

Celočíselné programování – metoda větví a mezí

Integer programming – branch and bound method

Adam Krhovják

Bakalářská práce
2011



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Adam KRHOVJÁK**
Osobní číslo: **A08150**
Studijní program: **B 3902 Inženýrská informatika**
Studijní obor: **Informační a řídicí technologie**

Téma práce: **Celočíselné programování – metoda větví a mezí**

Zásady pro vypracování:

1. Nastudujte a popište úlohu a metody lineárního programování, simplexovou tabulku, duální úlohu.
2. Nastudujte a popište úlohu a metody celočíselného lineárního programování.
3. Podrobně popište algoritmus větví a mezí (branch and bound).
4. Nastudujte a stručně popište programovací jazyk Java a vybrané programovací prostředí.
5. Vymyslete a vyřešte několik úloh celočíselného lineárního programování metodou větví a mezí.
6. Vytvořte Android aplikaci, která bude implementovat metodu větví a mezí.

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. LAŠČIAK, Adam, et al. Optimálne Programovanie. Bratislava : SNTL, 1983. 565 s.
2. HANZÁLEK, Zdeněk; ŠŮCHA, Přemysl. Celočíselné lineární programování. 24.8.2007 [cit. 2011-01-24]. Dostupné z WWW: http://edux2.felk.cvut.cz/modules/edux/get_file_from_dms.php?function=view&FileID=1734&s
3. SLAVÍČEK, Ondřej. Aplikace celočíselného programování v ekonomii. Olomouc, 2008. 73 s. Diplomová práce. Univerzita Palackého v Olomouci, Přírodovědná fakulta, Katedra matematické analýzy a aplikací matematiky. Vedoucí práce RNDr. Jitka Machalová, Ph.D. Dostupné z WWW: <http://mant.upol.cz/soubory/OdevzdanePrace/m08-03-os.pdf>.
4. SVRČEK, Jaroslav. Lineární programování v úlohách. 2. vydání. Olomouc : Vydavatelství Univerzity Palackého v Olomouci, 2003. 104 s. ISBN 80-244-0705-1.
5. ŠŮCHA, Přemysl. Celočíselné lineární programování. 11.3.2004 [cit. 2011-01-24]. Dostupné z WWW: <http://dce.felk.cvut.cz/sucha/articles/ilp.pdf>.
6. TRICK, Michael. Branch and Bound. 14.6.1998 [cit. 2011-01-24]. Dostupné z WWW: <http://mat.gsia.cmu.edu/orclass/integer/node13.html>.
7. ZAKHOUR, Sharon, et al. Java 6 : Výukový kurz. první. Brno : Computer Press, 2007. 525 s. ISBN 978-80-251-157.

Vedoucí bakalářské práce:

Ing. Libor Pekař

Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce:

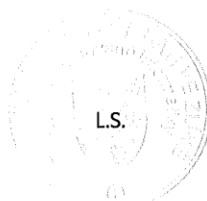
25. února 2011

Termín odevzdání bakalářské práce:

7. června 2011

Ve Zlíně dne 25. února 2011


prof. Ing. Vladimír Vašek, CSc.
děkan




prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Bakalářská práce se zabývá odvětvím optimalizace - celočíselným programováním. Těžištěm práce je problematika modelování s celočíselnými proměnnými, která má opodstatnění v mnoha praktických aplikacích. Celá teorie lineárního programování je bohatě prokládána konkrétními příklady, které umožňují snadnější pochopení dané problematiky. Dále je uveden rozsáhlý popis a algoritmus metody větví a mezí, jejíž matematický aparát je navíc podpořen mobilní aplikací běžící na platformě Android, která byla vytvořena v prostředí Eclipse za pomoci programovacího jazyka Java.

Klíčová slova: lineární programování, celočíselné programování, metoda větví a mezí, Java, Android

ABSTRACT

Bachelor thesis is concerned with the branch of optimization – an integer linear programming. The main focus of this thesis is a problem of modeling with integer variables which is useful in many practical applications. The whole theory of linear programming is generously interspaced with specific examples for easier understanding of the issue. This is followed by extensive description of branch and bound algorithm which is supported by mobile application running on Android platform and created in Eclipse using Java programming language.

Keywords: linear programming, integer programming, branch and bound method, Java, Android

Na tomto místě bych rád poděkoval vedoucímu své práce, Ing. Liboru Pekařovi, za jeho nekonečnou trpělivost a za cenné rady, bez kterých by tato práce jen těžko vznikla.

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 a jeden výtisk bude uložen u vedoucího práce;
- 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 s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- 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

.....
podpis diplomanta

OBSAH

ÚVOD	8
I TEORETICKÁ ČÁST	9
1 TEORIE LINEÁRNÍHO PROGRAMOVÁNÍ	10
1.1 ÚLOHA LINEÁRNÍHO PROGRAMOVÁNÍ VE STANDARDNÍM TVARU	10
1.2 PRIMÁRNÍ A DUÁLNÍ ÚLOHA LINEÁRNÍHO PROGRAMOVÁNÍ	12
1.3 KANONICKÝ TVAR ÚLOHY LINEÁRNÍHO PROGRAMOVÁNÍ	15
1.4 PRIMÁRNÍ ALGORITMUS SIMPLEXOVÉ METODY	16
1.5 DUÁLNÍ ALGORITMUS SIMPLEXOVÉ METODY	21
2 TEORIE CELOČÍSELNÉHO PROGRAMOVÁNÍ.....	25
2.1 METODY ŘEŠENÍ.....	25
2.2 ALGORITMUS METODY VĚTVÍ A MEZÍ	30
2.3 ŘEŠENÉ PŘÍKLADY METODOU VĚTVÍ A MEZÍ.....	31
3 TECHNOLOGIE JAVA	40
3.1 PROGRAMOVACÍ JAZYK JAVA	40
3.2 PLATFORMA JAVA	40
3.3 ECLIPSE IDE A PLATFORMA ANDROID.....	40
II PRAKTICKÁ ČÁST.....	42
4 VÝVOJ NA PLATFORMĚ ANDROID	43
4.1 STARTOVACÍ BALÍČEK SDK A INSTALACE ADT	43
4.2 KONFIGURACE A AKTUALIZACE ADT.....	44
4.3 PŘIDÁNÍ PLATFORMEM A JINÝCH KOMPONENT.....	44
5 MOBILNÍ APLIKACE CELOČÍSELNÉHO PROGRAMOVÁNÍ.....	46
5.1 POPIS APLIKACE BRANCHANDBOUND.APK.....	46
5.2 VÝPOČET PŘÍKLADU NA HTC WILDFIRE	50
5.2.1 Zadání příkladu a matematická formulace.....	51
5.2.2 Aplikace problému na emulátoru	52
ZÁVĚR	55
ZÁVĚR V ANGLIČTINĚ.....	56
SEZNAM POUŽITÉ LITERATURY	57
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	58
SEZNAM OBRÁZKŮ	59
SEZNAM TABULEK	60
SEZNAM PŘÍLOH	61

ÚVOD

Omezení úplnosti se může zdát poměrně neškodné, ale ve skutečnosti má dalekosáhlé důsledky. Význam modelování s celočíselnými proměnnými se ukázal být přínosný nejen na oblast celočíselné produkce, ale spolu s nimi můžeme správně modelovat fixní náklady, nejrozumnější logické požadavky a mnohé další aspekty.

Zvažme například výrobu televizních přijímačů. Řekněme, že model lineárního programování by nám mohl poskytnout výrobní plán 205,7 sad za týden. Za této situace by většina lidí neměla problém uvést, že produkce by měla být 205 sad za týden nebo dokonce říct: „Zhruba 200 sad za týden.“ Na druhou stranu předpokládejme, že jsme si koupili skladiště, abychom takto dokončené zboží mohli uskladnit a dále, že velikost skladiště přímo odpovídá velikosti vyrobené sady. Pak na základě modelu, který jsme navrhli, bychom koupili 0,7 skladiště v jedné lokaci, 0,6 v další a tak dále. Pochopitelně, že množství zakoupených skladišť koresponduje s celočíselnými hodnotami a rádi bychom tedy, aby náš model odrážel tuto skutečnost. Účelem těchto statí je popsat některé techniky řešící výše zmíněný problém a stejně tak i jejich možná úskalí a blíže ukázat metodu, na níž je založena zajímavá celočíselná aplikace. Nejdříve se však odrazme od základní terminologie.

I. TEORETICKÁ ČÁST

1 TEORIE LINEÁRNÍHO PROGRAMOVÁNÍ

1.1 Úloha lineárního programování ve standardním tvaru

Úlohu lineárního programování lze formulovat následujícím způsobem
maximalizovat

$$z(\mathbf{x}) = c_1x_1 + c_2x_2 + \dots + c_nx_n \quad (1.1)$$

za podmínek

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &+ \dots + a_{1n}x_n = b_1 \\ &\vdots \\ a_{r1}x_1 + a_{r2}x_2 &+ \dots + a_{rn}x_n = b_r \end{aligned} \quad (1.2)$$

$$\begin{aligned} a_{r+1,1}x_1 + a_{r+1,2}x_2 &+ \dots + a_{r+1,n}x_n \leq b_{r+1} \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 &+ \dots + a_{mn}x_n \leq b_m \\ x_j \geq 0, \quad j = 1, \dots, r, \quad r+1, \dots, n \end{aligned} \quad (1.3)$$

Lineární funkce (1.1) ukazuje základní rys úlohy lineárního programování - linearitu a zároveň představuje matematickou formulaci jejího ekonomického ekvivalentu, kterou nazýváme *účelová funkce*. *Podmínky omezení úlohy* pak reprezentují rovnice a nerovnice (1.2) a (1.3), které taktéž splňují předem zmíněnou vlastnost linearitu.

Z podmínek omezení úlohy je patrné, že nejednotvárnost forem zápisu směřuje k rozšíření teorie lineárního programování a způsobu řešení jednotlivých úloh. Takovýto přístup však není pohodlný, a proto je rozumné jakoukoliv formu omezení úlohy přepsat na úlohu s jediným typem omezení a to ve tvaru rovnic. Tento tvar plně postačuje pro vysvětlení problémů teorie a metod řešení úloh lineárního programování a nazývá se standardní. [1]

Standardní tvar úlohy lineárního programování můžeme vyjádřit takto:

maximalizovat

$$z(\mathbf{x}) = c_1x_1 + c_2x_2 + \dots + c_nx_n \quad (1.4)$$

za podmínek

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &+ \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 &+ \dots + a_{2n}x_n = b_2 \\ &\vdots \end{aligned} \quad (1.5)$$

$$\begin{aligned} a_{m1}x_1 + a_{m2}x_2 &+ \dots + a_{mn}x_n = b_m \\ x_j \geq 0, \quad j = 1, \dots, n \end{aligned} \quad (1.6)$$

Je na místě vysvětlit způsob, jak přejít od libovolného typu zápisu na zápis úlohy ve standardním tvaru. Řešení spočívá v zavedení přídatných nebo taktéž doplňkových

proměnných $x_{n+1}, x_{n+1}, \dots, x_{n+m-q}$, které přidáme ke každé z nerovnic, až získáme následující podobu úlohy

maximalizovat

$$z(\mathbf{x}) = c_1 x_1 + c_2 x_2 + \dots + c_n x_n \quad (1.7)$$

za podmínek

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ \vdots & \\ a_{r1}x_1 + a_{r2}x_2 + \dots + a_{rn}x_n &= b_r \\ a_{r+1,1}x_1 + a_{r+1,2}x_2 + \dots + a_{r+1,n}x_n + x_{n+1} &= b_{r+1} \\ \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n + x_{n+(m-r)} &= b_m \\ x_j &\geq 0, \quad j = 1, \dots, n \end{aligned} \quad (1.8)$$

$$x_j \geq 0, \quad j = 1, \dots, n \quad (1.9)$$

Rovnice (1.7) až (1.9) jsou ekvivalentním zápisem úlohy (1.1) až (1.3) a komplexně nesou název standardní tvar úlohy. Je pochopitelné, že se přídatné proměnné nemůžou v účelové funkci promítnout jinak než s nulovými koeficienty, čili je není třeba zapisovat.

Poznámka 1. Rovnice a nerovnice (1.1) až (1.3) modifikujeme v procesu řešení úlohy lineárního programování na tzv. *kanonický tvar*, o kterém se zmíníme později.

Výše zmíněnými úpravami lze libovolnou úlohu lineárního programování převést na úlohu s tímž významem.

Poznámka 2. V případě, že omezení úlohy má tvar nerovnic typu \geq příslušnou přídatnou proměnnou je nutno odečíst k získání ekvivalentní rovnice.

V rámci zjednodušení zápisu lze úlohu formulovat v některém z následujících tvarů:

$$z(\mathbf{x}) = \sum_{j=1}^n c_j x_j \quad (1.4)$$

Za podmínek

$$\sum_{j=1}^n a_{ij} x_j = b_i, \quad i = 1, \dots, m \quad (1.5)$$

$$x_j \geq 0, \quad j = 1, \dots, n \quad (1.6)$$

nebo

maximalizovat

$$z(\mathbf{x}) = \sum_{j=1}^n c_j x_j \quad (1.4)$$

za podmínek

$$\sum_{j=1}^n \mathbf{A}_j x_j = b_i \quad (1.5)$$

$$x_j \geq 0, \quad j = 1, \dots, n \quad (1.6)$$

Kde $\mathbf{A}_j = (a_{1j}, a_{2j}, \dots, a_{mj})^T$ vyjadřuje sloupcový vektor koeficientů omezujících podmínek.

Prostřednictvím maticového zápisu

maximalizovat

$$z = \mathbf{c}^T \mathbf{x} \quad (1.4)$$

za podmínek

$$\mathbf{Ax} = \mathbf{b} \quad (1.5)$$

$$\mathbf{x} \geq \mathbf{0} \quad (1.6)$$

Nebo souhrnně takto:

$$\max \{ z = \mathbf{c}^T \mathbf{x}, \mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \} \quad (1.10)$$

Poznamenejme, že ne vždy je maximalizace funkce podstatná, což je dáno existencí situací, ve kterých požadujeme minimum. Jelikož jsou úlohy těchto typů vzájemně převoditelné, získáme jednoduchou úpravou funkci $-z(\mathbf{x})$ a tu minimalizujeme.

Definice 1. Přípustným řešením úlohy $z = \mathbf{c}^T \mathbf{x}$ nazveme vektor $\mathbf{x} = (x_1; \dots; x_n)$, který vyhovuje omezujícím podmínkám $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$. [1]

Definice 2. Přípustné řešení úlohy $z = \mathbf{c}^T \mathbf{x}$ nazveme bazické, jestliže sloupcové vektory matice \mathbf{A} , které odpovídají kladným indexům vektoru \mathbf{x} , tvoří lineárně nezávislou soustavu. [1]

Definice 3. Přípustné řešení úlohy $z = \mathbf{c}^T \mathbf{x}$ nazveme optimálním, pokud v něm účelová funkce dosahuje maximální hodnoty. [3]

1.2 Primární a duální úloha lineárního programování

Podstatnou částí teorie lineárního programování je teorie duality. Proto je na místě zmínit základní principy této teorie lineárního programování.

V závislosti na ní se ukazuje, že ke každé úloze lineárního programování existuje jiná úloha, která je s ní určitým způsobem svázaná. Tuto úlohu pak nazýváme *duální úlohou* k původní úloze *primární*.

Zadejme úlohu lineárního programování ve standardním tvaru:

maximalizovat

$$z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} \quad (1.11)$$

za podmínek

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned} \quad (1.12)$$

Současně zvažme jinou úlohu lineárního programování ve tvaru:
minimalizovat

$$p(\mathbf{v}) = \mathbf{v}^T \mathbf{b} \quad (1.13)$$

za podmínek

$$\begin{aligned} \mathbf{A}^T \mathbf{v} &\geq \mathbf{c} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned} \quad (1.14)$$

Pak úlohu (1.13) nazýváme duální úlohou k úloze (1.11). Úlohu (1.11) pak nazveme primární.

Jestliže jsme v primární úloze požadovali minimum, pak v odpovídající duální úloze požadujeme maximum.

Primární úloha:
minimalizovat

$$z(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$$

za podmínek

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

Duální úloha:
maximalizovat

$$p(\mathbf{v}) = \mathbf{v}^T \mathbf{b}$$

za podmínek

$$\begin{aligned} \mathbf{A}^T \mathbf{v} &\leq \mathbf{c} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

Lze vidět, že jednotlivé složky vektoru $\mathbf{b}_i, i = 1, \dots, m$ jsou koeficienty účelové funkce duální úlohy a koeficienty účelové funkce primární úlohy jsou složky vektoru pravých stran soustavy úlohy duální. Je též patrné, že matice koeficientů \mathbf{A} je v obou případech až na malý rozdíl stejná - v duální úloze je transponovaná. [1]

Uvedené statě si demonstrováme na příkladu.

Příklad 1.

Mějme úlohu lineárního programování:

maximalizovat

$$z = 5x_1 + 8x_2 + 7x_3$$

za podmínek

$$3x_1 + 4x_2 + 5x_3 + x_4 = 7$$

$$x_1 + 2x_2 + 3x_3 + x_5 = 3$$

$$x_j \geq 0 \quad j = 1, \dots, 5$$

Duální úlohu získáme takto:

Omezující podmínky duální úlohy vypočteme tím, že roznásobíme matici \mathbf{A}^T vektorem \mathbf{v} zprava a po vynásobení výsledek položíme větší nebo roven vektoru \mathbf{c}^T . Pro náš případ:

$$\mathbf{A} = \begin{pmatrix} 3 & 4 & 5 & 1 & 0 \\ 1 & 2 & 3 & 0 & 1 \end{pmatrix} \quad \mathbf{c}^T = (5 \quad 8 \quad 7 \quad 0 \quad 0)$$

A tedy:

$$\mathbf{A}^T \mathbf{v} = \begin{pmatrix} 3 & 1 \\ 4 & 2 \\ 5 & 3 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot (v_1 \quad v_2) = \begin{pmatrix} 3v_1 + 1v_2 \\ 4v_1 + 2v_2 \\ 5v_1 + 3v_2 \\ 1v_1 + 0v_2 \\ 0v_1 + 1v_2 \end{pmatrix}$$

Odtud soustavu nerovnic zapíšeme takto:

$$3v_1 + 1v_2 \geq 5$$

$$4v_1 + 2v_2 \geq 8$$

$$5v_1 + 3v_2 \geq 7$$

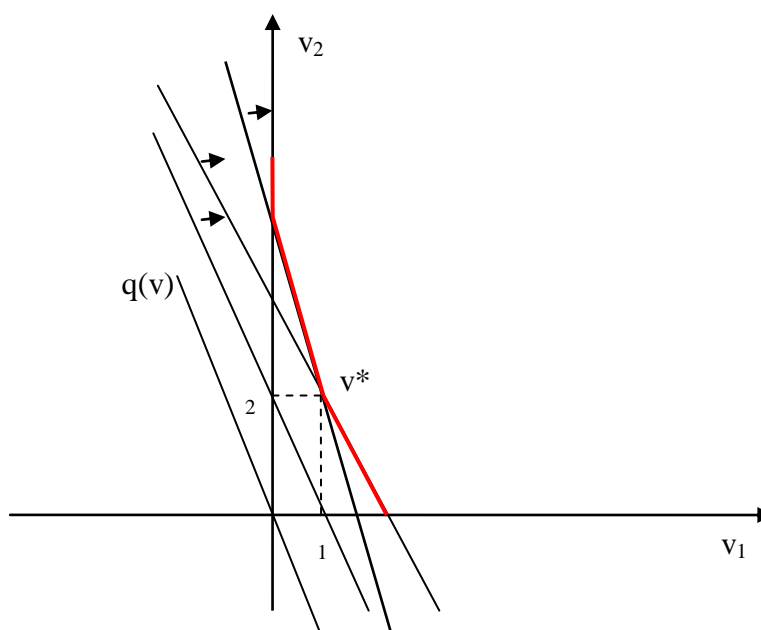
$$v_1 \geq 0$$

$$v_2 \geq 0$$

A je tedy zapotřebí nalézt takový vektor $\mathbf{v}^T = (v_1; v_2)$, který je právě dán soustavou těchto nerovnic. Poznamenejme ještě, že velikost vektoru \mathbf{v} je rovna počtu omezení primární úlohy.

$$p(\mathbf{v}) = 7u_1 + 3u_2$$

Za předpokladu, že duální úloha má dvě proměnné, jak je tomu v našem případě, lze ji řešit grafickým způsobem.



Obrázek 1: Grafické řešení duální úlohy

Z obrázku je patrné, že nejmenší hodnota účelové funkce je v bodě $(\mathbf{v}^*)^T = (1; 2)$. Optimální řešení původní úlohy leží v bodě $\mathbf{x}^T = (1; 1; 0; 0; 0)$. Protože duální úlohy jsou navzájem svázány, znamená to, že maximum účelové funkce primární úlohy je rovno minimu účelové funkce úlohy duální $p(\mathbf{v})$.

Chceme-li tedy nalézt duální úlohu, která lze formulovat k libovolné primární úloze lineárního programování, musíme původní úlohu přepsat do standardního tvaru v případě, že není úloha takto zadána a k té následně duální úlohu najít, jak už bylo předvedeno výše

1.3 Kanonický tvar úlohy lineárního programování

Simplexová metoda nám poskytne řešení úlohy lineárního programování. Pro její aplikaci si však nejdříve musíme představit jiný tvar úlohy lineárního programování než standardní. Chceme tedy úlohu lineárního programování:

maximalizovat

$$z = \mathbf{c}^T \mathbf{x} \quad (1.15)$$

za podmínek

$$\mathbf{Ax} = \mathbf{b} \quad (1.16)$$

$$\mathbf{x} \geq \mathbf{0} \quad (1.17)$$

Je-li možné přeskupit sloupce matice \mathbf{A} , tím způsobem, aby na konci modifikace obsahovaly jednotkovou submatici

$$\begin{aligned}
a_{11}x_1 + \dots + a_{1,n-m}x_{n-m} + x_{n-m+1} &= b_1 \\
a_{21}x_1 + \dots + a_{2,n-m}x_{n-m} + x_{n-m+2} &= b_2 \\
&\dots \\
a_{m1}x_1 + \dots + a_{m,n-m}x_{n-m} + x_n &= b_m
\end{aligned} \tag{1.18}$$

Pak soustavu lineárních rovnic nazveme *kanonickým tvarem* úlohy lineárního programování.

A odtud matice soustavy \mathbf{A}'

$$\mathbf{A}' = \begin{bmatrix} a_{11} & \dots & a_{1,n-m} & 1 & 0 & \dots & 0 \\ a_{21} & \dots & a_{2,n-m} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m1} & \dots & a_{m,n-m} & 0 & 0 & \dots & 1 \end{bmatrix} \tag{1.19}$$

Stejně tak jako u úlohy ve standardním tvaru je i zde nutné vymezit základní a nezákladní proměnné.

Mějme soustavu lineárních rovnic

$$\begin{aligned}
a_{11}x_1 + \dots + a_{1,n-m}x_{n-m} + x_{n-m+1} &= b_1 \\
a_{21}x_1 + \dots + a_{2,n-m}x_{n-m} + x_{n-m+2} &= b_2 \\
&\dots \\
a_{m1}x_1 + \dots + a_{m,n-m}x_{n-m} + x_n &= b_m
\end{aligned}$$

Potom proměnné $x_{n-m+1}, x_{n-m+2}, x_n$ nazveme základní nebo též *bazické*. Proměnné x_1, x_2, x_{n-m} nezákladní neboli *nebazické*.

Poznamenejme také, že na kanonický tvar lze převést omezení zadaná ve tvaru nerovností $\mathbf{Ax} \leq \mathbf{b}$, zavedením přídatných proměnných, které nabývají nezáporných hodnot, čímž získáme opět soustavu typu lineárních rovnic (1.18). [1],[3]

1.4 Primární algoritmus simplexové metody

Budeme-li mluvit o simplexové metodě, myslíme tím mechanismus, který hledá řešení postupným procházením základních řešení úlohy lineárního programování a současně přechod k „optimálnější“ bazickým řešením. Chceme-li při řešení úlohy lineárního programování uplatnit algoritmus simplexové metody, je nezbytné úlohu upravit tak, aby soustava rovnic a nerovnic, kterými je dáno omezení úlohy, byla soustavou výhradně lineárních rovnic. S poznatky z předchozí kapitoly můžeme základní algoritmus simplexové metody shrnout v těchto krocích: [1]

1. Nalezení výchozího přípustného bazického řešení.

2. Kontrola optimálnosti.

3. Za předpokladu, že řešení není optimální, následuje přechod na novou simplexovou tabulku.

Máme-li k dispozici kanonický tvar úlohy, sepíšeme jej do tzv. simplexové tabulky, což vidíme v tabulce 1.

Tabulka 1: Obecná simplexová tabulka

x_1	x_2	\dots	x_m	x_{m+1}	\dots	x_k	\dots	x_n		
1	0	\dots	0	$x_{1,m+1}$	\dots	x_{1k}	\dots	x_{1n}	x_{10}	x_1
0	1	\dots	0	$x_{2,m+1}$	\dots	x_{2k}	\dots	x_{2n}	x_{20}	x_2
									\vdots	
0	0	\dots	0	$x_{r,m+1}$	\dots	x_{rk}	\dots	x_{rn}	x_{r0}	x_r
									\vdots	
0	0	\dots	1	$x_{m,m+1}$	\dots	x_{mk}	\dots	x_{mn}	x_{m0}	x_m
0	0	\dots	0	$x_{0,m+1}$				x_{0n}	x_{00}	z

V pravém sloupci simplexové tabulky zapisujeme názvy bazických proměnných. Vrátime-li se o jeden sloupec zpět, můžeme vyčíst hodnoty bazických proměnných odpovídajícího bazického řešení. Koeficienty účelové, včetně její hodnoty ve zkoumaném řešení jsou zaznamenány v posledním řádku simplexové tabulky. Je zřejmé, že ostatní sloupce, které nezmiňujeme, avšak jsou patrné z obrázku, obsahují koeficienty příslušného kanonického tvaru. [1],[4]

Nyní je načas se podívat, jak zkoumanou úlohu interpretovat simplexovou tabulkou.

Příklad 2. maximalizovat

$$z = x_1 + 2x_2$$

za podmínek

$$-3x_1 + 4x_2 \leq 6$$

$$4x_1 + 3x_2 \leq 12$$

$$x_1, x_2, \geq 0$$

Zavedením nezáporných přídatných proměnných získáme x_3 a x_4 úlohu:

maximalizovat

$$z = x_1 + 2x_2$$

za podmínek

$$-3x_1 + 4x_2 + x_3 = 6$$

$$4x_1 + 3x_2 + x_4 = 12$$

$$x_1, x_2, x_3, x_4 \geq 0$$

Ze soustavy je patrný kanonický tvar soustavy naší úlohy:

$$-3x_1 + 4x_2 + x_3 = 6$$

$$4x_1 + 3x_2 + x_4 = 12$$

$$z - x_1 - 2x_2 = 0$$

A ten zapíšeme do simplexové tabulky:

Tabulka 2: Simplexová tabulka pro kanonický tvar příkladu 2

x_1	x_2	x_3	x_4		
-3	4	1	0	6	x_3
4	3	0	1	12	x_4
-1	-2	0	0	0	z

Ve chvíli, kdy máme k dispozici kanonický tvar úlohy sepsaný do simplexové tabulky, postupujeme podle následujícího principu, abychom zjistili, zda jsme již získali optimální kanonický tvar: [1]

V případě, že všechny prvky posledního řádku tabulky, které odpovídají proměnným x_1, x_2, \dots, x_n , jsou nezáporné, tedy $x_{0j} \geq 0$ pro všechna $j = 1, 2, \dots, n$, pak jsme získali simplexovou tabulku, která je tabulkou optimální a v tuto chvíli můžeme ukončit výpočet. Pokud ne, znamená to, že existuje alespoň jedno $j = 1, 2, \dots, n$ takové, že $x_{0j} < 0$.

Simplexová tabulka pak není optimální a je nutno přejít na novou.

Přechod na novou simplexovou tabulku provedeme elementární změnou báze. K tomu je nezbytné určit klíčový sloupec a klíčový řádek simplexové tabulky. K nalezení klíčového sloupce uplatníme následující pravidlo: [1], [4]

Klíčovým sloupcem při přechodu na novou simplexovou tabulku bude sloupec, kterému odpovídá nejmenší ze záporných hodnot x_{0j} , ($j = 1, 2, \dots, n$). Nastane-li situace, že

minimální hodnota je ve více řádcích stejná, volíme klíčový sloupec s nejmenší hodnotou indexu j .

Klíčový řádek elementární změny báze bude takový řádek, ve kterém je podíl prvků x_{i0} s odpovídajícími prvky vedoucího sloupce x_{ik} , kde $i = 1, 2, \dots, m$, minimální.

S ohledem na předem zmíněné určíme odpovídající sloupec, který náleží proměnné x_k vztahem:

$$\min_{x_{0i} < 0} x_{0i} = x_{k0} \quad (1.20)$$

a klíčový řádek pak určíme vztahem:

$$\min_{x_{ik} < 0} \frac{x_{i0}}{x_{ik}} = \frac{x_{s0}}{x_{sk}} \quad (1.21)$$

Za klíčový prvek simplexové tabulky pak budeme považovat prvek x_{sk} . Nyní můžeme uskutečnit přechod na nový kanonický tvar, resp. novou simplexovou tabulku pomocí elementárních úprav tak, aby klíčový sloupec byl sloupcem jednotkovým s jednotkovým prvkem na místě klíčového prvku. S tímto poznatkem pak vynásobíme vedoucí řádek reciprokou hodnotou klíčového prvku a takto upravený klíčový řádek pak vynásobený zápornou hodnotou příslušného prvku klíčového sloupce, přičteme k odpovídajícímu řádku. Což můžeme zapsat matematicky zapsat takto:

$$\begin{aligned} x'_{sj} &= \frac{x_{sj}}{x_{sk}} \\ x'_{ij} &= x_{ij} - x'_{sj}x_{ik} \end{aligned} \quad (1.22)$$

Předtím než přejdeme na novou simplexovou tabulku, provedeme kontrolu optimálnosti a jestliže jsme ještě nezískali optimální simplexovou tabulku, výše uvedené kroky opakujeme. Nyní se tedy můžeme vrátit k příkladu.

Příklad 3. (pokračování příkladu 1.)

Simplexová tabulka 2 není optimální a proto je nutný přechod na novou simplexovou tabulku. Vidíme totiž, že v obou sloupcích, které odpovídají proměnným x_1 a x_2 , jsou na posledním řádku oba prvky záporné. Ještě jednou zapišme výchozí simplexovou tabulku a v ní vyznačme klíčový prvek.

Klíčový sloupec je tedy dán:

$$\min\{-1, -2\} = -2$$

a odtud klíčový řádek pak:

$$\min\left\{\frac{6}{4}, \frac{12}{3}\right\} = \frac{6}{4}$$

Klíčový sloupec odpovídá proměnné x_2 a klíčový řádek pak proměnné x_3

Tabulka 3: Výběr klíčového prvku simplexové tabulky

x_1	x_2	x_3	x_4		
-3	4	1	0	6	x_3
4	3	0	1	12	x_4
-1	-2	0	0	0	z

čemuž odpovídá přípustné bazické řešení $(\mathbf{x}_1)^T = (0; 0; 6; 12)$

Provedeme elementární změnu báze a získáme následující simplexovou tabulku tak, že klíčový řádek jsme vydělili klíčovým prvkem (číslem 4). Tímto způsobem jsme modifikovaný řádek vynásobili číslem -3, což je prvek klíčového sloupce, nacházející se v druhém řádku a připočítali k prvnímu řádku. Následně jsme modifikovaný první řádek vynásobili číslem 2 a přičetli k třetímu řádku.

Tabulka 4: Volba klíčového prvku po elementární změně báze

x_1	x_2	x_3	x_4		
-3/4	1	1/4	0	6/4	x_3
25/4	0	-3/4	1	15/2	x_4
-5/2	0	1/2	0	3	z

Této simplexové tabulce pak odpovídá nové bazické řešení $(\mathbf{x}_2)^T = (0; 1,5; 0; 7,5)$ s hodnotou účelové funkce $z = 3$. Protože v posledním řádku stále nejsou všechny prvky kladné nebo rovny nule, znamená to, že ani tato simplexová tabulka není optimální, opět určíme klíčový řádek a klíčový sloupec a provedeme elementární změnu báze. Výsledek této změny pak zapíšeme do tabulky.

Tabulka 5: Optimální simplexová tabulka příkladu

x_1	x_2	x_3	x_4		
0	1	4/25	3/25	12/5	x_3
1	0	-3/25	4/25	6/5	x_4
0	0	1/5	2/5	6	z

Nové přípustné bazické řešení odpovídá vektoru $(\mathbf{x}_3)^T = (1, 2, 2, 4; 0, 0)$ a hodnotě účelové funkce $z = 6$. Zkoumáním posledního řádku tabulky zjistíme, že jsme již získali optimální simplexovou tabulku a tedy i optimální řešení úlohy.

1.5 Duální algoritmus simplexové metody

Na rozdíl od primárního algoritmu simplexové metody, který hledal řešení postupným zlepšováním bazického přípustného řešení úlohy, duální algoritmus zlepšuje přípustné řešení odpovídající duální úlohy, čímž nalezne optimální řešení. Najdeme-li optimální řešení duální úlohy, pak toto řešení odpovídá optimálnímu řešení primární úlohy. Jinými slovy, prostřednictvím duálního algoritmu získáme optimální řešení jak duální, tak i primární úlohy. Stejně, jako v případě primárního algoritmu simplexové metody, zapisujeme duálně přípustný kanonický tvar úlohy do simplexové tabulky. Jestliže pro všechna $x_{i0}, i = 1, 2, \dots, m$, jež jsou prvky simplexové tabulky, platí $x_{i0} \geq 0$, pak tabulka v tomto tvaru je optimální a můžeme výpočet ukončit. V případě, že existuje alespoň jedno $x_{i0} \leq 0$, pak určíme klíčový řádek vztahem: [1],[3]

$$\min_{x_{i0} < 0} x_{i0} = x_{k0} \quad (1.23)$$

Stejně tak pokud $\forall x_{kj}, j = 1, 2, \dots, n$ platí $x_{kj} \geq 0$, pak je úloha nepřípustná a výpočet ukončíme.

V případě, že existuje alespoň jedno $x_{kj} < 0$ pro $j = 1, 2, \dots, n$, pak klíčový sloupek je dán vztahem: [1],[3]

$$\min_{x_{kj} < 0} \frac{x_{0j}}{x_{kj}} = \frac{x_{0s}}{x_{ks}} \quad (1.24)$$

V důsledku pak tedy získáme optimální řešení primární úlohy nebo na základě předchozí věty zjistíme, že primární úloha nemá přípustné řešení.

Uvedené nejlépe vynikne na konkrétní situaci, a proto klíčový postup uvedeme na úloze typu:

$$\min \{z = \mathbf{c}^T \mathbf{x}, \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\} \quad (1.25)$$

Pro tento příklad má duálně kanonický tvar následující podobu:

$$-\mathbf{Ax} + \mathbf{Iy} = -\mathbf{b}$$

$$z - \mathbf{c}^T \mathbf{x} = 0$$

Příklad 4.

minimalizovat

$$z = 4x_1 + 6x_2 + 5x_3 + 3x_4$$

za podmínek

$$x_1 + x_2 + 3x_3 + 2x_4 \geq 5$$

$$x_1 + 4x_2 + 2x_3 + x_4 \geq 3$$

$$x_1, x_2, x_3, x_4 \geq 0$$

Duálně přípustný kanonický tvar bude, dle výše zmíněného, zapsán takto:

$$-x_1 - x_2 - 3x_3 - 2x_4 + x_5 = -5$$

$$-x_1 - 4x_2 - 2x_3 - x_4 + x_6 = -3$$

$$z - 4x_1 - 6x_2 - 5x_3 - 3x_4 = 0$$

Soustavu lineárních rovnic, jež odpovídá kanonickému tvaru, v tomto tvaru sepíšeme do simplexové tabulky. Na základě ní je patrné, že řešení není optimální. Vidíme totiž, že $x_{10} < 0$ a $x_{20} < 0$ a pseudooptimální řešení je pak:

$$(\mathbf{x}_1)^T = (0; 0; 0; 0; -5, -3)$$

Tabulka 6: Simplexová tabulka pro duálně přípustný kanonický tvar příkladu 4

x_1	x_2	x_3	x_4	x_5	x_6	
-1	-1	-3	-2	1	0	-5 x_5
-1	-4	-2	-1	0	1	-3 x_6
-4	-6	-5	-3	0	0	0 z

Pro přechod na novou tabulku je tedy nutné zjistit klíčový řádek a klíčový sloupec.

S ohledem na předchozí vztahy, je klíčový řádek dán

$$\min\{-5, -3\} = -5$$

Máme-li tedy klíčový řádek, klíčový sloupec zjistíme

$$\min\left\{\frac{-4}{-1}, \frac{-6}{-1}, -\frac{-5}{-3}, -\frac{-3}{-2}\right\} = \frac{3}{2}$$

Stejným mechanismem jako u primární simplexové metody přejdeme na novou tabulku

Tabulka 7: Výsledek elementární změny báze

x_1	x_2	x_3	x_4	x_5	x_6		
1/2	1/2	3/2	1	-1/2	0	5/2	x_4
-1/2	-7/2	-1/2	0	-1/2	1	-1/2	x_6
-5/2	-9/2	-1/2	0	-3/2	0	15/2	z

Vidíme, že pseudooptimálnímu řešení odpovídá vektor:

$$(\mathbf{x}_2)^T = (0; 0; 0; 5/2; 0; -1/2)$$

který ovšem není přípustný a je opět nutné přejít na novou tabulku, určit klíčový řádek a klíčový sloupec. Opět klíčový řádek je dán:

$$\min\{-1/2\} = -1/2$$

a klíčový sloupec:

$$\min\left\{\frac{-5}{-1}, \frac{-9}{-7}, -\frac{-1}{-1}\right\} = 1$$

Odtud získáme novou tabulku:

Tabulka 8: Optimální simplexová tabulka příkladu 4

x_1	x_2	x_3	x_4	x_5	x_6		
-1	-10	0	1	-2	3	1	x_3
1	7	1	0	1	-2	1	x_4
-2	-1	0	0	-1	-1	8	z

Dospěli jsme do stavu, kdy $x_{10} \geq 0$ a $x_{20} \geq 0$. Můžeme tedy výpočet ukončit, neboť jsme získali optimální řešení, kterému odpovídá vektor:

$$(\mathbf{x}_3)^T = (\mathbf{x}_{opt})^T = (0; 0; 1; 1; 0; 0)$$

2 TEORIE CELOČÍSELNÉHO PROGRAMOVÁNÍ

Existuje mnoho praktických problémů, ať už v ekonomii nebo při řízení a plánování výroby, které vyžadují celočíselné řešení. Všechny tyto problémy pak můžeme řešit prostřednictvím celočíselného lineárního programování. Je zřejmé, že tato úloha se od běžné úlohy lineárního programování liší tím, že na některé nebo všechny proměnné jsou přidány podmínky celočíselnosti. Pokud tedy není kladena podmínka celočíselnosti na všechny proměnné, které v dané úloze vystupují, mluvíme o ní jako o smíšené úloze celočíselného programování. V opačném případě se jedná o úlohu čistě nebo také ryze celočíselnou. V případě, že hodnoty proměnných jsou omezeny pouze na 0 a 1, mluvíme o binárním celočíselném programování, které se také nazývá bivalentní. Existují matematické modely, u nichž se dá celočíselné řešení nalézt pouhým zaokrouhlením. Avšak obecně takový přístup není možný. Mohli bychom se dopustit toho, že takto získané řešení nebude optimální a ba dokonce ani přípustné. [3],[5]

2.1 Metody řešení

Chceme-li řešit úlohu lineárního programování s ohledem na celočíselnost, je nutné zvolit některou z metod celočíselného lineárního programování. Problémem těchto metod je časová složitost algoritmu. Řešení odpovídající úlohy lineárního programování, bez důrazu na celočíselnost, lze nalézt v polynomiálním čase, zatímco jsou-li kladeny podmínky na celočíselnost, je nutné se smířit s faktem, že žádný doposud známý algoritmus není schopen takovouto úlohu ve zmíněném polynomiálním čase vyřešit. Je na místě uvést nejznámější metody řešení úloh celočíselného lineárního programování, mezi které patří: [5]

- Výčtové metody
- Metoda sečných nadrovin
- Metoda větví a mezí

Dále budeme vycházet z modelu

$$\max \{ z = \mathbf{c}^T \mathbf{x}, \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \mathbf{x} \in \mathbf{Z} \} \quad (2.1)$$

kde

\mathbf{c} a \mathbf{x} – n -rozměrné sloupcové vektory

\mathbf{b} – m -rozměrný sloupcový vektor

\mathbf{A} – matice $m \times n$

Výčtové metody – Způsob nalezení optimálního řešení je založen na principu prohledávání všech přípustných řešení, jejichž počet je sice konečný, avšak může být extrémně vysoký. Tato metoda tedy nevyhovuje problémům většího rozsahu a uplatnění nalézá pouze v malých úlohách. [3]

Příklad 5.

maximalizovat

$$z = x_1 - 2x_2$$

za podmínek

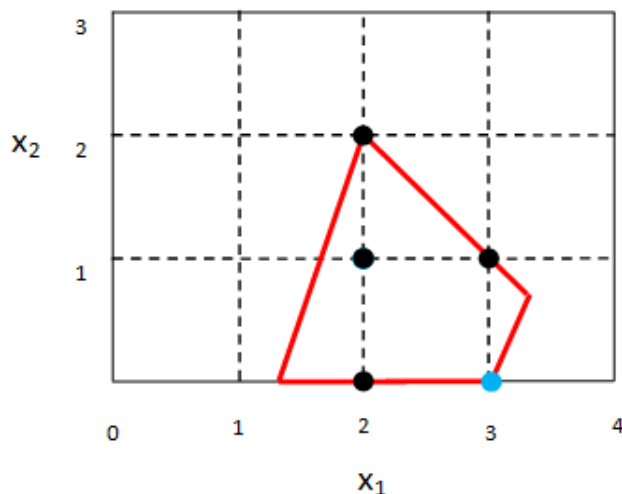
$$9x_1 - 3x_2 \geq 12$$

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

$$2x_1 - x_2 \leq 6$$

$$x_j \geq 0, x_j \in \mathbf{Z} \quad j = 1, 2$$

Z obrázku 2 vidíme, že máme celkem k dispozici pět přípustných řešení. V našem případě není množina přípustných řešení obsáhlá a zkoumáním zjistíme, že optimum se nachází v bodě $(\mathbf{x}_{opt})^T = (3; 0)$. Je zřejmé, že pro takovýto příklad tato metoda postačuje, ovšem u rozsáhlejších příkladů, kdy množina přípustných řešení obsahuje více než tisíc bodů, zcela postrádá smysl.



Obrázek 2: Množina přípustných řešení pro daná omezení příkladu 5

Metoda sečných nadrovin – pod tímto názvem se skrývá skupina algoritmů, které hledají řešení obdobným způsobem jako metoda větví a mezí. Tedy opakovaným řešením úlohy lineárního programování prostřednictvím simplexové metody. Obdržením celočíselného řešení výpočet končí. V opačném případě přidáme další omezující podmínku, čímž zúžíme

oblast přípustných řešení, přičemž každá nově přidaná podmínka musí splňovat tyto požadavky:

- Optimální řešení nalezené pomocí simplexové metody se stane nepřipustným.
- Žádné celočíselné řešení nalezené v předchozím kroku se nemůže stát nepřipustným

Takto vzniklá celočíselná úloha je opět řešena prostřednictvím simplexové metody jako úloha lineárního programování a vznikem celočíselného řešení proces ukončíme. K nejznámějším metodám patří Dantzigovy a Gomoryho řezy, lišící se v konstrukci podmínek. [5]

Metoda větví a mezí - jedná se o jednu z kombinatorických metod, jejímž principem je nalézt optimum postupným řešením úloh lineárního programování, do kterých se navazují celočíselné podmínky na neceločíselné proměnné. Uvedený proces se znázorňuje pomocí stromového grafu skládajícího se z vrcholů a větví. [1]

Předpokládejme tento problém:

$$\max \{z = \mathbf{c}^T \mathbf{x}, \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \mathbf{x} \in \mathbf{Z}\} \quad (2.2)$$

Mějme množinu Q , která obsahuje celočíselné řešení:

$$Q = \{\mathbf{x} | \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \mathbf{x} \in \mathbf{Z}\} \quad (2.3)$$

Symbolem $C(Q)$ vytyčme konvexní obal množiny.

Množinu spojitých řešení označme \tilde{Q}

$$\tilde{Q} = \{\mathbf{x} | \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\} \quad (2.4)$$

Množiny $C(Q)$ a \tilde{Q} vytvářejí neohraničené konvexní polyedrické množiny

Protože metoda větví a mezí je kombinatorickou metodou, což znamená, že vychází ze skutečnosti, že prostor řešení sestává z konečného počtu bodů, uplatňme kritérium optimálnosti, $\mathbf{c}^T \mathbf{x}^* \geq \mathbf{c}^T \mathbf{x}$ kde \mathbf{x}^* je řešením optimálním a \mathbf{x} je libovolné přípustné řešení. Dále pak uvažujme horní, resp. dolní hranici \bar{z}_i resp. \underline{z}_i , kde i je krok, vzniklou uspořádáním hodnot cílové funkce $z = \mathbf{c}^T \mathbf{x}$. [1]

Horní hranice, která je stanovena vzhledem na množinu řešení \tilde{Q} , se vytváří z hodnot cílové funkce, dané řešením neceločíselné úlohy.

$$\begin{aligned} \bar{z} &= z_i^* \text{ jestliže } x^* \text{ je optimálním řešením úlohy} \\ \bar{z} &= \infty \text{ jestliže } \tilde{Q} \text{ je prázdná množina } \vee z \text{ je neohraničená na } \tilde{Q} \end{aligned} \quad (2.5)$$

Dolní hranicí pak rozumíme hodnoty funkce dané existujícím maximálním celočíselným řešením. Tedy jinými slovy se jedná o nejlepší celočíselné řešení, které jsme doposud výpočtem dosáhli. Umožňuje nám posoudit další celočíselná řešení v procesu vyhledávání optima funkce, což můžeme zapsat takto:

$$\begin{aligned} \bar{z}_i &= z_i^* \text{ pro } z_i^* > z_{i-1} \\ \bar{z}_i &= z_{i-1} \text{ pro } z_i^* \leq z_{i-1} \end{aligned} \quad (2.6)$$

Předpokládejme řešení úlohy lineárního programování

$$X^T = \{x_1, x_2, \dots, x_n\}$$

Mějme množinu

$$G = \{g_1, g_2, \dots, g_n\}$$

přičemž platí

$$\forall n \in G: 0 < n < 1$$

Nechť g_K je prvek množiny G , tak, že pro všechny $g \in G$ platí $g_K \geq g$

pak K -tou složku vektoru X uvažme:

$$x_K = [x_K] + g_K, [x_K] \in \mathbf{Z}, g_K \in G \quad (2.7)$$

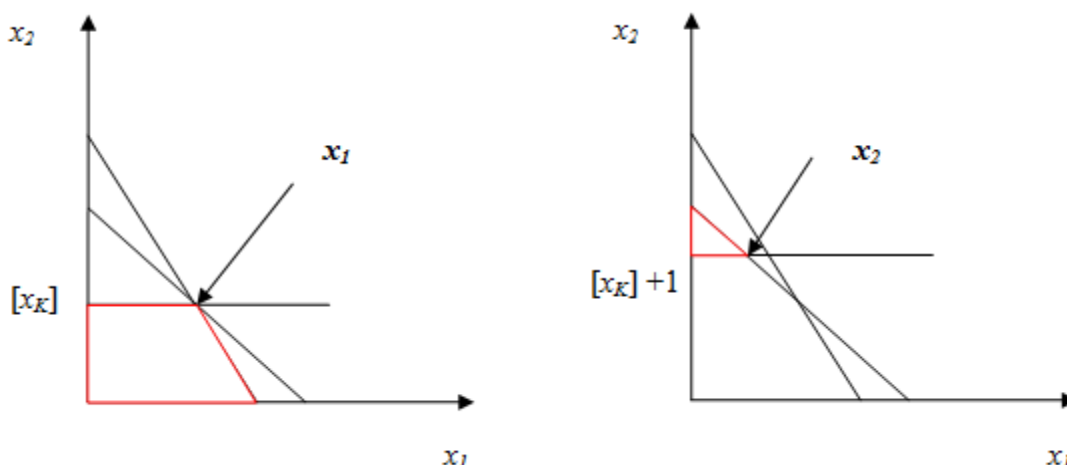
kde hranatá závorka reprezentuje celočíselnou část. Pro každé neceločíselné řešení postačuje zavést dvě omezující podmínky a to:

$$\begin{aligned} x_K &\leq [x_K] \\ x_K &\geq [x_K] + 1 \end{aligned}$$

Z čehož vyplývá, že prostor:

$$[x_K] < x_K < [x_K] + 1 \quad (2.8)$$

můžeme zavrhnout, protože žádné z případných celočíselných řešení neleží v této oblasti, což lze vyjádřit následujícím způsobem:



Obrázek 3: Tvorba příslušných omezení v metodě větví a mezí

Grafické znázornění na obrázku 3 ukazuje tvorbu příslušných omezení v závislosti na úloze dvou proměnných, kde červená hranice vymezuje množinu přípustných řešení.

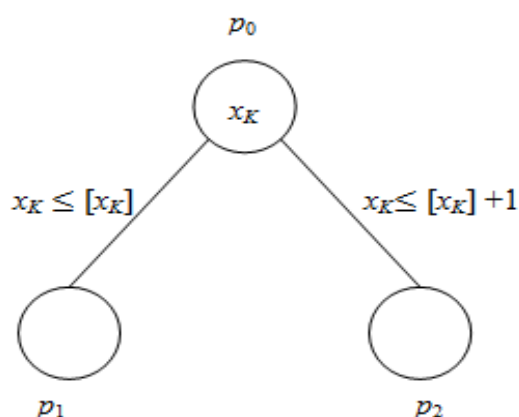
Přidáním těchto omezení k řešením na množině získáme \tilde{Q} obdržíme dvě nové úlohy, které můžeme definovat následujícím způsobem: [1], [5]

$$1. \max \{z = \mathbf{c}^T \mathbf{x}, \tilde{Q} \cap \{x | x_k \leq [x_k]\}\} \quad (2.9)$$

$$2. \max \{z = \mathbf{c}^T \mathbf{x}, \tilde{Q} \cap \{x | x_k \geq [x_k] + 1\}\} \quad (2.10)$$

Jednotlivé kroky výpočtového mechanismu pak znázorňuje stromový diagram $\Lambda = (p, b)$, kde p představuje vrcholy diagramu a b pak jednotlivé větve, spojující jednotlivé vrcholy. Strom vizualizuje dělení množin na podmnožiny. Vrchol znázorňuje přípustné řešení, ze kterého se uskutečňuje větvení, kde příslušnou větví rozumíme omezení celočíselnosti.

Uvažme následující diagram:



Obrázek 4: Větvení řešení

Z obrázku je patrné, že ve vrchole p_0 leží řešení spojitě úlohy, což znamená, že minimálně jedna ze složek vektoru \mathbf{x} není celočíselná. Do vrcholu obvykle zapisujeme horní hranici hodnoty cílové funkce z , a k jednotlivým větvím pak připisujeme příslušná omezení. Vrcholy, které jsou následovníky, nesou nové řešení úlohy při daném omezení.

Rozšířením o omezení na celočíselnost je tedy nutné řešit úlohu znovu. Aplikováním algoritmu simplexové metody na úlohu s nově vzniklými omezeními získáme řešení, které bude v celočíselných proměnných. Pokud nejsou všechny proměnné celočíselné, je nutné znovu zvolit neceločíselnou proměnnou, tu omezit a přistoupit k dalšímu větvení. [1],[5]

2.2 Algoritmus metody větví a mezí

S ohledem na předchozí výklad nyní uveďme v několika krocích postup při řešení úlohy metodou větví a mezí: [1]

- **Krok 1.** Nalezneme řešení spojitě úlohy lineárního programování, které bude odpovídat vrcholu $i = 1$ stromového diagramu, jehož horní mezí bude řešení spojitě úlohy a dolní mezí jeho zaokrouhlení na nejbližší nižší celá čísla. Jsou-li všechny proměnné celočíselné, výpočet končí a obdrželi jsme optimální řešení. V případě, že ne, vrchol 1 se stává větvícím.
- **Krok 2.** Pro větvení vybereme proměnnou s největší neceločíselnou částí a podle ní vytvoříme dva nové podproblémy, které představují možné následovníky.
- **Krok 3.** Řešíme 1. podproblém.
 - Získali jsme řešení, které není přípustné, větev zavrhneme a přejdeme k 4. kroku.
 - Jestliže je řešení přípustné, můžou nastat dvě situace:
 - Všechny proměnné jsou v celočíselných hodnotách. Pokračujeme 4. krokem.
 - Alespoň jedna proměnná je neceločíselná.

Tento vrchol pak přiřadíme do množiny adeptů.

- **Krok 4.** Řešíme 2. podproblém se stejnými možnými případy jako ve předcházejícím podproblému.

- **Krok 5.**
 - Vrchol i má dva možné potomky v množině adeptů. Jednoho z nich zvolíme jako větvící vrchol. Přejdeme k 6. kroku.
 - Vrchol i má jediného možného potomka v množině adeptů.
 - Vrchol i nemá žádného potomka v množině adeptů.
 - Jestliže je množina adeptů prázdná, algoritmus končí.
 - Pokud ne, jeden z vrcholů adeptů určíme za nejbližší větvící vrchol.
- **Krok 6.** Nejbližší větvící vrchol, pak vyloučíme z množiny adeptů a označíme ho jako vrchol $i+1$. Pokračujeme 2. krokem. Nejlepší celočíselné řešení je pak optimální.

2.3 Řešené příklady metodou větví a mezí

Je načase takto nabyté vědomosti zúročit na konkrétních příkladech. Abychom úlohu mohli lépe interpretovat, zvolíme situaci, kde vystupují pouze dvě proměnné, což nám také umožní geometrický výklad příkladu. Z předchozí kapitoly navíc vyplývá, že pro detailnější popis je úloha o dvou proměnných plně postačující, neboť s rostoucím počtem proměnných přímo úměrně roste čas, který musíme na vysvětlení takového problému vynaložit, i když v důsledku je princip stále stejný.

Příklad 6.

maximalizovat

$$z(\mathbf{x}) = 100x_1 + 150x_2$$

za podmínek

$$8000x_1 + 4000x_2 \leq 6$$

$$15x_1 + 30x_2 \leq 200$$

$$x_1, x_2 \geq 0, x_1, x_2 \in \mathbf{Z}$$

Prvním krokem, který provedeme, bude to, že úlohu vyřešíme bez ohledu na celočíselnost, což tedy znamená, že vyřešíme úlohu některou z metod lineárního programování, čímž se dopracujeme k řešení, které odpovídá vektoru:

$$(\mathbf{x}_1)^T = \left(\frac{20}{9}; \frac{50}{9} \right)$$

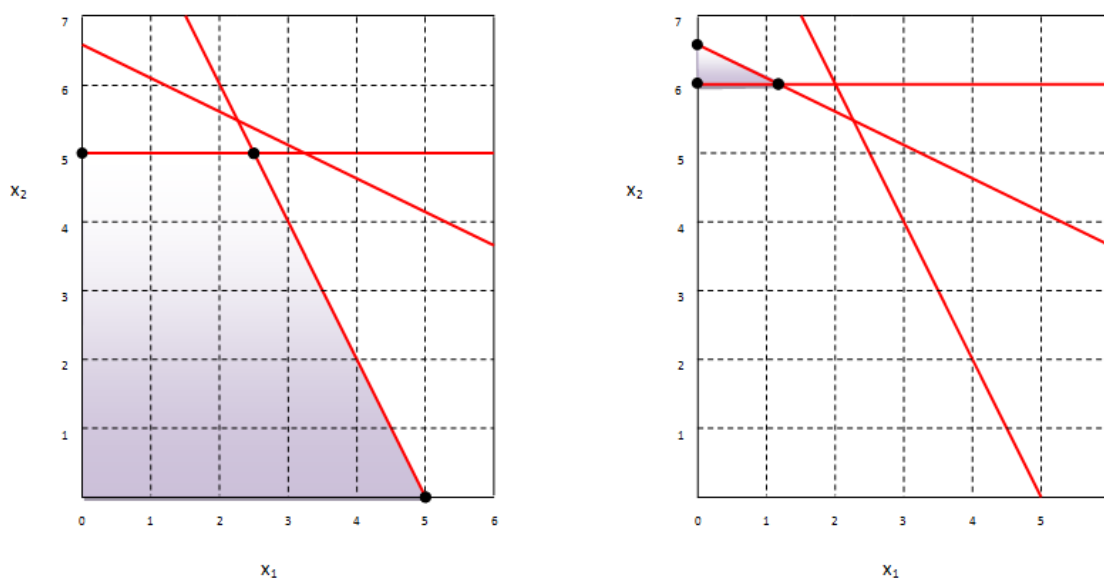
Na první pohled vidíme, že řešení není v celočíselných hodnotách a je tedy zapotřebí danou úlohu rozdělit na dvě nové subúlohy, ke kterým se přidá nová omezující podmínka.

K tomu zvolíme proměnnou, která se bude dále větvit. Na základě výše uvedeného algoritmu vybereme proměnnou, která nese větší desetinnou část, v našem případě se jedná o proměnnou x_2 , čímž získáme úlohy v následujícím tvaru:

$$\begin{aligned} &\text{maximalizovat} \\ &z(\mathbf{x}) = 100x_1 + 150x_2 \\ &8000x_1 + 4000x_2 \leq 6 \\ &15x_1 + 30x_2 \leq 200 \\ &x_2 \leq 5 \\ &x_1, x_2 \geq 0 \\ &x_1, x_2 \in \mathbb{Z} \end{aligned}$$

$$\begin{aligned} &\text{maximalizovat} \\ &z(\mathbf{x}) = 100x_1 + 150x_2 \\ &8000x_1 + 4000x_2 \leq 6 \\ &15x_1 + 30x_2 \leq 200 \\ &x_2 \geq 5 \\ &x_1, x_2 \geq 0 \\ &x_1, x_2 \in \mathbb{Z} \end{aligned}$$

Ukažme si ještě, jak tuto situaci znázorníme graficky. Přidáním omezujících podmínek získáme následující přípustné množiny:



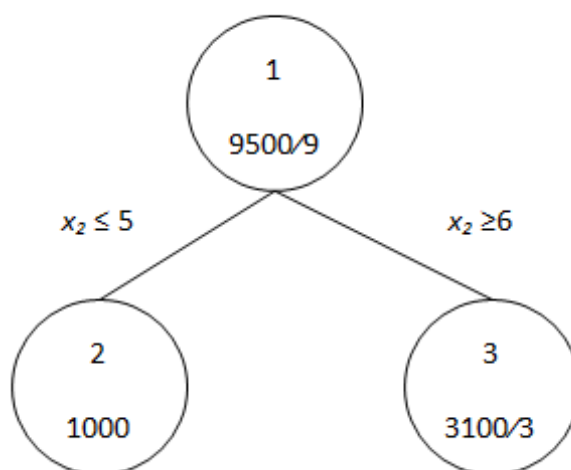
Obrázek 5: Množiny přípustných řešení po přidání podmínek $x_2 \leq 5$ a $x_2 \geq 6$

Stejně jako na začátku i nyní zanedbáme podmínky na celočíselnosti a získáme řešení:

$$(\mathbf{x}_2)^T = \left(\frac{5}{2}; 5 \right)$$

$$(\mathbf{x}_3)^T = \left(\frac{4}{3}; 6 \right)$$

Znázorníme si ještě získané výsledky stromovým diagramem, který bude mít následující podobu:



Obrázek 6: První větvení příkladu 6

Řešení první a druhé subúlohy leží v množině přípustných řešení, avšak ani jedno není v celočíselných hodnotách a tedy dle algoritmu vybereme k dalšímu větvení to s vyšší hodnotou účelové funkce. Z obrázku vidíme, že vyšší hodnotu nám dává řešení obsažené ve třetím vrcholu. Vzhledem k tomu, že vektor řešení obsahuje jedinou neceločíselnou složku, vybereme ji k dalšímu větvení.

Úloha nyní bude vypadat takto:

$$\begin{aligned}
 &\text{maximalizovat} \\
 &z(\mathbf{x}) = 100x_1 + 150x_2 \\
 &8000x_1 + 4000x_2 \leq 6 \\
 &15x_1 + 30x_2 \leq 200 \\
 &x_2 \geq 6 \\
 &x_1 \leq 1 \\
 &x_1, x_2 \geq 0 \\
 &x_1, x_2 \in \mathbf{Z}
 \end{aligned}$$

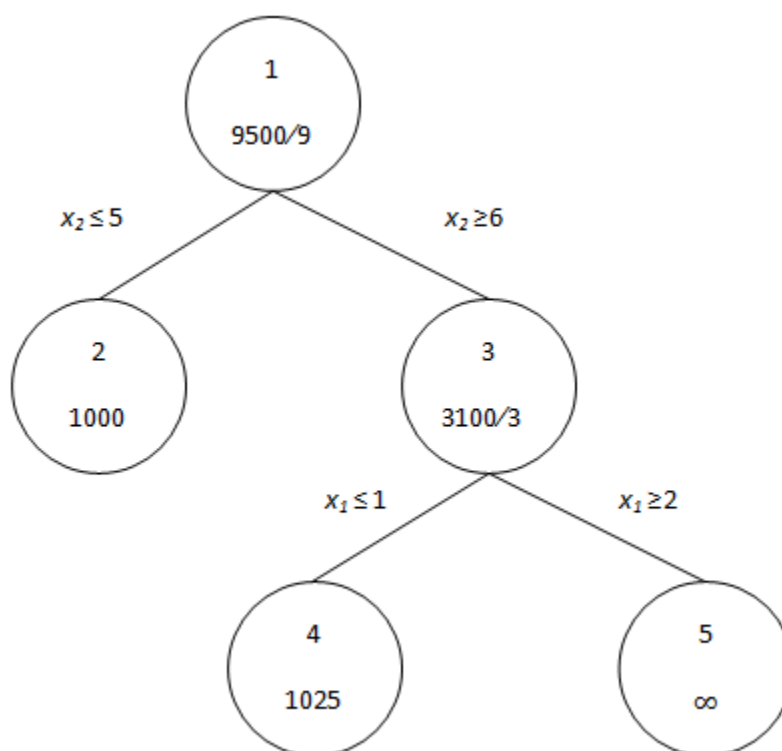
$$\begin{aligned}
 &\text{maximalizovat} \\
 &z(\mathbf{x}) = 100x_1 + 150x_2 \\
 &8000x_1 + 4000x_2 \leq 6 \\
 &15x_1 + 30x_2 \leq 200 \\
 &x_2 \geq 6 \\
 &x_1 \geq 2 \\
 &x_1, x_2 \geq 0 \\
 &x_1, x_2 \in \mathbf{Z}
 \end{aligned}$$

Řešením těchto subúloh získáme:

$$(\mathbf{x}_4)^T = \left(1; \frac{37}{6}\right)$$

Neexistuje

A diagram pak bude vypadat takto:



Obrázek 7: Druhé větvení příkladu 6

Jelikož ve vrcholu pět je množina přípustných řešení prázdná, můžeme pokračovat větvením ve vrcholu druhém nebo čtvrtém. Vzhledem k tomu, že vyšší hodnotu účelové funkce jsme získali ve čtvrtém vrcholu, budeme dále větvit ten, čímž připadají v potaz tyto nové subúlohy:

$$\begin{aligned}
 &\text{maximalizovat} \\
 &z(\mathbf{x}) = 100x_1 + 150x_2 \\
 &8000x_1 + 4000x_2 \leq 6 \\
 &15x_1 + 30x_2 \leq 200 \\
 &x_2 \geq 6 \\
 &x_1 \leq 1 \\
 &x_2 \leq 6 \\
 &x_1, x_2 \geq 0 \\
 &x_1, x_2 \in \mathbb{Z}
 \end{aligned}$$

$$\begin{aligned}
 &\text{maximalizovat} \\
 &z(\mathbf{x}) = 100x_1 + 150x_2 \\
 &8000x_1 + 4000x_2 \leq 6 \\
 &15x_1 + 30x_2 \leq 200 \\
 &x_2 \geq 6 \\
 &x_1 \geq 2 \\
 &x_2 \geq 7 \\
 &x_1, x_2 \geq 0 \\
 &x_1, x_2 \in \mathbb{Z}
 \end{aligned}$$

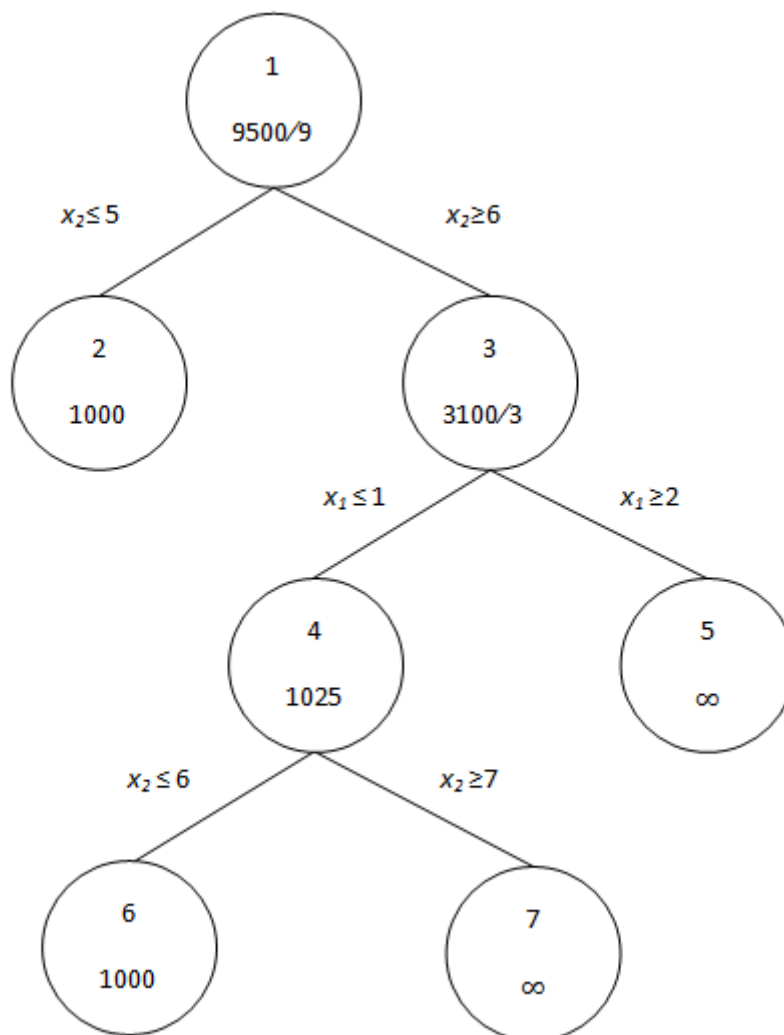
Opět zjistíme, že druhá z úloh nemá řešení:

$$(\mathbf{x}_6)^T = (1; 6)$$

Neexistuje

Vidíme však, že řešení šestého vrcholu, které je optimálním řešením neceločíselné úlohy lineárního programování, splňuje celočíselnou podmínku.

Pro úplnost, si ještě tento krok znázorníme v diagramu:



Obrázek 8: Stromový diagram příkladu 6

Protože v šestém vrcholu není již možné získat lepší řešení, tedy vyšší hodnotu účelové funkce a ani větvení druhého uzlu tento účel nesplní, získali jsme tak optimální celočíselné řešení

$$(\mathbf{x}_{opt})^T = (1; 6)$$

$$z_{opt} = 1000$$

Stejnou metodu si předvedme ještě na jednom příkladu. A i zde bude úloha úlohou se dvěma proměnnými.

Příklad 7.

maximalizovat

$$z(\mathbf{x}) = 2x_1 + x_2$$

za podmínek

$$x_1 + x_2 \leq 5$$

$$-x_1 + x_2 \leq 0$$

$$6x_1 + 2x_2 \leq 21$$

$$x_1, x_2 \geq 0, x_1, x_2 \in \mathbf{Z}$$

Úlohu nejdříve vyřešíme bez ohledu na celočíselnost, tedy jako úlohu lineárního programování a obdržíme řešení:

$$(\mathbf{x}_1)^T = \left(\frac{11}{4}; \frac{9}{4} \right)$$

Ani jedna z proměnných nenabývá celočíselných hodnot a proto přejdeme k větvení. Vzhledem k tomu, že větší neceločíselnou část obsahuje proměnná x_1 , vybereme k větvení právě ji. Tím tedy získáme dvě nové subúlohy vypadající takto:

maximalizovat

$$z(\mathbf{x}) = 2x_1 + x_2$$

$$x_1 + x_2 \leq 5$$

$$-x_1 + x_2 \leq 0$$

$$6x_1 + 2x_2 \leq 21$$

$$x_1 \leq 2$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbf{Z}$$

maximalizovat

$$z(\mathbf{x}) = 2x_1 + x_2$$

$$x_1 + x_2 \leq 5$$

$$-x_1 + x_2 \leq 0$$

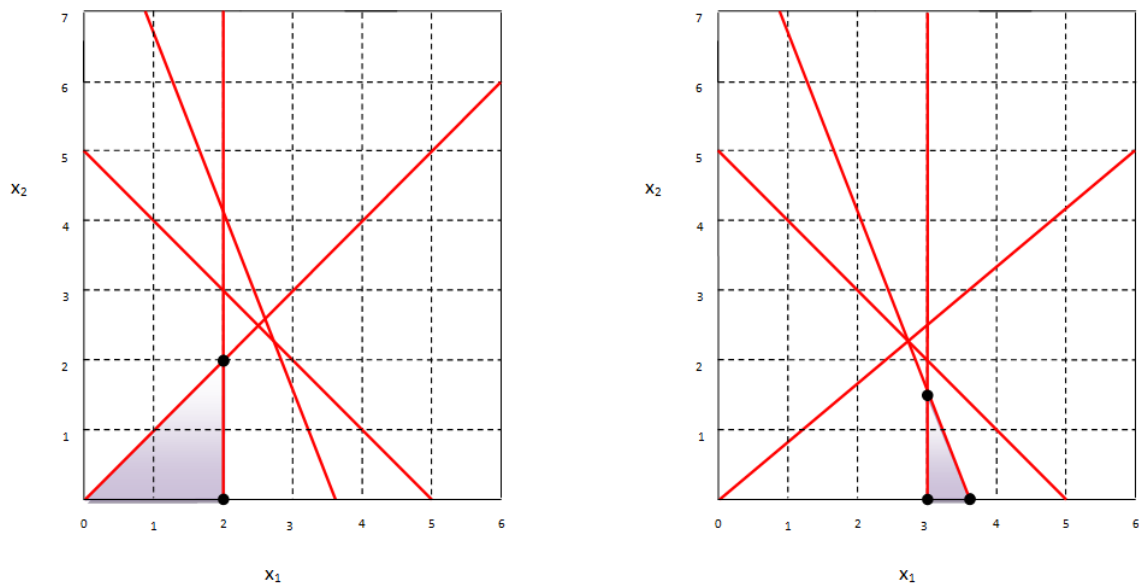
$$6x_1 + 2x_2 \leq 21$$

$$x_1 \geq 3$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbf{Z}$$

Stejně jako v předchozím příkladu si nově přidané podmínky znázorníme graficky:

Obrázek 9: Množiny přípustných řešení po přidání podmínek $x_1 \leq 2$ a $x_1 \geq 3$

Pro tyto subúlohy pak získáme následující optimální řešení:

$$(\mathbf{x}_2)^T = (2; 2)$$

$$(\mathbf{x}_3)^T = \left(3; \frac{3}{2}\right)$$

Lepší hodnotu účelové funkce získáme v bodě $(\mathbf{x}_3)^T$ a z této větve tedy vybereme proměnnou, jež má větší neceločíselnou část. Přidáním omezujících podmínek získá úloha následující podobu:

maximalizovat

$$z(\mathbf{x}) = 2x_1 + x_2$$

$$x_1 + x_2 \leq 5$$

$$-x_1 + x_2 \leq 0$$

$$6x_1 + 2x_2 \leq 21$$

$$x_1 \geq 3$$

$$x_2 \leq 1$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbf{Z}$$

maximalizovat

$$z(\mathbf{x}) = 2x_1 + x_2$$

$$x_1 + x_2 \leq 5$$

$$-x_1 + x_2 \leq 0$$

$$6x_1 + 2x_2 \leq 21$$

$$x_1 \geq 3$$

$$x_2 \geq 2$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbf{Z}$$

Zjistíme, že řešení druhé větve neexistuje a první má takovýto výsledek:

$$(\mathbf{x}_4)^T = \left(\frac{19}{6}; 1\right)$$

Neexistuje

Řešení není v celočíselných hodnotách a přistoupíme k dalšímu větvení, protože ve druhé větvi je množina přípustných řešení prázdná, větvení provedeme podle první větve.

Neceločíselnou část řešení obsahuje pouze proměnná x_1 a jejím rozvětvením přejdeme k těmto subúlohám:

maximalizovat

$$z(\mathbf{x}) = 2x_1 + x_2$$

$$x_1 + x_2 \leq 5$$

$$-x_1 + x_2 \leq 0$$

$$6x_1 + 2x_2 \leq 21$$

$$x_1 \geq 3$$

$$x_2 \leq 1$$

$$x_1 \leq 3$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbf{Z}$$

maximalizovat

$$z(\mathbf{x}) = 2x_1 + x_2$$

$$x_1 + x_2 \leq 5$$

$$-x_1 + x_2 \leq 0$$

$$6x_1 + 2x_2 \leq 21$$

$$x_1 \geq 3$$

$$x_2 \leq 1$$

$$x_1 \geq 4$$

$$x_1, x_2 \geq 0$$

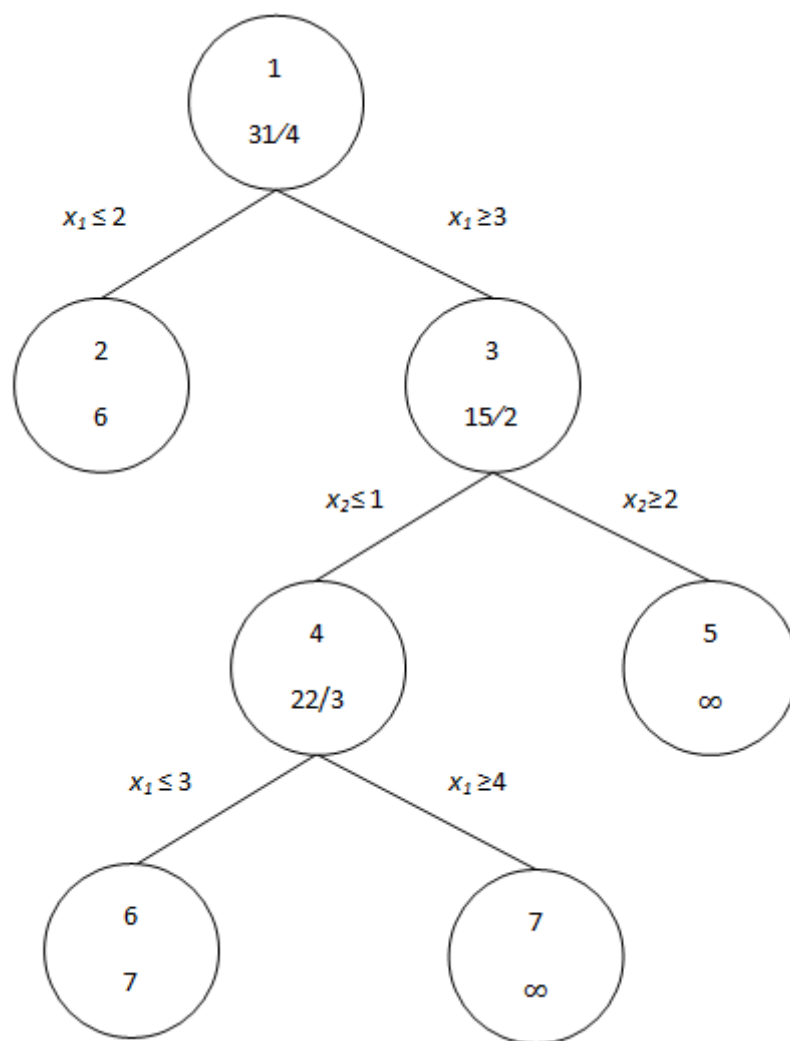
$$x_1, x_2 \in \mathbf{Z}$$

Nově přidané omezující podmínky v jednotlivých subúlohách nám změní řešení úlohy takto:

$$(\mathbf{x}_6)^T = (3; 1)$$

Neexistuje

Jedno z řešení je již v celočíselných hodnotách. Je tedy optimální? Protože ve vrcholech pět a sedm, vzhledem k existenci nepřipustného řešení, další větvení nelze provést a ani vrchol dva nepřinese zlepšení, je řešení ve vrcholu šest optimální. Celou úlohu si nyní interpretujme graficky prostřednictvím stromového diagramu:



Obrázek 10: Stromový diagram příkladu 7

3 TECHNOLOGIE JAVA

Bavíme-li se o technologii Java, nemyslíme tím pouze programovací jazyk, ale také platformu. V následujících podkapitolách si tyto jednotlivé principy vysvětlíme.

3.1 Programovací jazyk Java

Programovací jazyk Java je objektově orientovaným jazykem vyšší úrovně vyvinutý firmou Sun Microsystems. Díky tomu, že je opředěn všemi moderními trendy, se stal jedním z nejpoužívanějších programovacích jazyků na světě. Mezi jeho hlavní přednosti patří zejména hardwarová nezávislost. Vytvořený zdrojový kód v programovacím jazyce Java se nejprve ukládá jako neformátovaný textový soubor s příponou .java. Poté kompilátor zajistí překlad těchto zdrojových souborů do souborů s příponou .class, čímž je zajištěn překlad do speciální mezikódu, který je strojovým jazykem JVM, díky čemuž lze stejné soubory s příponou .class spouštět na platformě Microsoft Windows, Linux nebo Mac OS. [6],[8]

3.2 Platforma Java

Název operačního systému je často spojován v souvislosti se slovem platforma. Je tím míněno pracovní prostředí jak po stránce hardware, tak i software, zajišťující bezproblémovou činnost programů. Zatímto většinu platforem lze popsat jako kombinaci operačního systému a hardware, platforma Java se od těchto odlišuje tím, že je platformou čistě softwarovou. V kapitole o programovacím jazyce jsme si vysvětlili pozici a roli, kterou hraje Java Virtual Machines v rámci celé aplikace. Druhou část, kterou platforma Java zahrnuje, představuje aplikační programové rozhraní. Jedná se o rozsáhlou sbírku hotových komponent. Ty se pak člení do knihoven, které označujeme pojmem packages. Díky tomu, že platforma Java je nezávislá na hardwarové platformě, se může stát, že některé programy jsou pomalejší než nativní kód. V současné době je tedy snaha o to, přiblížit se výkonu nativního kódu a současně zachovat přenositelnost. [6]

3.3 Eclipse IDE a platforma Android

Eclipse je známo jako univerzální vývojové prostředí napsané v jazyce Java. Avšak vzhledem k tomu, že pro tento produkt byly vytvořeny knihovny, zčásti napsané v nativním kódu dané platformy, odpadla nejvíce diskutovaná část u programů vytvořených v Javě - svižnost. Jak jsme si uvedli v předchozích kapitolách, díky

architektury mají programy pomalejší odezvy. V Eclipse tento problém odpadá vzhledem ke skutečnosti, že velká část je napsána v jazyku samotného systému. Další nespornou výhodou je rozšiřitelnost prostřednictvím pluginů, čemuž také odpovídá skutečnost, že základní verze neobsahuje nástroj pro vizuální návrh grafického uživatelského rozhraní. Ten a mnohá další rozšíření je pak možné dodat právě ve formě pluginů. Díky nim můžeme rozšířit seznam podporovaných jazyků například o C++ nebo PHP. Právě ony ale také umožňují, za použitího programovacího jazyka Java, vývoj na platformě Android. Jedná se o softwarovou platformou pro mobilní zařízení, zahrnující operační systém a klíčové aplikace. [6],[9]

II. PRAKTICKÁ ČÁST

4 VÝVOJ NA PLATFORMĚ ANDROID

Dříve, než přejdeme k samotnému vývoji a popisu si ukážeme, jakým způsobem je třeba nakonfigurovat počítač, na kterém bude vývoj probíhat. Pomineme-li systémové požadavky, bude zapotřebí mít k dispozici vhodnou instalaci vývojového prostředí Eclipse. V závislosti na něm se pak budou odvíjet další kroky, které si nyní ukážeme.

4.1 Startovací balíček SDK a instalace ADT

Prvním krokem na cestě k úspěšnému vytvoření aplikace je stažení startovacího balíčku SDK. Ten je možné bezplatně získat na [7]. Stažením tohoto balíčku nezískáme plnohodnotné vývojové prostředí, nýbrž pouze klíčové nástroje, které nám umožní stažení zbývajících komponent. Máme-li tedy v rukou takto stažený balíček s příponou .zip nebo .tgz, rozbalíme jej na bezpečné místo našeho vývojového počítače. Ve výchozím nastavení jsou soubory rozbaleny do adresáře s názvem android-sdk-<machine-platform>. Lokaci SDK adresáře v systému si poznamenáme pro pozdější nastavování pluginu ADT.

Android nabízí vlastní plugin pro Eclipse IDE, zvaný ADT, který byl navržen tak, aby poskytl výkonné a integrované prostředí pro vývoj Android aplikací. Plugin tedy rozšiřuje možnosti Eclipse, čímž nám umožní nejen vytváření uživatelského rozhraní aplikace, ale také ladění za pomoci nástrojů SDK. Nejrychlejším způsobem jak začít je Eclipse spolu s ADT, a proto si tedy v následujících odstavcích ukážeme, jakým způsobem instalovat ADT plugin, který musí být kompatibilní se startovacím balíčkem SDK, do vývojového prostředí Eclipse.

K dispozici máme několik možností, kterými lze Eclipse o zmíněný ADT plugin rozšířit. Avšak nejbezpečnějším a nejjistějším způsobem bude manuální instalace prostřednictvím balíčku ADT, který je dostupný z [1]. Uvedené pak vystihne těchto několik kroků: [7]

- **Krok 1.** Spustíme Eclipse a v záložce *Help* vybereme *Install New Software*.
- **Krok 2.** V aktuálním okně klikneme v pravém horním rohu na tlačítko *Add*.
- **Krok 3.** Ve vyvolaném dialogu *Add repository* klikneme na *Archive*.
- **Krok 5.** Vybereme stažený balíček ADT.
- **Krok 6.** Do textového pole "Name" vložíme název pro lokální aktualizaci webu.
- **Krok 7.** Klikneme na tlačítko *OK*.
- **Krok 8.** V dialogu *Available Software* vybereme zatrhávací políčko *Developer Tools* a klikneme na tlačítko *Next*.

- **Krok 9.** V dalším okně uvidíme seznam nástrojů připravených ke stažení, klikneme na *Next*.
- **Krok 10.** Po přečtení a přijetí licenčních podmínek klikneme na *Finish*.
- **Krok 11.** Když je instalace kompletní, restartujeme Eclipse.

Výše zmíněné, krok za krokem, ukazuje, jakým způsobem stáhnout a instalovat ADT plugin do vývojového prostředí Eclipse. Poznamenejme však ještě, že pro správnou funkčnost je nutné mít tento plugin současně kompatibilní jak s balíčkem SDK tak s Eclipse IDE.

4.2 Konfigurace a aktualizace ADT

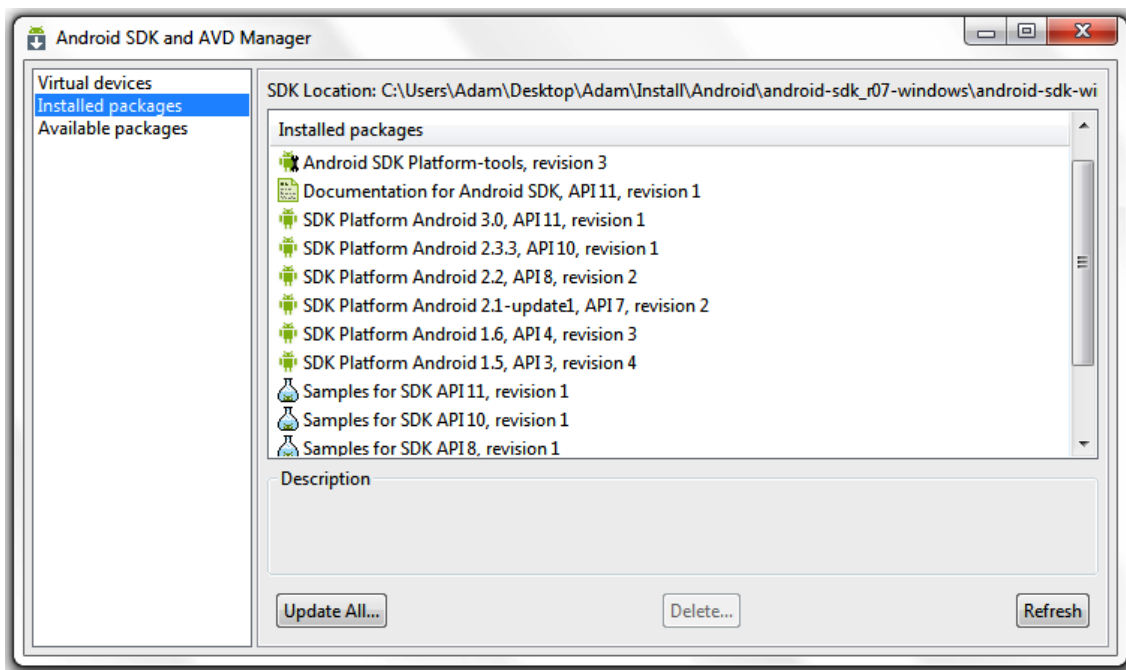
Poté, co jsme úspěšně stáhli a nainstalovali ADT plugin, bude dalším krokem jeho modifikace tak, aby ve výsledku ukazoval na adresář SDK. V této krátké podkapitole si v několika krocích ukážeme, jak tuto konfiguraci přímo provést z prostředí Eclipse. [7]

- **Krok 1.** Spustíme Eclipse a v záložce menu *Window* vybereme *Preferences*.
- **Krok 2.** V levém panelu vybereme položku *Android*.
- **Krok 3.** Do textového pole "SDK Location", za pomoci tlačítka *Browse* lokalizujeme náš SDK Adresář.
- **Krok 4.** Klikneme na *Apply* a poté *OK*.

Tímto je konfigurace dokončena. Poznamenejme však, že čas od času se objeví nová verze pluginu ADT s novými rysy, odladěna od chyb verze předchozí. Proto je vhodné starou verzi aktualizovat. I to přímo provedeme z vývojového prostředí Eclipse, kde v záložce menu *Help* zvolíme *Check for Updates*.

4.3 Přidání platformy a jiných komponent

Posledním krokem k plnému nastavení SDK je užití Android SDK and AVD Manageru ke stažení nezbytných komponent do našeho vývojového prostředí. V předchozích kapitolách jsme si vysvětlili jakým způsobem stáhnout SDK balíček a zakomponovat ho s pluginem ADT do vývojového prostředí Eclipse. K tomu, abychom mohli vyvíjet Android aplikace, bude zapotřebí ještě stáhnout alespoň jednu platformu Android a s ní spojené nástroje. Vše provedeme prostřednictvím zmíněného Android SDK a AVD Manageru, který spustíme uvnitř Eclipse v záložce menu *Window* kliknutím na *Android SDK and AVD Manager*. Vyvíjíme-li na platformě Windows, můžeme také tento nástroj spustit dvojitým kliknutím na *SDK Manager.exe*, který se nachází v kořeni adresáře Android SDK.



Obrázek 11: Android SDK and AVD Manager a instalované platformy Android.

Skrze zmíněný nástroj, jehož grafické uživatelské rozhraní vidíme na Obrázku 11, získáme po stažení přes položku *Available packages* v levém panelu okna, jak platformu Android, která zahrnuje potřebné knihovny, systémové obrázky, vzorový zdrojový kód a emulátor, na kterém můžeme vytvořené aplikace testovat, tak i potřebné USB ovladače, které umožní spouštět a ladit aplikace přímo na zařízení. Jeho prostřednictvím si také, pomocí položky *Virtual Devices*, vytvoříme AVD, které definuje systémový obraz a nastavení zařízení užitě emulátorem. [7]

5 MOBILNÍ APLIKACE CELOČÍSELNÉHO PROGRAMOVÁNÍ

V předchozí kapitole jsme si plně nakonfigurovali vývojové prostředí Eclipse IDE k vytváření Android aplikací. Tato část tedy bude věnována aplikaci, běžící na platformě Android, kterou je pak možné instalovat do mobilních zařízení, čímž nemáme namysli pouze mobilní telefony, ale také tablety.

5.1 Popis aplikace BranchAndBound.apk

Na tomto místě, za pomoci programovacího jazyka Java, detailněji rozpracujeme popis programu, primárně řešící maximalizační úlohu celočíselného programování s využitím kombinatorické metody větví a mezí. Vzhledem k rozsáhlosti programu bude vhodné si vždy stěžejní části zdrojového kódu představit, analyzovat je a vysvětlit jejich význam v rámci celé aplikace.

Nejprve je však nutné ukázat, jakými prvky je zajištěna komunikace s uživatelským problémem. Po spuštění se objeví standardní grafické rozhraní, jež nese tři editovací textová pole, jezdec, přepínač a dvě tlačítka. Jezdec nám umožní nastavit požadovanou hodnotu *NRC* resp. *NV*, která představuje informaci o počtu proměnných resp. počtu omezujících podmínek úlohy. Vzhledem k tomu, že oba parametry nemusejí být stejné, je nutno nastavovanou položku určit prostřednictvím přepínače. Tyto skutečnosti pak zabezpečuje tato metoda:

```
public void onItemClick(AdapterView<?> parent,
    View view, int pos, long id) {
    Toast.makeText(parent.getContext(), " " +
        parent.getItemAtPosition(pos).toString(), Toast.LENGTH_LONG).show();
    if (parent.getItemAtPosition(pos).toString().equals("NRC")) {
        choosen=1;
        m_Bar.setProgress(m_Bar.getProgress())
    }
    else{
        choosen=2;
        m_Bar.setProgress(m_Bar.getProgress());
    }
}
```

Nyní bude potřeba zadat koeficienty jednotlivých omezení a účelové funkce k čemuž nám poslouží editovací textové pole. Vzhledem k tomu, že aplikace využívá systémovou numerickou klávesnici zařízení, není potřeba ošetřovat situaci, kdy uživatel zadá jakoukoliv jinou posloupnost. Abychom se však vyvarovali nechtěnému zhroucení, bude

potřeba ověřit, zda obsah textového pole není prázdný řetězec, což provedeme následujícím způsobem:

```
OnClickListener mDone = new OnClickListener() {
    public void onClick(View v) {
        if(!m_Editor3.getText().toString().equals("")) {
            if (inr==0){
                puvodni_rozmer=spacef();
                prvotni_tabulka=input(puvodni_rozmer);
                row= m_Editor3.getText().toString().split(" ");
                for(int i=0;i<row.length;i++)
                    prvotni_tabulka[inr][i]=Integer.valueOf(row[i]);
                m_Editor3.setText("");
                m_Bar.setEnabled(false);
            }
            else if (inr==puvodni_rozmer[0]){
                row =m_Editor3.getText().toString().split(" ");
                for(int i=0;i<row.length;i++)
                    prvotni_tabulka[inr][i]=Integer.valueOf(row[i]);
                m_Editor3.setText("");
                m_Editor3.setEnabled(false);
            }
            else if(inr<puvodni_rozmer[0]){
                row =m_Editor3.getText().toString().split(" ");
                for(int i=0;i<row.length;i++)
                    prvotni_tabulka[inr][i]=Integer.valueOf(row[i]);
                m_Editor3.setText("");
            }
            inr++;
        }
        else{
            if(m_Editor3.isEnabled())
                showDialog(4)
        }
    }
};
```

V této chvíli jsme veškerá data o úloze předali aplikaci a nezbývá tedy nic jiného než začít řešit. S tím souvisí zanedbání celočíselnosti a následné řešení úlohy jako úlohy lineárního programování k čemuž nám poslouží členská metoda `simplex_method()`. Této metodě je nutné předat celkem tři parametry, které představují simplexová tabulka, její rozměr a rozměr původní úlohy. Ve výsledku pak tato metoda vrátí dvourozměrné pole, ze kterého pak metodou `optimumf()`, získáme optimální řešení úlohy lineárního programování. Její výpis pak vypadá takto:

```

public double []optimumf(double [][]SA_tr,int []bounds,int []former)
{
    int temp = 0;
    double []X = new double[bounds[1]];
    for(int i=0;i<former[1];i++)
    {
        for(int j=0;j<bounds[0];j++)
        {
            if(SA_tr[j][i]==1)
                X[temp++]= SA_tr[j][bounds[0]+bounds[1]];
        }
    }
    return X;
}

```

Obecně mohou nastat tři případy. Získané řešení výše uvedenou metodou je nepřipustné a můžeme výpočet ukončit konstatováním, že nemá-li úloha lineárního programování přípustné řešení, nemá jej ani úloha celočíselného programování. Dále může nastat situace, že jsme jsme již celočíselné řešení získali a tedy i v tomto případě můžeme výpočet ukončit, ovšem tentokrát s nalezeným celočíselným řešením. Za jiných okolností než těchto bude nutné přejít k větvení a proměnnou, podle které větvení provedeme, nám zajistí tento kousek kódu:

```

public int []option(double []x,int []p)
{
    int index =0;
    double temp=x[0]-floor(x[0]);
    double pom=x[0];
    for(int i=0;i<p[1];i++)
    {
        if(x[i]-floor(x[i])>temp)
        {
            temp=x[i]-floor(x[i]);
            pom=x[i];
            index=i;
        }
    }
    int []interval = new int[3];
    interval[0]=floor(pom);
    interval[1]=floor(pom)+1;
    interval[2]=index;
    return interval;
}

```


Nyní jsme ve fázi, kdy nabalením omezujících podmínek získáme dvě nové subúlohy. Pro tento účel byla vytvořena metoda `insert_line_left()` resp. `insert_line_right()`. Ty pak rozšiřují simplexovou tabulku o příslušnou omezující podmínku. Aplikováním simplexového algoritmu, který je implementován v členské metodě `simplex_metod()`, obdržíme dvě nová řešení, jejichž hodnoty získáme prostřednictvím zmíněné metody `optimumf()`. Dále bylo potřeba rozlišit, které z těchto nalezených řešení je „lepší“. Vzhledem k tomu, že veškerá data jsou soustředěna pomocí dvourozměrného pole, bylo potřeba k dalšímu větvení sestavit metody, které by zajistily správnou cestu a navíc si rozměr tohoto řešení uchovaly. Metody `which_table()`, `which_path()`, `which_space()` právě vylíčené skutečnosti řeší. Zajišťují správné pokračování v řešícím procesu a navíc metoda `which_path()` umožňuje uchovávat jednotlivé „cesty“ za řešením, což nám poslouží nejen k porovnávání dvou možných řešení, ale k rozeznání, zda je vůbec toto řešení přípustné. Všechny totiž v sobě implementují metodu `is_permmissible()`, která má za úkol ověřit, zda nalezené je přípustné a tedy vyhovuje příslušným omezením. Její výpis vypadá takto:

```
public boolean is_permmissible(double [][]fs,int []v,double []s,int []a,int []f)
{
    int hel=0;
    double r_s=0;
    for(int i=0;i<a[0];i++){
        r_s=0;
        for(int j=0;j<f[1];j++){
            r_s+=fs[i][j]*s[j];
        }
        if(vec_p[i]==0){
            if(r_s<fs[i][a[0]+a[1]] || r_s==fs[i][a[0]+a[1]])
                hel++;
        }
        if(v[i]==1){
            if(r_s>fs[i][a[0]+a[1]] || r_s==fs[i][a[0]+a[1]])
                hel++;
        }
    }
    if(hel==a[0])
        return true;
    else
        return false;
}
```

Podrobnějším zkoumáním zjistíme způsob, jakým daná metoda ověří přípustné řešení. Je proto nezbytné, aby jejím parametrem byla simplexová tabulka spolu s rozměrem, příslušejícím správnému iteračnímu kroku. Zadefinování pomocné proměnné `sum` slouží ke sledování hodnot levých stran rovnic a nerovnic s následným porovnáním. V případě, že ve všech řádcích simplexové tabulky hodnota proměnné `sum` splňuje příslušnou podmínku, vrací funkce `true`, v ostatních případech potom `false`.

V posledním kroku otestujeme, zda-li získané řešení vyhovuje celočíselné podmínce. Příkladem toho je metoda `is_integer_all()`. Uvedená metoda prochází postupně celý vektor přípustného řešení a v případě, že rozdíl mezi příslušnou složkou vektoru a její dolní hranicí je roven nule, inkrementuje obsah pomocné proměnné `pom`. Na konci pak stačí pouze porovnat zda je její hodnota odpovídá údajům o počtu proměnných. Je-li tato skutečnost pravdivá, vrátí metoda `true` na znamení, že jsme získali celočíselné řešení. V opačném případě nás `false` informuje o tom, že alespoň jedna ze složek vektoru je stále neceločíselná. Uvedené má následující podobu:

```
public boolean is_integer_all(double []x,int []p)
{
    int pom=0;
    for(int i=0;i<p[1];i++){
        if(x[i]-floor(x[i])==0)
            pom++;
    }
    if(pom==p[1])
        return true;
    else
        return false;
}
```

Tato metoda pak vstupuje do jádra algoritmu ve smyčce `while`, kde představuje ukončovací podmínku celého výpočtu. V případě, že tato metoda ponese pozitivní výsledek, příslušná podmínka v cyklu se stane neplatná a aplikace vyvolá dialog s optimálními složkami vektoru, jež představují řešení problému. Jinak cyklus pokračuje ve své další iteraci, opakováním všech doposud uvedených metod.

5.2 Výpočet příkladu na HTC Wildfire

Dostali jsme se do fáze, kdy jsme vybudovali rozsáhlý matematický aparát celočíselnému programování a navíc také prostředek, jehož použití si nyní na praktické situaci názorně předvedeme. Tato část je tedy návodem, jak za pomoci aplikace `BranchAndBound.apk`

řešit celočíselné problémy. Vzhledem k tomu, že ovládání aplikace se na jiném typu zařízení, než na kterém bude výklad probíhat, může lehce lišit, bude rozumné, si vždy lehce rozvést, možná úskalí a odchylky, s tím spojené.

5.2.1 Zadání příkladu a matematická formulace

Majitel autobazaru se rozhodl, za účelem vyššího zisku, rozšířit svou nabídku zánovních vozů, nákupem dvou nových typů automobilů. První typ automobilu nakupuje za cenu 200000 Kč/kus a druhý typ za cenu 100000 Kč/kus. Rozhodl se, že první typ by pak vystavil s cenou 205000 Kč/kus a druhý s cenou 107500 Kč/kus. Na nákup těchto automobilů má k dispozici celkem 1200000 Kč. Vzhledem k tomu, že některé vozy z předešlého měsíce nebyly prodány, má pro nově zakoupená vozidla plochu 80 m², s tím, že první typ zabírá na parkovišti plochu 5 m² a druhý pak 10 m². Nabízí se tedy otázka, kolik kusů od každého druhu automobilu má majitel autobazaru koupit, aby maximalizoval svůj zisk. Všechny tyto hodnoty pak zapíšeme do tabulky 9.

Tabulka 9: Omezení daná nákupem nových automobilů

automobil	plocha (m ²)	cena (Kč/kus)	zisk (Kč/ks)
typ 1	5	200000	5000
typ 2	10	100000	7500

Zformulujeme-li nyní tuto situaci matematickým zápisem, získáme maximalizační úlohu s omezeními typu \leq .

Tedy maximalizovat

$$z = 5000x_1 + 7500x_2$$

za podmínek

$$5x_1 + 10x_2 \leq 80$$

$$200000x_1 + 100000x_2 \leq 1200000$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in \mathbf{Z}$$

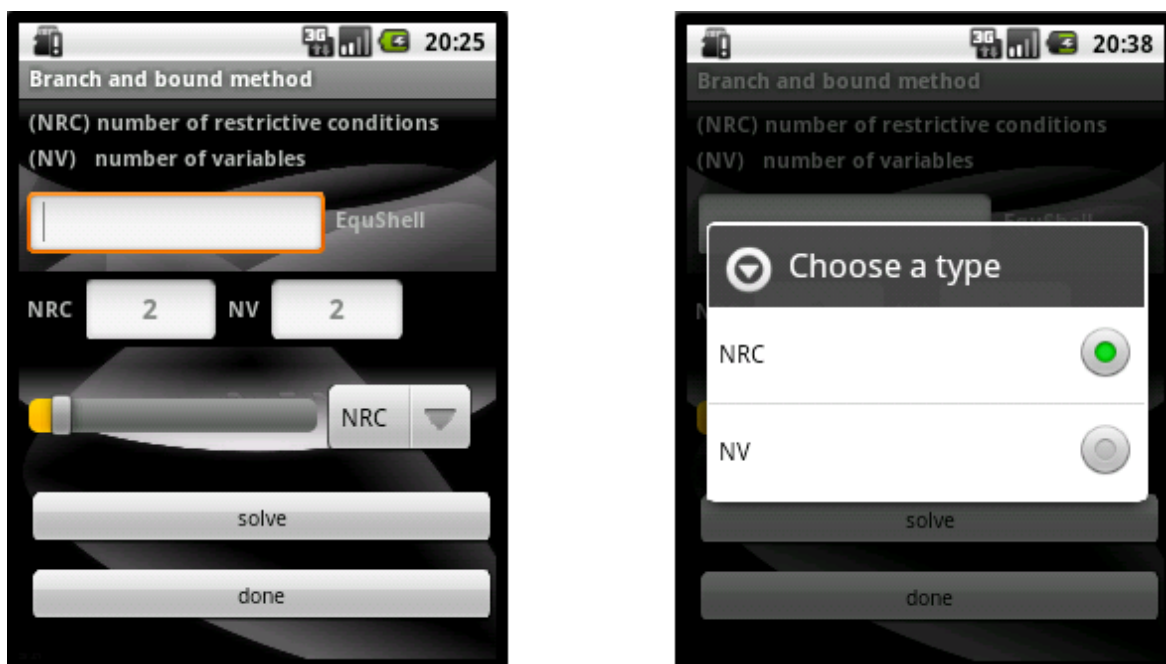
kde

x_1 ... počet automobilů typu 1

x_2 ... počet automobilů typu 2

5.2.2 Aplikace problému na emulátoru

Výše zmíněný problém nám poskytl chybějící ingredienci pro uvedení zmíněné mobilní aplikace, ve které si v několika krocích daný problém vyřešíme. Po spuštění aplikace se objeví standardní, grafické uživatelské rozhraní, jež v sobě nese základní komponenty nutné pro komunikaci s problémem. Jak je patrné z matematické formulace, v úloze vystupují dvě omezující podmínky a dvě proměnné, pod kterými se skrývá počet jednotlivých druhů automobilů. Užitím jezdce, si nejdříve nastavíme hodnotu *NRC*, představující počet omezení dané úlohy. Podobně budeme postupovat i při nastavování hodnoty *NV*, jež nese počet proměnných. Chceme-li přenastavovat hodnoty *NRC* a *NV* pomocí jezdce, je nejprve nutné zvolit v přepínači vpravo, kterou z nich chceme ovlivňovat, což je patrné z obrázku 12.



Obrázek 12: Vkládání informací o počtu omezujících podmínek a proměnných

V této chvíli můžeme začít zadávat koeficienty jednotlivých omezení, včetně hodnot koeficientů účelové funkce, do editovacího textového pole *EquShell*. Nutno dodat, že po přidání každé podmínky je nutné koeficienty uložit prostřednictvím tlačítka *done*. A konečně, když jsme všechny vstupní informace o úloze předali aplikaci, stačí kliknout na

tlačítko *solve*, čímž vyvoláme dialog, ve kterém vidíme optimální celočíselné řešení. Uvedený výklad pak znazorňuje obrázek 13.



Obrázek 13: Vkládání koeficientů s následným řešením problému.

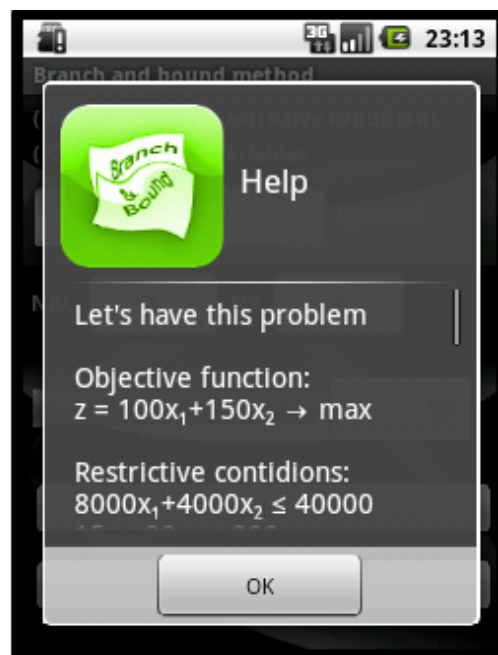
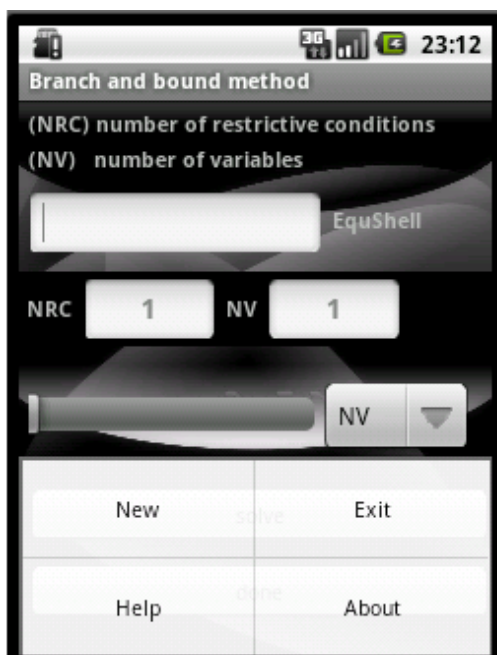
Za předpokladu, že takto ucelený návod nebude ve chvíli spuštění již k dispozici, nabízí aplikace pro osvěžení předchozích kroků jinou alternativu. A tou je vyvolání vysouvacího menu prostřednictvím tlačítka *menu*, které se nachází přímo na těle zařízení. V závislosti na typu zařízení se pak pozice tohoto tlačítka může odlišovat. U mobilních telefonů se nachází na čelní straně těsně nad trackballem. Pomocí něj je možné se posouvat nejen v menu, ale také mezi všemi dalšími komponentami dané aplikace. Na rozdíl od mobilních telefonů, se u tabletů toto tlačítko nachází na boční straně zařízení. V obou případech však jeho stiskem vyvoláme zmíněné vysouvací menu které obsahuje tyto položky:

- *New*
- *Exit*
- *Help*
- *About*

Jak je patrné z obrázku 14, vyvolané vysouvací menu nabízí položku *Help*. Pod ní se skrývá nápověda, ve které je znovu ilustrováno zadávání informací o celočíselném problému. V případě, že při zadávání jednotlivých koeficientů nebo informací o počtu proměnných, či omezujících rovnic dojde k manuální chybě, stiskem položky *New* ve

vysouvacím menu připravíme prostor pro opětovné zadání. Chceme-li okno aplikace opustit, máme k dispozici dva způsoby:

- Stiskem položky *Exit* vysouvacího menu
- Stiskem tlačítka *zpět* na těle zařízení



Obrázek 14: Vyvolané menu a nápověda aplikace BranchAndBound.apk

ZÁVĚR

Jedno latinské přísloví praví „*est rerum omnium magister usus*“, což v překladu znamená „*praxe je učitelkou všech věcí*“. Zní to možná trochu nadneseně, ale staří Římané to jistě nezapsali jen tak, aby v tom nebyl alespoň kousek pravdy. Samozřejmě, že za vznikem tohoto přísloví nestojí technika, řešící celočíselný problém, jež má rozsáhlé důsledky pro úlohy z ekonomické praxe, ale co však chci skutečně říct je to, čemu mnohému jsem se na cestě za poznáním naučil. Vždy jsem věřil číslům, rovnicím a logice, jež mají výsledek a dávají věcem smysl. O to víc pak bylo důležité zkrotit své myšlenky a dát jim řád tak, aby kousek po kousku odhalovaly celek práce. Nejdříve od teoretické roviny, jež měla připravit půdu pro klíčovou oblast této práce – celočíselné programování. A právě aplikace jedné z jeho metod tvoří výsledek seznámení se s touto problematikou. Na jejím základě bych zdůraznil hlavní aspekty, které poukazují na problém, že řešení úlohy celočíselného programování jako úlohy neceločíselného lineárního programování obecně není možné a dále pak, že řešení nelze vidět ani v pouhém zaokrouhlování. Vědomí, že i malý problém se může stát velmi obtížným a těžko schůdným, mě tedy vedl k vytvoření aplikace, s jejíž pomocí se určité druhy problému dají, s trochou nadsázky, řešit z „kapsy“. Shrnu-li nyní veškeré poznatky, kterými jsem se po dokončení obohatil, nezískám pouze nové poznatky z oblasti optimalizace, ale také cenné zkušenosti s novými technologiemi v oblasti programování.

ZÁVĚR V ANGLIČTINĚ

One of the Latin proverbs says "*est rerum omnium magister usus*" which could be translated as "*the practice is a teacher of all things*". It might sound a little lightly but the Romans certainly did not write it just so. It was at least some truth. Of course, it was not born due to technology which solves integer problem and which has a broad implications for the role of economic practice. However, what I really want to say is how much I have been learning on my way to discover. I have always believed the numbers, equations and logic which have results and make sense of the things. That was more important to tame my thoughts and give them order to bit by bit reveal the whole work. First is the theoretical plane, which should pave the way for a key part of this thesis – an integer linear programming and just one application of these methods is a result of familiarity with this issues. On this basis I emphasize the main aspects that show it is not possible to solve the problem of integer programming as the problem of linear programming and also we cannot see the solution in a mere rounding. Consciousness that even a small problem can become very difficult and hardly viable then led me to create an application in which, you can solve some specific kind of problems from your "pocket" with some exaggeration. If I summarize all the information after finishing my work I was enriched not only in the field optimization but I also obtain a valuable experience with a new technology in the field of programming.

SEZNAM POUŽITÉ LITERATURY

- [1] LAŠČIAK, Adam, et al. *Optimálne Programovanie*. Bratislava : SNTL, 1983. 565 s.
- [2] HANZÁLEK, Zdeněk; ŠŮCHA, Přemysl. *Celočíselné lineární programování*. 24.8.2007. [cit. 2011-01-24]. Dostupné z WWW: <http://edux2.felk.cvut.cz/modules/edux/get_file_from_dms.php?function=view&FileID=1734&source=p>.
- [3] SLAVÍČEK, Ondřej. *Aplikace celočíselného programování v ekonomii*. Olomouc, 2008. 73 s. Diplomová práce. Univerzita Palackého v Olomouci, Přírodovědná fakulta, Katedra matematické analýzy a aplikací matematiky. Vedoucí práce RNDr. Jitka Machalová, Ph.D. Dostupné z WWW>: <<http://mant.upol.cz/soubory/OdevzdanePrace/m08-03-os.pdf>>
- [4] SVRČEK, Jaroslav. *Lineární programování v úlohách*. 2. vydání. Olomouc : Vydavatelství Univerzity Palackého v Olomouci, 2003. 104 s. ISBN 80-244-0705-1.
- [5] ŠŮCHA, Přemysl. *Celočíselné lineární programování*. 11.3.2004 [cit. 2011-01-24]. Dostupné z WWW: <<http://dce.felk.cvut.cz/sucha/articles/ilp.pdf>>.
- [6] ZAKHOUR, Sharon, et al. *Java 6: Výukový kurz. první.* : Computer Press, 2007. 525 s. ISBN 978-80-251-157.
- [7] *Android.com* [online]. 16.4.2007 [cit. 2011-05-20]. Android Developers. Dostupné z WWW: <<http://developer.android.com/resources/tutorials/hello-world.html>>.
- [8] *Dione.zcu.cz* [online]. 2008 [cit. 2011-05-21]. Programovací jazyk Java. Dostupné z WWW: <<http://v1.dione.zcu.cz/java/uvod.html>>.
- [9] *Root.cz* [online]. 3.9.2002 [cit. 2011-05-06]. Eclipse 2. Dostupné z WWW: <<http://www.root.cz/clanky/eclipse-2-ide-na-vsechno/>>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ADT	Android Development Tools.
AVD	Android Virtual Device.
IDE	Integrated Development Environment
JVM	Java Virtual Machine
SDK	Software Developer's Kit.
USB	Universal Serial Bus
A	Matice typu $m \times n$
Z	Množina celých čísel
b	m -rozměrný sloupcový vektor
c	n - rozměrný sloupcový vektor
v	m - rozměrný sloupcový vektor
x	Vektor řešení úlohy
\mathbf{x}_{opt}	Optimální řešení úlohy
$[x_K]$	Dolní celá část čísla x_K
g_K	Neceločíselná část čísla x_K
\hat{z}	Horní mez
\tilde{z}	Dolní mez
z^*	Optimální hodnota účelové funkce
Q	Množina celočíselných řešení
\tilde{Q}	Množina spojitých řešení
Λ	Stromový diagram
$(1;3)$	Souřadnice bodu

SEZNAM OBRÁZKŮ

Obrázek 1: Grafické řešení duální úlohy	15
Obrázek 2: Množina přípustných řešení pro daná omezení příkladu 5	26
Obrázek 3: Tvorba příslušných omezení v metodě větví a mezí	29
Obrázek 4: Větvení řešení	29
Obrázek 5: Množiny přípustných řešení po přidání podmínek $x_2 \leq 5$ a $x_2 \geq 6$	32
Obrázek 6: První větvení příkladu 6	33
Obrázek 7: Druhé větvení příkladu 6	34
Obrázek 8: Stromový diagram příkladu 6	35
Obrázek 9: Množiny přípustných řešení po přidání podmínek $x_1 \leq 2$ a $x_1 \geq 3$	37
Obrázek 10: Stromový diagram příkladu 7	39
Obrázek 11: Android SDK and AVD Manager a instalované platformy Android.	45
Obrázek 12: Vkládání informací o počtu omezujících podmínek a proměnných	52
Obrázek 13: Vkládání koeficientů s následným řešením problému.....	53
Obrázek 14: Vyvolané menu a nápověda aplikace BranchAndBound.apk	54

SEZNAM TABULEK

Tabulka 1: Obecná simplexová tabulka	17
Tabulka 2: Simplexová tabulka pro kanonický tvar příkladu 2	18
Tabulka 3: Výběr klíčového prvku simplexové tabulky	20
Tabulka 4: Volba klíčového prvku po elementární změně báze	20
Tabulka 5: Optimální simplexová tabulka příkladu	21
Tabulka 6: Simplexová tabulka pro duálně přípustný kanonický tvar příkladu 4	22
Tabulka 7: Výsledek elementární změny báze	23
Tabulka 8: Optimální simplexová tabulka příkladu 4	24
Tabulka 9: Omezení daná nákupem nových automobilů	51

SEZNAM PŘÍLOH

P I : Zdrojové soubory na přiloženém CD