Celočíselné lineární programování

František Včelař

Bakalářská práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**František Včelař**
Osobní číslo:	**A13133**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Celočíselné lineární programování**
Téma anglicky:	**Integer Linear Programming**

Zásady pro vypracování:

1. **Uveďte formulace úlohy, typové příklady, matematické modely, proveďte klasifikace úloh lineárního programování (LP).**
2. **Formulujte kanonický tvar úloh LP,vysvětlete převod nerovnic na rovnice, bázické řešení a simplexovou tabulku.**
3. **Vysvětlete primární a duální úlohu LP, uveďte aspekty kombinované úlohy a celočíselnosti.**
4. **Analyzujte a aplikujte metodu řezných nadrovin a porovnejte s metodou větví a mezí.**
5. **Vypracujte a využijte programovou podporu pro řešení celočíselných úloh LP.**
6. **Vytvořte aplikační příklady využití celočíselných úloh LP.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **GASS, Saul I. Linearne programovanie: metódy a aplikácie : príručka pre vysoké školy na Slovensku. preprac. vyd. Bratislava: Alfa, 1972, 397 s. Edícia elektrotechnickej literatúry (Alfa)**
2. **RYCHETNÍK, Luděk, Jan ZELINKA a Věra PELZBAUEROVÁ. Sbírka příkladů z lineárního programování. Vyd. 1. Praha: SNTL – Nakladatelství technické literatury, 1968, 314 s.**
3. **PLESNÍK, Ján, Jitka DUPAČOVÁ a Milan VLACH. Lineárne programovanie. 1. vyd. Bratislava: Alfa, vydavateľstvo technickej a ekonomickej literatúry, 1990, 314 s. Edícia matematicko-fyzikálnej literatúry. ISBN 80-05-00679-9**
4. **GRYGAROVÁ, Libuše. Úvod do lineárního programování, skripta, Státní pedagogické nakladatelství, Praha 1975, 1. vydání, skripta**
5. **DUPÁČOVÁ, Jitka. Lineární programování, Státní pedagogické nakladatelství, Praha 1982, 1. vydání, skripta**
6. **JABLONSKÝ, Josef. Operační výzkum: kvantitativní modely pro ekonomické rozhodování. 1. vyd. Brno: Professional Publishing, 2002, 323 s. ISBN 80-86419-23-1**
7. **LINDA, Bohdan a Josef VOLEK. Lineární programování. Vyd. 3. Pardubice: Univerzita Pardubice, 2009, 139 s. ISBN 978-80-7395-207-5**

Vedoucí bakalářské práce: **prof. Ing. Roman Prokop, CSc.**
Ústav matematiky

Datum zadání bakalářské práce: **15. prosince 2017**

Termín odevzdání bakalářské práce: **25. května 2018**

Ve Zlíně dne 15. prosince 2017

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis diplomanta

ABSTRAKT

Práce je věnována metodám řešení standardních úloh lineárního programování. V teoretické části jsou popsány základní algoritmy pro řešení neceločíselných úloh. Všechny algoritmy jsou popsány nejdříve zcela obecně, nicméně pro jejich lepší pochopení neformálně. Následně jsou demonstrovány na příkladech, které jsou vypracovány dostatečně podrobně na to, aby byl případný čtenář schopen řešit obdobné úlohy samostatně. Ve zcela stejném duchu jsou pak popsány dvě základní metody pro řešení celočíselných úloh – metoda Gomoryho řezů a metoda větví a mezí –, které jsou založeny na znalosti jejich neceločíselných řešení.

Praktická část nabízí jednoduchý program s přívětivým uživatelským prostředí pro řešení úloh popsaných v teoretické části. Je určen jednak k řešení obdobných úloh, ale především ke kontrole samostatně řešených úloh, ať již neceločíselných, tak celočíselných.

Klíčová slova: (celočíselné) lineární programování (ILP, LP), simplexový algoritmus, dualita, duální simplexový algoritmus, metoda Gomoryho řezů, metoda větví a mezí

ABSTRACT

The bachelor thesis is devoted to methods of solving of standard linear programming problems. The theoretical part contains descriptions of basic algorithms for non-integer problems. All the algorithms are at first described generally, but in an informal way for the sake of simple understanding. Then they are demonstrated by examples, which are elaborated in such a detail, so that the reader should be able to solve similar problems individually. In a similar manner two of the basic methods for solving of integer problems – Gomory's cut method and branch & bound method, which are based on the knowledge of their non-integer solutions – are described.

The practical part presents a simple program with a user-friendly environment for solving of the problems described in the theoretical part. It is intended for solving of similar problems and mainly for checking of results of individually solved integer or non-integer problems.

Keywords: (integer) linear programming (ILP, LP), simplex algorithm, duality, dual simplex algorithm, Gomory's cut method, branch & bound method

Poděkování. Chci velmi poděkovat panu profesoru Prokopovi za shovívavé, ovšem když bylo třeba také nekompromisní, odborné vedení mé bakalářské práce. Děkuji mu rovněž za jeho drahocenný čas, který mi vždy ochotně věnoval.

Vážený pane profesore, děkuji Vám mnohokrát

autor

OBSAH

ÚVOD

Lineární programování se začalo vyvíjet ve 30. letech 20. století během druhé světové války, vývoj byl zpočátku motivovaný potřebou řešit složité strategické problémy, související s vedením války, kde bylo snahou snížit celkové výdaje na provoz armády nebo zefektivnit jejich využití. Rychle rostoucí poválečná ekonomika a rozmach průmyslu zapříčinil urychlení vývoje lineárního programování, neboť mnoho průmyslových odvětví kladlo nároky na zefektivnění zavedených postupů, a to od logistických problémů, marketingu, rozvrhu investic až po technologické procesy. Spolu s rychle se vyvíjející teorií lineárního programování se rozšiřuje a zrychluje také vývoj výpočetní techniky, která se hojně využívá pro řešení daných úloh. Tento vývoj umožňuje řešit rozsáhlé úlohy ve stále kratším čase, což dále vede k rozšiřování oblasti využití. Dnes je již lineární programování vnímáno jako standardní nástroj optimalizace, a to nejen v průmyslu a armádě, ale i v medicíně, ekonomii, dopravě, telekomunikaci, marketingu a v celé řadě dalších oblastí.

Za zakladatele této disciplíny v její současné formě jsou obecně považováni George Bernard Dantzig, který v roce 1947 vytvořil a popsal simplexovou metodu a také předložil obecnou formulaci úlohy teorie lineárního programování, a John von Neumann, který tentýž rok založil teorii duality. Nicméně již před nimi se mnoha aspekty lineárního programování zabýval fenomenální sovětský matematik Leonid Vitalijevič Kantorovič, který v roce 1975 získal spolu s ekonomem Tjallingem Koopmansem Nobelovu cenu za ekonomii, a to především za jejich příspěvek k teorii optimálního rozdělování zdrojů, v níž mají úlohy lineárního programování zásadní význam.

Z historického pohledu je zajímavé, že sám Dantzig ve své knize [1] připouští, že kdyby byly Kantorovičovy články brány od počátku vážně, byl by patrně tehdejší Sovětský svaz jak v teorii, tak i v praxi lineárního programování daleko před USA (podle jeho osobního odhadu přibližně o deset let); samozřejmě také díky mimořádné Kantorovičově matematické erudici a talentu. Dnes je neuvěřitelné, že jeho pozoruhodná monografie z roku 1939 z oblasti lineárního programování byla odmítnuta proto, že byla v příkrém rozporu s marxistickou ideologií.

Za zmínku stojí i přínos české školy lineárního programování a operačního výzkumu. Co se týká speciálně lineárního programování, zmiňme alespoň některá jména. Za nestora české školy lineárního programování lze jistě považovat prof. Františka Nožičku. Z dalších jsou to např. prof. Karel Zimmermann, prof. Jitka Dupačová či doc Petr Lachout. Speciálně k celočíselnému programování přispěl světoznámý matematik českého původu prof. em. Václav

(Vašek) Chvátal, který obohatil zejména metodu Gomoryho řezů. Jen pro zajímavost dodejme, že posledně jmenovaný je jeden z mála matematiků s českými kořeny, který má Erdösovo číslo rovno 1.

Ačkoliv je vznik celočíselného lineárního programování pozdějšího data, i ono má svá velká jména. Americký matematik Ralph E. Gomory formuloval v roce 1958 první matematicky korektní konečnou metodu celočíselného programování. Jeho vědecká dráha je pozoruhodná zejména tím, že – až na dva roky strávené ve funkci lektora na Princetonu – po celou vědeckou kariéru pracoval v průmyslových výzkumných centrech, především u IBM. To mu mj. umožnilo podílet se jako matematikovi specialistovi i na dvou významných fyzikálních objevech, které byly oceněny Nobelovou cenou.

Významnou obecnou metodu větví a mezí, kterou lze použít také na nelineární celočíselné úlohy, formulovaly v roce 1960 anglické matematičky Ailsa Land*ová* a Alison Doig*ová*. Jejich metodu uvedl v život roku 1965 americký teoretický kybernetik R. J. Dakin [1].

Speciální metody pro řešení celočíselných úloh specifické struktury, založené na pokročilých výsledcích teorie matic, jsou spojovány se jmény maďarských matematiků Dénese Königa a E. Egerváryho (Egerwarry).

V roce 1948, když pracoval Dantzig pro US Air Force, vyšla jeho první práce *Programming in Linear Structures*. Každá činnost letectva vyžadovala svůj „programm“, počínaje leteckým útokem, přes transport pum, až po distribuci kancelářského papíru. V tom smyslu znamenalo slovo „programming“ hledání optimálního programu jakékoli vojenské aktivity. Téhož roku navrhl Koopmans pro tuto činnost název „linear programming“. O rok později navrhl Robert Dorfman obecnější pojem „mathematical programming“. Za všechny tyto termíny tedy vděčíme paradoxně vojenskému žargonu generálního štábu AF a nikoliv číslicovým počítačům nebo kybernetice. Naštěstí pro matematiky počítače záhy do těchto disciplín masivně vstoupily, takže nemuseli hledat vhodnější názvy. Ostatně i na tom měl nemalou zásluhu Dantzig, neboť byl mezi prvními, kdo se snažil přemluvit generalitu v Pentagonu, aby armáda neprodleně začala investovat nemalé prostředky do vývoje číslicových počítačů. O pozoruhodné historii se lze více dočíst v [1] nebo v prvním dílu knihy, kterou letmo zmiňujeme na str. 41.

[1] Tam, kde se nám nepodařilo dohledat křestní jména na webu, v citacích ani v originálních článcích, jsme byli nuceni ponechat jen jejich iniciály.

I. TEORETICKÁ ČÁST

1 TEORETICKÉ ZÁKLADY

1.1 Optimalizační úloha

Obecnou optimalizační úlohu lze formulovat velmi jasně a srozumitelně i s minimem speciální matematické symboliky a terminologie následovně. Je-li zadána reálná funkce reálných proměnných $f: M \to \mathbb{R}$, je třeba nalézt takové $x_0 \in M$, že $f(x_0) = \max_{x \in M} f(x)$, resp. $f(x_0) = \min_{x \in M} f(x)$ (hodnoty ***extrémů***). Tedy hledáme takové $x_0 \in M$, v němž nabývá funkce *f* své největší, příp. nejmenší hodnoty. Obvykle nás zajímá jak hodnota x_0, tak i hodnota extrému, v ojedinělých případech pouze některá z nich. Obecně však hodnota x_0 nemusí vůbec existovat, anebo jich naopak může existovat i více. Za této situace se optimalizační úloha obvykle rozpadá na dvě podúlohy:

(a) *ověřit, zda extrém existuje, příp. kolik existuje bodů z M, v nichž je extrém dosažen*;

(b) *pokud extrém existuje, tento extrém nalézt spolu s body, v nichž je extrému dosaženo*.

Samozřejmě úlohy (a) i (b) závisí jak na struktuře množiny *M*, tak také na tvaru funkce *f*. Odtud plyne, že v každém konkrétním případě je třeba použít, event. vybudovat specifické matematické metody a postupy, které jsou vhodné k jejich řešení.

Velmi specifické optimalizační úlohy řeší tzv. ***operační analýza***, resp. ***výzkum***. Tento obor lze (aspoň velmi přibližně) charakterizovat jako oblast aplikované matematiky, která se věnuje řešení ekonomických, logistických, organizačních či dokonce vojenských úloh. Praktické aplikace rovněž na mnoha místech ovlivnily i její terminologii. Např. o optimalizované funkci *f* se obvykle hovoří jako o ***účelové funkci***. Zvláštní součástí operační analýzy jsou pak ***úlohy lineárního programování*** (dále obvykle jen LP). Lineární programování je s mnoha svými modifikacemi a rozšířeními velmi rozsáhlou oblastí. Zde se budeme zabývat v podstatě jen nejzákladnějšími úlohami LP a metodami jejich řešení.

Co se týká výše zmíněných úkolů (a) a (b), lze říci, že lépe na tom již LP ani nemůže být. Předně je plně vyřešena otázka (a). Známe přesně strukturu množiny M a známe i přesně množinu těch jejích bodů, v nichž je dosaženo extrému, a jaký je její tvar. Důsledkem této skutečnosti je pak to, že úlohu (b) lze řešit přímo algoritmicky. Přitom algoritmus nejenže najde body optima, ale rovněž indikuje, kdy optimum neexistuje, čili zpětně odpovídá i na otázku (a). V této práci budeme uvažovat pouze klasický a historicky první tzv. ***simplexový***

algoritmus, příp***. simplexovou metodu***. Pochopitelně dnes již existuje mnoho variant a vylepšení tohoto algoritmu, jakož i dalších algoritmů, jež pracují na odlišných principech [12].

Z hlediska toho, jakým způsobem je specifikována množina M, na níž je optimalizace prováděna, se někdy uvádí následující členění optimalizačních úloh.

Úloha hledání volného extrému. V takovémto typu úloh je množina M chápána jako přirozený definiční obor funkce f, případně jeho vhodná podmnožina, a žádná další omezení se na množinu M nekladou. Tyto úlohy stály na samém počátku vývoje moderní matematiky, zejména infinitesimálního počtu.

Úloha hledání vázaného extrému. Na rozdíl od předchozí úlohy, je hledání vázaného extrému zúženo na určitou podmnožinu definičního oboru funkce. To znamená, že hledáme extrém při splnění dodatečných podmínek, které mají tvar rovností. Tyto rovnosti jsou pak nazývány *vazbami* úlohy, přičemž jejich počet je menší než počet samotných proměnných.

Hledáme tedy extrém funkce

$$f(x_1, \dots, x_n) \tag{1.1}$$

za podmínek

$$g_i(x_1, \dots, x_n) = 0 \tag{1.2}$$

$$i = 1, \dots, m < n \tag{1.3}$$

Problematika hledání vázaného extrému tvoří jednu z nejrozsáhlejší části operačního výzkumu a tou je matematické programování. Její prvopočátky však souvisely z velké části s řešením problémů klasické mechaniky. Úlohy tohoto typu vedly mj. ke značně univerzální a obecně známé metodě Lagrangeových multiplikátorů.

Úloha s omezeními ve tvaru nerovností. Tyto úlohy zásadním způsobem zobecňují předchozí typ úloh, neboť omezující podmínky jsou obecně ve tvaru nerovností, přičemž jejich počet není nijak vázán počtem proměnných. Úloha má pak následující tvar.

Hledáme extrém funkce

$$f(x_1, \dots, x_n) \tag{1.4}$$

za podmínek

$$g_i(x_1, \dots, x_n) \geq 0, \quad i = 1, \dots, m \tag{1.5}$$

$$x_j \geq 0, \quad j = 1, \dots, n \tag{1.6}$$

Jak dále uvidíme, úlohy LP mají nejčastěji tvar (1.4)–(1.6), příp. (1.1)–(1.3) s dodatečnou podmínkou nezápornosti všech proměnných. Úlohy tohoto typu vedly mj. k rozvoji zobecnění metody Lagrangeových multiplikátorů. Tato metoda je založena na splnění tzv. Kuhn-Tuckerových podmínek. Bod, který splňuje Kuhn-Tuckerovy podmínky, se nazývá sedlovým bodem. Základním výsledkem je zde tzv. Kuhn-Tuckerova věta o sedlovém bodu, která tvrdí, že optimum leží v sedlovém bodu, právě když je tento bod nezáporný. Tím je optimalizační úloha převedena na úlohu hledání nezáporného sedlového bodu.

Je zřejmé, že z čistě teoretického hlediska by bylo možné úlohy LP řešit výše uvedenými metodami, nicméně jejich řešení by bylo zdlouhavé a neefektivní. Zejména válečné a následně průmyslové potřeby, které vyžadovaly řešení rozsáhlých úloh, si vynutily specifické a podstatně efektivnější metody řešení úloh LP.

1.2 Modelové úlohy lineárního programování

V tomto odstavci stručně popíšeme několik typických úloh LP, z nichž budou patrny všechny základní rysy obecné úlohy LP. Jejich výběr je pouze tradiční, nikoli ovšem reprezentativní, neboť tento seznam by byl dnes velmi rozsáhlý. Z těch úloh, které zde nemůžeme uvést, zmiňme např. ty, které vznikají v souvislosti s teorií her, zejména maticových, nebo úlohy, vznikající v rámci řešení grafových problémů. Zejména posledně jmenované úlohy mají často technické aplikace, a tedy úlohy LP mají dnes i mnohá diametrálně odlišná uplatnění než jen ekonomická a logistická.

Dopravní úloha. Tato úloha, standardně rovněž nazývaná ***dopravní problém***, je historicky první, která byla formulována jako problém LP a která stimulovala celý jeho rozvoj. Je ovšem stále aktuální. Uvedeme jeho nejběžnější popis.

Je m dodavatelů a n odběratelů určitého produktu. Každý dodavatel má k dispozici $a_i \geq 0$ $(i = 1, \dots, m)$ jednotek produktu a každý odběratel naopak požaduje b_j $(j = 1, \dots, n)$ jeho jednotek. Dále jsou známy ceny $c_{ij} \geq 0$ za přepravu jednotky produktu od i-tého dodavatele k j-tému odběrateli. Úkolem je stanovit přepravní plán tak, aby

- kapacity všech dodavatelů byly vyčerpány a současně požadavky všech odběratelů byly uspokojeny;

- celkové náklady na dopravu byly minimální.

Přepravní plán je zřejmě plně popsán hodnotami $x_{ij} \geq 0$, které určují, kolik jednotek produktu je přepraveno od *i*-tého dodavatele k *j*-tému odběrateli. K jejich optimálnímu určení jsou tyto hodnoty chápány jako proměnné.

První požadavek vede ke dvěma typům podmínek. Předně k platnosti evidentní bilanční rovnice

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j. \tag{1.7}$$

Tato rovnost je předpokládána, ale není následně součásti formulace vlastní optimalizační úlohy. Pokud je uvedená rovnice splněna, hovoříme také podrobněji o *vyvážené dopravní úloze*. Zadruhé musí být splněno m rovnic $\sum_{j=1}^{n} x_{ij} = a_i$, které říkají, že kapacity všech dodavatelů jsou kompletně rozvezeny a n rovnic $\sum_{i=1}^{m} x_{ij} = b_j$, které stanoví, že poptávky všech odběratelů jsou plně uspokojeny. Celkové náklady na dopravu vyjadřuje účelová funkce $f(\mathbf{x}) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$, což je lineární funkce všech svých proměnných.

Druhý požadavek můžeme nyní zapsat jako následující ***úlohu minimalizace***:

$$\begin{aligned} &\min\left(f(\mathbf{x})\right) = \min\left(\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}\right) \\ &\sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1, \ldots, m \\ &\sum_{i=1}^{m} x_{ij} = b_j, \quad j = 1, \ldots, n \\ &x_{ij} \geq 0, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n \end{aligned} \tag{1.8}$$

Platí, že úloha (1.8) má řešení, právě když je splněna rovnice (1.7), viz např. [1, 3]. Nicméně lze řešit rovněž *úlohu nevyváženou*, a to s pomocí jistých „umělých“ kroků, které ji převedou na úlohu vyváženou. Úlohu (1.8) lze samozřejmě řešit již zmíněným a univerzálním simplexovým algoritmem. Nicméně díky velmi specifické struktuře dopravní úlohy vznikly již brzy po její formulaci speciální algoritmy, které dokáží řešit tuto úlohu podstatně efektivněji. Jeden z nejznámějších algoritmů je podle svých duchovních otců D. Königa a E. Egerváryho nazýván obvykle ***mad'arskou metodou***.

Směšovací úloha. Tyto úlohy mají širokou škálu uplatnění a zahrnují vskutku míchání nejrůznějších směsí tak, aby splňovaly předepsané minimální požadavky, a přitom měly i minimální cenu. Jedná se např. o krmné směsi s předepsanou minimální nutriční hodnotou, palivové směsi s minimálním stanoveným oktanovým číslem či směsi medikamentů s minimálním předepsaným množstvím bazálního léčiva. Na tyto úlohy lze ale převést např. i problém minimalizace odpadu při výrobě nebo rozhodování o velikosti plánovaných investic.

Obecně máme k dispozici m surovin, přičemž každá surovina obsahuje n složek v různých množstvích. Označíme a_{ij} množství *j*-té složky v jednotce *i*-té suroviny ($i = 1, \dots, m$, $j = 1, \dots, n$). Dále známe ceny c_i za jednotku *i*-té suroviny. Konečně b_j označují předepsaná (celková) minimální množství jednotlivých složek ve směsi. Pokud budou proměnné x_i znamenat celková množství surovin ve směsi, potom nalezení složení směsi s minimální cenou bude odpovídat následující minimalizační úloze:

$$\begin{aligned} &\min\left(f(\mathbf{x})\right) = \min\left(\sum_{i=1}^{m} c_i x_i\right) \\ &\sum_{i=1}^{m} a_{ij} x_i \geq b_j, \quad j = 1, \dots, n \\ &x_i \geq 0, \quad i = 1, \dots, m \end{aligned} \tag{1.9}$$

Úloha může obsahovat i některá další omezení, ale pro naše účely je tvar (1.9) postačující.

Úloha výrobního plánování. Tuto úlohu popíšeme nejdříve pokud možno co nejobecněji, poněvadž právě na ni lze převést velmi širokou třídu problémů nejrozličnějších ekonomických interpretací. Kromě toho s výhodou využijeme její tvar v souvislosti se simplexovým algoritmem – např. proto, že v ***simplexové tabulce***, která je jeho grafickým vyjádřením, automaticky obdržíme jednotkovou bázi a právě tento tvar budeme považovat (aspoň pro naše účely) za základní tvar úlohy LP. Tato úloha má rovněž nejblíže k celočíselné úloze LP.

Výrobce produkuje n různých výrobků, přičemž k jejich výrobě má k dispozici m omezených zdrojů. Přitom pojem zdroje je zde chápán velmi obecně (zdroji mohou být skladové zásoby primárních surovin, pracovní síly, energie, kapacity výrobních linek atd.). Výrobní plán je plně popsán nezápornými hodnotami x_i ($i = 1, \dots, n$), kde x_i určuje počet *i*-tého výrobku. Dále cenu *i*-tého výrobku označíme c_i. Hodnota b_j označuje dostupné množství *j*-

tého zdroje ($j = 1, \dots, m$). Nakonec a_{ij} značí množství *j*-tého zdroje, které je potřebné k výrobě jednoho kusu *i*-tého výrobku. Potom úlohu nalezení optimálního výrobního plánu lze zapsat jako ***úlohu maximalizace*** celkového zisku výrobce:

$$\max\big(f(\mathbf{x})\big) = \max\big(\sum_{i=1}^{n} c_i x_i\big)$$

$$\sum_{i=1}^{n} a_{ij}\, x_i \le b_j, \quad j = 1, \dots, m \tag{1.10}$$

$$x_i \ge 0, \quad i = 1, \dots, n \tag{1.11}$$

Ke zcela přirozeným podmínkám (1.11) se zejména v případě úlohy (1.10) přidávají ještě některé další, které blíže specifikují obory hodnot proměnných x_i. Jedná se především o požadavek, aby byly hodnoty x_i celočíselné: $x_i \in \mathbb{Z}$. Pokud výrobce vyrábí velké série laciných výrobků, potom obvykle stačí získané hodnoty x_i zaokrouhlit směrem dolů, přičemž zisk $f(\mathbf{x}) = \sum_{i=1}^{n} c_i x_i$ výrobce se změní jen minimálně. Pokud je ovšem výrobcem např. zbrojařský koncern, který vyrábí spíše menší série velmi drahých zbraňových systémů, pak začne být požadavek celočíselnosti mimořádně zásadní – jinak kompletnímu tanku nemůže chybět hlaveň, stíhačce jeden ze dvou motorů či jeden z několika kulometů atp. Jindy jsou motivace k řešení celočíselných úloh velmi pragmatické, např. tehdy, když výrobce může dodávat pouze jednotlivá balení výrobku.

Nakonec uvedeme jeden příklad na úlohu (1.10) a krátce tím budeme demonstrovat ***metodu grafického řešení*** úloh LP. Úlohy, které lze takto řešit, jsou víceméně banální povahy a sotva jimi vyřešíme skutečně podstatné problémy. Mají však svou heuristickou a pedagogickou hodnotu. Umožní studentovi, příp. budoucímu uživateli metod LP získat jistou představu o tvaru množiny, na které se optimalizace provádí (mnohdy pouze konvexní polyedr), o tvaru množiny bodů, v nichž se mohou vyskytovat extrémy (krajní body, či hrana, event. celá stěna tyto body obsahující), nebo o tom, kdy je, příp. naopak není úloha řešitelná.

Ukázka grafického řešení úlohy LP. Továrna vyrábí dva druhy granulátu pomocí dvou strojů v jednosměnném osmihodinovém provozu. Technologický proces výroby obou druhů granulátu vyžaduje zapojení obou strojů do výroby každého z nich. Vedení firmy požaduje optimalizovat rozvrh směny tak, aby byl zisk směny co nejvyšší. Pro výrobu 1 tuny prvního granulátu je zapotřebí 1 hodiny běhu stroje prvního a dalších 3 hodin stroje druhého, pro 1 tunu druhého granulátu je potřeba 3 hodin běhu prvního stroje a 2 hodin stroje druhého.

Konečně cena jedné tuny prvního granulátu je 2 000 Kč a druhého 3 000 Kč. Vše zapíšeme do následující tabulky:

	Granulát č.1	*Granulát č.2*	*Délka směny*
Stroj č.1	1 hodina	3 hodiny	8 hodin
Stroj č.2	3 hodiny	2 hodiny	8 hodin

Obr. 1. Grafické zobrazení dat úlohy LP *– mnohdy formou tabulky*

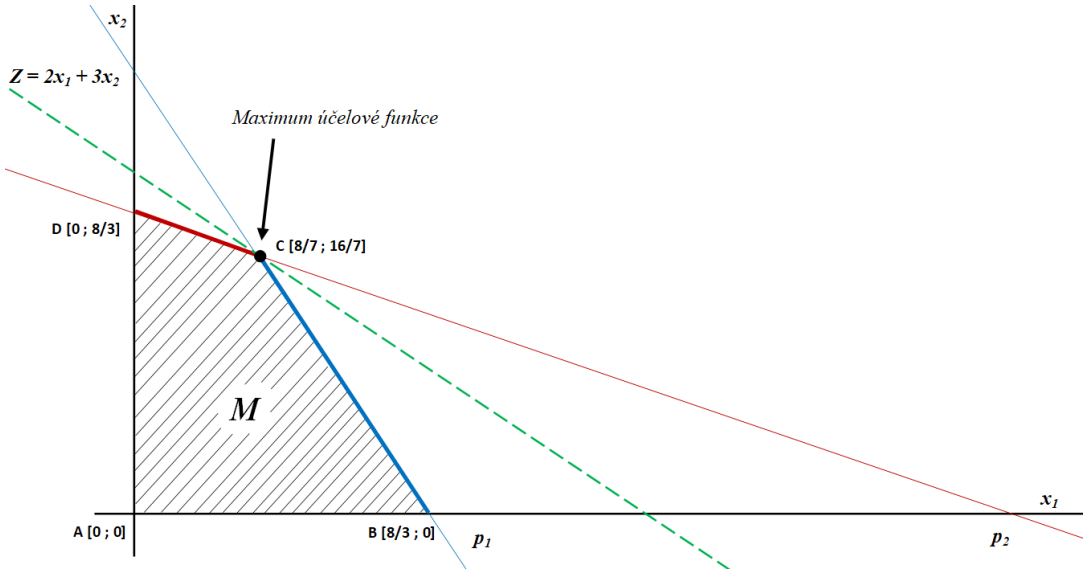
Abychom nemuseli přizpůsobovat měřítko, bereme za měnovou jednotku 1 000 Kč, na řešení úlohy to zřejmě nemá žádný vliv. Označují-li x_1 a x_2 vyrobená množství jednotlivých granulátů, vyjadřuje funkce $f(x_1, x_2) = 2x_1 + 3x_2$ zisk během jedné směny. Vyjádříme-li požadavky na omezení časových kapacit strojů ve tvaru nerovností, obdržíme následující optimalizační úlohu:

$$\max\left(f(x_1, x_2)\right) = \max\left(2x_1 + 3x_2\right)$$

$$\begin{aligned} x_1 + 3x_2 &\leq 8 \\ 3x_1 + 2x_2 &\leq 8 \end{aligned} \tag{1.12}$$

$$x_{1,2} \geq 0 \tag{1.13}$$

Na obr. 2 je zobrazena množina *M*, která je vymezena podmínkami (1.12) a (1.13).



Obr. 2. Ilustrace grafického řešení úlohy LP

Jestliže nyní do obrázku zobrazíme přímku $2x_1 + 3x_2 = c$ pro dostatečně velké $c > 0$, pak bude ležet nad M a bude s ní mít prázdný průnik. Bude-li se následně tato přímka pohybovat proti směru svého gradientu, bude se c spojitě zmenšovat, dokud se přímka nedotkne množiny M v bodě $\mathbf{C} = \left(\frac{8}{7}; \frac{16}{7}\right)$. V tomto bodě je pak evidentně dosaženo maxima funkce $f(x_1, x_2)$. Kompletní řešení lze shrnout takto:

$$\max\left(f(x_1, x_2)\right) = f\left(\frac{8}{7}, \frac{16}{7}\right) = 2\,.\frac{8}{7} + 3\,.\frac{16}{7} = \frac{64}{7} \approx 9{,}1429 \text{ [tisíce Kč]}$$

$x_1 + 3x_2 \leq 8 \quad \rightarrow \quad \frac{8}{7} + 3\,.\frac{16}{7} = 8$ – *podmínka je splněna jako rovnost*

$3x_1 + 2x_2 \leq 8 \quad \rightarrow \quad 3\,.\frac{8}{7} + 2\,.\frac{16}{7} = 8$ – *podmínka je splněna jako rovnost*

$x_1,\ x_2 \geq 0$ – *podmínka nezápornosti je splněna automaticky*

To, co nás pochopitelně zajímá nejvíce, uvádí první řádek: za stanovených omezení lze docílit maximálního zisku za jednu směnu přibližně 9 143 Kč.

1.3 Formulace základní úlohy lineárního programování

Úlohy z předchozího oddílu nám nyní umožňují provést prosté pozorování, na jehož základě jsme schopni formulovat úlohu LP naprosto obecně. Předně každá z optimalizovaných funkcí v úlohách (1.8)–(1.10) je lineární. Avšak rovněž tak všechna omezení, ať již ve tvaru rovnice či nerovnice, jsou lineární. Kromě toho nelze vyloučit ani možnost, že přímá interpretace úlohy vyžaduje některá omezení ve tvaru nerovností, jiná ve tvaru rovností. V dalším uvedeme obecnou úlohu LP formálně [2, 3, 8].

Obecnou úlohu LP, příp. ***úlohu ve smíšeném*** či ***kombinovaném tvaru*** lze zapsat následovně. Jsou-li $a_{ij}, b_i, c_j \in \mathbb{R}$ daná čísla, $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ a $\emptyset \neq I_1, I_2 \subseteq I$ jsou disjunktní podmnožiny (nikoli nutně neprázdné), pak požadujeme

$$\max\left(\sum_{j=1}^{n} c_j x_j\right), \quad \text{resp.} \quad \min\left(\sum_{j=1}^{n} c_j x_j\right) \tag{1.14}$$

$$\sum_{j=1}^{n} a_{ij}\, x_j \leq b_i, \quad i \in I_1 \tag{1.15}$$

$$\sum_{j=1}^{n} a_{ij}\, x_j \geq b_i, \quad i \in I_2 \tag{1.16}$$

$$\sum_{j=1}^{n} a_{ij}\, x_j = b_i, \quad i \in I \setminus (I_1 \cup I_2) \tag{1.17}$$

$$x_j \geq 0, \quad j \in \{1, \dots, n\} \tag{1.18}$$

Lineární funkci $f(\mathbf{x}) = f(x_1, \dots, x_n) = \sum_{j=1}^{n} c_j x_j$ nazýváme standardně ***účelovou funkcí***. Požadavky (1.15)–(1.17) jsou ***strukturální omezení***, příp. ***podmínky***. Požadavek (1.18) je pak ***podmínkou nezápornosti***.

V souladu s běžnými zvyklostmi se účelová funkce $z = f(x_1, \dots, x_n) = \sum_{j=1}^{n} c_j x_j$, příp. její hodnota označuje prostě jen „ z “; podobně např. optimální hodnotu účelové funkce budeme vždy značit „ $\tilde{z}$ “ (v tabulkách je obvyklé písmeno L). Protože nemůže dojít k nedorozumění, budeme zde tuto konvenci hojně využívat.

K uvedeným podmínkám (1.15)–(1.18) se kladou i některé další, tzv. ***nutné podmínky***. Pro nás nejdůležitější z těchto podmínek bude požadavek, aby všechny proměnné měly navíc celočíselné hodnoty: $x_j \in \mathbb{Z}$ pro $j = 1, \dots, n$. Jiným z těchto požadavků je např. podmínka, aby byly proměnné logického typu, neboli $x_j \in \{0,1\}$, $j = 1, \dots, n$. Konečně některé úlohy ze své podstaty vyžadují, aby nezáporné byly pouze některé proměnné. Z tohoto pohledu je pak nutnou podmínkou i požadavek, aby proměnné ze stanovené jejich podmnožiny mohly nabývat obecně libovolných reálných hodnot. Tento výčet je samozřejmě velmi neúplný.

Množinu M vektorů $\mathbf{x} \in \mathbb{R}^n$, které vyhovují podmínkám (1.15)–(1.18), nazýváme ***množinou přípustných řešení*** (úlohy LP). Již nyní je zřejmé, že množina M je uzavřená a konvexní. Rovnice (1.17) vyjadřují nadroviny a nerovnice (1.15), (1.16) a (1.18) vyjadřují uzavřené poloprostory. Množina M je tudíž průnikem konvexních uzavřených množin. Jedním z nejdůležitějších úkolů teorie LP bylo úplné popsání její struktury. Ukazuje se totiž, že právě struktura M je u úloh LP mnohem úžeji svázána s alokací a existencí optima, nežli je tomu u obecné optimalizační úlohy, kde jsou často mnohem podstatnější vlastnosti optimalizované funkce $f: M \to \mathbb{R}$ (např. diferencovatelnost atp.).

Hlavním cílem tohoto odstavce je klasifikace úloh LP, přičemž vyjdeme z obecné úlohy LP. Ovšem musíme být schopni vyjádřit úlohu v tzv. ***maticovém tvaru***. A k tomu zase potřebujeme pravidla, která nám umožní převod obecné úlohu (1.14)–(1.18) na jednotný tvar.

Pravidla přípustných transformací úloh LP

(1) Každou maximalizační úlohu lze převést **za stejných omezení** na úlohu minimalizační a naopak pomocí rovnic

$$-\max_{\mathbf{x} \in M} f(\mathbf{x}) = \min_{\mathbf{x} \in M} \big(-f(\mathbf{x})\big), \qquad -\min_{\mathbf{x} \in M} f(\mathbf{x}) = \max_{\mathbf{x} \in M} \big(-f(\mathbf{x})\big),$$

přičemž tyto rovnice platí vždy, pokud jeden z extrémů v nich existuje.

(2) Omezení (1.17) ve tvaru rovnosti můžeme nahradit dvěma současně platnými nerovnicemi $\sum_{j=1}^{n} a_{ij}\, x_j \leq b_i$ a $\sum_{j=1}^{n} a_{ij}\, x_j \geq b_i$, resp.

$$\sum_{j=1}^{n} a_{ij}\, x_j \leq b_i$$
$$\sum_{j=1}^{n} (-a_{ij})\, x_j \leq -b_i$$

(3) Omezení (1.15) ve tvaru nerovnosti převedeme na rovnost doplněním levé strany o nezápornou proměnnou $u_i \geq 0$ pro $i \in I_1$ na tvar

$$\sum_{j=1}^{n} a_{ij}\, x_j + u_i = b_i.$$

Proměnné u_i jsou k původní úloze dodány navíc a nazývají se ***doplňkové***, resp. ***skluzové***, příp. též ***přídatné*** (např. v úloze (1.10) bychom mohli tyto proměnné interpretovat jako rezervy ve zdrojích). Je důležité dodat, že pokud doplňkové proměnné potřebujeme, nijak se tím nezmění účelová funkce, poněvadž by v ní měly formálně tyto proměnné nulové koeficienty.

(4) Poslední pravidlo dodáváme spíše pro úplnost. Pokud bychom museli u některé proměnné x_j požadovat, aby mohla nabýt jakékoli reálné hodnoty (nikoli jen nezáporné), postačí vyjádřit ji jako rozdíl dvou nezáporných proměnných: $x_j = x_j^+ - x_j^-$. V tomto případě by se účelová funkce i všechna omezení musela změnit substitucí tohoto rozdílu do každého explicitního výskytu x_j. Hodnoty x_j, x_j^+, x_j^- mezi sebou nemají jednoznačný vztah, zjevně to ovšem nemůže mít žádný vliv na řešení původní úlohy.

Je důležité si uvědomit, že – na rozdíl od pravidel (1), (2) a (4) – pravidlo (3) není ekvivalentní v tom smyslu, že původní úloha má řešení, právě když má řešení úloha transformovaná. Nicméně transformovaná úloha vždy indikuje, zda má řešení úloha původní a pokud má, pak je jejím řešením část řešení úlohy transformované. Kromě toho, každé z uvedených pravidel (2)–(4) zvětšuje dimenzi úlohy buďto na straně proměnných, anebo na straně počtu omezení, což ovšem při dnešních možnostech výpočetní techniky není obvykle zásadním problémem (ale i nyní nikoliv bezvýhradně).

Již víme, že každou smíšenou úlohu LP můžeme převést např. na následující tvar (příp. při zvětšení počtu proměnných a omezení):

$$\begin{aligned} &\max\left(\sum_{j=1}^{n} c_j x_j\right) \\ &\sum_{j=1}^{n} a_{ij}\, x_j \le b_i, \quad i \in \{1, \dots, m\} \\ &x_j \ge 0, \quad j \in \{1, \dots, n\} \end{aligned} \tag{1.19}$$

Tuto úlohu nyní přepíšeme do maticového tvaru. Upozorněme, že v LP je obvyklé upřednostňovat sloupcové vektory před řádkovými. Důvod je jednoduchý: vyhneme se při zápisu a odvozování nadbytečným transpozicím, byť se jim nelze vyhnout zcela. Označme postupně

$$\mathrm{A} = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad \mathbf{0} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

Potom úloha (1.19) dostane následující velmi jednoduchý tvar, který ve zbytku práce budeme považovat za prioritní:

$$\begin{aligned} &\max\left(\boldsymbol{c}^{\mathrm{T}}\mathbf{x}\right) \\ &\mathrm{A}\mathbf{x} \le \boldsymbol{b} \\ &\mathbf{x} \ge \mathbf{0} \end{aligned} \tag{1.20}$$

Matici A nazýváme ***maticí* (*koeficientů*) *úlohy***, vektor $\boldsymbol{c}$ je ***vektor cen*** a vektor $\boldsymbol{b}$ je ***vektor omezení***. Úloha (1.20) se často označuje jako ***základní úloha* LP (*ve tvaru nerovností*)**. Důvody jsou patrně dva. Předně je tato úloha přesně ve tvaru velmi frekventované úlohy výrobního plánování (1.10). Zadruhé vzhledem k její ekonomické interpretaci lze mnohdy předpokládat, že všechny prvky matice A i vektory $\boldsymbol{b}$ a $\boldsymbol{c}$ jsou nezáporné. Pak je množina přípustných řešení v $\mathbb{R}^n$ uzavřená a omezená, tudíž kompaktní, a spojitá účelová funkce na ní musí nabýt maxima (lze dokázat, že množina přípustných řešení bude mít v takovém případě velmi specifický a jednoduchý tvar konvexního polyedru [3, 8, 12]).

Pochopitelně v závislosti na potřebách řešitele, lze na základní úlohu převést beze změny její dimenze kteroukoliv z následujících maximalizačních, resp. minimalizačních úloh:

$$\begin{aligned} &\max\left(\boldsymbol{c}^{\mathrm{T}}\mathbf{x}\right), \quad \mathrm{A}\mathbf{x} \ge \boldsymbol{b}, \quad \mathbf{x} \ge \mathbf{0} \\ &\min\left(\boldsymbol{c}^{\mathrm{T}}\mathbf{x}\right), \quad \mathrm{A}\mathbf{x} \ge \boldsymbol{b}, \quad \mathbf{x} \ge \mathbf{0} \end{aligned} \tag{1.21}$$

$$\min (\boldsymbol{c}^{\mathrm{T}}\mathbf{x}) , \quad \mathrm{A}\mathbf{x} \leq \boldsymbol{b} , \quad \mathbf{x} \geq \mathbf{0}$$

Zásadní význam v LP má maximalizační úloha ve **tvaru rovností**, v níž matice omezení obsahuje sloupcovou jednotkovou bázi. Pak hovoříme, že je ***úloha v kanonickém tvaru***. Úlohu (1.20) převedeme na kanonický tvar jednoduše tak, že původní matici omezení A doplníme o jednotkovou submatici I_m řádu m a do vektoru přípustných řešení přidáme m doplňkových proměnných. Tím úlohu (1.20) převedeme na následující kanonický tvar:

$$\begin{aligned} &\max (\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}}) \\ &\tilde{\mathrm{A}}\tilde{\mathbf{x}} = \boldsymbol{b} \\ &\tilde{\mathbf{x}} \geq \mathbf{0} \end{aligned} \tag{1.22}$$

Přitom $\tilde{\mathrm{A}} = (\mathrm{A}|\mathrm{I}_m)$ a $\tilde{\mathbf{x}} = (x_1, \dots, x_n, x_{n+1}, \dots, x_{n+m})^{\mathrm{T}}$, kde $x_{n+1}, \dots, x_{n+m}$ jsou doplňkové proměnné. Proměnné $x_1, \dots, x_n$, které odpovídají původní úloze (1.20), se v této souvislosti nazývají ***základní proměnné***.

Úloha (1.22) má mnoho užitečných vlastností. Co se týká teorie, mnoho obecných výsledků je odvozeno právě pro úlohu (1.22), přičemž ne všechna pro nás důležitá tvrzení platí rovněž pro úlohy ve tvarech (1.20), resp. (1.21). Pro praxi je pak podstatná zejména ta skutečnost, že v úloze (1.22) lze provádět elementární řádkové úpravy s jejími omezeními, aniž by se změnila množina přípustných řešení. Právě této skutečnosti využívá s výhodou simplexový algoritmus.

Nakonec připojme krátce popis dvou speciálních úloh s nutnými podmínkami, z nichž řešení první je hlavním cílem této práce.

Celočíselná úloha LP. V mnoha případech vyžaduje sama podstata úlohy, aby (alespoň některé) proměnné nabývaly jen celočíselných hodnot; nechť pro jednoduchost $x_j \in \mathbb{Z}$ pro $j = 1, \dots, n$. Tato podmínka přibude k ostatním omezením kterékoliv z úloh (1.20)–(1.22), přičemž způsobí to, že řešení takové úlohy bude ležet uvnitř množiny přípustných řešení úlohy, která tato omezení nemá. Podstatné ovšem je, že v závislosti na poloze nadroviny, která reprezentuje účelovou funkci (přesněji na směru gradientu účelové funkce, který je na ni kolmý), nemusí optimální řešení ležet v nejbližším celočíselném okolí neceločíselného řešení. Pouhým zaokrouhlením se hodnota účelové funkce příliš změnit nemusí, může však být daleko od hodnoty v optimálním celočíselném řešení.

Je zřejmé, že metody vyvinuté pro řešení neceločíselných úloh nejsou dostatečné. Proto se začala rozvíjet teorie i metody speciální, a to až do té míry, že vznikla celá samostatná disciplína zvaná ***celočíselné lineární programování***. Dále zde popíšeme jeho dvě standardní metody, které vycházejí z neceločíselných řešení: ***metodu*** (***Gomoryho***) ***řezů*** (také nazývanou ***metoda sečných nadrovin***) a ***metodu větví a mez***í.

Binární úloha LP. Ještě více specializovanou třídou celočíselného programování jsou ***binární úlohy***, tj. úlohy, které mají logické proměnné: $x_j \in \{0,1\}$ pro $j = 1, \dots, n$. V tomto případě je podmínka nezápornosti zřejmě redundantní. Příkladem může být např. tzv. ***přiřazovací problém***, který je speciálním případem dopravní úlohy (1.8), ačkoliv obvykle ve formě maximalizační úlohy. V této úloze je $m = n$ a $a_i = b_j = 1$ pro všechny hodnoty indexů *i* a *j*:

$$\max\left(\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}\right)$$

$$\sum_{j=1}^{n} x_{ij} = 1,$$
$$\sum_{i=1}^{n} x_{ij} = 1, \quad i,j = 1, \dots, n$$

$$x_{ij} = 0,1, \quad i,j = 1, \dots, n$$

Tato úloha má čtvercovou matici s výraznou blokovou strukturou, a tudíž neobsahuje jednotkovou sloupcovou bázi, jako je tomu u kanonického tvaru (1.22). Ačkoliv bychom ji i přesto mohli řešit obecnými metodami celočíselného LP, bylo by toto řešení nesmírně těžkopádné a u rozsáhlých úloh dokonce neschůdné. Nicméně právě struktura matice umožnila vyvinout nesrovnatelně efektivnější algoritmy pro řešení této úlohy, které jsou ovšem založeny na zcela jiných matematických principech nežli výše zmíněné obecné metody, totiž na kombinatorických vlastnostech matic. Teoretické základy k těmto metodám poskytli ještě před samotným vznikem LP maďarští matematikové E. Egerváry (Egerwarry) (1931) a D. König (1936).

K interpretaci této úlohy odkazujeme např. na [1, 2, 12].

1.4 Základní vlastnosti úlohy lineárního programování

V tomto oddílu uvedeme všechna tvrzení, která jsou potřebná k algoritmickému řešení úloh LP, tj. k simplexové metodě. Všechny zde uvedené věty lze nalézt často v úplnějším a komplexnějším tvaru, a to i včetně důkazů, v [3, 8, 12], některé pak rovněž v [1].

Všechna tvrzení jsou formulována pro obecnou úlohu ve tvaru rovností, spec. tedy pro úlohu (1.22), na kterou se budeme odkazovat jednoduše jako na „úlohu LP“. Pokud budou tvrzení platit rovněž pro úlohy (1.20) a (1.21), a bude-li to vhodné, upozorníme na to.

Věta 1. (a) *Množina přípustných řešení úlohy* LP *je uzavřená a konvexní a má konečný počet krajních bodů.*
(b) *Má-li úloha* LP *optimální řešení, potom alespoň jedno optimální řešení je krajním bodem množiny přípustných řešení.*
(c) *Pokud je množina přípustných řešení omezená, je množina všech optimálních řešení konvexním obalem množiny všech těch optimálních řešení, která jsou krajními body množiny přípustných řešení.*

Věta 1 neříká nic jiného nežli to, že při hledání optima úlohy LP **se lze omezit výhradně na krajní body** množiny přípustných řešení, kterých je vždy konečný počet. Tato věta platí v plném rozsahu rovněž pro úlohy (1.20) a (1.21).

Obvykle se rovněž předpokládá, že hodnost matice $\mathrm{A} \in \mathbb{R}^{m \times n}$ úlohy (1.20) má plnou řádkovou hodnost, tj. **její hodnost je rovna počtu omezení**:

$$h(\mathrm{A}) = m \tag{1.23}$$

Podmínka (1.23) říká, že ze sloupců matice A lze vybrat m sloupců, jež jsou lineárně nezávislé, čili tvoří bázi, kterou lze převést na jednotkovou bázi pouze elementárními úpravami na řádky matice A Jordanovou eliminací.

Bazické přípustné řešení úlohy LP je takové přípustné řešení, v němž indexům jeho nenulových složek odpovídají v matici $\widetilde{\mathrm{A}}$ lineárně nezávislé sloupce s týmiž indexy. Uvedený název souvisí s tím, že zmíněné nezávislé sloupce lze vždy doplnit některými sloupci matice $\widetilde{\mathrm{A}}$ tak, aby tvořily bázi. Kladné složky x_i řešení **x** jsou ***bazické proměnné***, nulové jsou pak ***nebazické proměnné***. Podmínka (1.23) tedy zaručuje, že pokud $m = n$, všechny základní proměnné se mohou stát bazickými.

Zásadní důležitost má následující charakterizační tvrzení.

Věta 2. *Bod je bazickým přípustným řešením úlohy* LP*, právě když je krajním bodem množiny přípustných řešení.*

Věta 2 platí pouze pro úlohu (1.22) ve tvaru rovností. Zřejmě také definice bazického řešení má smysl jenom v tomto případě.

Věty 1 a 2 spolu tvrdí, že při hledání optima úlohy LP **se lze omezit výhradně na bazická přípustná řešení**. Tato skutečnost vyjadřuje podstatu simplexového algoritmu. V hrubých rysech můžeme říci, že podle jistých pravidel postupujeme od jednoho přípustného bazického řešení k některému sousednímu takovým způsobem, že se v každém kroku zvýší hodnota účelové funkce, dokud není dosaženo maxima. Počet bazických řešení, tudíž i krajních bodů množiny přípustných řešení, je vzhledem k (1.23) u úlohy (1.22) zřejmě shora omezen kombinačním číslem $\binom{m+n}{m} = \binom{m+n}{n}$, což je také maximální počet různých kroků simplexového algoritmu. V literatuře se nicméně uvádí, že běžný počet kroků se pohybuje v mezích m a $3m$ (viz [12]), což je „velmi rozumná“ efektivnost.

Tvrzení (b) věty 1 a věta 2 tvoří základ prakticky všech obecných metod řešení úloh LP (snad kromě těch, které jsou vyvinuty pro řešení úloh zcela specifické struktury) a vyjadřují jednu z možných variant tzv. ***základní věty lineárního programování***, kterou lze shrnout do následujícího tvrzení.

Věta 3. *Má-li úloha* LP *optimální řešení, potom má i optimální bazické řešení. Pokud existuje jediné optimální řešení, je bazické.*

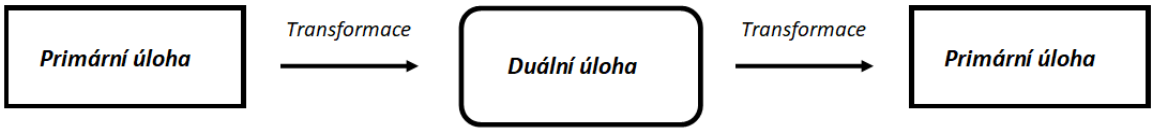
Závěrem tohoto odstavce uvedeme několik obecných poznámek, které se vztahují k našemu hlavnímu tématu poněkud volněji. Především v souvislosti s výše uvedenými větami je jasné, že algoritmus, řešící úlohu LP, musí startovat v některém bazickém přípustném řešení, které nemusí být automaticky k dispozici. Pak je nutno nejdříve vyřešit pomocnou úlohu, která takové řešení poskytne (čímž vzniká tzv. dvoufázový simplexový algoritmus). V našem případě to ale není nezbytné, poněvadž naším cílem je řešení základní úlohy (1.20), kterou převádíme na úlohu (1.22) doplněním původní matice úlohy o jednotkovou submatici I_m. Tím dostáváme automaticky jedno bazické přípustné řešení $\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{b} \end{pmatrix}$, samozřejmě pokud $\boldsymbol{b} \geq \mathbf{0}$, což je však u kanonických úloh obvyklý dodatečný předpoklad.

Bazické přípustné řešení je ***nedegenerované***, pokud má právě m kladných složek, v opačném případě, tj. má kladných složek méně, je ***degenerované***. Úloha LP je ***nedegenerovaná***, pokud má všechna bazická přípustná řešení nedegenerovaná, jinak je ***degenerovaná***. Zdůrazněme, že pojem nedegenerovanosti, resp. degenerovanosti má smysl pouze pro úlohy

typu (1.22) ve tvaru rovností, ačkoli se o nich následně hovoří i v souvislosti s odpovídající úlohou (1.20) ve tvaru nerovností. Degenerovanost má některé negativní stránky, zapříčiňuje především problém se zacyklením řešících algoritmů. Všechny tyto problémy lze ovšem vždy vhodným způsobem obejít. Je jasné, že postačující podmínkou degenerovanosti je nerovnost $h(\mathrm{A}) = m < n$. Nicméně tato podmínka není postačující, neboť degenerovanost je velmi komplexní jev.

1.5 Primární a duální úloha lineárního programování

V LP existuje velmi důležitý vzájemně jednoznačný vztah mezi maximalizačními a minimalizačními úlohami – ***dualita*** (tato korespondence ovšem nemá nic společného se skutečností, že se tyto úlohy dají navzájem převádět podle pravidla (1) z odd. 1.3). Každou maximalizační úlohu lze podle stanovených pravidel převést na minimalizační a naopak. Zcela obecně jsou tato pravidla popsána v [12]. Úloha dualitou převáděná je ***primární***, úloha transformovaná je ***duální***. Vztah duality je symetrický v tom smyslu, že duální úloha k úloze duální musí poskytnout opět úlohu primární v přesně původním tvaru. Tato vlastnost je pro dualitu fundamentální. Umožňuje nám např. dodatečnou analýzu a kontrolu výsledků primární úlohy, příp. řešit úlohu duální, jestliže je to z jistých důvodů vhodnější, a rovněž tvorbu speciálních algoritmů. Někdy je dokonce vhodné řešit duální úlohu proto, že má předem k dispozici tzv. duálně přípustné řešení, zatímco u primární úlohy přípustné bazické řešení ihned k dispozici nemáme. Dualita má pochopitelně také samostatnou teoretickou hodnotu.



Obr. 3. Ilustrace vztahu primární a duální úlohy

Pro všechny naše další účely však stačí popsat tzv. symetrickou dualitu pro základní úlohu (1.20) v maticovém tvaru podle následujícího schématu:

Primární úloha:	⟶	***Duální úloha:***	
$\max\ (\boldsymbol{c}^{\mathrm{T}}\mathbf{x})$		$\min\ (\boldsymbol{b}^{\mathbf{T}}\mathbf{y})$	
$\mathrm{A}\boldsymbol{x} \leq \boldsymbol{b}$		$\mathrm{A}^{\mathrm{T}}\mathbf{y} \geq \boldsymbol{c}$	(1.24)
$\mathbf{x} \geq \mathbf{0}$		$\mathbf{y} \geq \mathbf{0}$	

Poněvadž $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$, $\mathrm{A} \in \mathbb{R}^{m \times n}$ a $\mathrm{A}^{\mathrm{T}} \in \mathbb{R}^{n \times m}$, všechny dimenze u obou dvou úloh jsou kompatibilní s příslušnými maticovými operacemi a vznik duální úlohy z primární lze jednoduše popsat pravidlem: „otoč, co lze, vyměň, co jde“, tedy kromě podmínky nezápornosti. Abychom demonstrovali alespoň v našem případě vztah duality uvedený na obr. 3, stačí napsat následující jednoduché ekvivalence a využít duality (1.24):

$$\begin{array}{c} \min\ (\boldsymbol{b}^{\mathrm{T}}\mathbf{y}) \\ \mathrm{A}^{\mathrm{T}}\mathbf{y} \geq \boldsymbol{c} \\ \mathbf{y} \geq \mathbf{0} \end{array} \quad \Leftrightarrow \quad \begin{array}{c} \max\ (-\boldsymbol{b}^{\mathrm{T}}\mathbf{y}) \\ -\mathrm{A}^{\mathrm{T}}\mathbf{y} \leq -\boldsymbol{c} \\ \mathbf{y} \geq \mathbf{0} \end{array} \quad \overset{DUALITA}{\Longrightarrow} \quad \begin{array}{c} \min\ (-\boldsymbol{c}^{\mathrm{T}}\mathbf{x}) \\ (-\mathrm{A}^{\mathrm{T}})^{\mathrm{T}}\mathbf{x} \geq -\boldsymbol{b} \\ \mathbf{x} \geq \mathbf{0} \end{array} \quad \Leftrightarrow \quad \begin{array}{c} \max\ (\boldsymbol{c}^{\mathrm{T}}\mathbf{x}) \\ \mathrm{A}\mathbf{x} \leq \boldsymbol{b} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

Vztah mezi úlohami se mnohdy interpretuje v souvislosti s úlohou výrobního plánování. V primární úloze reprezentuje $\mathbf{x} \geq \mathbf{0}$ výrobní plán. Zeptejme se nyní, zda by se výrobci nakonec nevyplatilo zdroje pouze zpeněžit a nic nevyrábět. Nechť vektor $\mathbf{y} \geq \mathbf{0}$ označuje relativní ceny za jednotková množství jednotlivých zdrojů. Zřejmě logický je předpoklad, že výrobce nechce prodělat tím, že některý z výrobků nebude vyrábět, což vyjadřují omezení $\mathrm{A}^{\mathrm{T}}\mathbf{y} \geq \boldsymbol{c}$. Jeho celkový zisk z pouhého prodeje je potom $\boldsymbol{b}^{\mathrm{T}}\mathbf{y}$. Za těchto podmínek bude navíc platit, že $\boldsymbol{c}^{\mathrm{T}}\mathbf{x} \leq \boldsymbol{b}^{\mathrm{T}}\mathbf{y}$ (viz následující větu 4). Minimalizace v duální úloze představuje rovněž logický požadavek, že výrobce to se svými představami o míře zisku nesmí příliš přehnat v duchu zásady „kdo má velké oči, nemá nakonec nic“. Pokud bude za všech okolností $\boldsymbol{c}^{\mathrm{T}}\mathbf{x} < \boldsymbol{b}^{\mathrm{T}}\mathbf{y}$, pak ztrácí primární úloha smysl (nebude řešitelná) a výrobce může prodávat zdroje za ceny, které jsou na trhu aspoň přijatelné. Pokud ovšem budou existovat $\tilde{\mathbf{x}}$ a $\tilde{\mathbf{y}}$ taková, že nastane ekvilibrium $\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}} = \boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}}$, pak se mu zajisté vyplatí vyrábět přinejmenším z toho důvodu, že výrobek je prodejnější nežli zdroj. Taková $\tilde{\mathbf{x}}$ a $\tilde{\mathbf{y}}$ současně řeší obě úlohy, jak primární, tak i duální. Ukazuje se, že tato rovnost je nejen nutná, ale také postačující pro současnou řešitelnost obou úloh, což představuje zcela klíčová vlastnost duality.

Než zformulujeme základní vlastnosti duality, uvedeme jeden ilustrační příklad vzájemně duálních úloh.

Grafická ukázka vzájemného převodu duálních úloh.

Primární úloha: ***Duální úloha:***

$\max\ (\boldsymbol{c}^{\mathrm{T}}\mathbf{x}) = \max\ (3x_1 + 2x_2 + x_3)$ $\min\ (\boldsymbol{b}^{\mathrm{T}}\mathbf{y}) = \max\ (1200y_1 + 3500y_2 + 5000y_3)$

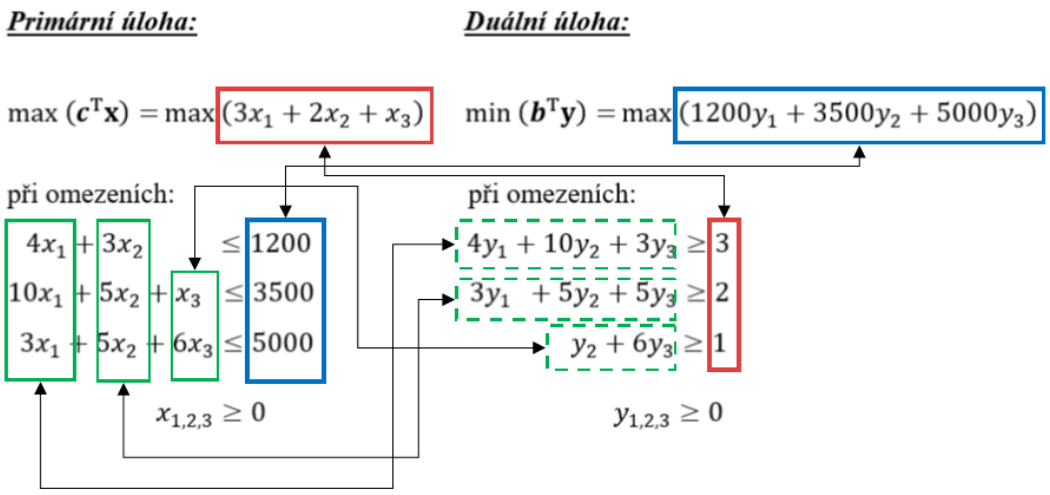
při omezeních:

$$\begin{aligned} 4x_1 + 3x_2 &\leq 1200 \\ 10x_1 + 5x_2 + x_3 &\leq 3500 \\ 3x_1 + 5x_2 + 6x_3 &\leq 5000 \\ x_{1,2,3} &\geq 0 \end{aligned}$$

při omezeních:

$$\begin{aligned} 4y_1 + 10y_2 + 3y_3 &\geq 3 \\ 3y_1 + 5y_2 + 5y_3 &\geq 2 \\ y_2 + 6y_3 &\geq 1 \\ y_{1,2,3} &\geq 0 \end{aligned}$$

Vzájemný převod úloh lze v tomto případě zanést do tohoto obrázku:



Obr. 4. Ukázka transformace primární úlohy na úlohu duální

Následující věta formuluje všechny pro nás podstatné vlastnosti duality.

Věta 4. (a) *Pro libovolná přípustná řešení úloh* (1.24) *vždy platí* $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$.
(b) *Má-li optimální řešení kterákoliv z úloh* (1.24)*, potom má optimální řešení také druhá úloha.*
(c) *Primární úloha má optimální řešení* $\tilde{\mathbf{x}}$ *a duální úloha má optimální řešení* $\tilde{\mathbf{y}}$*, právě když platí rovnost* $\mathbf{c}^T\tilde{\mathbf{x}} = \mathbf{b}^T\tilde{\mathbf{y}}$.

Tvrzení (a) věty 4, je triviálním důsledkem definice vzájemně duálních úloh a základních vztahů z maticové algebry:

$$\mathbf{c}^T\mathbf{x} \leq (A^T\mathbf{y})^T\mathbf{x} = \mathbf{y}^T(A^T)^T\mathbf{x} = \mathbf{y}^TA\mathbf{x} \leq \mathbf{y}^T\mathbf{b} = (\mathbf{y}^T\mathbf{b})^T = \mathbf{b}^T\mathbf{y}$$

Toto tvrzení má však mnohé teoretické důsledky, a proto se nazývá ***slabá věta o dualitě***. Např. z ní ihned plyne, že má-li primární úloha shora neomezenou účelovou funkci, potom je množina přípustných řešení duální úlohy prázdná, neboť její účelová funkce by musela mít „nekonečnou“ hodnotu v libovolném přípustném řešení. Platí samozřejmě také obdoba pro neomezenost zdola u duální úlohy.

Tvrzení (b), které je důsledkem slabé věty o dualitě, je ***silná věta o dualitě***. Tvrzení (c) má mj. význam v tom, že ať již získáme vektory $\tilde{\mathbf{x}}$ nebo $\tilde{\mathbf{y}}$ jakkoliv, třeba i pouhým hádáním, platí-li rovnost hodnot účelových funkcí, budeme mít jistotu, že jsou optimálními řešeními obou úloh.

Závěrem zmiňme, že zejména u rozsáhlých úloh je vhodné řešit obě úlohy pro kontrolu. Díky numerickému řešení může totiž dojít k jistým odchylkám od přesného řešení. Má tedy smysl obě řešení vzájemně porovnat.

2 SIMPLEXOVÝ ALGORITMUS

2.1 Primární simplexový algoritmus

Ačkoli již dříve byly úlohy LP řešeny metodami, které měly jen krůček k algoritmizaci, první skutečně algoritmický postup – ***simplexovou metodu*** – navrhl (a snad i pojmenoval) v roce 1947 americký matematik George Bernard Dantzig pro potřeby letectva USA. Z pochopitelných důvodů utajení ji ovšem mohl publikovat až v letech 1949–51. Její název vychází ze skutečnosti, že množiny přípustných řešení prvních úloh LP řešených touto metodou měly právě tvar simplexů. Pro úplnost zmiňme, že ***simplexy*** – jak napovídá již jejich název – jsou geometricky nejjednodušší útvary, které mají stejnou dimenzi, jakou má euklidovský prostor $\mathbb{R}^n$, v němž leží; tedy v přímce je to úsečka, v rovině trojúhelník, v trojrozměrném prostoru čtyřstěn, a to i včetně svých vnitřků atd. (existuje samozřejmě jejich přesná matematická definice). Obr. 2 na str. 17 však ukazuje, že již v triviálních případech bývá množina přípustných řešení znatelně komplikovanější. V průběhu let bylo učiněno několik pokusů metodu nazvat vhodněji, avšak tradice byla silnější, takže si ponechala svůj původní název, byť dnes nemá se simplexy prakticky nic společného.

Každá z metod celočíselného programování, které dále popíšeme, je založena na znalosti řešení úlohy bez omezení celočíselnosti. Nejdříve tedy musíme být schopni řešit tuto úlohu, o níž se často hovoří jako o ***úloze relaxované***. V našem případě lze k jejímu řešení vždy použít ***primární simplexový algoritmus***, jehož jednotlivé kroky se zaznamenávají do ***simplexových tabulek***. Základními podmínkami k jeho inicializaci je to, aby byla řešená úloha ve tvaru rovností a matice omezení obsahovala jednotkovou bázi. Jednotková sloupcová báze obsažená v matici úlohy, nikoli nutně tvořící submatici, se nazývá ***kanonická báze***.

Obě výše uvedené podmínky jsou splněny, je-li úloha (1.20) převedena na kanonický tvar (1.22). Předpokládejme tedy, že řešíme úlohu (1.22) v kanonickém tvaru. Základní myšlenka simplexového algoritmu plyne ze základní věty LP. Každá simplexová tabulka reprezentuje jedno bazické přípustné řešení. Jeden krok simplexového algoritmu transformuje tuto tabulku na novou tak, aby reprezentovala bazické přípustné řešení, v němž dojde k relativně největšímu přírůstku účelové funkce, je-li v bázi vyměněn jeden sloupec. Má-li úloha optimální řešení, pak jej algoritmus po konečném počtu kroků nutně nalezne.

Jeden krok algoritmu je realizován Jordanovou eliminací na řádcích vzhledem ke zvolenému prvku a_{ij} rozšířené matice soustavy omezení

$$\left(\widetilde{\mathrm{A}}\middle|\boldsymbol{b}\right)=\left(\begin{array}{ccc|c} a_{11} & \cdots & a_{1(n+m)} & b_1 \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & a_{ij} & \vdots & b_i \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & \cdots & a_{m(n+m)} & b_m \end{array}\right) \tag{2.1}$$

tak, že a_{ij} je změněn na 1 a ostatní prvky *j*-tého sloupce se vynulují. ***Prvek*** a_{ij}, který se nazývá ***klíčový***, nelze ovšem zvolit libovolně.

Nejdříve musíme umět vybrat ***klíčový sloupec***. K tomu učiníme následující jednoduchou úvahu, v níž budeme pro jednoduchost brát nedegenerovanou úlohu. Označme $\{j_1,\ldots,j_m\}=$ **j** množinu indexů sloupců, které tvoří kanonickou bázi a $\tilde{\mathbf{x}}$ jí příslušné přípustné bazické řešení, tj. $\tilde{\mathbf{x}}$ má kladné složky s indexy $j_l \in \mathbf{j}$, ostatní složky jsou nulové. Nyní chceme zaměnit jeden prvek stávající kanonické báze za sloupec s indexem $j \notin \mathbf{j}$. Jakmile se nulová proměnná $\tilde{x}_j$ zvětší o jistou přípustnou hodnotu $h>0$, jež vynuluje jednu bazickou proměnnou, potom se původní bazické složky řešení musí změnit na hodnoty $\tilde{x}_{j_l}-h$, neboť musí být zachovány rovnosti omezení: $\widetilde{\mathrm{A}}\tilde{\mathbf{x}}=\boldsymbol{b}$. Označíme-li takto vzniklé řešení $\tilde{\mathbf{x}}_j$, pak se hodnota účelové funkce změní o přírůstek

$$\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}}_j-\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}}=\left(\sum_{l=1}^{m}c_{j_l}\left(\tilde{x}_{j_l}-h\right)+c_j h\right)-\sum_{l=1}^{m}c_{j_l}\tilde{x}_{j_l}=$$

$$=\left(c_j-\sum_{l=1}^{m}c_{j_l}\right)h=-\left(\sum_{l=1}^{m}c_{j_l}-c_j\right)h \tag{2.2}$$

Hodnoty v poslední závorce (2.2) jsou tzv. ***relativní ceny*** a je zvykem značit je $\left(z_j-c_j\right)$. Co se týká jejich znaménka, poznamenejme, že se jedná jen o konvenci, jež nám umožňuje v simplexové tabulce odečíst se správným znaménkem příslušnou hodnotu účelové funkce. Je ovšem podstatné, že relativní ceny určují – tedy až na své znaménko – tu proměnnou, resp. jí příslušný sloupec, u níž je nejvyšší relativní přírůstek hodnoty účelové funkce. Tím máme dáno pravidlo, kterým je identifikován klíčový sloupec:

$$\textit{klíčový je j-tý sloupec, pro který}\quad j=\arg\max_{l}\left\{z_l-c_l \mid z_l-c_l<0\right\} \tag{2.3}$$

Pokud je takových řádků více, lze zvolit kterýkoliv z nich (alespoň prozatím).

Dále je nutno identifikovat ***klíčový řádek***, v němž pak leží klíčový prvek. Jediným požadavkem je zde to, aby poslední sloupec v (2.1) zůstal po Jordanově eliminaci nezáporný. Zřejmě musí být $a_{ij} > 0$ a podíl $\frac{b_i}{a_{ij}}$ musí být minimální. Snadno lze ukázat, třeba pouze na triviálních numerických experimentech, že v jakémkoliv jiném případě se objeví v posledním sloupci alespoň jedna záporná složka. Tedy klíčový řádek vybíráme podle pravidla:

$$\textit{klíčový je i-tý řádek, pro který} \quad i = \arg\min_k \left\{ \frac{b_k}{a_{kj}} \,\middle|\, a_{kj} > 0 \right\} \tag{2.4}$$

Pravidla (2.3) a (2.4) identifikují klíčový prvek a_{ij}. Poté proběhne jeden krok Jordanovy eliminace na celou simplexovou tabulku. Tento postup se opakuje, dokud nejsou všechny relativní ceny nezáporné. Tím algoritmus končí, neboť jakoukoliv další volbou nové bazické proměnné nelze již hodnotu účelové funkce zvýšit.

Než uvedeme konkrétní příklady, je nutno připojit několik poznámek. Předně (2.2) platí pouze pro nebazickou proměnnou $\tilde{x}_j$. Záměna jedné bazické proměnné za druhou bazickou proměnnou k žádné změně účelové funkce nepovede, a tudíž relativní cena každé bazické proměnné $\tilde{x}_j$ je nulová. Dále při Jordanově eliminaci se v simplexové tabulce automaticky přepočtou relativní ceny, takže je není nutno pokaždé znovu počítat, přičemž se současně spočte hodnota účelové funkce pro aktuální bázi, viz např. [1, 2, 12]. Konečně za jistých okolností může dojít podle výše uvedených pravidel pro výběr klíčového sloupce a následně řádku k zacyklení algoritmu. Později uvedeme, jak tato pravidla upravit tak, aby k tomu nedošlo, byť za i cenu jeho jistého zpomalení.

Dále je dobré si uvědomit, že primární simplexový algoritmus vskutku vybírá novou bazickou proměnnou $\tilde{x}_j$ podle toho, že je u ní **relativně** nejvyšší růst účelové funkce. Existuje totiž vždy (možná jiná) nebazická proměnná, v níž je růst účelové funkce nejvyšší **absolutně**. Abychom ji ale zjistili, museli bychom mj. provést dodatečné výpočty, které přímo simplexová tabulka neobsahuje. Kromě toho se v literatuře uvádí, že tato filozofie výběru nové proměnné skrývá určitá úskalí a v krajním případě se dokonce uskuteční největší možný počet kroků algoritmu. Z tohoto pohledu se jeví pravidlo (2.3) jako jistý kompromis, zřejmě však velmi rozumný (viz např. výstižnou ilustraci zmíněné situace v [2]).

Nyní uvedeme explicitní popis algoritmu pro řešení úlohy (1.20). Nejdříve doplníme úlohu m doplňkovými proměnnými $x_{n+1}, \ldots, x_{n+m}$, čímž ji převedeme na kanonický tvar

(1.22). Matice této úlohy tedy bude mít tvar $(\mathrm{A} \mid \mathrm{I}_m)$, kde I_m je jednotková matice příslušného řádu. Nechť je algoritmus reprezentován posloupností simplexových tabulek. Nultý krok přitom představuje výchozí tabulku. Tato tabulka je postupně transformována p kroky algoritmu, a tudíž p-tá tabulka reprezentuje již optimální řešení kanonické úlohy, z níž je možno zjistit i optimální řešení původní úlohy. Obecný tvar k-té tabulky ($k = 0, \ldots, p$), kde L_k je hodnota účelové funkce pro aktuální bázi, je následující:

Tab. 1. Obecný tvar primární simplexové tabulky

základní proměnné	*doplňkové proměnné*	
$\boldsymbol{c}^{\mathrm{T}}$	$\mathbf{0}^{\mathrm{T}}$	**max**
A_k	B_k	$\boldsymbol{b}_k$
$(\boldsymbol{z}_{\mathrm{A}} - \boldsymbol{c}_{\mathrm{A}})_k^{\mathrm{T}}$	$(\boldsymbol{z}_{\mathrm{B}} - \boldsymbol{c}_{\mathrm{B}})_k^{\mathrm{T}}$	L_k

V tab. 1 se první dva řádky během transformací nemění, takže se kromě počáteční tabulky neopisují. Speciálně je pro $k = 0$ v počáteční tabulce $\mathrm{A}_0 = \mathrm{A}$, $\mathrm{B}_0 = \mathrm{I}_m$ a $\boldsymbol{b}_0 = \boldsymbol{b}$. Poněvadž jsou doplňkové proměnné na začátku všechny bazické a jejich koeficienty v účelové funkci jsou formálně nulové, jsou relativní ceny u proměnných původní úlohy x_j, které nejsou bazické, rovny podle (2.2) jednoduše $\left(-c_j\right)$ a $L_0 = 0$. Počáteční tabulka bude mít tedy tvar:

Tab. 2. Tvar simplexové tabulky v 0*-té iteraci*

základní proměnné	*doplňkové proměnné*	
$\boldsymbol{c}^{\mathrm{T}}$	$\mathbf{0}^{\mathrm{T}}$	**max**
A	I_m	$\boldsymbol{b}$
$(-\boldsymbol{c})^{\mathrm{T}}$	$\mathbf{0}^{\mathrm{T}}$	0

V p-té tabulce, kterou algoritmus končí, musí být $(\boldsymbol{z}_{\mathrm{A}} - \boldsymbol{c}_{\mathrm{A}})_p^{\mathrm{T}} \geq \mathbf{0}$ a $(\boldsymbol{z}_{\mathrm{B}} - \boldsymbol{c}_{\mathrm{B}})_p^{\mathrm{T}} \geq \mathbf{0}$. Z teorie LP plyne, že v tomto případě je $\tilde{\mathbf{y}} = (\boldsymbol{z}_{\mathrm{B}} - \boldsymbol{c}_{\mathrm{B}})_p^{\mathrm{T}}$ optimální řešení duální úlohy z (1.24). Podle věty 4, (c) v odd. 1.5 je potom $\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}} = \boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}}$ pro optimální řešení $\tilde{\mathbf{x}}$ primární úlohy. Tuto rovnost lze použít jako kritérium kontroly správnosti výpočtů. Ovšem pokud tato rovnost bude platit, podle téže věty budeme mít jistotu, že jsme nalezli optimální řešení i v případě, že bychom se během výpočtů zmýlili.

Nyní vyřešíme pomocí simplexového algoritmu ve formě tabulek dva příklady.

Příklad 1. V tomto příkladu řešíme úlohu (1.12) ze str. 17, kde jsme již tuto úlohu vyřešili grafickou metodou. Připomeňme, že jde o následující úlohu:

$$\begin{aligned} &\max\ (2x_1 + 3x_2) \\ &x_1 + 3x_2 \leq 8 \\ &3x_1 + 2x_2 \leq 8 \\ &x_{1,2} \geq 0 \end{aligned} \tag{2.5}$$

Úlohu přepíšeme na kanonický tvar přidáním doplňkových proměnných x_3 a x_4. Tím přejde úloha na tento tvar:

$$\begin{aligned} &\max\ (2x_1 + 3x_2) \\ &x_1 + 3x_2 + x_3 = 8 \\ &3x_1 + 2x_2 + x_4 = 8 \\ &x_{1,2,3,4} \geq 0 \end{aligned} \tag{2.6}$$

Nedříve sestavíme a zobrazíme všechny simplexové tabulky a ihned za nimi poskytneme komentář k tomu, jak byla každá transformována z předchozí tabulky.

Tab. 3. Simplexové tabulky k př. 1

x_1	x_2	x_3	x_4	
2	3	0	0	**max**
1	**3**	1	0	8
3	2	0	1	8
−2	−3	0	0	0
$\frac{1}{3}$	1	$\frac{1}{3}$	0	$\frac{8}{3}$
$\mathbf{\frac{7}{3}}$	0	$-\frac{2}{3}$	1	$\frac{8}{3}$
−1	0	1	0	8

0	1	$\frac{3}{7}$	$-\frac{1}{7}$	$\frac{16}{7}$
1	0	$-\frac{2}{7}$	$\frac{3}{7}$	$\frac{8}{7}$
0	0	$\frac{5}{7}$	$\frac{3}{7}$	$\frac{64}{7}$

První počáteční tabulka je zřejmě ve tvaru tab. 2. Klíčový prvek v ní je $a_{12} = 3$. Nejnižší záporná relativní cena leží ve druhém sloupci, neboť dle (2.3): $2 = \arg\min\{(-2)_1, (-3)_2\}$. A leží v prvním řádku, protože podle (2.4): $1 = \arg\min\left\{\left(\frac{8}{3}\right)_1, \left(\frac{8}{2}\right)_2\right\}$. Vzhledem tomuto prvku je proveden krok Jordanovy řádkové eliminace. Ve druhé tabulce je klíčový prvek $\frac{7}{3}$. Leží v prvním sloupci, neboť je v něm již jediná záporná relativní cena. A leží ve druhém řádku, protože podle (2.4): $2 = \arg\min\left\{\left(\frac{8}{3}:\frac{1}{3}\right)_1, \left(\frac{8}{3}:\frac{7}{3}\right)_2\right\}$. Po provedení dalšího eliminačního kroku již má poslední tabulka jen nezáporné relativní ceny a simplexový algoritmus končí nalezením optimálního řešení. Optimální bazické řešení kanonické úlohy (2.6) je $\mathbf{x}^* = \left(\frac{8}{7}, \frac{16}{7}, 0, 0\right)^{\mathrm{T}}$, jemuž odpovídá optimální řešení původní úlohy $\tilde{\mathbf{x}} = \left(\frac{8}{7}, \frac{16}{7}\right)^{\mathrm{T}}$, které bylo nalezeno i grafickou metodou a kterému odpovídá nejvyšší hodnota účelové funkce $\frac{64}{7}$. Vektor $\tilde{\mathbf{y}} = \left(\frac{5}{7}, \frac{3}{7}\right)^{\mathrm{T}}$ je potom řešením duální úlohy, přičemž $\boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}} = 8 \cdot \frac{5}{7} + 8 \cdot \frac{3}{7} = \frac{64}{7} \approx 9{,}1429$.

Příklad 2. Uvažujeme následující maximalizační úlohu:

$$\begin{aligned} &\max\ (-12x_1 + 40x_2 + 15x_3) \\ &\qquad 2x_2 + 3x_3 \leq 24 \\ &x_1 + x_2 - x_3 \leq 18 \\ &-x_1 + 2x_2 + 3x_3 \leq 15 \\ &x_{1,2,3} \geq 0 \end{aligned} \tag{2.7}$$

Úlohu převedeme na kanonický tvar přidáním doplňkových proměnných x_4, x_5 a x_6. Tím přejde úloha na tento kanonický tvar:

$$\begin{aligned} &\max\ (-12x_1 + 40x_2 + 15x_3) \\ &\quad 2x_2 + 3x_3 + x_4 = 24 \\ &x_1 + x_2 - x_3 + x_5 = 18 \\ &-x_1 + 2x_2 + 3x_3 + x_6 = 15 \\ &x_{1,2,3,4,5,6} \geq 0 \end{aligned} \tag{2.8}$$

Nyní sestavíme a zobrazíme všechny simplexové tabulky:

Tab. 4. Simplexové tabulky k př. 2

x_1	x_2	x_3	x_4	x_5	x_6	
−12	40	15	0	0	0	**max**
0	2	3	1	0	0	24
1	1	−1	0	1	0	18
−1	**2**	3	0	0	1	15
12	−40	−15	0	0	0	0
1	0	0	1	0	−1	9
$\frac{3}{2}$	0	$-\frac{5}{2}$	0	1	$-\frac{1}{2}$	$\frac{21}{2}$
$-\frac{1}{2}$	1	$\frac{3}{2}$	0	0	$\frac{1}{2}$	$\frac{15}{2}$
−8	0	45	0	0	20	300
0	0	$\frac{5}{3}$	1	$-\frac{2}{3}$	$-\frac{2}{3}$	2
1	0	$-\frac{5}{3}$	0	$\frac{2}{3}$	$-\frac{1}{3}$	7
0	1	$\frac{1}{6}$	0	$\frac{1}{3}$	$\frac{1}{3}$	11
0	0	$\frac{95}{3}$	0	$\frac{16}{3}$	$\frac{52}{3}$	356

V počáteční tabulce je klíčový prvek $a_{32} = 2$, protože $2 = \arg\min\{(-40)_2, (-15)_3\}$, a tudíž leží ve druhém sloupci. Poněvadž $3 = \arg\min\left\{\left(\frac{24}{2}\right)_1, \left(\frac{15}{2}\right)_3\right\}$, leží ve třetím řádku. Podobně v následující tabulce je klíčovým prvkem $\frac{3}{2}$: poněvadž $1 = \arg\min\{(-8)_1\}$, leží

v prvním sloupci, a jelikož $2 = \arg\min\left\{\left(\frac{9}{1}\right)_1, \left(\frac{21}{2}:\frac{3}{2}\right)_2\right\}$, leží ve druhém řádku. V poslední tabulce jsou již všechny relativní ceny nezáporné. Optimální řešení kanonické úlohy je $\mathbf{x}^* = (7,11,0,2,0,0)^{\mathrm{T}}$, přičemž maximální hodnota účelové funkce je 356. Z tohoto řešení lze zjistit nejen to, že optimální řešení původní úlohy je $\tilde{\mathbf{x}} = (7,11,0)^{\mathrm{T}}$, ale i to, že první omezení je o hodnotu 2 nedočerpáno, zatímco ostatní dvě jsou dočerpána. Řešení je v tomto případě dokonce (neúmyslně) celočíselné. Nakonec vektor $\tilde{\mathbf{y}} = \left(0, \frac{16}{3}, \frac{52}{3}\right)^{\mathrm{T}}$ je řešením duální úlohy k úloze (2.7), přičemž také $\boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}} = 24 \cdot 0 + 18 \cdot \frac{16}{3} + 15 \cdot \frac{52}{3} = 356$.

Víme již, že pro kanonickou úlohu (1.22) je počet přípustných bazických řešení roven nejvýše $\binom{n+m}{m} = \binom{n+m}{n}$. U př. 1 je $\binom{2+2}{2} = \binom{4}{2} = 6$ a u př. 2 je $\binom{3+3}{3} = \binom{6}{3} = 20$. Zejména u př. 2 tedy běžel simplexový algoritmus dosti efektivně.

Příklad 3. Tento příklad jsme převzali z [12], kde je uveden jako neřešené cvičení.

$$\begin{aligned} &\max\ (3x_1 + 2x_2 + 4x_3) \\ &\begin{aligned} x_1 \ &+ x_2 + 2x_3 \le 4 \\ 2x_1 \ &+ \ x_3 \le 5 \\ 2x_1 \ &+ x_2 + 3x_3 \le 7 \end{aligned} \\ &x_{1,2,3} \ge 0 \end{aligned} \tag{2.9}$$

Úlohu převedeme na kanonický tvar přidáním doplňkových proměnných x_4, x_5 a x_6. Tím získáme následující kanonickou úlohu ve tvaru:

$$\begin{aligned} &\max\ (3x_1 + 2x_2 + 4x_3) \\ &\begin{aligned} x_1 \ &+ x_2 + 2x_3 + x_4 \qquad\quad = 4 \\ 2x_1 \ &+ \ x_3 \quad + x_5 \quad = 5 \\ 2x_1 \ &+ x_2 + 3x_3 \qquad + x_6 = 7 \end{aligned} \\ &x_{1,2,3} \ge 0 \end{aligned} \tag{2.10}$$

Algoritmus řešení úlohy (2.10) znázorňují následující simplexové tabulky:

Tab. 5. Simplexové tabulky k př. 3

x_1	x_2	x_3	x_4	x_5	x_6	
3	2	4	0	0	0	**max**
1	1	**2**	1	0	0	4
2	0	1	0	1	0	5
2	1	3	0	0	1	7
−3	−2	−4	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	0	0	2
$\frac{3}{2}$	$-\frac{1}{2}$	0	$-\frac{1}{2}$	1	0	3
$\mathbf{\frac{1}{2}}$	$-\frac{1}{2}$	0	$-\frac{3}{2}$	0	1	1
−1	0	0	2	0	0	8
0	1	1	2	0	−1	1
0	**1**	0	4	1	−3	0
1	−1	0	−3	0	2	2
0	−1	0	−1	0	2	10
0	0	1	−2	−1	**2**	1
0	1	0	4	1	−3	0
1	0	0	1	1	−1	2
0	0	0	3	1	−1	10
0	0	$\frac{1}{2}$	−1	$-\frac{1}{2}$	1	$\frac{1}{2}$
0	1	$\frac{3}{2}$	1	$-\frac{1}{2}$	0	$\frac{3}{2}$
1	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{5}{2}$
0	0	$\frac{1}{2}$	2	$\frac{1}{2}$	0	$\frac{21}{2}$

Optimální řešení kanonické úlohy je $\mathbf{x}^* = \left(\frac{5}{2}, \frac{3}{2}, 0, 0, 0, \frac{1}{2}\right)^{\mathrm{T}}$. Optimální řešení původní úlohy je $\tilde{\mathbf{x}} = \left(\frac{5}{2}, \frac{3}{2}, 0\right)^{\mathrm{T}}$ a optimálním řešením duální úlohy je vektor $\tilde{\mathbf{y}} = \left(2, \frac{1}{2}, 0\right)^{\mathrm{T}}$, přičemž

společná hodnota účelových funkcí obou úloh (1.24) je $\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}} = \frac{21}{2} = 4 \cdot 2 + 5 \cdot \frac{1}{2} = \boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}}$. Navíc z hodnot složek $\mathbf{x}^*$ plyne, že první a druhé omezení je plně dočerpáno, poslední omezení je nedočerpáno o hodnotu $\frac{1}{2}$.

Poslední příklad má oproti dvěma předchozím příkladům jednu zvláštnost. Z třetí a čtvrté iterace (podtabulky) v tab. 5 je zřejmé, že se jedná o degenerovanou úlohu (poznamenejme, že je obecně velmi obtížné předem poznat, zda je či není úloha degenerovaná, neboť tato skutečnost souvisí jen okrajově s hodností matice původní úlohy, v případě úlohy (2.9) je totiž $h(\mathrm{A}) = 3$). Ve druhé podtabulce není podle (2.4) jednoznačně určen klíčový řádek a ve třetí podtabulce není podle (2.3) jednoznačně určen klíčový sloupec. To jsou zcela klasické příznaky toho, že může dojít k zacyklení simplexového algoritmu, kdy nedochází iteracemi ke zvyšování hodnoty účelové funkce, jak ukazují třetí a čtvrtá tabulka. V těchto tabulkách je ve sloupci hodnot omezení potom alespoň jedna nula. Podle (2.4) je potom nutně vybrán řádek, který takovou nulu obsahuje, čímž bude přírůstek účelové funkce po iteraci nulový. Základním problémem je skutečnost, že báze, jíž u degenerovaného řešení odpovídají nenulové složky řešení, není určena jednoznačně. Algoritmus potom může iteracemi přeskakovat z jedné takové báze ke druhá stále dokola, aniž by bylo dosaženo maxima.

Existuje několik metod, jak vyvést simplexový algoritmus ze smyčky. Nejobvyklejší je tzv. ***metoda nejmenších indexů***, o níž je dokázáno, že vždy vyvede algoritmus z cyklu. Pokud se v iteraci vyskytne degenerované přípustné řešení, stačí tvrdošíjně vždy vybírat nejmenší indexy, které určují pravidla (2.3) a (2.4), dokud se v iteraci opět neobjeví nedegenerované řešení. Pak algoritmus pokračuje standardně až k nalezení optima, anebo pokud se zase objeví degenerované řešení, je nutno opět pokračovat metodou nejmenších indexů atd. až do konce. Podle některých autorů nebyla u rozsáhlých úloh degenerace pozorována, takže většina specializovaného software ani žádnou ochranu proti zacyklení nemá [2]. Jiní naopak tvrdí, že u takových úloh k degeneraci v jisté fázi dochází téměř vždy [1]. Ovšem z logického hlediska se jedná pouze o paradox – dva autoři se na tutéž skutečnost dívají z různých úhlů pohledu: jeden z numerického hlediska, druhý z hlediska teoretického.

Pokud bychom však trvali na ošetření možnosti zacyklení přinejmenším proto, že k němu může dojít u didaktických příkladů menších rozměrů, pak zaměníme pravidlo (2.3) následujícím pravidlem, které je zapsáno poněkud těžkopádně, ale jeho význam je z předchozího zřejmý – vybrat nejmenší index:

$$\textit{klíčový je } j\textit{-tý sloupec, pro který} \quad j = \min\left(\arg\min_{l} \{z_l - c_l \mid z_l - c_l < 0\}\right) \quad (2.11)$$

Obdobně přejde pravidlo (2.4) v následující pravidlo:

$$\textit{klíčový je } i\textit{-tý řádek, pro který} \quad i = \min\left(\arg\min_{k} \left\{\frac{b_k}{a_{kj}} \,\middle|\, a_{kj} > 0\right\}\right) \quad (2.12)$$

Poněvadž nejednoznačnost výběru klíčového sloupce, příp. řádku vždy předchází degenerovanosti přípustného řešení a pravidla (2.11) a (2.12) se běžně redukují na (2.3) a (2.4), můžeme nyní zformulovat primární simplexový algoritmus obvyklým způsobem do imperativní sekvence. Předpokládejme, že v úloze (1.20) je $\boldsymbol{b} \geq \boldsymbol{0}$. Má-li úloha optimální řešení, pak je v konečném počtu kroků najde **primární simplexový algoritmus**:

0. KROK: sestav počáteční simplexovou tabulku (tab. 2), jdi na 1. KROK;

1. KROK: vyber klíčový prvek a_{ij} podle pravidel (2.11) a (2.12), jdi na 2. KROK;

2. KROK: v aktuální tabulce proveď podle vybraného klíčového prvku Jordanovu eliminaci v celém sloupci, v němž tento prvek leží, jdi na 3. KROK;

3. KROK: jsou-li v posledním řádku všechny relativní ceny nezáporné, jdi na 4. KROK, jinak jdi na 1. KROK;

4. KROK: algoritmus je ukončen, poslední tabulka reprezentuje optimální řešení úlohy (primární i duální)

Ve dvou případech, je algoritmus ukončen jinak nežli 4. krokem. Zaprvé pokud ve sloupcích nad všemi zápornými relativními cenami leží pouze nekladné hodnoty. Pak podle (2.4), resp. (2.12) nelze vybrat žádný klíčový řádem, ovšem nebazické proměnné v těchto sloupcích mají schopnost stále zvyšovat hodnotu účelové funkce svým vlastním zvětšením, avšak současně hodnoty žádných jiných proměnných nemusí klesnout, aby byly splněny všechny rovnosti omezení kanonické úlohy. Jinak řečeno účelová funkce je na množině přípustných řešení shora neomezená. Ve druhém případě poběží algoritmus do nekonečna a i v tom případě to znamená, že je účelová funkce na množině přípustných řešení rovněž neomezená.

Poznámka. Pouze pro úplnost doplňme, proč nám u př. 3 postačilo řešit jej jen na základě pravidel (2.3) a (2.4) a vybrat klíčový sloupec jen na základě „přívětivějších" numerických hodnot v něm. V našem případě bychom dospěli k výsledku i při výběru náhodném. Tato skutečnost plyne z jistých teoretických výsledků kombinatorické povahy, na které lze nalézt odkazy v [12], str. 123–5. Geometricky řečeno rozměry úloh podobných té naší nejsou dostatečné k tomu, aby se simplexový algoritmus mohl zacyklit jinde než v bezprostředním okolí maxima Ještě jinak řečeno, množina přípustných řešení nemá dostatek krajních bodů k tomu, aby se to mohlo stát jinde.

Samotný Dantzig (spoluautor M. N. Thapa) píše patrně ve své poslední knize *Linear Programming 2: Theory and Extensions* z roku 2003 doslova, že z matematického hlediska je degenerace fascinujícím fenoménem. Otázka degenerace a jejího překonání je přinejmenším v teoretické rovině stále živá. Skutečnost, že se zacyklení algoritmu daří odstranit poměrně jednoduchými postupy, souvisí s tím, že tyto postupy obcházejí degeneraci v podstatě „velmi hrubou silou" bez ohledu na komplexní strukturu matice úlohy A, resp. rozšířené matice $\widetilde{A}$.

2.2 Několik poznámek k analýze citlivosti

Každá oblast optimalizace má nejen vlastní metody pro hledání optima, ale také metody, které jsou schopny zjišťovat, jak je toto optimum ovlivňováno změnami zadání a parametrů řešených optimalizačních úloh. Tyto postupy se souhrnně označují jako ***analýza citlivosti***, příp. ***postoptimalizační analýza***. Je zřejmé, že i LP má svoji analýzu citlivosti. Přitom otázky citlivosti řešeného problému mají nezřídka stejný význam jako samotné nalezení optima.

Analýza citlivosti úloh LP zahrnuje dva typy problémů. Především řeší obecně otázku, jak se změní optimální řešení při změně zadání úlohy, tedy při změně koeficientů matice omezení, jejich pravých stran či koeficientů účelové funkce. Zadruhé řeší otázku, v jakém rozsahu lze měnit některé koeficienty úlohy tak, aby se optimální báze nezměnila a měnila se pouze hodnota účelová funkce a jak tuto změnu interpretovat.

Zde se krátce zmíníme o tom, co lze vyčíst ze simplexové tabulky, jež reprezentuje optimální řešení. Již víme, že tato tabulka obsahuje kompletní řešení kanonické úlohy, přičemž jeho složky, odpovídající doplňkovým proměnným, představují, o kolik jsou jednotlivá omezení nedočerpána; v případě nulové hodnoty je kapacita omezení plně vyčerpána. Kromě toho tabulka v posledním řádku obsahuje pod doplňkovými proměnnými také řešení duální úlohy. Právě řešení duální úlohy má svůj vlastní význam, resp. odpovídající ekonomickou interpretaci. Proto se hodnoty jeho složek nejčastěji nazývají ***duální***, příp. ***stínové ceny***.

Duální ceny lze interpretovat v zásadě dvěma způsoby. Pro další úvahu musíme předpokládat, že změny pravých stran $\boldsymbol{b}$ omezení probíhají v takových přípustných mezích, aby se nezměnila optimální báze. Pokud by se báze změnila, neplatilo by ekvilibrium, které vyjadřuje rovnost v silné větě o dualitě. Z této věty tedy vyplývá, že pro optimální hodnotu účelové funkce je

$$\tilde{z} = \boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}} = \boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}} = \sum_{i=1}^{m} b_i\,\tilde{y}_i \tag{2.13}$$

Pro interpretaci předpokládejme, že původní úloha je úlohou optimálního výrobního plánu. Z rovnosti (2.13) ihned vidíme, že zvýšíme-li kapacitu b_i o jednotku, zvýší se hodnota účelové funkce o $\tilde{y}_i$. Tudíž, abychom zvýšili zisk navýšením kapacity *i*-tého zdroje, mezní cenou, za kterou má smysl přikoupit jednotku tohoto zdroje, je $\tilde{y}_i$. V opačném případě bychom navýšením sice zisk zvýšili, avšak ne o tolik, abychom v konečném důsledku neprodělali.

Jiná možnost, jak pohlížet na duální ceny, opět plyne z rovnice (2.13). Z ní totiž také plyne, že

$$\tilde{y}_i = \frac{\partial \tilde{z}}{\partial b_i}, \quad i = 1, \ldots, m \tag{2.14}$$

Z těchto rovností je vidět, že $\tilde{y}_i$ je rychlost – jinak řečeno citlivost – změny optimální hodnoty účelové funkce v závislosti na změně kapacity b_i.

Konkrétně v př. 1 z předchozího odd. je $\tilde{\mathbf{y}} = \left(\frac{5}{7}, \frac{3}{7}\right)^{\mathrm{T}}$. Tento výsledek tvrdí, že nákupem obou zdrojů lze zvýšit zisk, ovšem jednotku prvního zdroje má smysl nakupovat nejvýše za $\frac{5}{7}$ (měnové jednotky) a druhý maximálně za $\frac{3}{7}$. V opačném případě sice zisk vzroste, nicméně ve finále proděláme. Přitom nejvíce poroste zisk nákupem prvního zdroje. V př. 2 je $\tilde{\mathbf{y}} = \left(0, \frac{16}{3}, \frac{52}{3}\right)^{\mathrm{T}}$, což znamená, že pro zvýšení zisku má smysl přikupovat pouze druhý a třetí zdroj, zatímco první zdroj přikupovat nemá smysl, ledaže by nám jej někdo dával zdarma. Stejně bychom interpretovali i výsledky př. 3, kde $\tilde{\mathbf{y}} = \left(2, \frac{1}{2}, 0\right)^{\mathrm{T}}$. Nejrychleji budeme zvyšovat zisk dokupováním prvního zdroje, třetí zdroj naopak nemá smysl dokupovat vůbec.

2.3 Duální simplexový algoritmus

Standardní metody celočíselného programování, které jsou probrány v další části, jsou založeny na využití neceločíselného řešení relaxované úlohy jakožto řešení výchozího. Metody potom fungují tak, že se k původním omezením úlohy postupně přidávají další omezující

podmínky takovým způsobem, aby se – tak říkajíc – přenášela zodpovědnost za neceločíselnost kompletního řešení kanonické úlohy postupně na doplňkové proměnné. Tím jsou proměnné v původní úloze postupně „sunuty“ k jejímu celočíselnému optimálnímu řešení.

Výše zmíněnou úlohu řeší ***duální simplexový algoritmus***, který vychází z předpokladu, že ve výchozí tabulce jsou již všechny relativní ceny nezáporné. Duální simplexovou metodu vyvinuli v roce 1954 nezávisle na sobě C. E. Lemke a E. M. L. Beale. Vychází pochopitelně ze vzájemného vztahu duálních úloh (1.24). Poněvadž duální úlohu lze přepsat na tvar maximalizační úlohy

$$\begin{gathered} \max\left(-\boldsymbol{b}^{\mathrm{T}}\mathbf{y}\right) \\ -\mathrm{A}^{\mathrm{T}}\mathbf{y} \leq -\boldsymbol{c} \\ \mathbf{y} \geq \mathbf{0} \end{gathered}$$

můžeme výchozí tab. 2 primární simplexové metody zapsat následovně (význam změny maximalizace na **minimalizaci** bude zřejmý ihned z následujícího):

Tab. 6. Primární simplexová tabulka pro duální úlohu

základní proměnné	*doplňkové proměnné*	
$-\boldsymbol{b}^{\mathrm{T}}$	$\mathbf{0}^{\mathrm{T}}$	**min**
$-\mathrm{A}^{\mathrm{T}}$	I_n	$-\boldsymbol{c}$
$\boldsymbol{b}^{\mathrm{T}}$	$\mathbf{0}^{\mathrm{T}}$	0

V této tabulce je poslední řádek – obecně nazývaný také ***kriteriální řádek*** – nezáporný, čili primární simplexový algoritmus nemůže optimalizaci vůbec zahájit. Navíc pravý sloupec není nezáporný, neboli tab. 6 neobsahuje přípustné bazické řešení. Je-li ovšem původní úloha řešitelná, vyjadřuje tato tabulka úlohu s přípustným duálním řešením – totiž s nulovým řešením úlohy primární.

Z tab. 6 je zřejmá základní myšlenka duálního algoritmu řešit duální úlohu pomocí primární simplexové tabulky. Jak dále ukážeme, znamená to, že je třeba vybírat klíčový řádek v zásadě stejným způsobem, jako primární algoritmus vybírá klíčový sloupec, a klíčový sloupec vybírat analogicky k výběru klíčového řádku. Z geometrického pohledu prochází duální algoritmus přípustná řešení $\mathbf{y}$ duální úlohy, jimž odpovídají sice bazická, avšak ne-

přípustná řešení $\mathbf{x}$ primární úlohy tak dlouho, dokud nenalezne první přípustné bazické řešení primární úlohy [5]. V tu chvíli algoritmus nalezl optimální řešení obou úloh. Podle slabé věty o dualitě musí být díky nepřípustnosti primárního řešení $\boldsymbol{c}^{\mathrm{T}}\mathbf{x} > \boldsymbol{b}^{\mathrm{T}}\mathbf{y}$. Algoritmus potom funguje tak, že postupně zvyšuje hodnotu účelové funkce duální úlohy a současně přitom snižuje hodnotu účelové funkce úlohy primární, dokud není dosaženo rovnosti $\boldsymbol{c}^{\mathrm{T}}\mathbf{x} = \boldsymbol{b}^{\mathrm{T}}\mathbf{y}$. Podle silné věty o dualitě jsou tímto způsobem nutně nalezena optimální řešení obou úloh. Vzhledem k popsanému mechanismu funguje duální algoritmus de facto tak, že minimalizuje funkci $-\boldsymbol{b}^{\mathrm{T}}\mathbf{y}$, a proto je v tab. 6 poznačena minimalizace.

Pro vlastní formulaci algoritmu musíme ovšem vyjít z tvaru obecné tabulky, abychom zcela zbytečně nemuseli předělávat veškeré značení, neboli z tab. 1. Podstatnějším důvodem je však skutečnost, že v kontextu dále popsaných metod celočíselného LP budeme vždy vycházet právě z tvaru tab. 1, která bude reprezentovat optimální řešení relaxované neceločíselné úlohy. Pravidla pro výběr klíčového řádku a klíčového sloupce jsou následující:

$$\textit{klíčový je i-tý řádek, pro který} \quad i = \arg\min_{k} \{b_k \mid b_k < 0\}; \tag{2.15}$$

$$\textit{klíčový je j-tý sloupec, pro který} \quad j = \arg\min_{l} \left\{ \left| \frac{z_l - c_l}{a_{il}} \right| \,\middle|\, a_{il} < 0 \right\} \tag{2.16}$$

Je zřejmé, že pravidla (2.15) a (2.16) vybírají klíčové řádky a sloupce vskutku „duálně“ k tomu, jak vybírá primární simplexový algoritmus klíčové sloupce a řádky podle (2.3) a (2.4), příp. podle (2.11) a (2.12).

Duální simplexový algoritmus můžeme nyní zformulovat do imperativního tvaru:

0. KROK: sestav počáteční simplexovou tab. 6, jdi na 1. KROK;

1. KROK: vyber klíčový prvek a_{ij} podle pravidel (2.15) a (2.16), jdi na 2. KROK;

2. KROK: v aktuální tabulce proveď podle vybraného klíčového prvku Jordanovu eliminaci v celém sloupci, v němž tento prvek leží, jdi na 3. KROK;

3. KROK: jsou-li v posledním sloupci všechny složky nezáporné, jdi na 4. KROK, jinak jdi na 1. KROK;

4. KROK: algoritmus je ukončen, poslední tabulka reprezentuje optimální řešení úlohy (duální i primární)

Popsaný algoritmus je konečný, pokud má některá ze vzájemně duálních úloh řešení. Co se týká nejednoznačnosti při použití pravidel (2.15) a (2.16), také zde lze použít metodu nejmenších indexů.

Jediným případem, kdy duální algoritmus skončí předčasně, nastane, jakmile jsou v pravém sloupci simplexové tabulky stále záporné hodnoty, ale současně v jim příslušných řádcích jsou již všechny koeficienty nezáporné. Pak nemá duální úloha optimální řešení, a tudíž je nemá podle silné věty o dualitě ani úloha primární. Lze říci dokonce více. V tomto případě musí být množina přípustných řešení primární úlohy prázdná. Podle výše uvedené geometrické interpretace není schopen duální algoritmus nalézt přípustné bazické řešení primární úlohy, což podle základní věty LP znamená, že nemůže nalézt žádný krajní bod množiny přípustných řešení. A to je možné pouze tehdy, pokud je tato množina prázdná.

Duální algoritmus je nedílnou součástí základních metod celočíselného LP, kterým se věnujeme v další kapitole. V těchto metodách vznikají vždy simplexové tabulky s duálně přípustným řešením doplněním primární optimální simplexové tabulky o další omezení. Tabulka je doplněna o řádky a sloupce těchto omezení, přičemž nový kriteriální řádek obsahuje původní řádek jako svou část. Tím, že se doplněním omezení změní primární úloha, se samozřejmě změní také úloha duální. Nicméně kriteriální řádek i potom bude obsahovat přípustné duální řešení, čímž může být duální algoritmus inicializován.

Příklad 4. Konečná optimální simplexová tabulka úlohy (2.10) v př. 3 je následující:

Tab. 7. Optimální simplexová tabulka z př. 3

x_1	x_2	x_3	x_4	x_5	x_5	**max**
0	0	$\frac{1}{2}$	−1	$-\frac{1}{2}$	1	$\frac{1}{2}$
0	1	$\frac{3}{2}$	1	$-\frac{1}{2}$	0	$\frac{3}{2}$
1	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{5}{2}$
0	0	$\frac{1}{2}$	2	$\frac{1}{2}$	0	$\frac{21}{2}$

Nyní si představme, že potřebujeme řešit úlohu (2.10) s tím, že jsme nuceni navíc mezi její omezení přidat rovnici (2.17), tzn. aby úloha reprezentovaná tab. 7 splňovala následující dodatečné omezení s novou proměnnou x_7:

$$\frac{1}{2}x_3 + \frac{1}{2}x_5 - x_7 = \frac{1}{2} \tag{2.17}$$

Toto omezení nejdříve vynásobíme (-1) proto, aby mohla proměnná x_7 vstoupit do kanonické báze v následující tab. 8, čímž přejde na tvar

$$-\frac{1}{2}x_3 - \frac{1}{2}x_5 + x_7 = -\frac{1}{2} \tag{2.18}$$

a přidáme jej do tabulky. Tím vznikne tabulka, která má přípustné duální řešení a má tvar vhodný k provedení iterací duálního algoritmu, neboť všechny relativní ceny jsou nezáporné. Takto postupně získáme následující tabulky:

Tab. 8. Duální simplexový algoritmus v př. 4

x_1	x_2	x_3	x_4	x_5	x_6	x_7	**min**
0	0	$\frac{1}{2}$	−1	$-\frac{1}{2}$	1	0	$\frac{1}{2}$
0	1	$\frac{3}{2}$	1	$-\frac{1}{2}$	0	0	$\frac{3}{2}$
1	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	$\frac{5}{2}$
0	0	$-\frac{1}{2}$	0	$-\frac{1}{2}$	0	1	$-\frac{1}{2}$
0	0	$\frac{1}{2}$	2	$\frac{1}{2}$	0	0	$\frac{21}{2}$
0	0	0	−1	−1	1	1	0
0	1	0	1	−2	0	3	0
1	0	0	0	0	0	1	2
0	0	1	0	1	0	−2	1
0	0	0	2	0	0	1	10

První část tab. 8 obsahuje nepřípustné primární bazické řešení $\mathbf{x} = \left(\frac{5}{2}, \frac{3}{2}, 0,0,0,0, -\frac{1}{2}\right)^{\mathrm{T}}$, duální řešení $\mathbf{y} = \left(2, \frac{1}{2}, 0,0\right)^{\mathrm{T}}$ v ní je ale přípustné. Klíčovým je zde čtvrtý řádek, poněvadž $-\frac{1}{2} < 0$, a klíčový sloupec má pořadí $3 = \min\left(\arg\min\left\{\left(\left|\frac{1}{2}:\left(-\frac{1}{2}\right)\right|\right)_3, \left(\left|\frac{1}{2}:\left(-\frac{1}{2}\right)\right|\right)_5\right\}\right)$. Po provedení Jordanovy eliminace má příslušná kanonická úloha optimální celočíselné řešení $\mathbf{x}^* = (2,0,1,0,0,0,0)^{\mathrm{T}}$. Příslušné celočíselné řešení původní úlohy je pak $\tilde{\mathbf{x}} = (2,0,1)^{\mathrm{T}}$. V tomto případě je celočíselná i optimální hodnota účelové funkce 10.

Poznamenejme, že celočíselnost řešení v př. 4 není tak docela náhoda, poněvadž rovnice (2.18) je ve tvaru tzv. Gomoryho řezů, jimž se budeme věnovat v další kapitole, protože tvoří základ jedné ze standardních metod celočíselného programování. V tomto případě je totiž $\tilde{\mathbf{x}} = (2,0,1)^{\mathrm{T}}$ navíc optimálním celočíselným řešením úlohy (2.9) za dodatečného požadavku, aby $x_1, x_2, x_3 \in \mathbb{Z}$.

3 METODY CELOČÍSELNÉHO PROGRAMOVÁNÍ

3.1 Metoda Gomoryho řezů

Pro úlohy celočíselného LP existuje celá skupina metod jejich řešení – ***metody sečných nadrovin*** –, které jsou všechny založeny na společné filozofii. Vycházejí z optimálního obecně neceločíselného řešení relaxované úlohy. K této úloze se následně přidávají postupně další podmínky tak, aby nebylo vyloučeno žádné přípustné celočíselné řešení, ale současně byla vyloučena alespoň některá neceločíselná řešení tím, že se stanou nepřípustnými řešeními nové úlohy. Každá podmínka má tvar lineární rovnice, reprezentující nadrovinu, která „odřízne“ část množiny přípustných řešení původní úlohy. Tento postup se opakuje, dokud není nalezeno optimální celočíselné řešení.

Níže popsané metody pracují tak, že je úloha (1.20) převedena na kanonický tvar (1.22) a poté je nalezeno takové optimální řešení kanonické úlohy, které má celočíselné všechny základní proměnné. Ačkoliv je celočíselnost doplňkových proměnných obecně irelevantní, u níže popsané metody je její součástí.

První ze zmíněných metod, dnes nejčastěji nazývaná ***metoda Gomoryho řezů***, byla publikována v roce1958. Publikoval ji americký matematik Ralph Edward Gomory, který ji ještě téhož roku podstatně zobecnil v dalším článku. Zde se budeme věnovat pouze základní metodě. Tato metoda byla původně vyvinuta pro tzv. *celočíselně celočíselné úlohy* ve tvaru (1.22), ve které jsou celočíselné jak všechny prvky matice A, tak také vektoru $\boldsymbol{b}$. Předpoklad, že je úloha (1.22) celočíselně celočíselná, je z jedné strany pragmatické povahy, neboť potom budou celočíselné rovněž všechny doplňkové proměnné. Tím lze následně např. interpretovat kladné hodnoty doplňkových proměnných jako přebytky zdrojů v jednotkách.

Ovšem z druhé strany je předpoklad „celočíselné celočíselnosti“ nutnou podmínkou pro správnou funkci Gomoryho metody. Vynásobením každého omezení úlohy (1.20) vhodným číslem lze jistě docílit celočíselnosti A i $\boldsymbol{b}$. Nicméně nejedná se nadbytečnou formalitu, což souvisí se samotnou numerickou podstatou Gomoryho algoritmu. V [5] je ukázáno na protipříkladu, že opomenutí celočíselnosti A nebo $\boldsymbol{b}$ vede k tomu, že algoritmus poskytne zcela nesmyslné řešení. Na konci tohoto oddílu se pokoušíme o jisté vysvětlení.

Dále budeme potřebovat dva pomocné pojmy. Je-li $q \in \mathbb{R}$ libovolné reálné číslo, pak vždy existuje jeho jednoznačný rozklad

$$q = \lfloor q \rfloor + \{q\}, \tag{3.1}$$

kde $\lfloor q \rfloor$ je tzv. ***celá část*** q, tj. největší celé číslo takové, že $\lfloor q \rfloor \leq q$, a $\{q\}$ je ***zlomková část*** q, přičemž zřejmě $0 \leq \{q\} < 1$.

V souvislosti s výše uvedeným rozkladem budeme dále nuceni při přepisu nerovností na tvar rovností zavádět další proměnné, které se nazývají ***umělé proměnné***. Tyto proměnné mají zásadně pomocný charakter bez přímé, kupř. ekonomické interpretace. Tím se liší od proměnných doplňkových, které, ačkoli mají rovněž v jistém smyslu pomocný význam, mají také přímou interpretaci, např. zda došlo či nedošlo k dočerpání určitého zdroje.

Budeme vycházet z optimálního řešení relaxované úlohy. Uvažujme tedy úlohu (1.20), kde $\mathrm{A} \in \mathbb{Z}^{m \times n}$, $\boldsymbol{b} \in \mathbb{Z}^m$ (po případném vynásobení omezení vhodnými konstantami) a $\boldsymbol{c} \in \mathbb{R}^n$. Nechť následující simplexová tabulka reprezentuje optimální řešení k ní příslušné kanonické úlohy (1.22):

Tab. 9. Optimální simplexová tabulka relaxované kanonické úlohy

základní + doplňkové proměnné (0-tá iterace Gomoryho algoritmu)	**max**
M	$\boldsymbol{b}^*$
$(\boldsymbol{z}_\mathrm{M} - \boldsymbol{c}_\mathrm{M})^\mathrm{T} \geq \mathbf{0}^\mathrm{T}$	L_0

V této tabulce $\mathrm{M} \in \mathbb{R}^{m \times (n+m)}$ a $\mathbf{0} \leq \boldsymbol{b}^* \in \mathbb{R}^m$, přičemž prvky tohoto vektoru tvoří odpovídající složky optimálního bazického řešení, které jsou ovšem obecně neceločíselné.

Nyní naznačíme základní smysl a konstrukci Gomoryho řezů. Pokud nereprezentuje tab. 9 již celočíselné optimální řešení úlohy (1.20), resp. (1.22), pak má vektor $\boldsymbol{b}^*$ alespoň jednu neceločíselnou složku, řekněme $b^*_{i_0}$, odpovídající hodnotě jisté základní proměnné. Při použití rozkladu (3.1) můžeme napsat omezení, které představuje i_0-tý řádek tabulky, následně:

$$\sum_{j=1}^{n+m} \left(\lfloor m_{i_0 j} \rfloor + \{ m_{i_0 j} \} \right) x_j = \lfloor b^*_{i_0} \rfloor + \{ b^*_{i_0} \},$$

čili

$$\sum_{j=1}^{n+m}\{m_{i_0 j}\}\, x_j - \{b_{i_0}^*\} = \lfloor b_{i_0}^* \rfloor - \sum_{j=1}^{n+m}\lfloor m_{i_0 j} \rfloor\, x_j \tag{3.2}$$

Pokud předpokládáme, že je $\mathbf{x} \in \mathbb{Z}^{n+m}$ (a tento předpoklad je ospravedlnitelný právě díky té skutečnosti, že $A \in \mathbb{Z}^{m\times n}$ a $\boldsymbol{b} \in \mathbb{Z}^m$), napravo (3.2) je celé číslo, a tudíž je rovněž hodnota následujícího výrazu celočíselná:

$$\sum_{j=1}^{n+m}\{m_{i_0 j}\}\, x_j - \{b_{i_0}^*\} \in \mathbb{Z} \tag{3.3}$$

Kromě toho $\{b_{i_0}^*\} < 1$, neboli $-\{b_{i_0}^*\} > -1$. Tím spíše pak platí nerovnost

$$\sum_{j=1}^{n+m}\{m_{i_0 j}\}\, x_j - \{b_{i_0}^*\} > -1 \tag{3.4}$$

Z (3.3) a (3.4) ihned dostaneme, že

$$\sum_{j=1}^{n+m}\{m_{i_0 j}\}\, x_j - \{b_{i_0}^*\} \geq 0 \tag{3.5}$$

Následně pak obdržíme

$$\sum_{j=1}^{n+m}\left(-\{m_{i_0 j}\}\right) x_j \leq -\{b_{i_0}^*\} \tag{3.6}$$

Doplníme-li nakonec nerovnost (3.6) nezápornou umělou proměnnou $v_k \geq 0$, přejde tato nerovnice v rovnici

$$\sum_{j=1}^{n+m}\left(-\{m_{i_0 j}\}\right) x_j + v_k = -\{b_{i_0}^*\} \tag{3.7}$$

Rovnice (3.7), reprezentující nadrovinu, se nazývá ***Gomoryho řez*** (někdy se tak nazývá rovněž nerovnice (3.5) i přesto, že reprezentuje uzavřený poloprostor).

Základní myšlenka ***Gomoryho algoritmu*** spočívá v tom, že každé celočíselné řešení původní úlohy splňuje nerovnost (3.5), tudíž nadrovina (3.7) „uřízne jistou porci“ původní množiny přípustných řešení, avšak nikdy „neodřízne“ žádné celočíselné přípustné řešení.

Nechť nyní tab. 9 představuje 0-tou iteraci Gomoryho algoritmu. Do této tabulky je přidán Gomoryho řez jako další podmínka k původní úloze, přičemž za i_0-tý řádek k jeho vytvoření je zvolen ten, v němž má $b_{i_0}^*$ největší zlomkovou část pro bazickou základní (tj. původní) proměnnou v tab. 9. Tím získáme novou tabulku ve tvaru:

Tab. 10. Simplexová tabulka neceločíselné úlohy po doplnění prvního Gomoryho řezu

základní + doplňkové proměnné	v_1	**min**
M	0 . . . 0	$\boldsymbol{b}^*$
$-\{m_{i_0 1}\}$ . . . $-\{m_{i_0(n+m)}\}$	1	$-\{b_{i_0}^*\}$
$(\boldsymbol{z}_{\mathrm{M}} - \boldsymbol{c}_{\mathrm{M}})^{\mathrm{T}} \geq \mathbf{0}^{\mathrm{T}}$	0	L_1

V této tabulce je bazické řešení nepřípustné, protože v pravém sloupci je záporná hodnota $-\{b_{i_0}^*\}$. Reprezentuje však úlohu s přípustným duálním řešením $\mathbf{y} = (\tilde{\mathbf{y}}^{\mathrm{T}}, 0)^{\mathrm{T}}$, kde $\tilde{\mathbf{y}}$ je optimální řešení duální úlohy z (1.24). Tuto tabulku tedy transformujeme duálním simplexovým algoritmem opět na tvar tab. 9, ve které bude $\mathrm{M} \in \mathbb{R}^{(m+1)\times(n+m+1)}$ a $\mathbf{0} \leq \boldsymbol{b}^* \in \mathbb{R}^{m+1}$, přičemž v $\boldsymbol{b}^*$ je nyní o jednu celočíselnou složku navíc oproti tabulce původní. Tyto iterace opakujeme tak dlouho (řekněme **celkem *p*-krát**) dokud nevznikne tabulka

Tab. 11. Konečná simplexová tabulka Gomoryho algoritmu

základní + doplňkové proměnné v_1 . . . v_p	**min**
M	$\boldsymbol{b}^*$
$(\boldsymbol{z}_{\mathrm{M}} - \boldsymbol{c}_{\mathrm{M}})^{\mathrm{T}} \geq \mathbf{0}^{\mathrm{T}}$	L_p

v níž $\mathrm{M} \in \mathbb{R}^{(m+p)\times(n+m+p)}$ a $\mathbf{0} \leq \boldsymbol{b}^* \in \mathbb{R}^{m+p}$, přičemž všechny základní proměnné původní úlohy jsou již celočíselné. Tím Gomoryho algoritmus končí.

Než uvedeme příklady, popíšeme alespoň neformální algoritmus Gomoryho metody pro celočíselné řešení maximalizační úlohy (1.20). Nechť této úloze odpovídá kanonická úloha ve tvaru (1.22) s počáteční simplexovou tabulkou T_{init} ve tvaru tab. 2, str. 33. Potom **Gomoryho algoritmus** provádí následující kroky:

0. KROK: primárním simplexovým algoritmem transformuj tabulku T_{init} na tabulku T_0 ve tvaru tab. 9 a jdi na 1. KROK;

1. KROK: pokud T_0 obsahuje jen celočíselné základní proměnné, pak jdi na 3. KROK; v opačném případě polož $T_i := T_0$ a jdi na 2. KROK;

2. KROK: v tabulce T_i zvol v jejím pravém sloupci prvek $b_{i_0}^*$ s největší zlomkovou částí pro základní bazickou proměnnou a vytvoř Gomoryho řez (3.7), T_i doplň na tvar tab. 10 a duálním simplexovým algoritmem transformuj na tabulku T_{i+1}; pokud T_{i+1} obsahuje pouze celočíselné základní proměnné, pak jdi na 3. KROK; jinak polož $T_i := T_{i+1}$ a opakujeme tento krok;

3. KROK: tabulka T_i obsahuje optimální celočíselné řešení a algoritmus končí

Algoritmus poskytuje v poslední tabulce T_i již optimální celočíselné řešení úlohy (1.20).

Důležitými otázkami u každého algoritmu je jednak jeho konečnost, jednak jeho efektivnost (rychlost konvergence). V souvislosti s Gomoryho algoritmem se otázce konečnosti literatura buď vyhýbá, anebo ve zkratce praví, že je vždy konečný. Skutečností však je, že Gomory dokázal konečnost pro jinou (patrně pomalejší) variantu algoritmu, a to navíc za dalšího požadavku, který je např. splněn, je-li množina přípustných řešení výchozí úlohy (1.20) omezená (v praxi je obvykle splněn); podrobněji k tomu viz např. v [1]. Není nám známo, zda je konečnost dokázána pro výše uvedený algoritmus, ale na druhé straně jsme nikde v literatuře nezaznamenali, že by někdy selhal.

S výše uvedeným souvisí právě požadavek, aby byl ke konstrukci Gomoryho řezu zvolen řádek s nejvyšší zlomkovou částí prvku v posledním sloupci. Tento požadavek má zřejmě

urychlit algoritmus tím, že neceločíselné části řešení maximálním možným způsobem přesouvá do hodnot umělých proměnných. Patrně se jedná o požadavek heuristický, jenž nejspíše nemá přímou oporu v teoretické rovině.

Co se týká rychlosti Gomoryho algoritmu, je z numerických studií i praktických aplikací známo, že jeho konvergence není příliš dobrá. Důvod je vcelku zřejmý, Gomoryho řezy vyjadřují jen podmínku, která je splněna pro každé celočíselné přípustné řešení, ovšem **vůbec nic navíc!** Je proto až s podivem, že algoritmus vskutku konverguje. Nepatrného zrychlení lze docílit skutečností, že umělá proměnná, která během algoritmu jednou vystoupí z báze, se do ní již nikdy nevrátí. V té chvíli lze z tabulky tedy vynechat sloupec, který jí odpovídá. Běžné jsou případy, kdy i pro cvičné úlohy malých rozměrů, např. pouze 3×2, s „rozumnými“ koeficienty může počet iterací Gomoryho algoritmu několikanásobně překročit kterýkoliv z jejích rozměrů.

Zásadní je ovšem skutečnost, že pokud Gomoryho algoritmu skončí, potom našel vskutku optimální celočíselné řešení původní úlohy. Výsledná simplexová tabulka určitě reprezentuje optimální celočíselné bazické řešení jisté úlohy, protože všechny relativní ceny jsou nezáporné a pravý sloupec je celočíselný. Tato úloha však vznikla doplněním pouze takových omezení, která jsou splněna všemi celočíselnými řešeními původní úlohy, **a to včetně všech řešení optimálních** – a právě v tom spočívá hlavní myšlenka Gomoryho metody. Nalezené řešení, které je na konci jediným takovým, že splňuje všechny Gomoryho řezy, je tedy nutně optimálním celočíselným řešením původní úlohy (1.20).

Nyní uvedeme několik příkladů k demonstraci Gomory algoritmu. První příklad vyřešíme kompletně. V dalších příkladech se již budeme soustředit pouze na samotnou funkci Gomoryho algoritmu. Buď vyjdeme z již dostupné optimální simplexové tabulky primárního algoritmu relaxované úlohy, anebo průběh Gomoryho algoritmu jen okomentujeme, pokud by jeho kompletní prezentace manuálních výpočtů byla neúnosně zdlouhavá.

Příklad 5. Uvažujme následující maximalizační úlohu ve tvaru (1.20):

$$\begin{aligned} &\max\ (8x_1 + 9x_2) \\ &3x_1 + 6x_2 \leq 16 \\ &2x_1 + 2x_2 \leq 9 \\ &x_{1,2} \geq 0, \quad x_{1,2} \in \mathbb{Z} \end{aligned} \tag{3.8}$$

Po převedení (3.8) na kanonický tvar (1.22) doplněním skluzovými proměnnými $x_{3,4} \geq 0$ vyřešíme nejdříve relaxovanou úlohu primárním simplexovým algoritmem:

Tab. 12. Primární simplexový algoritmus v př. 5

x_1	x_2	x_3	x_4	
8	9	0	0	**max**
3	**6**	1	0	16
2	2	0	1	9
–8	–9	0	0	0
$\frac{1}{2}$	1	$\frac{1}{6}$	0	$\frac{8}{3}$
1	0	$-\frac{1}{3}$	1	$\frac{11}{3}$
$-\frac{7}{2}$	0	$\frac{3}{2}$	0	24
0	1	$\frac{1}{3}$	$-\frac{1}{2}$	$\mathbf{\frac{5}{6}}$
1	0	$-\frac{1}{3}$	1	$\frac{11}{3}$
0	0	$\frac{1}{3}$	$\frac{7}{2}$	$\frac{221}{6}$

Z tab. 12 plyne, že relaxace úlohy (3.8) má neceločíselné optimální řešení $\tilde{\mathbf{x}} = \left(\frac{11}{3}, \frac{5}{6}\right)^{\mathrm{T}}$ s optimální hodnotou účelová funkce $\frac{221}{6} = 36{,}8\overline{3}$, která je rovněž neceločíselná.

V tab. 12 v posledních dvou řádcích nad kriteriálním řádkem jsou zlomkové části pravých stran postupně $\frac{5}{6}$ a $\frac{2}{3}$ pro základní proměnné x_2 a x_1. Z prvního z nich tudíž plyne, že prvním Gomoryho řezem je podle (3.7) rovnice

$$-\frac{1}{3}x_3 - \frac{1}{2}x_4 + v_1 = -\frac{5}{6} \tag{3.9}$$

Do optimální tab. 12 přidáme do posledního řádku omezení (3.9) a na vzniklou tabulku aplikujeme duální algoritmus. Tím získáváme postupně následující sekvenci simplexových tabulek, z nichž poslední je opět optimální:

Tab. 13. Duální simplexový algoritmus pro první Gomoryho řez

x_1	x_2	x_3	x_4	v_1	**min**
0	1	$\frac{1}{3}$	$-\frac{1}{2}$	0	$\mathbf{\frac{5}{6}}$
1	0	$-\frac{1}{3}$	1	0	$\frac{11}{3}$
0	0	$\mathbf{-\frac{1}{3}}$	$-\frac{1}{2}$	1	$-\frac{5}{6}$
0	0	$\frac{1}{3}$	$\frac{7}{2}$	0	$\frac{221}{6}$
0	1	0	−1	1	0
1	0	0	$\frac{3}{2}$	−1	$\mathbf{\frac{9}{2}}$
0	0	1	$\frac{3}{2}$	−3	$\frac{5}{2}$
0	0	0	3	1	36

Z poslední části tab. 13 vyplývá, že optimální řešení $\tilde{\mathbf{x}} = \left(\frac{9}{2}, 0\right)^{\mathrm{T}}$ stále není celočíselné, ačkoliv optimální hodnota účelové funkce 36 již je.

Musíme pokračovat v algoritmu zavedením dalšího Gomoryho řezu. Druhý řádek nad řádkem kriteriálním odpovídá bazické základní proměnné x_1 se zlomkovou částí $\frac{1}{2}$. Opět podle (3.7) zkonstruujeme druhý Gomoryho řez s rovnicí

$$-\frac{1}{2}x_4 + v_2 = -\frac{1}{2} \tag{3.10}$$

Do optimální tab. 13 přidáme (3.10) a na vzniklou tabulku opět aplikujeme duální algoritmus, přičemž získáme následující simplexové tabulky:

Tab. 14. Duální simplexový algoritmus pro druhý Gomoryho řez

x_1	x_2	x_3	x_4	v_1	v_2	**min**
0	1	0	−1	1	0	0
1	0	0	$\frac{3}{2}$	−1	0	$\frac{9}{2}$
0	0	1	$\frac{3}{2}$	−3	0	$\frac{5}{2}$
0	0	0	$-\frac{1}{2}$	0	1	$-\frac{1}{2}$
0	0	0	3	1	0	36
0	1	0	0	1	−2	1
1	0	0	0	−1	3	3
0	0	1	0	−3	3	1
0	0	0	1	0	−2	1
0	0	0	0	1	6	33

V tab. 14 již nalézáme optimální celočíselné řešení úlohy (3.8): $\tilde{\mathbf{x}} = (3,1)^{\mathrm{T}}$, přičemž optimální hodnota účelové funkce je 33.

Příklad 6. Tento příklad je víceméně pouze komentářem k př. 4 z odd. 2.3. Předposlední řádek tab. 7 má tvar $(a_{31}, \dots, a_{36} \mid b_3) = \left(1,0,\frac{1}{2},0,\frac{1}{2},0 \,\middle|\, \frac{5}{2}\right)$. Odtud je ihned zřejmé, že omezení (2.18) je Gomoryho řezem ve tvaru (3.7), který přísluší zmíněnému řádku relaxované úlohy. K dosažení celočíselného optima postačila v tomto případě jediná iterace Gomoryho algoritmu. Jak jsme již zmínili, je taková situace velmi výjimečná. Nicméně zde k rychlému zakončení algoritmu přispělo patrně to, že neceločíselné řešení leželo v blízkosti (v grafovém smyslu) degenerovaných řešení s více nulovými, a tudíž automaticky celočíselnými složkami.

Příklad 7. Tento příklad je komentářem k př. 1 z odd. 2.3, který byl vyřešen rovněž grafickou metodou v odd. 1.2. S požadavkem celočíselnosti se nyní jedná o následující úlohu:

$$\begin{aligned} &\max\ (2x_1 + 3x_2) \\ &x_1 + 3x_2 \leq 8 \\ &3x_1 + 2x_2 \leq 8 \\ &x_{1,2} \geq 0, \quad x_{1,2} \in \mathbb{Z} \end{aligned} \tag{3.11}$$

Relaxovaná úloha byla vyřešena jednoduchým primárním simplexovým algoritmem s optimálním neceločíselným řešením $\tilde{\mathbf{x}} = \left(\frac{8}{7}, \frac{16}{7}\right)^{\mathrm{T}}$. Přímo z obr. 2, odd. 1.2 lze okamžitě nahlédnout, že celočíselné řešení v tomto případě vznikne pouhým zaokrouhlením uvedeného řešení směrem dolů: $\tilde{\mathbf{x}}_{\mathbb{Z}} = (1,2)^{\mathrm{T}}$. I přes tuto skutečnost zde Gomoryho algoritmus pracuje dosti pomalu. Jsou zapotřebí celkem čtyři Gomoryho řezy, přičemž duální algoritmy mají při řešení několik kroků, kterými se vrací dokonce k ryze neceločíselným bazickým řešením. Kdybychom měli vypsat všechny potřebné simplexové tabulky, včetně komentářů, potřebovali bychom cca další tři-čtyři strany. Proto u tohoto poučného konstatování skončíme.

Příklad 7 zde upozorňuje na jeden důležitý fakt. Gomoryho algoritmus se v zásadě neumí vypořádat s tak triviální situací, jakou je to, že optimální celočíselné řešení vznikne pouhým zaokrouhlením (i když nakonec řešení nutně musí najít).

Výše uvedená skutečnost jen podtrhuje všeobecně známý fakt, že Gomoryho metoda obecně konverguje značně pomalu. Proto vzniklo postupně mnoho jejích vylepšení. U zrodu jedné z takových metod, která se umí vyrovnat se situací z př. 7, stojí věhlasný matematik českého původu Vašek Chvátal (takto se oficiálně uvádí jako autor svých publikací). Tato metoda generuje řezy sofistikovanějším způsobem, tzv. Chvátalovy-Gomoryho řezy. Další vylepšení původní metody šla směrem k redukcím rozměrů instancí úlohy v průběhu algoritmu. Oba směry vylepšení lze spolu rovněž kombinovat; k tomu viz podrobně [5]. Z tohoto textu si dovolíme jednu citaci: „Historii Gomoryho řezů provází ve vědecké komunitě oscilace mezi přijetím a odmítáním. Nejprve byly považovány za zázračné, pak zcela odmítnuty ve prospěch metod branch & bound a nakonec opět povolány do boje s celočíselnými programy.“

Vývoj celočíselného programování nešel ovšem jen cestou vylepšování Gomoryho metody. V obecné rovině vznikla např. metoda větví a mezí, o které je pojednáno v dalším oddílu. Vývoj šel ovšem také vytvářením metod vyvinutých pro speciální úlohy, které jsou založeny na odlišných principech. Máme zde na mysli např. již zmíněný dopravní problém

s požadavkem celočíselnosti či jeho speciální případ přiřazovacího problému. Lze na ně sice aplikovat Gomoryho metodu, ale specifické algoritmy řeší tyto úlohy nesrovnatelně efektivněji [1, 2, 3, 8, 12].

Závěrem uvedeme argumentaci, která aspoň neformálně objasňuje nutnost předpokladu, že pro správnou funkci Gomoryho algoritmu musí být úloha (1.20) celočíselně celočíselná, čili že $\mathrm{A} \in \mathbb{Z}^{m\times n}$ a $\boldsymbol{b} \in \mathbb{Z}^{m}$. Uvažujme nyní dvě úlohy:

$$\begin{array}{ll} \max\left(\boldsymbol{c}^{\mathrm{T}}\mathbf{x}\right) & \max\left(\boldsymbol{c}^{\mathrm{T}}\mathbf{x}\right) \\ \mathrm{A}\mathbf{x} \leq \boldsymbol{b} & \widehat{\mathrm{A}}\mathbf{x} \leq \widehat{\boldsymbol{b}} \\ \mathbf{x} \geq \mathbf{0} & \mathbf{x} \geq \mathbf{0} \end{array} \tag{3.12}$$

Nechť matice $(\mathrm{A}|\boldsymbol{b}) \in \mathbb{Q}^{m\times(n+1)}$, $\left(\widehat{\mathrm{A}}\middle|\widehat{\boldsymbol{b}}\right) \in \mathbb{Z}^{m\times(n+1)}$, přičemž $\left(\widehat{\mathrm{A}}\middle|\widehat{\boldsymbol{b}}\right)$ vzniká z $(\mathrm{A}|\boldsymbol{b})$ pouze vynásobením řádků vhodnými kladnými konstantami. Obě úlohy jsou ekvivalentní: mají stejné množiny přípustných řešení a mají stejná optimální řešení. Jejich kanonické přepisy do tvaru (1.22) jsou potom následující:

$$\begin{array}{ll} \max\left(\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}}\right) & \max\left(\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}}\right) \\ (\mathrm{A}|\mathrm{I}_m)\tilde{\mathbf{x}} = \boldsymbol{b} & \left(\widehat{\mathrm{A}}\middle|\mathrm{I}_m\right)\tilde{\mathbf{x}} = \widehat{\boldsymbol{b}} \\ \tilde{\mathbf{x}} \geq \mathbf{0} & \tilde{\mathbf{x}} \geq \mathbf{0} \end{array} \tag{3.13}$$

Zatímco úlohy (3.12) jsou ekvivalentní, úlohy (3.13) zjevně ekvivalentní nejsou. Stačí, aby existoval prvek $a_{ij} \notin \mathbb{Z}$ matice A, a úlohy budou mít různé množiny přípustných řešení. Navíc přípustné bazické řešení $\tilde{\mathbf{x}} = (\mathbf{0}^{\mathrm{T}}, \boldsymbol{b}^{\mathrm{T}})^{\mathrm{T}}$ první úlohy je obecně neceločíselné a může být Gomoryho metodou odříznuto, přípustné bazické řešení $\tilde{\mathbf{x}} = \left(\mathbf{0}^{\mathrm{T}}, \widehat{\boldsymbol{b}}^{\mathrm{T}}\right)^{\mathrm{T}}$ druhé úlohy je celočíselné a nikdy odříznuto nebude.

Již z výše uvedené triviální úvahy je zřejmé, co se může stát. Teorie říká, že optimální simplexové tabulky T_1 a T_2, které odpovídají postupně kanonickým úlohám (3.13), budou mít stejný pravý sloupec a část kriteriálního řádku, v němž je uchováno řešení duální úlohy, ovšem obecně budou různé. Gomoryho algoritmus inicializovaný T_1 bude v konečném důsledku odřezávat i celočíselná řešení, mezi nimiž mohou být i řešení optimální, a v krajním případě může být ukončen nestandardně. Jinými slovy bude indikovat, že celočíselné optimální řešení vůbec neexistuje. Gomoryho algoritmus inicializovaný tabulkou T_2 nikdy žádné

celočíselné řešení neodřízne. Předpoklad „celočíselné celočíselnosti“ základní úlohy (1.20) je tudíž pro správnou inicializaci Gomoryho algoritmu zcela nezbytný: je třeba vždy položit $T_{init} := T_2$. Koneckonců převod úlohy (1.20) na celočíselně celočíselnou úlohu je triviální záležitostí, ovšem nicméně podstatný.

Příklad úlohy, o níž je známo, že Gomoryho algoritmus u ní bez převodu na celočíselně celočíselný tvar selže, jsme převzali z [5]. Úloha je velmi jednoduchá:

$$\begin{aligned} &\max\ (x_1 - x_2) \\ &-\tfrac{1}{3}x_1 \ + x_2 \le \tfrac{1}{3} \\ &\quad x_1 - \tfrac{1}{3}x_2 \le \tfrac{1}{3} \\ &x_{1,2} \ge 0, \quad x_{1,2} \in \mathbb{Z} \end{aligned} \tag{3.14}$$

Úlohu (3.14) převedeme na celočíselně celočíselnou úlohu do tvaru

$$\begin{aligned} &\max\ (x_1 - x_2) \\ &-x_1 + 3x_2 \le 1 \\ &3x_1 \ - x_2 \le 1 \\ &x_{1,2} \ge 0, \quad x_{1,2} \in \mathbb{Z} \end{aligned} \tag{3.15}$$

Relaxace obou úloh (3.14) i (3.15) mají společné neceločíselné optimální řešení $\tilde{\mathbf{x}} = \left(\frac{1}{3}, 0\right)^{\mathrm{T}}$ a úlohy k nim duální mají společné optimální řešení $\tilde{\mathbf{y}} = (1,0)^{\mathrm{T}}$ se stejnou hodnotou účelové funkce $\tilde{z} = \boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}} = \boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}} = \frac{1}{3}$. Relaxace k nim příslušných kanonických úloh mají pochopitelně rovněž společné neceločíselné řešení $\mathbf{x}^* = \left(\frac{1}{3}, 0, \frac{4}{9}, 0\right)^{\mathrm{T}}$.

Obě úlohy (3.14) i (3.15) mají totéž optimální celočíselné řešení $\tilde{\mathbf{x}}_{\mathbb{Z}} = (0,0)^{\mathrm{T}}$. Ovšem ke správné funkci Gomoryho algoritmu je vhodná pouze úloha (3.15). Pokud bychom ke spuštění Gomoryho algoritmu použili optimální simplexovou tabulku úlohy (3.14), pak bude již po přidání prvního řezu duální simplexový algoritmus zastaven nestandardně – v řádku se zápornou hodnotou v pravém sloupci jsou jen nezáporné prvky. To znamená, že řez je nepřípustný a Gomoryho algoritmus bude indikovat, že množina přípustných řešení relaxované

kanonické úlohy k (3.14) neobsahuje žádné celočíselné řešení (což je pochopitelně pravda). Relaxace kanonické úlohy k (3.15) samozřejmě celočíselná řešení má, třeba $\mathbf{x} = (0,0,1,1)^{\mathrm{T}}$.

Úlohu (3.14) se pokusíme využít v praktické části této práce k testu použitého software, zda je Gomoryho metoda ošetřena vůči úlohám, které nejsou celočíselně celočíselné.

Poznamenejme, že existuje varianta Gomoryho metody, která již nevyžaduje celočíselnou celočíselnost výchozí úlohy, v níž jsou řezy generovány sofistikovanějším způsobem (na základě několika řádků současně, nikoliv jen řádku jediného). V té souvislosti se potom zde popsaná metoda podrobněji nazývá ***první Gomoryho algoritmus***, zatímco zmíněná varianta je ***druhý Gomoryho algoritmu***. Gomoryho řezy druhého typu nemohou být příliš hluboké, aby neodřezávaly celočíselná řešení, nicméně existuje celá řada racionálních i intuitivních důvodů domnívat se, že oba algoritmy budou v základních parametrech (rychlost, operační a paměťové nároky) přibližně srovnatelné. Ovšem tuto domněnku nemůžeme nijak doložit.

3.2 Metoda větví a mezí

Tato metoda byla vyvinuta podstatně později než Gomoryho metoda, neboť její vznik byl bezesporu podmíněn rozvojem výpočetní techniky, jakož i myšlenek teoretické kybernetiky. Ve své podstatě se jedná o metodu zcela univerzální, kterou lze použít k řešení libovolného celočíselného optimalizačního problému. Myšlenka metody větví a mezí se objevila poprvé cca v polovině minulého století. Bylo pouze logickým vyústěním, že se tato obecná myšlenka uchytila právě v LP. První úplnou formulaci algoritmu podaly v roce 1960 anglické matematičky Ailsa Land*ová* a Alison Doig*ová*. Nicméně trvalo ještě pět let, nežli nová generace sálových počítačů (patrně od IBM) a programovacích jazyků (snad některá z prvních verzí COBOLu či FORTRANu) umožnila vznik první použitelné implementace algoritmu, kterou realizoval americký teoretický kybernetik a specialista na programovací jazyky R. J. Dakin a popsal ve svém článku z roku 1965.

Při popisu základní myšlenky algoritmu budeme opět vycházet z obecně neceločíselného řešení maximalizační úlohy (1.20), v níž nyní přibude požadavek celočíselnosti. Optimální řešení relaxované úlohy nechť je $\tilde{\mathbf{x}} = (\tilde{x}_1, \dots, \tilde{x}_n)^{\mathrm{T}}$. Tuto úlohu lze pro naše účely dostatečně popsat trojicí $(M, \tilde{\mathbf{x}}, \tilde{z})$, kde M je množina přípustných řešení a $\tilde{z} = \boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}}$ je maximální hodnota účelové funkce. Bude-li hodnota $\tilde{x}_l$ ($l \in \{1,2, \dots, n\}$) vybraná neceločíselná složka řešení $\tilde{\mathbf{x}}$, potom celočíselná hodnota jí příslušné proměnné x_l splňuje některé ze dvou omezení

$$x_l \leq \lfloor \tilde{x}_l \rfloor, \qquad \text{resp.} \qquad x_l \geq \lfloor \tilde{x}_l \rfloor + 1 \tag{3.16}$$

Označíme-li $M_1 = \{\mathbf{x} \in M \mid x_l \leq \lfloor \tilde{x}_l \rfloor\}$ a $M_2 = \{\mathbf{x} \in M \mid x_l \geq \lfloor \tilde{x}_l \rfloor + 1\}$, rozvětví se původní úloha na dvě podúlohy

$$(M_1, \tilde{\mathbf{x}}_1, \tilde{z}_1), \quad \text{resp.} \quad (M_2, \tilde{\mathbf{x}}_2, \tilde{z}_2), \tag{3.17}$$

kde všechny složky úloh (3.17) mají stejný význam jako výše (pokud je $M_i = \emptyset$, nemusí být $\tilde{\mathbf{x}}_i$ a $\tilde{z}_i$ přiřazeny žádné hodnoty). Jestliže $\tilde{z}_i > \tilde{z}_j$, $i, j \in \{0,1\}$, zvolíme úlohu $(M_i, \tilde{\mathbf{x}}_i, \tilde{z}_i)$, kterou opět rozložíme na dvě podúlohy stejným způsobem. Je podstatné, že v tomto procesu větvení na podúlohy nakonec nutně dospějeme k celočíselné úloze, ačkoliv nemusí obsahovat optimální celočíselné řešení úlohy původní. To, abychom dospěli k optimálnímu celočíselnému řešení původní úlohy, zajistí ***algoritmus větví a mezí***, který maximálně redukuje počet podúloh, jež nevedou k cíli, a který dále neformálně popíšeme (ponecháme stranou programátorské problémy, např. přečíslovávání úloh v běhu algoritmu i další „formality“).

Nejdříve zavedeme dva zásobníky: zásobník aktivních úloh $\mathcal{Z}$ a zásobník umrtvených úloh $\mathcal{U}$; a reálnou proměnnou z^*, v níž bude uchovávána aktuálně nejvyšší dosažená hodnota účelové funkce celočíselného řešení. Na konci algoritmu bude $\mathcal{Z} = \emptyset$ a $\mathcal{U}$ bude obsahovat všechny úlohy, které obsahují optimální celočíselná řešení a společnou maximální hodnotu účelové funkce původní úlohy. **Algoritmus větví a mezí** má následující imperativní formu:

0. KROK: polož $(M_0, \tilde{\mathbf{x}}_0, \tilde{z}_0) := (M, \tilde{\mathbf{x}}, \tilde{z})$, $\mathcal{Z} := \emptyset$, $\mathcal{U} := \emptyset$ a $z^* := -\infty$; jakmile je již $(M_0, \tilde{\mathbf{x}}_0, \tilde{z}_0)$ celočíselná, zařaď ji do $\mathcal{U}$, algoritmus končí; jinak je-li $\tilde{z}_i > \tilde{z}_j$, $i, j \in \{0,1\}$ v úlohách (3.17), zvol do 1. KROKU úlohu $(M_i, \tilde{\mathbf{x}}_i, \tilde{z}_i)$ a úlohu $\left(M_j, \tilde{\mathbf{x}}_j, \tilde{z}_j\right)$ zařaď do $\mathcal{Z}$; pokud některá z úloh nemá řešení, je vyřazena a zbylá je zvolena do 1. KROKU jako $(M_i, \tilde{\mathbf{x}}_i, \tilde{z}_i)$;

1. KROK: úloha $(M_i, \tilde{\mathbf{x}}_i, \tilde{z}_i)$ je rozvětvena doplněním omezení (3.16) na dvě podúlohy

$$(M_{i+1}, \tilde{\mathbf{x}}_{i+1}, \tilde{z}_{i+1}) \quad \text{a} \quad (M_{i+2}, \tilde{\mathbf{x}}_{i+2}, \tilde{z}_{i+2}); \tag{3.18}$$

má-li některá z úloh (3.18) celočíselné řešení, přejdi ke 2. KROKU; jinak je-li $\tilde{z}_k > \tilde{z}_l$, $k, l \in \{i+1, i+2\}$ v úlohách (3.18), zvol $(M_i, \tilde{\mathbf{x}}_i, \tilde{z}_i) := (M_k, \tilde{\mathbf{x}}_k, \tilde{z}_k)$, úlohu $(M_l, \tilde{\mathbf{x}}_l, \tilde{z}_l)$

zařaď do $\mathcal{Z}$ a opakuj 1. KROK; pokud některá z úloh nemá řešení, je vyřazena, zvolena je zbylá úloha jako $(M_i, \tilde{\mathbf{x}}_i, \tilde{z}_i)$ a opakuje se 1. KROK;

2. KROK: celočíselnou úlohu $(M_i, \tilde{\mathbf{x}}_i, \tilde{z}_i)$ zařaď do $\mathcal{U}$ a polož $z^* := \tilde{z}_i$; z obou zásobníků $\mathcal{Z}$ a $\mathcal{U}$ vyřaď všechny úlohy $(M_j, \tilde{\mathbf{x}}_j, \tilde{z}_j)$, pro něž je $\tilde{z}_j < \tilde{z}_i$; je-li $\mathcal{Z} = \emptyset$, algoritmus končí; jinak ze $\mathcal{Z}$ **vyjmi** $(M_j, \tilde{\mathbf{x}}_j, \tilde{z}_j)$ s největší hodnotou $\tilde{z}_j$, pak polož $(M_i, \tilde{\mathbf{x}}_i, \tilde{z}_i) := (M_j, \tilde{\mathbf{x}}_j, \tilde{z}_j)$ a vrať se k 1. KROKU

Po ukončení algoritmu je zásobník $\mathcal{U} \neq \emptyset$ a zůstanou v něm právě jen všechny celočíselné úlohy (s totožnou) maximální hodnotou účelové funkce, které řeší úlohu (1.20) s nutnou podmínkou celočíselnosti všech proměnných.

Velmi podstatná je i u algoritmu větví a mezí otázka jeho konečnosti. Zejména u tohoto algoritmu je ovšem zodpovězení této otázky podstatně jednodušší než u ostatních zde uvedených algoritmů. V praxi je obvykle množina přípustných řešení omezená. Je zřejmé, že v takovém případě musí být konečný i algoritmus, neboť každá složka libovolného přípustného řešení může nabývat jen konečně mnoha celočíselných hodnot.

Algoritmus větví a mezí představuje procházení binárního stromu s kořenem v původní úloze. Méně formálně lze říci, že pěstuje a zušlechťuje binární strom v následujícím smyslu. Každá úloha je rozvětvena na dvě podúlohy doplněním jednoho ze dvou omezení typu (3.16). Větvení dále probíhá vždy v podúloze, která má větší hodnotu účelové funkce, a tudíž větší naději k nalezení celočíselného optima, a to do té doby, dokud není nalezena celočíselná úloha (nikoli nutně optimální) – zde větvení končí. Následně jsou odřezány všechny větve, v jejichž uzlech nelze očekávat nalezení lepšího celočíselného řešení. Poté probíhá větvení podle listu s největší hodnotou účelové funkce. Tento proces větvení a „prořezávání“ stromu probíhá tak dlouho, dokud v listech stromu nezůstanou jen úlohy, které reprezentují optimální celočíselná řešení původní úlohy. Pokud je optimální celočíselné řešení jediné, zůstane pouze jediná cesta, vedoucí z kořene k tomuto řešení, vše ostatní je odříznuto. Pochopitelně při grafickém zobrazení tohoto stromu jsou v něm znázorněny i všechny „jalové“ větve, a to včetně aktuálního optimálního řešení a hodnoty účelové funkce.

Z uvedeného je nyní zřejmé, že je-li algoritmus větví a mezí konečný, nalezne tento algoritmus všechna celočíselná řešení původní úlohy, neboť zřejmě nevynechá žádnou celočíselnou kombinaci základních proměnných, jež by mohla vést k řešení optimálnímu. Z téhož ovšem plyne i zásadní nedostatek metody. Klade vysoké operační a paměťové nároky,

protože, dokud není algoritmus standardně ukončen, nejsme schopni poznat, že optimální řešení mohlo být již dávno nalezeno.

Všechny kroky algoritmu větví a mezí jsou jednoznačně určeny až na jedinou výjimku, proto není ve vlastním algoritmu ani zmíněna. Obvykle není specifikováno, jakým způsobem je vybrána proměnná do nových omezení (3.16). Algoritmus je kombinatorické povahy, a tudíž již z tohoto důvodu klade značné operační i paměťové nároky. Mnoho autorů přistupuje tedy na stanovisko, že je to v zásadě jedno, a vybírají neceločíselné proměnné jednoduše postupně podle jejich pořadí v původní úloze. Pouze v [5] lze najít některá sofistikovanější pravidla. Zmiňme zde alespoň pravidlo, kdy je vybrána proměnná podle „míry neceločíselnosti“, tj. proměnná s indexem

$$l = \arg\max_i \left\{ \min\left\{ \{\tilde{x}_i\}, 1 - \{\tilde{x}_i\} \right\} \,\middle|\, \tilde{x}_i \notin \mathbb{Z} \right\}$$

Stejně tak lze alespoň teoreticky připustit možnost, že větvící podúlohy budou mít stejné hodnoty účelových funkcí, anebo že bude v zásobníku $\mathcal{Z}$ více úloh se stejnou největší hodnotou účelové funkce. Nicméně se zdá, že většina programů tyto velmi nepravděpodobné eventuality v praktických úlohách nijak nemá ošetřeny. Koneckonců vždy je k dispozici univerzální pravidlo nejmenších indexů.

Komerční modulárně stavěné programy patrně řeší v každém větvení celou úlohu od začátku, ovšem někteří autoři upozorňují na skutečnost, že implementace, v nichž jsou rozvětvené úlohy „přeoptimalizovány“ duálním simplexovým algoritmem, výrazně urychlují celou metodu. Pro podmínku typu $x_l \geq \lfloor \tilde{x}_l \rfloor + 1$ lze postupovat přímým doplněním omezení stejně jako při doplnění Gomoryho řezu. U podmínky typu $x_l \leq \lfloor \tilde{x}_l \rfloor$ je tvar doplněného omezení poněkud komplikovanější (viz event. [5]), avšak úspora výpočtů je i tak nezanedbatelná (anebo je nutno celou úlohu řešit od samého začátku primárním simplexovým algoritmem).

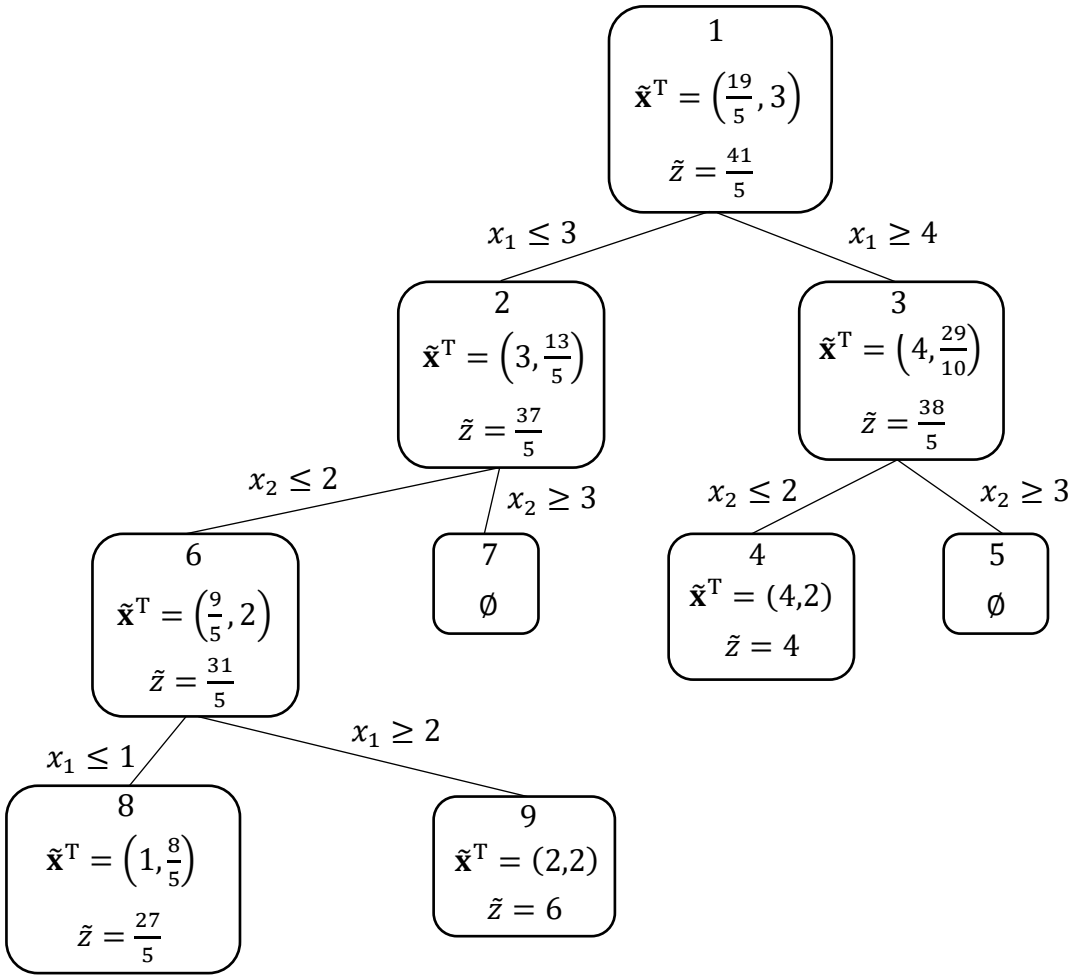
Nakonec uveďme, že mezi Gomoryho algoritmem a algoritmem větví a mezí je zásadní rozdíl. Zatímco Gomoryho algoritmus najde vždy pouze jedno optimální řešení, byť by jich existovalo i více, algoritmus větví a mezí najde současně řešení všechna.

Algoritmus mezí a větví je pochopitelně manuálně náročný, proto je následující ilustrační příklad velmi jednoduchý.

Příklad 8. Metodou větví a mezí řešme následující maximalizační úlohu:

$$\begin{aligned} &\max\ (-x_1 + 4x_2) \\ &-10x_1 + 20x_2 \leq 22 \\ &\quad 5x_1 + 10x_2 \leq 49 \\ &\quad\ x_1 \qquad\quad \leq 5 \\ &x_{1,2} \geq 0, \quad x_{1,2} \in \mathbb{Z} \end{aligned} \tag{3.19}$$

Ručními výpočty při výběru neceločíselných hodnot optimálního řešení relaxované úlohy vzestupně podle jejich indexů dává algoritmus následující strom řešení:



Obr. 5. Binární strom řešení úlohy (3.19)

Ve stromu na obr. 5 jsou úlohy číslovány v pořadí, v jakém jsou procházeny postupně algoritmem. Jak je vidět, v listech stromu leží jediná úloha 9 ze seznamu $\mathcal{U}$, úlohy 5 a 7 se do žádného ze seznamů $\mathcal{Z}$ a $\mathcal{U}$ nikdy nedostaly a úloha 8 byla ze $\mathcal{Z}$ vyňata v posledním

kroku. Úloha 9 obsahuje optimální řešení původní úlohy (3.19): maximální hodnoty účelové funkce $\tilde{z} = 6$ je dosaženo v bodě $\tilde{\mathbf{x}} = (2,2)^{\mathrm{T}}$.

Již v zásadě triviální př. 8 ukazuje, že metoda větví a mezí bude klást vyšší nároky na operační a paměťové kapacity než Gomoryho metoda.

3.3 Porovnání Gomoryho metody s metodou větví a mezí

Na závěr uvedeme některé další metody či přístupy k celočíselnému LP, ačkoliv o nich sotva můžeme říci více nežli to, že existují. Poté krátce srovnáme Gomoryho metodu s metodou větví a mezí z hlediska jejich vzájemných výhod a nevýhod.

Metody dynamického programování spočívají obecně v dekompozici úlohy na podúlohy podle zvoleného rozhodovacího procesu. Tyto podúlohy jsou ukládány pro další možné použití. Vzniklé podúlohy lze potom dále rozkládat na základě téhož rozhodovacího procesu, který může být rovněž adaptivní, čímž je ovlivňována efektivnost řešícího algoritmu. Jak je vidět, metoda větví a mezí patří právě mezi tyto metody s neadaptivním rozhodováním.

Speciální metody jsou určeny k řešení úloh se specifickou strukturou ad hoc. Např. jsme již letmo zmínili metody pro řešení dopravního problému, spec. pak pro řešení přiřazovacího problému. Tyto metody jsou založeny na diametrálně odlišných matematických principech, což jim umožňuje být nesrovnatelně efektivnější než zde probrané obecné postupy.

Heuristické metody jsou často používány tam, kde by jakékoliv exaktní řešení úplně selhalo, anebo by bylo časově neúnosně náročné. Jejich nevýhodou je, že jsou – jak naznačuje jejich název – často založeny především na semiempirických znalostech, takže nám nezaručí nalezení optima, nýbrž obvykle pouze suboptima, které je v tom či onom smyslu optimu blízké. Nicméně jejich předností je, že jsou schopny poskytnout vždy řešení aspoň nějaké!

Výhodou Gomoryho metody je to, že je rychlejší než metoda větví a mezí, neboť metodu větví a mezí znatelně zpomalují vysoké výpočetní a paměťové nároky, což je obecný handicap všech algoritmů kombinatorické povahy. Nevýhoda Gomoryho metody naopak spočívá v tom, že v případě nejednoznačnosti nalezne pouze jediné řešení (snad nejbližší neceločíselnému řešení v euklidovské metrice – je to ovšem jen dohad, protože již z triviálního př. 7 víme, že Gomoryho řezy se mohou chovat dosti nevyzpytatelně). Ovšem naopak metoda větví a mezí – pokud je dokončena – najde vždy řešení všechna.

Nevýhodou Gomoryho metody je naopak to, že je mnohem citlivější na interní zaokrouhlování implementace než metoda větví a mezí. Každá implementace může totiž poznat celočíselnou hodnotu jen tak, že tato hodnota nepřeteče jisté meze kolem celočíselné hodnoty. Často se stává, že Gomoryho metoda již celočíselné řešení našla, ale implementace algoritmu to nepozná a pokračuje v přidávání dalších řezů. Tyto redundantní řezy pak buď přivedou numerické řešení do tolerančních mezí, anebo je algoritmus ukončen nestandardně, má-li jeho implementace vůči této situaci nějakou pojistku.

Výhodou metody větví a mezí je rovněž skutečnost, že v podstatě beze změny ji lze použít na tzv. smíšené celočíselné úlohy, tj. takové, v nichž požadujeme celočíselnost jen u některých proměnných. Naopak Gomoryho metodu, alespoň v její původní verzi, lze použít pouze na čisté celočíselné úlohy.

Nicméně Gomoryho metoda byla několikrát modifikována, takže si již také umí poradit se smíšenými úlohami volbou vhodných řezů. Stejně tak si umí poradit s patologickou situací, kterou jsme pozorovali u př. 7. V této souvislosti připomeňme již zmíněné Chvátalovy-Gomoryho řezy, které odřezávají neceločíselná řešení rafinovaněji. Cenou za to je, že důkaz konečnosti algoritmu je zde mnohem komplikovanější.

Ačkoliv se zdá, že na metodě větví a mezí se toho příliš modifikovat nedá, přeci se dělaly pokusy tuto metodu provádět jinak. V algoritmu, který jsme uvedli v odd. 3.2, byly úlohy uchovávány v zásobnících, což je typické pro prohledávání stromu do hloubky. Byly činěny pokusy s prohledáváním do šířky, kdy stojí úlohy ve frontě. Numerické experimenty ovšem naznačují, že prohledávání do hloubky stále zůstává nadějnější, čemuž napomáhá především odřezávání jalových větví ve stromu řešení. Zejména v dnešní době je pak pro metodu větví a mezí rovněž podstatné, že je snadno paralelizovatelná, kdy jsou prováděny výpočty v jednotlivých větvích na spřažených počítačích. V závislosti na dostupnosti počítačové mašinerie mohou být pak řešeny úlohy vskutku impozantních dimenzí (v řádech desítek tisíc proměnných a stovek tisíc omezení), o jejichž řešení se mohlo průkopníkům LP sotva zdát.

Nakonec uveďme jedinou v rešerši zaznamenanou letmou zmínku, že nejefektivnější dnes existující postupy pro celočíselné LP kombinují obě metody: Gomoryho metodu s metodou větví a mezí. Z obecného pohledu by mělo patrně jít o postupy z oblasti metod dynamického programování, kdy adaptivní proces vybírá v jednotlivých fázích řešení úlohy jednu nebo druhou metodu. Nic podrobnějšího k popisu takového postupu, zejména na základě jakých kritérií jsou vybírány metody, jsme však bohužel nezaznamenali. Na tomto principu je pravděpodobně založena i funkce MATLAB*u*, kterou používáme v praktické části.

3.4 Doplněk k duálnímu simplexovému algoritmu

V obou výše popsaných algoritmech celočíselného LP je přítomen duální simplexový algoritmus jako pomocná, ovšem obligátní procedura. Tento algoritmus jsme plně popsali v odd. 2.3, nicméně z důvodu plynulosti výkladu jsme tam uvedli jen příklad, který demonstroval zmíněnou pomocnou funkci duálního algoritmu. Nikoliv pouze z kontrolních důvodů je však mnohdy podstatné řešit úlohu duální, neboť reálná situace přímo vyžaduje model v jejím tvaru. V krajním případě můžeme vyřešit úlohu duální, zatímco software, resp. hardware bude kolabovat při řešení primární úlohy. Zde tedy vyřešíme kompletně jednu duální úlohu, již můžeme porovnat s řešením úlohy primární, kterou máme již k dispozici, a to včetně jejího řešení. To nám dovolí podrobněji rovněž komentovat běh a tvar výsledků získaných duálním simplexovým algoritmem.

Použijeme úlohu (2.9) na str. 37:

$$\begin{aligned} &\max\ (3x_1 + 2x_2 + 4x_3) \\ &\begin{aligned} x_1 \ &+ x_2 + 2x_3 \le 4 \\ 2x_1 \ &+ \ x_3 \le 5 \\ 2x_1 \ &+ x_2 + 3x_3 \le 7 \end{aligned} \\ &x_{1,2,3} \ge 0 \end{aligned} \tag{3.20}$$

o níž víme, že má optimální řešení $\tilde{\mathbf{x}} = \left(\frac{5}{2}, \frac{3}{2}, 0\right)^{\mathrm{T}}$ s hodnotou účelové funkce $\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}} = \frac{21}{2} = \boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}}$ a optimálním řešením duální úlohy je $\tilde{\mathbf{y}} = \left(2, \frac{1}{2}, 0\right)^{\mathrm{T}}$. Úloha (3.20) je ekvivalentní úloze

$$\begin{aligned} &\min\ (-3x_1 - 2x_2 - 4x_3) \\ &\begin{aligned} x_1 \ &+ x_2 + 2x_3 \le 4 \\ 2x_1 \ &+ \ x_3 \le 5 \\ 2x_1 \ &+ x_2 + 3x_3 \le 7 \end{aligned} \\ &x_{1,2,3} \ge 0 \end{aligned} \tag{3.21}$$

Podle (1.24) má duální úloha k (3.20) evidentně explicitní tvar

$$\begin{aligned} &\min\ (4y_1 + 5y_2 + 7y_3) \\ &\begin{aligned} y_1 \ &+ 2y_2 + 2y_3 \geq 3 \\ y_1 \ &\quad + \ \ y_3 \geq 2 \\ 2y_1 \ &+ y_2 + 3y_3 \geq 4 \end{aligned} \\ &y_{1,2,3} \geq 0 \end{aligned} \tag{3.22}$$

Víme, že základním smyslem duálního algoritmu je řešit duální úlohu v primární simplexové tabulce. Proto přepíšeme (3.22) doplněním skluzových proměnných $y_{4,5,6} \geq 0$ na následující kanonickou úlohu

$$\begin{aligned} &\max\ (-4y_1 - 5y_2 - 7y_3) \\ &\begin{aligned} -y_1 \ &- 2y_2 - 2y_3 + y_4 \quad\quad\ = -3 \\ -y_1 \ &\quad - \ y_3 \quad + \ y_5 \quad = -2 \\ -2y_1 \ &- y_2 - 3y_3 \quad\quad + \ y_6 = -4 \end{aligned} \\ &y_{1,2,3} \geq 0 \end{aligned} \tag{3.23}$$

Zapíšeme-li úlohu do primární simplexové tabulky ve tvaru tab. 6 na str. 43, bude její kriteriální řádek nezáporný. Dalším požadavkem inicializace duálního algoritmu je nepřípustnost primárního bazického řešení a existence přípustného řešení duální úlohy. V této chvíli je ovšem duální úlohou úloha (3.21) s přípustným řešením $\mathbf{x} = (0,0,0)^{\mathrm{T}}$ a primární je duální úloha s účelovou funkcí z (3.23). Tudíž pro přípustná řešení obou úloh **x** a **y** podle slabé věty o dualitě platí nerovnost $-\boldsymbol{b}^{\mathrm{T}}\mathbf{y} \leq -\boldsymbol{c}^{\mathrm{T}}\mathbf{x}$; avšak díky nepřípustnosti **y** bude na začátku

$$-\boldsymbol{b}^{\mathrm{T}}\mathbf{y} = -4y_1 - 5y_2 - 7y_3 > -3x_1 - 2x_2 - 4x_3 = -\boldsymbol{c}^{\mathrm{T}}\mathbf{x} \tag{3.24}$$

Vzhledem k tomu, co bylo řečeno v odd. 2.3, bude duální algoritmus procházet duálně přípustná řešení, jimž odpovídají nepřípustná primární bazická řešení, dokud není nalezeno řešení přípustné a v (3.24) není dosaženo rovnosti. Pak podle silné věty o dualitě samozřejmě algoritmus skončil nalezením optimálního řešení. Odtud plyne, že v poslední optimální tabulce duálního algoritmu bude v kriteriálním řádku a v posledním sloupci optimální hodnota účelové funkce **s opačným znaménkem**, poněvadž algoritmus snižuje v (3.24) levou stranu

a současně zvyšuje pravou, což je třeba mít na zřeteli. Duální simplexový algoritmus poskytne následující sekvenci simplexových tabulek:

Tab. 15. Duální simplexový algoritmus, řešící úlohu (3.23)

y_1	y_2	y_3	y_4	y_5	y_6	
−4	−5	−7	0	0	0	**min**
−1	−2	−2	1	0	0	−3
−1	0	−1	0	1	0	−2
−2	−1	−3	0	0	1	**−4**
4	5	7	0	0	0	0
0	$-\frac{3}{2}$	$-\mathbf{\frac{1}{2}}$	1	0	$-\frac{1}{2}$	**−1**
0	$\frac{1}{2}$	$\frac{1}{2}$	0	1	$-\frac{1}{2}$	0
1	$\frac{1}{2}$	$\frac{3}{2}$	0	0	$-\frac{1}{2}$	2
0	3	1	0	0	2	−8
0	3	1	−2	0	1	2
0	**−1**	0	1	1	−1	**−1**
1	−4	0	3	0	−2	−1
0	0	0	2	0	1	−10
0	0	1	1	3	**−2**	**−1**
0	1	0	−1	−1	1	1
1	0	0	−1	−4	2	3
0	0	0	2	0	1	−10
0	0	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{3}{2}$	1	$\frac{1}{2}$
0	1	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
1	0	1	0	−1	0	2
0	0	$\frac{1}{2}$	$\frac{5}{2}$	$\frac{3}{2}$	0	$-\frac{21}{2}$

V jednotlivých tabulkách jsou zvýrazněny v pravých sloupcích prvky, které jsou vybrány podle pravidla (2.15), a následně v příslušných řádcích klíčové prvky podle pravidla (2.16), samozřejmě kromě poslední tabulky, jež obsahuje optimální řešení obou úloh (3.22) a (3.20).

Úloha (3.20) byla degenerovaná, což se projevilo i degenerovaností duální úlohy (3.22). Ve třetí tabulce byly výběry podle pravidel (2.15) a (2.16) nejednoznačné a následně ve čtvrté tabulce nedošlo ke snížení hodnoty účelové funkce. Poslední optimální tabulka již vyjadřuje, že bylo dosaženo ekvilibria: $-\boldsymbol{b}^{\mathrm{T}}\tilde{\mathbf{y}} = -\frac{21}{2} = -\boldsymbol{c}^{\mathrm{T}}\tilde{\mathbf{x}}$. Přitom v kriteriálním řádku nalézáme optimální řešení duální úlohy (původně de facto primární) $\tilde{\mathbf{x}} = \left(\frac{5}{2}, \frac{3}{2}, 0\right)^{\mathrm{T}}$. Bazické řešení $\tilde{\mathbf{y}} = \left(2, \frac{1}{2}, 0\right)^{\mathrm{T}}$ potom řeší původní duální úlohu. Duální algoritmus pochopitelně musel najít táž řešení jako algoritmus primární.

II. PRAKTICKÁ ČÁST

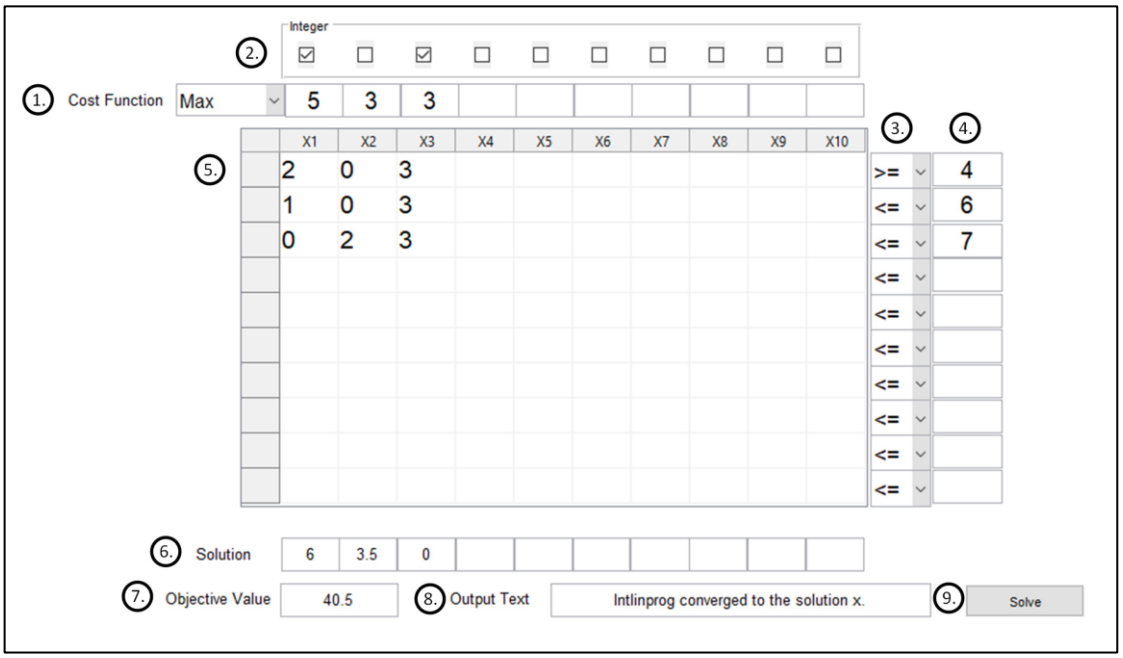
4 PROGRAMOVÁ PODPORA TEORETICKÉ ČÁSTI

4.1 Popis programu a jeho uživatelského prostředí

Úkolem praktické části této práce bylo vytvoření a odladění výukového programu pro řešení všech typů celočíselných i neceločíselných úloh LP ve tvaru nerovností, které byly popsány v teoretické části. Pracovně je nazván ***LinSolver***. Program je vypracován v prostředí MATLAB*u* R2016a jako Guide Application, který využívá vlastní funkci *intlinprog* pro řešení smíšených úloh celočíselného lineárního programování.

Program je koncipován k maximální jednoduchosti jeho použití, uživatele plně odstiňuje od znalosti prostředí MATLAB*u* a vzhled uživatelského prostředí vychází z obecného zápisu simplexové tabulky, což napomáhá intuitivnímu ovládání celého programu.

Program zvládá řešit všechny standardní úlohy LP, jak ve tvaru minimalizace, tak i maximalizace, též je schopen řešit celočíselné úlohy, kde je rovněž možné zadat požadavky na celočíselnost pouze pro konkrétní podmnožinu proměnných. Jeho limitem je počet proměnných zadané úlohy, která může obsahovat maximálně 10 proměnných. Program je určen pro výukové účely. Nikde v literatuře jsme nezaznamenali žádnou cvičnou úlohu s větším počtem proměnných než 10.



Obr. 6. Vzhled uživatelského prostředí programu

Rozhraní programu obsahuje 9 základních prvků, které lze vidět na obr. 6:

1. **Řádek účelové funkce** – zde volíme typ optimalizace (min/max) a zadáváme jednotlivé koeficienty účelové funkce.
2. **Požadavky celočíselnosti proměnných –** řádek checkboxů pro zavedení požadavků celočíselnosti jednotlivých proměnných.
3. **Nerovnosti omezujících podmínek –** umožňuje zadat směry nerovností omezení.
4. **Vektor omezení –** pole pro zadání jednotlivých prvků vektoru pravých stran omezení.
5. **Matice úlohy –** tabulka pro zadání jednotlivých prvků matice omezení.
6. **Hodnoty jednotlivých proměnných vektoru optimálního řešení –** výpis nalezených hodnot složek optimálního řešení.
7. **Optimální hodnota účelové funkce –** výpis nalezené optimální hodnoty účelové funkce.
8. **Výpis stavových hlášení –** tato hlášení vrací samotná funkce *intlinprog*.
9. **Tlačítko Solve –** tlačítko, které spustí výpočet úlohy.

Z poměrně skoupých informací víme, že funkce *intlinprog* používá i metodu Gomoryho řezů i metodu větví, včetně dalších metod, jako např. Chvátalovy-Gomoryho řezy. O tom, která metoda bude použita, rozhoduje vnitřní rozhodovací proces funkce *intlinprog* na základě tvaru úlohy. Proto ji lze použít bez problémů také na smíšené úlohy.

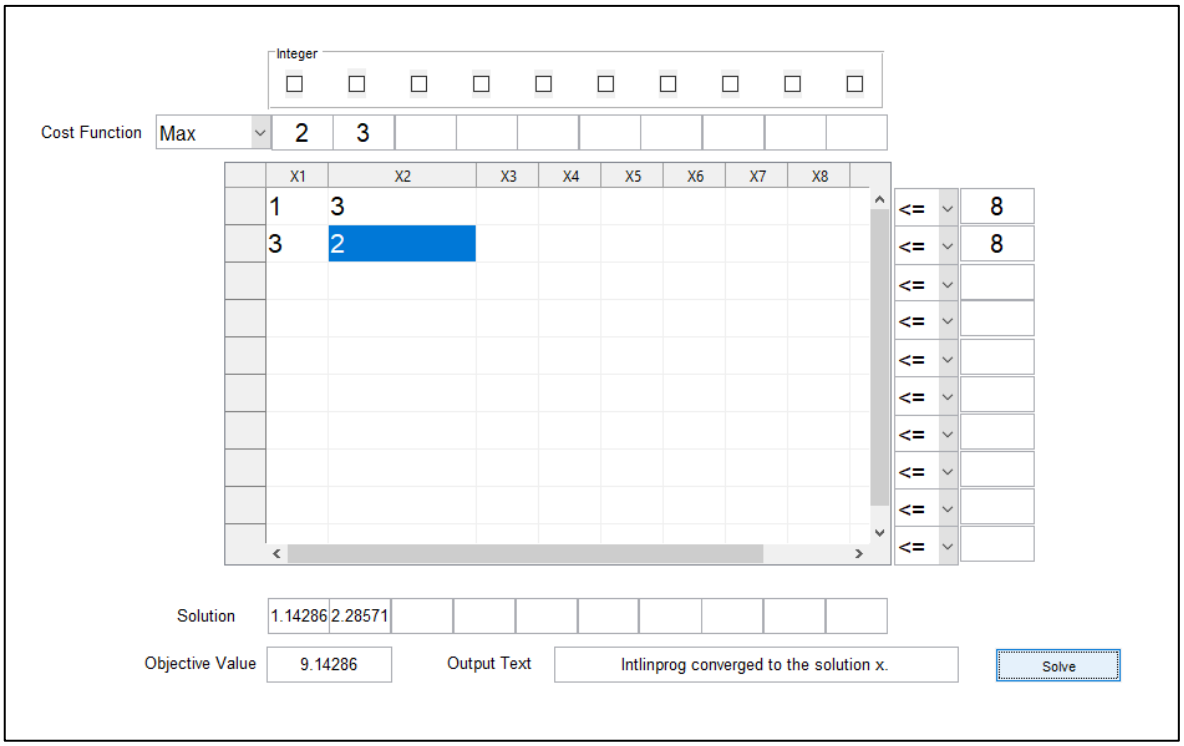
Upozornění. Instalace programu LinSolver může u slabších počítačů trvat i několik minut, protože je třeba nainstalovat příslušné části MATLAB*u*. Po jeho spuštění může trvat několik desítek vteřin, než bude program aktivní. Poté ovšem LinSolver vrací již optimální řešení úlohy prakticky okamžitě. Instalace programu byla odzkoušena pro OS **Windows 10**.

4.2 Ukázky řešení úloh z teoretické části

V tomto oddílu vyřešíme vybrané úlohy z teoretické části vyvinutým programem LinSolver a řešení porovnáme s řešeními, která jsme získali manuálními výpočty.

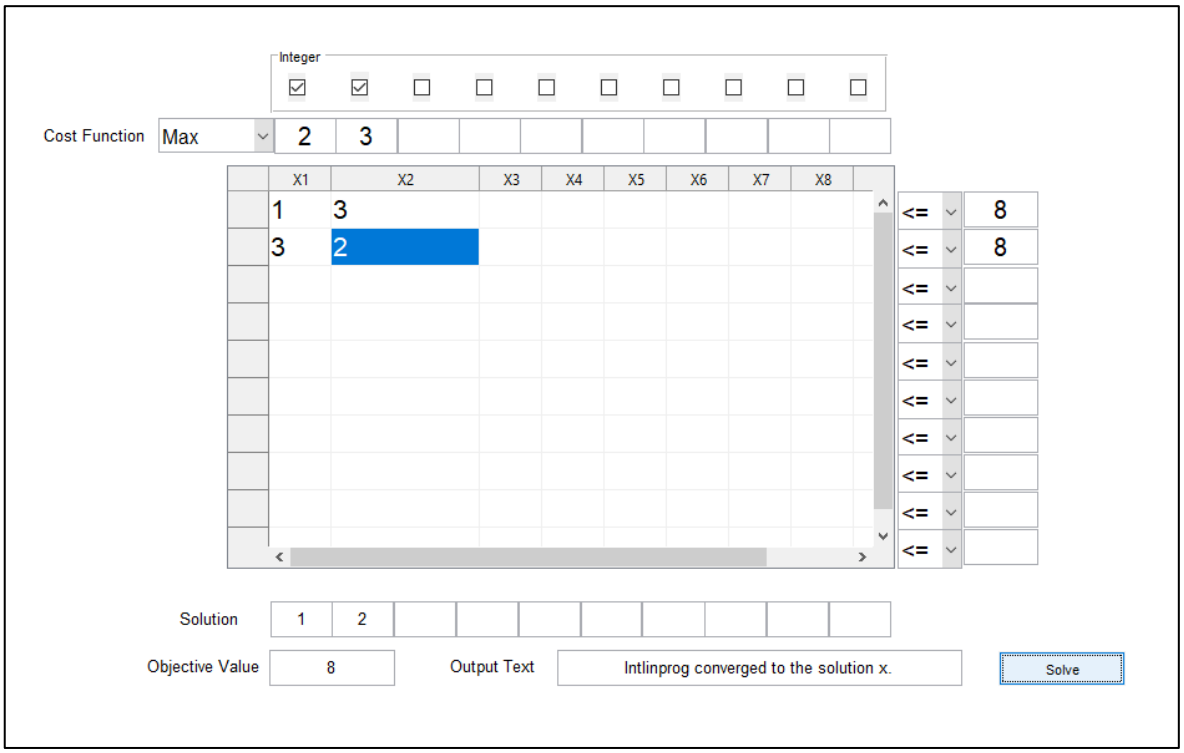
V souvislosti s degenerací poukážeme na jeden limit funkce *intlinprog*, ukážeme totiž, že funkce nemá systematickou ochranu před zacyklením. Program MATLAB v tomto případě nejen nenalezne správné řešení, avšak navíc podá i zcela zavádějící chybné hlášení. Nejsme si jisti, zda je to správná vizitka pro program, který je určen i pro vědecko-výzkumné účely.

Řešení úlohy (1.12) na str. 17, resp. (2.5) na str. 34 ukazuje obr. 7 na následující stránce.



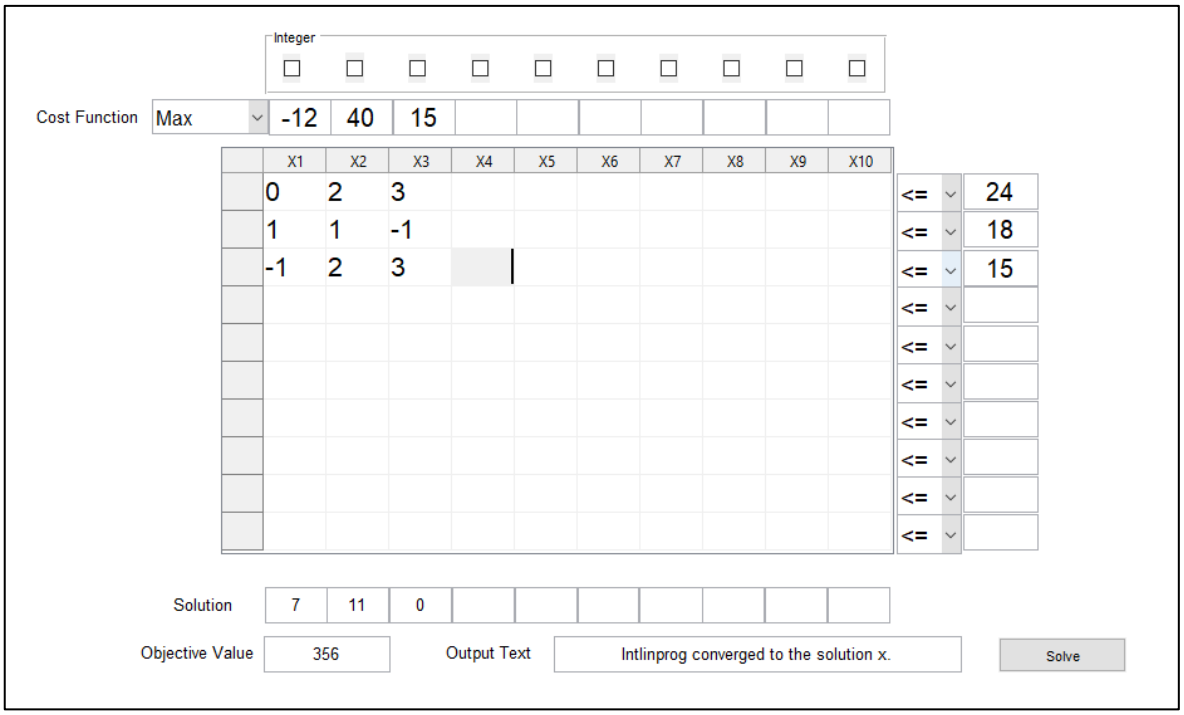
Obr. 7. Řešení úlohy (1.12)

Zde $\frac{8}{7} \approx 1{,}14286$ a $\frac{16}{7} \approx 2{,}28571$. Řešení úlohy (1.12) při dodatečném požadavku celočíselnosti obou proměnných je zobrazeno na obr. 8.



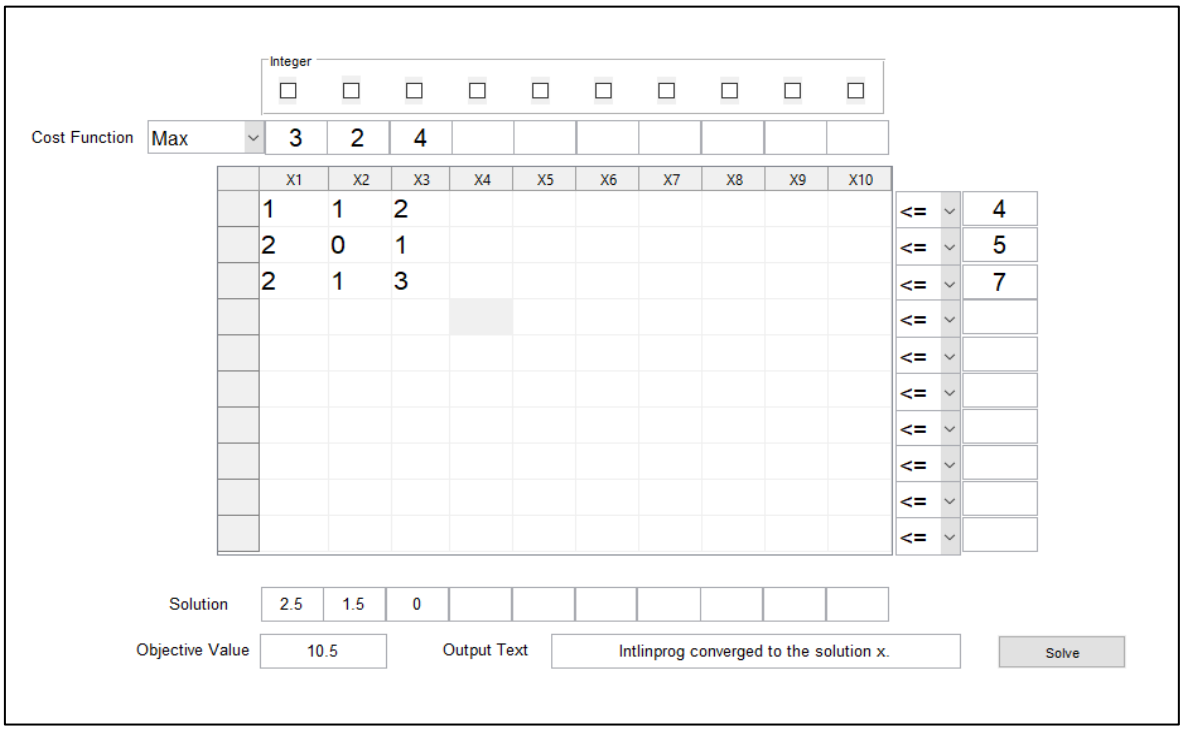
Obr. 8. Řešení úlohy (1.12) s dodatečným požadavkem celočíselnosti

Řešení úlohy (2.7) na str. 35 ukazuje obr. 9.



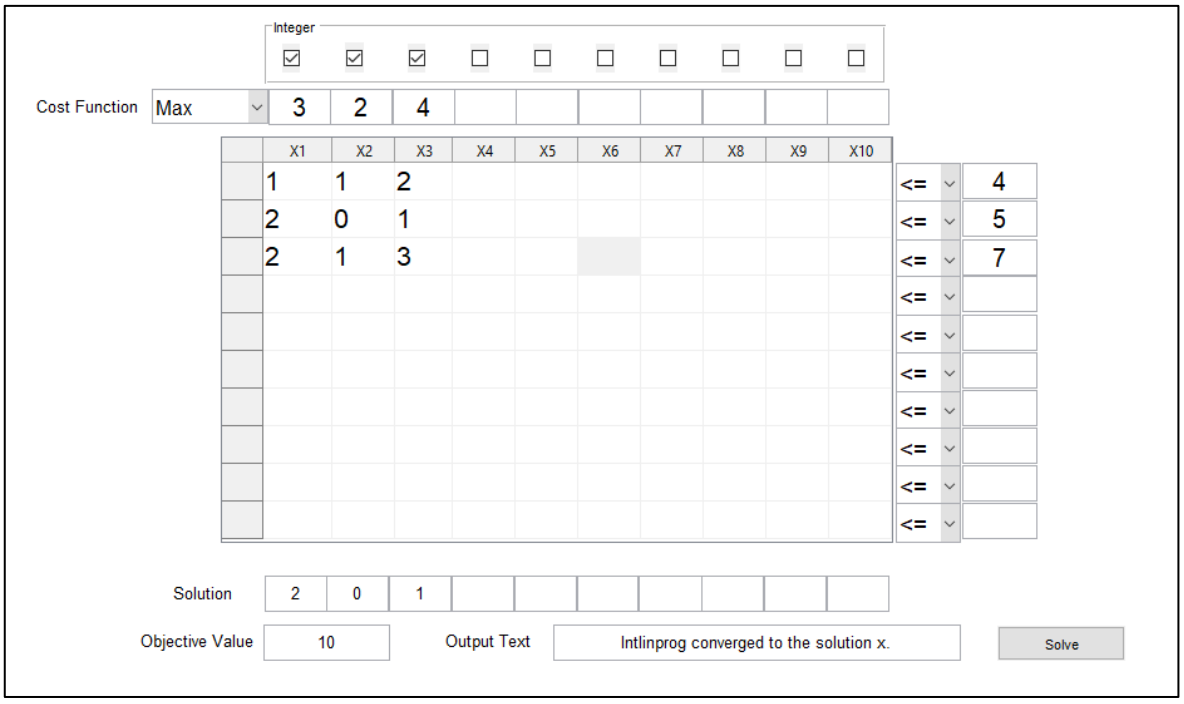
Obr. 9. Řešení úlohy (2.7)

Řešení úlohy (2.9) na str. 37 ukazuje obr. 10.



Obr. 10. Řešení úlohy (2.9)

Řešení úlohy (2.9) při požadavku celočíselnosti všech proměnných ukazuje obr. 10.



Obr. 11. Řešení úlohy (2.9) s dodatečným požadavkem celočíselnosti

Řešení relaxace celočíselné úlohy (3.8) na str. 53 ukazuje obr. 12.

Obr. 12. Řešení relaxace úlohy (3.8)

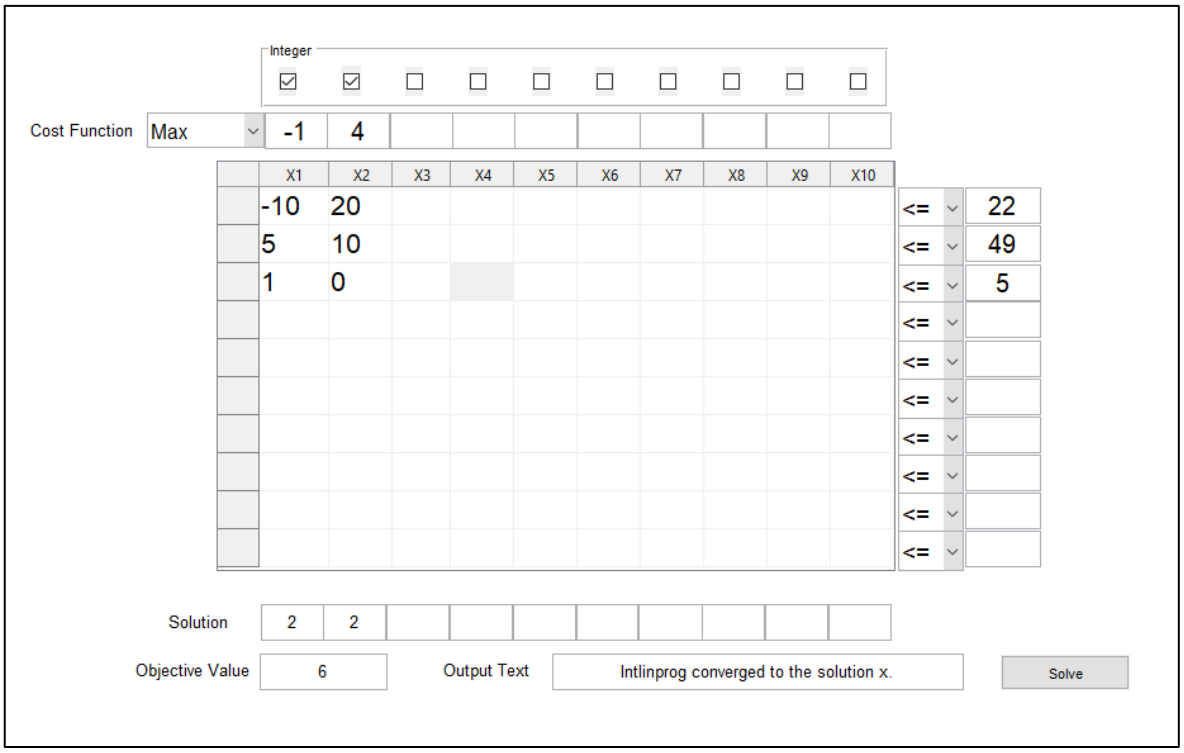
Řešení úlohy (3.8) při požadavku celočíselnosti obou proměnných ukazuje obr. 13.

Obr. 13. Řešení celočíselné úlohy (3.8)

Řešení celočíselné úlohy (3.15) na str. 59 ukazuje obr. 14.

Obr. 14. Řešení celočíselné úlohy (3.15)

Řešení celočíselné úlohy (3.19) na str. 64 ukazuje obr. 15.



Obr. 15. Řešení celočíselné úlohy (3.19)

Jak je okamžitě vidět z uvedených příkladů, program LinSolver pochopitelně nemohl ani v jediném případě dospět k jinému řešení, nežli které jsme získali již v teoretické části. Tím můžeme považovat program za **odladěný a plně funkční** (ovšem jak dále uvidíme, jedině v rámci schopností, které poskytuje samotný MATLAB).

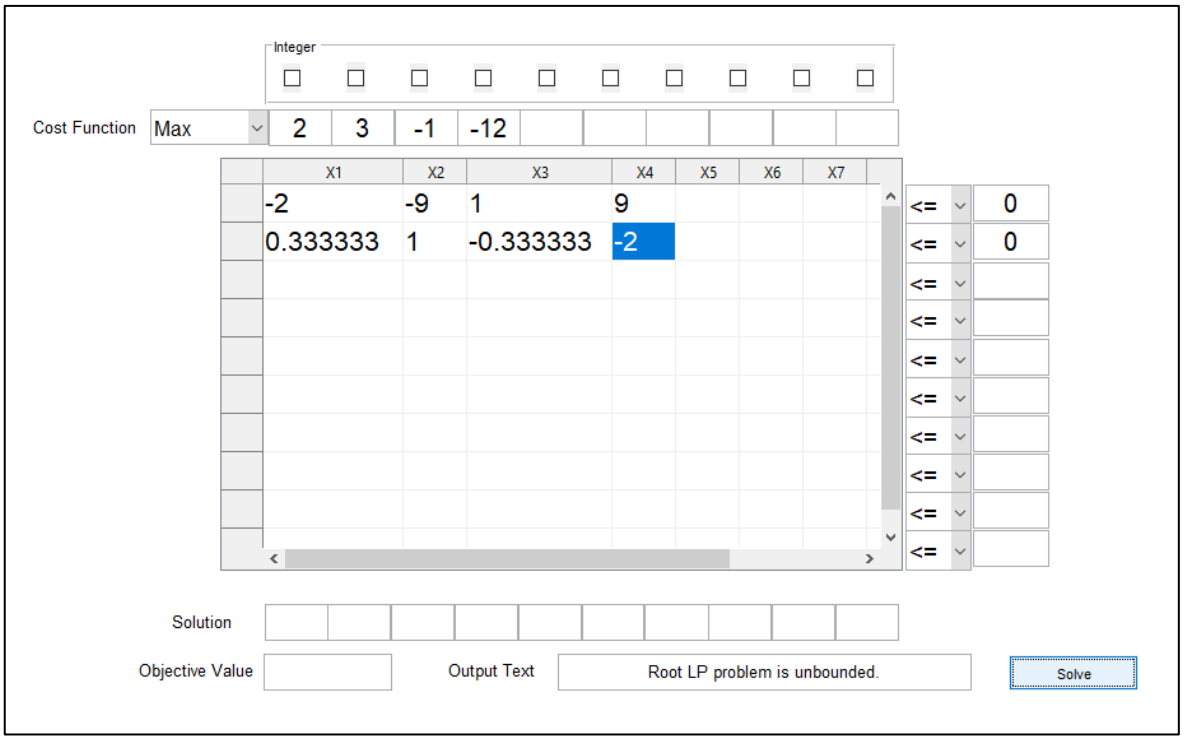
Jak je vidět z obr. 10, program vyřešil i úlohu (2.9), která je degenerovaná. Stejně úspěšně vyřešil i všechny další degenerované úlohy, které jsou v literatuře určeny k demonstraci zacyklení. K tomu ovšem vždy postačilo programové ošetření nejednoznačnosti ve výběru klíčového sloupce či řádku, protože tyto úlohy mají obvykle nejednoznačnost pouze ve výběru řádků, anebo jen sloupců. Potom postačí brát nejmenší index řádku, či sloupce, a toto ošetření bude fungovat stejně jako metoda nejmenších indexů, ačkoli systematická ochrana proti zacyklení bude chybět.

Dosud nejjednodušší úloha se zacyklením primárního simplexového algoritmu pochází od Alberta Williama Tuckera (téhož, jenž je znám Kuhnovou-Tuckerovou větou), ačkoliv za tu cenu, že je totálně degenerovaná, tj. jejím optimálním řešením je nulový vektor $\tilde{\mathbf{x}} = \mathbf{0}$, přičemž $\tilde{z} = 0$. Z tohoto důvodu ovšem nemá vlastnost, kterou jsme popsali v předchozím odstavci. Proto se jedná o jednu z mála známých úloh (tedy kromě variací na totéž téma)

malých rozměrů, které dokáží spolehlivě odhalit, je-li implementace simplexového algoritmu ošetřena proti smyčkám. Tuckerova úloha má délku cyklu 6 a je následující:

$$\begin{aligned} &\max\ (2x_1 + 3x_2 - x_3 - 12x_4) \\ &-2x_1 - 9x_2 \ + x_3 + 9x_4 \leq 0 \\ &\tfrac{1}{3}x_1 \ + x_2 - \tfrac{1}{3}x_3 - 2x_4 \leq 0 \\ &x_{1,2,3,4} \geq 0 \end{aligned} \tag{4.1}$$

Řešení úlohy (4.1) pomocí LinSolver*u* je na následujícím obrázku.



Obr. 16. Řešení Tuckerovy úlohy – ukázka zacyklení

Z obr. 16 je zřejmé, že funkce *intlinprog* interpretuje zacyklení **zcela mylně** jako neomezenost účelové funkce. Z toho lze dedukovat, že tato funkce nekontroluje aktuální hodnotu účelové funkce a zastavuje simplexový algoritmus nestandardně jen na základě počtu iterací. To však může učinit patrně jen na základě semiemprického odhadu. Pro naši úlohu je sice maximální počet iterací shora odhadnut přesně počtem $\binom{m+n}{n} = \binom{2+4}{4} = \binom{6}{4} = 15$, ovšem obecně může být tento počet pro hardware nezobrazitelný. Funkce *intlinprog* tudíž není proti zacyklení systémově ošetřena.

Závěrem uvedeme několik poznámek k hlášením, která může vrátit funkce *intlinprog*. Předně může ohlásit Intlinprog converged to the solution x. Lze očekávat, že tento výpis znamená, že simplexový algoritmus je ukončen standardně a nalezené řešení je vskutku optimální. Hlášení můžeme v tomto případě zřejmě považovat za zcela důvěryhodné.

Dalším možným je hlášení No feasible point found., což pochopitelně znamená, že je množina přípustných řešení prázdná. Z hlubší teorie LP (která nemohla být v první části této práce vyložena) vyplývá, že simplexová metoda umí vždy spolehlivě identifikovat, zda má úloha LP přípustná řešení, a to aniž by předtím bylo nutno jakkoliv algebraicky manipulovat s omezeními. Proto se lze domnívat, že i toto hlášení lze považovat za důvěryhodné.

Třetím možným hlášením je Root LP problem is unbounded. Nicméně obr. 16 v tomto případě ukazuje, že právě toto hlášení za zcela důvěryhodné považovat nelze. Může v ojedinělých případech totiž znamenat pravý opak. Lze tedy pouze konstatovat, že u běžných úloh bude toto hlášení pravděpodobně správné.

Poslední tři možná hlášení jsou u cvičných úloh velmi nepravděpodobná, avšak mohou se vyskytnout u rozsáhlejších praktických úloh, anebo také u úloh, v nichž jsou koeficienty, příp. celočíselnost proměnných voleny více či méně náhodně (např. při „experimentování“ s programem). Prvním je Intlinprog stopped prematurely. Integer feasible point found. Toto hlášení tvrdí, že výchozí relaxovaná úloha má neomezenou množinu přípustných řešení, jejíž průnik s celočíselným řešetem je nekonečný. Dalším je Intlinprog stopped prematurely. No integer feasible point found. V tomto případě je buď množina přípustných řešení prázdná, anebo je sice neprázdná, ale neobsahuje žádné přípustné celočíselné řešení.

Posledním je hlášení Intlinprog stopped by an output function or plot function. Z dokumentace je pouze zřejmé, že se jedná o softwarové hlášení. V ní je uvedeno výslovné varování, že význam pojmů „output function“ a „plot function“ je u *intlinprog* jiný, než je obvyklé, a že tyto funkce monitorují hodnoty v jednotlivých iteracích algoritmů. Význam tohoto hlášení není tedy zcela jasný. Lze se však domnívat, že pravděpodobně znamená tu skutečnost, že hardware by nebyl schopen úlohu dořešit.

ZÁVĚR

V úvodu se snažíme rozebrat o maličko více historii LP, než je pouhé konstatování, že jeho zakladatelem byl G. B. Dantzig a že v souvislosti s ním byla udělena Nobelova cena. Pokud autoři (prací jako je tato) historii zmiňují, mnohdy tak činí příliš zploštěle. Také v dalším textu se snažíme alespoň o minimální komentář k tvůrcům jednotlivých algoritmů.

V první kapitole teoretické části jsou shrnuty v nejnutnějším minimu tvrzení teorie LP, které jsou nezbytné k pochopení funkce univerzálního nástroje pro řešení úloh LP – simplexové metody. Jsou v ní rovněž shrnuty nejběžnější typy úloh, na jejichž základě lze formulovat obecný tvar úlohy LP. Kapitola je zakončena nezbytným úvodem do teorie duality, a to mj. jako účinného nástroje pro kontrolu správnosti řešení úloh LP, příp. rovněž proto, že řešení duální úlohy může být někdy vhodnější.

Druhá kapitola je věnována algoritmickému řešení velmi frekventované maximalizační úlohy s omezeními ve tvaru nerovností. Velké množství aplikačních problémů je přímo v jejím tvaru, anebo je lze s minimálním úsilím na tuto úlohu převést. Pro její řešení bez požadavku celočíselnosti jsou plně popsány dva algoritmy: primární a duální simplexový algoritmus. U algoritmů je vždy věnována pozornost jejich konečnosti a situacím, kdy by mohlo vlivem nejednoznačnosti, zejména vlivem degenerace úlohy, dojít k jejich zacyklení. Rovněž vždy uvádíme interpretaci situace, kdy je algoritmus ukončen nestandardně a jaké to má důsledky pro řešitelnost výchozí úlohy. Pozornost věnujeme možnostem ošetření patologických situací, proti nimž nemají obvykle komerční programy žádné pojistky. Oba algoritmy poskytují obecně neceločíselná řešení relaxované úlohy bez požadavku na jejich celočíselnost, z nichž se inicializují následně popsané algoritmy pro řešení celočíselných úloh.

Ve třetí kapitole se věnujeme dvěma základním obecným metodám celočíselného LP: metodě Gomoryho řezů a metodě větví a mezí. Jsou plně popsány oba jim příslušné algoritmy. Věnujeme pozornost předpokladům jejich použitelnosti, konečnosti a interpretaci situace, kdy musí být algoritmus ukončen nestandardně a jaký význam má tato skutečnost pro řešitelnost celočíselné úlohy. V rámci dostupných informací je Gomoryho algoritmus porovnán s algoritmem větví a mezí z hlediska jejich základních parametrů: rychlosti konvergence, operačních a paměťových nároků. Na konci kapitoly jsou velmi stručně zmíněny některé další metody celočíselného LP.

Všechny algoritmy v teoretické části jsou sice popsány neformálně, ale takovým způsobem, aby byly programovatelné a současně mohly být ošetřeny všechny nestandardní situ-

ace, které by mohly nastat, a následně mohla být podána příslušná hlášení: množina přípustných řešení je prázdná, účelová funkce je na množině přípustných řešení neomezená, optimální řešení neexistuje apod.

V závěru třetí kapitoly pro úplnost krátce zmiňujeme duální simplexový algoritmus aplikovaný na řešení celé úlohy, neboť v předchozím textu je tento algoritmus vždy používán výhradně jako procedura pomocná.

Čtvrtá kapitola, která tvoří praktickou část práce, je věnována programu ***LinSolver***, který je koncipován jako podpůrný program pro výuku lineárního programování. Proto má velmi jednoduché uživatelské prostředí. Je schopen řešit jakoukoli úlohu lineárního programování ve tvaru nerovností maximálních rozměrů 10×10, a to jak neceločíselnou, tak také celočíselnou či smíšenou. Využívá přitom funkci *intlinprog* MATLAB*u*, která je založena především na algoritmech popsaných v teoretické části.

Program LinSolver byl odladěn nejen na základě úloh řešených v teoretické části, avšak rovněž mnoha dalších úloh, které nabízí použitá literatura a u nichž jsou známa správná řešení. Ve všech případech program vrátil stejná řešení. Proto můžeme konstatovat, že je LinSolver plně funkční. Nicméně, jak jsme ukázali na konci praktické části, toto konstatování je omezeno možnostmi samotného MATLAB*u*, který je zřejmě schopen vracet i hlášení či řešení zcela scestná. Pozoruhodná je v této souvislosti tedy otázka, proč v programu, který dnes již zdaleka není určen jen k pedagogickým účelům, není ošetřeno hlídání aktuální hodnoty účelové funkce, neboť tato aktivita by algoritmus poznatelně nezpomalila.

SEZNAM POUŽITÉ LITERATURY

[1] DANTZIG, George B.. *Lineárne programovanie a jeho rozvoj*. 1. vyd. Bratislava: SVTL, 1966. 704 s.

[2] DEMEL, Jiří. *Operační výzkum* [online]. 2017 [cit. 2018-03-08], učební text ČVUT. Dostupné na: https:// kix.fsv.cvut.cz/~demel/ped/ov/ov.pdf

[3] DUPAČOVÁ, Jitka, LACHOUT, Petr. *Úvod do optimalizace.* 1.vyd. Praha: Mat-fyzpress, 2011. 88s. ISBN 978-7378-176-7

[4] GRYGAROVÁ, Libuše. *Úvod do lineárního programování*, skripta. 1. vyd. Praha: Státní pedagogické nakladatelství, 1975

[5] HLADÍK, Milan. *Celočíselné programování.* 2017 [cit. 2018-03-09], text k přednášce na MFF UK. Dostupné na: https://kam.mff.cuni.cz/~hladik/CP/text_cp.pdf

[6] JABLONSKÝ, Josef. *Operační výzkum: kvantitativní modely pro ekonomické rozhodování*. 1. vyd. Praha: Professional publishing, 2002. 323 s. ISBN 80-86419-23-1

[7] KUBIŠOVÁ, Andrea. *Operační výzkum*. 2014 [cit. 2018-03-10], učební text VŠPJ. Dostupné na: https://www.vspj.cz/ISBN/Skripta%20%20V%C5%A0PJ/Opera%C4%8Dn%C3%AD%20v%C3%BDzkum%20-%20Kubi%C5%A1ov%C3%A1%20Andrea.pdf

[8] LACHOUT, Petr. *Matematické programování*. 2011 [cit. 2018-03-11], pracovní text k přednášce na MFF UK. Dostupné na: http://www.karlin.mff.cuni.cz/~lachout/Vyuka/ TEXTY/111016-MP_skripta.pdf

[9] LINDA, Bohdan, VOLEK, Josef. *Lineární programování*. 3. vyd. Pardubice: Unverzita Pardubice, 2009, 139 s. ISBN 978-80-7395-207-5

[10] LUENBERGER, David G., YE, Yinyu. *Linear and nonlinear programming*. 3rd ed. New York: Springer, 2008. 546 s. ISBN 978-0387-74502-2

[11] MAŇAS, *Optimalizační metody*. 1. vyd. Praha: TKI SNTL, 1979. 260 s.

[12] PLESNÍK, Ján, DUPAČOVÁ, Jitka, VLACH, Milan: *Lineárne programovanie*. 1. vyd. Bratislava: Alfa, 1990. 320 s. ISBN 80-05-00679-9

[13] PEKAŘ, Libor. *Optimalizace*, skripta. FAI, UTB, Zlín 2013. 80 s.

[14] PROKOP, Roman. *Lineární programování*. FAI, UTB, Zlín. 26 s.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

$\mathbb{R}$	množina reálných čísel
$\mathbb{Q}$	množina racionálních čísel
$\mathbb{Z}$	množina celých čísel
$\mathbb{R}^n$	množina sloupcových n-rozměrných vektorů s reálnými prvky
$\mathbb{Q}^n$	množina sloupcových n-rozměrných vektorů s racionálními prvky
$\mathbb{Z}^n$	množina sloupcových n-rozměrných vektorů s celočíselnými prvky
$\mathbb{R}^{n\times m}$	množina matic typu $n \times m$ s reálnými prvky
$\mathbb{Q}^{n\times m}$	množina matic typu $n \times m$ s racionálními prvky
$\mathbb{Z}^{n\times m}$	množina matic typu $n \times m$ s celočíselnými prvky
$\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, ...	pevně zvolené vektory (příslušných rozměrů)
A, B, C, ...	pevně zvolené matice (příslušných typů)
$\boldsymbol{a}^{\mathrm{T}}$	řádkový vektor transponovaný ke sloupcovému vektoru $\boldsymbol{a}$
A^{T}	transponovaná matice k matici A
$\mathbf{x}$, $\mathbf{y}$, ...	proměnné vektory
$\widetilde{\mathrm{A}}$	matice omezení kanonické úlohy
$\tilde{\mathbf{x}}$	přípustné řešení kanonické úlohy
$\tilde{\mathbf{x}}$, $\tilde{\mathbf{y}}$	optimální řešení primární, resp. duální úlohy
$\mathbf{x}^*$	optimální řešení kanonické úlohy
$\mathbf{0}$	nulový sloupcový vektor (příslušného rozměru)
z	hodnota účelové funkce, příp. samotná účelová funkce
$\tilde{z}$	optimální hodnota účelové funkce
LP	lineární programování (linear programming)

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Instalační DVD s výukovým programem LinSolver, který je určen k řešení cvičných úloh lineárního programování (instalace odzkoušena pro OS **Windows 10**).