Aplikace metod lineárního programováni

Nikolas Remeš

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Nikolas Remeš**
Osobní číslo:	**A20316**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Aplikace metod lineárního programování**
Téma práce anglicky:	**Application of Linear Programming Methods**

Zásady pro vypracování

1. Proveďte formulaci a klasifikaci úloh lineárního programování.
2. Nastudujte metody jedno a dvoufázové simplexové metody.
3. Uveďte programovou realizaci metody simplexové tabulky.
4. Napište studijní oporu pro tuto problematiku.
5. Uveďte charakteristické úlohy pro jednotlivé metody.

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

Seznam doporučené literatury:

1. JABLONSKÝ, Josef. *Operační výzkum: kvantitativní modely pro ekonomické rozhodování*. 1. vyd. Praha: Professional-publishing, 2002. 323s. ISBN 80-86419-23-1.
2. LINDA, Bohdan, VOLEK, Josef. *Lineární programování*. 3. vyd. Pardubice: Univerzita Pardubice, 2009, 139 s. ISBN 978--80-7395-207-5.
3. LUENBERGER, David G., YE, Yinyu. *Linear and nonlinear programming*. 3rd ed. New York: Springer, 2008. 546 s. ISBN 978-0387-74502-2.
4. MAŇAS, *Optimalizační metody*. 1. vyd. Praha: TKI SNTL, 1979. 260 s
5. PLESNÍK, Ján, DUPAČOVÁ, Jitka, VLACH, Milan: *Lineárne programovanie*. 1. vyd. Bratislava: Alfa, 1990. 320 s. ISBN 80-05-00679-9.
6. PEKAŘ, Libor. *Optimalizace*, skripta. FAI, UTB, Zlín 2013. 80 s.
7. PROKOP, Roman. *Lineární programování*. FAI, UTB, Zlín. 26 s.

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

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

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

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním 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 bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji 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 – 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 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 bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem 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 bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13. 5. 2024 Nikolas Remeš, v. r.

ABSTRAKT

Tato práce se zabývá aplikací metod lineárního programování s cílem optimalizace různých procesů a rozhodovacích problémů. První část práce představuje teoretické základy lineárního programování a jeho aplikace v průmyslu, logistice a dalších odvětvích. Následně je provedena analýza konkrétních případů, ve kterých jsou tyto metody úspěšně využívány. Druhá část práce se zaměřuje na návrh a implementaci aplikace, která umožňuje uživatelům efektivně využívat lineární programování pro řešení konkrétních úkolů. Výsledky ukazují významné zlepšení v efektivitě a optimalizaci rozhodovacích procesů.

Klíčová slova: lineární programování, formulace úlohy, aplikace v praxi, Simplexova metoda, dualita v lineárním programování, optimalizace, průmysl.

ABSTRACT

This thesis explores the application of linear programming methods with the aim of optimizing various processes and decision problems. The first part introduces the theoretical foundations of linear programming and its applications in industries, logistics, and other fields. Subsequently, an analysis of specific cases where these methods are successfully utilized is conducted. The second part focuses on the design and implementation of an application that enables users to efficiently leverage linear programming to solve specific tasks. The results demonstrate significant improvements in the efficiency and optimization of decision-making processes.

Keywords: linear programming, issue formulation, practical application, Simplex method, duality in linear programming, optimization, industry.

Chtěl bych poděkovat a jsem velice vděčný v první řadě mému vedoucímu práce panu prof. Ing. Romanovi Prokopovi, CSc., že měl se mnou trpělivost a byl ochoten se mi věnovat. Dále chci poděkovat mé rodině, která byla vždy při mně, podporovala mě a vyšla mi ve všem vstříc. V neposlední řadě také mým kamarádům, příbuzným atd.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato bakalářská práce je určena pro studenty Fakulty aplikované informatiky, kteří se během studia s lineárním programováním potkávají.

V této práci byl kladen důraz na stručnou teorii a programovou realizaci

Práce je stavěna tak, aby bylo srozumitelná pro čtenáře, který s tímto tématem nepřichází do styku denně, a je schopen pochopil tuto matematickou disciplínu.

V první části bakalářské práce se budu se věnovat převážně teorii. Definici lineárního programování, historii, důležitým základním pojmům a klasifikaci úloh lineárního programování.

Ve druhé části chci seznámit čtenáře s formulací úloh a metodami LP a příkladnými ukázkami využití lineárního programování.

V další části se budu zabývat metodou simplexové tabulky a citlivostní analýzou.

Praktická část je věnována vývoji vlastního programu, který je schopen řešit úlohy lineárního programování, nazvaného „Simplex Solver“ a tvorbou studijních opor.

I. TEORETICKÁ ČÁST

1 LINEÁRNÍ PROGRAMOVÁNÍ

1.1 Definování lineárního programování

Lineární programování je matematická disciplína, která se zabývá řešením optimalizačních úloh v různých oblastech praktického života. Konkrétně může jít o řešení některého z níže uvedených reálných problémů: [1]

- Optimální výrobní program, jenž podniku může zajistit maximální peněžní zisk za předpokladu, že má omezené zásoby surovin,
- Směšovací problém, kdy podnik chce vyrobit směs za co nejmenší cenu s žádaným množstvím určitých složek,
- Dělení materiálu. Tedy jakým způsobem co nejefektivněji rozdělit různý materiál na menší části,
- Dopravní problém. Jak naplánovat distribuci zboží na prodejny, aby byly náklady na dopravu co nejmenší. [1]

Proto je nutné si před začátkem práce odpovědět na tyto počáteční otázky:

1. Co je cílem této optimalizace (čeho chceme dosáhnout),
2. Co jsou řídící veličiny (jak můžeme ovlivnit cíl),
3. Co nás omezuje (jaké jsou naše limity).

1.2 Historie lineárního programování

Lineární programování je relativně mladým odvětvím matematiky, jehož hlavní vývoj začal teprve během druhé světové války. První předchůdce však najdeme již v 18. století, a to hospodářskou tabulku francouzského ekonoma Francoise Quesnaye z roku 1758. V této tabulce se Quesnay rozhodl pomocí lineárního systému rovnic a omezení nerovností, které zajišťují nezápornost proměnných (tzv. omezení na znaménka), ukázat vzájemný vztah mezi pronajímateli, farmáři a řemeslníky. [2]

V knize *Mécanique analytique* publikované v roce 1788 se objevila jedna z metod, kterou lze použít k nalezení vázaného extrému, nazvaná jako metoda Lagrangeových multiplikátorů. Lagrangeova metoda transformuje problém na soustavu lineárních rovnic, během hledání extrémů lineárních funkcí omezených lineárními rovnostmi bez omezení

na znaménko, kterou lze řešit např. eliminací Gaussovou (objevila se v některých učebnicích koncem 18. století, nicméně pod jiným názvem). [3]

Dalším významným přínosem byla práce Fouriera z 20. let 19. století, kde autor představil algoritmus pro řešení soustav lineárních nerovnic. Jedním ze základů lineárního programování je také Farkasova teorie soustav lineárních nerovností z 19. a počátku 20. století. Kromě Farkase v této oblasti působili také Hermann Minkowski a Constantin Carathéodory. [4]

Pozoruhodné je také představení navrhovaného řešení dopravního problému, které jako první zformuloval francouzský matematik Gaspard Monge na konci 18. století. Toto řešení navrhl A. N. Tolstoy v r. 1930 a americký matematik Hitchcock v r. 1941 formuloval postup řešení obecného dopravního problému. Ve 30. letech se řešil i přiřazovací problém, který je spolu s dopravním problémem řazen mezi speciální problém lineárního programování. Hlavním úkolem přiřazovacího problému je co nejefektivněji spárovat dvě stejné velké skupiny. Můžeme si například představit množinu strojů a množinu zaměstnanců, ze kterých víme, jak dlouho trvá naučit se jednotlivé stroje ovládat. Pak je úkolem najít takové řešení, které minimalizuje celkový čas strávený školením zaměstnanců. Řešení přiřazovacího problému j založeno na kombinatorických úvahách, výsledný algoritmus je dnes kvůli národnosti autorů (König a Egerváry) znám jako Maďarská metoda. [5]

Další významný příspěvek s názvem Matematické metody v organizaci a plánování výroby (r. 1939) pochází od ruského matematika L. V. Kantoroviče. Kantorovič pak v r. 1975 obdržel spolu s ekonomem Koopmansem Nobelovu cenu za ekonomii za přínos k rozvoji lineárního programování a jeho ekonomických aplikací. [6]

Jedním z nejdůležitějších problému lineárního programování je bezesporu problém jídelníčku. Tento problém byl formulován Jerrym Cornfieldem během druhé světové války. Hlavní motivací bylo najít pro vojáky co nejlevnější, ale nutričně dostačující jídelníček. V roce 1947 byla s tímto problémem testována jednoduchá Dantzigova metoda, optimální řešení bylo nalezeno za 120 člověkodnů. Je jistě zajímavé, že Stiglerovo přibližné řešení se od optimálního řešení nalezeného simplexovou metodou lišilo pouze od 0.24 dolaru ročně v nákladech na potraviny. [7]

Dantzig, považovaný za zakladatele lineárního programování a jehož algoritmus zvaný simplexová metoda (1947) se stále používá k řešení problémů lineárního programování, sám

prohlásil, že sestavení algoritmu je výsledkem mnoha let diskusí s ekonomy jako T. Koopmansem nebo W. Leontiefem. [8]

S rozvojem výpočetní techniky v 50. a 60. letech 20. století dramaticky vzrostl výpočetní výkon a začala se také zvažovat účinnost a složitost algoritmů používaných k řešení úloh lineárního programování. Kvůli složitosti simplexové metody začaly vznikat jiné metody, sice výpočetně složitější, ale naopak efektivnější. Mezi ně patří např.: metoda elipsoidů metoda publikovaná v roce 1979 nebo metoda vnitřních bodů z roku 1984. [9]

1.3 Základní pojmy a vlastnosti lineárního programování

Tato část je čerpána ze zdroje [10].

V následující části práce jsou identifikovány základní pojmy lineárního programování a jeho vlastnosti.

Obecnou úlohu lineárního programování můžeme formulovat takto:

Maximalizujte (minimalizujte) za podmínek:

$$f(x) = c_1x_1 + c_2x_2 + \cdots + c_nx_n \textbf{ (1)}$$

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \textbf{ (2)}$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2$$

$$\dots$$

$$a_{m1}x_{12} + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m$$

$$x_1, x_2, \dots, x_n \geq 0 \textbf{ (3)}$$

- **(1)** je funkce, jejíž extrém (maximum či minimum) hledáme. Je pojmenovaná jako **účelová funkce**,
- **(2)** a **(3)** jsou omezující podmínky, kde podmínky **(2)** jsou **vlastní omezení** a podmínky **(3)** jsou **podmínky nezápornosti**. [11]

Řešit úlohu lineárního programování znamená vyhledat extrém, tzn. maximum nebo minimum, lineární funkce na množině dané soustavou lin. rovnic a nerovnic.

Standardní tvar úlohy je takový tvar, který sice ještě nebyl doplněn o pomocné proměnné, ale má vlastní omezení ve tvaru rovnic.

Kanonický tvar úlohy je takový tvar, kde celá soustava je doplněna o přídatné a pomocné proměnné, a jehož vlastní omezení jsou ve tvaru rovnic.

Přípustné řešení úlohy je takové řešení, které vyhovuje veškerým omezujícím podmínkám, tzn. omezením vlastním i podmínkám nezápornosti.

Přípustná množina úlohy (tj. množina všech řešení soustavy (2) a (3)) je množina všech přípustných řešení.

Optimální řešení úlohy je přípustné řešení, pro které je hodnota účelové funkce maximální či minimální. [11]

Optimální hodnota je maximální (nebo minimální) hodnota účelové funkce.

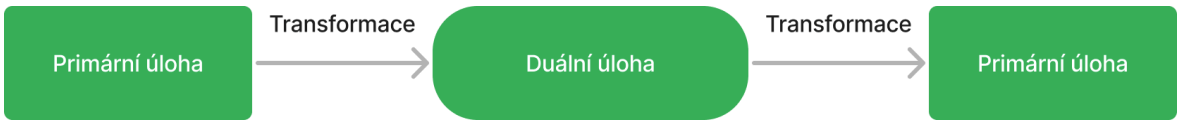
Konvexní množina je množina, v niž leží celá úsečka propojující dva body této množiny.

Konvexní polyedr čili mnohostěn je konvexní tvar omezen ohraničenou množinou.

1.4 Primární a duální úloha lineárního programování

V lineárním programování existuje pevný vzájemný vztah mezi minimalizačními a maximalizačními úlohami – **dualita**. Každou maximalizační úlohu je možné podle jasně daných pravidel transformovat na minimalizační a naopak. Úloha, která je dualitou transformována je **primární**, úloha převedená je **duální**. Tento vztah duality je symetrický v tom smyslu, že duální úloha k duální úloze musí poskytnout opět úlohu primární, přesně ve tvaru původním. [12]

Duální úloha nám umožňuje například kontrolu výsledků primární úlohy, dodatečnou analýzu, popř. řešit duální úlohu, pokud je to vhodnější z jistých důvodů. [12]



Obrázek 1: Znázornění vztahu primární a duální úlohy [zdroj: vlastní]

Primární úloha	→	**Duální úloha**
$\boldsymbol{c^T x \rightarrow max}$		$\boldsymbol{b^T y \rightarrow min}$
$\boldsymbol{Ax \leq b}$		$\boldsymbol{A^T y \geq c}$
$\boldsymbol{x \geq 0}$		$\boldsymbol{c \geq 0}$

Grafická ukázka převodu duálních úloh:

Primární úloha:

$$c^{\mathrm{T}}x \rightarrow max = 3x_1 + 2x_2 + x_3 \rightarrow max$$

při omezení:

$$\begin{aligned} 4x_1 + 3x_2 &\leq 1200 \\ 10x_1 + 5x_2 + x_3 &\leq 3500 \\ 3x_1 + 5x_2 + 6x_3 &\leq 5000 \end{aligned}$$

$$x_{1,2,3} \geq 0$$

Duální úloha:

$$b^{\mathrm{T}}y \rightarrow min = 1200x_{y1} + 3500y_2 + 5000y_3 \rightarrow max$$

při omezení:

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

$$y_{1,2,3} \geq 0$$

1.5 Klasifikace úloh lineárního programování

$$(x_i \geq 0) \qquad \begin{aligned} &\max c^T x \quad Ax \leq b, \\ &\min c^T x \quad Ax \leq b, \\ &\max c^T x \quad Ax \geq b, \\ &\min c^T x \quad Ax \geq b. \end{aligned}$$

Kombinovaná úloha LP

Bez ohledu na požadavek na účelovou funkci (maximum, minimum), mohou omezení být ve tvaru nerovností typu ≤, ≥, nebo ve tvaru rovnosti.

$$c^T x \quad x_i \geq 0.$$

Kombinovanou úlohou (smíšeným omezením) se tedy rozumí omezení:

$$a_{11}x_+ + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1,$$

$$a_{21}x_1 + a_{22}n_2 + \cdots + a_{2n}x_n \geq b_2,$$

$$a_{31}x_1 + a_{32}n_2 + \cdots + a_{3n}x_n = b_3.$$

Další modifikace

- nula-jedničkové LP – výsledek 0 nebo 1 (tedy ano nebo ne),
- celočíselné LP – výsledek v celých číslech.

2 FORMULACE ÚLOH LINEÁRNÍHO PROGRAMOVÁNÍ

Nalezněte hodnoty vektoru *x*, které

1. maximalizují lineární kritérium

$$c'x = c_1x_1 + c_2x_2 + \cdots + c_nx_n,$$

2. splňují vedlejší podmínky $Ax \leq b$, tedy

$$a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,n}x_n \leq b_i,$$

   pro $i = 1,2, \ldots, m$ (m podmínek pro n proměnných) a

3. jsou záporné. [13]

Zápis standardní úlohy je následující:

$$c^T x \rightarrow opt^{1}$$

$$Ax \leq b^{2}$$

$$x \geq 0^{3}$$

[1] hledání maxima či minima,

[2] nepřekročíme limitující omezení (např. nemůžeme vyrobit více produktů, než na které máme materiál) (≥ , ≤ , =),

[3] hodnota je nezáporná. [13]

Pojďme si uvést příklady úlohy, které lze pomocí lineárního programování řešit. Tyto úlohy ovšem nebudou obsahovat finální řešení. Cílem je získání poznatků o sestavení matematického modelu ze slovní úlohy. Tento matematický model se vytváří z účelové funkce, kterou lze maximalizovat nebo naopak minimalizovat v důsledku zadání úlohy. Dále je nutné si stanovit takzvané omezující podmínky, které zahrnují vlastní omezení a podmínky nezápornosti. [1]

2.1 Optimální výrobní program

„Podnik vyrábí čtyři typy výrobků V_1, V_2, V_3, V_4. Na jejich výrobu používá tři druhy surovin S_1, S_2 a S_3. Suroviny S_1 má k dispozici 2 t, suroviny S_2 5 t a suroviny S_3 3 t. Na výrobu jednoho kusu výrobku V_1 je potřeba 5 kg S_1, 8 kg S_2 a 7 kg S_3. Na výrobu jednoho kusu výrobku V_2 je potřeba 10 kg S_1, 3 kg S_2 a 5 kg S_3. Na výrobu jednoho kusu výrobku V_3 je potřeba 7 kg S_1, 6 kg S_2 a 12 kg S_3. Na výrobu jednoho kusu výrobku V_4 je potřeba 9 kg S_1, 2 kg S_2 a 4 kg S_3. Zisk z výrobku V_1 je 60 Kč, z výrobku V_2 30 Kč, z výrobku V_3 40 Kč a z výrobku V_4 80 Kč. Určete výrobní program tak, aby podnik vyrobil aspoň 60 ks výrobku V_2.“ [13]

Tabulka 1: Optimální výrobní program (výchozí hodnoty)

	Výrobek 1 [ks]	Výrobek 2 [ks]	Výrobek 3 [ks]	Výrobek 4 [kg]	Množství materiálu k dispozici [kg]
Surovina 1 [kg]	5	10	7	9	2000
Surovina 2 [kg]	8	3	9	2	5000
Surovina 3 [kg]	7	5	3	4	3000
Cena za ks [Kč]	60	30	40	80	

2.1.1 Sestavení matematického modelu úlohy

Na počátku je třeba vytvořit účelovou funkci. V účelové funkci bude každý z výrobků reprezentovaný proměnnou *x*. Tedy x_1, x_2, x_3 a x_4. Za jeden vyrobený kus Výrobku 1 obdrží podnik 60 Kč. Za jeden vyrobený kus Výrobku 2 obdrží podnik 30 Kč. Za jeden kus vyrobeného Výrobku 3 obdrží podnik 40 Kč. A za jeden kus Výrobku 4 obdrží podnik 80 Kč. Podnik chce dosáhnout maximálního zisku. Bude funkci tedy maximalizovat. Účelová funkce bude mít tento tvar:

$$f(x) = 60x_1 + 30x_2 + 40x_3 + 80x_4 \rightarrow max. \text{ [1]}$$

Další krok bude vymezovat limity skladových zásob. Na vyrobené množství Výrobku 1 spotřebuje podnik 5 kg suroviny S_1. Na vyrobené množství Výrobku 2 spotřebuje 10 kg suroviny S_2. Na vyrobené množství Výrobku 3 spotřebuje 7 kg suroviny S_3. A na vyrobené množství Výrobku 4 spotřebuje podnik 9 kg suroviny S_1. Suroviny S_1 je však pouze 2000 kg. Výsledná nerovnice bude mít tvar:

$$5x_1 + 10x_2 + 7x_3 + 9x_3 \leq 2\,000.[1]$$

Pro získání podmínek pro suroviny S_1, S_2 a S_3 budeme postupovat stejně. Musíme vzít v úvahu skutečnost, že nemůžeme vyrobit záporné množství produktů, zapíšeme to podle podmínky $x_i \geq 0, i = 1, 2, 3, 4$.

Slovní úloha se přiřadí následovně pomocí matematického zápisu:

$$f(x) = 60 + 30x_2 + 40x_3 + 80x_4 \rightarrow max\ [1]$$

s následujícími podmínkami:

$$5x_1 + 10x_2 + 7x_3 + 9x_3 \leq 2\,000$$

$$8x_1 + 3x_2 + 9x_3 + 2x_3 \leq 5\,000$$

$$7x_1 + 5x_2 + 3x_3 + 4x_3 \leq 3\,000$$

$$x_2 \geq 60$$

$$x_i \geq 0, i = 1, 2, 3, 4.\ [1]$$

2.2 Směšovací problém

„Podnik vyrábí směsi ze šesti různých složek S_1, …, S_6. Kvalita výsledné směsi je hodnocena pomocí parametrů P_1, …, P_4. Výsledná směs musí získat ve skupině parametrů P_1 alespoň 65 bodů, ve skupině parametrů P_2 alespoň 40 bodů, ve skupině P_3 alespoň 55 bodů a ve skupině P_4 alespoň 70 bodů. Složka S_2 má mít ve směsi zastoupení alespoň 40 kg a složka S_5 se musí pohybovat v rozmezí od 20 kg do 100 kg. Pořizovací cena jednotlivých složek a jejich kvalitativní ohodnocení ve skupinách parametrů P_1, P_2, P_3 a P_4 je uvedeno v Tabulce 2. Je nutné určit, jaké množství jednotlivých složek bude výsledná směs obsahovat.“ [13]

Tabulka 2: Směšovací problém (výchozí hodnoty) [13]

Složka	Kvalita složky ve skupině parametrů				Pořizovací cena složky [Kč/kg]
	P_1 [body/kg]	P_2 [body/kg]	P_3 [body/kg]	P_4 [body/kg]	
S_1	2	5	5	3	20
S_2	6	7	2	6	110
S_3	5	9	4	4	60
S_4	5	6	9	9	80
S_5	8	6	2	5	75
S_6	7	4	7	3	90

2.2.1 Sestavení matematického modelu úlohy

Nejprve musí být opět zkonstruována účelová funkce. Aby jednotlivé složky vytvořily směs, S_1, S_2, S_3, S_4, S_5 a S_6 nahradíme proměnnými x_1, x_2, x_3, x_4, x_5 a x_6. Za výrobní množství x_1 firma zaplatí pořizovací cenu $20x_1$ Kč, za x_2 je to $110x_2$ Kč, za x_3 to bude $60x_3$ Kč, za x_4 to je $80x_4$ Kč, za x_5 je to $75x_5$ Kč, a za x_6 je to $90x_6$ Kč. Výslednou funkci minimalizujeme, protože společnost chce získat požadovaný mix za co nejnižší cenu. To bude vypadat nějak takto:

$$f(x) = 20x_1 + 110x_2 + 60x_3 + 80x_4 + 75x_5 + 90x_6. \text{ [1]}$$

Dále je potřeba matematicky vyjádřit skutečnost, že jsme limitováni minimálním počtem bodů, které musí výsledná směs dosáhnout ve skupinách parametrů P_1, P_2, P_3 a P_4. Za 1 kg složky S_1 obsažené v celkové směsi obdrží směs 2 body ve skupině P_1, za 1 kg složky S_2 6 bodů, za 1 kg složky S_3 5 bodů, za 1 kg složky S_4 5 bodů, za 1 kg složky S_5 8 bodů a za 1 kg složky S_6 7 bodů. Celkový počet bodů musí být minimálně 65 pro parametr P_1. Výsledná nerovnice bude mít tedy tvar takový:

$$2x_1 + 6x_2 + 5x_3 + 5x_4 + 8x_5 + 7x_6 \geq 65. \text{ [1]}$$

Stejnými kroky budeme postupovat pro získání podmínek pro všechna ostatní bodová hodnocení P_2, P_3, P_4.

Nakonec je třeba vzít v úvahu podmínku, že minimální množství složky S_2 ve směsi je minimálně 40 kilogramů. To zapíšeme s podmínkou $x2 \geq 40$. Podobně zapíšeme podmínku pro složku S_5, která musí mít ve směsi hmotnost mezi 20 kg a 100 kg. Musíme mít na paměti, že nemůžeme mít záporná množství ostatních složek S_1, S_3, S_4 a S_6. Zapíšeme to pomocí $x_j \geq 0, j = 1, 3, 4, 6$. [1]

Matematický zápis slovní úlohy bude vypadat takto:

$$f(x) = 20x_1 + 110x_2 + 60x_3 + 80x_4 + 75x_5 + 90x_6 \rightarrow min$$

za podmínek:

$$2x_1 + 6x_2 + 5x_3 + 5x_4 + 8x_5 + 7x_6 \geq 65$$

$$5x_1 + 7x_2 + 8x_3 + 6x_4 + 6x_5 + 4_{x6} \geq 40$$

$$5x_1 + 2x_2 + 4x_3 + 9x_4 + 2x_5 + 7x_6 \geq 55$$

$$3x_1 + 6x_2 + 4x_3 + 9x_4 + 5x_5 + 3x_6 \geq 70$$

$$x_2 \geq 40, x_5 \geq 20, x_5 \leq 100, x_j \geq 0, j = 1, 3, 4, 6.\ [1]$$

2.3 Dělení materiálu

„Na skladě je 90 ks tyčí délky 120 cm a 130 ks tyčí délky 140 cm. Pro další výrobu je třeba z nich vyrobit tyče délky 50 cm a 70 cm. Jak je třeba rozřezat zásobu tyčí na skladě, potřebujeme-li získat co nejvíce tyčí délky 50 cm a alespoň 220 ks tyčí délky 70 cm? Odpad z výroby nesmí být větší než 5000 cm.“ [13]

Tabulka 3: Možné způsoby řezu tyčí [13]

	Možné způsoby řezu				
Skladové zásoby	120cm tyče		140cm tyče		
Způsob řezu	1.	2	3.	4.	5.
Počet tyčí o délce 50 cm [ks]	1	2	0	1	2
Počet tyčí o délce 70 cm [ks]	1	0	2	1	0
Množství řezného odpadu [cm]	0	20	0	20	40

2.3.1 Sestavení matematického modelu úlohy

Proměnné v úloze odpovídají způsobům řezu.

Sestavíme matematický model úlohy. Jako první vytvoříme účelovou funkci. V zadání je požadováno vyrobení co nejvíce 50cm tyčí. Tuto skutečnost použijeme k sestavení účelové funkce. Pokud rozřežeme 1 ks 120cm tyče podle způsobu 1, získáme 1 ks 50cm tyče. Podle prvního způsobu budeme řezat x_1 tyčí. x_1 tedy bude celkový počet tyčí délky 50 cm podle prvního způsobu. Podle 2. způsobu bude celkový počet 50cm tyčí $2x_2$.

$$f(x) = x_1 + 2x_2 + 0x_3 + x_4 + 2x_5$$

Za omezení se dá považovat jak množství tyčí, které je na skladě k dispozici, tak i stanovený spodní limit pro počet vyrobených tyčí délky 70 cm a samozřejmě i maximální množství odpadu, které nesmí být překročeno.

120cm tyče řezáme podle 1. a 2. způsobu. Podmínka pro počet 120cm tyčí, které jsou na skladě k dispozici:

$$x_1 + x_2 \leq 90$$

Tyče délky 140 cm řežeme 3., 4. a 5. způsobem. Podmínka pro počet tyčí délky 140 cm, které jsou na skladě:

$$x_3 + x_4 + x_5 \leq 130$$

Podmínka pro počet vyrobených tyčí délky 70 cm:

$$x_1 + 0x_2 + 2x_3 + x_4 + 0x_5 \geq 220$$

Podmínka pro odpad z výroby:

$$0x_1 + 20x_2 + 0x_3 + 20x_4 + 40x_5 \leq 5000$$

Výsledný matematický model bude tedy vypadat takto:

$$f(x) = x_1 + 2x_2 + 0x_3 + x_4 + 2x_5$$

při omezení:

$$x_1 + x_2 \leq 90$$

$$x_3 + x_4 + x_5 \leq 130$$

$$x_1 + 2x_3 + x_4 \geq 220$$

$$20x_2 + 20x_4 + 40x_5 \leq 5000$$

$$x_{1,2,3,4,5} \geq 0$$

2.4 Dopravní problém

„Ze skladů S_1, S_2, S_3 a S_4 je třeba rozvézt zboží do prodejen P_1, P_2, P_3, P_4 a P_5. V Tabulce 4 jsou uvedeny náklady na distribuci jedné jednotky zboží mezi jednotlivými sklady a prodejnami, dále tabulka obsahuje údaje o kapacitách všech skladů a požadavcích prodejen. Určete, jak má být zboží distribuováno do prodejen.“ [13]

Tabulka 4: Dopravní problém (výchozí hodnoty) [13]

		Prodejny	P_1	P_2	P_3	P_4	P_5
		Požadavky prodejen	200	500	400	300	600
Sklady	Kapacity skladů [ks]		Jednotkové náklady na přepravu ze skladu S_i do obchodu P_j [Kč/ks]				
S_1	800		6	10	5	8	7
S_2	400		4	8	12	6	9
S_3	600		11	15	10	9	5
S_5	200		8	6	2	8	14

2.4.1 Sestavení matematického modelu úlohy

Za prvé je třeba zkonstruovat účelovou funkci. Proměnné u dopravního problému budou představovat množství zboží, které bude distribuováno ze skladů do prodejen. Vypočítáme tedy celkové množství zboží jako součin skladů a prodejen: $S * P = 20$.

Při takovém množství proměnných je třeba vytvořit nové proměnné s odpovídajícími indexy, tedy x_{ij}, kde i = 1, 2, 3, 4; j = 1, 2, 3, 4, 5.

Jednotkové náklady na distribuci zboží ze skladu do prodejny je nutné zahrnout do účelové funkce. Přebíráme je z Tabulky 4. Náklady na distribuci zboží x_{11} ze skladu S_1 do prodejny P_1 jsou 6 Kč. Náklady na distribuci 1 ks zboží x_{12} ze skladu S_1 do prodejny P_2 jsou 10 Kč. Totéž uděláme pro osobní zboží.

Výsledná funkce, kterou budeme minimalizovat, bude následující:

$$f(x) = 6x_{11} + 10x_{12} + 5x_{13} + 8x_{14} + 7x_{15} +$$

$$4x_{21} + 8x_{22} + 12x_{23} + 6x_{24} + 9x_{25} +$$
$$11x_{31} + 15x_{32} + 10x_{33} + 9x_{34} + 5x_{35} +$$
$$8x_{41} + 6x_{42} + 2x_{43} + 8x_{44} + 14x_{45}. [1]$$

V dalším kroku musíme matematicky vyjádřit fakt, že jsme omezení kapacitou skladů S_1, S_2, S_3 a S_4. Ze skladu S_1 se do prodejny P_1 distribuuje x_{11} ks zboží, do prodejny P_2 se distribuuje x_{12} ks zboží, do prodejny P_3 se distribuuje x_{13} ks zboží, do prodejny P_4 se distribuuje x_{14} ks zboží a do prodejny P_5 se distribuuje x_{15} ks zboží. Kapacita skladu S_1 je 800 položek. Výsledná rovnice bude následující:

$$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} = 800.$$

Totožným způsobem postupujeme pro získání podmínek pro sklady S_2, S_3 a S_4.

V dalším kroku musíme matematicky vyjádřit fakt, že jsme omezení nároky prodejen P_1, P_2, P_3, P_4 a P_5. Dopravíme zboží ze skladů S_1, S_2, S_3 a S_4, konkrétně zboží x_{11}, x_{21}, x_{31} a x_{41} do prodejny P_1 s potřebou 200 položek zboží. Výsledná rovnice tedy bude taková:

$$x_{11} + x_{21} + x_{31} + x_{41} = 200. [1]$$

Totožným způsobem postupujeme pro získání podmínek pro prodejny P_2, P_3, P_4 a P_5.

V neposlední řadě musíme vzít v úvahu, že nelze distribuovat záporná množství zboží. To ošetříme podmínkou $x_{ij} \geq 0$, i = 1, 2, 3, 4; j = 1, 2, 3, 4, 5. [1]

Slovní úloha tedy získá matematickým zápisem níže uvedené zadání:

$$f(x) = 6x_{11} + 10x_{12} + 5x_{13} + 8x_{14} + 7x_{15} +$$
$$4x_{21} + 8x_{22} + 12x_{23} + 6x_{24} + 9x_{25} +$$
$$11x_{31} + 15x_{32} + 10x_{33} + 9x_{34} + 5x_{35} +$$
$$8x_{41} + 6x_{42} + 2x_{43} + 8x_{44} + 14x_{45} \rightarrow min.$$

za podmínek:

$$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} = 800$$

$$x_{21} + x_{22} + x_{23} + x_{24} + x_{25} = 400$$

$$x_{31} + x_{32} + x_{33} + x_{34} + x_{35} = 600$$

$$x_{41} + x_{42} + x_{43} + x_{44} + x_{45} = 200$$

$$x_{11} + x_{21} + x_{31} + x_{41} = 200$$

$$x_{12} + x_{22} + x_{32} + x_{42} = 500$$

$$x_{13} + x_{23} + x_{33} + x_{43} = 400$$

$$x_{14} + x_{24} + x_{34} + x_{44} = 300$$

$$x_{15} + x_{25} + x_{35} + x_{45} = 600$$

$$x_{ij} \geq 0, i = 1, 2, 3, 4; j = 1, 2, 3, 4, 5.$$ [1]

2.5 Metody a řešení lineárního programování

Existuje několik metod, které lze použít k řešení úloh lineárního programování. [14]

2.5.1 Metoda jednoduchého simplexu

Jedná se o metodu z nejstarších a nejznámějších metod pro řešení lineárních programů. Princip spočívá v postupném pohybu po vrcholech polyedru omezení směrem k optimálního řešení. Každý krok vede ke zlepšení cílové funkce, dokud není nalezeno optimální řešení. [14]

2.5.2 Duální simplexová metoda

Duální simplexová metoda pracuje s duálním lineárním programem a hledá optimální řešení pomocí podobného postupu jako jednoduchý simplex. Avšak namísto pohybu v primárním prostoru se pohybuje v duálním prostoru. [14]

2.5.3 Metoda Karmarkarova

Tato metoda kombinuje metodu vnitřního bodů s rychlými algoritmy lineární algebry, což umožňuje rychlou konvergenci k optimálnímu řešení. [14]

2.5.4 Metoda vnitřního bodu

Metoda se liší od klasického simplexového přístupu a je účinná zejména pro velké lineární programy. pracuje s vnitřními body polyedru omezení a umožňuje rychlejší konvergenci.

Každá z těchto metod má své výhody i nevýhody a vhodnost metody závisí na konkrétních charakteristikách problému lineárního programování. Je důležité vybrat nejvhodnější metodu pro danou úlohu s ohledem na rozměr problému, dostupné prostředky a požadovanou přesnost. [14]

V této bakalářské práci je kladen důraz na jednofázovou a dvoufázovou metodu.

3 METODA SIMPLEXOVÉ TABULKY

Metoda simplexové tabulky je jedním z nejvíce používaných algoritmů pro řešení lineárního programování. Byla vyvinuta Georgem Dantzigem v roce 1947 a od té doby se stala základním nástrojem v oblasti optimalizace. Tato metoda umožňuje nalézt optimální řešení úlohy lineárního programování pomocí systematického prohledávání možných řešení v polyedrovém prostoru. [8]

3.1 Základní principy

3.1.1 Standardní forma

U metody simplexové tabulky je vyžadováno, aby byla úloha lineárního programování převedena do standardního tvaru. Všechna omezení a cílová funkce jsou v tomto tvaru lineárního programu ve tvaru rovnic. [15]

Maximalizovat:

$c^T x$

za podmínek:

$Ax \leq b$

$x \geq 0$

kde:

- **c** je vektor koeficientů cílové funkce,
- **x** je vektor proměnných,
- **A** je matice koeficientů omezení,
- **b** je vektor pravých stran omezení,
- $\geq$ označuje nezápornost proměnných.

3.1.2 Základní řešení

Centrálním konceptem metody simplexové tabulky jsou základní řešení. Vrcholům polyedru, který je definován omezeními lineárního programu odpovídají tyto základní řešení. Sadou nezávislých proměnných, které mají nenulovou hodnotu, a všemi ostatními proměnnými, které jsou rovny nule, je charakterizováno každé základní řešení. [15]

3.1.3 Postup simplexové metody

1. **Inicializace**

   Začínáme s libovolným základním řešením. Určíme počáteční bázové proměnné, které odpovídají nenulovým hodnotám základního řešení.

2. **Test optimality**

   Zkontrolujeme, zda je současné řešení optimální (tzn. jestli hodnota účelové funkce je maximální (resp. minimální). Pokud ano, končíme.

3. **Nalezení směru**

   Pokud současné řešení není optimální, hledáme směr, jakým se pohybovat, aby se hodnota cílové funkce zlepšila. Tento směr je určen pomocí simplexového pravidla.

4. **Určení kroku**

   Po nalezení směru je nutné určit délku kroku, jakou se pohybujeme po směru. To zahrnuje výpočet tzv. „vstupního sloupce“ a „výstupního řádku“.

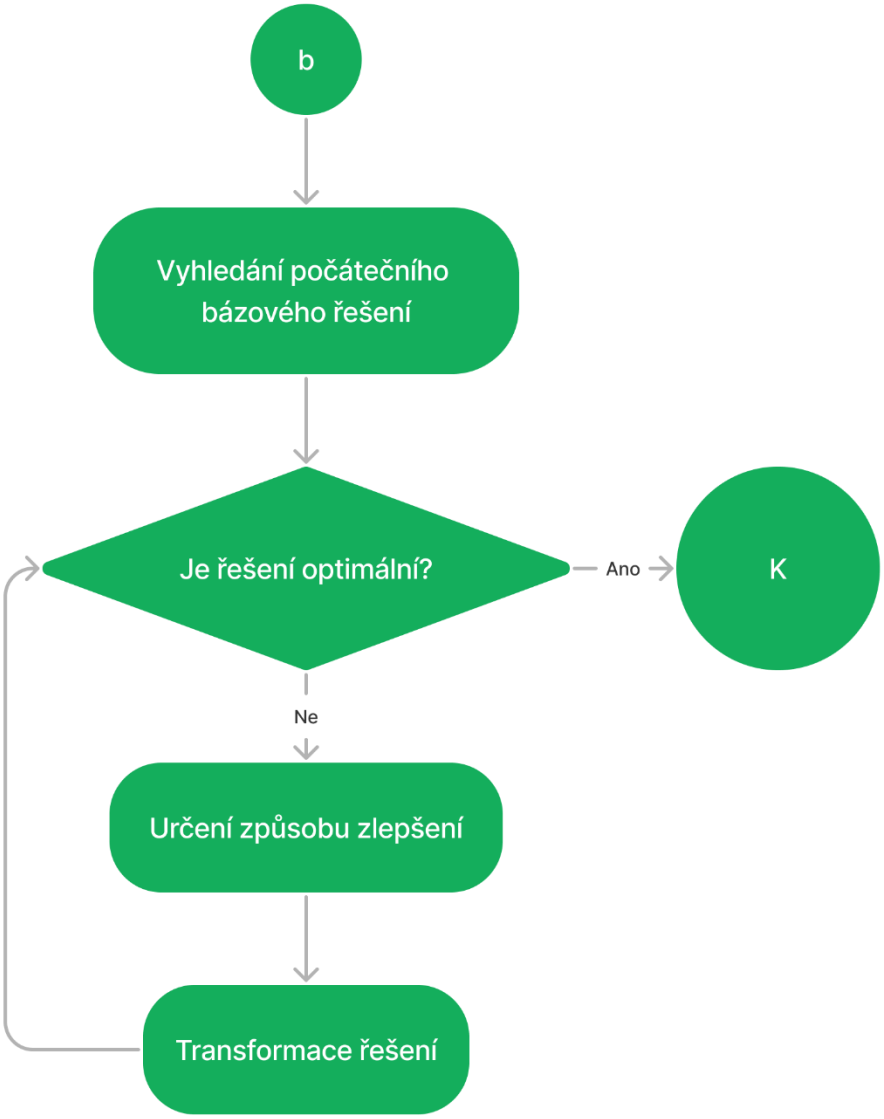
5. **Aktualizace**

   Aktualizujeme současné řešení podle kroku a směru.

6. **Znovu přehodnotíme**

   Pokud nové řešení splňuje všechna omezení pokračujeme zpět k bodu 2; jinak se vrátíme k bodu 3 hledáme nový směr.

Tento proces se opakuje, dokud není dosaženo optimálního řešení. [15]



Obrázek 2: Diagram hledání optimálního řešení [zdroj: vlastní]

3.2 Složitost a efektivita

„Simplexova metoda má obecně exponenciální složitost v nejhorším případě, což znamená, že může být poměrně pomalá pro velké instance problému. Nicméně, v praxi se často ukazuje jako efektivní, protože. řeší mnoho problému rychle a spolehlivě.“ [15]

3.3 Rozšíření a varianty

„Existuje mnoho rozšíření a variant simplexové metody, které zlepšují její výkonnost nebo ji upravují pro specifické typy problémů.

Některé z těchto variant zahrnují:

- ***Duální simplexova metoda:*** *Optimalizuje duální problém lineárního programu*
- ***Dvoufázová metoda:*** *Používá se k nalezení počátečního řešení pro lineární programy se speciálními vlastnostmi.“* [15]

3.4 Aplikace

„Simplexova metoda má široké spektrum aplikací v různých odvětvích, včetně průmyslu, financí, dopravy a logistiky. Používá se pro optimalizaci výrobních procesů, plánování výroby, řízení zásob, alokaci zdrojů a mnoho dalších úkolů, které lze formulovat jako lineární program.“ [15]

Pro řešení úloh lineárního programování se pořád používá simplexova metoda, avšak dnes již nikdo ruční výpočet neprovádí, protože existuje řada programů pro praktické řešení.

3.5 Příklad

Tento příklad je čerpán ze zdroje [12].

Zadání úlohy:

Určete hodnoty proměnné $x = [x_1, x_2]$, kde $x_1 \geq 0, x_2 \geq 0$, která maximalizuje kritérium J

$$J = 2x_1 + 3x_3$$

a vyhovuje podmínkám

$$x_1 + x_2 \leq 5$$

$$x_1 + 2x_2 \leq 8$$

Ověření platnosti základních podmínek:

1. Maximalizujeme kritérium – ✓ (ano, $2x_1 + 3x_2 \rightarrow max$),
2. Pravé strany musí být nezáporné – ✓ (ano, číslo 5 i 8 jsou kladná čísla),
3. Omezující podmínky jsou vymezeny nerovnostmi typu ≤ – ✓ (ano, v obou rovnicích podmínky jsou uvedeny znaménka ≤),
4. Proměnné x jsou nezáporné – ✓ (v zadání je definováno, že $x_1 \geq 0,\ x_2 \geq 0$).

Zbavíme se nerovnosti a maxima:

1. Nerovnosti se zbavíme tak, že k levé straně nerovnice přidáme další člen ($s_i > 0$), který bude vždy kladný, v našem případě bude rovnice vypadat takto:

$$x_1 + x_2 + s_1 = 5$$

$$x_1 + 2x_2 + s_2 = 8$$

2. Do simplexovy tabulky zapíšeme také kritérium. Pokud bychom použili základní tvar, tedy v tomto případě $2x_1 + 3x_2 \rightarrow max$ bylo by nutné do simplexovy tabulky na pravou stranu dát ∞. Z tohoto důvodu se upraví tento řádek rovnice do tvaru

$$-2x_1 - 3x_2 = 0$$

Protože maximum $2x_1 + 3x_2$ pro $x_1, x_2 \geq 0$ *je* ∞. Maximum pro $-2x_1 - 3x_2$ pro $x_1, x_2 \geq 0$ *je* 0.

Sestavení simplexovy tabulky:

Tabulka 5: Výchozí simplexova tabulka

x_1	x_2	x_3	x_4	b	
1	1	1	0	5	1. omezení
1	2	0	1	8	2. omezení
-2	-3↑	0	0	0	řádek kritéria

Úprava simplexovy tabulky:

1. Najdeme nejmenší záporný koeficient v řádku kritéria → *klíčový sloupec*.

   Nejmenší záporný koeficient je koeficient, na kterém hodnota kritéria závisí nejvíce. V tomto případě je to hodnota -3. Tomu odpovídá sloupec proměnné x_2.

2. Pro kladné prvky v klíčovém sloupci najdeme nejmenší podíl pravé strany a prvku v klíčovém sloupci → *klíčový řádek*. V případě, že je více klíčových řádků, lze vybrat libovolný z nich.

   Pro první omezení je to podíl $\frac{5}{1} = 5$, pro druhé omezení je to podíl $\frac{8}{2} = 4$. Menší podíl je v řádku s druhým omezením, a to bude tedy klíčový řádek.

3. Na průsečíku klíčového sloupce a klíčového řádku leží *klíčový prvek*.

   V našem případě je tedy klíčový prvek číslo 2.

4. Klíčový řádek celý vydělíme klíčovým prvkem a dostaneme

Tabulka 6: Dělení klíčového řádku klíčovým prvkem

x_1	x_2	x_3	x_4	b
1/2	1	0	1/2	1/2

5. Ostatní řádky tabulky redukujeme klíčovým řádkem, tj. klíčový řádek násobíme takovým číslem, aby po přičtení k danému řádku byla v klíčovém sloupci nula.

   Tedy: (*i*) poslední řádek má v klíčovém sloupci -3; klíčový řádek násobíme 3 a přičteme k prvnímu řádku; dostaneme poslední řádek nové tabulky (dole); (*ii*) první řádek má v klíčovém sloupci 1, klíčový řádek proto násobíme -1 a přičteme k prvnímu řádku, dostaneme první řádek v nové tabulce; (*iii*) druhý řádek je klíčový, v něm zůstává upravený klíčový řádek.

   Tady je nová tabulka:

Tabulka 7: Simplexova tabulka po první iteraci

x_1	x_2	x_3	x_4	b	**1. iterace**
1/2	0	1	- 1/2	1	1. omezení
1/2	1	0	1/2	4	2. omezení
- 1/2	0	0	3/2	12	řádek kritéria

V nové simplexové tabulce se hodnota kritéria J zvýšila na hodnotu 12. Přesto není řešení ještě ideální, protože v řádku kritéria se dále objevuje záporný koeficient. Z tohoto důvodu budeme znovu upravovat simplexovu tabulku.

6. V případě, že se v řádku kritérií objevuje záporný koeficient (v nové tabulce je to prvek $-\frac{1}{2}$, opakuje se postup od bodu 1 a to do doby, než jsou všechny prvky v řádku kritéria nezáporné.

V našem případě dostaneme konečnou tabulku po další kroku:

Tabulka 8: Výsledná simplexova tabulka

x_1	x_2	x_3	x_4	b	**2. iterace**
1	0	2	-1	2	1. omezení
0	1	-1	1	3	2. omezení
0	0	1	1	13	řádek kritéria

Protože všechny prvky v řádku kritéria jsou nezáporné, úloha končí.

Výsledek:

Optimální řešení nalezené simplexovou metodou v tomto případě je pro $x_1 = 2$ a $x_2 = 3$. Hodnota kritéria J = 13. Řešení dostaneme, když tabulku vyjádříme opět v rovnicích a z nich vypočteme x_1 a x_2.

Zkouška správnosti:

Účelová funkce: $2 * 2 + 3 * 3 = 13$

Obě omezení jsou splněna jako rovnost. Přídatné proměnné jsou nulové:

- $2 + 3 = 5$
- $2 + 2 * 3 = 8$

3.6 Jednofázová simplexová metoda

Abychom mohli jednofázovou simplexovou metodu použít, je to možné jen v případě, že všechna vlastní omezení úlohy LP budou definována jako nerovnice typu „≤“. Pro získání výchozího základního řešení stačí soustavu převést pomocí přídatných proměnných na soustavu rovnic, jsou-li v úloze lineárního programování všechna vlastní omezení ve tvaru nerovnic „≤“. [16]

3.7 Dvoufázová simplexová metoda

Jestliže úloha lineárního programování neobsahuje všechny omezení ve tvaru nerovnic „≤“, potom získání výchozího základního řešení úlohy není tak snadné a představuje vlastně

celou **I. fázi** výpočtu. Teprve až **II. fáze** se zabývá optimalizací účelové funkce. Tato fáze je již naprosto shodná s jednofázovou simplexovou metodou. [16]

Pomocná úloha má následující tvar:

$$z = 8x_1 + 2x_2 + 5x_3 \rightarrow min$$

při omezení:

$$x_1 \quad + x_3 \geq 24$$

$$2x_1 + x_2 + x_3 \geq 30$$

$$x_{1,23} \geq 0$$

Nalezněte výchozí řešení této úlohy a vypočtěte optimální řešení simplexovou metodou.

Nerovnice vyrovnáme na rovnice odečtením přídatných proměnných:

$$x_1 \quad + x_3 - x_4 \quad = 24$$

$$2x_1 + x_2 + x_3 \quad - x_5 = 30$$

$$x_{1,2,3,4,5} \geq 0$$

Soustava rovnic není v kanonickém tvaru. Kanonický tvar získáme přičtením **pomocných proměnných** k oběma rovnicím:

$$x_1 \quad + x_3 - x_4 \quad + y_1 \quad = 24$$

$$2x_1 + x_2 + x_3 \quad - x_5 \quad + y_2 = 30$$

$$x_{1,2,3,4,5} \geq 0$$

$$y_{1,2,3} \geq 0$$

Aby byla takto rozšířená soustava rovnic ekvivalentní s původní soustavou, musíme během výpočtu vynulovat pomocné proměnné. Sestavíme pomocnou účelovou funkci

$$z' = y_1 + y_2 \rightarrow min,$$

ve které minimalizujeme součet všech pomocných proměnných. Při zachování podmínek nezápornosti bude minimum $z' = 0$, budou-li všechny pomocné proměnné rovny nule. Pomocnou funkci převedeme do anulovaného tvaru:

$$z' - y_1 - y_1 = 0.$$

K vynulované funkci z' přičteme obě omezení rozšířené soustavy. Tím upravíme vektory základních proměnných y_1 a y_2 na jednotkové:

$$z' + 3x_1 + x_2 + 2x_3 - x_4 - x_5 = 54.$$

Dostáváme rozšířený model úlohy lineárního programování:

$x_1 \quad + x_3 - x_4 \quad + y_1 \quad = 24$

$2x_1 + x_2 + x_3 \quad - x_5 \quad + y_2 \quad = 30$

$-8x_1 - 2x_2 - 5x_3 \quad + z \quad = 0$

$3x_1 + 3x_2 + 2x_3 - x_4 - x_5 \quad + z' = 24$

$x_{1,2,3,4,5} \geq 0$

$y_{1,2} \geq 0$

Hodnoty proměnných jsou odtud $x_1 = 0,\ x_2 = 0,\ x_3 = 0,\ x_4 = 0,\ x_5 = 0,\ y_1 = 24,\ y_2 = 30,\ +z' = 50,\ z = 0.$ Výchozím řešením rozšířené úlohy LP je vektor $x = (0,0,0,0,0,24,30)$, pomocná účelová funkce má hodnotu $z' = 54$.

Výchozí řešení přepíšeme do simplexové tabulky. Základními proměnnými jsou pomocné proměnné y_1 a y_2.

Tabulka 9: První fáze výpočtu dvoufázové metody

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	b_i	t
y_1	**1**	0	1	-1	0	1	0	24	24/1
y_2	**2**	**1**	**1**	**0**	**-1**	**0**	**1**	**30**	**30/2←**
z_j	**-8**	-2	-5	0	0	0	0	0	-
z'_j	**3**↑	1	2	-1	-1	0	0	24	-
y_1	**0**	**-1/2**	**1/2**	**-1**	**1/2**	**1**	**-1/2**	**9**	**18←**
x_1	1	½	**1/2**	0	-1/2	0	1/2	15	30
z_j	0	-2	**-1**	0	-4	0	4	120	-
z'_j	0	-1/2	**1/2**↑	-1	1/2	0	-3/2	9	-
x_3	0	-1	1	-2	1	2	-1	18	-

x_1	1	1	0	1	-1	-1	1	6	-
z_j	0	1	0	-2	-3	2	3	138	-
z'_j	0	0	0	0	0	-1	-1	0	-

Úlohu řešíme dvoufázovou simplexovou metodou. V první fázi testujeme optimalitu podle pomocné účelové funkce z', kterou minimalizujeme. Klíčový sloupec určíme podle maximálního kladného koeficientu v pomocné účelové funkci. V Tabulce 10 je to $\max(3,1,2) = 3$. Vystupuje pomocná proměnná y_2. Ve druhé iteraci je vystupující proměnná pomocná proměnná y_1. Ve třetí iteraci jsou obě pomocné proměnné rovny nule a pomocná účelová funkce se vynulovala. Vypočetli jsme první přípustné řešení zadané úlohy, první fáze výpočtu tím skončila.

Vynecháme sloupce pomocných proměnných a řádek pomocné účelové funkce a pokračujeme minimalizací účelové funkce *z*. Vstupující proměnná je podle z_2=1 proměnná x_2, vystupující proměnná je x_1.

Tabulka 10: Druhá fáze výpočtu dvoufázové metody – optimální řešení

3.iterace	x_1	x_2	x_3	x_4	x_5	b_i	*t*
x_3	0	**-1**	1	-2	1	18	-
x_1	**1**	**1**	**0**	**1**	**-1**	**6**	**6←**
z_j	0	**1**↑	0	-2	-3	138	-
x_3	1	0	1	-1	0	24	-
x_2	1	1	0	1	-1	6	-
z_j	-1	0	0	-3	-2	132	-

Ve čtvrté iteraci jsou všechny koeficienty v řádce *z* nekladné. Optimálním řešením je vektor $x = (0,6,24,0,0), z = 132$. [13]

Shrnutí způsobu doplňování přídatných a pomocných proměnných ve dvoufázové metodě je v Tabulce 11.

Tabulka 11: Shrnutí způsobu doplnění přídatných a pomocných proměnných [16]

Typ omezení	Přídatná proměnná	Pomocná proměnná
≤	$+x$	
≥	$-x$	$+y$
=		$+y$

3.8 Zakončení výpočtu

Výpočet může skončit buď nalezením optimálního řešení (jedno nebo nekonečné množství) nebo konstatováním, že optimální řešení neexistuje.

Právě jedno optimální řešení

$$2x_1 + x_2 + x_3 \leq 25$$

$$x_1 + 2x_2 + x_3 \leq 8$$

$$z = 24x_1 + 20x_2 + 14x_3 \rightarrow max$$

K levým stranám omezení přičteme přídatné proměnné x_4 a x_5 a vynulujeme účelovou funkci. Výchozí řešení přepíšeme do Tabulky 12. Ve druhé iteraci jsou všechny koeficienty v řádce *z* nezáporné, vektor $x = (9,0,0,7,0)$ je tedy optimálním řešením.

Tabulka 12: Právě jedno optimální řešení

	x_1	x_2	x_3	x_4	x_5	b_i	*t*
x_4	**2**	1	1	1	0	25	25/2
x_5	**1**	**2**	**1**	**0**	**1**	**8**	**8←**
z_j	**-24↑**	-20	-14	0	0	0	-
x_4	0	-3	-1	1	-2	9	-
x_1	1	2	1	0	1	8	-
z_j	0	28	10	0	24	192	-

Redukované ceny u nezákladních proměnných jsou všechny rovny nule, toto optimální řešení je tedy jediné. Max. hodnota účelové funkce $z = 192$.

Nekonečně mnoho řešení

V Tabulce 13 je řešení úlohy lineárního programování, které jsme vypočetli ve druhé iteraci.

Tabulka 13: Optimální řešení lineárního programování

	x_1	x_2	x_3	x_4	x_5	b_i	*t*
x_4	**2**	1	1	1	0	25	25/2
x_5	**1**	**2**	**1**	**0**	**1**	**9**	**9←**
z_j	**-24↑**	-20	-24	0	0	0	-
x_4	0	-3	-1	1	-2	7	-
x_1	1	2	1	0	1	9	-
z_j	0	28	**0!**	0	24	216	-

Řešení v Tabulce 13 je optimální, ale ne jediné. U nezákladní proměnné x_3 je redukovaná cena rovna nule.

Znamená to, že zvýšení hodnoty této proměnné nezmění hodnotu účelové funkce, existuje tedy další optimální řešení, které někdy nazýváme alternativní optimální řešení. Vypočteme ho v Tabulce 14.

Tabulka 14: Výpočet alternativního optimálního řešení úlohy LP

2.ite-race	x_1	x_2	x_3	x_4	x_5	b_i	*t*
x_4	0	-3	**-1**	1	-2	7	-
x_1	**1**	**2**	**1**	**0**	**1**	**9**	**9←**
z_j	0	28	**0↑**	0	24	216	-
x_4	1	-1	0	1	-1	16	-
x_3	1	2	1	0	1	9	-
z_j	0	28	0	0	24	216	-

Vidíme, že řešení v Tabulce 14 je opět optimální s toutéž hodnotou z, ale s jinými hodnotami proměnných. Vypočetli jsme tedy dvě optimální řešení:

$$x^{(1)} = (9,0,0,7,0), z = 216,$$
$$x^{(2)} = (0,0,9,16,0), z = 216.$$

Obě řešení jsou základní, počet kladných proměnných je roven dvěma. Podle definice je optimálním řešením i každá konvexní kombinace vektorů optimálních řešení, např:

$$x^{(3)} = 1/2x^{(1)} + 1/2x^{(2)} = \left(\frac{9}{2}, 0, \frac{9}{2}, \frac{23}{2}, 0\right),$$
$$x^{(4)} = 1/5x^{(1)} + 2/5x^{(2)} + 2/5x^{(3)} = \left(\frac{18}{5}, 0, \frac{27}{5}, \frac{62}{5}, 0\right) \text{ apod.}$$

Neomezená účelová funkce

$$z = 5x_1 + x_2 + 3x_3 \rightarrow max$$
$$3x_1 + x_2 + 3x_3 \geq 5$$
$$2x_1 - 2x_2 + 2x_3 \leq 2$$
$$x_{1,2,3} \geq 0$$

Postup výpočtu je v Tabulce 15.

Tabulka 15: Neomezená úloha LP

	x_1	x_2	x_3	x_4	x_5	y_1	b_i	*t*
y_1	**3**	1	-3	-1	0	1	5	5/3
←x5	**2**	**-2**	**2**	**0**	**1**	**0**	**2**	**1**
z_j	**-5**	-1	-3	-1	0	0	5	-
z'_j	**3↑**	1	-3	-1	0	0	5	-
←y1	**0**	**4**	**-6**	**-1**	**-3/2**	**1**	**2**	**2**
x_1	1	**-1**	1	0	1/2	0	1	-
z_j	0	**-6**	2	0	5/2	0	5	-
z'_j	0	**4↑**	-6	-1	-3/2	0	2	-
x_2	0	1	**-3/2**	-1/4	-3/8	-	1/2	-

x_1	1	0	**-1/2**	-1/4	1/8	-	3/2	-
z_j	0	0	**-7↑**	-3/2	1/4	-	8	-

Ve třetí iteraci končí první fáze výpočtu. Podle řádky *z* je vstupující proměnná x_3, ale v klíčovém sloupci není ani jeden kladný koeficient. Proměnná x_3 tedy může nabývat jakýchkoliv nezáporných hodnot, aniž by se porušila přípustnost řešení, tzn. nezápornost pravých stran. Spolu s růstem proměnné x_3 roste neomezeně i hodnota účelové funkce.

Optimální řešení neexistuje.

Pozn. Můžeme se o tom přesvědčit dosazením $x_3 = t > 0$ do soustavy omezení v Tabulce 15, odkud je

$$x_1 = \frac{3}{2} + \frac{1}{2} * t,$$

$$x_2 = \frac{1}{2} + \frac{3}{2} * t,$$

$$z = 8 + 7 * t.$$

Dosadíme např.:

- $t = 1 \rightarrow x_1 = 2, x_2 = 2, z = 15,$
- $t = 100 \rightarrow x_1 = 51.5, x_2 = 150.5, z = 708,$
- $t = 10\,000 \rightarrow x_1 = 5001.5, x_2 = 15\,000, z = 70\,008$ atd.

Vidíme, že s rostoucí hodnotou $x_3 = t > 0$ roste i hodnota účelové funkce *z* nade všechny meze.

Úloha nemá řešení

$$2x_1 + 2x_2 + 3x_3 \leq 16$$

$$x_1 + 2x_2 + 2x_3 \geq 20$$

$$x_{1,2,3} \geq 0$$

$$z = 63x_1 + 27x_2 + 56x_3 \rightarrow min$$

V soustavě vlastních omezení je jen jedna nerovnice typu ≥, potřebujeme tedy pouze jednu pomocnou proměnnou, kterou přičteme ve druhém omezení. Dostáváme model v kanonickém tvaru:

$$
\begin{aligned}
2x_1 + 2x_2 + 3x_3 + x_4 &= 16\\
x_1 + 2x_2 + 2x_3 - x_5 + y_1 &= 20\\
-63x_1 - 27x_2 - 56x_3 + z &= 0\\
x_1 + 2x_2 + 2x_3 - x_5 + z' &= 20\\
x_{1,2,3,4,5} &\geq 0\\
y_1 &\geq 0
\end{aligned}
$$

Úlohu řešíme v Tabulce 16 dvoufázovou simplexovou metodou.

Tabulka 16: Neřešitelná úloha LP

	x_1	x_2	x_3	x_4	x_5	y_1	b_i	*t*
←x4	**2**	**2**	**3**	**1**	**0**	**0**	**16**	**8**
y_1	1	**2**	2	0	-1	1	20	10
z_j	-63	**-27↑**	-56	0	0	0	0	-
z'_j	1	**2**	2	0	-1	0	20	-
x_2	1	1	3/2	1/2	0	0	8	-
y_1	-1	0	-1	-1	-1	1	4	-
z_j	-36	0	-31/2	27/2	0	0	216	-
z'_j	-1	0	-1	0	-1	0	**4!**	-

Ve druhé iteraci není možno určit vstupující proměnnou, protože v řádce pomocné účelové funkce jsou všechny koeficienty nekladné. Našli jsme tedy minimum pomocné účelové funkce, které je ale větší než nula. Pomocná proměnná $y_1 = 4$, má tedy stále kladnou hodnotu. Původní úloha proto nemá přípustné řešení a její optimum tedy neexistuje. Podmínky jsou nekonzistentní, úloha je neřešitelná.

Pozn. Je to jediný případ. kdy má pomocná proměnná ekonomickou interpretaci. Ukazuje, o kolik je podmínka v omezení, ve kterém zůstala jako základní proměnná, nesplnitelná. V naší úloze jsou to 4 jednotky, takže druhé omezení bychom museli snížit alespoň o 4.

4 CITLIVOSTNÍ ANALÝZA

„Každý programový systém pro řešení úloh lineárního programování nabízí uživateli možnost zobrazit informace týkající se analýzy citlivosti optimálního řešení ve vztahu ke změnám ve vektoru pravých stran a vektoru cenových koeficientů.“ [16]

4.1 Koeficient citlivosti

Koeficient citlivosti daného omezení ukazuje, o kolik se eventuálně zvýší, popř. sníží hodnota účelové funkce, kdyby se zvýšila, popř. snížila pravá strana omezení o jednotku (v úloze maximalizace, a naopak v úloze minimalizace). [12]

Příklad

Tento příklad je čerpán ze zdroje [12].

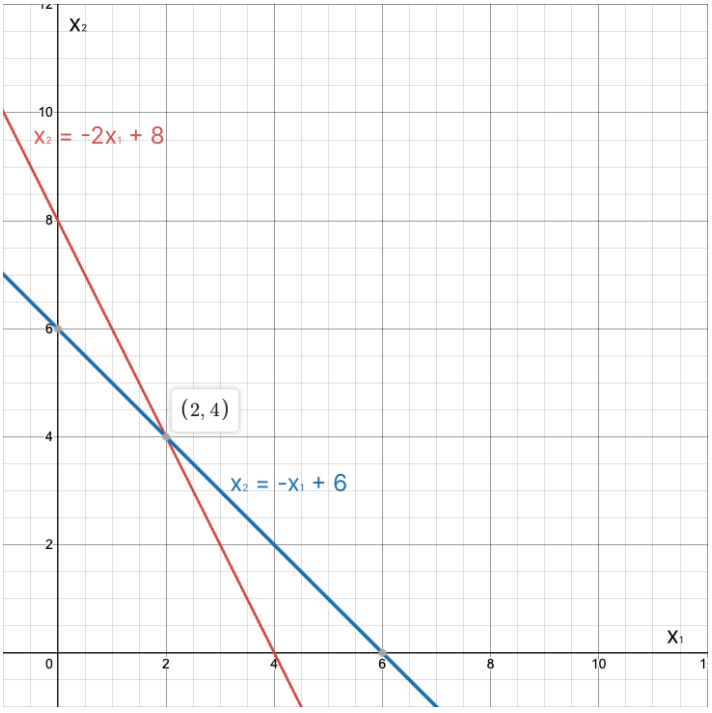
$$3x_1 + 2x_2 \rightarrow max$$

při omezení:

$$2x_1 + x_2 \leq 8$$

$$x_1 + x_2 \leq 6$$

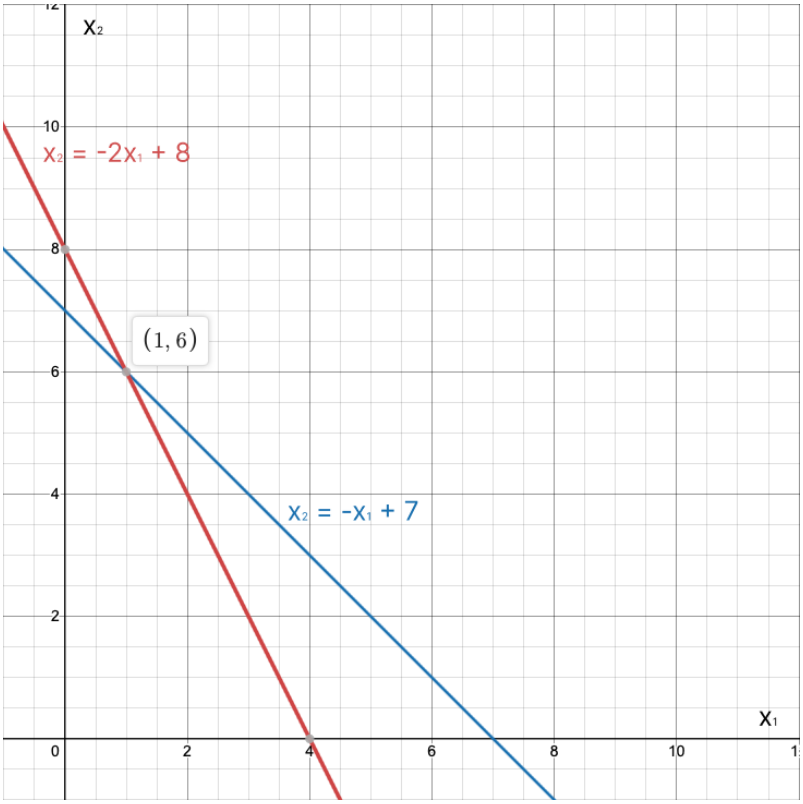
$$x_{1,2} \geq 0$$



Obrázek 3: Omezení zobrazená v grafu u citlivosti

Na základně daného kritéria dostaneme optimální řešení v bodě (2, 4) s hodnotou účelové funkce 14.

V případě, že pravou stranu druhého omezení zvýšíme o 1 (tedy na 7), bude omezení podle následujícího obrázku.



Obrázek 4: Omezení po zvýšení o 1 zobrazená v grafu

Optimální řešení se změní na (1, 6), tudíž hodnota účelové funkce vzroste na 15. Duální cena pro druhé omezení bude $15 - 14 = 1$.

Pro první omezení bude duální cena také rovna 1.

4.2 Koeficient stability

Koeficient stability udává, jak moc se musí zvýšit cena množství (pro maximalizaci, snížit cena množství pro minimalizaci), aby byla aktivní proměnnou, tj. aby změna množství způsobila změnu hodnoty kritéria. [12]

Příklad

Tento příklad je čerpán ze zdroje [12].

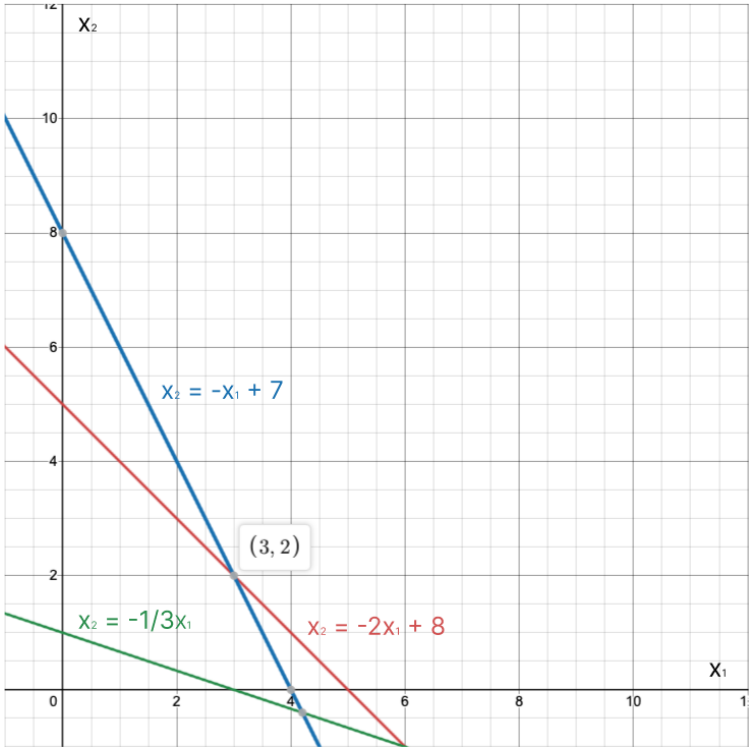
$$x_1 + 3x_2 \rightarrow max$$

při omezení:

$$x_1 + x_2 \leq 5$$

$$2x_1 + x_2 \leq 8$$

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



Obrázek 5: Omezení v grafu u stability

Kritérium roste s vrstevnicí $x_2 = -\frac{1}{3}x_1$ (zelená čára) a sice směrem doprava a nahoru. Vrstevnice s nejvyšší hodnotou prochází bodem (0, 5) hranice omezení. Ve směru proměnné *x* je omezena svou nezáporností.

Optimální řešení (0, 5) dá kritérium $c_1 * 0 + c_2 * 5$. Změna koeficientu c_1 hodnotu kritéria neovlivní. Aby se veličina x_1 stala aktivní, tj. aby změna c_1 začala ovlivňovat hodnotu kritéria, musel by optimální bod řešení přeskočit do dalšího vrcholu simplexu. tedy do bodu (3, 2).

Redukovaná cena veličiny *x* je dána hodnotou, o kterou by se musel zvýšit odpovídající koeficient cen (tady c_1), aby se veličina *x* stala aktivní.

V našem případě to evidentně nastane, když kriteriální vrstevnice bude rovnoběžná s rovnicí $y = -x + 5$, tj. s rovnicí $x + y = 5$. Tedy kritérium (1, 3) se musí změnit tak, aby druhá souřadnice cen kritéria zůstala stejná (tj. 3) a první souřadnice vytvořila vektor úměrný vektoru (1, 1), což jsou koeficienty zmíněného omezení. Požadovaný vektor tedy bude (3, 3) a jeho první souřadnice se změnila o hodnotu 2. **Redukovaná cena** pro veličinu *x* tedy bude 2.

Druhá veličina má redukovanou cenu 0.

II. PRAKTICKÁ ČÁST

5 TVORBA PROGRAMU PRO ŘEŠENÍ ÚLOHY LINEÁRNÍHO PROGRAMOVÁNÍ

Pro tuto bakalářskou práci jsem si vybral tvorbu programu pro řešení úlohy lineárního programování simplexovou metodou.

Budu zde popisovat, jaké technologie jsem si vybral pro vytvoření tohoto programu a jakým způsobem jsem program vytvářel.

5.1 Použité technologie pro vývoj softwaru

Celý software je vyvíjen v programovacím jazyce Java a frameworkem JavaFX pro tvorbu desktopových aplikací.

5.1.1 Programovací jazyk Java

Java je vysokoúrovňový, objektově orientovaný, programovací jazyk a počítačová platforma, se kterou jako první přišla společnost Sun Microsystems v roce 1995. Od skromných začátků se vyvinul do podoby spolehlivé platformy, na niž je postaveny mnoho služeb a aplikací, a dnes pohání velkou část digitálního světa.

Je to multiplatformní programovací jazyk, který má programátorům umožnit napsat jeden kód a spouštět ho kdekoliv (WORA – write once, run anywhere), což znamená, že umožňuje uživatel napsat a zkompilovat jeden kód a spouštět ho na jakékoliv platformě bez jeho další rekompilace. [17]

Obrázek 6: Logo Javy [18]

5.1.1.1 Java Development Kit (JDK)

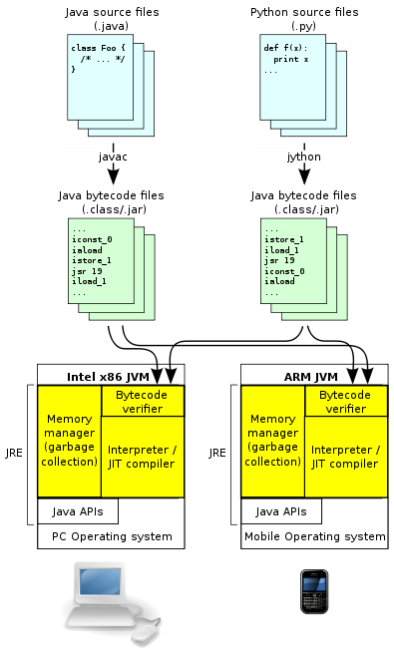
Java Development Kit neboli JDK, je produkt Java technologie od Oracle Corporation, který obsahuje soubor základních nástrojů pro vývoj aplikací pro platformu Java. JDK implementuje Java Language Specification (JLS) a Java Virtual Machine Specification (JVMS) a poskytuje Standard Edition (SE) aplikačního programového rozhraní (API) Java. [19]

5.1.1.2 Java Runtime Environment (JRE)

Java Runtime Environment neboli JRE, je software, kdy vyžadují Java programy pro svůj běh. Java Runtime Environment je vrstva, která komunikuje mezi programy, napsanými v Javě, a operačním systémem. Tváří se v podstatě jako takový překladač a zprostředkovatel. [20]

5.1.1.3 Java Virtual Machine (JVM)

Java Virtual Machine neboli JVM, je virtuální stroj, který umožňuje na počítači spouštět programy napsané v jazyce Java i programy napsané v jiných jazycích, které se také kompilují do Java bytecode. JVM funguje jako interpret mezi programovacím jazykem Java a základním hardwarem. Poskytuje běhové prostředí pro aplikace Java, které lze spustit na různých platformách a operačních systémech. [21]



Obrázek 7: Princip fungování JVM [22]

5.1.1.4 Proč jsem si vybral právě tento jazyk?

Byl to můj první jazyk, ve kterém jsem se naučil programovat. V Javy se snažím neustále vzdělávat a učit se novým technologiím. Od doby, kdy jsem se s Javou poprvé setkal, s ní intenzivně pracuji, ať už v soukromém, tak i v profesním prostředí.

Dalším důvodem je jeho univerzálnost. Že programy v Javě napsané lze spouštět na všech platformách jako je Linux, Windows, macOS, ale třeba i na mobilních technologiích.

5.1.2 JavaFX

JavaFX je moderní framework pro Javu určený k tvorbě jak desktopových aplikací, tak i bohatých webových aplikací. JavaFX. podporuje stolní počítače a webové prohlížeče v systémech Microsoft Windows, Linux (včetně Raspberry Pi) a macOS. Ale také mobilní zařízení s operačními systémy iOS a Android prostřednictvím služby Gluon Mobile. [23]

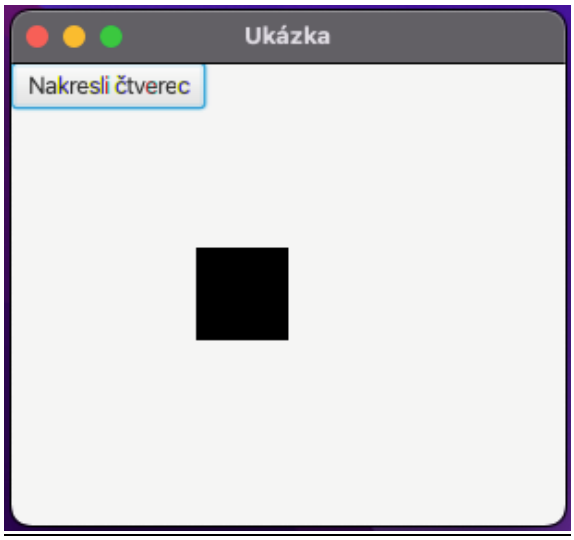
JavaFX je stále ve vývoji a s jednotlivou novou verzí přibývají nové prvky. Stále ji ale chybí důležité prvky pro vyvíjení aplikací na mobily. Například geolokace, orientace zařízení nebo natáčení kamerou. Při přehrávaní videa nebo zvuku mohou být použity pouze kodeky, které licencuje Oracle.

JavaFX v roce 2014 zcela nahradila zastaralý Swing, jako nástroj pro tvorbu GUI v Javě. [23]

Tvorba GUI je možná dvěma způsoby. Jako první možnost je použití software Scene Builder (popíšeme si později), díky kterému si může vývojář zpříjemnit práci a modelovat GUI způsobem „drag & drop“. Jako druhá možnost je psaní celého front-end.

```java
public class App extends Application {
    @Override
    public void start(Stage stage) {
        Button btn = new Button();
        btn.setText("Nakresli čtverec");

        Pane root = new Pane();
        root.getChildren().add(btn);

        btn.setOnAction((ActionEvent event) -> {
            Rectangle rec = new Rectangle( x: 100, y: 100, width: 50, height: 50);

            root.getChildren().add(rec);
        });

        Scene scene = new Scene(root, width: 300, height: 250);

        stage.setTitle("Ukázka");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

Obrázek 8: Příklad kódu v JavaFX [zdroj: vlastní]

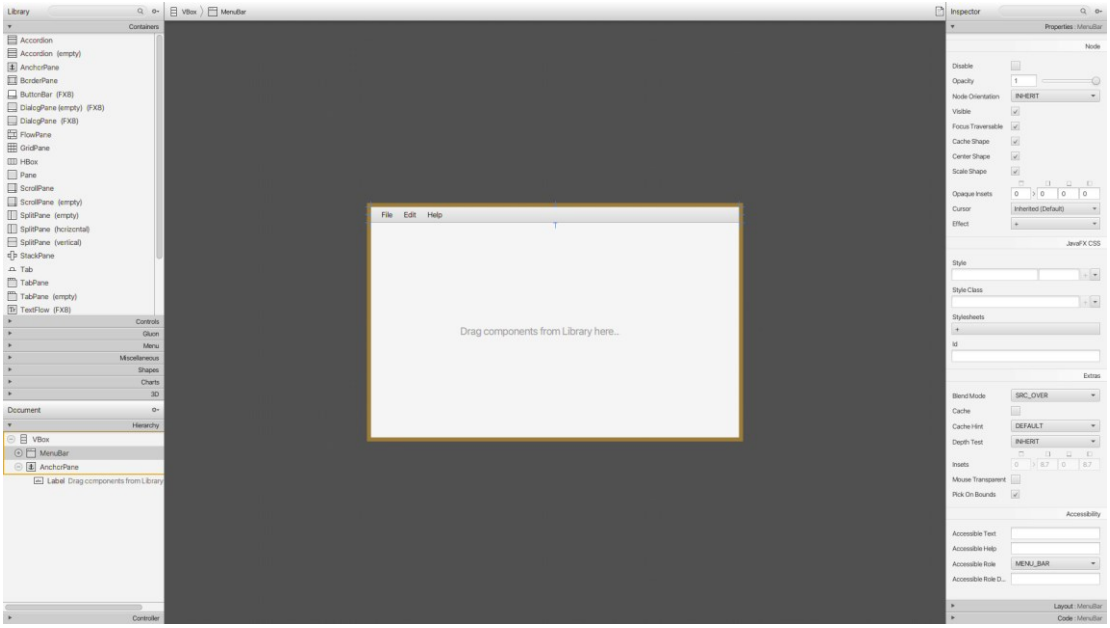


Obrázek 9: Příklad GUI napsaného v JavaFX [zdroj: vlastní]

5.1.3 Scene Builder

JavaFX Scene Builder je vizuální nástroj, který umožňuje uživateli jednoduše a rychle vytvářet grafické rozhraní bez psaní kódu. Uživatel může přetahovat jednotlivé komponenty pomocí funkce „drag & drop“, upravovat jejich vlastnosti, aplikovat CSS styly a FXML kód, který JavaFX využívá se automaticky generuje na pozadí. Výsledkem je soubor s příponou .fxml, který lze zkombinovat s projektem v jazyce Java navázáním uživatelského rozhraní na logiku aplikace. [24]

Obrázek 10: Ikona nástroje Scene Builder [25]



Obrázek 11: Grafické rozhraní softwaru Scene Builder [zdroj: vlastní]

5.1.4 IntelliJ IDEA

IntelliJ IDEA je vývojové prostředí napsané v jazyce Java for vývoj softwaru napsaného v jazycích Java, Kotlin, Groovy a dalších založených na JVM. Je Vyvíjen společností JetBrains (dříve známe jako IntelliJ) a je dostupná ve dvou verzích – Community (dostupná zdarma) a Ultimate (placená verze). Obě jsou použitelné komerčně, ovšem verze Community nepodporuje takové množství funkcí, co verze Ultimate. [25]

Základní rozdíly:

Tabulka 17: Základní rozdíly mezi verzemi Community a Ultimate [27]

Ultimate	Community
Java	Java
Groovy	Groovy
Kotlin	Kotlin
Python	Python
PHP	
SQL	
HTML	
XML, JSON, YAML	XML, JSON, YAML
JavaScript, TypeScript	
CSS, Sass, SCSS	
Spring	
Java EE	
Jakarta EE	
Hibernate, JPA	
JavaFX	JavaFX
Android	Android
Thymeleaf	
React	
Angular	
Node.js	
Maven	Maven
Gradle	Gradle

Git, GitHub, GitLab	Git, GitHub, GitLab
Subversion	Subversion
Docker	Docker

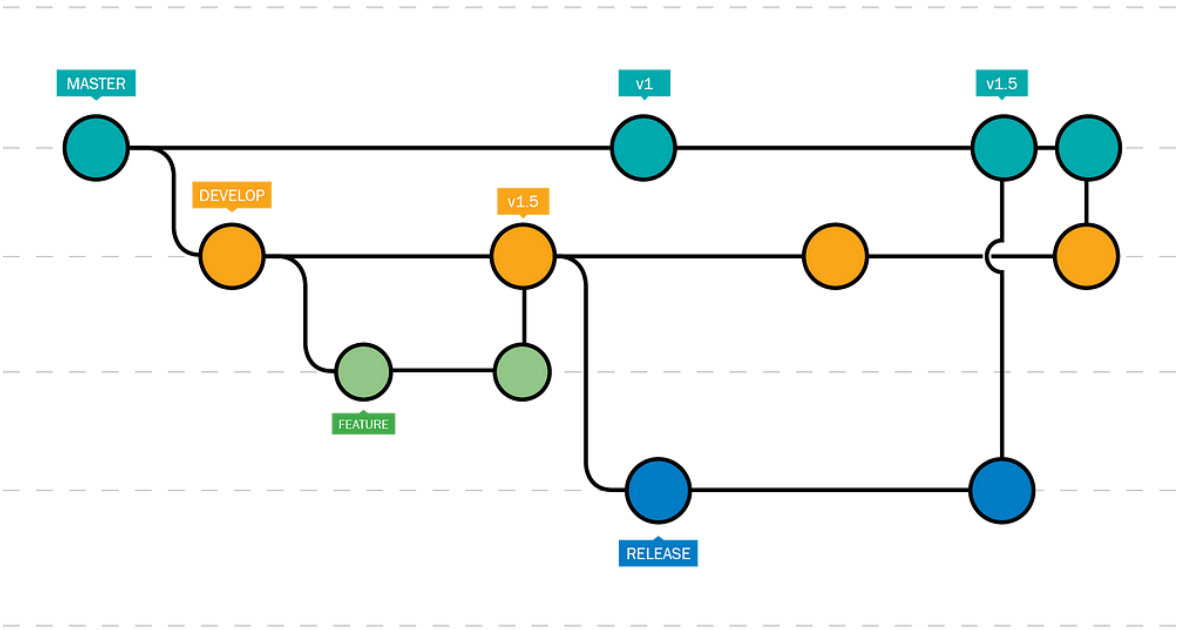
Pro vývoj aplikací v jazyce Java existuje spousta vývojových prostředí jako Eclipse, NetBeans, BlueJ nebo JDeveloper. Na střední škole jsme pracovali s IDE NetBeans, Na vysoké škole jsem potom vyzkoušel právě prostředí od JetBrains a tento nástroj mně vyhovuje nejvíce.

5.1.5 Git

Git je distribuovaný systém pro správu verzí, který sleduje změny v libovolné sadě počítačových souborů. standardně užitý pro koordinaci práce mezi programátory spolupracující na vývoji softwaru.

Mezi cíle systému Git patří rychlost, integrita dat a podpora distribuovaných nelineárních pracovních postupů (tisíce paralelních větví běžících na různých počítačích). [28]

Obrázek 12: Logo systému Git [29]



Obrázek 13: Princip fungování systému Git [30]

5.1.6 GitHub

GitHub je vývojová platforma, která umožňuje vývojářům vytvářet, uchovávat, spravovat a sdílet jejich kód. GitHub využívá systému Git. Je ve směs určen pro správu open source projektů. Je to světově nejrozšířenější platforma pro správu kódu. [31]

Obrázek 14: Logo softwaru GitHub [32]

5.2 Vývoj programu

Můj prvotní plán byl postavit aplikace na čisté Javy. Vytvořil jsem si tedy projekt a začal hned přemýšlet nad tím, jak simplexovou metodu převést do kódu. Tato část byla na celém programu asi to nejtěžší, ale po spoustě hodinách jsem našel řešení, tak toto vyřešit. V programu je to řešené tak, že si vypočítám hodnotu alpha podílem hodnoty, která se nachází na aktuálním optimalizovaném řádku a v klíčovém sloupci, a klíčové

hodnoty. Optimální hodnotu pak získám rozdílem hodnoty, kterou optimalizuju, a násobkem alphy a klíčové hodnoty.

Takže si teď pojďme popsat jak jednofázovou, tak dvoufázovou metodu.

5.2.1 Jednofázová metoda pohledem kódu

Pokud se budeme bavit o jednofázové metodě, tak ji rozdělím do dvou částí – maximalizace a minimalizace.

Maximalizace

Při maximalizaci si program vytvoří dvourozměrné pole, které nám později bude sloužit jako tabulka, a do proměnné *m* uloží počet omezení, do proměnné *n* počet proměnných v účelové funkci. Následně si inicializuje tabulku tak, že všechny omezení postupně ukládá do dvourozměrného pole a následně zde uloží i negované hodnoty v účelové funkci.

```
private static void solveOnePhaseProblem(double[] c, double[][] A, double[] b, HashMap<ResultHashMapIdentifier, double[]> results) {
    System.out.println(SolverType.ONEPHASE_SOLVER);
    final int m = A.length;
    final int n = c.length;
    final double[][] table = new double[m + 1][n + m + 1];

    //initialize table
    for (int i = 0; i < m; i++) {
        System.arraycopy(A[i], srcPos: 0, table[i], destPos: 0, n);
        table[i][n + i] = 1;
        table[i][n + m] = b[i];
    }
    for (int i = 0; i < n; i++) {
        table[m][i] = -c[i];
    }

    simplex(table, results, m, n);
}
```

Obrázek 15: Ukázka metody pro řešení jednofázové úlohy LP [zdroj: vlastní]

Minimalizace

Minimalizaci jsem se rozhodl řešit vytvořením duální úlohy k primární úloze a poté program pokračuje stejně jako u jednofázové úlohy.

```
private static void minimize(double[] c, double[][] A, double[] b, HashMap<ResultHashMapIdentifier, double[]> results) {
    System.out.println(SolverType.MINIMIZATION);
    double[] transposedC = new double[b.length];
    double[][] transposedA = new double[A[0].length][A.length];
    double[] transposedB = new double[c.length];

    //transpose c to b
    System.arraycopy(c, srcPos: 0, transposedB, destPos: 0, c.length);

    //transpose b to c
    System.arraycopy(b, srcPos: 0, transposedC, destPos: 0, b.length);

    //transpose A to A
    for (int i = 0; i < A.length; i++) {
        for (int j = 0; j < A[0].length; j++) {
            transposedA[j][i] = A[i][j];
        }
    }

    solveOnePhaseProblem(transposedC, transposedA, transposedB, results);
}
```

Obrázek 16: Ukázka vytvoření duální úlohy v kódu [zdroj: vlastní]

Následně voláme metodu *simplex*, která již řeší simplexovou tabulku.

Tato metoda jako první zkontroluje, zda řešení je již optimální. Pokud ano, tak celý proces řešení končí. Pokud však ne, tak přechází na zjištění klíčového sloupce a následně ověří, zda úloha není nekonečná. Pokud není, tak přechází vyhledání klíčového řádku.

Po zjištění klíčového řádku přechází k optimalizaci řešení. Tento proces jsme si již popsali výše.

```
/*
Klicova hodnota je ulozena v table[p][q]
Nova optimalni hodnota je zjistovana jako ze
 - zjistime alphu, coz je podil hodnoty, na aktualnim radku a v klicovem sloupci, a klicove hodnoty
 - optimalni hodnotu pak ziskame rozdilem hodnoty, kterou optimalizujeme, a nasobkem alphy a klicove hodnoty
 */
for (int i = 0; i <= m; i++) {
    if (i != p) {
        double alpha = table[i][q] / table[p][q];
        for (int j = 0; j <= n + m; j++) {
            table[i][j] =  table[i][j] - (alpha * table[p][j]);
        }
    }
}

//optimalizovani klicoveho radku
double pivot = table[p][q];
for (int j = 0; j <= n + m; j++) {
    table[p][j] = table[p][j] / pivot;
}
```

Obrázek 17: Ukázka optimalizace v kódu [zdroj: vlastní]

Tento celý proces se opakuje do doby, kdy program zjistí, že řešení je optimální. V tu chvíli se zpracuje výsledek a vypíše se nám.

5.2.2 Dvoufázová metoda pohledem kódu

U dvoufázové metody je potřeba si ještě, kromě proměnných *m*, *n* a dvourozměrného pole, vytvořit i proměnnou, která nám bude držet počet pomocných proměnných (v našem případě se bude jmenovat *auxiliaryVarsCount*.

Program vytvoří dvourozměrné pole, které bude sloužit jako tabulka a u které je ale potřeba rozlišit o jaké znaménko v omezení je jedná. Pokud omezení je se znaménkem ≤, tak se do tabulky přidá *1* jako přídatná proměnná. Pokud je znaménko omezení ≥, do tabulky se přidá *-1* jako přídatná proměnná a *1* jako pomocná proměnná. Pokud je ale znaménko =, tak se do tabulky přidá *1* jako pomocná proměnná.

```
//initialize table
for (int i = 0; i < m; i++) {
    System.arraycopy(A[i], srcPos: 0, table[i], destPos: 0, n);
    if (signs[i].equals(LOWER_OR_EQUALS)) {
        table[i][n + i] = 1;
    } else if (signs[i].equals(GREATER_OR_EQUALS)) {
        table[i][n + i] = -1;
        table[i][n + auxiliaryVarsCount + i] = 1;
    } else if (signs[i].equals(EQUALS)) {
        table[i][n + auxiliaryVarsCount + i] = 1;
    }

    //put right side of each constraint into table
    table[i][table[i].length - 1] = b[i];
}
```

Obrázek 18: Vytvoření tabulky u dvoufázové metody v kódu [zdroj: vlastní]

Následně se do tabulky přidaní negované hodnoty z účelové funkce a program přejde vytvoření pomocné účelové funkce.

Dále si do pole *constraintsWithAuxiliaryVar* uloží všechny omezení s pomocnou proměnnou. Poté jednotlivé prvky na jednotlivých indexech mezi omezeními sčítá a ukládá do pole s finální pomocnou účelovou funkcí. Toto pole potom uloží do tabulky.

```
//find count of auxiliary variables
double[][] constraintsWithAuxiliaryVar = new double[auxiliaryVarsCount][];
int counter = 0;
for (int i = 0; i < table.length - 2; i++) {
    if (table[i][n + m - 1 + i] == 1) {
        constraintsWithAuxiliaryVar[counter] = table[i];
        counter++;
    }
}

//create auxiliary function a put it into table
double[] auxiliaryFunc = new double[n + m];
double auxiliaryFuncValue = 0;
for (int i = 0; i < constraintsWithAuxiliaryVar.length; i++) {
    for (int j = 0; j < auxiliaryFunc.length; j++) {
        auxiliaryFunc[j] += constraintsWithAuxiliaryVar[i][j];
    }
    auxiliaryFuncValue += constraintsWithAuxiliaryVar[i][table[0].length - 1];
}
System.arraycopy(auxiliaryFunc, srcPos: 0, table[table.length - 1], destPos: 0, length: n + m);
table[table.length - 1][table[0].length - 1] = auxiliaryFuncValue;
```

Obrázek 19: Ukázka vytvoření pomocné účelové funkce v kódu [zdroj: vlastní]

Následně program volá funkci simplex, pro řešení pomocné účelové funkce.

Jakmile je řešení pomocné účelové funkce optimální, program vytvoří novou tabulku, která již neobsahuje pomocnou účelovou funkci, ani pomocné proměnné a opět volá funkci simplex pro započatí řešení účelové funkce.

Jakmile je řešení optimální, tak program zpracuje výsledek vypíše nám ho.

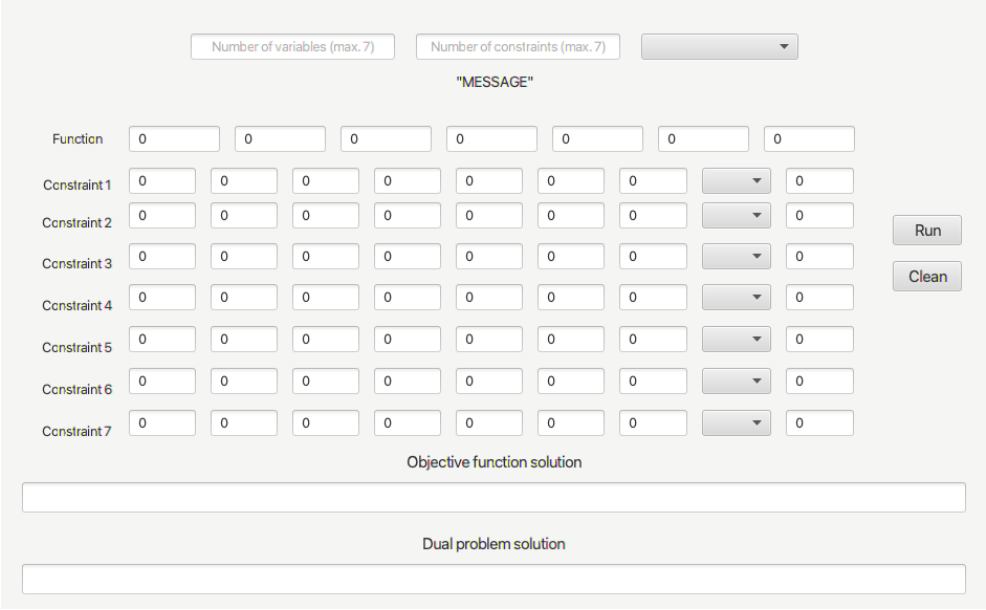
5.2.3 Grafické rozhraní

Pokud by program zůstal v takovém řešení, byla by to pouze konzolová aplikace, což jsem později přehodnotil, protože takové řešení není moc user-friendly. Tudíž jsem se rozhodl vytvořit k této logice i grafické rozhraní.

Rozhraní jsem tvořil v programu Scene Builder, který jsem zmiňoval výše. S tímto softwarem jsem pracoval jak na střední škole, tak i v zaměstnání, takže to byl pro mě nejrychlejší a nejjednodušší způsob, jak vytvořit GUI.

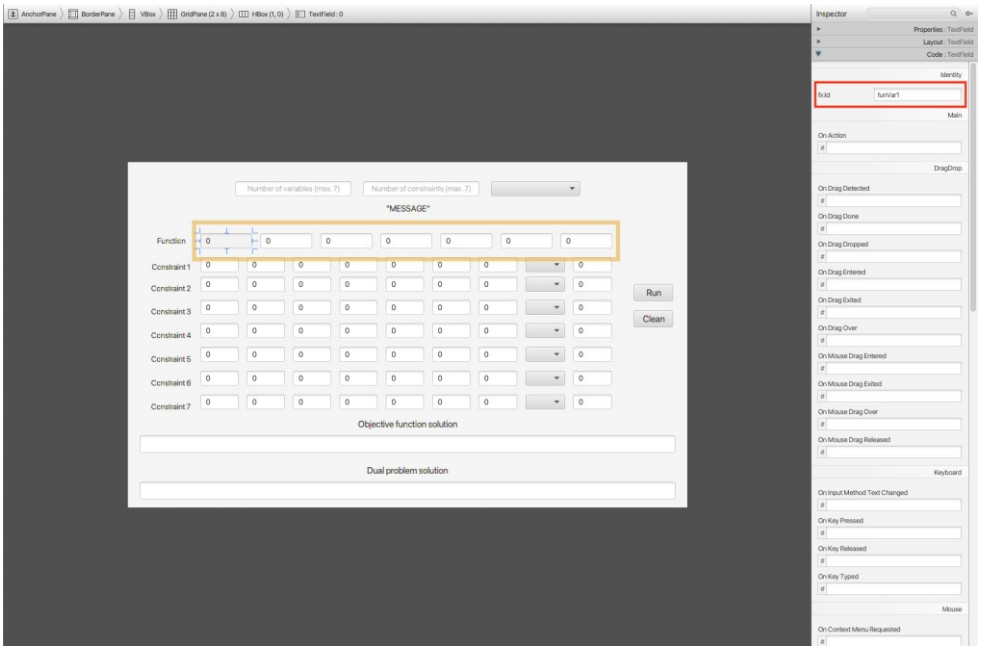
Grafické rozhraní je vytvořeno tak, aby bylo pro uživatele, co poměrně nejpřehlednější, ale zároveň se dalo zadávat možná co největší počet dat. Uživatel může zadávat maximálně 7 proměnných a 7 omezení.

Graficky je zpracováno tak, jak můžeme vidět na Obrázku 20.



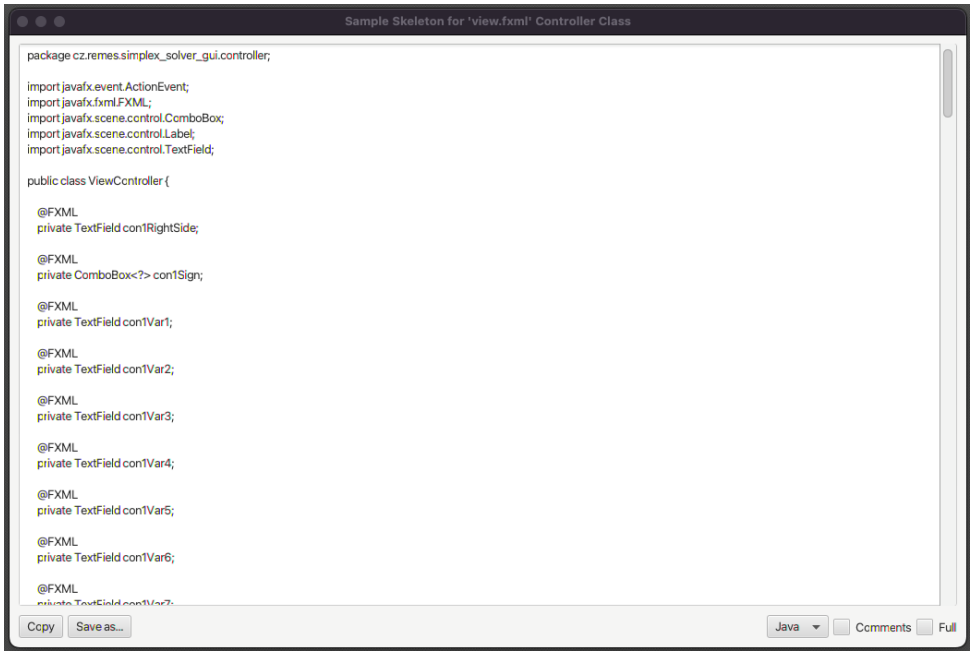
Obrázek 20: Grafické rozhraní [zdroj: vlastní]

Každý element má určité „fx:id“, což je identifikátor, se kterým následně pracuje back-end. Pro příklad, textové pole pro hodnotu první proměnné v účelové funkci má identifikátor „funVar1“.



Obrázek 21: Identifikátory elementů [zdroj: vlastní]

Scene Builder má jednu velkou výhodu, která usnadní práci, a to je funkce „Sample Skeleton for ‚view.fxml' Controller Class". Tato funkce vygeneruje vše potřebné pro spárování tohoto GUI s back-endem. Najdeme ji ve View → Show Sample Controller Skeleton.



Obrázek 22: Funkce Show Sample Controller Skeleton

V kódu to potom vypadá jako na Obrázku 23.

```
public class ViewController implements Initializable {
    3 usages
    @FXML
    public TextField con1RightSide;

    5 usages
    @FXML
    public ComboBox<String> con1Sign;

    3 usages
    @FXML
    public TextField con1Var1;

    3 usages
    @FXML
    public TextField con1Var2;

    3 usages
    @FXML
    public TextField con1Var3;

    3 usages
    @FXML
```

Obrázek 23: Identifikátory elementů v kódu

Každý controller obsahuje metodu *initialize*, která se spouští okamžitě po otevření určitého okna a nese v sobě logiku, kterou je zapotřebí provést, když se okno otevře. Já potřebuji do určitých elementů přidat data hned po otevření okna, např. do Choice Boxu hned na začátku potřebuji přidat volby, zda program bude počítat maximalizaci nebo minimalizaci. Do ostatních Choice Boxu potřebuji přidat znaménka, které si uživatel zvolí pro každé omezení.

```
@Override
public void initialize(URL location, ResourceBundle resources) {
    String[] signs = {"<=", ">=", "="};
    optimizationCb.getItems().addAll(OptimizationType.MAXIMIZE, OptimizationType.MINIMIZE);
    optimizationCb.setValue(OptimizationType.MAXIMIZE);

    con1Sign.getItems().addAll(signs);
    con1Sign.setValue(signs[0]);

    con2Sign.getItems().addAll(signs);
    con2Sign.setValue(signs[0]);

    con3Sign.getItems().addAll(signs);
    con3Sign.setValue(signs[0]);

    con4Sign.getItems().addAll(signs);
    con4Sign.setValue(signs[0]);

    con5Sign.getItems().addAll(signs);
    con5Sign.setValue(signs[0]);

    con6Sign.getItems().addAll(signs);
    con6Sign.setValue(signs[0]);

    con7Sign.getItems().addAll(signs);
    con7Sign.setValue(signs[0]);
}
```

Obrázek 24: Funkce *initialize* [zdroj: vlastní]

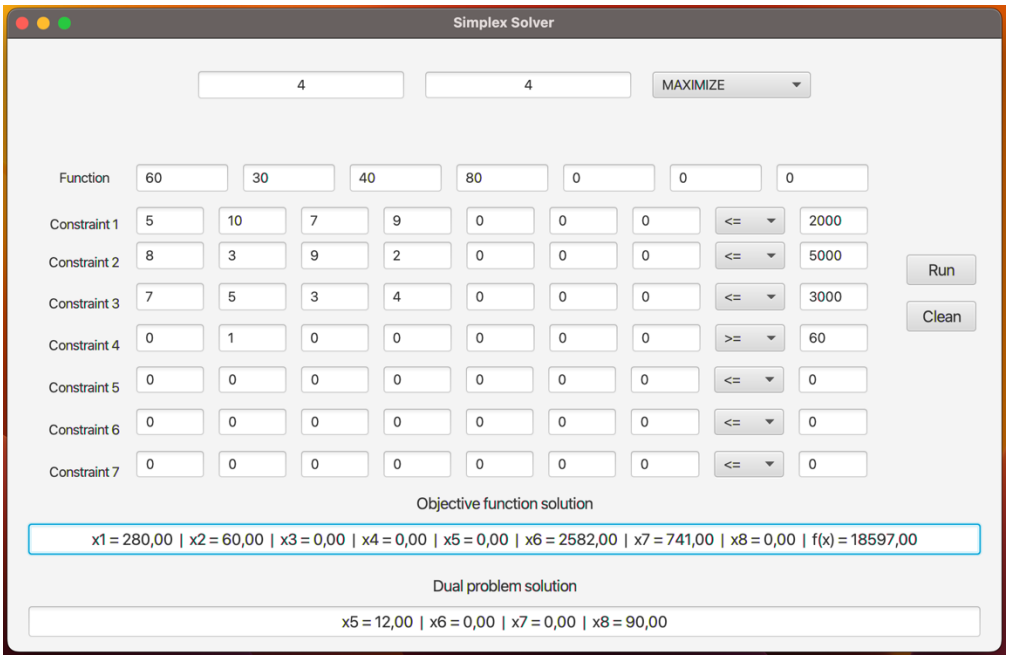
Další podstatná část controlleru je metoda *process*, díky které se provádí celý výpočet.

Celé GUI si na začátku vyčistíme, abychom případně nepracovali s nechtěnými daty. Následně se vytvoří jednotlivé proměnné a pole, se kterými budeme později pracovat. Za pomoci metody si zpracujeme data z grafického rozhraní. Do polí si uložíme data z GUI.

A voláme tu metodu *process* ze třídy *SimplexSolver*, která nám provádí výpočet. Následně je výsledek metodou *processResultToGui* vypsán uživateli do grafického rozhrání.

V případě, že by tento process na spadnul do výjimky na určitém problému, tak se uživateli tento problém také vypíše do GUI.

Další důležitá metoda, která je úzce svázána s uživatelským rozhraním je metoda *clean*, která uživateli vynuluje všechny hodnoty, doposud zadané.



Obrázek 25: Vyplněná tabulka s výsledkem

5.2.4 Použití programu Simplex Solver

Za prvé je třeba si ujasnit, kolik má uživatel proměnných v účelové funkci a kolik omezení. Číslo počtu proměnných uživatel do textové pole „Number of variables“. Číslo počtu omezení zadá do textového pole „Number of constraints“. Dále vybere, zda chce provádět *maximalizaci* či *minimalizaci*.

Do řádku „Function“ uživatel zadává postupně hodnoty jednotlivých proměnných $x_1, \dots, x_7$. V případě, že uživatel má účelovou funkci kratší než 7 proměnných, zapíše do nevyužitých textových polí *0*.

Do řádků Constraint zadává hodnoty každé proměnné u jednotlivých omezení. Zvolí znaménko, které náleží k jednotlivým omezením a zadá hodnotu pravé strany u omezení. Pokud nevyužije všechny textové pole z důvodu, že jeho omezení obsahují méně, než 7 proměnných, opět do nich zapíše *0*.

Poté může kliknout na tlačítko Run a program mu provede výpočet.

Pokud by během výpočtu nastala chyba, program tuto chybu zobrazí v místě nad hodnotami účelové funkce.

6 TVORBA STUDIJNÍCH OPOR

Další praktickou části bylo vytvoření studijních opor. Cílem byl vytvořit dokument několik řešených příkladů. Příklady jsem volil takové, aby si čtenář byl schopen propočítat všechny možné kombinace a typy příkladů, např. jednofázová a dvoufázová úloha, nebo příklady, které mají jedno optimální řešení, ale mohou mít i nekonečně mnoho řešení.

ZÁVĚR

V závěru je důležité poukázat na to, že lineární programování představuje mocný nástroj pro řešení optimalizačních problémů v různých oblastech lidské činnosti. Jeho aplikace sahají od výroby přes finance a dopravu, až po logistiku.

Je nezbytné si uvědomit, že lineární programování je pouze teoretický koncept, ale že má i v reálném světě své zastoupení. Využíván je k řešení praktických problémů a umožňuje organizacím dosahovat efektivnějších výsledků.

SEZNAM POUŽITÉ LITERATURY

[1] KREISEL, Tomáš. *Lineární programování a jeho využití ve vybraných úlohách.* Online, Bakalářská práce, vedoucí Mgr. Jana Řezníčková, Ph.D. Nám. T. G. Masaryka 5555, 760 01 Zlín, Česká republika: Univerzita Tomáše Bati, 2019. Dostupné z: https://digilib.k.utb.cz/bitstream/handle/10563/44568/kreisel_2019_dp.pdf. [cit. 2024-05-07].

[2] PHILLIPS, Almarin. The Tableau Économique as a Simple Leontief Model. Online. *The Quarterly Journal of Economics.* 1955, roč. 69, č. 1, s. 137-144. ISSN 00335533. Dostupné z: https://www.jstor.org/stable/1884854. [cit. 2024-05-07].

[3] DUFFIN, Richard J. On fourier's analysis of linear inequality systems. Online. In: BALINSKI, M. L. (ed.). *Pivoting and Extension.* Mathematical Programming Studies. Berlin, Heidelberg: Springer Berlin Heidelberg, 1974, s. 71-95. ISBN 978-3-642-00756-9. Dostupné z: https://doi.org/10.1007/BFb0121242. [cit. 2024-05-07].

[4] SCHRIJVER, Alexander. On the history of the transportation and maximum flow problems. Online. *Mathematical Programming.* 2002, roč. 91, č. 3, s. 437-445. ISSN 0025-5610. Dostupné z: https://doi.org/10.1007/s101070100259. [cit. 2024-05-07].

[5] VERSHIK, Anatoly. *L.V.Kantorovich and Linear Programming.* PDF. 2007.

[6] GARILLE, Susan Garner a GASS, Saul I. Stigler's Diet Problem Revisited. Online. *Operations Research.* 2001, roč. 49, č. 1, s. 1-13. ISSN 0030-364X. Dostupné z: https://doi.org/10.1287/opre.49.1.1.11187. [cit. 2024-05-07].

[7] PLESNÍK, Ján; DUPAČOVÁ, Jitka a VLACH, Milan. *Lineárne programovanie.* Bratislava: Alfa, 1990. ISBN 80-05-00679-9.

[8] ZUZAŇÁKOVÁ, Jana. Lineární programování v praxi. Online, Diplomová práce, vedoucí Mgr. Petr Zemánek, Ph.D. Žerotínovo nám. 617/9, 601 77 Brno: Masarykova univerzita, 2020. Dostupné z: https://is.muni.cz/th/i23nz/DP_Zuzanakova.pdf. [cit. 2024-05-07].

[9] LAGOVÁ, Milada a JABLONSKÝ, Josef. *Lineární modely.* Vyd. 3. Praha: Oeconomica, 2014. ISBN 978-80-245-2020-9.

[10] FIŠNAROVÁ, Simona. *Základy lineárního programování.* Online. Zemědělská 1, 613 00 Brno, 2012. Dostupné také z: http://user.mendelu.cz/fisnarov/imt/prednasky/lp.pdf. Výukový materiál.

[11] VČELAŘ, František. *Celočíselné lineární programování.* Bakalářská práce. Nám. T. G. Masaryka 5555, 760 01 Zlín, Česká republika: Univerzita Tomáše Bati, 2018.

[12] PECHERKOVÁ, Pavla; JOZOVÁ, Šárka a NAGY, Ivan. *Lineární programování I.* PDF. 2020. Dostupné také z: https://www.fd.cvut.cz/personal/nagyivan/LinPrg1/LP1Skripta.pdf. Výukové materiály.

[13] BRÁZDOVÁ, Markéta. *Řešené úlohy lineárního programování.* Vyd. 1. Pardubice: Univerzita Pardubice, 2011. ISBN 978-80-7395-361-4.

[14] REMEŠ, Nikolas. *[Jaké jsou metody a řešení lineárního programování?].* Online. In: ChatGPT. 3.5, 6. dubna 2024. Dostupné z: OpenAI, http://chat.openai.com. [cit. 2024-05-07].

[15] REMEŠ, Nikolas. *[Co víš o simplexově metodě?].* Online. In: ChatGPT. 3.5, 8. dubna 2024. Dostupné z: OpenAI, http://chat.openai.com. [cit. 2024-05-07].

[16] JABLONSKÝ, Josef. *Operační výzkum: kvantitativní modely pro ekonomické rozhodování.* Praha: Professional Publishing, 2002. ISBN 80-86419-42-8.

[17] TRONÍČEK, Zdeněk. *Učebnice jazyka Java.* Online. Programovací učebnice. 2011. Dostupné z: http://programovaci-ucebnice.g6.cz/ucebnice/UcebniceJazykaJava/0_Predmluva.xhtml. [cit. 2024-05-08].

[18] ANDERSON, Mark. *Java programming language logo.* Online. In: Https://en.wikipedia.org/wiki/Main_Page. 2001. Dostupné z: https://upload.wikimedia.org/wikipedia/en/thumb/3/30/Java_programming_language_logo.svg/800px-Java_programming_language_logo.svg.png. [cit. 2024-05-08].

[19] *Java Development Kit.* Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001. Dostupné z: https://en.wikipedia.org/wiki/Java_Development_Kit. [cit. 2024-05-08].

[20] *What is Java Runtime Environment.* Online. In: AMAZON. AWS. C2024. Dostupné z: https://aws.amazon.com/what-is/java-runtime-environment/. [cit. 2024-05-08].

[21] *What is Java Runtime Environment.* Online. In: LENOVO. Lenovo. C2024. Dostupné z: https://www.lenovo.com/us/en/glossary/jvm/. [cit. 2024-05-08].

[22] *Znázornění architektury JVM.* Online. In: Wikipedie. 2001. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/3/3a/Java_virtual_machine_architecture.svg. [cit. 2024-05-08].

[23] *JavaFX. Online.* In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001. Dostupné z: https://en.wikipedia.org/wiki/JavaFX. [cit. 2024-05-08].

[24] *JavaFX Scene Builder.* Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001. Dostupné z: https://cs.wikipedia.org/wiki/JavaFX_Scene_Builder. [cit. 2024-05-08].

[25] Scene Builder Logo. Online. In: GluonHQ. C2024. Dostupné z: https://gluonhq.com/wp-content/uploads/2015/02/SceneBuilderLogo.png. [cit. 2024-05-08].

[26] *IntelliJ IDEA.* Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001. Dostupné z: https://en.wikipedia.org/wiki/IntelliJ_IDEA#cite_note-3. [cit. 2024-05-08].

[27] *IntelliJ IDEA Ultimate vs IntelliJ IDEA Community Edition.* Online. In: JetBrains. C2000. Dostupné z: https://www.jetbrains.com/products/compare/?product=idea&product=idea-ce. [cit. 2024-05-08].

[28] *Git.* Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001. Dostupné z: https://en.wikipedia.org/wiki/Git#cite_note-kernel_SCM_saga-11. [cit. 2024-05-08].

[29] *Git logo.* Online. In: Git --everything-is-local. C2005. Dostupné z: http://git-scm.com/downloads/logos. [cit. 2024-05-08].

[30] RAZA, sam. Git workflow with Release branch. Online. In: Medium. 2017. Dostupné z: https://samraza.medium.com/git-workflow-strategy-37bd5c242133. [cit. 2024-05-08].

[31] GitHub. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001. Dostupné z: https://en.wikipedia.org/wiki/GitHub. [cit. 2024-05-08].

[32] GitHub logo. Online. In: GitHub. C2024. Dostupné z: https://github.com/logos. [cit. 2024-05-08].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

LP	Lineární programování
Max.	Maximální
GUI	Graphical User Interface – grafické uživatelské rozhraní
Pozn.	Poznámka
Tj.	To je
Popř.	Popřípadě
JDK	Java Development Kit
JVM	Java Virtual Machine
JRE	Java Runtime Environment
CSS	Cascading Style Sheets

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: MATEMATICKÝ PROGRAM PRO ŘEŠENÍ ÚLOH LINEÁRNÍHO PROGRAMOVÁNÍ POD NÁZVEM „SIMPLEX SOLVER“

Příloha obsahuje zdrojový kód výše zmíněného programu.

PŘÍLOHA II: STUDIJNÍ OPORA

Příloha obsahuje dokument, ve kterém jsou řešeny příklady úloh lineárního programování.