Univerzita Pavla Jozefa Šafárika v Košiciach

Prírodovedecká fakulta

PODPORA PRE RIEŠENIE ÚLOHY LINEÁRNEHO PROGRAMOVANIA

ŠTUDENTSKÁ VEDECKÁ KONFERENCIA

Študijný odbor:	Informatika
Školiace pracovisko:	Ústav informatiky
Vedúci práce:	RNDr. Pavol Široczki
Konzultant:	doc. RNDr. Roman Soták, PhD.

Košice 2015

Jozef Džama

Poďakovanie

Rád by som poďakoval vedúcemu práce RNDr. Pavlovi Široczkému a konzultantovi doc. RNDr. Romanovi Sotákovi, PhD. za vedenie, rady a pomoc počas tvorby mojej práce. Tiež chcem poďakovať RNDr. Františkovi Galčíkovi, PhD., najmä za nasmerovanie v oblasti kreslenia zlomkov vo Swingu a Bc. Miroslavovi Opielovi za rôzne rady.

Abstrakt

Cieľom tejto práce je vytvoriť interaktívnu podporu riešenia úloh lineárneho programovania s možnosťami použitia simplexovej metódy a jej variantov ako sú napríklad revidovaná a duálna simplexová metóda. Súčasťou práce je aj stručný prehľad iných metód používaných na riešenie úloh lineárnej optimalizácie a porovnanie časových zložitostí týchto metód so zložitosťou simplexovej metódy.

Obsah

Kapitola 1

Úvod

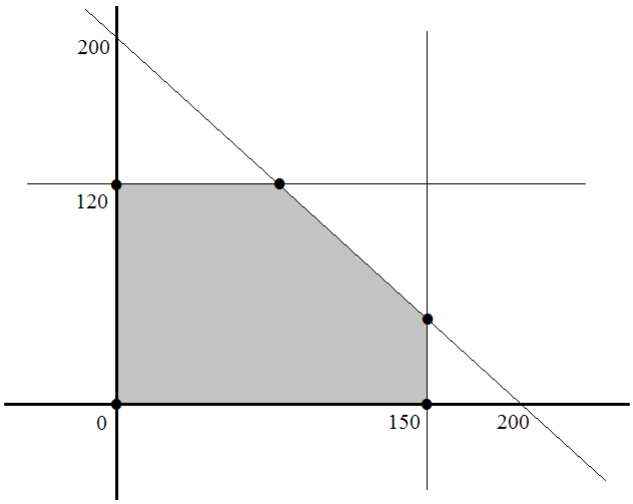
Problémy reálneho života je možné v mnohých prípadoch popísať (v zjednodušenej forme) matematickými modelmi, u ktorých je známy postup ako ich riešiť. Jedným z týchto modelov je lineárna optimalizácia, ktorou sa budeme zaoberať v tejto práci.

Uvažujme nasledujúcu situáciu. Majme továreň, v ktorej za jeden deň dokážu vyrobiť najviac 150 výrobkov typu A a 120 výrobkov typu B. V továrni navyše dokážu denne vyrobiť len 200 ľubovoľných výrobkov. Továreň predáva výrobok A za 3 eurá a výrobok B za 1 euro. Koľko akých výrobkov by mali vyrobiť, aby bol príjem čo najväčší? Riešenie tejto úlohy je jednoduché, pre pochopenie však stručne vysvetlíme ako je možné uvedenú úlohu vyriešiť graficky. Označme denné množstvo vyrobených výrobkov typu A ako x, množstvo výrobkov typu B ako y. Zo zadania vyplývajú ohraničenia $x \leq 150$, $y \leq 120$, $x + y \leq 200$, $x \geq 0$, $y \geq 0$. Posledný dôležitý vzťah popisuje takzvaná účelová funkcia vyjadrujúca príjem továrne: $3x + y \to \max$.

Nerovnosti zakreslíme do súradnicovej sústavy (pozri obr. 1.1). Sivá plocha znázorňuje body, ktoré spĺňajú všetky spomínané ohraničenia.

Z teórie lineárnej optimalizácie je známe, že maximálna hodnota účelovej funkcie sa bude nadobúdať v niektorom z vrcholov (krajných bodov) sivého päťuholníka; v našom príklade je to bod $x = 150$ a $y = 50$, teda maximálny príjem továrne je 500 eur. Táto úloha nielen znie ako zjednodušený problém nejakej výrobnej firmy, problémy podobného druhu sa naozaj riešia v oblastiach ako výroba, doprava, energie. Graficky sa však dajú riešiť len špeciálne úlohy (s malým počtom premenných). Ako však postupovať v prípade, že v úlohe vystupujú zložitejšie vzťahy alebo viac premenných?

Úlohy, kde účelová funkcia a ohraničenia sú popísané lineárnymi funkciami vystupujúcich premenných (tak ako v predchádzajúcej úlohe) sa nazývajú úlohami lineárnej optimalizácie. Lineárna optimalizácia je teda oblasť zaoberajúca sa metódami na ur-



Obrázok 1.1: Grafické riešenie úlohy lineárneho programovania.

čovanie najlepšieho výsledku v matematickom modeli určenom lineárnymi vzťahmi. Celočíselná optimalizácia obmedzuje optimálne riešenie na hodnoty z množiny celých čísel. Keďže riešenie týchto úloh vyžaduje relatívne veľký počet matematických operácií a teda je zdĺhavé, pričom chybovosť riešiteľa obvykle nie je nulová, bolo na ich riešenie vyvinuté obrovské množstvo softvéru. Okrem programov určených výlučne na riešenie týchto úloh (napríklad Linear Programming Solver), dokážu tieto úlohy po stiahnutí rozšírení riešiť aj všeobecnejšie zamerané programy ako napríklad MATLAB, Maple alebo Excel. Na našej univerzite bol v minulosti za účelom riešenia úloh lineárnej optimalizácie vyvinutý program CASSIM (Computer ASsisted SImplex Method), ktorý bol a je vhodný predovšetkým na výučbové účely. Dôvody prečo sa hodí na výučbu sú predovšetkým: používa simplexovú metódu (o nej neskôr obšírnejšie), ponúka krokovaný výpočet pomocou ponuky akcií z menu, upozorní používateľa pri pokuse o vykonanie kroku nezodpovedajúceho štandardnému postupu pri riešení úlohy simplexovou metódou a zobrazuje celú úlohu prehľadne v tabuľke. Tento program bol vyvinutý v rokoch 1993 až 1997, s čím súvisí viacero obmedzení pri používaní (predovšetkým spúšťanie v DOSBoxe a obmedzenia na rozmery vstupu). Keďže sa vďaka spomínaným vlastnostiam hodí na výučbu základných princípov simplexovej metódy viac ako iné existujúce programy, rozhodli sme sa vytvoriť nový program s podobnými výhodami, ktorý by bol naďalej použiteľný pri výučbe lineárnej optimalizácie. Našim východiskom je teda program CASSIM, chceme zachovať jeho pozitívne vlastnosti, ale zároveň zjednodušiť a vylepšiť používanie. Na to je nutná znalosť teórie týkajúcej sa riešenia úloh lineárnej optimalizácie, preto bude súčasťou našej práce aj teória z oblasti lineárnej optimalizácie. Navyše sa plánujeme zaoberať ďalšími metódami

riešenia úloh lineárneho programovania, ako sú napríklad metódy vnútorného bodu a elipsoidná metóda, pričom dôraz budeme okrem ich vysvetlenia klásť predovšetkým na porovnanie časovej zložitosti rôznych metód, prípadne na ich ďalšie výhody a nevýhody. Týmto témam sa venuje mnoho prác a štúdií, pokúsime sa na ich základe vyvodiť závery a porovnania v jednotlivých situáciách.

Ciele práce možno teda zhrnúť do týchto dvoch bodov:

1. Vytvoriť program vhodný na výučbu a na riešenie úloh lineárnej a celočíselnej optimalizácie využijúc ako vzor CASSIM.

2. Spracovať teóriu o simplexovej metóde a rozdiely (najmä z pohľadu časovej zložitosti) medzi rôznymi metódami na výpočet úloh lineárnej optimalizácie.

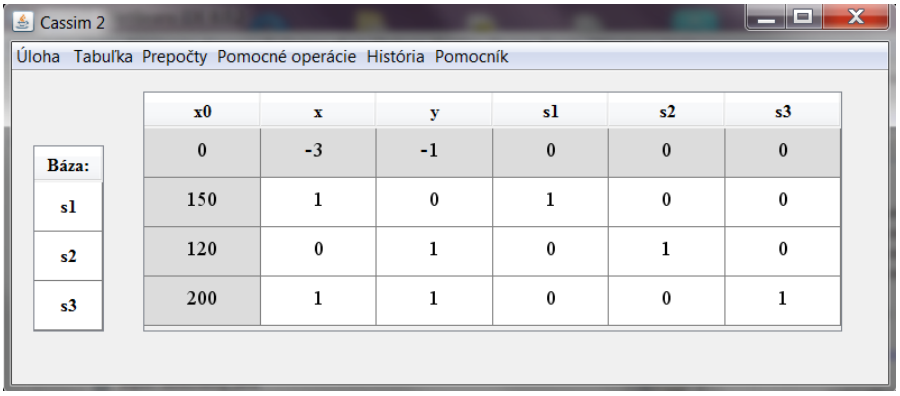
Kapitola 2

Návrh riešenia

Rozdeľme si celý problém na menšie podproblémy, ktorým sa budeme venovať. Prvým krokom je naštudovanie a spracovanie teórie ohľadne simplexovej metódy a ďalších metód, ktoré budú implementované. Následne môžeme prejsť k samotnému programu. Riešenie úloh simplexovou metódou v CASSIMe prebieha tak, že úloha je zapísaná v tabuľke a akcie vykonávame buď na celej tabuľke alebo vybranom riadku, stĺpci alebo bunke.

Zároveň si určíme hlavné požiadavky, ktoré budeme klásť na náš program. Už sme spomínali, ktoré vlastnosti prevezmeme z CASSIMu. Ďalšími požiadavkami sú: použiteľnosť na rôznych platformách, jednoduchá inštalácia a čo možno najmenej obmedzení na veľkosť vstupu a postup používateľa pri riešení.

Ako programovací jazyk sme si zvolili Javu, pretože je nezávislá od platformy, patrí medzi moderné objektovo orientované jazyky a prispelo aj to, že je u nás štandardne vyučovaná a máme s ňou nejaké skúsenosti.



Obrázok 2.2: Typická tabuľka simplexovej metódy v našom programe.

2.1 Simplexová metóda

Simplexová metóda je jedným z algoritmov na riešenie úloh lineárnej optimalizácie. Publikovaný bol Dantzigom v roku 1947. Používa sa na úlohu v takzvanom štandardnom tvare, teda úlohu zadanú účelovou funkciou $\vec{c}^{\top}\vec{x} \to \min$, ktorú minimalizujeme, s ohraničeniami v tvare $A^{\top}\vec{x} = \vec{b}^{\top}$, kde $x_i \geq 0$ pre každé x_i z vektora premenných $\vec{x} = (x_1, \ldots, x_n)$, zatiaľ čo $\vec{c} = (c_1, \ldots, c_n)$ sú koeficienty účelovej funkcie, A je matica rozmeru $m \times n$, $\vec{b} = (b_1, \ldots, b_m)$ sú konštanty a pre každé j platí $b_j \geq 0$. V geometrickom znázornení ohraničenia a nezápornosť x_i vytvoria (nie nutne ohraničený) konvexný mnohosten. Je známe, že ak má úloha optimálne riešenie, tak sa jedno z jej optimálnych riešení nachádza v jednom z krajných bodov (vrcholov) tohto mnohostena (bod optima nemusí byť jediný). To redukuje problém na konečný výpočet, keďže vrcholov je konečne veľa. Ďalšou vlastnosťou je, že pokiaľ nie je v krajnom bode hodnota účelovej funkcie minimálna, tak existuje taká hrana (prípustného konvexného mnohostena) obsahujúca tento bod, po ktorej smerom od daného bodu hodnota účelovej funkcie klesá. Simplexová metóda využíva tento prístup, teda prechádza cez hrany do krajných bodov so stále menšou hodnotou účelovej funkcie. Takýto postup sa opakuje, kým sa nedostaneme do bodu s minimálnou hodnotou účelovej funkcie alebo nenájdeme nekonečnú hranu pozdĺž ktorej hodnota účelovej funkcie klesá, vtedy vravíme, že úloha je neohraničená. Tento postup je, ako sme už naznačili, konečný, keďže počet krajných bodov prípustnej množiny je konečný.

Riešenie úloh lineárnej optimalizácie simplexovou metódou prebieha v dvoch fázach. V prvej fáze je nájdený štartovací krajný bod. Podľa zadania to môže byť buď triviálne alebo je na to potrebná pomocná úloha. V pomocnej úlohe sa krajný bod buď nájde, potom pokračujeme druhou fázou, alebo sa nenájde, vtedy úloha nemá riešenie a hovoríme, že úloha je neprípustná. Druhá fáza využíva nájdený štartovací bod a znižuje hodnotu účelovej funkcie vhodným pohybom do ďalších bodov pivotovaním podľa istých vzťahov. Popísať tu to všetko by bolo zdĺhavé. Výsledkom je buď optimálna hodnota účelovej funkcie alebo zistenie, že je úloha neohraničená.

2.2 Príprava na výpočet

Rozdeľme program na menšie časti, ktoré musíme vytvoriť. Najprv dáme používateľovi na výber čo chce vykonať (načítanie uloženého alebo zadávanie nového vstupu, otvorenie riešenia). V prípade, že si vyberie zadanie novej úlohy, otvorí sa nové okno

s tabuľkou na zadanie vstupu. Pred otvorením používateľ uvedie počet premenných a počet ohraničení. Potom používateľ môže zadávať vstup. Spomenieme obmedzenia pre zadávané reťazce – každá hodnota musí byť buď celé číslo, desatinné číslo alebo zlomok. Aby sme sa vyhli nekonzistentným dátam, po otvorení nastavíme vo všetkých bunkách hodnotu 0 a zmena je povolená len ak hodnotu vieme naparsovať a zapísať do zlomku. Používateľ môže v samostatnej tabuľke zmeniť aj názvy premenných. Názvy premenných, ktoré môžu byť použité v neskorších fázach výpočtu zakazujeme. Po kliknutí na bunky znázorňujúce ohraničenia, nezápornosť a pravú stranu účelovej funkcie sú na výber ponúknuté preddefinované možnosti podľa charakteru bunky. Ak úloha nie je v štandardnom tvare, tak ju program počas načítania automaticky do tohto tvaru prevedie (zmení účelovú funkciu, pridá nové premenné kvôli nezápornosti a ohraničeniam).

Na zlomky používame triedu BigFraction z balíčka org.apache.commons.math3. Objekt tejto triedy ukladá zlomok do dvoch BigIntegerov – reprezentujúcich čitateľa a menovateľa zlomku. Pôvodne sme plánovali použiť triedu Fraction s primitívnymi celočíselnými premennými (int), ale tá pri hľadaní spoločného menovateľa zlomkov rýchlo pretekala. Nami zvolená trieda taktiež dokáže efektívne robiť všetky nami používané matematické operácie so zlomkami.

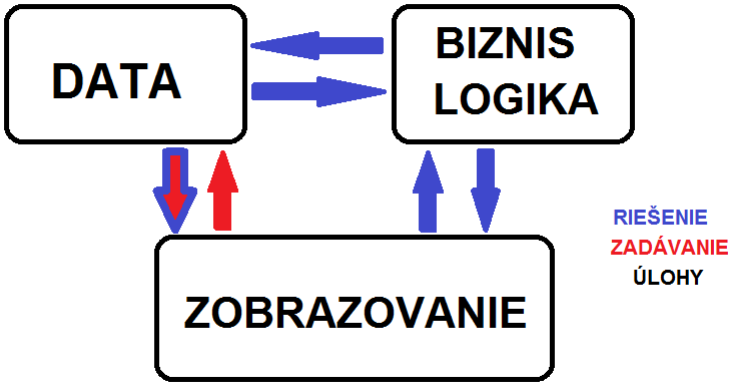
2.3 Výpočet

2.3.1 Štruktúra:

Po načítaní vstupu sa otvorí okno s výpočtom. Pri budovaní architektúry programu sa inšpirujeme známymi návrhovými vzormi Model-view-controller (MVC) a Data Access Object (DAO). Štruktúra bude vyzerať tak, že oddelíme dáta, biznis logiku a zobrazovanie. Zobrazované sú priamo dáta uložené do tabuliek, ale vykonávanie krokov riešenia alebo kontrol prebieha cez biznis logiku.

2.3.2 Dáta:

Dáta ukladáme v samostatnej triede. Potrebujeme zabezpečiť, aby boli dáta v celom programe rovnaké, na to je vhodný návrhový vzor singleton, teda budeme mať jedinú inštanciu tejto triedy v systéme. V tejto triede bude aktuálna tabuľka uložená v dvojrozmernom poli, indexy prvkov aktuálnej bázy v jednorozmernom poli a ďalšie



Obrázok 2.3: Štruktúra programu.

potrebné dáta (napríklad počet pomocných premenných v pomocnej úlohe alebo či riešime revidovanou metódou a podobne).

Rozhodli sme sa, že zadanie a pretransformovanú úlohu zo zadania budeme ukladať do súboru typu csv. Tento formát je na to vhodný vďaka svojej jednoduchosti a mala by byť zachovaná aj čitateľnosť súboru. Do súboru ukladáme pôvodné zadanie úlohy, tabuľku pretransformovanú do štandardného tvaru a kroky výpočtu vykonané používateľom v zakódovanej forme. Načítať je potom možné buď zadanie na úpravu alebo aj riešenie, kedy sú načítané aj kroky výpočtu a pomocou histórie cez opakovanie krokov je možné upraviť úlohu do stavu, v akom bola pri uložení.

2.3.3 Používateľské rozhranie:

Na zobrazovanie použijeme knižnicu Swing. Potrebujeme zobraziť dve tabuľky, prvú reprezentujúcu aktuálne koeficienty riešenej úlohy a druhú, ktorá podľa poľa s indexmi prvkov bázy zobrazí z poľa názvov stĺpcov tie, ktoré sa práve nachádzajú v báze. Vzhľad rozhrania ilustrujeme obrázkom 2.2. Tieto tabuľky sa po väčšine operácií menia, pri spustení pomocnej úlohy, Gomoryho rezu, revidovanej úlohy alebo po vymazaní nulového riadka sa menia aj ich rozmery. Komplikácie nastávajú pri vykresľovaní zlomkov do tabuľky. Java ani Swing na to nemajú pripravené vhodné metódy. Pomôže nám však trieda Graphics2D, pomocou ktorej si zobrazovanie zlomkov sami prispôsobíme.

Na zbytok používateľského rozhrania využijeme menu, kde prehľadne podelíme jednotlivé metódy do skupín. Súčasťou menu je aj jednoduché mapovanie jednotlivých akcií na klávesové skratky, ktorými ich môžeme vyvolať. Okrem menu využijeme

navyše ešte jedno tlačidlo na ukončenie pomocnej úlohy a Combo box (pole so zoznamom) na výber premennej pri revidovanej metóde. Textové informácie pre používateľa budeme zobrazovať cez modálne vyskakovacie okná.

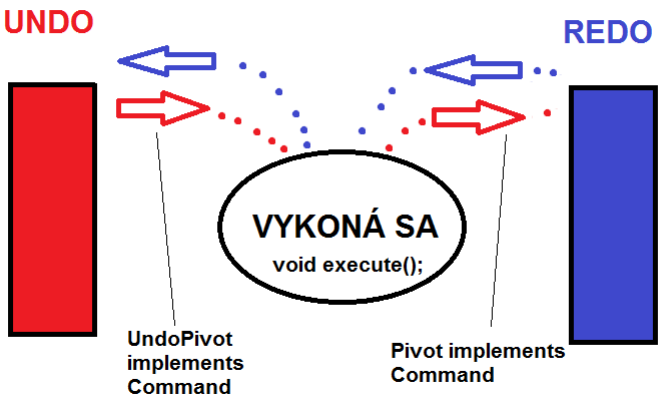
2.3.4 Biznis logika:

Biznis logiku bude riešiť samostatná servisná trieda. Tu budú implementované metódy na jednotlivé akcie - kroky výpočtu, ktoré budú vykonávať operácie na dátach. Pred väčšinou akcií bude zavolaná funkcia, ktorej úlohou bude kontrola či je tento krok v súlade so štandardným postupom v aktuálne používanej metóde. Ak to podľa vrátenej hodnoty nie je splnené, zobrazovanie reaguje tak, že upozorní používateľa na neštandardnosť akcie. Väčšinu akcií však dovolíme po potvrdení užívateľom vykonať. Operácie, ktoré nemenia dáta majú ako návratovú hodnotu index určujúci riadok alebo stĺpec, ktorý je následne zvýraznený. Príkladom je operácia minimum, ktorá pre daný stĺpec určí najvhodnejší riadok na vykonanie ďalšej akcie, teda určí jednu bunku, ktorá je potom zvýraznená. Druhý typ operácií sú tie, ktoré menia dáta. Tie sa vykonajú a vrátia objekt implementujúci interface Command. Podrobnejšie informácie o objekte Command sú uvedené v ďalšom odstavci.

2.3.5 História:

V programe ako tento sa môže používateľ ľahko zmýliť, prípadne vyskúšať, aký vplyv na výsledok má neštandardný postup riešenia. Keďže všetky operácie, ktoré budeme vykonávať sú reverzibilné (pri niektorých je potrebné uloženie dodatočných informácií – presnejšie maximálne hodnoty koeficientov jedného stĺpca) rozhodli sme sa, že implementujeme aj históriu výpočtu. Používame na to dva zásobníky a návrhový vzor Command pattern. Zvažovali sme aj ukladanie celých tabuliek s potrebnými informáciami, respektíve celého stavu singletonovskej triedy. Ale keďže reverzné operácie nie sú príliš zložité na implementáciu a ani ich vykonanie nie je časovo náročné, rozhodli sme sa šetriť pamäť, čo môže byť užitočné najmä pri úlohách veľkých rozmerov. Návrhový vzor Command sme použili takto:

- Vytvorili sme interface Command s metódami execute() na jeho vykonanie, toString() na vypísanie čo vykoná a getType() na určenie jeho typu. Následne sme naprogramovali jeho implementácie pre jednotlivé reverzné operácie.



Obrázok 2.4: História výpočtu.

- Receiver je trieda s biznis logikou alebo pre niektoré operácie priamo singletonovská trieda s dátami.

- Command – objekty sa vytvoria vždy po vykonaní metódy, ktorá zmení dáta a uložia sa do UNDO zásobníka alebo pri kroku späť vyberieme Command z UNDO zásobníka, vykonáme ho, pri tom sa vytvorí iný Command a uloží do REDO zásobníka. Napokon len zobrazíme aktuálne dáta. Podobne sa postupuje pri zopakovaní (REDO) kroku. Tieto operácie riadi hlavná trieda (trieda hlavného okna), teda plní funkcie Invokera a Clienta.

Kapitola 3

Časová zložitosť

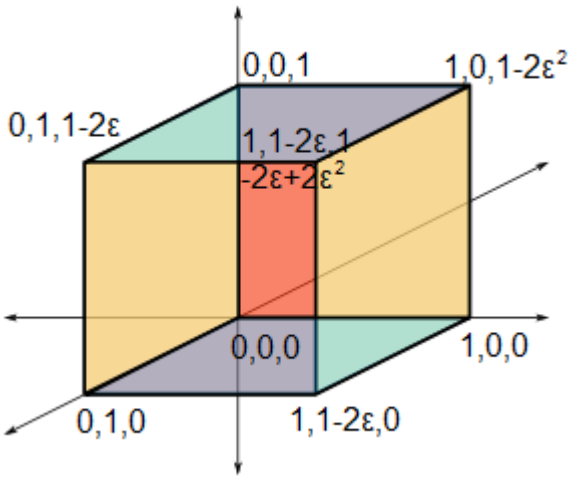
Napokon spracujeme teóriu o rôznych metódach riešenia úloh lineárnej optimalizácie. V bežnej praxi sa používajú aj programy využívajúce simplexovú metódu napriek tomu, že je známy príklad, kedy má simplexová metóda exponenciálnu časovú zložitosť. Ďalším typom metód ktorým sa budeme venovať sú metódy vnútorného bodu, kde patrí elipsoidná a Karmarkarova metóda.

3.1 Časová zložitosť simplexovej metódy:

Pri riešení simplexovou metódou býva čas výpočtu úloh bežnej praxe nie veľkých rozmerov prijateľný, v priemernom prípade je jej zložitosť polynomiálna. Mnohé numerické experimenty dokonca ukazujú, že na úlohu s m ohraničeniami stačí m simplexových iterácií. Navyše je táto metóda jednoduchá na pochopenie a implementáciu, čo znamená, že je súčasťou základných kurzov lineárnej optimalizácie, na podporu ktorých by mal tento program slúžiť, preto sme v našom programe využili práve simplexovú metódu.

Ako sme už spomenuli, v najhoršom prípade môže mať simplexová metóda až exponenciálnu časovú zložitosť. Príkladom je takzvaná Kleeho-Mintyho kocka, popísaná v roku 1972 (pozri [2]).

Príklad takejto úlohy zostrojíme z úlohy na (trojrozmernej) kocke, na ktorej vyriešenie potrebujeme jednu iteráciu simplexovej metódy (účelová funcia má tvar $x_3 \rightarrow$ max a pre $i \in \{1, 2, 3\}$: $0 \leq x_i \leq 1$). Avšak po perturbovaní (deformácii) kocky sa situácia zmení. Uvažujme teda zmenenú úlohu v tvare:



Obrázok 3.5: Kleeho-Mintyho kocka.

$$\begin{array}{rcrcrcl} \varepsilon^2 x_1 & + & \epsilon x_2 & + & x_3 & \rightarrow & max \\ x_1 & & & & & \leq & 1 \\ 2\varepsilon x_1 & + & x_2 & & & \leq & 1 \\ \varepsilon^2 x_1 & + & 2\varepsilon x_2 & + & x_3 & \leq & 1 \\ & & & & x_{1-3} & \geq & 0 \end{array}$$

Táto úloha zodpovedá pre $\varepsilon = 0$ pôvodnej úlohe. K riešeniu by sme teda došli hneď po prvom kroku. Ale v zmenenej úlohe budeme k riešeniu postupovať z vrcholu (0,0,0) až do vrcholu (0,0,1) postupne cez všetky vrcholy. To je 8 iterácií, čo zodpovedá exponenciálnej zložitosti.

3.2 Elipsoidná metóda a jej časová zložitosť

Historicky ďalšia metóda je elipsoidná metóda. Patrí do inej skupiny metód ako simplexová metóda, konkrétne medzi metódy vnútorného bodu. Jej cieľom je nájsť bod x v ohraničenom mnohostene $P = \{x : Cx \leq d\}$ alebo korektne určiť $P = \emptyset$. Hlavná myšlienka postupu:

1. Začneme s takou guľou v začiatku súradnicovej sústavy ($a_0 = 0$), ktorá je dosť veľká, aby obsahovala P. Platí, že dosť veľká guľa má polomer 2^L, pričom L zodpovedá počtu bitov potrebných na zakódovanie C a d.

2. Zistíme či $a_k \in P$. Ak áno, vrátime a_k.

3. Inak máme $a_k \notin P$, pretože $C_j a_k > d_j$ pre nejaké j. Potom rozdelíme elipsoid E_k cez jeho stred a_k nadrovinou rovnobežnou s $C_j x = d_j$.

4. Vypočítame nový elipsoid E_{k+1} so stredom v a_{k+1} obsahujúci zhruba polovicu E_k.

Táto metóda postupuje k riešeniu tak, že objem elipsoidu zmenšuje o faktor $e^{\frac{1}{2(n+1)}}$ v každej iterácii. Začínajúc s guľou s objemom $2^{O(nL)}$ je môžné ukázať, že algoritmus skončí, keď objem bude $2^{-\Omega(nL)}$. Poznamenáme, že po každých $O(n)$ iteráciách sa objem zmenší o faktor e, a teda po $O(n^2 L)$ iteráciách budeme mať žiadaný výsledok.

Ešte načrtneme myšlienku algoritmu, ktorý môže byť rozšírený na polynomiálny algoritmus na lineárnu optimalizáciu. Hľadáme optimálne riešenie úlohy $min\{c^\top x : Ax \leq b, x \geq 0\}$.

- Majúc aktuálny elipsoid $E(a_k, A_k)$, ak $a_k \notin \{c^\top x : Ax \leq b, x \geq 0\}$ keďže $A_j a_k > b_k$ pre nejaké j, potom môžme použiť nadrovinu $A_j x \leq A_j a_k$ na rozdelenie oblasti na dve časti.

- Ak $a_k \in \{c^\top x : Ax \leq b, x \geq 0\}$, tak použijeme nadrovinu $c^\top x \leq c^\top a_k$ na rozdelenie oblasti a necháme si tú menšiu oblasť, ktorá obsahuje optimálne riešenie. Toto je podstatná časť, keďže toto sa dá urobiť v polynomiálnom čase, tak vieme celú úlohu vyriešiť v polynomiálnom čase. Grafická ukážka je na obrázku 3.6. Táto metóda sa však v praxi zvyčajne nepoužíva, pretože je pomalá na bežne riešených úlohách.

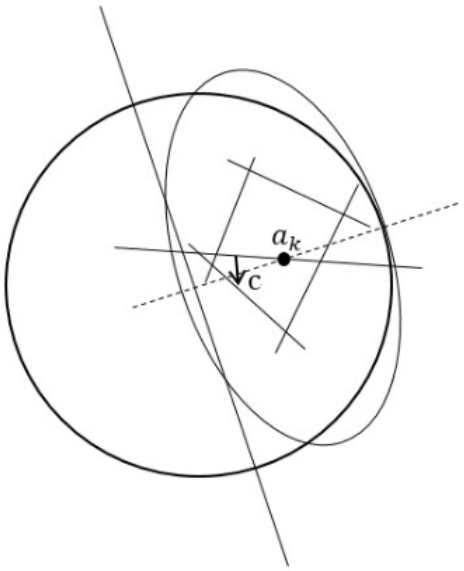
3.3 Karmarkarova metóda a jej časová zložitosť

Karmarkarova metóda vychádza z elipsoidnej metódy, teda tiež patrí medzi metódy vnútorného bodu, ale na rozdiel od elipsoidnej metódy je rýchla aj na bežne riešených úlohách v praxi, pričom má polynomiálnu zložitosť. V porovnaní so simplexovou metódou je viditeľne rýchlejšia na veľkých vstupoch. Ďalej ukážeme jej stručný prehľad.

Metóda sa aplikuje na lineárny problém v projektívnom tvare

$$\min\{\vec{c}^\top \vec{x} : A\vec{x} = \vec{0}, \vec{a}^\top \vec{x} = 1, \vec{x} \geq \vec{0}\} \tag{PLP}$$

kde $\vec{a} \in \mathbb{R}^n$ a $\vec{x} \in \mathbb{R}^n$ a A je matica rozmeru $p \times n$. Ohraničenie $A\vec{x} = \vec{0}$ budeme označovať ako homogénne. Základnou myšlienkou algoritmu je potenciálová funkcia, ktorá je homogénna stupňa 0 v $\vec{x}$ a nazývame ju *Karmarkarov potenciál*:

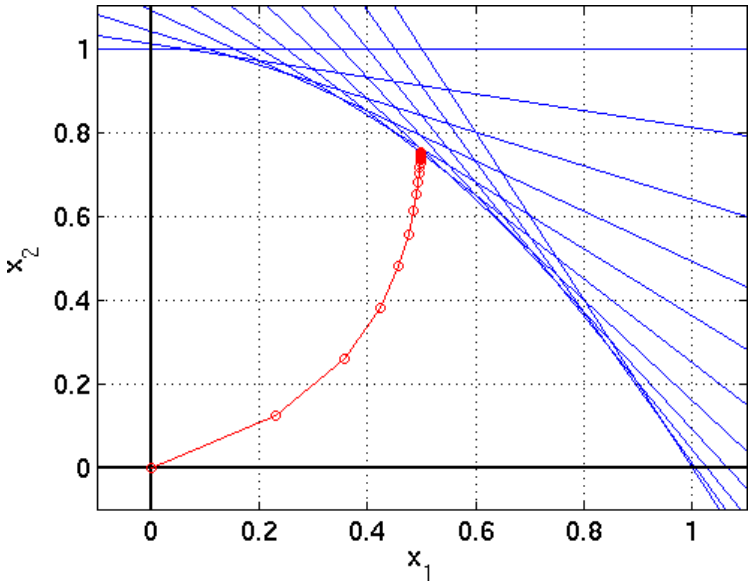


Obrázok 3.6: Grafická ukážka k elipsoidnej metóde.

$$\varphi_v(\vec{x}) = n \log(\vec{c}^\top \vec{x} - v\vec{a}^\top \vec{x}) - \sum_{i=1}^{n} \log x_i = \log(\vec{c}^\top \vec{x} - v\vec{a}^\top \vec{x})^n + \pi(x),$$

pričom $v \in \mathbb{R}$ je dolný odhad $v(PLP)$ (teda $v \leq v(PLP)$. Algoritmus minimalizuje φ_v, pričom upravuje hodnotu parametra v v každej iterácii. Výsledkom je konvergencia ceny k $v(PLP)$. Tento algoritmus má značnú schopnosť, a to že redukuje φ aspoň o $\frac{1}{4}$ v každej iterácii. Toto posúvanie sa je počítané nasledovne:

1. Škálujeme premennú posúvajúc $\vec{x}$ do bodu $(1, \ldots, 1)^\top \in \mathbb{R}^n$.
2. Vypočítame smer $\vec{d}$, opak k ortogonálnej projekcii gradientu $\vec{x} \mapsto \varphi_v(\vec{x})$ do základu homogénnych ohraničení.
3. Hľadáme v smere $\vec{d}$. Bod $\hat{\vec{x}}$ získame z $\varphi_v(\hat{\vec{x}}) < \varphi_v(\vec{x})$ a $A\hat{\vec{x}} = 0$ (pričom vo všeobecnosti $\vec{a}^\top \hat{\vec{x}} \neq 0$).
4. Vymeníme spať: $\vec{x} \leftarrow \hat{\vec{x}}/\vec{a}^\top \hat{\vec{x}}$.



Obrázok 3.7: Grafická ukážka ku karmarkarovej metóde. Úloha s dvoma premennými.

Kapitola 4

Záver

Po naštudovaní potrebnej teórie sme vytvorili program, v ktorom sme okrem iného implementovali jednotlivé kroky simplexovej metódy a ostatných metód a tiež ďalšie užitočné operácie. Následne sme spravili intuitívne používateľské rozhranie s klávesovými skratkami. Veríme, že nami vytvorený program bude v budúcnosti použiteľný v základných kurzoch lineárnej optimalizácie. Napokon sme naštudovali a spracovali teóriu o ďalších metódach riešenia úloh lineárnej optimalizácie. Stále ostáva priestor pre rôzne úpravy alebo vylepšenia, aby sme ciele splnili čo najlepšie.

Zoznam použitej literatúry

[1] Plesník, Dupačová, Vlach: Lineárne programovanie, Alfa, Bratislava 1990

[2] Klee, Minty: How good is the simplex algorithm? In Shisha, Oved. Inequalities III (Proceedings of the Third Symposium on Inequalities held at the University of California, Los Angeles, Calif., September 1–9, 1969, dedicated to the memory of Theodore S. Motzkin). New York-London: Academic Press(1972). pp. 159–175.

[3] http://link.springer.com/chapter/10.1007/978-3-662-05078-1_24

[4] http://people.orie.cornell.edu/dpw/orie6300/Lectures/lec20.pdf