě tehdy, když matice A_B má lineárně nezávislé řádky, kde $B = \{1, 2, \ldots, n\} \setminus N$, a jsme zpátky u definice bázického přípustného řešení pro rovnicový tvar.

Pro obecnou úlohu lineárního programování nemusí být žádné z optimálních řešení bázické, jak ukazuje třeba úloha

$$\text{maximalizovat } x_1 + x_2 \text{ za podmínky } x_1 + x_2 \leq 1.$$

To je rozdíl proti rovnicovému tvaru (viz větu 4.2.3) a jedna z výhod rovnicového tvaru.

Na intuitivní pojem „rohu“ konvexní množiny se dá nahlížet přinejmenším dvěma způsoby. Jeden z nich je zachycen v uvedené definici vrcholu konvexního mnohostěnu. Jiná definice uvažuje body, které se nedají „generovat úsečkami“. Pro odlišení se jim říká **extremální body**, tedy bod $\mathbf{x}$ je extremální bod konvexní množiny $C \subseteq \mathbb{R}^n$, pokud neexistují dva body $\mathbf{y}, \mathbf{z} \in C$ různé od $\mathbf{x}$ a takové, že $\mathbf{x}$ leží na úsečce $\mathbf{yz}$.

Pro konvexní mnohostěny se dá poměrně snadno ukázat, že extremální body jsou přesně totéž co vrcholy. Tudíž máme další možný ekvivalentní popis

bázických přípustných řešení.

Omezený konvexní mnohostěn je konvexním obalem svých vrcholů. Konvexní mnohostěn obecně nemusí mít vůbec žádné vrcholy – příkladem je poloprostor. Nicméně *omezený* konvexní mnohostěn P nějaké vrcholy vždy má a dokonce platí víc: P je konvexním obalem množiny svých vrcholů. To může z příkladů v dimenzích 2 a 3 intuitivně vypadat jako zjevné, ale důkaz není jednoduchý (v Zieglerově knize zmiňované v předchozím oddílu se toto tvrzení označuje jako „hlavní věta“ teorie mnohostěnů). Důsledkem tvrzení je skutečnost, že se každý konvexní mnohostěn dá vyjádřit jednak jako průnik konečně mnoha poloprostorů a jednak jako konvexní obal konečně mnoha bodů.

5

Simplexová metoda

V této kapitole vysvětlíme simplexovou metodu pro řešení úloh lineárního programování. Budeme využívat pojmy rovnicový tvar a bázické přípustné řešení z předchozí kapitoly.

Gaussova eliminace v lineární algebře má sice fundamentální význam teoretický a didaktický, jako východisko dalších postupů, ale v praxi se nahrazuje složitějšími a efektivnějšími algoritmy. Podobně na řešení větších úloh lineárního programování se nepoužívá základní verze simplexové metody, kterou zde vyložíme, ale varianty složitější. V tomto úvodním textu však nebudeme klást důraz na co nejefektivnější organizaci výpočtu a soustředíme se na základní myšlenky.

5.1 Úvodní příklad

Simplexovou metodu nejdřív ukážeme na řešení následující úlohy:

$$\begin{array}{lrcrcl} \text{maximalizovat} & x_1 & + & x_2 & & \\ \text{za podmínek} & -x_1 & + & x_2 & \leq & 1 \\ & x_1 & & & \leq & 3 \\ & & & x_2 & \leq & 2 \\ & x_1, x_2 \geq 0. & & & & \end{array} \tag{5.1}$$

Schválně bereme úlohu, která není v rovnicovém tvaru, máme sice nezáporné proměnné, ale nerovnice musíme nahradit rovnicemi zavedením po-

mocných proměnných. Rovnicový tvar je

$$\begin{array}{lrcrcrcrcrcl} \text{maximalizovat} & x_1 & + & x_2 & & & & & & & & \\ \text{za podmínek} & -x_1 & + & x_2 & + & x_3 & & & & & = & 1 \\ & x_1 & & & & & + & x_4 & & & = & 3 \\ & & & x_2 & & & & & + & x_5 & = & 2 \\ & x_1, x_2, \ldots, x_5 \geq 0 & & & & & & & & & & \end{array}$$

s maticí

$$A = \begin{pmatrix} -1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

V simplexové metodě se každá úloha přepíše do podoby *simplexové tabulky*, která se pak postupně upravuje. V našem případě začneme se simplexovou tabulkou

$$\begin{array}{rcrcrcr} x_3 & = & 1 & + & x_1 & - & x_2 \\ x_4 & = & 3 & - & x_1 & & \\ x_5 & = & 2 & & & - & x_2 \\ \hline z & = & & & x_1 & + & x_2. \end{array}$$

V prvních třech řádcích jsou rovnice z úlohy, v nichž jsou pomocné proměnné převedeny na levou stranu a zbytek na pravou. Poslední řádek, oddělený čarou, obsahuje novou proměnnou z, která vyjadřuje účelovou funkci.

Ke každé simplexové tabulce náleží jisté bázické přípustné řešení. V našem případě dosadíme 0 za proměnné x_1 a x_2 z pravé strany a bez počítání dostáváme $x_3 = 1, x_4 = 3, x_5 = 2$. Toto přípustné řešení je vskutku bázické s $B = \{3, 4, 5\}$; všimněme si, že A_B je jednotková matice. Proměnné x_3, x_4, x_5 na levé straně jsou bázické a proměnné x_1, x_2 na pravé straně nebázické. Hodnota účelové funkce $z = 0$, která odpovídá tomuto bázickému přípustnému řešení, se vyčte z posledního řádku tabulky.

Z počáteční simplexové tabulky budeme postupně odvozovat další tabulky podobného tvaru. Všechny budou obsahovat *stejné* informace o uvažované úloze, jen jinak zapsané. Postup skončí tabulkou, jejíž příslušné bázické přípustné řešení bude optimální, a jeho optimalita bude z tabulky přímo vidět.

Pustíme se do prvního kroku. Pokusíme se zvýšit hodnotu účelové funkce zvýšením některé z nebázických proměnných x_1 nebo x_2. V naší tabulce snadno odhalíme, že zvýšením hodnoty x_1 se zvýší hodnota z. Totéž platí i o x_2, protože obě proměnné mají kladné koeficienty v z-ové řádce tabulky. Vyberme si třeba x_2, a to budeme zvyšovat, zatímco x_1 zůstane 0. O kolik můžeme x_2 zvýšit? Pokud chceme zachovat přípustnost, musíme dát pozor,

aby x_3, x_4, x_5 neklesla pod 0. Rovnice určující jejich hodnoty mohou tudíž omezovat přírůstek x_2. Uvažme první rovnici

$$x_3 = 1 + x_1 - x_2.$$

Spolu s implicitním omezením $x_3 \geq 0$ nás tato rovnice nechá zvýšit x_2 (při zachování $x_1 = 0$) nejvýš na 1. Druhá rovnice

$$x_4 = 3 - x_1$$

neomezuje x_2 vůbec a třetí rovnice

$$x_5 = 2 - x_2$$

povoluje zvýšení na $x_2 = 2$, než x_5 začne být záporné. Nejpřísnější omezení vyplývá tedy z první rovnice.

Zvýšíme x_2, jak nejvíc to jde, a dostaneme $x_2 = 1$ a $x_3 = 0$. Hodnoty ostatních proměnných ze zbytku tabulky jsou

$$\begin{aligned} x_4 &= 3 - x_1 = 3, \\ x_5 &= 2 - x_2 = 1. \end{aligned}$$

V tomto novém přípustném řešení se x_3 stalo nulovým a x_2 nenulovým. Celkem přirozeně tedy převedeme x_3 na pravou stranu, kde žijí nebázické proměnné, a x_2 na levou stranu, kde sídlí proměnné bázické. Uděláme to pomocí rovnice $x_3 = 1 + x_1 - x_2$, z níž vyjádříme

$$x_2 = 1 + x_1 - x_3.$$

Pravou stranu dosadíme za x_2 do zbývajících rovnic tabulky, a tak získáme novou tabulku

x_2	=	1	+	x_1	−	x_3
x_4	=	3	−	x_1		
x_5	=	1	−	x_1	+	x_3
z	=	1	+	$2x_1$	−	x_3

s $B = \{2, 4, 5\}$, k níž náleží bázické přípustné řešení $\mathbf{x} = (0, 1, 0, 3, 1)$ s hodnotou účelové funkce $z = 1$.

Tento proces přepsání simplexové tabulky do jiné se nazývá **pivotovací krok**. V každém pivotovacím kroku některá nebázická proměnná, v našem případě x_2, *vstoupí* do báze, zatímco některá bázická proměnná, v našem případě x_3, z ní *vystoupí*.

V nové tabulce můžeme vyšší hodnotu účelové funkce docílit zvýšením x_1, kdežto zvýšení x_3 by vedlo k hodnotě menší. První rovnice zvyšování x_1

nijak neomezuje, ze druhé dostáváme $x_1 \leq 3$ a ze třetí rovnice $x_1 \leq 1$, tudíž nejpřísnější omezení vyplývá ze třetí rovnice. Stejně jako v předchozím kroku z ní x_1 vyjádříme a dosadíme za něj do zbývajících rovnic, čímž x_1 vstoupí do báze a přesune se na levou stranu, a x_5 z báze vystoupí a přemístí se na pravou stranu. Dostaneme tabulku

$$\begin{array}{lcrcrcr} x_1 & = & 1 & + & x_3 & - & x_5 \\ x_2 & = & 2 & & & - & x_5 \\ x_4 & = & 2 & - & x_3 & + & x_5 \\ \hline z & = & 3 & + & x_3 & - & 2x_5 \end{array}$$

s $B = \{1, 2, 4\}$, bázickým přípustným řešením $\mathbf{x} = (1, 2, 0, 2, 0)$ a $z = 3$.

Po ještě jednom pivotovacím kroku, v němž do báze vstoupí x_3 a vystoupí z ní x_4, skončíme u tabulky

$$\begin{array}{lcrcrcr} x_1 & = & 3 & - & x_4 & & \\ x_2 & = & 2 & & & - & x_5 \\ x_3 & = & 2 & - & x_4 & + & x_5 \\ \hline z & = & 5 & - & x_4 & - & x_5 \end{array}$$

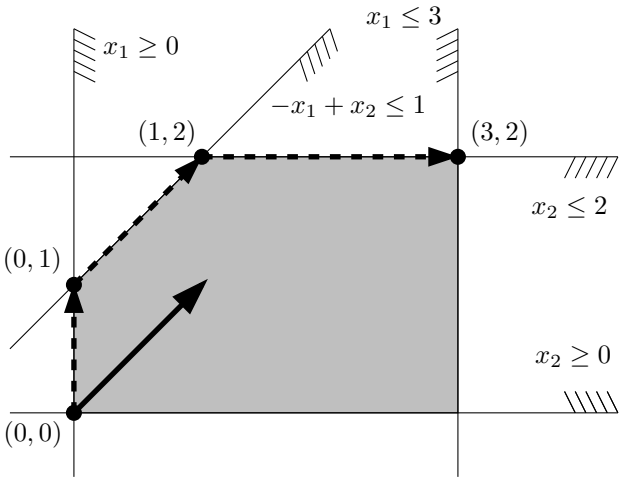
s bází $\{1, 2, 3\}$, bázickým přípustným řešením $\mathbf{x} = (3, 2, 2, 0, 0)$ a $z = 5$. V této tabulce nemůžeme zvětšit žádnou nebázickou proměnnou, aniž bychom snížili hodnotu účelové funkce, takže jsme uvízli. Ale naštěstí to zároveň znamená, že jsme našli optimální řešení. Proč?

Uvažme *libovolné* přípustné řešení $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_5)$ naší úlohy, s hodnotou účelové funkce rovnou nějakému $\tilde{z}$. Nyní $\tilde{\mathbf{x}}$ a $\tilde{z}$ splňují všechny rovnice v poslední tabulce, které jsme dostali z původních rovnic úlohy ekvivalentními úpravami, a tedy musí platit

$$\tilde{z} \ = 5 - \tilde{x}_4 - \tilde{x}_5.$$

To spolu s podmínkami nezápornosti $\tilde{x}_4, \tilde{x}_5 \geq 0$ dává $\tilde{z} \leq 5$. Tabulka dokonce poskytuje i důkaz, že $\mathbf{x} = (3, 2, 2, 0, 0)$ je *jediné* optimální řešení: pokud $z = 5$, tak $x_4 = x_5 = 0$, a to jednoznačně určuje hodnoty ostatních proměnných.

Geometrická ilustrace. Každému přípustnému řešení (x_1, x_2) původní úlohy (5.1) s nerovnicemi odpovídá přesně jedno přípustné řešení $(x_1, x_2, \ldots, x_5)$ upravené úlohy v rovnicovém tvaru, a obráceně. Množiny přípustných řešení jsou ve vhodném smyslu isomorfní, a vylíčený postup simplexové metody můžeme tudíž sledovat na rovinném obrázku pro původní úlohu (5.1):



Vidíme, jak se simplexová metoda přesouvá po hranách od jednoho vrcholu množiny přípustných řešení ke druhému, přičemž hodnota účelové funkce roste, až dosáhne optima. V uvedeném příkladu by šlo jít i kratší cestou, kdybychom se v prvním kroku rozhodli zvyšovat x_1 místo x_2.

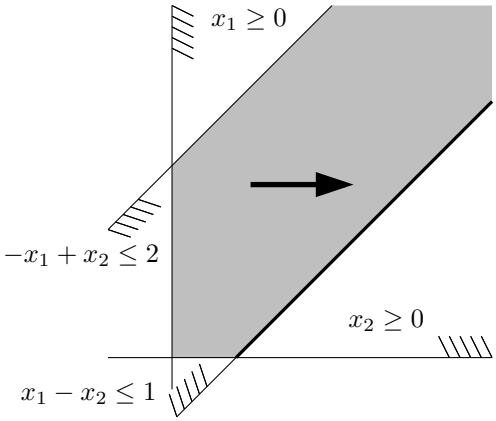
Možné potíže. V uvedeném jednoduchém příkladu proběhla simplexová metoda hladce a bez překážek našla optimální řešení. Obecně se ale musíme umět vypořádat s několika komplikacemi. Ukážeme je na příkladech v následujících oddílech.

5.2 Ošetření výjimek: neomezenost

Co se v simplexové metodě stane pro neomezenou úlohu ukážeme na úloze

$$\begin{array}{lrcrcl} \text{maximalizovat} & x_1 & & & & \\ \text{za podmínek} & x_1 & - & x_2 & \leq & 1 \\ & -x_1 & + & x_2 & \leq & 2 \\ & x_1, x_2 \geq 0. & & & & \end{array}$$

znázorněné na obrázku:



Po obvyklém převodu na rovnicový tvar zavedením pomocných proměnných x_3, x_4 a jejich využitím jako počáteční přípustné báze dostaneme počáteční simplexovou tabulku

$$\begin{array}{lcrcrcr} x_3 & = & 1 & - & x_1 & + & x_2 \\ x_4 & = & 2 & + & x_1 & - & x_2 \\ \hline z & = & & & x_1. & & \end{array}$$

Po prvním pivotovacím kroku se vstupující proměnnou x_1 a vystupující proměnnou x_3 vyjde tabulka

$$\begin{array}{lcrcrcr} x_1 & = & 1 & + & x_2 & - & x_3 \\ x_4 & = & 3 & & & - & x_3 \\ \hline z & = & 1 & + & x_2 & - & x_3. \end{array}$$

Když teď zkusíme zavést do báze x_2, zjistíme, že žádná z rovnic v tabulce neomezuje jeho zvětšování. Můžeme vzít x_2 libovolně velké, a dostaneme z také libovolně velké – úloha je neomezená.

Rozebereme tuto situaci trochu podrobněji. Z tabulky je vidět, že pro libovolné číslo $t \geq 0$ dostaneme přípustné řešení volbou $x_2 = t$, $x_3 = 0$, $x_1 = 1 + t$ a $x_4 = 3$, s hodnotou účelové funkce $z = 1 + t$. Jinak řečeno, polopřímka

$$\{(1,0,0,3) + t(1,1,0,0) : t \geq 0\}$$

je obsažena v množině přípustných řešení a „dosvědčuje“ neomezenost úlohy, neboť účelová funkce na ní nabývá libovolně velkých hodnot. Odpovídající polopřímka pro původní dvoudimenzionální úlohu je na obrázku vytažena tlustě.

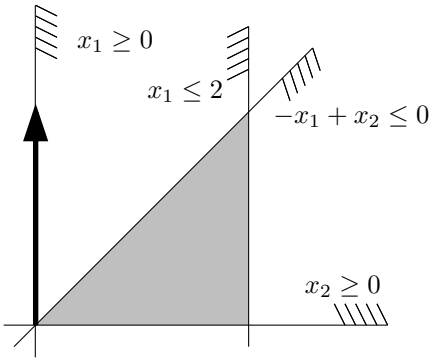
Podobná polopřímka je výstupem simplexové metody pro všechny neomezené úlohy.

5.3 Ošetření výjimek: degenerovanost

Zatímco v případě neomezené úlohy můžeme některé proměnné zvyšovat libovolně, k opačnému extrému někdy dochází v případě degenerované úlohy. Rovnice v tabulce nepovolují zvýšení žádné nebázické proměnné, takže ani zvýšení z pivotovacím krokem není možné.

Uvažme úlohu

$$\begin{array}{lrcrcl} \text{maximalizovat} & x_2 & & & & \\ \text{za podmínek} & -x_1 & + & x_2 & \leq & 0 \\ & x_1 & & & \leq & 2 \\ & x_1, x_2 \geq 0. & & & & \end{array} \tag{5.2}$$



Obvyklým způsobem dostaneme počáteční simplexovou tabulku

$$\begin{array}{lcrcrcl} x_3 & = & & & x_1 & - & x_2 \\ x_4 & = & 2 & - & x_1 & & \\ \hline z & = & & & & & x_2 \end{array}$$

(kvíz: jaké bázi tato tabulka odpovídá, a jaké je příslušné bázické řešení?). Jediný kandidát pro vstup do báze je zde x_2, ale první řádek tabulky ukazuje, že jeho hodnotu nelze zvýšit, aniž by se x_3 stalo záporným.

Uvíznutí v tomto případě bohužel neznamená, že jsme nalezli optimální řešení, takže musíme provést pivotovací krok s „nulovým postupem“. V našem příkladu vstup x_2 do báze (a výstup x_3) vyústí v jinou tabulku se stejným bázickým přípustným řešením:

$$\begin{array}{lcrcrcl} x_2 & = & & & x_1 & - & x_3 \\ x_4 & = & 2 & - & x_1 & & \\ \hline z & = & & & x_1 & - & x_3. \end{array}$$

Situace se nicméně vylepšila. Nebázickou proměnnou x_1 nyní lze zvýšit a po jejím vstupu do báze, výměnou za x_4, už dostaneme závěrečnou tabulku

$$\begin{array}{rcrcrcr} x_1 & = & 2 & & & - & x_4 \\ x_2 & = & 2 & - & x_3 & - & x_4 \\ \hline z & = & 2 & - & x_3 & - & x_4 \end{array}$$

s optimálním řešením $\mathbf{x} = (2, 2, 0, 0)$.

Situace, kdy jsme nuceni k pivotovacímu kroku neměnícímu hodnotu účelové funkce, může nastat jedině pro úlohu, kde jednomu bázickému přípustnému řešení odpovídá více přípustných bází. Takové úlohy se nazývají **degenerované**.

Je snadno vidět, že aby nějakému bázickému přípustnému řešení odpovídalo víc bází, musí v něm být některé *bázické* proměnné rovny nule.

V našem příkladu jsme mohli zvyšovat hodnotu účelové funkce už po jednom degenerovaném pivotovacím kroku. Obecně to může trvat mnohem déle. Dokonce se může stát, že se tabulka v posloupnosti degenerovaných pivotovacích kroků opakuje, takže by algoritmus mohl procházet nekonečnou posloupností tabulek bez jakéhokoli pokroku. Tento jev se nazývá **zacyklení**. Příklad úlohy, kde se simplexová metoda může zacyklit, je poměrně komplikovaný (nejmenší příklad má 6 proměnných a 3 rovnice) a tady ho uvádět nebudeme.

Pokud výpočet simplexovou metodou neskončí, musí se nutně zacyklit. To je proto, že možných simplexových tabulek pro jednu úlohu je jen konečně mnoho, konkrétně nejvýše $\binom{n}{m}$, což dokážeme v oddílu 5.5.

Jak zabránit zacyklení? O tom krátce pojednáme v oddílu 5.7.

5.4 Ošetření výjimek: nepřípustnost

Aby mohl simplexový algoritmus vůbec začít, potřebuje nějakou přípustnou bázi. V příkladech, které jsme zatím uváděli, jsme přípustnou bázi dostali víceméně zadarmo. Tak je tomu u všech úloh tvaru

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} \leq \mathbf{b} \text{ a } \mathbf{x} \geq \mathbf{0}$$

s $\mathbf{b} \geq \mathbf{0}$. Vskutku, pomocné proměnné, zavedené při převodu na rovnicový tvar, poslouží jako přípustná báze.

Obecně je ale nalezení přípustného řešení stejně obtížné jako nalezení optimálního řešení (viz poznámku v sekci 1.3). Naštěstí se počáteční přípustné báze dá najít pomocí samotné simplexové metody, která se použije na vhodnou pomocnou úlohu. U této pomocné úlohy je počáteční přípustná báze k dispozici automaticky, a z jejího optimálního řešení získáme přípustnou bázi původní úlohy.

Konstrukci pomocné úlohy vysvětlíme na příkladu následující úlohy v rovnicovém tvaru:

$$\begin{array}{lllllllll} \text{maximalizovat} & x_1 & + & 2x_2 & & & & & \\ \text{za podmínek} & x_1 & + & 3x_2 & + & x_3 & = & 4 \\ & & & 2x_2 & + & x_3 & = & 2 \\ & x_1, x_2, x_3 \geq 0. & & & & & & & \end{array}$$

Přípustné řešení budeme vyrábět z vektoru $(x_1, x_2, x_3) = (0, 0, 0)$. Ten je nezáporný, ale není přípustný, protože nesplňuje rovnice úlohy. Zavedeme pomocné proměnné x_4 a x_5 jako „opravy" nepřípustnosti: $x_4 = 4 - x_1 - 3x_2 - x_3$ vyjadřuje, o kolik původní proměnné x_1, x_2, x_3 nesplňují první rovnici, a $x_5 = 2 - 2x_2 - x_3$ hraje podobnou roli pro druhou rovnici. Podaří-li se nám najít nějaké nezáporné hodnoty x_1, x_2, x_3, pro něž tyto opravy vyjdou nulové, budeme mít přípustné řešení původní úlohy.

Úkol najít nezápornou x_1, x_2, x_3 s nulovými opravami můžeme vyjádřit úlohou lineárního programování:

$$\begin{array}{lllllllllllll} \text{maximalizovat} & & & & & & - & x_4 & - & x_5 & & & \\ \text{za podmínek} & x_1 & + & 3x_2 & + & x_3 & + & x_4 & & & = & 4 \\ & & & 2x_2 & + & x_3 & & & + & x_5 & = & 2 \\ & x_1, x_2, \ldots, x_5 \geq 0. & & & & & & & & & & \end{array}$$

Optimální hodnota účelové funkce $-x_4 - x_5$ je 0, právě když pro původní úlohu existují hodnoty x_1, x_2, x_3 s nulovými opravami (neboli přípustné řešení).

To je ta správná pomocná úloha. Proměnné x_4 a x_5 tvoří přípustnou bázi, s bázickým přípustným řešením $(0, 0, 0, 4, 2)$. Vyjádříme ještě účelovou funkci pomocí nebázických proměnných, tedy ve tvaru $z = -6 + x_1 + 5x_2 + 2x_3$, a už můžeme na pomocnou úlohu pustit simplexovou metodu.

Úloha je určitě omezená, protože účelová funkce nemůže být kladná, a simplexová metoda najde bázické přípustné řešení, které je optimální. Čtenář může cvičně zkontrolovat, že necháme-li v prvním pivotovacím kroku vstoupit do báze x_1 a ve druhém x_3, vyjde závěrečná simplexová tabulka

$$\begin{array}{llllllll} x_1 & = & 2 & - & x_2 & - & x_4 & + & x_5 \\ x_3 & = & 2 & - & 2x_2 & & & - & x_5 \\ \hline z & = & & & & - & x_4 & - & x_5. \end{array}$$

Příslušné optimální řešení $(2, 0, 2, 0, 0)$ dává bázické přípustné řešení původní úlohy $(x_1, x_2, x_3) = (2, 0, 2)$. Počáteční simplexovou tabulku pro původní úlohu dokonce dostaneme ze závěrečné tabulky úlohy pomocné,

vynecháním sloupců s x_4 a x_5 a změnou účelové funkce:

$$\begin{array}{rcrcr} x_1 & = & 2 & - & x_2 \\ x_3 & = & 2 & - & x_2 \\ \hline z & = & 2 & + & x_2. \end{array}$$

V našem velmi jednoduchém příkladu pak už jediný pivotovací krok dosáhne optima.

5.5 Simplexové tabulky obecně

To, co jsme zatím vysvětlili na příkladech, zformulujeme v této a následující sekci obecně a (povětšinou) s důkazy.

Uvažme obecnou úlohu v rovnicovém tvaru

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} = \mathbf{b} \text{ a } \mathbf{x} \geq \mathbf{0}.$$

Simplexová metoda vyrábí při řešení této úlohy posloupnost simplexových tabulek. Každá z nich odpovídá nějaké přípustné bázi B a určuje nějaké bázické přípustné řešení. (Pro jistotu připomeňme, že přípustná báze je m-prvková $B \subseteq \{1, 2, \ldots, n\}$, pro niž je matice A_B regulární a (jediné) řešení soustavy $A_B\mathbf{x}_B = \mathbf{b}$ je nezáporné.) Formálně zavedeme simplexovou tabulku jako jistou soustavu lineárních rovnic speciálního tvaru, v níž bázické proměnné a proměnná z, reprezentující hodnotu účelové funkce, stojí na levé straně a jsou vyjádřeny pomocí nebázických proměnných.

Simplexová tabulka $\mathcal{T}(B)$ určená přípustnou bází B je soustava $m+1$ lineární rovnice pro proměnné $x_1, x_2, \ldots, x_n$ a z, která má *stejnou množinu řešení* jako soustava $A\mathbf{x} = \mathbf{b}$, $z = \mathbf{c}^T\mathbf{x}$ a v maticovém zápisu vypadá takto:

$$\begin{array}{rcrcr} \mathbf{x}_B & = & \mathbf{p} & + & Q\,\mathbf{x}_N \\ \hline z & = & z_0 & + & \mathbf{r}^T\mathbf{x}_N, \end{array}$$

kde $\mathbf{x}_B$ je vektor bázických proměnných, $N = \{1, 2, \ldots, n\} \setminus B$, $\mathbf{x}_N$ je vektor nebázických proměnných, $\mathbf{p} \in \mathbb{R}^m$, $\mathbf{r} \in \mathbb{R}^{n-m}$, Q je matice typu $m \times (n-m)$ a $z_0 \in \mathbb{R}$.

Bázické přípustné řešení, příslušné k takové tabulce, se z ní dá okamžitě vyčíst: dostaneme jej dosazením $\mathbf{x}_N = \mathbf{0}$, tj. máme $\mathbf{x}_B = \mathbf{p}$. Z přípustnosti báze B plyne $\mathbf{p} \geq \mathbf{0}$. Účelová funkce pro toto bázické přípustné řešení má hodnotu $z_0 + \mathbf{r}^T\mathbf{0} = z_0$.

Hodnoty $\mathbf{p}, Q, \mathbf{r}, z_0$ můžeme snadno vyjádřit pomocí B a $A, \mathbf{b}, \mathbf{c}$:

5.5.1 Lemma. *Pro každou přípustnou bázi B existuje právě jedna simplexová tabulka, a ta je daná vztahy*

$$Q = -A_B^{-1}A_N,\ \mathbf{p} = A_B^{-1}\mathbf{b},\ z_0 = \mathbf{c}_B^T A_B^{-1}\mathbf{b}\ \text{ a }\ \mathbf{r} = \mathbf{c}_N - (\mathbf{c}_B^T A_B^{-1} A_N)^T.$$

Vzorečky je zbytečné si pamatovat, v případě potřeby se lehce odvodí. Důkaz je nezáživný a zařazujeme ho spíš pro pořádek. Řekneme ho stručněji než jiné části výkladu a některé detaily přenecháme pilnému čtenáři, a podobně si budeme počínat u dalších důkazů podobného druhu.

Důkaz. Nejdřív, jak se na ty formule přijde: přepíšeme soustavu $A\mathbf{x} = \mathbf{b}$ na $A_B\mathbf{x}_B = \mathbf{b} - A_N\mathbf{x}_N$ a zleva vynásobíme inverzní maticí A_B^{-1} (to jsou ekvivalentní úpravy), čímž dostaneme

$$\mathbf{x}_B = A_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N.$$

Dosadíme-li toto za $\mathbf{x}_B$ do rovnice $z = \mathbf{c}^T\mathbf{x} = \mathbf{c}_B^T\mathbf{x}_B + \mathbf{c}_N^T\mathbf{x}_N$, vyjde

$$\begin{aligned} z &= \mathbf{c}_B^T(A_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N) + \mathbf{c}_N^T\mathbf{x}_N = \\ &= \mathbf{c}_B^T A_B^{-1}\mathbf{b} + (\mathbf{c}_N^T - c_B^T A_B^{-1}A_N)\mathbf{x}_N. \end{aligned}$$

Tudíž vzorce v lemmatu opravdu dávají simplexovou tabulku, a zbývá ověřit jednoznačnost.

Nechť $\mathbf{p}, Q, \mathbf{r}, z_0$ určují simplexovou tabulku pro přípustnou bázi B a $\mathbf{p}', Q', \mathbf{r}', z_0'$ také. Protože každá volba $\mathbf{x}_N$ jednoznačně určuje $\mathbf{x}_B$, musí platit rovnost $\mathbf{p} + Q\mathbf{x}_N = \mathbf{p}' + Q'\mathbf{x}_N$ pro všechna $\mathbf{x}_N \in \mathbb{R}^{n-m}$. Volba $\mathbf{x}_N = \mathbf{0}$ dá $\mathbf{p} = \mathbf{p}'$, a dosazujeme-li za $\mathbf{x}_N$ postupně všechny vektory $\mathbf{e}_i$ standardní báze, vyjde i $Q = Q'$. Podobně se dokáže $z_0 = z_0'$ a $\mathbf{r} = \mathbf{r}'$. □

5.6 Simplexová metoda obecně

Optimalita. Stejně jako v konkrétním příkladu v oddílu 5.1 nahlédneme:

> Je-li $\mathcal{T}(B)$ simplexová tabulka, v jejíž poslední řádce jsou všechny koeficienty u nebázických proměnných nekladné, neboli
>
> $$\mathbf{r} \le \mathbf{0},$$
>
> potom je příslušné bázické přípustné řešení *optimální*.

Skutečně, bázické přípustné řešení příslušné takové tabulce má účelovou funkci rovnou z_0, a pro libovolné jiné přípustné řešení $\tilde{\mathbf{x}}$ máme $\tilde{\mathbf{x}}_N \ge 0$ a $\mathbf{c}^T\tilde{\mathbf{x}} = z_0 + \mathbf{r}^T\tilde{\mathbf{x}}_N \le z_0$.

Pivotovací krok: kdo vstoupí a kdo vystoupí. V každém kroku simplexové metody přejdeme od „staré“ báze B a simplexové tabulky $\mathcal{T}(B)$ k „nové“ bázi B' a odpovídající simplexové tabulce $\mathcal{T}(B')$. Vždy jedna nebázická proměnná x_v vstoupí do báze a jedna bázická proměnná x_u z báze vystoupí[1], tedy $B' = (B \setminus \{u\}) \cup \{v\}$.

Nejdřív vždycky vybereme vstupující proměnnou x_v, a to podle následujícího kritéria.

> Do báze smí vstoupit taková nebázická proměnná, jejíž koeficient v posledním řádku simplexové tabulky je *kladný*.

Jenom u takových nebázických proměnných totiž jejich zvětšením vzroste hodnota účelové funkce.

Toto kritérium typicky splňuje několik nebázických proměnných, takže pro vstupující proměnnou je obvykle víc možností. O jejím výběru řekneme více v sekci 5.7.

Když se už rozhodneme, že vstupující proměnná bude x_v, zbývá vybrat vystupující proměnnou.

> Vystupující proměnná x_u musí být taková, že její nezápornost spolu s příslušnou rovnicí simplexové tabulky nejpřísněji omezuje zvýšení vstupující proměnné x_v.

Konkrétní vyjádření tohoto požadavku může vypadat složitě kvůli zmatku s indexy, ale myšlenku jsme už viděli v příkladu a je zcela jednoduchá. Pišme $B = \{j_1, j_2, \ldots, j_m\}$, $j_1 < j_2 < \cdots < j_m$, a $N = \{\ell_1, \ell_2, \ldots, \ell_{n-m}\}$, $\ell_1 < \ell_2 < \cdots < \ell_{n-m}$. Řečeno slovy, j_i je i-tý nejmenší prvek množiny B a ℓ_k je k-tý nejmenší prvek množiny N. Pak má i-tá rovnice simplexové tabulky tvar

$$x_{j_i} = p_i + \sum_{k=1}^{n-m} q_{ik} x_{\ell_k}.$$

Index v zvolené vstupující proměnné potřebujeme teď psát ve tvaru $v = \ell_t$; obšírněji řečeno, definujeme $t \in \{1, 2, \ldots, n-m\}$ jako ten index, pro nějž $v = \ell_t$. Podobně index u vystupující proměnné (který ještě nebyl vybrán) budeme psát ve tvaru $u = j_s$.

Protože všechny nebázické proměnné x_{ℓ_k}, $k \neq t$, mají zůstat nulové, podmínka nezápornosti proměnné x_{j_i} omezuje možné hodnoty vstupující proměnné x_{ℓ_t} prostřednictvím nerovnosti $-q_{it} x_{\ell_t} \leq p_i$. Pokud $q_{it} \geq 0$,

[1] Písmena u a v tady nejsou vektory (abeceda není zas tak dlouhá). Máme v jako *v*stupující a u jako *u*stupující, chcete-li.

tato nerovnost zvyšování x_{ℓ_t} nijak neomezuje, zatímco pro $q_{it} < 0$ dává ohraničení $x_{\ell_t} \leq -p_i/q_{it}$.

Vystupující proměnná x_{j_s} je tedy vždycky taková, pro niž

$$q_{st} < 0 \quad \text{a} \quad -\frac{p_s}{q_{st}} = \min\left\{-\frac{p_i}{q_{it}} : q_{it} < 0,\ i = 1, 2, \ldots, m\right\}. \tag{5.3}$$

To jest, v simplexové tabulce si všímáme jen řádků, v nichž je koeficient x_v záporný. V nich tím koeficientem vydělíme složku vektoru $\mathbf{p}$, změníme znaménko, a ze všech takových podílů hledáme minimum.

Jestliže x_v nemá záporný koeficient v žádném řádku, tj. minimum na pravé straně rovnice (5.3) je přes prázdnou množinu, je úloha neomezená a výpočet končí.

Pro důkaz toho, že simplexová metoda opravdu funguje, je potřeba následující lemma.

5.6.1 Lemma. *Jestliže B je přípustná báze a $\mathcal{T}(B)$ je odpovídající simplexová tabulka, a jestliže vstupující proměnná x_v a vystupující proměnná x_u byly vybrány podle popsaných kriterií (a jinak libovolně), je $B' = (B \setminus \{u\}) \cup \{v\}$ zase přípustná báze.*

Pokud žádné x_u nesplňuje kritérium pro volbu vystupující proměnné, pak je úloha neomezená. Pro všechna $t \geq 0$ dostaneme dosazením t za x_v a 0 za ostatní nebázické proměnné přípustné řešení, a hodnota účelové funkce pro tato řešení pro $t \to \infty$ jde do nekonečna.

Důkaz je jeden z těch, které nejsou pro základní porozumění látce potřeba.

Důkaz (náznak). Je potřeba ověřit, že $A_{B'}$ je regulární. To platí, právě když $A_B^{-1}A_{B'}$ je regulární (protože u A_B regularitu předpokládáme). Matice $A_{B'}$ má $m-1$ sloupců stejných jako A_B, a pro ně jsou odpovídající sloupce $A_B^{-1}A_{B'}$ rovny (navzájem různým) sloupcům jednotkové matice. Zbývající sloupec matice $A_B^{-1}A_{B'}$ se vyskytuje v simplexové tabulce $\mathcal{T}(B)$ jako sloupec nebázické proměnné x_v (protože $Q = -A_B^{-1}A_N$ podle lemmatu 5.5.1). V řádku odpovídajícím vystupující proměnné x_u má tento sloupec nenulové číslo q_{st}, tak jsme vystupující proměnnou vybírali, a ostatní sloupce $A_B^{-1}A_{B'}$ tam mají 0, čili matice je opravdu regulární.

Dále je třeba zkontrolovat přípustnost báze B'. Tady se využije toho, že nové bázické přípustné řešení, to pro B', se dá napsat pomocí starého, a nezápornost jeho bázických proměnných jsou přesně ty podmínky, které jsme použili při výběru vystupující proměnné. Prakticky stejně se ukáže část lemmatu, pojednávající o neomezených úlohách. Podrobnosti si dovolíme vynechat. □

Geometrický pohled. Jak jsme viděli v části 4.4, bázická přípustná řešení odpovídají vrcholům mnohostěnu přípustných řešení. Není těžké ověřit, že

pivotovací krok simplexové metody odpovídá přesunu z jednoho vrcholu do jiného po hraně mnohostěnu (přičemž hrana je 1-dimenzionální stěna, tedy geometricky úsečka spojující ony dva vrcholy, viz část 4.4). Výjimkou mohou být degenerované pivotovací kroky, kdy se zůstává v témže vrcholu a mění se jen přípustná báze.

Organizace výpočtu. Když z dané simplexové tabulky $\mathcal{T}(B)$ najdeme novou přípustnou bázi B', novou simplexovou tabulku $\mathcal{T}(B')$ bychom mohli vypočítat podle vzorců z lemmatu 5.5.1. Tak se to ovšem nedělá, protože to je neefektivní.

Při *ručním výpočtu* se v pivotovacích krocích stará simplexová tabulka přepočítává na novou. Jeden takový přístup jsme už předvedli na příkladu. Vezme se ta rovnice staré tabulky, která má na levé straně vystupující proměnnou x_u, a v ní se na levou stranu převede proměnná vstupující, x_v. To bude v nové tabulce rovnice pro x_v. Z této rovnice se pak za x_v dosadí do všech ostatních rovnic staré tabulky i do rovnice pro z, a tím je konstrukce nové tabulky hotová.

Tradičně se úprava simplexové tabulky prezentuje trochu jinak, způsobem velmi podobným Gaussově eliminaci. Poněvadž dnes se podle simplexové metody počítá ručně asi už výhradně na cvičeních z lineárního programování, zůstaneme u výše uvedené formulace.

V počítačových implementacích simplexové metody se neudržuje simplexová tabulka, nýbrž jenom bázické složky přípustného řešení, tj. vektor $\mathbf{p} = A_B^{-1}\mathbf{b}$, a inverzní matice A_B^{-1}, a ostatní složky simplexové tabulky se dopočítají, až když jsou potřeba (všimněme si, že pro výběr vstupující proměnné a test optimality potřebujeme jen poslední řádek, a pro výběr vystupující proměnné jen sloupec vystupující proměnné a vektor $\mathbf{p}$).

Takovému výpočetnímu postupu se v literatuře říká *revidovaná simplexová metoda*. Pro m mnohem menší než n je zpravidla podstatně efektivnější než udržování celé tabulky. Speciálně na udržování A_B^{-1} stačí $O(m^2)$ operací na každý pivotovací krok, kdežto přepočítání celé simplexové tabulky vyžaduje řádově mn operací.

Protože o efektivní implementaci simplexové metody nám zde tolik nejde, nebudeme revidovanou simplexovou metodu popisovat přesně. Poznamenejme ještě, že udržovat inverzní matici A_B^{-1} zpravidla není z hlediska efektivity a zaokrouhlovacích chyb nejvhodnější, a v praxi se místo toho pracuje například s LU-rozkladem matice A_B.

Hledání počáteční přípustné báze. Pokud u dané úlohy není k dispozici nějaká „zjevná“ počáteční báze, hledáme ji postupem naznačeným v sekci 5.4. Pro výchozí úlohu v obvyklém rovnicovém tvaru

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} = \mathbf{b} \text{ a } \mathbf{x} \geq \mathbf{0}$$

napřed zařídíme, aby $\mathbf{b} \geq \mathbf{0}$ (rovnice, kde $b_i < 0$, vynásobíme -1). Pak přidáme m nových proměnných x_{n+1} až x_{n+m} a řešíme nejdřív pomocnou úlohu

$$\begin{array}{ll} \text{maximalizovat} & -(x_{n+1} + x_{n+2} + \cdots + x_{n+m}) \\ \text{za podmínek} & \bar{A}\,\bar{\mathbf{x}} = \mathbf{b} \\ & \bar{\mathbf{x}} \geq \mathbf{0}, \end{array}$$

kde $\bar{\mathbf{x}} = (x_1, \ldots, x_{n+m})$ je vektor všech proměnných včetně pomocných a matice $\bar{A} = (A \,|\, I_m)$ vznikne připsáním jednotkové matice k A zprava. Z jakéhokoli přípustného řešení původní úlohy dostaneme přípustné řešení úlohy pomocné dosazením $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$. Takové přípustné řešení má účelovou funkci rovnou 0, a je tedy optimální. Tudíž původní úloha je přípustná, právě když optimální řešení pomocné úlohy splňuje $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$.

V pomocné úloze tvoří proměnné x_{n+1} až x_{n+m} přípustnou bázi, a můžeme nasadit simplexovou metodu. Ta vrátí nějaké optimální řešení pomocné úlohy. Pokud pro něj neplatí $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$, jsme hotovi – původní úloha je nepřípustná.

Předpokládejme tedy, že máme optimální řešení pomocné úlohy splňující $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$. Simplexová metoda vždycky vrací bázické přípustné řešení, a v případě uvažované pomocné úlohy v bázi obvykle nebude žádná z proměnných x_{n+1} až x_{n+m}. Taková báze je i přípustnou bází pro původní úlohu a umožňuje pro ni nastartovat simplexovou metodu.

V jistých degenerovaných případech se může stát, že v bázi, vrácené simplexovou metodou pro pomocnou úlohu, zbude některá z proměnných x_{n+1} až x_{n+m}, a takovou bázi nemůžeme přímo použít pro úlohu původní. To ale je jen kosmetická vada. Vrácené optimální řešení má nanejvýš m nenulových složek, a jejich sloupce v matici A jsou lineárně nezávislé. Jestliže je těchto sloupců méně než m, můžeme je snadno doplnit na bázi dalšími lineárně nezávislými sloupci matice A, viz lemma 4.2.1. V simplexové metodě se to elegantně implementuje speciálními pivotovacími kroky, v nichž se z báze vyhazují případné zbytky proměnných x_{n+1} až x_{n+m}, ale tím se zde nebudeme zabývat.

5.7 Pivotovací pravidla, zacyklení, efektivita

Pivotovací pravidlo je jakékoli pravidlo pro výběr vstupující proměnné v případě, že je více možností, což obvykle je. Výjimečně může být více možností také pro výběr vystupující proměnné, a některá pivotovací pravidla specifikují i ten, ale to zpravidla není tak podstatné.

Počet pivotovacích kroků nutných k vyřešení úlohy lineárního programování na pivotovacím pravidle podstatně závisí. (Viz třeba příklad v sekci 5.1.) Problém je samozřejmě v tom, že předem nevíme, která volba bude nakonec nejvýhodnější.

Zde uvedeme některá běžná pivotovací pravidla. Termínem „zlepšující proměnná“ je myšlena jakákoli nebázická proměnná s kladným koeficientem v řádce účelové funkce, tedy kandidát na vstupní proměnnou.

NEJVĚTŠÍ KOEFICIENT. Vyber zlepšující proměnnou s největším koeficientem v řádku účelové funkce z. To je původní, Dantzigem navržené pravidlo, které maximalizuje zlepšení z *na jednotku zvýšení* vstupní proměnné.

NEJVĚTŠÍ PŘÍRŮSTEK. Vyber zlepšující proměnnou, která vede k největšímu *absolutnímu* zvýšení z. Toto pravidlo je výpočetně náročnější než NEJVĚTŠÍ KOEFICIENT, ale lokálně maximalizuje zvýšení z.

NEJSTRMĚJŠÍ HRANA. Vyber zlepšující proměnnou, jejímž zavedením do báze se průběžné bázické přípustné řešení posune ve směru, který svírá nejmenší úhel s vektorem $\mathbf{c}$. Zapsáno vzorcem, má se maximalizovat podíl

$$\frac{\mathbf{c}^T(\mathbf{x}_{\text{nové}} - \mathbf{x}_{\text{staré}})}{\|\mathbf{c}\| \cdot \|\mathbf{x}_{\text{nové}} - \mathbf{x}_{\text{staré}}\|},$$

kde $\mathbf{x}_{\text{staré}}$ je bázické přípustné řešení pro momentální simplexovou tabulku a $\mathbf{x}_{\text{nové}}$ je bázické přípustné řešení pro tabulku, kterou bychom dostali vstupem uvažované zlepšující proměnné do báze. (Připomeňme, že $\|\mathbf{v}\| = (v_1^2 + v_2^2 + \cdots + v_n^2)^{1/2} = \sqrt{\mathbf{v}^T\mathbf{v}}$ značí euklidovskou délku vektoru $\mathbf{v}$, a výraz $\mathbf{u}^T\mathbf{v}/(\|\mathbf{u}\| \cdot \|\mathbf{v}\|)$ je kosinus úhlu, sevřeného vektory $\mathbf{u}$ a $\mathbf{v}$.)

Toto je v praxi šampion mezi pivotovacími pravidly, podle rozsáhlých výpočetních studií je většinou efektivnější než všechna ostatní pravidla popsaná zde i než mnohá další.

NEJMENŠÍ INDEX. Vyber zlepšující proměnnou s nejmenším indexem, a pokud je víc možností pro vystupující proměnnou, také vyber tu s nejmenším indexem. Toto je takzvané BLANDOVO PRAVIDLO a má hlavně teoretický význam, protože zabraňuje zacyklení, což ještě zmíníme trochu podrobněji.

NÁHODNÁ HRANA. Vyber ze všech zlepšujících proměnných náhodně, všechny se stejnou pravděpodobností. To je nejjednodušší příklad *pravděpodobnostního pravidla*, v němž se při výběru nějakým způsobem používají náhodná čísla.

Boj se zacyklením. Jak jsme už řekli, může se stát, že simplexová metoda se pro některé úlohy zacyklí (a to je teoreticky jediná možnost, jak může selhat). V praxi se na takovou situaci narazí velice zřídka, pokud vůbec, takže mnohé implementace možnost zacyklení prostě ignorují.

Z teoretického hlediska jsou přinejmenším dva způsoby, jak možnost zacyklení vyloučit. Jedna z nich je už zmíněné Blandovo pravidlo, o němž se dá dokázat, že při jeho důsledném použití se simplexová metoda nikdy nezacyklí (důkaz není jednoduchý a dělat ho nebudeme). Bohužel z hlediska rychlosti výpočtu je Blandovo pravidlo jedno z nejméně efektivních pivotovacích pravidel a v praxi se téměř nepoužívá.

Jiná možnost se v literatuře najde pod záhlavím *symbolické perturbace* nebo *lexikografické pravidlo*. Tento přístup zde jenom načrtneme.

Zacyklení se může vyskytnout jenom u degenerovaných úloh. U nich se může objevit nejednoznačnost při volbě výstupní proměnné. Lexikografické pravidlo v případě takové nejednoznačnosti rozhoduje následovně. Předpokládejme, že S je množina řádkových indexů s takových, že

$$q_{st} < 0 \text{ a } -\frac{p_s}{q_{st}} = \min\left\{-\frac{p_i}{q_{it}} : q_{it} < 0,\ i = 1, 2, \ldots, m\right\}$$

(přitom t je sloupcový index odpovídající vstupující proměnné). Jinak řečeno, všechny indexy z S jsou kandidáty na výstupní proměnnou. Podle lexikografického pravidla se vybere vždycky ten index $s \in S$, pro nějž je vektor

$$\left(\frac{q_{s1}}{q_{st}}, \ldots, \frac{q_{s(n-m)}}{q_{st}}\right)$$

nejmenší v lexikografickém uspořádání. (Připomeňme, že vektor $\mathbf{x} \in \mathbb{R}^k$ předchází vektor $\mathbf{y} \in \mathbb{R}^k$ v **lexikografickém uspořádání**, pokud $x_1 < y_1$, nebo pokud $x_1 = y_1$ a $x_2 < y_2$,..., nebo pokud $x_1 = y_1$, $x_2 = y_2$,..., $x_{k-1} = y_{k-1}$ a $x_k < y_k$.) Poněvadž matice A má hodnost m, dá se ukázat, že žádné dva z uvedených vektorů se nemohou rovnat, takže volba vystupující proměnné podle tohoto pravidla je vždy jednoznačná.

Je známo, že lexikografické pravidlo nikdy nevede k zacyklení. Z výpočetního hlediska může být pro hodně degenerované úlohy dosti náročné, protože na to, abychom rozhodli o vystupující proměnné, musíme někdy porovnat mnoho komponent zmíněných vektorů.

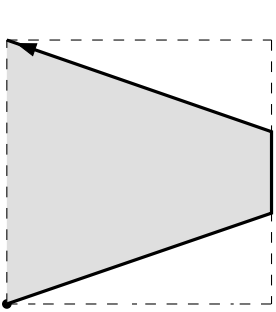
Lexikografické pravidlo má následující geometrický podklad. Pro úlohy v rovnicovém tvaru degenerovanost znamená, že množina F všech řešení soustavy rovnic $A\mathbf{x} = \mathbf{b}$ obsahuje bod, který má více než $n - m$ nulových složek. Tudíž F není v obecné poloze vzhledem k souřadnicovým osám. Účinek lexikografického pravidla je vpodstatě stejný, jako kdybychom afinní podprostor F o nepatrný kousek posunuli, čehož by se dosáhlo maličkou změnou vektoru $\mathbf{b}$. Takové posunutí přivede F do „obecné polohy“, čímž zmizí degenerovanost, a přitom se optimální řešení úlohy změní jen libovolně málo. Lexikografické pravidlo simuluje efekt vhodného „nekonečně malého“ posunutí (neboli „perturbace“) podprostoru F.

Efektivita simplexové metody. V praxi funguje simplexová metoda zpravidla velmi uspokojivě i pro velké úlohy. Výpočetní experimenty na-

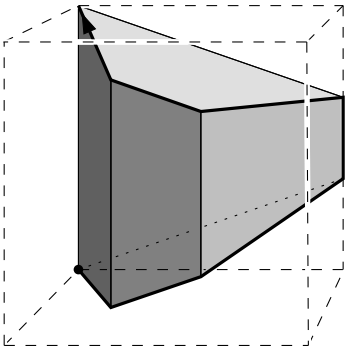
značují, že jí pro úlohy v rovnicovém tvaru s m omezeními typicky stačí k nalezení optima $2m$ až $3m$ pivotovacích kroků.

Ve své době byl proto velikým překvapením výsledek Kleeho (čti Klího) a Mintyho, kteří zkonstruovali úlohu lineárního programování v rovnicovém tvaru s $3n$ proměnnými a $2n$ omezeními, pro niž simplexová metoda s původním Dantzigovým pivotovacím pravidlem a s určitou počáteční přípustnou bází potřebuje $2^n - 1$ pivotovacích kroků, tedy exponenciálně mnoho.

Množina přípustných řešení je šikovně deformovaná n-dimenzionální krychle, takzvaná *Klee-Mintyho krychle*, sestrojená tak, že simplexová metoda projde postupně všechny její vrcholy. Místo formálního popisu konstrukci ilustrujeme obrázkem pro dimenze 2 a 3:

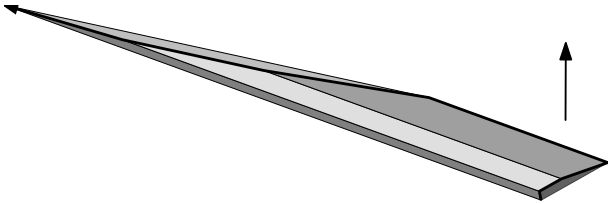


$n = 2$



$n = 3$

Pro lepší představu je Klee-Mintyho krychle vepsána do krychle obyčejné. Simplexová metoda se bude ubírat cestou vyznačenou tlustě. Přesněji řečeno, tyto konkrétní Klee-Mintyho krychle přinutí jít po vyznačené cestě jen některá z pivotovacích pravidel a například původní Dantzigovo pravidlo neošálí. Verze Klee-Mintyho krychle, která funguje pro Dantzigovo pravidlo, vypadá bizarněji:



Později byly podobné příklady s exponenciálně mnoha kroky objeveny i pro další pivotovací pravidla, kromě jiného pro všechna uvedená zde s výjimkou pravidla NÁHODNÁ HRANA. Mnoho lidí se pokoušelo navrhnout pi-

votovací pravidlo, pro které by se dalo dokázat, že počet kroků simplexové metody je při jeho použití vždycky omezen nějakou polynomiální funkcí m a n, ale nikomu se to zatím nepodařilo. Nejlepší známý výsledek v tomto směru je jisté pravděpodobnostní pivotovací pravidlo (jiné než NÁHODNÁ HRANA), pro něž je pro každou vstupní úlohu průměrný počet kroků omezen funkcí $e^{C\sqrt{n \ln n}}$, kde C je jistá nevelká konstanta. To je podstatně lepší než třeba 2^n, ale zase mnohem horší než polynomiální funkce.

Moc dobrý odhad není znám dokonce ani pro nejchytřejší možné pivotovací pravidlo, říkejme mu „vševědovo pravidlo“, které by vždycky vybralo vůbec nejkratší možnou posloupnost pivotovacích kroků vedoucí k optimálnímu řešení. Takzvaná *Hirschova domněnka*, jeden ze slavných otevřených matematických problémů, praví, že vševědovu pravidlu vždycky stačí řádově nejvýš n pivotovacích kroků, ale nejlepší známý výsledek v tom směru dává horní odhad jen $n^{1+\ln n}$. To je lepší než zmíněné $e^{C\sqrt{n \ln n}}$, ale horší než každá polynomiální funkce n, a opravdové pivotovací pravidlo to nedává, protože nikdo neví, jak vševědova rozhodnutí simulovat nějakým algoritmem.

Navzdory Klee-Mintyho krychli a podobným naschválným příkladům se simplexová metoda úspěšně používá. Pozoruhodné teoretické výsledky naznačují, že zmíněné naschválné příklady jsou opravdu vzácné. Ví se například, že vygenerujeme-li úlohu lineárního programování v rovnicovém tvaru vhodným (přesně definovaným) způsobem náhodně, potom počet pivotovacích kroků je s velkou pravděpodobností řádově nejvýš m^2. Jiný, nedávný výsledek praví, že začneme-li s libovolnou vstupní úlohou a potom její koeficienty o trochu pozměníme, a to náhodně, potom na výsledné úloze bude počet kroků simplexové metody s velkou pravděpodobností polynomiální (konkrétní odhad na ten polynom závisí na tom, jak specifikujeme ono „trochu“). Přesná formulace těchto dvou výsledků vyžaduje řadu technických pojmů, které zde nechceme zavádět, a tudíž si ji odpustíme.

5.8 Shrnutí

Pro přehlednost celou simplexovou metodu ještě znovu sepíšeme.

Algoritmus SIMPLEXOVÁ METODA

1. Převeď vstupní úlohu na rovnicový tvar

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} = \mathbf{b} \text{ a } \mathbf{x} \geq \mathbf{0}$$

   s n proměnnými a m rovnicemi, kde A má hodnost m.

2. Není-li k dispozici nějaká přípustná báze, zařiď $\mathbf{b} \geq \mathbf{0}$ a vyřeš simple-

xovou metodou pomocnou úlohu

$$\begin{array}{ll} \text{maximalizovat} & -(x_{n+1} + x_{n+2} + \cdots + x_{n+m}) \\ \text{za podmínek} & \bar{A}\,\bar{\mathbf{x}} = \mathbf{b} \\ & \bar{\mathbf{x}} \geq \mathbf{0}, \end{array}$$

kde x_{n+1} až x_{n+m} jsou pomocné proměnné, $\bar{\mathbf{x}} = (x_1, \ldots, x_{n+m})$ a $\bar{A} = (A \,|\, I_m)$. Vyjde-li maximum záporné, je původní úloha nepřípustná, **konec**. Jinak je prvních n složek optimálního řešení bázickým přípustným řešením původní úlohy (viz sekce 5.6).

3. Pro přípustnou bázi $B \subseteq \{1, 2, \ldots, n\}$ sestav simplexovou tabulku tvaru

$$\begin{array}{rcccl} \mathbf{x}_B & = & \mathbf{p} & + & Q\,\mathbf{x}_N \\ \hline z & = & z_0 & + & \mathbf{r}^T\mathbf{x}_N. \end{array}$$

4. Jestliže v simplexové tabulce platí $\mathbf{r} \leq \mathbf{0}$, vrať optimální řešení ($\mathbf{p}$ udává jeho bázické složky, nebázické jsou 0), **konec**.

5. Jinak vyber *vstupující proměnnou* x_v, jejíž koeficient ve vektoru $\mathbf{r}$ je kladný. Je-li víc možností, použij některého pivotovacího pravidla, viz sekce 5.7.

6. Jestliže je sloupec vstupující proměnné x_v v tabulce celý nezáporný, je úloha neomezená, **konec**.

7. Jinak vyber *vystupující proměnnou* x_u. Prober všechny řádky simplexové tabulky, v nichž je koeficient u x_v záporný, a v každém z nich vyděl složku vektoru $\mathbf{p}$ tím koeficientem a změň znaménko. Řádek vystupující proměnné x_u je takový, v němž je tento podíl minimální (je-li víc možností, rozhodni se podle pivotovacího pravidla, případně libovolně, když to pivotovací pravidlo neurčuje).

8. Dosavadní přípustnou bázi B nahraď novou přípustnou bází $(B \setminus \{u\}) \cup \{v\}$. Aktualizuj také simplexovou tabulku tak, aby odpovídala této nové bázi (viz sekce 5.6). Vrať se na krok 4.

6

Dualita lineárního programování

6.1 Věta o dualitě

Zde zformulujeme asi nejdůležitější teoretický výsledek o úlohách lineárního programování.

Podívejme se na úlohu

$$\begin{array}{lrcrcr} \text{maximalizovat} & 2x_1 & + & 3x_2 & & \\ \text{za podmínek} & 4x_1 & + & 8x_2 & \leq & 12 \\ & 2x_1 & + & x_2 & \leq & 3 \\ & 3x_1 & + & 2x_2 & \leq & 4 \\ & x_1, x_2 \geq 0. & & & & \end{array} \tag{6.1}$$

Aniž bychom hledali optimum, z první nerovnice a podmínek nezápornosti můžeme hned usoudit, že maximální hodnota účelové funkce není větší než 12, protože pro nezáporná x_1 a x_2 vždycky máme

$$2x_1 + 3x_2 \leq 4x_1 + 8x_2 \leq 12.$$

Lepší horní odhad dostaneme, když první nerovnici napřed vydělíme dvěma:

$$2x_1 + 3x_2 \leq 2x_1 + 4x_2 \leq 6.$$

A ještě lepší, když k první rovnici přičteme druhou a vydělíme třemi, což vede k nerovnosti

$$2x_1 + 3x_2 = \frac{1}{3}(4x_1 + 8x_2 + 2x_1 + x_2) \leq \frac{1}{3}(12 + 3) = 5,$$

tudíž účelová funkce nemůže být větší než 5.

Jak dobrý odhad můžeme takto dostat? A co znamená „takto“? Začněme druhou otázkou: Snažíme se z omezujících podmínek odvodit nerovnici tvaru

$$d_1x_1 + d_2x_2 \leq h,$$

kde $d_1 \geq 2$, $d_2 \geq 3$ a h je co nejmenší. Pak totiž můžeme prohlásit, že pro všechna $x_1, x_2 \geq 0$ platí

$$2x_1 + 3x_2 \leq d_1x_1 + d_2x_2 \leq h,$$

a tedy h je horní odhad na maximum účelové funkce. Jak můžeme takové nerovnice odvozovat? Tři nerovnice dané v úloze zkombinujeme s nějakými nezápornými koeficienty y_1, y_2, y_3 (nezápornost je potřeba proto, aby se neobrátil směr nerovnosti). Dostáváme

$$(4y_1 + 2y_2 + 3y_3)x_1 + (8y_1 + y_2 + 2y_3)x_2 \leq 12y_1 + 3y_2 + 4y_3,$$

a tudíž $d_1 = 4y_1 + 2y_2 + 3y_3$, $d_2 = 8y_1 + y_2 + 2y_3$ a $h = 12y_1 + 3y_2 + 4y_3$.

Jak volit koeficienty y_1, y_2, y_3 co nejlépe? Musíme mít $d_1 \geq 2$, $d_2 \geq 3$ a přitom dostat h co nejmenší. To je zase úloha lineárního programování:

$$\begin{array}{lrcrcrcl} \text{minimalizovat} & 12y_1 & + & 3y_2 & + & 4y_3 & & \\ \text{za podmínek} & 4y_1 & + & 2y_2 & + & 3y_3 & \geq & 2 \\ & 8y_1 & + & y_2 & + & 2y_3 & \geq & 3 \\ & y_1, y_2, y_3 \geq 0. & & & & & & \end{array}$$

Jmenuje se *duální úloha* k úloze (6.1), s níž jsme začali. Duální úloha zde „hlídá“ původní úlohu shora, v tom smyslu, že každé přípustné řešení (y_1, y_2, y_3) duální úlohy dává jistý horní odhad na maximum účelové funkce v (6.1).

Jak dobře hlídá? Dokonale! Optimální řešení duální úlohy je $\mathbf{y} = (\frac{5}{16}, 0, \frac{1}{4})$ s hodnotou účelové funkce 4,75, a to je i optimální hodnota úlohy (6.1), které se nabývá pro $\mathbf{x} = (\frac{1}{2}, \frac{5}{4})$.

Obsahem věty o dualitě je, že duální úloha *vždycky* hlídá dokonale. Zopakujme výše uvedené úvahy obecně pro úlohu tvaru

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\,\mathbf{x} \leq \mathbf{b} \text{ a } \mathbf{x} \geq \mathbf{0}, \qquad \text{(P)}$$

kde A je matice s m řádky a n sloupci. Snažíme se zkombinovat m nerovnic soustavy $A\mathbf{x} \leq \mathbf{b}$ pomocí nezáporných koeficientů $y_1, y_2, \ldots, y_m$ tak, aby výsledná nerovnice měla j-tý koeficient aspoň c_j, $j = 1, 2, \ldots, n$, a přitom aby pravá strana byla co nejmenší. To vede k **duální úloze**

$$\text{minimalizovat } \mathbf{b}^T\mathbf{y} \text{ za podmínek } A^T\mathbf{y} \geq \mathbf{c} \text{ a } \mathbf{y} \geq \mathbf{0}, \qquad \text{(D)}$$

kdo nevěří, může si to rozepsat do složek. V této souvislosti se o původní úloze (P) mluví jako o **primární úloze**.

Ze způsobu, jak jsme úlohu (D) vyrobili, plyne:

6.1.1 Tvrzení. *Každé přípustné řešení* $\mathbf{y}$ *duální úlohy* (D) *dává horní odhad na maximum účelové funkce v úloze* (P). *Jinak řečeno, pro každé přípustné řešení* $\mathbf{x}$ *úlohy* (P) *a každé přípustné řešení* $\mathbf{y}$ *úlohy* (D) *platí*

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}.$$

Speciálně, je-li (P) *neomezená, musí* (D) *být nepřípustná, a je-li* (D) *neomezená, pak je nepřípustná* (P).

Tomuto tvrzení se zpravidla říká *slabá věta o dualitě*, slabá proto, že vyjadřuje jen ono hlídání úlohy (P) úlohou (D), a nemluví o jeho dokonalosti. Tu vyjadřuje až věta o dualitě bez přívlastků (někdy též zvaná *silná věta o dualitě*).

Věta o dualitě lineárního programování

Pro úlohy

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} \leq \mathbf{b} \text{ a } \mathbf{x} \geq \mathbf{0} \qquad \text{(P)}$$

a

$$\text{minimalizovat } \mathbf{b}^T\mathbf{y} \text{ za podmínek } A^T\mathbf{y} \geq \mathbf{c} \text{ a } \mathbf{y} \geq \mathbf{0} \qquad \text{(D)}$$

nastane právě jedna z následujících možností:

1. Ani (P), ani (D) nemá přípustné řešení.
2. (P) je neomezená a (D) nemá přípustné řešení.
3. (P) nemá přípustné řešení a (D) je neomezená.
4. Jak (P), tak (D) mají přípustné řešení. Pak existuje optimální řešení $\mathbf{x}^*$ úlohy (P) a optimální řešení $\mathbf{y}^*$ úlohy (D) a platí

   $$\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*.$$

   To jest, *maximum (P) je rovno minimu (D).*

Na první pohled může věta o dualitě vypadat složitě. Pro její lepší pochopení může být užitečné začít speciálním případem, zvaným Farkasovo

lemma, které probereme v části 6.4. Tato jednodušší věta má několik poměrně názorných interpretací a obsahuje v sobě to nejdůležitější z věty o dualitě.

Dokazování věty o dualitě, jímž se budeme zabývat v sekcích 6.3 a 6.4, dá trochu práce, na rozdíl od slabé věty o dualitě, která je při správném přístupu snadná.

Hlavní tvrzení věty o dualitě spočívá v rovnosti $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$ pro čtvrtou z možných situací, tj. když jak (P) tak (D) jsou přípustné.

Poněvadž úloha lineárního programování může být buď přípustná a omezená, nebo přípustná a neomezená, nebo nepřípustná, máme 3 možnosti pro (P) a 3 možnosti pro (D). To na první pohled dává 9 možných kombinací pro (P) a (D). Tři možnosti, „(P) neomezená a (D) přípustná omezená“, „(P) neomezená a (D) neomezená“ a „(P) přípustná omezená a (D) neomezená“, jsou vyloučeny slabou větou o dualitě. V důkazu (silné) věty o dualitě vyloučíme možnosti „(P) přípustná omezená a (D) nepřípustná“ a „(P) nepřípustná a (D) přípustná omezená“. Zbývají tak čtyři případy uvedené ve větě o dualitě, a všechny čtyři se mohou skutečně vyskytnout.

Ještě jednou přípustnost versus optimalita. V kapitole 1 jsme poznamenali, že nalezení přípustného řešení obecné úlohy lineárního programování je výpočetně stejně obtížné, jako nalezení optima. Tam jsme to stručně zdůvodnili pomocí binárního vyhledávání. Věta o dualitě poskytuje mnohem elegantnější argument. Úloha (P) má optimální řešení, právě když má přípustné řešení úloha, vzniklá kombinací omezujících podmínek z (P), omezujících podmínek z (D) a nerovnosti mezi účelovými funkcemi:

$$\begin{array}{ll} \text{maximalizovat} & \mathbf{c}^T\mathbf{x} \\ \text{za podmínek} & A\mathbf{x} \leq \mathbf{b} \\ & A^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y} \\ & \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0} \end{array}$$

(na účelové funkci nezáleží a proměnné jsou $x_1, \ldots, x_n, y_1, \ldots, y_m$). Navíc pro každé přípustné řešení $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ je $\tilde{\mathbf{x}}$ optimálním řešením úlohy (P).

6.2 Dualizace pro každého

Věta o dualitě platí pro každou úlohu lineárního programování, jenom se musí k dané úloze správně vytvořit úloha duální. To se dá dělat tak, že se daná úloha nejdřív převede na speciální tvar (P) pomocí triků zmíněných v sekcích 1.1 a 4.1, a pak je duální úloha tvaru (D). Ten můžeme mnohdy ještě zjednodušit, například lze nahradit rozdíl dvou nezáporných proměnných jednou proměnnou neomezenou.

Jednodušší, než to dělat vždy znovu, je řídit se následujícím receptem (jehož správnost se dokazuje právě popsaným postupem). Řekněme, že primární úloha má proměnné $x_1, x_2, \ldots, x_n$, z nichž některé jsou nezáporné, některé nekladné a některé neomezené. Omezující podmínky nechť jsou $P_1, P_2, \ldots, P_m$, kde P_i je tvaru

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \left\{ \begin{array}{l} \leq \\ \geq \\ = \end{array} \right\} b_i.$$

Má se *maximalizovat* $\mathbf{c}^T\mathbf{x}$.

Potom duální úloha má proměnné $y_1, y_2, \ldots, y_m$, kde y_i odpovídá omezující podmínce P_i a splňuje

$$\left\{ \begin{array}{l} y_i \geq 0 \\ y_i \leq 0 \\ y_i \in \mathbb{R} \end{array} \right\} \text{ pokud v podmínce } P_i \text{ je } \left\{ \begin{array}{l} \leq \\ \geq \\ = \end{array} \right\}.$$

Omezující podmínky duální úlohy jsou $Q_1, Q_2, \ldots, Q_n$, kde Q_j odpovídá proměnné x_j a zní

$$a_{1j}y_1 + a_{2j}y_2 + \cdots + a_{mj}y_m \left\{ \begin{array}{l} \geq \\ \leq \\ = \end{array} \right\} c_j \text{ pokud } x_j \text{ splňuje } \left\{ \begin{array}{l} x_j \geq 0 \\ x_j \leq 0 \\ x_j \in \mathbb{R} \end{array} \right\}.$$

Účelová funkce je $\mathbf{b}^T\mathbf{y}$ a má se *minimalizovat*.

Všimněte si, že v první části receptu (od primárních omezení k duálním proměnným) se nerovnosti obracejí, zatímco ve druhé části (od primárních proměnných k duálním omezením) je směr zachován.

Recept na dualizaci

	Primární úloha	Duální úloha
proměnné	$x_1, x_2, \ldots, x_n$	$y_1, y_2, \ldots, y_m$
matice	A	A^T
pravé strany	$\mathbf{b}$	$\mathbf{c}$
účelová funkce	$\max \mathbf{c}^T\mathbf{x}$	$\min \mathbf{b}^T\mathbf{y}$
nerovnosti	$x_j \geq 0$ $x_j \leq 0$ $x_j \in \mathbb{R}$	j-té omezení má $\geq$ $\leq$ $=$
	i-té omezení má $\geq$ $\leq$ $=$	$y_i \leq 0$ $y_i \geq 0$ $y_i \in \mathbb{R}$

Chceme-li dualizovat *minimalizační* úlohu, můžeme napřed přejít k úloze maximalizační změnou znaménka účelové funkce a pak pracovat podle receptu.

Tak se také dá zjistit, že pravidla fungují symetricky „tam“ a „pozpátku“. Tím míníme, že vyjdeme-li z nějaké úlohy lineárního programování, zkonstruujeme úlohu duální, a k ní zase úlohu duální, dostaneme zpátky původní (primární) úlohu. Speciálně, úlohy (P) a (D) v naší formulaci věty o dualitě jsou *navzájem duální*. Takže jiná možnost, jak dualizovat minimalizační úlohu, je považovat ji za úlohu duální a použít recept v obráceném směru.

Fyzikální interpretace duality. Mějme úlohu

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} \leq \mathbf{b}.$$

Podle dualizovacího receptu zní duální úloha

$$\text{minimalizovat } \mathbf{b}^T\mathbf{y} \text{ za podmínek } A^T\mathbf{y} = \mathbf{c} \text{ a } \mathbf{y} \geq \mathbf{0}.$$

Předpokládejme, že primární úloha je přípustná a omezená a nechť $n = 3$. Na $\mathbf{x}$ se díváme jako na bod ve třídimenzionálním prostoru a $\mathbf{c}$ interpretujme jako vektor gravitace, směřuje tedy dolů.

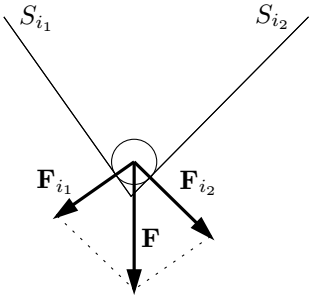
Každá z nerovnic soustavy $A\mathbf{x} \leq \mathbf{b}$ určuje poloprostor. Průnik oněch poloprostorů je neprázdný konvexní mnohostěn omezený zdola. Každá jeho (dvoudimenzionální) stěna je dána některou z rovnic $\mathbf{a}_i^T\mathbf{x} = b_i$, kde vektory $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m$ jsou řádky matice A, interpretované ovšem jako sloupcové vek-

tory (přitom ovšem ne každé nerovnici soustavy $A\mathbf{x} \leq \mathbf{b}$ musí odpovídat stěna). Označme takovou stěnu S_i.

Představme si, že hranice mnohostěnu je z tvrdého papíru a že někde uvnitř mnohostěnu upustíme malinkou kuličku. Ta spadne a skutálí se do nejdolejšího vrcholu (nebo možná zůstane na vodorovné hraně či stěně). Výslednou polohu kuličky označme $\mathbf{x}^*$, čili $\mathbf{x}^*$ je optimální řešení původní úlohy. V této stabilní poloze se kulička dotýká několika stěn, typicky 3. Buď D množina takových i, že kulička se dotýká stěny S_i. Pro $i \in D$ tedy máme

$$\mathbf{a}_i^T \mathbf{x}^* = b_i. \tag{6.6}$$

Na kuličku působí gravitace silou $\mathbf{F}$, úměrnou vektoru $\mathbf{c}$. Gravitační síla se rozkládá na síly, jimiž kulička působí na stěny, kterých se dotýká. Síla $\mathbf{F}_i$, kterou kulička působí na stěnu S_i, je kolmá k S_i a směřuje ven z mnohostěnu (zanedbáváme-li tření), viz schématický dvoudimenzionální obrázek:



Síly působící na kuličku jsou v rovnováze, a proto $\mathbf{F} = \sum_{i \in D} \mathbf{F}_i$. Normála stěny S_i směrem ven z mnohostěnu je $\mathbf{a}_i$, čili $\mathbf{F}_i$ je úměrné $\mathbf{a}_i$, a tedy pro nějaká nezáporná čísla y_i^* dostáváme

$$\sum_{i \in D} y_i^* \mathbf{a}_i = \mathbf{c}.$$

Definujeme-li $y_i^* = 0$ pro $i \notin D$, můžeme psát $\sum_{i=1}^m y_i^* \mathbf{a}_i = \mathbf{c}$, v maticovém zápisu $A^T \mathbf{y}^* = \mathbf{c}$. Tedy $\mathbf{y}^*$ je přípustné řešení duální úlohy.

Podívejme se na součin $(\mathbf{y}^*)^T (A\mathbf{x}^* - \mathbf{b})$. Pro $i \notin D$ je i-tá složka $\mathbf{y}^*$ rovna 0, a pro $i \in D$ je nulová i-tá složka $A\mathbf{x}^* - \mathbf{b}$ podle (6.6). Takže součin je 0 a odtud $(\mathbf{y}^*)^T \mathbf{b} = (\mathbf{y}^*)^T A\mathbf{x}^* = \mathbf{c}^T \mathbf{x}^*$.

Vidíme, že $\mathbf{x}^*$ je optimální řešení primární úlohy, $\mathbf{y}^*$ je přípustné řešení duální úlohy a platí $\mathbf{c}^T \mathbf{x}^* = \mathbf{b}^T \mathbf{y}^*$. Podle slabé věty o dualitě je $\mathbf{y}^*$ též optimální řešení duální úlohy a máme situaci přesně jako ve větě o dualitě. Speciální třídimenzionální případ věty o dualitě jsme právě „fyzikálně nahlédli“.

Poznamenejme, že duální úloha má také interpretaci ekonomickou. Duálním proměnným se v ní říká *stínové ceny* a zájemce ji najde například v Chvátalově učebnici citované v kapitole 8.

6.3 Důkaz věty o dualitě ze simplexové metody

Věta o dualitě lineárního programování se dá poměrně rychle odvodit ze správnosti simplexové metody: ze závěrečné tabulky jde totiž dostat nejen optimální řešení původní úlohy, ale i optimální řešení úlohy duální.

Pro nás to má jeden háček – sice jsme popsali, jak se dá v simplexové metodě zabránit zacyklení (Blandovým pravidlem nebo symbolickými perturbacemi), ale pro ani jeden z těchto přístupů jsme konečnost simplexové metody nedokázali a nebudeme to dělat ani tady. Tento oddíl tedy dokazuje větu o dualitě pro ty, kdo uvěřili, že zacyklení simplexové metody lze vždycky vyloučit (některý z důkazů obsahuje téměř každá rozsáhlejší učebnice lineárního programování).

Přesněji řečeno, dokážeme toto:

> *Je-li primární úloha (P) přípustná a omezená, pak duální úloha (D) je také přípustná a omezená a má stejnou optimální hodnotu účelové funkce jako (P).*

Poněvadž (P) je duální úlohou k úloze (D), můžeme v uvedeném tvrzení role (P) a (D) zaměnit. To spolu s úvahami uvedenými za větou o dualitě (o možných kombinacích, které mohou pro (P) a (D) nastat) větu dokazuje.

Uvažme primární úlohu

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínek } A\mathbf{x} \leq \mathbf{b} \text{ a } \mathbf{x} \geq \mathbf{0}. \qquad \text{(P)}$$

Převodem na rovnicový tvar zavedením pomocných proměnných x_{n+1} až x_{n+m} vznikne úloha

$$\text{maximalizovat } \bar{\mathbf{c}}^T\bar{\mathbf{x}} \text{ za podmínek } \bar{A}\bar{\mathbf{x}} = \mathbf{b} \text{ a } \bar{\mathbf{x}} \geq \mathbf{0},$$

kde $\bar{\mathbf{x}} = (x_1, \ldots, x_{n+m})$, $\bar{\mathbf{c}} = (c_1, \ldots, c_n, 0, \ldots, 0)$ a $\bar{A} = (A \,|\, I_m)$. Je-li posledně uvedená úloha přípustná a omezená, simplexová metoda najde nějaké optimální řešení $\bar{\mathbf{x}}^*$, které odpovídá nějaké přípustné bázi B. Prvních n složek vektoru $\bar{\mathbf{x}}^*$ tvoří optimální řešení $\mathbf{x}^*$ úlohy (P). Podle kriteria optimality platí v závěrečné simplexové tabulce $\mathcal{T}(B)$ nerovnost $\mathbf{r} \leq \mathbf{0}$. Tvrzení, které se snažíme dokázat, pak už ihned plyne z následujícího lemmatu.

6.3.1 Lemma. *V popsané situaci je vektor* $\mathbf{y}^* = (\bar{\mathbf{c}}_B^T\bar{A}_B^{-1})^T$ *přípustným řešením duální úlohy (D) a splňuje rovnost* $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$ *jako ve větě o dualitě.*

Důkaz. Podle lemmatu 5.5.1 je $\bar{\mathbf{x}}^*$ dáno vztahy $\bar{\mathbf{x}}_B^* = \bar{A}_B^{-1}\mathbf{b}$ a $\bar{\mathbf{x}}_N^* = \mathbf{0}$, takže

$$\mathbf{c}^T\mathbf{x}^* = \bar{\mathbf{c}}^T\bar{\mathbf{x}}^* = \bar{\mathbf{c}}_B^T\bar{\mathbf{x}}_B^* = \bar{\mathbf{c}}_B^T(\bar{A}_B^{-1}\mathbf{b}) = (\bar{\mathbf{c}}_B^T\bar{A}_B^{-1})\mathbf{b} = (\mathbf{y}^*)^T\mathbf{b} = \mathbf{b}^T\mathbf{y}^*.$$

Rovnost $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$ platí a zbývá ověřit přípustnost $\mathbf{y}^*$, neboli $A^T\mathbf{y}^* \geq \mathbf{c}$ a $\mathbf{y}^* \geq \mathbf{0}$.

Podmínka $\mathbf{y}^* \geq \mathbf{0}$ se dá přepsat jako $I_m\mathbf{y}^* \geq \mathbf{0}$, a z toho je vidět, že obě podmínky dohromady jsou ekvivalentní podmínce

$$\bar{A}^T\mathbf{y}^* \geq \bar{\mathbf{c}}. \tag{6.8}$$

Levá strana po dosazení za $\mathbf{y}^*$ dává $\bar{A}^T(\bar{\mathbf{c}}_B^T\bar{A}_B^{-1})^T = (\bar{\mathbf{c}}_B^T\bar{A}_B^{-1}\bar{A})^T$. Označme tento $(n+m)$-složkový vektor $\mathbf{w}$. Pro jeho bázické složky máme

$$\mathbf{w}_B = (\bar{\mathbf{c}}_B^T\bar{A}_B^{-1}\bar{A}_B)^T = (\bar{\mathbf{c}}_B^T I_m)^T = \bar{\mathbf{c}}_B,$$

takže pro bázické složky platí v (6.8) dokonce rovnost. Pro nebázické složky máme

$$\mathbf{w}_N = (\bar{\mathbf{c}}_B^T\bar{A}_B^{-1}\bar{A}_N)^T = \bar{\mathbf{c}}_N - \mathbf{r} \geq \bar{\mathbf{c}}_N,$$

protože $\mathbf{r} = \bar{\mathbf{c}}^N - (\bar{\mathbf{c}}_B^T\bar{A}_B^{-1}\bar{A}_N)^T$ podle lemmatu 5.5.1 a $\mathbf{r} \leq \mathbf{0}$ podle kritéria optimality. Takovéto kouzlení se simplexovou metodou dokazuje lemma a tím i větu o dualitě. □

6.4 Důkaz věty o dualitě z Farkasova lemmatu

Jiný přístup k větě o dualitě lineárního programování je dokázat napřed zjednodušenou verzi, zvanou *Farkasovo lemma* (vyslovuje se Farkašovo), a potom větu odvodit dosazením šikovně poskládané matice do onoho lemmatu. Pěkné na tom je, že Farkasovo lemma má názornou geometrickou interpretaci.

6.4.1 Tvrzení (Farkasovo lemma). *Buď A libovolná reálná matice s m řádky a n sloupci a buď $\mathbf{b} \in \mathbb{R}^m$ libovolný vektor. Pak nastane právě jedna z následujících dvou možností:*

(F1) *Soustava $A\mathbf{x} = \mathbf{b}$ má nezáporné řešení.*

(F2) *Existuje vektor $\mathbf{y} \in \mathbb{R}^m$ takový, že $\mathbf{y}^T A \geq \mathbf{0}^T$ a přitom $\mathbf{y}^T\mathbf{b} < 0$.*

Je snadno vidět, že nemohou nastat obě možnosti zároveň. Vektor $\mathbf{y}$ jako v (F2) totiž určuje lineární kombinaci rovnic dosvědčující, že $A\mathbf{x} = \mathbf{b}$ žádné nezáporné řešení mít nemůže: všechny koeficienty na levé straně této lineární kombinace jsou nezáporné, a přitom pravá strana je záporná. Například pro soustavu

$$\begin{array}{rcrcr} x_1 & - & 2x_2 & = & 3 \\ -x_1 & + & 4x_2 & = & -5 \end{array}$$

můžeme vzít $\mathbf{y} = (3, 2)$. Sečteme-li trojnásobek první rovnice a dvojnásobek druhé, dostaneme rovnici $x_1 + 2x_2 = -1$, která nezápornými čísly zjevně splnitelná není.

Abychom mohli na Farkasovo lemma nahlížet geometricky, potřebujeme pojem konvexního obalu (viz sekce 4.3). Dále pro vektory $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n \in \mathbb{R}^m$ definujeme **konvexní kužel** jimi generovaný jako množinu všech jejich nezáporných lineárních kombinací, tj.

$$\Big\{t_1\mathbf{a}_1 + t_2\mathbf{a}_2 + \cdots + t_n\mathbf{a}_n : t_1, t_2, \ldots, t_n \geq 0\Big\}.$$

Jinými slovy, tento konvexní kužel je konvexní obal polopřímek $p_1, p_2, \ldots, p_n$, kde $p_i = \{t\mathbf{a}_i : t \geq 0\}$ vychází z počátku a prochází bodem $\mathbf{a}_i$.

6.4.2 Tvrzení (Farkasovo lemma geometricky). *Mějme libovolné vektory $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n, \mathbf{b} \in \mathbb{R}^m$. Pak nastane právě jedna z následujících dvou možností:*

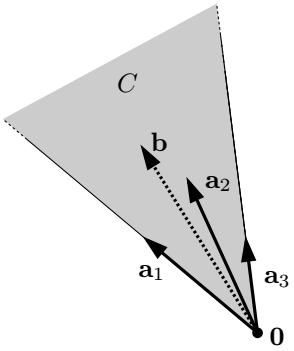
(F1′) *Bod $\mathbf{b}$ leží v konvexním kuželu C generovaném $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$.*

(F2′) *Existuje nadrovina h procházející bodem $\mathbf{0}$, tvaru*

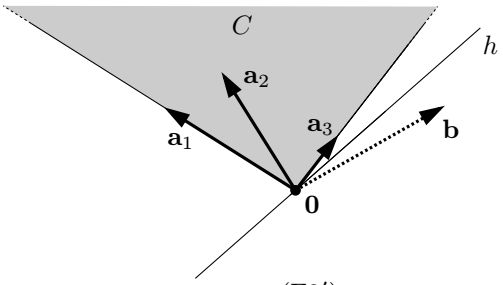
$$h = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{y}^T\mathbf{x} = 0\}$$

pro nějaké $\mathbf{y} \in \mathbb{R}^m$, taková že všechny vektory $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ (a tedy i celý kužel C) leží na jedné její straně, tj. $\mathbf{y}^T\mathbf{a}_i \geq 0$ pro všechna $i = 1, 2, \ldots, n$, a $\mathbf{b}$ leží na straně druhé (ostře), tj. $\mathbf{y}^T\mathbf{b} < 0$.

Obrázek ilustruje obě možnosti pro $m = 2$ a $n = 3$:



(F1′)



(F2′)

Abychom viděli, že obě verze znamenají totéž, stačí vzít za $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ sloupce matice A. Existenci nezáporného řešení soustavy $A\mathbf{x} = \mathbf{b}$ můžeme přepsat jako $\mathbf{b} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \cdots + x_n\mathbf{a}_n,\ x_1, x_2, \ldots, x_n \geq 0$, a to je přesně totéž jako $\mathbf{b} \in C$. To, že (F2) a (F2′) znamenají totéž, snad už další vysvětlení nepotřebuje.

Důkazů Farkasova lemmatu je známo řada typů. Jeden z nich, využívající elementárních prostředků matematické analýzy a svým způsobem nejpřirozenější, ukážeme v následujícím oddílu.

K důkazu věty o dualitě si napřed pořídíme vhodnější variantu Farkasova lemmatu pro soustavu nerovnic, trikem natrénovaným z převodu na rovnicový tvar.

6.4.3 Lemma (Farkasovo lemma, varianta). *Soustava nerovnic* $A\mathbf{x} \leq \mathbf{b}$ *má nezáporné řešení* $\mathbf{x}$*, právě když každé* nezáporné $\mathbf{y}$*, pro něž* $\mathbf{y}^T A \geq \mathbf{0}^T$*, splňuje také* $\mathbf{y}^T\mathbf{b} \geq \mathbf{0}$.

Důkaz. Je-li dána matice A typu $m \times n$, utvořme matici $\bar{A} = (A \,|\, I_m)$. Potom $A\mathbf{x} \leq \mathbf{b}$ má nezáporné řešení, právě když $\bar{A}\,\bar{\mathbf{x}} = \mathbf{b}$ má nezáporné řešení. Posledně jmenované je podle Farkasova lemmatu (tvrzení 6.4.1) ekvivalentní podmínce, že každé $\mathbf{y}$, pro něž $\mathbf{y}^T\bar{A} \geq \mathbf{0}^T$, splňuje také $\mathbf{y}^T\mathbf{b} \geq 0$. A konečně $\mathbf{y}^T\bar{A} \geq \mathbf{0}^T$ říká přesně totéž jako $\mathbf{y}^T A \geq \mathbf{0}^T$ a $\mathbf{y} \geq \mathbf{0}$, takže dohromady máme požadovanou ekvivalenci. □

Důkaz věty o dualitě. Předpokládejme, že primární úloha (P) má optimální řešení $\mathbf{x}^*$. Dokážeme, že (D) má optimální řešení, jehož hodnota se rovná optimální hodnotě pro (P).

Buď $\gamma = \mathbf{c}^T\mathbf{x}^*$ optimální hodnota (P). Tudíž systém nerovnic

$$A\mathbf{x} \leq \mathbf{b},\ \mathbf{c}^T\mathbf{x} \geq \gamma \tag{6.9}$$

má nezáporné řešení, kdežto pro libovolné $\varepsilon > 0$ systém

$$A\mathbf{x} \leq \mathbf{b},\ \mathbf{c}^T\mathbf{x} \geq \gamma + \varepsilon \tag{6.10}$$

žádné nezáporné řešení nemá. Definujeme $(m+1)\times n$ matici $\hat{A}$ a vektor $\hat{\mathbf{b}}_\varepsilon \in \mathbb{R}^m$:

$$\hat{A} = \begin{pmatrix} A \\ -\mathbf{c}^T \end{pmatrix}, \quad \hat{\mathbf{b}} = \begin{pmatrix} \mathbf{b} \\ -\gamma - \varepsilon \end{pmatrix}.$$

Pak (6.9) je ekvivalentní $\hat{A}\mathbf{x} \leq \hat{\mathbf{b}}_0$ a (6.10) je ekvivalentní $\hat{A}\mathbf{x} \leq \hat{\mathbf{b}}_\varepsilon$.

Použijeme lemma 6.4.3. Pro $\varepsilon > 0$ systém $\hat{A}\mathbf{x} \leq \hat{\mathbf{b}}_\varepsilon$ nezáporné řešení nemá, takže existuje nezáporný vektor $\hat{\mathbf{y}} = (\mathbf{u}, z) \in \mathbb{R}^{m+1}$, pro nějž $\hat{\mathbf{y}}^T\hat{A} \geq \mathbf{0}^T$, ale $\hat{\mathbf{y}}^T\hat{\mathbf{b}}_\varepsilon < 0$. Tyto podmínky můžeme přepsat na

$$A^T\mathbf{u} \geq z\mathbf{c}^T,\ \mathbf{b}^T\mathbf{u} < z(\gamma + \varepsilon). \tag{6.11}$$

Použitím Farkasova lemmatu na případ $\varepsilon = 0$ (kdy systém má nezáporné řešení) vidíme, že právě zavedený vektor $\hat{\mathbf{y}}$ musí splňovat $\hat{\mathbf{y}}^T \hat{\mathbf{b}}_0 \geq 0$, což je totéž jako

$$\mathbf{b}^T \mathbf{u} \geq z\gamma.$$

Musí platit $z > 0$, protože $z = 0$ by protiřečilo ostré nerovnosti v (6.11). Pak můžeme položit $\mathbf{v} := \frac{1}{z}\,\mathbf{u} \geq \mathbf{0}$, a z (6.11) dostaneme

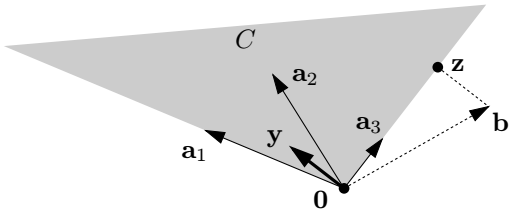
$$A^T \mathbf{v} \geq \mathbf{c}^T,\ \mathbf{b}^T \mathbf{v} < \gamma + \varepsilon.$$

To znamená, že $\mathbf{v}$ je přípustné řešení duální úlohy (D) s hodnotou účelové funkce menší než $\gamma+\varepsilon$. Tuto úvahu můžeme udělat pro libovolně malé $\varepsilon > 0$, a proto optimální hodnota (D) musí být přesně γ. Podle věty 4.2.3 se tato hodnota nabývá pro nějaké přípustné řešení $\mathbf{y}^*$. Důkaz věty o dualitě je hotov. □

6.5 Důkaz Farkasova lemmatu

V tomto oddílu dokážeme geometrickou verzi Farkasova lemmatu (tvrzení 6.4.2) pomocí elementární geometrie a analýzy. Jsou dány vektory $\mathbf{a}_1, \ldots, \mathbf{a}_n$ v $\mathbb{R}^m$. Nechť C je jimi generovaný konvexní kužel, tj. množina všech jejich lineárních kombinací s nezápornými koeficienty. Dokázat Farkasovo lemma znamená ukázat, že pro každý vektor $\mathbf{b} \notin C$ existuje nadrovina oddělující tento vektor od C, která prochází počátkem $\mathbf{0}$. Jinak řečeno, potřebujeme nalézt vektor $\mathbf{y} \in \mathbb{R}^m$, pro který platí $\mathbf{y}^T\mathbf{b} < 0$ a $\mathbf{y}^T\mathbf{x} \geq 0$ pro všechna $\mathbf{x} \in C$.

Základní schéma důkazu je jednoduché. Zvolíme $\mathbf{z}$ jako bod z C ležící nejblíž k $\mathbf{b}$ (měříme euklidovskou vzdálenost) a ověříme, že vektor $\mathbf{y} = \mathbf{z} - \mathbf{b}$ vyhovuje našim požadavkům, viz obrázek:



Hlavní technickou částí důkazu je ukázat, že takový nejbližší bod $\mathbf{z}$ opravdu existuje. V principu by se skutečně mohlo stát, že žádný bod není nejbližší (taková situace nastává například pro bod 0 na reálné ose a otevřený interval $(1, 2)$, protože interval obsahuje body, jejichž vzdálenost k 0 je libovolně blízká 1, ale žádný bod se vzdáleností přesně 1).

6.5.1 Lemma. *Nechť C je konvexní kužel v $\mathbb{R}^m$ generovaný konečně mnoha vektory $\mathbf{a}_1, \ldots, \mathbf{a}_n$ a nechť $\mathbf{b} \notin C$ je bod. Potom existuje bod $\mathbf{z} \in C$, který je k $\mathbf{b}$ nejbližší (takový bod je právě jeden, ale to nebudeme potřebovat).*

Důkaz tvrzení 6.4.2 z lemmatu 6.5.1. Jak jsme ohlásili, definujeme $\mathbf{y} = \mathbf{z} - \mathbf{b}$, kde $\mathbf{z}$ je bod C ležící nejblíž k $\mathbf{b}$. Nejdřív je potřeba ověřit, že $\mathbf{y}^T\mathbf{z} = 0$. Pokud $\mathbf{z} = \mathbf{0}$, je to jasné. Pokud $\mathbf{z} \neq \mathbf{0}$, tak můžeme za předpokladu, že $\mathbf{z}$ není kolmý na $\mathbf{y}$, maličko posunout $\mathbf{z}$ podél polopřímky $\{t\mathbf{z} : t \geq 0\} \subseteq C$ a dostat se ještě blíž k $\mathbf{b}$. Formálněji, předpokládejme, že $\mathbf{y}^T\mathbf{z} > 0$ a definujme $\mathbf{z}' = (1-\alpha)\mathbf{z}$ pro nějaké malé $\alpha > 0$. Spočítáme, že $\|\mathbf{z}' - \mathbf{b}\|^2 = (\mathbf{y} - \alpha\mathbf{z})^T(\mathbf{y} - \alpha\mathbf{z}) = \|\mathbf{y}\|^2 - 2\alpha\mathbf{y}^T\mathbf{z} + \alpha^2\|\mathbf{z}\|^2$. Pro všechna dostatečně malá $\alpha > 0$ platí $2\alpha\mathbf{y}^T\mathbf{z} > \alpha^2\|\mathbf{z}\|^2$, a tudíž i $\|\mathbf{z}' - \mathbf{b}\|^2 < \|\mathbf{y}\|^2 = \|\mathbf{z} - \mathbf{b}\|^2$. To je ale ve sporu s předpokladem, že $\mathbf{z}$ je nejbližší bod. Případ $\mathbf{y}^T\mathbf{z} < 0$ se vyřeší podobně. Tedy vskutku $\mathbf{y}^T\mathbf{z} = 0$.

Abychom ověřili $\mathbf{y}^T\mathbf{b} < 0$, spočítáme $0 < \mathbf{y}^T\mathbf{y} = \mathbf{y}^T\mathbf{z} - \mathbf{y}^T\mathbf{b} = -\mathbf{y}^T\mathbf{b}$.

Dále vezměme $\mathbf{x} \in C$, $\mathbf{x} \neq \mathbf{z}$. Úhel $\angle\mathbf{bzx}$ musí mít velikost aspoň $90°$, protože jinak by body dostatečně blízké k $\mathbf{z}$ na úsečce $\mathbf{zx}$ ležely k $\mathbf{b}$ blíž než $\mathbf{z}$, čili $(\mathbf{b}-\mathbf{z})^T(\mathbf{x}-\mathbf{z}) \leq 0$ (úvaha je podobná té, jíž jsme výše ověřovali $\mathbf{y}^T\mathbf{z} = 0$ a formální ověření ponecháme čtenáři). Tudíž $0 \geq (\mathbf{b}-\mathbf{z})^T(\mathbf{x}-\mathbf{z}) = -\mathbf{y}^T\mathbf{x} + \mathbf{y}^T\mathbf{z} = -\mathbf{y}^T\mathbf{x}$ a Farkasovo lemma je dokázáno. □

Zbývá dokázat lemma 6.5.1. Důkaz rozdělíme do několika kroků, které jsou zajímavými tvrzeníčky samy o sobě.

6.5.2 Lemma. *Nechť $X \subseteq \mathbb{R}^m$ je neprázdná uzavřená množina a $\mathbf{b} \in \mathbb{R}^m$ je bod. Potom v X existuje (alespoň jeden) bod s nejmenší vzdáleností k $\mathbf{b}$.*

Důkaz. To je jednoduché, ale vyžaduje to základní poznatky o kompaktních množinách v $\mathbb{R}^d$. Zvolíme libovolné $\mathbf{x}_0 \in X$, označíme $r = \|\mathbf{x}_0 - \mathbf{b}\|$ a nechť $K = \{\mathbf{x} \in X : \|\mathbf{x} - \mathbf{b}\| \leq r\}$. Zjevně pokud existuje nejbližší bod k $\mathbf{b}$ v K, je tento bod i nejbližším bodem k $\mathbf{b}$ v celém X. Vzhledem k tomu, že K je průnikem X s uzavřenou koulí o poloměru r, je to množina uzavřená a omezená, tedy kompaktní. Definujeme funkci $f\colon K \to \mathbb{R}$ předpisem $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{b}\|$. Potom je f spojitá funkce na kompaktní množině a každá taková funkce nabývá minima, neboli existuje $\mathbf{z} \in K$ takové, že $f(\mathbf{z}) \leq f(\mathbf{x})$ pro všechna $\mathbf{x} \in K$. Takový bod $\mathbf{z}$ je nejbližším bodem K k $\mathbf{b}$. □

Takže zbývá dokázat:

6.5.3 Lemma. *Každý konečně generovaný konvexní kužel je uzavřený.*

Toto lemma není tak zřejmé, jak by se mohlo na první pohled zdát. Jako výstražný příklad uvažme uzavřený disk D v rovině, na jehož hranici leží bod $\mathbf{0}$. Potom kužel generovaný D, neboli množina $\{t\mathbf{x} : \mathbf{x} \in D, t \geq 0\}$ sestává z otevřené poloroviny a bodu $\mathbf{0}$, a tedy není uzavřený. To samozřejmě není s lemmatem ve sporu, jen to ukazuje, že je potřeba nějak využít konečnost.

Zavedeme pojem *primitivní kužel* v $\mathbb{R}^m$: to je konvexní kužel generovaný nějakými $k \leq m$ lineárně nezávislými vektory. Než přistoupíme k důkazu lemmatu 6.5.3, vyřešíme následující speciální případ:

6.5.4 Lemma. *Každý primitivní kužel P v $\mathbb{R}^m$ je uzavřený.*

Důkaz. Nechť $P_0 \subseteq \mathbb{R}^k$ je kužel generovaný vektory $\mathbf{e}_1, \ldots, \mathbf{e}_k$ standardní báze $\mathbb{R}^k$. Jinak řečeno, P_0 je nezáporný ortant, který nepochybně uzavřený je (například proto, že je průnikem uzavřených poloprostorů $x_i \geq 0, i = 1, 2, \ldots, k$).

Nechť $P \subseteq \mathbb{R}^m$ je primitivní kužel generovaný lineárně nezávislými vektory $\mathbf{a}_1, \ldots, \mathbf{a}_k$. Definujeme lineární zobrazení $f\colon \mathbb{R}^k \to \mathbb{R}^m$ předpisem $f(\mathbf{x}) = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \cdots + x_k\mathbf{a}_k$. Toto f je díky lineární nezávislosti $\mathbf{a}_j$ prosté a platí $P = f(P_0)$. Takže stačí dokázat následující tvrzení: *Je-li $f\colon \mathbb{R}^k \to \mathbb{R}^m$ prosté lineární zobrazení a $P_0 \subseteq \mathbb{R}^k$ je uzavřená množina, pak obraz $f(P_0)$ je uzavřený.*

Obor hodnot f označíme $L = f(\mathbb{R}^k)$. Protože je f prosté, je to lineární izomorfismus mezi $\mathbb{R}^k$ a L, a existuje tedy inverzní lineární zobrazení $g = f^{-1}\colon L \to \mathbb{R}^k$. Platí $P = g^{-1}(P_0)$. Poněvadž každé lineární zobrazení euklidovského prostoru je spojité (to se dá ověřit pomocí maticového zápisu lineárního zobrazení), je i g spojité. Vzor uzavřené množiny při spojitém zobrazení je uzavřená množina (zatímco *obraz* uzavřené množiny obecně uzavřený být nemusí), takže P je uzavřená podmnožina L. Protože L je uzavřená podmnožina $\mathbb{R}^m$ (lineární podprostor), musí být P uzavřený, jak jsme chtěli. □

Lemma 6.5.3 je nyní důsledkem lemmatu 6.5.4, skutečnosti, že sjednocení konečně mnoha uzavřených množin je uzavřené, a následujícího lemmatu:

6.5.5 Lemma. *Nechť C je konvexní kužel v $\mathbb{R}^m$ generovaný konečně mnoha vektory $\mathbf{a}_1, \ldots, \mathbf{a}_n$. Potom se C dá vyjádřit jako sjednocení konečně mnoha primitivních kuželů.*

Důkaz. Ověříme, že každé $\mathbf{x} \in C$ je obsaženo v primitivním kuželu generovaném vhodnou lineárně nezávislou podmnožinou vektorů $\mathbf{a}_i$. Můžeme

předpokládat, že $\mathbf{x} \neq \mathbf{0}$ (jelikož $\{\mathbf{0}\}$ je primitivní kužel generovaný prázdnou množinou vektorů).

Nechť $I \subseteq \{1, 2, \ldots, n\}$ je množina nejmenší možné velikosti taková, že $\mathbf{x}$ leží v konvexním kuželu generovaném $A_I = \{\mathbf{a}_i : i \in I\}$ (to je standardní trik lineární algebry a konvexní geometrie). Tedy existují nezáporné koeficienty α_i, $i \in I$, takové, že $\mathbf{x} = \sum_{i \in I} \alpha_i \mathbf{a}_i$. Koeficienty α_i jsou dokonce kladné, protože pokud by některé α_i bylo 0, mohli bychom i z I vynechat. Teď potřebujeme dokázat, že množina A_I je lineárně nezávislá. Pro spor předpokládejme, že existuje netriviální lineární kombinace $\sum_{i \in I} \beta_i \mathbf{a}_i = \mathbf{0}$, ve které nejsou všechna β_i rovna 0. Potom existuje reálné t takové, že všechny výrazy $\alpha_i - t\beta_i$ jsou nezáporné a alespoň jeden nulový. (Nejprve můžeme uvážit případ, kdy je nějaké β_i kladné, začít s $t = 0$, postupně t zvětšovat a sledovat, co se stane. Případ záporného β_i se řeší podobně, jen se t bude z počáteční nulové hodnoty zmenšovat.) Potom rovnice

$$\mathbf{x} = \sum_{i \in I} (\alpha_i - t\beta_i)\mathbf{a}_i$$

vyjadřuje $\mathbf{x}$ jako lineární kombinaci méně než $|I|$ vektorů s kladnými koeficienty. □

7

Nejen simplexová metoda

Pro úlohu lineárního programování byly postupně navrženy desítky různých algoritmů. Většina z nich se příliš neosvědčila a jen málokteré se ukázaly jako vážný soupeř pro simplexovou metodu, algoritmus historicky první. Přinejmenším dvě metody ale v době objevu vyvolaly velké vzrušení a určitě stojí za zmínku.

První z nich, *elipsoidová metoda*, nemůže soutěžit se simplexovou metodou v praxi, ale má ohromný význam teoretický. Je to první algoritmus na lineární programování, o němž se podařilo dokázat, že vždy běží v polynomiálním čase (což se o simplexové metodě neumí dodnes a pro mnoho jejích variant to ani neplatí).

Druhá je *metoda vnitřního bodu*, nebo spíše bychom měli říkat metody vnitřního bodu, protože je to celá skupina algoritmů. Pro některé z nich byl také dokázán polynomiální odhad na dobu běhu, ale navíc tyto algoritmy dnes v praxi soutěží se simplexovou metodou velmi úspěšně. Zdá se, že pro některé typy úloh je lepší simplexová metoda, pro jiné zase metody vnitřního bodu.

Poznamenejme ještě, že pro lineární programování se používá i několik dalších algoritmů příbuzných simplexové metodě. *Duální simplexová metoda* se dá zhruba popsat jako simplexová metoda aplikovaná na duální úlohu, ale detaily organizace výpočtu, které jsou v praxi klíčové pro rychlost algoritmu, jsou poněkud jiné. Duální simplexová metoda se zvlášť hodí pro úlohy v rovnicovém tvaru, které mají $n - m$ podstatně menší než m.

Primárně-duální metoda podobně jako duální simplexová metoda přechází mezi přípustnými řešeními duální úlohy, ale nedělá pivotovací kroky, nýbrž v každém kroku řeší jistou pomocnou úlohu, odvozenou z úlohy primární. Pro úlohy pocházející z kombinatorických optimalizačních problémů má pomocná úloha často názorný význam a dá se řešit kombinatorickými prostředky. Primárně-duální metoda je základem řady přibližných algoritmů

pro výpočetně náročné problémy kombinatorické optimalizace.

7.1 Elipsoidová metoda

Elipsoidová metoda byla navržena kolem roku 1970 jako algoritmus na jisté nelineární optimalizační problémy. V roce 1979 nastínil v krátkém článku Leonid Chačijan, jak se touto metodou dá řešit úloha lineárního programování, a to v polynomiálním čase. Světový tisk kolem toho nadělal senzaci, protože novináři výsledek překroutili a prezentovali jej jako nevídaný průlom v praktických výpočetních metodách[1]. Přitom elipsoidová metoda pro praxi lineárního programování zajímavá nikdy nebyla – Chačijanův objev byl opravdu velice významný, ale pro teorii výpočetní složitosti. O důkaz řešitelnosti úlohy lineárního programování v polynomiálním čase se předtím lidé marně pokoušeli desítky let. Elipsoidová metoda také byla koncepčně naprosto odlišná od předchozích přístupů, což byly hlavně varianty simplexové metody.

Velikost vstupu, polynomiální algoritmy. Abychom mohli přesněji popsat, co se míní polynomiálním algoritmem na lineární programování, musíme napřed definovat **velikost vstupu** pro úlohu lineárního programování. Zhruba řečeno, je to celkový počet bitů potřebných k zapsání vstupu.

Nejdřív definujeme pro celé číslo i jeho *bitovou velikost* jako

$$\langle i \rangle = \lceil \log_2(|i| + 1) \rceil + 1,$$

což je počet bitů ve dvojkovém zápisu i včetně znaménka. Pro racionální číslo r, tj. zlomek $r = p/q$, bitovou velikost definujeme jako $\langle r \rangle = \langle p \rangle + \langle q \rangle$.

[1] Poučný citát z všeobecně zajímavého článku

E. L. Lawler: The Great Mathematical Sputnik of 1979, *Math. Intelligencer* 2(1980) 191–198:

Zdá se, že článek v Timesech vyjadřoval jistá neotřesitelná přesvědčení pisatele, Malcolma W. Browna. Browne zatelefonoval Georgi Dantzigovi ze Stanfordské Univerzity, průkopníkovi a autoritě v oboru lineárního programování, a snažil se ho přimět ke všelijakým prohlášením. Dantzigova verze interview stojí za zopakování.

> „A co problém obchodního cestujícího?“ zeptal se Browne. „Pokud nějaká souvislost existuje, já o ní nevím,“ řekl Dantzig. („Ruský objev přinesl metodu na řešení problémů souvisejících s takzvaným problémem obchodního cestujícího,“ napsal Browne.) „A kryptografie?“ zeptal se Browne. „Pokud nějaká souvislost existuje, já o ní nevím,“ řekl Dantzig. („Může ovlivnit i teorii kódů,“ napsal Browne.) „Je ruská metoda prakticky použitelná?“ zeptal se Browne. „Není,“ řekl Dantzig. („Matematikové popisují objev ... jako metodu, jíž mohou počítače nacházet řešení třídy velmi obtížných problémů, k nimž se dosud přistupovalo metodou pokusů a omylů,“ napsal Browne.)

Pro n-složkový vektor $\mathbf{v}$, jehož složky jsou racionální čísla, položíme $\langle \mathbf{v} \rangle = \sum_{i=1}^{n} \langle v_i \rangle$, a podobně $\langle A \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \langle a_{ij} \rangle$ pro racionální matici typu $m \times n$. Uvážíme-li úlohu U lineárního programování třeba ve tvaru

$$\text{maximalizovat } \mathbf{c}^T\mathbf{x} \text{ za podmínky } A\mathbf{x} \leq \mathbf{b},$$

a omezíme-li se na případ, kdy A, $\mathbf{b}$ i $\mathbf{c}$ jsou racionální (což je z výpočetního hlediska předpoklad rozumný), pak bitová velikost úlohy U je $\langle U \rangle = \langle A \rangle + \langle \mathbf{b} \rangle + \langle \mathbf{c} \rangle$.

Říkáme, že nějaký algoritmus je **polynomiální algoritmus na úlohu lineárního programování**, pokud existuje mnohočlen $p(x)$ takový, že algoritmus pro každou úlohu U lineárního programování s racionálním A, $\mathbf{b}$ i $\mathbf{c}$ najde správné řešení v nejvýš $p(\langle U \rangle)$ krocích. Kroky se počítají v některém z obvyklých modelů výpočtu, například jako kroky Turingova stroje (většinou na konkrétně zvoleném modelu nezáleží, co je polynomiální na jednom modelu, je polynomiální i na všech ostatních rozumných modelech). Ihned zdůrazněme, že jedna aritmetická operace se nepočítá jako jeden krok! Jako kroky se počítají operace s jednotlivými bity, takže například sečtení dvou k-bitových celých čísel vyžaduje řádově k kroků.

Odbočme na chvíli od lineárního programování a uvažme Gaussovu eliminaci, dobře známý algoritmus na řešení soustavy lineárních rovnic. Pro soustavu lineárních rovnic tvaru $A\mathbf{x} = \mathbf{b}$, kde (pro jednoduchost) A je matice typu $n \times n$ a A i $\mathbf{b}$ jsou racionální, definujeme velikost vstupu přirozeně jako $\langle A \rangle + \langle \mathbf{b} \rangle$. Je Gaussova eliminace polynomiální algoritmus? To je záludná otázka! Tento algoritmus sice potřebuje řádově nejvýš n^3 aritmetických operací, ale háček je v tom, že v průběhu výpočtu by se mohly objevit příliš velké mezivýsledky. Kdyby v průběhu Gaussovy eliminace vznikla například celá čísla s 2^n bity, což se při nešikovné implementaci skutečně může stát, výpočet by potřeboval exponenciálně mnoho kroků, i když by zahrnoval jen n^3 aritmetických operací. (To se ovšem týká *přesného* výpočtu, kdežto mnohé programy počítají v plovoucí čárce a čísla tudíž průběžně zaokrouhlují. Pak ale není záruka, že vypočtené řešení je správné.) Abychom čtenáře zbytečně nestrašili: ví se, jak Gaussovu eliminaci implementovat v polynomiálním čase. Chtěli jsme jen poukázat na to, že to není samozřejmé (a ani příliš jednoduché), a upozornit na jeden typ potíží, které mohou při dokazování polynomiálnosti algoritmu vzniknout.

Elipsoidová metoda i další algoritmy, jež zmíníme v sekci 7.2, jsou polynomiální, kdežto simplexová metoda s Blandovým pravidlem (ani s řadou dalších pivotovacích pravidel) polynomiální není[2].

[2]To se dokazuje pomocí Klee-Mintyho krychle, viz sekce 5.7, a využije se toho, že se n-dimenzionální Klee-Mintyho krychle dá reprezentovat vstupem velikosti polynomiální v n, což je ovšem nutné ověřit.

Silně polynomiální algoritmy. Pro algoritmy, jejichž vstup je popsán souborem celých nebo racionálních čísel, jako jsou například algoritmy na lineární programování, se kromě počtu elementárních operací s jednotlivými bity uvažuje i počet aritmetických operací (sčítání, odčítání, násobení, dělení, umocňování). To často dává realističtější obraz o době běhu algoritmu na skutečných počítačích, protože soudobé počítače většinou provádějí základní aritmetické operace jako elementární krok, tedy pokud operandy nejsou příliš velká čísla.

Vhodná verze Gaussovy eliminace jednak je polynomiální algoritmus ve smyslu diskutovaném výše, a jednak má počet aritmetických operací omezený polynomem, konkrétně polynomem Cn^3 pro vhodnou konstantu C, kde n je počet rovnic soustavy a počet neznámých. Říkáme, že Gaussova eliminace je **silně polynomiální** algoritmus na řešení soustav lineárních rovnic.

Silně polynomiální algoritmus pro úlohu lineárního programování by byl takový, který by jednak byl polynomiální ve smyslu definovaném před chvílí, a jednak by pro každou úlohu s n proměnnými a m omezeními našel řešení s použitím nejvýš $p(m+n)$ aritmetických operací, kde $p(x)$ je nějaký pevný mnohočlen. Ale žádný silně polynomiální algoritmus pro lineární programování není znám.

Elipsoidová metoda není silně polynomiální. Pro každé přirozené číslo M se dá najít úloha lineárního programování s pouhými 2 proměnnými a 2 omezeními, pro niž elipsoidová metoda provede aspoň M aritmetických operací (vstup pro takovou úlohu musí ovšem obsahovat čísla, jejichž bitová velikost roste do nekonečna pro M jdoucí do nekonečna). Speciálně se počet aritmetických operací elipsoidové metody nedá omezit žádným polynomem v $m+n$.

Elipsoidy. Dvoudimenzionální elipsoid je elipsa plus její vnitřek. Obecný elipsoid se dá nejpřirozeněji zavést jako afinní transformace koule. Označíme

$$B^n = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x}^T\mathbf{x} \leq 1\}$$

n-dimenzionální kouli jednotkového poloměru. Pak n-dimenzionální **elipsoid** je každá množina tvaru

$$E = \{M\mathbf{x} + \mathbf{s} : \mathbf{x} \in B^n\},$$

kde M je libovolná regulární matice typu $n \times n$ a $\mathbf{s} \in \mathbb{R}^n$ je libovolný vektor. Zobrazení $\mathbf{x} \mapsto M\mathbf{x} + \mathbf{s}$ je složením lineárního isomorfismu a posunutí. Je to **afinní zobrazení**, přesněji řečeno *regulární* afinní zobrazení $\mathbb{R}^n \to \mathbb{R}^n$.

Úpravou definice můžeme elipsoid popsat nerovnicí:

$$\begin{aligned} E &= \{\mathbf{y} \in \mathbb{R}^n : M^{-1}(\mathbf{y} - \mathbf{s}) \in B^n\} = \\ &= \{\mathbf{y} \in \mathbb{R}^n : (\mathbf{y} - \mathbf{s})^T (M^{-1})^T M^{-1} (\mathbf{y} - \mathbf{s}) \leq 1\} = \\ &= \{\mathbf{y} \in \mathbb{R}^n : (\mathbf{y} - \mathbf{s})^T Q^{-1} (\mathbf{y} - \mathbf{s}) \leq 1\}, \end{aligned} \tag{7.1}$$

kde jsme označili $Q = MM^T$. Z nauky o maticích se dobře ví, že taková Q je pozitivně definitní matice, tj. čtvercová symetrická matice splňující $\mathbf{x}^T Q\mathbf{x} > 0$ pro všechny nenulové vektory $\mathbf{x}$. Obráceně, každá pozitivně definitní matice Q se dá rozložit jako $Q = MM^T$ pro nějakou regulární M. Tudíž alternativní definice elipsoidu je, že to je množina popsaná (7.1) pro nějakou pozitivně definitní Q.

Geometricky je $\mathbf{s}$ střed elipsoidu E. Je-li Q diagonální matice a $\mathbf{s} = \mathbf{0}$, dostáváme elipsoid v *osové poloze*, tvaru

$$\left\{\mathbf{y} \in \mathbb{R}^n : \frac{y_1^2}{q_{11}} + \frac{y_2^2}{q_{22}} + \cdots + \frac{y_n^2}{q_{nn}} \leq 1\right\},$$

osy elipsoidu jsou pak rovnoběžné s osami souřadnic. Čísla $\sqrt{q_{11}}$, $\sqrt{q_{22}}$,…, $\sqrt{q_{nn}}$ jsou *délky poloos* elipsoidu E, což by mělo znít povědomě těm, kdo jsou zvyklí na rovnici elipsy ve tvaru $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$. Z teorie vlastních čísel je známo, že každá pozitivně definitní matice Q se dá převést na diagonální tvar ortogonální změnou báze, tj. existuje ortogonální matice T tak, že TQT^{-1} je diagonální, s vlastními čísly Q na diagonále. Geometricky je T otočení soustavy souřadnic, které přivede elipsoid do osové polohy.

Lev na Sahaře. Jedna z tradičních matematických anekdot dává návod, jak lovit lva na Sahaře. Oplotíme Saharu, dalším plotem ji rozdělíme na polovinu, a najdeme jednu polovinu, ve které lev není. Druhou polovinu pak rozpůlíme dalším plotem, a tak pokračujeme, až už bude oplocený kousek tak malý, že se v něm lev nemůže hýbat a je ulovený, anebo když tam žádný lev není, dokázali jsme, že nebyl ani na celé Sahaře. I když o kvalitách návodu lze diskutovat, pro nás je podstatné, že dává poměrně dobrý popis elipsoidové metody. Ve skutečné elipsoidové metodě se ale trvá na tom, aby momentálně oplocený kousek byl vždy tvaru elipsoidu, i za cenu toho, že lev se někdy může vracet na místa, odkud byl už dřív vyhnán – zaručí se jen, že plocha jeho území se stále zmenšuje.

Elipsoidová metoda neřeší přímo úlohu lineárního programování, nýbrž soustavu lineárních nerovnic $A\mathbf{x} \leq \mathbf{b}$. Jak ale víme, úloha lineárního programování se dá na tento problém převést, viz sekci 6.1. Pro jednodušší vysvětlení budeme navíc předpokládat, že se uvedený problém řeší za následujících změkčených podmínek:

Změkčená úloha. *Spolu s maticí A a vektorem $\mathbf{b}$ jsou dána čísla $R > \varepsilon > 0$. Předpokládá se, že množina $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$ je obsažena v kouli $B(\mathbf{0}, R)$ se středem v $\mathbf{0}$ o poloměru R. Jestliže P obsahuje nějakou kouli o poloměru ε, musí algoritmus vrátit nějaký bod $\mathbf{y} \in P$. Jestliže ale P žádnou kouli o poloměru ε neobsahuje, může algoritmus vrátit buď nějaký bod $\mathbf{y} \in P$, nebo odpověď NEMÁ ŘEŠENÍ.*

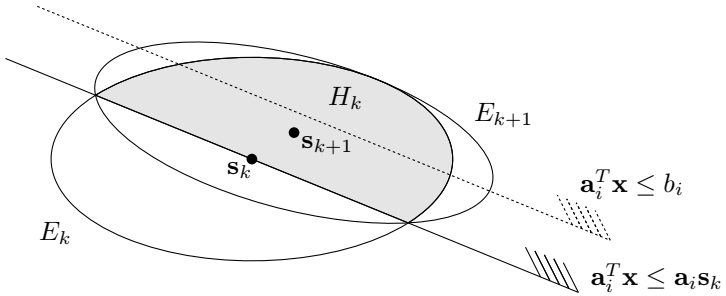
Koule $B(\mathbf{0}, R)$ tedy hraje roli Sahary a předpokládáme že lev, pokud je přítomen, je velký aspoň ε. Je-li na Sahaře jen menší lev, můžeme ho také ulovit, ale nemusíme.

Za těchto předpokladů vytváří elipsoidová metoda posloupnost elipsoidů $E_0, E_1, \ldots, E_t$, kde $P \subseteq E_k$ pro každé k, takto:

1. Polož $k = 0$ a $E_0 = B(\mathbf{0}, R)$.

2. Nechť elipsoid E_k je tvaru $E_k = \{\mathbf{y} \in \mathbb{R}^n : (\mathbf{y}-\mathbf{s}_k)^T Q_k^{-1} (\mathbf{y}-\mathbf{s}_k) \leq 1\}$. Pokud $\mathbf{s}_k$ splňuje všechny nerovnice soustavy $A\mathbf{x} \leq \mathbf{b}$, vrať $\mathbf{s}_k$ jako řešení, **konec**.

3. Jinak zvol nějakou nerovnici soustavy, kterou $\mathbf{s}_k$ porušuje; nechť je to i-tá nerovnice, tj. máme $\mathbf{a}_i^T \mathbf{s}_k > b_i$. Definuj E_{k+1} jako elipsoid nejmenšího možného objemu, který obsahuje „půlelipsoid“ $H_k = E_k \cap \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}_i^T \mathbf{x} \leq \mathbf{a}_i^T \mathbf{s}_k\}$, viz obrázek:



4. Je-li objem E_{k+1} menší než objem koule o poloměru ε, vrať NEMÁ ŘEŠENÍ, **konec**. Jinak zvětši k o 1 a pokračuj krokem 2.

Označme H'_k průnik elipsoidu E_k s poloprostorem $\{x \in \mathbb{R}^n : \mathbf{a}_i \mathbf{x} \leq b_i\}$ definovaným i-tou nerovnicí soustavy. Jestliže $P \subseteq E_k$, potom též $P \subseteq H'_k$, a tím spíš i $P \subseteq H_k$. Proč se za E_{k+1} bere nejmenší elipsoid obsahující H_k, místo aby se vzal nejmenší elipsoid obsahující H'_k? Čistě proto, aby vzorce definující E_{k+1} byly co nejjednodušší a usnadnila se analýza algoritmu.

Elipsoid E_{k+1}, tedy elipsoid nejmenšího objemu obsahující H_k, je vždy určen jednoznačně. Pro ilustraci uvedeme, že je dán vztahy

$$\mathbf{s}_{k+1} \quad = \quad \mathbf{s}_k - \frac{1}{n+1} \cdot \frac{Q\mathbf{s}_k}{\sqrt{\mathbf{s}_k^T Q \mathbf{s}_k}},$$

$$Q_{k+1} \quad = \quad \frac{n^2}{n^2-1}\left(Q_k - \frac{2}{n+1}\cdot\frac{Q\mathbf{s}_k\mathbf{s}_k^T Q}{\mathbf{s}_k^T Q\mathbf{s}_k}\right).$$

Bez důkazu necháme také fakt klíčový pro správnost a efektivitu algoritmu: vždy platí

$$\frac{\text{objem}(E_{k+1})}{\text{objem}(E_k)} \le e^{-1/(2n+2)}.$$

Tudíž elipsoid E_k bude mít objem aspoň $e^{-k/(2n+2)}$-krát menší než počáteční koule $B(\mathbf{0}, R)$, a protože objem n-dimenzionální koule je úměrný n-té mocnině poloměru, pro k splňující $R \cdot e^{-k/n(2n+2)} < \varepsilon$ je objem E_k určitě menší než objem koule o poloměru ε. Takové k dává horní odhad pro maximální počet iterací t, z čehož vyjde $t \le n(2n+2)\ln(R/\varepsilon)$. To je jistě omezeno polynomem v $n + \langle R\rangle + \langle\varepsilon\rangle$.

Tolik jednoduchá a pěkná myšlenka elipsoidové metody, a teď přijdou zvládnutelné, ale nepříjemné komplikace. Především elipsoid E_{k+1} nemůžeme počítat přesně, přinejmenším v racionální aritmetice, protože definující vzorce obsahují odmocniny, a i kdyby neobsahovaly, koeficienty by po mnoha iteracích mohly mít příliš mnoho bitů a algoritmus by nebyl polynomiální. To se obejde tím, že se E_{k+1} počítá jen přibližně, s určitou vhodně zvolenou přesností. Přitom se musí dát pozor, aby P určitě zůstal v E_{k+1} obsažen – proto se přibližný E_{k+1} malinko zvětší. S tím souvisí i to, že pokud se například pro řezání elipsoidu mnohokrát za sebou používá stále stejná nerovnice, elipsoidy začnou být příliš protáhlé (jehlovité), a je nutné je uměle zkrátit.

Další potíž je ta, že ve skutečnosti nechceme řešit změkčený problém s R a ε, nýbrž libovolnou soustavu lineárních nerovnic. Tady přijde ke slovu omezení na bitovou velikost složek matice A a vektoru $\mathbf{b}$. Platí následující:

(E1) (Existuje-li řešení, existuje i nepříliš velké řešení.) Označíme-li $\varphi = \langle A\rangle + \langle\mathbf{b}\rangle$ velikost vstupu pro soustavu $A\mathbf{x} \le \mathbf{b}$, potom tato soustava má řešení, právě když má řešení soustava

$$\begin{aligned} A\mathbf{x} &\le \mathbf{b} \\ -K \le x_1 &\le K \\ -K \le x_2 &\le K \\ &\vdots \\ -K \le x_n &\le K, \end{aligned}$$

kde $K = 2^\varphi$. Pro posledně jmenovanou soustavu jsou všechna řešení určitě obsažena v kouli $B(\mathbf{0}, R)$, kde $R = K\sqrt{n}$.

(E2) (Existuje-li řešení, pak řešení mírně uvolněné soustavy obsahuje i malou kouli.) Položme $\eta = 2^{-5\varphi}$, $\varepsilon = 2^{-6\varphi}$ a buď $\boldsymbol{\eta}$ n-složkový vektor se všemi složkami rovnými η. Potom soustava $A\mathbf{x} \le \mathbf{b}$ má řešení, právě když má řešení soustava $A\mathbf{x} \le \mathbf{b} + \boldsymbol{\eta}$, a v takovém případě množina řešení posledně jmenované soustavy obsahuje kouli o poloměru ε.

Je celkem jasné, jak využít těchto faktů při řešení obecné soustavy $A\mathbf{x} \le \mathbf{b}$ elipsoidovou metodou. Místo původní soustavy řešíme elipsoidovou metodou

změkčený problém, ale pro novou soustavu $A\mathbf{x} \leq \mathbf{b} + \boldsymbol{\eta}$, $-K - \eta \leq x_1 \leq K + \eta, -K - \eta \leq x_2 \leq K + \eta, \ldots, -K - \eta \leq x_n \leq K + \eta$, přičemž K, R a ε se vhodně zvolí (nejdříve přidáme k původnímu systému omezení $-K \leq x_i \leq K$ jako v (E1), a potom na takto rozšířený systém aplikujeme (E2)). Důležité je, že bitová velikost R a ε i velikost vstupu pro novou soustavu jsou omezeny polynomiální funkcí původní velikosti vstupu φ, a proto elipsoidová metoda poběží v polynomiálním čase.

Fakta (E1) a (E2) nebudeme dokazovat, ale naznačíme základní myšlenku. Pro fakt (E1) to uděláme pro $n = 2$, tj. v rovině. Mějme soustavu m nerovnic tvaru

$$a_{i1}x + a_{i2}y \leq b_i, \quad i = 1, 2, \ldots, m.$$

Buď p_i přímka $\{(x, y) \in \mathbb{R}^2 : a_{i1}x + a_{i2}y = b_i\}$. Jednoduše se spočítá, že průsečík p_i a p_j, pokud existuje, má souřadnice

$$\left(\frac{a_{i2}b_j - a_{j2}b_i}{a_{i2}a_{j1} - a_{i1}a_{j2}}, \frac{a_{j1}b_i - a_{i1}b_j}{a_{i2}a_{j1} - a_{i1}a_{j2}} \right).$$

Pokud například všechna a_{ij} a b_i jsou celá čísla s absolutní hodnotou nejvýš 1000, jsou souřadnice všech průsečíků zlomky s čitatelem i jmenovatelem omezeným $2{\times}10^6$. Jestliže tedy má množina řešení naší soustavy tří nerovnic aspoň jeden vrchol, musí tento vrchol ležet ve čtverci o straně $4{\times}10^6$ se středem v počátku. Jestliže množina řešení vrchol nemá a je neprázdná, dá se například ukázat, že musí obsahovat některou z přímek p_i, a že každá p_i zasahuje do onoho čtverce. Tak nějak se dá ověřit fakt (E1) pro naši speciální soustavu. Pro obecnou soustavu v dimenzi n je myšlenka stejná, a pro vyjádření souřadnic vrcholů se využije Cramerova pravidla a odhad pro velikost determinantu matice.

Fakt (E2) dá poněkud víc práce, ale idea je podobná. Každé řešení $\tilde{\mathbf{x}}$ původní soustavy $A\mathbf{x} \leq \mathbf{b}$ vyhovuje i upravené soustavě $A\mathbf{x} \leq \mathbf{b} + \boldsymbol{\eta}$, a také všechna $\mathbf{x}$ z koule $B(\tilde{\mathbf{x}}, \varepsilon)$ jí vyhovují, protože změna $\mathbf{x}$ o ε nemůže změnit žádnou složku $A\mathbf{x}$ o víc než η.

Pokud $A\mathbf{x} \leq \mathbf{b}$ řešení nemá, podle vhodné varianty Farkasova lemmatu (kterou jsme neuváděli, ale která se snadno odvodí z variant zmíněných v sekci 6.4) existuje nezáporné $\mathbf{y} \in \mathbb{R}^m$ splňující $\mathbf{y}^T A = \mathbf{0}$ a $\mathbf{y}^T \mathbf{b} = -1$. Zase pomocí Cramerova pravidla se ukáže, že existuje i $\mathbf{y}$ s nepříliš velkými složkami, a to potom dosvědčí neřešitelnost i pro soustavu $A\mathbf{x} \leq \mathbf{b} + \boldsymbol{\eta}$.

Tím končíme nástin elipsoidové metody. Pokud byly některé části pro čtenáře příliš neúplné a mlhavé, můžeme jen doporučit nějaké obsáhlejší pojednání, například ve vynikající knize

> M. Grötschel, L. Lovász, L. Schrijver: *Geometric Algorithms and Combinatorial Optimization*, druhé vydání, Springer, Heidelberg 1994.

Proč elipsoidy? Používají se v elipsoidové metodě proto, že to je asi nejjednodušší třída n-dimenzionálních konvexních množin, která je uzavřená na regulární afinní zobrazení. Populárně řečeno, je dost bohatá na to, aby mohla

aproximovat všechny konvexní mnohostěny včetně placatých a jehlovitých. Kdyby si to někdo přál, elipsoidy se dají nahradit například simplexy, ale vzorce v algoritmu a jeho analýze budou o poznání nepříjemnější než pro elipsoidy.

Elipsoidová metoda nemusí znát celou úlohu. Soustava nerovnic $A\mathbf{x} \leq \mathbf{b}$ se dá elipsoidové metodě zadat i pomocí **oddělovacího orákula**. To je algoritmus (černá skříňka), který jako vstupy přijímá body $\mathbf{s} \in \mathbb{R}^n$, a pokud $\mathbf{s}$ je řešením soustavy, vrací odpověď ANO, zatímco když $\mathbf{s}$ není řešením, vrací jednu (libovolnou) nerovnici soustavy, již $\mathbf{s}$ porušuje. (Taková nerovnice odděluje $\mathbf{s}$ od množiny řešení, a odtud název oddělovací orákulum). Elipsoidová metoda oddělovacímu orákulu postupně zadává středy $\mathbf{s}_k$ generovaných elipsoidů a porušenou nerovnici vždy využije na vytvoření elipsoidu dalšího.

Zmiňujeme to proto, že pro některé zajímavé optimalizační problémy se oddělovací orákulum dá efektivně implementovat, přestože příslušná soustava má exponenciálně mnoho nerovnic nebo dokonce nekonečně mnoho nerovnic (takové soustavy jsme zatím vůbec neuvažovali).

Zřejmě nejzásadnějším příkladem situace, kdy se umí nekonečná soustava lineárních nerovnic řešit elipsoidovou metodou, je **semidefinitní programování**. V něm se neuvažuje neznámý *vektor* $\mathbf{x}$, nýbrž neznámá čtvercová *matice* $X = (x_{ij})_{i,j=1}^n$. Optimalizuje se nějaká lineární funkce proměnných x_{ij}, a omezující podmínky jsou jednak lineární rovnice a nerovnice pro ta x_{ij}, a jednak (to je rozdíl proti lineárnímu programování) požadavek, že matice X musí být *pozitivně semidefinitní*. Posledně jmenovaný požadavek se dá vyjádřit soustavou nekonečně mnoha lineárních nerovnic, totiž $\mathbf{a}^T X \mathbf{a} \geq 0$ pro každé $\mathbf{a} \in \mathbb{R}^n$. Pro tuto soustavu lze sestrojit efektivní oddělovací orákulum na základě algoritmů na výpočet vlastních vektorů matice a elipsoidová metoda najde optimum v polynomiálním čase (ve skutečnosti to ovšem není tak jednoduché, jak by se z našeho minipopisu snad mohlo zdát).

Semidefinitním programováním se pak dají v polynomiálním čase řešit četné výpočetní problémy, některé přesně a některé aspoň přibližně, a někdy se tak dostane jediný známý polynomiální algoritmus. Příkladem je **problém maximálního řezu** (MAXCUT), kdy se množina vrcholů daného grafu $G = (V, E)$ má rozdělit na dvě části tak, aby co nejvíce hran šlo mezi nimi. Přes semidefinitní programování se dostane aproximační algoritmus, zvaný Goemansův-Williamsonův, který vždy najde rozdělení, kde počet hran jdoucích napříč je aspoň 87,8% počtu optimálního. To je nejlepší známý aproximační algoritmus pro tuto úlohu a nejspíš i nejlepší možný aproximační algoritmus s polynomiální dobou běhu.

Semidefinitní programování je patrně nejvýznamnějším novým nástrojem v teorii optimalizace za posledních zhruba 20 let. Více o tomto tématu se najde například v článku

> L. Lovász: Semidefinite programs and combinatorial optimization, ve sborníku *Recent Advances in Algorithms and Combinatorics* (editoři B. Reed a C. Linhares-Sales), str. 137–194, Springer, New York, 2003.

Elipsoidová metoda se v semidefinitním programování a podobných aplikacích dá nahradit i vhodnými metodami vnitřního bodu, čímž vzniknou algoritmy efektivní nejen teoreticky, ale i prakticky. O tom pojednává například článek

> K. Krishnan, T. Terlaky: Interior point and semidefinite approaches in combinatorial optimization, in: D. Avis, A. Hertz, O. Marcotte (editoři): *Graph Theory and Combinatorial Optimization*, Springer, Berlin 2005, str. 101–158.

Teorie versus praxe. Pojem polynomiálního algoritmu navrhl v 70. letech 20. století Jack Edmonds jako formální protějšek intuitivního pojmu efektivního algoritmu. Dnes je většinou teoretikova první otázka u každého algoritmu: je polynomiální, nebo ne?

Jak je možné, že elipsoidová metoda, která je polynomiální, je v praxi mnohem pomalejší, než simplexová metoda, která polynomiální není? Jeden důvod je ten, že elipsoidová metoda je sice polynomiální, ale stupeň polynomu je poměrně vysoký. Druhý a hlavní důvod je, že simplexová metoda je pomalá jenom na speciálně zkonstruovaných úlohách, s nimiž se v praxi téměř nesetká, kdežto elipsoidová metoda se zřídkakdy chová lépe, než zaručuje odhad pro nejhorší případ. Teoreticky je ale „dobré chování na všech vstupech až na řídké výjimky“ velmi obtížné zachytit. Navíc zaručený odhad efektivity pro *všechny* vstupy je přece jenom uspokojivější než jen empiricky podložená víra, že algoritmus většinou funguje dobře.

Pojem polynomiálního algoritmu má tedy z praktického hlediska velké nedostatky, ale snaha o konstrukci polynomiálního algoritmu v teorii většinou časem vede i k algoritmům efektivním v praxi. V oblasti lineárního programování jsou působivým příkladem metody vnitřního bodu.

7.2 Metody vnitřního bodu

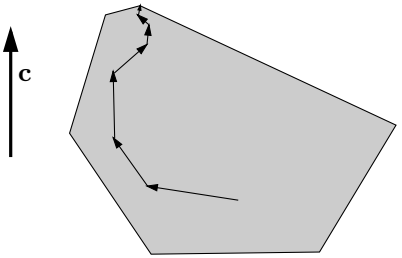
Po senzaci s elipsoidovou metodou se lineární programování znovu dostalo do novin v roce 1984. Tehdy Narendra Karmakar, výzkumník firmy IBM, navrhl algoritmus, jenž se dnes řadí do velké skupiny metod vnitřního bodu, ukázal, že je polynomiální, a publikoval výsledky výpočetních experimentů naznačující, že algoritmus je v praxi podstatně rychlejší než simplexová metoda. I když se jeho prohlášení o praktické efektivitě ukázala jako příliš optimistická, metody vnitřního bodu se dnes na lineární programování používají a mnohdy nad simplexovou metodou vítězí.

Metody vnitřního bodu mají mnoho variant a používaly se přinejmenším od roku 1950 na nelineární optimalizační problémy. V oblasti lineárního programování se člení podle základního přístupu na *metody centrální cesty*, *metody snižování potenciálu* a *metody afinního škálování*, a téměř pro každý z přístupů existuje *primární verze*, *duální verze*, *primárně-duální verze* a někdy i *samoduální verze*. V různých pramenech se přitom při výkladu

přesně stejného algoritmu často vychází z různé intuice, například to, co se někde prezentuje jako krok Newtonovy iterační metody, se jinde odvozuje lineární aproximací z metody Lagrangeových multiplikátorů a ještě jinde se to podává jako projekce gradientu do množiny přípustných řešení. To říkáme proto, aby čtenář, který v literatuře najde o metodách vnitřního bodu něco zdánlivě zcela jiného než zde, nebyl zmaten víc, než je nutné.

Simplexovou metodu jsme vysvětlili i s důkazy, až na záležitost se zacyklením, u elipsoidové metody jsme aspoň zformulovali algoritmus a některá přesná tvrzení o něm, ale u metod vnitřního bodu se pouze pokusíme vyvolat ve čtenáři určitý základní dojem, jak fungují. Je to nejen proto, že už jsme skoro u konce našeho pojednání o lineárním programování, ale také proto, že tyto metody jsou komplikované a používají některé matematické nástroje, které se do úvodního textu ani nehodí. Je vůbec překvapivé, na jak pokročilé matematice je založen moderní lineárně programovací software – ani soustavy lineárních rovnic neřeší jen starou dobrou Gaussovou eliminací, nýbrž kombinaci několika složitých metod.

Simplexová metoda se plíží po hranici množiny přípustných řešení. Elipsoidová metoda svírá množinu přípustných řešení zvenku a až do posledního kroku zůstává mimo ni. Metody vnitřního bodu kráčejí vnitřkem množiny přípustných řešení směrem k optimu a hranici se s velkým úsilím vyhýbají, a až na závěr, když už se dostanou k optimu velice blízko, přeskočí na hranici do optimálního vrcholu, viz schématický obrázek:



Pracovat pouze s vnitřními body množiny přípustných řešení je klíčová myšlenka, která dala metodám vnitřního metodu jejich jméno. Vnitřní body mají řadu příjemných matematických vlastností, určitý druh „obecné polohy“, a umožňují vyhnout se záludnostem kombinatorické struktury hranice, které se badatelé bezúspěšně snažili překonat při pokusech o polynomiální algoritmus založený na simplexové metodě. Základní otázka v metodách vnitřního bodu je, jak se vyhýbat hranici a přitom postupovat dostatečně rychle směrem k optimu.

Naznačíme, jak pracuje **metoda centrální cesty**, která se dnes považuje v praxi za nejúspěšnější. Uvažme úlohu

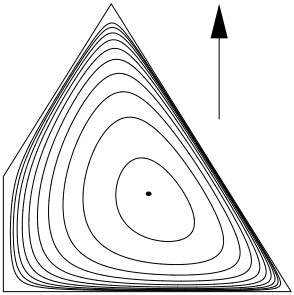
maximalizovat $\mathbf{c}^T\mathbf{x}$ za podmínek $A\mathbf{x} \leq \mathbf{b}$ a $\mathbf{x} \geq \mathbf{0}$.

Z daného vnitřního bodu množiny P přípustných řešení bychom mohli zkusit jít přímo ve směru $\mathbf{c}$, ale tak bychom brzy narazili na hranici. Proto je třeba směr včas změnit tak, aby se cesta hranici vyhýbala a přitom stále tíhla k optimu. Uvažme úlohu s týmiž omezeními, ale s pozměněnou účelovou funkcí

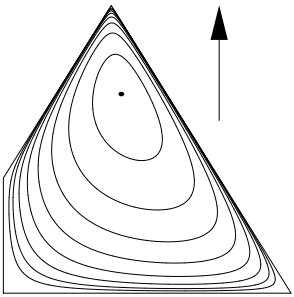
$$f_\mu(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \mu\left(\sum_{j=1}^{n} \ln(x_j) + \sum_{i=1}^{m} \ln\left(b_i - \mathbf{a}_i^T\mathbf{x}\right)\right),$$

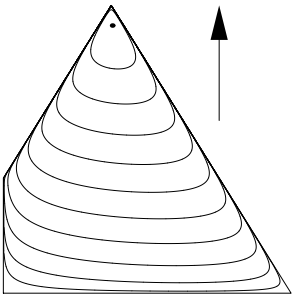
kde $\mathbf{a}_i$ je i-tý řádek matice A a $\mu \geq 0$ je reálný parametr, který budeme později volit podle potřeby. Účelová funkce je nelineární, obsahuje logaritmy, a je udělaná tak, že když se $\mathbf{x}$ blíží ke hranici množiny P, tj. když se některá složka blíží 0 nebo rozdíl pravé a levé strany některé z nerovnic se blíží 0, účelová funkce jde k $-\infty$. Výrazu v závorkách za μ v definici f_μ se říká *bariérová funkce* nebo určitěji *logaritmická bariéra*.

Pro P omezenou a s neprázdným vnitřkem lze ukázat, že pro každé $\mu > 0$ má úloha s účelovou funkcí f_μ jednoznačně určené optimální řešení, které označíme $\mathbf{x}_\mu^*$. Je-li μ velmi velké číslo, člen $\mathbf{c}^T\mathbf{x}$ v účelové funkci bude mít nepatrný vliv a $\mathbf{x}_\mu^*$ bude bod „co nejdál od hranice“, takzvaný *analytický střed* množiny P. Následující obrázek ukazuje (pro dvoudimenzionální úlohu) vrstevnice funkce f_μ pro $\mu = 100$ (vektor $\mathbf{c}$ je znázorněn šipkou a bod $\mathbf{x}_\mu^*$ puntíkem):



Naopak pro malé μ se $\mathbf{x}_\mu^*$ blíží optimu původní úlohy lineárního programování, viz ilustrace pro $\mu = 0.5$ a $\mu = 0.1$:



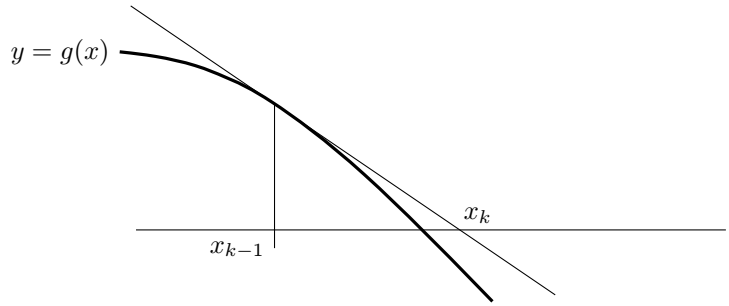


Množina $\{\mathbf{x}^*_\mu : \mu > 0\}$ se nazývá **(primární) centrální cesta** a algoritmus se ji snaží sledovat a dospět tím k optimu.

Body $\mathbf{x}^*_\mu$ se nepočítají přesně, bylo by to příliš obtížné a není to ani potřeba. Zvolíme vhodnou klesající posloupnost $\mu_1 > \mu_2 > \cdots$ jdoucí k 0. Máme-li z $(k-1)$-ní iterace nějaký bod $\mathbf{x}_{k-1} \in P$, v další iteraci se snažíme udělat krok směrem k bodu $\mathbf{x}^*_{\mu_k}$. To znamená, že chceme popojít k maximu funkce $f_{\mu_k}(\mathbf{x})$. Ta je uvnitř P spojitě diferencovatelná, a maximum se pozná podle toho, že tam je gradient f_{μ_k} roven 0 (neboli všechny první parciální derivace jsou tam 0). Takže $\mathbf{x}^*_{\mu_k}$ je (jediným) nulovým bodem vektorové funkce

$$F(\mathbf{x}) = \left(\frac{\partial f_{\mu_k}(\mathbf{x})}{\partial x_1}, \frac{\partial f_{\mu_k}(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f_{\mu_k}(\mathbf{x})}{\partial x_n} \right)^T .$$

Jeden z algoritmů pro hledání nulových bodů diferencovatelných funkcí je **Newtonova metoda**. Pro funkci $g\colon \mathbb{R} \to \mathbb{R}$ jedné proměnné je jeden krok znázorněn na obrázku:



V bodě x_{k-1} se funkce g aproximuje tečnou, a příští bod x_k se určí jako průsečík této tečny s osou x. To se dá zobecnit i na funkci $\mathbb{R}^n \to \mathbb{R}^n$, jejíž graf se v každém kroku aproximuje tečnou nadrovinou.

V metodě centrální cesty dostaneme bod $\mathbf{x}_k$ z bodu $\mathbf{x}_{k-1}$ jedním krokem Newtonovy metody pro výše zavedenou funkci F. Ve skutečnosti ale neuděláme celý krok, ale jdeme jen třeba 60 % cesty směrem k bodu, do nějž by šla Newtonova metoda. Jak se ukáže, tento bod se najde jako řešení jisté soustavy lineárních rovnic. Velká část teorie metod vnitřního bodu se věnuje co nejefektivnějšímu řešení této soustavy, protože to je výpočetně nejnáročnější část.

V některých variantách se dokonce nepoužívá Newtonova metoda, ale metody vyšších řádů, jež aproximují funkci F ne nadrovinou, nýbrž algebraickou plochou vyššího stupně, spočtenou z vyšších derivací funkce F.

Tak, jak jsme metodu centrální cesty zatím popsali, potřebuje do začátku vnitřní bod množiny přípustných řešení. Ten by se dal najít dvoufázovým postupem, podobně jako se v simplexové metodě hledá počáteční přípustná báze. Je ale znám přístup elegantnější. Z dané úlohy U se zkonstruuje nová úloha U' tak, že jednak je U' vždy přípustná a omezená a navíc máme k dipozici explicitně daný bod na její centrální cestě, a jednak se z optimálního řešení U'

pozná, zda byla U přípustná a omezená, a pokud ano, dá se z něj získat také optimální řešení U. Přitom pokud U má n proměnných a m omezení, bude U' mít $O(m+n)$ proměnných a omezení. Tudíž na U' můžeme bez dalších příprav spustit některou metodu centrální cesty a tím vyřešit i U. Výjimečně pozorného čtenáře by mohlo napadnout, proč nepoužít tuto transformaci U na U' i v simplexové metodě? Hlavní důvod je, že pro získání optimálního řešení U ve skutečnosti nestačí jakékoliv optimální řešení U', ale je potřeba optimální řešení „v nejobecnější možné poloze“. Optimálních řešení U' tedy je typicky hodně, vyplňují složitý konvexní mnohostěn, a potřebuje se nějaký vnitřní bod tohoto mnohostěnu. Přesně takové optimální řešení umějí najít metody centrální cesty (kdežto simplexová metoda to nedovede). Už se ale nebudeme pouštět do dalších technických detailů a zde náčrt metod vnitřního bodu ukončíme.

O několika konkrétních metodách vnitřního bodu se ví, že jsou to (slabě) polynomiální algoritmy. Teoretický odhad na maximální počet iterací je $O(L\sqrt{n})$, kde L je maximum z bitových velikostí koeficientů úlohy a n je počet proměnných. Jsou známy příklady, zkonstruované pomocí Klee-Mintyho krychle, pro něž metody centrální cesty opravdu vyžadují $\Omega(\sqrt{n}\log n)$ iterací. V praxi se ale pro pevné L zdá být počet iterací omezen konstantou nebo $O(\log n)$. To je trochu podobná situace jako pro simplexovou metodu, kde chování v nejhorším případě je mnohem horší než chování pro typické vstupy.

Metody vnitřního bodu se úspěšně používají nejen pro lineární programování, ale i pro semidefinitní programování a další důležité třídy optimalizačních úloh, např. konvexní kvadratické programování. Více o nich čtenář najde například v úvodním článku

> T. Terlaky: An easy way to teach interior-point methods, *European Journal of Operational Research* 130,1(2001), 1–19,

v přehledovém článku

> F. A. Potra, S. J. Wright: Interior-point methods, *Journal of Computational and Applied Mathematics* 124(2000), str. 281–302,

(v době psaní tohoto textu byl dostupný na webových stránkách autorů), nebo v několika knihách tam citovaných. Mnoho materiálu je k dispozici na Internetu.

8

Software a další čtení

Nejznámější komerční (a drahý) program na řešení úloh lineárního a celočíselného programování se jmenuje CPLEX. Volně dostupné kódy s podobnými funkcemi, i když ne tak propracované, jsou například `lp_solve`, GLPK a CLP. Také knihovna geometrických algoritmů CGAL (`www.cgal.org`) obsahuje řešič úloh lineárního programování i konvexního kvadratického programování, a to jak v plovoucí čárce, tak i v přesné racionální aritmetice.

Webová stránka

`www-neos.mcs.anl.gov/neos/server-solver-types.html`

uvádí přehled řešičů, jimž se dá po síti poslat zadání úlohy a které (při troše štěstí) vrátí optimum.

Knih se slovy lineární programování v titulu je v nabídce knihkupectví Amazon víc než knih se slovem astrologie, takže je jasné, že z literatury můžeme zmínit jen úzký výběr.

Moderní a přístupná učebnice lineárního programování je

> D. Bertsimas and J. Tsitsiklis: *Introduction to Linear Optimization*, Athena Scientific, Belmont, Massachusetts, 1997.

Kniha

> J. Matoušek, B. Gärtner: *Understanding and Using Linear Programming*, Springer, Berlin 2006

je rozšířenou verzí textu, který právě čtete.

Ze starších zdrojů zmíníme tři. Velmi kvalitně a na vysoké teoretické úrovni pojednává lineární i celočíselné programování kniha

A. Schrijver: *Theory of Linear and Integer Programming*, Wiley-Interscience, New York 1986.

Za jednu z nejlepších učebnic se dodnes považuje

V. Chvátal: *Linear Programming*, W. H. Freeman, New York 1983.

A kdo má rád klasické prameny, může sáhnout po spisu

G. B. Dantzig: *Linear Programming and Extensions*, Princeton University Press, Princeton 1963.

V češtině je o lineárním programování k dispozici například skriptum

J. Rohn: *Lineární algebra a optimalizace*, Nakladatelství Karolinum, Praha 2004.

Obsahuje důkaz konečnosti Blandova pravidla a několik odkazů na další české učebnice.