Masarykova univerzita

Přírodovědecká fakulta

BAKALÁŘSKÁ PRÁCE

Jan Kovář

Algoritmické řešení úlohy lineárního programování v rovině

Vedoucí práce: doc. RNDr. Martin Čadek, CSc.

Studijní program: Aplikovaná matematika

Studijní obor: Matematika - ekonomie

2011

Masarykova univerzita

Přírodovědecká fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jan Kovář**

Studijní program - obor: **Aplikovaná matematika - Matematika - ekonomie**

Ředitel Ústavu matematiky a statistiky PřF MU Vám ve smyslu Studijního a zkušebního řádu MU určuje bakalářskou práci s tématem:

Algoritmické řešení úlohy lineárního programování v rovině

Algoritmic approach to linear programming in the plane

Oficiální zadání: Úloha lineárního programování v rovině spočívá v nalezení maxima lineární funkce na průniku polorovin. Cílem práce je popsat a případně naprogramovat náhodnostní algoritmus pro řešení této úlohy a současně najít nějaké jeho aplikace.
Literatura: Doporučená literatura
de Berg, Mark - Cheong, Otfried - van Kreveld, Marc - Overmars, Mark. Computational geometry. 3rd ed. Berlin, Heidelberg : Springer, 2008. ISBN 978-3-540-77973-5.

Vedoucí bakalářské práce: doc. RNDr. Martin Čadek, CSc.
Datum zadání bakalářské práce: říjen 2010
Datum odevzdání bakalářské práce: dle harmonogramu ak. roku 2010/2011

V Brně dne 15.10.2010

v.z. prof. RNDr. Jiří Rosický, DrSc.
Ředitel Ústavu matematiky a statistiky

Zadání bakalářské práce převzal dne: 30. 11. 2010

Podpis studenta

Na tomto místě bych rád poděkoval doc. RNDr. Martinu Čadkovi, CSc. za ochotu, laskavý přístup a cenné rady poskytnuté během odborného vedení této práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů.

V Brně dne

Jan Kovář

Název práce: Algoritmické řešení úlohy lineárního programování v rovině

Autor: Jan Kovář

Ústav matematiky a statistiky Přírodovědecké fakulty, MU

Vedoucí bakalářské práce: doc. RNDr. Martin Čadek, CSc.

Klíčová slova: lineární programování, náhodnostní algoritmus, optimalizace

Abstrakt

Práce se zabývá maximalizací lineární funkce na průniku konečně mnoha polorovin. Je odvozen náhodnostní algoritmus k řešení tohoto problému, který je založen na iteračním principu postupného přidávání polorovin. Následně je analyzována jeho očekávaná časová složitost. Je dokázáno, že roste lineárně vzhledem k počtu polorovin obsažených v úloze. Součástí práce je také implementace algoritmu.

Title: Algoritmic approach to linear programming in the plane

Author: Jan Kovář

Department of Mathematics and Statistics, Faculty of Science, MU

Supervisor: doc. RNDr. Martin Čadek, CSc.

Keywords: linear programming, randomized algorithm, optimization

Abstract

The work deals with maximizing a linear function at the intersection of a finite number of half-planes. A randomized algorithm solving this problem is derived. It is based on the principle of iteration, when the half-planes are progressively added. Subsequent analysis of the expected time complexity is provided. The proof that the expected running time increases linearly due to the number of half-planes contained in the problem is given. The work also includes an implementation of the algorithm.

Obsah

Úvod

Úloha lineárního programování je speciálním případem hledání vázaných extrémů reálné funkce více proměnných. Jedná se o situaci, kdy vyšetřovaná funkce je lineární a snažíme se ji maximalizovat na množině, která je zadána soustavou lineárních nerovnic. V rovině pak hledáme bod ležící v průniku konečně mnoha daných polorovin, ve kterém lineární funkce dvou proměnných nabývá maxima.

Nejznámějším algoritmem k řešení úlohy lineárního programování je simplexová metoda. Zde se však budeme zabývat jiným algoritmem. Budeme vycházet především z [1, kapitola 4]. Jedná se o náhodnostní algoritmus. To znamená, že průběh algoritmu nezávisí pouze na jeho vstupu, ale také na nějaké náhodné události.

Na začátku budou vysvětleny základní pojmy a souvislosti problému. Následně bude odvozen algoritmus pro jednodimenzionální úlohu lineárního programování, a to především proto, že jeho různé modifikace jsou použitelné při řešení úlohy lineárního programování v rovině. V další části se budeme zabývat omezenou úlohou v rovině, pro kterou bude uveden detailní postup odvození algoritmu. Posléze bude analyzován problém omezenosti úlohy. Odvodíme algoritmus k rozpoznání neomezené úlohy. Postup pro řešení omezené úlohy využijeme ke konstrukci algoritmu pro obecnou úlohu lineárního programování. Následně vysvětlíme koncept očekávané časové složitosti a dokážeme, že očekávaná časová složitost odvozeného algoritmu je lineární vzhledem k počtu polorovin obsažených v úloze. V dalších dvou kapitolách uvedeme srovnání se simplexovou metodou a možné rozšíření algoritmu pro úlohy lineárního programování, které neřešíme v rovině, ale v prostorech vyšších dimenzí.

Přínosem práce oproti základnímu zdroji [1, kapitola 4] je detailní zpracování, a to jak postupného odvození, tak výsledných algoritmů. Uvedené algoritmy jsou proto snadno implementovatelné. Navíc byl odhalen případ neexistence lexikograficky nejmenšího řešení, který autoři [1] nevzali v úvahu.

Součástí práce je také implementace algoritmu. Program byl vytvořen v prostředí *Delphi 6.0* a zahrnuje mimo jiné přehledný grafický výstup. Důraz je kladen na demonstraci principu algoritmu, a proto je aplikace vhodná pro výukové účely. Funkce programu jsou popsány v poslední kapitole.

Kapitola 1

Základní pojmy

Definice 1.1. Nechť $I = \{1, 2, \ldots, n\}, J = \{1, 2, \ldots, d\}, \overline{I} \subseteq I, \overline{J} \subseteq J$. *Úlohou lineárního programování* rozumíme úlohu najít bod $(x_1, \ldots, x_d) \in \mathbb{R}^d$, ve kterém nabývá lineární funkce

$$c_1x_1 + c_2x_2 + \cdots + c_dx_d$$

svého maxima (minima) za podmínek

$$\begin{aligned} a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{id}x_d &\leq b_i, \quad i \in I \setminus \overline{I}, \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{id}x_d &= b_i, \quad i \in \overline{I}, \\ x_j &\geq 0, \quad j \in \overline{J}, \end{aligned}$$

kde $a_{ij}, b_i, c_i \in \mathbb{R}$ pro každé $i \in I, j \in J$ a $x_1, x_2, \ldots, x_d$ jsou proměnné.

Lze jednoduše ukázat, že libovolnou úlohu lineárního programování lze převést na ekvivalentní úlohu lineárního programování ve tvaru[1] dle definice 1.2.

Definice 1.2. *Úlohou lineárního programování* rozumíme úlohu najít bod $(x_1, \ldots, x_d) \in \mathbb{R}^d$, ve kterém nabývá lineární funkce

$$c_1x_1 + c_2x_2 + \cdots + c_dx_d$$

svého maxima za podmínek

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1d}x_d &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2d}x_d &\leq b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nd}x_d &\leq b_n, \end{aligned}$$

kde $c_1, \ldots, c_d, a_{11}, \ldots, a_{nd}, b_1, \ldots, b_d \in \mathbb{R}$ a $x_1, x_2, \ldots, x_d$ jsou proměnné.
Označíme-li $A = (a_{ij})$ pro $i = 1, \ldots, n$, $j = 1, \ldots, d$, $\boldsymbol{b} = (b_1, \ldots, b_n)^T$, $\boldsymbol{c} = (c_1, \ldots, c_d)$, $\boldsymbol{x} = (x_1, \ldots, x_d)^T$, pak lze pomocí vektorů úlohu zkráceně zapsat

$$max\left\{\boldsymbol{cx} \mid A\boldsymbol{x} \leq \boldsymbol{b}\right\}.$$

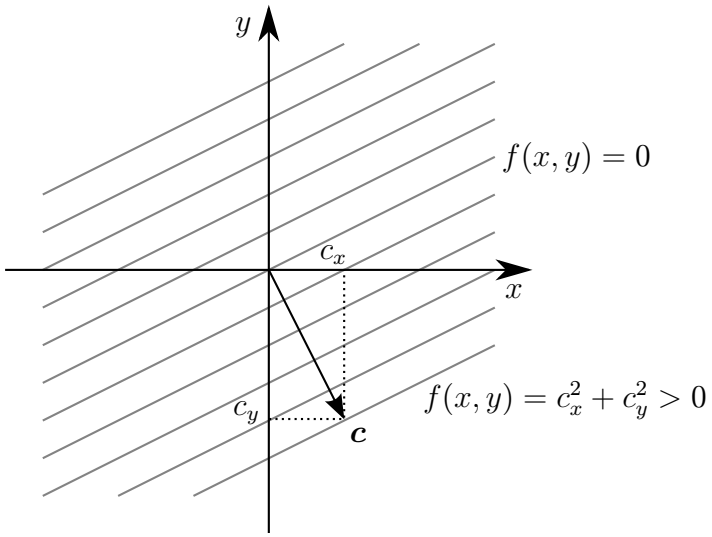
[1] Požadujeme-li navíc, aby všechny proměnné byly nezáporné, nazývá se tento tvar *kanonický*.

Funkce $f(\boldsymbol{x}) = f(x_1, x_2, \ldots, x_d) = c_1x_1 + c_2x_2 + \cdots + c_dx_d$ se nazývá *účelová funkce*. Řekneme, že $\overline{\boldsymbol{x}}$ je *přípustné řešení* úlohy lineárního programování, jestliže platí $A\overline{\boldsymbol{x}} \leq \boldsymbol{b}$. *Optimálním řešením* nazveme takové přípustné řešení $\boldsymbol{x}^*$, které maximalizuje účelovou funkci, tedy $\boldsymbol{x}^*$ je optimální řešení, jestliže $A\boldsymbol{x}^* \leq \boldsymbol{b}$ a pro každé $\overline{\boldsymbol{x}}$ splňující $A\overline{\boldsymbol{x}} \leq \boldsymbol{b}$ platí $\boldsymbol{c}\overline{\boldsymbol{x}} \leq \boldsymbol{c}\boldsymbol{x}^*$.

Zde se budeme zabývat maximalizační úlohou lineárního programování v rovině, tedy speciálním případem výše uvedené definice pro $d = 2$. Proměnné budeme značit x a y, složky vektoru $\boldsymbol{c}$ označme c_x, c_y. Pak účelová funkce je tvaru $f(x, y) = c_xx + c_yy$.

V následujícím textu budeme automaticky předpokládat, že každá úloha lineárního programování je zadána ve tvaru dle definice 1.2. Navíc předpokládejme, že pro každé $i \in \{1, \ldots, n\}$ je alespoň jeden z koeficientů a_{i1}, a_{i2} nenulový. V opačném případě by daná nerovnice buď byla splněna vždy, a proto byla nadbytečná, anebo by nebyla splněna nikdy, a tudíž by úloha neměla přípustné řešení. Dále předpokládejme, že účelová funkce je nenulová, jinak by úloha neměla smysl.

Řešení $(x, y)^T$ můžeme reprezentovat jako body v dvourozměrném euklidovském prostoru. Z analytické geometrie víme, že rovnice $a_{i1}x + a_{i2}y = b_i$ zadává v $\mathbb{R}^2$ přímku a nerovnice $a_{i1}x + a_{i2}y \leq b_i$ polorovinu, jejíž hraniční přímkou je tato přímka. Množina všech přípustných řešení úlohy lineárního programování je tedy průnikem n polorovin. Jelikož polorovina je konvexní oblast a průnik konvexních oblastí je konvexní oblast, je množina všech přípustných řešení konvexní oblastí.



Obrázek 1.1: Vrstevnice účelové funkce

Účelová funkce f je lineární funkcí dvou proměnných. Zobrazíme-li v $\mathbb{R}^2$ její vrstevnice (tedy křivky dané rovnicemi $c_xx + c_yy = k$ pro $k \in \mathbb{R}$), dostaneme rovnoběžné přímky, jejichž normálovým vektorem je vektor $\boldsymbol{c}$. Zřejmě $c_xx + c_yy = 0$ je rovnice vrstevnice procházející počátkem. V každém bodě ležícím na této přímce je hodnota účelové funkce nulová. Uvažme vrstevnici procházející bodem $[c_x, c_y]$. Její rovnice má tvar $c_xx + c_yy = c_x^2 + c_y^2$. Hodnota účelové funkce v bodech ležících na této

přímce je rovna $c_x^2+c_y^2$, a tedy je kladná, což znamená, že je větší než hodnota účelové funkce v bodech vrstevnice procházející počátkem. Vektor $\boldsymbol{c}$ proto udává směr, ve kterém účelová funkce roste, tedy směr, ve kterém se snažíme o maximalizaci.

Pro každé $i \in \{1, \ldots, n\}$ označme poloroviny $h_i = \{(x, y) \in \mathbb{R}^2 \mid a_{i1}x+a_{i2}y \leq b_i\}$. Nechť $H = \{h_1, h_2, \ldots h_n\}$. Pro úlohu lineárního programování pak budeme používat označení $(H, \boldsymbol{c})$, zkráceně též pouze H, bude-li z kontextu jasné, o jaký vektor $\boldsymbol{c}$ se jedná.

V závislosti na množině přípustných řešení a existenci optimálního řešení můžeme rozlišit následující 4 situace.

- Úloha nemá přípustné řešení (tedy ani optimální řešení), což nastává, pokud je průnik polorovin prázdný.
- Úloha má jediné optimální řešení.
- Úloha má více optimálních řešení. Tato situace může nastat v případě, kdy je vektor $\boldsymbol{c}$ kolmý na přímku tvořící hranici konvexní množiny přípustných řešení.
- Množina přípustných řešení je neprázdná, ale úloha nemá optimální řešení. Tato úloha lineárního programování se nazývá *neomezená*. V tomto případě existuje polopřímka ρ, která je celá obsažena v množině přípustných řešení a podél níž hodnota účelové funkce neomezeně roste.

Příklady jednotlivých situací jsou znázorněny na obrázku 1.2. Na které straně od hraniční přímky polorovina leží, je vyznačeno vystínováním.

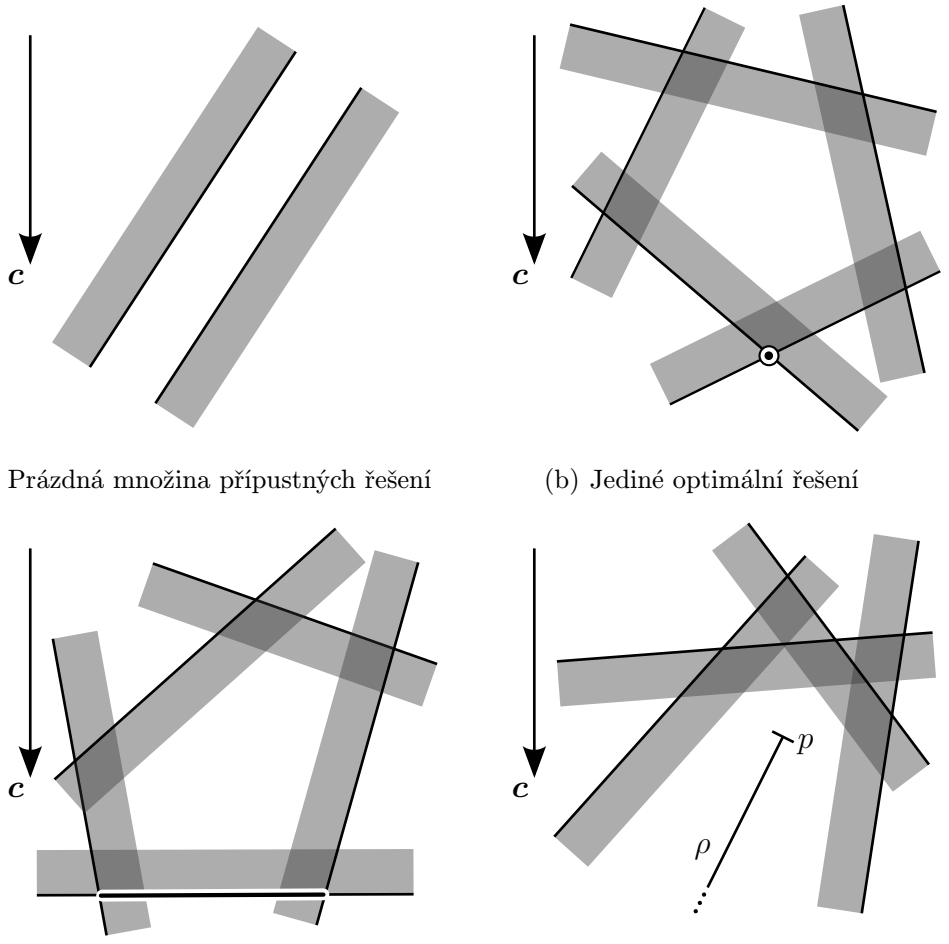
V této práci budeme prezentovat různé algoritmy, stručně se proto podívejme na pojmy související se složitostí algoritmů. Vycházejme přitom z [2]. Každému vstupu můžeme přiřadit jeho velikost. V našem případě, kdy na vstupu je úloha lineárního programování, budeme za velikost vstupu považovat n, tedy počet nerovnic určujících množinu všech přípustných řešení.

Definice 1.3. *Délkou výpočtu* algoritmu pro daný vstup rozumíme počet vykonaných elementárních operací. Nechť $f : \mathbb{N} \to \mathbb{N}$ je funkce. Řekneme, že algoritmus A má *časovou složitost* $f(n)$, jestliže pro každé $n \in \mathbb{N}$ je $f(n)$ maximální délka výpočtu pro všechny vstupy velikosti n.

Při analýze časové složitosti algoritmu je však rozhodující asymptotické chování, zanedbáváme proto aditivní a multiplikativní konstanty. Zajímá nás tedy, jak rychle funkce roste s rostoucí velikostí vstupu.

Definice 1.4. Řekneme, že funkce $g : \mathbb{N} \to \mathbb{N}$ *roste nejvýše tak rychle* jako funkce $f : \mathbb{N} \to \mathbb{N}$, píšeme $g = \mathcal{O}(f)$, jestliže existuje $c \in \mathbb{R}$ a $n_0 \in \mathbb{N}$ takové, že pro každé $n \in \mathbb{N}$, $n \geq n_0$ platí $g(n) \leq cf(n)$.

Při analýze časové složitosti pak říkáme, že algoritmus A má časovou složitost $\mathcal{O}(f(n))$, což znamená, že časová složitost algoritmu A roste nejvýše tak rychle jako funkce f. Speciálně, jestliže algoritmus A má časovou složitost $\mathcal{O}(n)$, říkáme že má *lineární* časovou složitost; pokud má složitost $\mathcal{O}(n^2)$, říkáme že má *kvadratickou* časovou složitost.



(a) Prázdná množina přípustných řešení

(b) Jediné optimální řešení

(c) Více optimálních řešení

(d) Neomezená úloha

Obrázek 1.2: Varianty množin všech přípustných řešení a optimálních řešení

Kapitola 2

Jednodimenzionální úloha

Než začneme odvozovat algoritmus pro řešení úlohy lineárního programování v rovině, podívejme se na jednodimenzionální úlohu, tedy úlohu na přímce. V následujícím textu se s touto úlohou několikrát setkáme, a proto je vhodné rozebrat ji zvlášť v této části.

Nechť je dána jednodimenzionální úloha lineárního programování, tedy úloha najít bod $x \in \mathbb{R}$, ve kterém účelová funkce $f(x) = cx$ nabývá svého maxima za podmínek

$$\begin{aligned} a_1 x &\leq b_1, \\ &\vdots \\ a_n x &\leq b_n, \end{aligned}$$

kde $c, a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{R}$, $c \neq 0$. Množina všech přípustných řešení je průnikem polopřímek.

Pro libovolné $j \in \{1, \ldots, n\}$ uvažme j-tou nerovnici $a_j x \leq b_j$. Mohou nastat následující případy:

- Jestliže $a_j > 0$, pak můžeme nerovnici upravit do podoby $x \leq \frac{b_j}{a_j}$ a jedná se o polopřímku omezenou zprava.
- Jestliže $a_j < 0$, pak můžeme nerovnici upravit do podoby $x \geq \frac{b_j}{a_j}$ a jedná se o polopřímku omezenou zleva.
- Jestliže $a_j = 0$, pak v případě, že $b_j \geq 0$, je j-tá nerovnice splněna pro každé x a může být vynechána, v opačném případě, tj. je-li $b_j < 0$, není j-tá nerovnice nikdy splněna a úloha tak nemá přípustné řešení.

Pokud pro žádnou z nerovnic nenastane případ $a_j = 0$ a $b_j < 0$, uvažme následující dvě množiny indexů.

$$\begin{aligned} J_l &= \{j \in \mathbb{N} \mid 1 \leq j \leq n \wedge a_j < 0\}, \\ J_r &= \{j \in \mathbb{N} \mid 1 \leq j \leq n \wedge a_j > 0\}. \end{aligned}$$

Označíme-li

$$x_l = \begin{cases} \max\left\{\frac{b_j}{a_j} \mid j \in J_l\right\}, & \text{jestliže } J_l \neq \emptyset, \\ -\infty, & \text{jestliže } J_l = \emptyset, \end{cases}$$
$$x_r = \begin{cases} \min\left\{\frac{b_j}{a_j} \mid j \in J_r\right\}, & \text{jestliže } J_r \neq \emptyset, \\ \infty, & \text{jestliže } J_r = \emptyset, \end{cases}$$

pak množina všech přípustných řešení jednodimenzionální úlohy lineárního programování je ekvivalentní vztahům

$$x \geq x_l \qquad a \qquad x \leq x_r.$$

Ze všech polopřímek omezených zleva (respektive zprava) jsme vybrali tu, jejíž koncový bod je největší (resp. nejmenší), neboť je podmnožinou všech polopřímek omezených zleva (resp. zprava).

Uvažme nejprve variantu, že obě uvedené množiny indexů jsou neprázdné, tedy $x_l \neq -\infty$ a $x_r \neq \infty$. Pokud $x_l > x_r$, pak úloha nemá přípustné řešení. V opačném případě je množinou všech přípustných řešení úsečka s krajními body x_l a x_r. Vzhledem k linearitě účelové funkce je optimálním řešením buď bod x_l, anebo bod x_r. Stačí tedy porovnat hodnoty účelových funkcí v těchto bodech.

Jestliže je některá z množin J_l nebo J_r prázdná, může být úloha neomezená a nemusí tedy existovat optimální řešení. Jednotlivé možnosti závisí na tom, zda je účelová funkce rostoucí, tj. $c > 0$, anebo klesající, tj. $c < 0$. Mohou nastat následující případy.

- $J_l = \emptyset$ a $J_r \neq \emptyset$, tedy $x_l = -\infty$ a množina všech přípustných řešení je polopřímka omezená zprava. Je-li účelová funkce rostoucí, je optimálním řešením bod x_r. Pokud je účelová funkce klesající, úloha je neomezená.
- $J_l \neq \emptyset$ a $J_r = \emptyset$, tedy $x_r = \infty$ a množina všech přípustných řešení je polopřímka omezená zleva. Jestliže je účelová funkce klesající, pak optimálním řešením je bod x_l. V opačném případě je úloha neomezená.
- $J_l = \emptyset$ a $J_r = \emptyset$, tedy $x_l = -\infty$, $x_r = \infty$ a množina všech přípustných řešení je celá přímka. Úloha je neomezená v případě rostoucí i klesající účelové funkce.

Uvedené poznatky o jednodimenzionální úloze lineárního programování můžeme shrnout v algoritmu 1. S tímto algoritmem se v různých obměnách setkáme při následné analýze úlohy lineárního programování v rovině.

Algoritmus 1 Algoritmus k řešení jednodimenzionální úlohy

ONEDIMLP$(c, a_1, \ldots, a_n, b_1, \ldots, b_n)$

Vstup: Reálná čísla $c, a_1, \ldots, a_n, b_1, \ldots, b_n$, kde $f(x) = cx$ je účelová funkce úlohy s množinou všech přípustných řešení danou nerovnicemi $a_j x \leq b_j, j \in \{1, \ldots, n\}$.

Výstup: Optimální řešení jednodimenzionální úlohy, případně zpráva o tom, že úloha nemá přípustné řešení či je neomezená.

```
1: x_l ← −∞; x_r ← ∞;
2: for i = 1 to n do
3:    if a_i > 0 then
4:       x_r ← min(x_r, b_i/a_i);
5:    else if a_i < 0 then
6:       x_l ← max(x_l, b_i/a_i);
7:    else
8:       if b_i < 0 then
9:          return Úloha nemá přípustné řešení.
10: if x_l ≠ −∞ ∧ x_r ≠ ∞ then
11:    if x_l > x_r then
12:       return Úloha nemá přípustné řešení.
13:    else
14:       if c > 0 then
15:          return x_r
16:       else
17:          return x_l
18: else
19:    if x_l = −∞ ∧ x_r ≠ ∞ ∧ c > 0 then
20:       return x_r
21:    else if x_l ≠ −∞ ∧ x_r = ∞ ∧ c < 0 then
22:       return x_l
23:    else
24:       return Úloha je neomezená.
```

Kapitola 3

Algoritmus pro omezenou úlohu lineárního programování

Vraťme se nyní k úloze lineárního programování v rovině. V této části se budeme zabývat omezenou úlohou. Algoritmus, který zde popíšeme, je založen na postupném přidávání lineárních omezení (polorovin). V jednotlivých krocích jsou řešeny úlohy lineárního programování, jejichž množiny přípustných řešení jsou dány prvními i polorovinami. Účelová funkce všech těchto úloh je stejná. Ze známého optimálního řešení úlohy dané prvními $i-1$ polorovinami algoritmus odvodí optimální řešení úlohy, která vznikne přidáním další poloroviny, tj. poloroviny h_i. Avšak algoritmus vyžaduje, aby každá z postupně řešených úloh měla právě jedno optimální řešení, s výjimkou případu, kdy je množina přípustných řešení prázdná. Toho docílíme následujícími dvěma úmluvami.

Protože chceme docílit toho, aby v každém kroku úloha nebyla neomezená, přidáme k úloze další dvě lineární omezení. Na jejich základě získáme počáteční řešení. Pokud je původní úloha omezená, existuje dostatečně velká konstanta M taková, že přidáním podmínek $|x| \leq M$ a $|y| \leq M$ nebude ovlivněno optimální řešení. Tento zápis nám však dává čtyři lineární omezení, při následující specifikaci v závislosti na vektoru $\boldsymbol{c}$ stačí přidat pouze dvě, a to m_1 a m_2.

	$c_x > 0$	$c_x \leq 0$
$c_y > 0$	$m_1 \colon x \leq M$ $m_2 \colon y \leq M$	$m_1 \colon x \geq -M$ $m_2 \colon y \leq M$
$c_y \leq 0$	$m_1 \colon x \leq M$ $m_2 \colon y \geq -M$	$m_1 \colon x \geq -M$ $m_2 \colon y \geq -M$

Viděli jsme, že může nastat situace, kdy má úloha lineárního programování více optimálních řešení (obrázek 1.2(d)). V takovém případě budeme hledat lexikograficky nejmenší řešení. Je ale možné, že úloha, která má více optimálních řešení, nemá lexikograficky nejmenší řešení. Touto situací se budeme zabývat v následující kapitole. V této části proto v případě více optimálních řešení předpokládejme existenci lexikograficky nejmenšího řešení.

Definice 3.1. Nechť $a, b \in \mathbb{R}^2, a = [a_x, a_y], b = [b_x, b_y]$. Řekneme, že bod a je *lexikograficky menší* než bod b, jestliže platí: $a_x < b_x \vee (a_x = b_x \wedge a_y < b_y)$.

Uvažujme úlohu lineárního programování $(H, \boldsymbol{c})$, kde $H = \{h_1, h_2, \ldots, h_n\}$ a m_1, m_2 jsou výše uvedená dodatečná lineární omezení. Zaveďme následující označení:

- $H_i = \{m_1, m_2, h_1, h_2, \ldots h_i\}$ pro každé $i \in \{0, \ldots, n\}$,
- $C_i = \bigcap H_i = m_1 \cap m_2 \cap h_1 \cap \cdots \cap h_i$ pro každé $i \in \{0, \ldots, n\}$, tedy C_i je množina všech přípustných řešení úlohy $(H_i, \boldsymbol{c})$,
- v_i je optimální (lexikograficky nejmenší) řešení úlohy $(H_i, \boldsymbol{c})$ pro každé $i \in \{0, \ldots, n\}$,
- l_i je hraniční přímka poloroviny h_i pro každé $i \in \{1, \ldots, n\}$.

Pokud je úloha $(H, \boldsymbol{c})$ omezená a m_1, m_2 vhodně zvolená dodatečná lineární omezení, tj. neovlivňující optimální řešení, pak jsou úlohy $(H_n, \boldsymbol{c})$ a $(H, \boldsymbol{c})$ ekvivalentní. Jestliže pro nějaké $j \in \{0, \ldots, n\}$ nemá úloha $(H_j, \boldsymbol{c})$ přípustné řešení, tj. je-li $C_j = \emptyset$, pak nemá přípustné řešení ani žádná z úloh $(H_i, \boldsymbol{c})$ pro $i \in \{j+1, \ldots, n\}$, neboť dle definice C_i zřejmě platí

$$C_0 \supseteq C_1 \supseteq \cdots \supseteq C_n.$$

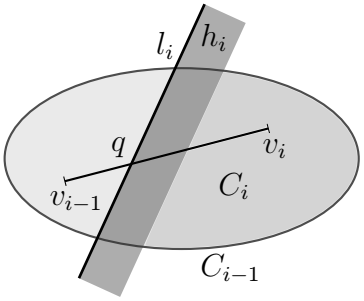
Speciálně pak nemá přípustné řešení ani úloha $(H, \boldsymbol{c})$ v případě, že je omezená.

Věta 3.1. *Pro každé $i \in \{1, \ldots, n\}$ platí:*
(i) jestliže $v_{i-1} \in h_i$, pak $v_i = v_{i-1}$,
(ii) pokud $v_{i-1} \notin h_i$, pak buď $v_i \in l_i$, nebo $C_i = \emptyset$.

Důkaz.

(i) Nechť $v_{i-1} \in h_i$. Víme, že $v_{i-1} \in C_{i-1}$ a $C_i = C_{i-1} \cap h_i$. Proto také $v_{i-1} \in C_i$. Jelikož $C_i \subseteq C_{i-1}$ a v_{i-1} je optimálním řešením na C_{i-1}, je v_{i-1} optimálním řešením také na C_i, a tedy $v_i = v_{i-1}$.

(ii) Důkaz vedeme sporem: předpokládejme, že $v_{i-1} \notin h_i$, $C_i \neq \emptyset$ a $v_i \notin l_i$. Protože C_i je podmnožinou C_{i-1}, je v_i prvkem C_{i-1}. Jelikož také $v_{i-1} \in C_{i-1}$ a současně C_{i-1} je konvexní oblastí, leží v C_{i-1} celá úsečka s krajními body v_i a v_{i-1}. Podle předpokladu leží v_{i-1} v polorovině opačné k h_i. Naopak zřejmě v_i leží v polorovině h_i, neboť $v_i \in C_i$ a $C_i = C_{i-1} \cap h_i$. Z toho vyplývá, že úsečka $v_i v_{i-1}$ protíná l_i. Označme tento průsečík q. Protože q leží v polorovině h_i, je také prvkem množiny C_i.
Jelikož v_{i-1} je optimální řešení na C_{i-1}, musí být hodnota účelové funkce v bodě v_{i-1} větší, anebo rovna hodnotě účelové funkce v bodě v_i. Protože účelová funkce je lineární na $\mathbb{R}^2$, je také lineární, případně konstantní, na každé úsečce v $\mathbb{R}^2$.
Nejprve předpokládejme, že $f(v_{i-1}) > f(v_i)$. Pak z linearity účelové funkce na úsečce $v_i v_{i-1}$ dostáváme vztah $f(v_{i-1}) > f(q) > f(v_i)$. Z druhé nerovnosti však dostáváme spor s optimalitou v_i na C_i.



V případě, že $f(v_{i-1}) = f(v_i)$, je účelová funkce na úsečce $v_i v_{i-1}$ konstantní, a tedy $f(v_{i-1}) = f(q) = f(v_i)$. Víme, že v_{i-1} je lexikograficky nejmenší optimální řešení na C_{i-1}, speciálně v_{i-1} je lexikograficky menší než v_i. Proto je také q lexikograficky menší než v_i, což je spor.

□

Věta 3.1 nám dává částečný návod k nalezení optimálního řešení úlohy H_i. V případě (i) jsou optimální řešení úloh H_i a H_{i-1} totožná. Nastane-li však případ (ii), věta 3.1 nám dává pouze nápovědu, kde příslušné řešení hledat. Při hledání se můžeme omezit pouze na přímku l_i.

Předpokládejme nyní, že $v_{i-1} \notin h_i$. Hledané optimální řešení úlohy H_i musí být současně i přípustným řešením, tj. $v_i \in C_i$, a proto musí ležet ve všech polorovinách[1] $h_1, \ldots, h_{i-1}$. Z uvedených omezení dostáváme novou úlohu lineárního programování v rovině. Označme ji K. Účelová funkce je stejná jako u původní úlohy, množina všech přípustných řešení je dána vztahy:

$$\begin{aligned} a_{i1}x + a_{i2}y &= b_i, \\ a_{11}x + a_{12}y &\leq b_1, \\ a_{21}x + a_{22}y &\leq b_2, \\ &\vdots \\ a_{i-1,1}x + a_{i-1,2}y &\leq b_{i-1}. \end{aligned} \tag{3.1}$$

Jedná se však o speciální případ, který můžeme snadno vyřešit, neboť úlohu lze převést na jednodimenzionální úlohu lineárního programování. Uvažujme nyní o průniku přímky l_i a poloroviny h_j pro $1 \leq j \leq i-1$. Upravujme tedy nerovnici $a_{j1}x + a_{j2}y \leq b_j$ za předpokladu, že platí $a_{i1}x + a_{i2}y = b_i$. Rozlišme nyní dva případy.

Předpokládejme, že $a_{i2} \neq 0$. Pak vyjádřením y z rovnice (3.1) dostáváme

$$y = \frac{b_i - a_{i1}x}{a_{i2}}.$$

[1] V polorovině h_i leží automaticky, neboť dle věty 3.1 leží na její hraniční přímce.

Dosaďme za y do nerovnice poloroviny h_j a upravujme.

$$\begin{aligned} a_{j1}x + a_{j2}y &\leq b_j \\ a_{j1}x + a_{j2}\frac{b_i - a_{i1}x}{a_{i2}} &\leq b_j \\ \frac{a_{j1}a_{i2}x + a_{j2}b_i - a_{j2}a_{i1}x}{a_{i2}} &\leq b_j \\ \frac{x\left(a_{j1}a_{i2} - a_{j2}a_{i1}\right) + a_{j2}b_i}{a_{i2}} &\leq b_j \\ \frac{a_{j1}a_{i2} - a_{j2}a_{i1}}{a_{i2}}\,x &\leq b_j - \frac{a_{j2}b_i}{a_{i2}} \end{aligned} \tag{3.2}$$

Označme

$$D_j^x = \frac{a_{j1}a_{i2} - a_{j2}a_{i1}}{a_{i2}} \qquad \text{a} \qquad E_j^x = b_j - \frac{a_{j2}b_i}{a_{i2}}.$$

Pak v závislosti na D_j^x dostáváme následující omezení pro x:

- $D_j^x > 0$: $x \leq \frac{E_j^x}{D_j^x}$,
- $D_j^x < 0$: $x \geq \frac{E_j^x}{D_j^x}$,
- $D_j^x = 0$: Buď $E_j^x \geq 0$, a tedy nedostáváme žádné omezení pro x (platí $l_i \subset h_j$), anebo $E_j^x < 0$ (tj. $l_i \cap h_j = \emptyset$), pak úloha K nemá přípustné řešení a podle věty 3.1 je C_i prázdnou množinou.

Předpokládejme, že $a_{i2} = 0$. Postupujeme analogicky jako v předchozím případě. Z rovnice (3.1) vyjádříme x, dosadíme do nerovnice poloroviny h_j a po úpravách dostaneme

$$a_{j2}y \leq b_j - \frac{a_{j1}b_i}{a_{i1}}. \tag{3.3}$$

Označme

$$D_j^y = a_{j2} \qquad \text{a} \qquad E_j^y = b_j - \frac{a_{j1}b_i}{a_{i1}}.$$

V závislosti na D_j^y dostáváme následující omezení pro y:

- $D_j^y > 0$: $y \leq \frac{E_j^y}{D_j^y}$,
- $D_j^y < 0$: $y \geq \frac{E_j^y}{D_j^y}$,
- $D_j^y = 0$: Buď $E_j^y \geq 0$, a tedy nedostáváme žádné omezení pro y (platí $l_i \subset h_j$), anebo $E_j^y < 0$ (tj. $l_i \cap h_j = \emptyset$), pak úloha K nemá přípustné řešení a podle věty 3.1 je C_i prázdnou množinou.

Pro $1 \leq j \leq i-1$ zaveďme následující označení.

$$D_j = \begin{cases} D_j^x, & \text{jestliže } a_{i2} \neq 0, \\ D_j^y, & \text{jestliže } a_{i2} = 0, \end{cases} \qquad E_j = \begin{cases} E_j^x, & \text{jestliže } a_{i2} \neq 0, \\ E_j^y, & \text{jestliže } a_{i2} = 0. \end{cases}$$

Uvažme nyní výše uvedená omezení pro všechna $j \in \{1, \ldots, i-1\}$. Pokud pro nějaké takové j platí $D_j = 0$ a $E_j < 0$, pak $C_i = \emptyset$. V opačném případě můžeme tato lineární omezení rozdělit do dvou množin na omezení shora a omezení zdola. Zaveďme tedy označení následujících množin indexů.

$$J_l = \{j \in \mathbb{N} \mid 1 \leq j \leq i-1 \wedge D_j < 0\},$$
$$J_r = \{j \in \mathbb{N} \mid 1 \leq j \leq i-1 \wedge D_j > 0\}.$$

Úloha K je tedy ekvivalentní jednodimenzionální úloze $\overline{K}$ s proměnnou z, jejíž množina všech přípustných řešení je dána vztahy

$$z \leq \frac{E_j}{D_j}, \quad j \in J_r,$$
$$z \geq \frac{E_j}{D_j}, \quad j \in J_l$$

a jejíž účelová funkce je

$$\overline{f}(z) = \begin{cases} f\left(z, \frac{b_i - a_{i1}z}{a_{i2}}\right), & \text{jestliže } a_{i2} \neq 0, \\ f\left(\frac{b_i}{a_{i1}}, z\right), & \text{jestliže } a_{i2} = 0. \end{cases}$$

Jestliže z je optimální řešení úlohy $\overline{K}$, pak pro optimální řešení úlohy K, tedy v_i, platí následující vztah:

$$v_i = \begin{cases} \left[z, \frac{b_i - a_{i1}z}{a_{i2}}\right], & \text{jestliže } a_{i2} \neq 0, \\ \left[\frac{b_i}{a_{i1}}, z\right], & \text{jestliže } a_{i2} = 0. \end{cases}$$

Získali jsme tak jednodimenzionální úlohu lineárního programování, kterou již umíme řešit na základě kapitoly 2 tohoto textu.

V případě, že $J_l \neq \emptyset$ a $J_r \neq \emptyset$, označíme

$$z_l = \max\left\{\frac{E_j}{D_j} \;\middle|\; j \in J_l\right\} \qquad \text{a} \qquad z_r = \min\left\{\frac{E_j}{D_j} \;\middle|\; j \in J_r\right\},$$

množina všech přípustných řešení úlohy $\overline{K}$ je pak ekvivalentně dána vztahy

$$z \geq z_l, \qquad z \leq z_r.$$

Jestliže $z_l > z_r$, pak úloha $\overline{K}$ nemá přípustné řešení, a tedy podle věty 3.1 je $C_i = \emptyset$. Pokud $z_l \leq z_r$, je množinou všech přípustných řešení úlohy $\overline{K}$ úsečka $z_l z_r$. Vzhledem

k linearitě účelové funkce je optimálním řešením jeden z krajních bodů této úsečky. Je jím bod z_l, pokud $\overline{f}(z_l) \geq \overline{f}(z_r)$ (v případě rovnosti je z_l lexikograficky menší). Jestliže $\overline{f}(z_l) < \overline{f}(z_r)$, optimálním řešením úlohy $\overline{K}$ je bod z_r.

Jestliže však $J_l = \emptyset$ nebo $J_r = \emptyset$, na rozdíl od obecného případu jednodimenzionální úlohy analyzovaného v kapitole 2, nemůže nastat situace, kdy je úloha neomezená. To je zaručeno dodatečnými lineárními omezeními, která byla volena tak, že účelová funkce je shora omezená na množině $C_0 = m_1 \cap m_2$, a tedy i na jejích podmnožinách $C_1, \ldots, C_n$.

Na základě dosavadních poznatků formulujme algoritmus 2 pro řešení omezené úlohy lineárního programování.

Věta 3.2. *Časová složitost algoritmu 2 je* $\mathcal{O}(n^2)$.

Důkaz. Časová složitost každé iterace cyklu **for** (ř. 2–35) závisí na vyhodnocení podmínky na řádku 3. První větev (ř. 4) má konstantní časovou složitost. Druhá část větvení (ř. 6–35) obsahuje 3 **for** cykly (ř. 7–8, 10–11, 13–20), z nichž každý má časovou složitost $\mathcal{O}(i)$. Ostatní části této větve mají konstantní časovou složitost. Celkem je tedy časová složitost i-té iterace cyklu **for** (ř. 2–35) v závislosti na splnění podmínky na řádku 3 buď konstantní, anebo $\mathcal{O}(i)$. V nejhorším případě může v každé iteraci dojít k vykonání druhé větve, a proto je celková složitost algoritmu 2 rovna $\mathcal{O}(1 + 2 + \cdots + n) = \mathcal{O}(\frac{n(n+1)}{2}) = \mathcal{O}(n^2)$. □

Algoritmus 2 Algoritmus pro omezenou úlohu lineárního programování

BOUNDEDLP$(H, \boldsymbol{c}, m_1, m_2)$

Vstup: Úloha lineárního programování $(H, \boldsymbol{c})$ a dodatečná lineární omezení m_1, m_2 taková, že úloha $(\{m_1, m_2\}, \boldsymbol{c})$ je omezená.

Výstup: Optimální řešení úlohy $(H \cup \{m_1, m_2\}, \boldsymbol{c})$, případně oznámení, že úloha nemá přípustné řešení.

$v_0 \leftarrow$ průnik hraničních přímek polorovin m_1 a m_2;
for $i = 1$ **to** n **do**
  if $v_{i-1} \in h_i$ **then**
    $v_i \leftarrow v_{i-1}$;
  else
    if $a_{i2} \neq 0$ **then**
      for $j = 1$ **to** $i - 1$ **do**
        $D_j \leftarrow \frac{a_{j1}a_{i2} - a_{j2}a_{i1}}{a_{i2}}$; $E_j \leftarrow b_j - \frac{a_{j2}b_i}{a_{i2}}$;
    else
      for $j = 1$ **to** $i - 1$ **do**
        $D_j \leftarrow a_{j2}$; $E_j \leftarrow b_j - \frac{a_{j1}b_i}{a_{i1}}$;
    $z_l \leftarrow -\infty$; $z_r \leftarrow \infty$;
    for $j = 1$ **to** $i - 1$ **do**
      if $D_j > 0$ **then**
        $z_r \leftarrow min(z_r, \frac{E_j}{D_j})$;
      else if $D_j < 0$ **then**
        $z_l \leftarrow max(z_l, \frac{E_j}{D_j})$;
      else
        if $E_j < 0$ **then**
          return Nemá přípustné řešení
    if $z_l = -\infty$ **then**
      $z \leftarrow z_r$;
    else if $z_r = \infty$ **then**
      $z \leftarrow z_l$;
    else if $z_l > z_r$ **then**
      return Nemá přípustné řešení
    else
      if $\overline{f}(z_l) \geq \overline{f}(z_r)$ **then**
        $z \leftarrow z_l$;
      else
        $z \leftarrow z_r$;
    if $a_{i2} \neq 0$ **then**
      $v_i \leftarrow \left[z, \frac{b_i - a_{i1}z}{a_{i2}}\right]$;
    else
      $v_i \leftarrow \left[\frac{b_i}{a_{i1}}, z\right]$;
return v_n

Kapitola 4

Neomezená úloha lineárního programování

Doposud jsme se zabývali omezenou úlohou lineárního programování. Můžeme se však setkat i s úlohami, které jsou neomezené, jak jsme např. viděli na obrázku 1.2(d). V takovém případě nám algoritmus 2 nevrátí správný výsledek. Díky dodatečně přidaným lineárním omezením m_1, m_2 bude výstupem nějaký bod v rovině, přestože neomezená úloha nemá optimální řešení. Pokusíme se tedy odvodit algoritmus, pomocí kterého dokážeme rozpoznat, zda je daná úloha lineárního programování neomezená.

Úloha lineárního programování je neomezená právě tehdy, když existuje polopřímka ρ, která je podmnožinou množiny všech přípustných řešení a podél níž účelová funkce neomezeně roste. Nechť se jedná o polopřímku

$$\rho = \{p + \lambda \boldsymbol{s} \mid \lambda \in \mathbb{R}; \lambda \geq 0\},$$

kde $p = [p_x, p_y]$ je její počáteční bod a $\boldsymbol{s} = (s_x, s_y)$ její směrový vektor.

Pro každé i označme $\boldsymbol{n_i}$ normálový vektor přímky l_i, který směřuje do poloroviny h_i.

Věta 4.1. *Úloha lineárního programování $(H, \boldsymbol{c})$ je neomezená právě tehdy, když existuje vektor $\boldsymbol{s}$ splňující $\langle \boldsymbol{s}, \boldsymbol{c} \rangle > 0$, $\langle \boldsymbol{s}, \boldsymbol{n_i} \rangle \geq 0$ pro každé $i \in \{1, \cdots, n\}$ a současně $\bigcap H' \neq \emptyset$, kde $H' = \{h_i \in H \mid \langle \boldsymbol{s}, \boldsymbol{n_i} \rangle = 0\}$.*

Důkaz. Nechť úloha $(H, \boldsymbol{c})$ je neomezená. Pak existuje polopřímka ρ s výše uvedenými vlastnostmi a se směrovým vektorem $\boldsymbol{s}$. Účelová funkce podél polopřímky ρ neomezeně roste, proto musí vektory $\boldsymbol{s}$ a $\boldsymbol{c}$ svírat úhel v intervalu $\langle 0, \frac{\pi}{2} \rangle$. Protože pro úhel φ svíraný vektory $\boldsymbol{s}$ a $\boldsymbol{c}$ platí

$$\cos \varphi = \frac{\langle \boldsymbol{s}, \boldsymbol{c} \rangle}{\|\boldsymbol{s}\| \cdot \|\boldsymbol{c}\|},$$

z vlastností funkce cos dostáváme $\langle \boldsymbol{s}, \boldsymbol{c} \rangle > 0$. Protože polopřímka ρ leží v množině všech přípustných řešení úlohy $(H, \boldsymbol{c})$, musí vektor $\boldsymbol{s}$ svírat s každým z normálových vektorů $\boldsymbol{n_i}$ úhel v intervalu $\langle 0, \frac{\pi}{2} \rangle$. Analogicky jako výše tedy dostáváme $\langle \boldsymbol{s}, \boldsymbol{n_i} \rangle \geq 0$.

Úloha $(H, \boldsymbol{c})$ je neomezená, a proto je její množina přípustných řešení neprázdná, tj. $\bigcap H \neq \emptyset$. Protože $H' \subseteq H$, je také $\bigcap H' \neq \emptyset$. Vektor $\boldsymbol{s}$ je tedy hledaným vektorem s požadovanými vlastnostmi.

Nechť nyní existuje vektor $\boldsymbol{s}$, pro který platí $\langle \boldsymbol{s}, \boldsymbol{c} \rangle > 0$, $\langle \boldsymbol{s}, \boldsymbol{n_i} \rangle \geq 0$, $i \in \{1, \cdots, n\}$ a $\bigcap H' \neq \emptyset$. Z posledního vztahu vyplývá, že existuje bod $p_0 \in \bigcap H'$. Uvažme polopřímku $\rho_0 = \{p_0 + \lambda \boldsymbol{s} \mid \lambda \in \mathbb{R}; \lambda \geq 0\}$. Pro každou polorovinu h_i z H' je normálový vektor $\boldsymbol{n_i}$ její hraniční přímky l_i kolmý na vektor $\boldsymbol{s}$ (neboť $\langle \boldsymbol{s}, \boldsymbol{n_i} \rangle = 0$), a tedy polopřímka ρ_0 je s přímkou l_i rovnoběžná. Proto leží celá polopřímka ρ_0 v každé z polorovin h_i z H'. Pro každou polorovinu h_i z $H \setminus H'$ je $\langle \boldsymbol{s}, \boldsymbol{n_i} \rangle > 0$, a tedy vektory $\boldsymbol{s}$ a $\boldsymbol{n_i}$ svírají úhel z intervalu $\left\langle 0, \frac{\pi}{2} \right)$. Z toho vyplývá, že pro každou polorovinu h_i z $H \setminus H'$ existuje $\lambda_i \in \mathbb{R}, \lambda_i \geq 0$ takové, že $(p_0 + \lambda \boldsymbol{s}) \in h_i$ pro každé $\lambda \geq \lambda_i$. Označíme-li $\lambda_{max} = \max\{\lambda_i \mid h_i \in H \setminus H'\}$, pak $(p_0 + \lambda \boldsymbol{s}) \in \bigcap H$ pro každé $\lambda \geq \lambda_{max}$. Zvolme $p = p_0 + \lambda_{max} \boldsymbol{s}$. Pak polopřímka

$$\rho = \{p + \lambda \boldsymbol{s} \mid \lambda \in \mathbb{R}; \lambda \geq 0\}$$

je podmnožinou množiny všech přípustných řešení úlohy $(H, \boldsymbol{c})$ a zároveň účelová funkce podél ní neomezeně roste, neboť $\langle \boldsymbol{s}, \boldsymbol{c} \rangle > 0$. Úloha $(H, \boldsymbol{c})$ je tedy neomezená. □

Podaří-li se nám najít vektor splňující podmínky věty 4.1, je úloha neomezená. V takovém případě budeme požadovat, aby byla výstupem algoritmu polopřímka svědčící o neomezenosti úlohy, tj. polopřímka, která leží v množině všech přípustných řešení a podél níž účelová funkce neomezeně roste. Tuto polopřímku můžeme sestrojit postupem uvedeným v druhé části důkazu věty 4.1. Pokud dokážeme, že neexistuje vektor splňující podmínky věty 4.1, jedná se o omezenou úlohu lineárního programování.

Podívejme se nyní, jak vypadají jednotlivé složky vektoru $\boldsymbol{n_i}$, který jsme zadefinovali jako normálový vektor hraniční přímky l_i poloroviny h_i směřující do poloroviny h_i. Víme, že přímka l_i je dána rovnicí

$$a_{i1}x + a_{i2}y = b_i.$$

Protože se jedná o obecnou rovnici přímky, je vektor (a_{i1}, a_{i2}) normálovým vektorem přímky l_i. Normálovým vektorem je však také vektor $(-a_{i1}, -a_{i2})$. Jelikož nám nezáleží na velikosti vektoru, ale pouze na směru, stačí nám zjistit, který z těchto vektorů směřuje do poloroviny h_i. Normálový vektor hraniční přímky nezávisí na posunutí poloroviny. Proto můžeme místo poloroviny $h_i = \{(x, y) \in \mathbb{R}^2 \mid a_{i1}x + a_{i2}y \leq b_i\}$ uvažovat polorovinu $h_i' = \{(x, y) \in \mathbb{R}^2 \mid a_{i1}x + a_{i2}y \leq 0\}$. Normálový vektor zůstane stejný, navíc směřuje do poloroviny h_i právě tehdy, když směřuje do poloroviny h_i'. Hraniční přímka l_i' poloroviny h_i' prochází počátkem soustavy souřadnic. Umístíme-li počáteční bod vektoru (a_{i1}, a_{i2}), respektive $(-a_{i1}, -a_{i2})$, do počátku soustavy souřadnic, je koncovým bodem tohoto vektoru bod $[a_{i1}, a_{i2}]$, respektive $[-a_{i1}, -a_{i2}]$. Přičemž vektor vycházející z počátku soustavy souřadnic směřuje do poloroviny h_i' právě tehdy, když v ní leží jeho koncový bod. Dosazením obou koncových bodů do

nerovnice poloroviny h_i' získáváme:

$$\begin{aligned} a_{i1}^2 + a_{i2}^2 &\leq 0, \\ -(a_{i1}^2 + a_{i2}^2) &\leq 0. \end{aligned}$$

První nerovnost není splněna nikdy[1], druhá platí vždy. Proto můžeme za normálový vektor $\boldsymbol{n_i}$ směřující do poloroviny h_i považovat vektor $(-a_{i1}, -a_{i2})$.

Na základě věty 4.1 hledejme vektor $\boldsymbol{s} = (s_x, s_y)$ splňující

$$\begin{aligned} \langle \boldsymbol{s}, \boldsymbol{c} \rangle &> 0, \\ \langle \boldsymbol{s}, \boldsymbol{n_i} \rangle &\geq 0, \quad i \in \{1, \ldots, n\}. \end{aligned} \tag{4.1}$$

S využitím výše odvozených normálových vektorů $\boldsymbol{n_i} = (-a_{i1}, -a_{i2})$ a rozepsáním skalárních součinů dostáváme ekvivalentní vztahy

$$c_x s_x + c_y s_y > 0, \tag{4.2}$$

$$-a_{i1} s_x - a_{i2} s_y \geq 0, \quad i \in \{1, \cdots, n\}. \tag{4.3}$$

Zřejmě pokud vektor $\boldsymbol{s}$ splňuje tyto vztahy, pak je splňuje také vektor $\lambda \boldsymbol{s}$ pro libovolné $\lambda \in \mathbb{R}, \lambda > 0$. Jelikož nám nezáleží na velikosti vektorů, ale pouze na jejich směru, můžeme nerovnici (4.2) nahradit rovnicí

$$c_x s_x + c_y s_y = 1. \tag{4.4}$$

Tím nijak neomezíme možný směr vektoru $\boldsymbol{s}$, omezíme pouze jeho velikost, a to tak, že požadujeme, aby jeho koncový bod ležel na přímce dané rovnicí (4.4) (počáteční bod leží v počátku soustavy souřadnic). Ke každému vektoru $\boldsymbol{s}$ splňujícímu nerovnice (4.2) a (4.3) existuje vhodné λ takové, že vektor $\lambda \boldsymbol{s}$ splňuje rovnici (4.4) a nerovnice (4.3). Konkrétně při volbě

$$\lambda = \frac{1}{c_x s_x + c_y s_y}$$

dostáváme pro vektor $\boldsymbol{u} = \lambda \boldsymbol{s}$

$$c_x u_x + c_y u_y = c_x \frac{s_x}{c_x s_x + c_y s_y} + c_y \frac{s_y}{c_x s_x + c_y s_y} = \frac{c_x s_x + c_y s_y}{c_x s_x + c_y s_y} = 1.$$

Vektor $\boldsymbol{u}$ tedy splňuje rovnici (4.4). Navíc z platnosti nerovnic (4.3) pro vektor $\boldsymbol{s}$ získáme vynásobením každé z nerovnic číslem λ platnost týž nerovnic pro vektor $\boldsymbol{u}$. Naopak každý vektor $\boldsymbol{s}$ splňující rovnici (4.4) automaticky splňuje nerovnici (4.2). Nahrazení nerovnice (4.2) rovnicí (4.4) tedy neznamená žádné nové omezení pro směr vektoru $\boldsymbol{s}$.

Předpokládejme nyní, že $c_y \neq 0$. Pak vyjádřením s_y z rovnice (4.4) získáváme vztah

$$s_y = \frac{1}{c_y} - \frac{c_x}{c_y} s_x. \tag{4.5}$$

[1]Na začátku jsme požadovali, aby alespoň jedno z čísel a_{i1}, a_{i2} bylo nenulové.

Dosaďme za s_y do každé z nerovnic (4.3) a upravujme.

$$-a_{i1}s_x - \frac{a_{i2}}{c_y} + \frac{a_{i2}c_x}{c_y}\,s_x \geq 0$$
$$\left(\frac{a_{i2}c_x}{c_y} - a_{i1}\right) s_x \geq \frac{a_{i2}}{c_y}$$

Označme

$$F_i^x = \frac{a_{i2}c_x}{c_y} - a_{i1}, \qquad G_i^x = \frac{a_{i2}}{c_y}.$$

Pokud existuje $j \in \{1, \ldots, n\}$ takové, že $F_j^x = 0$ a současně $G_j^x > 0$, pak neexistuje vektor $\boldsymbol{s}$, který by splňoval vztahy (4.1), a úloha $(H, \boldsymbol{c})$ je podle věty 4.1 omezená. V opačném případě dostáváme následující omezení pro s_x.

$$\begin{aligned} s_x &\geq \frac{G_i^x}{F_i^x}, \quad i \in I_l, \\ s_x &\leq \frac{G_i^x}{F_i^x}, \quad i \in I_r, \end{aligned} \tag{4.6}$$

kde I_l a I_r jsou následující množiny indexů:

$$I_l = \left\{i \in \mathbb{N} \mid 1 \leq i \leq n \wedge F_i^x > 0\right\},$$
$$I_r = \left\{i \in \mathbb{N} \mid 1 \leq i \leq n \wedge F_i^x < 0\right\}.$$

Označíme-li

$$x_l = \begin{cases} \max\left\{\frac{G_i^x}{F_i^x} \;\middle|\; i \in I_l\right\}, & \text{jestliže } I_l \neq \emptyset, \\ -\infty, & \text{jestliže } I_l = \emptyset, \end{cases}$$
$$x_r = \begin{cases} \min\left\{\frac{G_i^x}{F_i^x} \;\middle|\; i \in I_r\right\}, & \text{jestliže } I_r \neq \emptyset, \\ \infty, & \text{jestliže } I_r = \emptyset, \end{cases}$$

pak všechny nerovnice (4.6) můžeme nahradit dvojicí jim ekvivalentních vztahů

$$s_x \geq x_l, \qquad s_x \leq x_r.$$

Pokud $x_r < x_l$, neexistuje vektor $\boldsymbol{s}$, který by splňoval vztahy (4.1), a úloha $(H, \boldsymbol{c})$ je podle věty 4.1 omezená. V opačném případě získáme vektor $\boldsymbol{s}$ vyhovující vztahům (4.1) volbou libovolného s_x splňujícího $x_l \leq s_x \leq x_r$ a dopočtením s_y ze vztahu (4.5). Např. volbou $s_x = x_l$ (v případě, že $x_l \neq -\infty$) dostáváme vektor

$$\boldsymbol{s} = \left(x_l, \frac{1}{c_y} - \frac{c_x}{c_y}\,x_l\right).$$

Za předpokladu, že $c_y = 0$ dostáváme vyjádřením s_x z rovnice (4.4) vztah $s_x = \frac{1}{c_x}$. Jeho dosazením do každé z nerovnic (4.3) po úpravě získáme

$$-a_{i2}s_y \geq \frac{a_{i1}}{c_x}.$$

Označíme-li

$$F_i^y = -a_{i2}, \qquad G_i^y = \frac{a_{i1}}{c_x},$$

můžeme dále postupovat zcela analogicky jako v případě, že $c_y \neq 0$. Buď zjistíme, že vektor $\boldsymbol{s}$ s požadovanými vlastnostmi neexistuje a úloha $(H, \boldsymbol{c})$ je podle věty 4.1 omezená, anebo získáme vektor $\boldsymbol{s}$ vyhovující vztahům (4.1).

Jestliže jsme získali vektor $\boldsymbol{s}$ splňující vztahy (4.1), stále se ještě nemusí jednat o neomezenou úlohu lineárního programování. Dle věty 4.1 musí být splněna ještě jedna podmínka, a to $\bigcap H' \neq \emptyset$. Pokud tato podmínka splněna není, tj. je-li $\bigcap H' = \emptyset$, je také $\bigcap H = \emptyset$, neboť $H' \subseteq H$, a tedy úloha $(H, \boldsymbol{c})$ nemá přípustné řešení.

Označme I' množinu indexů $I' = \{i \in \mathbb{N} \mid h_i \in H'\}$. Průnik všech polorovin z H' je pak řešením nerovnic

$$a_{i1}x + a_{i2}y \leq b_i, \quad i \in I'.$$

Podle definice H' je

$$\langle \boldsymbol{s}, \boldsymbol{n_i} \rangle = -a_{i1}s_x - a_{i2}s_y = 0, \quad i \in I'.$$

Normálový vektor hraniční přímky l_i libovolné poloroviny h_i z H' je tedy kolmý na vektor $\boldsymbol{s}$. To znamená, že hraniční přímky všech polorovin z H' jsou rovnoběžné s vektorem $\boldsymbol{s}$.

V případě, že $s_y \neq 0$, tj. pokud není $\boldsymbol{s}$ rovnoběžný s osou x, můžeme místo polorovin $h_i, i \in I'$ uvažovat jejich průniky s osou x, což jsou polopřímky určené nerovnicemi

$$a_{i1}x \leq b_i, \quad i \in I'. \tag{4.7}$$

Vzhledem k tomu, že hraniční přímky všech polorovin $h_i, i \in I'$ jsou rovnoběžné, mají všechny tyto polopřímky neprázdný průnik právě tehdy, když mají neprázdný průnik všechny poloroviny $h_i, i \in I'$.

Situace je opět zcela analogická té, se kterou jsme se již v tomto textu několikrát setkali. Hledáme však pouze množinu všech přípustných řešení jednodimenzionální úlohy lineárního programování, aniž by nás zajímala její účelová funkce či optimální řešení. Nerovnice (4.7) můžeme nahradit dvojicí nerovnic

$$x \geq x_l, \qquad x \leq x_r,$$

kde

$$x_l = \max\left(\left\{\frac{b_i}{a_{i1}} \;\middle|\; i \in I' \wedge a_{i1} < 0\right\} \cup \{-\infty\}\right),$$
$$x_r = \min\left(\left\{\frac{b_i}{a_{i1}} \;\middle|\; i \in I' \wedge a_{i1} > 0\right\} \cup \{\infty\}\right).$$

Jestliže $x_l \leq x_r$, je $\bigcap H' \neq \emptyset$ a podle věty 4.1 je úloha $(H, \boldsymbol{c})$ neomezená. V opačném případě je $\bigcap H' = \emptyset$, proto také $\bigcap H = \emptyset$, a tedy úloha $(H, \boldsymbol{c})$ nemá přípustné řešení.

V případě, že $s_y = 0$, tj. $\boldsymbol{s}$ je rovnoběžný s osou x, postupujeme analogicky, s tím rozdílem, že uvažujeme průniky s osou y, což jsou polopřímky určené nerovnicemi

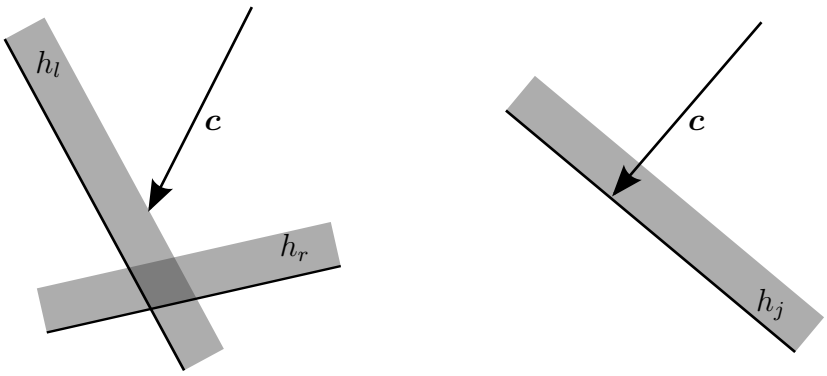
$$a_{i2}y \leq b_i, \quad i \in I'.$$

Uvažujme nyní o omezené úloze lineárního programování. Vraťme se k situaci, kde jsme zjistili, že hledaný vektor $\boldsymbol{s}$ neexistuje a úloha je tedy omezená. Předpokládejme, že $c_y \neq 0$, v opačném případě bychom postupovali zcela analogicky. Omezenost úlohy mohla být prokázána ve dvou odlišných situacích.

První možností je, že omezenost úlohy jsme zjistili na základě platnosti vztahu $x_l > x_r$. Za této situace[2] označme

- h_l takovou polorovinu h_j, že výraz $\frac{G_i^x}{F_i^x}$, $i \in I_l$ nabývá maxima pro $i = j$,
- h_r takovou polorovinu h_j, že výraz $\frac{G_i^x}{F_i^x}$, $i \in I_r$ nabývá minima pro $i = j$.

Uvážíme-li úlohu $(\{h_l, h_r\}, \boldsymbol{c})$, získáme pro ni stejné závěry o neexistenci vhodného vektoru $\boldsymbol{s}$ jako pro úlohu $(H, \boldsymbol{c})$. Podle věty 4.1 je tedy i úloha $(\{h_l, h_r\}, \boldsymbol{c})$ omezená. Poloroviny h_l a h_r proto představují svědky omezenosti úlohy $(H, \boldsymbol{c})$, které můžeme použít místo dodatečných lineárních omezení v algoritmu pro omezenou úlohu lineárního programování. Situace je znázorněna na obrázku 4.1(a).



(a) Poloroviny h_l a h_r jsou svědky (b) Jediná polorovina h_j je svědkem

Obrázek 4.1: Svědci omezenosti úlohy $(H, \boldsymbol{c})$

Druhý možný případ, kdy jsme mohli zjistit, že úloha je omezená, nastal, pokud existuje j, pro které je $F_j^x = 0$ a $G_j^x > 0$. Pak vyjádřením F_j^x a G_j^x po úpravě dostáváme:

$$a_{j2}c_x - a_{j1}c_y = \langle c, (a_{j2}, -a_{j1}) \rangle = 0, \tag{4.8}$$

$$\frac{a_{j2}}{c_y} > 0. \tag{4.9}$$

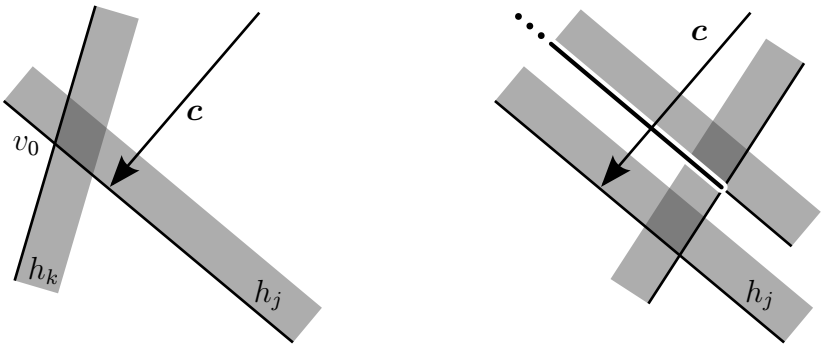
Víme, že (a_{j1}, a_{j2}) je normálový vektor přímky l_j, a proto $(a_{j2}, -a_{j1})$ je její směrový vektor. Ze vztahu (4.8) tedy vyplývá, že vektor $\boldsymbol{c}$ je kolmý na přímku l_j. Navíc na základě vztahu (4.9) platí, že c_y a $-a_{j2}$ mají opačná znaménka. Jelikož $(-a_{j1}, -a_{j2})$ je normálový vektor přímky l_j směřující do poloroviny h_j, vektor $\boldsymbol{c}$ nesměřuje do poloroviny h_j. Nastává situace znázorněná na obrázku 4.1(b).

[2]Tento případ mohl nastat pouze pokud $x_l \neq -\infty$ a $x_r \neq \infty$.

Z výše uvedeného postupu vyplývá, že ani pro úlohu lineárního programování $(\{h_j\}, \boldsymbol{c})$ neexistuje vektor $\boldsymbol{s}$ požadovaných vlastností, a proto je podle věty 4.1 i úloha $(\{h_j\}, \boldsymbol{c})$ omezená. Polorovina h_j je tedy svědkem toho, že úloha $(H, \boldsymbol{c})$ je omezená. Situace je však odlišná od té, kdy jsme měli k dispozici dva svědky. Může se stát, že úloha je sice omezená, ale nemá lexikograficky nejmenší řešení.

Jelikož přímka l_j je kolmá na vektor $\boldsymbol{c}$, je hodnota účelové funkce ve všech bodech ležících na přímce l_j stejná. Body na přímce l_j však můžeme lexikograficky uspořádat. Pokud se nám podáří nalézt takovou polorovinu h_k, že úloha $(\{h_j, h_k\}, \boldsymbol{c})$ má lexikograficky nejmenší řešení, pak v případě, že množina všech přípustných řešení původní úlohy $(H, \boldsymbol{c})$ je neprázdná, musí mít lexikograficky nejmenší řešení i původní úloha. V takovém případě můžeme místo dodatečných omezení m_1, m_2 v algoritmu 2 použít poloroviny h_j, h_k a za počáteční řešení v_0 považovat průnik jejich hraničních přímek. Situace je znázorněna na obrázku 4.2(a).

Ukážeme-li, že neexistuje polorovina $h_k \in H \setminus \{h_j\}$ výše uvedených vlastností, znamená to, že úloha nemá lexikograficky nejmenší řešení, případně nemá přípustné řešení. Označme H'' množinu všech polorovin z H, jejichž hraniční přímky jsou rovnoběžné s přímkou l_j (a tedy kolmé na vektor $\boldsymbol{c}$). Jestliže je $\bigcap H'' = \emptyset$, pak úloha H nemá přípustné řešení, neboť $H'' \subseteq H$. Pokud však $\bigcap H'' \neq \emptyset$, úloha H má nekonečně mnoho přípustných řešení a nemá lexikograficky nejmenší optimální řešení. Příklad takové úlohy je znázorněn na obrázku 4.2(b).



(a) Existuje lexikograficky nejmenší řešení

(b) Neexistuje lexikograficky nejmenší řešení

Obrázek 4.2: Existence lexikograficky nejmenšího řešení v případě poloroviny s hraniční přímkou kolmou na vektor $\boldsymbol{c}$

Hledejme tedy polorovinu h_k, jejíž hraniční přímka protíná přímku l_j a současně platí, že úloha $(\{h_j, h_k\}, \boldsymbol{c})$ má lexikograficky nejmenší řešení. Uvažme průniky přímky l_j s polorovinami z $H \setminus \{h_j\}$. V případě, že přímka l_j není kolmá na osu x, můžeme nám známým způsobem řešit jednodimenzionální úlohu lineárního programování pro x-ové souřadnice těchto průniků. V opačném případě řešíme analogickou úlohu pro y-ové souřadnice. Postup je téměř stejný jako v případě odvození algoritmu pro omezenou úlohu lineárního programování, vycházíme proto ze vztahů (3.2) a (3.3). Nemusíme však řešit celou úlohu. Zajímá nás pouze, jestli

některý z průniků má podobu polopřímky omezené zleva, neboť hledáme lexikograficky nejmenší řešení. Jakmile najdeme polorovinu h_k, jejíž průnik s přímkou l_j je polopřímka omezená zleva, můžeme přejít k algoritmu 2. Pokud žádný z průniků nemá podobu polopřímky omezené zleva, ověříme, zda $\bigcap H'' = \emptyset$. To je stejný problém jako zjištění, zda je $\bigcap H' = \emptyset$, s tím rozdílem, že místo vektoru $\boldsymbol{s}$ použijeme vektor $(c_y, -c_x)$, který je kolmý na vektor $\boldsymbol{c}$. Oba problémy můžeme řešit pomocí algoritmu 5. Uvedený postup ke zjištění existence lexikograficky nejmenšího řešení je využit v algoritmu 4.

Pokud úloha má přípustné řešení, ale nemá lexikograficky nejmenší optimální řešení, budeme požadovat jako výstup lexikograficky největší optimální řešení. Pokud ani to neexistuje, výstupem bude nějaké optimální řešení. To získáme na základě algoritmu 5. Můžeme použít velmi podobný postup jako v předchozím odstavci. Hledáme tedy polorovinu h_k, jejíž průnik s přímkou l_j je polopřímka omezená zprava. K tomu můžeme použít algoritmus 4, který mírně modifikujeme, a to tak, že nerovnosti na řádcích 3 a 7 zaměníme za opačné. V případě, že nalezneme polorovinu h_k, jejíž hraniční přímka protíná přímku l_j a současně platí, že úloha $(\{h_j, h_k\}, \boldsymbol{c})$ má lexikograficky největší řešení, můžeme využít upravený algoritmus 2 k řešení této úlohy. Záměnou neostré nerovnosti na řádku 28 algoritmu 2 za ostrou zajistíme, aby algoritmus hledal lexikograficky největší optimální řešení místo nejmenšího. Jelikož se jedná jen o drobné úpravy, nebudeme zde upravené algoritmy explicitně uvádět. Pro použití v následných algoritmech je zde pouze pojmenujme, a to FindLexBoundMax$(H, \boldsymbol{c}, j)$ a BoundedLPMax$(H, \boldsymbol{c}, m_1, m_2)$.

Doposud získané poznatky o řešení úlohy lineárního programování v rovině můžeme shrnout ve schématu na obrázku 4.3. Zbývá tedy uvést algoritmy řešící jednotlivé podproblémy z tohoto schématu. Ty však můžeme snadno formulovat na základě výsledků získaných v této kapitole.

Věta 4.2. *Časová složitost algoritmu 3 je $\mathcal{O}(n)$.*

Důkaz. Algoritmus obsahuje tři **for** cykly, přičemž při vykonání algoritmu proběhnou vždy právě dva. Časová složitost každého z těchto **for** cyklů je $\mathcal{O}(n)$. Ostatní části algoritmu mají konstantní časovou složitost. Proto je celková časová složitost algoritmu 3 rovna $\mathcal{O}(n)$. □

Věta 4.3. *Časová složitost algoritmu 4 je $\mathcal{O}(n)$.*

Důkaz. Zřejmě vždy proběhne maximálně jeden **for** cyklus časové složitosti $\mathcal{O}(n)$, a proto je i celková časová složitost algoritmu lineární. □

Věta 4.4. *Časová složitost algoritmu 5 je $\mathcal{O}(n)$.*

Důkaz. Algoritmus obsahuje dva nevnořené cykly **for**. Časová složitost prvního cyklu je $\mathcal{O}(n)$. Mohutnost množiny I' je nejvýše n, a proto i časová složitost cyklu **for all** je $\mathcal{O}(n)$. Ostatní části algoritmu mají konstantní časovou složitost. Celková časová složitost algoritmu 5 je tedy $\mathcal{O}(n)$. □

Existuje $\boldsymbol{s}$ splňující (4.1)?

- ne, 2 svědci h_l a h_r → Použij svědky jako m_1, m_2 a spusť algoritmus 2.
  - *řešení*
  - *úloha nemá přípustné řešení*
- ne, 1 svědek h_j → Existuje h_k tak, že $(\{h_j, h_k\}, \boldsymbol{c})$ má lexikograficky nejmenší řešení?
  - ano → Použij h_j, h_k jako m_1, m_2 a spusť algoritmus 2.
    - *řešení*
    - *úloha nemá přípustné řešení*
  - ne → Existuje h_k tak, že $(\{h_j, h_k\}, \boldsymbol{c})$ má lexikograficky největší řešení?
    - ano → Použij h_j, h_k jako m_1, m_2 a spusť modifikovaný algoritmus 2.
      - *lexikograficky největší řešení (nejmenší neexistuje)*
      - *úloha nemá přípustné řešení*
    - ne → Je $\bigcap H'' = \emptyset$?
      - ano → *úloha nemá přípustné řešení*
      - ne → *nějaké optimální řešení (lex. nejm. ani nejv. neexistuje)*
- ano → Je $\bigcap H' = \emptyset$?
  - ano → *úloha nemá přípustné řešení*
  - ne → *úloha je neomezená*

Obrázek 4.3: Schéma shrnující doposud získané poznatky

Algoritmus 3 Algoritmus k nalezení vektoru $\boldsymbol{s}$ splňujícího podmínky (4.1)

FINDVECTOR$(H, \boldsymbol{c})$

Vstup: Úloha lineárního programování $(H, \boldsymbol{c})$.

Výstup: Vektor $\boldsymbol{s}$ splňující podmínky (4.1), pokud takový existuje. V opačném případě algoritmus vrací hodnotu nil, navíc do proměnných[3] wit_1, wit_2 vloží indexy svědků omezenosti úlohy, v případě jednoho svědka vloží do wit_2 hodnotu nil.

1: **if** $c_y \neq 0$ **then**
2: **for** $i = 1$ **to** n **do**
3: $F_i \leftarrow \frac{a_{i2}c_x}{c_y} - a_{i1}$; $G_i \leftarrow \frac{a_{i2}}{c_y}$;
4: **else**
5: **for** $i = 1$ **to** n **do**
6: $F_i \leftarrow -a_{i2}$; $G_i \leftarrow \frac{a_{i1}}{c_x}$;
7: $w_l \leftarrow -\infty$; $w_r \leftarrow \infty$;
8: $wit_1 \leftarrow nil$; $wit_2 \leftarrow nil$;
9: **for** $i = 1$ **to** n **do**
10: **if** $(F_i > 0) \wedge \left(\frac{G_i}{F_i} > w_l\right)$ **then**
11: $w_l \leftarrow \frac{G_i}{F_i}$; $wit_1 \leftarrow i$;
12: **else if** $(F_i < 0) \wedge \left(\frac{G_i}{F_i} < w_r\right)$ **then**
13: $w_r \leftarrow \frac{G_i}{F_i}$; $wit_2 \leftarrow i$;
14: **else**
15: **if** $(F_i = 0) \wedge (G_i > 0)$ **then**
16: $wit_1 \leftarrow i$; $wit_2 \leftarrow nil$;
17: **return** nil
18: **if** $w_l > w_r$ **then**
19: **return** nil
20: **else**
21: **if** $w_l \neq -\infty$ **then**
22: $w \leftarrow w_l$;
23: **else if** $w_r \neq \infty$ **then**
24: $w \leftarrow w_r$;
25: **else**
26: $w \leftarrow 0$;
27: **if** $c_y \neq 0$ **then**
28: **return** $\left(w; \frac{1}{c_y} - \frac{c_x}{c_y} w\right)$
29: **else**
30: **return** $\left(\frac{1}{c_x}; w\right)$

Algoritmus 4 Algoritmus ke zjištění existence lexikograficky nejmenšího řešení

FindLexBound$(H, \boldsymbol{c}, j)$

Vstup: Úloha lineárního programování $(H, \boldsymbol{c})$, index j poloroviny h_j s hraniční přímkou kolmou na vektor $\boldsymbol{c}$.

Výstup: Index k poloroviny h_k takové, že úloha $(\{h_j, h_k\}, \boldsymbol{c})$ má lexikograficky nejmenší řešení, nebo hodnota nil, pokud taková polorovina neexistuje.

```
1: if a_{j2} ≠ 0 then
2:   for all i ∈ {1,...,n} \ {j} do
3:     if (a_{i1}a_{j2} − a_{i2}a_{j1}) / a_{j2} < 0 then
4:       return i
5: else
6:   for all i ∈ {1,...,n} \ {j} do
7:     if a_{i2} < 0 then
8:       return i
9: return nil
```

Jestliže je úloha neomezená, požadujeme, aby výstupem algoritmu byla polopřímka ρ, která je celá obsažena v množině všech přípustných řešení a podél níž účelová funkce neomezeně roste. Předpokládejme tedy, že úloha je neomezená, na základě algoritmu 3 jsme získali vektor $\boldsymbol{s} = (s_x, s_y)$ splňující podmínky (4.1) a z algoritmu 5 máme bod $p_0 \in \bigcap H'$, $p_0 = [p_{0x}, p_{0y}]$.

Vycházejme z důkazu věty 4.1. Pro každou polorovinu $h_i \in H \setminus H'$ existuje λ_i takové, že $p_0 + \lambda \boldsymbol{s}$ leží v polorovině h_i pro libovolné $\lambda \geq \lambda_i$. Zřejmě toto splňuje takové λ_i, že $p_0 + \lambda_i \boldsymbol{s}$ je průsečík přímky l_i s polopřímkou $\rho_0 = \{p_0 + \lambda \boldsymbol{s} \mid \lambda \in \mathbb{R}; \lambda \geq 0\}$, pokud existuje. V opačném případě leží v polorovině h_i celá polopřímka ρ_0 neboť vektor $\boldsymbol{s}$ splňuje vlastnosti (4.1). Průsečík λ_i snadno vypočteme. Dosazením do rovnice přímky l_i získáme rovnici

$$a_{i1}(p_{0x} + \lambda_i s_x) + a_{i2}(p_{0y} + \lambda_i s_y) = b_i,$$

z níž po úpravách dostaneme výraz

$$\lambda_i = \frac{b_i - (a_{i1}p_{0x} + a_{i2}p_{0y})}{a_{i1}s_x + a_{i2}s_y}.$$

Poznamenejme, že pro libovolné i, pro něž $h_i \in H \setminus H'$, je výraz $a_{i1}s_x + a_{i2}s_y$ nenulový. Poloroviny $h_i \in H$, pro které je tento výraz nulový, mají hraniční přímky rovnoběžné s vektorem $\boldsymbol{s}$, a tudíž patří do H'.

Hledanou polopřímkou je tedy polopřímka

$$\rho = \{p + \lambda \boldsymbol{s} \mid \lambda \in \mathbb{R}; \lambda \geq 0\},$$

kde $p = p_0 + \lambda_{max}\boldsymbol{s}$, $\lambda_{max} = \max\{\lambda_i \mid i \in \{1, \ldots, n\} \setminus I'\}$.

K nalezení polopřímky ρ formulujme algoritmus 6. Jeho časová složitost je zřejmě lineární.

[3] Proměnné wit_1, wit_2 chápeme jako globální proměnné.

Algoritmus 5 Algoritmus zjišťující, zda je průnik podmnožiny G množiny H prázdný

IsEmpty$(H, \boldsymbol{c}, \boldsymbol{s})$

Vstup: Úloha lineárního programování $(H, \boldsymbol{c})$ a vektor $\boldsymbol{s}$ zadávající podmnožinu $G = \{h_i \in H \mid \langle \boldsymbol{s}, \boldsymbol{n_i} \rangle = 0\}$.

Výstup: Jestliže $\bigcap G \neq \emptyset$, je výstupem libovolný bod[4] $p_0 \in \bigcap G$. Pokud $\bigcap G = \emptyset$, výstupem je hodnota *nil*.

1: $I' \leftarrow \emptyset$;
2: **for** i=1 **to** n **do**
3: **if** $a_{i1}s_x + a_{i2}s_y = 0$ **then**
4: $I' \leftarrow I' \cup \{i\}$;
5: **if** $s_y \neq 0$ **then**
6: $j \leftarrow 1$; $c^* \leftarrow c_x$;
7: **else**
8: $j \leftarrow 2$; $c^* \leftarrow c_y$;
9: $w_l \leftarrow -\infty$; $w_r \leftarrow \infty$;
10: **for all** $i \in I'$ **do**
11: **if** $a_{ij} > 0$ **then**
12: $w_r \leftarrow \min(w_r, \frac{b_i}{a_{ij}})$;
13: **else if** $a_{ij} < 0$ **then**
14: $w_l \leftarrow \max(w_l, \frac{b_i}{a_{ij}})$;
15: **if** $w_l > w_r$ **then**
16: **return** *nil*
17: **else**
18: **if** $w_l \neq -\infty \wedge w_r \neq \infty$ **then**
19: **if** $c^* \leq 0$ **then**
20: $w \leftarrow w_l$;
21: **else**
22: $w \leftarrow w_r$;
23: **else if** $w_l \neq -\infty$ **then**
24: $w \leftarrow w_l$;
25: **else if** $w_r \neq \infty$ **then**
26: $w \leftarrow w_r$;
27: **else**
28: $w \leftarrow 0$;
29: **if** $s_y \neq 0$ **then**
30: **return** $[w, 0]$
31: **else**
32: **return** $[0, w]$

Algoritmus 6 Algoritmus k nalezení polopřímky, na níž je úloha neomezená

FINDRHO$(H, \boldsymbol{c}, \boldsymbol{s}, p_0)$

Vstup: Neomezená úloha lineárního programování $(H, \boldsymbol{c})$, vektor $\boldsymbol{s}$ splňující podmínky (4.1) a bod $p_0 \in \bigcap H'$.

Výstup: Bod p takový, že polopřímka $\rho = \{p + \lambda \boldsymbol{s} \mid \lambda \in \mathbb{R}; \lambda \geq 0\}$ leží v množině všech přípustných řešení a účelová funkce podél ní neomezeně roste.

1: $\lambda_{max} \leftarrow 0$;
2: **for all** $i \in \{1, \ldots, n\} \setminus I'$ **do**
3: $\quad \lambda_{max} \leftarrow \max\left\{\lambda_{max}, \frac{b_i - (a_{i1}p_{0x} + a_{i2}p_{0y})}{a_{i1}s_x + a_{i2}s_y}\right\}$;
4: **return** $p_0 + \lambda_{max}\boldsymbol{s}$

S využitím doposud odvozených algoritmů jsme schopni uvést algoritmus 7 k řešení obecné úlohy lineárního programování v rovině.

Věta 4.5. *Časová složitost algoritmu 7 je* $\mathcal{O}(n^2)$.

Důkaz. Tvrzení přímo vyplývá z časové složitosti algoritmů využitých v algoritmu 7. □

[4]V situaci, kdy zjišťujeme, zda-li je $\bigcap H'' = \emptyset$, je tento bod libovolným optimálním řešením.

Algoritmus 7 Algoritmus k řešení úlohy lineárního programování v rovině

LP$(H, \boldsymbol{c})$

Vstup: Úloha lineárního programování $(H, \boldsymbol{c})$.

Výstup: Lexikograficky nejmenší optimální řešení úlohy (největší, pokud neexistuje, či nějaké, pokud ani nejv. neexistuje), zpráva o neomezenosti úlohy a polopřímka, na níž je úloha neomezená, nebo zpráva o tom, že úloha nemá přípustné řešení.

1: $\boldsymbol{s} \leftarrow$ FindVector$(H, \boldsymbol{c})$;
2: **if** $\boldsymbol{s} = nil$ **then**
3: **if** $wit_2 \neq nil$ **then**
4: **return** BoundedLP$(H \setminus \{h_{wit_1}, h_{wit_2}\}, \boldsymbol{c}, h_{wit_1}, h_{wit_2})$
5: **else**
6: $k \leftarrow$ FindLexBound$(H, \boldsymbol{c}, wit_1)$;
7: **if** $k \neq nil$ **then**
8: **return** BoundedLP$(H \setminus \{h_j, h_k\}, \boldsymbol{c}, h_{wit_1}, h_k)$
9: **else**
10: $k \leftarrow$ FindLexBoundMax$(H, \boldsymbol{c}, wit_1)$;
11: **if** $k \neq nil$ **then**
12: **return** BoundedLPMax$(H \setminus \{h_j, h_k\}, \boldsymbol{c}, h_{wit_1}, h_k)$
13: **else**
14: $p \leftarrow$ IsEmpty$(H, \boldsymbol{c}, (c_y, -c_x))$;
15: **if** $p = nil$ **then**
16: **return** Úloha nemá přípustné řešení.
17: **else**
18: **return** p
19: **else**
20: $p_0 \leftarrow$ IsEmpty$(H, \boldsymbol{c}, \boldsymbol{s})$;
21: **if** $p_0 = nil$ **then**
22: **return** Úloha nemá přípustné řešení.
23: **else**
24: $p \leftarrow$ FindRho$(H, \boldsymbol{c}, \boldsymbol{s}, p_0)$
25: **return** Úloha je neomezená, a to na polopřímce $\rho = \{p + \lambda \boldsymbol{s} \mid \lambda \in \mathbb{R}; \lambda \geq 0\}$.

Kapitola 5

Očekávaná časová složitost

Viděli jsme, že v nejhorším případě může být délka výpočtu algoritmu k řešení úlohy lineárního programování kvadratická vzhledem k velikosti vstupu. Délka výpočtu dané úlohy závisí na pořadí polorovin. Pro některá uspořádání může být kvadratická, pro jiná lineární. V této části upravíme algoritmus do podoby náhodnostního algoritmu a budeme se zabývat očekávanou časovou složitostí. Pro danou úlohu poloroviny náhodně uspořádáme tak, že všechny permutace polorovin budou mít stejnou pravděpodobnost zvolení.

K nalezení náhodné permutace daného pole formulujme algoritmus 8. Předpokládáme, že máme k dispozici generátor náhodných čísel RANDOM(k), jehož výstup je náhodné celé číslo z intervalu $\langle 1, k\rangle$, a to každé se stejnou pravděpodobností. Časová složitost algoritmu 8 je zřejmě lineární.

Algoritmus 8 Algoritmus pro náhodnou permutaci

RNDPERM($H, \boldsymbol{c}$)
Vstup: Pole $A = (A[1], \ldots, A[n])$
Výstup: Náhodná permutace pole A

```
1: for i = 1 to n − 1 do
2:     rand ← n − RANDOM(n − i + 1) + 1;
3:     pom ← A[i];
4:     A[i] ← A[rand];
5:     A[rand] ← pom;
6: return A
```

Pomocí procedury pro náhodnou permutaci upravme algoritmus 7 do podoby náhodnostního algoritmu. Jediná změna v algoritmu 9 oproti algoritmu 7 se nachází na řádcích 4, 8 a 12.

Délka výpočtu algoritmu 9 závisí na náhodné volbě permutace polorovin. Příčinou možné kvadratické délky výpočtu je algoritmus pro omezenou úlohu lineárního programování na řádku 4, respektive 8 či 12. Analyzujme tedy délku výpočtu algoritmu 2 v případě náhodnostního algoritmu.

Důvodem kvadratické časové složitosti algoritmu 2 je počet možných vykonání

Algoritmus 9 Náhodnostní algoritmus k řešení úlohy lin. programování v rovině

RndLP$(H, \boldsymbol{c})$

Vstup: Úloha lineárního programování $(H, \boldsymbol{c})$.

Výstup: Lexikograficky nejmenší optimální řešení úlohy (největší, pokud neexistuje, či nějaké, pokud ani nejv. neexistuje), zpráva o neomezenosti úlohy a polopřímka, na níž je úloha neomezená, nebo zpráva o tom, že úloha nemá přípustné řešení.

1: $\boldsymbol{s} \leftarrow$ FindVector$(H, \boldsymbol{c})$;
2: **if** $\boldsymbol{s} = nil$ **then**
3: **if** $wit_2 \neq nil$ **then**
4: **return** BoundedLP(RndPerm$(H \setminus \{h_{wit_1}, h_{wit_2}\}), \boldsymbol{c}, h_{wit_1}, h_{wit_2})$
5: **else**
6: $k \leftarrow$ FindLexBound$(H, \boldsymbol{c}, wit_1)$;
7: **if** $k \neq nil$ **then**
8: **return** BoundedLP(RndPerm$(H \setminus \{h_j, h_k\}), \boldsymbol{c}, h_{wit_1}, h_k)$
9: **else**
10: $k \leftarrow$ FindLexBoundMax$(H, \boldsymbol{c}, wit_1)$;
11: **if** $k \neq nil$ **then**
12: **return** BoundedLPMax(RndPerm$(H \setminus \{h_j, h_k\}), \boldsymbol{c}, h_{wit_1}, h_k)$
13: **else**
14: $p \leftarrow$ IsEmpty$(H, \boldsymbol{c}, (c_y, -c_x))$;
15: **if** $p = nil$ **then**
16: **return** Úloha nemá přípustné řešení.
17: **else**
18: **return** p
19: **else**
20: $p_0 \leftarrow$ IsEmpty$(H, \boldsymbol{c}, \boldsymbol{s})$;
21: **if** $p_0 = nil$ **then**
22: **return** Úloha nemá přípustné řešení.
23: **else**
24: $p \leftarrow$ FindRho$(H, \boldsymbol{c}, \boldsymbol{s}, p_0)$
25: **return** Úloha je neomezená, a to na polopřímce $\rho = \{p + \lambda \boldsymbol{s} \mid \lambda \in \mathbb{R}; \lambda \geq 0\}$.

druhé větve podmínky na řádku 3. Časová složitost i-té iterace cyklu **for** (ř. 2–35) je v závislosti na splnění podmínky na řádku 3 buď konstantní, anebo $\mathcal{O}(i)$. V případě náhodnostního algoritmu je splnění této podmínky náhodnou veličinou. Proto pro $i \in \{1, \ldots, n\}$ označme X_i náhodnou veličinu, která nabývá hodnoty 1, jestliže $v_{i-1} \notin h_i$ (časová složitost i-té iterace je lineární), nebo hodnoty 0, jestliže $v_{i-1} \in h_i$ (časová složitost i-té iterace je konstantní).

Celková časová složitost vykonání druhé větve během všech iterací dohromady je proto

$$\sum_{i=1}^{n} X_i \mathcal{O}(i),$$

zatímco celková časová složitost vykonání první větve během všech iterací je lineární, a tedy časová složitost algoritmu 2 je rovna náhodné veličině

$$O(n) + \sum_{i=1}^{n} X_i \mathcal{O}(i).$$

Definice 5.1. Nechť časová složitost algoritmu A pro daný vstup je náhodná veličina Y, pak *očekávanou časovou složitostí* algoritmu A rozumíme střední hodnotu náhodné veličiny Y, tj. $E(Y)$.

Věta 5.1. *Očekávaná časová složitost algoritmu 9 je* $\mathcal{O}(n)$.

Důkaz. Analyzujme očekávanou časovou složitost algoritmu 2. Jestliže ukážeme, že je lineární, bude lineární i očekávaná časová složitost algoritmus 9, neboť ostatní části algoritmu kromě řádku 4, respektive 8, mají lineární časovou složitost.

Z předchozích výsledků vyplývá, že očekávaná časová složitost algoritmu 9 pro daný vstup velikosti n je[1]

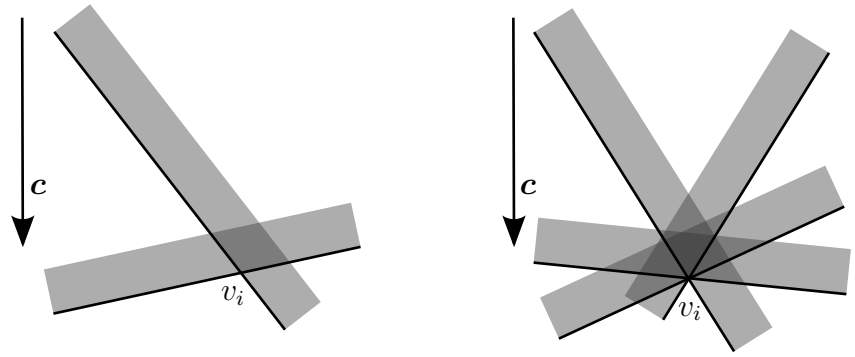
$$E\left(O(n) + \sum_{i=1}^{n} X_i \mathcal{O}(i)\right) = O(n) + \sum_{i=1}^{n} E(X_i)\mathcal{O}(i).$$

Náhodná veličina X_i nabývá hodnot 0, nebo 1. Má tedy alternativní rozdělení pravděpodobnosti, a proto střední hodnota náhodné veličiny X_i je rovna pravděpodobnosti, že náhodná veličina X_i nabývá hodnoty 1, tj.

$$E(X_i) = P(X_i = 1) = P(v_{i-1} \notin h_i).$$

Podívejme se na algoritmus 2 po i iteracích hlavního cyklu. Máme tedy nalezeno optimální řešení v_i úlohy $(H_i, \boldsymbol{c})$. Bod v_i je průsečíkem nejméně dvou z hraničních přímek polorovin z H_i. Odebereme-li jednu polorovinu, pak může dojít ke změně optimálního řešení pouze v případě, že bod v_i leží na hraniční přímce této poloroviny. Pokud v_i leží na hraničních přímkách právě dvou polorovin, pak odebráním jedné z nich dojde ke změně množiny všech přípustných řešení a může dojít ke změně

[1] Využijme linearitu střední hodnoty náhodné veličiny, tedy že pro náhodnou veličinu Y a konstanty $a, b \in \mathbb{R}$ platí $E(a + bY) = a + bE(Y)$.



(a) v_i průsečíkem hraničních přímek právě dvou polorovin

(b) v_i průsečíkem hraničních přímek více než dvou polorovin

Obrázek 5.1: Bod v_i jako průsečík hraničních přímek polorovin

optimálního řešení. Situaci je znázorněna na obrázku 5.1(a). Jestliže v_i je průsečíkem hraničních přímek více než dvou polorovin, pak pouze dvě z nich tvoří hranici množiny všech přípustných řešení, jak ilustruje obrázek 5.1(b). Pouze odebráním jedné z těchto dvou polorovin dojde ke změně množiny všech přípustných řešení a může dojít ke změně optimálního řešení. Odebrání jiné poloroviny, tedy takové, na jejíž hraniční přímce bod v_i neleží, nevede v žádné z předchozích situací ke změně optimálního řešení. Pouze pro dvě poloroviny z H_i tedy platí, že odebráním jedné z nich může dojít ke změně optimálního řešení.

Situace, kdy $v_{i-1} \notin h_i$ nastane právě tehdy, když odebráním poloroviny h_i dojde ke změně optimálního řešení (neboli po $i-1$ iteracích je jiné optimální řešení než po i iteracích). Tedy $v_{i-1} \notin h_i$ může nastat pouze tehdy, pokud h_i je jednou ze dvou výše uvedených polorovin, jejichž odebráním může dojít ke změně optimálního řešení. Jelikož jsou poloroviny uspořádány náhodně, je pravděpodobnost, že h_i je jednou z těchto polorovin, nejvýše $\frac{2}{i}$. Proto

$$E(X_i) = P(v_{i-1} \notin h_i) \leq \frac{2}{i}.$$

Dosazením do vztahu pro očekávanou časovou složitost algoritmu 2 dostáváme

$$O(n) + \sum_{i=1}^{n} E(X_i)\mathcal{O}(i) \leq O(n) + \sum_{i=1}^{n} \frac{2}{i}\mathcal{O}(i) = \mathcal{O}(n),$$

a tedy očekávaná časová složitost je lineární. □

Zdůrazněme, že lineární časová složitost je očekávána pro každý vstup. Nejedná se o průměrnou časovou složitost přes všechny možné vstupy. Očekávání se vztahuje k náhodné volbě obsažené v algoritmu, nikoliv k možnému vstupu. Pro libovolnou úlohu je očekávaná časová složitost algoritmu 9 lineární.

Kapitola 6

Simplexová metoda

K řešení úloh lineárního programování se většinou využívá simplexová metoda. Stručně ji proto v této části popíšeme a srovnáme s náhodnostním algoritmem odvozeným v této práci. Vycházejme z [3, kapitola 2], kde lze také nalézt podrobnější popis simplexové metody.

Abychom mohli řešit úlohu lineárního programování simplexovou metodou, je třeba ji nejprve převést do tzv. *standardního tvaru*, tedy do tvaru

$$max\{\boldsymbol{cx} \mid A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0\}^1.$$

Konkrétně v případě našeho výchozího tvaru (podle definice 1.2) úlohu

$$max\{\boldsymbol{cx} \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$$

převedeme na ekvivalentní úlohu

$$max\left\{(\boldsymbol{c}, -\boldsymbol{c})\begin{pmatrix}\boldsymbol{x'}\\ \boldsymbol{x''}\end{pmatrix} \;\middle|\; (A \mid -A \mid E)\begin{pmatrix}\boldsymbol{x'}\\ \boldsymbol{x''}\\ \boldsymbol{x^*}\end{pmatrix} = \boldsymbol{b}, \begin{pmatrix}\boldsymbol{x'}\\ \boldsymbol{x''}\\ \boldsymbol{x^*}\end{pmatrix} \geq 0\right\}.$$

Původní úlohu jsme transformovali tak, že jsme přidali vektor doplňkových proměnných $\boldsymbol{x}^*$, $\boldsymbol{x}^* \geq 0$ a převedli tak soustavu nerovnic $A\boldsymbol{x} \leq b$ na soustavu rovnic $A\boldsymbol{x} + \boldsymbol{x}^* = b$. Dále jsme každou z původních proměnných x_i nahradili dvojicí nezáporných proměnných x_i' a x_i'' tak, že $x_i = x_i' - x_i''$.

Vidíme, že transformací úlohy lineárního programování v rovině do standardního tvaru získáme úlohu s $n+4$ proměnnými. Chceme-li tedy řešit úlohu lineárního programování v rovině pomocí simplexové metody, musíme řešit úlohu vyšší dimenze.

Množina všech přípustných řešení úlohy lineárního programování je průnikem konečně mnoha afinních poloprostorů[2] a představuje proto polyedr. Simplexová metoda postupně prochází vrcholy tohoto polyedru a hledá optimální řešení.

Základním krokem je nalezení výchozího řešení, tj. libovolného přípustného řešení, které představuje vrchol polyedru. Dále zkoumáme, jak by se měnila hodnota

[1] Zápisem $\boldsymbol{x} \geq 0$ pro vektor $\boldsymbol{x} = (x_1, \ldots, x_d)^T$ rozumíme $x_i \geq 0$, $i = 1, \ldots, d$.

[2] Afinním poloprostorem nazýváme množinu vektorů $\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}\boldsymbol{x} \leq b\}$ pro nějaký vektor $\boldsymbol{a} \in \mathbb{R}^d$ a číslo $b \in \mathbb{R}$.

účelové funkce, pokud bychom se pohybovali po některé z hran polyedru vycházející z aktuálního vrcholu. Pokud přírůstek hodnoty účelové funkce pro žádnou hranu není kladný, pak se nacházíme v optimálním řešení. V opačném případě se vydáme po některé hraně polyedru, podél níž hodnota účelové funkce roste. Pokud je tato hrana polopřímkou, pak je úloha neomezená. Jestliže se jedná o úsečku, pak dojdeme do jiného vrcholu polyedru, který představuje jiné přípustné řešení s vyšší hodnotou účelové funkce. Předchozí postup opakujeme pro nově nalezené řešení.

Přestože při použití v praxi je simplexová metoda poměrně efektivní nástroj pro řešení úlohy lineárního programování, v nejhorším případě může délka výpočtu růst exponenciálně vzhledem k velikosti vstupu (viz např. [4, kapitola 5.2]). Časová složitost simplexové metody je tedy $\mathcal{O}(2^n)$.

Jak simplexová metoda, tak předchozí náhodnostní algoritmus jsou iteračními algoritmy. Zatímco simplexová metoda v jednotlivých iteracích prochází přípustná řešení odpovídající vrcholům polyedru, řešení nalezená během iterací náhodnostního algoritmu nemusí být přípustnými řešeními původní úlohy. Každý algoritmus tak k problému přistupuje zcela odlišným způsobem, což vede také k odlišným závěrům o časových složitostech obou algoritmů.

Kapitola 7

Úloha lineárního programování ve vyšších dimenzích

Doposud jsme se zabývali úlohou lineárního programování v rovině, popřípadě na přímce. Odvozený algoritmus lze však zobecnit a na stejném principu je možné řešit i úlohy ve vyšších dimenzích. Každá nerovnice v úloze dimenze d zadává afinní poloprostor v $\mathbb{R}^d$. Stejně jako v dvourozměrném případě úlohu řešíme iteračně postupným přidáváním nerovnic. Pro $i = 1, 2, \ldots, n$ označme h_i jednotlivé poloprostory, g_i jejich hranice.

Lze ukázat že i ve vyšších dimenzích platí analogie věty 3.1, viz [1, kapitola 4.6]. Je-li v_i řešení úlohy dimenze d po i iteracích, pak se přidáním poloprostoru h_{i+1} optimální řešení nezmění, pokud v_i v tomto poloprostoru leží. V opačném případě se nové řešení nachází na hranici g_{i+1} nově přidaného poloprostoru, anebo úloha nemá přípustné řešení. Tato hranice je nadrovinou dimenze $d-1$. Řešení v_{i+1} musí ležet ve všech průnicích hranice g_{i+1} s poloprostory $h_1, \ldots, h_i$. Na této množině tak hledáme bod, ve kterém účelová funkce nabývá maxima. Abychom tedy nalezli řešení úlohy po $i+1$ iteracích, musíme vyřešit úlohu dimenze $d-1$. Účelovou funkci této úlohy snadno získáme omezením původní účelové funkce na množinu g_{i+1}.

Uvažme úlohu lineárního programování v trojrozměrném euklidovském prostoru. Nejprve musíme zjistit, zda je úloha neomezená. Vycházíme přitom z analogie věty 4.1. Hledáme proto vhodný vektor $\boldsymbol{s}$ určující v množině všech přípustných řešení polopřímku, podél níž účelová funkce neomezeně roste. Požadujeme tedy, aby odchylka vektorů $\boldsymbol{s}$ a $\boldsymbol{c}$ byla v intervalu $\langle 0, \frac{\pi}{2})$ a současně odchylka vektorů $\boldsymbol{s}$ a $\boldsymbol{n_i}$ byla v intervalu $\langle 0, \frac{\pi}{2}\rangle$ pro $i = 1, 2, \ldots, n$, kde $\boldsymbol{n_i}$ je normálový vektor hraniční roviny poloprostoru h_i směřující do tohoto poloprostoru. Jelikož nám záleží pouze na směru vektoru $\boldsymbol{s}$, můžeme požadovat, aby jeho koncový bod ležel v dané rovině kolmé na vektor $\boldsymbol{c}$. Každému směru, ve kterém účelová funkce roste, odpovídá právě jeden bod této roviny. Vyjádřením výše uvedených požadavků na vektor $\boldsymbol{s}$ pak získáme úlohu lineárního programování v rovině. Tu umíme řešit v očekávaném lineárním čase.

Jestliže nalezneme vektor $\boldsymbol{s}$ požadovaných vlastností, původní úloha je neomezená. V opačném případě můžeme během výpočtu získat svědky skutečnosti, že dvoudimenzionální úloha k nalezení vektoru $\boldsymbol{s}$ nemá přípustné řešení, a tedy svědky

omezenosti původní úlohy. Až na singulární případy jsou tito svědci 3. Získáme je ve chvíli, kdy po přidání poloroviny h_i jednodimenzionální úloha na přímce l_i k nalezení řešení v_i nemá přípustné řešení. Konkrétně se jedná o polorovinu h_i společně se svědky neexistence přípustného řešení jednodimenzionální úlohy. Singulární případy bychom řešili analogicky jako v kapitole 4 při zjišťování existence lexikograficky nejmenšího řešení.

Máme-li k dispozici svědky omezenosti úlohy v $\mathbb{R}^3$, můžeme přejít k řešení omezené úlohy. Vyjdeme z počátečního řešení, které je průnikem hraničních rovin svědků omezenosti. Postupně v náhodném pořadí přidáváme poloprostory. Předpokládejme že po i iteracích máme řešení v_i. Pokud v_i leží v poloprostoru h_{i+1}, pak $v_{i+1} = v_i$. V opačném případě řešíme úlohu lineárního programování v hraniční rovině g_{i+1} poloprostoru h_{i+1}. Jednotlivé poloroviny této úlohy jsou určeny jako průniky roviny g_{i+1} s poloprostory $h_1, \ldots, h_i$. Počet polorovin v této úloze je i, a proto její očekávaná časová složitost je $\mathcal{O}(i)$. Očekávaná délka výpočtu i-té iterace je tedy buď konstantní, pokud $v_{i-1} \in h_i$, anebo $\mathcal{O}(i)$. Situace je analogická odvození očekávané časové složitosti v kapitole 5. Náhodnostní algoritmus k řešení úlohy lineárního programování v $\mathbb{R}^3$ má proto lineární očekávanou časovou složitost.

Podobný postup lze využít i pro úlohy dimenze d. Zde získáme d svědků omezenosti úlohy. V jednotlivých iteracích řešíme úlohy dimenze $d-1$. Indukcí tak získáme stejný závěr o lineární očekávané časové složitosti náhodnostního algoritmu k řešení úlohy lineárního programování v $\mathbb{R}^d$.

Očekávaná časová složitost uvedeného algoritmu je lineární pro libovolnou fixní dimenzi d. To však neznamená, že délka výpočtu je stejná pro všechny dimenze. Lze ukázat, že očekávaná časová složitost algoritmu roste exponenciálně s rostoucí dimenzí, viz [1, kapitola 4.6]. Algoritmus je proto vhodný pouze pro úlohy nižších dimenzí.

Kapitola 8

Implementace

Algoritmus k řešení úlohy lineárního programování v rovině odvozený v této práci byl implementován v prostředí *Delphi 6.0*. Vznikl tak program využívající teoretické poznatky získané v předchozích kapitolách.

Předností aplikace je její názornost, a to díky grafickému výstupu. Je zaměřena na demonstraci principu algoritmu, a proto je vhodná pro výukové účely. Program umožňuje přehledné znázornění jednotlivých kroků výpočtu algoritmu.

Aplikace je určena pro operační systém *Windows* a spouští se pomocí souboru *LPinPlane.exe*, který se nachází na přiloženém kompaktním disku. Po spuštění zadáme vektor $\boldsymbol{c}$ představující účelovou funkci a jednotlivé nerovnice úlohy. Pomocí tlačítek můžeme nerovnice uspořádat do námi požadovaného pořadí, anebo je uspořádat náhodně. Úlohu je možné uložit do souboru, ze kterého ji můžeme později otevřít, a nemusíme ji tak zadávat po jednotlivých nerovnicích. Úlohu si můžeme nechat vykreslit kliknutím na příslušné tlačítko. Poloroviny jsou zobrazovány stejným způsobem jako v této práci, tedy hraniční přímkou a vystínováním příslušné strany. Po kliknutí na některou z nerovnic je odpovídající hraniční přímka zvýrazněna. Režim zadávání nerovnic je znázorněn na obrázku 8.1.

V nabídce *Nástroje - Možnosti* můžeme volit mezi omezenou a obecnou úlohou. V případě omezené úlohy se jedná o implementaci algoritmu 2, kde za dodatečná lineární omezení jsou považovány první dvě poloroviny. Uživatel tak musí sám vhodně vybrat první dvě nerovnice, a to tak, aby společně s vektorem $\boldsymbol{c}$ tvořily omezenou úlohu s lexikograficky nejmenším řešením. Naopak při volbě obecné úlohy si program najde svědky omezenosti sám a změní pořadí polorovin tak, že svědky umístí na první dvě pozice. Při volbě obecné úlohy se program dokáže vypořádat i s neomezenou úlohou. Jedná se o implementaci algoritmu 9. Po spuštění programu je zvolena obecná úloha.

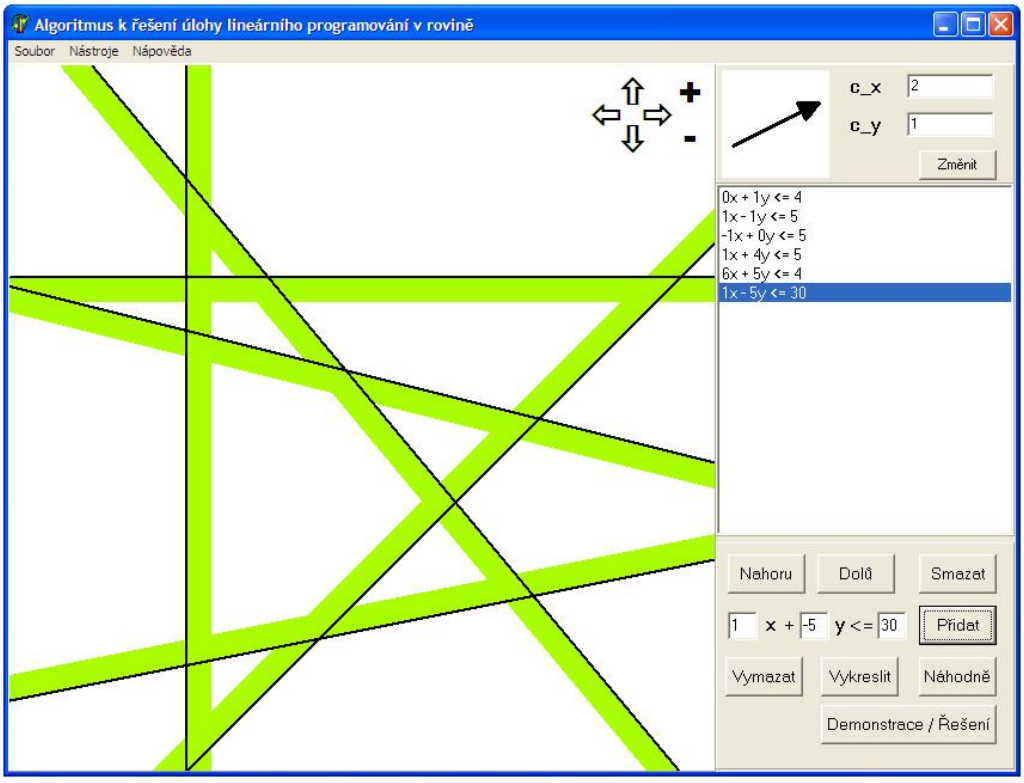
Kliknutím na tlačítko *Demonstrace / Řešení* přejdeme do demonstračního režimu. V případě, že v průběhu výpočtu dojde k volání algoritmu 2, můžeme zde procházet jednotlivé kroky algoritmu, a to buď ručně, anebo automaticky v nastaveném časovém intervalu. Průběžně jsou zobrazovány jednotlivé dílčí úlohy a jejich řešení. Hraniční přímka nově přidané poloroviny je zvýrazněna. Program v průběhu demonstrace je znázorněn na obrázku 8.2. V případě, že k volání algoritmu 2 nedo-

jde (např. u neomezené úlohy), je výsledek zobrazen ihned.

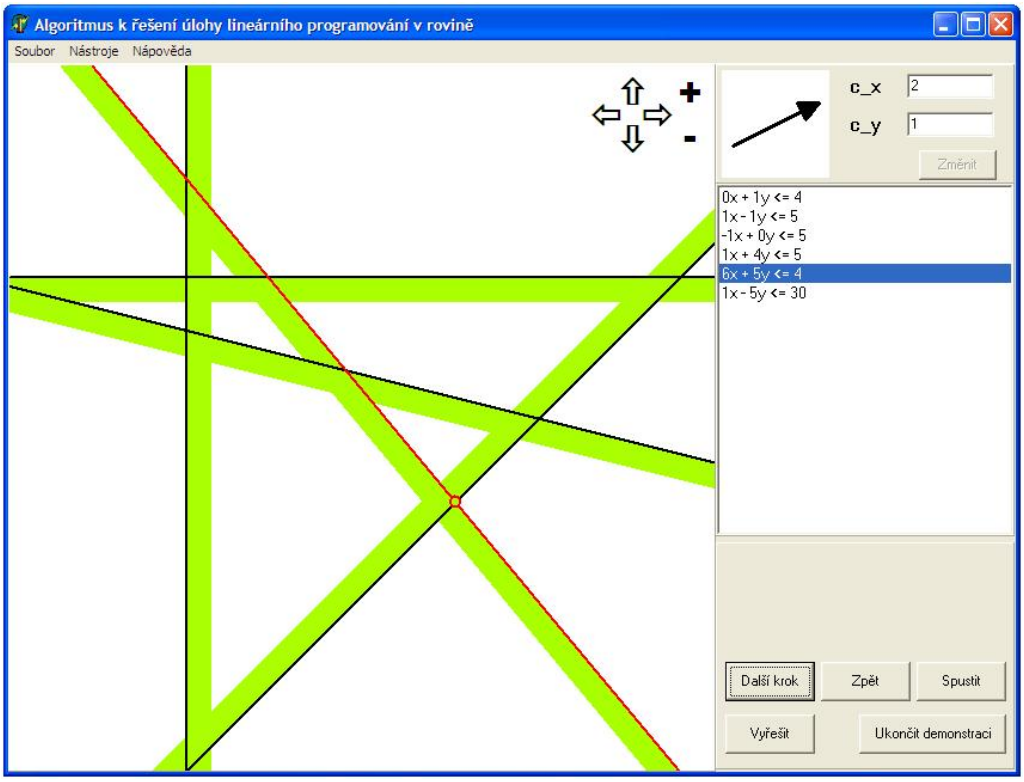
Aplikace také umožňuje nastavení některých parametrů ovlivňujících její funkci a vzhled. Kromě již zmíněné volby mezi omezenou a obecnou úlohou lze nastavit časový interval mezi jednotlivými kroky při automatické demonstraci či barvy jednotlivých prvků grafického výstupu.

Program umožňuje zkoumat, jak se mění průběh výpočtu algoritmu v závislosti na pořadí vstupních nerovnic. Je možné srovnávat různá uživatelem určená pořadí s náhodným pořadím, což může být přínosné pro objasnění časové složitosti algoritmu.

Vytvořená aplikace tedy není pouhou strohou implementací algoritmu, ale zahrnuje také některé užitečné funkce usnadňující pochopení jeho principu, a to především prostřednictvím přehledného grafického výstupu.



Obrázek 8.1: Program *LPinPlane.exe* v režimu zadávání nerovnic



Obrázek 8.2: Program *LPinPlane.exe* v průběhu demonstračního režimu

Seznam použité literatury

[1] BERG, Mark, et al. *Computational Geometry : Algorithms and Applications.* Third Edition. Berlin : Springer-Verlag, 2008. 386 s. ISBN 978-3-540-77973-5.

[2] CORMEN, Thomas H., et al. *Introduction to algorithms.* 2nd ed. Cambridge (Mass.) : MIT Press, 2001. 1180 s. ISBN 0-262-03293-7.

[3] JIRSÍK, Tomáš. *Simplexová metoda a její využití v ekonomii* [online]. Brno : Masarykova univerzita, 2009. 44 s. Bakalářská práce. Masarykova univerzita, Přírodovědecká fakulta. Dostupné z WWW: <http://is.muni.cz/th/211086/prif_b/>.

[4] KASANA, Harvir Singh; KUMAR, Krishna Dev. *Introductory Operations Research : Theory and Applications.* Berlin : Springer-Verlag, 2004. 581 s. ISBN 3-540-40138-5.