Univerzita Komenského v Bratislave Fakulta matematiky,
fyziky a informatiky

Riešenie problému distribuovaného nákupu

Bakalárska práca

2017
Lukáš Ivan

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Riešenie problému distribuovaného nákupu

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Michal Forišek, PhD.

Bratislava, 2017
Lukáš Ivan

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Lukáš Ivan
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Riešenie problému distribuovaného nákupu
A solution to the distributed shopping problem

Cieľ: Práca má nasledujúce ciele: 1. Naštudovať a spracovať problematiku súvisiacu s problémom distribuovaného nákupu. 2. Implementovať aplikáciu ktorá bude zbierať praktické dáta a generovať prakticky relevantné inštancie. 3. Preskúmať a implementovať čo najefektívnejší algoritmus riešiaci tento problém.

Vedúci: RNDr. Michal Forišek, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 02.11.2016

Dátum schválenia: 04.11.2016 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Týmto by som chcel poďakovať školiteľovi mojej bakalárskej práce RNDr. Michalovi Forišekovi, PhD. za jeho odborné vedenie, metodickú pomoc a cenné rady, ktoré mi poskytol pri jej vypracovaní.

Abstrakt

Cieľom práce je návrh a implementácia aplikácie na riešenie problému distribuovaného nákupu. Riešenie problému má základ v optimalizácii celkovej ceny prepravného a ceny tovaru pri nákupe viacerých položiek od viacerých predajcov, pričom predajcovia môžu predávať tie isté produkty. Výsledkom práce je konzolová aplikácia, ktorá sa prostredníctvom internetu pripája na predajné portály a získava informácie o predajcoch a jednotlivých produktoch za účelom transformácie údajov do modelu úlohy lineárneho programovania. Aplikácia rieši optimalizáciu pomocou existujúcich riešiteľov. V práci analyzujeme aj možnosti použitia iných metód ako sú metódy lineárneho programovania, ktoré však pre riešenie nášho problému nie sú vhodné. Porovnali sme rôzne dostupné riešitele lineárneho programovania so zameraním sa na výkon a optimalizačnú presnosť pri riešení nášho problému a pre praktické použitie sme určili najvhodnejšieho riešiteľa. Aplikáciu je možné v praxi použiť na niekoľko predajných portálov s každým z piatich použitých riešiteľov úloh lineárneho programovania v súlade s ich licenciou.

Kľúčové slová: problém distribuovaného nákupu, optimalizácia, úloha lineárneho programovania, predajné portály

Abstract

The goal of this thesis is to design and implement an application for solving the problem of distributed shopping. Solution has basis in optimizing total price of shipping and the price of products, when buying more items from several sellers, when one product can be sold by more sellers. Result of the thesis is a console application, which connects through the internet to shopping websites and collects information about sellers and individual products for transforming data into a linear programming model. The application solves the optimization with help of already existing solvers. Moreover, we are analyzing other methods than linear programing in this thesis, which however are not usable for solution of this problem. We compared several available solvers of the linear programming, with a focus on speed and precision of the optimization in solving of our problem and we chose most appropriate solver for a practical solution. The application is useable in practice with several shopping websites and with each of the five used linear programming solvers with respect to their license.

Keywords: problem of the distributed shopping, optimization, linear programming, shopping websites

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam zdrojových kódov

Úvod

Cieľom tejto práce je riešiť problém distribuovaného nákupu. V praxi to znamená, že kupujúci chce nakúpiť množinu produktov. Niektoré produkty predáva viacero predávajúcich, každý za inú cenu. Kupujúci má však záujem za množinu produktov zaplatiť čo najmenej. Do celkovej ceny za produkty vstupuje tiež náklad na ich získanie. V prípade kamenných obchodov sú to pre kupujúceho náklady na dopravu pri nákupe produktov a v prípade internetových obchodov je to cena za prepravu tovaru od predajcov ku kupujúcemu.

Internetové portály, napríklad ebay.com, ktoré združujú predajcov, neponúkajú pre kupujúceho možnosti optimalizácie celkovej ceny nákupu viacerých položiek. Naopak, v mnohých prípadoch portály umožňujú predajcom zmenu poradia na portáli pri vyhľadávaní a následnom zoradení výsledkov podľa ceny. Vo výsledkoch takéhoto vyhľadávania sa stáva, že v poradí sa lepšie umiestni tovar, ktorý je za nižšiu cenu ale po pripočítaní poštovného zaplatí kupujúci viac ako za tovar s vyššou cenou ale nižším poštovným.

Významom práce bolo navrhnúť a implementovať algoritmus, ktorý by kupujúcemu optimalizoval nákup na základe minimálnej celkovej ceny a ponúkol rozpis, od ktorého predajcu je výhodné produkty kúpiť. V súčasnosti len veľmi málo známych predajných portálov kupujúcim ponúka takúto službu a preto bolo dôležité algoritmus nielen navrhnúť ale aj implementovať do takého stavu, aby vedel pracovať s reálnymi dátami. Dôležitým prínosom našej práce je overenie funkčnosti riešenia pre optimalizáciu nákupu na nákupnom portáli a porovnanie efektivity nášho riešenia. Naším podnetom na spracovanie danej témy bola skutočnosť, že výsledok práce je možné využívať v reálnom prostredí a kupujúcemu môže priniesť úžitok vo forme ušetrenej sumy finančných prostriedkov. Sekundárnym motívom bolo využitie algoritmu pre vlastné potreby pri nákupoch z internetových obchodov.

Z dôvodu, že práca je zameraná na riešenie nákupu prostredníctvom internetu, základnou podmienkou využívania algoritmu v praxi pri nákupe na internete je, aby bol algoritmus dostatočne rýchly.

Problém sme definovali nasledovne: Máme množinu predajcov, z ktorých každý predáva podmnožinu z požadovaných produktov za nejakú cenu. Aby sme mohli nakúpiť z obchodu nejaký tovar, musíme zaplatiť poštovné, ktoré sa nemení alebo sa mení len

minimálne s ohľadom na počet prepravovaných položiek. Položky sú dostatočne malé, aby sa zmestili do jednej zásielky.

Vstupné dáta pre účely tejto práce sme získali z reálnych zdrojov, napríklad z portálov pre zberateľov malých predmetov ako sú poštové známky, hracie karty a podobne, kde v mnohých prípadoch prebieha sprostredkovanie kúpy, predaja alebo výmeny týchto predmetov. Praktické využitie algoritmu vidíme pri nákupe viacerých predmetov z jednej kategórie, v prípade, že existuje komunita predajcov zastrešená portálovým riešením. Výsledný algoritmus je ľahko upraviteľný aj na väčšiu diverzitu zdrojových údajov.

Práca je členená do štyroch kapitol a niekoľkých podkapitol. V prvej kapitole všeobecne popisujeme súčasný stav optimalizácie nákupu prostredníctvom internetu na najväčších predajných portáloch, uvádzame matematickú definíciu riešeného problému s ukážkou vstupu a výstupu. V druhej kapitole sa zaoberáme spôsobom získania vstupných údajov a návrhom štruktúry dát potrebných pre riešenie problému. V tretej kapitole analyzujeme metódy prístupu k riešeniu z hľadiska optimalizácie algoritmu, výhody a nevýhody zvolených metód a alternatívne postupy riešenia. Zároveň v nej uvádzame výsledky porovnania možných riešiteľov problému distribuovaného nákupu a odporúčame najvhodnejšieho riešiteľa. V štvrtej kapitole popisujeme používateľské rozhranie a dôvody výberu implementovaného používateľského rozhrania pre naše riešenie. Zistenia a výsledky, ku ktorým sme dospeli sumarizujeme v kapitole Záver.

Výstupom práce je aplikácia použiteľná v prostredí viacerých operačných systémov, ktorá umožňuje kupujúcemu optimálne nakúpiť produkty v rôznych množstvách na jednom internetovom portáli. Po zadaní požadovaných položiek kupujúcim aplikácia zozbiera a spracuje údaje potrebné pre optimalizáciu celkovej ceny nákupu. Prostredníctvom riešiteľa nájde najvhodnejších predajcov a výsledky prezentuje kupujúcemu.

Kapitola 1

Súčasný stav a definícia problému

V súčasnej dobe, kedy pribúda počet internetových obchodov sa výrazne mení správanie a návyky spotrebiteľov. Nové technológie prispievajú k tomu, že spotrebitelia majú možnosť jednoduchšou formou ponúknuť svoje už nepotrebné veci. V minulosti túto možnosť zabezpečovali len burzy a bazáre, ktorých sa zúčastňovalo dostatočné množstvo kupujúcich. Tovar tak bolo možné ponúknuť veľkému množstvu záujemcov. Dnes je pre predaj tovaru ideálne prostredie internetu. Agregačné portály umožňujú predávajúcim ponúkať tovar podľa kategórií, ktoré bez problémov kupujúci nájde pri hľadaní žiadaného produktu. Rozšírená oblasť predaja a kúpy prostredníctvom internetu je aj v zberateľstve. Tematické burzy zberateľských predmetov nahradili webové stránky.

Náklady na predaj alebo kúpu sa nezmenili, zmenili len svoju štruktúru. Na sprostredkovanie obchodu bolo v minulosti potrebné dopraviť tovar na burzu čo predstavovalo náklad pre predávajúceho. Náklad pre kupujúceho tvorila doprava na burzu a preprava zakúpeného tovaru domov. V súčasnosti je preprava tovaru realizovaná prepravnými spoločnosťami, ktoré si za svoje služby účtujú poplatky. Tieto poplatky sa premietajú do ceny tovaru. Najmarkantnejšie sa to prejaví pri nákupe množstva rôznych predmetov nízkej ceny od predávajúcich, ktorí sú rozmiestnení vo väčšom regióne, napríklad pri nákupe zberateľských predmetov alebo v predmetov zo zábavného priemyslu.

V tejto kapitole sa zaoberáme problematikou distribuovaného nákupu predmetov zo zábavného priemyslu, konkrétne v oblasti nákupu hracích kariet, kde v súčasnosti nie je žiadne známe univerzálne riešenie optimalizácie nákupu z pohľadu kupujúceho. Uvádzame matematickú definíciu, ktorá formálne popisuje problém.

1.1 Optimalizácia nákupu na predajných portáloch

Predajné portály dokážu nahradiť prostredie burzy a bazáru. Predávajúci ponúkajú nový alebo použitý tovar prostredníctvom webových nespoplatnených stránok, alebo je táto služba spoplatnená až keď dôjde k predaju tovaru, ako poplatok z ceny tovaru. Týmto spôsobom je možnosť predaja prístupná pre širokú verejnosť, čo často spôsobuje, že tovar rovnakého druhu a kvality je dostupný u viacerých predajcov.

Kupujúcemu uľahčuje výber prostredie portálu, kde si výsledky vyhľadávania môže zoraďovať podľa rôznych kritérií, napríklad podľa ceny. Problém nastane vtedy, keď kupujúci má záujem kúpiť viac produktov, ktoré zároveň predávajú viacerí predajcovia. Napríklad sadu zberateľských kariet predáva viac predajcov a každý z nich disponuje kartami z jednej sady, pričom karty sa môžu u jednotlivých predajcov opakovať. V záujme kupujúceho je vybrať karty tak, aby platil čo najmenej. Ak by kúpil viac kariet od jedného predajcu, ušetril by na prepravnom, lebo cena za prepravu častokrát nezávisí od počtu zakúpených kariet. Ako kritériá výberu si kupujúci určí teda cenu každej požadovanej karty, výskyt viacerých požadovaných kariet u jedného predajcu a cenu prepravy.

Pri kúpe malého počtu kariet je tento problém riešiteľný aj priamo užívateľom. V prípade, že kupujúci potrebuje väčšie množstvo kariet, existuje viac predajcov s rôznymi cenami prepravy, stáva sa tento problém ťažko riešiteľný bez výpočtovej sily počítača.

Vyhľadávaním na predajných portáloch sme zistili, že cenovú optimalizáciu nákupu so zohľadnením ceny prepravy, kupujúcemu neumožňuje žiaden zo súčasných najväčších portálov. Do vyhľadávania sme zahrnuli portály amazon.com, ebay.com, alibaba.com, heureka.sk a mtgcardmarket.com.

Pre hracie karty Magic, The Gathering™ sme identifikovali aplikáciu Cart optimizer pre portál www.tcgplayer.com [7]. Jej použitie je do značnej miery komplikované a nepraktické. Po vytvorení niekoľkých nákupov sa nám podarilo zistiť, že jeho výsledky sú značne nepresné, dokonca bez použitia porovnávajúceho algoritmu. Je to jednoúčelová aplikácia, ktorá optimalizuje nákup hracích kariet so zohľadnením poštovného a reputácie predajcov. Popis fungovania algoritmu nie je uvedený a ani nie je dostupný jeho zdrojový kód.

Ďalším vyhľadávaním sme našli pre portál magiccardmarket.eu internetovú stránku magiccardwantlist.wente.dk [5]. Internetová aplikácia ponúka optimalizáciu nákupu hracích kariet Magic, The Gathering™ dvomi spôsobmi. Prvým spôsobom je vyhľadanie všetkých možných alternatív, pri ktorom zadania pre viac kariet môžu trvať veľmi dlho, niekedy aj mnoho rokov. Druhý spôsob je riešenie heuristickou metódou. Bližší popis autor na svojej stránke neuvádza. Aplikácia je však proprietárna a slúži len pre portál magiccardmarket.eu. Rozhodli sme sa, že toto riešenie porovnáme s nami

implementovaným algoritmom.

1.2 Výber portálu pre riešenie

Ako referenčný portál pre praktické použitie riešenia sme zvolili portál magiccardmarket.eu. Dôvodom pre tento výber bola skutočnosť, že ceny predávaných položiek na tomto portály sú veľmi nízke a predajcovia sú rozmiestnení po celej Európe, prepravné je vo väčšine prípadov realizované prostredníctvom pošty a je zrovnateľné s cenou jednotlivých kariet. V mnohých prípadoch je cena karty dokonca nižšia ako cena prepravy. Modelovými situáciami sme si potvrdili, že častokrát je výhodnejšie kúpiť drahšie karty od jedného predajcu ako lacnejšie od viacerých. Ďalším dôvodom pre výber portálu, bolo veľké množstvo predávaných položiek, dobrá štruktúra údajov prístupných pre kupujúceho, najmä jednoznačnosť položiek pri všetkých predávajúcich. Ako sme uviedli v predchádzajúcej kapitole, existuje aj jednoúčelové riešenie problému nákupu kariet práve pre tento portál, ktoré sme zahrnuli do porovnania výsledkov efektivity nášho riešenia.

1.3 Matematická definícia problému

Po identifikácii problému nákupu sme zadefinovali jeho matematickú podobu. Máme množinu n produktov, očíslovaných $J = \{1, 2, 3 \dots n\}$ a množinu m rôznych obchodov, očíslovaných $I = \{1, 2, 3 \dots m\}$. Našim cieľom bolo nakúpiť isté množstvo týchto produktov, dané vektorom $Q = [q_1, q_2, q_3 \dots q_n]$. Obchody sme popísali maticou cien $C \in \mathbb{N}^{m \times n}$, kde $C_{i,j}$ je cena produktu j v obchode i, maticou množstiev produktov na sklade $N \in \mathbb{N}^{m \times n}$, kde $N_{i,j}$ je množstvo dostupného produktu j v obchode i, a funkciami ktoré udávajú poštovné $f_i(x)$, kde x je celkový počet kariet nakupovaných v obchode i.

Maticu reprezentujúcu nákup $R \in \mathbb{N}^{n \times m}$ označujeme ako *platnú*, ak pre každé $j \in J$ platí rovnosť $\sum_{i \in I} R_{i,j} = Q_j$, a pre každé $i \in I, j \in J$ platí nerovnosť $0 \leq R_{i,j} \leq N_{i,j}$.

Potom hľadáme takú platnú maticu nákupu, pre ktorú je cena podľa

$$price(R) = \sum_{i \in I} \left(\sum_{j \in J} R_{i,j} \cdot C_{i,j} + f_i\big(\sum_{j \in J} R_{i,j}\big) \right) \tag{1.1}$$

čo najmenšia možná. V tejto práci sme sa zaoberali len prípadmi, pre ktoré platí, že každá funkcia poštovného je neklesajúca a má diskrétny obor hodnôt, teda vieme túto funkciu popísať sledom intervalov a hodnôt im priradením.

1.4 Príklad vstupu a výstupu

Pre názornosť sme zobrazili vstupné dáta modelovej situácie, kedy kupujúci má záujem kúpiť štyri kusy produktu A a jeden kus produktu B. Výberom z množiny predávajúcich, ktorí ponúkajú produkt A alebo produkt B, dostaneme zoznam troch predávajúcich. Zoznam cien produktov jednotlivých predávajúcich ich početnosť je v tabuľke 1.1. Ceny poštovného v závislosti od počtu naraz zasielaných kusov pre jednotlivých predajcov sú uvedené v tabuľke 1.2.

Tabuľka 1.1: Príkladový vstup: ceny rôznych predajcov

Predajca	Typ produktu	Cena	Počet
1	A	0.6	2
1	B	1.3	2
2	A	0.4	4
2	B	1	1
3	B	0.8	1

Tabuľka 1.2: Príkladový vstup: poštovné

Predajca	Poštovné ≤ 4 kus	Poštovné > 4 kus
1	1.5	4
2	1.7	4
3	1.5	4

Výstupom je nasledovná štuktúra:

Predávajúci 1: 1*A
Predávajúci 2: 3*A; 1*B

Riešením tejto situácie pre kupujúceho je nakúpiť jeden kus produktu A u predávajúceho číslo jeden, a u predávajúceho číslo dva kúpiť tri produkty A a jeden produkt B.

Kapitola 2

Vstupné dáta

V nasledujúcej kapitole popisujeme spôsob, akým sme pristúpili k získaniu údajov pre testovanie a demonštráciu praktickej časti našej práce. Ako zdroj vstupných dát sme mali k dispozícii rôzne internetové obchody, napríklad ebay.com alebo portály pre predaj zberateľských predmetov, napríklad magiccardmarket.eu, ktorý sme sa rozhodli použiť ako najvhodnejší. Cieľom výsledného algoritmu bolo, aby pracoval s údajmi rôznych portálov, preto bola dôležitá transformácia údajov do jednotnej formy. Táto transformácia je iná pre každý portál, nakoľko dáta sú dostupné v rôznej štruktúre a forme. Navrhli sme štruktúru údajov, ktorú je možné naplniť údajmi z väčšiny portálov. V ďalšej časti kapitoly sme popísali princíp parsovania údajov z nami zvoleného portálu.

2.1 Štruktúra dát

Pre demonštráciu algoritmu tejto práce sme vybrali portál pre zberateľské alebo hracie karty magiccardmarket.eu, podľa ktorého sme spracovali štruktúru vstupných údajov. Vybrali sme ho z dôvodu, že ide o jeden z najväčších lokálnych obchodov pre zberateľské hracie karty. Portál združuje predajcov danej komodity v rámci celej Európy. Výhodou tohto portálu je, že väzba predmetu na predajcov je jednoznačne stanovená a je zrejmé, ktorí predajcovia konkrétnu hraciu kartu predávajú. Každá hracia karta má niekoľko vlastností, ktoré nás zaujímali vo vzťahu s riešením nákupu - názov, cena a identifikačné číslo.

Cena prepravného je závislá od lokality a na mnohých portáloch si ju stanovuje predajca sám. V našom prípade portál magiccardmarket.eu určuje jednotnú cenu pre každého predajcu a cena je závislá od štyroch atribútov - krajina predajcu, krajina kupujúceho, hmotnosť zásielky a spôsob zásielky.

To znamená, že dvaja predajcovia z jednej krajiny majú voči kupujúcemu rovnakú cenu prepravy za rovnaký počet kariet. Pre názorný príklad uvádzame na obrázku 2.1 poštovné od predávajúcich z Írska účtované kupujúcemu na Slovensku.

Origin: Ireland; **Destination**: Slovakia

Shipping Method	Certified	Max. Value	Maximum Weigth	Stamp price	Price
Standard Post Letter	No	25,00 €	50 g	1,10 €	1,40 €
Standard Post Large Envelope	No	25,00 €	100 g	2,15 €	2,65 €
Standard Post Packet	No	25,00 €	250 g	5,00 €	5,50 €
Standard Post Packet	No	25,00 €	500 g	6,00 €	6,50 €
Standard Post Packet Playmat	No	25,00 €	500 g	6,00 €	6,50 €
Standard Post Packet	No	25,00 €	1000 g	8,50 €	9,00 €
Standard Post Packet	No	25,00 €	2000 g	11,75 €	12,25 €

Obr. 2.1: Príklad ceny poštovného z Írska na Slovensko (zdroj: https://www.magiccardmarket.eu/Help/Shipping_Costs).

Pre stanovenie hmotnosti zásielky sme použili v algoritme transformáciu na počet kariet podľa hmotnosti štandardnej obálky a jednotlivých kariet. Na základe týchto informácií sme navrhli štruktúru atribútov predajcu pre potreby nákupu nasledovne - meno/názov predajcu, identifikátor, krajina predajcu a počet ponúkaných kariet.

Pre každý typ karty bolo potrebné nastaviť hranice prepravného v súvislosti s počtom kariet, nakoľko hmotnosti kariet sa môžu odlišovať. Preto sme zaviedli vstupnú konštantu práve pre túto veličinu a to spôsobilo, že vo výsledku sú hranice pre poštovné dynamické a sú prepočítavané podľa hmotnosti kariet vždy pri spustení algoritmu a vyhľadaní predajcov, ktorý požadované karty ponúkajú.

Vytvorili sme program - HTML parser, ktorý sa pripája na konkrétny internetový portál - magiccardmarket.eu a importuje údaje potrebné pre uloženie do pripravenej štruktúry. Vstupný parameter pre algoritmus sme zvolili krajinu kupujúceho, v našom prípade Slovensko.

Pre možné použitie algoritmu na iné portály sme transformáciou ceny prepravného na počet kariet eliminovali rozdielnosť účtovania prepravného, ktoré môže byť účtované rôznym spôsobom v závislosti na krajine, hmotnosti, individuálnom účtovaní a podobne.

2.2 Parsovanie dát

Pred transformáciou údajov pre riešenie problému bolo potrebné dáta spracovať z formy v akej sú zobrazené na nákupnom portáli. Portál magiccardmarket.eu neposkytuje verejné API pre sťahovanie údajov a preto jediné riešenie, ktoré prichádzalo do úvahy bolo stiahnuť HTML zdrojové kódy stránky a následne ich vyparsovať. Pre každú jednu požadovanú kartu sme museli stiahnuť zoznam predajcov, ktorí ju predávajú. Tieto zoznamy boli na portáli stránkované, preto pre stiahnutie všetkých údajov bolo potrebné použiť simuláciu AJAX požiadaviek.

Ako nasledujúci krok sme parsovali HTML kód stránky, čo bolo možné implementovať viacerými spôsobmi. Preskúmali sme viac možností a ako najjednoduchšie a najefektívnejšie pre rýchly vývoj sa nám osvedčil skriptovací jazyk Python vo verzii 3. Po preskúmaní možností na parsovanie HTML v tomto jazyku sme vybrali knižnicu BeautifulSoup 4 [18]. Táto Python knižnica na parsovanie údajov z HTML a XML je jedna z najlepších a najviac používaných open source riešení daného problému. Je to vlastne nadstavba nad HTML parser a poskytuje výbornú navigáciu, vyhľadávanie a úpravy v parsovanom súbore údajov.

Z HTML kódu stránky sme potrebovali získať informácie o cene a dodávateľovi pre každú kartu. Štruktúra tabuľky so zoznamom predajcov pre kartu bola rozsiahlejšia. Obsahovala údaje - Počet predaných kariet, predajca, jazyk predajcu, stav karty, extra informácia, poznámka, cena karty a počet dostupných kariet.

Pre potreby algoritmu sme potrebovali len identifikátor predajcu, krajinu kde sa nachádza ponúkaná karta, cenu karty a počet dostupných kariet. Ostatné atribúty neboli pre účely tohto algoritmu potrebné, avšak v budúcnosti bude možné ich zapracovať ako filtre pri volaní AJAX požiadaviek a týmto spôsobom zúžiť výber kariet napríklad podľa ich kvality - na portáli sa predávajú nové aj použité karty.

Príklad jedného riadku zo zoznamu predajcov, z ktorého boli údaje získavané je zdrojový kód 2.1.

Po vyparsovaní zoznamu predajcov pre jednotlivé karty bolo potrebné implementovať získanie tabuľky poštovného. Poštovné sme potrebovali získať pre všetky krajiny, ktoré sa nachádzali v zozname predajcov. Pri parsovaní údajov pre poštovné bolo potrebné vyriešiť niekoľko problémov.

Ako prvý sme riešili spôsob doručenia. Tabuľka poštovného obsahovala viacero možných spôsobov doručenia - listom, prvou triedou, doporučene, balíkom, atď. Štruktúra a názvy typov služieb poštovného boli rôzne, v závislosti od krajiny. Problém sme eliminovali spôsobom, že sme do parseru stiahli všetky služby doručenia a následne sme ich utriedili podľa hmotnosti, ktorú je možné danou službou prepravovať. Následne sme pre každú krajinu vybrali najlacnejšiu službu pre každú hmotnostnú kategóriu. Zdrojový kód 2.2 je príklad štruktúry tabuľky pre poštovné zo stránky magiccardmarket.eu. Údaje boli získané pre poštovné z krajiny Írsko do krajiny Slovensko.

Ako ďalšie sme riešili, že v tabuľke nebol uvedený údaj aký maximálny počet kariet je možné zaslať naraz pomocou uvedenej služby. Museli sme zadefinovať hmotnosť jednej karty a z toho dopočítavať chýbajúci údaj. Stanovili sme X gramov ako hmotnosť karty a všetky údaje sme upravili na základe tejto hodnoty. Hodnotu je možné meniť pri spustení algoritmu, je to požadovaný vstup od používateľa.

V súčasnej implementácii nie je možné kupovať karty pre rôzne kartové hry kvôli rozdielnej hmotnosti kariet, aj keď prevádzkovateľ portálu magiccardmarket.eu prevádzkuje aj portál pre obchod s kartami hier Pokémon a Warcraft. Veľmi jedno-

```
<tr>
   <td class="Seller "><span class="horList nowrap"><span class="
      horListItem" onmouseover="showMsgBox(this,'173 Sales |&
      nbsp;460 Available items')" onmouseout="hideMsgBox()"
      ><span><span class="horListItem-sellCount">173</span></span></
      span><span class="horListItem"><span style="display: inline-block;
      width: 16px; height: 16px; background-image: url('/img/
      d0b251b7b836101af646d7262d20530f/spriteSheets/ssMain.png');
      background-position: -626px -20px;" class="icon" onmouseover="
      showMsgBox(this,'Item location: Italy')" onmouseout="hideMsgBox()"
      ></span></span><span class="horListItem"><a href="/Users/
      war-to-proxy">war-to-proxy</a></span></span></td>
   <td class="Language ">
      <div class="centered"><a href="/Help/Card_Language" class="noborder"
          target="_blank"><span style="display: inline-block; width: 16
         px; height: 16px; background-image: url('/img/95
         b012cb06cf4cb42b20c908e55b3bbe/spriteSheets/ssMain2.png');
         background-position: -144px -80px;" class="icon" onmouseover="
         showMsgBox(this,'Italian')" onmouseout="hideMsgBox()"></span></
         a></div>
   </td>
   <td class="Condition ">
      <div class="centered"><a href="/Help/Card_condition" class="noborder
         " target="_blank"><span style="display: inline-block; width: 16
         px; height: 16px; background-image: url('/img/95
         b012cb06cf4cb42b20c908e55b3bbe/spriteSheets/ssMain2.png');
         background-position: -0px -64px;" class="icon" onmouseover="
         showMsgBox(this,'Mint')" onmouseout="hideMsgBox()"></span></a></
         div>
   </td>
   <td class="Extra nowrap centered">
      <div class="centered vAlignParent"></div>
   </td>
   <td class="Comment ">
      <div>new seller --&gt; + 1300 positive feedback on ebay (
         war_to_proxy)</div>
   </td>
   <td class="Price st_price">
      <div id="price233279069">
         <div class="algn-r nowrap">0,99</div>
      </div>
   </td>
   <td class="Available st_ItemCount centered">1</td>
   <td class="Buy "><span style="display: inline-block; width: 16px;
      height: 16px; background-image: url('/img/95
      b012cb06cf4cb42b20c908e55b3bbe/spriteSheets/ssMain2.png');
      background-position: -64px -16px;" class="icon" onmouseover="
      showMsgBox(this,'You have to be logged in to be able to buy items.
      ')" onmouseout="hideMsgBox()"></span></td>
</tr>
```

Zdrojový kód 2.1: Riadok z tabuľky predajcov z magiccardmarket.eu [13]

duchou úpravou, zmenou URL pre volanie AJAX požiadaviek, je možné nakupovať aj iné karty, pretože štruktúra a zobrazované údaje na všetkých portáloch sú rovnaké. Mnohí predajcovia predávajú karty viacerých typov a pre prípad najoptimálnejšieho nákupu rôznych druhov kariet je nevyhnutné upraviť algoritmus tak, aby zohľadňoval aj rozdielne hmotnosti kariet. Toto však nebolo cieľom našej práce. Parsovací algoritmus je implementovaný tak, že je schopný získať údaje pre portály - www.magiccardmarket.eu, en.yugiohcardmarket.eu, en.pokemoncardmarket.eu, en.wowcardmarket.eu, en.spoilscardmarket.eu.

Údaje získané podľa tohto popisu sme uložili na ďalšie spracovanie. V ďalších krokoch je potrebné ich transformovať do formátu zrozumiteľného pre algoritmus na riešenie problému, čo popisujeme v nasledujúcej kapitole.

```
<table class="MKMTable HelpShippingTable">
   <thead>
      <tr>
         <th>Shipping Method</th>
         <th>Certified</th>
         <th>Max. Value</th>
         <th>Maximum Weight</th>
         <th>Stamp price</th>
         <th>Price</th>
      </tr>
   </thead>
   <tbody>
      <tr>
         <td>Standard Post Letter</td>
         <td>No</td>
         <td>25,00</td>
         <td>50 g</td>
         <td>1,35</td>
         <td>1,65</td>
      </tr>
      <tr>
         <td>Standard Post Large Envelope</td>
         <td>No</td>
         <td>25,00</td>
         <td>100 g</td>
         <td>2,50</td>
         <td>3,00</td>
      </tr>
   </tbody>
</table>
```

Zdrojový kód 2.2: Riadok z poštovného z magiccardmarket.eu [13]

Kapitola 3

Metódy spracovania

V tejto kapitole sme porovnali rôzne metódy spracovania vstupných dát. Zaoberali sme sa efektívnosťou riešenia problému a popísali sme kroky, akým spôsobom sme dospeli k záveru pre výber vhodného riešiteľa. V úvode kapitoly popisujeme výhody a nevýhody použitia algoritmu hrubej sily. V ďalšej časti sa venujeme výberu najlepšieho riešiteľa problému distribuovaného nákupu. V závere kapitoly porovnávame na praktických vstupoch rôzne riešitele úlohy celočíselného programovania a odporúčame konkrétneho riešiteľa pre použitie ako súčasť riešenia.

3.1 Hrubá sila

Jedným z možných riešení tohto problému je riešenie pomocou metódy hrubej sily. Princíp metódy spočíva v systematickom prechádzaní celého priestoru možných riešení problému a hľadaní najlepšieho riešenia, v našom prípade najnižšieho súčtu cien za tovar a prepravu. Predpokladajme, že cena prepravného sa mení vzhľadom na množstvo objednaných produktov minimálne a je hlavne závislá od predávajúceho, od ktorého nakupujeme. V momente ako vyberieme množinu predávajúcich, od ktorých je možné nakúpiť požadovaný tovar, vieme ľahko zistiť, ktoré produkty je najvýhodnejšie nakúpiť od jednotlivých predajcov jednoduchým pažravým algoritmom - greedy algorithm. Princíp algoritmu spočíva v predpoklade, že lokálne optimálna voľba je zároveň globálne optimálnou voľbou [9].

Vyberieme si množinu predajcov. S istotou môžeme tvrdiť, že za prepravu zaplatíme poštovné v presnej výške, a teda si môžeme pažravo dovoliť pre každý žiadaný produkt nájsť obchod s najnižšou cenou a nakúpiť tento produkt výlučne za túto cenu. Uvedené vieme docieliť jednoduchým lineárnym prechodom cez ceny obchodov.

Väčším problémom je vybrať práve takúto množinu obchodov. Metóda hrubej sily je vhodná pre menšiu množinu obchodov, čo znamená prehľadať všetky skupiny obchodov, na každej aplikovať popísaný pažravý algoritmus a porovnaním výsledkov získať

najlepšiu možnú konfiguráciu. Metóda však nemusí zvládnuť veľké množstvá údajov. Výpočet by mohol trvať neprijateľne dlho a pre účely internetového nákupu je táto vlastnosť prekážkou.

Výhodou metódy hrubej sily je, že vždy nájde najlepší možný výsledok. Túto vlastnosť sme využili v neskoršej časti práce na zvýšenie presnosti výsledku. Pretože problém použitia metódy je len vo veľkosti vstupných údajov, tak kedykoľvek v algoritme, keď bolo potrebné vypočítať medzivýsledok na vybranej čiastke vstupných údajov, sme využili pozitívne vlastnosti metódy.

Nakoľko sme predpokladali nemenné poštovné, hrubú silu môžeme použiť len pri nákupe malého množstva produktov. V opačnom prípade výsledok metódy bude neplatný.

3.1.1 Zlepšenie hrubej sily

Ďalšia možnosť, pri ktorej sme použili metódu hrubej sily, bol prípad keď sme jednoznačne vedeli ohraničiť veľkosť vyberanej množiny obchodov. V prípade, že sa algoritmus dostane do časti, kde z danej množiny obchodov nevyberá viac ako n-člennú množinu, prehľadáva len množiny, ktoré majú n alebo menej prvkov. Tento výber nám umožnil výrazné zrýchlenie behu algoritmu. Prípad nastáva vtedy, keď cena za prepravné vo vstupnej množine je vysoká v porovnaní s cenami produktov, a teda výsledok obsahujúci priveľa predajcov je neoptimálny.

Danú skutočnosť využívame tiež pri čiastočných výsledkoch pri používaní optimalizačnej metódy, v prípade ak je množina ohraničená cenou x a algoritmus je schopný vybrať množinu na spracovanie. Takéto cenové ohraničenie vzniká pri optimalizácii hlavne vtedy, ak už je nám známe riešenie alebo medziriešenie, ktoré vo výsledku stojí x a teda všetky riešenia, ktoré stoja viac ako x, nie je potrebné brať do úvahy.

3.2 Riešiteľ problému boolovskej splniteľnosti

Jednou z možností na zlepšenie metódy hrubej sily je použitie riešiteľa problému boolovskej splniteľnosti. Problém splniteľnosti je definovaný ako zistenie, či existuje taká realizácia premenných typu boolean, že všetky podmienky sú pravdivé. V praxi to znamená, že sa pýtame, či sa vstupné podmienky môžu niekedy naplniť.

Riešitele boolovskej splniteľnosti majú dlhú históriu. Implementácia týchto algoritmov je dobre popísaná a existuje množstvo modifikácií v závislosti na tom, aký prístup majú k spotrebe pamäte, alebo aké majú nároky na rýchlosť spracovania.

V našom riešení je možné uvedeného riešiteľa použiť iba v prípade, ak problém nákupu alebo jeho časť je možné prekonvertovať na problém splniteľnosti.

Ako prvé sme stanovili prepravné v určitej výške a vygenerovali sme podmienky tak, aby riešiteľ uvažoval už iba o možnosti, kedy by sme zaplatili za prepravné najviac túto sumu. Ďalej sme postupne aplikovali riešiteľa na vstupy skonštruované s odlišnými hodnotami a na základe výsledkov sme vyhľadávali hodnotu prepravného. Hodnotu sme vyhľadávali binárnym spôsobom, pokiaľ sme nenašli najlepší výsledok.

Nevýhoda metódy bola jej rýchlosť. Aplikáciu riešiteľa bolo potrebné opakovať na veľkých vstupoch a navyše konštrukcia vstupných podmienok bola tiež viacnásobná.

Pri implementácii algoritmu sme zistili, že akékoľvek konvertovanie problému nákupu alebo jeho časti na problém splniteľnosti bolo pre beh algoritmu časovo náročné. Z uvedeného dôvodu sme sa rozhodli použiť iného riešiteľa.

3.3 Úloha celočíselného programovania

Optimalizačné úlohy sa často transformujú na formálny matematický zápis a vzniká takzvaná úloha matematického programovania. Predtým, ako sa zadefinuje presný matematický model, si musíme uvedomiť, čo zahŕňa všeobecná optimalizačná úloha. Každá takáto úloha má optimalizačný ciel a ohraničujúce podmienky - kritérium, ktoré sa snažíme dosiahnuť, a za akých podmienok sa kritérium dá docieliť. Formálne matematické vyjadrenie cieľov a ohraničujúcich podmienok sa nazýva matematický model.

Úloha matematického programovania je ale príliš všeobecná, a vnikli rôzne zúženia, ktoré sa skúmajú a riešia. Jedným z takýchto zúžení je úloha lineárneho programovania (ďalej len ÚLP), ktorou sa v tejto kapitole zaoberáme. Model ÚLP je široko skúmaný a existuje viacero všeobecných riešiteľov, ktoré sa úlohu snažia riešiť čo najrýchlejšie.

3.3.1 Matematický model

Pre ÚLP je cieľ limitovaný na jednu funkciu, ktorá je lineárna - od jej vstupných parametrov závisí len lineárne. Tieto parametre nazývame rozhodovacie premenné $x_1, x_2..., x_n$ a vyjadrujú menlivé parametre úlohy, napríklad v našom prípade počet nakúpených produktov.

Cieľ optimalizácie vieme vyjadriť takzvanou účelovou funkciou $f(x_1, x_2, ..., x_n)$, ktorej hodnotu sa snažíme externalizovať - minimalizovat alebo maximalizovať. Ako sme si už spomenuli, v ÚLP musí byť funkcia f lineárna a teda vieme zapísať účelovú funkciu ako $f(x_1, x_2, ..., x_n) = \sum_i a_i \cdot x_i$, kde a_i sú konštanty.

Rozhodovacie premenné sú vymedzené takzvanými ohraničujúcimi podmienkami. Tieto podmienky vieme vyjadriť sústavou lineárnych rovníc alebo nerovníc reprezentujúcich reálne ohraničenia rozhodovacích premenných [8].

Normalizácia

Model ÚLP vieme výrazne zjednodušiť normalizáciou bez ujmy na vyjadrovacej sile. Obmedzíme všetky rozhodovacie premenné na nezáporné čísla a ohraničujúce podmienky na podmienky tvaru $\sum_i a_i x_i \leq b$, kde a_i, b sú konštanty. Potom vieme ÚLP problém všeobecne zapísať ako

$$max\{c^T x | Ax \leq b, x \geq 0, x \in \mathbb{R}^n\} \tag{3.1}$$

kde c, b sú vektory konštánt a A je matica obsahujúca konštanty [16].

Úloha celočíselného programovania

Pre naše použitie potrebujeme schopnosť tieto premenné a koeficienty obmedziť na celé čísla vzhľadom na to, že nevieme kúpiť iba časť produktu. Keď takto obmedzíme parametre ÚLP, hovoríme o úlohe celočíselného programovania (Ďalej len ÚCP). Tento problém je ťažšie riešiteľný ako ÚLP, keďže patrí do triedy NP-úplných problémov, čo znamená že neexistuje algoritmus, riešiaci túto úlohu v polynomiálnom čase.

Existuje však viacero ÚCP riešiteľov a riešiteľov úloh striktne všeobecnejších, ako je napríklad úloha miešaného celočíselného programovania, ktorá sa podobá na ÚCP, ale niektoré premenné a koeficienty môžu byť aj reálne čísla. Tieto riešitele využívajú viaceré techniky na čo najrýchlejšie spracovanie a riešenie ÚCP, a po dlhej dobe výskumu, viaceré riešitele vedia ÚCP riešiť veľmi efektívne.

3.3.2 Prevedenie problému do ÚCP

Na vyjadrenie problému v ÚCP potrebujeme rozhodovacie premenné, ohraničujúce podmienky a účelovú funkciu. Ako prvé bolo potrebné zadefinovať rozhodovacie premenné a ich význam pre konkrétnu inštanciu úlohy, keďže aj ohraničujúce podmienky aj účelová funkcia závisia na premenných. Podarilo sa nám identifikovať potrebu pre dva diametrálne odlišné významy premenných na popísanie problému. Tieto dva významy sme odlíšili rôznou konvenciou pomenovania, ktoré budeme označovať $x_{i,j}$ ako premenné reprezentujúce položky a $y_{i,k}$ ako premenné reprezentujúce poštovné.

Položky

Položkami sme nazvali premenné v tvare $x_{i,j}$, a vyjadrujú počet kusov nakúpeného tovaru j u predajcu i. Každá takáto premenná je vlastne hodnota z matice reprezentujúcej nákup R, ktorá je popísaná v kapitole 1.3, konkrétne pre každé i, j platí $x_{i,j} = R_{i,j}$. Zvolené rozhodovacie premenné nám umožnili popísať stav nákupu a vyjadriť jeho nutné podmienky. Premenné sme zvolili tak, aby každá popisovala práve jeden pohyblivý stav v našom probléme, čím sme zaručili, že v žiadnej inštancii problému nevytvoríme príliš

veľa premenných, pretože všetkými pohyblivými stavmi v probléme, vieme určite vyjadriť každú možnú situáciu. Nie je prípustné ani žiadnu premennú vynechať, nakoľko jej odstránením by sme dostali veličinu, ktorú nevieme vyjadriť pomocou zvyšných premenných.

Pre všetky položky existujú dve podmienky, ktoré bolo potrebné namodelovať. Prvá podmienka zaručuje, že nie je možné nakúpiť viac položiek ako je predajca ponúka, teda pre všetky i, j platí $x_{i,j} \leq N_{i,j}$ (N je spomínané v kapitole 1.3). Podmienku $0 \leq x_{i,j}$ nezadávame do modelu ÚCP, nakoľko táto podmienka platí pre všetky rozhodovacie premenné ÚCP. Druhá potrebná podmienka ktorá musí platiť, popisuje presne aký celkový počet položiek jednotlivého druhu je požadované nakúpiť. Druhú podmienku sme zapísali v tvare $\sum_i x_{i,j} = Q_j$, čo znamená, že súčet kusov všetkých položiek daného typu je presne rovný požadovanému množstvu.

Poštovné

Ku každému predajcovi ďalej bolo potrebné namodelovať aj poštovné. Konštrukcia poštovného bola ale komplikovanejšia ako konštrukcia položiek. Pre každého predajcu sme zostavili zoznam rôznych metód prepravy, zoradili sme ich podľa ceny vzostupne a očíslovali sme ich od 1 po k. Tieto metódy popisujú správanie sa funkcií poštovného z kapitoly 1.3. Každá metóda má priradený interval $\langle w_{i,k-1}, w_{i,k})$, ktorý popisuje na ktoré množstvá posielaných položiek sa táto metóda vzťahuje a v rámci tohto intervalu má pevnú cenu $f_i(w_{i,k-1})$. ku každej kombinácii predajcu a metódy poštovného sme vyrobili jednu premennú v tvare $y_{i,k}$, kde i je predajca a k je metóda poštovného. Tieto premenné sme limitovali na hodnoty pravda alebo nepravda, v závislosti od toho, či daná metóda je používaná alebo nie. Problémom bolo, že intervaly platnosti metód sú ohraničené z dvoch strán, a na namodelovanie ÚCP sme potrebovali ohraničenie len z jednej strany.

Aby sme dostali intervaly ohraničené len z jednej strany, prepočítali sme intervaly z diskrétnych na kumulatívne, inak povedané namiesto celkovej ceny poštovného sme každej metóde priradili príplatok od predchádzajúcej a primerane sme upravili interval platnosti. Pre každú metódu k sme jej cenu znížili o cenu metódy $k - 1$ a upravili interval platnosti na $\langle w_{k-1}, \infty)$. Takto sme zachovali cenu poštovného požadovaného pre každé možné nakupované množstvo, a zároveň sme zaručili ohraničenie intervalu len na jednej strane. Následne pre takto definované intervaly a premenné už bolo možné, vyrobiť podmienky, ktoré zaručovali, že všetky požiadavky boli vždy splnené.

Keďže sa počet nakupovaných položiek od predajcu i v ohraničujúcich podmienkach vyskytoval viackrát, zaviedli sme označenie $P_i = \sum_j x_{i,j}$. Následne sme vytvorili tieto podmienky pre všetkých predajcov i a poštovných metód k:

- $y_{i,k} \leq 1$, ktorá zaručí, že $y_{i,k}$ bude nadobúdať len hodnoty pravda (1) alebo ne-

pravda (0). Niektoré zápisy modelu ÚCP priamo dovoľujú špecifikovať premennú ako boolovskú, a vtedy je táto podmienka nepotrebná,

- $w_{i,k-1} \cdot y_{i,k} \leq P_i$ popisuje, že metódu poštovného k nepoužívame ak sa počet nakupovaných kariet dá poslať lacnejšou metódou, a teda $y_{i,k}$ nadobúda hodnotu nepravda,

- $M \cdot y_{i,k} + w_{i,k-1} - 1 \geq P_i$, kde M je veľká konštanta (väčšia ako maximálna hodnota P_i) zabezpečuje, že $y_{i,k}$ nadobudne hodnotu pravda, ak nakupujeme položiek viac ako je minimálna hodnota pre metódu poštovného k. Ak platí $wi, k - 1 \geq P_i$, tak $y_{i,k}$ musí určite byť pravda (1), inak ľavá strana bude menšia ako pravá

Účelová funckia

Hlavným cieľom riešenia problému bolo znížiť celkovú cenu nákupu, čo zadefinovalo účelovú funkciu pre model ÚCP. Po zadefinovaný všetkých premenných sme túto účelovú funkciu vedeli vyjadriť presne v závislosti od rozhodovacích premenných. Celková cena sa skladá z ceny nakúpených položiek, a ceny poštovného. Ceny jednotlivých položiek sme v kapitole 1.3 označili ako $C_{i,j}$, a ceny jednotlivých poštovných metód sú hodnoty $f_i(w_{i,k-1})$. Hodnoty sme prenásobili priradenými rozhodovacími premennými aby sme zistili, ktoré položky a poštovné sú v riešení nakupované. Suma týchto členov nám vytvorila vzorec pre účelovú funkciu:

$$\sum_{i,j} C_{i,j} \cdot x_{i,j} + \sum_{i,k} f_i(w_{i,k-1}) \cdot y_{i,k} \tag{3.2}$$

3.4 ÚCP riešitele

Hľadanie ÚCP riešiteľov vhodných pre naše potreby sme realizovali v dvoch jednoduchých fázach. Najskôr sme našli čo najviac renomovaných riešiteľov z rôznych zdrojov, napríklad z porovnania výkonu riešiteľov [10, 15]. Následne sme o každom zistili podrobnejšie informácie a posúdili sme riešitele na základe nasledujúcich kritérií:

- je dostupný pre operačný systém Windows,

- jeho inštalácia nie je komplikovaná,

- je dostupný pre bežného používateľa - open source alebo voľne dostupný.

V tomto procese sme identifikovali niekoľko potenciálnych riešiteľov, ktoré spĺňajú viacero alebo všetky naše kritéria. Niektoré riešitele sme vybrali aj keď nespĺňali jedno z kritérií , hlavne kritérium pre porovnanie rýchlosti s ostatnými.

- **Gurobi** [12] - komerčný ÚCP riešiteľ vytvorený v jazyku C, ktorý je rýchly, robustný a dokáže riešiť lineárne a kvadratické programovanie aj na celých číslach. Gurobi nie je voľne dostupný a jeho cena je pomerne vysoká. Do porovnania riešiteľov sme ho vybrali hlavne kvôli jeho výkonu a možnosti použitia na akademické účely.

- **lp_solve** [4] - open source riešiteľ licencovaný LGPL a vytvorený v ANSI C, ktorý je založený na upravenej simplex metóde a na metóde Branch-and-bound pre celé čísla.

- **Symphony** [17] - je to časť COIN-OR projektu [2], z ktorého využíva Clp ako riešiteľ ÚCP relaxácií. Symphony je jeden z mála riešiteľov, ktorý vie využiť na riešenie problémov viacero vlákien.

- **GLPK** [3] - GNU Linear Programming Kit je open source balík na riešenie ÚCP problémov vytvorený v C. Je to časť GNU Projektu licencovaná GNU licenciou. Zavádza GNU MathProg ako formát špecifikácie problémov, ale vie spracovať aj bežné formáty, ako napríklad formát MPS.

- **SCIP** [14] - je súčasťou SCIP Optimization Suite a je jeden z najrýchlejších nekomerčných riešiteľov [10]. Ako jeden z mála riešiteľov umožňuje používateľovi kontrolovať proces riešenia a poskytuje podrobné informácie počas riešenia. Avšak napriek faktu že je open source, je licencovaný ZIB licenciou, ktorá nepovoľuje iné ako nekomerčné akademické použitie.

3.4.1 Výroba vstupu pre riešiteľe

Existuje niekoľko formátov, ktoré sú schopné riešitele spracovať. Aby sme mohli pre nás vybrať najvhodnejší, bolo potrebné sa s jednotlivými formátmi oboznámiť a analyzovať ich. Do úvahy prichádzali nasledovné formáty:

- API (application programming infterface) - skoro všetky riešitele majú podporu na programovú API na riešenie problémov priamo z programu. Vzniká ale problém, že pre každý podporovaný riešiteľ by sme museli vyrobiť abstrakčnú vrstvu, aby sme ho vedeli správne použiť. Knižnica PuLP [6] pre Python túto abstrakčnú vrstvu poskytuje, avšak len pre vybrané riešitele - GLPK, Symphony, CPLEX a Gurobi. Keďže sme chceli použiť viac riešiteľov pre porovnanie, nebolo možné túto abstrakčnú vrstvu použiť.

- LP formát - podporovaný viacerými riešiteľmi, ale nie všetkými. Má výbornú čitateľnosť a ľahko sa generuje, ale pre naše použitie je nevhodný, keďže ho nepodporujú všetky riešitele. Jeho výhodou je, že popisuje problém presne tak,

ako vzniká. V našom riešení máme údaje v pamäti uložené v podobnej forme ako LP.

- MPS formát - jeden z najstarších formátov. Má pomerne veľa obmedzení, ale práve preto ho všetky riešitele vedia spracovať. Jedným z obmedzení je limit pre mená premenných na osem znakov.
- free MPS formát - vylepšená verzia MPS formátu. Odstraňuje väčšinu obmedzení formátu MPS. Pre potreby nášho algoritmu obmedzenia MPS formátu neboli relevantné. Formát free MPS však nepodporujú všetky nami vybrané riešitele.

Z uvedeného prehľadu vyplýva, že ako vhodný pre použitie v našej práci bol práve MPS formát. Z údajov získaných z predajného portálu a uložených v pamäti, algoritmus vygeneroval súbor vo formáte MPS, ktorý následne slúžil ako vstupný súbor pri spúšťaní jednotlivých riešiteľov.

Popis MPS formátu

MPS formát sa skladá z viacero sekcií:

- NAME - deklaruje meno daného problému.
- ROWS (riadky) - v tejto časti sa deklarujú názvy a porovnávacie znamienka rovníc vyskytujúcich sa v probléme. Jej prvky majú tvar "Z MENO", kde Z je kód pre znamienko. Táto sekcia kóduje aj objektívnu funkciu, v tomto prípade je kód znamienka N.
- COLLUMNS (stĺpce) - deklaruje ako vyzerajú ľavé strany rovníc zadefinovaných v predchádzajúcej sekcii. Definuje premenné a ich koeficienty.
- RHS (pravá strana) - veľmi podobným spôsobom ako v predchádzajúcej sekcii definuje pravú stranu rovníc.
- RANGES (intervaly) - kóduje jednoduché obmedzenia v tvare a <= x <= b.
- BOUNDS (obmedzenia) - kóduje veľmi jednoduché obmedzenia premenných konštantami.

3.4.2 Porovnanie ÚCP riešiteľov

V závislosti na vybratých produktoch, náš model problému distribuovaného nákupu generuje veľké počty premenných a podmienok. Výber riešiteľa pre účely práce sme preto posudzovali hlavne z hľadiska rýchlosti riešenia. Na základe uvedeného sme pre porovnávacie testy pripravili niekoľko vstupov pre všetkých riešiteľov.

Ako testovacie dáta sme vybrali niektoré karty z najčastejšie predávaných kariet na portáli magiccardmarket.eu, konkrétne Polluted Mire, Snuff Out a Bad Moon. Výber najpredávanejších kariet nám zabezpečil dostatočné množstvo podmienok a premenných. Postupne sme pre namodelované prípady získali údaje o predajcoch a poštovnom a pripravili sme súbor v MPS formáte, ktorý sme postupne nechali spracovať všetkými riešiteľmi. Modelové prípady boli nasledovné:

- Vstup číslo 1: 10x Bad Moon - 19584 podmienok a 10625 premenných,
- Vstup číslo 2: 4x Bad moon, 4x Snuff Out - 26959 podmienok a 14745 premenných,
- Vstup číslo 3: 10x Bad moon, 10x Polluted Mire - 29377 podmienok a 16254 premenných,
- Vstup číslo 4: 5x Bad moon, 5x Snuff Out, 5x Polluted Mire - 35020 podmienok a 19508 premenných,
- Vstup číslo 5: 10x Bad moon, 10x Snuff Out, 10x Polluted Mire - 35020 podmienok a 19508 premenných.

Modelové prípady sme riešili vymenovanými riešiteľmi. Beh jednotlivých riešiteľov sme obmedzili časovým limitom 300 sekúnd. V prípade, že riešiteľ nedokázal ukončiť riešenie problému v rámci limitu, do priemerného času sme mu zarátali čas 300 sekúnd. Po tomto limite sme porovnali cenu, ktorú by používateľ zaplatil ako celkovú cenu nedokončeného riešenia s etalónovým výsledkom jedného riešiteľa, ktorý každý zo vstupov vyriešil v časovom limite. Uvedený časový limit bol nastavený na základe používateľského správania, kde je predpoklad, že do tohto limitu je kupujúci ochotný počkať na výsledok, pokiaľ mu to prinesie zodpovedajúcu úsporu.

Gurobi je komerčný riešiteľ, ktorý poskytuje zadarmo verziu pre akademické účely a my sme túto možnosť využili na porovnanie výkonu s voľne použiteľnými alternatívami. Je to najrýchlejší riešiteľ mixovaných celočíselných problémov na trhu [11]. Všetky príkladové problémy Gurobi vyriešil v časom limite s priemerným časom behu 0,54 sekundy. Našiel riešenie rýchlo a efektívne, a teda je vhodným etalónom pre porovnanie výsledkov ostatných riešiteľov.

lp_solve sa ukázal výrazne pomalý. Nevyriešil v časovom limite správne ani jeden z testovacích vstupov. Priemerne výsledky boli o takmer 300 % horšie ako optimálne výsledky riešiteľa Gurobi. To znamená, že pri stanovenom limite behu riešiteľa by kupujúci zaplatil takmer trojnásobok ceny za karty a poštovné.

Na naše účely potrebujeme riešenie nájdené rýchlejšie a lp_solve ani v pomerne dlhom čase nevie nájsť uspokojivé riešenie, preto je nevhodný ako riešiteľ nášho problému. Dokonca pri reálnom nákupe nie je možné použiť optimalizované výsledky. Keďže ani jeden vstup nedobehol v časovom limite riešiteľa sme neuviedli ani v grafe 3.1. Jeho jedinou výhodou je, že je veľmi ľahko nainštalovateľný a bol by vhodný pre ľahkú distribúciu celého riešenia.

Symphony má výhodu v tom, že je schopný bežať vo viacerých vláknach naraz, a teda lepšie využiť väčšinu procesorov počítača používateľa. V súčasnosti má väčšina počítačov viac jadier, často štyri až osem logických jadier, čo môže byť aj nevýhodné. Počas výpočtu je plná výpočtová sila hostiteľského systému zaneprázdnená, čo spôsobuje výrazne zhoršený výkon iných procesov a aplikácií. Pri dlhých výpočtoch sa stáva systém skoro nepoužiteľný.

Symphony mal priemernú dĺžku výpočtu 39,33 sekúnd a všetky vstupy boli vyriešené v časovom limite. Presnosť je zhodná s Gurobi. Symphony mal pomalšie časy na jednoduhších vstupoch ale na zložitejších vstupoch (číslo 4 a číslo 5) výrazne prekonal aj GLPK. V sumáre na našich testovaných vstupoch sa umiestnil najlepšie z nekomerčných riešiteľov.

SCIP Optimization Suite je balíček optimalizačných utilít, z ktorých sme využili SCIP s nalinkovaným SoPlex ako riešiteľom miešaných celočíselných problémov. Jeho výkon sa ukázal ako výborný, v časovom limite spracoval všetky vstupy.

Celkovo na jeden vstup potreboval v priemere 9,62 sekundy a všetky výsledky boli optimálne. Vo verzii nalinkovanej na SoPlex je aj ľahko dostupný a jeho vysoký výkon je takmer neporovnateľný s ostatnými nekomerčnými riešiteľmi, preto je ideálny na použitie. SCIP bol výrazne rýchlejší na zložitejších vstupoch, na jednoduchších boli časy trochu pomalšie ako časy GLPK riešiteľa. V sumáre sa umiestnil na druhom mieste. Využitiu v našom riešení však bráni licencia riešiteľa. SCIP je licencovaný ZIB licenciou, ktorá povoľuje použitie riešiteľa len členom akademickej a nekomerčnej inštitúcie a väčšina používateľov nemá k nemu prístup.

GLPK je open source projekt z GNU vytvorený v ANSI C. Je to časť GNU projektu a vyžíva viaceré známe techniky na riešenie problémov. Priemerne na jeden vstup potreboval 68,54 sekundy, pričom vyriešil všetky vstupy v časovom limite.

GLPK je dostupný, skompilovaný pre viaceré operačné systémy ako aj pomerne presný. GLPK riešiteľ bol výrazne rýchlejší od Symphony na vstupoch číslo 1,2 a 3, ale na vstupoch 4 a 5 bol jeho čas hraničný z hodnotou 150 a 180 sekúnd. Pre účely nákupu sú však tieto hodnoty prijateľné.

Modelový prípad nákupu zberateľa

Pre potvrdenie výsledkov sme zadefinovali modelový prípad pre zberateľa predmetov. Tento modelový prípad vychádzal z potreby zberateľa kariet doplniť svoju zbierku. Pripravili sme vstup pre nákup šiestich rôznych kariet, z každej karty po jednom kuse. Vstupné údaje scenára boli nasledovné:

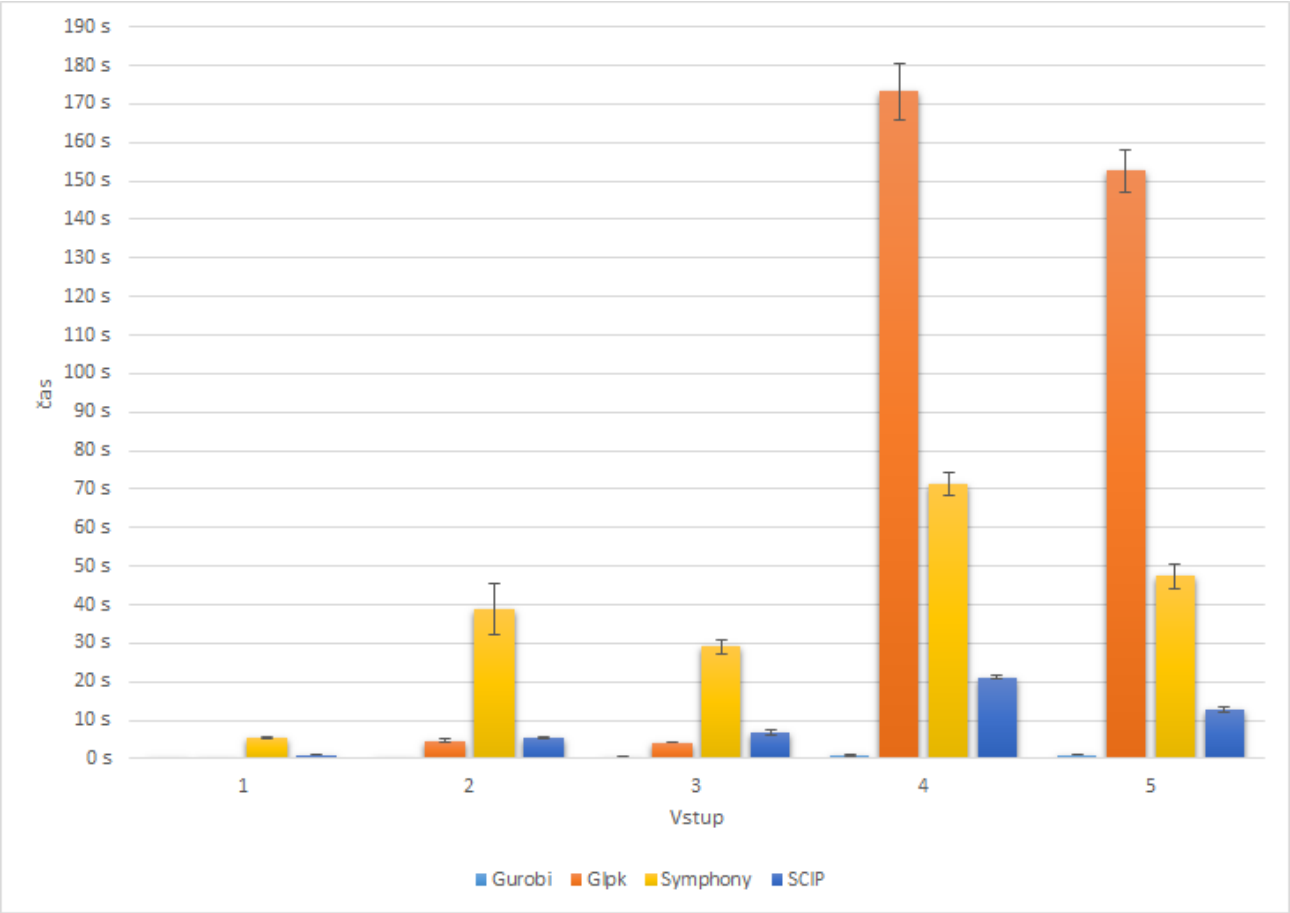
1x Eternal Witness, 1x Yavimaya Coast, 1x Beseech the Queen, 1x Hill Giant, 1x Pegasus Refuge, 1x Starlit Angel: 70274 podmienok a 39826 premennych.

Portál magiccardwantlist.wenke.dk poskytol svojou heuristikou výsledok 8,54 €. Náš etalónový riešiteľ Gurobi dokázal nájsť riešenie 6,37 €. Výsledok existujúcej heuristiky bol o viac 34 % horší. V porovnaní rýchlostí riešiteľov, tak ako v predchádzajúcom porovnaní priemerných časov bol najrýchlejší Gurobi, keď na vyriešenie tohto vstupu bolo potrebných 0,84 sekundy. Druhý najrýchlejší bol SCIP s časom 6,93 sekundy. Ostatné riešitele v limite 300 sekúnd nedokázali nájsť optimálne riešenie. Najmenšiu odchýlku od etalónového výsledku Gurobi zaznamenal riešiteľ Symphony vo výške 9,9 %. Na štvrtom mieste sa umiestnil GLPK riešiteľ, ktorý mal odchýlku 14,9 %. Na poslednom mieste bol Lp_solve, ktorý v časovom limite 300 sekúnd bol schopný poskytnúť optimalizovaný výsledok s odchýlkou 133,3 %.

Pre modelový prípad nákupu zberateľa sa potvrdilo, že Symphony je z voľne dostupných riešiteľov najvýkonnejší, nedokázal síce vyriešiť vstup v časovom limite ale odchýlka od etalónového výsledku Gurobi bola najnižšia a zároveň bola o takmer 25 % nižšia ako odchýlka existujúceho webového riešenia.

3.5 Výsledky

Testovali sme päť vstupov uvedených v predchádzajúcej kapitole pomocou každého riešiteľa. V priebehu testovania boli vstupy optimalizované každým riešiteľom 21 krát. Na základe týchto testov sme spočítali medián a štandardnú odchýlku pre každého riešiteľa a vstup. Výsledky sú zobrazené v grafe 3.1. Na začiatku testovania sme vygenerovali podmienky a premenné pre všetky vstupy a uložili sme ich v pamäti, aby sme predišli zmene údajov na portáli magiccardmarket.wenke.eu. Obraz údajov sme pripravili dňa 13.5.2017 a súčasne sme pomocou rýchlejšej heuristickej metódy optimalizovali naše vstupy na portáli magiccardwantlist.wente.dk. Tento portál pracuje s obrazom údajov, ktorý je aktualizovaný v intervaloch, alebo na vyžiadanie používateľa, preto porovnanie výsledkov bolo spracované pre rovnaké údaje. V vstupe číslo 1 bol výsledok optimalizácie rovnaký, výsledná cena za karty a prepravu bola zhodná. Vo všetkých ostatných vstupoch ÚCP riešitele našli optimálnejší nákup ako heuristická metóda použitá na stránke magiccardwantlist.wente.dk. Výsledky metódy hrubej sily, ktorá je na tomto portáli tiež implementovaná nebolo možné zahrnúť do porovnania,

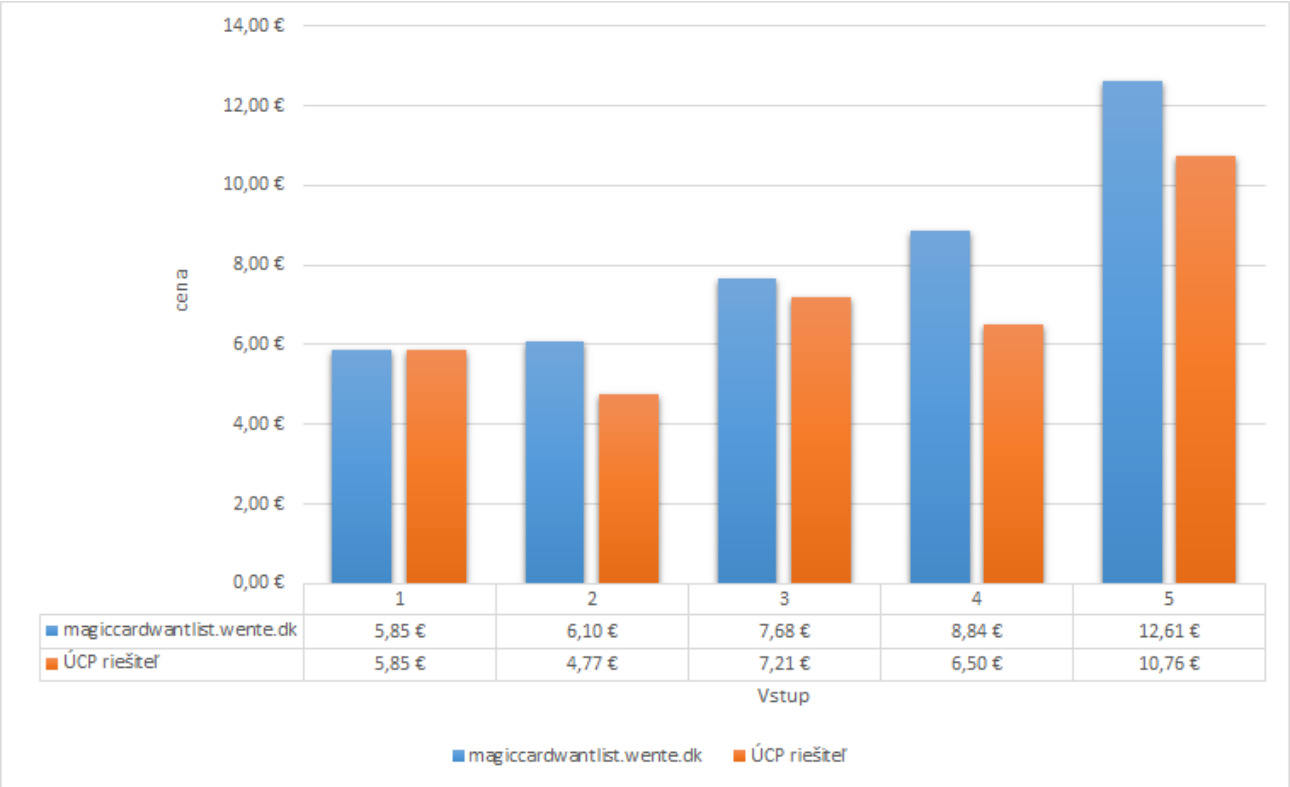


Obr. 3.1: Čas behu riešiteľov

pretože čas behu na nami zadaných vstupoch bol príliš veľký. Výsledky porovnania optimalizácie ceny sú vyobrazené v grafe 3.2.

Naša aplikácia podporuje všetky testované riešitele, no ako súčasť celkového riešenia a vzhľadom na komplexnosť inštalácie riešiteľov bolo cieľom našej práce vybrať jedeného konkrétneho riešiteľa, ktorého sme chceli odporučiť pre použitie s našou aplikáciou.

Z uvedených výsledkov je zrejmé, že Gurobi je neporovnateľne rýchlejší ako nekomerčné riešitele, ale z dôvodov spomínaných v kapitole 3.4.2 ho používateľom nemôžeme odporučiť. Ako druhý najlepší sa ukázal SCIP ale jeho ZIB licencia tiež limituje prístupnosť pre bežného používateľa. Ako najpomalší a s najväčšou odchýlkou sa umiestnil Lp_solve. V porovnaní rýchlosti sa najlepšie z nekomerčných riešiteľov umiestnil Symphony. Druhý najlepší nekomerčný bol GLPK riešiteľ. Rozdiel medzi nimi nebol však vôbec veľký. Pre rôzne testované scenáre bolo poradie takmer zhodné, v niektorých prípadoch bol dokonca GLPK riešiteľ rýchlejší ako Symphony. Zvažovali sme implementovať do výsledného riešenia oba riešitele a na základe vstupu rozhodnúť automaticky napríklad podľa počtu podmienok a premenných, ktorého riešiteľa aplikácia použije. Kvôli skutočnosti, že pre používateľa je náročnejšie inštalovať oba



Obr. 3.2: Porovnanie výslednej ceny existujúceho riešenia a IP riešiteľov

spomínané riešitele rozhodli sme sa implementovať v aplikácii vstupný parameter, ktorý určí riešiteľa už pri spustení aplikácie. Popis akým spôsobom je uvedený v nasledujúcej kapitole. Ako ďalší rozvoj tejto práce vnímame možnosť pomocou heuristiky vybrať jeden z vhodnejších riešiteľov a tak získať výsledky podstatne rýchlejšie.

Symphony je teda najvýkonnejší riešiteľ z nekomerčne prístupných riešiteľov testovaných našou aplikáciou. Ako ukázali výsledky testov nami modelovaných prípadov, zďaleka nedosahuje výkonnosť komerčných riešiteľov. Na základe testov medzi nekomerčnými riešiteľmi však dosiahol v našom testovacom prostredí najlepšie výsledky. Inštalácia Symphony je jednoduchá aj pre menej skúseného používateľa, nakoľko predkompilované binárne súbory pre Windows, Linux a OS X sú ľahko dostupné.

Kapitola 4

Používateľské rozhranie

Táto časť práce sa zaoberá popisom používateľského rozhrania. Zdôvodňuje výber používateľského rozhrania pre vytvorené riešenie. Ide o časť programu, ktorá je prioritne zodpovedná za prehľadnosť a jednoduchosť zadávania vstupných údajov a zobrazenie výstupných údajov v našom prípade výsledkov.

Z hľadiska prevádzky riešenia sme zvažovali dve možnosti ako implementovať používateľské rozhranie:

- Prvou možnosťou bolo webové rozhranie - webová aplikácia, kde by mal používateľ možnosť cez rozhranie internetového prehliadača zadať úlohy vzdialenému serveru na nájdenie najlepšieho nákupného scenára. Server sa pripojí na portál s burzou predmetov, nájde optimálne riešenie a výsledok zobrazí v grafickej podobe.

- Druhou možnosťou bolo vytvorenie natívnej aplikácie - desktopová aplikácia. Nevýhoda prístupu použitia aplikácie spočíva hlavne v tom, že je určená pre jeden konkrétny operačný systém. V tomto prípade vznikajú problémy s použitými technológiami alebo knižnicami, ktoré musia byť prístupné pre správny beh programu. Ak riešenie obsahuje časti, ktoré je zložitejšie nainštalovať na cieľovú platformu, do značnej miery sa komplikuje inštalačný proces a zvládnu ho len skúsenejší používatelia.

4.1 Výber spôsobu prevádzky

Pri výbere spôsobu prevádzky bol jedným z hlavných faktorov rozmer zložitosti riešenia najlepšieho nákupného scenára a hlavne jeho časová náročnosť. V prípade webovej aplikácie by riešenie nákupných scenárov viacerými používateľmi naraz mohlo zaťažiť zdroje servera natoľko, že by mohlo dôjsť k preťaženiu. Konfigurácia servera by musela byť pomerne náročná na zdroje - pamäť, jadrá procesoru a prenájom takéhoto

servera by bol finančne náročný. Na uvedené problémy existuje riešenie, napríklad obmedzenie času behu programu na úkor presnosti optimalizačného výsledku, prípadne implementácia fronty na úlohy a následného zaslania informácie o výsledku používateľovi notifikačným e-mailom.

Naopak, pri natívnej aplikácii je možné prezentovať priebeh vyhľadávania nákupného scenára spolu s čiastkovými výsledkami, prípadne umožniť používateľovi vstúpiť do priebehu výpočtu a zastaviť ho podľa požadovanej kvality optimalizácie. V tomto prípade, na súčasných počítačoch je možné využiť všetky jadrá procesora a viacvláknovo riešiť daný problém, musí to však umožňovať riešiteľ a všetky jeho súčasti. V prípade menej výkonných počítačov je tu riziko, že výpočet bude trvať príliš dlho.

Pri tvorbe natívnej aplikácie je potrebné zvážiť, kto je primárny odberateľ a používateľ riešenia a prevažne aký operačný systém preferuje. V prípade tejto práce odberateľ a operačný systém nebol definovaný, preto sme pri zvažovaní výberu operačného systému zobrali do úvahy možnosť vytvoriť riešenie pre viac platforiem. Umožnilo nám to použiť skriptovací jazyk Python a vytvoriť v ňom zdrojový kód. Výber operačného systému pre spúšťanie riešenia je závislý od interpretera jazyka Python a podporovaných operačných systémov odporúčaného riešiteľa. Riešiteľa Symphony je možné spustiť na nasledujúcich OS:

- GNU/Linux (gcc, 32 a 64 bit),
- Microsoft Windows,
- CYGWIN (w/ gcc and cl compilers, 32 a 64 bit),
- MinGW (w/ gcc and cl compilers, 32 a 64 bit),
- Visual C++,
- Mac OSX (gcc and clang).

Pri navrhovaní riešenia sa ukázalo, že dôležitejšie je zachovať možnosť spúšťania aplikácie na viacerých platformách ako využitie špecifických vlastností OS a preto sme sa rozhodli použiť Python interpreter. Python interpreter vo verzii 3 je spustiteľný na operačných systémoch Microsoft Windows, Linux, MacOSX a iných.

4.2 Používateľské rozhranie

Ako používateľské prostredie sme zvažovali použiť buď grafické rozhranie alebo príkazový riadok v konzole. Vzhľadom na všetky skutočnosti popísané v kapitole 4.1 sme zvolili ako používateľské rozhranie príkazový riadok - CLI (command line interface), pri ktorom používateľ komunikuje s programom prostredníctvom textových príkazov. Ako

textové príkazy sú prenášané jednotlivé parametre - zoznam kariet. Python v súčasnej dobe disponuje viacerými knižnicami pre tvorbu grafického používateľského rozhrania, ale implementácia grafického rozhrania nebolo cieľom našej práce.

Popis CLI rozhrania

Pre potreby behu algoritmu je nutné zadať určité vstupné parametre. Parametre, ktoré je možné odovzdať programu sú nasledovné:

- pozičné argumenty v tvare: "počet karta"je potrebné špecifikovať, pre ktoré karty požadujeme nájsť optimálne riešenie nákupu. Tento argument je možné opakovať pre špecifikovanie ďalších kariet. V prípade, že názov karty má viac slov, slová musia byť spojené v úvodzovkách.

  Príklad:

  ```
  >python optimizer.py 4 "Bad Moon" 2 "Snuff Out"
  ```

- -s, --solver {riešiteľ}: umožňuje zvoliť aký riešiteľ je použitý na výpočet. Ako prípustné hodnoty je možné uviesť gurobi, scip, lp_solve, symphony, glpk. Ak nebude zadaný parameter {riešiteľ}, východzia hodnota je symphony.

  Príklad:

  ```
  >python optimizer.py --solver gurobi 4 "Bad Moon" 2 "Snuff Out"
  ```

- -t, --timelimit {číslo}: beh riešiteľa bude obmedzený na {číslo} sekúnd.

  Príklad:

  ```
  >python optimizer.py --timelimit 60 4 "Bad Moon" 2 "Snuff Out"
  ```

- -c, --country {country}: krajina, do ktorej bude tovar dopravovaný, kde {country} je oficiálna skratka krajiny (napríklad UK pre Veľkú Britániu). Pokiaľ parameter nie je zadaný, východisková hodnota je SK (Slovensko).

  Príklad:

  ```
  >python optimizer.py --country UK 4 "Bad Moon" 2 "Snuff Out"
  ```

- -f {file}: načíta vstupné karty zo súboru namiesto z pozičných argumentov

  Príklad:

  ```
  >python optimizer.py -f vstup.txt
  ```

  Príklad vstupného súboru:

  ```
  1 4 Bad Moon
  2 2 Snuff Out
  3 3 Plague Belcher
  ```

- -wmps {file}: zapíše vygenerovaný MPS problém do súboru, slúži na debugovacie účely.

  Príklad:

  ```
  >python optimizer.py -wmps out.mps 4 "Bad Moon" 2 "Snuff Out"
  ```

Výhodou CLI rozhrania v našej aplikácii bola rýchlosť jeho implementácie. Nevýhoda tohto rozhrania je, že používateľ musí poznať štruktúru parametrov vstupujúcich do aplikácie. Vstupných parametrov nie je veľa, z tohto dôvodu osvojenie si formátu zápisu parametrov nie je pre používateľa časovo náročné.

Záver

V našej práci sme sa zaoberali riešením optimalizačného problému distribuovaného nákupu produktov, v prípade keď poštovné je nezanedbateľnou čiastkou výslednej ceny a zároveň aplikáciou tohto problému pre praktické využitie. Cieľom práce bolo naštudovať a spracovať danú oblasť, a tiež implementovať prakticky relevantné riešenie. Pri overovaní možných postupov riešenia daného problému sme zistili, že problematika je značne neriešená a pre bežného používateľa neexistujú prakticky využiteľné riešenia až na pár existujúcich jednoúčelových riešení. Portál pre testovanie nášho riešenia sme vybrali magiccardmarket.eu práve kvôli tomu, aby sme mohli porovnať výsledky nášho riešenia s existujúcim.

V snahe nájsť čo najefektívnejšie riešenie sme preskúmali viacero možných spôsobov riešenia problematiky. Pri použití metód spracovania - hrubá sila a riešenie problému boolovskej splniteľnosti sme zistili že nie sú dostatočne efektívne alebo praktické na hľadanie riešenia. Ako ďalšiu možnosť sme použili zadefinovanie a konverziu problému na ÚCP, a jej následné vyriešenie ÚCP riešiteľmi. Pri hľadaní dostatočne rýchleho riešiteľa sme porovnali niekoľko dostupných riešiteľov. Riešitele sme porovnali na základe troch vlastností: rýchlosť, licencia pre použitie, jednoduchosť inštalácie. Na základe výsledkov porovnania sme určili ktorý riešiteľ je optimálne použitie pri našom riešení.

Výsledkom práce je vytvorenie aplikácie, ktorá na reálnych dátach optimalizuje kupujúcemu celkovú cenu nákupu. Aplikácia je zložená z dvoch základných modulov: modul na zber údajov a ich transformáciu pre riešiteľa a riešiteľ problému.

Zberač dát, ktorý sme implementovali na komunikáciu s internetovým portálom www.magiccardmarket.eu, získava z portálu informácie a pripravuje ich na použitie do druhej časti aplikácie s cieľom zistenia optimálneho riešenia. Zberač dát umožňuje prácu s prakticky využiteľnými dátami, a výsledok formátuje používateľovi do zrozumiteľnej podoby. Druhý modul aplikácie transformuje spracované dáta na úlohu lineárneho programovania a odovzdá ju ÚCP riešiteľovi na vyriešenie. Táto časť aplikácie ďalej vyberie z výstupu riešiteľa dáta popisujúce riešenie a vráti ich zberaču dát na zobrazenie výsledku používateľovi.

V závere môžeme skonštatovať, že zadané ciele práce sa nám podarilo naplniť. Problém distribuovaného nákupu sme naštudovali a popísali jeho matematický model. Identifikovali sme miesto praktického použitia a vyrobili aplikáciu, ktorá z internetu získa

reálne údaje podľa zadania používateľa a vyrieši optimalizáciu celkovej nákupnej ceny pomocou ÚCP riešiteľa. Ako sa ukázalo porovnaním výsledkov, náš program, ktorý je v využiteľný aj na iné typy portálov, bol efektívnejší ako existujúce proprietárne riešenie.

Výsledky práce je možné rozšíriť o možnosť komunikovať s viacerými internetovými portálmi a získať viacero vstupných zdrojov, čo umožní optimalizovať nákup aj iných produktov ako zberateľských kariet. Ideálnym výsledkom by bola možnosť praktického využitia aj pri nákupoch cez agregačné portály alebo cez portály, ktoré sprostredkovávajú obchod s tovarom rôzneho druhu. Ďalším rozšírením funkcionality aplikácie by mohlo byť pridanie možnosti spracovania komplexnejších vstupov, napríklad nákup len u predávajúcich s dobrou reputáciou, zohľadnenie stavu tovaru - kvality, prípadne rýchlosti dodávky tovaru. Výsledok práce je možné upraviť aj o schopnosť modelovať aj iné charakteristické funkcie poštovného.

Komfort ovládania aplikácie používateľom je možné zvýšiť implementáciou grafického rozhrania, ktoré by používateľovi uľahčilo zadávanie a vyhľadávanie zvolených produktov a požadovaného množstva. Grafické rozhranie by zvýšilo prehľadnosť výsledkov ich logickým priestorovým umiestnením. Aplikácia by zároveň výsledky optimálneho nákupu priamo objednala na obchodnom portáli bez nutnosti manuálneho zadania nákupu používateľom.

Literatúra

[1] Clp: Coin-or linear programming.
URL <https://projects.coin-or.org/Clp>

[2] COIN-OR: Computational Infrastructure for Operations Research.
URL <https://www.coin-or.org/>

[3] GLPK (GNU Linear Programming Kit).
URL <https://www.gnu.org/software/glpk/>

[4] lp_solve reference guide.
URL <http://lpsolve.sourceforge.net/5.5/>

[5] The Magic Wantlist Optimizer.
URL <http://magiccardwantlist.wente.dk/>

[6] PuLP documentation.
URL <http://pythonhosted.org/PuLP/>

[7] TCG Player store for playing cards.
URL <http://www.tcgplayer.com/>

[8] Berežný, Š.; Hajduová, Z.; Kravecová, D.: *Úvod do lineárneho programovania: vysokoškolská učebnica*. Oficyna Wydawnicza Wyższa Szkoła Humanitas, 2013.

[9] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: Introduction to algorithms second edition. 2001.

[10] Gearhart, J. L.; Adair, K. L.; Detry, R.; aj.: Comparison of open-source linear programming solvers. *Tech. Rep. SAND2013-8847*, 2013.

[11] Gurobi, O.: Gurobi Benchmark.
URL <http://www.gurobi.com/pdfs/benchmarks.pdf>

[12] Gurobi, O.: Gurobi Optimization - The Best Mathematical Programming Solver.
URL <http://www.gurobi.com/>

[13] KG, S. L. . C.: Europe's largest online marketplace for Magic the Gathering cards.
URL <https://www.magiccardmarket.eu/>

[14] Maher, S. J.; Fischer, T.; Gally, T.; aj.: *The SCIP Optimization Suite 4.0*. 17-12, Takustr.7, 14195 Berlin, 2017.

[15] Meindl, B.; Templ, M.: Analysis of commercial and free and open source solvers for linear optimization problems. *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, 2012.

[16] Press, C.: *Linear and Integer Programming: Theory and Practice, Second Edition*. Advances in Applied Mathematics, CRC Press, 2001, ISBN 9781482294712.
URL <https://books.google.sk/books?id=Sd3MBQAAQBAJ>

[17] Ralphs, T.; Mahajan, A.; Vigerske, S.; aj.: coin-or/SYMPHONY: Version 5.6.16. Leden 2017, doi:10.5281/zenodo.248734.
URL <https://doi.org/10.5281/zenodo.248734>

[18] Richardson, L.: Beautiful soup documentation. 2007.
URL <https://media.readthedocs.org/pdf/beautiful-soup-4/latest/beautiful-soup-4.pdf>

Príloha
Zdrojový kód aplikácie

Príloha obsahuje zdrojový kód aplikácie na optimalizáciu distribuovaného nákupu, tak ako je popísaná v kapitole 4.

Súbory zdrojového kódu sú uložené na priloženom CD v archíve pod názvom **optimizer.zip**. V tomto archíve sa nachádzajú aj súbory **README.md** a **CITAJMA.md** pre ľahšiu orientáciu použíteľa, ktoré obsahujú aj zoznam potrebných externých knižníc a aplikácii potrebných pre beh aplikácie.