UNIVERZITA KOMENSKÉHO V BRATISLAVE
Fakulta matematiky, fyziky a informatiky
Ekonomická a finančná matematika

Vnorená L-Shaped metóda s adaptívnou agregáciou rezov optimality

Diplomová práca

Diplomant: Bc. Michal Dobiaš

Vedúci diplomovej práce: Mgr. Soňa Kilianová, PhD.

Bratislava 2011

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky
Katedra aplikovanej matematiky a štatistiky

VNORENÁ L-SHAPED METÓDA S ADAPTÍVNOU AGREGÁCIOU REZOV OPTIMALITY

Diplomová práca

Bc. Michal DOBIAŠ

Študijný odbor: 9.1.9 Aplikovaná matematika

Študijný program: Ekonomická a finančná matematika

Kód práce: d0c95cc8-8074-4be3-a6cf-99cd8e1a90a3

Vedúci práce: Mgr. Soňa Kilianová, PhD.

BRATISLAVA 2011

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Michal Dobiaš
Študijný program: ekonomická a finančná matematika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.1.9. aplikovaná matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov : Vnorená L-Shaped metóda s adaptívnou agregáciou rezov optimality

Cieľ : Cieľom práce je štúdium základných algoritmov lineárneho stochastického programovania pre dvojstupňové a viacstupňové úlohy. Na základe už existujúceho modifikovaného algoritmu pre dvojstupňové úlohy navrhnúť jeho zovšeobecnenú verziu pre viacstupňové úlohy. Numerická implementácia algoritmov s cieľom porovnania efektívnosti navrhnutého algoritmu - vnorenej L-Shaped metódy s adaptívnou agregáciou rezov optimality - v porovnaní so základnými algoritmami.

Vedúci : Mgr. Soňa Kilianová, PhD.

Dátum zadania: 10.02.2010

Dátum schválenia: 15.03.2011

.....
prof. RNDr. Daniel Ševčovič, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Dátum potvrdenia finálnej verzie práce, súhlas s jej odovzdaním (vrátane spôsobu sprístupnenia)

.....
vedúci práce

Čestné prehlásenie

Prehlasujem, že som diplomovú prácu vypracoval samostatne s použitím konzultácií, nadobudnutých vedomostí a uvedenej odbornej literatúry.

Bratislava 20. 3. 2011 .....

Vlastnoručný podpis

Poďakovanie

Ďakujem svojej vedúcej diplomovej práce Mgr. Soni Kilianovej, PhD. za starostlivé vedenie, odborné rady a cenné pripomienky, ktoré mi pomohli pri vypracovávaní tejto práce.

Michal Dobiaš

DOBIAŠ, Michal: Algoritmy stochastického programovania.
|Diplomová práca| - Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky; Katedra aplikovanej matematiky a štatistiky.
- Vedúci diplomovej práce: Mgr. Soňa Kilianová, PhD.
- Bratislava: FMFI UK, 2011 /81 s./

Abstrakt

V diplomovej práci sa zaoberáme algoritmami na riešenie dvojstupňových a viacstupňových stochastických lineárnych úloh (SLÚ) a uvádzame základnú matematickú teóriu príslušných úloh a algoritmov. Predstavíme a odôvodníme algoritmus L-Shaped metódy pre riešenie dvojstupňových úloh, ako i jeho viacrezovú a adaptívnu modifikáciu. Následne uvádzame vnorenú L-Shaped metódu pre viacstupňové SLÚ a navrhneme jej adaptívnu verziu vychádzajúc z obdobnej modifikácie L-Shaped metódy pre dvojstupňové SLÚ. Efektívnosť algoritmických modifikácií vnorenej L-Shaped metódy vyhodnotíme v numerickom experimente na vzorke dostupných štandardných úloh.

Kľúčové slová:

Stochastické programovanie, L-Shaped metóda, Bendersova dekompozícia, Adaptívna agregácia rezov optimality

Abstract

In this master thesis we consider algorithms for solving two-stage and multistage stochastic linear problems. We introduce basic mathematical theory for these problems and algorithms. L-Shaped method algorithm is given along with insight of its functional mechanism of cutting planes. Multicut L-Shaped method and adaptive modification of L-Shaped method is introduced consequently. Then we consider Nested L-Shaped method (Benders decomposition) algorithm for solving multistage stochastic linear problems. We propose adaptive modification of Nested L-Shaped method based on similar L-Shaped method (two stage) modification. Effectiveness of modificated Nested L-Shaped method is compared by the means of numerical experiment on some of the available standard problem instances.

Keywords:

Stochastic programming, L-Shaped method, Benders decomposition, Adaptive optimality cut aggregation

Obsah

Úvod **2**

1 Lineárne úlohy **3**
1.1 Formulácia lineárnych úloh 3
1.2 Motivácia pre riešenie stochastických lineárnych úloh s kompenzáciou 4

2 Dvojstupňové stochastické lineárne úlohy (SLÚ) s pevnou kompenzáciou **7**
2.1 Formulácia úlohy a deterministický ekvivalent 7
2.2 Množiny prípustnosti 9
2.3 Špecifické typy stochastických lineárnych úloh s pevnou kompenzáciou 10
2.4 Hodnotová funkcia druhého stupňa 12
2.5 Podmienky optimality 13

3 Algoritmy pre dvojstupňové SLÚ s pevnou kompenzáciou **14**
3.1 L-Shaped metóda 14
3.1.1 Algoritmus 14
3.1.2 Princíp fungovania algoritmu 16
3.2 Viacrezová L-Shaped metóda 18
3.2.1 Algoritmus 19
3.3 L-Shaped metóda s adaptívnou agregáciou rezov optimality 20
3.3.1 Algoritmus 22
3.3.2 Pravidlá agregácie 23
3.3.3 Modifikácia pôvodného algoritmu 23

4 Viacstupňové stochastické lineárne úlohy s pevnou kompenzáciou **27**
4.1 Formulácia úlohy a deterministický ekvivalent 27
4.2 Stromy scenárov 29

5 Algoritmy pre viacstupňové SLÚ s pevnou kompenzáciou **30**
5.1 Vnorená L-Shaped metóda (Bendersov dekompozičný algoritmus) . 30
5.1.1 Algoritmus 31
5.2 Vnorená L-Shaped metóda s adaptívnou agregáciou rezov optimality 33
5.2.1 Algoritmus 35
5.2.2 Pravidlá agregácie 37

6 Numerický experiment **39**
6.1 Riešené úlohy 39
6.1.1 Pltexp 39
6.1.2 Bonds 40
6.1.3 Scfxm 42
6.1.4 Scsd 42

6.1.5 Numerické instancie úloh 43
6.2 Experiment 44
6.2.1 Systém a nastavenia 44
6.2.2 Numerické výsledky 44

Záver **52**

Zoznam použitej literatúry **52**

Tabuľky **53**

Listing programu **60**

Úvod

Témou tejto diplomovej práce sú algoritmy stochastického programovania. Stochastické programovanie sa využíva v širokej škále aplikácií. Ako príklad možno uviesť optimálnu alokáciu zdrojov [1], [2], [6], plánovanie prenosovej kapacity telekomunikačných liniek [13], či plánovanie objemu výroby [1], [6], [13]. Problémy z reálneho sveta, ktoré vedú na úlohy stochastického programovania sú veľmi rozmanité, z čoho vyplýva aj rozmanitý charakter takýchto úloh. V tejto práci sa zameriame na lineárne stochastické úlohy s určitými vlastnosťami, ktoré budú uvedené pri ich formulácii.

Úlohy stochastického programovania v praxi často nadobúdajú značné rozmery, preto je dôležité mať k dispozícii efektívne algoritmy na ich riešenie. Za účelom zvyšovania efektivity sa vyvíjajú stále nové algoritmy a modifikácie existujúcich algoritmov, napríklad [11]. Preto sme sa aj my v tejto práci rozhodli navrhnúť a otestovať určité modifikácie v algoritme vnorenej L-Shaped metódy. Tento upravený algoritmus je v práci uvedený pod názvom 'vnorená L-Shaped metóda s adaptívnou agregáciou rezov optimality'.

Cieľmi tejto diplomovej práce boli:

1. **Štúdium teórie a algoritmov lineárneho stochastického programovania.**
Cieľ bol realizovaný štúdiom literatúry a odborných článkov uvádzaných v práci ([2], [3], [7], [8], [11], [14]).

2. **Návrh na modifikáciu algoritmu L-Shaped metódy s adaptívnou agregáciou rezov optimality pre dvojstupňové SLÚ na riešenie viacstupňových SLÚ**
Cieľ bol realizovaný naprogramovaním základnej vnorenej L-Shaped metódy na riešenie viacstupňových stochastických lineárnych úloh (SLÚ), ako i následným navrhnutím a naprogramovaním algoritmu vnorenej L-Shaped metódy s adaptívnou agregáciou rezov optimality (AARO) vychádzajúc zo zdrojov [2] a [11]. Oba programové kódy v jazyku MATLAB sú súčasťou príloh tejto práce. V rámci prípravných prác bola v MATLABe naprogramovaná taktiež L-Shaped metóda na riešenie dvojstupňových stochastických lineárnych úloh a jej viacrezová modifikácia. Tieto programové kódy nie sú v práci uvedené, samotné algoritmy však možno nájsť v častiach 3.1 a 3.2.

3. **Numerická implementácia navrhnutého algoritmu a vyhodnotenie jeho výkonnosti**
V rámci práce bol vykonaný numerický experiment, v ktorom sme na viacerých štandardných úlohách porovnali efektívnosť naprogramovaných algoritmov, konkrétne vnorenú L-Shaped metódu, viacrezovú vnorenú L-Shaped metódu a jej adaptívnu

verziu.

Za vlastný prínos práce pokladáme splnené ciele 2 a 3, ako aj odhalenie pravdepodobného nedostatku v Kroku 2b (v tejto práci to zodpovedá Kroku 3b) Algoritmu 3.3.1 uvedeného v zdroji [11], ktorý spomenieme v časti 3.3. Taktiež zásobník (vylepšenie algoritmu z [2]).

Štruktúra práce je nasledovná.

Prvá kapitola pojednáva o formáte zápisu deterministických lineárnych úloh a následne predstavuje motiváciu riešenia SLÚ s kompenzáciou.

V druhej kapitole sformulujeme dvojstupňovú SLÚ s pevnou kompenzáciou a jej deterministický ekvivalent. Následne uvedieme základnú teóriu a predstavíme špecifické prípady pre SLÚ s pevnou kompenzáciou.

Tretia kapitola sa skladá z troch častí. Najprv v nej predstavíme základný algoritmus L-Shaped metódy na riešenie dvojstupňových SLÚ s pevnou kompenzáciou a popíšeme princíp jeho fungovania. Ďalej uvedieme viacrezovú modifikáciu L-Shaped metódy a napokon sa venujeme L-Shaped metóde s AARO pre dvojstupňové SLÚ.

V štvrtej kapitole najprv sformulujeme viacstupňovú SLÚ s pevnou kompenzáciou, následne zadefinujeme pojem strom scenárov.

Piata kapitola uvádza algoritmus vnorenej L-Shaped metódy pre viacstupňové SLÚ podľa [2]. Následne navrhujeme modifikáciu algoritmu využívajúceho AARO.

Šiesta kapitola sa týka numerického experimentu. Najprv tu uvedieme ciele numerického experimentu a popíšeme riešené úlohy. Následne uvedieme numerické výsledky s interpretáciou. V tejto kapitole tiež porovnávame efektívnosť vnorenej L-Shaped metódy s AARO a vnorenej L-Shaped metódy z literatúry [2].

Ďalej nasleduje záver, zoznam použitej literatúry, tabuľková príloha a listing zdrojových kódov v jazyku MATLAB.

1 Lineárne úlohy

Hoci deterministické úlohy lineárneho programovania ako také nie sú hlavnou náplňou tejto práce, úzko s ňou súvisia. Predstavujú model, s ktorým budeme často pracovať a nadväzovať naň. Považujeme preto za užitočné, najprv objasniť určité základy týkajúce sa formulácie lineárnych úloh. Predíde sa tak prípadným nejasnostiam, ktoré by mohli vzniknúť z nejednotnosti zápisu v literatúre. Riešenie lineárnych úloh v reálnej praxi nám taktiež poslúži ako motivačný príklad pre potrebu zavedenia a riešenia stochastických lineárnych úloh s kompenzáciou.

1.1 Formulácia lineárnych úloh

Štandardná formulácia lineárnych úloh má podľa [7] často tvar

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \boldsymbol{c}^\top \boldsymbol{x} \\ A\boldsymbol{x} \ & \propto \ \boldsymbol{b}, \\ \boldsymbol{l} \leq \boldsymbol{x} \ & \leq \ \boldsymbol{u}, \end{aligned} \tag{1.1}$$

kde $A \in \mathbb{R}^{m \times n}$ je matica koeficientov, $\boldsymbol{c} \in \mathbb{R}^n$ účelový gradient, $\boldsymbol{b} \in \mathbb{R}^m$ vektor pravej strany, $\boldsymbol{l} \in \mathbb{R}^n$ a $\boldsymbol{u} \in \mathbb{R}^n$ predstavujú dolné a horné ohraničenie na $\boldsymbol{x}$. Všetky uvedené dáta sú nestochastické. V prípade neohraničenosti niektorého x_i máme $l_i = -\infty$, prípadne $u_i = \infty$. Operátor $\propto$ zastrešuje možnosti ohraničení typov '$\leq$', '$\geq$' a '$=$', ktoré sa v riadkoch môžu nachádzať v ľubovoľnom poradí.

Uvažujme ohraničenia pre $\boldsymbol{x}$. Zavedením substitúcie $\boldsymbol{x} = \boldsymbol{x}^+ - \boldsymbol{x}^-$, kde $\boldsymbol{x}^+ \in \mathbb{R}^n_+$, $\boldsymbol{x}^- \in \mathbb{R}^n_+$ a pridaním doplnkových premenných $\boldsymbol{y} \in \mathbb{R}^n_+$ a $\boldsymbol{z} \in \mathbb{R}^n_+$, podľa [7], vieme ohraničenia $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$ transformovať do tvaru:

$$\begin{aligned} \boldsymbol{x}^+ - \boldsymbol{x}^- \ - \boldsymbol{y} \ \ &= \ \boldsymbol{l} \\ \boldsymbol{x}^+ - \boldsymbol{x}^- \ + \boldsymbol{z} \ &= \ \boldsymbol{u} \\ \boldsymbol{x}^+, \ \boldsymbol{x}^-, \ \boldsymbol{y}, \ \boldsymbol{z} \ &\geq \ \boldsymbol{0}. \end{aligned}$$

Tým sa úloha (1.1) transformuje na tvar:

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \boldsymbol{c}^\top \boldsymbol{x} \\ A\boldsymbol{x} \ & \propto \ \boldsymbol{b}, \\ \boldsymbol{x} \ & \geq \ \boldsymbol{0}. \end{aligned} \tag{1.2}$$

Vidno, že úlohu v tvare (1.2) možno spätne previesť na úlohu v tvare (1.1), k tomu stačí v (1.1) položiť $\boldsymbol{l} = (0, ..., 0)^\top$ a $\boldsymbol{u} = (\infty, ..., \infty)^\top$. Ďalej sa pozrime na ohraničenia $A\boldsymbol{x} \propto \boldsymbol{b}$. Predpokladajme, že obsahujú nejaké nerovnosti typu '$\leq$' a/alebo '$\geq$', prípadne rovnosti. Nerovnosť typu $a_{i1}x_1 + ... + a_{in}x_n \leq b_i$ vieme zavedením doplnkovej premennej $w_i \in \mathbb{R}_+$ previesť na ekvivalentnú rovnosť $a_{i1}x_1 + ... + a_{in}x_n + w_i = b_i$.

Opačnú nerovnosť vieme analogicky previesť na rovnosť tak, že kladnú doplnkovú premennú odčítame.

Uvedenými transformáciami teda vieme úlohu tvaru (1.2) previesť do nasledovného ekvivalentného[1] tvaru:

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \boldsymbol{c}^\top \boldsymbol{x} \\ A\boldsymbol{x} &= \boldsymbol{b}, \\ \boldsymbol{x} &\geq \boldsymbol{0}. \end{aligned} \tag{1.3}$$

Formuláciu (1.3) budeme považovať za všeobecný tvar úlohy lineárneho programovania, do ktorého vieme previesť úlohu lineárneho programovania zapísanú v ľubovoľnom inom tvare.

1.2 Motivácia pre riešenie stochastických lineárnych úloh s kompenzáciou

S potrebou riešenia úloh lineárneho programovania sa stretávame v širokej škále aplikácií z reálneho sveta, pretože principiálne predstavujú vhodný model pre rozmanité typy problémov. V praxi sa však často stáva, že nemáme k dispozícii fixné, nestochastické dáta, ktoré sú predpokladom pre priamu aplikáciu modelov typu (1.3). V mnohých prípadoch sa jedná o náhodné premenné, pre ktoré poznáme typ rozdelenia, pričom jeho parametre odhadujeme z dát, ktoré máme k dispozícii. Môže sa zdať rozumné, takéto náhodné premenné nahradiť odhadom ich strednej hodnoty a následne riešiť deterministickú lineárnu úlohu, to však vo všeobecnosti nevedie k správnym výsledkom [2].

Uvažujme nasledovný produkčný problém, uvádzaný ako príklad v [7]:

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \boldsymbol{c}^\top \boldsymbol{x} \\ A\boldsymbol{x} &= \boldsymbol{b}, \\ T\boldsymbol{x} &= \boldsymbol{h}, \\ \boldsymbol{x} &\geq \boldsymbol{0}. \end{aligned} \tag{1.4}$$

T a $\boldsymbol{h}$ obsahujú premenné, predstavujúce produktivity, prípadne dopyty alebo kapacity, pričom tieto môžu mať stochastický charakter. Vektor $\boldsymbol{x}$ predstavuje rozhodnutie, ktoré sa musí vykonať ešte pred realizáciou náhodných dát. Pre rozhodnutie $\boldsymbol{x}$, ktoré by sme získali riešením uvažujúc iba deterministické ohraničenie $A\boldsymbol{x} = \boldsymbol{b}$, by po realizácii náhodných dát s vysokou pravdepodobnosťou platilo $T\boldsymbol{x} \neq \boldsymbol{h}$. V

[1] Poznamenávame, že ohraničenie typu rovnosť vieme jednoducho prepísať na dve ohraničenia typu nerovnosť ($ax = b \Leftrightarrow (ax \leq b \wedge ax \geq b)$).

prípade riešenia aproximatívnej úlohy tvaru

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \boldsymbol{c}^\top \boldsymbol{x} \\ A\boldsymbol{x} &= \boldsymbol{b} \\ \overline{T}\boldsymbol{x} &= \overline{\boldsymbol{h}} \\ \boldsymbol{x} &\geq \boldsymbol{0}, \end{aligned} \tag{1.5}$$

kde náhodné T a $\boldsymbol{h}$ nahrádzame ich strednými hodnotami $\overline{T}$ a $\overline{\boldsymbol{h}}$, nastáva podobná situácia, kedy po realizácii náhodných dát budú ohraničenia obsahujúce náhodné premenné s vysokou pravdepodobnosťou porušené. V princípe zo samotnej povahy úlohy vyplýva, že dodržanie ohraničení $T\boldsymbol{x} = \boldsymbol{h}$ nemožno zaručiť, pretože rozhodnutie o $\boldsymbol{x}$ sa vykonáva ešte predtým, než sú konkrétne hodnoty náhodných premenných vystupujúcich v týchto ohraničeniach známe. V praxi sa zvykne za nedodržanie stanovených obmedzení (kapacity, dopyty) platiť, respektíve sú s tým spojené určité náklady. To, aké vysoké náklady vzniknú, závisí práve od veľkosti rozdielu $\boldsymbol{h} - T\boldsymbol{x}$ po realizácii náhodných premenných. Práve táto miera nedodržania ohraničení vo vzťahu k pridruženým nákladom nie je zohľadnená pri riešení úlohy so zanedbaním stochastických ohraničení ani pri aproximatívnom riešení (1.5).

Možným východiskom je pridanie takzvanej kompenzácie. Zaveďme ohraničenie v tvare $W\boldsymbol{y} = \boldsymbol{h} - T\boldsymbol{x},\ \boldsymbol{y} \geq 0$, kde $W = (I, -I)$, pričom I je matica identity. Ďalej majme lineárnu účelovú funkciu $\boldsymbol{q}^\top \boldsymbol{y}$ predstavujúcu kompenzačné - penalizačné náklady. Dostávame tak úlohu:

$$Q(\boldsymbol{x}; T, \boldsymbol{h}) \triangleq \min_{\boldsymbol{y}} \left\{ \boldsymbol{q}^\top \boldsymbol{y} |\ W\boldsymbol{y} = \boldsymbol{h} - T\boldsymbol{x},\ \boldsymbol{y} \geq \boldsymbol{0} \right\} \tag{1.6}$$

Veľmi často je potom vhodné riešiť optimalizačnú úlohu, kde sa minimalizujú celkové priemerné náklady, čiže suma aktuálnych nákladov $\boldsymbol{c}^\top \boldsymbol{x}$ a očakávanej hodnoty kompenzačných nákladov. Takáto úloha má nasledovný tvar

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \boldsymbol{c}^\top \boldsymbol{x} \ + \ \mathbb{E}\, Q(\boldsymbol{x}; T, \boldsymbol{h}) \\ A\boldsymbol{x} &= \boldsymbol{b}, \\ \boldsymbol{x} &\geq \boldsymbol{0}. \end{aligned} \tag{1.7}$$

Úlohy typu (1.7) sa nazývajú *dvojstupňové stochastické lineárne úlohy (SLÚ) s pevnou kompenzáciou*. Dvojstupňové preto, lebo tu vystupujú dva stupne rozhodovania. *Prvý stupeň* označuje fázu, kedy robíme rozhodnutie o $\boldsymbol{x}$ bez toho, aby sme poznali ako sa vyvinie stav náhodných premenných. *Druhý stupeň* nastáva po realizácii náhodných premenných, kedy robíme rekurzné, resp. korekčné rozhodnutie o $\boldsymbol{y}$, ktorým máme možnosť dodatočne ovplyvniť výsledok. Pevná kompenzácia znamená, že matica W, nazývaná *kompenzačná matica*, v tomto prípade nie je stochastická. Poznamenávame, že matica W nemusí mať vo všeobecnosti tvar $W = (I, -I)$, ktorý prináleží špecifickému prípadu takzvanej jednoduchej kompenzácie. O dvojstupňových SLÚ s pevnou kompenzáciou budeme podrobne písať v ďalšej kapitole, kde uvedieme aj príslušnú teóriu.

V praxi sa stretávame aj s problémami, ktoré vo svojej prirodzenej forme vedú k rozhodovaniu vo viac než dvoch stupňoch. Vo všeobecnosti preto existujú aj *viac-stupňové SLÚ s pevnou kompenzáciou*. Podrobnému popisu a teórii k nim sa budeme venovať až neskôr v Kapitole 4.

2 Dvojstupňové stochastické lineárne úlohy (SLÚ) s pevnou kompenzáciou

Prvý typ stochastických úloh, ktorými sa v tejto práci budeme zaoberať, sú takzvané *dvojstupňové stochastické lineárne úlohy (SLÚ) s pevnou kompenzáciou*. Takéto úlohy patria medzi základné úlohy stochastického programovania.

Stochastická lineárna úloha je úloha lineárneho programovania, v ktorej môžu byť niektoré dáta, čiže matice alebo vektory, náhodné. Pojem kompenzácia tu úzko súvisí s tým, že úloha sa skladá z dvoch stupňov, ktorým prislúchajú aj dva typy rozhodnutí. V časti označovanej ako *prvý stupeň* robíme takzvané *prvostupňové rozhodnutia*. Tieto sú robené za neistoty, čiže pred realizáciou náhodných dát. Následne, po realizácii náhodných dát, máme možnosť spraviť v *druhom stupni* tzv. *druhostupňové* alebo *kompenzačné rozhodnutia* a tým ovplyvniť konečný výstup.

2.1 Formulácia úlohy a deterministický ekvivalent

Pod základnou dvojstupňovou stochastickou lineárnou úlohou s pevnou kompenzáciou budeme podľa [2] rozumieť úlohu v nasledovnom tvare:

$$\begin{aligned} \min_{\boldsymbol{x}} \; & z = \boldsymbol{c}^\top \boldsymbol{x} + \mathbb{E}_{\boldsymbol{\xi}}[\min_{\boldsymbol{y}} \boldsymbol{q}(\omega)^\top \boldsymbol{y}(\omega)] \\ & A\boldsymbol{x} = \boldsymbol{b}, \\ & T(\omega)\boldsymbol{x} + W\boldsymbol{y}(\omega) = \boldsymbol{h}(\omega), \\ & \boldsymbol{x} \geq \boldsymbol{0}, \; \boldsymbol{y}(\omega) \geq \boldsymbol{0}, \end{aligned} \tag{2.1}$$

kde $\boldsymbol{c} \in \mathbb{R}^{n_1}$, $\boldsymbol{b} \in \mathbb{R}^{m_1}$, $A \in \mathbb{R}^{m_1 \times n_1}$, $W \in \mathbb{R}^{m_2 \times n_2}$, ďalej $\forall \omega : T(\omega) \in \mathbb{R}^{m_2 \times n_1}$, $\boldsymbol{q}(\omega) \in \mathbb{R}^{n_2}$, $\boldsymbol{h}(\omega) \in \mathbb{R}^{m_2}$ a $\boldsymbol{\xi}(\omega)^\top = (\boldsymbol{q}(\omega)^\top, \boldsymbol{h}(\omega)^\top, T_1(\omega), ..., T_{m_2}(\omega))$, kde $T_i(\omega)$ predstavuje i-ty riadok matice $T(\omega)$. Počet prvkov vektora $\boldsymbol{\xi}(\omega)^\top$ je $N = m_2 + n_2 + (m_2 \times n_1)$. Na záver podotýkame, že ω označuje náhodné udalosti určujúce náhodný vektor ξ.

Matica W sa nazýva *kompenzačná matica* a matica $T(\omega)$ *technologická matica*. O matici W budeme predpokladať, že nie je stochastická, v opačnom prípade by totiž mohli nastať problémy s charakterizovaním množiny prípustných riešení[2]. $\mathbb{E}_{\boldsymbol{\xi}}$ predstavuje očakávanú hodnotu vzhľadom na náhodný vektor $\boldsymbol{\xi}$. Ďalej definujeme $\Xi \in \mathbb{R}^N$ ako nosič rozdelenia pravdepodobnosti pre $\boldsymbol{\xi}$.

Úlohu (2.1) možno sformulovať v odlišnom tvare, ktorý sa zvykne nazývať *deterministický ekvivalent*. Uvádzame ho v tvare podľa [2]:

$$\begin{aligned} \min_{\boldsymbol{x}} \; & z = \boldsymbol{c}^\top \mathbf{x} + \mathcal{Q}(\mathbf{x}) \\ & A\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{2.2}$$

[2]Práve nestochastická, čiže *pevná* kompenzačná matica W má za následok, že v názve úlohy vystupuje pojem *pevná kompenzácia*.

kde

$$\mathcal{Q}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\xi}} Q(\boldsymbol{x}, \boldsymbol{\xi}(\omega)), \tag{2.3}$$

pričom $Q(\boldsymbol{x}, \boldsymbol{\xi}(\omega))$ je optimálna hodnota účelovej funkcie úlohy druhého stupňa v tvare:

$$\begin{aligned} &\min_{\boldsymbol{y}} \boldsymbol{q}(\omega)^\top \boldsymbol{y} \\ &W\boldsymbol{y} = \boldsymbol{h}(\omega) - T(\omega)\boldsymbol{x}, \ \boldsymbol{y} \geq \boldsymbol{0}. \end{aligned} \tag{2.4}$$

Z predošlej formulácie je zrejmé, že ak poznáme funkciu $\mathcal{Q}(\boldsymbol{x})$, potom stochastická úloha (2.2) je vlastne úlohou deterministického nelineárneho programovania. Pri všetkých numerických algoritmoch na riešenie dvojstupňových SLÚ, ktoré neskôr uvedieme, budeme predpokladať vektor $\boldsymbol{\xi}$ s konečným rozdelením. Jeho nosič Ξ je potom daný ako

$$\Xi \triangleq \{\boldsymbol{\xi}_i : i = 1, ..., K\}, \tag{2.5}$$

kde K je počet možných realizácií $\boldsymbol{\xi}$. Jednotlivé realizácie $\boldsymbol{\xi}_i$ sa potom zvyknú označovať pojmom *scenáre*.

Poznamenávame, že v prípade stochastickej lineárnej úlohy s vektorom $\boldsymbol{\xi}$ majúcim konečné rozdelenie, je deterministický ekvivalent vždy lineárna úloha. Hlavnou motiváciou pre riešenie týchto stochastických úloh iným spôsobom, než je priama aplikácia metód lineárneho programovania na deterministický ekvivalent, býva jeho neúnosne veľký rozmer pri veľkom počte scenárov.

Proces optimalizácie v dvojstupňovej úlohe (2.2) – (2.4) možno vysvetliť nasledovne. Úloha (2.2) predstavuje prvý stupeň optimalizačného problému. Tu volíme rôzne prípustné hodnoty $\boldsymbol{x}$, následne pre každú z nich v druhom stupni (2.4) riešime pridružený optimalizačný problém pre všetky realizácie náhodného vektora $\boldsymbol{\xi}$. Z optimálnych hodnôt účelových funkcií druhého stupňa, ktoré získame, ďalej vypočítame strednú hodnotu vzhľadom na realizácie $\boldsymbol{\xi}$. Tak pre jednotlivé voľby $\boldsymbol{x}$ získame hodnotu $\mathcal{Q}(\boldsymbol{x})$. Zo všetkých prípustných $\boldsymbol{x}$ prvého stupňa vyberieme za optimálne to, pre ktoré bude hodnota účelovej funkcie prvého stupňa, teda $\boldsymbol{c}^\top \boldsymbol{x}$ v súčte s vypočítanou hodnotou $\mathcal{Q}(\boldsymbol{x})$, minimálna. Takto získané $\boldsymbol{x}$ nám dáva optimálne rozhodnutie do budúcnosti v prípade, že máme možnosť urobiť korekčné rozhodnutie $\boldsymbol{y}(\omega)$, no realizácia ω nie je v čase rozhodovania o $\boldsymbol{x}$ známa.

Vychádzajúc z [10], ak by pre nejaké $\boldsymbol{x}$ a $\boldsymbol{\xi}_i \in \Xi$ bola úloha druhého stupňa (2.4) neprípustná, definujeme $Q(\boldsymbol{x}, \boldsymbol{\xi}_i) = +\infty$. Tiež by mohol nastať prípad, že úloha druhého stupňa je zdola neohraničená a teda $Q(\boldsymbol{x}, \boldsymbol{\xi}_i) = -\infty$. V takom prípade sa jedná o patologický prípad, kedy pre určitú hodnotu $\boldsymbol{x}$ a realizáciu náhodného vektora $\boldsymbol{\xi}$ môže byť hodnota účelovej funkcie druhého stupňa neobmedzene znižovaná. Modely s takouto vlastnosťou by sa nemali uvažovať.

Ako sa ukáže neskôr, k úlohe (2.4) je užitočné uvažovať aj duálnu úlohu, ktorá má tvar:

$$\begin{aligned} &\max_{\boldsymbol{\pi}} \ (\boldsymbol{h}(\omega) - T(\omega)\boldsymbol{x})^\top \boldsymbol{\pi} \\ &W^\top \boldsymbol{\pi} \leq \boldsymbol{q}(\omega). \end{aligned} \tag{2.6}$$

Z teórie duality lineárneho programovania uvedenej napríklad v [9] je známe, že ak má jedna z úloh (2.4), (2.6) optimálne riešenie, potom má optimálne riešenie aj druhá, pričom hodnoty účelových funkcií oboch úloh sa v optime rovnajú. Ak nastáva v niektorej z úloh neohraničenosť, potom druhá je neprípustná.

2.2 Množiny prípustnosti

Predpokladajme najprv, že vektor $\boldsymbol{\xi}$ má konečné rozdelenie, resp. $\boldsymbol{\xi} \in \Xi$, kde nosič Ξ je daný vzťahom (2.5). V tomto prípade budeme výnimočne uvažovať maticu W, ktorá môže byť stochastická. Uvedený výsledok je totiž platný aj pre takýto prípad, preto ho nebudeme zbytočne zužovať na prípad pevnej kompenzácie.

Podľa [2], nech $K_1 \triangleq \{\boldsymbol{x} | A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ je množina definovaná ohraničeniami úlohy prvého stupňa (2.2), ktorá nezávisí od realizácie náhodného vektora $\boldsymbol{\xi}$, a nech $K_2 \triangleq \{\boldsymbol{x} | \mathcal{Q}(\boldsymbol{x}) < \infty\}$ je množina prípustných riešení úlohy druhého stupňa (2.4). Je užitočné zaviesť tiež označenie pre ich prienik $K \triangleq K_1 \cap K_2$. Ďalej definujeme $K_2(\boldsymbol{\xi}) \triangleq \{\boldsymbol{x} | Q(\boldsymbol{x}, \boldsymbol{\xi}) < \infty\}$ ako elementárne množiny prípustnosti. Potom

$$K_2^P \triangleq \{\boldsymbol{x} |\ \forall \boldsymbol{\xi} \in \Xi\ \ \exists \boldsymbol{y} \geq \boldsymbol{0} : W\boldsymbol{y} = \boldsymbol{h} - T\boldsymbol{x}\} = \bigcap_{\boldsymbol{\xi} \in \Xi} K_2(\boldsymbol{\xi}) \tag{2.7}$$

bude predstavovať takú množinu prvostupňových rozhodnutí $\boldsymbol{x}$, pre ktorej každý prvok vieme nájsť nejaké prípustné druhostupňové rozhodnutie $\boldsymbol{y}$ pri ľubovoľnej realizácii náhodnej premennej $\boldsymbol{\xi}$.

O ďalších dôležitých vlastnostiach množín K_2^P a K_2 hovorí nasledujúca veta.

Veta 2.1. ([2], str. 86)

(a) Pre všetky realizácie $\boldsymbol{\xi}$ je elementárna množina prípustnosti $K_2(\boldsymbol{\xi})$ uzavretý konvexný polyéder, preto množina K_2^P je uzavretá a konvexná.

(b) Ak množina Ξ je konečná, potom K_2^P je tiež polyedrická a je totožná s K_2.

Dôkaz je uvedený v [2].

Z uvedenej vety plynie, že za daných predpokladov je množina prípustných riešení druhého stupňa K_2 totožná s K_2^P a teda podľa (2.7) je vyjadrená prienikom množín $K_2(\boldsymbol{\xi})$.

V práci sa budeme zaoberať numerickými algoritmami, pričom budeme riešiť úlohy s konečným rozdelením vektora $\boldsymbol{\xi}$. Napriek tomu, pre doplnenie teórie bude užitočné uviesť aj niektoré vlastnosti pre spojitý náhodný vektor $\boldsymbol{\xi}$. Pre tento prípad by mohla nastať situácia, kedy $K_2^P \subset K_2$ a teda tieto množiny by nemuseli byť totožné, čo predstavuje problém. Ako sa uvádza v [2], pre nestochastickú maticu W takýto prípad nastáva len zriedka. Z definície množín K_2 a K_2^P možno nahliadnuť, že pokiaľ by hodnotová funkcia $Q(\boldsymbol{x}, \boldsymbol{\xi})$ bola ohraničená s pravdepodobnosťou jedna, pričom jej očakávaná hodnota cez $\boldsymbol{\xi}$, čiže $\mathcal{Q}(\boldsymbol{x})$ by bola neohraničená, boli by

množiny K_2 a K_2^P rôzne. Aby sa tomuto problému predišlo, zavádza sa dodatočný predpoklad na vektor $\boldsymbol{\xi}$, obsiahnutý v nasledovnom tvrdení.

Tvrdenie 2.1. ([2], str. 87) *Nech $\boldsymbol{\xi}$ má konečné druhé momenty, potom ak $P(\omega|Q(\boldsymbol{x},\boldsymbol{\xi}) < \infty) = 1$, tak $\mathcal{Q}(\boldsymbol{x}) < \infty$.*

Všeobecný dôkaz možno nájsť v [14].

Pre úlohu s nestochastickou maticou W možno vďaka platnosti Tvrdenia 2.1 sformulovať dve dôležité vety. Prvá z nich zaručuje, že pri daných predpokladoch budú množiny K_2 a K_2^P totožné.

Veta 2.2. ([2], str. 88) *V dvojstupňovej úlohe stochastického programovania s pevnou kompenzáciou, kde $\boldsymbol{\xi}$ má konečné druhé momenty, sú množiny K_2 a K_2^P totožné.*

Dôkaz je uvedený v [2].

V skutočnosti existujú aj všeobecnejšie postačujúce podmienky zaručujúce totožnosť K_2 a K_2^P, než dáva Veta 2.2. V rámci tejto práce ich však nebudeme uvádzať. Nasledovná veta uvádza užitočné výsledky ohľadom vlastností množiny K_2 ($\equiv K_2^P$).

Veta 2.3. ([2], str. 88–89) *Ak matica W je nestochastická a $\boldsymbol{\xi}$ má konečné druhé momenty, platí:*

(a) K_2 je uzavretá, konvexná množina.

(b) Ak matica T je nestochastická, množina K_2 je polyedrická.

(c) Nech Ξ_T je nosič rozdelenia T. Ak $\boldsymbol{h}(\omega)$ a $T(\omega)$ sú nezávislé a Ξ_T je polyedrická množina, potom množina K_2 je polyedrická.

Dôkaz je uvedený v [2].

Všimnime si, že Veta 2.3 - časť (a) nám dáva pre množinu K_2^P analogický výsledok s Vetou 2.1 - časť (a), pričom tu platí zároveň pre K_2. Ďalej je zaujímavé, že pri spojitom vektore $\boldsymbol{\xi}$ dostávame polyedrickú vlastnosť K_2 ($\equiv K_2^P$) až po zavedení ďalšieho predpokladu na maticu T, prípadne na vzťah vektora $\boldsymbol{h}$ a matice T. Pre diskrétny prípad sme túto vlastnosť mali bez ohľadu na T a $\boldsymbol{h}$.

2.3 Špecifické typy stochastických lineárnych úloh s pevnou kompenzáciou

Vo všeobecnosti sa rozlišujú tri špeciálne prípady spadajúce pod dvojstupňové stochastické lineárne úlohy s pevnou kompenzáciou typu (2.1).

Prvý z nich sa vyznačuje tým, že jeho množina prípustných riešení prvého stupňa je obsiahnutá v množine prípustných riešení druhého stupňa, teda $K_1 \subset K_2$. Vtedy hovoríme, že v úlohe je *relatívne úplná kompenzácia*. Nakoľko je vo všeobecnosti problematické overovať vzťah množín K_1 a K_2, výhody ktoré úlohy tohto typu poskytujú, sa v praxi nezvyknú príliš využívať. Preto sa v reálnych úlohách zvykne predpokladať skôr tzv. *úplná kompenzácia*, ktorá je charakterizovaná pomocou vlastností matice W. Konkrétne tu platí:

$$\forall \boldsymbol{t} \in \mathbb{R}^{m_2}\ \exists \boldsymbol{y} \geq \boldsymbol{0} :\ W\boldsymbol{y} = \boldsymbol{t}. \tag{2.8}$$

Je zrejmé, že nakoľko druhostupňové ohraničenia úlohy (2.1) majú tvar $T(\omega)\boldsymbol{x} + W\boldsymbol{y}(\omega) = \boldsymbol{h}(\omega),\ \boldsymbol{y}(\omega) \geq \boldsymbol{0}$, dostávame tu pre ľubovoľný vektor $\boldsymbol{x}$ prípustné riešenie $\boldsymbol{y} \geq \boldsymbol{0}$, čo je z praktického hľadiska veľmi výhodné. Pre identifikáciu úloh s takouto vlastnosťou slúži nasledovná lema.

Lema 2.1. ([7], str. 201) *Matica $W \in \mathbb{R}^{m_2 \times n_2}$ spĺňa podmienku úplnej kompenzácie (2.8) práve vtedy, keď má hodnosť m_2 a pre ľubovoľnú množinu $\{W_{i_1}, W_{i_2}, ..., W_{i_{m_2}}\}$ lineárne nezávislých stĺpcov W, sú lineárne ohraničenia*

$$W\boldsymbol{y} = 0 \tag{2.9}$$

$$y_{i_k} \geq 1,\ \ k = 1, ..., m_2, \tag{2.10}$$

$$\boldsymbol{y} \geq 0 \tag{2.11}$$

prípustné.

Pre overovanie podmienok (2.9)–(2.11) možno použiť napríklad algoritmus, ktorý uvádzajú autori v [7] na s. 26.

Napokon rozlišujeme tzv. úlohy s *jednoduchou kompenzáciou*. Jedná sa o špeciálny prípad úplnej kompenzácie, kde $W = (I, -I)$, $T(\omega) \equiv T$, $\boldsymbol{q}(\omega) \equiv (\boldsymbol{q}^+, \boldsymbol{q}^-)$, $\boldsymbol{\xi}(\omega) = \boldsymbol{h}(\omega)$. Potom funkcia $Q(\boldsymbol{x}, \boldsymbol{\xi}(\omega))$ predstavuje optimálnu hodnotu účelovej funkcie úlohy v tvare:

$$\begin{aligned} \min_{\boldsymbol{y}}\ & {\boldsymbol{q}^+}^\top \boldsymbol{y}^+ + {\boldsymbol{q}^-}^\top \boldsymbol{y}^- \\ & I\boldsymbol{y}^+ - I\boldsymbol{y}^- = \boldsymbol{h}(\omega) - T\boldsymbol{x} \\ & \boldsymbol{y}^+, \quad \boldsymbol{y}^- \geq \boldsymbol{0}. \end{aligned} \tag{2.12}$$

Duálna úloha k (2.12) má tvar:

$$\begin{aligned} \max_{\boldsymbol{p}}\ & (\boldsymbol{h}(\omega) - T\boldsymbol{x})^\top \boldsymbol{p} \\ & \boldsymbol{p} \ \leq\ \ \boldsymbol{q}^+ \\ & \boldsymbol{p} \ \geq -\boldsymbol{q}^- \end{aligned} \tag{2.13}$$

Úloha (2.12) je vždy prípustná, pričom riešenie existuje práve vtedy, keď je duálna úloha (2.13) prípustná. Vtedy zjavne musí platiť $\boldsymbol{q}^+ + \boldsymbol{q}^- \geq \boldsymbol{0}$.

2.4 Hodnotová funkcia druhého stupňa

Vlastnosti funkcií $Q(\boldsymbol{x}, \boldsymbol{\xi})$, ako aj tzv. hodnotovej funkcie druhého stupňa $\mathcal{Q}(\boldsymbol{x})$, ktorá predstavuje ich očakávanú hodnotu vzhľadom na $\boldsymbol{\xi}$, sú kľúčové z hľadiska konštrukcie algoritmov pre riešenie úlohy (2.1). Najprv uvedieme vlastnosti pre $Q(\boldsymbol{x}, \boldsymbol{\xi})$ za predpokladu $Q(\boldsymbol{x}, \boldsymbol{\xi}) \neq -\infty$.

Veta 2.4. ([2], str. 89) *Pre stochastickú úlohu s pevnou kompenzáciou je* $Q(\boldsymbol{x}, \boldsymbol{\xi})$

(a) po častiach lineárna konvexná funkcia v $(\boldsymbol{h}, T)$*;*

(b) po častiach lineárna konkávna funkcia v $\boldsymbol{q}$*;*

(c) po častiach lineárna konvexná funkcia v $\boldsymbol{x}$ *pre všetky* $\boldsymbol{x} \in K = K_1 \cap K_2$*.*

Dôkaz je uvedený v [2].

Povaha $\mathcal{Q}(\boldsymbol{x})$ je obzvlášť dôležitá pre konštrukciu efektívnych algoritmov. $\mathcal{Q}(\boldsymbol{x})$ sa nachádza v účelovej funkcii prvostupňovej optimalizačnej úlohy (2.2), čo vzhľadom k tomu, že na jej vyčíslenie je potrebné riešiť optimalizačné úlohy druhého stupňa pre všetky realizácie $\boldsymbol{\xi}$, predstavuje výpočtovo náročný problém. Tento problém, ako sa ukáže, možno pri určitých predpokladoch zjednodušiť. O vlastnostiach $\mathcal{Q}(\boldsymbol{x})$ hovorí nasledovná veta.

Veta 2.5. ([2], str. 90–91) *Pre stochastickú úlohu s pevnou kompenzáciou, v ktorej* $\boldsymbol{\xi}$ *má konečné druhé momenty, platí:*

(a) $\mathcal{Q}(\boldsymbol{x})$ *je Lipschitzovská, konvexná a konečná funkcia na* K_2*.*

(b) Ak $\boldsymbol{\xi}$ *má konečnú distribúciu,* $\mathcal{Q}(\boldsymbol{x})$ *je po častiach lineárna funkcia.*

(c) Ak $F(\boldsymbol{\xi})$ *je absolútne spojitá distribučná funkcia,* $\mathcal{Q}(\boldsymbol{x})$ *je diferencovateľná na* K_2*.*

Dôkaz v [2].

Z uvedených vlastností $\mathcal{Q}(\boldsymbol{x})$ a Viet 2.1, 2.2 plynie viacero dôležitých záverov. Pokiaľ náhodné premenné predstavujúce vektor $\boldsymbol{\xi}$ sú diskrétne a majú konečnú distribúciu, z Vety 2.1 - časť (b) máme, že K_2 a K_2^P sú totožné polyedrické množiny, ďalej z Vety 2.5 dostávame, že $\mathcal{Q}(\boldsymbol{x})$ je po častiach lineárna funkcia, konvexná na K_2. V takom prípade vieme úlohu (2.1) riešiť dekompozičnými postupmi, z ktorých odvodené algoritmy sú hlavnou náplňou tejto práce.

V prípade, že $\boldsymbol{\xi}$ nemá konečnú distribúciu, dostávame pre väčšinu typických rozdelení konečné druhé momenty a absolútne spojitú distribučnú funkciu. Potom podľa Vety 2.2 množiny K_2 a K_2^P splývajú a z Vety 2.5 máme, že $\mathcal{Q}(\boldsymbol{x})$ je konvexná a diferencovateľná funkcia. Na riešenie úlohy (2.1) potom možno, ako je uvedené v [2], aplikovať techniky nelineárneho programovania. V niektorých špecifických prípadoch možno vyjadriť $\mathcal{Q}(\boldsymbol{x})$ analyticky, avšak vo všeobecnosti by bolo treba $\mathcal{Q}(\boldsymbol{x})$

rátať numerickou integráciou $Q(\boldsymbol{x}, \boldsymbol{\xi})$, pričom takýto prístup je efektívny iba pre $\boldsymbol{\xi}$ malých rozmerov. Iný, často jediný efektívne použiteľný prístup spočíva v diskretizácii vektora $\boldsymbol{\xi}$ a pristupovaní k úlohe ako v prípade konečnej distribúcie. Takto dostávame aproximatívne riešenie pôvodnej úlohy s predpokladom, že postupným zjemňovaním delenia bude toto konvergovať k presnému riešeniu.

2.5 Podmienky optimality

V tejto časti uvedieme základné vety, ktoré dávajú výsledky ohľadom existencie a charakteristických vlastností riešenia úlohy (2.2)–(2.4).

Postačujúcu podmienku pre existenciu riešenia (2.2)–(2.4) nám dáva nasledovná veta. Označenie rc tu pri aplikácii na funkciu $\mathcal{Q}$ predstavuje zostupovú hodnotu (angl. recession value), čiže $\sup_{\boldsymbol{x} \in dom(\mathcal{Q})}(\mathcal{Q}(\boldsymbol{x}+\boldsymbol{v}) - \mathcal{Q}(\boldsymbol{x}))$, kde $dom(\mathcal{Q})$ predstavuje časť definičného oboru $\mathcal{Q}$, na ktorej $\mathcal{Q}$ nadobúda konečné hodnoty.

Veta 2.6. ([2], str. 94) *Predpokladajme, že náhodné prvky vektora $\boldsymbol{\xi}$ majú konečné druhé momenty a platí:*

(a) množina K je ohraničená; alebo

(b) kompenzačná funkcia $\mathcal{Q}$ je eventuelne lineárna vo všetkých zostupných smeroch v K, čiže $\mathcal{Q}(\boldsymbol{x} + \lambda\boldsymbol{v}) = \mathcal{Q}(\boldsymbol{x} + \overline{\lambda}\boldsymbol{v}) + (\lambda - \overline{\lambda})rc\mathcal{Q}(\boldsymbol{v})$ pre nejaké $\overline{\lambda} \geq 0$ (závislé od $\boldsymbol{x}$), $\forall \lambda \geq \overline{\lambda}$ a nejakú konštantnú zostupovú hodnotu $zk\mathcal{Q}(\boldsymbol{v})$, pre všetky $\boldsymbol{v}$ také, že $\boldsymbol{x} + \lambda\boldsymbol{v} \in K$ pre všetky $\boldsymbol{x} \in K$ a $\lambda \geq 0$.

Dôkaz vety je uvedený v [2].

Za predpokladu existencie optimálneho riešenia (2.2)–(2.4) nám nasledovná veta charakterizuje vlastnosti tohto optimálneho riešenia cez Karash-Kuhn-Tuckerove podmienky.

Veta 2.7. ([2], str. 95) *Predpokladajme, že (2.2) má konečnú optimálnu hodnotu účelovej funkcie. Riešenie $\boldsymbol{x}^* \in K_1$, je optimálne pre (2.2) práve vtedy keď existuje nejaké $\boldsymbol{\lambda}^* \in \mathbb{R}^{m_1}$, $\boldsymbol{\mu}^* \in \mathbb{R}^{n_1}_+$, $\boldsymbol{\mu}^{*\top}\boldsymbol{x}^* = 0$, také že* $-\boldsymbol{c} + A^\top\boldsymbol{\lambda}^* + \boldsymbol{\mu}^* \in \partial\mathcal{Q}(\boldsymbol{x}^*)$.

Dôkaz je uvedený v [2].

Touto vetou ukončíme teóriu dvojstupňových SLÚ a v ďalšom prejdeme na príslušné algoritmy.

3 Algoritmy pre dvojstupňové SLÚ s pevnou kompenzáciou

V tejto kapitole najprv predstavíme základný algoritmus *L-Shaped metódy* používaný na riešenie dvojstupňových SLÚ s pevnou kompenzáciou. Následne uvedieme jeho viacrezovú a adaptívnu modifikáciu.

3.1 L-Shaped metóda

Uvádzame schému algoritmu L-Shaped metódy podľa [2] pre riešenie dvojstupňových lineárnych úloh s pevnou kompenzáciou typu (2.1), ktorá zodpovedá štandardu z literatúry, pričom býva označovaná tiež ako Van Slyke - Wetsova [1969] L-Shaped metóda.

3.1.1 Algoritmus

Krok 0: Inicializácia.
Polož $r = s = \nu = 0$ (ν predstavuje index makroiterácie).

Krok 1: Riešenie hlavnej úlohy.
Polož $\nu = \nu + 1$. Rieš úlohu:

$$\min_{(\boldsymbol{x},\theta)} z = \boldsymbol{c}^\top \boldsymbol{x} + \theta \tag{3.1}$$

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{3.2}$$

$$\boldsymbol{G}_l \boldsymbol{x} \geq g_l,\ l = 1, ..., r, \tag{3.3}$$

$$\boldsymbol{E}_l \boldsymbol{x} + \theta \geq e_l,\ l = 1, ..., s, \tag{3.4}$$

$$\boldsymbol{x} \geq \boldsymbol{0}, \theta \in \mathbb{R}. \tag{3.5}$$

Dostávame optimálne riešenie $(\boldsymbol{x}^\nu, \theta^\nu)$. Ak ešte neboli pridané *rezy optimality* (3.4), θ^ν sa položí rovné $-\infty$ a neuvažuje sa pri výpočte $\boldsymbol{x}^\nu$.

Krok 2: Kontrola prípustnosti v úlohách druhého stupňa, pridávanie rezov prípustnosti (3.3).
Pre všetky scenáre $k = 1, ..., K$ rieš úlohu:

$$\min_{(\boldsymbol{v}^+,\boldsymbol{v}^-)} w' = \boldsymbol{e}^\top \boldsymbol{v}^+ + \boldsymbol{e}^\top \boldsymbol{v}^- \tag{3.6}$$

$$W\boldsymbol{y} + I\boldsymbol{v}^+ - I\boldsymbol{v}^- = \boldsymbol{h}_k - T_k \boldsymbol{x}^\nu \tag{3.7}$$

$$\boldsymbol{y} \geq \boldsymbol{0},\ \boldsymbol{v}^+ \geq \boldsymbol{0},\ \boldsymbol{v}^- \geq \boldsymbol{0}, \tag{3.8}$$

kde $\boldsymbol{e} = (1, ..., 1)^\top$. Ak pre nejaké k nastane $w' > 0$, definujeme:

$$\boldsymbol{G}_{r+1} = (\boldsymbol{\sigma}^\nu)^\top T_k \tag{3.9}$$

$$g_{r+1} = (\boldsymbol{\sigma}^\nu)^\top \boldsymbol{h}_k, \tag{3.10}$$

kde $\boldsymbol{\sigma}^\nu$ je vektor hodnôt duálnych premenných úlohy (3.6)–(3.8). Z (3.9) a (3.10) generuj ohraničenie typu (3.3) - rez prípustnosti, polož $r = r+1$, pridaj ohraničenie do množiny (3.3), choď na Krok1. Inak ($\forall k : w' = 0$) pokračuj.

Krok 3: Riešenie úloh druhého stupňa, pridávanie rezov optimality (3.4). Pre všetky scenáre $k = 1, ..., K$ rieš úlohu druhého stupňa:

$$\begin{aligned} \min_{\boldsymbol{y}} \; & w = \boldsymbol{q}_k^\top \boldsymbol{y} \\ & W\boldsymbol{y} = \boldsymbol{h}_k - T_k \boldsymbol{x}^\nu, \\ & \boldsymbol{y} \geq \boldsymbol{0}. \end{aligned} \tag{3.11}$$

Pre optimálne riešenie (3.11) v k-tej iterácii zober hodnotu jeho duálnej premennej $\boldsymbol{\pi}_k^\nu$. Vypočítaj:

$$\boldsymbol{E}_{s+1} = \sum_{k=1}^{K} \boldsymbol{p}_k (\boldsymbol{\pi}_k^\nu)^\top T_k \tag{3.12}$$

$$e_{s+1} = \sum_{k=1}^{K} \boldsymbol{p}_k (\boldsymbol{\pi}_k^\nu)^\top \boldsymbol{h}_k \tag{3.13}$$

Polož $w^\nu = e_{s+1} - \boldsymbol{E}_{s+1}\boldsymbol{x}^\nu$. Ak $\theta^\nu \geq w^\nu$, máme optimálne riešenie $\boldsymbol{x}^\nu$, koniec. Inak $s = s + 1$, generuj rez optimality a pridaj ho do množiny ohraničení (3.4). Vráť sa na Krok 1.

Sekvenciu Krok 1 $\rightarrow$ Krok 2 $\rightarrow$ (Krok 3) $\rightarrow$ Krok 1 označujeme pojmom *makroiterácia*, pričom jednotlivým makroiteráciám prislúcha index ν. Riešenie úlohy (3.6)–(3.8) pre jednotlivé scenáre $k = 1, ..., K$ v Kroku 2 spolu s výpočtom rezov prípustnosti (3.9)–(3.10) a taktiež riešenie druhostupňovej úlohy (3.11) pre jednotlivé scenáre v Kroku 3 spolu s výpočtom rezov optimality (3.12)–(3.13) označujeme pojmom *mikroiterácie*. Jednotlivé mikroiterácie označujeme rovnako ako scenáre indexom k.

O konvergencii uvedeného algoritmu hovorí nasledovná veta, ktorej dôkaz možno nájsť v [2], s. 159-162.

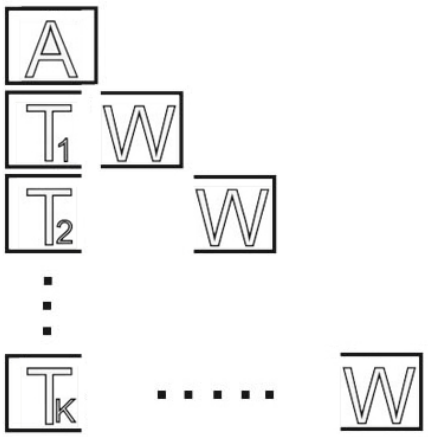
Veta 3.1. ([2], s. 162) *Nech $\boldsymbol{\xi}$ je konečná náhodná premenná. Potom algoritmus L-Shaped metódy konverguje v konečnom počte iterácií k optimálnemu riešeniu, ak toto existuje, v opačnom prípade dokazuje neprípustnosť (2.1).*

3.1.2 Princíp fungovania algoritmu

Nutným predpokladom pre použiteľnosť uvedeného algoritmu je konečné rozdelenie vektora $\boldsymbol{\xi}$. Vtedy vieme napísať deterministický ekvivalent (2.1) v nasledovnej extenzívnej forme:

$$\begin{aligned} \min_{\boldsymbol{x},\{\boldsymbol{y}_k\}} \; & \boldsymbol{c}^\top \boldsymbol{x} + \sum_{k=1}^{K} p_k \boldsymbol{q}_k{}^\top \boldsymbol{y}_k \\ & A\boldsymbol{x} = \boldsymbol{b}, \\ & T_k \boldsymbol{x} + W \boldsymbol{y}_k = \boldsymbol{h}_k, \;\; k = 1, \ldots K, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \;\; \boldsymbol{y}_k \geq \boldsymbol{0}, \;\; k = 1, \ldots K. \end{aligned} \tag{3.14}$$

Indexom k označujeme jednotlivé scenáre. Je zrejmé, že táto extenzívna forma predstavuje úlohu lineárneho programovania, ktorú možno riešiť rozličnými triedami metód. Patria sem dobre známe pivotné metódy, konkrétne simplexová, či duálna simplexová metóda a tiež metódy vnútorného bodu, popísané napr. v [9], [12]. V prípade malého rozmeru extenzívnej formy, resp. pri malom počte možných realizácií náhodného vektora $\boldsymbol{\xi}$, by bolo použitie takýchto metód efektívne. Na druhej strane, s rastúcim počtom možných realizácií náhodného vektora môže veľkosť extenzívnej formy narásť do obrovských rozmerov, preto je v tomto prípade rozumné využiť špeciálnu blokovú štruktúru úlohy (3.14), ktorá je zachytená na Obr. 1. Na



Obr. 1: Bloková štruktúra extenzívnej formy (3.14).

úlohy so špecifickou blokovou štruktúrou možno aplikovať rôzne, primárne či duálne dekompozičné metódy. Medzi takéto metódy patrí aj uvedený algoritmus L-Shaped metódy, ktorý za svoj názov vďačí práve uvedenej blokovej štruktúre pripomínajúcej písmeno 'L'.

Hlavná myšlienka algoritmu spočíva v postupnej aproximácii nelineárneho člena $\mathcal{Q}(\boldsymbol{x})$ v (2.2) pomocou linearizácie dotykovými nadrovinami. Výpočet $\mathcal{Q}(\boldsymbol{x})$ si vyžaduje vyriešiť optimalizačné úlohy druhého stupňa (2.4) pre všetky realizácie ω, čo

je z výpočtového hľadiska náročné. Preto sa v Kroku 1 rieši *hlavná úloha* ('master problem'), v ktorej sa postupne buduje aproximácia nelineárneho člena pridávaním rezov optimality $\boldsymbol{E}_l\boldsymbol{x} + \theta \geq e_l$. Úlohy druhého stupňa (2.4) sa riešia v Kroku 3 iba ako podproblém hlavnej úlohy.

Vznik rezov optimality a funkciu hlavnej úlohy si teraz po predstavení základnej idey ozrejmíme podrobnejšie. Úlohu (2.2) vieme prepísať do tvaru

$$\begin{aligned} \min_{(\boldsymbol{x},\theta)} \ & \boldsymbol{c}^\top \boldsymbol{x} + \theta \\ & A\boldsymbol{x} = \boldsymbol{b}, \\ & \mathcal{Q}(\boldsymbol{x}) \leq \theta, \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{aligned} \tag{3.15}$$

Funkciu $\mathcal{Q}(\boldsymbol{x})$ vieme nahradiť jej linearizáciou v okolí nejakého bodu $\hat{\boldsymbol{x}}$. Teda $\mathcal{Q}(\boldsymbol{x}) \approx \boldsymbol{u}^\top \boldsymbol{x} + \alpha$, kde $\boldsymbol{u}$ je subgradient $\mathcal{Q}(\boldsymbol{x})$ vypočítaný v bode $\hat{\boldsymbol{x}}$ a α predstavuje posun $\mathcal{Q}(\hat{\boldsymbol{x}}) - \boldsymbol{u}^\top \hat{\boldsymbol{x}}$. Ak aproximujeme funkciu $\mathcal{Q}(\boldsymbol{x})$ pomocou r nadrovín, dostávame úlohu v tvare

$$\begin{aligned} \min_{(\boldsymbol{x},\theta)} \ & \boldsymbol{c}^\top \boldsymbol{x} + \theta \\ & A\boldsymbol{x} = \boldsymbol{b}, && (3.16) \\ & \boldsymbol{u}_l^\top \boldsymbol{x} + \alpha_l \leq \theta, \ l = 1, ..., r, && (3.17) \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned}$$

ktorú iteratívne riešime, pridávajúc v každej iterácii ďalšiu nadrovinu (3.17). Takto sa postupne zlepšuje aproximácia funkcie $\mathcal{Q}(\boldsymbol{x})$. Po iteráciu ν tak získame postupnosť $\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, .., \hat{\boldsymbol{x}}_\nu$ a k nej pridruženú postupnosť subgradientov $\boldsymbol{u}_1 \in \partial\mathcal{Q}(\hat{\boldsymbol{x}}_1), \boldsymbol{u}_2 \in \partial\mathcal{Q}(\hat{\boldsymbol{x}}_2), .., \boldsymbol{u}_\nu \in \partial\mathcal{Q}(\hat{\boldsymbol{x}}_\nu)$, obdobne dostávame postupnosť posunov $\alpha_1, \alpha_2, .., \alpha_\nu$. Teraz sa pozrieme na výpočet subgradientu $\boldsymbol{u}$ a posunu α v lineárnej aproximácii $\boldsymbol{u}^\top \boldsymbol{x} + \alpha \approx \mathcal{Q}(\boldsymbol{x})$.

Označme $\boldsymbol{y}_k^*(\boldsymbol{x})$ a $\boldsymbol{\pi}_k^*(\boldsymbol{x})$ parametrické optimálne riešenia primárnej a duálnej úlohy druhého stupňa (2.4) a (2.6) v k-tom scenári, kde $k = 1, ..., K$. Z teórie duality (slabá veta o dualite) pre účelové funkcie primárnej a duálnej úlohy druhého stupňa platí[3]

$$\boldsymbol{q}(\omega_k)^\top \boldsymbol{y}_k{}^*(\boldsymbol{x}) = Q(\boldsymbol{x}, \omega_k) \geq (\boldsymbol{h}_k - T_k\boldsymbol{x})^\top \boldsymbol{\pi}_k \quad \forall \boldsymbol{x}, \forall \boldsymbol{\pi}_k, \tag{3.18}$$

pričom v optime

$$\boldsymbol{q}(\omega_k)^\top \boldsymbol{y}_k{}^*(\boldsymbol{x}) = Q(\boldsymbol{x}, \omega_k) = (\boldsymbol{h}_k - T_k\boldsymbol{x})^\top \boldsymbol{\pi}_k^*(\boldsymbol{x}) \quad \forall \boldsymbol{x}, \tag{3.19}$$

kde $\boldsymbol{\pi}_k$ je duálna premenná z úlohy (2.6) prislúchajúca k scenáru k. Dosadením konkrétneho $\hat{\boldsymbol{x}}$ do (3.19) vieme ľahko dostať

$$\mathcal{Q}(\hat{\boldsymbol{x}}) = \sum_{k=1}^{K} p_k Q(\hat{\boldsymbol{x}}, \omega_k) = \sum_{k=1}^{K} p_k (\boldsymbol{h}_k - T_k\hat{\boldsymbol{x}})^\top \hat{\boldsymbol{\pi}}_k, \tag{3.20}$$

[3] $Q(\boldsymbol{x}, \omega_k)$ už je optimálna hodnota účelovej funkcie pre parametrickú úlohu v $\boldsymbol{x}$, optimalizuje sa cez y_k.

kde $\hat{\boldsymbol{\pi}}_k = \boldsymbol{\pi}_k^*(\hat{\boldsymbol{x}})$. Po dosadení $\hat{\boldsymbol{\pi}}_k$ do (3.18) pre $\mathcal{Q}(\boldsymbol{x})$ máme

$$\mathcal{Q}(\boldsymbol{x}) = \sum_{k=1}^{K} p_k Q(\boldsymbol{x}, \omega_k) \geq \sum_{k=1}^{K} p_k (\boldsymbol{h}_k - T_k \boldsymbol{x})^\top \hat{\boldsymbol{\pi}}_k \quad \forall \boldsymbol{x}. \tag{3.21}$$

Zo vzťahov (3.20) a (3.21) je zrejmé, že výraz $\sum_{k=1}^{K} p_k (\boldsymbol{h}_k - T_k \boldsymbol{x})^\top \hat{\boldsymbol{\pi}}_k$, ktorý je lineárny v $\boldsymbol{x}$, predstavuje nutne opornú nadrovinu funkcie $\mathcal{Q}(\boldsymbol{x})$ s dotykom v bode $\hat{\boldsymbol{x}}$.

Ak teda chceme dostať lineárnu aproximáciu $\mathcal{Q}(\boldsymbol{x}) \approx \boldsymbol{u}^\top \boldsymbol{x} + \alpha$, stačí položiť

$$\boldsymbol{u} = -\sum_{k=1}^{K} p_k T_k^\top \hat{\boldsymbol{\pi}}_k$$

$$\alpha = \sum_{k=1}^{K} p_k \boldsymbol{h}_k{}^\top \hat{\boldsymbol{\pi}}_k.$$

V jednotlivých iteráciách sa teda takto v Kroku 3 algoritmu L-shaped metódy vypočítajú koeficienty (3.12), (3.13) pre rezy optimality (3.4).

Ostáva objasniť úlohu a výpočet rezov prípustnosti (3.3) a tiež ukončovaciu podmienku algoritmu. Ak v Kroku 2 pre nejaké $\hat{\boldsymbol{x}}$ a k nastane $w' > 0$, $\hat{\boldsymbol{x}}$ je neprípustné. Pre optimálnu hodnotu účelovej funkcie duálnej úlohy k (3.6)–(3.8) s premennou $\boldsymbol{\sigma}$ máme $\hat{\boldsymbol{\sigma}}^\top (\boldsymbol{h}_k - T_k \hat{\boldsymbol{x}}) > 0$. Z duálnej úlohy tiež dostávame $\hat{\boldsymbol{\sigma}}^\top W \leq 0$. Vieme, že pre každé prípustné $\boldsymbol{x}$ existuje nejaké $\boldsymbol{y} \geq \boldsymbol{0}$, pre ktoré platí $W\boldsymbol{y} = \boldsymbol{h} - T\boldsymbol{x}$. Využitím Farkasovej lemy podľa [8] dostávame:

$$\exists \boldsymbol{y} \geq \boldsymbol{0} : W\boldsymbol{y} = \boldsymbol{h} - T\hat{\boldsymbol{x}} \Leftrightarrow \left(\forall \boldsymbol{\sigma} : W^\top \boldsymbol{\sigma} \leq 0 \Rightarrow \boldsymbol{\sigma}^\top (\boldsymbol{h} - T\hat{\boldsymbol{x}}) \leq 0\right). \tag{3.22}$$

Vidno, že pridaním ohraničenia $\hat{\boldsymbol{\sigma}}^\top (\boldsymbol{h}_k - T_k \hat{\boldsymbol{x}}) \leq 0$ vylúčime $\boldsymbol{x}$, ktoré by viedli k neprípustnosti v úlohe druhého stupňa (2.4). To zodpovedá výpočtu koeficientov podľa (3.9), (3.10) a pridaniu rezov prípustnosti v tvare (3.3). Teraz už vieme definovať kompletnú hlavnú úlohu (3.1)–(3.5), ktorej riešením v jednotlivých iteráciách ν dostávame zakaždým nové optimálne riešenie $\hat{\boldsymbol{x}}_\nu$, $\hat{\theta}_\nu$. Algoritmus skončí vtedy, keď nastane $\boldsymbol{u}_\nu^\top \hat{\boldsymbol{x}}_\nu + \alpha_\nu \leq \hat{\theta}_\nu$, čo sa dá interpretovať tak, že funkcia $\mathcal{Q}(\boldsymbol{x})$ je už dobre aproximovaná a preto sa $\hat{\boldsymbol{x}}_\nu$ v poslednej makroiterácii ν realizuje v mieste, kde sa oporná nadrovina pridaná v predposlednej makroiterácii $\nu - 1$ dotkla $\mathcal{Q}(\boldsymbol{x})$.

Poznamenávame, že okrem uvedenej Van Slyke - Wetsovej metódy [1969] založenej na linearizácii primárnej úlohy (ktorá je vlastne Bendersov dekompozičný algoritmus [1960], rozšírený o riešenie problému s prípustnosťou), je tiež známa Dantzig - Wolfeho metóda založená na linearizácii duálnej úlohy, ktorá je podrobne popísaná v [4].

3.2 Viacrezová L-Shaped metóda

V algoritme L-Shaped metódy boli v Kroku 3 riešené druhostupňové úlohy pre všetky scenáre $k = 1, ..., K$ a následne bol pridaný jeden rez optimality s koeficientami počítanými podľa (3.12) a (3.13). Existuje aj odlišný prístup, kedy sa rezy

optimality pridávajú pre každý scenár zvlášť. Najprv uvedieme schému takéhoto algoritmu podľa [2], potom ho porovnáme s predchádzajúcim algoritmom.

3.2.1 Algoritmus

Krok 0: Inicializácia.
Polož $r = \nu = 0$ a $s(k) = 0$ pre všetky $k = 1, ..., K$.

Krok 1: Riešenie hlavnej úlohy.
Plož $\nu = \nu + 1$. Rieš úlohu:

$$\min_{\boldsymbol{x}, \{\theta_k} z = \boldsymbol{c}^\top \boldsymbol{x} + \sum_{k=1}^{K} \theta_k \tag{3.23}$$

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{3.24}$$

$$\boldsymbol{G}_l \boldsymbol{x} \geq g_l, \ l = 1, ..., r, \tag{3.25}$$

$$\boldsymbol{E}_{l(k)} \boldsymbol{x} + \theta_k \geq e_{l(k)}, \ l(k) = 1, ..., s(k), \ k = 1, ..., K, \tag{3.26}$$

$$\boldsymbol{x} \geq \boldsymbol{0}, \ \theta_k \in \mathbb{R}, \ k = 1, ..., K. \tag{3.27}$$

Dostávame optimálne riešenie $(\boldsymbol{x}^\nu, \theta_1^\nu, ..., \theta_K^\nu)$. Ak pre niektoré k neboli pridané rezy optimality (3.26), θ_k^ν sa položí rovné $-\infty$ a neuvažuje sa pri výpočte $\boldsymbol{x}^\nu$.

Krok 2: Kontrola prípustnosti v úlohách druhého stupňa, pridávanie rezov prípustnosti (3.25).
Rovnako ako v predchádzajúcom algoritme 3.1.1.

Krok 3: Riešenie úloh druhého stupňa, pridávanie rezov optimality (3.26).
Pre všetky scenáre $k = 1, ..., K$ rieš úlohu druhého stupňa (3.11). Nech $\boldsymbol{\pi}_k^\nu$ je vektor simplexových multipikátorov pridružený k optimálnemu riešeniu úlohy k. Ak platí nerovnosť

$$\theta_k^\nu < p_k(\boldsymbol{\pi}_k^\nu)^\top (\boldsymbol{h_k} - T_k \boldsymbol{x^\nu}), \tag{3.28}$$

definuj

$$\boldsymbol{E}_{s(k)+1} = p_k(\boldsymbol{\pi}_k^\nu)^\top T_k \tag{3.29}$$

$$e_{s(k)+1} = p_k(\boldsymbol{\pi}_k^\nu)^\top \boldsymbol{h}_k \tag{3.30}$$

Polož $s(k) = s(k) + 1$. Ak (3.28) neplatí pre žiadne $k = 1, ..., K$, $\boldsymbol{x}^\nu$ je optimálne riešenie - stop, inak sa vráť na Krok 1.
Makroiterácie ako aj mikroiterácie sú v tomto algoritme definované rovnako ako v algoritme (3.1.1), pričom ν je index makroiterácií a k index mikroiterácií a scenárov.

Vo všeobecnosti sa v algoritme (3.2.1) pri porovnaní so základnou L-Shaped metódou dosahuje presnejšia aproximácia funkcie $\mathcal{Q}(\boldsymbol{x})$ v každej makroiterácii, čím sa redukuje počet makroiterácií potrebných pre vyriešenie úlohy (2.2). Na druhej

strane, tým že v každej makroiterácii pridávame rezy optimality pre všetky scenáre, dosahuje hlavná úloha (3.23)–(3.27) vačšie rozmery než (3.1)–(3.4). To, ktorý algoritmus je efektívnejší závisí od konkrétnej úlohy. V [3] sú uvedené nasledovné závislosti pre maximálny počet makrotiterácií v oboch algoritmoch. Pre viacrezový algoritmus je horná hranica počtu makroiterácií daná vzťahom

$$1 + K(a^{m_2} - 1), \tag{3.31}$$

pričom pre algoritmus L-Shaped metódy máme

$$[1 + K(a - 1)]^{m_2}, \tag{3.32}$$

kde K je počet realizácií $\boldsymbol{\xi}$, m_2 je počet ohraničení v úlohách druhého stupňa (2.4), a predstavuje takzvané *spádové číslo* úlohy dané nasledovnou definíciou.

Definícia 1. ([3], str. 9) *Nech $a(\boldsymbol{\xi})$ predstavuje maximálny počet rôznych spádov $Q(\boldsymbol{x}, \boldsymbol{\xi})$ v ľubovoľnom smere rovnobežnom s niektorou súradnou osou pre dané $\boldsymbol{\xi}$. To znamená maximálny počet rôznych buniek (polyedrického rozkladu $K_1 \cap K_2$ vzhľadom na $Q(\boldsymbol{x}, \boldsymbol{\xi})$ pre dané $\boldsymbol{\xi}$) pretnutých ľubovoľnou polpriamkou (rovnobežnou s niektorou súradnou osou) začínajúcou v ľubovoľnom bode z $K_1 \cap K_2$. Definujme $a = \max_{\boldsymbol{\xi} \in \Xi} a(\boldsymbol{\xi})$ ako spádové číslo druhostupňovej úlohy (2.4).*

Poznamenávame, že parametre K, a, m_2 sú dané úlohou, nezávisia od použitého algoritmu.

3.3 L-Shaped metóda s adaptívnou agregáciou rezov optimality

V tejto časti vychádzajúc z [11] popíšeme a odôvodníme algoritmus, ktorý môže v niektorých prípadoch znamenať signifikantné zlepšenie výpočtového času v porovnaní s L-Shaped metódou a viacrezovou L-Shaped metódou. V Kapitole 5.2 z neho budeme vychádzať pri zostavení algoritmu s adaptívnou agregáciou rezov optimality pre viacstupňové SLÚ s pevnou kompenzáciou.

Algoritmy L-Shaped metódy a viacrezovej L-Shaped metódy uvedené v predchádzajúcich častiach 3.1 a 3.2 predstavujú z hľadiska úrovne agregácie rezov optimality dva extrémne prípady. V L-shaped metóde sa v každej makroiterácii pridáva do hlavnej úlohy jeden rez optimality spoločný pre všetky scenáre (realizácie $\boldsymbol{\xi}$). Preto tu dochádza k informačnej strate v dôsledku zlučovania duálnej informácie všetkých scenárov pre výpočet jediného rezu optimality. Vo viacrezovej L-Shaped metóde je naopak duálna informácia maximálne využitá, pretože sa pre každý scenár pridáva do hlavnej úlohy osobitný rez optimality.

Označme $\mathcal{S}$ množinu scenárov a $K \triangleq |\mathcal{S}|$ počet scenárov pre danú úlohu. V makroiterácii ν dostávame pre L-Shaped metódu hlavnú úlohu rozmerov maximálne $(m_1+\nu)\times(n_1+1)$ a pre viacrezovú L-Shaped metódu maximálne $(m_1+\nu K)\times(n_1+K)$. Pri veľkom počte scenárov tak môže v druhom prípade výpočtový čas značne narásť. To viedlo autorov [11] ku konštrukcii algoritmu s čiastočným zoskupením rezov optimality. Množina všetkých scenárov $\mathcal{S}$ je tu rozčlenená do L skupín tak, že platí $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup ...\mathcal{S}_L$ a $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ pre všetky $i \neq j$. Každej takejto skupine s indexom $l \in \{1, ..., L\}$ je v algoritme priradená premenná θ_l a sú pre ňu generované rezy optimality v tvare $\boldsymbol{E}_l^\nu \boldsymbol{x} + \theta_l \geq e_l^\nu$, kde $\boldsymbol{E}_l^\nu = \sum_{s\in\mathcal{S}_l} p_s(\boldsymbol{\pi}_s^\nu)^\top T_s$ a $e_l^\nu = \sum_{s\in\mathcal{S}_l} p_s(\boldsymbol{\pi}_s^\nu)^\top \boldsymbol{h}_s$.

V prípade L-Shaped metódy nastáva maximálna úroveň zoskupenia ($L = 1$), pre viacrezovú metódu máme minimálne zoskupenie ($L = K$). Podľa výsledkov uvedených v [11] sa optimálny výpočtový čas dosahuje pre nejaký stupeň zoskupenia $1 < L < K$, ktorý vopred nemožno stanoviť. Preto sa v adaptívnom algoritme na začiatku zvolí nejaká úroveň zoskupenia, ktorá sa postupne môže zvyšovať na základe novej informácie získanej o funkcii $\mathcal{Q}(\boldsymbol{x})$. Takéto priebežné zoskupovanie sa vykonáva na základe konkrétnych pravidiel, ktoré uvedieme neskôr.

Definujme $S(\nu) = \{S_1^{(\nu)}, S_2^{(\nu)}, ..., S_{L_\nu}^{(\nu)}\}$[4], kde $S_i^{(\nu)} \cap S_j^{(\nu)} = \emptyset$ pre všetky $i \neq j$ a $\bigcup_{i=1}^{L_\nu} S_i^{(\nu)} = \mathcal{S}$. Ďalej zaveďme indexovú množinu pre skupiny scenárov v makroiterácii ν ako $\Lambda(\nu) = \{1, ..., L_\nu\}$ a funkciu vracajúcu indexy pre prvky $d \in S(\nu)$:

$$\lambda(d;\nu) : S(\nu) \to \Lambda(\nu) \text{ takú, že } \lambda(S_j^{(\nu)};\nu) = j.$$

Pre pravdepodobnosti jednotlivých skupín scenárov $S_i^{(\nu)}$ platí $p_{S_i^{(\nu)}} = \sum_{s\in S_i^{(\nu)}} p_s$, kde p_s je pravdepodobnosť scenára s a $\sum_{i=1}^{L_\nu} p_{S_i^{(\nu)}} = \sum_{s\in S} p_s = 1$. Dostávame nasledovnú *hlavnú úlohu* v makroiterácii ν:

$$\min_{\boldsymbol{x},\{\theta_l\}_{l\in\Lambda(\nu)}} \quad \boldsymbol{c}^\top\boldsymbol{x} + \sum_{l\in\Lambda(\nu)} \theta_l, \tag{3.33}$$

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{3.34}$$

$$\boldsymbol{G}_j\boldsymbol{x} \geq g_j,\ j = 1, ..., r(\nu), \tag{3.35}$$

$$\boldsymbol{E}_{l,j}\boldsymbol{x} + \theta_l \geq e_{l,j},\ j = 1, ..., s(\nu),\ l \in \Lambda(\nu) \tag{3.36}$$

$$\boldsymbol{x} \geq 0, \tag{3.37}$$

kde $r(\nu)$ označuje počet rezov prípustnosti (3.35) pridaných po makroiteráciu ν a $s(\nu)$ označuje počet makroiterácií, v ktorých boli pridávané rezy prípustnosti (3.36) po makroiteráciu ν. Teraz pristúpime k predstaveniu samotnej iteračnej schémy adaptívneho algoritmu L-Shaped metódy podľa [11] s miernou modifikáciou v Kroku 3b (v [11] označený ako Krok 2b), ktorú odôvodníme neskôr.

[4]Pozor, $\mathcal{S} \neq S(\nu)$. V prípade $S(\nu)$ ide o dvoj-úrovňovú množinu.

3.3.1 Algoritmus

Krok 0: Inicializácia.
Nastav $\nu = 0$, $s(0) = 0$, $r(0) = 0$, inicializuj $S(0)$ na začiatočný stav.

Krok 1: Riešenie hlavnej úlohy.
Rieš hlavnú úlohu (3.33)–(3.37). Nech $(\boldsymbol{x}^\nu, \{\theta_l^\nu\}_{l\in\Lambda(\nu)})$ je optimálne riešenie hlavnej úlohy v iterácii ν. Ak pre nejaké $l \in \Lambda(\nu)$ neboli doposiaľ pridané ohraničenia (3.36), θ_l^ν sa nastaví na $-\infty$ a je vo výpočte ignorovaná. Nastav $\nu = \nu + 1$.

Krok 2: Kontrola prípustnosti v úlohách druhého stupňa a pridávanie rezov prípustnosti (3.35)[5]
Rovnako ako v algoritme 3.1.1.

Krok 3: Aktualizuj úroveň agregácie.
Krok 3a: Agregácia scenárov.
Vytvor agregát $S(\nu)$ z $S(\nu-1)$ využitím pravidiel agregácie. Každý prvok $S(\nu)$ je zjednotením nejakých prvkov z $S(\nu-1)$. Ak sa podľa agregačných pravidiel skupiny $d_1, d_2, ..., d_m \in S(\nu-1)$ zagregujú do $d \in S(\nu)$, potom $d = \bigcup_{i=1}^m d_i$ a $p_d = \sum_{i=1}^m p_{d_i}$. Z hlavnej úlohy (3.33)–(3.37) sa odstránia premenné $\theta_{\lambda(d_1;\,\nu-1)}, ..., \theta_{\lambda(d_m;\,\nu-1)}$ a pridá sa namiesto nich nová premenná $\theta_{\lambda(d;\,\nu)}$.

Krok 3b: Aktualizácia rezov optimality.
Pre všetky iterácie $j = 1, ..., s(\nu)$, v ktorých boli pridané *rezy optimality*, definuj

$$e_{j,\lambda(d;\,\nu)} = \sum_{l=1}^{m} e_{j,\lambda(d_l;\,\nu-1)}, \tag{3.38}$$

a

$$\boldsymbol{E}_{j,\lambda(d;\,\nu)} = \sum_{l=1}^{m} \boldsymbol{E}_{j,\lambda(d_l;\,\nu-1)}. \tag{3.39}$$

Pre všetky iterácie $j = 1, ..., s(\nu)$ nahraď rezy typu (3.36) prislúchajúce k $d_1, ..., d_m$ novým rezom $\boldsymbol{E}_{j,\lambda(d;\,\nu)}\boldsymbol{x} + \theta_{\lambda(d;\,\nu)} \geq e_{j,\lambda(d;\,\nu)}$ prislúchajúcim k d. Vypočítaj hodnotu $\theta^\nu_{\lambda(d;\,\nu)} = \sum_{l=1}^m \theta^{\nu-1}_{\lambda(d_l;\,\nu-1)}$.

Krok 4: Riešenie podúloh pre jednotlivé scenáre.
Pre všetky scenáre $k = 1, ..., K$ rieš úlohu druhého stupňa (3.11). Nech $\boldsymbol{\pi}_k^\nu$ je vektor duálnych multiplikátorov priradených k riešeniu (3.11). Pre každé $d \in S(\nu)$ ak

$$\theta^\nu_{\lambda(d;\,\nu)} < p_d \sum_{k\in d} (\boldsymbol{\pi}_k^\nu)^\top (\boldsymbol{h}_k - T_k \boldsymbol{x}^\nu), \tag{3.40}$$

[5]Tento krok bol v [11] uvedený na konci algoritmu, tu sme zjednotili jeho formuláciu s algoritmami z [2].

definuj

$$e_{n,\lambda(d;\nu)} = \sum_{k \in d} p_k (\boldsymbol{\pi}_k^\nu)^\top \boldsymbol{h}_k \tag{3.41}$$

a

$$\boldsymbol{E}_{n,\lambda(d;\nu)} = \sum_{k \in d} p_k (\boldsymbol{\pi}_k^\nu)^\top T_k, \tag{3.42}$$

kde $n = s(\nu) + 1$, do hlavnej úlohy (3.33)–(3.37) pridaj rez optimality typu (3.36).

Ak podmienka (3.40) nie je splnená pre žiadne $d \in S(\nu)$ koniec, $\boldsymbol{x}^\nu$ je optimálne. Inak polož $s(\nu) = s(\nu) + 1$ a choď na Krok 1.

Konvergencia uvedeného algoritmu plynie priamo z konvergencie viacrezovej L-Shaped metódy. Pre kompletný opis algoritmu zostáva ešte uviesť spomínané pravidlá pre zoskupovanie scenárov (zároveň rezov optimality).

3.3.2 Pravidlá agregácie

- *Hladina nadbytočnosti δ.*
  Stáva sa, že niektoré rezy optimality, ktoré boli pridané do hlavnej úlohy, obsahujú málo informácie o optimálnom riešení. Takéto rezy môžu byť zlúčené, pričom nedôjde k signifikantnej strate informácie. Uvažujme iteráciu ν a nejakú skupinu scenárov $d \in S(\nu)$. Označme ďalej f_d počet iterácií kedy všetky rezy optimality prislúchajúce k d boli nadbytočné. Za nadbytočné sa považujú ohraničenia, ktoré vychádzajú ako neaktívne, čiže sa nerealizujú. Potom podľa tohto pravidla budeme zlučovať všetky skupiny d, pre ktoré platí $f_d/s(\nu) > \delta$, kde $\delta \in (0, 1)$ je hladina nadbytočnosti, ktorú si určíme. Ako sa uvádza v [11], toto pravidlo funguje dobre ak je celkový počet iterácií dostatočne veľký, v opačnom prípade treba zaviesť takzvanú zahrievaciu periódu, počas ktorej nebudeme zlučovať žiadne rezy optimality.

- *Ohraničenie minimálneho počtu zoskupení*
  Aby sa zabránilo predčasnému zoskupeniu všetkých scenárov, zavádza sa ohraničenie pre minimálny počet zoskupení $|S(\nu)|$. Toto sa nastaví pred spustením algoritmu a následne zostáva nezmenené.

- *Maximálny počet zoskupených rezov*
  Pre zamedzenie príliš rýchleho zoskupovania môže ďalej slúžiť ohraničenie, ktoré udáva, koľko scenárov je možné zlúčiť do jednej skupiny, čiže maximálna veľkosť skupiny.

3.3.3 Modifikácia pôvodného algoritmu

V nasledovnom ešte objasníme spomínanú modifikáciu Kroku 3b (v [11] je označený ako Krok 2b), ktorú sme navrhli na základe výsledkov numerických experimentov.

Algoritmus uvedený v [11] má koeficienty starých rezov agregované nasledovným spôsobom:

$$e_{j,\lambda(d;\nu)} = p_d \sum_{l=1}^{m} (1/p_{d_l}) e_{j,\lambda(d_l;\nu-1)} \ \forall d \in S(\nu),\ j = 1, ..., s(\nu), \tag{3.43}$$

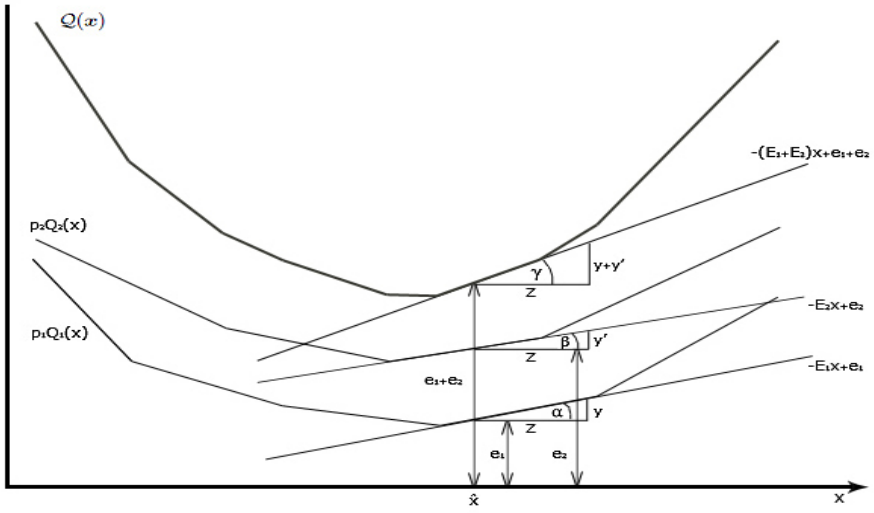
$$\boldsymbol{E}_{j,\lambda(d;\nu)} = p_d \sum_{l=1}^{m} (1/p_{d_l}) \boldsymbol{E}_{j,\lambda(d_l;\nu-1)} \ \forall d \in S(\nu),\ j = 1, ..., s(\nu), \tag{3.44}$$

kde p_{d_l}, $l = 1, ..., m$ sú pravdepodobnosti skupín $d_1, d_2, ..., d_m \in S(\nu - 1)$ a p_d je pravdepodobnosť agregovanej skupiny $d = \bigcup_{i=1}^{m} d_i$, $d \in S(\nu)$. Výpočet hodnoty $\theta^{\nu}_{\lambda(d;\nu)} = \sum_{l=1}^{m} \theta^{\nu-1}_{\lambda(d_l;\nu-1)}$ v algoritme z [11] nie je, čo sa nám v kontexte schémy, ktorú uvádzajú nezdá vcelku korektné. V nasledujúcom poľahky ukážeme, že takýmto spôsobom agregácie nedostávame presný rez ale iba akúsi hrubú aproximáciu.

Predstavme si jednoduchú dvojstupňovú SLÚ s pevnou kompenzáciou, ktorá má dva scenáre. V prípade výpočtu koeficientov podľa viacrezovej L-Shaped metódy (3.30) a (3.29) dostávame koeficienty $e_1 = p_1\pi_1 h_1$, $E_1 = p_1\pi_1 T_1$ pre prvý scenár a $e_2 = p_2\pi_2 h_2$, $E_2 = p_2\pi_2 T_2$ pre druhý scenár. V prípade výpočtu pre šandardnú L-Shaped metódu dostávame podľa (3.13), (3.12) koeficienty $e = p_1\pi_1 h_1 + p_2\pi_2 h_2$ a $E = p_1\pi_1 T_1 + p_2\pi_2 T_2$. Ak by sme agregovali rezy viacrezovej metódy s použitím (3.43) a (3.44), dostávame $\tilde{e} = (p_1+p_2)(\frac{1}{p_1}e_1 + \frac{1}{p_2}e_2) = (p_1+p_2)\pi_1 h_1 + (p_1+p_2)\pi_2 h_2$, podobne $\tilde{E} = (p_1 + p_2)\pi_1 T_1 + (p_1 + p_2)\pi_2 T_2$ zjavne $\tilde{e} \neq e$ a $\tilde{E} \neq E$. Po agregácii starých rezov dostávame teda iný rez, ako by sme dostali v prípade, keď by sme ho rátali pôvodne pre štandardnú (jednorezovú) L-Shaped metódu - teda by sme od začiatku predpokladali agregáciu scenárov.

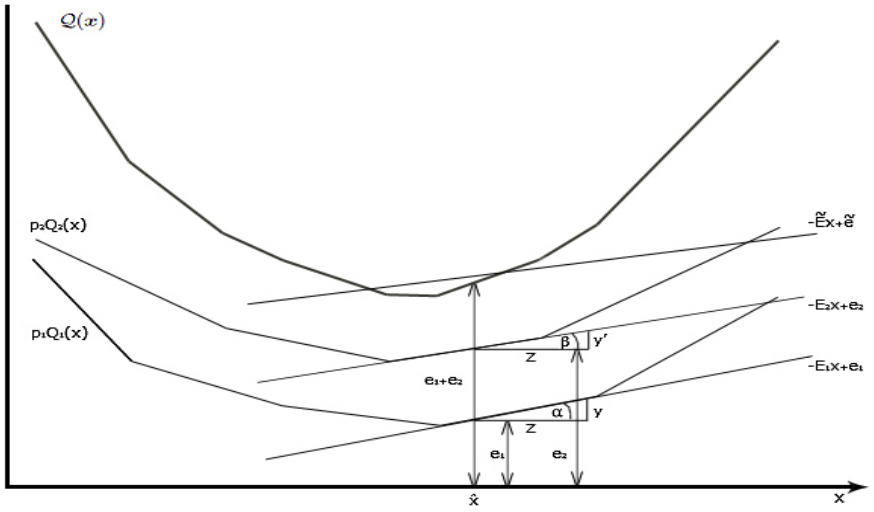
Autori v [11] robili vzorkovanie ('sampling') pre úlohy s veľkým počtom scenárov, kde sa v rámci vzoriek nachádzali zakaždým rozličné scenáre. Preto prirodzene museli dostať pre každú vzorku scenárov trocha iné riešenie a optimálnu hodnotu účelovej funkcie. Z tohto dôvodu si pravdepodobne nevšimli, že pri použití takejto agregácie môže dôjsť k nepresnému vyriešeniu úlohy. Pri testovaní adaptívneho algoritmu vnorenej L-shaped metódy (Algoritmus 5.2.1) pre viacstupňové úlohy sa hodnota účelovej funkcie pri použití takejto agregácie často nezhodovala so skutočnou hodnotou. Preto sme navrhli spôsob uvedený v predošlom algoritme v Kroku 3b. Ukázalo sa, že staré rezy možno jednoducho sčítať, namiesto váženia pravdepodobnosťami ako v (3.43) a (3.44), čo sa pokúsime graficky ilustrovať na Obr. 2.

Uvažujme prípad dvojstupňovej SLÚ s pevnou kompenzáciou, ktorá má 2 scenáre a jednorozmernú premennú x. V prípade viacrezovej L-Shaped metódy ('multicut') aproximujeme zvlášť čiastkové funkcie $p_1 Q_1(x)$ a $p_2 Q_2(x)$. V prípade agregácie scenárov ('singlecut') aproximujeme jedným rezom funkciu $\mathcal{Q}(x) = p_1 Q_1(x) + p_2 Q_2(x)$.



Obr. 2: Agregácia rezov v prípade jednoduchého sčítavania.

Na Obrázku 2 vidíme čiastkové funkcie $p_1Q_1(x)$ a $p_2Q_2(x)$ aproximované rezmi $-E_1x + e_1$ a $-E_2x + e_2$ v okolí bodu $\hat{x}$. Nakoľko platí $\mathcal{Q}(x) = p_1Q_1(x) + p_2Q_2(x)$, dostávame tg $\alpha = \frac{y}{z} = -E_1$, tg $\beta = \frac{y'}{z} = -E_2$, tg $\gamma = \frac{y+y'}{z} = -(E_1 + E_2)$. Vidíme teda, že súčet koeficientov $-E_1 + (-E_2)$ nám dáva dobrú smernicu rezu pre agregovanú verziu, aproximujúcu $\mathcal{Q}(x)$. Odôvodnenie súčtu $e_1 + e_2$ je evidentné priamo z obrázku.

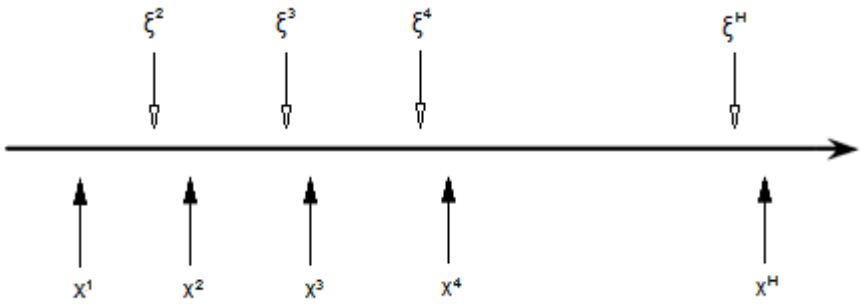


Obr. 3: Agregácia rezov v prípade aproximácie cez pravdepodobnosti.

V prípade použitia aproximácie z [11] môže nastať aj prípad zachytený na Obr. 3, kedy rez nebude dobre aproximovať funkciu $\mathcal{Q}(x)$. Môže nastať prípad, kedy rez $\mathcal{Q}(x)$ napríklad pretne.

4 Viacstupňové stochastické lineárne úlohy s pevnou kompenzáciou

V predchádzajúcej časti sme písali o stochastických lineárnych úlohách, kde rozhodovanie prebieha v dvoch stupňoch. Teraz sa pozrieme na všeobecnejší model, v ktorom rozhodovací proces pozostáva z ľubovoľného počtu H stupňov. Graficky je takýto proces znázornený na Obr. 4. Jednotlivé rozhodnutia $\boldsymbol{x}^i$ sa tu striedajú v



Obr. 4: Viacstupňový proces rozhodovania.

čase s realizáciami náhodných dát $\boldsymbol{\xi}^i$. Každé rozhodnutie $\boldsymbol{x}^i$ tu možno považovať za korekčné rozhodnutie vzhľadom na všetky predošlé rozhodnutia $\boldsymbol{x}^j$, $j = 1, ..., i-1$. Na rozdiel od dvojstupňových SLÚ, kde $\boldsymbol{\xi}$ označovalo náhodný vektor, tu bude $\boldsymbol{\xi} = (\boldsymbol{\xi}^1, ..., \boldsymbol{\xi}^H)$ označovať stochastický proces, pričom $\boldsymbol{\xi}^t$ pre $t = 2, ..., H$ sú náhodné vektory a $\boldsymbol{\xi}^1$ predstavuje vo všeobecnosti prvostupňové dáta, ktoré sú deterministické.

4.1 Formulácia úlohy a deterministický ekvivalent

Pod viacstupňovou stochastickou lineárnou úlohou s pevnou kompenzáciou budeme podľa [2] rozumieť úlohu v nasledovnom tvare:

$$\begin{aligned}
&\min_{(\boldsymbol{x}^1,...,\boldsymbol{x}^H)} \boldsymbol{c}^1\boldsymbol{x}^1 + \mathbb{E}_{\boldsymbol{\xi}^2}[\min\, \boldsymbol{c}^2(\omega^2)\boldsymbol{x}^2(\omega^2) + ... + \mathbb{E}_{\boldsymbol{\xi}^H}[\min\, \boldsymbol{c}^H(\omega^H)\boldsymbol{x}^H(\omega^H)]...] \\
&\qquad W^1\boldsymbol{x}^1 = \boldsymbol{h}^1, \\
&\qquad T^1(\omega^2)\boldsymbol{x}^1 + W^2\boldsymbol{x}^2(\omega^2) = \boldsymbol{h}^2(\omega^2), ... \\
&\qquad T^{H-1}(\omega^H)\boldsymbol{x}^{H-1}(\omega^{H-1}) + W^H\boldsymbol{x}^H(\omega^H) = \boldsymbol{h}^H(\omega^H), \\
&\qquad \boldsymbol{x}^1 \geq 0;\ \boldsymbol{x}^t(\omega^t) \geq \boldsymbol{0},\ t = 2, ..., H;
\end{aligned} \tag{4.1}$$

kde $\boldsymbol{c}^1$ je vektor z $\mathbb{R}^{n_1}$, $\boldsymbol{h}^1$ je vektor z $\mathbb{R}^{m_1}$, $\boldsymbol{\xi}^t(\omega^t)^\top = (\boldsymbol{c}^t(\omega^t)^\top, \boldsymbol{h}^t(\omega^t)^\top, T_1^{t-1}(\omega^t), ..., T_{m_t}^{t-1}(\omega^t))$ sú náhodné vektory rozmeru N_t definované na (Ω, Σ^t, P), kde $\Sigma^t \subset \Sigma^{t+1}$ pre všetky $t = 2, ..., H$. Matice W^t sú známe, nenáhodné, rozmeru $m_t \times n_t$. Rozhodnutia $\boldsymbol{x}^t$ závisia od vývoja do času t, ktorý značíme ω^t, prípadne neskôr ω. Ďalej definujeme $\Xi^t \subset \mathbb{R}^{N_t}$ ako nosič náhodného vektora $\boldsymbol{\xi}^t$.

Teraz popíšeme deterministický ekvivalent k úlohe (4.1) podľa [2]. Uvažujme stupne $t = 1, ..., H$, pričom rozhodnutia v jednotlivých stupňoch nech sú $\boldsymbol{x}^t(\omega^t)$. Keďže celá súvislosť medzi stupňami spočíva práve v týchto rozhodnutiach, vieme zadefinovať rekurziu ako v dynamickom programovaní. Majme terminálnu podmienku v tvare

$$\begin{aligned} Q^H(\boldsymbol{x}^{H-1}, \boldsymbol{\xi}^H(\omega)) &= \min_{\boldsymbol{x}^H} \boldsymbol{c}^H(\omega)\boldsymbol{x}^H(\omega) \\ W^H\boldsymbol{x}^H(\omega) &= \boldsymbol{h}^H(\omega) - T^{H-1}(\omega)\boldsymbol{x}^{H-1}, \\ \boldsymbol{x}^H(\omega) &\geq \mathbf{0}. \end{aligned} \tag{4.2}$$

Ak položíme $\mathcal{Q}^{t+1}(\boldsymbol{x}^t) = \mathbb{E}_{\boldsymbol{\xi}^{t+1}}[Q^{t+1}(\boldsymbol{x}^t, \boldsymbol{\xi}^{t+1}(\omega))]$ pre všetky t, potom pre $t = 2, ..., H-1$, dostávame rekurziu

$$\begin{aligned} Q^t(\boldsymbol{x}^{t-1}, \boldsymbol{\xi}^t(\omega)) &= \min_{\boldsymbol{x}^t} \boldsymbol{c}^t(\omega)\boldsymbol{x}^t(\omega) + \mathcal{Q}^{t+1}(\boldsymbol{x}^t) \\ W^t\boldsymbol{x}^t(\omega) &= \boldsymbol{h}^t(\omega) - T^{t-1}(\omega)\boldsymbol{x}^{t-1}, \\ \boldsymbol{x}^t(\omega) &\geq \mathbf{0}. \end{aligned} \tag{4.3}$$

Potom úloha, ktorú riešime, má tvar

$$\begin{aligned} \min_{\boldsymbol{x}^1} \boldsymbol{c}^1\boldsymbol{x}^1 + \mathcal{Q}(\boldsymbol{x}^1) \\ W^1\boldsymbol{x}^1 = \boldsymbol{h}^1, \\ \boldsymbol{x}^1 \geq \mathbf{0}. \end{aligned} \tag{4.4}$$

Zrejme pre $H = 2$ dostávame formuláciu ekvivalentnú s dvojstupňovou SLÚ s pevnou kompenzáciou (2.2)–(2.4). Všimnime si, že deterministický ekvivalent (4.2)–(4.4) vlastne rozložil pôvodnú úlohu (4.1) na skupinu podúloh. Dostávame teda dekompozíciu pôvodnej úlohy, ktorá sa využíva v algoritmoch[6].

Nakoľko nám v tejto práci ide predovšetkým o numerické algoritmy pre riešenie úloh, budeme ďalej predpokladať, že vektor $\boldsymbol{\xi}^t$ môže nadobúdať iba konečný počet hodnôt, čiže má diskrétne konečné rozdelenie. Potom nosič pre $\boldsymbol{\xi}^t$ možno zapísať v tvare

$$\Xi^t \triangleq \{\boldsymbol{\xi}_i^t : i = 1, ..., n_t \in \mathbb{N}\},$$

kde $t = 1, ..., H$ a n_t je počet možných realizácií $\boldsymbol{\xi}^t$.

Definujme množiny prípustnosti pre (4.3) v tvare

$$K^t \triangleq \{\boldsymbol{x}^t | \mathcal{Q}^{t+1}(\boldsymbol{x}^t) < \infty\}. \tag{4.5}$$

Nasledovná veta zaručuje pre funkcie $\mathcal{Q}^{t+1}(\boldsymbol{x}^t)$ a množiny prípustnosti K^t vlastnosti užitočné pre konštrukciu algoritmov.

Veta 4.1. ([2], str. 129) *Množiny K^t a funkcie $\mathcal{Q}^{t+1}(\boldsymbol{x}^t)$ sú konvexné pre $t = 1, ..., H-1$ a ak Ξ^t je konečná pre $t = 1, ..., H$, potom K^t a $\mathcal{Q}^{t+1}(\boldsymbol{x}^t)$ sú polyedrické.*

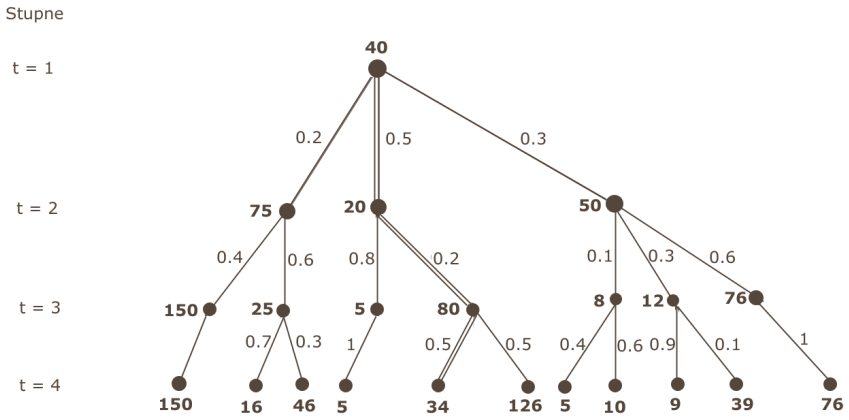
Dôkaz je uvedený v [2], str. 129.

[6]Napríklad Bendersov dekompozičný algoritmus.

4.2 Stromy scenárov

Konečná distribúcia vektorov $\boldsymbol{\xi}^t$ pre $t = 1, ..., H$ umožňuje reprezentovať náhodný proces $\boldsymbol{\xi}$ ako *strom scenárov*. *Koreň* tohto stromu zodpovedá vektoru $\boldsymbol{\xi}^1$ s jedinou realizáciou. Jednotlivé úrovne stromu predstavujú zhora od koreňa smerom nadol[7] vzostupne usporiadané stupne v úlohe (4.2)–(4.4). Vo všeobecnosti sa počet uzlov v úrovni t rovná počtu možných realizácií $\boldsymbol{\xi}^t$. Každý uzol v stupni $t = 2, ..., H$ je prepojený s práve jedným uzlom v stupni $t-1$, ktorý predstavuje jeho *predchodcu*. Zároveň každý uzol v stupni $t = 1, ..., H-1$ je prepojený s nejakými uzlami v stupni $t+1$, ktoré sa označujú ako jeho *nasledovníci*. Množinu všetkých uzlov na úrovni $t = 1, ..., H$ budeme označovať Ω_t.

Pri konečnom počte scenárov K sú jednotlivé scenáre reprezentované cestami v strome scenárov od koreňa až po poslednú úroveň $t = H$, ktorej uzly sa nazývajú *listy*. Počet listov v strome scenárov zodpovedá celkovému počtu scenárov K.



Obr. 5: Strom scenárov.

Na Obr. 5 je pre názornosť zachytený konkrétny príklad stromu scenárov. Čísla na spojovacích čiarach predstavujú pravdepodobnosti nastatia danej vetvy. Čísla zvýraznené tučným písmom predstavujú realizácie náhodných dát (v praxi to zvyknú byť vektory a matice). Zdvojenou čiarou je vyznačená realizácia jedného z celkového počtu jedenástich možných scenárov.

Vo všeobecnosti rastie počet uzlov stromu exponenciálne v závislosti od jeho hĺbky, teda počtu stupňov. V praxi sa stretávame s úlohami, kde počet stupňov je veľmi veľký. Preto aj rozmery stromov scenárov bývajú v praxi veľké, čo si vyžaduje použitie efektívnych algoritmov.

[7] Závisí od orientácie vyobrazenia konkrétneho stromu.

5 Algoritmy pre viacstupňové SLÚ s pevnou kompenzáciou

V tejto časti najprv uvedieme základnú, takzvanú vnorenú verziu L-Shaped metódy pre riešenie viacstupňových SLÚ (4.2)–(4.4). Neskôr využijeme poznatok z dvojstupňových SLÚ, že rôzna úroveň agregácie rezov optimality (resp. scenárov) vedie v závislosti od riešenej úlohy k rozličnej výpočtovej efektivite a teda k rozdielnemu počítačovému času riešenia. Výsledky numerických experimentov uvedené v [11] poukazujú na možné výhody použitia prístupu s adaptívnou agregáciou rezov optimality pre dvojstupňové SLÚ s pevnou kompenzáciou. Preto sa pokúsime takýto prístup rozšíriť aj na viacstupňové SLÚ s pevnou kompenzáciou. Navrhneme a uvedieme schému vnorenej L-Shaped metódy s adaptívnym stupňom agregácie rezov optimality. Neskôr na niekoľkých numerických experimentoch vyhodnotíme efektivitu takéhoto prístupu v porovnaní s klasickou a viacrezovou vnorenou L-Shaped metódou. Podľa nám dostupných informácií nebol takýto adaptívny prístup pre viacstupňové úlohy doposiaľ navrhnutý ani implementovaný. Viacrezová vnorená L-Shaped metóda bola ako jedna z možností implementovaná napríklad H.I. Gassmanom v efektívnom solveri MSLiP, ktorý je naprogramovaný v jazyku FORTRAN. Viacej o tomto solveri sa dá nájsť v [5][8]. Podotýkame, že existujú rozličné iné vylepšenia vnorenej metódy, ktoré už boli napríklad v MSLiP implementované.

Bendersov dekompozičný algoritmus bol navrhnutý Bendersom (1962) pre riešenie dvojstupňových SLÚ. Následne bol Van-Slykem a Wetsom (1969) rozšírený o riešenie problému s prípustnosťou druhostupňových úloh a Birgem (1985) modifikovaný pre viacstupňové SLÚ a nazvaný vnorená L-Shaped metóda.

5.1 Vnorená L-Shaped metóda (Bendersov dekompozičný algoritmus)

Pred uvedením samotnej iteračnej štruktúry algoritmu bude užitočné najprv zadefinovať *hlavnú úlohu*, ktorá tu zohráva analogickú úlohu s (3.1)–(3.5) pre klasickú (dvojstupňovú) L-Shaped metódu.

Pre každý stupeň $t = 1, ..., H-1$ a každý uzol[9] $k = 1, ..., K^t$ v stupni t vieme

[8] Najnovšiu verziu a dokumentáciu tohto solveru, ktorý patrí k najlepším na svete, možno získať z webovej stránky autora: http://myweb.dal.ca/gassmann/

[9] Viď. 4.2 Stromy scenárov.

podľa [2] definovať *hlavnú úlohu* v tvare

$$\min_{(\boldsymbol{x}_k^t, \theta_k^t)} \ (\boldsymbol{c}_k^t)^\top \boldsymbol{x}_k^t + \theta_k^t \tag{5.1}$$

$$W^t \boldsymbol{x}_k^t = \boldsymbol{h}_k^t - T_k^{t-1} \boldsymbol{x}_{a(k)}^{t-1}, \tag{5.2}$$

$$\boldsymbol{G}_{k,j}^t \boldsymbol{x}_k^t \geq g_{k,j}^t, \ j = 1, ..., r_k^t, \tag{5.3}$$

$$\boldsymbol{E}_{k,j}^t \boldsymbol{x}_k^t + \theta_k^t \geq e_{k,j}^t, \ j = 1, ..., s_k^t, \tag{5.4}$$

$$\boldsymbol{x}_k^t \geq 0, \tag{5.5}$$

kde $a(k)$ je predchodca uzla k v stupni $t-1$, $\boldsymbol{x}_{a(k)}^{t-1}$ je priebežné riešenie pre daný uzol a kde pre $t = 1$, $\boldsymbol{b} \triangleq \boldsymbol{h}^1 - T^0 \boldsymbol{x}^0$ predstavuje počiatočné podmienky úlohy. V poslednom stupni H nie je prítomná premenná θ_k^H ani ohraničenia (5.3)–(5.4). Takúto hlavnú úlohu pre daný stupeň t a uzol k budeme označovať ako $NLDS(t, k)$. Označme ďalej $\mathcal{D}_k^t$ množinu nasledovníckych uzlov pre uzol $k \in \{1, ..., K^t\}$ v stupni $t \in \{1, ..., H-1\}$ stromu scenárov. Pre zjednodušenie zápisu vzťahov v algoritme bude užitočné zaviesť ďalej maticu $\boldsymbol{G}_k^t \triangleq ((\boldsymbol{G}_{k,1}^t)^\top, ..., (\boldsymbol{G}_{k,r_k^t}^t)^\top)^\top$ a vektor $\boldsymbol{g}_k^t \triangleq (g_{k,1}^t, ..., g_{k,r_k^t}^t)^\top$.

Teraz možno pristúpiť k formulácii algoritmu podľa [2] s miernou modifikáciou, ktorú následne objasníme.

5.1.1 Algoritmus

Krok 0: Inicializácia.
Nastav $t = 1$, $k = 1$, $r_k^t = s_k^t = 0$, pridaj ohraničenie $\theta_k^t = 0$ k (5.1)–(5.5) pre všetky t a k, polož *SMER* = *VPRED*. (r_k^t predstavuje počítadlo rezov prípustnosti a s_k^t počítadlo rezov optimality pre uzol k v stupni t).

Krok 1: Riešenie hlavnej úlohy.
Rieš priebežnú úlohu $NLDS(t, k)$.
∘ Ak je neprípustná:
  ∘ ak $t = 1$, koniec, celá úloha (4.2)–(4.4) je neprípustná.
  ∘ ak $t > 1$, polož $r_{a(k)}^{t-1} = r_{a(k)}^{t-1} + 1$. Nech podmienka neprípustnosti je daná duálnym riešením $\boldsymbol{\pi}_k^t, \boldsymbol{\rho}_k^t \geq \boldsymbol{0}$, takým, že $(\boldsymbol{\pi}_k^t)^\top W^t + (\boldsymbol{\rho}_k^t)^\top \boldsymbol{G}_k^t \leq \boldsymbol{0}$, ale $(\boldsymbol{\pi}_k^t)^\top (\boldsymbol{h}_k^t - T_k^{t-1} \boldsymbol{x}_{a(k)}^{t-1}) + (\boldsymbol{\rho}_k^t)^\top \boldsymbol{g}_k^t > 0$. Vypočítaj koeficienty rezov prípustnosti (5.3) ako

$$\boldsymbol{G}_{a(k),\ r_{a(k)}^{t-1}}^{t-1} = (\boldsymbol{\pi}_k^t)^\top T_k^{t-1} \tag{5.6}$$

$$g_{a(k),\ r_{a(k)}^{t-1}}^{t-1} = (\boldsymbol{\pi}_k^t)^\top \boldsymbol{h}_k^t + (\boldsymbol{\rho}_k^t)^\top \boldsymbol{g}_k^t. \tag{5.7}$$

Pridaj rez prípustnosti s (5.6) a (5.7) do $NLDS(t-1, a(k))$. Ulož t, k a *SMER* do *zásobníka* typu LIFO[10]. Nastav *SMER* = *VZAD*. Polož $t = t-1$, $k = a(k)$, vráť sa na Krok 1.

[10] Skratka pre Last in first out, prvý vložený prvok sa vyberie ako posledný.

◦ Ak je prípustná:
Aktualizuj hodnoty $\boldsymbol{x}_k^t$, θ_k^t. Ulož hodnoty duálnych multiplikátorov pre ohraničenia (5.2)–(5.4) ako $(\boldsymbol{\pi}_k^t, \boldsymbol{\rho}_k^t, \boldsymbol{\sigma}_k^t)$.

◦ Ak je zásobník neprázdny, vyber z neho naposledy uložené t, k a *SMER*. Pokračuj na Krok 2.

◦ Inak

◦ ak $k < K^t$, nastav $k = k + 1$, vráť sa na Krok 1. [posun na ďalší uzol v stupni]

◦ ak $k = K^t$, [posledný uzol v danom stupni]

◦ ak $SMER = VPRED$ a $t < H$, nastav $t = t + 1$, $k = 1$ a vráť sa na Krok 1.

◦ ak $t = H$, nastav $SMER = VZAD$. Pokračuj na Krok 2.

Krok 2: Výpočet rezov optimality.
Pre všetky uzly $k = 1, ..., K^{t-1}$ v $t - 1$ vypočítaj

$$\boldsymbol{E}_{k,n}^{t-1} = \sum_{l \in \mathcal{D}_k^t} \frac{p_l^t}{p_k^{t-1}} (\boldsymbol{\pi}_l^t)^\top T_l^{t-1}$$

a

$$e_{k,n}^{t-1} = \sum_{l \in \mathcal{D}_k^t} \frac{p_l^t}{p_k^{t-1}} \left[(\boldsymbol{\pi}_l^t)^\top \boldsymbol{h}_l^t + \sum_{i=1}^{r_l^t} (\rho_{li}^t)^\top g_{li}^t + \sum_{i=1}^{s_l^t} (\sigma_{li}^t)^\top e_{li}^t \right], \tag{5.8}$$

kde $n = s_k^{t-1} + 1$, ďalej ρ_{li}^t a σ_{li}^t predstavujú i-te skalárne zložky príslušných vektorov $\boldsymbol{\rho}_l^t$ a $\boldsymbol{\sigma}_l^t$. Súčasná podmienená stredná hodnota účelových funkcií všetkých podúloh v $\mathcal{D}_k^t$ je potom $\overline{\theta}_k^{t-1} = e_{k,n}^{t-1} - \boldsymbol{E}_{k,n}^{t-1} \boldsymbol{x}_k^{t-1}$. Ak sa ohraničenie $\theta_k^{t-1} = 0$ nachádza v $NLDS(t-1, k)$, odstráň ho, pridaj ohraničenie (5.4) do $NLDS(t-1, k)$ a polož $s_k^{t-1} = 1$. Ak $\overline{\theta}_k^{t-1} > \theta_k^{t-1}$, pridaj ohraničenie (5.4) s $\boldsymbol{E}_{k,n}^{t-1}$ a $e_{k,n}^{t-1}$ do *NLDS(t - 1, k)*, nastav $s_k^{t-1} = s_k^{t-1} + 1$.

Krok 3
Ak $t = 2$ a do $NLDS(1)$ neboli pridané žiadne ohraničenia, koniec, $\boldsymbol{x}_1^1$ je optimum, inak polož $t = t - 1$, $k = 1$. Ak $t = 1$, nastav $SMER = VPRED$. Choď na Krok 1.

Vo všeobecnosti existuje veľa možností pre voľbu poradia, v ktorom sa vyberajú jednotlivé uzly stromu scenárov. Konkrétny spôsob výberu uzlov sa zvykne označovať ako *sekvenčný protokol*. V prípade uvedeného algoritmu je implementovaný protokol označovaný ako *rýchlo vpred - rýchlo vzad* ("fast forward - fast back").

Algoritmus tu najprv postupne vyberá jednotlivé uzly v doprednom smere od koreňa k listom a rieši k nim pridružené hlavné úlohy. Riešenia $\boldsymbol{x}$ z predchádzajúcich stupňov sa posielajú vždy do nasledujúceho stupňa (vždy do hlavných úloh

prislúchajúcich nasledovníkom uzla, pre ktorý bola vyriešená hlavná úloha v nižšom stupni), kde vystupujú v ohraničeniach.

Po vyriešení všetkých hlavných úloh v poslednom stupni sa smer pohybu zmení na spätný a riešia sa postupne úlohy pre uzly od listov smerom ku koreňu. Duálne riešenia $(\boldsymbol{\pi}, \boldsymbol{\rho}, \boldsymbol{\sigma})$ z hlavných úloh vyšších stupňov sa posielajú do hlavných úloh v najbližšom nižšom stupni (vždy do hlavnej úlohy prislúchajúcej predchodcovi uzlov, pre ktoré boli vyriešené hlavné úlohy vo vyššom stupni), kde slúžia pre výpočet koeficientov rezov optimality.

Môže sa stať, že v niektorej hlavnej úlohe nastáva neprípustnosť. Pokiaľ sa jedná o úlohu v prvom stupni (prislúcha koreňu stromu scenárov), celá úloha je neprípustná a algoritmus skončí. V opačnom prípade sa neprípustnosť rieši pridávaním rezov prípustnosti do hlavných úloh v nižších stupňoch, čím sa postupne zužuje obor hodnôt pre ich riešenia $\boldsymbol{x}$. Pri odstraňovaní neprípustnosti sa môže aktuálny smer pohybu v strome scenárov na čas porušiť a meniť. Po vyriešení neprípustnosti sa smer pohybu vráti do posledného stavu pred jej detekciou.

Algoritmus uvedený v [2] si v prípade neprípustného riešenia po pridaní rezu prípustnosti nepamätá, odkiaľ bol rez pridávaný. Čiže sa vo všeobecnosti nevracia do uzla, z ktorého bol rez prípustnosti naposledy generovaný. Takto môže zbytočne traverzovať časť stromu scenárov od prípustnej úlohy kam bol naposledy pridaný rez prípustnosti až po uzol z ktorého bol tento rez naposledy generovaný. Tomuto problému sa dá vyhnúť implementáciou zásobníka[11]. Na koniec zásobníka vkladáme pozíciu uzla, odkiaľ generujeme rez prípustnosti, a tiež smer, ktorým sa aktuálne pohybujeme. Smer môže byť buď dopredný, kedy algoritmus postupuje od koreňa stromu scenárov k listom alebo spätný, čiže postup od listov ku koreňu (presný význam bude jasnejší z iteračnej schémy algoritmu). Pri následnom dosiahnutí prípustnosi sa pozrieme najprv na zásobník, pričom ak je neprázdny, vyberieme uzol a smer z jeho konca.

5.2 Vnorená L-Shaped metóda s adaptívnou agregáciou rezov optimality

Nasledovný algoritmus možno čiastočne považovať za vlastný prínos tejto práce. Pri jeho návrhu sme vychádzali z vnorenej L-Shaped metódy z časti 5.1, ktorú sme rozšírili o princípy L-Shaped metódy s adaptívnou agregáciou rezov optimality pre dvojstupňové úlohy (časť 3.3).

Pred formuláciou tohto algoritmu je potrebné objasniť niektoré špecifiká mnohostupňových SLÚ a vnorenej L-Shaped metódy, ktoré majú kľúčový význam pre pochopenie jeho princípu. Strom scenárov mnohostupňovej SLÚ obsahuje tri rozdielne typy uzlov, ktoré môžno pomenovať *koreň*, *uzly strednej vrstvy* a *listy*. Sekvenčný protokol vnorenej L-Shaped metódy prechádza postupne jednotlivé uzly stromu sce-

[11] Štandardne používaná dátová štruktúra. Možno ho implementovať napríklad ako pole, pričom dáta vždy ukladáme na koniec a vyberáme z konca.

nárov a rieši k nim pridružené hlavné úlohy $NLDS(t,k)$. Špecificky uzly strednej vrstvy sú vyberané dva razy - pri ceste vpred a následne pri ceste vzad, čiže aj $NLDS(t,k)$ pre $t = 2, ..., H-1$ je riešená vždy pre dopredný smer a spätný smer. Koreň a listy sú vyberané iba raz a to v doprednom smere. V prípade neprípustnosti v niektorom uzle môže nastávať ďalšie cyklenie, ktoré situáciu komplikuje, avšak pre naše potreby postačí takto zjednodušená predstava. Pre zostavenie algoritmu je potrebné definovať v každom uzle po stupeň $H-1$ akúsi analógiu hlavnej iterácie, ktorú sme mali v prípade L-Shaped metódy pre dvojstupňové SLÚ. Tam bol počet hlavných iterácií definovaný v koreňovom uzle ako počet riešení hlavnej úlohy. V našom prípade sa ukázalo ako výhodné považovať za hlavnú iteráciu pre daný uzol práve jeho výber sekvenčným protokolom v doprednom smere, čiže riešenie hlavnej úlohy k nemu pridruženej pri pohybe v strome scenárov smerom od koreňa k listom. Jedna z výhod takéhoto prístupu spočíva v tom, že umožňuje definovať hlavnú iteráciu pre všetky uzly stromu rovnako. Významný je tiež fakt, že každému výberu uzla v doprednom smere možno za predpokladu prípustnosti priradiť práve jedno pridávanie rezov optimality do daného uzla počas spätného pohybu sekvenčného protokolu od listov ku koreňu. To je analogické s dvojstupňovým prípadom, kedy v každej makroiterácii riešime hlavnú úlohu v koreňovom uzle a za predpokladu prípustnosti práve raz pridávame rezy optimality. Po týchto úvodných myšlienkach by už mal byť princíp algoritmu zreteľnejší, pristúpime teda k jeho formalizácii.

Uvažujme množinu $\mathcal{D}_k^t$ nasledovníckych uzlov pre uzol $k \in \{1, ..., K^t\}$, ktorý sa nachádza v stupni $t \in \{1, ..., H-1\}$ stromu scenárov. Túto vieme rozčleniť do $L_k^t \leq |\mathcal{D}_k^t|$ skupín $\mathcal{D}_{k,1}^t, ..., \mathcal{D}_{k,L_k^t}^t$ tak, že platí $\mathcal{D}_k^t = \mathcal{D}_{k,1}^t \cup \mathcal{D}_{k,2}^t \cup ... \mathcal{D}_{k,L_k^t}^t$, pričom $\mathcal{D}_{k,i}^t \cap \mathcal{D}_{k,j}^t = \emptyset$ pre všetky $i \neq j$. Označme ν_k^t aktuálny počet dopredných výberov uzla k v stupni t sekvenčným protokolom. To znamená - koľký raz algoritmus rieši pre tento uzol hlavnú úlohu[12] pri smere pohybu od koreňa k listom stromu. Pre zjednodušenie značenia ďalej zaveďme nasledovnú konvenciu. Ak máme ľubovoľný zápis (napríklad funkciu) v tvare $X_{i_1,..,i_n}^j(..;\nu;..)$, potom označenie ν predstavuje zjednodušený zápis pre $\nu_{i_1}^j$.

Teraz vieme zadefinovať zoskupenie nasledovníckych uzlov pre ν_k^t-ty dopredný výber ako $D_k^t(\nu) = \{D_{k,1}^t(\nu), ..., D_{k,L}^t(\nu)\}$, kde L sme označili $L = L_k^t(\nu)$ pre zjednodušenie zápisu, pričom platí $D_{k,i}^t(\nu) \cap D_{k,j}^t(\nu) = \emptyset$ pre všetky $i \neq j$ a $\bigcup_{i=1}^L D_{k,i}^t(\nu) = \mathcal{D}_k^t$. Pre pravdepodobnosti skupín uzlov $D_{k,i}^t(\nu)$ v strome scenárov platí $p_{k,D_{k,i}^t(\nu)}^t = \sum_{d \in D_{k,i}^t(\nu)} p_{k,d}^t$, kde $p_{k,d}^t$ je pravdepodobnosť nasledovníka d uzla k v stupni t, a $\sum_{i=1}^L p_{k,D_{k,i}^t(\nu)}^t = \sum_{d \in D_k^t} p_{k,d}^t = 1$.

Zaveďme indexovú množinu pre skupiny nasledovníckych uzlov pre uzol k v stupni t ako $\Lambda_k^t(\nu) = \{1, ..., L\}$ a funkciu vracajúcu indexy pre prvky $d \in D_k^t(\nu)$:

$$\lambda_k^t(d;\nu) : D_k^t(\nu) \to \Lambda_k^t(\nu) \text{ takú, že } \lambda_k^t(D_{k,j}^t(\nu);\nu) = j.$$

Kvoli zjednodušeniu značenia ďalej použijeme nasledujúcu konvenciu. V ľubovoľnom zápise tvaru $Y_{i,..,\lambda(.;.),..}^j(..;..)$, označenie $\lambda(.;.)$ predstavuje zjednodušený zápis

[12] Táto bude formálne definovaná neskôr.

pre $\lambda_i^j(.;.)$.

Pre každý stupeň $t = 1, ..., H-1$ a každý uzol $k = 1, ..., K^t$ v stupni t vieme definovať *hlavnú úlohu* pre jeho dopredný výber $\nu = \nu_k^t$ v tvare

$$\min_{\boldsymbol{x}_k^t, \{\theta_{k,l}^t\}_{l \in \Lambda_k^t(\nu)}} \quad (\boldsymbol{c}_k^t)^\top \boldsymbol{x}_k^t + \sum_{l \in \Lambda_k^t(\nu)} \theta_{k,l}^t \tag{5.9}$$

$$W^t \boldsymbol{x}_k^t = \boldsymbol{h}_k^t - T_k^{t-1} \boldsymbol{x}_{a(k)}^{t-1}, \tag{5.10}$$

$$\boldsymbol{G}_{k,j}^t \boldsymbol{x}_k^t \geq g_{k,j}^t, \ j = 1, ..., r_k^t(\nu), \tag{5.11}$$

$$\boldsymbol{E}_{k,l,j}^t \boldsymbol{x}_k^t + \theta_{k,l}^t \geq e_{k,l,j}^t, \ j = 1, ..., s_k^t(\nu), \ l \in \Lambda_k^t(\nu), \tag{5.12}$$

$$\boldsymbol{x}_k^t \geq 0, \tag{5.13}$$

kde $a(k)$ je predchodca *uzla* k, $\boldsymbol{x}_{a(k)}^{t-1}$ je priebežné riešenie v uzle $a(k)$ v stupni $t-1$, $r_k^t(\nu)$ je počet rezov prípustnosti (5.11) pridaných po dopredný výber $\nu = \nu_k^t$ pre uzol k v stupni t a $s_k^t(\nu)$ je počet dopredných výberov uzla k v stupni t, v ktorých boli pridávané rezy optimality (5.12). Pre $t = 1$, $\boldsymbol{b} \triangleq \boldsymbol{h}^1 - T^0 \boldsymbol{x}^0$ predstavuje počiatočné podmienky úlohy. V poslednom stupni H nie sú prítomné $\theta_{k,l}^H$ ani ohraničenia (5.11)–(5.12). Takúto hlavnú úlohu pre daný stupeň t a uzol k budeme označovať ako $NLDSA(t,k)$. Pre zjednodušenie zápisu vzťahov v algoritme bude užitočné zaviesť ďalej maticu $\boldsymbol{G}_k^t \triangleq ((\boldsymbol{G}_{k,1}^t)^\top, ..., (\boldsymbol{G}_{k,r_k^t(\nu)}^t)^\top)^\top$ a vektor $\boldsymbol{g}_k^t \triangleq (g_{k,1}^t, ..., g_{k,r_k^t(\nu)}^t)^\top$.

5.2.1 Algoritmus

Krok 0: Inicializácia.
Pre všetky $t \in \{1, ..., H-1\}$ a $k \in \{1, ..., K^t\}$ polož $\nu_k^t = 0$, $s_k^t(0) = 0$, $r_k^t(0) = 0$, inicializuj $D_k^t(0)$ na začiatočné stavy. Pridaj ohraničenia $\theta_{k,l}^t = 0$ k (5.9)–(5.13) pre všetky t, k a $l \in \Lambda_k^t(0)$. Polož $SMER = VPRED$ a nastav $t = 1$, $k = 1$.

Krok 1: Riešenie hlavnej úlohy.
Rieš súčasnú úlohu $NLDSA(t,k)$. Ak $SMER = VPRED$, nastav $\nu_k^t = \nu_k^t + 1$, $D_k^t(\nu) = D_k^t(\nu-1)$, $r_k^t(\nu) = r_k^t(\nu-1)$, $s_k^t(\nu) = s_k^t(\nu-1)$[13].
∘ Ak je neprípustná:

∘ ak $t = 1$, koniec, celá úloha (4.2)–(4.4) je neprípustná.

∘ ak $t > 1$, nech podmienka neprípustnosti je daná duálnym riešením $\boldsymbol{\pi}_k^t, \boldsymbol{\rho}_k^t \geq \boldsymbol{0}$, takým, že $(\boldsymbol{\pi}_k^t)^\top W^t + (\boldsymbol{\rho}_k^t)^\top \boldsymbol{G}_k^t \leq \boldsymbol{0}$, ale $(\boldsymbol{\pi}_k^t)^\top (\boldsymbol{h}_k^t - T_k^{t-1} \boldsymbol{x}_{a(k)}^{t-1}) + (\boldsymbol{\rho}_k^t)^\top \boldsymbol{g}_k^t > 0$. Označme $r = r_{a(k)}^{t-1}(\nu)$. Nastav $r = r + 1$. Vypočítaj koeficienty rezov prípustnosti (5.11) ako

$$\boldsymbol{G}_{a(k),r}^{t-1} = (\boldsymbol{\pi}_k^t)^\top T_k^{t-1}, \tag{5.14}$$

$$g_{a(k),r}^{t-1} = (\boldsymbol{\pi}_k^t)^\top \boldsymbol{h}_k^t + (\boldsymbol{\rho}_k^t)^\top \boldsymbol{g}_k^t. \tag{5.15}$$

[13] $D_k^t(\nu)$, $s_k^t(\nu)$ a $r_k^t(\nu)$ sa skopírujú pre nový výber uzla $\nu = \nu_k^t$, aby boli k dispozícii ich aktuálne stavy po najbližšiu zmenu agregácie resp. pridávanie rezov optimality.

Pridaj rez prípustnosti s (5.14), (5.15) do $NLDSA(t-1, a(k))$. Ulož t, k a *SMER* do zásobníka typu LIFO. Nastav $SMER = VZAD$. Polož $t = t-1$, $k = a(k)$ a vráť sa na Krok 1.

∘ Ak je prípustná:
Máme riešenie $(\boldsymbol{x}^t_{k,\nu}, \{\theta^t_{k,l,\nu}\}_{l\in\Lambda^t_k(\nu)})$[14]. Aktualizuj hodnoty $\boldsymbol{x}^t_k = \boldsymbol{x}^t_{k,\nu}$, $\{\theta^t_{k,l}\}_{l\in\Lambda^t_k(\nu)} = \{\theta^t_{k,l,\nu}\}_{l\in\Lambda^t_k(\nu)}$. Ulož hodnoty duálnych riešení pre ohraničenia (5.10)–(5.12) ako $\boldsymbol{\pi}^t_k \in \mathbb{R}^{m_t}$, $\boldsymbol{\rho}^t_k \in \mathbb{R}^r$ a $\boldsymbol{\sigma}^t_k \in \mathbb{R}^{r\times L}$, kde $r = r^t_k(\nu)$.

- ∘ Ak je zásobník neprázdny, vyber z konca zásobníka naposledy uložené t, k a *SMER*. Pokračuj na Krok 2.
- ∘ Inak
  - ∘ ak $k < K^t$, nastav $k = k+1$, vráť sa na Krok 1.
  - ∘ ak $k = K^t$,
    - ∘ ak $SMER = VPRED$ a $t < H$, nastav $t = t+1$, $k = 1$ a vráť sa na Krok 1.
    - ∘ ak $t = H$, nastav $SMER = VZAD$. Pokračuj na Krok 2.

Krok 2: Pre všetky uzly $k = 1, ..., K^{t-1}$ v stupni $t-1$ vykonaj:
Krok 2a: Agregácia skupín uzlov.
Vytvor nový agregát $D^{t-1}_k(\nu)$ z $D^{t-1}_k(\nu-1)$ podľa pravidiel agregácie (s využitím informácie v $\boldsymbol{\sigma}^{t-1}_k$). Každý prvok $D^{t-1}_k(\nu)$ je zjednotením nejakých prvkov z $D^{t-1}_k(\nu-1)$. Nech $d_1, d_2, ..., d_m \in D^{t-1}_k(\nu-1)$, kde $m \le L(\nu-1, k, t-1)$, sú nejaké skupiny uzlov. Ak sa podľa agregačných pravidiel $d_1, d_2, ..., d_m$ zagregujú do $d \in D^{t-1}_k(\nu)$, potom $d = \bigcup_{i=1}^m d_i$ a $p^t_{k,d} = \sum_{i=1}^m p^t_{k,d_i}$. Z hlavnej úlohy $NLDSA(t-1,k)$ odstráň premenné $\theta^{t-1}_{k,\lambda(d_1;\nu-1)}, ..., \theta^{t-1}_{k,\lambda(d_m;\nu-1)}$ a pridaj namiesto nich novú premennú $\theta^{t-1}_{k,\lambda(d;\nu)}$.

Krok 2b: Aktualizácia rezov optimality.
Pre všetky dopredné výbery uzla k v stupni $t-1$, v ktorých boli pridané *rezy optimality*, tj. pre všetky $j \in \{1, ..., s^{t-1}_k(\nu)\}$, definuj

$$e^{t-1}_{k,\lambda(d;\nu),j} = \sum_{i=1}^m e^{t-1}_{k,\lambda(d_i;\nu-1),j} \tag{5.16}$$

a

$$\boldsymbol{E}^{t-1}_{k,\lambda(d;\nu),j} = \sum_{i=1}^m \boldsymbol{E}^{t-1}_{k,\lambda(d_i;\nu-1),j}. \tag{5.17}$$

Pre všetky $j \in \{1, ..., s^{t-1}_k(\nu)\}$ nahraď v $NLDSA(t-1,k)$ rezy prislúchajúce k $d_1, ..., d_m \in D^{t-1}_k(\nu-1)$ novým rezom

$$\boldsymbol{E}^{t-1}_{k,\lambda(d;\nu),j}\boldsymbol{x}^{t-1}_k + \theta^{t-1}_{k,\lambda(d;\nu)} \ge e^{t-1}_{k,\lambda(d;\nu),j} \tag{5.18}$$

[14]Index ν znamená, že sa jedná o riešenie s daným poradovým číslom.

prislúchajúcim k $d \in D_k^{t-1}(\nu)$. Hodnoty $\theta^{t-1}_{k,\lambda(d_1;\,\nu-1)}, ..., \theta^{t-1}_{k,\,\lambda(d_m;\,\nu-1)}$ nahraď ich súčtom, ulož novú hodnotu $\theta^{t-1}_{k,\lambda(d;\,\nu)} = \sum_{i=1}^{m} \theta^{t-1}_{k,\lambda(d_i;\,\nu-1)}$.

Krok 2c: Výpočet nových rezov optimality.
Nasledujúce vzťahy (5.19) a (5.20) slúžia na výpočet koeficientov rezov optimality pre aktuálne zoskupenie nasledovníckych uzlov $D_k^{t-1}(\nu)$ prislúchajúce uzlu k v stupni $t-1$. Polož:

$$\boldsymbol{E}^{t-1}_{k,l,n} = \sum_{s \in D} \frac{p^t_s}{p^{t-1}_k} (\boldsymbol{\pi}^t_s)^\top T^{t-1}_s \quad \forall l \in \Lambda^{t-1}_k(\nu) \tag{5.19}$$

a

$$e^{t-1}_{k,l,n} = \sum_{s \in D} \frac{p^t_s}{p^{t-1}_k} \left[(\boldsymbol{\pi}^t_s)^\top \boldsymbol{h}^t_s + \sum_{i=1}^{r^t_s(\nu)} (\rho^t_{s,i})^\top g^t_{s,i} + \sum_{j=1}^{s^t_s(\nu)} \sum_{i \in \Lambda^t_s(\nu)} (\sigma^t_{s,i,j})^\top e^t_{s,i,j} \right] \quad \forall l \in \Lambda^{t-1}_k(\nu), \tag{5.20}$$

kde $n = s^{t-1}_k(\nu) + 1$, $D = D^{t-1}_{k,l}(\nu)$, $\rho^t_{s,i}$ predstavuje i-tu zložku vektora $\boldsymbol{\rho}^t_s$ a $\sigma^t_{s,i,j}$ predstavuje (i,j)-ty prvok matice $\boldsymbol{\sigma}^t_s$.

Súčasná podmienená stredná hodnota účelových funkcií podúloh pre l-ty agregát je potom $\overline{\theta}^{t-1}_{k,l} = e^{t-1}_{k,l,n} - \boldsymbol{E}^{t-1}_{k,l,n} \boldsymbol{x}^{t-1}_k$, kde $l \in \Lambda^{t-1}_k(\nu)$.
∘ Ak sa v $NLDSA(t-1,k)$ nachádzajú počiatočné ohraničenia $\theta^{t-1}_{k,l} = 0$, odstráň ich, pridaj namiesto nich ohraničenia (5.12) s koeficientami (5.19) a (5.20), aktualizuj $s^{t-1}_k(\nu) = s^{t-1}_k(\nu) + 1$.
∘ Inak, ak pre nejaké $l \in \Lambda^{t-1}_k(\nu)$ platí $\overline{\theta}^{t-1}_{k,l} > \theta^{t-1}_{k,l}$, pridaj pre tieto l ohraničenia (5.12) s koeficientami (5.19) a (5.20), aktualizuj $s^{t-1}_k(\nu) = s^{t-1}_k(\nu) + 1$.

Krok 3
Ak $t = 2$ a do $NLDSA(1)$ neboli pridané žiadne ohraničenia, koniec, $\boldsymbol{x}^1_1$ je optimum, inak polož $t = t-1$, $k = 1$. Ak $t = 1$, nastav $SMER = VPRED$. Choď na Krok 1.

5.2.2 Pravidlá agregácie

Nasledovné pravidlá agregácie vychádzajú z pravidiel uvedených pre dvojstupňovú L-Shaped metódu s adaptívnou agregáciou rezov optimality.

- *Hladina nadbytočnosti δ.*
  Uvažujme nejaký uzol k v stupni $t \in \{1, ..., H-1\}$ stromu scenárov a jeho $\nu = \nu^t_k$ dopredný výber sekvenčným protokolom. Majme nejakú skupinu nasledovníckych uzlov $d \in D^t_k(\nu)$.

– a) Podľa [11] by toto agregačné pravidlo vychádzalo nasledovne. Nech $f^t_{k,d}$ predstavuje počet dopredných výberov, kedy všetky rezy optimality prislúchajúce k d boli nadbytočné (nulové duálne premenné). Formálne to možno zapísať nasledovne:

$$f^t_{k,d} := \sum_{i=1}^{v^t_k} \chi^t_k(i;d),$$

kde

$$\chi^t_k(i;d) = \begin{cases} 1 & \text{ak } \sigma^t_{k,\lambda(d;i),j} = 0 \text{ pre všetky } j \in \{1,...,s^t_k(i)\}, \\ 0 & \text{inak.} \end{cases}$$

– b) Nakoľko spôsob a) sa ukázal byť príliš reštriktívny pre nami riešené úlohy (nedochádzalo k agregácii), zaviedli sme mierne modifikovanú verziu:

$$f^t_{k,d} := \sum_{i=1}^{v^t_k} \xi^t_k(i;d), \tag{5.21}$$

kde

$$\xi^t_k(i;d) = \frac{\sum_{j=1}^{s^t_k(i)} \bar{\chi}^t_k(j;d)}{s^t_k(i)}, \tag{5.22}$$

$$\bar{\chi}^t_k(j;d) = \begin{cases} 1 & \text{ak } \sigma^t_{k,\lambda(d;i),j} = 0, \\ 0 & \text{inak.} \end{cases}$$

To nie je nič iné, než upravený spôsob a) s tým rozdielom, že ak nie sú všetky duálne premenné pre rezy prislúchajúce danej skupine $d \in D^t_k(\nu)$ v rámci iterácie nulové, nezapočíta sa 0 ako v prípade a), ale pomer nulových premenných k ich celkovému počtu v danej iterácii. Čiže namiesto binárnej charakteristickej funkcie $\chi^t_k(i;d)$ z a) máme funkciu $\xi^t_k(i;d)$ vracajúcu reálne čísla z intervalu (0,1). V experimentoch totiž nastávali prípady, kde bolo veľa nulových rezov v rámci iterácie, no neboli nulové všetky, čo spôsobilo nepoužiteľnosť pravidla a).

Potom podľa verzie a), prípadne b) tohto pravidla budeme zlučovať všetky skupiny d, pre ktoré platí $f^t_{k,d}/s^t_k(\nu) > \delta$, kde $\delta \in (0,1)$ je *hladina nadbytočnosti*, ktorú si určíme.

- *Maximálny počet zoskupených uzlov* (AggMax).
Pre zamedzenie príliš rýchleho zoskupenia všetkých uzlov možno použiť ohraničenie, ktoré udáva, koľko najviac uzlov je možné zlúčiť do jednej skupiny, čiže určuje maximálnu veľkosť skupiny.

6 Numerický experiment

Hlavným cieľom numerického experimentu bude porovnať na viacerých štandardných úlohách z literatúry efektívnosť naprogramovaných algoritmov. Konkrétne pôjde o porovnanie vnorenej L-Shaped metódy, viacrezovej vnorenej L-Shaped metódy a vnorenej L-Shaped metódy s adaptívnou agregáciou rezov optimality na riešenie viac-stupňových SLÚ. Efektívnosť algoritmov budeme vyhodnocovať na základe systémového času a počtu makroiterácií potrebných na vyriešenie jednotlivých úloh. Ako vedľajší cieľ si stanovujeme porovnať na viacerých štandardných úlohách, v ktorých nastáva neprípustnosť, algoritmus vnorenej L-Shaped metódy podľa literatúry [2] a jeho mierne upravenú verziu so zásobníkom, ktorú v tejto práci uvádzame ako Algoritmus 5.1.1.

6.1 Riešené úlohy

V tejto časti stručne predstavíme úlohy, ktorých instancie s rozličným počtom scenárov budeme riešiť v numerickom experimente jednotlivými algoritmami. Jedná sa o úlohy z rozmanitých oblastí aplikácie, ktoré sú v prevažnej väčšine prípadov popísané v zbierkach úloh [1] a [6].

6.1.1 Pltexp

Popis úlohy pltexp je prevzatý z [6]. Jedná sa o stochastický model rozširovania produkčnej kapacity. Cieľom je priradiť nové produkčné kapacity pre sériu tovární, tak aby sa maximalizoval profit pri neistom dopyte.

Predpokladajme výrobný systém s I továrňami a J produktmi. Každá továreň $i \in \{1, ..., I\}$ má výrobnú kapacitu $REGCAP_i$ a kapacitu pre nadčasy $OTCAP_i$. Kapacita pre daný produkt j v rámci každej továrne je limitovaná na $\alpha_{i,j}$ percent celkovej kapacity. Nech cena kapitálu pre produkt j v továrni i je CC_{ij}, prevádzkové náklady sú RG_{ij} a nadčasové prevádzkové náklady OT_{ij}. Predpokladajme amortizačný faktor β a kapitál $BUDG$. Ďalej predpokladajme diskrétne rozdelený dopyt po tovare j s prvkami (d_{js}, p_s), $s = 1, ..., S$, pričom predaná jednotka prináša REV_j dolárov.

Označme:
$y_{ijt} = 1$ ak je produkt j vyrobený v továrni i počas periódy t, 0 inak,
r_{ijt} = pravidelná produkcia výrobku j v továrni i počas periódy t,
o_{ijt} = nadčasová produkcia výrobku j v továrni i počas periódy t,
$\delta^+_{ijt}, \delta^-_{ijt}$ = zmena v celkovej produkcii výrobku i z továrne j počas periódy t.
SC_{ij} = náklady na odstránenie výroby produktu i z továrne j. Formulácia viacstup-

ňovej lineárnej verzie úlohy je potom nasledovná

$$\max_{\{y_{ij1},r_{ijt},o_{ijt},\delta^+_{ijt},\delta^-_{ijt}\}} \sum_i \sum_j -\beta CC_{ij}y_{ij1} + (REV_j - RG_{ij})r_{ij1} + (REV_j - OT_{ij})o_{ij1} +$$
$$\sum_{t>1}\sum_i\sum_j -\beta CC_{ij}\delta^+_{ijt} + SC_{ij}\delta^-_{ijt} + (REV_j - RG_{ij})r_{ijt} + (REV_j - OT_{ij})o_{ijt} \tag{6.1}$$

$$\begin{aligned}
\delta^+_{ijt} - \delta^-_{ijt} - y_{ijt} - y_{ij(t-1)} = 0,\ i = 1,...,I,\ j = 1,...,J,\ t > 1,\\
\sum_i r_{ijt} + o_{ijt} \le d_{jt},\ j = 1,...,J,\ t = 1,...,T,\\
\alpha_{ij}(r_{ijt} + o_{ijt}) \le y_{ijt}(REGCAP_i + OTCAP_i),\\
i = 1,...,I,\ j = 1,...,J,\ t = 1,...,T,\\
\sum_j \alpha_{ij} r_{ijt} \le REGCAP_i,\ i = 1,...,I,\ t = 1,...,T,\\
\sum_j \alpha_{ij} o_{ijt} \le OTCAP_i,\ i = 1,...,I,\ t = 1,...,T,\\
\sum_i\sum_j CC_{ij}y_{ij1} \le BUDG_1, \sum_i\sum_j CC_{ij}\delta^+_{ijt} \le BUDG_t,\ t = 1,...,T,\\
y_{ijt} \in \{0,1\},\ r_{ijt},\ o_{ijt},\ \delta^+_{ijt}.\delta^-_{ijt} \ge 0,\ i = 1,...,I,\ j = 1,...,J,\ t = 1,...,T.
\end{aligned}$$

V numerickom experimente boli použité dostupné trojstupňové a štvorstupňové instancie tejto úlohy, ktoré sú súčasťou balíka SLÚ nazývaného POSTS.

6.1.2 Bonds

V tomto prípade sa jedná o úlohu investícií do dlhopisov, ktorej popis sme prevzali z [1]. Predpokladajme, že dlhopisy maturujú v určitých štandardných časových periódach z množiny $\mathcal{D}^S$, pričom najdlhší čas maturity nech je D. V modeli budeme mať dlhopisy maturujúce v $d \in \mathcal{D} = \{1, 2, ..., D\}$ mesiacoch.

Nech $v_t^{d,+}$ je hodnota nových pôžičiek, ktoré si subjekt berie v čase t s maturitou $d \in \mathcal{D}^S$ a $v_t^{d,-}$ je hodnota peňazí, ktoré naopak subjekt požičiava. Potom, ak v_t^d je hodnota dlhopisových transakcií v čase t s maturitou d, máme

$$v_t^d = \begin{cases} v_{t-1}^{d+1} + v_t^{d,+} - v_t^{d,-} & \text{ak } d \in \mathcal{D}^S \\ v_{t-1}^{d+1} & \text{ak } d \in \mathcal{D}\backslash\mathcal{D}^S \end{cases}$$

Celková hodnota dlhopisových transakcií v čase t je

$$x_t = \sum_{d\in\mathcal{D}} v_t^d. \tag{6.2}$$

Ak je táto hodnota pozitívna, bude použitá na financovanie obchodného rizika počas periódy t. Premenná x_t sa zvykne zapisovať ako stochastický proces

$$x_t = x_{t-1} + \xi_t, \tag{6.3}$$

kde ξ_t je náhodná premenná. Na obmedzenie predaja dlhopisov sa zavádza ohraničenie

$$\sum_{d\in\mathcal{D}^S} v_t^{d,+} - \sum_{d=1}^{M} v_{t-1}^{d} \leq \xi_t. \tag{6.4}$$

Pri daných náhodných úrovniach výnosov $\eta_t^{d,-}, \eta_t^{d,+}$ a η_t^x prislúchajúcim k $v_t^{d,-}, v_t^{d,}$ a x_t chceme maximalizovať očakávaný výnos

$$E_{\eta,\xi}\left\{\sum_{t=1}^{T}\left(\sum_{d\in\mathcal{D}^S}\left[-\eta_t^{d,-}v_t^{d,-} + \eta_t^{d,+}v_t^{d,+}\right] - \eta_t^x x_t\right)\right\} \tag{6.5}$$

V nasledovnej formulácii sa úloha mení na minimalizačnú, prvostupňové premenné a ohraničenia sú oddelené od druhostupňových.

$$\min_{\{v_t^{d,-},v_t^{d,+},x_t\}_{t=0}^{T}} \sum_{d\in\mathcal{D}^S}\left[-\eta_0^{d,-}v_0^{d,-} + \eta_0^{d,+}v_0^{d,+}\right] - \eta_0^x x_0 + \tag{6.6}$$
$$E_{\eta,\xi}\left\{\sum_{t=1}^{T}\left(\sum_{d\in\mathcal{D}^S}\left[-\eta_t^{d,-}v_t^{d,-} + \eta_t^{d,+}v_t^{d,+}\right] - \eta_t^x x_t\right)\right\}$$

$$\begin{aligned}
v_0^d - v_{-1}^{d+1} - v_0^{d,+} + v_0^{d,-} &= 0,\ \forall d \in \mathcal{D}^S,\\
v_0^d - v_{-1}^{d+1} &= 0,\ \forall d \in \mathcal{D}\backslash\mathcal{D}^S,\\
x_0 - \sum_{d\in\mathcal{D}} v_0^d &= 0,\\
x_0 - x_{-1} &= \xi_0,\\
\sum_{d\in\mathcal{D}^S} v_0^{d,0} - \sum_{d=1}^{M} v_{-1}^d &\leq \xi_0,\\
v_0^{d,+}, v_0^{d,-} &\geq 0,\ \forall d \in \mathcal{D},\\
v_t^d - v_{t-1}^{d+1} - v_t^{d,+} + v_t^{d,-} &= 0,\ \forall d \in \mathcal{D}^S, t = 1,...,T,\\
v_t^d - v_{t-1}^{d+1} &= 0,\ \forall d \in \mathcal{D}\backslash\mathcal{D}^S, t = 1,...,T,\\
x_t - \sum_{d\in\mathcal{D}} v_t^d &= 0,\ \forall t = 1,...,T,\\
x_t - x_{t-1} &= \xi_t,\ \forall t = 1,...,T,\\
\sum_{d\in\mathcal{D}^S} v_t^{d,+} - \sum_{d=1}^{M} v_{t-1}^d &\leq \xi_t,\ \forall t = 1,...,T,\\
v_t^{d,+}, v_t^{d,-} &\geq 0,\ \forall d \in \mathcal{D}, t = 1,...,T.
\end{aligned}$$

Pre čas $t = -1$ máme dané hodnoty všetkých rozhodovacích premenných a v čase $t = 0$ zas všetky hodnoty η a ξ.

6.1.3 Scfxm

Podľa dostupných informácií je SCFXM reálna úloha plánovania výroby, ktorej presný pôvod je neznámy. Referenciu na úlohu možno nájsť v [6]. Nepodarilo sa nám zistiť presnejšiu špecifikáciu ani formuláciu SCFXM. Podľa [6] bola úloha pôvodne formulovaná ako dvojstupňová, neskôr bola rozšírená a vznikli viacstupňové verzie.

6.1.4 Scsd

SCSD predstavuje viacstupňový model, ktorého popis je uvedený v [6]. Úlohou je nájsť systém nosníkov s minimálnou hmotnosťou. Sú dané body spojov v rovine a kritická hmotnosť, ktorú tento systém musí uniesť. Štruktúra je tvorená tyčami, ktoré sa umiestňujú medzi dvojice spojov, pričom sú dané ohraničenia pre maximálne zataženie jednotlivých nosníkových tyčí.

V [6] je uvedená nasledovná formulácia dvojstupňovej úlohy, ktorú možno priamočiaro rozšíriť na viacstupňovú.

Predpokladajme, že $\boldsymbol{P}_i = (P_{ix}, P_{iy})$ sú horizontálne a vertikálne sily pôsobiace v každom z N možných spojov v priestore. Existuje $K = (N-1)N/2$ možných umiestnení nosníkov. Nech δ_k a γ_k sú kosínusy uhlov, ktoré zvierajú jednotlivé nosníky $k = 1, ..., K$ s rovinami x a y. Premenné u_k predstavujú sily napínajúce jednotlivé nosníky a s_k predstavujú plochy ich priečnych rezov. Hľadáme štruktúru s minimálnou vlastnou hmotnosťou schopnú uniesť požadované zaťaženie. V deterministickom modely sa vychádza z predpokladu, že v každom optimálnom riešení sa dosiahne maximálne pnutie, čiže $|u_k| = \sigma s_k$, kde σ je hraničný koeficient pnutia materiálu, pri ktorom sa začína deformovať. Účelová funkcia predstavuje súčin hustoty materiálu ρ, plochy prierezu a dĺžky jednotlivých nosníkov L_k. Deterministická formulácia vyzerá nasledovne:

$$\min_{\{u_k^+, u_k^-\}_{k=1}^K} \quad W = \frac{\rho}{\sigma} \sum_{k=1}^{K} L_k (u_k^+ - u_k^-) \tag{6.7}$$

pri ohraničeniach

$$\sum_{k=1}^{K} a_{ik}^1 (u_k^+ - u_k^-) = P_{ix}, i = 1, ..., N, \tag{6.8}$$

$$\sum_{k=1}^{K} a_{ik}^2 (u_k^+ - u_k^-) = P_{iy}, i = 1, ..., N, \tag{6.9}$$

$$u_k^+,\ u_k^- \geq 0, \tag{6.10}$$

kde $a^1_{ik} = \pm\delta_k$ a $a^2_{ik} = \pm\gamma_k$ ak tyč k vchádza, respektíve vychádza zo spoja i, vo všetkých ostatných prípadoch sú to nuly. P_{ix} a P_{iy} predstavujú množinu $2N$ externých síl na nosníky. V stochastickej verzii sú potom hodnoty týchto externých síl a prípadne tiež hmotnosti tyčí náhodné.

6.1.5 Numerické instancie úloh

Instancie úloh PLTEXP, SCFXM a BONDS boli získané zo série štandardných úloh POSTS, ktorá je dostupná na internete.[15] Pre úlohu SCSD8 sme vytvorili instancie prispôsobením počtu scenárov a stupňov, pričom dáta pre ich výrobu boli získané prostredníctvom emailovej korešpondencie od Prof. J. Birgeho.

Úloha	**H**	**U/ST**	**K**	**R**	**S**	**fv**
pltexpA4.6	4	6	216	26894	71912	19.599
pltexpA4.16	4	16	4096	454334	1214492	-18.849
pltexpB4.6	4	6+2+2	24	4430	13160	-17.928
pltexpB5.6	5	6+2+2+2	48	9422	26216	-23.846
bonds3.5	3	5	25	1282	1342	-3027.706
scfxm3.6	3	6	36	4020	5295	18615.932
scfxm3.16	3	16	256	24720	32435	18438.891
scfxm4.6	4	6	216	23460	30783	18616.224
scfxm4.16	4	16	4096	393360	515763	18438.891
scsd3.27	3	27	729	15130	105910	30
scsd3.64	3	64	4096	83210	582470	32.5
scsd4.8	4	8	512	11690	81830	51.5
scsd4.27	4	27	19683	408790	2861530	48

Tabuľka 1: Riešené instancie úloh.

Označenie stĺpcov v Tabuľke 1 má nasledovný význam: H je počet stupňov, U/ST je počet uzlov na stupeň (v prípade odlišností medzi stupňami je použitý symbol '+' a vypísané počty uzlov pre každý stupeň), K je celkový počet scenárov, R a S predstavujú riadkový, respektíve stĺpcový rozmer deterministického ekvivalentu, fv je funkčná hodnota účelovej funkcie v optime.

[15] Adresa pre POSTS: http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html.

6.2 Experiment

6.2.1 Systém a nastavenia

Celý numerický experiment bol vykonaný na počítači Intel Core 2 Duo E4500, 2.2GHZ, 2GB RAM s operačným systémom Windows XP Professional. Algoritmy boli naprogramované a testované v prostredí MATLAB R2007b s nainštalovaným toolboxom MPT ('Multi-Parametric Toolbox'). Z MPT bol využitý lineárny solver GLPK, ktorý je založený na primárno-duálnej metóde vnútorného bodu. Ukázal sa byť spoľahlivejší a podstatne rýchlejší než simplexová metóda a metóda vnútorného bodu (Mehrotra predictor-corrector), ktoré sú štandardne implementované v MATLABe. Solver GLPK sme používali na riešenie všetkých lineárnych úloh, čiže v našom prípade hlavných úloh $NLDS(t,k)$ a $NLDSA(t,k)$.

Všetky instancie úloh boli v experimente riešené 10-krát, pričom výsledný systémový čas bol získaný ako aritmetický priemer dosiahnutých časov. Absolútna presnosť lineárneho solvera bola ponechaná na východzej hodnote eps $= 1.0e-7$ pri dĺžke kroku $\delta = 1.0e-6$.

6.2.2 Numerické výsledky

Na testovanie viacrezovej vnorenej L-Shaped metódy bol použitý adaptívny algoritmus vnorenej L-Shaped metódy s deaktivovanou dynamickou agregáciou rezov (bol použitý ten istý programový kód). To znamená, že algoritmus bežal od začiatku do konca so štartovacou - nulovou úrovňou agregácie. V listingoch programov v prílohe preto nie je algoritmus viacrezovej vnorenej L-Shaped metódy zvlášť uvádzaný.

V Tabuľke 2 sú uvedené výsledky porovnania vnorenej L-Shaped metódy (VLSM) a viacrezovej vnorenej L-Shaped metódy (VLSM VR). Označenie stĺpcov v Tabuľke 2 je nasledovné: 'CPU' označuje systémový čas (v sekundách), 'it' počet makroiterácií, 'rp' počet pridaných rezov prípustnosti a 'ro' počet pridaných rezov optimality do vyriešenia úlohy. Stĺpec 'α' predstavuje porovnanie systémového času VLSM VR voči VLSM (hodnota 'CPU' pre VLSM je 100%), stĺpec 'agr.' objasníme neskôr. Vidno, že vo väčšine prípadov bola rýchlejšia VLSM. Viacrezová verzia bola výrazne rýchlejšia iba v prípade scsd3.27 a scsd4.27. Rýchlejšia bola tiež pri úlohách bonds3.5 a scsd4.8, tu však boli rozdiely systémového času voči VLSM takmer zanedbateľné.

Čo sa týka počtu makroiterácií, okrem dvoch výnimiek (scfxm3.6 a scfxm4.6) dosahovala nižší počet VLSM VR. Tento výsledok sa dá očakávať vzhľadom na lepšiu aproximáciu účelovej funkcie v prípade viacrezovej metódy. O tejto aproximácii pojednávajú napríklad autori v [3]. Odkazujeme tiež na časť 3.2 tejto práce (Viacrezová L-Shaped metóda), kde sú tieto výsledky pre dvojstupňové úlohy spomínané. V prípade scfxm3.6 a scfxm4.6, v ktorých počet makroiterácií vo VLSM VR bol vyšší než pri VLSM, bola situácia pravdepodobne skomplikovaná neprípustnosťou niektorých hlavných úloh a teda potrebou pridávať rezy prípustnosti. Táto skutočnosť ovplyvnila iteračný priebeh algortimu a presnejšia aproximácia účelovej

	VLSM				VLSM VR					VLSM AARO
Úloha	CPU	it	rp	ro	CPU	it	rp	ro	α	agr.
pltexpA4.6	4,57	4	0	49	4,63	4	0	258	101%	0
pltexpA4.16	65,38	4	0	273	86,83	4	0	4368	133%	0
pltexpB4.6	0,94	4	0	19	0,96	4	0	42	102%	0
pltexpB5.6	2,06	4	0	43	2,15	4	0	90	104%	0
bonds3.5	0,14	4	0	7	0,12	4	0	30	86%	0
scfxm3.6	23,25	30	186	86	28,24	38	202	592	121%	1
scfxm3.16	95,49	30	287	206	-	-	-	-	-	
scfxm4.6	38,31	30	186	380	46,11	32	203	2012	120%	1
scfxm4.16	553,83	42	354	1378	-	-	-	-	-	
scsd3.27	20,25	14	0	126	14,71	8	0	2295	73%	1
scsd3.64	70,56	10	0	309	104,15	8	0	12480	148%	1
scsd4.8	12,36	10	0	256	11,60	8	0	1856	94%	1
scsd4.27	2501,40	22	0	4645	1128,40	8	0	61317	45%	1

Tabuľka 2: Porovnanie vnorenej L-Shaped metódy a viacrezovej vnorenej L-Shaped metódy.

funkcie tu zrejme neznamenala dostatočnú výhodu voči iným nešpecifickým vplyvom. Spomedzi všetkých testovaných úloh boli rezy prípustnosti pridávané iba v scfxm úlohách.

Posledný stĺpec Tabuľky 2 označený ako 'agr.' udáva, či mala daná úloha potenciál pre použitie metódy s adaptívnou agregáciou rezov optimality (VLSM AARO). V prípade, že počas riešenia danej úlohy nenastávala pre veľmi nízku hladinu nabytočnosti ($\delta \approx 1.0e-5$) žiadna agregácia, je pri danej úlohe v tomto stĺpci 0, inak 1. Úlohy s nulovou hodnotou 'agr.' neboli následne v testoch VLSM AARO uvažované. Pre úlohy scfxm3.16 a scfxm4.16 viacrezová metóda (VLSM VR) neskonvergovala pravdepodobne z numerických príčin použitého lineárneho solvera GLPK[16]. Tieto úlohy neboli v testoch VLSM AARO uvažované, jednak preto, že by nebolo možné ich porovnať s VLSM VR a taktiež kvôli možným numerickým problémom pri VLSM AARO, nakoľko táto metóda štartuje z úplnej dezagregácie rezov, čo je totožné s VLSM VR. V Tabuľke 3 uvádzame výsledky porovnania VLSM (jednorezová verzia) z literatúry [2] a našej upravenej verzie využívajúcej zásobník pri pridávaní rezov prípustnosti. Stĺpce 'rp', 'ro', 'it' a 'CPU' v Tabuľke 3 majú rovnaký význam ako v Tabuľke 2, stĺpec 'zásobník' udáva, či bol použitý zásobník (1 = bol, 0 = nebol). Rezy prípustnosti boli z úloh, ktoré sme testovali, pridávané iba v scfxm úlohách. Vo všetkých ostatných (prípustných) úlohách je iteračná sekvencia takto upravenej

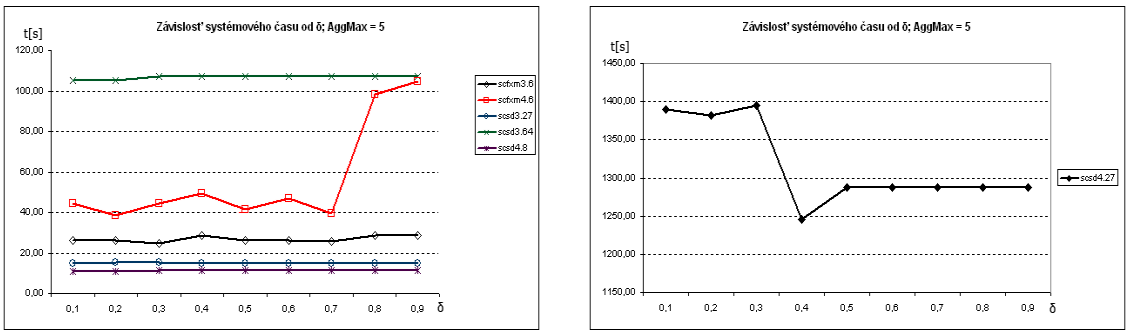
[16]Pri testovaní rôznych úloh sme skúšali aj iné solveri (napr. Linprog z MATLABu) a ukázalo sa, že konvergencia algoritmu môže závisieť od použitého solvera.

VLSM					
Úloha	zásobník	CPU	it	rp	ro
scfxm3.6	1	21,08	30	186	86
	0	59,22	130	255	167
sfxm4.6	1	38,31	30	186	380
	0	50,52	135	243	440
scfxm3.16	1	95,49	30	287	206
	0	1008,40	239	361	353
scfxm4.16	1	553,83	42	354	1378
	0	-	-	-	-

Tabuľka 3: Porovnanie vnorenej L-Shaped metódy so zásobníkom a bez zásobníka.

verzie totožná s pôvodnou verziou, zavedenie zásobníka sa neuplatňuje. Vidno, že úspora systémového času a makroiterácií je veľmi výrazná. Pri scfxm3.6 dosahuje úspora systémového času 65%, pri scfxm4.6 je to 24% a pri scfxm3.16 dokonca 90%. Úloha scfxm4.16 pri verzii bez zásobníka v nastavenom čase neskonvergovala. Počet makroiterácií sa pri scfxm3.6 zredukoval o 77%, pri scfxm4.6 o 78% a scfxm 3.16 o 87%.

Podotýkame, že všetky testované algoritmy vrátane VLSM AARO boli implementované so zásobníkom. Dávame do pozornosti možnosť a zaujímavosť porovnania efektu tejto modifikácie na ďalších úlohách s neprípustnosťou.

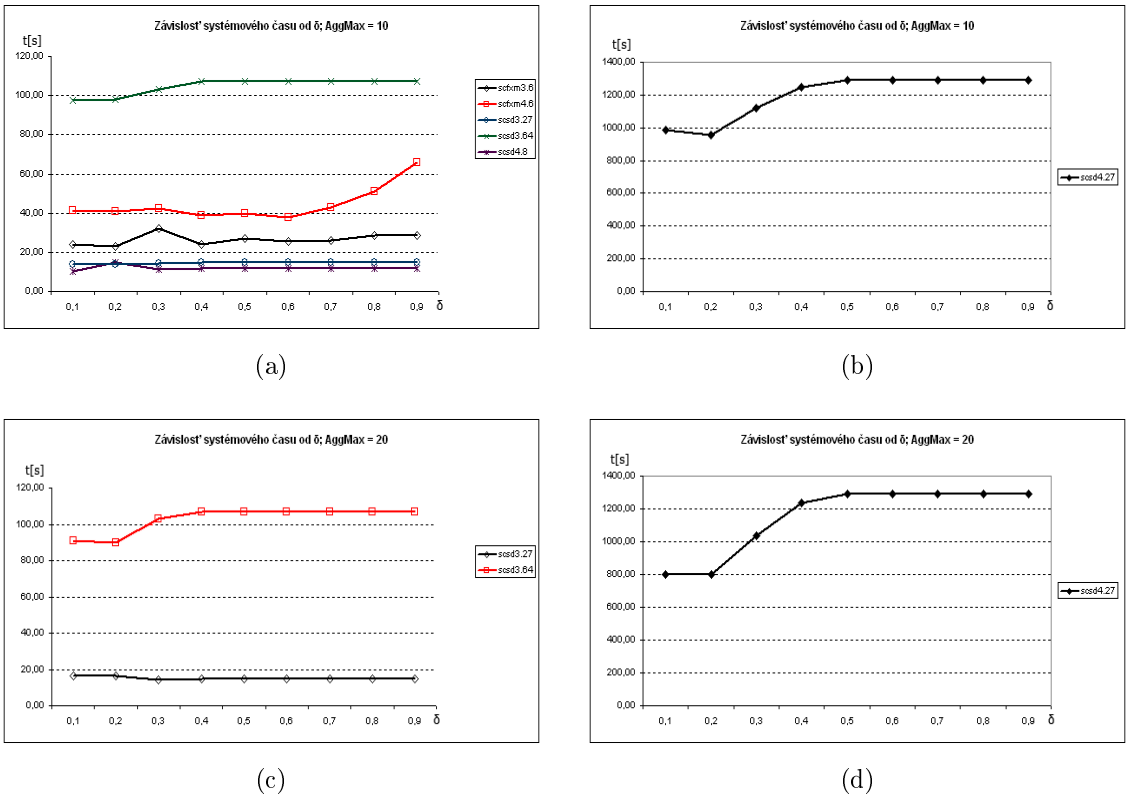


(a) (b)

Obr. 6: VLSM AARO závislosť systémového času od δ.

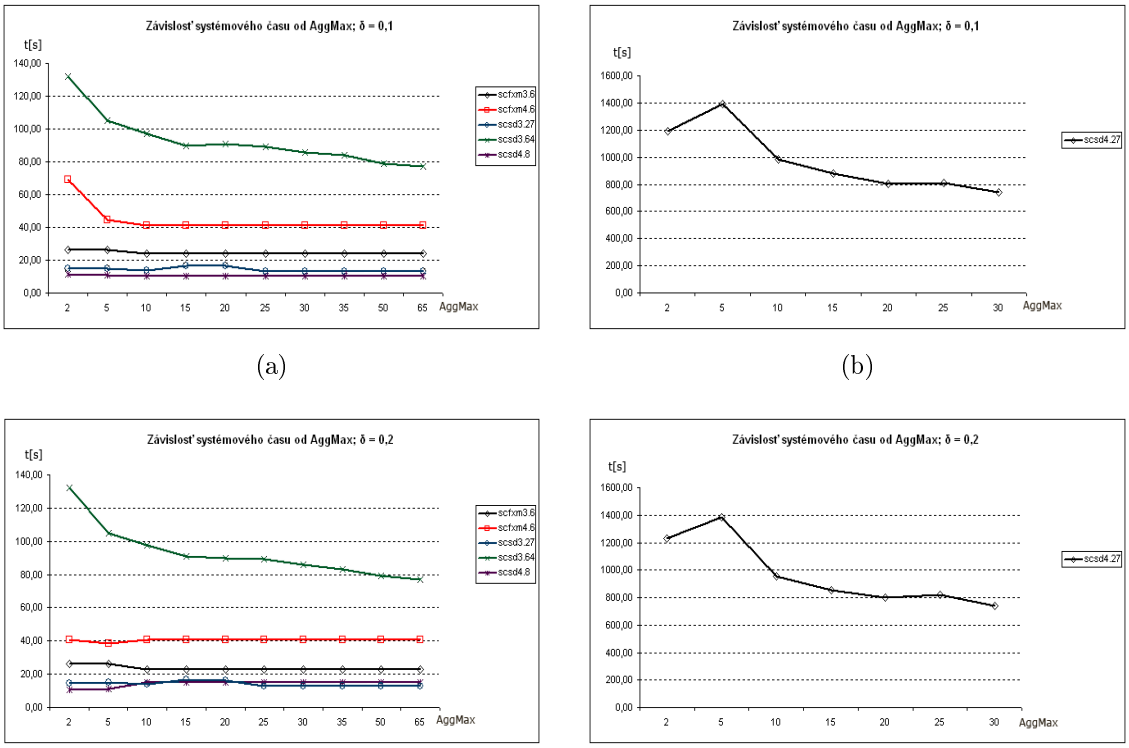
Ďalej popíšeme výsledky dosiahnuté pre algoritmus VLSM AARO. Na Obrázkoch 6 (a)–(b) sú znázornené závislosti systémového času od hladiny nadbytočnosti δ pri hodnote AggMax = 5. Úlohu scsd3.27 bolo potrebné vyčleniť do separátneho grafu kvoli vysokému času riešenia. Obrázky 7 (a)–(b) a 7 (c)–(d) zachytávajú situáciu pre AggMax = 10, respektíve AggMax = 20.

S rastúcou hodnotou parametra AggMax v grafoch postupne nie sú zobrazené



Obr. 7: VLSM AARO závislosť systémového času od δ.

úlohy, ktoré majú počet uzlov na stupeň ňižší než hodnota AggMax, pretože tento parameter tu už prestáva hrať úlohu (možno agregovať všetky uzly) a výsledky sa opakujú. Z týchto grafov sa nám nepodarilo vyvodiť jednoznačnú rastúcu, či klesajúcu závislosť systémového času od δ. Výsledky sú podmienené konkrétnou riešenou úlohou. Vo väčšine prípadov je závislosť takmer konštantná alebo mierne rastúca po určitú hodnotu δ. Konštantné úseky v pravej časti niektorých grafov zodpovedajú tomu, že pre príslušné hodnoty δ už nenastávala agregácia. Z týchto grafou, teda možno vyvodiť aspoň skutočnosť, že pre agregáciu je často potrebné voliť δ dostatočne nízke. Podľa našich experimentov odporúčame pre agregáciu voliť $\delta \leq 0.3$.

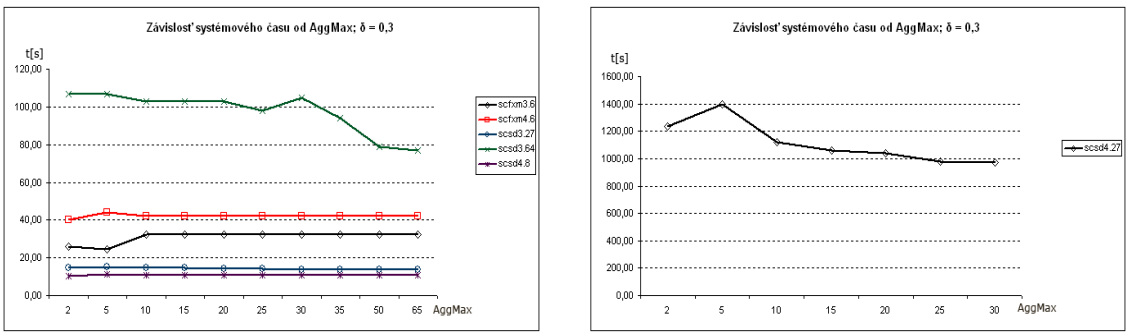


(a) (b)

(c) (d)

Obr. 8: VLSM AARO závislosť systémového času od AggMax.

Na Obrázkoch 8 až 10 sú zachytené výsledky pre závislosť systémového času od AggMax pri rôznych hodnotách δ.

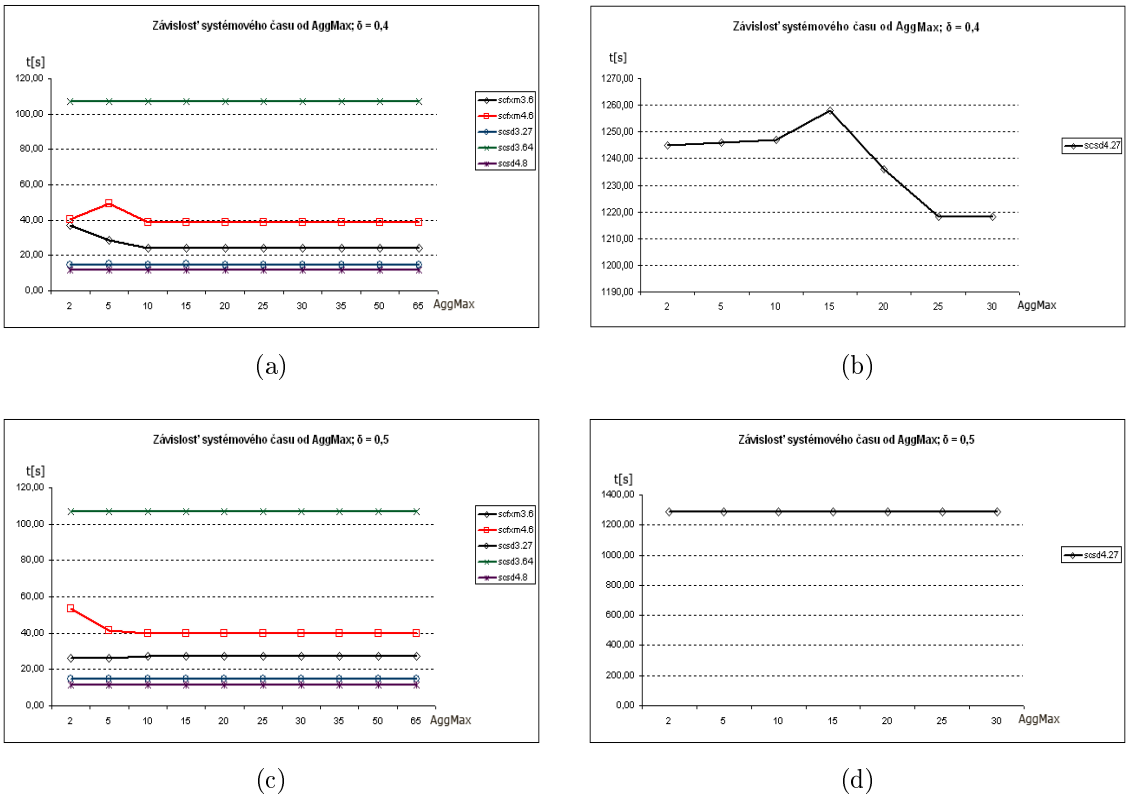


(a) (b)

Obr. 9: VLSM AARO závislosť systémového času od AggMax.

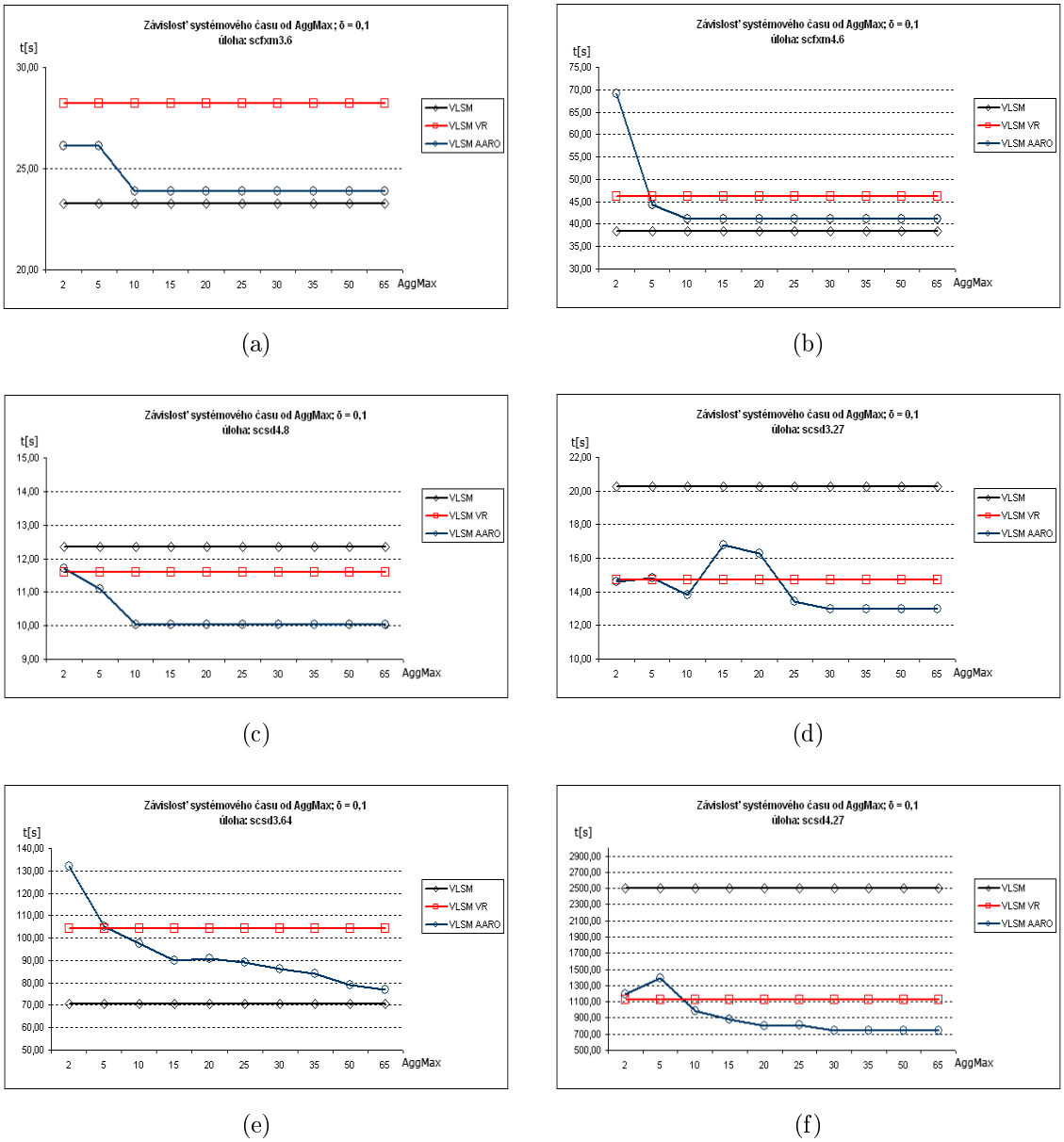
Z príslušných grafov si možno všimnúť, že pri zväčšujúcom sa δ nadobúdajú grafy postupne tvar konštantných čiar. To predstavuje už spomínanú skutočnosť, že pri väčších hodnotách δ sa vytráca agregácia. Ďalej si možno všimnúť, že systémový čas má tendenciu klesať s rastúcim AggMax, pričom minimum sa zväčša nadobúda

pri maximálnej hodnote AggMax. Pre testované úlohy vychádza ako optimálne voliť AggMax veľké alebo tento parameter v algoritme neuvažovať, a teda povoliť úplnú agregáciu.



Obr. 10: VLSM AARO závislosť systémového času od AggMax.

Obrázok 11 zachytáva porovnanie VLSM, VLSM VR a priebeh VLSM AARO pre rôzne hodnoty AggMax s dostatočne nízkou voľbou hladiny nadbytočnosti $\delta = 0.1$. Nízku hladinu δ sme zvolili preto aby sa v dostatočnej miere uplatňovala agregácia nasledovníckych uzlov. Ako vidno, VLSM AARO dosahuje zlepšenie voči VLSM VR takmer vždy, no jeho výhodnosť v porovnaní s VLSM nie je jednoznačná. Zlepšenie voči obom algoritmom dosiahol VLSM AARO v prípadoch scsd4.8, scsd3.27 a scsd 4.27. Malé zlepšenie (ktoré tu nevidno) bolo zaznamenané taktiež v scfxm3.6 pri AggMax $= 10$ a $\delta = 2$, čo možno nájsť v tabuľkovej prílohe. Najväčšie zlepšenie systémového času (voči najrýchlejšiemu z dvojice VLSM a VLSM VR) 34% bolo dosiahnuté v úlohe scsd4.27. V scsd4.8 bolo dosiahnuté zlepšenie približne 12% a v scsd3.27 približne 14%.



Obr. 11: Porovnanie VLSM, VLSM VR a VLSM AARO ($\delta = 0.1$, pre rôzne AggMax).

Na záver tejto časti ešte uvádzame niekoľko porovnaní algoritmov VLSM, VLSM VR a VLSM AARO v tabuľkovej forme. V Tabuľke (4) udvádzame porovnanie algoritmov z hľadiska systémového času ('CPU') pre viaceré hodnoty parametra AggMax vo VLSM AARO, následne v Tabuľke (5) udvádzame ich porovnanie z hľadiska počtu makroiterácií. Symbolom '*' značíme v oboch tabuľkách opakovanie poslednej číselnej hodnoty v danom riadku.

[CPU]	**VLSM**; **VLSM VR**		**VLSM AARO**, $\delta = 0.1$, AggMax:						
Úloha			5	10	15	20	30	50	65
scfxm3.6	23,25	28,24	26,10	23,90	*	*	*	*	*
scfxm4.6	38,31	46,11	44,30	41,20	*	*	*	*	*
scsd3.27	20,25	14,71	14,80	13,80	16,80	16,30	13,00	*	*
scsd3.64	70,56	104,15	105,20	97,40	139,00	91,00	86,00	79,00	77,00
scsd4.8	12,36	11,60	11,10	10,04	*	*	*	*	*
scsd4.27	2501,40	1128,40	1390,00	985,00	882,00	801,00	742,00	*	*

Tabuľka 4: Porovnanie vnorenej L-Shaped metódy, viacrezovej vnorenej L-Shaped metódy a vnorenej L-Shaped metódy s AARO na základe systémového času (CPU).

[Makroit.]	**VLSM**	**VLSM VR**	**VLSM AARO**, $\delta = 0.1$, AggMax:						
Úloha			5	10	15	20	30	50	65
scfxm3.6	30	38	36	32	*	*	*	*	*
scfxm4.6	30	32	34	32	*	*	*	*	*
scsd3.27	14	8	8	8	10	10	8	*	*
scsd3.64	10	8	8	8	12	8	8	8	8
scsd4.8	10	8	8	8	*	*	*	*	*
scsd4.27	22	8	10	8	8	8	8	*	*

Tabuľka 5: Porovnanie vnorenej L-Shaped metódy, viacrezovej vnorenej L-Shaped metódy a vnorenej L-Shaped metódy s AARO na základe počtu makroiterácií.

Podrobnejšie výsledky pre rôzne nastavenia AggMax a δ možno nájsť v tabuľkovej prílohe.

Záver

Numerický experiment viedol k zaujímavým výsledkom. V prvom rade treba spomenúť modifikáciu algoritmu VLSM týkajúcu sa pridávania rezov prípustnosti s použitím zásobníka. V našom experimente sme dosiahli redukciu systémového času do 90% a zníženie počtu makroiterácií do 87%. Nepodarilo sa nám zistiť, či sa takáto modifikácia v praxi používa, no v dostupnej literatúre, ani v literatúre [2] z ktorej sme študovali algoritmus VLSM sme ju nenašli.

Pri porovnaní VLSM a VLSM VR bol na väčšine nami testovaných úloh rýchlejší algoritmus VLSM, pričom počet makroiterácií vyšiel až na dve výnimky (scfxm3.6 a scfxm4.6) podľa očakávaní nižší pre VLSM VR. Spomínané výnimky možno pripísať potrebe pridávať rezy prípustnosti.

Predbežné testy VLSM AARO ukázali, že algoritmus v mnohých prípadoch neagregoval rezy optimality, čiže pre viacero úloh nie je veľmi vhodný. Pôvodné agregačné pravidlo z [11] bolo potrebné mierne modifikovať, nakoľko sa pre naše úlohy ukázalo byť príliš reštriktívne (viď. 5.2.2 Pravidlá agregácie - Hladina nadbytočnosti δ.). Po vyradení úloh, v ktorých nebolo možné testovať agregáciu (Tabuľka 2), bol realizovaný numerický experiment s rôznymi nastaveniami AggMax a δ. Vo všeobecnosti vyšlo rozumné v algoritme voliť parameter AggMax veľký približne ako počet scenárov na stupeň alebo ho vôbec neuvažovať. Parameter δ odporúčame na základe výsledkov experimentu v záujme dosiahnutia agregácie voliť malý $\delta \leq 0.3$. Výkonnosť algoritmu možno hodnotiť nasledovne. V prevažnej väčšine testovaných úloh bol algoritmus rýchlejší ako VLSM VR, no algoritmus VLSM ho často predbehol. V niektorých prípadoch dosiahol zlepšenie voči obom algoritmom - 34% bolo dosiahnuté v úlohe scsd4.27, 12% v scsd4.8 a približne 14% v scsd3.27. Nepatrné zlepšenie 1.5% bolo pre jedno nastavenie dosiahnuté aj v scfxm3.6.

Na základe uvedeného nemožno algoritmus považovať za príliš výhodnú voľbu pre bežné úlohy (také s ktorými sme sa stretli v numerickom experimente) z nasledovných dôvodov. Jeho konštrukcia je rádovo komplexnejšia než konštrukcia VLSM, pričom VLSM dosahuje vo viacerých prípadoch lepší systémový čas. Potenciálna výhoda algoritmu je obmedzená na úlohy, v ktorých nastane agregácia nasledovníckych uzlov na základe použitého agregačného pravidla, pričom nepoznáme (zatiaľ) nijaké objektívne kritérium, ktoré by takéto úlohy vyčlenilo. Z informácií uvedených v [11] ako aj z testov pre nami navrhnutú viacstupňovú verziu vychádza, že algoritmus môže mať určitý potenciál pre veľmi veľké úlohy (scsd4.27 má 19683 scenárov, STORM, 20TERM.. samplované v [11] sú veľmi veľké).

Pripomíname, že v našom experimente sme použili odlišný spôsob agregácie rezov optimality, než použili autori v [11]. Rozdiel je objasnený v časti 3.3. Pôvodný spôsob agregácie z [11] u nás často neviedol k presnej hodnote účelovej funkcie a teda k vyriešeniu úlohy s požadovanou presnosťou. Naša úprava agregácie môže mať vplyv na dosiahnuté výsledky algoritmu.

Zoznam použitej literatúry

[1] K. A. Ariyawansa, A. J. Felt,*On a new collection of stochastic linear programming test problems*, Informs Journal on Computing, 2004, Vol. 16, No. 3, pp. 291-299

[2] J. R. Birge, F. Louveaux, *Introduction to Stochastic Programming*, Springer, 1997.

[3] J. R. Birge, F. Louveaux, *A Multicut Algorithm for Two-Stage Stochastic Linear Programs*, European Journal of Operational Research, 1988, Vol. 34, No. 3, pp. 384–392.

[4] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.

[5] H. I. Gassman, *MSLiP: a computer code for the multistage stochastic linear programming problem*, Mathematical Programming, 1990, Vol. 47, No. 1–3, pp. 407–423.

[6] D. Holmes, *A Collection of Stochastic Programming Problems*, Department of Industrial and Operations Engineering, The University of Michigan, Technical Report 94–11, 1994.

[7] P. Kall, J. Mayer, *Stochastic Linear Programming: Models, Theory, and Computation*, Springer, 2005.

[8] J. Linderoth, *Lectures on Stochastic Programming*, 2003, Lecture 1–22, http://chentserver.uwaterloo.ca/aelkamel/che720/che720-stochastic-optimization/stochastic_opt_2/

[9] J. Plesník, J.Dupačová, M.Vlach, *Lineárne programovanie*, Alfa, Bratislava, 1990.

[10] A. Shapiro, D. Dentcheva, A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, SIAM, 2009.

[11] S. Trukhanov, L. Ntaimo, A. Schaefer, *On Adaptive Multicut Aggregation for Two-Stage Stochastic Linear Programs with Recourse*, European Journal of Operational Research, 2010, Vol. 206, No. 2, pp. 395–406.

[12] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, Springer, 2008.

[13] S. W. Wallace, W. T. Ziemba, *Applications of Stochastic Programming*, SIAM, 2005.

[14] D. Walkup and R.J-B Wets, *Stochastic programs with recourse*, SIAM Journal on Applied Mathematics, 1967, Vol. 15, No. 1, pp. 1299–1314.

Tabuľky

V nasledujúcich tabuľkách sú zachytené výsledky pre algoritmus vnorenej L-Shaped metódy s adaptívnou agregáciou rezov optimality. Označenie je nasledovné:
δ - hladina nadbytočnosti pre agregáciu rezov optimality
AggMax - maximálny počet agregovaných rezov
CPU - priemerný systémový čas
makroit. - počet makroiterácií
rp - počet rezov prípustnosti
ro - počet rezov optimality

Tabuľka T1

	AggMax $= 2; \delta = 0,1$				AggMax $= 2; \delta = 0,2$				AggMax $= 2; \delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	26,10	34	200	500	26,10	34	200	500	25,80	34	200	461
scfxm4.6	69,10	52	201	1737	40,66	34	199	642	40,28	32	199	554
scsd3.27	14,61	8	0	2240	14,75	8	0	2240	14,75	8	0	2294
scsd3.64	132,00	10	0	12467	132,00	10	0	12467	107,00	8	0	12480
scsd4.8	11,70	8	0	1679	10,69	8	0	1679	10,50	8	0	1752
scsd4.27	1194,00	8	0	60515	1229,00	8	0	60515	1234,00	8	0	61262

Tabuľka T2

	AggMax $= 2; \delta = 0,4$				AggMax $= 2; \delta = 0,5$				AggMax $= 2; \delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	37,10	64	200	574	26,40	36	200	428	24,90	32	200	395
scfxm4.6	40,20	32	199	593	53,61	44	199	662	38,08	32	199	649
scsd3.27	14,75	8	0	2294	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480,00
scsd4.8	11,60	8	0	1856	11,60	8	0	1856	11,60	8	0	1856
scsd4.27	1245,00	8	0	61262	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T3

	AggMax $= 2; \delta = 0,7$				AggMax $= 2; \delta = 0,8$				AggMax $= 2; \delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	32,60	48	201	530	28,50	38	202	596	28,50	38	202	592
scfxm4.6	40,55	34	199	730	64,03	56	199	859	42,89	38	199	745
scsd3.27	14,75	8	0	2295	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480,00	107,00	8	0	12480	107,00	8	0	12480
scsd4.8	11,60	8	0	1856	11,60	8	0	1856	11,60	8	0	1856
scsd4.27	1287,90	8	0	61317	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T4

	AggMax $= 5; \delta = 0,1$				AggMax $= 5; \delta = 0,2$				AggMax $= 5; \delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	26,10	36	201	267	26,10	36	201	268	24,60	42	201	299
scfxm4.6	44,30	34	200	823	38,73	32	200	443	44,28	38	199	479
scsd3.27	14,80	8	0	2164	15,10	8	0	2164	15,20	10	0	2297
scsd3.64	105,20	8	0	12220	105,00	8	0	12220	107,00	8	0	12480
scsd4.8	11,10	8	0	1471	11,00	8	0	1460	11,40	8	0	1816
scsd4.27	1390,00	10	0	70177	1382,00	10	0	70072	1395,00	10	0	61645

Tabuľka T5

	AggMax $= 5; \delta = 0,4$				AggMax $= 5; \delta = 0,5$				AggMax $= 5; \delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	28,50	42	200	329	26,20	36	200	332	26,25	36	200	375
scfxm4.6	49,52	42	200	527	41,44	34	199	550	46,84	40	199	599
scsd3.27	14,90	8	0	2291	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.8	11,60	8	0	1856	11,60	8	0	1856	11,60	8	0	1856
scsd4.27	1246,00	8	0	61235	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T6

	AggMax $= 5; \delta = 0,7$				AggMax $= 5; \delta = 0,8$				AggMax $= 5; \delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	25,80	34	200	417	28,50	38	202	596	28,50	38	202	592
scfxm4.6	39,45	32	199	602	98,45	86	200	954	104,77	90	211	917
scsd3.27	14,75	8	0	2295	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.8	11,60	8	0	1856	11,60	8	0	1856	11,60	8	0	1856
scsd4.27	1287,90	8	0	61317	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T7

	AggMax $= 10; \delta = 0,1$				AggMax $= 10; \delta = 0,2$				AggMax $= 10; \delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	23,90	32	200	153	22,89	32	200	154	32,10	50	201	250
scfxm4.6	41,20	32	199	989	40,73	34	199	452	42,36	36	199	514
scsd3.27	13,80	8	0	2019	13,90	8	0	2019	14,50	8	0	2286
scsd3.64	97,40	8	0	11896	97,80	8	0	11896	103,00	8	0	12481
scsd4.8	10,04	8	0	1256	15,00	12	0	1266	11,00	8	0	1797
scsd4.27	985,00	8	0	54455	955,00	8	0	54455	1123,00	8	0	61185

Tabuľka T8

	AggMax $= 10; \delta = 0,4$				AggMax $= 10; \delta = 0,5$				AggMax $= 10; \delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	24,00	32	200	270	27,10	38	201	347	25,76	34	200	323
scfxm4.6	38,70	32	199	515	39,75	34	199	574	37,92	32	199	649
scsd3.27	14,60	8	0	2286	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.8	11,60	8	0	1856	11,60	8	0	1856	11,60	8	0	1856
scsd4.27	1247,00	8	0	61187	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T9

	AggMax $= 10; \delta = 0,7$				AggMax $= 10; \delta = 0,8$				AggMax $= 10; \delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scfxm3.6	25,80	34	200	417	28,50	38	202	596	28,50	38	202	592
scfxm4.6	43,11	36	201	756	50,83	40	201	868	65,80	56	211	893
scsd3.27	14,75	8	0	2295	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.8	11,60	8	0	1856	11,60	8	0	1856	11,60	8	0	1856
scsd4.27	1287,90	8	0	61317	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T10

	AggMax $= 15; \delta = 0,1$				AggMax $= 15; \delta = 0,2$				AggMax $= 15; \delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	16,80	10	0	1886	16,60	10	0	1886	14,70	8	0	2282
scsd3.64	89,81	8	0	11570	90,86	8	0	11581	103,00	8	0	12481
scsd4.27	882,00	8	0	50670	851,00	8	0	50670	1062,00	8	0	61141

Tabuľka T11

	AggMax $= 15; \delta = 0,4$				AggMax $= 15; \delta = 0,5$				AggMax $= 15; \delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	15,00	8	0	2281	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.27	1258,00	8	0	61145	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T12

	AggMax $= 15; \delta = 0,7$				AggMax $= 15; \delta = 0,8$				AggMax $= 15; \delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	14,75	8	0	2295	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.27	1287,90	8	0	61317	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T13

	AggMax $= 20; \delta = 0,1$				AggMax $= 20; \delta = 0,2$				AggMax $= 20; \delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	16,30	10	0	1776	16,30	10	0	1751	14,10	8	0	2277
scsd3.64	91,00	8	0	11246	90,00	8	0	11246	103,00	8	0	12481
scsd4.27	801,00	8	0	46885	797,00	8	0	46885	1038,00	8	0	61097

Tabuľka T14

	AggMax = 20; $\delta = 0,4$				AggMax = 20; $\delta = 0,5$				AggMax = 20; $\delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	14,60	8	0	2276	14,70	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.27	1236,00	8	0	61099	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T15

	AggMax = 20; $\delta = 0,7$				AggMax = 20; $\delta = 0,8$				AggMax = 20; $\delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	14,70	8	0	2295	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	104,10	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.27	1287,90	8	0	61317	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T16

	AggMax = 25; $\delta = 0,1$				AggMax = 25; $\delta = 0,2$				AggMax = 25; $\delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	13,40	8	0	1596	12,70	8	0	1596	14,10	8	0	2272
scsd3.64	89,00	8	0	10921	89,00	8	0	10921	98,00	8	0	12482
scsd4.27	809,00	10	0	60515	820,00	10	0	44702	978,00	8	0	61052

Tabuľka T17

	AggMax = 25; $\delta = 0,4$				AggMax = 25; $\delta = 0,5$				AggMax = 25; $\delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	14,75	8	0	2271	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.27	1218,00	8	0	61056	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T18

	AggMax = 25; $\delta = 0,7$				AggMax = 25; $\delta = 0,8$				AggMax = 25; $\delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	14,75	8	0	2295	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.27	1287,90	8	0	61317	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T19

	AggMax = 30; $\delta = 0,1$				AggMax = 30; $\delta = 0,2$				AggMax = 30; $\delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	13,00	8	0	1541	12,80	8	0	1541	13,80	8	0	2270
scsd3.64	86,00	8	0	10679	86,00	8	0	10679	105,00	10	0	12499
scsd4.27	742,00	8	0	41636	741,00	8	0	41636	974,00	8	0	61052

Tabuľka T20

	AggMax $= 30; \delta = 0,4$				AggMax $= 30; \delta = 0,5$				AggMax $= 30; \delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	14,60	8	0	2269	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.27	1218,00	8	0	61137	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T21

	AggMax $= 30; \delta = 0,7$				AggMax $= 30; \delta = 0,8$				AggMax $= 30; \delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.27	14,75	8	0	2295	14,75	8	0	2295	14,75	8	0	2295
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480
scsd4.27	1287,90	8	0	61317	1287,90	8	0	61317	1287,90	8	0	61317

Tabuľka T22

	AggMax $= 35; \delta = 0,1$				AggMax $= 35; \delta = 0,2$				AggMax $= 35; \delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.64	84,00	8	0	10270	83,00	8	0	10270	94,00	10	0	12482

Tabuľka T23

	AggMax $= 35; \delta = 0,4$				AggMax $= 35; \delta = 0,5$				AggMax $= 35; \delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480

Tabuľka T24

	AggMax $= 35; \delta = 0,7$				AggMax $= 35; \delta = 0,8$				AggMax $= 35; \delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480

Tabuľka T25

	AggMax $= 50; \delta = 0,1$				AggMax $= 50; \delta = 0,2$				AggMax $= 50; \delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.64	79,00	8	0	9295	79,00	8	0	9295	79,00	8	0	9295

Tabuľka T26

	AggMax $= 50; \delta = 0,4$				AggMax $= 50; \delta = 0,5$				AggMax $= 50; \delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480

Tabuľka T27

	AggMax $= 50; \delta = 0,7$				AggMax $= 50; \delta = 0,8$				AggMax $= 50; \delta = 0,9$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.64	107,00	8	0	12480	107,00	8	0	12480	107,00	8	0	12480

Tabuľka T28

	AggMax = 65; $\delta = 0,1$				AggMax = 65; $\delta = 0,2$				AggMax = 65; $\delta = 0,3$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.64	77,00	8	0	8386	77,00	8	0	8386	77,00	8	0	8386

Tabuľka T29

	AggMax = 65; $\delta = 0,4$				AggMax = 65; $\delta = 0,5$				AggMax = 65; $\delta = 0,6$			
Úloha	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro	CPU	makroit.	rp	ro
scsd3.64	107,00	8	0	12480,00	107,00	8	0	12480	107,00	8	0	12480

Listing programu

Nasledujú programové kódy Vnorenej L-Shaped metódy a jej adaptívnej verzie v jazyku MATLAB.

Listing 1: Vnorená L-Shaped metóda

```
function [solved, optim_x, optim_fval, optim_theta, n_fc, n_oc, n_it] =...
fBenders_Singlecut(W,T,h,c,p,lb,ub,D,a,K,H, max_makroit, max_it, tol_fun, bound_tol, STATIC, solver1, solver2)
%solved: #1 vyriesenie, #2 nepripustnost, #3 neohranicenost, #4 prekrocene max_makroit
%krok 0 a Init
counter = 0;
all_nodes = 0;
num_feas_cuts = 0;
num_opt_cuts  = 0;
ctr = 0;
StackSize = 0;
for t= 1:H
  for k= 1:K(t)
   r{t}{k} = 0;
   s{t}{k} = 0;
   if t < H
      W{t}{k}   = [W{t}{k}   zeros(size(W{t}{k},1),1)];
      lb{t}{k} = [lb{t}{k} -Inf]; %theta je zdola neohranicena
      ub{t}{k} = [ub{t}{k}  Inf];
      theta_zeroLHS{t}{k} = zeros(1,size(c{t}{k},2)); %nuly pre x
      c{t}{k} = [c{t}{k} 1];
      theta_zeroLHS{t}{k} = [theta_zeroLHS{t}{k} 1];  %1 pre theta
      theta_zeroRHS{t}{k} = 0;
   else
     theta_zeroLHS{t}{k} = [];
     theta_zeroRHS{t}{k} = [];
   end
   E_con{t}{k} = [];
   e_con{t}{k} = [];
   D_con{t}{k} = [];
   d_con{t}{k} = [];
  end
end
t= 1;
k= 1;
dir = 1; % 1 -> VPRED, -1 -> VZAD
it = 1;

while 1
```

```
%krok 1

 %********************** NLDS(t,k) *********************************
    hv = h{t}{k};
    Wv = W{t}{k};
    cv = c{t}{k};
    lbv = lb{t}{k};
    ubv = ub{t}{k};

 if t > 1
    Tv = T{t}{k};
    rhs = (hv-Tv*x{t-1}{a{t}{k}});
 else
    rhs = hv;
 end

if isempty(theta_zeroLHS{t}{k}), theta_zeroLHS{t}{k} = []; end
if isempty(theta_zeroRHS{t}{k}), theta_zeroRHS{t}{k} = []; end
%if isempty(d_con{t}{k}) && (isempty(D_con{t}{k}) == 0), d_con{t}{k}= 0; end;
 if solver1 == 4
  [xstar,fval,lambda,exitflag,how]=mpt_solveLP(cv,[D_con{t}{k}; E_con{t}{k}],[d_con{t}{k}; e_con{t}{k}],...
       [theta_zeroLHS{t}{k}; Wv],[theta_zeroRHS{t}{k}; rhs],[],solver1,lbv,ubv);
 else
  options  = optimset('LargeScale','off','Simplex','on','Display','off','MaxIter',max_it,'TolFun',tol_fun);
  [xstar,fval,exitflag,output,lambda]=linprog(c{t}{k},[D_con{t}{k}; E_con{t}{k}],[d_con{t}{k}; e_con{t}{k}],...
       [theta_zeroLHS{t}{k}; W{t}{k}],[theta_zeroRHS{t}{k}; rhs],lb{t}{k},ub{t}{k},[],options);
 end

%*********************************************************************
if t==H & k == K(t)
  all_nodes = 1;
end
%lambda
if dir == 1,
    mode = 'FW';
else
    mode = 'BW';
end
disp(['MP solved for NLDSA(' int2str(t) ',' int2str(k) ') - ' mode]);
 if t < H
   sizex = size(cv,2)-1;
   if size(xstar,1) > 0
    theta_t_k = xstar(sizex+1);
   else
    theta_t_k = 0;
```

```
  end
else
  sizex = size(cv,2);
  theta_t_k = 0;
end

if size(xstar,1) > 0
 x_t_k     = xstar(1:sizex);
else
 x_t_k = zeros(sizex,1);
end
w = 0;
%kontrola pripustnosti a neohranicenosti
if (exitflag ~= 1)
  return_v = 1;
  if size(xstar,2) > 0
    fv= cv(1:sizex)*xstar(1:sizex);
  else
    fv = 0;
  end
  if abs(fv) > bound_tol
    disp('MP is unbounded.');
    optim_x = []; optim_fval = []; optim_theta = [];
    solved = 3;
    return;
  elseif (exitflag == -5 || exitflag == -2 || exitflag == -1 || exitflag == 0)
    if t == 1
      disp('Problem is infeasible.'); %return;
    end
    %disp('NLDSA() is not feasible.');
    return_v = 0;
    %hladanie 'basic dual solution'
    n_rowsW    = size(W{t}{k},1);
    e1         = ones(1,n_rowsW);
    I          = eye(n_rowsW);
    f_feas     = [zeros(1,sizex),e1,e1];
    sizef      = sizex+n_rowsW*2;
    if size(D_con{t}{k},2) > 0
      Aineq_feas = [D_con{t}{k}(:,1:sizex) zeros(size(D_con{t}{k},1), n_rowsW*2)];
      bineq_feas =  d_con{t}{k};
    else
      Aineq_feas = [];
      bineq_feas = [];
    end
    Aeq_feas   = [W{t}{k}(:,1:sizex),I,-I];
```

```
      if t > 1
       beq_feas   = h{t}{k}-T{t}{k}*x{t-1}{a{t}{k}};
      else
       beq_feas =  h{t}{k};
      end

     if solver2 == 4
      [v,w,lambda1,exitflag1,how]=mpt_solveLP(f_feas,Aineq_feas,bineq_feas,...
      Aeq_feas,beq_feas,[],solver2,zeros(1,sizef),Inf*ones(1,sizef));
     else
      options1  = optimset('LargeScale','off','Simplex','on','Display','off','MaxIter',max_it,'TolFun',1e-5);
      [v,w,exitflag1,output,lambda1] = linprog(f_feas,Aineq_feas,bineq_feas,Aeq_feas,beq_feas,zeros(1,sizef),...
Inf*ones(1,sizef),[],options1);
     end
    end
  else
    return_v = 1;
  end
  if w == 0
    return_v = 1;
  end
  if size(theta_zeroLHS{t}{k},1) > 0
    start = 2;
  else
    start = 1;
  end

  n_EQ      = size(Wv,1)+size(theta_zeroLHS{t}{k},1);
  ind1      = size(D_con{t}{k},1);
  ind2      = size(D_con{t}{k},1)+size(E_con{t}{k},1);
  if solver1 == 4
    pi_t_k  =   -lambda((size(lambda,1)-n_EQ+start):(size(lambda,1)));
    rho_t_k =   -lambda(1:ind1);
    sigma_t_k = -lambda(ind1+1:ind2);
  else
    if (size(lambda.eqlin,1) > 0) || (size(lambda.eqlin,2) > 0)
      pi_t_k    = -lambda.eqlin(start:n_EQ);
    end
    if (size(lambda.ineqlin,1) > 0) || (size(lambda.ineqlin,2) > 0)
      rho_t_k   = -lambda.ineqlin(1:ind1);
      sigma_t_k = -lambda.ineqlin(ind1+1:ind2);
    end
  end
 %*********************************************************************
 if return_v == 0  %nepripustnost
```

```
  if t == 1
    disp('Problem is infeasible');
    optim_x = []; optim_fval = []; optim_theta = [];
    solved = 2;
    return;
  else
    r{t-1}{a{t}{k}} = r{t-1}{a{t}{k}} + 1;
    if solver2 == 4
      ind1 = size(Aeq_feas,1);
      ind2 = size(Aineq_feas,1);
      pi_t_k1  =   -lambda1((size(lambda1,1)-ind1+1):(size(lambda1,1)));
      rho_t_k1 =   -lambda1(1:ind2);
    else
      pi_t_k1  = -lambda1.eqlin(1:size(lambda1.eqlin,1));
      rho_t_k1 = -lambda1.ineqlin(1:size(lambda1.ineqlin,1));
    end
    D_con{t-1}{a{t}{k}} = [D_con{t-1}{a{t}{k}}; -pi_t_k1'*T{t}{k} 0];
    if isempty(d_con{t}{k}), suc = 0;
    else
      suc = rho_t_k1'*d_con{t}{k};
    end
    d_con{t-1}{a{t}{k}} = [d_con{t-1}{a{t}{k}}; -(pi_t_k1'*h{t}{k} + suc)];
    %if isempty(d_con{t-1}{a{t}{k}}), d_con{t-1}{a{t}{k}} = 0; end
    disp(['Feasibility cut added to (' int2str(t-1) ',' int2str(a{t}{k}) ')']);
    num_feas_cuts = num_feas_cuts + 1;

    StackSize = StackSize + 1;
    Stack{StackSize} = [t k dir];   %prida poziciu do zasobnika
    dir = -1;
    k = a{t}{k};
    t = t - 1;
    %navrat na krok 1
  end
else %pripustnost
  x{t}{k} = x_t_k;
  theta{t}{k} = theta_t_k;
  pi{t}{k} = pi_t_k;
  rho{t}{k} = rho_t_k;
  sigma{t}{k} = sigma_t_k;

  if StackSize > 0
    t = Stack{StackSize}(1);      %vyberie zo zasobnika posledny
    k = Stack{StackSize}(2);      %uzol, z ktoreho bol pridany rez pripustnosti
    dir = Stack{StackSize}(3);
    StackSize = StackSize - 1;
```

```
elseif k < K(t)
  k = k + 1;
   %navrat na krok 1
else %k == K(t)
 if ((dir == 1) & (t < H))
      t= t+1;
      k= 1;     %poz. toto asi v knihe BL chyba
      %navrat na krok 1
 elseif (dir == -1 & t == 1)
       t= t+1;
       k= 1;
       dir= 1;
 else
     if t==H
      dir = -1; %dir = BACK
      it = it+1;
     end
    %krok 2
      Tv = T{t}{k};
      num_constr = 0;
      for j = 1:K(t-1)
       sizex = size(Tv,2);
       E{t-1}{j} = zeros(1,sizex);
       e{t-1}{j} = 0;
       for k= D{t-1}{j} %D{t-1}{j} -> descendant scenarios of (t-1,j) scenario
          Tv = T{t}{k};
          hv = h{t}{k};

          E{t-1}{j} = E{t-1}{j} + p{t}{k}/p{t-1}{j}*(pi{t}{k})'*Tv;

         if isempty(rho{t}{k})
          sum1 = 0;
         else
          sum1 = rho{t}{k}'*d_con{t}{k};
         end
         if isempty(sigma{t}{k})
           sum2 = 0;
         else
           sum2 = sigma{t}{k}'*e_con{t}{k};
         end
         e{t-1}{j} = e{t-1}{j} + (p{t}{k}/p{t-1}{j})*(pi{t}{k}'*hv + sum1 + sum2);
       end
       if isempty(e{t-1}{j}), e{t-1}{j} = 0; end

       theta_opt{t-1}{j} = e{t-1}{j} - E{t-1}{j}*x{t-1}{j};
```

```
 if size(theta_zeroRHS{t-1}{j},1) > 0
   theta_zeroRHS{t-1}{j} = [];
   theta_zeroLHS{t-1}{j} = [];
   %vymaz ohranicenia theta(j,t-1) = 0 ak existuje
   s{t-1}{j} = 1;
   %pridaj ohranicenie s Ej a ej
   E_con{t-1}{j} = [E_con{t-1}{j}; -E{t-1}{j} -1];
   e_con{t-1}{j} = [e_con{t-1}{j}; -e{t-1}{j}];
   disp(['Optimality cut added to (' int2str(t-1) ',' int2str(j) ')']);
   num_constr = num_constr + 1;
   num_opt_cuts = num_opt_cuts + 1;

 elseif theta_opt{t-1}{j} > theta{t-1}{j}
   s{t-1}{j} = s{t-1}{j} + 1;
   %pridaj ohranicenie s Ej a ej
   E_con{t-1}{j} = [E_con{t-1}{j}; -E{t-1}{j} -1];
   e_con{t-1}{j} = [e_con{t-1}{j}; -e{t-1}{j}];
   disp(['Optimality cut added to (' int2str(t-1) ',' int2str(j) ')']);
   num_constr = num_constr + 1;
   num_opt_cuts = num_opt_cuts + 1;
 end
end

if t==2 & num_constr == 0
  disp('Optimum.');
  z = 0;
  for i= 1:H
    for j= 1:K(i)
     cv = c{i}{j};
     if i<H
      xsz= size(cv,2)-1;
     else
      xsz= size(cv,2);
     end
       z= z + p{i}{j}*cv(1:xsz)*x{i}{j};
    end
  end
  optim_x     = x{1}{1}
  optim_fval  = z;
  optim_theta = theta_opt{1}{1};
  solved      = 1;
  n_fc = num_feas_cuts;
  n_oc = num_opt_cuts;
  n_it = it;
```

```
          return;
        else
          t= t-1;
          k= 1;
          if t == 1
            dir = 1;
            it = it+1;
          end
        end
      end
    %end
   end
  end
   if it > max_makroit
     optim_x = []; optim_fval = []; optim_theta = [];
     solved = 4;
     return;
   end
end %cyklus krok 1
```

Listing 2: Vnorená L-Shaped metóda s adaptívnou agregáciou rezov optimality

```
function [solved, optim_x, optim_fval, optim_theta, n_fc, n_oc, n_it, AGG_COEF] =...
fBenders_Adaptive_Multicut(W,T,h,c,p,lb,ub,Desc,a,K,H, max_makroit, max_it, tol_fun, bound_tol,theta_eps,
solver1, solver2)
%solved: #1 vyriesenie, #2 nepripustnost, #3 neohranicenost, #4 prekrocene max_makroit
%options = optimset('LargeScale','on','Display','off','MaxIter',max_it,'TolFun',tol_fun);

RedundTreshold = 0.3; %ak pomer neaktivity ohraniceni presiahne toto cislo,
                      %zlucia sa do jednoho (volit medzi 0 a 1)

MaxAgg  = 10;  %maximalny pocet atomickych rezov ktore mozu byt z
               %zagregovane do 1 skupiny

WarmUpPeriod = 1;    %zahrievacia perioda pocas ktorej neagreguje

StackSize = 0;

nFeasibCuts = 0;
nOptimCuts  = 0;

AggStartSize = 0;
AggEndSize   = 0;

%step 0 a Init
```

```
for t= 1:H
  for k= 1:K(t)
   r{t}{k} = 0;
   s{t}{k} = 0;
   ni{t}{k} = 1;
   num_agg{t}{k} = 0;



   if t < H
     agg_map{t}{k}= zeros(1,size(Desc{t}{k},2));

     NumAtomic{t}{k} = ones(1,size(Desc{t}{k},2));

     [D{t}{k}{1}, L{t}{k}{1}]= SetAggD(agg_map{t}{k});

     fd_s{t}{k} = zeros(1,L{t}{k}{1});
     ValidCuts{t}{k} = [];

     %pozor indexy su posunute voci zapisu algoritmu 0->1
     AggStartSize = AggStartSize + L{t}{k}{1};
     theta_zeroLHS{t}{k} = [];
     theta_zeroRHS{t}{k} = [];
     for i= 1:L{t}{k}{1}
       vect    = zeros(1, L{t}{k}{1});
       vect(i) = 1;
       theta_zeroLHS{t}{k} = [theta_zeroLHS{t}{k}; [zeros(1,size(c{t}{k},2)) vect]];  %1 pre theta
       theta_zeroRHS{t}{k} = [theta_zeroRHS{t}{k}; 0];
     end

     W{t}{k}  = [W{t}{k}   zeros(size(W{t}{k},1),L{t}{k}{1})];
     lb{t}{k} = [lb{t}{k} -Inf*ones(1,L{t}{k}{1})]; %theta je zdola neohranicena
     ub{t}{k} = [ub{t}{k}  Inf*ones(1,L{t}{k}{1})];
     c{t}{k}  = [c{t}{k} ones(1,L{t}{k}{1})];
   else
     agg_map{t}{k}= [];
     L{t}{k}{1} = 0;
     D{t}{k}{1} = [];

     theta_zeroLHS{t}{k} = [];
     theta_zeroRHS{t}{k} = [];
   end
   E_con{t}{k} = [];
   e_con{t}{k} = [];
   D_con{t}{k} = [];
```

```
   d_con{t}{k} = [];
  end
end

t= 1;
k= 1;
dir = 1; % 1 -> VPRED, -1 -> VZAD
it = 1;

while 1

%krok 1

 %********************** NLDSA(t,k) *********************************
  if t > 1
    rhs = h{t}{k}-T{t}{k}*x{t-1}{a{t}{k}};
  else
    rhs = h{t}{k};
  end

  hv = h{t}{k}; Wv = W{t}{k}; cv = c{t}{k}; lbv = lb{t}{k}; ubv = ub{t}{k};

 if isempty(theta_zeroLHS{t}{k}), theta_zeroLHS{t}{k} = []; end
 if isempty(theta_zeroRHS{t}{k}), theta_zeroRHS{t}{k} = []; end
 %if isempty(d_con{t}{k}), d_con{t}{k}= zeros(size(D_con{t}{k},1),1); end;
 if solver1 == 4
   [xstar,fval,lambda,exitflag,how]=mpt_solveLP(cv,[D_con{t}{k}; E_con{t}{k}],[d_con{t}{k}; e_con{t}{k}],...
   [theta_zeroLHS{t}{k}; Wv],[theta_zeroRHS{t}{k}; rhs],[],solver1,lbv,ubv);
 else
   options  = optimset('LargeScale','on','Simplex','off','Display','off','MaxIter',max_it,'TolFun',tol_fun);
   [xstar,fval,exitflag,output,lambda]=linprog(c{t}{k},[D_con{t}{k}; E_con{t}{k}],[d_con{t}{k}; e_con{t}{k}],...
   [theta_zeroLHS{t}{k}; W{t}{k}],[theta_zeroRHS{t}{k}; rhs],lb{t}{k},ub{t}{k},[],options);
 end

 %*********************************************************************

  if dir == 1,
     mode = 'FW';
 else
     mode = 'BW';
 end

   if dir == 1
     ni{t}{k} = ni{t}{k} + 1;
     D{t}{k}{ni{t}{k}} = D{t}{k}{ni{t}{k}-1};
```

```
     L{t}{k}{ni{t}{k}} = L{t}{k}{ni{t}{k}-1};
   end

%  disp(['MP solved for NLDSA(' int2str(t) ',' int2str(k) ') - ' mode]);

 if t < H
    sizex = size(c{t}{k},2)-L{t}{k}{ni{t}{k}};
    if size(xstar,1) > 0
      theta_t_k = xstar(sizex+1:sizex+L{t}{k}{ni{t}{k}})';
    else
      theta_t_k = 0;
    end
  else
    sizex = size(c{t}{k},2);
    theta_t_k = 0;
 end

  if size(xstar,1) > 0
   x_t_k     = xstar(1:sizex);
  else
   x_t_k = zeros(sizex,1);
  end

  w = 0;
  if (exitflag ~= 1)
    return_v = 1;
    if size(xstar,2) > 0
      fv= cv(1:sizex)*xstar(1:sizex);
    else
      fv = 0;
    end
    if abs(fv) > bound_tol
      disp('MP is unbounded.');
      optim_x = []; optim_fval = []; optim_theta = [];
      solved = 3;
      return;
    elseif (exitflag == -5 || exitflag == -2 || exitflag == -1 || exitflag == 0)
      if t == 1
        disp('Problem is infeasible.'); return;
      end
      return_v = 0;
      %hladanie 'basic dual solution'
      n_rowsW    = size(W{t}{k},1);
      e1         = ones(1,n_rowsW);
      I          = eye(n_rowsW);
```

```
    f_feas     = [zeros(1,sizex),e1,e1];
    sizef      = sizex+n_rowsW*2;
    if size(D_con{t}{k},2) > 0
      Aineq_feas = [D_con{t}{k}(:,1:sizex) zeros(size(D_con{t}{k},1), n_rowsW*2)];
      bineq_feas =  d_con{t}{k};
    else
      Aineq_feas = [];
      bineq_feas = [];
    end
    Aeq_feas   = [W{t}{k}(:,1:sizex),I,-I];
    if t > 1
     beq_feas   = h{t}{k}-T{t}{k}*x{t-1}{a{t}{k}};
    else
     beq_feas =  h{t}{k};
    end

    if solver2 == 4
      [v,w,lambda1,exitflag1,how]=mpt_solveLP(f_feas,Aineq_feas,bineq_feas,...
      Aeq_feas,beq_feas,[],solver2,zeros(1,sizef),Inf*ones(1,sizef));
    else
      options1  = optimset('LargeScale','on','Simplex','off','Display','off','MaxIter',max_it,'TolFun',1e-5);
      [v,w,exitflag1,output,lambda1] = linprog(f_feas,Aineq_feas,bineq_feas,Aeq_feas,beq_feas,zeros(1,sizef),...
        Inf*ones(1,sizef),[],options1);
    end
  end
 else
  return_v = 1;
 end

 if w == 0
  return_v = 1;
 end

if size(theta_zeroLHS{t}{k},1) > 0
  start = size(theta_zeroLHS{t}{k},1)+1;
 else
  start = 1;
end

 n_EQ      = size(W{t}{k},1)+size(theta_zeroLHS{t}{k},1);
 ind1      = size(D_con{t}{k},1);
 ind2      = size(D_con{t}{k},1)+size(E_con{t}{k},1);

  if solver1 == 4
   pi_t_k  =   -lambda((size(lambda,1)-n_EQ+start):(size(lambda,1)));
```

```
    rho_t_k =   -lambda(1:ind1);
    sigma_t_k = -lambda(ind1+1:ind2);
  else
   if (size(lambda.eqlin,1) > 0) || (size(lambda.eqlin,2) > 0)
     pi_t_k    = -lambda.eqlin(start:n_EQ);
   end

   if (size(lambda.ineqlin,1) > 0) || (size(lambda.ineqlin,2) > 0)
     rho_t_k   = -lambda.ineqlin(1:ind1);
     sigma_t_k = -lambda.ineqlin(ind1+1:ind2);
   end
  end

if return_v == 0  %nepripustnost
  if t == 1
    disp('Problem is infeasible');
    optim_x = []; optim_fval = []; optim_theta = [];
    solved = 2;
    return;
  else
    r{t-1}{a{t}{k}} = r{t-1}{a{t}{k}} + 1;

    if solver2 == 4
      ind1 = size(Aeq_feas,1);
      ind2 = size(Aineq_feas,1);
      pi_t_k1  = -lambda1((size(lambda1,1)-ind1+1):(size(lambda1,1)));
      rho_t_k1 = -lambda1(1:ind2);
    else
      pi_t_k1  = -lambda1.eqlin(1:size(lambda1.eqlin,1));
      rho_t_k1 = -lambda1.ineqlin(1:size(lambda1.ineqlin,1));
    end

    D_con{t-1}{a{t}{k}} = [D_con{t-1}{a{t}{k}}; -pi_t_k1'*T{t}{k} zeros(1,L{t-1}{a{t}{k}}{ni{t-1}{a{t}{k}}})];
    if isempty(d_con{t}{k}), suc = 0;
    else
      suc = rho_t_k1'*d_con{t}{k};
    end

    d_con{t-1}{a{t}{k}} = [d_con{t-1}{a{t}{k}}; -(pi_t_k1'*h{t}{k} + suc)];

    %disp(['Feasibility cut added to (' int2str(t-1) ',' int2str(a{t}{k}) ')']);
    nFeasibCuts = nFeasibCuts + 1;
    StackSize = StackSize + 1;
    stack{StackSize} = [t k dir];
```

```
    dir = -1;
    k = a{t}{k};
    t = t - 1;
    %navrat na krok 1
  end
else %pripustnost
  x{t}{k} = x_t_k;              %uloz hodnotu x
  if size(theta_zeroLHS{t}{k},1) > 0
   theta{t}{k} = 0;
  else
   theta{t}{k} = theta_t_k;     %uloz hodnotu theta
  end

  %uloz hodnoty dualnych premennych
  pi{t}{k} = pi_t_k;
  rho{t}{k} = rho_t_k;
  sigma{t}{k} = sigma_t_k;

  %ak su v zasobniku nejake uzly, vyber posledny pridany
  if StackSize > 0
    t = stack{StackSize}(1);
    k = stack{StackSize}(2);
    dir = stack{StackSize}(3);
    StackSize = StackSize - 1;

  else

  if k < K(t)
     k = k + 1;
     %navrat na krok 1
  else %k == K(t)
    if (dir == 1 & t < H)
        t= t+1;
        k= 1;     %poz. toto v knihe BL chyba
        %return to step 1
    elseif (dir == -1 & t == 1)
        t= t+1;
        k= 1;
        dir= 1;
    else
      if t==H
        dir = -1; %dir = BACK
        it = it+1;
      end
```

```
        %krok 2
        nNewOptimCuts = 0;
        for j = 1:K(t-1)

         if (s{t-1}{j} > 0) && (L{t-1}{j}{ni{t-1}{j}-1} > 1)
           [agg_map, fd_s{t-1}{j}] = CreateAggMap(sigma{t-1}{j}, s{t-1}{j}, L{t-1}{j}{ni{t-1}{j}-1},...
           ValidCuts{t-1}{j}, fd_s{t-1}{j}, NumAtomic{t-1}{j}, RedundTreshold, MaxAgg, WarmUpPeriod,
MinAggGroups);

           if (sum(agg_map) > 1) %agreguju sa minimalne 2 skupiny
             num_agg{t-1}{j} = 1;
             disp('Agregujem.');
             %krok 2a
             [D{t-1}{j}{ni{t-1}{j}}, L{t-1}{j}{ni{t-1}{j}}, fd_s{t-1}{j}, NumAtomic{t-1}{j}]  =...
              AggD(D{t-1}{j}{ni{t-1}{j}-1},L{t-1}{j}{ni{t-1}{j}-1}, agg_map, fd_s{t-1}{j}, NumAtomic{t-1}{j});

             [W{t-1}{j}, lb{t-1}{j}, ub{t-1}{j}, c{t-1}{j}, D_con{t-1}{j}] =...
             UpdateData(W{t-1}{j}, lb{t-1}{j}, ub{t-1}{j}, c{t-1}{j}, L{t-1}{j}{ni{t-1}{j}-1},...
             L{t-1}{j}{ni{t-1}{j}}, D_con{t-1}{j});
             %krok 2b

             offset = Offset(D,L,ni,t-1,j,1);
             [E_con{t-1}{j},e_con{t-1}{j},ValidCuts{t-1}{j}] =
         AggCuts(s{t-1}{j}, t,j,E_con{t-1}{j},...

 e_con{t-1}{j},p,D{t-1}{j}{ni{t-1}{j}-1},L{t-1}{j}{ni{t-1}{j}-1},ValidCuts{t-1}{j},agg_map,offset);  %ok
             %update theta
             theta_agg = 0;
             for i = 1:size(agg_map,2)
               if agg_map(i) == 1
                  theta_agg = theta_agg + theta{t-1}{j}(i);
               end
             end

             theta_pom = [];
             ctr = 0;
             for i = 1:size(agg_map,2)
               if agg_map(i) == 0
                   theta_pom = [theta_pom theta{t-1}{j}(i)];
               elseif (agg_map(i) == 1) && (ctr == 0)
                   ctr = 1;
                   theta_pom = [theta_pom theta_agg];
               end
             end
```

```
    theta{t-1}{j} = theta_pom;
  end
end

sizex = size(T{t}{j},2);
theta_opt{t-1}{j} = [];

%pridavanie novych rezov
%krok 2c
for d = 1:L{t-1}{j}{ni{t-1}{j}} %vsetky d z Lambda..
E{t-1}{j}{d} = zeros(1,sizex);
e{t-1}{j}{d} = 0;

offset = Offset(D,L,ni,t-1,j,0);

    for k1= D{t-1}{j}{ni{t-1}{j}}{d}+offset %D{t-1}{j} -> nasledovnicke uzly pre uzol (t-1,j)
      E{t-1}{j}{d} = E{t-1}{j}{d} + p{t}{k1}/p{t-1}{j}*(pi{t}{k1})'*T{t}{k1};
        if isempty(rho{t}{k1})
          sum1 = 0;
        else
          sum1 = rho{t}{k1}'*d_con{t}{k1};
        end

        if isempty(sigma{t}{k1})
          sum2 = 0;
        else
          sum2 = sigma{t}{k1}'*e_con{t}{k1};
        end

      e{t-1}{j}{d} = e{t-1}{j}{d} + (p{t}{k1}/p{t-1}{j})*(pi{t}{k1}'*h{t}{k1} + sum1 + sum2);
    end


    theta_opt{t-1}{j} =[theta_opt{t-1}{j} (e{t-1}{j}{d} - E{t-1}{j}{d}*x{t-1}{j})];

    if size(theta_zeroLHS{t-1}{j},1) > 0

      theta_zeroRHS{t-1}{j}(1,:) = [];
      theta_zeroLHS{t-1}{j}(1,:) = [];
      %vymaz ohranicenie theta(j,t-1) = 0 ak existuje
      %pridaj ohranicenie s Ej a ej
      vect    = zeros(1, L{t-1}{j}{ni{t-1}{j}});
      vect(d) = -1;
      E_con{t-1}{j} = [E_con{t-1}{j}; -E{t-1}{j}{d} vect];
      e_con{t-1}{j} = [e_con{t-1}{j}; -e{t-1}{j}{d}];
```

```
              ValidCuts{t-1}{j} = [ValidCuts{t-1}{j}; 1];
%             disp(['Optimality cut added to (' int2str(t-1) ',' int2str(j) ')']);
              nNewOptimCuts = nNewOptimCuts + 1;
              nOptimCuts = nOptimCuts + 1;

            elseif (theta_opt{t-1}{j}(d) > theta{t-1}{j}(d) + theta_eps)
              disp(['Rez pridany do' int2str(t-1) int2str(j)]);
              %pridaj ohranicenie s Ej a ej
              vect    = zeros(1, L{t-1}{j}{ni{t-1}{j}});
              vect(d) = -1;
              E_con{t-1}{j} = [E_con{t-1}{j}; -E{t-1}{j}{d} vect];
              e_con{t-1}{j} = [e_con{t-1}{j}; -e{t-1}{j}{d}];
              ValidCuts{t-1}{j} = [ValidCuts{t-1}{j}; 1];
              nOptimCuts = nOptimCuts + 1;
%             disp(['Optimality cut added to (' int2str(t-1) ',' int2str(j) ')']);
              nNewOptimCuts = nNewOptimCuts + 1;
            else
              vect    = zeros(1, L{t-1}{j}{ni{t-1}{j}});
              zer     = zeros(1,size(E_con{t-1}{j},2));
              E_con{t-1}{j} = [E_con{t-1}{j}; zer vect];
              e_con{t-1}{j} = [e_con{t-1}{j}; 0];
              ValidCuts{t-1}{j} = [ValidCuts{t-1}{j}; 0];
              nNewOptimCuts = nNewOptimCuts + 1;
            end
          end %for d = ..
          theta_opt{t-1}{j}(1)
          theta{t-1}{j}(1)
          theta_zeroRHS{t-1}{j} = [];
          theta_zeroLHS{t-1}{j} = [];

           if nNewOptimCuts > 0
              s{t-1}{j} = s{t-1}{j} + 1;
           end
        end   %for j=1:K(t)

        %krok 3
        if t==2 & nNewOptimCuts == 0
          disp('Optimum.');
          z = 0;
          for i= 1:H
            for j= 1:K(i)
            if i<H
              xsz= size(c{i}{j},2)-L{i}{j}{ni{i}{j}};
              AggEndSize = AggEndSize + L{i}{j}{ni{i}{j}};
            else
```

```
              xsz=  size(c{i}{j},2);
            end
              z= z + p{i}{j}*c{i}{j}(1:xsz)*x{i}{j};
            end
          end
          optim_x     = x{1}{1}
          optim_fval  = z;
          optim_theta = [theta_opt{1}{1}(1:L{1}{1}{ni{1}{1}-1})];
          solved      = 1;
          n_fc = nFeasibCuts;
          n_oc = nOptimCuts;
          n_it = it;
          AGG_COEF = AggEndSize/AggStartSize;
          return;
        else
          t= t-1;
          k= 1;
          if t == 1
            dir = 1; %direction FORE
            it = it+1;
          end
        end
      end
    end
  end
   if it > max_makroit
     optim_x = []; optim_fval = []; optim_theta = [];
     solved = 4;
     return;
   end
end %cyklus krok 1
%function fBenders_Adaptive
%funkcia na nastavenie zaciatocnej agregacie
%agg_map je napr. v tvare 0 1 2 1 0 0 0 2 0 1 2 2 0 0 3 2 0 3, tj
%rovnake cisla okrem nul vytvoria spolu skupiny
%multicut same nuly
function [D, L] = SetAggD(agg_map)
  L = sum(agg_map == 0) + max(agg_map);
  n_Nodes   = size(agg_map,2);
  set_index = zeros(1,n_Nodes);
  counter   = 1;
  for i = 1:n_Nodes
    if agg_map(i) == 0
      D{counter} = [i];
      counter    = counter + 1;
```

```
    else
      if set_index(agg_map(i)) == 0
        set_index(agg_map(i)) = counter;
        D{counter} = [i];
        counter = counter + 1;
      else
        D{set_index(agg_map(i))} = [D{set_index(agg_map(i))}  i];
      end
    end
  end
end

%funkcia na update agregacie
%agg_map je napr. v tvare 0 0 1 1 0 0 0 1 0 0 0 1
%pozor indexy tu uz predstavuju index mnoziny, nie uzla
function [D, L, fd_s1, nAtomic1] = AggD(D0, L0, agg_map, fd_s0, nAtomic0)
  L      = sum(agg_map == 0) + (sum(agg_map == 1) > 0);
  n_Sets = L0;
  if L0 ~= size(agg_map,2), display('Pozor nesedi pocet skupin L s agg_map!'); end %kontrola poctu skupin
  agg_index = 0;
  counter   = 1;
  fd_s1 = [];
  nAtomic1 = [];
  for i = 1:n_Sets
    if agg_map(i) == 0
      D{counter} = D0{i};
      fd_s1    = [fd_s1 fd_s0(i)];
      nAtomic1 = [nAtomic1 nAtomic0(i)];
      counter = counter + 1;
    else
      if agg_index == 0
        agg_index = counter;
        D{counter} = D0{i};
        fd_s1 = [fd_s1 0];
        nAtomic1 = [nAtomic1 nAtomic0(i)];
        counter = counter + 1;
      else
        D{agg_index} = [D{agg_index} D0{i}];
        nAtomic1(agg_index) = nAtomic1(agg_index) + nAtomic0(i);
      end
    end
  end

%funkcia na update koeficientov rezov optimality
 function [E,e,valid_cuts1] = AggCuts(s,t,k,E0,e0,p,D,L,valid_cuts0,agg_map,offset)
```

```
if sum(agg_map) > 0
 agg_ind    = [];
 no_agg_ind = [];
 for i=1:size(agg_map,2),
   if agg_map(i) == 1,
    agg_ind = [agg_ind i];
   else
    no_agg_ind = [no_agg_ind i];
   end
 end
valid_cuts1 = [];
E= []; e= [];
E1= E0(:,1:(size(E0,2)-L));
m= size(agg_ind,2);
L1 = L-m+1;
pos = 0;
for it = 1:s
  E_agg = zeros(1, size(E1,2));
  e_agg = 0;
  is_valid = 0;
  ctr = 0;
  pos2 = pos;
  for i= 1:L
      pos = pos + 1;
      if agg_map(i) == 1
        ctr = ctr + 1;
        E_agg =  E_agg + E1(pos,:);
        e_agg =  e_agg + e0(pos);
        is_valid = is_valid + valid_cuts0(pos);
      end
  end
  if is_valid > 1, is_valid = 1; end;

  ctr1 = 0;
  agg_inserted = 0;
  for i = 1:L
      pos2 = pos2 + 1;
      if (agg_map(i) == 1) && (agg_inserted == 0)
        ctr1 = ctr1 + 1;
        agg_inserted = 1;
        vect = zeros(1,L1);
        vect(ctr1) = -1;
        E = [E; E_agg vect];
        e = [e; e_agg];
        valid_cuts1 = [valid_cuts1; is_valid];
```

```
        elseif agg_map(i) == 0
          ctr1 = ctr1 + 1;
          vect = zeros(1,L1);
          vect(ctr1) = -1;
          E = [E; E1(pos2,:) vect];
          e = [e; e0(pos2)];
          valid_cuts1 = [valid_cuts1; valid_cuts0(pos2)];
        end
    end
  end
  else
    E = E0;
    e = e0;
    valid_cuts1 = valid_cuts0;
  end

function [agg_map,fd_s1] = CreateAggMap(sigma, s, L, valid_cuts, fd_s0,...
                                        num_atomic, r_treshold, MaxAgg, WarmUpPeriod, MinAggGroups)

  eps = 1.0e-9;
  zer = ones(1,L);
  cuts_exist = zeros(1,L);
  group = 1;
  for j=1:size(valid_cuts,1)
      if valid_cuts(j) == 1
        if abs(sigma(j)) > eps
          zer(group) = 0;
        end
        cuts_exist(group) = 1;
      end

      group = group + 1;
      group = mod(group - 1, L) + 1;
  end

  zer = zer.*cuts_exist;
  fd_s1 = fd_s0 + zer;
  ratio = fd_s1 / s;
  agg   = ratio > r_treshold;

  if (L-sum(agg)+1 > MinAggGroups) & (s > WarmUpPeriod)

    agg_map   = zeros(1,L);
    %osetrenie MaxAgg
     atomic_size = 0;
```

```
      for j=1:size(agg,2)
        if agg(j) == 1,
          atomic_size = atomic_size + num_atomic(j);
          if atomic_size <= MaxAgg
             agg_map(j) = 1;
          else
             break;
          end
        end
      end

      if sum(agg_map) == 1, agg_map = zeros(1,L); end
   else
     agg_map = zeros(1,L);
   end
%funkcia na update dat po agregacii - zmeni aj thety
function [W, lb, ub, c, D] = UpdateData(W0, lb0, ub0, c0, L1, L2, D0)
  dif  = L1-L2;
  s    = size(W0,2);
  W    = W0(:,1:(s-dif));
  s    = size(lb0,2);
  lb   = lb0(1:(s-dif));
  s    = size(ub0,2);
  ub   = ub0(1:(s-dif));
  s    = size(c0,2);
  c    = c0(1:(s-dif));
  s    = size(D0,2);
  D    = D0(:,1:(s-dif));

%funkcia na transformaciu indexov
%D{t-1}{j}{ni{t-1}{j}-1}{d}
function [off] = Offset(D,L,v,t,j,pos)
  off = 0;
  for i= 1:j-1
   ni = v{t}{i}-pos;   %-1
   for q= 1:L{t}{i}{ni}
    off= off+size(D{t}{i}{ni}{q}, 2);
   end
  end
```