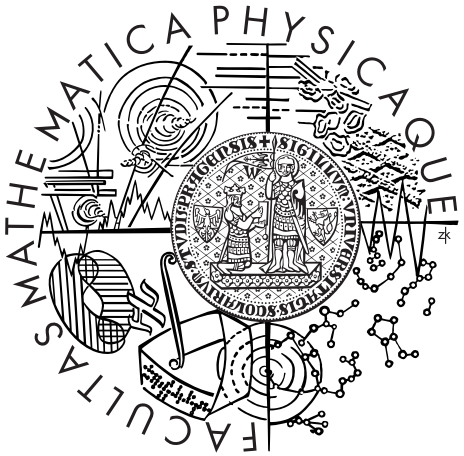
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Lukáš Adam

Nelinearity v úlohách stochastického programování: aplikace na řízení rizika

Katedra pravděpodobnosti a matematické statistiky

Vedoucí diplomové práce:	prof. RNDr. Jitka Dupačová, DrSc.
Studijní program:	Matematika
Studijní obor:	Pravděpodobnost, matematická statistika a ekonometrie
Studijní plán:	Ekonometrie

Praha 2011

Na tomto místě bych chtěl poděkovat vedoucí diplomové práce prof. RNDr. Jitce Dupačové, DrSc. za její cenné rady a připomínky, které mi při tvorbě práce nesmírně pomohly.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 3.4.2011 Lukáš Adam

Název práce: Nelinearity v úlohách stochastického programování: aplikace na řízení rizika

Autor: Lukáš Adam

Katedra: Katedra pravděpodobnosti a matematické statistiky

Vedoucí diplomové práce: prof. RNDr. Jitka Dupačová, DrSc.

Abstrakt: V této práci se zabýváme úlohami dvojstupňového stochastického programování. Práce je rozdělena do čtyř kapitol. V první z nich zavádíme nezbytné teoretické minimum pro další kapitoly, nejprve pro lineární a poté pro polyedrické účelové funkce. V druhé a třetí kapitole se zabýváme některými algoritmy a softwarem, který je určený pro úlohy stochastického programování. Většina algoritmů vychází z L–shaped metody, jejíž varianty uvádíme postupně pro lineární, polyedrický, kvadratický a konvexní případ. V poslední kapitole aplikujeme vybrané algoritmy a software na praktický problém obsahující podmíněnou hodnotu v riziku a provedeme jejich porovnání.

Klíčová slova: dvojstupňové programování, algoritmy, software, CVaR.

Title: Nonlinearities in stochastic programming problems. Application to risk control

Author: Lukáš Adam

Department: Department of Probability and Mathematical Statistics

Supervisor: prof. RNDr. Jitka Dupačová, DrSc.

Abstract: In this work we are interested in two–stage stochastic problems. Presented work is divided into four chapters. In the first one we introduce the necessary background for further chapters, at first for linear, later on for polyhedral objective function. In second and third chapters we present several algorithms and software, which can be used to solve stochastic problems. Most of the algoritms are based on L–shaped method, whose variant range from linear to polyhedral, quadratic and finally covnex problem. In the last chapter we apply some of mentioned algorithms and software on a practical problem involving conditional value at risk and compare them.

Keywords: two–stage programming, algorithms, software, CVaR.

Obsah

Úvod

Diplomová práce je věnována optimalizaci, při které jsou koeficienty náhodné povahy. Důraz je kladen na lineární dvojstupňové programování, které je dále rozšířeno do polyedrického případu a v některých částech jsou zmíněna další rozšíření, ať už do kvadratického nebo konvexního případu. Dále jsou v práci rozebírány algoritmy a software pro výpočty výše zmíněných úloh a dále aplikace těchto úloh do finančního sektoru, konkrétně pro výpočet podmíněné hodnoty v riziku. Na závěr je přiložena numerická studie srovnávající vybrané algoritmy a vybraný software.

Práce je rozdělena do čtyř kapitol a apendixu. V první kapitole definujeme problém lineárního dvojstupňového programování, ukážeme základní vlastnosti účelové funkce, čímž položíme základ pro druhou kapitolu, ve které se zabýváme algoritmy. Dále ukazujeme, ve kterých případech existuje přípustné řešení a kdy stanovujeme podmínky, za kterých je toto řešení optimální. Teorie lineárních problémů je dále rozpracována do problémů polyedrických.

Druhá kapitola je věnována algoritmům, hlavně algoritmu L–shaped. Nejprve ukazujeme základní verzi pro lineární případ, pro kterou je nutný předpoklad omezenosti množiny přípustných řešení. Poté tento předpoklad neuvažujeme a ukážeme, jaké úpravy jsou nutné pro základní verzi. Dále ukazujeme další algoritmy, které na L–shaped algoritmus navazují a měly by ho zrychlovat. Jedná se například o multicut verzi nebo regularizovanou dekompozici. V dalších částech algoritmus rozpracováváme postupně do polyedrického, kvadratického a konečně obecného konvexního případu. Bohužel L–shaped algoritmus je možný použít pouze v případě polyedričnosti množiny přípustných řešení. Pokud je tato množina konvexní, přikládáme ještě progressive hedging algoritmus. L–shaped algoritmus použijeme dále ve čtvrté kapitole, ve které ho aplikujeme na model s polyedrickou účelovou funkcí.

Ve třetí kapitole zmiňujeme software, který je schopen řešit úlohy dvojstupňového programování. Nejdříve se zabýváme deterministickým ekvivalentem, který je možné řešit pomocí obecných modelovacích jazyků, kterým je například GAMS. Další část se zabývá SMPS formátem, což je formát přímo určený k ukládání problémů stochastického programování. Zatímco obecné modelovací jazyky nejsou schopny využít speciální tvar úloh dvojstupňového programování, existuje software, který využívá algoritmy vyvinuté pro dvojstupňové úlohy. Mezi tento software patří například SLP–IOR nebo SPInE.

Čtvrtá kapitola je vyústění předchozích tří, uvažuje v ní optimalizační problém, v jehož účelové funkci se vyskytuje podmíněná hodnota v riziku. Tento problém převádíme na dvojstupňovou úlohu, na kterou dále aplikujeme L–shaped algoritmus, který se v tomto případě značně zjednoduší. Na závěr ukazujeme, jak je který software efektivní v řešení tohoto problému.

V apendixu podáváme stručný přehled vybraných definic a vět, které jsou potřeba pro důkaz vět ve vlastní práci.

Kapitola 1

Teoretický úvod

V této kapitole se zaobíráme základy dvojstupňového programování. Existuje mnoho knih, ve kterých je toto téma podrobně rozebráno, citujme například [5], [10], [21] nebo [22]. Koncepce kapitoly odpovídá koncepci [22].

Kapitola je rozdělena do dvou částí, v první z nich se zabíráme lineárním dvojstupňovým programováním, v druhé části pak přejdeme k polyedrickým účelovým funkcím. Pro oba případy podáváme základní definice, uvádíme duální úlohu a ukazujeme vlastnosti účelové funkce. Na závěr předkládáme podmínky, za kterých je přípustné řešení optimální.

1.1 Lineární případ

Pro dvojstupňové programování existují dva základní předpoklady. Prvním z nich je znalost rozdělení všech náhodných veličin, druhým předpokladem je pak podmínka, že volba x nemůže ovlivnit realizaci náhodného vektoru ξ. Nejprve tyto problémy definujme.

Definice 1.1. Definujme problém dvojstupňového lineárního programování následovně: jako první stupeň definujme problém

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \; & c'x + \mathrm{E}\,\mathcal{Q}(x,\xi) \\ s.t. \; & Ax = b, \\ & x \geq 0, \end{aligned} \tag{1.1}$$

kde $\mathcal{Q}(x,\xi)$ je optimální řešení problému druhého stupně

$$\begin{aligned} \min_{y\in\mathbb{R}^m} \; & q'y \\ s.t. \; & Tx + Wy = h, \\ & y \geq 0 \end{aligned} \tag{1.2}$$

a $\xi = (q, h, T, W)$ je náhodný vektor se známým rozdělením, jehož nosič označme Ξ. Připomeňme, že nosič náhodného vektoru je nejmenší uzavřená množina taková, že platí $\mathrm{P}(\xi \in \Xi) = 1$. Z kontextu bude většinou zřejmé, jestli se jedná o náhodnou veličinu ξ nebo o její realizaci $\xi(\omega)$. Pokud nebude hrozit omyl, budeme v obou případech psát pouze ξ.

Položme dále $K_1 = \{x \geq 0;\ Ax = b\}$ a $K_2 = \{x;\ \mathrm{E}\,\mathcal{Q}(x,\xi) < \infty\}$. Tyto množiny budeme dále označovat množiny přípustných řešení pro první, respektive pro druhý stupeň. Při optimalizaci úlohy (1.1)–(1.2) se stačí omezit na $x \in X = K_1 \cap K_2$. Pokud je $X = \emptyset$, pak optimální hodnota úlohy (1.1) je $+\infty$.

Druhý stupeň (1.2) můžeme převést na duální problém, který má tvar

$$\begin{aligned} &\max_{\pi} \pi'(h - Tx) \\ &s.t.\ W'\pi \leq q. \end{aligned} \tag{1.3}$$

Pokud mají oba problémy (1.2) a (1.3) alespoň jedno přípustné řešení, mají i optimální řešení a optimální hodnoty jejich účelových funkcí jsou stejné.

Dvojstupňové programování má jednoduchou interpretaci. Na začátku musíme provést rozhodnutí x a po realizaci náhodného vektoru ξ provedeme korekci volbou vektoru y. Nejjednodušším příkladem je jednoduchá kompenzace, při které volíme $W = [I, -I]$, T a q jsou deterministické a q je nezáporný. Pro jednoduchost označme $h(\xi) = \xi,\ q' = (q_1', q_2'),\ y' = (y_1', y_2')$. Potom problém (1.2) vypadá následovně:

$$\begin{aligned} &\min_{y_1, y_2} q_1' y_1 + q_2' y_2 \\ &s.t.\ Iy_1 - Iy_2 = \xi - Tx, \\ &\quad y_1 \geq 0,\ y_2 \geq 0. \end{aligned}$$

Díky nezápornosti q jsou optimálními řešeními $y_1 = (\xi - Tx)^+$ a $y_2 = (\xi - Tx)^-$ a tedy problém (1.1) můžeme přepsat do tvaru

$$\begin{aligned} &\min_{x} c'x + \mathrm{E}[q_1'(\xi - Tx)^+ + q_2'(\xi - Tx)^-] \\ &s.t.\ Ax = b, \\ &\quad x \geq 0. \end{aligned}$$

Vidíme, že zatímco podmínka $Ax = b$ musí platit vždy, nesplnění podmínky $Tx = \xi$ je penalizováno pomocí vektoru q.

Z hlediska optimalizace bychom chtěli, aby měl problém (1.1) hezké vlastnosti, mezi něž patří například konvexita účelové funkce. Pro její konvexitu je postačující, aby člen $\phi(x) = \mathrm{E}\,\mathcal{Q}(x,\xi)$ byl konvexní. Na následujících několika stránkách se budeme zabývat vlastnostmi této funkce.

Pokud v druhém stupni (1.2) zafixujeme ξ, můžeme na tento stupeň pohlížet jako na problém lineárního programování s parametrem x. Jako první položme

$$\Pi(q, W) = \{\pi;\ W'\pi \leq q\}$$

množinu přípustných řešení duálního problému (1.3). Je zřejmé, že tato množina je konvexní, uzavřená a polyedrická. Množina přípustných řešení primární úlohy druhého stupně (1.2) na x závisí, zatímco množina přípustných řešení duálu (1.3) závisí pouze na ξ. Pokud je množina $\Pi(q, W)$ neprázdná, dostáváme při fixním ξ vlastnost, že pro libovolné x je $\mathcal{Q}(x,\xi) > -\infty$.

Položme dále

$$s_{q,W}(\chi) = \inf\{q'y;\ Wy = \chi, y \geq 0\}.$$

Platí $Q(x,\xi) = s_{q,W}(h - Tx)$ a $s_{q,W}$ je konvexní funkce. Pokud $\Pi(q, W) \neq \emptyset$, pak platí

$$s_{q,W}(\chi) = \sup_{\pi \in \Pi(q,W)} \pi'\chi,$$

neboli $s_{q,W}$ je opěrnou funkcí množiny $\Pi(q,W)$. Dále z $\Pi(q,W) \neq \emptyset$ plyne, že $s_{q,W}$ je vlastní a pokud je $s_{q,W}$ konečná alespoň v jednom bodě, je polyedrická a pozitivně homogenní. Pokud je naopak $\Pi(q,W) = \emptyset$, $s_{q,W}$ nabývá pouze nekonečných hodnot, přičemž hodnoty $-\infty$ se nabývá na konvexní množině.

Připomeňme, že funkce $s_{q,W}$ je polyedrická právě tehdy, pokud je vlastní, konvexní, zdola polospojitá, $\operatorname{dom} s_{q,W}$ je konvexní uzavřená polyedrická množina a $s_{q,W}$ je na $\operatorname{dom} s_{q,W}$ po částech lineární. Podmínky na $\operatorname{dom} s_{q,W}$ jsou splněny, protože $\operatorname{dom} s_{q,W} = \operatorname{pos} W$. Jako optimální hodnota lineárního programování s parametrem, je $s_{q,W}$ po částech lineární. Konečně $s_{q,W}$ je zdola polospojitá, neboť její doména je uzavřená neprázdná konvexní množina a $s_{q,W}$ je na ní konvexní a spojitá.

Věta 1.2. *Pro pevné ξ je funkce $\mathcal{Q}(\cdot,\xi)$ konvexní. Pokud je množina $\Pi(q,W)$ neprázdná a problém* (1.2) *má přípustné řešení alespoň pro jedno x, pak je funkce $\mathcal{Q}(\cdot,\xi)$ polyedrická.*

Důkaz. Protože platí $\mathcal{Q}(x,\xi) = s_{q,W}(h - Tx)$, plynou požadované vlastnosti z vlastností funkce $s_{q,W}$. □

Definujme dále

$$\mathcal{D}(x,\xi) = \arg \max_{\pi \in \Pi(q,W)} \pi'(h - Tx)$$

množinu optimálních řešení duálního problému (1.3).

Věta 1.3. *Nechť pro nějaké $x_0 \in \mathbb{R}^n$ a $\xi \in \Xi$ je $\mathcal{Q}(x_0,\xi)$ konečná. Potom $\mathcal{Q}(\cdot,\xi)$ je subdiferencovatelná v x_0 a platí*

$$\partial \mathcal{Q}(x_0,\xi) = -T'\mathcal{D}(x_0,\xi)$$

Důkaz. Protože $\mathcal{Q}(x_0,\xi)$ je konečná, je $\Pi(q,W)$ neprázdná. Z toho ihned plyne, že funkce $s_{q,W}$ je polyedrická. Jsou tedy splněny předpoklady věty A.8 a pro $\chi_0 = h - Tx_0$ platí

$$\partial s_{q,W}(\chi_0) = \arg\max_{\pi} \{\pi'\chi_0 - (s_{q,W})^*(\pi)\},$$

kde $(s_{q,W})^*$ je sdružená funkce k funkci $s_{q,W}$. Sdružená funkce je definována v dodatku v definici A.6.

Pro funkci

$$f(\pi) = \begin{cases} 0 & \pi \in \Pi(q,W) \\ +\infty & \text{jinak} \end{cases}$$

platí $s_{q,W} = f^*$. Funkce f je zřejmě vlastní, a protože $\Pi(q,W)$ je konvexní a uzavřená, je f konvexní a zdola polospojitá. Podle věty A.7 pak platí $f = \operatorname{lsc} f = f^{**} = s^*_{q,W}$.

Nakonec platí

$$\partial s_{q,W}(\chi_0) = \arg\max_{\pi} \{\pi'\chi_0 - f(\pi)\} = \arg \max_{\pi \in \Pi(q,W)} \{\pi'\chi_0\}$$
$$\partial \mathcal{Q}(x_0,\xi) = -T' \arg \max_{\pi \in \Pi(q,W)} \{\pi'(h - Tx_0)\}.$$

□

Z věty 1.3 ihned plyne, že $\mathcal{Q}(\cdot, \xi)$ je diferencovatelná v x_0, pokud pro x_0 a ξ má duální problém (1.3) pouze jediné optimální řešení. Tato věta bude mít klíčový význam v L-shaped algoritmu, ve kterém budeme $\mathcal{Q}(\cdot, \xi)$ aproximovat zdola pomocí maxima lineárních funkcí.

Definice 1.4. Řekneme, že dvojstupňový problém (1.1)–(1.2) má pevnou kompenzaci, pokud W nezávisí na realizaci ξ. Stejný problém má úplnou kompenzaci, pokud pro každé χ existuje $y \geq 0$ takové, že platí $Wy = \chi$. Při relativně úplné kompenzaci musí platit, že pro všechna $x \in K_1$ skoro jistě existuje $y \geq 0$ splňující $Wy = h - Tx$.

Pro úplnou kompenzaci platí, že $\operatorname{pos} W$ je roven celému prostoru. Připomeňme, že $\operatorname{pos} W$ jsou lineární kombinace s kladnými koeficienty sloupců matice W. Relativní kompenzace je ekvivalentní s následující podmínkou:

$$\forall x \in K_1 \; \exists \; \Xi_0 \subset \Xi, \; \mathrm{P}(\Xi_0) = 0 \; \forall \xi \in \Xi \setminus \Xi_0 : \mathcal{Q}(x, \xi) < +\infty.$$

Protože je při pevné kompenzaci matice W pevná, budeme v tomto případě místo $\Pi(q, W)$ psát pouze $\Pi(q)$ a místo $s_{q,W}$ pouze s_q.

Zabývejte se nyní tím, kdy je funkce $\phi(x) = \mathrm{E}\, \mathcal{Q}(x, \xi)$ definována. Pokud nemá druhý stupeň (1.2) žádné přípustné řešení, platí $\mathcal{Q}(x, \xi) = +\infty$. Pokud naopak druhý stupeň neomezeně klesá, dostaneme $\mathcal{Q}(x, \xi) = -\infty$. Aby byla $\phi(x)$ korektně definována musí být $\mathcal{Q}(x, \cdot)$ měřitelná a dále pak buď $\mathrm{E}[\mathcal{Q}(x, \xi)_+]$ nebo $\mathrm{E}[\mathcal{Q}(x, \xi)_-]$ musí být konečné. Měřitelnost $\mathcal{Q}(x, \cdot)$ je zajištěna tím, že $\mathcal{Q}(x, \cdot)$ je optimální hodnota úlohy lineárního programování. Zabývejme se nyní druhou podmínkou.

Lemma 1.5. *Nechť kompenzace je pevná a nechť $\Pi(q_0)$ je neprázdná pro nějaké q_0. Potom existuje konstanta $\kappa > 0$ závisející pouze na W taková, že pro každé q, pro které je $\Pi(q)$ neprázdná množina, platí*

$$s_q(\cdot) \leq s_{q_0}(\cdot) + \kappa ||q - q_0|| \; || \cdot ||.$$

Důkaz. Položme v Hoffmanově větě A.9 $\mathcal{M}(b) = \Pi(q_0)$, čímž dostaneme existenci konstanty $\kappa > 0$ závislé pouze na W takové, že platí

$$\operatorname{dist}(\pi, \Pi(q_0)) \leq \kappa ||(W'\pi - q_0)_+||,$$

kde π je libovolné. Hoffmanovu větu můžeme použít, protože W je pevná a $\Pi(q_0) \neq \emptyset$.

Pro $\pi \in \Pi(q)$ platí $(W'\pi - q_0)_+ \leq (q - q_0)_+$, a tedy

$$||(W'x - q_0)_+|| \leq ||(q - q_0)_+|| \leq ||q - q_0||.$$

Celkově dostaneme

$$\operatorname{dist}(\pi, \Pi(q_0)) \leq \kappa ||q - q_0||,$$

z čehož plyne

$$\Pi(q) \subset \Pi(q_0) + \kappa ||q - q_0|| B,$$

kde B je uzavřená jednotková koule.

Protože $\Pi(q)$ i $\Pi(q_0)$ jsou neprázdné množiny, jejich opěrné funkce jsou s_q a s_{q_0}. Protože opěrná funkce k B je norma, dostaneme podle poznámky za definicí A.3

$$s_q(\cdot) \leq s_{q_0}(\cdot) + \kappa ||q - q_0|| \; || \cdot ||.$$

□

Důsledek 1.6. Nechť kompenzace je pevná. Potom existuje konstanta $\kappa > 0$ závisející pouze na W taková, že pro každé q, pro které je $\Pi(q)$ neprázdná množina a pro každé $\chi \in \operatorname{pos} W$, platí

$$s_q(\chi) \leq \kappa ||q|| \; ||\chi||.$$

Důkaz. V lemmatu 1.5 zvolme $q_0 = 0$. Protože platí $0 \in \Pi(0)$, jsou splněny předpoklady lemmatu, a tedy existuje konstanta κ taková, že pro libovolné q splňující $\Pi(q) \neq \emptyset$ platí $s_q(\chi) \leq s_0(\chi) + \kappa ||q|| \; ||\chi|| = \kappa ||q|| \; ||\chi||$, neboť pro $\chi \in \operatorname{pos} W$ je $s_0(\chi) = 0$. □

Věta 1.7. *Nechť je kompenzace pevná a nechť platí*

$$\mathrm{E}[||q|| ||h||] < +\infty, \; \mathrm{E}[||q|| ||T||] < +\infty. \tag{1.4}$$

Potom pro $x \in \mathbb{R}^n$ je $\mathrm{E}[\mathcal{Q}(x,\xi)_+] < +\infty$, právě když je skoro jistě splněno

$$h - Tx \in \operatorname{pos} W.$$

Důkaz. Nechť $\mathrm{E}[\mathcal{Q}(x,\xi)_+] < +\infty$ a pro spor předpokládejme existenci Ξ_1 takové, že $\mathrm{P}(\Xi_1) > 0$ a pro všechna $\xi \in \Xi_1$ platí $h - Tx \notin \operatorname{pos} W$. To ale znamená $\mathcal{Q}(x,\xi) = +\infty$, a tedy $\mathrm{E}[\mathcal{Q}(x,\xi)_+] = +\infty$, čímž jsme došli ke sporu.

Dokažme nyní obrácenou implikaci. Dle důsledku 1.6 existuje kladná konstanta κ taková, že pro všechna q splňující $\Pi(q) \neq \emptyset$ a pro všechna $\chi \in \operatorname{pos} W$ platí $s_q(\chi) \leq \kappa ||q|| \; ||\chi||$. Pokud pro nějaké ξ platí $\Pi(q) = \emptyset$, pak pro všechna $\chi \in \operatorname{pos} W$ je $s_q(\chi) = -\infty$. Dohromady dostáváme, že pro libovolné $\chi \in \operatorname{pos} W$ platí $[s_q(\chi)]_+ \leq \kappa ||q|| \; ||\chi||$.

Volbou $\chi = h - Tx$ poté dostáváme

$$\begin{aligned} s_q(h - Tx)_+ &\leq \kappa ||q|| \; ||h - Tx|| \leq \kappa ||q|| (||h|| + ||T|| \; ||x||) \\ \mathrm{E}[s_q(h - Tx)_+] &\leq \kappa \, \mathrm{E}[||q|| \; ||h||] + \kappa ||x|| \, \mathrm{E}[||q|| \; ||T||] < +\infty. \end{aligned}$$

□

Příklad 1.8. Na předchozí větě si ukažme důležitost pevné kompenzace. Předpokládejme, že h má diskrétní rozdělení, kde pro $k \in \mathbb{N}$ platí $\mathrm{P}(h = k) = \frac{1}{c}\frac{1}{k^2}$, kde $c = \sum_{k=1}^{+\infty} \frac{1}{k^2}$. Dále nechť $W = \frac{1}{h^2}$ a $q = \frac{1}{h}$. Pokud položíme $x = 0$, dostaneme následující optimalizační úlohu

$$\begin{aligned} \min_y & \frac{1}{h} y \\ s.t. \; & \frac{1}{h^2} y = h, \\ & y \geq 0. \end{aligned}$$

Předchozí věta má splněné předpoklady, neboť $\mathrm{E}\, qh = \mathrm{E}\, 1 = 1 < +\infty$. Úloha má jediné řešení $\bar{y} = h^3$ a optimální hodnota účelové funkce je $\mathcal{Q}(0,\xi) = h^2$, takže $\mathrm{E}\, \mathcal{Q}(0,\xi) = \mathrm{E}\, h^2 = +\infty$.

Předchozí úloha je ekvivalentní s úlohou

$$\begin{aligned} \min_y & \frac{1}{h} y \\ s.t. \; & y = h^3, \\ & y \geq 0. \end{aligned}$$

Tato úloha má pevnou kompenzaci, ale zase není splněn předpoklad $\mathrm{E}\, qh < +\infty$.

Věta 1.9. *Nechť je kompenzace pevná, pro skoro každé q množina $\Pi(q)$ neprázdná a nechť platí* (1.4). *Potom je $\phi(x)$ korektně definována a je konvexní, vlastní, zdola polospojitá, na* $\operatorname{dom}\phi$ *Lipschitzovsky spojitá a její doména je konvexní uzavřená množina, pro kterou platí*

$$\operatorname{dom}\phi = \{x \in \mathbb{R}^n; h - Tx \in \operatorname{pos} W s.j.\} \tag{1.5}$$

Důkaz. Protože $\Pi(q)$ je neprázdná skoro jistě, platí skoro jistě $\mathcal{Q}(x,\xi) = s_q(h - Tx)$. Protože $\Pi(q)$ je uzavřená, existuje bod této množiny, který je nejblíže nule. Označme tento bod $\pi(q)$. Potom podle věty A.9 existuje konstanta κ, která nezávisí na q, taková, že platí

$$||\pi(q)|| = \operatorname{dist}(0, \Pi(q)) \leq \kappa||(-q)_+|| \leq \kappa||q||.$$

Potom skoro jistě platí

$$\begin{aligned} s_q(h - Tx) &\geq \pi(q)'(h - Tx) \geq -||\pi(q)||\ ||h - Tx|| \\ &\geq -||\pi(q)||\ (||h|| + ||T||\ ||x||) \\ &\geq -\kappa||q||\ (||h|| + ||T||\ ||x||). \end{aligned} \tag{1.6}$$

Protože $\phi(x) = \mathrm{E}\,\mathcal{Q}(x,\xi) = \mathrm{E}\,s_q(h - Tx)$, pro všechna x platí $\phi(x) > -\infty$.

Uvažujeme–li posloupnost $x_n \to x$, pak platí

$$\begin{aligned} \liminf \phi(x_n) &= \liminf \int \mathcal{Q}(x_n,\xi) d\,\mathrm{P} \geq \int \liminf \mathcal{Q}(x_n,\xi) d\,\mathrm{P} \\ &\geq \int \mathcal{Q}(x,\xi) d\,\mathrm{P} = \phi(x). \end{aligned}$$

V první nerovnosti jsme použili Fatouovo lemma A.15, kde pro výpočet minoranty využijeme (1.6). Pro všechna n pak bude platit

$$\mathcal{Q}(x_n,\xi) \geq -\kappa||q||\ (||h|| + ||T||\ ||x_n||) \geq -\kappa||q||\ (||h|| + ||T||\sup_k ||x_k||),$$

což je integrovatelná minoranta nezávislá na n. Tímto jsme dokázali, že ϕ je zdola polospojitá v bodě x. Pokud bychom tento postup opakovali pro všechny body, dostali bychom dolní polospojitost na celém prostoru.

Konvexita ϕ plyne přímo z definice. Uvažujme dva body x_1, x_2 a $\lambda \in [0,1]$. Potom platí

$$\begin{aligned} \phi(\lambda x_1 + (1-\lambda)x_2) &= \mathrm{E}\,\mathcal{Q}(\lambda x_1 + (1-\lambda)x_2, \xi) \\ &\geq \mathrm{E}[\lambda\mathcal{Q}(x_1,\xi) + (1-\lambda)\mathcal{Q}(x_2,\xi)] = \lambda\phi(x_1) + (1-\lambda)\phi(x_2). \end{aligned}$$

Konvexnost a uzavřenost $\operatorname{dom}\phi$ plyne ihned z konvexnosti a dolní polospojitosti ϕ. Tvar množiny (1.5) plyne přímo z věty 1.7.

Víme, že platí následující

$$\begin{aligned} s_q(\chi_1) - s_q(\chi_2) &\leq \kappa||q||(||\chi_1|| - ||\chi_2||) \leq \kappa||q||\ ||\chi_1 - \chi_2|| \\ s_q(h - Tx_1) - s_q(h - Tx_2) &\leq \kappa||q||\ ||T||\ ||x_1 - x_2|| \\ \phi(x_1) - \phi(x_2) &\leq \kappa\,\mathrm{E}(||q||\ ||T||)||x_1 - x_2||. \end{aligned}$$

Díky symetrii dostaneme $|\phi(x_1) - \phi(x_2)| \leq \kappa\,\mathrm{E}(||q||\ ||T||)||x_1 - x_2||$, neboli ϕ je Lipschitzovsky spojitá. □

Věta 1.10. *Předpokládejme, že ϕ je vlastní a její doména má neprázdný vnitřek. Potom pro každé $x_0 \in \operatorname{dom}\phi$ platí*

$$\partial\phi(x_0) = -\operatorname{E}[T'\mathcal{D}(x_0,\xi)] + \mathcal{N}_{\operatorname{dom}\phi}(x_0),$$

kde $\mathcal{N}_C(x)$ označuje normální kužel pro množinu C a bod x. Definici normálního kužele přikládáme v apendixu.

Důkaz. Z věty plyne A.13 plyne, že $\mathcal{Q}(\cdot,\cdot)$ je měřitelná funkce. Protože je $\mathcal{Q}(\cdot,\xi)$ zdola polospojitá, plyne z věty A.12, že $\mathcal{Q}(x,\xi)$ je náhodná zdola polospojitá funkce. Protože podle věty 1.2 je $\mathcal{Q}(\cdot,\xi)$ i konvexní, jsou splněny předpoklady věty A.14, a tedy platí

$$\partial\phi(x_0) = \operatorname{E}\partial\mathcal{Q}(x_0,\xi) + \mathcal{N}_{\operatorname{dom}\phi}(x_0).$$

Protože $\phi(x_0)$ je konečná, je skoro jistě konečná i $\mathcal{Q}(x_0,\xi)$ a k dokončení důkazu stačí použít větu 1.3. □

Značné zjednodušení dostaneme, pokud ξ bude nabývat pouze konečně mnoha hodnot. Tímto případem se budeme zabývat ve zbytku této části, nejdříve zmíníme vlastnosti ϕ, dále pak podmínky optimality pro původní úlohu. Nechť ξ nabývá hodnot $\xi_1,\dots,\xi_K$ postupně s pravděpodobnostmi $p_1,\dots,p_K$. Potom ϕ můžeme napsat jako

$$\phi(x) = \operatorname{E}\mathcal{Q}(x,\xi) = \sum_{k=1}^{K} p_k \mathcal{Q}(x,\xi_k). \tag{1.7}$$

V tomto výpočtu položme $+\infty + (-\infty) = +\infty$, čímž dosáhneme toho, že $\phi(x)$ je korektně definována pro libovolné $x \in K_1$.

Původní problém (1.1) můžeme přepsat do jeho deterministického ekvivalentu

$$\begin{aligned} \min_{x,y_1,\dots,y_K} \; & c'x + \sum_{k=1}^{K} p_k q_k' y_k \\ s.t.\; & T_k x + W_k y_k = h_k,\; k = 1,\dots,K, \\ & Ax = b, \\ & x \geq 0,\; y_k \geq 0,\; k = 1,\dots,K. \end{aligned} \tag{1.8}$$

Zatímco v (1.7) jsme potřebovali dodefinovat součet $-\infty$ a $+\infty$, v případě (1.8) to není nutné, protože pokud pro všechna $x \in K_1$ existuje k takové, že $\mathcal{Q}(x,\xi_k) = +\infty$, pak problém (1.8) nemá přípustné řešení a jeho optimální hodnota je automaticky $+\infty$.

O výhodách a nevýhodách zápisu (1.8) pojednáme později, obecně můžeme říct, že výhoda je v tom, že dostaneme problém klasického lineárního programování, který můžeme řešit běžně dostupnými solvery. Nevýhoda je v jeho velikosti, kdy při velkém počtu možných realizací náhodného vektoru ξ se může stát, že se problém nevejde do paměti počítače a prakticky je tedy neřešitelný.

Věta 1.11. *Nechť je Ξ konečná množina a ϕ má konečnou hodnotu v alespoň jednom bodě $\bar{x}$. Potom ϕ je polyedrická a pro každé $x_0 \in \operatorname{dom}\phi$ a platí*

$$\partial\phi(x_0) = \sum_{k=1}^{K} p_k \partial\mathcal{Q}(x_0,\xi_k).$$

Důkaz. Protože $\phi(\bar{x})$ je konečná, jsou konečné i $\mathcal{Q}(\bar{x}, \xi_k)$ pro všechna k. Potom $\Pi(q_k, W_k)$ jsou neprázdné a podle věty 1.2 jsou $\mathcal{Q}(\cdot, \xi_k)$ polyedrické. Lineární kombinace s kladnými koeficienty polyedrických funkcí je polyedrická funkce a tedy ϕ je též polyedrická.

Protože $\Pi(q_k, W_k)$ jsou neprázdné, implikuje $x_0 \in \operatorname{dom}\phi$ konečnost $\phi(x_0)$. Můžeme použít Moreau-Rockafellarovu větu A.5 a protože $p_k\mathcal{Q}(\cdot, \xi_k)$ jsou polyedrické, platí požadované tvrzení. □

Věta 1.12. *Nechť Ξ je konečná a nechť $\bar{x} \in X$. Potom $\bar{x}$ je optimální řešení problému* (1.8) *právě tehdy, když existují $\pi_k \in \mathcal{D}(\bar{x}, \xi_k)$ a μ takové, že*

$$\begin{aligned} \sum_{k=1}^{K} p_k T_k' \pi_k + A'\mu &\leq c, \\ \bar{x}'(c - \sum_{k=1}^{K} p_k T_k' \pi_k - A'\mu) &= 0. \end{aligned} \tag{1.9}$$

Důkaz. Uvažujme problém (1.8) a sestrojme k němu duální úlohu. Problém (1.8) můžeme přepsat jako

$$\begin{aligned} &\min_{x, y_1, \ldots, y_K} (c', q_1', \ldots, q_K')(x', p_1 y_1', \ldots, p_K y_K')' \\ &s.t. \begin{pmatrix} A & 0 & \ldots & 0 \\ p_1 T_1 & W_1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ p_K T_K & 0 & \ldots & W_K \end{pmatrix} \begin{pmatrix} x \\ p_1 y_1 \\ \ldots \\ p_K y_K \end{pmatrix} = \begin{pmatrix} b \\ p_1 h_1 \\ \ldots \\ p_1 h_K \end{pmatrix} \\ &\qquad x \geq 0,\ p_k y_k \geq 0,\ k = 1, \ldots, K. \end{aligned} \tag{1.10}$$

Duální úloha tohoto problému pak bude

$$\begin{aligned} &\max_{\mu, \pi_1, \ldots, \pi_K} (b', p_1 h_1', \ldots, p_K h_K')(\mu', \pi_1', \ldots, \pi_K')' \\ &s.t. \begin{pmatrix} A' & p_1 T_1' & \ldots & p_K T_K' \\ 0 & W_1' & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & W_K' \end{pmatrix} \begin{pmatrix} \mu \\ \pi_1 \\ \ldots \\ \pi_K \end{pmatrix} \leq \begin{pmatrix} c \\ q_1 \\ \ldots \\ q_K \end{pmatrix} \end{aligned} \tag{1.11}$$

Z duality lineárního programování víme, že přípustné řešení primárního problému je optimální, pokud existuje přípustné řešení duálního problému a jsou splněny podmínky komplementarity. Celkem to znamená existenci $\bar{x}, \bar{y}_k, \pi_k, \mu$ splňujících

$$\begin{aligned} \sum_{k=1}^{K} p_k T_k' \pi_k + A'\mu &\leq c, \\ \bar{x}'(c - \sum_{k=1}^{K} p_k T_k' \pi_k - A'\mu) &= 0, \\ W_k' \pi_k &\leq q_k,\ k = 1, \ldots K, \\ p_k \bar{y}_k'(W_k' \pi_k - q_k) &= 0,\ k = 1, \ldots K, \end{aligned}$$

kde navíc $x \in K_1$ a pro všechna k je splněno $T_k x + W_k y_k = h_k$. První dvě podmínky přesně odpovídají (1.9), zatímco poslední dvě podmínky znamenají, že π_k je optimální řešení duálního problému druhého stupně (1.3), neboli $\pi_k \in \mathcal{D}(\bar{x}, \xi_k)$. □

1.2 Polyedrický případ

V této části dojde k jistému zobecnění, kdy se od lineární účelové funkce posuneme k polyedrickým funkcím. Nejdříve uveďme ekvivalentní definici polyedrické funkce.

Definice 1.13. Funkce f_1 je polyedrická, pokud existují vektory c_i a skaláry $\alpha_i,\ i = 1, \dots, I_1$ a dále vektory a_j a skaláry $b_j,\ j = 1, \dots, J_1$ takové, že platí

$$f_1(x) = \begin{cases} \max_{i=1,\dots,I_1}(\alpha_i + c_i'x) & \text{pro } a_j'x \le b_j,\ j = 1, \dots, J_1 \\ +\infty & \text{jinak} \end{cases} \tag{1.12}$$

a dom f_1 je neprázdná.

Funkce f_2 je náhodná polyedrická funkce, pokud existují náhodné vektory $q_i = q_i(\xi)$ a náhodné skaláry $\gamma_i = \gamma_i(\xi),\ 1, \dots, I_2$ a dále náhodné vektory $d_j = d_j(\xi)$ a náhodné skaláry $r_j = r_j(\xi),\ j = 1, \dots, J_2$ takové, že platí

$$f_2(y, \xi) = \begin{cases} \max_{i=1,\dots,I_2}(\gamma_i(\xi) + q_i(\xi)'y) & \text{pro } d_j(\xi)'y \le r_j(\xi),\ j = 1, \dots, J_2 \\ +\infty & \text{jinak} \end{cases} \tag{1.13}$$

a dom $f_2(\cdot, \xi)$ je neprázdná skoro jistě.

První stupeň definujme jako

$$\begin{aligned} &\min_{x \in \mathbb{R}^n}\ g_1(x) + \mathrm{E}\,\mathcal{Q}(x, \xi) \\ &s.t.\ x \ge 0, \end{aligned} \tag{1.14}$$

kde g_1 je polyedrická funkce a $\mathcal{Q}(x, \xi)$ je optimální hodnota problému druhého stupně

$$\begin{aligned} &\min_{y \in \mathbb{R}^m}\ g_2(y, \xi) \\ &s.t.\ T(\xi)x + W(\xi)y = h(\xi), \\ &\quad y \ge 0, \end{aligned} \tag{1.15}$$

kde g_2 je náhodná polyedrická funkce.

Lineární a polyedrické problémy mezi sebou můžeme převádět. Každý lineární problém je zjevně polyedrický. Na druhou stranu první stupeň polyedrického problému (1.14) můžeme napsat jako

$$\begin{aligned} &\min_{x,u}\ u + \mathrm{E}\,\mathcal{Q}(x, \xi) \\ &s.t.\ \alpha_i + c_i'x \le u,\ i = 1, \dots, I_1, \\ &\quad a_j'x \le b_j,\ j = 1, \dots, J_1, \\ &\quad x \ge 0 \end{aligned} \tag{1.16}$$

a druhý stupeň (1.15) jako

$$\begin{aligned} &\min_{y,v}\ v \\ &s.t.\ T(\xi)x + W(\xi)y = h(\xi), \\ &\quad \gamma_i(\xi) + q_i(\xi)'y \le v,\ i = 1, \dots, I_2, \\ &\quad d_j(\xi)'y \le r_j(\xi),\ j = 1, \dots, J_2, \\ &\quad y \ge 0. \end{aligned} \tag{1.17}$$

Pro jednoduchost se budeme zabývat pouze prvním způsobem zápisu.

Definujme dále

$$f_1(x) = \begin{cases} g_1(x) & \text{pro } x \geq 0 \\ +\infty & \text{jinak.} \end{cases}$$

a

$$f_2(y,\xi) = \begin{cases} g_2(y,\xi) & \text{pro } y \geq 0 \\ +\infty & \text{jinak.} \end{cases}$$

Potom problém (1.14) je ekvivalentní problému

$$\min_{x\in\mathbb{R}^n} \; f_1(x) + \mathrm{E}\,\mathcal{Q}(x,\xi) \tag{1.18}$$

a problém (1.15) je ekvivalentní problému

$$\begin{aligned} &\min_{y\in\mathbb{R}^m} \; f_2(y,\xi) \\ &\quad s.t.\; T(\xi)x + W(\xi)y = h(\xi). \end{aligned} \tag{1.19}$$

Protože g_2 je dle definice vlastní, je vlastní i f_2. Aby mělo smysl řešit úlohu (1.19), předpokládejme pro zbytek této části, že $\operatorname{dom} f_2(\cdot,\xi)$ je neprázdná skoro jistě. Toto dále implikuje, že f_2 je polyedrická. Podobně předpokládejme, že $\operatorname{dom} f_1$ je neprázdná.

Podobně jako v lineárním případě vytvoříme pro druhý stupeň (1.19) duální problém. Lagrangeova funkce vypadá následovně

$$\begin{aligned} L(y,\pi;x,\xi) &= f_2(y,\xi) + \pi'(h(\xi) - T(\xi)x - W(\xi)y) \\ \inf_y L(y,\pi;x,\xi) &= \pi'(h(\xi) - T(\xi)x) - \sup_y\{\pi'W(\xi)y - f_2(y,\xi)\} \\ &= \pi'(h(\xi) - T(\xi)x) - f_2^*(\pi'W(\xi),\xi), \end{aligned}$$

kde f_2^* je sdružená funkce k funkci f_2. Duální problém k (1.19) je tedy tvaru

$$\max_\pi[\pi'(h(\xi) - T(\xi)x) - f_2^*(W(\xi)'\pi,\xi)]. \tag{1.20}$$

V odvození (1.20) jsme nikde nepoužili, že $f_2(\cdot,\xi)$ je polyedrická. Duální tvar (1.20) platí pro libovolnou $f_2(\cdot,\xi)$ konvexní. Ke konvexitě $f_2(\cdot,\xi)$ stačí, aby byla konvexní i funkce $g_2(\cdot,\xi)$.

Zjednodušme zápis $f_2^*(\cdot,\xi)$ na $f_2^*(\cdot)$ a znovu označme $\mathcal{D}(x,\xi)$ množinu optimálních řešení duálního problému (1.20). Formulujme nyní sadu vět, které jsou rozšířeními vět z předchozí části o lineárních problémech.

Věta 1.14. *Nechť $\xi \in \Xi$ je pevné a nechť $\mathcal{Q}(\bar{x},\xi)$ je konečné pro nějaké $\bar{x}$. Potom $\mathcal{Q}(\cdot,\xi)$ je polyedrická a subdiferencovatelná pro každé x, pro které je $\mathcal{Q}(x,\xi)$ konečná a platí*

$$\partial\mathcal{Q}(x,\xi) = -T(\xi)'\mathcal{D}(x,\xi).$$

Důkaz. Definujme funkci $\psi(\pi) = f_2^*(W'\pi)$. Pokud je $\mathcal{Q}(x,\xi)$ konečná, pak platí

$$\mathcal{Q}(x,\xi) = \max_\pi[\pi'(h - Tx) - f_2^*(W'\pi)] = \max_\pi[\pi'(h - Tx) - \psi(\pi)] = \psi^*(h - Tx).$$

Protože sdružená funkce k polyedrické funkci je znovu polyedrická, je polyedrická i funkce $\mathcal{Q}(\cdot,\xi)$.

Podle důsledku A.8 platí

$$\begin{aligned}\partial\psi^*(h - Tx) &= \arg\max_{\pi}[(h - Tx)'\pi - \psi^{**}(\pi)] \\ &= \arg\max_{\pi}[(h - Tx)'\pi - f_2^*(W'\pi)] = \mathcal{D}(x, \xi)\end{aligned}$$

a tvrzení dostaneme použitím řetízkového pravidla. □

Definice 1.15. Problém (1.14)–(1.15) má pevnou kompenzaci, pokud je W deterministická matice a množina $\mathcal{Y} = \operatorname{dom} f_2(\cdot, \xi)$ nezávisí na ξ. Dále definujme

$$W(\mathcal{Y}) = \{Wy; y \in \mathcal{Y}\}.$$

Druhá podmínka v definici pevné kompenzace je automaticky splněna v lineárním případě, neboť $\operatorname{dom} q(\xi)'y = \mathbb{R}^m$.

Lemma 1.16. *Nechť má problém pevnou kompenzaci, nechť existuje x takové, že skoro jistě $\mathcal{Q}(x, \xi) < \infty$ a nechť existuje ξ_0 takové, že $\mathcal{Q}(x, \xi_0)$ je konečné. Potom existuje $\kappa > 0$ takové, že pro skoro všechna ξ platí*

$$\begin{aligned}\mathcal{Q}(x, \xi) \leq \mathcal{Q}(x, \xi_0) &+ \sum_{i=1}^{I_2} |\gamma_i(\xi) - \gamma_i(\xi_0)| + \sum_{i=1}^{I_2} ||q_i(\xi) - q_i(\xi_0)||\ ||y_0|| \\ &+ \kappa \sum_{i=1}^{I_2} ||q_i(\xi)||(||h(\xi) - h(\xi_0)|| + ||x||\ ||T(\xi) - T(\xi_0)||)\end{aligned}$$

Důkaz. Definujme multifunkci $\mathcal{M}(\xi)$, kdy každému ξ přiřadíme množinu těch bodů $y \in \mathcal{Y}$, které jsou přípustnými řešeními (1.19). Protože $\mathcal{Q}(x, \xi_0)$ je konečné, existuje y_0, které minimalizuje (1.19) pro x a ξ_0. Z tohoto ihned plyne, že $\xi_0 \in \operatorname{dom} \mathcal{M}$.

Multifunkci $\mathcal{M}$ můžeme formálně zapsat jako

$$\mathcal{M}(\xi) = \{y; \begin{pmatrix} W \\ -W \\ d_j' \end{pmatrix} y \leq \begin{pmatrix} h(\xi) - T(\xi)x \\ -h(\xi) + T(\xi)x \\ r_j \end{pmatrix}\}. \tag{1.21}$$

Použijeme-li Hoffmanovu větu A.9, dostaneme, že existuje konstanta $\kappa > 0$ taková, že pro každé ξ, pro které platí $\mathcal{Q}(x, \xi) < \infty$ existuje $y(\xi)$ splňující $y(\xi) \in \mathcal{Y}$, $Wy(\xi) = h(\xi) - T(\xi)x$ a

$$\operatorname{dist}(y_0, y(\xi)) \leq \kappa \left\| \begin{pmatrix} Wy_0 - h(\xi) + T(\xi)x \\ -Wy_0 + h(\xi) - T(\xi) \\ d_1'y_0 - r_1 \\ \cdots \\ d_{I_2}'y_0 - r_{J_2} \end{pmatrix}_+ \right\| = \kappa \left\| \begin{pmatrix} Wy_0 - h(\xi) + T(\xi)x \\ 0 \\ 0 \\ \cdots \\ 0 \end{pmatrix} \right\|.$$

Protože $Wy_0 = h(\xi_0) - T(\xi_0)x$, dostaneme

$$||y_0 - y(\xi)|| \leq \kappa ||h(\xi_0) - T(\xi_0)x - h(\xi) + T(\xi)x||. \tag{1.22}$$

Zde vidíme oprávněnost definice pevné kompenzace. Abychom mohli použít větu A.9, potřebujeme, aby byla matice W a vektory d_j deterministické.

Díky vztahu (1.22) a díky optimalitě y_0 pro problém (1.15), ve kterém uvažujeme pro x a ξ_0, můžeme zapsat

$$\begin{aligned}\mathcal{Q}(x,\xi) &\le \max_{1,\dots,I_2}\{\gamma_i(\xi)+q_i(\xi)'y(\xi)\}\\ &= \max_{i=1,\dots,I_2}\{\gamma_i(\xi)-\gamma_i(\xi_0)+q_i(\xi)'(y(\xi)-y_0)\\ &\qquad +(q_i(\xi)-q_i(\xi_0))'y_0+\gamma_i(\xi_0)+q_i(\xi_0)'y_0\}\\ &\le \sum_{i=1}^{I_2}|\gamma_i(\xi)-\gamma_i(\xi_0)|+\sum_{i=1}^{I_2}||q_i(\xi)-q_i(\xi_0)||\;||y_0||+\mathcal{Q}(x,\xi_0)\\ &\qquad +\kappa\sum_{i=1}^{I_2}||q_i(\xi)||(||h(\xi)-h(\xi_0)||+||x||\;||T(\xi)-T(\xi_0)||)\end{aligned}$$

□

Věta 1.17. *Nechť má úloha* (1.14)–(1.15) *pevnou kompenzaci a nechť pro* $i=1,\dots,I_2$ *platí*

$$\mathrm{E}\,|\gamma_i|<+\infty,\ \mathrm{E}\,||q_i||<+\infty,\ \mathrm{E}[||q_i||||h||]<+\infty,\ \mathrm{E}[||q_i||||T||]<+\infty, \tag{1.23}$$

kde γ_i *a* q_i *jsou příslušné náhodné polyedrické funkci* $f_2(y,\xi)$ *z* (1.13).

Potom $\mathrm{E}[\mathcal{Q}(x,\xi)_+]$ *je konečná právě tehdy, pokud*

$$h(\xi)-T(\xi)x\in W(\mathcal{Y})\ \text{s. j.} \tag{1.24}$$

Důkaz. Nechť $\mathrm{E}[\mathcal{Q}(x,\xi)_+]<+\infty$ a pro spor předpokládejme existenci Ξ_1 takové, že $\mathrm{P}(\Xi_1)>0$ a pro všechna $\xi\in\Xi_1$ platí $h(\xi)-T(\xi)x\notin W(\mathcal{Y})$. To ale znamená $\mathcal{Q}(x,\xi)=+\infty$, a tedy $\mathrm{E}[\mathcal{Q}(x,\xi)_+]=+\infty$, čímž jsme došli ke sporu.

Dokažme nyní obrácenou implikaci. Podmínka (1.24) je ekvivalentní tomu, že skoro jistě platí $\mathcal{Q}(x,\xi)<\infty$. Pokud neexistuje ξ_0 takové, že $\mathcal{Q}(x,\xi_0)$ je konečné, pak skoro jistě platí $\mathcal{Q}(x,\xi)=-\infty$, a tedy $\mathrm{E}[\mathcal{Q}(x,\xi)_+]=0$. Předpokládejme tedy, že takové ξ_0 existuje. Tím jsou splněny předpoklady lemmatu 1.16 a zkombinováním výsledků tohoto lemmatu a předpokladů této věty dostaneme požadované tvrzení. □

Podobně jako v lineárním případě definujme

$$\Pi(\xi)=\{\pi;\, f_2^*(W'\pi,\xi)<\infty\}$$

Věta 1.18. *Nechť je kompenzace pevná,* $\Pi(\xi)$ *neprázdná skoro jistě a nechť platí* (1.23). *Potom* ϕ *je korektně definovaná, vlastní, konvexní, zdola polospojitá a Lipschitzovsky spojitá na* $\operatorname{dom}\phi$. *Doména* $\operatorname{dom}\phi$ *je konvexní uzavřená podmnožina* $\mathbb{R}^n$ *a platí*

$$\operatorname{dom}\phi=\{x\in\mathbb{R}^n;\, h-Tx\in W(\mathcal{Y})\ \text{s.j.}\}.$$

Navíc pro každé $x_0\in\operatorname{dom}\phi$ *platí*

$$\partial\phi(x_0)=-\mathrm{E}[T'\mathcal{D}(x_0,\xi)]+\mathcal{N}_{\operatorname{dom}\phi}(x_0).$$

Důkaz. Důkaz je podobný jako u vět 1.9 a 1.10. □

Předpokládejme opět, že ξ nabývá pouze konečně mnoha hodnot. Nechť pro $k = 1, \ldots, K$ nabývá s pravděpodobnostmi p_k hodnot ξ_k. Potom za úmluvy $+\infty + (-\infty) = +\infty$ platí

$$\begin{aligned} \mathrm{E}\, \mathcal{Q}(x, \xi) &= \sum_{k=1}^{K} p_k \mathcal{Q}(x, \xi_k) \\ &= \min_{y_1, \ldots, y_K} \sum_{k=1}^{K} p_k f_2(y_k, \xi_k) \\ &\quad s.t.\ T_k x + W_k y_k = h_k,\ k = 1, \ldots, K, \\ &\quad\quad y_k \geq 0,\ k = 1, \ldots, K, \end{aligned}$$

kde $(h_k, T_k, W_k) = (h(\xi_k), T(\xi_k), W(\xi_k))$.

Věta 1.19. *Nechť náhodný vektor ξ může nabývat pouze konečného množství realizací a nechť ϕ má konečnou hodnotu v alespoň jednom bodě x. Potom ϕ je polyedrická funkce a pro každé $x_0 \in \operatorname{dom} \phi$ platí*

$$\partial \phi(x_0) = \sum_{k=1}^{K} p_k \partial \mathcal{Q}(x_0, \xi_k) = -\sum_{k=1}^{K} p_k T_k' \arg\max_{\pi} [\pi'(h_k - T_k x_0) - f_2^*(W(\xi)'\pi)].$$

Důkaz. Protože $\phi(x) = \sum_{k=1}^{K} p_k \mathcal{Q}(x, \xi)$, je $\phi(x)$ polyedrická díky větě 1.14. Tvar subdiferenciálu pak plyne z vět A.5 a 1.14. □

Věta 1.20. *Předpokládejme, že ξ nabývá pouze konečně mnoho realizací. Potom $\bar{x}$ je optimální řešení (1.14) tehdy a jen tehdy, pokud existují $\pi_k \in \mathcal{D}(\bar{x}, \xi_k),\ k = 1, \ldots, K$ takové, že je splněno*

$$0 \in \partial f_1(\bar{x}) - \sum_{k=1}^{K} p_k T_k' \pi_k.$$

Důkaz. Protože podle věty 1.18 je ϕ konvexní funkce, je účelová funkce problému (1.18) konvexní. Protože se jedná o minimalizaci bez podmínek, $\bar{x}$ je optimální řešení právě tehdy, když $0 \in \partial(f_1(\bar{x}) + \phi(\bar{x}))$. K dokončení důkazu stačí vzhledem k polyedričnosti f_1 a ϕ použít Moreau-Rockafellarovu větu A.5 a větu 1.19. □

Kapitola 2

Metody pro řešení

V této kapitole uveďme algoritmy, pomocí kterých je možné řešit úlohy dvojstupňového programování, pokud náhodný vektor ξ nabývá pouze konečného množství možných bodů. Zachovejme značení z předcházející kapitoly, jednotlivé body označme $\xi_1, \ldots, \xi_K$ a tyto body se budou nabývat postupně s pravděpodobnostmi $p_1, \ldots, p_K$. Zároveň připomeňme definice $K_1 = \{x \geq 0; Ax = b\}$ a $K_2 = \{x; \forall k \exists y_k \geq 0 : Wy_k = h_k - Tx_k\}$. Pokud nosič není konečný, je možné provést diskretizaci náhodného vektoru, čímž úlohu převedeme na předchozí případ.

Pro diskrétní případ existují dvě možnosti přístupu. Prvním z nich je vyřešení deterministického ekvivalentu (1.8), popřípadě jeho maticového zapsání (1.10) místo původního problému (1.1)–(1.2). Bohužel tento problém obsahuje nemalé množství neznámých a provedení výpočtu by mohlo být časově náročné. Druhou možností je využít speciální struktury dvojstupňové úlohy. V této kapitole se budeme věnovat výhradně druhému způsobu.

Většina kapitoly se bude zabývat algoritmem L–shaped a algoritmy, které jsou od něj odvozené. Nejdříve uvedeme základní variantu, posléze některé jeho rozšíření a nakonec budeme místo lineární uvažovat polyedrickou, kvadratickou a konečně konvexní účelovou funkci. Bohužel L–shaped algoritmus je možné použít pouze, pokud je množina přípustných řešení polyedrická. Pokud tento předpoklad není splněn, je nutné použít jiný algoritmus, jako příklad uvádíme v poslední části této kapitoly progressive hedging algoritmus. Pěkný přehled algoritmů je možné najít například v [8].

L–shaped algoritmus dostal svůj název podle speciální struktury, kterou využívá. Pokud se podíváme na maticový zápis deterministického ekvivalentu (1.10) a uvažujeme případ $K = 1$, v matici je jeden nulový blok a zbylé tři nenulové bloky tvoří písmeno L. Kromě názvu L–shaped se často používá i označení duální dekompozice. Tento název vychází z (1.11), kdy první řádek odpovídá prvnímu stupni (1.1) a každý další řádek odpovídá duální úloze (1.3) pro jednotlivé realizace ξ.

Před samotnými algoritmy zmiňme jednu poznámku. Zatímco v předchozí kapitole byla v některých větách nutným předpokladem pevná kompenzace, tedy matice W nebyla náhodná, v dále zmíněných algoritmech tento předpoklad není nutný a algoritmy je možné aplikovat i pro W stochastickou.

2.1 L-shaped algoritmus

Zatímco v deterministickém ekvivalentu řešíme jeden velký problém, v L-shaped algoritmu řešíme $K+1$ menších problémů. Jeden z problému je hlavní problém, který, zjednodušeně řečeno, odpovídá prvnímu stupni dvojstupňovému problému. Zbylých K problémů jsou pak subproblémy a odpovídají jednotlivým realizacím náhodného vektoru ξ.

V prvním stupni (1.1) řešíme problém $\min_{x\in X} c'x + \phi(x)$. Tento problém ale nahradíme problémem, který má stejné optimální řešení $\bar{x}$ a stejnou optimální hodnotu

$$\begin{aligned} &\min_{x\in X,\theta} \; c'x + \theta \\ &\quad s.t.\; \theta \geq \phi(x). \end{aligned} \tag{2.1}$$

Pro funkci ϕ neznáme explicitní zápis, využijeme ale věty 1.11, která říká, že ϕ je polyedrická funkce. To znamená, že ji na její doméně můžeme zapsat jako maximum konečného množství lineárních funkcí. Místo podmínky $\theta \geq \phi(x)$ přidejme do problému (2.1) podmínky $\theta \geq e_l - E_l x$, kde jednotlivé $e_l - E_l x$ jsou opěrné funkce pro funkci $\phi(x)$. Tyto podmínky budeme nazývat řezy optimality.

Protože pomocí řezů optimality můžeme funkci ϕ aproximovat pouze na její doméně, musíme dále aproximovat ještě její doménu. Zřejmě platí $\operatorname{dom}\phi = K_2$ a o množině K_2 víme, že je polyedrická, což znamená, že ji můžeme zapsat pomocí systému nerovností $D_l x \geq d_l$. Podobně jako pro řezy optimality, budeme tyto podmínky postupně přidávat do hlavního problému. Podmínky tohoto typu nazveme řezy přípustnosti.

Hlavní problém pak bude odpovídat problému (2.1), ve kterém místo podmínky $\theta \geq \phi(x)$ budeme uvažovat už přidané řezy přípustnosti a optimality. Optimální řešení hlavního problému označme (x^ν, θ^ν), kde ν označuje pořadí iterace.

V jednotlivých subproblémech nejprve zkoumáme, jestli je splněno $x^\nu \in K_2$. Pokud toto není splněno, přidáme řez přípustnosti. Pokud naopak platí $x^\nu \in K_2$, automaticky platí $x^\nu \in X$. Vyřešíme jednotlivé subproblémy, jejichž řešení označme y_k^ν a zkoumáme, jestli je $(x^\nu, y_1^\nu, \ldots, y_K^\nu)$ optimálním řešením (1.8). Pokud tomu tak je, algoritmus ukončíme, v opačném případě přidáme řez optimality.

Pro úvod předpokládejme, že množina K_1 je omezená. Pokud tento předpoklad není splněn, algoritmus se značně zesložití, protože se bude muset uvažovat případ, kdy hlavní problém není zdola omezený a do dalších kroků se místo s bodem bude muset pokračovat se směrem, ve kterém úloha neomezeně klesá. Tomuto případu se budeme věnovat později. Nyní uveďme jednotlivé kroky základního algoritmu.

Krok 0: V tomto kroku inicializujeme proměnné $r = s = \nu = 0$. Tyto proměnné postupně uvádějí počet řezů přípustnosti, počet řezů optimality a pořadí iterace, ve které se algoritmus nachází. Do tohoto kroku se později už nevracíme.

Krok 1: Nejdříve zvětšíme ν o jedna a řešíme optimalizační úlohu

$$\begin{aligned} &\min_{x,\theta} \; c'x + \theta \\ &s.t.\; Ax = b, \\ &\quad D_l x \geq d_l,\; l = 1, \ldots, r, \\ &\quad E_l x + \theta \geq e_l,\; l = 1, \ldots, s, \\ &\quad x \geq 0. \end{aligned} \tag{2.2}$$

Pokud je $r = 0$ nebo $s = 0$, příslušné nerovnosti v problému neuvažujeme. Dále pokud je $s = 0$, do účelové funkce zařadíme pouze člen $c'x$, problém vyřešíme, jeho optimální řešení označíme x^ν a položíme $\theta^\nu = -\infty$. Pokud neexistuje přípustné řešení hlavního problému (2.2), nemá přípustné řešení ani problém (1.8). Pokud naopak pro (2.2) přípustné řešení existuje, existuje díky předpokladu na omezenost K_1 i optimální řešení, které označme (x^ν, θ^ν).

Krok 2: Víme, že $x^\nu \in K_1$ a zkoumáme, jestli platí $x^\nu \in K_2$. Pro každé $k = 1, \ldots, K$ vyřešíme úlohu

$$\begin{aligned} \min_{y, v^+, v^-} \; & e'v^+ + e'v^- \\ s.t. \; & W_k y - v^+ + v^- = h_k - T_k x^\nu, \\ & y \geq 0, \; v^+ \geq 0, \; v^- \geq 0, \end{aligned} \tag{2.3}$$

kde $e = (1, \ldots, 1)'$. Optimální hodnotu (2.3) označíme ω_k a optimální řešení duální úlohy příslušné k (2.3) označíme σ_k^ν. Pokud pro nějaké k platí $\omega_k > 0$, položíme

$$\begin{aligned} D_{r+1} &= (\sigma_k^\nu)' T_k, \\ d_{r+1} &= (\sigma_k^\nu)' h_k, \end{aligned} \tag{2.4}$$

zvětšíme r o jedničku, čímž do hlavního problému přidáme řez přípustnosti a vrátíme se zpět do prvního kroku. Pokud pro všechna k je $\omega_k = 0$, posuneme se do třetího kroku.

Krok 3: Víme, že x^ν je přípustným řešením pro první i druhý stupeň, tedy $x \in X$, a chceme zjistit, zda se jedná o optimální řešení. Pro $k = 1, \ldots, K$ uvažujme subproblém

$$\begin{aligned} \min_{y} \; & q_k' y \\ s.t. \; & W_k y = h_k - T_k x^\nu, \\ & y \geq 0. \end{aligned} \tag{2.5}$$

Nalezněme optimální řešení duálního problému příslušejícího (2.5) a toto řešení označme π_k^ν. Dále položme

$$\begin{aligned} E_{s+1} &= \sum_{k=1}^{K} p_k (\pi_k^\nu)' T_k \\ e_{s+1} &= \sum_{k=1}^{K} p_k (\pi_k^\nu)' h_k \\ \tau^\nu &= e_{s+1} - E_{s+1} x^\nu. \end{aligned} \tag{2.6}$$

Pokud platí $\theta^\nu \geq \tau^\nu$, algoritmus končí, x^ν spolu s optimálními řešeními (2.5) je optimální řešení (1.8). V opačném případě zvětšíme s o jedničku, přidáme do prvního kroku řez optimality $\theta \geq e_l - E_l x$ a vrátíme se do prvního kroku.

Vysvětleme si nyní jednotlivé kroky algoritmu podrobněji. Věnujme se postupně oprávněnosti generování řezů přípustnosti a optimality, ukončovacího kritéria a nakonec konvergenci algoritmu.

Jak už bylo řečeno, v druhém kroku zkoumáme, zda platí $x^\nu \in K_2$. Pokud pro všechna k platí $\omega_k = 0$, znamená to, že existují $y_k \geq 0$ taková, že platí $W_k y_k = h_k - T_k x^\nu$, z čehož ihned plyne $\mathcal{Q}(x^\nu, \xi_k) < \infty$ a tedy $x^\nu \in K_2$. Pokud

naopak existuje k splňující $\omega_k > 0$, pak $h_k - T_k x^\nu \notin \operatorname{pos} W_k$. Protože $\operatorname{pos} W_k$ je konvexní uzavřená množina, můžeme od ní bod $h_k - T_k x^\nu$ ostře oddělit.

Platí $(\sigma_k^\nu)'(h_k - T_k x^\nu) = \omega_k > 0$. Protože σ_k^ν jsou optimální řešení duálního problému k problému (2.3), platí $W_k' \sigma_k^\nu \leq 0$. Pro libovolné $\chi \in \operatorname{pos} W_k$ existuje $y \geq 0$ takové, že $\chi = W_k y$. Potom ale platí $(\sigma_k^\nu)'\chi = (\sigma_k^\nu) W_k y = (W_k' \sigma_k^\nu)' y \leq 0$. Z tohoto ihned plyne, že oddělovací nadrovina je určena pomocí vektoru σ_k^ν a pro libovolné $x \in K_2$ musí platit $(\sigma_k^\nu)'(h_k - T_k x) \leq 0$, což je ale porušeno pro x^ν.

Věnujme se nyní třetímu kroku. Z duality lineárního programování víme, že platí

$$\begin{aligned} \mathcal{Q}(x^\nu, \xi_k) &= (\pi_k^\nu)'(h_k - T_k x^\nu) \\ \tau^\nu = \phi(x^\nu) &= \sum_{k=1}^{K} p_k (\pi_k^\nu)'(h_k - T_k x^\nu) \\ \phi(x) &= \sum_{k=1}^{K} p_k \sup_{\pi_k} \{\pi_k'(h_k - T_k x);\ W_k' \pi_k \leq q_k\} \geq \sum_{k=1}^{K} p_k (\pi_k^\nu)'(h_k - T_k x). \end{aligned} \tag{2.7}$$

Z tohoto zápisu plyne, že $\sum_{k=1}^{K} p_k (\pi_k^\nu)'(h_k - T_k x)$ je dolním odhadem $\phi(x)$. Tyto dvě funkce mají navíc společnou funkční hodnotu v bodě x^ν, jedná se tedy o opěrnou funkci.

Pokud platí $\tau^\nu \leq \theta^\nu$, je dvojice (x^ν, θ^ν) přípustným řešením problému (2.1). Protože problémy (2.2) a (2.1) mají stejnou účelovou funkci a množina přípustných řešení (2.2) je nadmnožinou množiny přípustných řešení (2.1), plyne z optimality (x^ν, θ^ν) pro (2.2) i optimalita (x^ν, θ^ν) pro (2.1).

Pokud naopak platí $\tau^\nu > \theta^\nu$, je třeba přidat řez optimality. Jeho zápis dostaneme z poslední podmínky v (2.7). Podmínka problému (2.1) $\theta \geq \phi(x)$ implikuje nutnost splnění $\theta \geq \sum_{k=1}^{K} p_k (\pi_k^\nu)'(h_k - T_k x)$. Tato podmínka je porušena dvojicí (x^ν, θ^ν). Jako ukončovací kritérium stačí požadovat dokonce pouze $\tau^\nu = \theta^\nu$, ale ve všech knihách je uveden tvar s nerovností. Pravděpodobně je tomu kvůli tomu, že počítače jsou schopny pracovat pouze s omezenou přesností a rovnost by nemusela nikdy nastat.

Tímto jsme dokázali oprávněnost jednotlivých řezů a ukončovacího kritéria. Věnujme se nyní konvergenci algoritmu, ke které stačí dokázat, že algoritmus může vygenerovat pouze omezený počet řezů. Druhý krok vygeneruje pouze konečné množství řezů, protože kužel $\{y;\ W_k' y \leq 0\}$ je polyedrický, třetí krok vygeneruje konečné množství řezů, protože existuje pouze konečný počet duálně přípustných bázických řešení problému (2.5).

Spolu s omezeností K_1 dostáváme výsledek, že algoritmus L-shaped najde optimální řešení problému (1.8) v konečném počtu kroků, popřípadě zjistí, že neexistuje přípustné řešení. Tento případ je podrobněji rozebírán například v [8], kde se navíc do prvního kroku zařadila podmínka $\theta \geq \theta_0$, kde θ_0 je jakýkoli dolní odhad pro $\phi(x)$.

Odvoďme nyní ještě jednu možnost, jak spočítat řezy optimality. Protože ϕ

je konvexní funkce, z vlastností subdiferenciálu, vět (1.11) a (1.3) plyne

$$\begin{aligned}\phi(x) &\geq \phi(x^\nu) + [\partial\phi(x^\nu)]'(x - x^\nu) \\ &= \sum_{k=1}^{K} p_k(\pi_k^\nu)'(h_k - T_k x^\nu) + [\sum_{k=1}^{K} p_k(-T_k)'\pi_k^\nu]'(x - x^\nu) \\ &= \sum_{k=1}^{K} p_k(\pi_k^\nu)'h_k - \sum_{k=1}^{K} p_k(\pi_k^\nu)'T_k x \\ &= \sum_{k=1}^{K} p_k(\pi_k^\nu)'(h_k - T_k x).\end{aligned}$$

Odvození tímto způsobem využijeme i v dalších částech.

Věnujme se nyní úpravě algoritmu pro polyedrickou množinu K_1, která ale nemusí být omezená. Tento algoritmus byl poprvé navržen v [23] a jeho implementaci můžeme nalézt například v OSLSE, což byla knihovna metod pro řešení stochastické optimalizace, jejíž vývoj byl ale bohužel ukončen roku 2002. Podrobnější informace můžeme nalézt například v [11]. Uveďme nyní jednotlivé kroky tohoto algoritmu.

Krok 0: Identický jako v základní verzi.

Krok 1: Nejdříve zvětšíme ν o jedna a řešíme optimalizační úlohu (2.2). Pokud je $s = 0$, položíme $\theta^\nu = -\infty$ a v účelové funkci uvažujeme pouze člen $c'x$. Pokud neexistuje přípustné řešení, ani problém (1.8) nemá přípustné řešení. Pokud existuje optimální řešení, označme ho (x^ν, θ^ν). Pokud není problém zdola omezen, pokračujme do druhého kroku s polopřímkou $x^\nu + tu^\nu,\ t \geq 0$, na které účelová funkce neomezeně klesá.

Krok 2: Identický jako v základní verzi.

Krok 3: Pro $k = 1, \ldots, K$ vyřešíme (2.5). Pokud tento problém není alespoň pro jedno k zdola omezený, není zdola omezený ani původní problém (1.8). Pokud pro všechna k existuje optimální řešení a v prvním kroku vyšlo optimální řešení, provedeme stejně jako v předchozím algoritmu test optimality a algoritmus buď zastavíme nebo přidáme řez optimality. Pokud pro všechna k existuje optimální řešení, ale problém v prvním kroku nebyl omezený, přesuneme se do čtvrtého kroku. Připomeňme, že problém řešený v tomto kroku vždy musí mít pro všechna k alespoň jedno přípustné řešení.

Krok 4: Pro $k = 1, \ldots, K$ vyřešme postupně problémy

$$\begin{aligned}\min_{y,v^+,v^-}\ & e'v^+ + e'v^- \\ s.t.\ & W_k y - v^+ + v^- = -T_k u^\nu, \\ & y \geq 0,\ v^+ \geq 0,\ v^- \geq 0.\end{aligned} \tag{2.8}$$

Optimální hodnoty (2.8) označme q_k^ν a optimální řešení duálního problému příslušného k problému (2.8) označme μ_k^ν. Pokud pro nějaké k je $q_k^\nu > 0$ znamená to, že směr u^ν není přípustným směrem množiny X. V tomto případě vygenerujeme podobně jako v (2.4) řez přípustnosti, pouze místo σ_k^ν uvažujeme μ_k^ν. V případě vygenerování řezu se vrátíme do prvního kroku, v opačném případě postoupíme do pátého kroku.

Krok 5: Pro $k = 1, \ldots, K$ vyřešme postupně problémy

$$\begin{aligned} \min_y \; & q_k' y \\ s.t. \; & W_k y = -T_k u^\nu, \\ & y \geq 0, \end{aligned} \tag{2.9}$$

jejichž optimální hodnoty označme v_k^ν a optimální hodnoty duální úlohy označme λ_k^ν. Pokud bude platit $c'u^\nu + \sum_{k=1}^K p_k q_k' v_k^\nu \geq 0$, vygenerujeme podobně jako v (2.6) řez optimality, pouze místo π_k^ν uvažujeme λ_k^ν. V tomto případě se vrátíme do prvního kroku, v opačném případě není úloha (1.8) zdola omezená.

Oprávněnost jednotlivých řezů nebudeme dokazovat, důkaz je podobný jako v základním algoritmu. Ukážeme pouze jak vznikl problém (2.9) a oprávněnost podmínky v pátém kroku, na jejíž základě přidáme další řez nebo algoritmus ukončíme.

Nejdříve si ukažme, za jakých podmínek nebude problém (1.8) zdola omezený. Předpokládejme, že účelová funkce (1.8) neomezeně klesá na polopřímce $(\bar{x} + t\tilde{x}, \bar{y}_1 + t\tilde{y}_1, \ldots, \bar{y}_K + t\tilde{y}_K)$, kde $t \geq 0$. Potom musí být počátek polopřímky přípustné řešení (1.8), neboli $A\bar{x} = b$, $T\bar{x} + W\bar{y}_k = h_k$ a $\bar{x}$ i $\bar{y}_k$ jsou nezáporné. Dále směr $(\tilde{x}, \tilde{y}_1, \ldots, \tilde{y}_K)$ musí být přípustným směrem množiny přípustných řešení, neboli $A\tilde{x} = 0$ a $T_k\tilde{x} + W_k\tilde{y}_k = 0$ a $\tilde{x}$ a $\tilde{y}_k$ jsou nezáporné. Nakonec musí účelová funkce v tomto směru klesat, neboli

$$c'\tilde{x} + \sum_{k=1}^K p_k q_k' \tilde{y}_k < 0.$$

Nyní předchozí podmínky aplikujme pro $\bar{x} = x^\nu$ a $\tilde{x} = u^\nu$. Splnění podmínek $Ax^\nu = b$, $Au^\nu = 0$, $x \geq 0$ a $u^\nu \geq 0$ máme zaručeno díky neomezenosti tohoto směru v úloze, kterou jsme řešili v prvním kroku. Existence $\bar{y}_k \geq 0$ splňující $T_k x^\nu + W_k \bar{y}_k = h_k$ plyne přímo z třetího kroku. Z tohoto ihned plyne, že pokud jsme v pátém kroku, pak problém (1.8) neomezeně klesá ve směru $(x^\nu + tu^\nu, \bar{y}_1 + t\tilde{y}_1, \ldots, \bar{y}_K + t\tilde{y}_K)$ právě tehdy, pokud jsou splněny následující podmínky

$$\begin{aligned} c'u^\nu + \sum_{k=1}^K p_k q_k' \tilde{y}_k &< 0 \\ T_k\tilde{x} + W_k\tilde{y}_k &= 0 \\ \tilde{y}_k &\geq 0. \end{aligned}$$

Z tohoto zápisu je ihned patrný tvar (2.9) a následné kritérium.

2.2 Varianty L-shaped metody

Ačkoli L-shaped metoda je značným zlepšením oproti řešení deterministického ekvivalentu, pořád tento algoritmus trpí několika nedostatky. Jedním z nich je vlastnost, že první iterace mohou být od sebe daleko vzdáleny a algoritmu může trvat dlouho, než se dostane do blízkosti optimálního řešení. Z tohoto důvodu existuje multicut verze L-shaped metody, popřípadě metody penalizující vzdálenosti mezi jednotlivými iteracemi.

V této části se věnujme multicut verzi, při které v jedné iteraci přidáváme více řezů. Zatímco v původním algoritmu jsme zdola odhadovali $\sum_{k=1}^{K} p_k \mathcal{Q}(x, \xi_k)$, v multicut verzi odhadujeme v jedné iteraci každou z funkcí $p_k \mathcal{Q}(x, \xi_k)$. Místo jednoho parametru θ zavedeme K parametrů $\theta_1, \ldots, \theta_K$. Odhad množiny K_2 zůstává stejný jako v předchozí verzi. Pro jednoduchost předpokládejme, že množina K_1 je omezená. Pokud by nebyla, rozšíření se provede stejně jako v části 2.1.

V prvním kroku pak řešíme problém

$$\begin{aligned} \min_{x,\theta_k} \; & c'x + \sum_{k=1}^{K} \theta_k \\ s.t. \; & Ax = b, \\ & D_l x \geq d_l, \; l = 1, \ldots, r, \\ & E_{l(k)} x + \theta_k \geq e_{l(k)}, \; l(k) = 1, \ldots, s(k), \\ & x \geq 0, \end{aligned} \tag{2.10}$$

jehož optimální řešení označme $(x^\nu, \theta_1^\nu, \ldots, \theta_K^\nu)$. Podobně jako u základní verze, pokud je některé $s(k) = 0$, v účelové funkci neuvažujeme θ_k a položíme $\theta_k^\nu = \infty$.

Druhý krok je identický se základní verzí, v třetím se liší pouze vyhodnocení a případný řez. Pokud je pro všechna k splněno

$$\theta_k^\nu \geq p_k (\pi_k^\nu)'(h_k - T_k x^\nu), \tag{2.11}$$

algoritmus končí, optimální řešení bylo nalezeno. Pro všechna k, pro které předchozí nerovnost neplatí, položíme

$$\begin{aligned} E_{s(k)+1} &= p_k (\pi_k^\nu)' T_k \\ e_{s(k)+1} &= p_k (\pi_k^\nu)' h_k, \end{aligned} \tag{2.12}$$

zvětšíme $s(k)$ o jedna a vrátíme se do prvního kroku.

Existují i jiné cesty, jak zlepšit efektivitu algoritmu, ať už se jedná o základní nebo multicut verzi. Pokud pracujeme s problémem, který má úplnou nebo relativně úplnou kompenzaci, můžeme vynechat druhý krok, s ním spojené řezy a pracovat pouze s prvním a třetím krokem. Pokud jsou známé některé podmínky pro proměnné v modelu, může být výhodné těchto podmínek využít a explicitně je do modelu doplnit. Tento proces ale vyžaduje dobrou znalost modelu.

Existují i složitější variace L-shaped metody. Místo na primární problém ji například můžeme použít na duální problém. Existují i metody shrnuté v [5], které se ve třetím kroku zabývají použitím výsledků z předchozí iterace. Toto je založeno na tom, že problémy řešené v tomto kroku jsou si skrze jednotlivé iterace podobné, což by mohlo znamenat, že optimální báze si budou podobné.

Další možností zrychlení konvergence je pracování se vzdálenostmi jednotlivých iterací x^ν. V části 2.3 přidáme podmínky na maximální vzdálenost dvou za sebou následujících iterací. V části 2.4 pak tuto vzdálenost budeme v účelové funkci penalizovat kvadratickým členem.

2.3 L–shaped algoritmus s lokálně omezeným krokem

Tento algoritmus byl poprvé zpracován v [14]. Vychází z L-shaped algoritmu, hlavní změnou je přidání podmínky na maximální vzdálenost dvou sousedních

iterací, kdy musí platit $||x^{\nu+1} - x^{\nu}||_\infty \leq \Delta^{\nu\mu}$. Protože použitá norma je supremová, po menší úpravě přidané podmínky neztrácíme, na rozdíl od regularizované dekompozice, linearitu.

Dalším rozdílem oproti L-shaped metodě je rozdělení iterací na hlavní a vedlejší. Hlavní iterace budeme značit pomocí jednoho indexu x^ν, vedlejší pak pomocí dvou indexů $x^{\nu\mu}$. Jako další možnost byla implementována možnost vyškrtávat některé řezy, čímž se snižují požadavky na používanou paměť. Na druhou stranu při špatném vyškrtávání hrozí, že některé řezy se budou muset generovat několikrát.

Pro jednoduchost předpokládejme, že úloha má úplnou nebo relativně úplnou kompenzaci, nemusíme tedy uvažovat druhý krok algoritmu L-shaped. Na druhou stranu není nutné předpokládat omezenost K_1, protože problém (2.13), který řešíme v prvním kroku, má omezenou množinu přípustných řešení. Popišme nyní jednotlivé kroky algoritmu.

Krok 0: V tomto kroku inicializujme jednotlivé parametry. Konkrétně se jedná o volbu $\zeta \in (0, \frac{1}{2})$, $\nu = 1$, $\mu = 0$, $\Delta^{10} > 0$ a počáteční iteraci $x^0 \in K_1$.

Krok 1: Nejprve vyřešíme optimalizační úlohu

$$\begin{aligned} \min_{x,\theta_k} \; & c'x + \sum_{k=1}^{K} \theta_k \\ s.t. \; & Ax = b, \\ & E_{l(k)}x + \theta_k \geq e_{l(k)}, \; l(k) = 1, \ldots, s(k), \\ & ||x - x^\nu||_\infty \leq \Delta^{\nu\mu}, \\ & x \geq 0. \end{aligned} \tag{2.13}$$

Tento problém není díky normě lineární. Pokud označíme jednotlivé složky vektoru $x = (x_1, \ldots, x_n)$, měli bychom podmínku s normou rozepsat pomocí následujících nerovností

$$\begin{aligned} x_i - x_i^\nu &\leq \Delta_i^{\nu\mu}, \; i = 1, \ldots, n \\ x_i - x_i^\nu &\geq -\Delta_i^{\nu\mu}, \; i = 1, \ldots, n. \end{aligned} \tag{2.14}$$

Díky jednoduchosti ale ponecháváme původní zápis a předpokládáme, že při implementaci bude norma rozepsána pomocí zápisu (2.14).

Pokud neexistuje přípustné řešení (2.13), algoritmus ukončíme, ani původní problém (1.8) nemá přípustné řešení. Pokud existuje přípustné řešení, musí vzhledem k omezenosti množiny přípustných řešení existovat i optimální řešení, které označíme $(x^{\nu\mu}, \theta_1^{\nu\mu}, \ldots, \theta_K^{\nu\mu})$.

Označme $\phi^*(x) = c'x + \phi(x)$ účelovou funkci (1.8) a její odhad v dané iteraci $m_{\nu\mu}(x) = c'x + \sum_{k=1}^{K} \inf\{\theta_k; \; E_{l(k)}x + \theta_k \geq e_{l(k)}\}$. Pokud je splněno ukončovací kritérium

$$\phi^*(x^\nu) - m_{\nu\mu}(x^{\nu\mu}) \leq \varepsilon_{tol}(1 + |\phi^*(x^\nu)|),$$

zastavíme algoritmus, dospěli jsme k optimálnímu řešení za dané chyby. Pokud zvolíme $\varepsilon_{tol} = 0$, řešení je optimální. V opačném případě postoupíme do druhého kroku.

Krok 2: Podobně jako v L-shaped algoritmu zjistíme subgradienty $\mathcal{Q}(\cdot, \xi_k)$ v bodě $x^{\nu\mu}$ a na základě tohoto subgradientu vytvoříme nové řezy, které přidáme do modelu. Pokud je splněna podmínka

$$\phi^*(x^{\nu\mu}) \leq \phi^*(x^\nu) - \zeta[\phi^*(x^\nu) - m_{\nu\mu}(x^{\nu\mu})],$$

přesuneme se do třetího kroku, v opačném případě do kroku čtvrtého.

Krok 3: Splnění podmínky v druhém kroku znamená přechod k další hlavní iteraci. Položíme tedy $x^{\nu+1} = x^{\nu\mu}$. Dále můžeme škrtnou některé řezy, pokud se zachovají dvě podmínky. První z nich je $m_{\nu+1,0}(x^{\nu+1}) = \phi^*(x^{\nu+1})$ a druhá klade podmínku na funkci $m_{\nu+1,0}$, která musí být konvexní, po částech lineární dolní odhad ϕ^*. Pro detaily odkazuji znovu na článek [14], zjednodušeně lze říci, že řez nesmí být několik posledních iterací aktivní. Dále existuje možnost zvětšit parametr $\Delta^{\nu+1}$. Nakonec zvětšíme ν o jedna, položíme $\mu = 0$ a přesuneme se do prvního kroku.

Krok 4: Pokud podmínka v druhém kroku není splněna, nepřistupujeme k další hlavní nýbrž vedlejší iteraci. I zde můžeme škrtnout některé řezy, ale $\Delta^{\nu,\mu+1}$ můžeme pouze snížit. Zvětšíme nakonec μ o jedna, ν ponecháme stejné a přesuneme se do prvního kroku.

Tento algoritmus byl použit v asynchronní verzi v programu Condor, který umožňuje provádět výpočty na propojených počítačích. Více o tomto algoritmu pojednáme v kapitole o software v části 3.4. V už zmíněném článku jsou dokázány věty o konvergenci hlavních iterací k optimálnímu řešení.

2.4 Regularizovaná dekompozice

Ačkoli byl tento algoritmus navržen původně v [20] pro úlohu minimalizující součet polyedrických funkcí, ukážeme si zjednodušenou variantu pro lineární dvojstupňové programování. V další části poté pro všechny metody založené na L-shaped ukážeme, jak se dají rozšířit z lineárních do polyedrických účelových funkcí.

Regularizovaná dekompozice vychází z multicut L-shaped metody, stejně jako v ní odhadujeme $\mathcal{Q}(\cdot, \xi_k)$ zdola pomocí lineárních funkcí. Kromě hlavních iterací x^ν používáme i vedlejší iterace a^ν. Do účelové funkce se přidá kvadratický člen $\frac{1}{2}||x-a^\nu||^2$, díky němuž ztrácíme linearitu, ale podobně, jako v algoritmu s lokálně omezeným krokem, se algoritmus zjednoduší. Pokud bude mít problém řešený v prvním kroku přípustné řešení, bude mít i optimální řešení, nemůže se stát, že by tento problém nebyl zdola omezený. To znamená, že není nutné rozlišovat, jestli je K_1 omezená množina nebo ne.

Další rozdíl je ten, že si v paměti budeme uchovávat maximální počet řezů a neaktivní řezy se budou mazat. Algoritmus můžeme zapsat do následujících kroků.

Krok 0: Položíme $r = \nu = 0$, $s(k) = 0$ pro $k = 1, \ldots, K$. Najdeme libovolné $a_1 \in K_1$ a posuneme se do prvního kroku.

Krok 1: Zvětšíme ν o jedna a vyřešíme problém

$$\begin{aligned} \min_{x,\theta_k} \; & c'x + \sum_{k=1}^{K} \theta_k + \frac{1}{2}||x - a^\nu||^2 \\ s.t. \; & Ax = b, \\ & D_l(x) \geq d_l, \; l = 1, \ldots, r, \\ & E_{l(k)}x + \theta_k \geq e_{l(k)}, \; l(k) = 1, \ldots, s(k), \\ & x \geq 0, \end{aligned} \tag{2.15}$$

jehož optimální řešení označíme $(x^\nu, \theta_1^\nu, \ldots, \theta_K^\nu)$. Pokud pro nějaké k platí $s(k) = 0$, neuvažujeme v účelové funkci θ_k a položíme $\theta_k^\nu = \infty$. Pokud problém (2.15) nemá přípustné řešení, nemá přípustné řešení ani (1.8). Pokud platí $c'x^\nu + \sum \theta_k^\nu = c'a^\nu + \phi(a^\nu)$, algoritmus zastavíme, a^ν je optimální řešení. V opačném případě se přesuneme do druhého kroku.

Krok 2: Vyřešíme (2.3) a stejně jako v originálním L-shaped vygenerujeme řez přípustnosti nebo se přesuneme do třetího kroku. Při přidání řezu přípustnosti položíme navíc $a^{\nu+1} = a^\nu$.

Krok 3: Pro $k = 1, \ldots, K$ vyřešíme problém (2.5) a spočteme $\mathcal{Q}(x^\nu, \xi_k)$. Pokud není splněno (2.11), přidáme řezy optimality (2.12), zvětšíme $s(k)$ o jedna a posuneme se do čtvrtého kroku. V opačném případě položíme $a^{\nu+1} = x^\nu$ a vrátíme se do prvního kroku.

Krok 4: Pokud platí $c'x + \phi(x^\nu) \leq c'a^\nu + \phi(a^\nu)$, položíme $a^{\nu+1} = x^\nu$, v opačném případě položíme $a^{\nu+1} = a^\nu$. V každém případě se vrátíme do prvního kroku.

V původním článku [20] byla za předpokladu existence optimálního řešení dokázána konvergence a^ν k tomuto optimálnímu řešení po konečném počtu kroků. Pokud problém není zdola omezen, algoritmus vygeneruje posloupnost přípustných řešení a^ν takových, že $c'a^\nu + \phi(a^\nu) \to -\infty$. Ve stejném článku byl implementován i speciální postup, pomocí kterého stačilo uchovávat v paměti pouze $n + 2K$ řezů a nepotřebné řezy mazat.

Existuje možnost uvažovat $\frac{\alpha}{2}||x - a^\nu||^2$ místo $\frac{1}{2}||x - a^\nu||^2$, kde α je libovolný kladný parametr, čímž nastavíme velikost penalizace pro vzdálenost jednotlivých iterací. Dále je možné místo pevného α volit α^ν a podobně jako v algoritmu s lokálně omezeným krokem tento parametr v závislosti na postupu algoritmu měnit.

2.5 Polyedrický případ

Všechny doposud popsané algoritmy v této kapitole se týkaly lineárních problémů. V následujících částech budeme postupně uvažovat polyedrickou, kvadratickou a konečně konvexní účelovou funkci.

Ukažme si nyní rozšíření L–shaped algoritmu do polyedrického případu, uvažujme tedy problém (1.14)–(1.15). Rozšíření je triviální, neboť postup L–shaped algoritmu je identický jak v případě f_1 a f_2 lineárních, tak i polyedrických funkcí. Podmínky ale musí být tvořeny lineárními funkcemi, požadujeme tedy, aby podmínky v tomto případě byly stejné jako v lineárním případě.

Věnujme se nyní komentářům k jednotlivým krokům algoritmu. Podle věty 1.19 je ϕ polyedrická, z čehož ihned plyne, že ji můžeme napsat jako maximum konečného počtu lineárních funkcí. Z této vlastnosti plyne i konvexita ϕ. V prvním kroku tedy řešíme problém

$$\begin{aligned} \min_{x,\theta}\ & f_1(x) + \theta \\ s.t.\ & Ax = b, \\ & D_l x \geq d_l,\ l = 1, \ldots, r, \\ & E_l x + \theta \geq e_l,\ l = 1, \ldots, s, \\ & x \geq 0. \end{aligned} \tag{2.16}$$

Druhý krok je identický, protože se v něm nepracuje s účelovou funkcí, ale pouze s podmínkami. Ve třetím kroku uvažujme problém

$$\begin{aligned} \min_y \ & f_2(y, \xi_k) \\ s.t. \ & W_k y = h_k - T_k x^\nu, \\ & y \geq 0. \end{aligned} \tag{2.17}$$

Pro tento problém jsme schopní vytvořit duál (1.20), jehož optimální řešení označme π_k^ν. Protože sdružená funkce k polyedrické funkci je polyedrická funkce, je duální úloha k úloze (2.17) polyedrická. Na lineární ji můžeme převést podobným trikem, jako se problém (1.14) převedl na (1.16). Stejně můžeme na lineární úlohy převést i úlohy (2.16) a (2.17).

Předpokládejme, že algoritmus nebyl ve třetím kroku ukončen, ale byl přidán řez optimality generovaný pomocí lineární funkce $\tilde{\phi}(x) = \sum_{k=1}^{K} p_k (\pi_k^\nu)'(h_k - T_k x)$. Potom platí

$$\tilde{\phi}(x^\nu) = \sum_{k=1}^{K} p_k (\pi_k^\nu)'(h_k - T_k x^\nu) = \sum_{k=1}^{K} p_k \mathcal{Q}(x^\nu, \xi_k) = \phi(x^\nu).$$

Využijme znovu větu (1.19), podle které umíme spočítat subdiferenciál funkce ϕ v bodě x^ν jako

$$\partial\phi(x^\nu) = -\sum_{k=1}^{K} p_k T_k' \pi_k^\nu,$$

což je směrnice lineární funkce $\tilde{\phi}$. To spolu s konvexností ϕ a rovností $\phi(x^\nu) = \tilde{\phi}(x^\nu)$ znamená, že $\tilde{\phi}$ je opěrná funkce pro ϕ. Tímto jsme ukázali oprávněnost i třetího kroku.

Kromě lineárních účelových funkcí můžeme do polyedrických rozšířit i podmínky. Uvažujme problémy

$$\begin{aligned} \min_x \ & f_1(x) + \sum_{k=1}^{K} p_k \mathcal{Q}(x, \xi_k) \\ s.t. \ & g_1(x) \leq 0, \\ & x \geq 0, \end{aligned} \tag{2.18}$$

kde $\mathcal{Q}(x, \xi_k)$ je optimální hodnota problému

$$\begin{aligned} \min_y \ & f_2(y, \xi_k) \\ s.t. \ & g_2(x, y, \xi_k) \leq 0, \\ & y \geq 0. \end{aligned} \tag{2.19}$$

kde $f_1(x)$, $g_1(x)$ jsou polyedrické funkce, $f_2(y, \xi_k)$ je náhodná polyedrická funkce a pro každé x a k je $g_2(x, \cdot, \xi_k)$ polyedrická funkce.

Následující argumentaci provedeme pouze pro problém (2.18), pro (2.19) by se provedla obdobně. Protože $g_1(x)$ je též konvexní funkce a dolní úrovňové množiny konvexní funkce jsou konvexní množiny, problém (2.18) je konvexní problém.

Ukažme nyní, jak tento problém můžeme převést na lineární problém. Pro jednoduchost předpokládejme, že $\operatorname{dom} f_1$ a $\operatorname{dom} g_1$ jsou celé prostory. Potom můžeme napsat

$$f_1(x) = \max_{i=1,\dots,I}\{a_i x + b_i\}$$
$$g_1(x) = \max_{j=1,\dots,J}\{c_j x + d_j\}.$$

Potom problém (2.18) můžeme přepsat na problém

$$\begin{aligned} &\min_{x,u}\ u \\ &s.t.\ u \geq a_i x + b_i,\ i = 1,\dots,I, \\ &\quad c_j x + d_j \leq 0,\ j = 1,\dots,J, \\ &\quad x \geq 0, \end{aligned} \tag{2.20}$$

což je lineární problém.

Případ (2.18)–(2.19) je bohužel poslední problém, pro který dostáváme konečnou konvergenci. Jak uvidíme později, při rozšíření účelové funkce na konvexní funkci už nemáme zajištěnou ani obecnou konvergenci.

Stejně jako klasickou metodu lze na polyedrický případ rozšířit i ostatní varianty L–shaped metody. Pro úplnost připomeňme, že regularizovaná dekompozice byla obecněji navržena pro součet polyedrických funkcí, ale v práci jsme ji kvůli kompatibilitě zjednodušili.

2.6 Kvadratický případ

Zabývejme se nyní algoritmem pro kvadratické problémy, jak byl poprvé navržen v [15]. Mějme kvadratický problém dvojstupňového programování, kde první stupeň je tvaru

$$\begin{aligned} &\min_{x}\ c'x + \frac{1}{2}x'Cx + \sum_{k=1}^{K} p_k \mathcal{Q}(x,\xi_k) \\ &s.t.\ Ax = b, \\ &\quad x \geq 0, \end{aligned} \tag{2.21}$$

kde podobně jako v předchozích má náhodný vektor ξ konečný nosič a $\mathcal{Q}(x,\xi_k)$ je optimální hodnota problému druhého stupně

$$\begin{aligned} &\min_{y}\ q_k'y + \frac{1}{2}y'D_k y \\ &s.t.\ W_k y = h_k - T_k x, \\ &\quad y \geq 0. \end{aligned} \tag{2.22}$$

Předpokládejme, že matice C a D_k jsou pozitivně semidefinitní.

Dá se ukázat, že $f(x) = c'x + \frac{1}{2}x'Cx + \sum_{k=1}^{K} p_k \mathcal{Q}(x,\xi_k)$ je na $K_2 = \{x; \phi(x) < \infty\}$ konvexní a po částech kvadratická funkce. To znamená, že polyedrickou množinu K_2 můžeme rozdělit na konečný počet polyedrických podmnožin, které množinu K_2 pokrývají, průnik jakýchkoli dvou podmnožin má prázdný vnitřek a na každé této podmnožině je f kvadratická funkce. Tyto podmnožiny budeme nazývat buňky a budeme je značit C_ν.

Algoritmus bude znovu iterační, přičemž v každé iteraci vyřadíme alespoň jednu buňku, na které se optimální řešení nenalézá. Algoritmus nyní formulujme.

Krok 0: Položme $S_1 = K_1$, $\nu = 1$ a nalezněme libovolné $x^0 \in K_1$.

Krok 1: Nalezněme buňku C_ν takovou, že $x^{\nu-1} \in C_\nu$ a $C_\nu \cap \operatorname{int} S_\nu \neq \emptyset$. Potom pro všechna $x \in C_\nu$ platí $f(x) = z_\nu(x)$, kde z_ν je kvadratická funkce. Předpokládejme, že z_ν je kvadratická na celém S_ν.

Krok 2: Nalezněme

$$\begin{aligned} x^\nu &= \arg\min_{x \in S_\nu} z_\nu(x) \\ w^\nu &= \arg\min_{x \in C_\nu} z_\nu(x) \end{aligned} \tag{2.23}$$

Pokud není druhý problém v (2.23) zdola omezený, není omezený ani problém (2.21)–(2.22). Pokud platí $\nabla' z_\nu(w^\nu)(x^\nu - w^\nu) = 0$, algoritmus ukončíme, w^ν je optimální řešení, v opačném případě položíme

$$S_{\nu+1} = S_\nu \cap \{x; \nabla' z_\nu(w^\nu)(x - w^\nu) \leq 0\}, \tag{2.24}$$

zvětšíme ν o jedna a vrátíme se do prvního kroku.

Uveďme oprávněnost ukončovacího kritéria a jednotlivých řezů. Podrobnější důkaz a rozbor algoritmu je uveden v [15]. Nechť w^ν splňuje ukončovací kritérium. Potom pro všechny $x \in S_\nu$ platí

$$0 = \nabla z'_\nu(w^\nu)(x^\nu - w^\nu) \leq z_\nu(x^\nu) - z_\nu(w^\nu) \leq z_\nu(x) - z_\nu(w^\nu),$$

neboli w^ν minimalizuje z_ν na S_ν, z čehož ihned plyne $\nabla z'_\nu(w^\nu)(x - w^\nu) \geq 0$. Protože $\nabla z_\nu(w^\nu) \in \partial f(w^\nu)$, platí pro všechny $x \in S_\nu$ dále

$$f(x) \geq f(w^\nu) + \nabla z'_\nu(w^\nu)(x - w^\nu) \geq f(w^\nu),$$

neboli w^ν minimalizuje f na S_ν.

Nyní stačí ukázat oprávněnost generování řezů (2.24). Pro libovolné $x \in S_1$ platí

$$f(x) \geq f(w^\nu) + \nabla z'_\nu(w^\nu)(x - w^\nu).$$

Nutná podmínka pro to, aby x minimalizoval f na S_1 je tedy $\nabla z'_\nu(w^\nu)(x - w^\nu) \leq 0$. Z toho plyne, že f má na S_ν minimum ve stejném bodě jako na $S_1 = K_1$.

Algoritmus skončí po konečném počtu kroků, protože přidané řezy závisí pouze z_ν a těch může být vygenerován pouze konečný počet, který odpovídá počtu buněk.

2.7 Konvexní případ

Rozšiřme nyní L-shaped algoritmus do konvexního případu. Znovu pracujme s lineárními podmínkami a konečným nosičem Ξ. Jako první stupeň uvažujme

$$\min_{x \in X} f_1(x) + \sum_{k=1}^{K} p_k \mathcal{Q}(x, \xi_k), \tag{2.25}$$

kde $\mathcal{Q}(x, \xi_k)$ je optimální hodnota problému druhého stupně

$$\begin{aligned} &\min_{y \in Y^k} f_2^k(y) \\ &s.t.\ W_k y = h_k - T_k x. \end{aligned} \tag{2.26}$$

Předpokládejme, že množiny X a Y^k jsou polyedrické, X je navíc omezená a funkce f_1 a f_2^k jsou konvexní.

Podobně jako v lineárním a polyedrickém případu platí, že $\mathcal{Q}(\cdot, \xi_k)$ jsou konvexní a zdola polospojité funkce pro libovolné k. Subdiferenciál $\mathcal{Q}(\cdot, \xi_k)$ je za předpokladu konečnosti $\mathcal{Q}(x, \xi_k)$ roven

$$\partial \mathcal{Q}(x, \xi_k) = -T_k' \mathcal{D}(x, \xi_k), \tag{2.27}$$

kde $\mathcal{D}(x, \xi_k)$ je množina optimálních řešení duálního problému druhého stupně

$$\max_{\pi} \pi'(h_k - T_k x) - (\bar{f}_2^k)^*(W_k' \pi), \tag{2.28}$$

kde $\bar{f}_2^k$ je rozšíření funkce f_2^k na celý prostor

$$\bar{f}_2^k(y) = \begin{cases} f_2^k(y) & \text{pro } y \in Y^k \\ +\infty & \text{jinak.} \end{cases}$$

Podrobné znění těchto tvrzení i s důkazy je možné najít v [21] v části 3.2.5. Vzorec (2.28) je možné odvodit podobně jako (1.20) v polyedrickém případě.

Hlavní problém bude vypadat následovně

$$\begin{aligned} &\min_{x,\theta} \theta \\ &s.t.\ \bar{\alpha}_\nu + (\bar{g}_\nu)' x \le \theta,\ \nu \in J_1 \\ &\qquad \beta_{\nu k} + r_{\nu k}' x \le 0,\ \nu \in J_2^k \\ &\qquad x \in X, \end{aligned} \tag{2.29}$$

kde první typ podmínek jsou řezy optimality a druhý typ jsou řezy přípustnosti. Množiny J_1 a J_2^k budou v sobě uchovávat indexy řezů. Množina J_2^k je indexována přes k, protože aproximujeme množiny $\{x; \mathcal{Q}(x, \xi_k) < \infty\}$.

Zatímco v lineárním případě stačí odhadovat funkci ϕ a druhou část účelové funkce $c'x$ přidat do problému (2.2), který se řeší v prvním kroku, nyní je výhodnější odhadovat $f_1 + \phi$, čímž si problém (2.29) zachová linearitu. Na druhou stranu musíme v (2.30) zvlášť počítat subgradienty pro f_1. Uveďme nyní, jak dostaneme jednotlivé koeficienty v řezech.

Protože f_1 je konvexní, zřejmě platí

$$\begin{aligned} f_1(x) &\ge \alpha_{\nu 0} + (g_{\nu 0})' x \\ g_{\nu 0} &\in \partial f_1(x^\nu) \\ \alpha_{\nu 0} &= f_1(x^\nu) - (g_{\nu 0})' x^\nu. \end{aligned} \tag{2.30}$$

Díky (2.27) můžeme zdola odhadnout i jednotlivé funkce $\mathcal{Q}(x, \xi_k)$, kde pro libovolné $\pi_\nu^k \in \mathcal{D}(x^\nu, \xi_k)$ platí

$$\begin{aligned} \mathcal{Q}(x, \xi_k) &\ge \alpha_{\nu k} + (g_{\nu k})' x \\ g_{\nu k} &= -T_k' \pi_\nu^k \\ \alpha_{\nu k} &= \mathcal{Q}(x^\nu, \xi_k) + (\pi_\nu^k)' T_k x^\nu. \end{aligned} \tag{2.31}$$

Dohromady dostaneme

$$\begin{aligned} \bar{g}_\nu &= g_{\nu 0} + \sum_{k=1}^{K} p_k g_{\nu k} \\ \bar{\alpha}_\nu &= \alpha_{\nu 0} + \sum_{k=1}^{K} p_k \alpha_{\nu k}. \end{aligned} \tag{2.32}$$

Řezy přípustnosti na účelové funkci nezáleží, zůstávají tedy stejné jak pro lineární, tak pro konvexní případ.

Díky požadavku na konečnost množiny X máme zajištěno to, že problém (2.29) buď nebude mít přípustné řešení nebo bude existovat optimální řešení. Nemůže se stát, že účelová funkce bude neomezeně klesat. Bohužel za obecných předpokladů není zajištěna konvergence algoritmu, tomuto problému se budeme věnovat na závěr této části. Nyní ve zkratce sepišme algoritmus.

Krok 0: Položme $\nu = 1$, $J_1 = J_2^k = \emptyset$ a $\theta^1 = -\infty$.

Krok 1: Pro $k = 1, \ldots, K$ vyřešme problémy (2.26), ve kterých položíme $x = x^\nu$. Pro všechna k, pro která platí $\mathcal{Q}(x^\nu, \xi_k) = \infty$, přidáme řez přípustnosti a do indexové množiny J_2^k přidáme index ν. Pokud pro všechna k platí $\mathcal{Q}(x^\nu, \xi_k) < \infty$, přidáme řez optimality založený na (2.30)–(2.32).

Krok 2: Pokud je $f_1(x^\nu) + \sum_{k=1}^{K} p_k \mathcal{Q}(x^\nu, \xi_k) = \theta^\nu$, ukončíme algoritmus, našli jsme optimální řešení.

Krok 3: Vyřešme hlavní problém (2.29). Pokud neexistuje přípustné řešení, nemá přípustné řešení ani původní problém (2.25). V opačném případě označme optimální řešení $(x^{\nu+1}, \theta^{\nu+1})$, zvětšeme ν o jedna a vraťme se do prvního kroku.

Pro konvergenci algoritmu můžeme použít následující větu.

Věta 2.1. *Jestliže problém* (2.25) *nemá přípustné řešení, potom algoritmus skončí po konečně mnoha krocích v kroku 3. Předpokládejme dále, že existuje konstanta C taková, že pro všechny ν a pro $k = 1, \ldots, K$ platí $||g_{\nu k}|| \leq C$, pokud $\mathcal{Q}(x^\nu, \xi_k) < \infty$. Pokud má problém* (2.25) *přípustné řešení, potom algoritmus buď skončí v kroku 2 nebo nalezne posloupnost bodů $x^\nu \in X$ takových, že platí*

$$\lim_{\nu \to \infty} f(x^\nu) = \bar{f},$$

kde $\bar{f}$ je optimální hodnota účelové funkce problému (2.25).

Důkaz. Důkaz byl proveden v [21], jedná se o větu 7 v kapitole 3. □

Stejně jako základní metodu lze do polyedrického případu rozšířit i ostatní varianty L-shaped metody. Pro úplnost připomeňme, že regularizovaná dekompozice byla obecněji navržena pro součet polyedrických funkcí, ale v práci jsme ji kvůli kompatibilitě zjednodušili.

2.8 Progressive hedging algoritmus

V této kapitole se budeme poprvé zabývat problémem, u kterého nebudeme vyžadovat polyedrickou množinu přípustných řešení. Zároveň poprvé zmíníme podmínky neanticipativity.

Úkolem je vyřešit problém

$$\begin{aligned} &\min_{x, y_k} f_1(x) + \sum_{k=1}^{K} p_k f_2(y_k, \xi_k) \\ &s.t.\ (x, y_k) \in C_k \subset \mathbb{R}^{n+m}, \end{aligned} \tag{2.33}$$

kde $f_1(\cdot)$ a $f_2(\cdot, \xi_k)$ jsou konvexní funkce a C_k jsou konvexní množiny.

Algoritmus byl poprvé popsán v [18] pro vícestupňové programování. Protože se v této práci zabýváme pouze dvojstupňovými úlohami, algoritmus je v práci uveden ve zjednodušené formě. Nicméně mezi verzemi algoritmu pro dvojstupňové a vícestupňové úlohy nejsou velké rozdíly. Základní myšlenkou algoritmu je místo x zavést K proměnných x_k a následně řešit problém

$$\begin{aligned} &\min_{x_k,y_k} \sum_{k=1}^{K} p_k[f_1(x_k) + f_2(y_k,\xi_k)] \\ &s.t.\ (x_k,y_k) \in C_k. \end{aligned} \tag{2.34}$$

Tento problém se rozpadne na K menších problémů, které se snadněji řeší. Bohužel problémy (2.34) a (2.33) nemají stejnou optimální hodnotu účelové funkce. Z tohoto důvodu se musí přidat podmínky neanticipativity, které říkají, že jednotlivé x_k musí být identické. Možností, jak tohoto dosáhnout, je víc, mezi nejjednodušší patří přidání $K-1$ rovnic $x_1 = x_2, \ldots, x_{K-1} = x_K$.

Jako další definujme pojem strategie, pod kterým rozumíme zobrazení z množiny Ξ do množiny $\mathbb{R}^{n+m} = \{(x,y); x \in \mathbb{R}^n, y \in \mathbb{R}^m\}$. Jednotlivé strategie budeme označovat $X(\xi_k) = (x_k, y_k)$, množinu všech strategií označme $\mathcal{E}$. Množinu přípustných strategií označme $\mathcal{C} = \{X \in \mathcal{E}; (x_k,y_k) \in C_k\}$. Pod množinou implementovatelných strategií uvažujme ty strategie, které splňují podmínky neanticipativity, neboli patří do množiny $\mathcal{N} = \{X \in \mathcal{E}; x_1 = \cdots = x_K\}$.

Problém (2.33) tedy můžeme převést na problém

$$\begin{aligned} &\min_{X}\ f_1(x_k) + \sum_{k=1}^{K} p_k f_2(y_k,\xi_k) \\ &s.t.\ X \in \mathcal{C} \cap \mathcal{N}. \end{aligned} \tag{2.35}$$

V algoritmu budeme postupně vytvářet implementovatelné strategie a přípustné strategie. Přípustnou strategii dostaneme řešením vhodné optimalizační úlohy, ve které jedna z podmínek bude $X \in \mathcal{C}$. Z libovolné přípustné strategie $X(\xi_k) = (x_k, y_k)$ můžeme dostat implementovatelnou strategii $\hat{X}$ pomocí $\hat{X}(\xi_k) = (\sum_{k=1}^{K} p_k x_k, y_k)$. Operátor tohoto přiřazení označme $J : X \mapsto \hat{X}$. Zřejmě se jedná o projekci na podprostor implementovatelných strategií $\mathcal{N}$. Označme dále $K = I - J$, kde I je identické zobrazení. Potom $\mathcal{N} = \{X \in \mathcal{E}; KX = 0\}$.

Označme $f^k(x_k,y_k) = f_1(x_k) + f_2(y_k,\xi_k)$ účelové funkce problémů, do kterých se rozpadne (2.34) a účelovou funkci (2.34) označme $F(X) = \sum_{k=1}^{K} p_k f^k(x_k,y_k)$. Potom původní problém (2.33) můžeme zapsat jako problém

$$\begin{aligned} &\min_{X}\ F(X) \\ &s.t.\ X \in \mathcal{C}, \\ &\quad KX = 0. \end{aligned} \tag{2.36}$$

Lagrangeova funkce problému (2.36) je rovna

$$F(X) + \langle Y, KX \rangle,$$

kde $Y \in \mathcal{E}$ jsou Lagrangeovy multiplikátory, $X \in \mathcal{C}$ a skalární součin je definován jako $\langle X^1, X^2 \rangle = \sum_{k=1}^{K} p_k[(x_k^1)' x_k^2 + (y_k^1)' y_k^2]$. Protože K je projekce, platí

$\langle Y, KX\rangle = \langle KY, X\rangle = \langle KY, KX\rangle$. Díky tomuto stačí místo $\langle Y, KX\rangle$ uvažovat $\langle X, W\rangle$, kde $W \in \mathcal{M}$, kde $\mathcal{M} = \mathcal{N}^{\perp}$.

Zaveďme novou Lagrangeovu funkci, která bude tvaru

$$L(X, W) = F(X) + \langle X, W\rangle + \frac{1}{2} r||X - \hat{X}^{\nu}||^2, \tag{2.37}$$

kde $X \in \mathcal{C}$ a $W \in \mathcal{M}$. Poslední člen v součtu je založen na teorii rozšířených Langrangeových multiplikátorů, kterou můžeme nalézt například v [3]. Dá se říci, že se jedná o penalizující člen a parametr $r > 0$ určuje velikost penalizace.

V algoritmu budeme postupně konstruovat strategie $X^{\nu}, \hat{X}^{\nu}, W^{\nu}$. První z nich bude přípustná strategie, která nemusí být implementovatelná. Na druhou stranu $\hat{X}^{\nu}$ bude vždy implementovatelná strategie, která ale nemusí být přípustná. Pokud by $\hat{X}^{\nu}$ byla i přípustná strategie, nemusí být ještě nutně optimálním řešením problému (2.33).

Z věty 2.3, kterou zmíníme později, plyne, že $\hat{X}^{\nu} \to \bar{X}$, kde $\bar{X}$ je optimální řešení problému (2.33). Problém je v tom, že konvergence není konečná, algoritmus se tedy zastaví po splnění nějakého ukončovacího kritéria. Jako optimální strategii je potom možné zvolit X^{ν} nebo $\hat{X}^{\nu}$. Doporučuje se zvolit $\hat{X}^{\nu}$. Ačkoli tato strategie nemusí být přípustná, u strategie X^{ν} neexistuje jednoznačné x, takže není možné udělat počáteční rozhodnutí. Pokud bychom u X^{ν} udělali z první souřadnice střední hodnotu, dostaneme právě strategii $\hat{X}^{\nu}$. Uveďme nyní celý algoritmus.

Krok 0: Zvolme $\nu = 1$, najděme libovolné $X^1 \in \mathcal{C}$ a položme $W^1 = 0$.

Krok 1: Spočtěme $\hat{X}^{\nu} = JX^{\nu}$, což je implementovatelná strategie, ale nemusí to být nutně přípustná strategie. Pokud budeme chtít zastavit algoritmus, za optimální řešení vezmeme $\hat{X}^{\nu}$.

Krok 2: Spočtěme $X^{\nu+1}$ jako optimální řešení problému

$$\begin{aligned} &\min_{X}\ F(X) + \langle X, W^{\nu}\rangle + \frac{1}{2} r||X - \hat{X}^{\nu}||^2 \\ &s.t.\ X \in \mathcal{C}. \end{aligned} \tag{2.38}$$

Tento problém se rozloží na vyřešení K problémů

$$\begin{aligned} &\min_{x,y}\ p_k[f_1(x) + f_2(y, \xi_k) + x' W^{\nu}(\xi_k) + \frac{1}{2} r|x - \hat{X}^{\nu}(\xi_k)|^2] \\ &s.t.\ (x, y) \in C_k. \end{aligned} \tag{2.39}$$

Strategie $X^{\nu+1}$ bude přípustná, ale nemusí být implementovatelná.

Krok 3: Spočtěme $W^{\nu+1} = W^{\nu} + rKX^{\nu+1}$. Pro tuto strategii bude platit $W^{\nu+1} \in \mathcal{M}$. Zvětšeme ν o jedna a vraťme se do prvního kroku.

Ještě před samotnou větou o konvergenci zmiňme větu o vztahu problémů (2.33) a (2.34).

Věta 2.2. *Pokud označíme* $\hat{\alpha} = \min_X\{F(X); X \in \mathcal{C}\}$ *optimální hodnotu* (2.34) *a pro* $U \in \mathcal{M}$ *označíme* $\Phi(U) = \min_X\{F(X); X \in \mathcal{C}, KX = U\}$, *pak platí*

$$\hat{\alpha} = \min_{U \in \mathcal{M}} \Phi(U).$$

Optimální hodnota (2.33) *je zřejmě rovna* $\Phi(0)$.

Důkaz. Důkaz byl proveden v [18]. □

Věta 2.3. *Nechť pro všechna k jsou C_k uzavřené, neprázdné a konvexní množiny, funkce f^k jsou konvexní a dolní úrovňové množiny $\{(x,y) \in C_k; f^k(x) \leq \alpha\}$ jsou omezené pro všechna α. Nechť $X^1 \in \mathcal{C}$ je libovolný startovací bod progressive hedging algoritmu.*

Potom jsou posloupnosti $\{\hat{X}^\nu\}$ a $\{W^\nu\}$ omezené právě tehdy, pokud existuje optimální řešení problému (2.33). V tomto případě navíc platí $\hat{X}^\nu \to \bar{X}$ a $W^\nu \to \bar{W}$, kde $\bar{X}$ je optimální řešení (2.36) a $\bar{W}$ jsou optimální Lagrangeovy multiplikátory příslušné stejnému problému.

Pokud zavedeme normu $||(X,W)||_r = (||X||^2 + r^{-2}||W||^2)^{\frac{1}{2}}$, pak platí

$$||(\hat{X}^{\nu+1}, W^{\nu+1}) - (\bar{X}, \bar{W})||_r \leq ||(\hat{X}^\nu, W^\nu) - (\bar{X}, \bar{W})||_r.$$

Předchozí nerovnost je ostrá, pokud neplatí $(\hat{X}^\nu, W^\nu) = (\bar{X}, \bar{W})$.

Důkaz. Proveden v [18]. □

Na závěr kapitoly ukažme, že ačkoli jsou algoritmy zapsané krok po kroku, v praktické implementaci je dobré je spíše než jako posloupnost kroků uvažovat jako celek. Toto využijeme později v algoritmu, který popíšeme v části 4.2. Ačkoli máme k dispozici explicitní zápis pro účelovou funkci problému (2.39), je vhodnější tuto funkci počítat iteračně. Dá se totiž ukázat, že pro účelovou funkci problému (2.39) pro iteraci $\nu + 1$ stačí vzít účelovou funkci stejného problému z iterace ν a připočíst k ní lineární člen.

Kapitola 3

Software

Tato kapitola je věnována softwaru a solverům, které jsou schopny řešit úlohy stochastického programování. Existují dvě možnosti přístupu k řešení úlohy. Při první se úloha převede na deterministický ekvivalent, který se následně vyřeší. V tomto případě můžeme použít obecné solvery, které si jsou schopné poradit i s úlohami nelineárního programování. Druhou možností je pak využití speciálního softwaru, který umí využít specifik úlohy dvojstupňového programování. Tento software je bohužel limitován pouze do lineárního případu, snad s dvěma výjimkami, které jdou na lineární případ převést. Jedná se o modely s individuálními pravděpodobnostními omezeními, kde ale může být stochastická pouze pravá strana, na kterou se kladou poté další požadavky. Úloha se pak převede do lineárního případu pomocí kvantilové funkce. Druhou možností je model s CVaRem, který budeme podrobněji rozebírat v části 4.1.

Bohužel nebudeme moci zcela využít potenciál této kapitoly, protože největší výhody stochastického softwaru se projeví až ve vícestupňovém programovaní, které je rozšířením dvojstupňového. Protože by některé části této kapitoly byly pro dvojstupňové úlohy zbytečně složité, a nebylo by zřejmé, proč byla daná věc udělána tak, jak byla udělána, na úvod kapitoly jsou zmíněny nejdůležitější pojmy, hlavně strom scénářů a SMPS formát, který byl vyvinut jako standardizace zápisů stochastických problémů a slouží k předání úlohy mezi software a solverem.

Jako další jsou rozebrány některé typy softwaru, nejprve obecné algebraické modelovací jazyky a poté specializovaný stochastický software. Přehled této skupiny softwaru je možné nalézt například v [13] nebo [24].

3.1 Strom scénářů

Stromy scénářů přímo souvisí s vícestupňovými úlohami, při kterých nejprve zvolíme x_1, načež se dozvíme realizaci náhodného vektoru ζ_2, který korigujeme volbou x_2. Po této volbě se dozvíme realizaci ζ_3, což při N–stupňové úloze pokračuje až po x_N.

Za předpokladu diskrétního rozdělení $\zeta_2, \ldots, \zeta_N$ jsme schopni sestrojit scénářový strom. Kořen odpovídá první fázi, která je deterministická, jednotlivé uzly v hloubce i pak odpovídají situaci, kdy je známá realizace ζ_2 až ζ_i a máme za úkol zvolit x_i. Pokud máme dva uzly na úrovni i a jejich poslední společný předek je v hloubce i', pak realizace $\zeta_2, \ldots, \zeta_i$ se budou poprvé lišit až v náhodném vektoru $\zeta_{i'+1}$. Pod scénářem budeme rozumět jednu realizaci náhodných veličin

$(\zeta_2, \ldots, \zeta_N)$, neboli cestu z kořene stromu do jeho libovolného listu.

Zde vidíme první specifikum dvojstupňového programování, ve kterém je K scénářů, které odpovídají jednotlivým realizacím $\xi_1, \ldots, \xi_K$. Každá větev scénářového stromu bude odpovídat jednomu scénáři, přičemž kořen odpovídá prvnímu stupni, můžeme říci, že ζ_1 má pouze jednu možnou realizaci, je tedy deterministická. Pro dvojstupňové programování bude strom jednoduchý, z kořene povede K potomků, kteří budou zároveň listy.

Pokud bychom chtěli najít optimální $x_1, \ldots, x_N$ za předem dané realizace náhodných vektorů, museli bychom vyřešit takzvaný výchozí deterministický problém, který by například při dvojstupňovém programování vypadal následovně

$$\begin{aligned} \min_{x \in X, y} \; & c'x + q_k'y \\ s.t. \; & T_k x + W_k y = h_k, \\ & y \geq 0. \end{aligned} \tag{3.1}$$

Tímto jsme položili dostatečný základ pro další části této kapitoly a můžeme navázat SMPS formátem.

3.2 SMPS formát

Tento formát byl poprvé navržen v [4] jako reakce na nutnost vytvoření jednotného vstupu pro solvery řešící stochastickou optimalizaci. SMPS formát se skládá ze tří souborů, jedná se o core, time a stoch soubory, která mají postupně přípony .cor, .tim a .sto.

Jedná se o tři textové soubory, které mají přesně popsanou strukturu. Na jednom řádku se nachází jeden až sedm vstupů, důležitá je hlavně pozice v řádku, na které se vstup nachází. Pokud řádek začíná na první pozici, jedná se o hlavičku, například příkazem `ROWS` se definují názvy řádků matice. Pokud záznam začíná na druhé pozici nebo později, jedná se o záznam, který patří pod první předcházející hlavičku. Podle pozice se jednotlivé vstupy rozdělují na kód, tři jmenné a tři číselné pole. Například kód se může nacházet na pozici 2–4 a může se jednat například o určení typu nerovnosti nebo specifikování, že na dalších polích bude pro proměnnou specifikována horní mez.

Z tohoto je vidět, že jednotlivé sloupce matice se neoddělují čárkou nebo středníkem, ale vstupy mají vyhrazeny pozice v řádku, což má za následek omezenou přesnost zadávání čísel, která je ale dostatečných patnáct číslic. Nyní popišme všechny tři soubory.

V core souboru je popsán výchozí deterministický problém. Zatímco (3.1) závisí na k, v core souboru popíšeme pouze tvar rovnic, ale hodnoty stochastických vektorů nemusíme specifikovat. To se provede později v stoch souboru, kde nejprve určíme, které vektory jsou náhodné, a poté určíme jejich rozdělení. Na druhou stranu je v core souboru nutné specifikovat hodnoty deterministických parametrů.

Core soubor je založen na MPS formátu a obsahuje postupně jméno problému, pojmenování jednotlivých řádků matice, které odpovídají podmínkám v modelu, a určení, jestli se jedná o nerovnost nebo rovnost. Dále jsou pojmenované jednotlivé sloupce matice, které odpovídají proměnným a následně jsou zadány číselné hodnoty. Pokud pro nějaký prvek matice není zadána hodnota, automaticky se

předpokládá, že je nulová. Toto je důležitý předpoklad pro stochastické programování, ve kterém matice výchozího deterministického problému obsahuje velké množství nul. Pokud by se tyto nuly musely zapisovat, obsah souboru by značně vzrostl. Pro jednotlivé řádky poté následuje definice pravých stran a pro sloupce můžeme definovat dolní a horní meze.

V time souboru definujeme rozdělení celého problému na jednotlivé stupně. Optimální je, pokud jsou řádky a sloupce v core souboru uspořádány tak, že s rostoucím indexem roste i stupeň, ve kterém se daný řádek nebo sloupec nalézá. V tomto případě stačí pro každý stupeň přiřadit index prvního řádku a sloupce, který patří do daného stupně. Pokud předchozí předpoklad splněn není, musí se každému řádku a sloupci explicitně přiřadit stupeň. Pro dvojstupňové programování bude tento soubor obsahovat pouze pět řádků.

V stoch souboru se určuje rozdělení náhodných veličin a postaví se strom scénářů, který může být nagenerovaný třemi způsoby.

Při prvním způsobu se použije příkaz `SCENARIOS` a vypíše se první scénář, jeho pravděpodobnost a hodnoty parametrů, kterých se při tomto scénáři nabývá. Pro další scénáře určíme otcovský scénář a první stupeň, ve kterém se od otcovského scénáře liší. Pro lišící se stupně vypíšeme hodnoty parametrů a samozřejmě pravděpodobnost scénáře.

Druhou možností je použití příkazu `NODES`, při kterém nepopisujeme scénáře, ale jednotlivé uzly. Každému uzlu přiřadíme jeho otce, pravděpodobnost, že se z otce přejde do daného syna a hodnoty náhodných veličin, které se v daném uzlu nabývají. Tato možnost se může například použít, pokud jednotlivé scénáře nemají stejnou délku.

Poslední možnost se využije, pokud jsou jednotlivé náhodné veličiny nezávislé mezi jednotlivými stupni. Klíčová slova jsou v tomto případě `INDEP` nebo `BLOCKS`. V prvním případě definujeme nezávislé náhodné veličiny, v druhém pak nezávislé náhodné vektory. Pro jednorozměrné náhodné veličiny je možno definovat diskrétní, rovnoměrné spojité, normální, gamma, beta a lognormální rozdělení. Ostatní je nutné definovat přes speciálně napsané subrutiny. Pro náhodné vektory je možné zadefinovat pouze diskrétní, normální a subrutiny.

Protože úloha dvojstupňového programování obsahuje pouze dva stupně, z nichž první je deterministický, jsou si všechny tři popsané způsoby ekvivalentní. Dále je nutno podotknout, že možnost zavedení spojitého nebo vlastního rozdělení neznamená, že bude existovat solver, který je schopný danou úlohu vyřešit. Jedná se pouze o zápis modelu.

Protože jednoduchá kompenzace vykazuje určité speciální rysy, jako například úplná kompenzace nebo možnost rozdělení účelové funkce na jednotlivé komponenty, je pro ní udělán speciální příkaz, do kterého se mimo jiné zadají i penalizace pro nesplnění podmínky. Podobně existují i speciální příkazy pro modely s pravděpodobnostními omezeními. Bohužel ale neexistuje příkaz, kterým by se zadefinoval model s CVaRem. Pro podrobnější informace odkazuji například na online manuál [6].

Tento formát je relativně starý, první článek byl vydán roku 1987, z čehož plyne, že je optimalizován pro menší úlohy, které byly tehdejší počítače schopné zpracovat. Při zpracování větších úloh může dojít k tomu, že stoch soubor bude neúnosně velký. O tomto pojednáme více v kapitole 4.

3.3 Algebraické modelovací jazyky

Ačkoli algebraické modelovací jazyky nejsou primárně určeny pro stochastické programování, lze jimi řešit deterministický ekvivalent nebo na jejich základě může být postaveno rozšíření, které si se stochastickými problémy poradit umí. Patří mezi ně například GAMS, AMPL nebo AIMMS. Protože mám zkušenosti pouze s prvním z nich, veškeré příklady této části se budou týkat právě GAMSu.

V porovnání s moderními programovacími jazyky, které jsou postaveny na principu objektově orientovaného programování a pracují s třídami, pro které jsou zadefinované vlastní metody, kód v algebraických modelovacích jazycích se snaží co nejvíce přiblížit matematickému zápisu rovnic. Kód je uspořádán lineárně, není tedy například možné vytvořit funkci, kterou budeme volat z různých částí programu. Existuje možnost napsání dalšího .gms souboru, který budeme volat přes `$include`, ale toto trpí několika velkými nedostatky, například viditelností proměnných.

Důležitým pojmem jsou množiny, pomocí kterých se indexují jednotlivé proměnné nebo rovnice. Zde uvedeme příklad dalšího rozdílu, kdy v programovacích jazycích se pole standardně čísluje od 0, zatímco v algebraických modelovacích jazycích se prvky množiny nechovají jako čísla, ale jako symboly. Tímto vzniká například problém v přistupování k jednotlivým prvkům pole, které je v GAMSu složité.

Zápis kódu v algebraických modelovacích jazycích nemá předepsanou strukturu, ale jedna z nejčastějších struktur kódu je následující: deklarace množin, parametrů a tabulek, která může být spojena s inicializací, deklarace proměnných a rovnic, popsání jednotlivých rovnic, vytvoření modelu a na závěr zavolání optimalizace pomocí příkazu `SOLVE`. Je samozřejmě možné vybrat požadovaný solver a v průběhu programu provést pomocné výpočty.

Algebraický modelovací jazyk si následně vytvoří účelovou funkci a podmínky problému a proměnným přiřadí dolní a horní meze, a pokud je zadána, tak i počáteční hodnotu pro první iteraci optimalizace, což lze využít hlavně v nelineární optimalizaci. Po vytvoření problému se provede presolve, ve kterém se program snaží zjistit neřešitelnost problému, vyřadit nadbytečné podmínky nebo zmenšit horní či zvětšit dolní mez pro jednotlivé proměnné.

Po provedení presolve se model uloží v MPS formátu, který je následně poslán solveru, který ho vyřeší. Na standardním výstupu jsou vidět optimální hodnoty proměnných a účelové funkce a pro jednotlivé podmínky jsou spočteny optimální Lagrangeovy multiplikátory.

Algebraické modelovací jazyky nejsou primárně určeny pro stochastické problémy, jejich síla spočívá ve výkonných solverech, které jsou schopny řešit lineární, nelineární nebo třeba i celočíselné programování. Pokud bychom v nich přesto chtěli řešit stochastické problémy, je toto možné pouze přes deterministické ekvivalenty.

Vytvořit deterministický ekvivalent pro dvojstupňové programování je jednoduché, například v lineárním případě stačí zapsat (1.8). Větší problém pak vzniká u vícestupňových problémů, kdy se musí konstruovat scénářové stromy. K této konstrukci jde využít například podmínek neanticipativity, které jsme zmínili v části 2.8.

Jak už bylo řečeno jednou z největších nevýhod deterministického ekvivalentu

je velikost problému, s kterou souvisí i délka jeho generování. Jednou z možností, jak zkrátit délku generování je například použití speciálního softwaru pro generování lineárních a kvadratických stochastických problémů LEQGEN. Více informací o tomto softwaru je možné nalézt v [7]. O možnosti spojení GAMSu s objektově orientovaným jazykem pojednáme více v části 4.5.

3.4 Condor

Condor je software, který umožňuje provádět výpočty na síti počítačů. K výpočtům je kromě vlastního počítače možné použít i ostatní počítače, které jsou zapojeny do stejné sítě a nejsou právě využívány. Vlastník počítače si samozřejmě může nastavit podmínky, za kterých Condoru povolí vykonávání výpočtů ostatních uživatelů sítě, což s sebou nese i možnost přerušení právě probíhajícího výpočtu. Condor sám o sobě není schopný provádět optimalizaci, je ho tedy nutné propojit s jiným softwarem, který je tohoto schopen.

Spojení L–shaped algoritmu s Condorem je velice účinné, protože L–shaped algoritmus je případem paralelního algoritmu, kdy jednotlivé kroky na sebe nemusí navazovat, ale většina výpočtů může probíhat paralelně. Konkrétněji na jednom počítači bude spuštěn hlavní problém, který v případě lineárního programování odpovídá (2.2), zatímco řešení subproblémů (2.3) a (2.5) a generování řezů přípustnosti a optimality se provádí na jiných počítačích.

Jako základ si vezměme článek [14], ve kterém se provedla asynchronní verze L–shaped algoritmu s lokálně omezeným krokem. Pro jednoduchost se omezíme pouze na asynchronní verzi základního L–shaped algoritmu, která je jednodušší a má stejný princip. Pro jednoduchost předpokládejme dále, že K_1 je omezená, a že kompenzace je relativně úplná, a tedy nemusíme uvažovat řezy přípustnosti. Ačkoli v původním článku mohl hlavní počítač odeslat na jiný počítač více úloh najednou, předpokládejme, že tento počet je omezen na jednu úlohu.

V L–shaped algoritmu se v jedné iteraci nejprve provedly všechny výpočty, které pro danou iteraci byly potřeba a teprve potom se přešlo do další iterace. V asynchronní verzi není tento postup nejlepší, pokud by hlavní počítač čekal na vyřešení všech subproblémů, nebyl by proces efektivní, protože zlomek řešení by se mohl zpozdit nebo být terminován. Toto je vyřešeno parametrem σ, který udává, jaká část řešení jednotlivých subproblémů se musí vrátit, aby mohl hlavní počítač přistoupit k další iteraci. Toto má za následek vlastnost, že zatímco hlavní počítač vyřešil například pátý hlavní problém, stále mohou z ostatních počítačů docházet řezy z první iterace.

Díky tomuto je vhodné pozměnit ukončovací kritérium algoritmu. Parametr θ^ν spočítáme totiž v hlavním problému, zatímco τ^ν spočítáme až v okamžiku, kdy se vrátí řešení všech subproblémů. Zaveďme tedy proměnnou $\tau_{\min}$, ve které budeme ukládat minimální hodnotu účelové funkce (1.8) pro jednotlivé iterace $x^1, x^2, \ldots$ a příslušné řešení druhého stupně. Ukončovací kritérium budeme ale provádět, kdykoli se nám v nějaké iteraci vrátí část řezů, která je úměrná parametru σ. Zatímco v původním L–shaped algoritmu jsme v ukončovacím kritériu porovnávali τ^ν a θ^ν, v této verzi budeme porovnávat $\min\{\tau^1, \ldots, \tau^{\nu_1}\}$ a θ^{ν_2}, kde $\nu_2 \geq \nu_1$. Dále místo absolutního kritéria $\tau_{\min} \leq \theta^\nu$ používáme relativní kritérium $\tau_{\min} - \theta^\nu \leq \varepsilon_{tol}(1 + |\tau_{\min}|)$.

Algoritmus můžeme rozdělit do čtyř základních metod. Jako první se volá inicializační metoda `f0()`.

`f0()` V této metodě inicializujeme proměnné. Zvolíme toleranci $\varepsilon_{tol} \geq 0$, práh pro postup do další iterace $\sigma \in (0, 1]$, pořadí iterace $\nu = 1$, boolovskou proměnnou určující, zda hlavní problém už řešil další iteraci $\texttt{spaceval}^1 = \texttt{false}$, $\tau_{\min} = \infty$ a počet subproblémů, které už byly v dané iteraci vyřešeny $i^1 = 0$. Vyřešíme problém (2.2), jehož optimální řešení označme (x^1, θ^1). Nakonec zavoláme metodu $\texttt{f1}(\texttt{x}^1, 1)$.

$\texttt{f1}(\texttt{x}^\nu, \nu)$ V tomto kroku pouze pro $k = 1, \ldots, K$ zavoláme jednotlivé subproblémy, tedy $\texttt{f2}(\texttt{x}^\nu, \nu, \texttt{k})$. Pokud je to možné, problémy neřešíme postupně, ale současně je odešleme na různé počítače.

$\texttt{f2}(\texttt{x}^\nu, \nu, \texttt{k})$ Toto je jediná metoda, která probíhá na vedlejších počítačích, zbylé tři se vyhodnocují na hlavním počítači. V této metodě vygenerujeme jednotlivé řezy stejně jako v třetím kroku L–shaped algoritmu. Po vyřešení optimalizace zavolá každý subproblém $\texttt{f3}(\texttt{x}^\nu, \nu, \texttt{k}, \texttt{data})$, kde proměnná `data` v sobě obsahuje číselné výsledky řezů.

$\texttt{f3}(\texttt{x}^\nu, \nu, \texttt{k}, \texttt{data})$ V samotné metodě nejdříve přidáme do modelu řezy, které vygenerovala předcházející metoda a zvětšíme i^ν o jedna. Pokud $i^\nu = K$, dostali jsme poslední informaci o ν-té iteraci a můžeme vyhodnotit $\tau_{\min} = \min\{\tau_{\min}, \tau^\nu\}$. Pokud platí $i^\nu \geq \sigma K$ a zároveň neplatí $\texttt{spaceval}^\nu$, přišlo nám dostatek řezů z jednotlivých subproblémů, ale ještě jsme nepřešli k další iteraci. Položíme tedy $\texttt{spaceval}^\nu = \texttt{true}$, provedeme ukončovací kritérium, pokud je tedy splněno

$$\tau_{\min} - \theta^\nu \leq \varepsilon_{tol}(1 + |\tau_{\min}|),$$

algoritmus ukončíme. V opačném případě zvětšíme ν o jedna a vyřešíme (2.2), čímž dostaneme x^ν a zavoláme metodu $\texttt{f1}(\texttt{x}^\nu, \nu)$.

V poslední metodě je v původním článku menší chyba, kdy program při volbě $\sigma > \frac{K-1}{K}$ nikdy nezavolá druhou iteraci, ale běh programu skončí vždy první iterací. Řešení je jednoduché, nahradit if-else-if konstrukci dvěma if konstrukcemi.

Pro další informace o paralelních algoritmech odkazuji na manuál GAMSu [19], konkrétně na apendix I. V dalších dvou částech se budeme zabývat softwarem, který už je schopen vytvořit úlohu stochastického programování a sám ji i vyřešit. Konkrétně se jedná o SLP–IOR a SPInE.

3.5 SLP–IOR

SLP–IOR je software, který je specializován převážně na lineární stochastické programování. První verze vyžadovaly nainstalovaný GAMS, ať už plnou nebo studentskou verzi, později ale byly implementovány vlastní solvery a připojení GAMSu se z nutnosti stalo pouze výhodou.

SLP–IOR v sobě obsahuje grafické rozhraní, ve kterém se daný problém vytvoří a zároveň se zadají data. Postup vytváření přibližně odpovídá vytváření SMPS formátu z části 3.2. Nejprve se vybere typ modelu, který se chce použít, poté se vytvoří výchozí deterministický problém, který se následně rozdělí na jednotlivé stupně. Následně se zadají deterministická data, určí se, které parametry jsou stochastické a na závěr se zadá rozdělení náhodných veličin. Na podrobnější popis vytváření modelu odkazuji na manuál [9].

Kromě manuálního vytvoření problému podporuje SLP–IOR i načtení SMPS formátu. Ve většině případů ale tento soubor nebude existovat. Pokud máme k dispozici alespoň MPS formát, popřípadě zapsání problému v jazyce GAMS, je možné tento problém nahrát a dodatečně k němu vytvořit stochastickou strukturu. Existuje samozřejmě i obrácený způsob, ze stochastického souboru je SLP–IOR schopný vytvořit jak SMPS soubor, tak i deterministický ekvivalent, neboli MPS soubor.

Bohužel SLP–IOR je klikací aplikace, což znamená, že při vytváření složitější úlohy není možné data vypsat ručně, ale je nutné je importovat přes SMPS formát, který se předem vytvoří. Toto má dále za následek, že případné výpočty se musí provést v jiném programu.

Věnujme se nyní volbě modelu, při které je dobré co nejpodrobněji specifikovat použitý model. SLP–IOR je schopný řešit jak jedno-, tak dvoj- i vícestupňové lineární problémy. Pro dvojstupňové programování postupně existují možnosti výběru z pevné, úplné a jednoduché kompenzace. Čím více specifikujeme kompenzaci, tím více specializovaný solver je možné použít, a tím rychlejší by měl být výpočet. SLP–IOR navíc obsahuje možnost rozeznání, zda má problém jednoduchou kompenzaci. Jednotlivé typu modelů jdou mezi sebou samozřejmě konvertovat.

SLP–IOR je ve speciálních případech schopný řešit i nelineární problémy, pro jednostupňové úlohy se za vcelku silných předpokladů jedná o modely s pravděpodobnostními omezeními nebo CVaRem. Ačkoli je možné vytvořit v podstatě libovolný problém, není jisté, zda bude existovat solver, který bude schopen zadaný problém vyřešit. Pokud je problém lineární a rozdělení náhodné veličiny diskrétní, je možné problém převést na deterministický problém, pro který solvery vždy existují. Pokud má náhodná veličina spojité rozdělení, dává SLP–IOR možnost její diskretizace.

Jako další funkci SLP–IOR zmiňme možnost připojení vlastního solveru a baterii testů pro jeho následné otestování. Pro jednotlivé solvery spojené se SLP–IOR odkazuji znovu na manuál [9], konkrétně kapitoly 17-18.

3.6 SPInE

V části 3.3 jsme se zabývali algebraickými modelovacími jazyky, přičemž jeden z nich byl i AMPL. Vytvořením grafického rozhraní a přidáním SPInE vznikla stochastická varianta SAMPL. Na rozdíl od SLP–IOR se nejedná o klikací aplikaci, ale díky AMPL je umožněno programování a díky SPInE i lineární stochastické programování.

Kód je rozdělený do několika samostatných částí. První z nich je modelová část, ve které definujeme výchozí deterministický problém a pomocí několika speciálních příkazů stochastickou část modelu. Druhá část je datová, ve které zadáváme data, které je možné zadat manuálně nebo načíst z textového souboru nebo souboru vytvořeného pomocí MS Excel či MS Access. Obě části mohou být spojeny do jedné, data se dají definovat i v modelové části. Poslední částí jsou pak skripty sloužící pro složitější optimalizaci.

Vraťme se nyní k tvorbě stochastické části modelu. Jako první je nutné zadat počet scénářů a jejich pravděpodobnosti, dále pak deklarovat náhodné parametry a určit strukturu scénářového stromu. Pro dvojstupňové úlohy je speciální příkaz `twostage`, z ostatních zmiňme například `binary`, který vytvoří binární scénářo-

vý strom. Tyto speciální tvary bohužel není možné zapsat do SMPS formátu, vytvoření modelu v SLP–IOR je tedy složitější než v SPInE. SPInE si zadané informace upraví a je z nich schopen vytvořit SMPS formát.

Pro složitější optimalizaci je možné použít skripty, díky nimž lze například řešit stejnou úlohu opakovaně pro změnu jednoho parametru. Jako příklad si uveďme použití skriptu je soubor stoch.run, který je dodávaný s programem AMPL Studio, a ve kterém je pomocí cyklů jednoduše naprogramován L–shaped algoritmus.

Se skripty se váží i další dvě vlastnosti AMPL Studia. První z nich je debugger, který mi například v GAMSu velmi chybí. Je tedy možné nastavit v kódu breakpoint, na kterém se běh programu zastaví a poté prohlídnout hodnoty vybraných parametrů. Toto činí hledání chyb ve složitém kódu o hodně jednodušší. Spolu s debuggerem je přidán i kompilátor, je tedy možné zjistit, jestli je program syntakticky dobře napsaný bez toho, aby se musel spustit.

Druhou vlastností je to, že AMPL Studio je program vzniklý spojením několika modulů, které se mohou volat na sobě nezávisle. Mezi tyto moduly například patří generování scénářů, vytvoření SMPS formátu, vyřešení problému za pomoci solveru nebo analýza výsledků. Při použití skriptů je možné jednotlivé moduly volat jednotlivě. Přesně tato vlastnost byla využita i ve skriptu stoch.run.

Schopnosti programu jsou velmi podobné schopnostem SLP–IOR. Po vytvoření problému je SAMPL schopný vytvořit SMPS formát, který poté pošle solveru. Defaultní solver je FortMP, ale existuje možnost připojení libovolného solveru, který je schopen načítat standardní formát.

Kapitola 4

Finanční aplikace

V této kapitole ukážeme aplikaci předchozích kapitol na reálná data, konkrétně na model, který má v účelové funkci podmíněnou hodnotu v riziku a následně tento model vyřešíme za pomoci vybraného softwaru, který jsme popsali v kapitole 3.

V první části podmíněnou hodnotu v riziku zadefinujme, ukažme některé její vlastnosti, vztah s hodnotou v riziku a ukážeme, jak vypadá deterministický ekvivalent, pomocí kterého jde podmíněná hodnota v riziku počítat. V druhé části ukážeme jiný přístup k problému, jednostupňový problém rozepíšeme do dvojstupňového a následně ho vyřešíme pomocí L–shaped algoritmu.

Ve třetí části popíšeme model, který budeme v dalších částech optimalizovat. Zmíníme též některá specifika tohoto modelu. Ve čtvrté části budeme tento model řešit pomocí aplikace SLP–IOR. Zároveň v této části popíšeme praktické vytvoření SMPS formátu a ukážeme jeho zastaralost. V páté části se budeme věnovat algebraickému modelovacímu jazyku GAMS, v kterém budeme řešit deterministický ekvivalent a následně v něm vytvoříme algoritmus popsaný v druhé části této kapitoly. Na závěr pak GAMS propojíme s objektově orientovaným jazykem C# a ukážeme, jak lze dosáhnout urychlení algoritmu.

Šestá část bude mít trochu jinou koncepci než předchozí dvě části. Spíše než softwarem se budeme zabývat multicut verzí L–shaped algoritmu a ukážeme, proč je tento algoritmus nevhodný pro úlohy, které obsahují podmíněnou hodnotu v riziku. Zároveň zmíníme, že v extrémních případech může být L–shaped algoritmus horší než deterministický ekvivalent.

4.1 Podmíněná hodnota v riziku

Jednou z nejpoužívanějších měr rizika je v současné době podmíněná hodnota v riziku neboli CVaR. V této části CVaR zadefinujeme, ukážeme některé jeho vlastnosti a přednosti před další mírou rizika, hodnotou v riziku VaR. Nejprve ukažme definice obou měr.

Mějme funkci $f(x, \xi)$, kde ξ je náhodný vektor a $x \in X$ je vektor, který budeme v další části nazývat portfolio. Kladné hodnoty $f(x, \xi)$ reprezentují ztráty, zatímco záporné hodnoty reprezentují zisk. Označme $\Psi(x, \zeta) = \mathrm{P}(f(x, \xi) \leq \zeta)$ distribuční funkci ztrátové funkce f.

Definice 4.1. Pro pevné portfolio x definujme hodnotu v riziku na hladině α jako

$$\mathrm{VaR}_\alpha(x) = \min\{\zeta; \Psi(x, \zeta) \geq \alpha\}.$$

Hladina α se většinou volí velká, například $\alpha = 0.95$. Hodnota VaR_α tedy udává α–kvantil rozdělení $z_x(\xi) = f(x, \xi)$. Jedná se o často používanou míru rizika, která říká, že v $\alpha 100\%$ případů nebude ztráta větší než $\mathrm{VaR}_\alpha(x)$. Na druhou stranu trpí VaR_α několika nedostatky. Jak bylo ukázáno v [2], VaR_α není koherentní mírou rizika, konkrétně nesplňuje požadavek na subaditivitu, a tedy se v obecném případě nejedná o konvexní funkci proměnné x, což s sebou nese problém při optimalizaci. Dále nedává VaR_α žádnou informaci o tom, co se děje v oněch nejhorších $(1 - \alpha)100\%$ případech, jestli jsou ztráty pouze o něco málo větší než VaR_α nebo jestli jsou katastrofické.

Z tohoto důvodu byla v [16] a [17] vyvinuta podmíněná hodnota v riziku, která výše zmíněné nedostatky odstraňuje. Uveďme nejprve její definici.

Definice 4.2. Uvažujme distribuční funkci

$$\Psi_\alpha(x, \zeta) = \begin{cases} 0 & \zeta < \mathrm{VaR}_\alpha \\ \frac{\Psi(x,\zeta) - \alpha}{1 - \alpha} & \zeta \geq \mathrm{VaR}_\alpha \,. \end{cases}$$

Potom pro pevně dané portfolio x rozumíme podmíněnou hodnotou v riziku $\mathrm{CVaR}_\alpha(x)$ na hladině α střední hodnotu náhodné veličiny, která má distribuční funkcí $\Psi_\alpha(x, \zeta)$.

Obecně můžeme říci, že CVaR_α je střední hodnota ztrát v nejhorších $(1 - \alpha)100\%$ případech. Důležitost definice 4.2 se ukazuje v případě, kdy má distribuční funkce v bodě VaR_α skok nebo je na nějakém okolí bodu VaR_α konstantní. Z definice je též zřejmé, že pro libovolné portfolio platí $\mathrm{CVaR}_\alpha(x) \geq \mathrm{VaR}_\alpha(x)$, tedy spočtením hodnoty CVaR_α dostaneme i horní odhad pro VaR_α. Jak bude uvedeno v další větě, při výpočtu CVaR_α můžeme dostat jako vedlejší výsledek i hodnotu VaR_α.

Věta 4.3. *Označme*

$$F_\alpha(x, z) = z + \frac{1}{1 - \alpha} \mathrm{E}[f(x, \xi) - z]^+.$$

Potom

$$\mathrm{CVaR}_\alpha(x) = \min_z F_\alpha(x, z) \tag{4.1}$$

a $\mathrm{VaR}_\alpha(x)$ *je levý bod intervalu* $\arg\min_z F_\alpha(x, z)$.

Důkaz. Byl proveden v [17]. □

Vidíme, že pro optimalizaci modelů, ve kterých se vyskytuje podmíněná hodnota v riziku, bude mít předchozí věta klíčový význam. Uveďme nyní některé vlastnosti funkce $F_\alpha(x, z)$.

Věta 4.4. *Pokud je* $f(x, \xi)$ *konvexní v* x*, pak* $\mathrm{CVaR}_\alpha(x)$ *je konvexní v* x *a* $F_\alpha(x, z)$ *je sdruženě konvexní v obou argumentech.*

Podobně pokud je $f(x, \xi)$ *sublineární v* x*, pak* $\mathrm{CVaR}_\alpha(x)$ *je sublineární v* x *a* $F_\alpha(x, z)$ *je sdruženě sublineární v obou argumentech.*

Důkaz. Konvexita $F_\alpha(x,z)$ dokážeme přímo z definice. Položíme–li $\lambda \in [0,1]$, pak platí

$$\begin{aligned}
&F_\alpha(\lambda x_1 + (1-\lambda)x_2, \lambda z_1 + (1-\lambda)z_2) \\
&\quad = \lambda z_1 + (1-\lambda)z_2 + \frac{1}{1-\alpha}\,\mathrm{E}[f(\lambda x_1 + (1-\lambda)x_2, \xi) - \lambda z_1 - (1-\lambda)z_2]^+ \\
&\quad \le \lambda z_1 + (1-\lambda)z_2 + \frac{1}{1-\alpha}\,\mathrm{E}[\lambda f(x_1,\xi) + (1-\lambda) f(x_2,\xi) - \lambda z_1 - (1-\lambda)z_2]^+ \\
&\quad \le \lambda z_1 + (1-\lambda)z_2 + \frac{1}{1-\alpha}\lambda\,\mathrm{E}[f(x_1,\xi) - \lambda z_1]^+ \\
&\qquad + \frac{1}{1-\alpha}(1-\lambda)\,\mathrm{E}[f(x_2,\xi) - z_2]^+ \\
&\quad = \lambda F_\alpha(x_1,z_1) + (1-\lambda)F_\alpha(x_2,z_2).
\end{aligned}$$

Využili jsme toho, že funkce $[\cdot]^+$ je neklesající a subaditivní funkce. Konvexita $\mathrm{CVaR}_\alpha(x)$ je pak důsledkem věty A.4.

Sublinearita se dokáže podobným způsobem. □

Pokud bychom definovali

$$\rho(y) = \min_z \{z + \mathrm{E}[y-z]^+\},$$

pak není těžké ukázat, že ρ je koherentní míra rizika. Skutečně subaditivita a pozitivní homogenita plyne přímo z věty 4.4. Monotónnost plyne z toho, že kladná část jakožto funkce je neklesající. Konečně invariance na translaci plyne z toho, že pro konstantu a platí

$$\rho(y+a) = \min_z \{z + \mathrm{E}[y+a-z]^+\} = \min_{z'} \{z' + a + \mathrm{E}[y-z']^+\} = \rho(y) + a.$$

Jedním z nejčastějších případů optimalizace, ve kterých se vyskytuje podmíněná hodnota v riziku, je

$$\min_{x\in X} c'x + \mathrm{CVaR}_\alpha(x), \tag{4.2}$$

kde $c = \lambda r$, λ je známý parametr určující váhu členu $r'x$, r je známý parametr, X je polyedrická množina a $\mathrm{CVaR}_\alpha(x)$ přísluší lineární funkci $f(x,\xi) = q(\xi)'x$. Díky větě 4.4 víme, že se jedná o konvexní problém.

Předpokládejme dále, že náhodná veličina ξ má rozdělení, které je soustředěno na konečném počtu bodů. Pokud označíme $q_k = q(\xi_k)$, potom můžeme díky (4.1) přepsat úlohu (4.2) jako

$$\min_{x\in X, z} c'x + z + \frac{1}{1-\alpha}\sum_{k=1}^{K} p_k [q_k'x - z]^+. \tag{4.3}$$

Tuto úlohu můžeme přidáním pomocných proměnných přepsat na problém

$$\begin{aligned}
\min_{x\in X, z, w_k} \quad & c'x + z + \frac{1}{1-\alpha}\sum_{k=1}^{K} p_k w_k \\
s.t.\ \ & w_k \ge q_k'x - z, \\
& w_k \ge 0,
\end{aligned} \tag{4.4}$$

což je lineární problém, na který můžeme použít nejrůznější solvery určené pro lineární optimalizaci.

Druhou možností řešení je použití L–shaped algoritmu, což bylo poprvé zpracováno v [12]. Tomuto postupu věnujme nyní další část.

4.2 Algoritmus pro CVaR

V algoritmu nebudeme vycházet z (4.4), ale z (4.3). Tento model můžeme přepsat do dvojstupňové úlohy

$$\min_{x\in X,z} c'x + z + \frac{1}{1-\alpha}\sum_{k=1}^{K} p_k \mathcal{Q}(x,z,\xi_k), \tag{4.5}$$

kde $\mathcal{Q}(x,z,\xi_k)$ je optimální hodnota problému

$$\begin{aligned} &\min_{y} y \\ &s.t.\ y \geq q_k'x - z, \\ &\quad y \geq 0. \end{aligned} \tag{4.6}$$

K problému (4.6) sestrojme duální problém, který bude tvaru

$$\begin{aligned} &\max_{\pi} (q_k'x - z)\pi \\ &s.t.\ 0 \leq \pi \leq 1. \end{aligned} \tag{4.7}$$

Tento problém je triviální, jeho optimální řešení je podle znaménka $q_k'x - z$ buď $\bar{\pi} = 1$ nebo $\bar{\pi} = 0$. Dále má problém (4.6) úplnou kompenzaci, takže je v L–shaped algoritmu možné vynechat druhý krok, ve kterém generujeme řezy přípustnosti.

V prvním kroku se bude řešit upravená verze problému (2.2), který vzhledem k absenci řezů přípustnosti bude mít tvar

$$\begin{aligned} &\min_{x\in X,z,\theta} c'x + z + \frac{1}{1-\alpha}\theta \\ &\quad s.t.\ E_l x + F_l z + \theta \geq e_l,\ l = 1,\ldots,\nu-1. \end{aligned} \tag{4.8}$$

Dále upravme vytváření řezů přípustnosti (2.6). Předpokládejme, že jsme v iteraci ν a pro zjednodušení zápisu označme

$$\mathcal{K}^\nu = \{k; \pi_k^\nu = 1\} = \{k; q_k'x^\nu - z^\nu > 0\}, \tag{4.9}$$

kde π_k^ν má stejný význam jako v původním L–shaped algoritmu, jedná se o optimální řešení (4.7). Vidíme, že díky tomuto zápisu nemusíme v třetím kroku L–shaped algoritmu počítat žádný problém, ale stačí pouze porovnat dvě čísla a podle výsledku porovnání zařadit nebo nezařadit index do množiny $\mathcal{K}^\nu$.

Koeficienty u jednotlivých řezů můžeme spočítat pomocí

$$\begin{aligned} E_\nu &= \sum_{k=1}^{K} p_k(\pi_k^\nu)'(-q_k) = -\sum_{k\in\mathcal{K}^\nu} p_k q_k \\ F_\nu &= \sum_{k=1}^{K} p_k(\pi_k^\nu)' = \sum_{k\in\mathcal{K}^\nu} p_k \\ e_\nu &= 0. \end{aligned} \tag{4.10}$$

Řez přidaný do (4.8) bude mít tedy tvar

$$\theta + z\sum_{k\in\mathcal{K}^\nu} p_k - x'\sum_{k\in\mathcal{K}^\nu} p_k q_k \geq 0. \tag{4.11}$$

Přímým aplikováním L–shaped metody na (4.5) jsme došli ke stejnému algoritmu, ke kterému jiným postupem došli i autoři [12]. Pro jednoduchost znovu předpokládejme, že množina X je neprázdná a omezená. Tento předpoklad je ve finančních modelech často splněn, například pokud vektor x vyjadřuje váhy jednotlivých instrumentů v portfoliu. Pokud nejsou povolené prodeje na krátko, pak se požaduje $0 \leq x \leq 1$. Pro jistotu sepišme jednotlivé kroky tohoto algoritmu.

Krok 0: Položme $\nu = 0$, zvolme $\mathcal{K}^0 = \{1, \ldots, K\}$, z čehož plyne přidání první podmínky $\theta + z - (\sum_{k=1}^{K} p_k q_k)' x \geq 0$.

Krok 1: Zvětšeme ν o jedna, vyřešme problém (4.8), jehož optimální řešení označme $(x^\nu, z^\nu, \theta^\nu)$. Díky předpokladům na množinu X máme zaručenu existenci optimálního řešení.

Krok 2: Podle (4.9) spočtěme množinu $\mathcal{K}^\nu$. Spočtěme dále

$$\omega^\nu = \sum_{k \in \mathcal{K}^\nu} p_k (q_k' x^\nu - z^\nu).$$

Pokud $\omega^\nu \leq \theta^\nu$, algoritmus ukončeme, (x^ν, z^ν) je optimální řešení (4.3). Pokud předchozí nerovnost neplatí, přidejme řez optimality (4.11) a vraťme se do prvního kroku.

Pravidla pro konvergenci jsou stejná jako u klasického L–shaped algoritmu. Pokud algoritmus skončí v bodě dva, dostaneme výsledek $(x^\nu, z^\nu, \theta^\nu)$. Zopakujme, co jednotlivé proměnné znamenají: x^ν je skladba optimálního portfolia, z^ν je podle věty 4.3 VaR_α a $z^\nu + \frac{1}{1-\alpha}\theta^\nu$ je hodnota CVaR_α.

V kroku nula jsme jako počáteční $\mathcal{K}^0$ zvolili celou množinu $\{1, \ldots, K\}$. Samozřejmě je možné volit i jiné množiny, například $\mathcal{K}^0 = \emptyset$, což má za následek podmínku $\theta \geq 0$. Pokud řešíme stejnou úlohu opakovaně, ale v každé úloze změníme nějaký parametr, například hladinu α, je možné použít techniku žhavého startu, kdy jako počáteční $\mathcal{K}^0$ zvolíme optimální $\mathcal{K}^\nu$ z poslední optimalizace.

Spolu s článkem [12] byl vyvinut i solver CVaRMin, který je schopný řešit pouze jednostupňové problémy, ve kterých se CVaR nachází v účelové funkci nebo v podmínkách. Nicméně na těchto problémech je CVaRMin minimálně desetkrát rychlejší než obecné solvery nebo solvery specializované na stochastickou dekompozici.

Uveďme k tomuto algoritmu ještě jednu praktickou poznámku. Ve výše zmíněném algoritmu nejdříve spočítáme množinu $\mathcal{K}^\nu$ a teprve po jejím spočtení počítáme jednotlivé E_ν, F_ν, e_ν a kontrolujeme, jestli je splněno ukončovací kritérium. Toto by však bylo zbytečně pomalé. Lepší je provádět částečné součty a pouze kontrolovat, jestli index patří do množiny $\mathcal{K}^\nu$ a pokud ano, tak k částečnému součtu přidáme další díl. Tímto způsobem stačí množinu $\{1, \ldots, K\}$ projít pouze jednou.

4.3 Použitý model

V této části popíšeme model, který jsme optimalizovali a nastíníme možnosti, jak se tento model dá optimalizovat. Tyto možnosti poté podrobněji rozebereme v dalších částech.

Základ tohoto modelu je model zpracovaný v seminární práci [1], který nyní v krátkosti popíšeme. V tomto modelu jsme ze zadaného portfolia vybírali subportfolio, které by mělo být co nejvíce imunní vůči krizi. Krizové scénáře jsme

měli zadány pomocí $N = 7$ úrokových měr a směnných kurzů, pro které jsme pro jednoduchost předpokládali, že jsou na sobě nezávislé. Označme jednotlivé úrokové míry a směnné kurzy jako náhodné veličiny a všechny náhodné veličiny jako náhodný vektor.

Portfolio bylo tvořeno celkem $n = 180$ instrumenty. Mezi tyto instrumenty patřila depozita, úvěry a swapy, přičemž jednotlivé instrumenty byly charakterizovány pomocí nominálu, maturity, měny a v případě úvěrů i ratingu. Jako měnu jsme uvažovali pouze koruny, eura a dolary. Pro každou úrokovou míru byl zadán určitý počet výnosových křivek a pro oba směnné kurzy byl zadán určitý počet možných směnných kurzů.

V modelu bylo pro korunovou, eurovou i dolarovou tržní úrokovou míru zadáno 6 výnosových křivek, stejný počet scénářů byl zadán i pro směnné kurzy mezi korunou a eurem a korunou a dolarem a nakonec bylo zadáno 5 scénářů pro kreditní riziko a 8 scénářů pro cenu likvidity. Za předpokladu nezávislosti se tedy jednalo celkem o 311040 scénářů. S takovýmto velkým počtem jsme si bohužel nebyli schopni poradit a předpokládali jsme tedy, že každá náhodná veličina může nabývat pouze 5 scénářů, čímž jsme dostali celkový počet 78125 scénářů.

Uvažovali jsme dvě hlavní kritéria, podle kterých jsme se rozhodovali, jestli instrument zařadíme do optimálního subportfolia. Prvním z nich byla profitabilita, která je definována jako rozdíl mezi budoucí hodnotou za prvního, v současnosti známého, scénáře a budoucí hodnotou za náhodného scénáře. Druhým kritériem pak byla současná hodnota.

V modelu jsme uvažovali dva typy podmínek. Prvním typ byl určen podmínkou na váhy jednotlivých instrumentů $Ax = b$, mezi podmínky jsme zařadili podmínku na minimální součet jednotlivých nominálů jednotlivých typů instrumentů v portfoliu. Dále jsme v počátečních obdobích kontrolovali, abychom měli více peněz než musíme vydat. Dále jsme uvažovali dolní a horní meze pro jednotlivé proměnné. Matice A měla rozměry 6×184. Druhým typem podmínky bylo omezení na maximální hodnotu CVaR na předem stanovené hladině α.

Pro tuto práci jsem výchozí model mírně předělal, v dalších odstavcích se budeme zabývat výhradně upravenou verzi. Do upraveného modelu jsem pro jednoduchost zařadil pouze profitabilitu, neuvažoval jsem současnou hodnotu. Zároveň jsem neuvažoval druhý typ podmínek, místo omezení na CVaR jsem CVaR přidal do účelové funkce, ve které byla lineární kombinace střední hodnoty profitability a CVaRu příslušného profitabilitě. Tímto jsem dosáhl toho, že optimalizovaný model byl tvaru (4.3).

Zmiňme nyní některé vlastnosti upraveného modelu. Dosud nezmíněná klíčová vlastnost je to, že profitabilita instrumentu závisela maximálně na čtyřech náhodných veličinách. Tato vlastnost dělala problémy při používání SLP–IORu.

Dále vidíme, že matice A je relativně malá. Problém je ale v tom, že účelová funkce není díky CVaRu lineární, nýbrž polyedrická. Jednou z možností řešení je zavedení pomocných proměnných a přepsání modelu na model (4.4). V tomto případě by bylo nutné zavést K pomocných proměnných a K pomocných podmínek. Při velkém K pak úloha nebude prakticky řešitelná. Tento postup jsme využili v původním modelu, který byl naprogramovaný v algebraickém jazyku GAMS.

Pro tuto práci využijeme možnost přepsání modelu do dvojstupňového modelu (4.5)–(4.6), který následně budeme řešit několika způsoby. Prvním způsobem

bude vytvoření SMPS formátu a jeho následné nahrání do SLP–IORu, druhým způsobem pak bude využití algoritmu popsaného v části 4.2, který se naprogramuje v GAMSu. Bohužel hledání množiny $\mathcal{K}^\nu$ se v GAMSu ukázalo jako zbytečně zdlouhavé, takže model byl znovu naprogramován v jazyku C#, ve kterém se prováděly veškeré výpočty a GAMS se volal pouze na řešení problému (4.8). Pomocí C# se vytvořil i výše zmíněný SMPS formát.

Před podrobnějším popisem těchto způsobů okomentujme velikost počtu iterací K. Při porovnání jednotlivých solverů nebo softwaru se většinou za základ bere malý problém, což je na jednu stranu u starších článků je pochopitelné, na druhou stranu myslím, že v současnosti je nutné vzít jako základ větší problémy. Důvod k tomuto je jednoduchý, pro malé problémy je možné vytvořit deterministický ekvivalent, který se následně vyřeší. Tento postup bude sice pomalejší, ale pro malé problémy bude tento časový rozdíl zanedbatelný. Rozdíl by se mohl stát významnějším, pokud bychom řešili za sebou větší množství malých úloh, což ale například SLP–IOR neumožňuje.

V práci postupně volíme K jako $2^7, 3^7, 4^7, 5^7$ a $6^5 * 5 * 8$, kde poslední počet přibližně odpovídá 6^7. Na základě výsledků pro tyto počty scénářů se dále snažíme odhadnout, jak se algoritmus bude chovat pro ještě větší problémy. Věnujme se nyní jednotlivým typům softwaru.

4.4 SLP–IOR

Jak už bylo zmíněno, SLP–IOR je klikací aplikace, a tedy v ní není možné provádět automatizované výpočty, ale je potřeba manuální zásah uživatele. Uveďme příklad, kdy chceme několikrát vyřešit stejnou úlohu, ve které vždy změníme jeden parametr. V GAMSu stačí udělat jeden cyklus, přičemž program si sám nahraje data, problém vyřeší a výsledek poté uloží. V SLP–IORu je nutné pro každou úlohu provést každou operaci manuálně.

Jako další problém klikací aplikace vidím nemožnost cokoli v aplikaci spočítat, ale nutnost provést všechny výpočty v jiné aplikaci. V modelu jsme pro každý instrument počítali profitabilitu, která pro každý typ instrumentu závisela na dvou až čtyřech náhodných veličinách. Stačilo si tedy pro každý instrument vytvořit čtyřrozměrné pole, ve kterém jsme měli uloženou profitabilitu. Toto ale není možné udělat v SMPS formátu, do kterého pro každý instrument musíme uložit K možných realizací náhodného vektoru. To při $n = 180$ instrumentech a $K = 311040$ realizacích náhodného vektoru odpovídá přibližně 55 miliónům záznamů, z nichž každý se opakuje minimálně stokrát.

Zabývejme se nejprve generováním samotného SMPS formátu, který můžeme najít v tabulce 4.1. V prvním sloupci uvádíme počet scénářů, v druhém sloupci pak dobu, za kterou se vygeneroval SMPS formát. Vidíme, že u největšího problému trvalo pouhé vygenerování více než 4 minuty. Další dva sloupce uvádějí velikosti core a stoch souboru v MB. Velikost core souboru je pořád stejná, protože v tomto souboru popisujeme pouze výchozí deterministický problém, který nezávisí na počtu realizací náhodného vektoru. Velikost time souboru jsem nevypisoval, protože je pouze 1 kB. Vidíme, že velikost core a time souborů jsou zanedbatelné ve srovnání se stoch souborem, který při největším počtu realizací je větší než, pro textový soubor astronomických, 2 GB.

		Generování 1		Generování 2	
K	Čas [s]	Core [MB]	Stoch[MB]	Core [MB]	Stoch[MB]
128	0.101	0.031	0.763	0.020	0.511
2187	1.543	0.031	14.821	0.020	9.926
16384	12.368	0.031	114.874	0.020	76.919
78125	56.759	0.031	553.945	0.020	370.894
311040	4:06.130	0.031	2214.273	0.020	1482.518

Tabulka 4.1: Doba generování a velikost SMPS formátu

K	128	2187	16384	78125	311040
Čas [s]	2	2	55	4:30	?

Tabulka 4.2: Doba načítání SMPS formátu

V posledních dvou sloupcích jsem vypsal jednoduchou modifikaci SMPS formátu, ve kterém jsem ale jednotlivé záznamy neukládal na předem určené pozice v řádku, ale odděloval jsem tyto záznamy jedním středníkem, což odpovídá .csv souboru, ve kterém se jednotlivé vstupy oddělují čárkou nebo středníkem. Vidíme, že velikost stoch souboru klesla přibližně o třetinu.

Bohužel SLP–IOR je v plné verzi dostupný pouze v 32 bitové verzi a na 64 bitovém operačním systému se mi ho nepodařilo spustit ani v režimu kompatibility. Toto mělo za následek, že jsem veškeré testování tohoto softwaru musel provést na jiném, slabším, počítači. Časy v této části nebudou tak důležité, ale pro úplnost uveďme, že optimalizace stejné úlohy v GAMSu trvala na tomto pomalejším počítači zhruba o čtvrtinu déle.

V SLP–IORu jsem provedl podobné experimenty jako v článku [12], v kterém se pro problémy, které měly v účelové funkci podmíněnou hodnotu v riziku, měřilo, jak dlouho je budou jednotlivé solvery optimalizovat. Bohužel tyto výsledky nemůžu potvrdit.

První problém, s kterým jsem se setkal, bylo načtení SMPS formátu. Časy, které byly potřebné k tomuto načtení, uvádím v tabulce 4.2. Jak uvidíme dále, nastane paradoxní situace, kdy doba potřebná pro SLP–IOR k pouhému načtení problému byla delší než doba, za kterou jiný software problém vygeneroval a vyřešil. Otazník u nejvyššího počtu scénářů je uveden proto, že načítání tohoto problému trvalo proměnlivou délku, v některých případech se problém nepodařilo načíst vůbec a SLP–IOR zamrzl. Odhadovaná doba načítaní tohoto problému je kolem 20 minut.

Zaměřme nyní naší pozornost na jednotlivé solvery. Asi nejpřekvapivější bylo zjištění, že solver využívající kvadratickou dekompozici QDECOM nebyl schopný vyřešit ani jeden problém a u všech problémů zjistil údajnou neexistenci přípustného řešení. Pro ostatní solvery je pro menší K obtížné určit přesný čas samotného optimalizování, neboť to se skládá z několika samostatných částí, které končí zavoláním solveru. Jediným údajem, který je ale k dispozici je doba běhu solveru. Zároveň se doba běhu výrazně lišila v průběhu jednotlivých pokusů.

Věnujme se nyní případu, kdy $K = 16384$, což je relativně malý problém, s kterým GAMS neměl problém. SLP–IOR má k dispozici šest solverů, které by tento problém měly být schopny vyřešit. Jedná se o BPMPD, CPlex, MSLiP, OSL, QDECOM a SAA. Solverem CPlex jsem se nezabýval, protože se při jeho

použití vygeneruje deterministický ekvivalent, který se následně vyřeší. Tento přístup zkoumám v další části.

Jak už bylo řečeno QDECOM nevyřešil ani jeden problém a SAA pro defaultní volbu nastavitelných parametrů spočítal téměř správný výsledek, který se od optimálního řešení lišil pouze pro několik instrumentů. SAA je zkratkou pro Sample average approximation, tento solver vybírá z K scénářů předem daný počet scénářů, na jehož základě pak spočte horní a dolní mez pro optimální hodnotu účelové funkce. Podrobnější informace o tomto solveru je možné najít v [8]. Výsledek by šel zlepšit vhodnou volbou parametrů. MSLiP kvůli špatně nastaveným vstupním parametrům ani optimalizaci nespustil, bohužel se mi nepovedlo najít, který parametr je třeba zvětšit, aby mohla být optimalizace nastartována.

Zbývají poslední dva solvery. BPMPD byl relativně rychlý, optimalizace mu trvala odhadem 2 minuty, ale výsledky nebyly úplně přesné, chyba byla však zanedbatelná, nejčastěji se jednalo o výpis 10^{-9} místo 0. Tento solver dopadl dle mého názoru jednoznačně nejlépe. Se solverem OSL nemám bohužel nejlepší zkušenosti. I tento malý problém optimalizoval kolem 12 minut, při $K = 78125$ jsem ho nechal běžet 50 minut a pak jsem ho musel terminovat díky nedostatku virtuální paměti a nedostatku místa na disku.

Speciálně pro tento typ problémů byl vyvinut solver CVaRMin. Bohužel SMPS formát nepodporuje úlohy s podmíněnou hodnotou v riziku a při nahrání dvojstupňového problému se mi nepodařilo tento problém přetransformovat do formátu, který CVaRMin potřebuje, což mělo za následek, že jsem ho nemohl vyzkoušet.

4.5 GAMS

V této části budeme využívat jako optimalizační software výhradně GAMS, pro který budeme porovnávat několik algoritmů. Jako první uvažujme nejjednodušší možnost zápisu (4.4). Pro velké K nebyl GAMS schopný tento problém vyřešit, jako další jsme tedy v GAMSu naprogramovali L–shaped algoritmus. I toto bylo jednoduché, v GAMSu stačil přidat cyklus, ve kterém se počítá množina $\mathcal{K}^\nu$ a přidávají se jednotlivé řezy. Protože se tento způsob neukázal jako příliš efektivní, naprogramovali jsme stejný algoritmus i v programovacím jazyce C#, ze kterého se v každé iteraci volal GAMS. V C# se prováděly veškeré výpočty a GAMS se volal pouze pro optimalizaci. Pro všechny tři metody jsme zaznamenávali celkový čas, který byl nutný k vyřešení problému a celkový počet iterací.

Nejprve jsme pro každou náhodnou veličin zvolili pouhé dvě realizace a toto číslo jsme postupně zvyšovali až na konečných 311040 scénářů. Tento problém byl už poměrně složitý, protože bylo nutné v každé iteraci procházet pole o velikosti daného počtu scénářů a pro každý prvek v tomto poli se musela pro každý instrument provést série operací. Zároveň jsme optimalizaci provedli pro šest hodnot parametru h, který uvádí poměrovou část celkové nominální hodnoty portfolia, kterou je nutné ve výsledném subportfoliu zanechat. Výsledky uvádíme v tabulce 4.3.

První dva sloupce udávají parametr vynechání h a počet scénářů K. V dalších šesti sloupcích je střídavě doba běhu programu a počet iterací, které program musel vykonat. Dvojice sloupců postupně odpovídá řešení deterministického algoritmu, použití speciálního algoritmu pro CVaR a použití identického algoritmu, ale ve spojení s C#.

h	K	CPLEX		GAMS		C#	
0	128	0.407	7	1.727	10	2.852	10
0.1	128	0.356	11	2.291	14	4.319	14
0.2	128	0.349	11	2.205	14	4.344	14
0.3	128	0.355	11	2.152	14	4.189	14
0.4	128	0.414	11	2.186	14	4.131	14
0.5	128	0.372	11	2.014	13	3.950	13
0	2187	1.540	358	18.138	45	18.571	45
0.1	2187	1.600	373	14.608	37	15.472	37
0.2	2187	1.742	435	19.338	49	20.391	49
0.3	2187	1.640	470	19.593	48	19.820	48
0.4	2187	1.841	464	15.866	42	17.440	42
0.5	2187	1.850	473	16.384	42	17.369	42
0	16384	17.667	2663	1:14.060	42	38.611	42
0.1	16384	26.101	3494	1:50.894	64	56.817	64
0.2	16384	19.666	2984	2:03.248	67	1:01.387	67
0.3	16384	30.704	3546	1:45.881	58	52.483	58
0.4	16384	28.848	3390	1:06.133	36	31.965	36
0.5	16384	35.904	3836	1:24.989	47	43.168	47
0	78125	4:32.370	13948	6:25.737	52	2:34.848	52
0.1	78125	5:41.063	15382	11:42.975	93	4:39.386	93
0.2	78125	5:30.011	14932	6:25.400	53	3:01.012	53
0.3	78125	8:12.615	17788	6:59.819	55	2:44.145	55
0.4	78125	5:33.441	16218	7:14.018	60	3:01.343	60
0.5	78125	8:24.959	18247	6:50.092	58	2:57.148	58
0	311040	x	x	57:10.192	111	22:42.622	111
0.1	311040	x	x	54:36.342	111	23:06.111	111
0.2	311040	x	x	32:24.348	67	13:36.060	67
0.3	311040	x	x	31:01.003	62	13:34.051	62
0.4	311040	x	x	31:33.420	63	12:41.594	63
0.5	311040	x	x	37:07.353	64	13:09.851	64

Tabulka 4.3: Porovnání softwaru

Délka výpočtů je pouze orientační, v sekundách se měří reálná doba, kterou program běžel. Dvojtečkou jsou odděleny minuty od sekund, x znamená, že algoritmus nebyl úlohu schopný spočítat. Zvláště u menšího počtu scénářů pak tato doba mohla být zkreslena jinými procesy, které zatěžovaly procesor. Počet iterací v prvním případě udává počet iterací, který musel CPLEX vykonat při řešení deterministického ekvivalentu. Počet iterací v dalších dvou případech je identický, protože se jedná o stejný algoritmus. V tomto případě se jedná o počet cyklů, které algoritmus provedl, přičemž v každém cyklu se volala optimalizační úloha.

Je důležité si uvědomit, že velikost této úlohy nezáleží na počtu scénářů K, ale pouze na počtu cyklů. Jinými slovy, pokud by pro $K = 128$ trvalo algoritmu nalezení optimálního řešení stejný počet cyklů jako pro $K = 311040$, pak by GAMS v obou případech běžel přibližně stejnou dobu. Rozdíl by však byl v C#, kdy by se v druhém případě prohledávalo výrazně větší pole. Díky této vlastnosti je pro porovnání vlastností algoritmů důležitý i počet cyklů.

Vidíme, že pro menší problémy je nejrychlejší vyřešení deterministického ekvivalentu pomocí CPLEXu. U větších problémů ale začíná být CPLEX pomalý a největší problémy nebyl schopný vyřešit vůbec a běh optimalizace skončil chybou o nedostatku paměti. Toto je způsobeno tím, že při optimalizovaní se invertují matice, jejichž rozměr odpovídá, po případném presolve, počtu podmínek. Zatímco algoritmus v druhém a třetím případě prohledává stále větší a větší pole, v prvním případě je nutné invertovat stále větší a větší matice.

Pokud bychom porovnávali algoritmus z části 4.2, který byl jednou napsaný pouze v GAMSu a jednou v C# společně s GAMSem, je výhoda jednoznačně na straně kombinace obou programů. Protože teoretický maximální počet vygenerovaných řezů je 2^K, je zajímavé, že počet hlavních iterací závisí na logaritmu počtu scénářů přibližně lineárně.

Jako další jsme naprogramovali techniku žhavého startu, při které se využívá výsledků minulé optimalizace. Například v [12] se při změně parametru úlohy nezačínalo od $K^0 = \{1, \ldots, K\}$, ale položilo se $K^0 = K^\nu$, kde K^ν byla množina z minulé optimalizace, která určovala optimální řešení. V tomto případě je tato metoda též atraktivní, protože změna parametru h ovlivňuje pouze tvar množiny přípustných řešení X, ale nikoli účelové funkce. Navíc ji ovlivňuje takovým způsobem, že pro libovolné $h_1 \leq h_2$ platí $X(h_1) \supset X(h_2)$, kde $X(h)$ je množina přípustných řešení za parametru h. To znamená, že veškeré řezy zůstávají v dolním odhadem účelové funkce. Na druhou stranu tyto řezy už nemusejí být opěrnými funkcemi.

Výsledky metody popsané v minulém odstavci můžeme vidět v tabulce 4.4. První dva sloupce znovu určují velikost hranice vynechaní h a počet scénářů K. V dalších dvou sloupcích jsou zapsány časy a počet iterací, pokud bychom použili techniku žhavého startu a h bychom volili nejdříve jako 0 a postupně ho zvyšovali. V posledních dvou sloupcích jsme jako výchozí h zvolili 0.5 a postupně ho snižovali. Poslední dva sloupce se tedy pro jednotlivá K musí číst jakoby odspodu.

Pokud h nejdříve snižujeme, zvětšuje se množina přípustných řešení, a tedy všechny řezy zůstávají nejenom dolním odhadem, ale i opěrnými funkcemi. Očekával jsem, že tento postup bude mít lepší výsledky než zvyšování h, ale předpoklad se neukázal jako platný, oba postupy dávají přibližně stejné výsledky.

Můžeme porovnat některé políčka tabulek 4.4 a 4.3. Například políčko pro

h	K	Rostoucí h		Klesající h	
0	128	2.902	10	2.137	7
0.1	128	2.929	10	0.349	2
0.2	128	0.334	2	0.986	4
0.3	128	0.657	3	0.329	2
0.4	128	0.339	2	0.973	4
0.5	128	1.003	4	3.960	13
0	2187	19.117	45	19.595	25
0.1	2187	11.890	26	14.865	29
0.2	2187	16.125	34	16.613	33
0.3	2187	14.341	28	11.347	24
0.4	2187	13.491	26	11.393	26
0.5	2187	12.693	23	17.385	42
0	16384	36.707	42	41.300	38
0.1	16384	55.032	54	57.879	52
0.2	16384	54.135	55	55.718	54
0.3	16384	48.682	47	43.684	44
0.4	16384	24.709	23	18.479	20
0.5	16384	34.178	31	41.337	47
0	78125	2:29.557	52	2:46.862	48
0.1	78125	4:44.238	94	4:31.819	81
0.2	78125	2:33.667	45	2:43.337	53
0.3	78125	2:14.903	42	2:15.004	43
0.4	78125	2:32.247	45	1:52.681	37
0.5	78125	2:26.726	43	2:55.147	58
0	311040	25:26.102	111	8:20.150	38
0.1	311040	8:43.769	38	21:25.654	98
0.2	311040	12:28.154	55	11:35.112	54
0.3	311040	12:44.721	55	10:32.124	50
0.4	311040	11:34.290	50	10:25.204	50
0.5	311040	11:44.770	50	12:58.727	64

Tabulka 4.4: Žhavý start

K	L–shaped	Žhavý start 1	Žhavý start 2
128	79	31	32
2187	263	182	179
16384	314	252	255
78125	371	321	320
311040	478	359	354

Tabulka 4.5: Součet počtu iterací přes jednotlivá h

K	CPLEX	GAMS	C#	Žhavý start 1	Žhavý start 2
128	2.253	12.575	23.785	8.164	8.734
2187	10.213	103.927	109.063	87.657	91.198
16384	158.890	565.205	284.431	253.443	258.397
78125	2274.459	2738.041	1137.882	1021.338	1024.850
311040	x	14632.658	5930.289	4961.806	4516.971

Tabulka 4.6: Součet časů přes jednotlivá h

$h = 0$ ve třetím a čtvrtém sloupci v tabulce 4.4 odpovídá políčku v tabulce 4.3 pro stejné h a K a pro sedmý a osmý řádek. To je kvůli tomu, že v žhavém startu toto byla první optimalizace a tedy se vlastně o žhavý start nejednalo. Skutečně počet iterací je stejný, ale čas výpočtu se liší. Jak už bylo řečeno, toto je způsobeno počítačem, který v průběhu optimalizace pravděpodobně vykonával jiné procesy. Z tohoto důvodu je spíše než čas důležitější počet iterací, které musely být vykonány.

Naštěstí je počet iterací relativně malý. Pokud by tomu tak nebylo, nebo pokud bychom metodou žhavého startu řešili větší počet úloh, bylo by nutné nějakým způsobem vyškrtávat řezy, aby problém nenarostl do velkých rozměrů. Toto ale naštěstí v práci nebylo nutné.

Celkový počet nutných iterací pro pevné K a všechna h jsou uvedeny v tabulce 4.5, v tabulce 4.6 pak uvádíme celkový čas, tentokrát v sekundách. Vidíme, že propojení GAMSu s C# dává lepší výsledky, stejně tak technika žhavého startu, kdy čas potřebný na výpočet skupiny největších problémů je oproti algoritmu v GAMSu třetinový.

4.6 Multicut verze

V části 4.2 jsme na problém (4.3) aplikovali základní variantu L–shaped algoritmu. Ukažme nyní aplikaci multicut verze L–shaped algoritmu na stejný problém. Pomocí (2.10) můžeme problém řešený v první fázi algoritmu zapsat jako

$$\begin{aligned} \min_{x \in X, z} \; & c'x + z + \frac{1}{1-\alpha} \sum_{k=1}^{K} w_k \\ s.t. \; & D_l x + C_l z \geq d_l, \\ & E_l x + F_l z + w_k \geq e_l. \end{aligned} \tag{4.12}$$

Je zřejmé, že řezy přípustnosti nejsou potřeba, věnujme se tedy řezům opti-

mality. Úpravami dostaneme, že řez přípustnosti může být pouze tvaru

$$w_k \geq p_k \bar{\pi}_k (q_k' x - z),$$

kde $\bar{\pi}_k$ je nějaké řešení (4.7). Víme, že toto řešení může být pouze $\bar{\pi}_k = 0$ nebo $\bar{\pi}_k = 1$. Pomineme-li možný řez $w_k \geq 0$, pro každé k se může sestrojit maximálně jeden řez, který bude tvaru

$$w_k \geq p_k (q_k' x - z).$$

Úlohu (4.12) můžeme tedy přepsat na úlohu

$$\begin{aligned} \min_{x \in X, z} \; & c'x + z + \frac{1}{1-\alpha} \sum_{k \in \mathcal{K}^\nu} w_k \\ s.t. \; & w_k \geq p_k (q_k' x - z), \; k \in \mathcal{K}^\nu, \\ & w_k \geq 0, \; k \in \mathcal{K}^\nu, \end{aligned} \tag{4.13}$$

kde $\mathcal{K}^\nu$ je nějaká podmnožina množiny $\{1, \ldots, K\}$.

Otázkou zůstává jak zvolit počáteční $\mathcal{K}^1$. Pokud zvolíme $\mathcal{K}^1 = \{1, \ldots, K\}$, pak je problém (4.13) identický s modelem deterministického ekvivalentu (4.4). To znamená, že při špatné volbě některé z množin $\mathcal{K}^\nu$, nepřináší aplikování multicut verze L–shaped algoritmu žádnou výhodu oproti deterministickému ekvivalentu.

Další možností je zvolit $\mathcal{K}^1 = \emptyset$, což je standardní postup v L–shaped algoritmu, kdy na počátku nejsou známy žádné řezy. Potom problém (4.13) můžeme přepsat jako

$$\min_{x \in X, z} \; c'x + z. \tag{4.14}$$

Tento problém není díky proměnné z zdola omezen. Je možné použít variantu L–shaped algoritmu pro neomezenou množinu K_1, kterou upravíme pro multicut verzi. V práci jsem zvolil jiné řešení, které není zcela korektní, na druhou stranu ale kromě největšího problému vykazuje lepší výsledky než jakákoli metoda z předchozích částí.

Je zřejmé, že množinu $\mathcal{K}^1$ můžeme zvolit jako libovolnou podmnožinu množiny $\{1, \ldots, K\}$. V první iteraci budeme v GAMSu řešit problém (4.14). GAMS samozřejmě zjistí, že tento problém není zdola omezen, ale přesto přiřadí proměnným x a z nějakou optimální hodnotu. Proměnná x^1 bude ležet mezi nulou a jedničkou a $z^1 = 0$. Výpočet této úlohy trvá zanedbatelný čas. Položíme dále

$$\mathcal{K}^2 = \{k; \; q_k' x^1 - z^1 \geq 0\} \tag{4.15}$$

a pokračujeme druhou iterací.

Číselné výsledky můžeme vidět v tabulce 4.7. Největší problém nebyl zahrnut, protože ho GAMS nebyl schopný vyřešit, zajímavé je, že tentokráte běh skončil díky maximální povolené době běhu programu, zatímco u deterministického ekvivalentu běh skončil díky nedostatku paměti.

Kromě tří sloupců, které mají identický význam jako v předchozích tabulkách jsem do tabulky zařadil i mohutnost množiny $\mathcal{K}^2$, která je vypočítaná pomocí (4.15). Pokud je v tomto řádku pouze jedno číslo, algoritmus skončil druhou iterací, pokud jsou v něm dvě čísla, algoritmus provedl iterace tři. Více iterací nebylo v žádném případě potřeba.

h	K	Čas [s]	$\lvert\mathcal{K}^2\rvert$	h	K	Čas [t]	$\lvert\mathcal{K}^2\rvert$
0	128	0.441	64	0	16384	10.184	4236/4780
0.1	128	0.407	88	0.1	16384	11.594	4499/4503
0.2	128	0.367	96	0.2	16384	12.677	4800/4805
0.3	128	0.370	98	0.3	16384	7.519	5147
0.4	128	0.373	100	0.4	16384	9.994	5462
0.5	128	0.361	128	0.5	16384	10.611	5595
0	2187	1.884	810/856	0	78125	2:29.663	24235/25847
0.1	2187	1.084	819	0.1	78125	2:26.571	24818/25000
0.2	2187	1.242	833	0.2	78125	3:00.984	25443/25446
0.3	2187	1.258	832	0.3	78125	1:49.108	26075
0.4	2187	1.307	843	0.4	78125	1:55.736	26800
0.5	2187	1.246	838	0.5	78125	2:40.379	26922

Tabulka 4.7: Multicut verze

Vidíme, že v takto modifikovaném problému se oproti deterministickému ekvivalentu nacházela přibližně třetina podmínek, takže upravený algoritmus dosáhl výrazně lepších výsledků. Protože se v účelové funkci nalézal CVaR na hladině 0.95, nejnižší teoretická mohutnost množiny $\mathcal{K}^2$ je $\frac{1}{20}K$. Tady vidíme problém multicut verze L–shaped algoritmu při použití pro CVaR. Multicut verze sníží počet nutných řezů pouze na poměrnou část, pro velké problémy nebude možné multicut verzi aplikovat ani pro nejlepší volbu $\mathcal{K}^1$.

Uvažujme nyní modifikaci předchozí úlohy, do které přidáme podmínku $\alpha \leq z \leq \beta$, kde α je dostatečně malé a β dostatečně velké číslo. Tím dosáhneme toho, že množina K_1 je omezená a je tedy možné použít přímo multicut verzi L–shaped algoritmu. Zároveň při vhodné volbě parametrů α a β nemá tato podmínka na optimální řešení vliv.

Aplikujme nyní na tento problém L–shaped algoritmus. V první iteraci vyřešíme problém

$$\min_{x \in X, z} c'x + z$$
$$s.t.\ \alpha \leq z \leq \beta,$$

jehož optimální řešení bude dvojice (x^1, z^1), přičemž bude platit $0 \leq x^1 \leq 1$ a $z^1 = \alpha$. Vzhledem k tomu, že množinu $\mathcal{K}^2$ počítáme vztahu $\mathcal{K}^2 = \{k;\ q'_k x^1 - z^1 \geq 0\}$, bude při dostatečně malém parametru α platit $\mathcal{K}^2 = \{1, \ldots, K\}$. Tím jsme ukázali, že v případě přidání podmínky $\alpha \leq z \leq \beta$ se multicut verze L–shaped algoritmu převede na deterministický ekvivalent.

Tímto příkladem jsme ukázali doposud nezmíněný předpoklad L–shaped algoritmu, že funkce ϕ je dostatečně pěkná. Pokud tomu tak není, může se stát, že L–shaped algoritmus bude muset spočítat velké množství řezů. Není těžké zkonstruovat funkci ϕ takovou, aby při použití L–shaped algoritmu bylo nutno spočítat všech K řezů. V tomto případě je L–shaped algoritmus dokonce výrazně horší než deterministický ekvivalent.

Jako poslední uveďme poznámku k modelu, který jsme řešili v této kapitole. Při použití L–shaped algoritmu nebylo kromě dvou případů nutno spočítat více než 100 řezů. Na druhou stranu teoretický maximální počet řezů je 2^K, což i pro malé K je obrovské číslo.

Závěr

V práci jsou shrnuty základní poznatky teorie lineárního a polyedrického dvojstupňového programovaní. Dále uvádíme výběr algoritmů, pomocí kterých je možné tyto úlohy řešit. Tyto algoritmy jsou založeny na L–shaped metodě, kterou uvádíme v základní verzi, ukazujeme několik variant a uvádíme rozšíření do polyedrického, kvadratického a konvexního případu. Dále zmiňujeme vybraný software, pomocí kterého lze tyto úlohy řešit.

V poslední kapitole se zabýváme aplikací dvojstupňového programování na model, který má v účelové funkci podmíněnou hodnotu v riziku. Na tento model se pak aplikují vybrané algoritmy ve vybraném softwaru. Konkrétně se jedná o vypočtení deterministického ekvivalentu a použití základní a multicut verze L–shaped algoritmu, ze softwaru pak byl použit GAMS, SLP–IOR a propojení GAMSu se C#. Zároveň jsme zkoumali techniku žhavého startu, při které se při větším počtu optimalizací využívají výsledky předchozí optimalizace.

Stejný postup lze použít i pro minimalizaci jiných polyedrických měr rizika, mezi které patří například střední absolutní odchylka MAD. Pro tyto postupy lze též očekávat podobné výsledky. Aplikování dekompozičních metod je v tomto případě výhodné díky velkému rozsahu řešených problémů, k čemuž značnou mírou přispívá polyedrická účelová funkce.

Při práci se SLP–IOR jsem se potkal s několika problémy, osobně si myslím, že tento software je navržen spíše pro menší problémy. Na druhou stranu se CPLEX, defaultní solver GAMSu pro lineární programování, ukázal jako velice výkonný solver, který byl schopen vyřešit deterministický ekvivalent i pro problémy, se kterými si SLP–IOR nedokázal v rozumné době poradit.

Po naprogramování L–shaped algoritmu v GAMSu se ukázalo, že GAMS není vhodný jazyk pro prohledávání velkých polí. Z tohoto důvodu jsem naprogramoval stejný model v C#, ze kterého jsem pro optimalizaci volal GAMS. Tento přístup se pro větší problémy ukázal býti výrazně lepším. Technika žhavého startu se též ukázala jako značné vylepšení, kdy se u největšího problému podařilo snížit počet nutných iterací o čtvrtinu.

Jako poslední jsme testovali multicut verzi L–shaped algoritmu, z které vychází například regularizovaná dekompozice. Tento algoritmus se díky velkému počtu řezů ukázal pro velké problémy nepoužitelný. Pro menší problémy ale dosahoval ze všech algoritmů nejlepších výsledků.

Literatura

[1] Adam, L., Lendel, G., and Sůva, P. (2011). Krizové scénáře vývoje ekonomiky. Seminární práce MFF UK.

[2] Artzner, P., Delbaen, F., Eber, J. M., and Heath, D. (1999). Coherent measures of risk. *Mathematical Finance*, 9(3):203–228.

[3] Bazaraa, M. S., Sherali, H. D., and Shetty, C. M. (2006). *Nonlinear programming. Theory and algorithms. 3rd ed.* Hoboken, NJ: John Wiley & Sons. 853 p.

[4] Birge, J. R., Dempster, M. A. H., Gassmann, H. I., Gunn, E. A., King, A. J., and Wallace, S. W. (1987). A standard input format for multiperiod stochastic linear programs. *Mathematical Programming Society Committee on Algorithms Newsletter*, 17:1–19.

[5] Birge, J. R. and Louveaux, F. (1997). *Introduction to stochastic programming.* Springer Series in Operations Research. New York, NY: Springer. 421 p.

[6] Gassmann, H. I. (2005). The SMPS format for stochastic linear programs, http://myweb.dal.ca/gassmann/smps2.htm.

[7] Gondzio, J. and Kouwenberg, R. (2001). High-performance computing for asset-liability management. *Operations Research*, 49(6):879–891.

[8] Kall, P. and Mayer, J. (2005). *Stochastic linear programming. Models, theory, and computation.* International Series in Operations Research & Management Science. New York, NY: Springer. 397 p.

[9] Kall, P. and Mayer, J. (2007). SLP-IOR, User's guide, www.ior.uzh.ch/ /research/stochOpt.

[10] Kall, P. and Wallace, S. W. (1994). *Stochastic programming.* New York, NY: Wiley. 307 p.

[11] King, A. J., Wright, S. E., Parija, G. R., and Entriken, R. (2005). The IBM stochastic programming system. Wallace, S. W. et al., Applications of stochastic programming. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM). Philadelphia, PA: MPS, Mathematical Programming Society. MPS/SIAM Series on Optimization 5:21-36.

[12] Künzi-Bay, A. and Mayer, J. (2006). Computational aspects of minimizing conditional value-at-risk. *Computational Management Science*, 3(1):3–27.

[13] Kopa, M., editor (2008). *On selected software for stochastic programming.* Matfyzpress. Praha. 117 p.

[14] Linderoth, J. and Wright, S. (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24(2-3):207–250.

[15] Louveaux, F. (1978). Piecewise convex programs. *Mathematical programming*, 15:53–62.

[16] Rockafellar, R. T. and Uryasev, S. (2000). Optimization of conditional value-at-risk. *Journal of Risk*, 2(3):1–26.

[17] Rockafellar, R. T. and Uryasev, S. (2002). Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, 26(7):1443–1471.

[18] Rockafellar, R. T. and Wets, R. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147.

[19] Rosenthal, R. E. (2010). Gams – a user's guide, www.gams.com/dd/docs/ /bigdocs.

[20] Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333.

[21] Ruszczyński, A. and Shapiro, A., editors (2003). *Stochastic programming.* Handbooks in Operations Research and Management Science 10. Asterdam: Elsevier. 688 p.

[22] Shapiro, A., Dentcheva, D., and Ruszczyński, A. (2009). *Lectures on stochastic programming. Modeling and theory.* MPS/SIAM Series on Optimization 9. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM); Philadelphia, PA: Mathematical Programming Society. 436 p.

[23] Van Slyke, R. M. and Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17:638–663.

[24] Wallace, S. W. and Ziemba, W. T., editors (2005). *Applications of stochastic programming.* MPS/SIAM Series on Optimization 5. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM). Philadelphia, PA: MPS, Mathematical Programming Society. 709 p.

Seznam tabulek

Příloha A

Dodatky

V příloze je uveden výběr z [22].

Definice A.1. Pro rozšířenou reálnou funkci $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ definujeme doménu $\operatorname{dom} f = \{x \in \mathbb{R}^n;\ f(x) < +\infty\}$. Řekneme, že f je vlastní, pokud $\operatorname{dom} f \neq \emptyset$ a pro všechna $x \in \mathbb{R}^n$ platí $f(x) > -\infty$.

Dále řekneme, že f je zdola polospojitá v bodě x_0, pokud

$$f(x_0) \leq \liminf_{x \to x_0} f(x).$$

Funkce f je zdola polospojitá, pokud je zdola polospojitá v každém bodě.

Definice A.2. Množina $C \subset \mathbb{R}^n$ je kužel, pokud pro každé $x \in C$ a $t \geq 0$ je $tx \in C$. Polární kužel příslušný kuželi C definujeme jako

$$C^* = \{z \in \mathbb{R}^n;\ z'x \leq 0,\ \forall x \in C\}.$$

Recesní kužel pro obecnou množinu C definujeme jako

$$\{h \in \mathbb{R}^n;\ x + th \in C,\ \forall x \in C,\ \forall t > 0.\}$$

Pro kužel $C \subset \mathbb{R}^n$ platí $C^{**} = (C^*)^* = C$ právě tehdy, když C je uzavřený a konvexní. Konvexní množina $C \subset \mathbb{R}^n$ je omezená, pokud je její recesní kužel tvořen pouze nulou.

Definice A.3. Pro obecnou množinu $C \subset \mathbb{R}^n$ dále definujme opěrnou funkci $s(h) = s_C(h) = \sup_{z \in C} z'h$.

Pokud je C konvexní a uzavřená, pak pod normálním kuželem pro množinu C a bod $x_0 \in C$ rozumíme množinu

$$\mathcal{N}_C(x_0) = \{z;\ z'(x - x_0) \leq 0,\ \forall x \in C\}.$$

Opěrná funkce je konvexní, pozitivně homogenní a zdola polospojitá funkce. Pokud jsou C_1 a C_2 konvexní uzavřené množiny, pak pro jejich nosiče platí $s_{C_1} \leq s_{C_2}$ právě tehdy, když $C_1 \subset C_2$. Normální kužel $\mathcal{N}_C(x_0)$ je vždy uzavřený a konvexní.

Nechť $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ je konvexní funkce a nechť x_0 je vnitřní bod $\operatorname{dom} f$. Pokud $f(x_0) > -\infty$, pak f je Lipschitzovsky spojitá na nějakém okolí x_0 a směrové derivace v x_0 jsou konečné.

Věta A.4. *Nechť $\psi(x, y)$ je sdruženě konvexní funkce v obou argumentech. Potom $\vartheta(x) = \min_y \psi(x, y)$ je též konvexní funkce.*

Řekneme, že $z \in \mathbb{R}^n$ je subgradient funkce f v bodě x_0, pokud pro všechna $x \in \mathbb{R}^n$ platí

$$f(x) - f(x_0) \geq z'(x - x_0).$$

Množinu všech subgradientů nazveme subdiferenciál, který značíme $\partial f(x_0)$. Funkce f je subdiferencovatelná v bodě x_0, pokud je $\partial f(x_0) \neq \emptyset$. Pro subdiferenciál součtu funkcí platí následující věta.

Věta A.5 (Moreau-Rockafellar). *Nechť pro $i = 1, \ldots, m$ jsou $f_i : \mathbb{R}^n \to \overline{\mathbb{R}}$ konvexní vlastní funkce a $f = f_1 + \cdots + f_m$. Pokud pro x_0 jsou $f_i(x_0)$ konečné, pak*

$$\partial f_1(x_0) + \cdots + \partial f_m(x_0) \subset \partial f(x_0).$$

Předchozí inkluzi můžeme napsat jako rovnost, pokud je splněna alespoň jedna z následujících podmínek

(i) $\cap_{i=1}^m \operatorname{rint}(\operatorname{dom} f_i)$ *je neprázdná*

(ii) pro nějaké k jsou $f_1, \ldots, f_k$ polyedrické a průnik množin $\cap_{i=k+1}^m \operatorname{rint}(\operatorname{dom} f_i)$ a $\cap_{i=1}^k \operatorname{dom} f_i$ je neprázdný

(iii) existuje $\bar{x} \in \operatorname{dom} f_m$ takový, že pro všechna $i = 1, \ldots, m-1$ platí $\bar{x} \in \operatorname{int}(\operatorname{dom} f_i)$

Pokud je f subdiferencovatelná v x_0, pak $\mathcal{N}_{\operatorname{dom} f}(x_0)$ je identický s recesním kuželem pro množinu $\partial f(x_0)$. Dále pro všechna $x \in \mathbb{R}^n$ platí $f(x) > -\infty$, a tedy f je vlastní. Dále platí, že f je diferencovatelná v x_0 právě tehdy, když je její subdiferenciál tvořen pouze jediným bodem.

Definice A.6. Pro $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ definujme sdruženou funkci

$$f^*(z) = \sup_{x \in \mathbb{R}^n} \{z'x - f(x)\}.$$

Věta A.7 (Fenchel-Moreau). *Nechť $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ je vlastní konvexní funkce. Potom $f^{**} = \operatorname{lsc} f$, kde $\operatorname{lsc} f$ je největší zdola polospojitá funkce, která je menší nebo rovna než f.*

Sdružená funkce je vždy konvexní a zdola polospojitá. Pokud pro nějaké $x \in \mathbb{R}^n$ platí $f(x) = -\infty$, pak $f^* \equiv -\infty$ a $f^{**} \equiv +\infty$. Pro vlastní a konvexní funkci f platí $f = f^{**}$ právě tehdy, když je f zdola polospojitá.

Důsledek A.8. Nechť f je konvexní, vlastní a zdola polospojitá. Pokud $f(x) < +\infty$, pak platí

$$\partial f(x) = \arg\max_z \{z'x - f^*(z)\}. \tag{A.1}$$

Důkaz. Nechť $z \in \partial f(x)$. Potom pro všechna y platí $z'y - f(y) \leq z'x - f(x)$, což je ekvivalentní s nerovností $\sup_y \{z'y - f(y)\} = z'x - f(x)$ neboli

$$f^*(z) = z'x - f(x).$$

Protože sdružená funkce je vždy zdola polospojitá, konvexní a v tomto případě navíc i vlastní, platí z Fenchel-Moreauovy věty A.7 $f^{***} = f^*$. Z tohoto vztahu nám ihned vyjde

$$z \in \partial f^{**}(x) \iff f^{**}(x) = z'x - f^*(z)$$

Protože $f^{**}(x) = \sup_z\{z'x - f^*(z)\}$, dostaneme $\partial f^{**}(x) = \arg\max_{z\in\mathbb{R}^n}\{z'x - f^*(z)\}$. Zbytek tvrzení pak plyne z předpokladů a Fenchel-Moreauovy věty. □

Věta A.9 (Hoffman). *Pro matici A o rozměrech $m \times n$ definujme multifunkci $\mathcal{M}(b) = \{x \in \mathbb{R}^n;\ Ax \le b\}$. Potom existuje $\kappa > 0$ závisející pouze na A takové, že pro všechna $x \in \mathbb{R}^n$ a $b \in \operatorname{dom}\mathcal{M}$ platí*

$$\operatorname{dist}(x, \mathcal{M}(b)) \le \kappa||(Ax - b)_+||.$$

Definice A.10. Multifunkce $\mathcal{G} : \Omega \rightrightarrows \mathbb{R}^n$ je zobrazení z množiny Ω do množiny podmnožin $\mathbb{R}^n$. Řekneme, že $\mathcal{G}$ má uzavřené hodnoty, pokud pro všechna $\omega \in \Omega$ je $\mathcal{G}$ uzavřená množina. Multifunkce s uzavřenými hodnotami je měřitelná, pokud pro každou uzavřenou množinu $A \subset \mathbb{R}^n$ je $\mathcal{G}^{-1}(A) = \{\omega \in \Omega;\ \mathcal{G}(\omega) \cap A \neq \emptyset\}$ $\mathcal{F}$-měřitelná.

Definice A.11. Řekneme, že funkce $F(x, \omega) : \mathbb{R}^n \times \Omega \to \overline{\mathbb{R}}$ je náhodná zdola polospojitá, pokud má epigrafická funkce $\omega \mapsto \operatorname{epi} F(\cdot, \omega)$ uzavřené hodnoty a je měřitelná.

Věta A.12. *Nechť σ-algebra $\mathcal{F}$ je P-úplná. Potom funkce $F(x, \omega) : \mathbb{R}^n \times \Omega \to \overline{\mathbb{R}}$ je náhodná zdola polospojitá právě tehdy, když je pro každé $\omega \in \Omega$ funkce $F(\cdot, \omega)$ zdola polospojitá a funkce $F(\cdot, \cdot)$ je měřitelná vzhledem k σ-algebře $\mathcal{B} \times \mathcal{F}$.*

Věta A.13. *Nechť $F(x, \omega) : \mathbb{R}^n \times \Omega \to \overline{\mathbb{R}}$ je náhodná zdola polospojitá funkce. Potom $\vartheta(\omega) = \inf_{x\in\mathbb{R}^n} F(x, \omega)$ a $X^*(\omega) = \arg\min_{x\in\mathbb{R}^n} F(x, \omega)$ jsou měřitelné funkce.*

Věta A.14. *Nechť funkce $F(x, \omega) : \mathbb{R}^n \times \Omega \to \overline{\mathbb{R}}$ je náhodná zdola polospojitá a $F(\cdot, \omega)$ je konvexní pro $\omega \in \Omega$ skoro jistě. Nechť dále $f(x) = \mathrm{E}\, F(x, \omega)$ je vlastní a $\operatorname{dom} f$ má neprázdný vnitřek. Potom pro $x_0 \in \operatorname{dom} f$ platí*

$$\partial f(x_0) = \int_\Omega \partial F(x_0, \omega) dP(\omega) + \mathcal{N}_{\operatorname{dom} f}(x_0).$$

Lemma A.15 (Fatou). *Nechť pro posloupnost funkcí $\{f_n\}$ existuje P–integrovatelná funkce g taková, že pro všechna n platí $f_n(\cdot) \ge g(\cdot)$. Potom*

$$\liminf \int_\Omega f_n d\,\mathrm{P} \ge \int_\Omega \liminf f_n d\,\mathrm{P}\,.$$