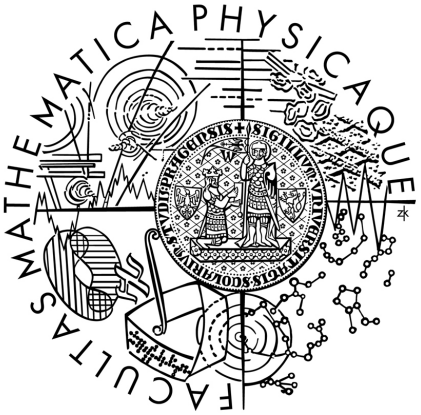
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Jiří Peinlich

Metody řešení úloh semi-infinitního programování

Katedra aplikované matematiky

Vedoucí diplomové práce: Doc. RNDr. Libuše Grygarová, DrSc.

Studijní program: Informatika, Diskrétní matematika a optimalizace

2008

Děkuji paní Doc. RNDr. Libuši Grygarové, DrSc. za její rady a odborné vedení při tvorbě této práce.
Děkuji Alence za podporu, kterou jsem pro psaní potřeboval.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Jiří Peinlich

Obsah

Název práce: Metody řešení úloh semi-infinitního programování
Autor: Jiří Peinlich
Katedra (ústav): Katedra aplikované matematiky
Vedoucí diplomové práce: Doc. RNDr. Libuše Grygarová, DrSc.
e-mail vedoucího: libuse@kam.mff.cuni.cz

Abstrakt: Cílem práce je dát přehled základních přístupů k řešení úloh lineárního semi-infinitního programování. Dále se práce podrobněji zabývá různými variantami metod sečných nadrovin pro úlohu lineárního semi-infinitního programování. Práce zahrnuje implementaci dvou variant této metody v jazyce Octave a na příkladech je ukázáno, jak tyto metody pracují.
Klíčová slova: optimalizační metody, metody lineárního semi-infinitního programování, metody sečných nadrovin

Title: Methods for solving semi-infinite programming problems
Author: Jiří Peinlich
Department: Department of Applied Mathematics
Supervisor: Doc. RNDr. Libuše Grygarová, DrSc.
Supervisor's e-mail address: libuse@kam.mff.cuni.cz

Abstract: The aim of this work is to give an overview of methods for solving linear semi-infinite programming problems. The work also discusses various types of cutting plane method for semi-infinite programming problems. The work involves implementation of two types of this method in programming language Octave and the behavior of these methods is shown on examples.
Keywords: optimization methods, linear semi-infinite programming methods, cutting-plane methods

Kapitola 1

Úvod

V této diplomové práci se zaměřuji na zpracování metod řešení úlohy lineárního semi-infinitního programování. Lineární semi-infinitní programování je zajímavým odvětvím matematického programování. Řešení problémů lineárního semi-infinitního programování často vyžaduje speciální teoretické a numerické techniky. Oblast lineárního semi-infinitního programování, stejně jako její aplikace, se v posledních několika letech velmi rychle rozvíjí.

Mějme $\mathbf{c} \in \mathbb{R}^n$, T pevnou nekonečně-prvkovou množinu indexů. Pro každé $t \in T$ mějme dán reálný vektor $\mathbf{a}(t) \in \mathbb{R}^n$ a reálné číslo $b(t) \in \mathbb{R}$. Pak úlohou lineárního semi-infinitního programování nazveme následující minimalizační úlohu:

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}(t)^T\mathbf{x} \geq b(t), \forall t \in T\right\}, \end{gathered} \tag{1.1}$$

Problém 1.1 vypadá, že má velmi jednoduchou strukturu, ale ve skutečnosti obsahuje mnoho obecných optimalizačních problémů. Mějme například $f : T \to \mathbb{R}$ libovolnou reálnou funkci definovanou na libovolné množině T. Potom problém nalezení globálního minima f na T lze jednoduše převést na úlohu 1.1 následujícím způsobem

$$\begin{gathered} \min_M -y, \\ M = \{y \in \mathbb{R} : -y \geq -f(t), \forall t \in T\}. \end{gathered}$$

Netriviální a mnohem praktičtější příklad dostaneme, pokud bude v předchozím problému $T = \mathbb{R}^n$ a dále $f : \mathbb{R}^n \to \mathbb{R}$ konvexní a diferencovatelná.

Potom totiž platí (viz [13]), že

$$f(x) = \max_{t \in \mathbb{R}^n} \{f(t) + \nabla f(t)^T (x - t)\}.$$

Minimalizace funkce f na $\mathbb{R}^n$ je pak ekvivalentní následující úloze 1.1

$$\min_M x_0,$$
$$M = \{(x_0, \mathbf{x}) \in \mathbb{R}^{n+1} : x_0 - f(t)^T \mathbf{x} \geq f(t) - \nabla f(t)^T t, \forall t \in \mathbb{R}^n\}.$$

Analogicky lze převést libovolnou úlohu konvexní optimalizace na úlohu semi-infinitního programování v $\mathbb{R}^n$.

Úloha lineárního semi-infinitního programování vyvstává například v problémech lineární Čebyševovy aproximace. V těchto problémech bývá funkce f, definovaná na kompaktní podmnožině $Y \subset \mathbb{R}^m$, aproximovaná lineární kombinací $p_n(\mathbf{x}, \mathbf{y}) = \sum_{n=1}^{n} x_j g_j(y)$ funkcí g_j. Pak musíme mimo jiné zjistit

$$\min_M e,$$
$$M = \{e \in \mathbb{R} : \pm(f(\mathbf{y}) - p_n(\mathbf{x}, \mathbf{y})) \leq e, \forall \mathbf{y} \in Y\}.$$

Aplikace semi-infinitního programování lze najít v mnoha oborech lidské činnosti. Semi-infinitní hry vyvstávají v těch situacích z teorie her, kdy jeden hráč má k dispozici nekonečně mnoho čistých strategií, zatímco druhý hráč má pouze konečný počet čistých strategií. V geometrii lze jako úlohu lineárního semi-infinitního programování formulovat například problém oddělení a silného oddělení dvojic podmnožin normovaného prostoru. V samoučících systémech vyvstává problém lineárního semi-infinitního programování v případě, že množina hypotéz systému je nekonečná. V telekomunikacích se úloha lineárního semi-infinitního programování používá pro výpočet, jak zvyšovat kapacitu optimální mobilní sítě. O dalších aplikacích úlohy semi-infinitního programování se lze dočíst v [5].

V první části práce seznámím čtenáře se základy semi-infinitního programování a se základními metodami, které tuto úlohu řeší. V druhé části práce se zaměřuji na metodu sečných nadrovin pro úlohu lineárního semi-infinitního programování. Tato metoda má oproti ostatním metodám řadu výhod. Velkou výhodou metody sečných nadrovin je lineární

rychlost konvergence (viz [4]). Autoři [3] a [17] uvádějí metodu sečných nadrovin jako jednu z klíčových metod pro řešení úlohy lineárního semi-infinitního programování. V poslední části práce popisuji implementaci dvou různých metod sečných nadrovin pro úlohu lineární semi-infinitní optimalizace. Součástí diplomové práce jsou i zdrojové texty implementovaných algoritmů v jazyce Octave na přiloženém optickém disku.

Kapitola 2

Lineární semi-infinitní programování

2.1 Úloha lineárního semi-infinitního programování

Úloha lineárního semi-infinitního programování vzniká jako zobecnění úlohy lineárního programování. Tak jako se ve většině disciplín matematického programování uvažuje množina přípustných řešení omezená pomocí konečného počtu podmínek, je i v úloze lineárního programování množina přípustných řešení omezená konečně mnoha lineárními podmínkami. Úloha lineárního semi-infinitního programování zobecňuje úlohu lineárního programování tím, že se v těchto optimalizačních úlohách uvažuje množina omezení množiny přípustných řešení nekonečná.

Úloha semi-infinitního programování se dá dále zobecnit tak, že se navíc uvažuje nekonečná množina proměnných. Úlohám, ve kterých je nekonečná jak množina proměnných, tak množina omezení množiny přípustných řešení, se říká úlohy infinitního programování.

Úlohy, ve kterých uvažujeme nekonečnou množinu proměnných, ale konečnou množinu podmínek, se označují jako úlohy zobecněného semi-infinitního programování. Struktura takovýchto úloh je obecně složitější než struktura úloh semi-infinitního programování.

Definice 1 *Mějme dán reálný vektor* $\mathbf{c} \in \mathbb{R}^n$. *Dále nechť* T *je pevná nekonečně-prvková množina indexů. Pro každé* $t \in T$ *mějme dán reálný vektor* $\mathbf{a}(t) \in \mathbb{R}^n$ *a reálné číslo* $b(t) \in \mathbb{R}$. *Pak úlohou lineárního semi-infinitního programování nazveme minimalizační úlohu*

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}(t)^T\mathbf{x} \geq b(t), \forall t \in T\right\}. \end{gathered} \tag{2.1}$$

V této úloze máme za úkol minimalizovat lineární funkci proměnných $\mathbf{x} \in \mathbb{R}^n$ vzhledem k nekonečné množině lineárních podmínek. Pokud by v úloze 2.1 byla množina T konečná množina indexů, změnila by se úloha na úlohu lineárního programování. Pokud je množina T spočetná, mluvíme o úloze spočetného lineárního semi-infitního programování. Pokud je v úloze 2.1 množina T Euklidův prostor konečné dimenze a pokud jsou funkce $\mathbf{a}(t), b(t)$ spojité na T, mluvíme o úloze spojitého lineárního semi-infinitního programování.
Uvedená definice úlohy semi-infinitního programování je vhodná k vytváření duální úlohy. Pro zjednodušení zápisu uveďme ještě definici ekvivalentní. V textu se budeme odkazovat na tu z definic, kterou budeme potřebovat.

Definice 2 *Mějme* $\mathbf{c}$, T, $\mathbf{a}(t), b(t)$ *definováno jako v definici 2.1. Definujme nyní reálnou funkci dvou proměnných* $g(\mathbf{x}, t)$ *jako* $g(\mathbf{x}, t) = b(t) - \mathbf{a}(t)\mathbf{x}$. *Úlohu lineárního semi-infinitního programování pak můžeme definovat takto*

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \\ M = \{\mathbf{x} \in \mathbb{R}^n : g(\mathbf{x}, t) \leq 0, \forall t \in T\}. \end{gathered} \tag{2.2}$$

2.2 Stručný úvod do teorie duality v semi-infinitním programování

Jednou ze základních technik práce s úlohami matematického programování je tvorba a následná analýza příslušných duálních úloh. Zabývejme se nejprve úlohou spočetného lineárního semi-infinitního programování. Uvažujme úlohu lineárního semi-infinitního programování ve

tvaru

$$\begin{gathered}\min_{M} \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}_i^T\mathbf{x} \geq b_i, i = 1, 2, \ldots\right\}.\end{gathered} \tag{2.3}$$

Předtím, než přistoupíme k prezentaci duální úlohy, je třeba definovat prostor proměnných duální úlohy. Prostor proměnných duální úlohy lze volit různě, a proto v závislosti na volbě prostoru se pak mění i duální úloha. Označme Z jako prostor všech posloupností v $\mathbb{R}$. Označme dále prostor všech posloupností v $\mathbb{R}$, ve kterých je pouze konečný počet nenulových prvků, jako W. Bilineární formu definujeme na $Z \times W$ standardním způsobem jako $\langle \mathbf{z}, \mathbf{w} \rangle = \sum z_i w_i, \mathbf{z} \in Z, \mathbf{w} \in W$. W bude prostor proměnných v duální úloze a můžeme tedy přistoupit k formulaci duální úlohy. K úloze 2.3 lze vytvořit následující duální úlohu (viz [7])

$$\begin{gathered}\max_{M} \sum_i w_i b_i \\ M = \left\{ w \in W : \sum_i w_i \mathbf{a}_i = \mathbf{c}, w_i \geq 0 \right\}.\end{gathered} \tag{2.4}$$

Po zformulování primární a duální úlohy můžeme již přistoupit k předložení podmínek, za jakých nabývají cílové funkce úloh 2.3 a 2.4 stejné optimální hodnoty. Nejprve definujme dvě množiny

$$M_A := \{\sum_i w_i \mathbf{a}_i : \mathbf{w} \in W, w_i \geq 0\},$$

$$M_B := \{(\sum_i w_i \mathbf{a}_i, \sum_i w_i b_i) : \mathbf{w} \in W, w_i \geq 0\}.$$

Všimněme si, že tyto množiny jsou kužely v $\mathbb{R}^n$, respektive v $\mathbb{R}^{n+1}$. Pokud označíme souhrnně podmínky $\mathbf{a_i}$ z úlohy 2.3 jako A, pak

$$M_A := \{A\mathbf{x} : \mathbf{x} \in \mathbb{R}^n, \mathbf{x} \geq \mathbf{0}\},$$

$$M_B := \{(A\mathbf{x}, \langle \mathbf{x}, \mathbf{b} \rangle) : \mathbf{x} \in \mathbb{R}^n, \mathbf{x} \geq \mathbf{0}\}.$$

Níže uvedené věty ukazují, za jakých podmínek má cílová funkce duální úlohy 2.4 stejnou optimální hodnotu jako cílová funkce primární úlohy 2.3.

Věta 1 *Pokud je optimální hodnota cílové funkce úlohy 2.4 konečná a pokud je množina M_B uzavřená, pak jsou optimální hodnoty cílových funkcí úloh 2.3, 2.4 stejné.*

Věta 2 *Pokud je optimální hodnota cílové funkce úlohy 2.4 konečná a pokud je* $\mathbf{c}$ *vnitřním bodem množiny M_A, pak jsou hodnoty cílových funkcí úloh 2.3, 2.4 stejné.*

Věta 3 *Pokud je optimální hodnota cílové funkce úlohy 2.3 konečná a pokud neexistuje žádné $\bar{\mathbf{x}} \neq 0$ s vlastnostmi $\mathbf{c}^T\bar{\mathbf{x}} = 0$, $\mathbf{a}_i^T\bar{\mathbf{x}} \geq 0$, $i = 1, 2, \ldots$, pak jsou hodnoty cílových funkcí úloh 2.3, 2.4 stejné.*

Pro zajímavost uveďme ještě standardní definici duální úlohy pro úlohu spojitého lineárního semi-infinitního programování (viz [7], [18]). Pro jednoduchost předpokládejme v definici úlohy lineárního semi-infinitního programování množinu T jako uzavřený interval $[0, 1]$, tj. uvažujme úlohu

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}(t)^T\mathbf{x} \geq b(t), \forall t \in [0,1]\right\}. \end{gathered} \tag{2.5}$$

O funkcích $\mathbf{a}(t)$, $b(t)$ dále předpokládejme, že jsou spojité. Označme prostor všech Borelovských měr na intervalu $[0, 1]$ jako $M_r[0, 1]$. Pak lze duální úlohu k úloze 2.5 formulovat následovně

$$\begin{gathered} \max_M \int_0^1 b(t)dw_{(t)}, \\ M = \left\{w \in M_r[0,1] : \int_0^1 \mathbf{a}(t)dw_{(t)} = \mathbf{c}, w \geq 0\right\}. \end{gathered} \tag{2.6}$$

Obdobně jako v případě spočetného semi-infinitního programování se dají definovat kužely v $\mathbb{R}^n$, respektive v $\mathbb{R}^{n+1}$

$$M_A := \left\{\int_0^1 \mathbf{a}(t)dw_{(t)} : w \geq 0, w \in M_r[0,1]\right\},$$

$$M_B := \left\{\int_0^1 \bar{\mathbf{a}}(t)dw_{(t)} : w \geq 0, w \in M_r[0,1]\right\},$$

kde $\bar{\mathbf{a}}$ píšeme místo $(\mathbf{a}(t), b(t))$ a platí věta

Věta 4 *Jakákoli z následujících podmínek stačí k tomu, aby měly cílové funkce úlohy 2.5 a 2.6 stejnou hodnotu.*

- *Optimální hodnota cílové funkce úlohy 2.6 má konečnou hodnotu a množina M_B je uzavřená.*
- *Optimální hodnota cílové funkce úlohy 2.6 má konečnou hodnotu a* $\mathbf{c}$ *je vnitřním bodem M_A.*
- *Optimální hodnota cílové funkce úlohy 2.5 má konečnou hodnotu a neexistuje* $\mathbf{x} \neq \emptyset$*, pro které by platilo* $\mathbf{c}^T\mathbf{x} = 0$ *a* $\mathbf{a}(t)\mathbf{x} \geq 0, t \in [0,1]$.

2.3 Základní rozdělení algoritmů pro úlohu lineárního semi-infinitního programování

K řešení úlohy lineárního semi-infinitního programování se přistupuje několika velice odlišnými způsoby. Bylo by nad rámec diplomové práce podrobně vysvětlovat všechny takovéto přístupy. V této diplomové práci se zaměřuji na uvedení základních typů řešení úlohy lineárního semi-infinitního programování. U každého typu uvádím základní myšlenky řešení spolu s odkazy na další literaturu, která ten který přístup popisuje podrobněji. Snažím se přístupy přiblížit v takové míře, aby si následně čtenář mohl vybrat metodu, která mu bude připadat přijatelná jak z hlediska rychlosti konvergence, tak z hlediska náročnosti implementace.

Existují následující základní typy algoritmů

- algoritmy odvozené od simplexové metody;
- algoritmy vnitřních bodů;
- redukční algoritmy;
- diskretizační algoritmy;
- algoritmy založené na metodě sečných nadrovin.

2.3.1 Algoritmy odvozené od simplexové metody

Algoritmy založené na simplexové metodě silně souvisí s teorií duality v lineárním semi-infinitním programování a dají se vytvořit jak pro primární, tak pro duální úlohu. Protože je však duální úloha poměrně

odlišná od primární úlohy, dá se předpokládat i odlišnost takovýchto algoritmů. V této práci se budu věnovat algoritmu pro duální úlohu spočetného lineárního programování. Algoritmus pro primární úlohu je mírně složitější a lze jej dohledat spolu s algoritmy pro spojité lineární semi-infinitní programování například v [7].

Uvažujme úlohu spočetného lineárního semi-infinitního programování v následující podobě

$$\begin{gathered}\min_M \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}_i^T\mathbf{x} \geq b_i, i = 1, 2, \ldots\right\}.\end{gathered} \tag{2.7}$$

Dále uvažujme duální úlohu k úloze 2.7 v následující podobě

$$\begin{gathered}\max_M \sum_i w_i b_i, \\ M = \left\{\mathbf{w} \in W : \sum_i w_i \mathbf{a}_i = \mathbf{c}, w_i \geq 0\right\},\end{gathered} \tag{2.8}$$

kde W buď prostor všech posloupností v $\mathbb{R}^n$, ve kterých je pouze konečný počet nenulových členů.

Začneme identifikováním bázických řešení. Nechť je $\mathbf{w} = \{w_i\}$ přípustným řešením úlohy 2.8. Budeme označovat

$$s(\mathbf{w}) := \{i : w_i \neq 0\}.$$

O $\mathbf{w}$ budeme říkat, že je bázickým, pokud bude množina $\{\mathbf{a}_i : i \in s(\mathbf{w})\}$ lineárně nezávislá. Odtud také plyne, že maximální počet prvků množiny $s(\mathbf{w})$ bude n. O bázickém řešení $\mathbf{w}$ řekneme, že je nedegenerované, pokud $\{\mathbf{a}_i : i \in s(\mathbf{w})\}$ generuje celé $\mathbb{R}^n$. Pro nedegenerované bázické řešení tedy bude mít množina $s(\mathbf{w})$ právě n prvků. Pro dané bázické řešení budeme matici, jejíž sloupce budou $a_i, i \in s(\mathbf{w})$, označovat $\hat{A}$. Dále $\hat{\mathbf{w}}, \hat{\mathbf{b}}$ nechť jsou vektory o souřadnicích $w_i, b_i, i \in s(\mathbf{w})$. Pokud je $\mathbf{w}$ nedegenerované, pak $\hat{A}$ je čtvercová, invertibilní a platí $\hat{A}\hat{\mathbf{w}} = \mathbf{c}$. Tedy $\hat{\mathbf{w}} = \hat{A}^{-1}\mathbf{c}$.

Přechodový funkcionál pro dané bázické řešení $\mathbf{w}$ definujme následovně

$$w_i^* := b_i - \mathbf{a}_i^T(\hat{A}^{-1})^T\hat{b}.$$

S přihlédnutím k tomu, že je problém 2.8 definován jako maximalizační a k tomu, že w_i^* je nulové pro $i \in s(\mathbf{w})$, můžeme říci, že přípustné nedegenerované bázické řešení $\mathbf{w}$ je optimální právě tehdy, když je $\mathbf{w}^*$ nekladné. Předpokládejme, že tedy není $\mathbf{w}^*$ nekladné, tj. když existuje nějaké k tak, že

$$b_k > \mathbf{a}_k(\hat{A}^{-1})^T\hat{\mathbf{b}}$$

.
Nyní chceme získat nové přípustné řešení, označme ho w'_k. Chceme zajistit, aby w'_k mělo stejnou množinu $s(w')$, jako předchozí prvek w, ale aby v množině $s(w')$ byl navíc i prvek k. Položme $w'_k = \delta > 0$, pak

$$\sum_{s(\mathbf{w})} w'_i\mathbf{a}_i + \delta\mathbf{a}_k = c.$$

Tedy

$$\mathbf{w}' = \hat{\mathbf{w}}' + \delta e_k,$$

kde

$$\hat{\mathbf{w}}' = \hat{A}^{-1}(c - \delta\mathbf{a}_k),$$

odkud dostáváme

$$\sum b_i w'_i - \sum b_i w_i = \delta b_k - \delta\hat{\mathbf{b}}^T\hat{A}^{-1}\mathbf{a}_k > 0.$$

Jinými slovy je $\mathbf{w}'$ zlepšení $\mathbf{w}$. Navíc je pro dostatečně malé δ $\mathbf{w}'$ přípustným řešením a otázkou tedy zůstává, jak velké má δ být. Ve skutečnosti se δ musí položit tak velké, aby nějaké w'_i nabylo nulové hodnoty a $\mathbf{w}'$ se tím stalo bázickým řešením. Je tedy nutné položit

$$\delta := \min_{i \in s(\mathbf{w})} \left\{ \frac{w_i}{[(\hat{A}^{-1})^T\mathbf{a}_k]_i} \right\}.$$

Pro více informací je možno čtenáře odkázat například na [7].

2.3.2 Metody vnitřních bodů

Metody vnitřních bodů pro řešení úlohy lineárního semi-infinitního programování obvykle vycházejí z metod vnitřních bodů pro řešení úlohy lineárního programování. Tyto algoritmy řeší úlohu lineárního programování tak, že tvoří posloupnost přípustných řešení konvergující k optimálnímu řešení. Implementace metod vnitřních bodů je v porovnání

s metodami vnějších bodů obvykle mnohem složitější, ale velkou výhodou metod vnitřních bodů oproti ostatním metodám je, že v každém kroku iterativního postupu je získané meziřešení přípustné zadané úloze. V této sekci se podíváme na zobecnění jedné z metod vnitřních bodů pro úlohu semi-infinitního programování.

Pro jednodušší výklad je výhodné pracovat s algoritmem nejdříve v obecnějším tvaru. Následně si pak vysvětlíme, jak se algoritmus aplikuje na semi-infinitní programování.

Nechť T je libovolná neprázdná množina indexů a nechť W je vektorový prostor všech funkcí z T do $\mathbb{R}^q$. Nechť U, Z jsou vektorové prostory konečné dimenze. Definujme vektorový prostor V jako kartézský součin U a W, to jest

$$V = U \otimes W.$$

O V budeme dále předpokládat, že se jedná o unitární prostor. Mějme na V zavedený skalární součin, který zde budeme značit jako $\langle \cdot, \cdot \rangle$.
Mají-li funkce $\mathbf{w}(t), \mathbf{w}'(t) \in W$ vlastnost

$$w_i(t) \geq 0, w_i'(t) \geq 0, \forall t \in T, i = 1, \ldots n,$$

pak má stejnou vlastnost i jejich konvexní kombinace. Definujme konvexní kužel

$$W_+ := \{\mathbf{w} \in W | w_i(t) \geq 0, t \in T, i = 1, \ldots, q\}$$

s vnitřkem

$$W_+^0 := \{\mathbf{w} \in W | w_i(t) > 0, t \in T, i = 1, \ldots, q\}.$$

Označme

$$V_+ := U \otimes W_+$$

a

$$V_+^0 := U \otimes W_+^0.$$

Definujme částečné uspořádání $\geq$ na V následovně

$$\mathbf{a} \geq \mathbf{b} \text{ pro } \mathbf{a}, \mathbf{b} \in V \text{ právě tehdy když } \mathbf{a} - \mathbf{b} \in V_+.$$

Pokud dále označíme nulový prvek V jako $\emptyset$, pak platí

$$\mathbf{a} \in V, \mathbf{a} \geq \emptyset \text{ právě tehdy když } \mathbf{a} \in V_+.$$

Obdobně definujme $\mathbf{a} > \mathbf{b}$ pro $\mathbf{a}, \mathbf{b} \in V$, pokud je $\mathbf{a} - \mathbf{b} \in V_+^0$. Zřejmě, pokud je $\mathbf{a} > \mathbf{b}$, potom je také $\mathbf{a} \geq \mathbf{b}$. Předpokládejme dále, že je W_+^0 neprázdný. Pro pevně zvolený prvek $\mathbf{e} \in W_+^0$, budeme každému prvku $(\mathbf{u}, \mathbf{e}) \in V_+^0$ říkat preferovaný prvek V.
Zaměřme nyní svou pozornost na úlohy typu

$$\begin{gathered} \min_M \langle (\mathbf{c}, \mathbf{d}), (\mathbf{u}, \mathbf{w}) \rangle, \\ M = \{(\mathbf{u}, \mathbf{w}) \in V : L((\mathbf{u}, \mathbf{w})) = \mathbf{b}, (\mathbf{u}, \mathbf{w}) \geq \emptyset\}, \end{gathered} \tag{2.9}$$

kde $\mathbf{c} \in U$, $\mathbf{d} \in W$ a $L : V \to Z$ je spojitý lineární operátor.
Pro takovou třídu úloh se podívejme na algoritmus uvedený v práci [11], který je zobecněním algoritmu [12].

V každém kroku algoritmu definujeme nějaký endomorfismus na V_+^0, který převádí aktuální bod $(\mathbf{x}, \mathbf{z}) \in V_+^0$ na preferovaný bod $(\mathbf{x}, \mathbf{e}) \in V_+^0$, který je „daleko“ od hranice V_+^0. Následně se provede krok v transformovaném prostoru, který nejvíce sníží hodnotu transformové cílové funkce. Poté provedeme inverzní transformaci, abychom se dostali zpět do původního prostoru a získali tak nový bod, který se stane v dalším kroku aktuálním bodem.
Nyní popíšeme, jak definujeme endomorfismus. Nejprve definujeme $\mathbf{z}^{-1}$ jako

$$\mathbf{z}^{-1} := (z_1^{-1}, \ldots, z_q^{-1}),$$

kde

$$z_i^{-1}(t) = \frac{e_i(t)}{z_i(t)}, \forall t \in T, i = 1, \ldots, q,$$

a definujme také $\mathbf{z}^*$ jako

$$\mathbf{z}^* := (z_1^*, \ldots, z_q^*),$$

kde

$$z_i^*(t) := \frac{z_i(t)}{e_i(t)}, \forall t \in T, i = 1, \ldots, q.$$

Všimněme si, že $\mathbf{z}^{-1}$ a $\mathbf{z}^*$ jsou prvky W_+^0. Pro každé dva prvky $\mathbf{w}, \mathbf{v} \in W$ definujeme

$$\odot : W \times W \to W$$

jako

$$(\mathbf{w} \odot \mathbf{v})_i(t) = w_i(t) v_i(t), \forall t \in T, i = 1, \ldots, q.$$

Zřejmě je $\odot$ komutativní a asociativní, navíc pokud $\mathbf{w}, \mathbf{v} \in W_+^0$ pak i $\mathbf{w} \odot \mathbf{v} \in W_+^0$.
Vzhledem k aktuálnímu bodu $(\mathbf{x}, \mathbf{z}) > \emptyset$ definujme funkce

$$F_z((\mathbf{u}, \mathbf{w})) := (\mathbf{u}, \mathbf{z}^{-1} \odot \mathbf{w}),$$

$$F_z^*((\mathbf{u}, \mathbf{w})) := (\mathbf{u}, \mathbf{z}^* \odot \mathbf{w}).$$

Je vidět, že tyto funkce jsou k sobě navzájem inverzní. Za předpokladu, že pro každé $(\mathbf{u}, \mathbf{w}) \in V$ platí

$$\langle (\mathbf{c}, \mathbf{d}), (\mathbf{u}, \mathbf{w}) \rangle = \langle F_z^*((\mathbf{c}, \mathbf{d})), F_z((\mathbf{u}, \mathbf{w})) \rangle,$$

má problém

$$\begin{gathered} \min_M \langle F_z^*((\mathbf{c}, \mathbf{d})), F_z((\mathbf{u}, \mathbf{w})) \rangle, \\ M = \left\{ (\mathbf{u}, \mathbf{w}) \in V : \begin{array}{rcl} L \circ F_z^*(Fz((\mathbf{u}, \mathbf{w}))) & = & \mathbf{b}, \\ F_Z^*(Fz((\mathbf{u}, \mathbf{w}))) & \geq & \emptyset \end{array} \right\}, \end{gathered} \tag{2.10}$$

(kde $L \circ F$ označuje složení L a F) stejnou minimální hodnotu cílové funkce jako problém 2.9. Pokud tedy přepíšeme $\mathbf{z}^{-1} \odot \mathbf{w}$ jako $\mathbf{y}$, pak $(\mathbf{u}, \mathbf{y}) = F_z(\mathbf{u}, \mathbf{w})$. Díky tomu, že V a V_+ jsou invariantní vzhledem k F_z a její inverzi, můžeme úlohu 2.10 převést na tvar

$$\begin{gathered} \min_M \langle F_z^*((\mathbf{c}, \mathbf{d})), (\mathbf{u}, \mathbf{y}), \rangle, \\ M = \{ (\mathbf{u}, \mathbf{y}) \in V : L \circ F_z^*((\mathbf{u}, \mathbf{y})) = \mathbf{b}, (\mathbf{u}, \mathbf{y}) \geq \emptyset \}. \end{gathered} \tag{2.11}$$

Nyní si popíšeme metodu vnitřního bodu pro úlohu 2.9. Mějme daný bod $(\mathbf{x}, \mathbf{z}) \in V$ množiny přípustných řešení úlohy 2.9, který leží v V_+^0. Každá iterace algoritmu přetransformuje V endomorfismem F_z a zobrazí tak bod $(\mathbf{x}, \mathbf{z}) \in V$ na preferovaný bod $(\mathbf{x}, \mathbf{e}) \in V$. V dalším kroku iterace z bodu $(\mathbf{x}, \mathbf{e})$ přejdeme na nový bod v $F_z(V)$ s menší hodnotou cílové funkce 2.11. Následně se aplikuje inverzní zobrazení F_z^*, které zobrazí nově nalezený bod zpět do V.

Směr přechodového kroku budeme volit pomocí ortogonální projekce $-F_z^*((\mathbf{c}, \mathbf{d}))$ na jádro zobrazení $L \circ F_z^*$. Krok v tomto směru zaručuje pokles cílové funkce jak v transformovaném, tak v původním problému (viz [11]). Velikost snížení cílové funkce je pak úměrná délce přechodového kroku. Ortogonální projekce je nutná k zajištění toho, aby nový bod byl pořád

přípustným řešením původní úlohy.

Pro zajištění konvergence je tedy nutné, aby byly splněny tyto předpoklady:

- $V = U \otimes W$ je unitární prostor se skalárním součinem, kde U je vektorový prostor a W je prostor funkcí $T \to \mathbb{R}^q$. $L : V \to Z$ je lineární operátor zobrazující prostor V do prostoru Z.
- Existuje $\mathbf{a} \in V$, pro které platí $\mathbf{a} > \emptyset$.
- Pro každé $(\mathbf{u}, \mathbf{w}) \in V$ platí

$$\langle (\mathbf{c}, \mathbf{d}), (\mathbf{u}, \mathbf{w}) \rangle = \langle F_z^*((\mathbf{c}, \mathbf{d})), F_z((\mathbf{u}, \mathbf{w})) \rangle.$$

- Pro libovolně zvolený $\mathbf{z} \in W$ je jádro zobrazení $L \circ F_z^*$ Hilbertův prostor.

Samotný algoritmus vypadá následovně

Algoritmus 1: 1. Položíme $k = 0$.

2. Vezmeme přípustný bod $(\mathbf{x}, \mathbf{z})^{(k)} > \emptyset$ a převedeme jej na preferovaný bod $(\mathbf{x}, \mathbf{e})$ za použití funkce $F_{\mathbf{z}^{(k)}}$, tj.

$$(\mathbf{x}, \mathbf{e}) = F_{\mathbf{z}^{(k)}}((\mathbf{x}, \mathbf{z})^{(k)}).$$

3. Promítneme směr největšího klesání v transformované lineární funkci $-F^*_{\mathbf{z}^{(k)}}((\mathbf{c}, \mathbf{d}))$ ortogonálně do jádra zobrazení $L \circ F^*_{\mathbf{z}^{(k)}}$. Takto získáme (c_p, z_p).

4. Nalezneme krok délky $\alpha > 0$ tak, že $\mathbf{e} + \alpha z_p > \mathbf{0}$ a označme

$$\mathbf{z}' := \mathbf{e} + \alpha z_p.$$

5. Provedeme inverzní transformaci, abychom získali $(\mathbf{x}, \mathbf{z})^{(k+1)}$, tj.

$$(\mathbf{x}, \mathbf{z})^{(k+1)} = F^*_{\mathbf{z}^{(k)}}((\mathbf{x} + \alpha c_p, \mathbf{z}')).$$

6. Zkontrolujeme podmínku k ukončení a zastavíme, pokud je splněna. V opačném případě položíme $k := k + 1$ a vrátíme se na krok 2.

V další části této kapitoly se podíváme na to, jak se dá předešlý algoritmus aplikovat na úlohu semi-infinitního porogramování. Uvažujme úlohu semi-infinitního programování 2.1 ve tvaru

$$\min_M \mathbf{c}^T\mathbf{x},$$
$$M = \left\{ \mathbf{x} \in \mathbb{R}^n : \begin{array}{cc} \sum_{j=1}^n a_{1j}(t)x_j \geq b_1(t) & \text{pro } t \in [l_1, v_1] \\ \vdots & \vdots \\ \sum_{j=1}^n a_{qj}(t)x_j \geq b_q(t) & \text{pro } t \in [l_q, v_q] \end{array} \right\},$$

kde

$$a_{ij}(t), b_i(t) \in C^\infty[l_i, v_i], i = 1, \ldots, q, j = 1, \ldots n, \mathbf{c}, \mathbf{x} \in \mathbb{R}^n.$$

Položme

$$\mathbf{A}(s) = \begin{pmatrix} a_{11}(s) & \ldots & a_{1n}(s) \\ \vdots & \ddots & \vdots \\ a_{q1}(s) & \ldots & a_{qn}(s) \end{pmatrix}, \mathbf{b}(s) = \begin{pmatrix} b_1(s) \\ \vdots \\ b_q(s) \end{pmatrix}.$$

Tuto úlohu můžeme zkráceně zapsat v maticovém tvaru

$$\min_M \mathbf{c}^T\mathbf{x},$$
$$M = \left\{ \begin{array}{c} \mathbf{x} \in \mathbb{R}^n : \\ \mathbf{A}(t)^T\mathbf{x} \geq \mathbf{b}(t), \forall t \in T \end{array} \right\}. \tag{2.12}$$

K tomu, abychom tuto úlohu přeformulovali do podoby problému 2.9, zaveďme doplňkové proměnné $z_i \in C^\infty[l_i, v_i]$ do každé z i podmínek. Problém 2.12 tak převedeme do tvaru

$$\min_M \mathbf{c}^T\mathbf{x}$$
$$M = \left\{ \begin{array}{c} \mathbf{x} \in \mathbb{R}^n : \\ \mathbf{A}(t)\mathbf{x} - \mathbf{z}(t) = \mathbf{b}(t), \forall t \in T, \mathbf{z} \geq \emptyset \\ z \in \prod_{i=1}^q C^\infty[l_i, v_i]. \end{array} \right\}.$$

To už je úloha v podobě 2.9, pokud zvolíme

$$U = \mathbb{R}^n, W = Z = \prod_{i=1}^q C^\infty[l_i, v_i]$$

a

$$L(\mathbf{x}, \mathbf{z}) = A(.)x - z(.).$$

Součin $\langle ., . \rangle$ definujeme na V jako

$$\langle (\mathbf{c}, \mathbf{d}), (\mathbf{x}, \mathbf{z}) \rangle = \mathbf{c}^T \mathbf{x} + \sum_{i=1}^{q} \int_{l_i}^{v_i} d_i(t) z_i(t) dt.$$

Nechť e je dáno jako $e_i \in C^\infty[l_i, v_i]$ takové, že

$$e_i(t) = 1 \forall t \in [l_i, v_i], i = 1, \ldots, q.$$

Poznamenejme, že

$$\langle (\mathbf{c}, 0), (\mathbf{x}, \mathbf{z}) \rangle = \mathbf{c}^T \mathbf{x}.$$

Dále je také vidět, že

$$\langle (\mathbf{c}, \mathbf{d}), (\mathbf{u}, \mathbf{w}) \rangle = \langle F^*_{\mathbf{z}}((\mathbf{c}, \mathbf{d})), F_{\mathbf{z}}((\mathbf{u}, \mathbf{w})) \rangle,$$

protože podle definice součinu dostáváme

$$\begin{aligned} \langle (\mathbf{c}, \mathbf{d}), (\mathbf{u}, \mathbf{w}) \rangle &= \mathbf{c}^T \mathbf{u} + \sum_{i=1}^{q} \int_{l_i}^{v_i} d_i(t) w_i(t) = \\ &= \mathbf{c}^T \mathbf{u} + \sum_{i=1}^{q} \int_{l_i}^{v_i} d_i(t) z_i(t) (\frac{1}{z_i(t)}) w_i(t) = \\ &= \langle F^*_{\mathbf{z}}((\mathbf{c}, \mathbf{d})), F_{\mathbf{z}}((\mathbf{u}, \mathbf{w})) \rangle. \end{aligned}$$

Podívejme se nyní na implementaci kroků uvedeného algoritmu. V krocích 2 a 5 se dělí a násobí kladná funkce kladnou funkcí. V kroku 3 se promítá vektor na jádro zobrazení. Přesnou implementaci tohoto promítání lze najít v [11]. Zde uveďme pouze, že autoři doporučují před promítáním najít bázi jádra zobrazení a následně ortogonalizovat tuto bázi Gram-Schmidtovým ortogonalizačním procesem.
Hlavní výpočetní složitost algoritmu je v počítání součinu funkcí v ortogonalizačním procesu. Každý součin vyžaduje q integrací a počet součinů, které je potřeba během jedné iterace vypočíst, je řádu $O(n^2)$.

2.3.3 Lokální redukční metody

Redukční metody vycházejí z teorie publikované v [10]. Od ostatních metod se redukční metody liší ve dvou hlavních vlastnostech. Na jednu stranu dokáží redukční metody najít pouze lokální extrémy uvažované

úlohy, na druhou stranu však vykazují velmi vysokou rychlost konvergence. V praxi se pak redukční metody nasazují pro závěrečné zpřesnění výsledku po použití nějaké globálně konvergenční metody s pomalejší rychlostí konvergence.

Hlavním principem lokálních redukčních metod je v tom, že nahradíme problém 2.2 problémem s konečným počtem omezení, který lokálně redukuje zadanou úlohu.
Pro potřeby této kapitoly uvažujme úlohu 2.2, to jest úlohu

$$\begin{gathered} \min_{M} \mathbf{c}^T \mathbf{x}, , \\ M = \{\mathbf{x} \in \mathbb{R}^n : g(\mathbf{x}, t) \leq 0, \forall t \in T\}. \end{gathered} \tag{2.13}$$

O množině T navíc předpokládejme, že má tvar

$$\text{T= } \{\text{t} \in \mathbb{R}^m : h_j(t) \geq 0 \text{ pro } j \in J\},$$

kde J je konečná množina. Funkce $g : \mathbb{R}^n \times T \to \mathbb{R}$ je dvakrát spojitě diferencovatelná vzhledem k x a spojitě diferencovatelná vzhledem k t. Funkce h_j jsou dvakrát spojitě diferencovatelné vzhledem k t.

Pro dané $\bar{\mathbf{x}} \in \mathbb{R}^n$ uvažujme úlohu

$$\max_{t \in T} g(\bar{\mathbf{x}}, t). \tag{2.14}$$

Nechť $T^{\bar{\mathbf{x}}} = \{t^1, \ldots, t^k\}$ je množina všech lokálních řešení, která splňují podmínku

$$|g(\bar{\mathbf{x}}, t^l) - g^*| \leq \delta^{ML}, \forall l \in L(\bar{\mathbf{x}}),$$

kde $L(\bar{\mathbf{x}})$ reprezentuje indexovou množinu $T^{\bar{\mathbf{x}}}$, δ^{ML} je kladná konstanta a g^* je globální řešení 2.14. Definujme množinu indexů aktivních podmínek z popisu množiny T v bodě $\bar{t}$ jako

$$J_0(\bar{t}) = \{j \in J : h_j(\bar{t}) = 0\}$$

a Lagrangeovu funkci přiřazenou problémemu 2.14 jako

$$\bar{L}(\bar{\mathbf{x}}, t, u) = g(\bar{\mathbf{x}}, t) + \sum_{j=1}^{|J|} u_j h_j(t), u_j \geq 0, j \in J, t \in T,$$

kde u_j reprezentují Lagrangeovy multiplikátory. Pro nějaké $\bar{t} \in T$ nechť jsou vektory

$$\{\nabla h_j(\bar{t})\}_{j \in J_0(\bar{t})}$$

lineárně nezávislé. Položme

$$F(\bar{t}) = \{\xi \in \mathbb{R}^m : \xi^T \nabla h_j(\bar{t}) = 0, j \in J_0(\bar{t})\}.$$

Definice 3 *1. Bod $\bar{t} \in T$ se nazývá kritickým bodem úlohy 2.14, pokud existují taková $\bar{u}_j, j \in J_0(\bar{t})$, že*

$$\nabla_t \bar{L}(\bar{\mathbf{x}}, \bar{t}, \bar{u}) = \nabla_t g(\bar{\mathbf{x}}, \bar{t}) + \sum_{j \in J_0(\bar{t})} \bar{u}_j \nabla h_j(\bar{t}) = 0.$$

2. Bod $\bar{t} \in T$ se nazývá nedegenerovaný kritický bod, pokud jsou splněny podmínky

- *$\bar{t} \in T$ je kritický bod,*
- *$\bar{u}_j \neq 0$ pro $j \in J_0(\bar{t})$,*
- *$\xi^T \nabla^2_{tt} \bar{L}(\bar{x}, \bar{t}, \bar{u}) \xi \neq 0$ pro každé $\xi \in F(\bar{t}) \backslash \{0\}$.*

Aby se na úlohu semi-infintního programování dala použít redukční metoda, musí být množina všech kritických bodů $T^{\bar{x}}$ konečná.

Definice 4 *Úlohu 2.14 nazveme regulární, pokud každý kritický bod úlohy 2.14 je nedegenerovaný kritický bod.*

Pokud je $\bar{\mathbf{x}} \in M$ a pokud je 2.14 regulární, pak každé lokální maximum v úloze 2.14 je nedegenerované, a tedy se jedná o izolované lokální maximum. Protože je T kompaktní množina, existuje jen konečný počet lokálních maxim v problému 2.14. Díky tomu, že každé $t^l \in T^{\bar{x}}$ je izolovaným bodem, dá se aplikovat věta o implicitních funkcích. Existují tedy okolí $U(\bar{\mathbf{x}})$ bodu x a $V(t^l)$ bodu t^l a implicitní funkce $t^1(\mathbf{x}), \ldots, t^k$ definované následovně

- $t^l(\mathbf{x}) : U(\bar{\mathbf{x}}) \to V(t^l) \cap T$ pro $l \in L(\bar{\mathbf{x}})$,
- $t^l(\bar{\mathbf{x}}) = t^l,$ pro $l \in L(\bar{\mathbf{x}})$,
- $\forall \mathbf{x} \in U(\bar{\mathbf{x}}), t^l(\mathbf{x})$ je nedegenerované a izolované lokální maximum problému 2.14.

Lze dokázat ekvivalence (viz [10])

$$\{\mathbf{x} \in U(\bar{\mathbf{x}}) : g(\mathbf{x}, t) \leq 0, \forall t \in T\} \Leftrightarrow$$
$$\{\mathbf{x} \in U(\bar{\mathbf{x}}) : g^l(\mathbf{x}) \equiv g(\mathbf{x}, t^l(\mathbf{x})) \leq 0, l \in L(\bar{\mathbf{x}})\}.$$

Tedy, pokud je problém 2.14 regulární, je možné nahradit nekonečnou množinu podmínek problému 2.13 konečnou množinou podmínek, která je dostatečná k definování přípustné lokální oblasti. Tento lokálně redukovaný problém se definuje následovně

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \text{ kde} \\ M = \{\mathbf{x} \in U(\bar{\mathbf{x}}) : g^l(\mathbf{x}) \leq 0, l \in L(\bar{\mathbf{x}})\}. \end{gathered} \tag{2.15}$$

Pro každé $x \in U(\bar{\mathbf{x}})$ je $g^l(\mathbf{x})$ dvakrát spojitě diferencovatelná funkce vzhledem k x a x je přípustným bodem tehdy a jen tehdy, když $g^l(\mathbf{x}) \leq 0$. Lze dokázat, že $x^* \in U(\bar{x})$ je ostré lokální minimum problému 2.13 právě tehdy, když je x^* ostré izolované lokální minimum problému 2.15. Pro více informací lze čtenáře odkázat na [10].

2.3.4 Algoritmy diskretizační

Jedny ze základních metod řešení úlohy semi-infinitního programování jsou diskretizační metody. Uvažujme úlohu lineárního semi-infinitního programování ve tvaru

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \\ M = \{\mathbf{x} \in \mathbb{R}^n : g(\mathbf{x}, t) \leq 0, \forall t \in T\}. \end{gathered} \tag{2.16}$$

Dále uvažujme D libovolnou podmožinu T. Označme hustotu D v T jako

$$d(D, T) := \sup_{t \in T} \inf_{d \in D} |t - d|.$$

Pro každé D definujme problém

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \\ M = \{\mathbf{x} \in \mathbb{R}^n : g(\mathbf{x}, t) \leq 0, \forall t \in D\}. \end{gathered} \tag{2.17}$$

Množině D budeme říkat dělení množiny T. V případě, že je množina T nekonečná a dělení D konečné, mluvíme o diskretizaci problému 2.16.

Jedna z možností, jak najít řešení problému 2.16, je postupně řešit úlohy 2.17 pro $D_i, i = 0, 1, \ldots$, kde $\{D_i\}$ je posloupnost konečných podmožin T s vlastností

$$\lim_{i \to \infty} d(D_i, T) = 0. \tag{2.18}$$

Takovémuto postupu řešení úlohy 2.16 se říká diskretizační metoda. Posloupnost dělení $\{D_i\}$ můžeme definovat předem. Většinou jsou pak tato dělení ekvidistantní s vlastností $D_i \subset D_{i+1}$. Na druhou stranu může každé následující dělení záviset na výsledcích předchozích iterací.

Pro zajištění konvergence diskretizační metody bohužel nestačí jen zajistit splnění podmínky 2.18 (viz například [14]). Různé diskretizační metody se tak od sebe liší tím, jak přesně dělí množinu T. Jako příklad jednoho z kritérií můžeme předvést například výsledek uvedený v [15].

Označme

$$\lambda(x^F, D) := M \cap \{\mathbf{x} \in \mathbb{R}^n : \mathbf{c}^T\mathbf{x} \leq \mathbf{c}^T\mathbf{x}^\mathbf{F}\}.$$

Za předpokladu 2.18 a pokud dále pro nějaké $\mathbf{x}^F$ je množina $\lambda(\mathbf{x}^F, D)$ omezená, pak je hromadný bod posloupnosti řešení diskretizačních úloh řešením problému 2.16.

Podrobnější informace o diskretizačních metodách lze nalézt například v [8].

2.3.5 Algoritmy založené na metodě sečných nadrovin

Metody sečných nadrovin, které jsou hlavním cílem této práce, budou podrobněji vysvětleny v další kapitole. Zde uvedu pouze stručný úvod. Metody sečných nadrovin vycházejí z nějakého počátečního $\mathbf{x}^0$, které nepatří do množiny přípustných řešení úlohy 2.1. Přidáváním řezů se následně posunují nalezené $\mathbf{x}^k$ stále blíže k přípustné množině úlohy 2.1. Celý algoritmus pak vypadá takto:

1. Zvolíme počáteční jednoduchou optimalizační úlohu a tuto úlohu vyřešíme.
2. Přidáváme k existující úloze další podmínky tak, aby právě nalezené řešení v nově vytvořené množině řešení už neleželo (odtud metoda sečen).

3. Skončíme, pokud je nalezené meziřešení přípustným řešením úlohy 2.1 (případně pokud už toto řešení leží dostatečně blízko množiny přípustných řešení úlohy 2.1).

Kapitola 3

Metody sečných nadrovin

3.1 Úvod

Tato kapitola obsahuje rozsáhlejší zpracování metod, které jsou založeny na metodě sečných nadrovin. V první sekci je představena základní metoda sečných nadrovin pro úlohu semi-infinitního programování. Následují algoritmy upravující tuto základní metodu pro řešení úlohy semi-infinitního programování. Metody jsou seřazeny od nejjednodušší metody, která pro výpočet vyžaduje nalezení nejvíce porušené podmínky, a z takovéto porušené podmínky pak vytváří řez, přes metody, které se snaží tento požadavek postupně odstraňovat, až k metodě, která pro vytváření sečné nadroviny využívá libovolnou porušenou podmínku. V závěru je pak prezentován algoritmus, který se snaží zrychlit algoritmus vytvářející řezy z libovolné porušené podmínky.

3.2 Metoda sečných nadrovin

Metoda sečných nadrovin je jednou ze základních technik řešení úloh celočíselného a konvexního programování a některých druhů nekonvexního programování. Metoda se používá zejména v případech, kdy je množina přípustných řešení složitě popsána. Řešení touto metodou spočívá v jednoduchém iterativním postupu. V prvním kroku se vytvoří počáteční úloha, ve které je množina přípustných řešení nejčastěji konvexní polyedr obsahující původní množinu přípustných řešení a ve které je cílová funkce stejná jako v původní úloze. Úlohu definovanou v k-tém

kroku pak vyřešíme (například simplexovou metodou) a zkontrolujeme, zda-li nalezené optimální řešení padne do původní množiny přípustných řešení. Pokud ano, pak je toto řešení i optimálním řešením zadané úlohy. V opačném případě se vytvoří sečná nadrovina tak, aby se v jednom poloprostoru, který této nadrovině přísluší, nacházelo námi nalezené řešení a v druhém poloprostoru se nacházela původní množina přípustných řešení. Takto vytvořená sečná nadrovina se pak přidá k popisu již vyřešené podúlohy. Tím vznikne nová úloha s menší množinou přípustných řešení a můžeme pokračovat v iteraci řešením této úlohy v kroku k+1 algoritmu. V případě, kdy by neexistovalo optimální řešení v k-tém kroku úlohy, neexistovalo by ani optimální řešení zadané úlohy.

3.2.1 Metoda sečných nadrovin pro úlohu semi-infinitního programování

Potřebné definice

Jedna z prvních aplikací metody sečných nadrovin pro lineární semi-infinitní programování byla metoda prezentovaná v [1]. Uvažujme úlohu lineárního semi-infinitního programování v podobě

$$\begin{gathered} \min_{M} \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T(t)\mathbf{x} \geq b(t), t \in T, \mathbf{x} \in H\right\}. \end{gathered} \tag{3.1}$$

Navíc o úloze předpokládejme následující

1. $\mathbf{c} \in \mathbb{R}^n, \mathbf{c} \neq 0$;
2. T je kompaktní podmožinou $\mathbb{R}^m$;
3. $H \neq \emptyset$ je kompaktní konvexní podmožinou $\mathbb{R}^n$;
4. $\mathbf{a} : T \to \mathbb{R}^n$ je spojitá n-vektorová funkce na T;
5. $b : T \to \mathbb{R}$ je spojitá funkce na T;
6. existuje $\hat{\mathbf{x}} \in H$, pro které platí

$$\mathbf{a}^T(t)\hat{\mathbf{x}} > b(t)$$

pro každé $t \in T$ a které není optimální řešení úlohy 3.1.

Pro řešení úlohy 3.1 předložili autoři v [1] následující algoritmus.

Algoritmus 2:

1. Definujeme úlohu P_0 následovně

$$\min_M \mathbf{c}^T\mathbf{x},$$
$$M = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \in H\}.$$

   Položíme $k = 1$.

2. Získáme $\mathbf{x}^k \in \mathbb{R}^n$ jako řešení úlohy P_{k-1}.

3. Vyřešíme

$$\min_{t\in T}(\mathbf{a}^T(t)\mathbf{x}^k - b(t)).$$

   Nechť t^k je její řešení s vlastností

$$\mathbf{a}^T(t^k)\mathbf{x}^k - b(t^k) < 0.$$

   Pokud takové t^k neexistuje, ukončíme výpočet a vrátíme $\mathbf{x}^k$ jako optimální řešení úlohy 3.1.

4. Vytvoříme úlohu P_k tak, že přidáme podmínku

$$\mathbf{a}^T(t^k)\mathbf{x} \geq b(t^k) \tag{3.2}$$

   k úloze P_{k-1}. Položíme $k = k + 1$ a vrátíme se na 2.

V případě, že úloha P_{k-1} nemá optimální řešení, nemá řešení ani původní úloha 3.1. V takovém případě není třeba pokračovat ve výpočtu. Gustafson a Kortanek v práci [1] dokazují, že pokud se algoritmus nezastaví po konečném počtu kroků, je limita posloupnosti $\{\mathbf{x}^k\}_{k=1}^{\infty}$ optimálním řešením úlohy 3.1. Pokud je v úloze 3.1 množina H konvexním polyedrem, je každá z úloh P_k úlohou lineárního programování. V takovém případě není krok 2 obtížný. Naproti tomu v kroku 3 máme za úkol vyřešit úlohu

$$\min_{t\in T}(\mathbf{a}^T(t)x_k - b(t)).$$

Tato úloha bude obecně náročná vzhledem k obecnosti funkcí a a b. Pokud je $\mathbf{x}^k$ přípustné řešení aktuální iterace, krok 3 vyžaduje, abychom našli nejvíce porušenou podmínku a z té pak algoritmus vytváří řez.

Příklad

Uvažujme úlohu

$$\begin{gathered}\min_M\{-x_1-x_2\},\\ M=\left\{(x_1,x_2)\in\mathbb{R}^n:\ \begin{array}{c}(-x_1+x_2)*t-x_2\geq -4t^2+4t-4,\\ \forall t\in[0,1], x\in H\end{array}\right\},\\ H=\left\{\begin{array}{lcl}(x_1,x_2)\in\mathbb{R}^2 & | & x_1+2x_2 \leq 20\\ & & x_1,x_2 \geq 0\end{array}\right\}.\end{gathered} \tag{3.3}$$

Je vidět, že jde o úlohu typu 3.1, kde

$$\begin{gathered}n=2, c=(-1,-1),\\ \mathbf{a}: t\to(-t,t-1),\\ b: t\to -4t^2+4t-4.\end{gathered}$$

Než se pustíme do řešení tohoto příkladu, nejprve si jej orientačně nakreslíme. Místo $\min\{-x_1-x_2\}$ uvažujme $\max\{x_1+x_2\}$ a po úpravě dostáváme

$$\begin{gathered}\max_M\{x_1+x_2\}\\ M=\left\{(x_1,x_2)\in\mathbb{R}^2: tx_1+(1-t)x_2\leq 4t^2-4t+4, t\in T, x\in H\right\},\end{gathered}$$

což je maximalizační úloha s nekonečně mnoha podmínkami. Každá z podmínek (pro pevné $t \in T$) je sama o sobě lineární nerovností. Abychom pochopili, jak přesně úloha vypadá, uvažujme nad tím, jaké podmínky dostaneme v závislosti na $t \in T$. Zvolme pro začátek nějaké přípustné hodnoty $t \in T$.

Pro $t = 0$ dostáváme nerovnost

$$x_2 \leq 4,$$

pro $t = 1$ dostáváme nerovnost

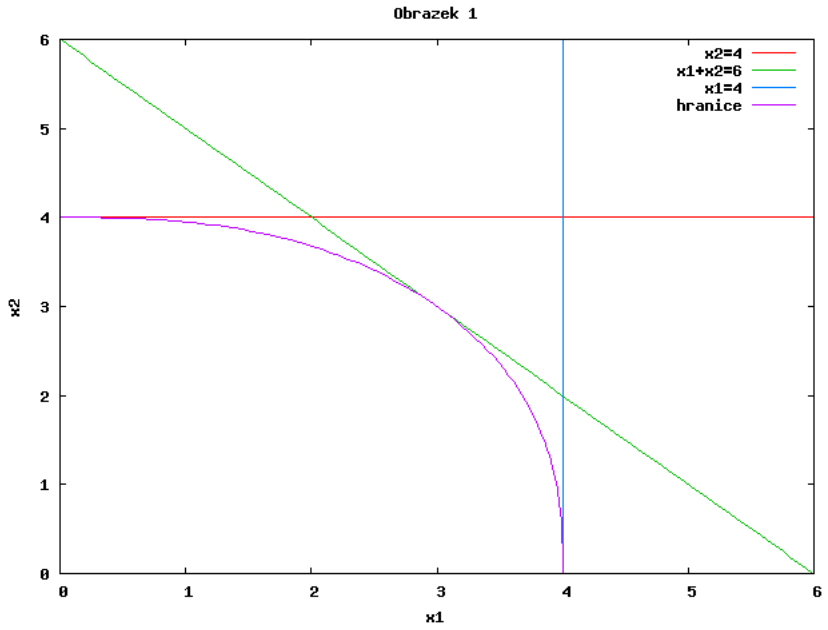
$$x_1 \leq 4,$$

a konečně pro $t = \frac{1}{2}$ dostáváme nerovnost

$$x_1 + x_2 \leq 6.$$

Při zkoumání levé strany nerovností zjistíme, že pokud pohybujeme proměnnou $t \in T$, měníme směrnici příslušné nerovnosti. Na pravé straně zase vidíme spojitou funkci $4t^2 - 4t + 4$, která je v intervalu $< 0, \frac{1}{2} >$ klesající, v intervalu $< \frac{1}{2}, 1 >$ rostoucí a minimum této funkce nastává právě v hodnotě $t = \frac{1}{2}$.

Nyní už můžeme přistoupit k tomu, že si úlohu nakreslíme.



Obrázek 3.1: Základní nákres problému 3.3

Na obrázku 3.1 vidíme znázorněnou množinu přípustných řešení úlohy 3.3. Na obrázku jsou znázorněny podmínky, které jsme před chvílí zmínili. Všechny podmínky úlohy pak vytvoří hranici množiny přípustných řešení, která je na obrázku také znázorněna. Z obrázku taktéž snadno vidíme, jaké řešení budeme metodou sečných nadrovin hledat. Protože maximalizujeme funkci $x_1 + x_2$, bude námi hledané řešení bod $\mathbf{x} = [3; 3]$. Dále lze z obrázku vypozorovat, že podmínka $x_1 + 2x_2 \leq 20$ z definice množiny H už množinu přípustných řešení více neovlivní. Tato podmínka je v příkladu kvůli omezenosti množiny H.

Nyní, když už víme, k čemu směřujeme, pokusme se úlohu 3.3 vyřešit metodou sečných nadrovin. Budeme postupovat podle algoritmu 2.

V prvním kroku algoritmu máme za úkol definovat počáteční úlohu. Ze zadání úlohy 3.3 vidíme, že počáteční úloha bude vypadat takto

$$\begin{gathered}\min_{M}(-x_1 - x_2),\\ M = \left\{(x_1, x_2) \in \mathbb{R}^2 : x_1 + 2x_2 \leq 20, x_1, x_2 \geq 0\right\}.\end{gathered} \tag{3.4}$$

Ve druhém kroku algoritmu máme vyřešit úlohu 3.4. Úlohu můžeme vyřešit například simplexovou metodou. Tato úloha má jediné optimální řešení, a sice $\mathbf{x}^1 = (20, 0)$ s optimální hodnotou cílové funkce -20.

Protože

$$\begin{aligned}(\mathbf{a}^T(t)\mathbf{x} - b(t)) &= (-x_1 + x_2)t - x_2 + 4t^2 - 4t + 4\\ &= 4t^2 + (-x_1 + x_2 - 4)t + 4 - x_2,\end{aligned}$$

řešíme ve třetím kroku algoritmu po dosazení za $\mathbf{x}$ hodnotu předchozího vypočteného řešení $\mathbf{x}^1 = (20, 0)$ úlohu

$$\min_{t \in T}(4t^2 - 24t + 4).$$

V této úloze se nabývá minima na množině $T = \{t \mid 0 \leq t \leq 1\}$ v $t^1 = 1$ s hodnotou cílové funkce -16. Protože je tato hodnota menší než nula, neskončíme, ale zavedeme první řez.

K úloze 3.4 přidáme novou podmínku, která bude po dosazení do obecného vyjádření 3.2 vypadat takto

$$-x_1 \geq -4.$$

Přidáním podmínky k úloze 3.4 získáme úlohu

$$\begin{gathered}\min_{M}(-x_1 - x_2),\\ M = \left\{(x_1, x_2) \in \mathbb{R}^2 : \begin{array}{rcr} x_1 + 2x_2 & \leq & 20\\ -x_1 & \geq & -4\\ x_1, x_2 & \geq & 0 \end{array}\right\}.\end{gathered} \tag{3.5}$$

Vrátíme se na krok 2 algoritmu 2. Nová úloha 3.5 má řešení $\mathbf{x}^2 = [4, 8]$ s hodnotou cílové funkce -12. V kroku 3 budeme nyní řešit úlohu

$$\min_{t \in T}(4t^2 - 4).$$

Tato úloha nabývá na množině T minimální hodnoty pro $t^2 = 0$ s hodnotou cílové funkce -4. Opět je tato hodnota menší než nula, a tak budeme opět přidávat řez. Řezem bude podmínka

$$-x_2 \geq -4.$$

Nová úloha bude po přidání podmínky k úloze 3.5 vypadat takto

$$\min_{M}(-x_1 - x_2)$$
$$M = \left\{ (x_1, x_2) \in \mathbb{R}^2 : \begin{array}{rcl} x_1 + 2x_2 & \leq & 20 \\ -x_1 & \geq & -4 \\ -x_2 & \geq & -4 \\ x_1, x_2 & \geq & 0 \end{array} \right\}. \tag{3.6}$$

Budeme-li uvedeným postupem pokračovat dále, získáme posloupnost $\{\mathbf{x}^k\}_{k=0}^{\infty}$:

$$\mathbf{x}^0 = [20, 0],$$
$$\mathbf{x}^1 = [4, 8],$$
$$\mathbf{x}^2 = [4, 4],$$
$$\mathbf{x}^3 = [4, 2],$$
$$\mathbf{x}^4 = [2, 4],$$
$$\mathbf{x}^5 = [2.5, 3.5],$$
$$\mathbf{x}^6 = [2.75, 3.25],$$
$$\mathbf{x}^7 = [2.87, 3.125],$$
$$\mathbf{x}^8 = [2.9375, 3.0625]$$
$$\vdots$$

Je vidět, že posloupnost konverguje k optimálnímu řešení úlohy 3.3, tj. k hodnotě $[3, 3]$.

3.3 Metody uvolňující požadavek na nejvíce porušenou podmínku

3.3.1 Úvod

Jak bylo uvedeno v závěru předchozího paragrafu, algoritmus sečných nadrovin (algoritmus 2) naráží na velmi obtížný problém hledání nejvíce porušené podmínky z množiny podmínek omezujících množinu přípustných řešení. Tato kapitola nabízí několik možných úprav základního algoritmu, které tento problém do jisté míry pomáhají řešit tím, že již nebude nutné hledat tuto nejvíce porušenou podmínku.

3.3.2 Základní úprava

Jedna z prvních změn algoritmu 2 se objevila v práci [2]. Autoři zde mírně upravili svůj algoritmus. Změnili krok 1 a krok 3. Podívejme se na tento obměněný algoritmus.

Algoritmus 3:

1. Definujeme úlohu P_0 stejně jako v kroku 1 algoritmu 2. Položíme $k = 1$. Zvolíme posloupnost $\{\epsilon^k\}_{k=1}^{\infty}$ nezáporných reálných čísel tak, aby splňovala podmínku

$$\lim_{k\to\infty} \epsilon^k \to 0.$$

2. Najdeme $\mathbf{x}^k \in \mathbb{R}^n$ jako optimální řešení úlohy P_{k-1}.

3. Položíme

$$\delta(\mathbf{x}^k) := \min_{t\in T}(\mathbf{a}^T(t)\mathbf{x}^k - b(t)).$$

   Pokud je $\delta(\mathbf{x}^k) \geq 0$, algoritmus končí a $\mathbf{x}^k$ je optimálním řešením úlohy 3.1. Jinak nalezneme $t^k \in T$, které splňuje podmínky

$$\mathbf{a}^T(t^k)\mathbf{x}^k - b(t^k) < 0,$$
$$\mathbf{a}^T(t^k)\mathbf{x}^k - b(t^k) \leq \delta(\mathbf{x}^k) + \epsilon^k.$$

4. Vytvoříme úlohu P_k tak, že přidáme podmínku

$$\mathbf{a}^T(t_k)\mathbf{x} \geq b(t_k)$$

k úloze P_{k-1}. Položíme $k := k+1$ a vrátíme se na krok 2.

Jak je vidět, krok 3 uvedeného algoritmu je jednoduchým zeslabením základního požadavku na nalezení nejvíce narušené podmínky. Požadavek na nalezení nejvíce narušené podmínky byl nahrazen požadavkem nalezení podmínky, která je v závislosti na ϵ dostatečně blízko nejvíce porušené podmínce.

3.3.3 Další uvolnění

V předchozím paragrafu byla navržena metoda, která mírně zmírňovala požadavek na hledání nejvíce porušené podmínky pro vytváření řezu při řešení úlohy lineárního semi-infinitního programování. V této sekci bude uvedena metoda prezentovaná v práci [3]. Metoda dovoluje zkonstruovat řez v libovolném $t_{k+1} \in T$, pro které je příslušná podmínka dostatečně porušená. Takto se lze vyhnout hledání podmínky, která je nejvíce porušená. Algoritmus skončí po konečně mnoha iteracích a vydá přibližné řešení se zvolenou přesností.

Uvažujme úlohu lineárního semi-infinitního programování ve tvaru

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}(t)^T\mathbf{x} \geq b(t), \forall t \in T\right\}. \end{gathered} \tag{3.7}$$

Pro danou množinu $T_k := \{t^1, t^2, \ldots, t^k\} \subset T$ definujeme úlohu lineárního programování 3.8 následovně

$$\begin{gathered} \min_M \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}(t^i)x \geq b(t^i), i = 1, 2, \ldots, k, \right. \\ \left. \mathbf{x} \geq 0. \right. \end{gathered} \tag{3.8}$$

Algoritmus 4:

1. Nastavíme $k := 1$, zvolíme jakékoliv $t_1 \in T$ a položíme $T_1 := \{t_1\}$. Zvolíme $\delta > 0$ libovolné, dostatečně malé.

2. Vyřešíme úlohu 3.8 na množině T_k a optimální řešení označme $\mathbf{x}^k = (x_1^k, x_2^k, \ldots, x_n^k)^T$.

3. Definujeme
$$\phi_k(t) := \mathbf{a}(t)^T \mathbf{x}^k - b(t).$$
Nalezneme jakékoli $t^{k+1} \in T$ takové, že $\phi_k(t^{k+1}) < -\delta$. Pokud takové t^{k+1} neexistuje, algoritmus skončí a vrátí $\mathbf{x}^k$ jako řešení. Jinak položíme $T_{k+1} := T_k \cup \{t_{k+1}\}$.

4. Položíme $k := k + 1$ a vrátíme se na krok 2.

Pokud by v 3.8 neexistovalo řešení, pak by neexistovalo řešení ani v 3.7. V takovém případě není důvod pokračovat v iteracích. Proto lze bez újmy na obecnosti předpokládat, že má 3.8 optimální řešení. Dále platí, že $t^{k+1} \notin T_k$, a pokud metoda skončí v kroku 3, pak výstupní řešení $\mathbf{x}^k$ bude také řešit úlohu 3.7 s dostatečně velkou přesností δ.

Duální úlohu k úloze 3.8 lze formulovat následovně

$$\max_M \mathbf{b}^T \mathbf{y},$$
$$M = \left\{ \mathbf{y} \in \mathbb{R}^k : \begin{array}{c} \sum_{i=1}^k a_j(t_i) y_i \leq c_j, j = 1, 2, \ldots, n \\ y_i \geq 0, i = 1, 2, \ldots, k. \end{array} \right\}. \tag{3.9}$$

Předpokládejme nyní, že 3.8 je řešitelná s optimální hodnotou $V(LP_k)$ a že úloha 3.9 je také řešitelná s optimální hodnotou $V(DLP_k)$. Pak platí níže uvedená věta.

Věta 5 *Pokud není optimální řešení DLP_{k+1} degenerované, pak $V(LP_{k+1}) > V(LP_k)$.*

Důkaz. Důkaz lze nalézt v textu [3]. □

Velikost přírůstku cílové funkce mezi úlohami $V(LP_{k+1})$ a $V(LP_k)$ lze dále zkoumat spolu s předpoklady, za kterých algoritmus skončí po konečně mnoha krocích s přesností δ (viz [3]). Pro zajištění konvergence algoritmu je potřeba zajistit omezenost množiny přípustných řešení úlohy 3.8, a dále aby v duální úloze 3.9 nedocházelo k degeneraci. Zbylé požadavky na konvergenci algoritmu se pak zajišťují volením δ dostatečně malým. Vydá-li nám algoritmus v kroku k^* nějaké řešení $\mathbf{x}^* \in \mathbb{R}^n$, otázkou zůstává, jak blízko bude $\mathbf{x}^*$ optimálnímu řešení úlohy 3.7. Označme

$$B_{k^*} := \{j : x_j^* > 0\}.$$

Věta 6 *Pro jakékoli $\delta > 0$ platí, že pokud existuje*

$$\bar{\mathbf{x}} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)^T,$$

pro které

$$\bar{x}_j \geq -\frac{x_j^*}{\delta}, \forall j \in B_{k^*},$$
$$\bar{x}_j \geq 0, \forall j \in B_{k^*}$$

tak, že

$$\sum_{j=1}^{k} \bar{x}_j f_j(t) \geq 1, \forall t \in T, \tag{3.10}$$

pak

$$|V(LP_{k^*}) - V(LSIP)| \leq \delta \left| \sum_{j=1}^{n} c_j \bar{x}_j \right|.$$

Důkaz. Důkaz lze nalézt v [3]. □

3.3.4 Největší uvolnění

V předchozí kapitole byla uvedena metoda, která hledá přibližné řešení úlohy 2.1 v případě, kdy neumíme najít nebo nechceme hledat nejvíce porušenou podmínku. Metoda dovolovala tvořit řez z podmínek, které jsou o nějaké přípustné δ vzdálené od nejvíce porušené podmínky. Toto umožňuje uživateli už celkem velkou volnost v hledání porušené podmínky. Někdy však ani takto relaxovaný požadavek nemusí vést k cíli. V případě, že uživatel není schopný najít nebo nechce hledat ani podmínku, která je vzdálena o δ od maximálně porušené podmínky, může se uživatel pokusit implementovat metodu prezentovanou níže.

Metoda určená pro řešení úlohy lineárního semi-infinitního programování, kterou nyní uvedeme, je založena na metodě Elzinga and Moora *Central-cutting-plane algorithm* pro řešení úlohy konvexního programování. Byla prezentovaná v textu [4].Tento algoritmus je schopný sestrojit sečnou nadrovinu z jakékoli podmínky z množiny porušených podmínek z popisu množiny přípustných řešení, a to bez ovlivnění lineární rychlosti konvergence metody. Pro zajištění rychlosti konvergence není tedy třeba hledat nejvíce porušenou podmínku, jak je tomu u jiných

metod.

Uvažujme stejnou úlohu lineárního semi-infinitního programování, jako v kapitole 3.2.1

$$\begin{gathered}\min_{M} \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T(t)\mathbf{x} \geq b(t), t \in T, \mathbf{x} \in H\right\}.\end{gathered} \tag{3.11}$$

Navíc předpokládejme splnění podmínek

1. $\mathbf{c} \in \mathbb{R}^n, \mathbf{c} \neq 0$,
2. T je kompaktní podmožinou $\mathbb{R}^m$,
3. $H \neq \emptyset$ je kompaktní konvexní podmožinou $\mathbb{R}^n$,
4. $\mathbf{a} : T \to \mathbb{R}^n$ je spojitá n-vektorová funkce na T,
5. $b : T \to \mathbb{R}$ je spojitá funkce na T,
6. existuje $\hat{\mathbf{x}} \in H$, pro které platí

   $$\mathbf{a}^T(t)\hat{\mathbf{x}} > b(t)$$

   pro každé $t \in T$ a které není optimální řešení úlohy 3.11.

Pro každé $\mathbf{x} \in \mathbb{R}^n$, nechť $\|\mathbf{x}\|$ označuje Euklidovskou normou, tj. $\|\mathbf{x}\| = \sqrt{(\mathbf{x}^T\mathbf{x})}$.

Označme V_P hodnotu cílové funkce v optimálním řešení úlohy 3.11

Algoritmus 5: 1. Zvolíme $\bar{v} > V_P$ a konstantu $\beta \in (0,1)$. Definujeme problém 3.12 následovně

$$\begin{gathered}\max_{M} \sigma, \\ M = \left\{(\mathbf{x}, \sigma) \in \mathbb{R}^{n+1} : \mathbf{c}^T\mathbf{x} + \|\mathbf{c}\|\sigma \leq \bar{v}, \mathbf{x} \in H\right\}.\end{gathered} \tag{3.12}$$

   Zvolme $\mathbf{y}^0 \in H$ a nechť $k = 1$.

2. Nechť $(\mathbf{x}^k, \sigma^k) \in \mathbb{R}^n \times \mathbb{R}$ je optimálním řešením úlohy SD_{k-1}. Jestliže je $\sigma^k = 0$, ukončíme výpočet a $\mathbf{x}^k$ je v takovém případě hledaným řešením úlohy 3.11. V opačném případě přejdeme na krok 3.

3. Pro zmenšení rozměru úlohy smažeme podmínky z SD_{k-1}, které vyhovují některému z mazacích pravidel uvedených níže. Vyřešíme problém SD_{k-1}.

4. Nyní záleží na tom, zda je $\mathbf{x}^k$ přípustným řešením úlohy 3.11:

   I Platí-li
   $$\mathbf{a}^T(t)\mathbf{x}^k \geq b(t) \text{ pro všechna } t \in T, \tag{3.13}$$
   pak přidáme podmínku
   $$\mathbf{c}^T\mathbf{x} + \|\mathbf{c}\|\sigma \leq c^T\mathbf{x}^k \tag{3.14}$$
   k úloze SD_{k-1} a položíme $\mathbf{y}^k := \mathbf{x}^k$.

   II V opačném případě nalezneme $t^k \in T$ takové, že pro něj platí
   $$\mathbf{a}^T(t^k)\mathbf{x}^k - b(t^k) \leq 0. \tag{3.15}$$
   Přidáme podmínku
   $$\mathbf{a}^T(t^k)\mathbf{x} - \|\mathbf{a}(t^k)\|\sigma \geq b(t^k) \tag{3.16}$$
   k úloze SD_{k-1} a nastavíme $\mathbf{y}^k := \mathbf{y}^{k-1}$.
   V obou případech položíme $k := k + 1$ a vrátíme se na krok 2.

První pravidlo mazání Smažeme podmínku $\mathbf{c}^T\mathbf{x} + \|c\|\sigma \leq \bar{v}$ nebo jakoukoli podmínku vytvořenou krokem 4 I v předchozích iteracích, pokud $\mathbf{x}^k$ je přípustným řešením úlohy 3.11, to znamená pokud
$$\min_{t\in T}(\mathbf{a}^T(t)\mathbf{x}^k - b(t)) \geq 0.$$

Druhé pravidlo mazání Smažeme takovou podmínku z množiny přípustných řešení úlohy SD_{k-1}, pokud zároveň platí

1. podmínka byla generována 4 II v j-té iteraci, kde $j < k$,
2. $\sigma^k \leq \beta\sigma^j$,
3. podmínka nebyla v úloze SD_{k-1} v bodě $(\mathbf{x}^k, \sigma^k)$ splněna jako rovnost (tj. platilo $\mathbf{a}^T(t^j)x_k - \|\mathbf{a}(t^j)\|\sigma^k > b(t^j)$).

Příklad

Řešme ještě jednou příklad uvedený v kapitole 3.2.1

$$\begin{gathered}\min_{M}(-x_1 - x_2),\\ M = \left\{(x_1,x_2)\in\mathbb{R}^n : \begin{array}{c}(-x_1+x_2)*t - x_2 \geq -4t^2+4t-4,\\ \forall t\in[0,1], x\in H\end{array}\right\},\\ H = \left\{\begin{array}{lcl}(x_1,x_2)\in\mathbb{R}^2 \quad | \quad x_1+2x_2 & \leq & 20\\ x_1,x_2 & \geq & 0\end{array}\right\}.\end{gathered} \tag{3.17}$$

Tentokrát budeme příklad řešit pomocí algoritmu 5. Připomeňme, že se optimální hodnoty cílové funkce úlohy 3.17 nabývá v bodě $[3,3]$.

Zvolme $\bar{v} := 1000$. Podle prvního kroku algoritmu vytvoříme počáteční úlohu

$$\begin{gathered}\max_{M}\sigma,\\ \text{M=}\left\{\begin{array}{lcl}(\mathbf{x},\sigma)\in\mathbb{R}^{n+1}: & & \\ -x_1-x_2+1.414 & \leq & 1000\\ x_1+2x_2 & \leq & 20\\ x_1,x2 & \geq & 0\end{array}\right\}.\end{gathered} \tag{3.18}$$

Zvolme dále $\mathbf{y}^0 \in H$ libovolně. Nechť je tedy $\mathbf{y}^0 := (1,1)$.

V dalším kroku algoritmu máme najít optimální řešení úlohy 3.18. Optimální řešení této úlohy je bod $(\mathbf{x}^1,\sigma^1) = (20,0,721.249)$ s optimální hodnotou cílové funkce 721.249. Tato hodnota je větší než nula, a můžeme tedy přistoupit k zavádění řezu.

Před zaváděním řezu máme nejprve ve třetím kroku algoritmu použít mazací pravidla k odstranění přebytečných podmínek. Nejsou však splněny předpoklady ani jednoho z pravidel mazání, a proto přistoupíme ke kroku 4 algoritmu.

Protože není bod $\mathbf{x}^1$ přípustným řešením úlohy 3.17, budeme zavádět řez podle kroku II. Máme nalézt libovolné $t^1 \in [0,1]$ tak, aby byla splněna nerovnost

$$4t^2 + (-x_1 + x_2 - 4)t + 4 - x_2 < 0.$$

Tato nerovnost je splněna například pro $t^1 := 0.429$. K úloze 3.18 tedy přidáme podmínku

$$-0.429x_1 + -0.571x_2 + -0.714\sigma \geq -3.02$$

a dostaneme následující úlohu

$$M = \left\{ \begin{array}{l} \max\limits_{M} \sigma, \\ (\mathbf{x}, \sigma) \in \mathbb{R}^{n+1} : \end{array} \begin{array}{rcrcrcr} -x_1 & - & x_2 & + & 1.414\sigma & \leq & 1000 \\ -0.429x_1 & - & 0.571x_2 & - & 0.714\sigma & \geq & -3.02 \\ x_1 & + & 2x_2 & & & \leq & 20 \\ & & & & x_1, x_2 & \geq & 0 \end{array} \right\}. \tag{3.19}$$

Zbývá položit $\mathbf{y}^1 := \mathbf{y}^0 = (1,1)$ a můžeme přejít zpět na krok 2 algoritmu.

Optimální řešení úlohy 3.19 je bod $(\mathbf{x}^2, \sigma^2) = (0,0,4.229)$ s optimální hodnotou cílové funkce 4.229. Tato hodnota je opět větší než nula a budeme tedy znovu zavádět řez.

Nejprve však ověříme, zdali nemůžeme nějakou podmínku z množiny M úlohy 3.19 odstranit. Podle prvního mazacího pravidla smažeme podmínku

$$-x_1 - x_2 + 1.414\sigma \leq 1000,$$

protože $\mathbf{x}^2$ je přípustné řešení úlohy 3.17. Když zkontrolujeme, zda-li nebudeme mazat nějakou další podmínku i podle druhého mazacího pravidla. Jediná podmínka vytvořená krokem II algoritmu je podmínka

$$-0.429x_1 + -0.571x_2 + -0.714\sigma \geq -3.02.$$

Podmínku však mazat nebudeme, neboť je v tomto kroku splněna jako rovnost.

Protože je $\mathbf{x}^2$ přípustným bodem úlohy 3.17, přistoupíme k zavádění řezu podle kroku I. Podle tohoto kroku přidáme následující podmínku

$$-x_1 - x_2 + 1.414\sigma \leq 0.$$

Celkově tedy v této iteraci algoritmu dostáváme úlohu

$$\max_{M} \sigma,$$

$$\mathrm{M}=\left\{ \begin{array}{rcrcrcr} (\mathbf{x},\sigma)\in\mathbb{R}^{n+1}: & & & & & & \\ -x_1 & - & x_2 & + & 1.414\sigma & \leq & 0 \\ -0.429x_1 & - & 0.571x_2 & - & 0.714\sigma & \geq & -3.02 \\ x_1 & + & 2x_2 & & & \leq & 20 \\ & & & & x_1,x_2 & \geq & 0 \end{array} \right\}. \tag{3.20}$$

Položíme $\mathbf{y}^2 := (0,0)$ a přistoupíme k další iteraci algoritmu.

V další iteraci algorimu dostaneme optimální řešení úlohy 3.20 $(\mathbf{x}^3,\sigma^3) = (3.234,0,2.287)$, kdy je $\mathbf{x}^3$ opět přípustným řešením úlohy 3.17. Stejně jako v předchozí iteraci smažeme podmínku podle prvního pravidla a druhá podmínka je splněna jako rovnost, takže ji nemůžeme vyloučit. Zavedeme řez podle kroku I. Dostaneme novou úlohu

$$\max_{M} \sigma,$$

$$M=\left\{ \begin{array}{rcrcrcr} (\mathbf{x},\sigma)\in\mathbb{R}^{n+1}: & & & & & & \\ -x_1 & - & x_2 & + & 1.414\sigma & \leq & -3.23 \\ -0.429x_1 & - & 0.571x_2 & - & 0.714\sigma & \geq & -3.02 \\ x_1 & + & 2x_2 & & & \leq & 20 \\ & & & & x_1,x_2 & \geq & 0 \end{array} \right\}. \tag{3.21}$$

Položíme $\mathbf{y}^3 := (3.234,0)$ a přejdeme na další iteraci.

Výsledky dalšího průběhu výpočtu závisí na volených t^k. Jeden z možných průběhů může vypadat takto:

Iterace	podmínek v úloze	$(\mathbf{x}^k, \sigma^k)$	zvolené t^k
1	2	$(20, 0, 721.24892)$	0.42889
2	3	$(0, 0, 4.22869)$	
3	3	$(3.23392, 0, 2.28673)$	
4	3	$(4.98272, 0, 1.23658)$	0.93626
5	4	$(2.8516, 1.8495, 1.0374)$	
6	3	$(0, 9.46060, 3.36550)$	0.11779
7	4	$(3.00931, 2.84116, 0.81274)$	
8	4	$(3.34361, 3.15448, 0.45794)$	0.56448
9	5	$(2.39832, 3.62172, 0.11991)$	0.32491
10	5	$(2.769658, 3.199420, 0.083873)$	
11	4	$(2.796498, 3.232536, 0.042394)$	0.37518
12	5	$(2.888885, 3.127470, 0.033429)$	0.45521
13	6	$(3.037114, 2.958899, 0.019046)$	
14	6	$(3.0439981, 2.9655211, 0.0095502)$	0.50369
15	6	$(2.9171825, 3.0814844, 0.0018764)$	0.48019
16	5	$(2.9669193, 3.0313794, 0.0016161)$	
		$\vdots$	

Pokud algoritmus neskončí po konečném počtu kroků, pak generuje posloupnost bodů $\{y_k\}_{k=\hat{k}}^{\infty}$. Limita této posloupnosti je optimálním řešením úlohy (LP). Důkaz tohoto tvrzení, jakož i důkaz rychlosti konvergence, kopíruje důkaz Elzinga a Moora pro jejich algoritmus pro řešení úloh konvexní programování.

Lemma 1 *Bez ohledu na to, zda-li se použijí mazací pravidla, platí, že pokud algoritmus neskončí, pak je* $\lim_{k\to\infty} \sigma^k = 0$.

Lemma 2 *Pokud je* $\tilde{\mathbf{x}}$ *přípustné řešení úlohy 3.11 a pro všechna* $t \in T$ *platí* $\mathbf{a}^T(t)\tilde{\mathbf{x}} > b(t)$, *pak existuje* $\tilde{\sigma} > 0$ *takové, že* $(\tilde{\mathbf{x}}, \tilde{\sigma})$ *vyhovuje všem nerovnostem, které představují řezy přidané kroky 4 a II algoritmu.*

Lemma 3 *Pokud se algoritmus zastaví v kroku* k^*, *pak* $\mathbf{y}^{k^*-1}$ *je přípustným řešením úlohy 3.11. Pokud se algoritmus nezastaví, pak existuje* $\hat{k}$ *takové, že* $\mathbf{y}^k$ *je přípustné řešení úlohy 3.11 pro každé* $k > \hat{k}$.

Věta 7 *Pokud se algoritmus zastaví v kroku* k^*, *pak* $\mathbf{y}^{k^*-1}$ *je optimální řešením úlohy 3.11. V opačném případě existuje limita posloupnosti* $\{\mathbf{y}^k\}_{k=\hat{k}}^{\infty}$, *a ta je optimálním řešením úlohy 3.11.*

Elzinga and Moore dokázali lineární rychlost konvergence pro jejich algoritmus. Analogickým způsobem lze dokázat lineární rychlost konvergence prezentovaného algoritmu.

Věta 8 *Mezi přípustnými body má v objektivní funkci algoritmus lineární rychlost konvergence.*

Kapitola 4

Programátorské zpracování

4.1 Úvod

Pro praktickou část diplomové práce jsem zvolil programovací jazyk Octave. Pro volbu tohoto programovacího jazyka pro mě bylo důležité, že se jedná o jazyk určený k provádění numerických výpočtů. Druhou velmi významnou vlastností Octave je, že se šíří pod licencí GNU General Public License a je jej tedy možné svobodně šířit.

V jazyce Octave jsem naprogramoval jednak kostry základního algoritmu metody sečných nadrovin pro úlohu semi-infinitní optimalizace a kostru algoritmu, který vytváří řez z libovolné porušené podmínky z množiny porušených podmínek. Metody jsem implementoval s ohledem na návrhový vzor *most*. Pro jednotlivé metody jsem dále implementoval aplikace na určité třídy úloh lineární semi-infinitní optimalizace.

Zdrojové kódy byly odladěny pod distribucí Octave, která je volně ke stažení. Vývoj probíhal v prostředí Cygwin v distribuci Octave 3.0.3.

Upozornění k verzi Octave 3.0.2

Během psaní diplomové práce jsem objevil chybu v distribuci Octave projevující se jen v prostředí Cygwin. Chyba byla velmi rychle opravena a po aplikaci příslušného opravného balíčku byly zdrojové kódy odladěny. Bohužel do odevzdání diplomové práce ještě nebyla aktualizovaná distribuce Octave pro Cygwin, a je proto nutné Octave nainstalovat přímo

ze zdrojových kódů dostupných již pro verzi 3.0.5. Podle informací od tvůrců Octave se chyba projevuje jen v distribuci pro Cygwin. Navíc by se měla distribuce Octave pro Cygwin v nejbližší době aktualizovat na verzi 3.0.3, ve které je již chyba opravena. Zdrojové kódy byly testovány ve virtuálním stroji, na kterém byl nainstalován systém Ubuntu, v distribuci Octave pro Ubuntu se chyba neprojevovala.

4.2 Octave

Octave je otevřený programovací jazyk, který se zaměřuje na numerické výpočty. Vznikl z důvodu potřeby alternativy k placeným numerickým balíkům, jako jsou například Matlab nebo Mathematica. Octave je na rozdíl od komerčních programů volně šiřitelný program, který lze svobodně používat, měnit nebo kopírovat za podmínky dodržování licence GNU General Public License, pod kterou je tento programovací jazyk distribuován.

Jazyk Octave byl vytvořen několika nadšenci jako program pro psaní vysokoškolské učebnice týkající se návrhu chemických reaktorů. Po několika letech je z něj již solidní prostředí, které v některých oblastech může konkurovat komerčním programům. Aktuální verze implementace jazyku Octave zvládá počítat s reálnými, komplexními nebo celočíselnými skaláry nebo maticemi, zvládá řešit soustavy nelineárních algebraických rovnic, integrovat funkce na konečných nebo nekonečných intervalech. Octave umí počítat mnoho různých matematických problémů od statistických, přes finanční matematiku, diferenciální počty, interpolaci a mnoho dalšího. Z optimalizační oblasti matematiky můžeme zmínit řešení úlohy lineárního, kvadratického programování a některé úlohy nelineárního programování.

Naprogramované metody tak mohou čtenáři posloužit jako rozšíření funcí implementace jazyku Octave také o řešení úlohy lineárního semi-infinitního programování.

4.3 Návrhový vzor most

Návrhové vzory (anglicky *design patterns*) představují představují v softwarovém inženýrství obecně definované řešení problémů vyvstávající při designu software. Návrhový vzor slouží jako šablona, která řeší nějaký obecný designový problém. Znalost návrhových vzorů není pro programování nezbytná, na druhou stranu může jejich znalost velice zefektivnit práci programátora, který již nemusí znovu vymýšlet to, co již jiní vymysleli. Navíc si může být programátor při použití správného návrhového vzoru jistý, že problémovou situaci řeší efektivně a korektně.

Most je návrhový vzor, který používáme v situacích, ve kterých je potřeba oddělit implementaci problému od jeho abstrakce. Protože algoritmy sečných nadrovin nedávají (a ani nemohou dát) přesnou odpověď na otázku, jak přesně hledat podmínku, ze které budeme vytvářet řez, implementoval jsem algoritmy obecněji tak, aby si mohl uživatel měnit metody hledající porušenou podmínku podle svých představ.

Pro bližší informace o návrhovém vzoru strategie lze čtenáře odkázat na knihu [19].

4.4 Metody sečných nadrovin

Obě implementované metody byly naprogramovány podobným způsobem. Hlavní kostra algoritmu je v jediné funkci, která vykonává kroky algoritmu příslušné metody tak, že volá funkce, které mají daný krok za úkol. Konkrétní implementace algoritmu pak spočívá v přepsání těchto funkcí funkcemi vlastními, které počítají konkrétní úlohu, která je implementována.

4.4.1 Základní metoda sečných nadrovin

Na tomto místě uvedu několik nejdůležitějších funkcí a stručně popíši, k čemu dané funkce slouží. Přesný tvar vstupních a požadavky na výstupní parametry těchto funkcí lze nalézt v dokumentaci.

- Základní metoda sečných nadrovin je definovaná ve funkci

```
function x = cutting_plane(sip_problem, steps).
```

Funkce je hlavní funkcí, kterou se zpouští algoritmus. Funkce má dva parametry. Prvním parametrem je problém, který se bude řešit. Druhým parametrem je maximální počet kroků, které se mají v metodě provést. Metoda vrátí nalezené optimální řešení. Pokud metoda řešení nenajde vrátí hodnotu Inf.

- Funkce

```
function program=create_first_program(sip_problem),
```

slouží pro vytváření první úlohy lineárního programování podle kroku 1 algoritmu 2.

- Funkce

```
function find_minimum_tk(xk,sip_problem),
```

slouží pro hledání porušené podmínky podle kroku 3 algoritmu 2.

- Funkce

```
function optimal(xk,tk,sip_problem),
```

slouží k ověření, zda-li je nalezené řešení optimálním řešením zadané úlohy.

- Funkce

```
function create_constraint(tk,sip_problem).
```

slouží pro vytvoření podmínky, která se bude přidávat k úloze.

Implementace základní metody

Ve stejném souboru, ve kterém je implementovaná základní metoda sečných nadrovin, je implementovaná aplikace základní metody, tj. jsou definovány funkce zmíněné v předchozím odstavci. Pokud tedy uživatel tyto funkce nepřepíše, umí aktuální implementace algoritmu řešit úlohy

typu

$$\begin{gathered}\min_{M} \mathbf{c}^T\mathbf{x}, \\ M = \left\{\mathbf{x} \in H : \mathbf{t}^T A\mathbf{x} + \mathbf{a}^T\mathbf{x} \geq \frac{1}{2}\mathbf{t}^T L\mathbf{t} + \mathbf{l}^t + l, \mathbf{t} \in T\right\}, \\ H = \left\{\mathbf{x} \in \mathbb{R}^n : A_1\mathbf{x} \leq \mathbf{b_1}, \mathbf{x} \geq 0\right\}, \\ T = \left\{\mathbf{t} \in \mathbb{R}^m : A_2\mathbf{t} \leq \mathbf{b_2}, \mathbf{t} \geq 0\right\}.\end{gathered} \tag{4.1}$$

Podrobnější informace o tom, jak přesně zadat úlohu a jak program spustit, lze nalézt v dokumentaci přiložené ke zdrojovým kódům, případně v komentářích zdrojových kódů.

4.4.2 Implementace mírně uvolňujících metod

Algoritmy zmíněné v kapitolách 3.3.2 a 3.3.3 lze implementovat pomocí metody zmíněné výše. Zde uvedeme velmi stručný návod pro implementaci daných metod.

- Funkci

  ```
  function find_minimum_tk(xk,sip_problem)
  ```

  je třeba implementovat tak, aby hledala příslušnou narušenou podmínku vzhledem ke kroku 3 algoritmů 3 respektive 4.

- Ve funkci

  ```
  function optimal(xk,tk,sip_problem)
  ```

  je třeba implementovat potřebné unkončovací kritérium pro jednotlivé algoritmy.

4.5 Centrální metoda sečných nadrovin

Zde uvedeme nejdůležitější funkce naprogramované pro metodu sečných nadrovin prezentovanou v kapitole 3.3.4.

- Centrální metoda je definovaná ve funkci

  ```
  function x = central_cutting_plane(sip_problem, steps =
     20, beta = 0.1).
  ```

První dva parametry funkce jsou stejné jako v předchozí metodě. Poslední parametr je vstupní parametr metody, která ovlivňuje mazání podmínek podle druhého mazacího pravidla algoritmu 5. Pro implementování algoritmu je nutné implementovat funkce a$(t), b(t)$ z definice algoritmu podle návodu v dokumentaci a upravit metodu .

- Funkce

```
function program = create_first_central_program(
   sip_problem)
```

  slouží pro vytvoření počáteční úlohy podle kroku 3.12 algoritmu 5.

- Funkce

```
function optimal(xk,tk,sip_problem)
```

  slouží k ověření, zda-li je nalezené řešení optimálním řešením zadané úlohy.

- Funkce

```
function bool=feasible(x,problem)
```

  slouží pro ověření, zda-li je vstupní prvek x přípustným řešením zadaného problému podle kroku 4 algoritmu 5.

- Funkce

```
function t=find_any_tk(program,x,problem)
```

  má za úkol hledat nějakou porušenou podmínku podle kroku II algoritmu 5.

4.5.1 Implementace centrální metody sečných nadrovin

Implementace centrální metody se liší od předchozí implementace v několika bodech. Na jednu stranu řeší implementace metody obecnější úlohu, na druhou stranu se však implementuje hledání porušené podmínky a přípustnosti řešení metodou Monte Carlo. Implementace řeší

úlohy typu

$$\begin{gathered} \min_{M} \mathbf{c}^T \mathbf{x}, \\ M = \left\{ \mathbf{x} \in H : a(t)^T \mathbf{x} + \mathbf{a}^T \mathbf{x} \geq b(t), t \in [0,1] \right\}, \\ H = \left\{ \mathbf{x} \in \mathbb{R}^n : A_1 \mathbf{x} \leq \mathbf{b_1}, \mathbf{x} \geq 0 \right\}. \end{gathered} \tag{4.2}$$

Podrobnější informace o tom, jak přesně zadat úlohu a jak program spustit, lze opět najít v dokumentaci přiložené ke zdrojovým kódům, případně komentáře zdrojových kódů.

Kapitola 5

Závěr

Cílem práce bylo nastudovat a přiblížit několik přístupů k řešení úloh lineárního semi-infinitního programování. V práci jsem prezentoval základní druhy metod pro řešení úlohy lineárního semi-infinitního programování. Podrobněji jsem se věnoval metodě sečných nadrovin pro úlohu lineárního semi-infinitního programování. Ukázal jsem, jakými způsoby je možné v této metodě uvolňovat základní požadavek na nalezení nejvíce porušené podmínky. Na dvou příkladech jsem ukázal, jak pracují dvě varianty metody sečných nadrovin pro úlohu lineárního semi-infinitního programování. Schéma těchto dvou metod jsem implementoval pro určité třídy úloh lineárního semi-infinitního programování v matematickém programovacím jazyce Octave.

K sepsání práce jsem sestudoval 16 prací různých autorů. Uvedenou literaturu jsem zpracoval a sjednotil značení, abych získal kompaktní celek. Zaměřil jsem se na metody sečných nadrovin na základě hodnocení v práci [3] a vlastního uvážení. Základní metodu a její modifikaci jsem ilustroval vlastními příklady. Vlastní je také mé programové zpracování těchto metod.

5.1 Možná rozšíření

Při zpracovnání teoretické části bylo třeba vynechat množství materiálů týkajících se jiných metod, než metod sečných nadrovin. Další teoretické rozšiřování práce by se tak mohlo zaměřovat podrobněji na jiné typy úlohy lineárního semi-infinitního programování, a to zejména na metody

řešící duální úlohu. Některé z uvedených metod lze zobecnit pro úlohu nelineárního semi-infinitního programování, jiné rozšíření práce by se tak mohlo zabývat úlohou nelineárního semi-infinitního programování.

Vzhledem k většímu rozsahu práce a také zaměření na metody sečných nadrovin jsem do práce nezařadil implementaci dvoufázového algoritmu, který nejprve vyhledává přibližné řešení metodou diskretizace, a následně zpřesňuje toto řešení pomocí redukční metody. V dalším pokračování v této oblasti bych uvažoval o jeho publikaci.

Literatura

[1] Gustafson S.-Å., Kortanek K. O.: *Numerical Solution of a Class of Semi-infinite Programming Problems*, NRLQ, Vol. 20 (1973), str. 477-504

[2] Gustafson S.-Å., Kortanek K. O.: *Computational Schemes for Semi-Infinite Programs*, Technical Report, Department of Mathematics, Carnegie-Mellon University, Pittsburgh, PA.

[3] Wu S. Y., Fang S. C., Lin C. J.: *Relaxed Cutting Plane Method for Solving Linear Semi-Infinite Programming Problems*, Journal of optimization theory and applications: Vol.99, No.3, pp. 759 779, December 1998

[4] Gribik P. R.: *Central-cutting-plane algorithm for semi-infinite programming problems*, Department of Mathematics Carnegie-Mellon University Pittsburg, PA 15213 USA

[5] Goberna M. A.: *Linear Semi-infinite Optimization:Recent Advances* Dep. de Estadística e Investigación Operativa, Universidad de Alicante, Spain

[6] Hettich R. (1978): *Lecture Notes in Control and Information Sciences*, Springer-Verlag Berlin

[7] Anderson, E. J. , Nash P. (1987) *Linear programming in infinite-dimensional spaces ISBN 0 471 91250 6.*

[8] Still G.: Discretization in semi-infinite programming: The rate of approximation*, University of Twente, 2001*

[9] Dong-Hui L., Qi L., Tam J., Wu S.-Y.: A Smoothing Newton Method for Semi-Infinite Programming*, Journal of Global Optimization 30: 169-194, 2004*

[10] R.Hettich and H.Th.Jongen, Semi-infinite programing: Theory, methods and applications*, Lectures Notes in Control and Information Science, vol.7, springer ed., 1978, pp 1-11.*

[11] M.C.Ferris and A.B Philpott, An Interior Point Algorithm for semi-infinite linear programming*, Cambridge University Engineering Department, Cambridge, England ,1988*

[12] R.J. Vanderbei, M.S. Meketon and B.A.Freedman A modification of Karmarkar's algorithm*, Algorithmica 1 (1986) 395-407*

[13] K. Glashoff Duality theory of semi-infinite programming*, Hamburg 2000*

[14] R. Reemtsen, S. Gorner Numerical Methods for Semi-Infinite Programming: A Survey', Semi-Infinite Programing*, R. Reemtsen and J.-J. Ruckmann (eds.). Kluwer, Bsoton-London-Dordrecht, 1998, pp. 195-275.*

[15] R. Reemtsen Some other aproximation methods for semi-infinite optimization problems*, J. Comp. Appl. Math 53 (1994), 87-108*

[16] Ana I.P.N. Pereira, Edite M.G.P. Fernandes A reduction method for semi-infinite programming*, Polytechnic Institute of Braganca, Braganca, Portugal,2001*

[17] HETTICH, R., and KORTANEK, K. O., Semi-Infinite Programming: Theory, Method, and Applications, SIAM Review, Vol. 35, pp. 380-429, 1993

[18] Klaus Glashoff *Duality theory of semi-infinite programming*, University of Hamburg, Institute for Applied Mathematics, Bundesstrase 55, D-2000 Hamburg 13, West-Germany

[19] Rudolf Pecinovský *Návrhové vzory*, Computer Press, a.s. Brno 2007

Dodatek A

Popis přiloženého CD

Na přiloženém CD lze nalézt text této práce spolu se zdrojovými kódy naprogramovaných metod. CD obsahuje složky:

`dipl/`, ve které se nachází tyto soubory:

- `diplomova_prace.pdf` elektronická verze této práce ve formátu pdf

`cutting_plane/`, která dále obsahuje:

- `cutting_plane.m` naprogramovaná metoda sečných nadrovin
- `cutting_plane.txt` dokumentace k naprogramované metodě sečných nadrovin
- `central_cutting_plane.m` naprogramovaná centrální metoda sečných nadrovin
- `central_cutting_plane.txt` dokumentace k centrální metodě sečných nadrovin

`octave/`, ve které se nachází zdrojové kódy jazyka Octave.