

Web-aplikace pro řešení problému lineárního programování

Web-application Solving the Linear Programming Problem

Bc. Tomáš Hampl



Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2012/2013

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Tomáš Hampl**
Osobní číslo: **A11518**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Automatické řízení a informatika**
Forma studia: **kombinovaná**

Téma práce: **Web-aplikace pro řešení problému lineárního programování**

Zásady pro vypracování:

1. Důkladně se seznámte se speciální metodou optimalizace Lineární programování a vypracujte literární rešerši na uvedené téma.
2. Navrhněte a realizujte interaktivní webovou aplikaci, která umožní vyřešit alespoň základní úlohu lineárního programování metodou simplexové tabulky.
3. Po zadání vstupních dat uživatelem aplikace poskytně informace o řešení zadaného problému a pokud toto existuje, dovoří ho najít jednorázově nebo s možností zobrazovat jednotlivé kroky řešení.
4. Průběh i výsledek řešení bude možno zobrazit v přehledné výstupní sestavě s možností uložit ji také ve formátu PDF.
5. Vytvořenou aplikaci důkladně otestujte a na vybraných problémech demonstруйте správnost řešení.
6. Věnujte dostatečnou pozornost zabezpečení aplikace.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:

1. KOŘENÁŘ, Václav a Milada LAGOVÁ. Optimalizační metody. Praha: Oeconomica, 2003. ISBN 80-245-0609-2.
2. LINDA, Bohdan; VOLEK, Josef. Lineární programování. Pardubice: Univerzita Pardubice, 2009. ISBN 978-80-7395-207-5.
3. SVRČEK, Jaroslav. Lineární programování v úlohách. Olomouc: Univerzita Palackého v Olomouci, 2003. ISBN 80-244-0705-1.
4. HÁJEK, Michal. Soubor úloh ke cvičení do předmětu Optimalizace – lineární a dynamické programování. Zlín, 2009. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky.
5. KOSEK, Jiří. PHP: tvorba interaktivních internetových aplikací. Podrobný průvodce. Praha: Grada, 1999. ISBN 8071693731.

Vedoucí diplomové práce:

doc. Ing. František Gazdoš, Ph.D.

Ústav řízení procesů

Datum zadání diplomové práce:

24. února 2013

Termín odevzdání diplomové práce:

11. června 2013

Ve Zlíně dne 24. února 2013

prof. Ing. Vladimír Vašek, CSc.
děkan



prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Tato práce se zabývá speciální metodou optimalizace Lineární programování a vytvořením webové aplikace řešící tyto úlohy. Práce je rozdělena do dvou částí.

Teoretická část je zaměřena na vysvětlení vzniku, využití a definici úlohy lineárního programování. Dále jsou popsány postupy pro sestavení matematického modelu, jeho převodu na základní tvar a podrobněji vysvětleny nejčastěji používané metody řešení těchto typů úloh.

Praktická část zahrnuje analýzu požadavků, návržení a implementaci webové aplikace, která umožňuje řešení úloh Lineárního programování. Nedílnou součástí je také popis zabezpečení aplikace a výpočet vzorových slovních úloh pro demonstraci aplikace.

Klíčová slova: Optimalizace, Lineární programování, Simplexová metoda, webová aplikace, Java.

ABSTRACT

This thesis deals with a special method of optimization of linear programming and creating the web application solving these tasks. The thesis is divided into two parts.

The theoretical part is focused on explaining the creation, use and definition of linear programming. Furthermore there are described procedures for establishing mathematical model, it's transfer to the basic form and the most often used methods to solve these types of problems are explained in detail.

The practical part includes requirements analysis, proposal and implementation of the web application that allows solving of linear programming tasks. The description of the application security and calculation of sample problem tasks to demonstrate the application are inseparable parts of this thesis.

Keywords: Optimization, linear programming, simplex method, web application, Java.

Na tomto místě bych rád poděkoval především vedoucímu diplomové práce panu doc. Ing. Františku Gazdošovi, Ph.D. za metodickou, odbornou pomoc, ochotu, cenné názory a věcné připomínky. Také děkuji rodině, přítelkyni a zejména otci za podporu při studiu a vytvoření patřičného zázemí.

Motto: „Nejllepším možným způsobem“

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	9
I TEORETICKÁ ČÁST.....	10
1 OPTIMALIZACE	11
1.1 MOŽNOSTI ŘEŠENÍ OPTIMALIZAČNÍCH PROBLÉMŮ	11
1.2 HISTORIE.....	12
1.3 OPERAČNÍ VÝZKUM.....	14
1.4 OPTIMALIZAČNÍ ÚLOHA.....	15
1.4.1 Základní typy optimalizačních metod	16
1.4.2 Omezující podmínky	16
1.4.3 Účelová funkce.....	17
1.4.4 Cíl optimalizace	18
1.4.5 Typy řešení.....	18
2 LINEÁRNÍ PROGRAMOVÁNÍ.....	19
2.1 TYPICKÉ ÚLOHY LINEÁRNÍHO PROGRAMOVÁNÍ	19
2.2 KONSTRUKCE MODELU LINEÁRNÍHO PROGRAMOVÁNÍ.....	19
2.3 OBECNÝ TVAR ÚLOHY LINEÁRNÍHO PROGRAMOVÁNÍ	20
2.4 TVARY ÚLOH LINEÁRNÍHO PROGRAMOVÁNÍ	21
2.4.1 Kanonický tvar	21
2.4.2 Symetrický tvar	21
2.4.3 Smíšený tvar.....	21
2.4.4 Převod mezi tvary	22
2.5 TYPY ŘEŠENÍ ÚLOH LINEÁRNÍHO PROGRAMOVÁNÍ.....	23
2.6 BÁZICKY PŘÍPUSTNÁ ŘEŠENÍ ÚLOHY LINEÁRNÍHO PROGRAMOVÁNÍ	24
2.6.1 Sloupcový vektor	24
2.6.2 Bázické řešení	24
2.6.3 Přípustné bázické řešení	24
2.7 GEOMETRICKÝ POHLED NA LINEÁRNÍ PROGRAMOVÁNÍ	25
2.7.1 Bod n-rozměrného reálného prostoru.....	25
2.7.2 Konvexní kombinace bodů	25
2.7.3 Další kombinace bodů.....	25
2.7.4 Geometrické vyjádření kombinací	26
2.7.5 Konvexní množina	26
2.8 GRAFICKÉ ŘEŠENÍ (ŘEŠENÍ DVOUROZMĚRNÝCH ÚLOH)	27
2.9 SIMPLEXOVÁ METODA.....	28
2.9.1 Sestavení simplexové tabulky	29
2.9.2 Zjištění optimálního řešení.....	30
2.9.3 Volba klíčového prvku	31
2.9.4 Transformace řešení	32
2.9.5 Výpočet hodnot proměnných	33
2.9.6 Porušení podmínky nezápornosti	34

2.10	PODMÍNKA CELOČÍSELNOSTI	35
2.11	DUÁLNÍ ÚLOHA	35
2.11.1	Duální neboli stínové ceny	36
2.12	NEDOČERPÁNÍ OMEZENÍ	36
2.13	CITLIVOSTNÍ ANALÝZA	37
II	PRAKTICKÁ ČÁST	38
3	ANALÝZA POŽADAVKŮ	39
3.1	CÍLE PRAKTICKÉ ČÁSTI	39
3.2	FUNKČNÍ POŽADAVKY	39
3.3	NEFUNKČNÍ POŽADAVKY	40
3.4	PŘÍPADY UŽITÍ	40
4	NÁVRH ŘEŠENÍ	44
4.1	ARCHITEKTURA APLIKACE	44
4.2	KLIENSKÁ ČÁST	44
4.3	JAVA PLATFORMA	44
4.4	SERVEROVÁ ČÁST	45
4.4.1	Aplikační server	46
4.4.2	Propojení aplikačního a webového serveru	46
4.4.3	JSF framework	47
4.4.4	Pretty Faces	48
4.5	ALGORITMUS ŘEŠENÍ ÚLOHY	48
4.6	BEZPEČNOST	49
5	IMPLEMENTACE	50
5.1	APACHE MAVEN	50
5.2	IMPLEMENTAČNÍ ČÁSTI A JEJICH TESTOVÁNÍ	51
5.3	ALGORITMY LINEÁRNÍHO PROGRAMOVÁNÍ	51
5.4	ZABEZPEČENÍ APLIKACE PROTI ÚTOKŮM	54
5.5	UŽIVATELSKÉ ROZHRAŇÍ	56
6	PŘÍKLADY VYBRANÝCH PROBLÉMŮ	60
6.1	PŘÍKLAD Č.1 – PIVOVAR V USA	60
6.2	PŘÍKLAD Č. 2 – DOPRAVNÍ PROBLÉM	61
6.3	PŘÍKLAD Č. 3 – PROBLÉM NUTRIČNÍCH HODNOT	63
	ZÁVĚR	65
	ZÁVĚR V ANGLIČTINĚ	CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.
	SEZNAM POUŽITÉ LITERATURY	69
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	71
	SEZNAM OBRÁZKŮ	72
	SEZNAM TABULEK	73
	SEZNAM PŘÍLOH	74

ÚVOD

Lidé se den co den musí rozhodovat a řešit různé situace a tím aniž by si to uvědomovali, hledají optimální rozhodnutí. Tato rozhodnutí nalezneme také sami u sebe, např. při určení důležitostí jednotlivých úkolů v zaměstnání nebo také při nakupování potřebných věcí do domácnosti, vždy hledáme nejvýhodnější řešení.

Hledání optimálního řešení využívá dnes mnoho oblastí, jako je světové tržní prostředí, kde v posledních letech dochází ke změnám díky převyšující nabídce nad poptávkou a tím je nutné řešit výrobní kapacity či produktivitu. Čím více se otvírají světové trhy, zvyšují se i konkurenční boje mezi firmami. Tyto firmy můžeme zařadit buď do cenové, nebo necenové konkurence v závislosti na jejich podnikové politice. Je proto velký tlak, aby se podniky snažily vyrábět co nejefektivněji a tím docílily co nejlepšího postavení na trhu, ale také co nejvyššího zisku. Při vedení společnosti můžeme řešit sdružené problémy pomocí lineárního programování a tím najít optimální cestku k efektivnějšímu řízení podniku.

Další oblastí, kde najdeme využití optimálního rozhodování je ekonomika. Ekonomické rozhodování není nejjednodušší úkol a musíme vždy brát v potaz, jaká fakta máme a co vypovídají. Máme proto několik způsobů, jak můžeme řešit daný problém. První způsob je intuitivní a využíváme jej na základně získaných zkušeností. Avšak někdy nám mohou potřebné zkušenosti chybět a rozhodnutí mohou mít fatální následky. Druhý způsob můžeme nazvat jako metodu optimalizace, kterou provádíme prostřednictvím matematického modelování. Přestože tato metoda není dokonalá, výsledky bývají velmi blízké skutečnosti a nedochází u ní k velkým omylům, jejichž dopady by představovaly hrozbu pro podnik, jako při chybě založené na špatném intuitivním rozhodnutí. Poslední způsob rozhodování je sloučení obou předchozích způsobů rozhodování, kde zjistíme pomocí metody optimalizace potřebné podklady pro směr rozhodování a zároveň jej upravíme dle získaných zkušeností.

Náplní diplomové práce je aplikace teoretických poznatků zabývajících se problematikou lineárního programování. Práce je rozdělena na dvě části a to teoretickou, která pojednává o úlohách lineárního programování a jejich řešení za pomoci simplexové metody, a na část praktickou která se zabývá návrhem a implementací webové aplikace umožňující řešení těchto úloh.

I. TEORETICKÁ ČÁST

1 OPTIMALIZACE

1.1 Možnosti řešení optimalizačních problémů

V současné době je možné řešit optimalizační úlohy komfortně za použití výpočetní techniky a patřičného softwaru. Takový software je možné buď zakoupit, případně stáhnout zdarma, nainstalovat a po spuštění používat, nebo je možné využít webových aplikací, které danou výpočetní metodu podporují. Webové aplikace jsou v dnešní době velice moderní, neboť nejsou závislé na platformě ani operačním systému a je možné je použít i na chytrých telefonech nebo vestavných zařízeních, které již dnes běžně mají instalován prohlížeč webových stránek. Práce je zaměřena na metody lineárního programování, proto i zmíněné aplikace budou porovnávány zejména na tomto typu úloh.

V matematických aplikacích jako je Mathematica[12], Matlab[13], Maple[14] a také v tabulkovém kalkulátoru Excel je možné řešit úlohy lineárního programování. Tyto aplikace jsou často finančně nákladné a vyžadují určitou znalost uživatele v jejich prostředí. Obvykle neobsahují tabulkový vstup pro úlohy lineárního programování a vestavěné či doinstalované funkce pro výpočet metod lineárního programování vyžadují přesně definovaný vstup, který je obvykle ve formátu matic. Nejsou tedy nejvhodnější k rychlému použití, nebo v případě kdy uživatel chce ověřit svůj ručně vypočtený postup, například při studiu optimalizačních metod. Pro zadávání vstupních dat ve formátu matic musí uživatel znát alespoň přepis obecného tvaru do maticového formátu, což často zahrnuje i nutné úpravy a je to tedy část, kde může uživatel udělat chybu.

Dále je k dispozici řada programů, které umožňují výpočet problémů lineárního programování a to včetně pohodlného zadávání proměnných, jako je například placený software LINDO (Linear INteractive and Discrete Optimizer)[15], který v nejvyšší verzi dokáže řešit úlohy s několika desítkami tisíc omezujících podmínek. Další jsou neplacené aplikace Linear Program Solver (LiPS)[16] a Lpsolver[17]. První zmíněná umožňuje 3 způsoby zadávání dat a také citlivostní analýzu, uživatelské prostředí je přívětivé a intuitivní. Aplikace Lpsolver je spíše jednoduchý projekt, který je naprogramován v jazyce Java, což zaručuje jeho nezávislost na využívané platformě, ale jeho grafické prostředí není příliš přívětivé a funkce se omezují pouze na výpočet a zobrazení mezikroků.

Existují však také webové aplikace pro řešení úloh lineárního programování, což je především aplikace Simplex method tool[18], za kterou stojí profesor Stefan Waner. Ta umožňuje pouze textové zadávání. Výsledky je možné zobrazit v desetinných, zlomcích, či řešit úlohu celočíselně. Jednotlivé mezikroky jsou zobrazeny textově. Mezi další patří webová aplikace Simplex Me[19] a je společnou prací autorů Alexandra Altenhofena, Stephana Diedericha a Michaela Schäfermeyera, která umožňuje zadání tabulkou či textem (ze souboru) a zobrazený výpočet je možné stáhnout ve formě PDF, nebo odeslat na email.

Další webovou aplikací je Simplex On Line Calculator[20] portálu Mathtools, která umožňuje přehledné zadání do tabulky a animované zobrazování jednotlivých kroků výpočtu. Toto zobrazování je velice vhodné pro naučení a pochopení výpočtu problému lineárního programování pomocí simplexové metody. Tato aplikace existuje i ve verzi pro mobilní telefony se systémem android, je však již placená.

V případě, že uživateli postačí pouze omezený počet proměnných a podmínek, nabízí webové aplikace velice rychlý způsob, jak vyřešit konkrétní problém, případně slouží pro ověřování správnosti ručního výpočtu. Pro problémy s velkým množstvím omezení a proměnných je vhodné přiklonit se ke specializovanému softwaru, který je na tuto problematiku přímo určen.

1.2 Historie

Z monografie P. Pavlíkové [1], kde je přehledně zpracována historie lineárního programování lze vyčíst fakta, která následují v této kapitole.

Hledáním optimálního řešení se začalo lidstvo zabývat ve druhé polovině 18. století v oblastech vědy jako je geodezie, kartografie (tj. umění, věda a technologie vytváření map), astronomie, ale především v teoretické mechanice. Dále se více rozvíjelo ve druhé polovině 19. století, kde tzv. optimalizační úlohy začaly pronikat do mnoha dalších oblastí zejména do matematiky a to konkrétně do teorie pravděpodobnosti, teorii čísel, teorii řešení soustav rovnic a nerovnic, teorii grafů, teorii her apod.

Co se týče osobností v historii, první myšlenky spojené s lineárním programováním byly zaznamenány u francouzského matematika a fyzika Jeana Baptista Josepha Fouriera (1768 – 1830). Fourier se zabýval otázkami teoretické mechaniky související s prací virtuálních sil a podmínek rovnováhy, což ho zavedlo k řešení problému soustav nerovnic. Ve svých spisech zřetelně poukazoval na možnost, že polyedrická množina

v R^3 představuje množinu přípustných řešení soustavy lineárních nerovnic o třech neznámých.

Další osobností byl Carl Friedrich Gauss (1777 – 1855), slavný německý matematik a fyzik, který se taktéž zabýval soustavou lineárních rovnic, po něm pojmenovaná Gaussova eliminační metoda což je zvláštní druh úlohy lineárního programování.

Na přelomu 19. a 20. století přispěl svou prací další významný matematik maďarský Gyula Farkáš (1847 – 1930), který navázal na práci Jeana Baptista Josepha Fouriera a vytvořil tak teorii řešení soustav lineárních nerovnic a to z pohledu mechanického, který během let změnil na plně matematický. Při studiu konvexních množin v rámci geometrické teorie čísel vynikl matematik německo-polsko-židovského původu Hermann Minkowski (1864 – 1909), který první neaplikoval jako základní prvek své teorie geometrickou tvorbu lineárních nerovnic. Na toto poukázal v roce 1917 další z řady maďarských matematiků Alfred Haar (1895 – 1933), který aplikoval teorii konvexních množin jako primární princip pro řešení takovýchto problémů.

Za největší úspěch do roku 1935 je považována disertační práce izraelsko-amerického matematika Theodora Samuela Motzkina (1908 – 1970) Beiträge zur Theorie der linearen Ungleichungen. Tato práce se zabývá řešením soustav lineárních nerovnic a Motzkin v ní shrnul a popsal veškeré existující publikované příspěvky zabývající se lineárními nerovnicemi a proto je brána jako velký zlom v historii tohoto oboru. Tato práce pro svou současně analytickou a geometrickou stránku byla významnou inspirací pro následující rozvoj oboru po válce.

Až v roce 1939 vzniklo první dílo orientované na lineární optimalizaci nazvané Matematické metody plánování a organizace výroby. Jeho autorem je ruský matematik Leonid Vitalevič Kantorovič (1912 – 1986), bohužel tato práce byla doceněna daleko později než v předválečném období, kdy se tato práce měla stát klíčem pro plánování a organizaci výroby. I přesto, že dosáhla uznání mnohem později, byla tato práce prvním impulsem v oblasti lineárního programování, kdy náhlý vzestup tohoto oboru zapříčinila druhá světová válka a následná nutnost oprav válkou poničených ekonomik.

Tento obor začal nabývat na důležitosti v momentě, kdy prvně americká a britská armáda vynaložila značné investice do operačního výzkumu. V historii byli i další významné události spojené s touto problematikou, jako je kombinatorická optimalizace přiřazovacího problému, kterou představili v roce 1931 matematici z Maďarska Denes König

(1884 – 1944) a Eugene Egerváry (1891 – 1958). Dále koncept postupu řešení dopravní záležitosti konstruoval o deset let později americký matematik a fyzik Frank Lauren Hitchcock. Další klíčové dílo *Theory of Games and Economic Behavior*, které napsali maďarský matematik John von Neumann (1903 – 1957) a rakouský ekonom Oskar Morgenstern (1902 – 1977), bylo zveřejněno v roce 1944 a značně přispělo k dalšímu vývoji lineárního programování.

Velice důležitým okamžikem byl rok 1947, kdy americký matematický vědec jménem George Bernard Dantzig (1914 – 2005) předložil světu významnou Simplexovou metodu, která je schopna obecně řešit příklady lineárního programování. Prvotní účel spočíval ve využití této metody u amerického letectva a nadále tato metoda byla testovaná na různých obtížných úlohách. Jednou z prvních byl problém nutričních hodnot, ve kterém měla metoda řešit sporné úseky a jehož výsledkem by byl takový jídelníček, který bude mít dostatečné množství předepsaných látek, ale přitom bychom se stravovali co nejlevněji. Tato metoda zrekapitulovala dosud známé důležité poznatky a zároveň byla předvedena v nové formě s nově připojenou účelovou funkcí – Jordanova modifikace Gaussovy eliminační metody.

I posléze se vyvíjela nová řešení úloh lineárního programování a vznikly asymptoticky rychlejší algoritmy. První se jmenuje Elipsoidova metoda, kterou zveřejnil roku 1979 sovětský matematik Leonid Chačijan (1952 – 2005), dalším novým algoritmem přispěl známý indický matematik Narendra Karmarkar (nar. 1957), který roku 1984 představil Metodu vnitřních bodů.

I z českých řad se do historie zapsal významný matematik František Nožička (1918 – 2004), jehož přínos byl v oblastech lineárního a dynamického programování a v teorii optimalizačních procesů.

1.3 Operační výzkum

Název operační výzkum vznikl z projektu probíhajícího během Druhé Světové války, z anglického názvu „Research in Military Operations“. Cílem tohoto projektu bylo snížit náklady a zvýšit účinnost vojenských operací. Jednotlivá zadání byla převedena na matematické úlohy obsahující matematické modely. Výsledky těchto úloh byly následně interpretovány na původní problém. Tyto metody však později našly uplatnění v komerčním sektoru pro plánování a optimalizaci. Použití lineárního programování

Lze také spatřit v řízení firem a v mikroekonomii, neboť firmy se snaží v rámci svých zdrojů minimalizovat náklady a maximalizovat zisky [2].

Operační výzkum tedy řeší problémy z mnoha odvětví za použití různých druhů metod, které jsou typické pro jednotlivá odvětví. Pro přehled lze uvést některé z metod operačního výzkumu:

- Teorie front řeší problematiku požadavků na službu a její kapacitu. V důsledku časového nesouladu vznikají u služby fronty. Požadavky mohou přicházet zcela nebo částečně náhodně a také doba obsluhy může být částečně časově náhodná.
- Teorie zásob, dnes zahrnutá do oboru logistika, řeší problémy zásobování tak, aby docházelo k minimálním nákladům spojených na jejich řízení a pořízení, ale současně nesmí dojít ke zpoždění, tedy je třeba, aby zásoby byly ve správný čas, na správném místě a za minimální náklady. Tato metoda se dnes označuje jako „Just in time“.
- Teorie rozvrhování, v současnosti často označována jako teorie řízení projektů, se zabývá přidělením jednotlivých dílčích činností, tak aby celkový projekt byl dokončen včas a veškeré činnosti na sebe navazovaly.
- Matematické programování je část operačního výzkumu, ve které se často používá matematický model úlohy. Protože se jedná o úlohu hledání lokálního extrému, zmíněný model je nazýván optimalizačním modelem, přičemž může být různých typů v závislosti na zadané úloze. V případě, že úloha obsahuje více kritérií, je model více kritériální. Pokud je kritériální funkce nebo nějaká další rovnice v úloze nelineární jedná se o úlohu nelineárního, programování a v opačném případě se jedná o úlohu lineárního programování, kterou se tato práce bude podrobněji zabývat [3].

1.4 Optimalizační úloha

Základem optimalizačních úloh je nalezení nejlepšího přípustného řešení problému, to spočívá v nalezení množiny bodů nejlépe vyhovující zadaným podmínkám, tedy nalezení řešení, které existuje mezi všemi *přípustnými řešeními* [4].

Pro řešení optimalizační úlohy je nutné nejprve definovat problém, následně jej popsat a nakonec sestavit matematický model této úlohy.

1.4.1 Základní typy optimalizačních metod

Iterační metody jsou založeny na principu postupného přibližování k extrému funkce ve stále dokola se opakujících krocích. Rozhodnutí o ukončení opakování výpočtu může být provedeno na základě několika kritérií, jako jsou dosažení určité přesnosti nebo překročení limitu počtu kroků. Takovýto výpočet ale může „uvíznout“ v lokálním extrému funkce a výsledek poté není optimálním řešením. Pokud iterační algoritmus sám o sobě není schopen se z lokálního extrému dostat, je možné výpočet opakovat několikrát z různých počátečních bodů prohledávaného prostoru.

Analytické metody využívají klasických nástrojů matematiky a jejího vyšetřování průběhu funkce. U jednorozměrných případů se využívá první a druhá derivace pro určení inflexních bodů a následně lokálních extrémů funkce. V případě více rozměrných úloh se určuje extrém pomocí parciálních derivací gradientu funkce.

Substituční metody využívají tvaru některých úloh vícerozměrné funkce, kdy je možné zjistit závislost mezi proměnnými (rozměry) a tuto závislost popsat funkcí. Díky tomu je možné snížit rozměr úlohy. Pokud se podaří úlohu zjednodušit na jednorozměrnou úlohu, lze ji řešit jinou metodou [4].

1.4.2 Omezující podmínky

Množina *přípustných řešení* není obvykle známa přímo, ale pomocí množiny, kterou nazýváme *prostor řešení* a jejích omezujících podmínek. Ta část prostoru řešení, splňující všechny omezující podmínky najednou, je právě *množina přípustných řešení*. Tyto podmínky je možné zapsat v maticové formě jako $\bar{A} * \bar{x} = \bar{b}$, kde \bar{b} je sloupcový vektor omezení, \bar{x} je sloupcový vektor proměnných a \bar{A} je matice koeficientů proměnných [2]. Obvykle se ale zapisují pomocí obecných lineárních nerovnic ve formátu proměnných x ve tvaru:

$$\begin{aligned} a_{1,1} * x_1 + a_{1,2} * x_2 + \dots a_{1,n} * x_n &\leq b_1 \\ a_{2,1} * x_1 + a_{2,2} * x_2 + \dots a_{2,n} * x_n &\leq b_2 \\ &\vdots \\ a_{m,1} * x_1 + a_{m,2} * x_2 + \dots a_{m,n} * x_n &\leq b_m \end{aligned} \quad (1)$$

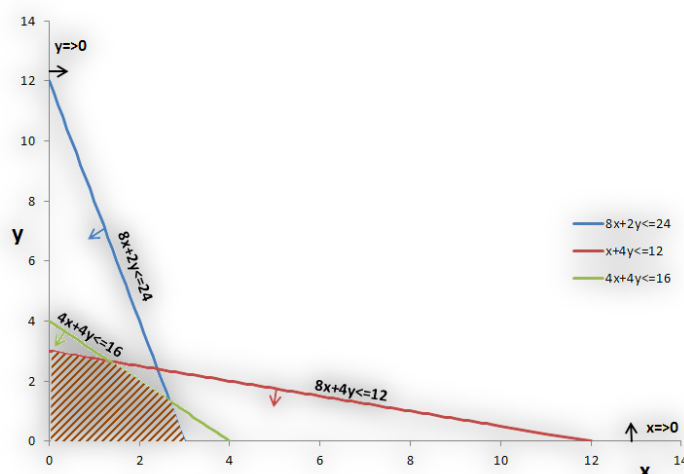
kde místo operátoru \leq může být také \geq nebo $=$ [2].

Tento typ podmínek se označuje jako *strukturní omezující podmínky*. Dalším typem podmínek jsou *podmínky pro jednotlivé proměnné*. To je obvykle podmínka nezápornosti, která se zapisuje ve tvaru nerovnic:

$$x_1, x_2, \dots, x_n \geq 0 \quad (2)$$

Pokud úloha obsahuje pouze 2 proměnné, prostor řešení je dvourozměrný a omezující podmínky je možné zobrazit graficky, což je obvykle snadnější pochopitelné. Například Obrázek 1 zobrazuje oblast přípustných řešení nerovnice (3), je to tedy průnik omezujících podmínek.

$$\begin{aligned} 8x + 2y &\leq 24 \\ x + 4y &\leq 12 \\ 4x + 4y &\leq 16 \\ x &\geq 0 \\ y &\geq 0 \end{aligned} \quad (3)$$



Obrázek 1 – Grafická reprezentace oblasti přípustných řešení

1.4.3 Účelová funkce

Účelová funkce, někdy také označována jako cenový funkcionál, je funkce, jejíž extrém hledáme. Tato funkce rozhoduje které řešení z množiny *přípustných řešení* je horší a které lepší. Účelová funkce je zobrazení přiřazující každému řešení z množiny *přípustných řešení* hodnotu, tzv. *hodnota účelové funkce* [2].

1.4.4 Cíl optimalizace

Cílem optimalizace může být maximalizace nebo minimalizace. Optimální řešení je takové, které má maximální nebo minimální (dle cíle úlohy) *hodnotu účelové funkce*. Takovýchto řešení může být i více za předpokladu, že jsou z množiny přípustných řešení a mají stejnou hodnotu účelové funkce.

1.4.5 Typy řešení

- Řešení, které vyhovuje podmínkám nezápornosti, je přípustné řešení soustavy lineárních rovnic.
- Řešení, které maximalizuje či minimalizuje cenový funkcionál, je řešení optimální.
- Řešení, které má stejný nebo menší počet kladných složek než počet lineárně nezávislých rovnic tvořících vlastní omezení, je základní řešení.
- Řešení, které má více než $n - m$ nulových složek, je degradované základní řešení [5].

2 LINEÁRNÍ PROGRAMOVÁNÍ

Lineární programování, často označováno zkratkou LP nebo LPP z anglického „linear programming“ nebo „linear programming problem“, je matematická metoda optimalizace pro stanovení způsobu, jak dosáhnout nejlepšího výsledku v matematickém modelu s více požadavky (omezeními), které mají lineární vztah.

2.1 Typické úlohy lineárního programování

Typické úlohy operačního výzkumu řešené pomocí lineárního programování, jsou úlohy:

- tok sítí při omezených propustnostech přepravních linek (network flow problems) a
- úloha nutričního problému (the diet problem).

Problém toku sítí při omezených propustnostech

Tato úloha může být kupříkladu využita k nalezení optimálních cest v silničním systému, k optimální přepravě tekutin v potrubí a prakticky ke všemu co cestuje skrze nějakou síť.

Problém nutričních hodnot

V tomto modelu existuje více omezujících podmínek a cílem je najít optimální hodnotu, jako například sestavení jídelníčku, kde jsou na výběr různá jídla, za různou cenu, s různým množstvím vitamínů, minerálů a dalších živin. Optimálním řešením může být jídelníček, který splní týdenní množství vitamínů, minerálů i dalších živin a má nejnižší možnou cenu. Tento model je zejména vhodný na pochopení úlohy lineárního programování pro laiky, a proto se často používá jako úvodní problém v předmětu operační výzkum nebo optimalizace [6].

2.2 Konstrukce modelu lineárního programování

Konstrukce modelu úlohy je klíčová část řešení úlohy pomocí lineárního programování. Chybný model úlohy samozřejmě dává i chybný výsledek, neboť model neodpovídá zadání úlohy. Pro sestavení modelu existuje následující postup a několik zásad, jejichž dodržování vede k předčasnému odstranění chyb:

1. Před formulací modelu matematickým zápisem je vhodné definovat problém a popsat jeho podstatné rysy verbálně, neboli sestavit *verbální model*, který obsahuje:
 - a. cíl optimalizace (např. minimalizace nákladů),

- b. podmínky (vstupní např. zásoby, výstupní např. maximální možný odbyt),
 - c. procesy, které chceme optimalizovat, (např. výroba konkrétního výrobku)
2. K veškerým koeficientům musí být přiřazeny správné jednotky a stanovena kvanta mezi procesy a podmínkami. Takový vztah může být kupříkladu norma spotřeby látky v cm na 1 ks výrobku.
3. Sestavit slovní popis omezujících podmínek a přepsat tento popis na soustavu rovnic a nerovnic.
4. Převést verbální model do matematického zápisu jak podrobněji znázorňuje Tabulka 1 a to tak, že každému procesu je přiřazena jedna proměnná a podmínka nezápornosti, každé podmínce je přiřazeno vlastní omezení ve tvaru nerovnice nebo rovnice a cíl modelu se vyjádří účelovou funkcí.
5. Zkontrolovat obě strany omezujících podmínek, jsou-li vyjádřeny ve stejných jednotkách.
6. Provést kontrolu výsledku a zamyslet se, zda opravdu modeluje to, co se modelovat má [2][3].

Tabulka 1 – Vztah matematického a verbálního modelu

Verbální model		Matematický model
Proces	→	Proměnná
Podmínka	→	Vlastní omezení
Cíl	→	Účelová funkce

2.3 Obecný tvar úlohy lineárního programování

Jak již bylo zmíněno v kapitole 1.4.3, úloha obsahuje účelovou funkci. Ta je ve tvaru:

$$L(x) = c_1x_1 + c_2x_2 \cdots c_nx_n \quad (4)$$

kde c_1, c_2, \dots, c_n jsou koeficienty účelové funkce, x_1, \dots, x_n proměnné a n je velikosti prostoru řešení.

Úloha dále obsahuje cíl optimalizace popsáný v kapitole 1.4.4, který se zapisuje ve tvaru:

maximize: $L(x)$, $\max L(x)$ nebo $L(x) \rightarrow \max$ pro maximalizaci účelové funkce, nebo

minimize: $L(x)$, $\min L(x)$ nebo $L(x) \rightarrow \min$ pro minimalizaci účelové funkce.

Poslední částí úlohy lineárního programování jsou omezující podmínky podrobněji popsane v kapitole 1.4.2.

Celé úloha lineárního programování má poté obecný tvar:

$$\begin{aligned} \max L(x) &= c_1 x_1 + c_2 x_2 \cdots c_n x_n \\ a_{1,1} * x_1 + a_{1,2} * x_2 + \cdots a_{1,n} * x_n &\leq b_1 \\ a_{2,1} * x_1 + a_{2,2} * x_2 + \cdots a_{2,n} * x_n &\leq b_2 \\ &\vdots \\ a_{m,1} * x_1 + a_{m,2} * x_2 + \cdots a_{m,n} * x_n &\leq b_m \\ x_1, x_2, \cdots, x_n &\geq 0 \end{aligned} \quad (5)$$

2.4 Tvary úloh Lineárního programování

Dle tvaru omezujících podmínek je možné kategorizovat tvary úloh lineárního programování.

2.4.1 Kanonický tvar

Za kanonický tvar označujeme takovou úlohu lineárního programování, jejíž:

- všechny podmínky jsou ve tvaru rovnic,
- všechny proměnné mají podmínku nezápornosti,
- cíl optimalizace je maximalizace, tj. maximalizuje se účelová funkce.

Tento tvar není obvyklým zadáním, ale je velice důležitý pro řešení úlohy. Simplexová metoda, což je běžná metoda výpočtu, vyžaduje úlohu lineárního programování v tomto tvaru. Běžným postupem je tedy obecnou úlohu převést na tento tvar a poté ji řešit simplexovou metodou [7].

2.4.2 Symetrický tvar

Pokud jsou veškeré omezující podmínky ve tvaru nerovností s operátorem \leq nebo veškeré s operátorem \geq , označujeme tento tvar úlohy lineárního programování za symetrický [5].

2.4.3 Smíšený tvar

Anglicky označováno jako „mixed constrain LP problem“, neboli smíšený tvar úlohy lineárního programování je takový tvar úlohy, kdy podmínky obsahují různé druhy operátorů $=$, \leq nebo \geq [5].

2.4.4 Převod mezi tvary

Z převodů mezi tvary úloh se v praxi obvykle používá pouze převod na kanonický tvar a to z následujících důvodů:

- tento tvar je vyžadován pro simplexovou metodu výpočtu,
- lze ji předat pouze pomocí tří proměnných, matice \bar{A} a vektorů \bar{x} a \bar{b} (viz. 2.4.1),
- s rovnicemi v podmínkách je možné provádět řádkové ekvivalentní úpravy bez vedlejšího vlivu na *prostor přípustných řešení* [5].

Převod minimalizace na maximalizaci

Převod minimalizace na maximalizaci se provádí vynásobením účelové funkce konstantou -1, tedy otočení znamének všech jejich koeficientů [5].

Převod podmínkových nerovností \leq na rovnice

Jednotlivé podmínkové nerovnice je nutné jednu po druhé převést na rovnice tak, aby tvar úlohy odpovídal kanonickému tvaru. Nerovnice méně nebo rovno \leq , lze převést na rovnice $=$, přidáním další proměnné x_i (kde i je další dosud nepoužitý index) na levou stranu nerovnice. Tato proměnná se nazývá *doplňková proměnná* a je to množství, které na levé straně nerovnice chybí k tomu, aby se z ní stala rovnice. Tedy její hodnota je doplněk (rezerva) rozdílu původní levé a pravé strany, s jakou je nerovnost splněna. Na tuto proměnnou je nutné dát podmínku nezápornosti [2].

Převod podmínkových nerovností \geq na nerovnosti \leq

Pokud podmínková nerovnice obsahuje operátor více nebo rovno \geq , je možné ji převést na tvar méně nebo rovno \leq a ten následně na rovnost, jak je popsáno v předchozím odstavci. Dle matematické poučky víme, že: “při násobení nebo dělení nerovnic záporným číslem musíme otočit původní znaménko nerovnosti“. Toho je možné využít a převod provést vynásobením nerovnice číslem -1 [8].

Existuje i druhá možnost, což je obdobný postup jako je převod nerovností \leq na rovnice, ale *doplňková proměnná* se vloží se záporným znaménkem. Tato možnost však komplikuje kontrolu porušení podmínky nezápornosti popsanou dále v textu [2].

Příklad převodu na kanonický tvar

Smíšený tvar úlohy:

$$\begin{aligned} \max L(x) &= x_1 + \frac{1}{2}x_2 + 3x_3 & (6) \\ x_1 + x_2 + x_3 &\leq 300 \\ \frac{2}{10}x_1 + \frac{100}{15}x_2 + \frac{2}{100}x_3 &\leq 50 \\ x_3 &\leq 20 \\ x_2 &\geq 100 \end{aligned}$$

se převede dle pravidel v kapitole 2.4.4 na následující kanonický tvar:

$$\begin{aligned} \max L(x) &= x_1 + \frac{1}{2}x_2 + 3x_3 & (7) \\ x_1 + x_2 + x_3 + x_4 &= 300 \\ \frac{2}{10}x_1 + \frac{100}{15}x_2 + \frac{2}{100}x_3 + x_5 &= 50 \\ x_3 + x_6 &= 20 \\ -x_2 + x_7 &= -100 \end{aligned}$$

Tučně zobrazené proměnné jsou *doplňkové proměnné*.

2.5 Typy řešení úloh lineárního programování

U úlohy lineárního programování můžou nastat 3 výsledné situace:

1. Úloha nemá žádné přípustné řešení (*prostor přípustných řešení* je prázdná množina).
2. Úloha má přípustné řešení, ale žádné není optimální.
3. Úloha má optimální řešení, kterých může být i více, ale mají stejnou hodnotu účelové funkce.

V prvním případě to znamená, že průnik podmínek řešení je prázdná množina, tedy $M = \emptyset$.

V druhém případě již průnik podmínek řešení není prázdná množina, tedy $M \neq \emptyset$, ale množina optimálních řešení je prázdná, tedy $M^* = \emptyset$. V takovém případě existuje *přípustné bázecké řešení*.

V třetím případě prostor přípustných řešení i množina optimálních řešení není prázdná, tedy $M \neq \emptyset, M^* \neq \emptyset$. V takovém případě existuje *bázecké optimální řešení*.

2.6 Bázicky přípustná řešení úlohy lineárního programování

V následujícím popisu předpokládejme tvar úlohy v kanonickém tvaru a maticové formě $\bar{A} * \bar{x} = \bar{b}$

2.6.1 Sloupcový vektor

Všechny sloupce matice \bar{A} generují tzv. *sloupcový vektor*. Lineární kombinace jednotlivých sloupců matice \bar{A} je řešením soustavy $\bar{A} * \bar{x} = \bar{b}$ a výsledkem je vektor jejich pravých stran. Hodnoty proměnných x_1, x_2, \dots, x_n jsou tvořeny koeficienty této lineární kombinace, pokud však existuje.

Úloha má řešení pouze v případě, že je možné z matice \bar{A} pomocí lineární kombinace sloupců vygenerovat vektor \bar{b} [2].

2.6.2 Bázické řešení

Je řešení $\bar{A} * \bar{x} = \bar{b}$, kdy sloupce matice \bar{A} odpovídající nenulovým složkám tohoto řešení jsou lineárně nezávislou soustavou vektorů.

Bázické řešení se nazývá *nedegradované*, pokud je počet nenulových složek bázického řešení roven hodnoti matice \bar{A} .

Bázické řešení se nazývá *degradované*, pokud je počet nenulových složek bázického řešení menší hodnoti matice \bar{A} .

Pokud je počet nenulových složek bázického řešení větší než hodnoti matice \bar{A} , nejedná se o bázické řešení [7].

2.6.3 Přípustné bázické řešení

Oproti bázickému řešení je navíc doplněno o podmínku nezápornosti $x \geq 0$. Je možné dokázat, že přípustné řešení je bázické, pokud je krajním bodem množiny přípustných řešení [2].

Pokud uvažujeme obdélníkové matice X a Y takové, že X má počet řádků m , počet sloupců n , hodnot m a Y je tvořená m lineárně nezávislými sloupci z matice X , potom Y je matice báze. Proměnné odpovídající jejím sloupcům se nazývají bázické a ostatní proměnné jsou nebázické.

Počet různých bázičských řešení je roven:

$$\binom{n}{m} = \frac{n!}{m! - (n - m)!} \quad (8)$$

2.7 Geometrický pohled na lineární programování

2.7.1 Bod n-rozměrného reálného prostoru

Bod je určen uspořádaným zápisem n souřadnic, kde n je rozměr prostoru, ve kterém se tento bod nachází. Souřadnice jsou reálným číslem, a proto je nazýván *bod n-rozměrného reálného prostoru* \mathbb{R}^n . Tuto n -tici zapisujeme (x_1, x_2, \dots, x_n) .

Takový bod označíme například písmenem a . V případě, že bodů bude více, je vhodné je indexovat dolním indexem $a_1, a_2 \dots a_n$ a souřadnice každého takto označeného bodu budou označeny dvěma indexy $a_i = (a_1^i, a_2^i, \dots, a_n^i)$.

2.7.2 Konvexní kombinace bodů

Konvexní kombinace bodů $a_1, a_2 \dots a_n$ je taková kombinace bodů, kdy každý z bodů má hodnotu h_i a součet těchto hodnot je roven 0.

Matematicky popsáno: Mějme v n -rozměrném reálném prostoru k reálných bodů $a_1, a_2 \dots a_k$ a k reálných čísel $h_1, h_2 \dots h_k$, kde k je kladné celé číslo, tj. $k = 1, 2, \dots, n$.

Pak bod $h_1 a_1 + h_2 a_2 + \dots h_n a_n$ je *konvexní kombinací bodů* $a_1, a_2 \dots a_k$ při splnění podmínek:

1. $h_i > 0$, kde k je kladné celé číslo, tj. $k = 1, 2, \dots, n$,
2. $\sum_{i=1}^k h_i = 1$ [2].

2.7.3 Další kombinace bodů

Existují i jiné typy kombinací bodů $a_1, a_2 \dots a_k$:

- *lineární kombinace* pokud jsou $h_1, h_2 \dots h_k$ jakákoli reálná čísla,
- *afinní kombinace* pokud platí $\sum_{i=1}^k h_i = 1$ [2].

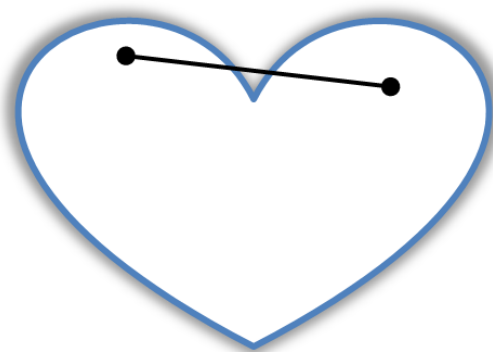
2.7.4 Geometrické vyjádření kombinací

Některými typickými kombinacemi bodů $a_1, a_2 \dots a_k$ jsou:

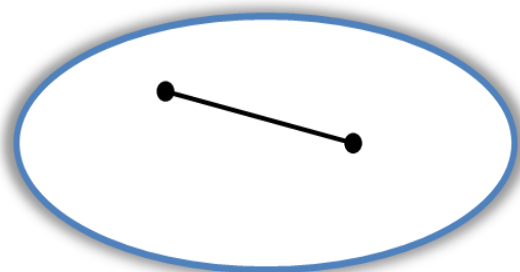
- Konvexní kombinací bodů a, b je úsečka obsahující krajní body a a b .
- Afinní kombinací bodů a, b je přímka procházející body a a b .
- Konvexní kombinací bodů a, b, c neležících na přímce je trojúhelník s krajními body a, b, c .
- Afinní kombinací bodů a, b, c je rovina procházející body a, b, c [2].

2.7.5 Konvexní množina

Konvexní množina L je podmnožinou reálného afinního prostoru pro kterou platí, že každé dva body $a_1, a_2 \in L$ patří do této konvexní množiny jejich lineární kombinace. Geometrický význam je úsečka spojující libovolné dva krajní body $a_1, a_2 \in L$, patří do této množiny L po celé délce. Příklad konvexní množiny zobrazuje Obrázek 3 a příklad nekonvexní zobrazuje Obrázek 2 [7].



Obrázek 2 – Nekonvexní množina



Obrázek 3 – Konvexní množina

Pokud není možné bod vyjádřit pomocí konvexní kombinace dvou jiných bodů, označujeme *krajní bod* konvexní množiny. Například kruh obsahuje nekonečný počet krajních bodů, tyto krajní body tvoří kružnici [7].

Konvexní množinu obsahující konečný počet krajních bodů označujeme jako *konvexní polyedr*. Taková množina je průnikem konečného počtu uzavřených poloprostorů a nadrovin definovaných omezujícími podmínkami [7].

2.8 Grafické řešení (řešení dvourozměrných úloh)

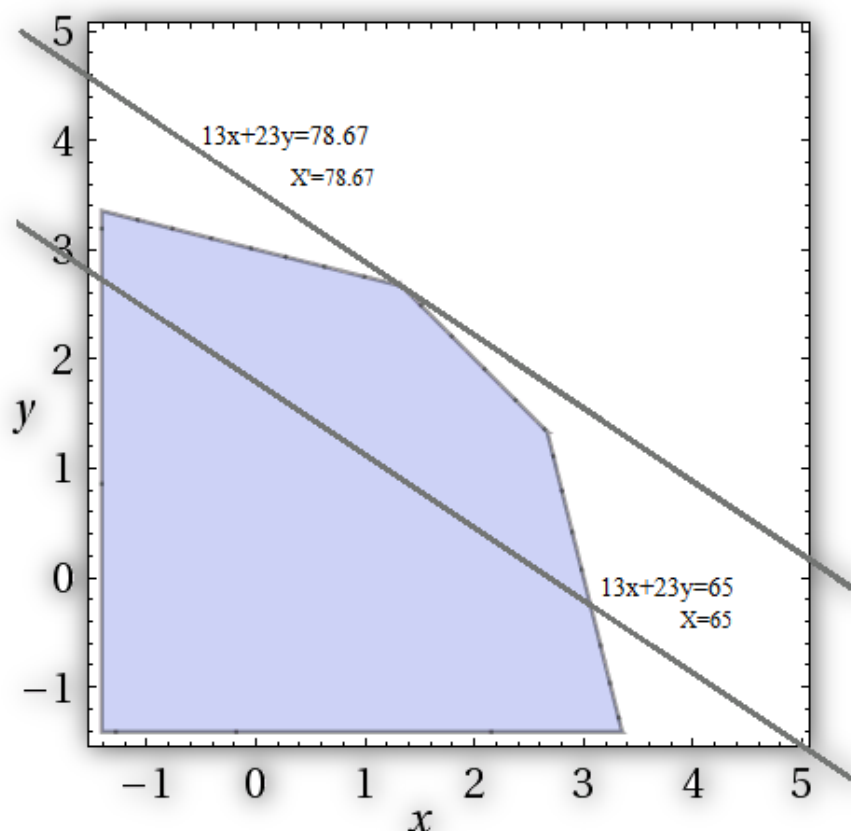
Počet proměnných *účelové funkce* definuje počet rozměrů *prostoru řešení*. V případě, že *účelová funkce* obsahuje pouze dvě proměnné, je možné prostor řešení zobrazit graficky a tuto úlohu také graficky řešit. Pak řešení probíhá v následujících krocích:

1. Zakreslíme jednotlivá omezení v podobě polorovin.
2. Najdeme průnik těchto polorovin, které zobrazují množinu *přípustných řešení*, viz kapitola 1.4.2.
3. V případě, že je průnik prázdný, úloha nemá optimální řešení a není možné pokračovat.
4. Zvolíme libovolnou hodnotu X a zakreslíme účelovou funkci rovnou této hodnotě.
5. Sestrojíme rovnoběžku zakreslené přímky z bodu 4. tak, aby průnik s metodou řešení nebyl prázdný, a současně nová hodnota účelové funkce X' byla ve směru optimalizace [9].

Příklad řešme graficky:

$$\begin{aligned} \max L(x) &= 13x + 15y & (9) \\ 8x + 2y &\leq 24 \\ x + 4y &\leq 12 \\ 4x + 4y &\leq 16 \\ x &\geq 0 \\ y &\geq 0 \end{aligned}$$

Příklad řešení dvourozměrné úlohy lineárního programování grafickou metodou, zobrazuje Obrázek 4. Jak popisuje postup, nejprve byly zakresleny tři omezující podmínky a z nich sestaven vybarvený prostor řešení, následně byla náhodně zvolena hodnota účelové funkce $X = 65$ a její přímka zakreslena. K této přímce byla sestrojena rovnoběžka tak, aby s prostorem řešení měla právě jeden společný bod. Protože se jedná o maximalizační úlohu, rovnoběžka je zkonstruována směrem zvyšujícím hodnoty x a y tedy tak, aby výsledná hodnota účelové funkce X' byla co možná nejvyšší.



Obrázek 4 – Příklad řešení dvourozměrné úlohy LP

2.9 Simplexová metoda

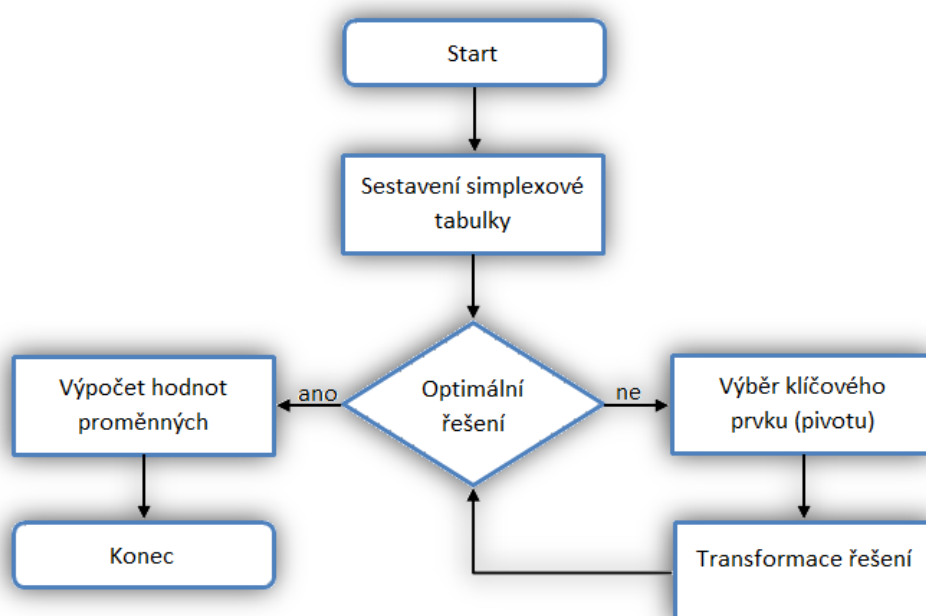
Simplexová metoda je nejčastěji používanou metodou pro řešení úloh lineárního programování, tedy pro nalezení optimálního řešení. Protože prostor bázecky přípustných řešení je nekonečný, tato metoda jej neprohledává celý, ale v každém kroku zvyšuje hodnotu účelové funkce a díky tomu v několika krocích konverguje k optimálnímu řešení [7].

Jedná se o iterační metodu výpočtu a je ji tedy možné popsat v následujících krocích:

1. Ze zadání úlohy se provede sestavení simplexové tabulky.
2. Je prozkoumán aktuální stav úlohy a proveden test optimálnosti. Pokud řešení je optimální, pokračuje se krokem číslo 5 a v případě, že optimální není, výpočet pokračuje krokem číslo 3.

3. Je proveden výběr klíčového sloupce i a řádku j , prvek a_{ij} je klíčovým prvkem, tzv. *pivot* a pomocí něj je proveden krok 4.
4. Je provedena transformace pomocí výměny jednoho bázeckého vektoru, která odpovídá stejné, obvykle však větší hodnotě účelové funkce a řešení pokračuje další iterací na krok 2.
5. Ze simplexové tabulky jsou vypočteny hodnoty jednotlivých proměnných [5].

Obrázek 5 – Znázorňuje vývojový diagram popsaného algoritmu.



Obrázek 5 – vývojový diagram algoritmu simplexové metody

2.9.1 Sestavení simplexové tabulky

Sestavení simplexové tabulky předpokládá úlohu zadanou v kanonickém tvaru. Pokud úloha v tomto tvaru není, je nutné ji do něj převést (dle kapitoly 2.4.4.). Je tedy dána úloha:

$$\begin{aligned}
 \max L(x) &= c_1 x_1 + c_2 x_2 + \dots + c_n x_n & (10) \\
 a_{1,1} * x_1 + a_{1,2} * x_2 + \dots + a_{1,n} * x_n &\leq b_1 \\
 a_{2,1} * x_1 + a_{2,2} * x_2 + \dots + a_{2,n} * x_n &\leq b_2 \\
 &\vdots \\
 a_{m,1} * x_1 + a_{m,2} * x_2 + \dots + a_{m,n} * x_n &\leq b_m \\
 x_1, x_2, \dots, x_n &\geq 0
 \end{aligned}$$

Tabulka 2 uvádí formát, do kterého se úloha přepíše.

Tabulka 2 – Obecná simplexová tabulka

x_1	x_2	...	x_n	x_{n+1}	x_{n+2}	x_{n+m}	b
a_{11}	a_{12}	...	a_{1n}	1	...	0	b_1
a_{21}	a_{22}	...	a_{2n}	0	...	0	b_2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
a_{m1}	a_{m2}	...	a_{mn}	0	...	1	b_m
$-c_1$	$-c_2$...	$-c_n$	0	...	0	f

Příklad sestavení simplexové tabulky. Následující úloha:

$$\begin{aligned}
 \max L(x) &= (1/2)x + 3y + z + 4w & (11) \\
 x + y + z + w &\leq 40 \\
 2x + y - z - w &\leq 10 \\
 w - y &\leq 10 \\
 x \geq 0, y \geq 0, z \geq 0, w &\geq 0
 \end{aligned}$$

následně se přepíše do formátu, který zobrazuje Tabulka 3.

Tabulka 3 – Příklad sestavení simplexové tabulky

x	y	z	w	$z0$	$z1$	$z2$	p	
1	1	1	1	1	0	0	0	40
2	1	-1	-1	0	1	0	0	10
0	-1	0	1	0	0	1	0	10
-0.5	-3	-1	-4	0	0	0	1	0

2.9.2 Zjištění optimálního řešení

Zjištění, je-li řešení maximální a výpočet se má ukončit, je provedeno porovnáním znamének koeficientů $c_1 \cdots c_n$. Mohou nastat 3 následující situace:

1. Jestliže existuje alespoň jeden koeficient se záporným znaménkem ($c_i < 0$), není dosaženo maxima a řešení není optimální.
2. Jsou-li všechny koeficienty kladné, je dosaženo maxima.
3. Je-li některý koeficient roven nule ($c_i = 0$), existuje více optimálních řešení.

U úloh převáděných ze smíšeného tvaru je nutné provádět kontrolu splnění *podmínek nezápornosti pro jednotlivé proměnné*. Tato kontrola bývá předřazena kontrole zjištění optimálnosti, a pokud je některá proměnná záporná, je třeba provést takovou transformaci řešení, aby porušení této podmínky bylo odstraněno [5].

2.9.3 Volba klíčového prvku

Klíčový prvek „pivot“ je nalezen volbou klíčového sloupce a následně volbou klíčového řádku. Prvek, který se nachází v klíčovém sloupci i klíčovém řádku, je klíčový prvek.

Volba sloupce

Klíčový sloupec se volí z množiny nebázických sloupců, neboli ze sloupců původně zadaných proměnných. Volba sloupce má zásadní vliv na hodnotu účelové funkce po transformaci řešení. Špatnou volbou sloupce je dokonce možné hodnotu účelové funkce snížit, to nastane v situaci, kdy je vybrán sloupec i prvkem $c_i > 0$. V maximalizační úloze je ale nutné hodnotu účelové funkce zvyšovat. Sloupce je možné vybrat dle jednoho z následujících pravidel:

1. volit libovolný sloupec se záporným koeficientem c_i ,
2. volit první sloupec zleva se záporným koeficientem c_i ,
3. volit sloupec se nejvyšším záporným koeficientem c_i ,
4. volit sloupec tak aby přírůstek účelové funkce byl maximální

Jakákoliv volba sloupce se záporným koeficientem c_i maximalizuje účelovou funkci, ale vhodně zvolený sloupec může snížit počet iterací. Pravidlo číslo 2. se používá pro zamezení tzv. degenerace řešení. Nejvíce používaným pravidlem pro volbu sloupce je pravidlo 3., tedy volba nejvíce záporného koeficientu. Jeho nalezení je výpočetně snadné a průměrný přírůstek je lepší než u předchozích pravidel. Pravidlo 4. dosahuje pro daný krok lepšího přírůstku oproti všem ostatním, ale nalezení takového sloupce je výpočetně náročné a v některých případech nezaručuje nejmenší počet kroků výpočtu [2].

Příklad výběru sloupce dle pravidla 3 znázorňuje Tabulka 4.

Tabulka 4 – Příklad volby sloupce v simplexové tabulce

x	y	z	w	$z0$	$z1$	$z2$	p	
1	1	1	1	1	0	0	0	40
2	1	-1	-1	0	1	0	0	10
0	-1	0	1	0	0	1	0	10
-0.5	-3	-1	-4	0	0	0	1	0

Volba řádku

Po volbě sloupce s následuje volba řádku. Proces volby řádku probíhá nejprve výpočtem pomocných proměnných pro každý řádek a následně výběrem řádku na základě těchto pomocných proměnných. Pomocné proměnné řádku vypočteme jako podíl $\frac{b_i}{a_{is}}$, kde i je číslo řádku a s je zvolený sloupec. Z nich vybereme tu pomocnou proměnnou, která má nejmenší kladnou hodnotu. Index i této proměnné označuje zvolený řádek. Prvek a_{is} je nazýván klíčový prvek nebo také „pivot“ [5].

Tabulka 5 zobrazuje příklad výběru sloupce a výpočet pomocných proměnných pro každý řádek a sloupec, jehož zvolení znázorňuje Tabulka 4.

Tabulka 5 – Příklad volby sloupce v simplexové tabulce

x	y	z	w	$z0$	$z1$	$z2$	p		$\frac{b_i}{a_{i4}}$
1	1	1	1	1	0	0	0	40	$40/1 = 40$
2	1	-1	-1	0	1	0	0	10	$10/-1 = -10$
0	-1	0	1	0	0	1	0	10	$10/1 = 10$
-0.5	-3	-1	-4	0	0	0	1	0	

2.9.4 Transformace řešení

Transformace řešení je speciální případ Jordanovy eliminační metody. Speciální proto, že z tohoto algoritmu bude proveden pouze první krok a to následujícím způsobem:

1. všechny prvky klíčového řádku dělíme klíčovým prvkem,
2. další řádky odčítáme od takového násobku klíčového řádku, který zajistí, že klíčový sloupec bude obsahovat nuly.
3. Tento násobek lze zjistit výpočtem $k = \frac{a_{is}}{a_{rs}}$ a nová hodnota prvku v jiném než klíčovém sloupci se vypočte dle vzorce $a_{ij} = a_{ij} - (a_{rj} * k)$, kde i je index zpracovávaného řádku, j index zpracovávaného sloupce, s je sloupec klíčového prvku a r je řádek klíčového prvku [2].

Tabulka 6 zobrazuje výsledek transformace předchozího kroku úlohy, který zobrazuje Tabulka 5 za pomoci klíčového prvku, včetně jeho výběru.

Tabulka 6 – Příklad transformace řešení

x	y	z	w	$z0$	$z1$	$z2$	p	
1	2	1	0	1	0	-1	0	30
2	0	-1	0	0	1	1	0	20
0	-1	0	1	0	0	1	0	10
-0.5	-7	-1	0	0	0	4	1	40

2.9.5 Výpočet hodnot proměnných

Pokud je řešení označeno jako optimální nebo je třeba provést kontrolu *nezápornosti proměnných*, je nutné vypočítat z tabulky hodnoty *základních proměnných*, což se provede následujícím postupem.

x_s je základní proměnná v případě, že její hodnotu je možné z tabulky simplexové metody určit. Zda je proměnná x_s základní, je možné určit dle sloupce s , který musí obsahovat pouze jednu nenulovou hodnotu $a_{sj} > 0$, hodnota proměnné se poté rovná podílu složky vektoru pravých stran b_j ve stejném řádku a této nenulové hodnoty $x_s = \frac{a_{sj}}{b_j}$. Hodnota f v posledním řádku sloupce vektoru pravých stran je hodnota účelové funkce.

Tabulka 7 – Výsledná simplexová tabulka

x	y	z	w	$z0$	$z1$	$z2$	p	
1	1	0,5	0	0,5	0	-0,5	0	15
2	0	-1	0	0	1	1	0	20
0,5	0	0,5	1	0,5	0	0,5	0	25
3	0	2,5	0	3,5	0	0,5	1	145

Příklad výpočtu hodnot proměnných zobrazuje Tabulka 7:

$$y = \frac{15}{1} = 15; z = 0; w = \frac{25}{1} = 25$$

$$p = \frac{145}{1} = 145$$

Žádná proměnná neporušuje podmínku *nezápornosti* a výsledky jsou tedy platné optimální řešení.

2.9.6 Porušení podmínky nezápornosti

V případě úloh převáděných ze smíšeného tvaru na kanonický, může často dojít k porušení podmínky nezápornosti proměnné. Pokud tato situace nastane, její vyřešení má vždy přednost před volbou nejvhodnějšího sloupce a řádku, jak bylo popsáno v kapitole 2.9.3. Pokud by se odstranění porušení této podmínky zanedbalo, výsledek řešení úlohy není optimální a současně neleží v prostoru řešení [10].

V případě, že je tabulka sestavena dle doporučení v kapitole 2.4.4, je možné kontrolu porušení podmínky nezápornosti provádět velice rychle a efektivně, a to kontrolou nezápornosti vektoru pravých stran, tedy sloupce hodnot b , což je poslední sloupec simplexové tabulky.

Tabulka 8 zobrazuje následující situaci. Dle dříve zmíněných pravidel by jako klíčový sloupec byl zvolen sloupec proměnné w . V tabulce je ale evidentně porušena podmínka nezápornosti proměnných z_1 a z_2 . Toto porušení lze řešit dvěma možnými způsoby.

1. Jak popsal Ron Larson ve své knize Linear Algebra neexistuje žádný návod, jak vybrat klíčový prvek a v tomto případě je to potřeba řešit metodou pokusu a omylu [10].
2. Autoři Harshbarger a Reynolds v knize Mathematical Applications for the Management, Life, and Social Sciences uvádějí, že je možné matematicky prokázat, že v případě existence optimálního řešení existuje na řádku r se zápornou hodnotou b_r také minimálně jeden další záporný prvek a_{ir} . Prvek a_{ir} umožní odstranit porušení podmínky nezápornosti. V případě že je v řádku záporných prvků více, zvolíme ten s nejvyšší absolutní hodnotou. Tento prvek zvolíme jako *klíčový prvek* a provedeme obvyklý krok transformace řešení (viz kapitola 2.9.4), čímž odstraníme porušení podmínky nezápornosti [8].

Následující příklad:

$$\begin{aligned} \max L(x) &= (1/2)x + 3y + z + 4w & (12) \\ x + y + z + w &\leq 40 \\ 2x + y - z - w &\geq 10 \\ w - y &\geq 10 \\ x \geq 0, y \geq 0, z \geq 0, w &\geq 0 \end{aligned}$$

bude převeden na simplexovou tabulku, kterou zobrazuje Tabulka 8.

Tabulka 8 – Volba klíčového prvku v případě porušení podmínky nezápornosti

x	y	z	w	z_0	z_1	z_2	p	
1	1	1	1	1	0	0	0	40
-2	-1	1	1	0	1	0	0	-10
0	1	0	-1	0	0	1	0	-10
-0.5	-3	-1	-4	0	0	0	1	0

2.10 Podmínka celočíselnosti

Z předchozích kapitol je zřejmé, že hodnoty proměnných ať již v průběhu či na konci výpočtu mohou nabývat neceločíselných hodnot. Mnohé úlohy v praxi však vyžadují, aby optimální hodnoty proměnných byly celočíselné. Kupříkladu při optimalizaci výrobního programu není možné vyrobit 42,55 výrobků. Jako první se obvykle objeví myšlenka zaokrouhlení, je ale možné prokázat, že v mnoha případech zaokrouhlené hodnoty není možné akceptovat, protože nepatří do prostoru přípustných řešení. Řešení této problematiky je několik. Obvykle je úloha nejprve vyřešena standardně v desetinných číslech a poté jsou výsledné hodnoty vhodně upraveny na celočíselné za použití některé ze speciálních metod. Z výše uvedeného tedy vyplývá, že řešení celočíselných úloh je obtížnější, neboť simplexová metoda tuto podmínku sama o sobě zajistit nedokáže [7].

Nejjednodušším způsobem je upravit řádek simplexové tabulky tak, aby neobsahoval desetinná čísla, čehož je možné docílit vynásobením řádku vhodným koeficientem, který toto zajistí. Ideální hodnota koeficientu je nejmenší společný násobek čísel daného řádku. Tedy desetinná čísla se převedou na zlomky, určí se nejmenší společný násobek jejich číselníků a touto hodnotou se vynásobí všechna čísla. Přepočtený řádek poté obsahuje celočíselné hodnoty.

2.11 Duální úloha

Ke každé úloze lineárního programování existuje tzv. *duální úloha*. Mezi zadanou obecnou úlohou a duální úlohou platí následující vztahy.

- Cíl optimalizace je opačný.
- Počet proměnných v zadané úloze je roven počtu omezujících podmínek v duální úloze.

- Matice *strukturních podmínek* duální úlohy je rovna transponované matici zadané úlohy.
- Hodnoty proměnných zadané úlohy jsou v duální úloze hodnoty pravých stran.
- Hodnoty pravých stran zadané úlohy jsou v duální úloze hodnoty proměnných [2].

Pro lepší porozumění nejprve zavedeme *standartní tvar* úlohy lineárního programování, který splňuje následující podmínky:

- všechny podmínky jsou ve tvaru nerovnic menší rovno „ \leq “,
- všechny proměnné mají podmínku nezápornosti,
- cíl optimalizace je maximalizace, tj. maximalizuje se účelová funkce.

Dle kapitoly 2.4.4 víme, že každou úlohu je možné na takovýto tvar převést, je možné jej zapsat

$$\begin{aligned} c^T x &\rightarrow \max \\ AX &\leq b \\ x &\geq 0 \end{aligned} \quad (13)$$

a k němu odpovídající duální úlohy:

$$\begin{aligned} b^T &\rightarrow \min \\ A^T y &\geq c \\ y &\geq 0 \end{aligned} \quad (14)$$

Pokud má základní úloha optimální řešení, pak i duální úloha má optimální řešení a hodnota účelové funkce je stejná [2].

2.11.1 Duální neboli stínové ceny

Optimální hodnoty duálních proměnných y_r se nazývají duální, někdy také stínové ceny. Jejich hodnota pro každý řádek y_r udává, o kolik se zvýší hodnota účelové funkce v případě, že omezení pravé strany b_r se zvýší o malou hodnotu x . Pokud k původnímu b_r přičteme x , účelová funkce se zvýší o $y_r \cdot x$ oproti své původní hodnotě. Velká změna x však může způsobit překročení meze stability řešení [2].

2.12 Nedočerpání omezení

V kapitole 2.4.4 bylo popsáno vytvoření *doplňkové proměnné*. Pro zopakování je to doplněk (rezerva) levé strany nerovnice větší nebo rovno \leq , aby se z ní stala rovnice.

Výsledná hodnota této proměnné má ekonomický význam, protože vyjadřuje velikost nedočerpání či překročení minimálního limitu určitého zdroje [3].

2.13 Citlivostní analýza

V zadání úlohy lineárního programování jsou jednotlivé koeficienty zadány konkrétní hodnotou a po celou dobu výpočtu jsou tyto koeficienty neměnné. Z hlediska praxe může být zajímavou otázkou, jaký vliv má změna hodnoty jednoho koeficientu na výsledek, tedy na optimální řešení. Na tuto otázku přináší odpověď právě *citlivostní analýza*, což je vztah mezi změnou parametru modelu úlohy a výslednou optimální hodnotou. Pro každou změnu parametru modelu je tedy nutné znovu vypočítat novou optimální hodnotu. Tento vztah se obvykle vyjadřuje graficky [4].

II. PRAKTICKÁ ČÁST

3 ANALÝZA POŽADAVKŮ

3.1 Cíle praktické části

Hlavním cílem praktické části je vytvoření interaktivní webové aplikace pro řešení problému lineárního programování za pomoci simplexové metody. Aplikace bude umožňovat uživateli zadat do webové stránky vstupní data úlohy a poskytnout mu informace o řešení problému za předpokladu, že řešení existuje. Protože simplexová metoda je iterační algoritmus, budou informace o řešení problému zobrazeny jednorázově nebo po jednotlivých krocích, kdy se spolu s posledním krokem výpočtu zobrazí výsledek řešení. Zobrazení výpočtů a kroků musí být v přehledné výstupní sestavě s možností uložit celé řešení do formátu PDF.

Na základě cílů praktické části je možné sestavit seznam funkčních a nefunkčních požadavků na interaktivní webovou aplikaci. Tento seznam požadavků je určen pro finální verzi webové aplikace a není striktně závazný tak, jak tomu bývá v praxi. To znamená, že je možné v aplikaci realizovat některé funkce navíc oproti tomuto seznamu, či některé vypustit. Tyto odklony od požadavků jsou obvykle z mnoha různých důvodů, jako např. v momentě, kdy zákazník v průběhu realizace projektu změní své požadavky, což má za následek jejich přidání či vyřazení. Dále se může stát, že při detailním prozkoumání určité problematiky je změněn postup celého projektu a následně se přehodnocují některé části aplikace. I přes tyto možné odchylky je seznam požadavků velice důležitou částí specifikace navrhované aplikace.

3.2 Funkční požadavky

Funkční požadavky jsou požadavky na funkce webové aplikace. Funkce jaké má webová aplikace vykonávat jsou následující:

- výpočet problému lineárního programování simplexovou metodou,
- zadávání problému lineární úlohy textovým zápisem,
- zadávání problému lineární úlohy tabulkovým zápisem,
- volba řešící metody (desetinná čísla, zlomky, celočíselně),
- volba zobrazení výsledku (po krocích či jednorázově),
- zobrazování výpočtu simplexovou metodou krok po kroku,
- jednorázové zobrazení výpočtu simplexovou metodou,

- zobrazení kontroly výpočtu,
- zobrazení nedočerpání omezení,
- zobrazení hodnot stínových cen,
- možnost spustit citlivostní analýzu,
- stažení kompletního výpočtu ve formátu PDF.

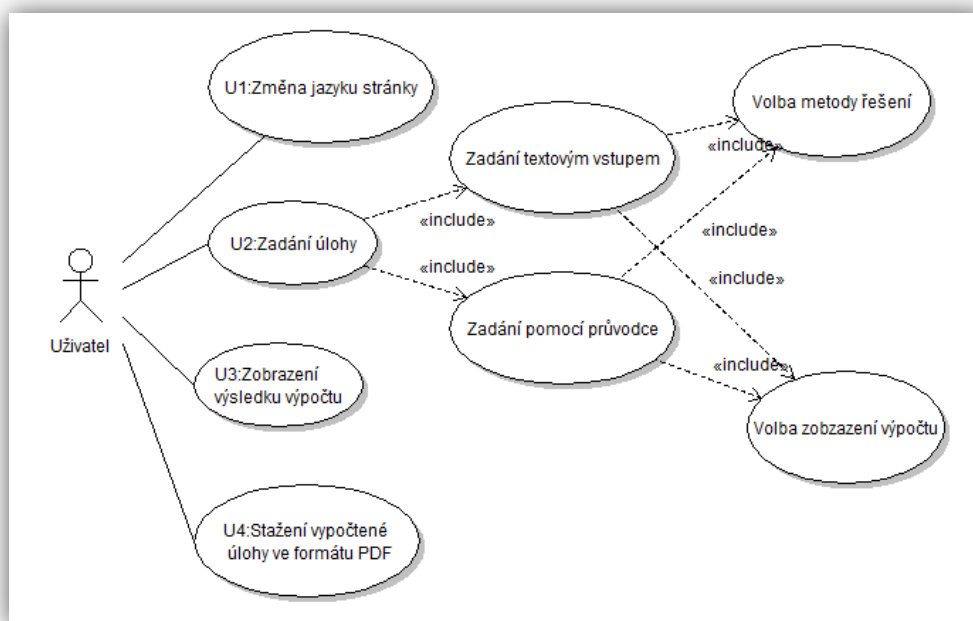
3.3 Nefunkční požadavky

Nefunkční požadavky nejsou požadavky přímo na funkce aplikace, ale také plní důležitou součást analýzy požadavků. Dávají si za cíl, aby vytvořená aplikace byla kvalitní, stabilní a její používání bylo pro uživatele komfortní. Nefunkční požadavky na aplikaci jsou:

- webový typ aplikace,
- možnost používání více uživateli najednou,
- bezpečnost aplikace proti útokům,
- česká lokalizace,
- přehledné výstupní sestavy,
- intuitivní ovládání,
- rozšiřitelnost (možnost rozšířit funkcionalitu),
- přenositelnost (možnost spustit aplikaci na různých serverových systémech).

3.4 Případy užití

Obrázek 6 znázorňuje případy užití, tzv. Use Case Diagram, tento diagram znázorňuje jaké funkce má uživatel systému k dispozici. Protože aplikace neřeší role ani přihlašování uživatelů, je zde pouze jeden aktér, ale i v takovémto případě je vhodné využít diagram případu užití pro ujasnění funkcí systému a jejich popis.



Obrázek 6 – Diagram případů užití

Tabulka 9 – Scénář případu užití změny jazyku stránky

U1	Změna jazyku stránky
Cíl případu užití:	Změnit lokalizaci aplikace dle zvolené hodnoty.
Hlavní aktér:	Uživatel
Vstupní podmínky:	-
Výstupní podmínky:	Lokalizace aplikace je změněna dle zvolené hodnoty.
Hlavní scénář:	<ol style="list-style-type: none"> 1. Uživatel se nachází na libovolné stránce a zvolí konkrétní jazyk ze seznamu dostupných. 2. Systém provede kontrolu existence jazyka. 3. Systém nastaví zvolený jazyk. 4. Systém přesměruje uživatele na úvodní stránku aplikace.
Alternativní scénář 2a :	2a1. Pokud systém vyhodnotí, že zvolený jazyk neexistuje, změna jazyku se neprovede.
(Zvolený jazyk neexistuje)	2a2. Systém přesměruje uživatele na úvodní stránku aplikace.

Tabulka 10 – Scénář případu užití zadání úlohy

U2	Zadání úlohy
Cíl případu užití:	Zadat parametry úlohy.
Hlavní aktér:	Uživatel
Vstupní podmínky:	Uživatel se nachází na stránce pro zadání průvodcem nebo textovým vstupem.
Výstupní podmínky:	Aplikace provede sestavení simplexové tabulky úlohy a provede její výpočet.
Hlavní scénář:	<ol style="list-style-type: none"> 1. Uživatel zadá požadovanou úlohu k řešení. 2. Uživatel zvolí způsob zobrazení výpočtu. 3. Uživatel zvolí metodu řešení. 4. Systém provede kontrolu správnosti zadání. 5. Systém provede sestavení simplexové tabulky. 6. Systém provede výpočet úlohy. 7. Systém zobrazí řešení úlohy.
Alternativní scénář 2a : (úloha je v nesprávném tvaru)	<p>2a1. Pokud systém vyhodnotí, že úloha je zadána v nesprávném tvaru ukončí další postup.</p> <p>2a2. Systém zobrazí informaci o chybě a ponechá zadaný vstup.</p>
Alternativní scénář 4b: (úloha nemá řešení)	<p>4b1. Pokud systém vyhodnotí, že úloha nemá řešení, ukončí další postup.</p> <p>4b2. Systém zobrazí informaci o chybě a ponechá zadaný vstup.</p>

Tabulka 11 – Scénář případu užití zobrazení výsledku

U3	Zobrazení výsledku výpočtu
Cíl případu užití:	Zobrazit řešení, výsledek a další informace k zadané úloze.
Hlavní aktér:	Uživatel
Vstupní podmínky:	Uživatel zadal do systému řešitelnou úlohu.
Výstupní podmínky:	Aplikace zobrazí řešení, postup a další informace k zadané úloze.
Hlavní scénář:	<ol style="list-style-type: none"> 1. Systém provede zobrazení tabulek s postupem. 2. Systém provede zobrazení tabulky s výsledky. 3. Systém provede zobrazení dalších informací, jako jsou stínové ceny, kontrola podmínek a rezerva omezení citlivostní analýza.

Tabulka 12 – Scénář případu užití stažení vypočtené úlohy ve formátu PDF

U4	Stažení vypočtené úlohy ve formátu PDF
Cíl případu užití:	Zobrazit či stáhnout řešení, výsledek a další informace k zadané úloze ve formátu PDF.
Hlavní aktér:	Uživatel
Vstupní podmínky:	Uživatel se nachází na stránce pro zobrazení výsledku, kam byl automaticky přesměrován systémem po zadání řešitelné úlohy.
Výstupní podmínky:	Aplikace zobrazí řešení, postup a další informace k zadané úloze ve formátu PDF.

4 NÁVRH ŘEŠENÍ

4.1 Architektura aplikace

Aplikace má být webovou aplikací, což je architektura typu klient-server. Webové aplikace ke svému fungování potřebuje několik zásadních částí a to zejména server, který zpracovává požadavky uživatele, provádí výpočty a odesílá výsledky, je zde spouštěn vlastní naprogramovaný kód, poté protokol HTTP či HTTPS, který zajišťuje přenos dat mezi klientem a serverem a webový prohlížeč, což je klientská strana systému, která zobrazuje data zaslaná serverem. Data jsou obvykle ve formátu HTML a spojení mezi klientem a serverem je přes síť internet či intranet.

4.2 Klientská část

Klientskou částí je internetový prohlížeč přímo u uživatele systému, na který nejsou kladeny speciální požadavky. Aplikace bude využívat standard XHTML 1.0 Transitional ¹ HTML a CSS level 3², což běžný internetový prohlížeč v dnešní době dokáže bezproblémově zobrazovat. Klientská část slouží pouze pro interakce s uživatelem a zobrazování výstupů aplikace.

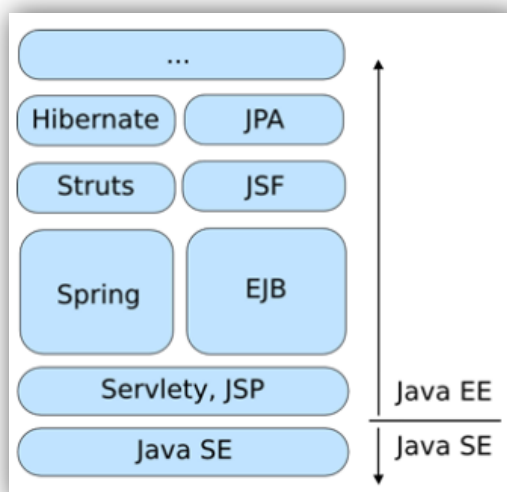
4.3 Java platforma

Java platforma je nejen programovací jazyk, který do ní také patří, ale i další součásti, což je například virtuální stroj Java (JVM-Java Virtual Machine), kompilátor, Garbage Collector (GC), standardní knihovny a další vybavení pro tvorbu a běh aplikací v jazyce Java. Java platforma se rozděluje na 4 verze přizpůsobené určitému použití. Jsou to:

- Java SE – využívá se pro aplikace běžící na standardním PC,
- Java EE – rozšíření Javy SE o další specifikace pro tvorbu webových aplikací, vzájemnou vazbu zobrazuje Obrázek 7,
- Java ME – aplikace pro mobilní telefony,
- Java FX – interaktivní webové aplikace spouštěny na straně klienta.

¹ URL adresa s popisem standardu: <http://www.w3.org/TR/xhtml1/>

² URL adresa s popisem standardu: <http://www.w3.org/TR/selectors/>

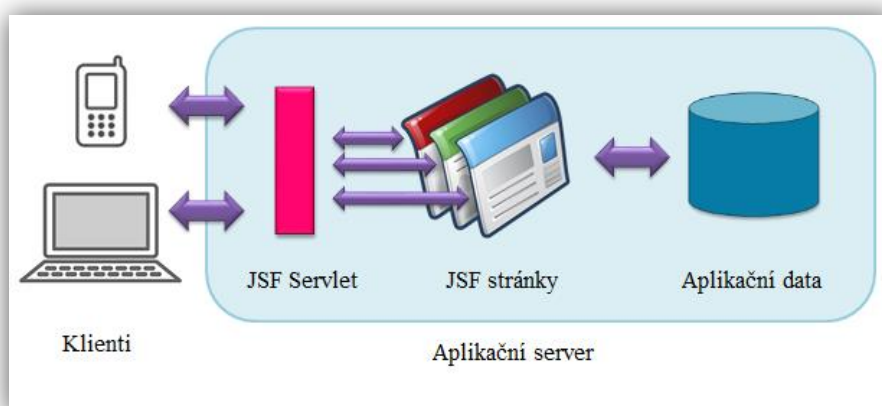


Obrázek 7 – Vazba Java SE a Java EE

Platforma Java je praxí prověřená a dostatečně bezpečná i stabilní na to, aby byla vhodná pro provozování této aplikace, průzkumy hovoří o 3 miliónech zařízení, které používají Javu. Platforma Java zajišťuje přenositelnost, tedy nezávislost na operačním systému serveru.

4.4 Serverová část

Serverová část aplikace bude naprogramována v jazyce Java, konkrétně Java EE 6. Tato část bude provádět veškeré operace, výpočty a řízení navigace. Klientská část tedy pouze zobrazuje grafické rozhraní a odesílá data. Serverovou část je možné rozdělit na další části, tak jak to zobrazuje Obrázek 8. Jsou to aplikační server, JSF Servlet, jednotlivé JSF stránky a aplikační data.



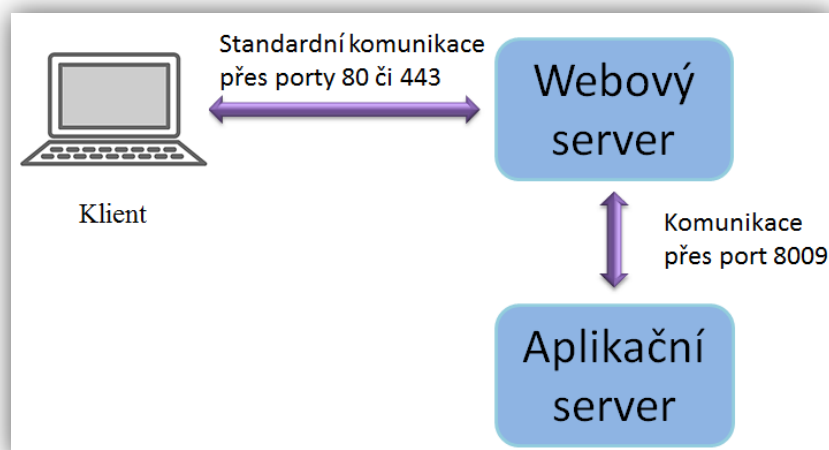
Obrázek 8 – Celková architektura aplikace

4.4.1 Aplikační server

Webové aplikace v prostředí Java EE se nespouští přímo, kde by bylo nejdříve nutné naprogramovat obsluhu HTTP protokolu, ale využívá se tzv. aplikační server, což je programové vybavení, které provádí obsluhu a zpracování HTTP protokolu. Požadavky, případně odpovědi, předává vlastní aplikaci přes jasně definované rozhraní, které se nazývají Servlety. Aplikační server ale také může zajišťovat připojení do databázových úložišť, řídit bezpečnost a účty uživatelů a další různé služby, které závisí na konkrétním typu a verzi aplikačního serveru.

4.4.2 Propojení aplikačního a webového serveru

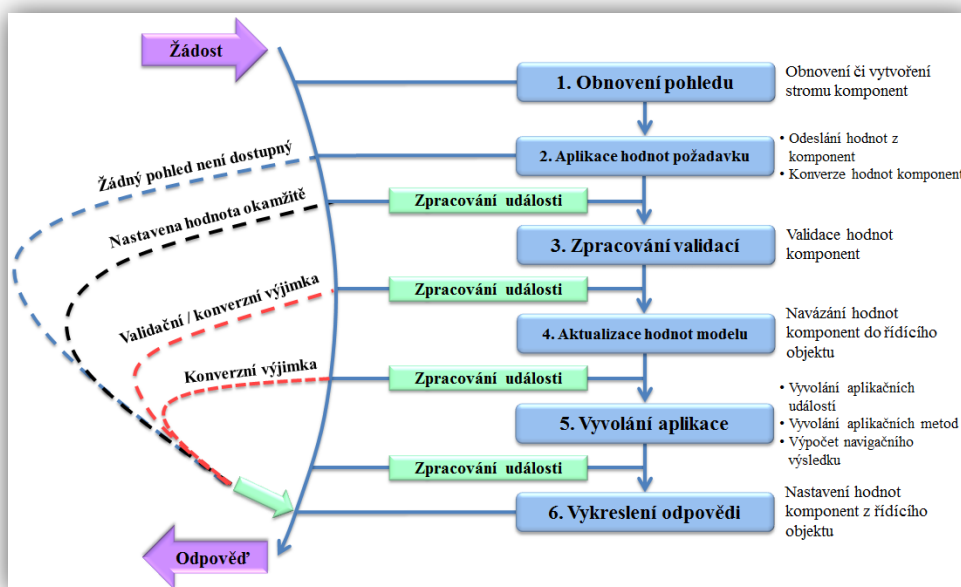
V produkčním prostředí je vhodné před aplikační server předřadit webový server, zejména z důvodů lepší škálovatelnosti a vyššího výkonu. Webový i aplikační server poté komunikují opět prostřednictvím sítě a jsou nakonfigurovány tak, aby vzájemně spolupracovaly. Nejčastěji používaným webovým serverem je Apache HTTPD, který je možný nakonfigurovat ke spolupráci s Apache Tomcat za pomoci dvou metod, je to *mod_proxy*, což je modul implementující reverzní proxy, nebo brána pro server Apache HTTPD a *mod_jk*, což je modul přímo určen k připojení aplikačního serveru Apache Tomcat. Podobným způsobem je možné propojit i jiné webové servery například Microsoft IIS za pomoci modulu ISAPI. Výsledné propojení a komunikační cestu zobrazuje Obrázek 9. Tato konfigurace má i další výhody, jako například možnost provozovat aplikační server na jiném hardware nežli je webový a tím zvětšit výkonnost, dokonce je možné provozovat aplikační server na více hardwarech a tím docílit navýšení výkonu. O směrování požadavků se poté stará webový server. Další výhoda nastane při výpadku či nasazování nové verze aplikace, kdy dochází ke krátkému výpadku aplikačního serveru. V této situaci webový server automaticky přesměruje uživatele na statickou stránku, kde může být například zobrazeno hlášení o údržbě serveru. Zákazník či uživatel tedy nevnímá celý server jako zcela nedostupný, ale vidí informaci o tom, že jeho služby jsou momentálně nedostupné.



Obrázek 9 – Propojení aplikačního a webového serveru

4.4.3 JSF framework

Jako hlavní Servlet pro výstupy ve formátu XHTML bude využit Servlet knihovny Java Server Faces (JSF), který je součástí Java EE od verze 5. Java Server Faces neobsahuje pouze Servlet, ale celý Framework pro tvorbu komponentních uživatelských rozhraní obsahující šablonovací systém, podporu lokalizace, řízení instancí objektů, validaci a konverzi dat, podporu zabezpečení a další funkce pro tvorbu webových aplikací. Obrázek 10 zobrazuje zjednodušený životní cyklus jednoho požadavku na JSF Servlet. Po přijmutí požadavku nejprve Servlet provede dohledání požadované šablony a na základě ní provede obnovení či vytvoření objektového stromu komponent. V kroku 2. je tento strom komponent naplněn hodnotami odeslanými uživatelem. V dalším kroku je provedena kontrola správnosti hodnot (validace) jednotlivých komponent, což je například kontrola formátu emailu, rozsah zadaného čísla či jeho formát a podobně. Po validaci následuje konverze a nastavení těchto hodnot do řídicího objektu (tzv. managed bean). Dále se spustí programátorem definované události a aplikační metody, na základě kterých je rozhodnuto o navigačním výsledku. Dle navigačního výsledku je zvolena šablona pro odpověď na požadavek, do této šablony se doplní hodnoty z řídicího objektu/objektů a tento se odešle klientovi.



Obrázek 10 – Životní cyklus požadavku na Servlet JSF

Pro generování výstupních souborů ve formátu PDF bude naprogramován vlastní Servlet.

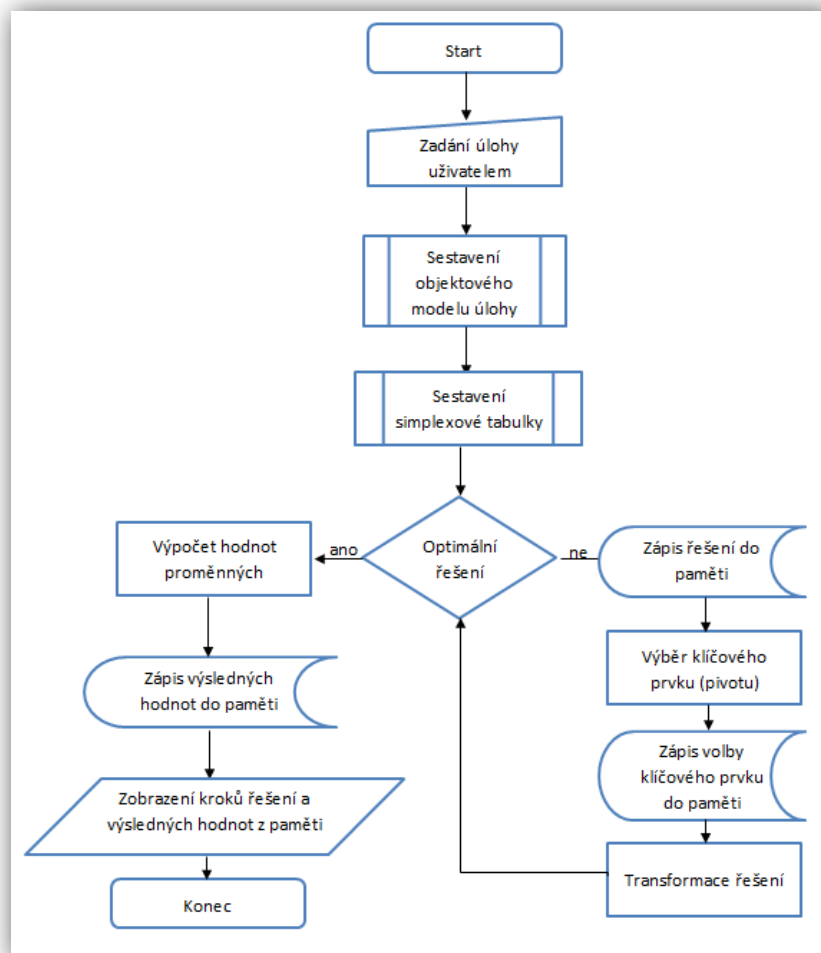
4.4.4 Pretty Faces

Pretty Faces je OpenSource knihovna pro přepisování URL adres s vylepšenou podporou pro Framework JSF. Umožňuje vytvářet uložitelné URL adresy a tzv. „pěkné“ URL adresy. Vylepšuje technologii řízení za pomoci Frameworku JSF a umožňuje spouštět akce po načtení stránky. Tato knihovna je vhodným doplňkem Frameworku JSF, protože opravuje jeho největší nedostatky v generování a vytváření URL adres [11].

4.5 Algoritmus řešení úlohy

Nejdůležitější částí aplikace je samotný výpočet zadané úlohy lineárního programování. Tento úkon není možné realizovat v jednom kroku, a proto je nutné vhodně navrhnout strukturu a objekty aplikace tak, aby výsledný program byl přehledný pro programátora a maximálně funkční pro uživatele. Obrázek 11 zobrazuje jednotlivé dílčí kroky, které musí aplikace vykonat. V každé části může nastat chyba, o které musí být uživatel vhodně informován. V prvním kroku po zadání úlohy uživatelem probíhá kontrola vstupních dat a sestavení objektového modelu zadané úlohy. Po jeho sestavení je zároveň dokončena kontrola správnosti vstupních údajů po syntaktické a sémantické stránce. V dalším kroku je výstup objektového modelu úlohy předán do třídy, která řeší vlastní výpočet a jeho provedení. Jednotlivé kroky výpočtu jsou ukládány do paměti, včetně souřadnic

klíčového prvku a po dosažení optimálního řešení jsou do paměti také uloženy výsledné hodnoty. Následně jsou hodnoty jednotlivých kroků zobrazeny uživateli, v nich zobrazeny volby klíčových prvků a nakonec jsou vypsané výsledné hodnoty řešení.



Obrázek 11 – Algoritmus řešení úlohy včetně zobrazení kroků

Objektový model úlohy je zobrazen diagramem tříd, který pro svou velikost je umístěn na obrázku v příloze PI.

4.6 Bezpečnost

Aplikace sama o sobě neobsahuje žádné citlivé ani tajné údaje, neboť neobsahuje perzistentní data, tj. data uložená mimo dočasnou paměť počítače, jako je například databáze nebo diskové úložiště. Z hlediska bezpečnosti tedy není nutné řešit zabezpečení dat a hesel. Je ale nutné se zaměřit na zabezpečení aplikace jako takové, aby potenciální útočník neměl možnost se díky aplikaci dostat k jiným datům uloženým na serveru, případně aby nebylo možné donutit server spustit kód podstrčený útočníkem.

5 IMPLEMENTACE

Samotná implementace výsledné aplikace pro řešení problému lineárního programování byla provedena za použití technologií popsanych v předchozí kapitole. Samotné algoritmy pro výpočet úlohy lineárního programování vychází z postupů popsanych v teoretické části práce. Celý projekt byl implementován v prostředí NetBeans IDE³, které umožňuje pohodlně pracovat se všemi zmíněnými technologiemi včetně nástroje Apache Maven.

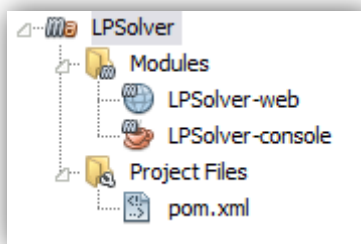
5.1 Apache Maven

Apache Maven⁴, zkráceně pouze Maven, je nástroj pro správu řízení a automatizaci sestavování softwarových aplikací v jazyce Java. Tento nástroj postupně nahrazuje původní aplikaci Apache Ant⁵ pro sestavování Java programů. Maven ovšem není další verzí, ale zcela novým přístupem, který výrazně zjednodušuje práci programátora i dalších lidí, kteří přijdou do styku se zdrojovými soubory aplikace. Programátor nejprve provede definici svého projektu, kde zadá potřebné údaje, jako je název, verze, autor, skupina a také jeho struktura, zda se jedná o jeden projekt nebo o více vzájemně provázaných modulů. Definici vytvářené aplikace zobrazuje Obrázek 12, který obsahuje rodičovský projekt LPSolver a dva podřízené moduly LPSolver-console a LPSolver-web. Dále se definují jednotlivé závislosti na ostatní knihovny. Všechny tyto definice se uvádí ve formátu xml a souboru pom.xml, který musí být součástí každého Maven modulu. Příklad definice závislosti uvádí Zdrojový kód 1. Veškeré závislosti Maven automaticky získává z internetu, není tedy nutné je stahovat ručně a přesně definovat cestu k jejich úložišti, jako tomu je u nástroje Ant. Knihovny stahované z internetového úložiště se nahrávají do lokálního úložiště v počítači tak, aby nemusely být znovu stahovány při každém sestavování aplikace.

³ <https://netbeans.org/>

⁴ <http://maven.apache.org/>

⁵ <http://ant.apache.org/>



Obrázek 12 – Struktura projektu

Zdrojový kód 1- zápis závislosti v prostředí Maven

```
1 <dependency>
2   <groupId>com.ocpsoft</groupId>
3   <artifactId>prettyfaces-jsf2</artifactId>
4   <version>3.3.3</version>
5 </dependency>
```

5.2 Implementační části a jejich testování

Po implementaci jednotlivých funkčních částí docházelo průběžně k jejich testování, a pokud se v aplikaci vyskytovaly chyby, byly ihned provedeny jejich opravy. Nejprve byla implementována a testována samotná část simplexového algoritmu. Následně byla doplněna kontrola vstupu, možnost zadání úlohy v textovém formátu, řešení úlohy celočíselně, ve zlomcích či v desetinných číslech, řízení chyb úlohy, zobrazení stínových cen a kontrola omezení. Tato funkcionalita je obsažena v samostatně spustitelné aplikaci LPPSolver-console, kterou je také možno využít jako knihovnu pro další projekt. Díky této vlastnosti je funkcionalita použita v aplikaci LPPSolver-web a nedochází tak k redundanci zdrojového kódu. V aplikaci LPPSolver-web bylo dále implementováno a testováno řízení navigace webových stránek, export do formátu PDF, lokalizace a zadání úlohy za pomoci průvodce. Celá aplikace LPPSolver-web byla následně znovu testována jako celek.

5.3 Algoritmy lineárního programování

V následující části budou popsány nejdůležitější části implementovaného kódu v aplikaci pro řešení úloh lineárního programování.

Volba sloupce

Zdrojový kód 2 je algoritmus pro nalezení klíčového sloupce. Metoda *findColum* vrací celočíselnou souřadnici minimální hodnoty posledního řádku simplexové tabulky a k tomu využívá dvě lokální proměnné *col*, což je číslo sloupce s prozatím nejnižší hodnotou v posledním řádku a *mm* která obsahuje prozatím nejnižší hodnotu v posledním řádku. Algoritmus prochází poslední řádek simplexové tabulky a v každém kroku provádí test, zda aktuální hodnota je menší než aktuální minimum. Pokud je nalezena nižší hodnota, provede se kontrola sloupce, zda obsahuje alespoň jednu kladnou hodnotu, pokud i tato podmínka vyhoví, je provedeno přepsání čísla sloupce a aktuální minimální hodnoty. Výchozí hodnotou proměnné *mm* je maximální možné číslo, čímž je zajištěno, že algoritmus vždy najde nižší hodnotu.

Zdrojový kód 2 – algoritmus volby sloupce

```
1 private int findColum() {
2     int col = Integer.MAX_VALUE;
3     double mm = Double.MIN_VALUE;
4     for (int j = 0; j < m.getM()[m.getM().length - 1].length - 1; j++) {
5         if (m.getM()[m.getM().length - 1][j] < mm) {
6             for (int i = 0; i < m.getM().length; i++) {
7                 if (m.getM()[i][j] > 0) {
8                     col = j;
9                     mm = m.getM()[m.getM().length - 1][j];
10                    break;
11                }
12            }
13        }
14    }
15    return col;
16 }
```

Volba řádku

Zdrojový kód 3 zobrazuje metodu *findRow*, která slouží pro nalezení klíčového řádku. Metoda má jeden povinný argument a to je sloupec, ve kterém má hledání probíhat. Metoda vrací celočíselnou hodnotu vybraného řádku a využívá tři lokální proměnné. Proměnnou *row*, která obsahuje číslo optimálního řádku, *minPlus* obsahující hodnotu optimálního podílu a *ds*, která vyjadřuje hodnotu podílu aktuálního řádku. Jak bylo popsáno v teoretické části, nejmenší kladný podíl zvoleného a nejpravějšího sloupce

je řádek obsahující klíčový prvek. Metoda tedy prochází požadovaný sloupec a provádí výpočet podílu pro každý řádek. Je-li tento podíl kladný a menší nežli dosud nejnižší hodnota, je tento podíl nastaven jako dosud nejnižší a uloženo číslo jeho řádku.

Zdrojový kód 3 – algoritmus volby řádku

```
1 private int findRow(int x) {
2     int row = Integer.MAX_VALUE;
3     double minPlus = Double.MAX_VALUE;
4     for (int i = 0; i < m.getM().length - 1; i++) {
5         double ds = m.getM()[i][m.getM()[i].length - 1] / m.getM()[i][x];
6         if (ds >= 0) {
7             if (ds < minPlus) {
8                 minPlus = ds;
9                 row = i;
10            }
11        }
12    }
13    return row;
14 }
```

Transformace řešení

Po volbě klíčového sloupce a řádku přichází na řadu transformace simplexové tabulky na tabulku s vyšší hodnotou optimální funkce. To zajišťuje metoda *modifyTable*, kterou zobrazuje Zdrojový kód 4. Tato metoda má dva povinné argumenty, je to číslo řádku a sloupce. Metoda používá tři lokální proměnné *pivotRowV*, ta obsahuje číslo klíčového řádku, *pivot* to je hodnota klíčového prvku a *koeficient* obsahující hodnotu, kterou je nutné vynásobit klíčový řádek tak, aby po odečtení od modifikovaného řádku v klíčovém sloupci zbyla 0. Algoritmus prochází simplexovou tabulku řádek po řádku a provádí jejich modifikace dle postupů popsaných v teoretické části. Klíčový řádek je pouze dělen klíčovým prvkem a od ostatních řádků jsou odečtené vhodné násobky klíčového řádku.

Zdrojový kód 4 – algoritmus transformace řešení

```
1 private void modifyTable(int row, int col) {
2     double[] pivotRowV = new double[m.getM()[row].length];
3     double pivot = m.getM()[row][col];
4     System.arraycopy(m.getM()[row], 0, pivotRowV, 0, m.getM()[row].length);
5     for (int i = 0; i < m.getM().length; i++) {
6         if (i == row) {
```

```
7   for (int j = 0; j < pivotRowV.length; j++) {
8       m.getM()[i][j] = m.getM()[i][j] / pivot;
9   }
10  } else {
11      double koeficient = (m.getM()[i][col] / pivot);
12      for (int j = 0; j < m.getM()[i].length; j++) {
13          m.getM()[i][j] = m.getM()[i][j] - pivotRowV[j] * koeficient;
14      }
15  }
16  }
17 }
```

Test optimality

Metodu pro test optimality popisuje Zdrojový kód 5. Tato metoda využívá dvou dalších metod, kterými jsou *breakingConstrainRow* a *getLastLineMoM*. První z metod vrací číslo řádku větší než nula, pokud existuje řádek porušující podmínky nezápornosti a v takovémto případě není řešení optimální. V opačném případě proběhne volání druhé z metod, jejímž výstupem je minimální hodnota posledního řádku. Pokud je toto minimum menší než 0, řešení není optimální, v opačném případě se jedná o optimální řešení.

Zdrojový kód 5 – metoda pro test optimality

```
1 private boolean checkIsOptimal() {
2     int row = breakingConstrainRow();
3     if (row >= 0) {
4         return false;
5     }
6     double min = getLastLineMoM();
7     if (min < 0) {
8         return false;
9     }
10    return false;
11 }
```

5.4 Zabezpečení aplikace proti útokům

Jak již bylo zmíněno v kapitole 4.6, systém zabezpečení aplikace proti útokům se musí zaměřit pouze na zneužití aplikace k podstrčení spustitelného kódu, nebo k zpřístupnění dat uložených na serveru.

XSS

Z principu funkce aplikace v jazyce Java je obvykle možné vyloučit podstrčení spustitelného kódu na stranu serveru, jelikož aplikace v jazyce Java pracuje s kompilovaným kódem. Není tedy možné podstrčit část skriptu do obsahu stránky tak, aby ji později server spustil, jak tomu je například v případě vytváření webových aplikací pomocí skriptovacích jazyků. Tato technologie útoku se nazývá XSS (Cross Site Scripting). Útočník také může napadnout klientskou část aplikace, a tak v prohlížeči oběti spouštět skripty, odcizovat uživatelské relace, deformovat stránky, vkládat nepřátelský obsah, přesměřovat uživatele, vkládat malware atd. Aby ale tohoto útoku bylo možné využít, musí být obsah stránky vypisován z datového úložiště, jako je například databáze, kterou tato aplikace neobsahuje. Proti XSS existují dva základní obranné mechanismy, kterými jsou:

1. Kontrola dat odeslaných uživatelem.
2. Překódování výstupů na HTML entity.

I přesto, že aplikace bez těchto mechanismů je odolná proti XSS útokům, oba tyto mechanismy jsou implementovány. Vstupní data jsou kontrolována a při nevhodném formátu je provádění kódu zastaveno, poté je chyba oznámena uživateli. Pokud by jakákoli data obsahovala části skriptu pro manipulaci klientské části, dojde k jejich překódování a zobrazení jako text ve stránce, což zajišťuje JSF komponenta „outputText“, která je využita pro všechny textové výstupy ve stránce.

SQL injection

Dalším velice častým útokem na webové aplikace je útok pojmenovaný SQL injection. Jedná se o útok na databázový systém přes neošetřená vstupní data. Implementovaná aplikace má veškerá používaná vstupní data ošetřena a v současné verzi neobsahuje připojení na databázový systém, proto tedy je před tímto útokem chráněna.

Přístup k ostatním dokumentům aplikace

Aplikace mimo přístupných dat obsahuje také konfigurační soubory a soubory šablon webových stránek, které musí být nedostupné z klientské části a slouží pouze pro běh serveru. Skrytí těchto konfiguračních souborů je provedeno zablokováním přístupu uživatele na určité adresy a složky serveru. Toto zablokování je provedeno v konfiguračním souboru web.xml a zobrazuje jej Zdrojový kód 6.

Zdrojový kód 6 – bezpečnostní restrikce aplikačního serveru

```
1 <!--deny -->
2     <security-constraint>
3         <display-name>Deny pages folder</display-name>
4         <web-resource-collection>
5             <web-resource-name>All Access</web-resource-name>
6             <url-pattern>/faces/pages/*</url-pattern>
7             <url-pattern>/pages/*</url-pattern>
8             <url-pattern>/Tomcat.dpf</url-pattern>
9         </web-resource-collection>
10        <auth-constraint/>
11    </security-constraint>
```

5.5 Uživatelské rozhraní

Uživatelské rozhraní neboli GUI z anglického *Graphical User Interface*, je takové rozhraní mezi strojem a člověkem, které umožňuje uživateli ovládat aplikaci za pomoci interaktivních ovládacích prvků. V případě webové aplikace se jedná o webové rozhraní, které používá standardní webové ovládací prvky, jako jsou textové vstupy, tlačítka, menu a podobně. Výstupem aplikace je webová stránka v jazyce HTML/CSS, která bude zobrazena přímo ve webovém prohlížeči. V případě stažení úlohy ve formátu PDF bude zobrazen dotaz na uložení tohoto datového souboru.

Všeobecné prvky ovládání

Jak zobrazují Obrázek 13, Obrázek 14 a Obrázek 15, každá stránka webové aplikace obsahuje společné prvky. Jsou to hlavička stránky, menu a patička stránky. Hlavička stránky obsahuje nadpis a navigační odkazy pro změnu jazyka, i přesto že prvotní detekce jazyka probíhá automaticky dle nastavení prohlížeče je možné jej ručně změnit. Menu stránky obsahuje navigační odkazy na textové zadání, zadávání pomocí průvodce a stránku o aplikaci. Patička stránky obsahuje informaci o datu vytvoření stránky a odkazy na validátory XHTML a CSS jazyků.

Stránka – zadání pomocí textu

Úvodní stránka webové aplikace pro řešení problému lineárního programování je zároveň stránkou pro textové zadání úlohy a zobrazuje ji Obrázek 13. Tato stránka umožňuje především zadání problému lineárního programování v definovaném textovém formátu, který je na stránce popsán. Formát je definovaný proto, aby aplikace přesně dokázala

rozeznat jednotlivé koeficienty, proměnné, cíl úlohy a sestavit simplexovou tabulku. Dále je možné zvolit typ výpočtu a řešení po krocích.

Web-aplikace pro řešení problému lineárního programování Cs | En

Zadání pomocí textu

Formát zadávání:

- Účelová funkce se zadává ve tvaru $\{v\} = \{ex\}$ subject to $\{cond\}$
- kde $\{type\}$ nabývá hodnot maximize nebo minimize
- kde $\{v\}$ je proměnná jejíž první písmeno je z abecedy $\{a-z\}$
- kde $\{ex\}$ je matematický výraz, který neobsahuje žádné funkce pouze operace $+, -, *, /$
- kde $\{cond\}$ jsou podmínky lineární úlohy oddělené čárkou
 - tvar jedné podmínky je $\{ex\} \{op\} \{n\}$
 - kde $\{ex\}$ je matematický výraz, který neobsahuje žádné funkce pouze operace $+, -, *, /$
 - kde $\{op\}$ je operátor $>, <, >=, <=$ nebo $=$
 - kde $\{n\}$ je nezáporné číslo

Vzorové zadání úlohy: Maximize $p = (1/2)x + 3y + z + 4w$ subject to $x + y + z + w \leq 40, 2x + y - z - w \geq 10, w - y \geq 10$

Maximize $p = (1/2)x + 3y + z + 4w$
 subject to
 $x + y + z + w \leq 40,$
 $2x + y - z - w \geq 10,$
 $w - y \geq 10$

Řešit po krocích: ☐ Typ výpočtu: Simplex desetinná čísla Počítat

application © copyright 2013, Tomas Hampel
 webdesign © copyright 2011, onebusypixel.com W3C XHTML 1.0 W3C CSS

Obrázek 13 – Stránka - textové zadání

Stránka – zadání pomocí průvodce

Tato stránka obsahuje dva kroky a je tedy tvořena jako průvodce. V prvním kroku je uživatel vyzván k zadání počtu proměnných a počtu omezení. Tento krok zobrazuje Obrázek 14. Po stisknutí tlačítka pokračovat, je zobrazena tabulka pro zadání úlohy, kterou zobrazuje Obrázek 15. Do této tabulky uživatel vyplní hodnoty koeficientů proměnných v jednotlivých nerovnicích, vybere cíl optimalizace a zvolí znaménka nerovnic. Tento způsob zadání úlohy je tedy komfortnější, protože je nutné zadávat pouze čísla a jediné omezení, co je nutné dodržovat, je formát vstupních čísel. Umožněno je zadat celé číslo, desetinné číslo, nebo zlomek. Dále je možné zvolit typ výpočtu a řešení po krocích.

Obrázek 14 – Stránka – zadání pomocí průvodce část 1

Obrázek 15 – Stránka – zadání pomocí průvodce část 2

Výpočet a výsledek

Výpočet probíhá pomocí simplexové metody a je tedy zapsán ve tvaru přehledných simplexových tabulek. Tento výstup zobrazuje Obrázek 16. Na konci výpočtu za výslednou simplexovou tabulkou je zobrazena tabulka s hodnotami jednotlivých proměnných, výsledek dosazení do nerovnic a zobrazení rezervy jejich omezení (nedočerpání omezení), a tabulka s přehledem hodnot stínových cen. Veškerá čísla v simplexových tabulkách jsou zobrazena dle metody výpočtu, tedy v desetinných číslech,

zlomcích nebo celočíselně. Pod výpočtem jsou navigační odkazy na zobrazení předchozího či následujícího kroku a možnost stažení celé úlohy ve formátu PDF.

Tabulka 1 :

x	y	z	w	z0	z1	z2	p	
1	1	1	1	1	0	0	0	40
-2	-1	1	1	0	1	0	0	-10
0	1	0	-1	0	0	1	0	-10
-0.5	-3	-1	-4	0	0	0	1	0

Tabulka 2 :

x	y	z	w	z0	z1	z2	p	
0	0.5	1.5	1.5	1	0.5	0	0	35
1	0.5	-0.5	-0.5	0	-0.5	0	0	5
0	1	0	-1	0	0	1	0	-10
0	-2.75	-1.25	-4.25	0	-0.25	0	1	2.5

Tabulka 3 :

x	y	z	w	z0	z1	z2	p	
0	2	1.5	0	1	0.5	1.5	0	20
1	0	-0.5	0	0	-0.5	-0.5	0	10
0	-1	0	1	0	0	-1	0	10
0	-7	-1.25	0	0	-0.25	-4.25	1	45

Tabulka 4 :

x	y	z	w	z0	z1	z2	p	
0	1	0.75	0	0.5	0.25	0.75	0	10
1	0	-0.5	0	0	-0.5	-0.5	0	10
0	0	0.75	1	0.5	0.25	-0.25	0	20
0	0	4	0	3.5	1.5	1	1	115

Řešení :

x	y	z	w	z0	z1	z2	p
10	10	0	20	0	0	0	115

Zkouška :

40 <= 40.0

10 >= 10.0

10 >= 10.0

Rezerva omezení :

0

0

0

Střhové ceny :

w	z0	z1	z2
0.0	3.5	1.5	1.0

[předchozí krok](#)

[stáhnout v PDF](#)

application © copyright 2013, Tomas Hampel
webdesign © copyright 2011, onebusypixel.com

W3C XHTML 1.0 W3C CSS

Obrázek 16 – Výpočet a výsledek úlohy

6 PŘÍKLADY VYBRANÝCH PROBLÉMŮ

V následující kapitole bude vyřešeno několik slovních úloh za pomoci webové aplikace. Tyto úlohy slouží pro závěrečné otestování a také pro demonstraci fungování aplikace. Veškeré příklady byly vypočteny primárně ve vytvořené aplikaci a pro kontrolu také v aplikacích Simplex method tool [18] a Simplex Me [19]. Výsledky ze všech tří aplikací se shodovaly, tudíž jsou považovány za správné. Správnost sestavených matematických modelů byla ověřena dle učebních textů pana Ženčáka [9].

6.1 Příklad č.1 – Pivovar v USA

Slovní zadání:

Pivovar v USA potřebuje optimalizovat svou produkci, která je tvořena dvěma druhy amerického piva – světlým a tmavým. Recept na oba druhy je rozdílný a vyžaduje různé množství vstupních surovin. Oba druhy také přináší různý zisk. Složení obou druhů piva, zisk za jeden barel a omezující limity množství surovin jsou v následující tabulce.

Tabulka 13 – Tabulka k zadání úlohy č. 1

Druh piva	Obilí [libry]	Chmel[unce]	Slad[libry]	Zisk[\$]
Světlé pivo (na 1 barel)	5	4	35	13
Tmavé pivo (na 1 barel)	15	4	20	23
Limit na suroviny	480	160	1190	

Cílem je najít takové složení výroby, které maximalizuje zisk [9].

Formulace úlohy:

$$\begin{aligned}
 \max L(x) &= 13x_1 + 23x_2 & (15) \\
 5x_1 + 15x_2 &\leq 480 \\
 4x_1 + 4x_2 &\leq 160 \\
 35x_1 + 20x_2 &\leq 1190 \\
 x_1, x_2 &\geq 0
 \end{aligned}$$

Textový vstup pro aplikaci LPPSolver:

```

Maximize p = 13x1+23x2
subject to
5x1+15x2≤480
4x1+4x2≤160
35x1+20x2≤1190

```

Výsledek výpočtu:

Tabulka 14 – Výsledek výpočtu příkladu č. 1

x1	x2	z0	z1	z2	p
12	28	0	0	210	800

Tabulka 15 – Zkouška a rezerva omezení příkladu č. 1

Zkouška:	Rezerva omezení:
480 ≤ 480	0
160 ≤ 160	0
980 ≤ 1190	210

Tabulka 16 – Stínové ceny příkladu č.1

z1	z2
2	0

Výrobní plán, který přinese maximální zisk 800\$, je výroba 12 barelů světlého a 28 barelů tmavého piva. Při tomto plánu bude spotřeba obilí 480 liber, spotřeba chmelu 160 liber a spotřeba sladu 980 liber.

6.2 Příklad č. 2 – Dopravní problém

Slovní zadání:

Firma pěstuje jablka ve třech sadech. Smlouvu o výkupu podepsala se dvěma ovocnými družstvy. Vzdálenost mezi sady a družstvy, maximální možné množství sklizeného ovoce v jednotlivých sadech a kapacity družstev jsou uvedeny v následující tabulce. Úkolem je minimalizovat přepravní náklady dané počtem ujetých kilometrů tak, aby byla ze sadů odvezena veškerá úroda a naplněny kapacity družstev [9].

Tabulka 17 – Tabulka k zadání úlohy č. 2

	Družstvo č.1	Družstvo č.2	Úroda
Sad č.1	22	15	170
Sad č.2	20	40	130
Sad č.3	28	35	200
Kapacita	150	350	

Formulace úlohy:

$$\begin{aligned} \min L(x) &= 22x_{11} + 15x_{12} + 20x_{21} + 40x_{22} + 28x_{31} + 35x_{32} \quad (16) \\ x_{11} + x_{21} + x_{31} &= 150 \\ x_{12} + x_{22} + x_{32} &= 350 \\ x_{11} + x_{12} &= 170 \\ x_{21} + x_{22} &= 130 \\ x_{31} + x_{32} &= 200 \end{aligned}$$

Textový vstup pro aplikaci LPPSolver:

Minimize p=22x11+15x12+20x21+40x22+28x31+35x32
 subject to
 x11+x21+x31=150
 x12+x22+x32=350
 x11+x12=170
 x21+x22=130
 x31+x32=200

Výsledek výpočtu:

Tabulka 18 – Výsledky výpočtu příkladu č. 2

x11	x12	x21	x22	x31	x32	p
0	170	130	0	20	180	12010

Tabulka 19 – Zkouška a rezerva omezení příkladu č. 2

Zkouška:	Rezerva omezení:
150 = 150.0	0
350 = 350.0	0
170 = 170.0	0
130 = 130.0	0
200 = 200.0	0

Tabulka 20 – Stínové ceny příkladu č. 2

X32
0

Výsledný plán přepravy jablek do jednotlivých družstev je následující. Ze sadu č.1 se převeze 0 jablek do družstva č. 1 a 170 do družstva č.2. Ze sadu č. 2 bude převezeno 130 jablek do družstva č. 1 a 0 do družstva č. 2. Nakonec ze sadu č. 3 se převeze 20 jablek do družstva č. 1 a 180 do družstva č. 3. Výsledný počet je 12 010 ujetých kilometrů.

6.3 Příklad č. 3 – Problém nutričních hodnot

Slovní zadání:

Pacient by měl denně sníst potraviny o úhrnné energetické hodnotě alespoň 12 000 kJ. Strava by měla obsahovat minimálně 475 gramů sacharidů, 67 gramů bílkovin a 83 gramů tuků. Kilogram potravin A má 3 110 kJ, obsahuje 0 g sacharidů, 200 g bílkovin a 0 g tuků a stojí 159 Kč. Kilogram potravin B má 9 630 kJ a obsahuje 500 g sacharidů, 0 g bílkovin a 0 g tuků a stojí 20 Kč. Kilogram potravin C má 2 000 kJ, obsahuje 0 g sacharidů, 0 g bílkovin a 100 g tuků, stojí 40 Kč. Kilogram potravin D má 20 000 kJ, obsahuje 0 g sacharidů, 60 g bílkovin a 830 g tuků, stojí 200 Kč. Sestavte z těchto surovin nejlevnější dietní plán zachovávající požadovaná omezení [9].

Formulace úlohy:

$$\begin{aligned}
 \min L(x) &= 159a + 20b + 40c + 200d & (17) \\
 500b &\geq 475 \\
 200a + 60d &\geq 67 \\
 100c + 830d &\geq 83 \\
 3110a + 9630b + 2000c + 20000d &\geq 12000 \\
 a, b, c, d &\geq 0
 \end{aligned}$$

Textový vstup pro aplikaci LPPSolver:

```

minimize p=159a+20b+40c+200d
subject to
500b≥475
200a+60d≥67
100c+830d≥83
3110a+9630b+2000c+20000d≥12000

```

Výsledek výpočtu:

Tabulka 21 – Výsledky výpočtu příkladu č. 3

a	b	c	d	z0	z1	z2	z3	p
0.305	0.95	0	0.1	0	0	0	97.05	87.495

Tabulka 22 – Zkouška a rezerva omezení příkladu č. 3

Zkouška:	Rezerva omezení:
475 >= 475.0	0
67 >= 67.0	0
83 >= 83.0	0
12097.05 >= 12000.0	-97.05
475 >= 475.0	0

Tabulka 23 – Stínové ceny příkladu č. 3

z0	z1	z2	z3
0.04	0.795	0.183494	0

Nejlevnější dietní plán je sestaven z 0,305 gramů potravin A, 0,95 gramů potravin B, 0 gramů potravin C a 0,1 gramů potravin D. Celková denní cena činí 87,495 Kč.

ZÁVĚR

Při řešení diplomové práce byla prozkoumána problematika optimalizačních metod a to zejména úloh lineárního programování. Hluběji byla prozkoumána simplexová metoda, která je nejčastěji používanou metodou při řešení tohoto typu úloh.

Teoretická část dále obsahuje zmínky o historii optimalizace, popis optimalizační úlohy a její základní typy. Na informace o optimalizaci navazuje podrobnější popis úloh lineárního programování, metodika sestavování jejich modelu, definice kanonického tvaru a postupy pro převod různých tvarů úloh do kanonického. Zkráceně je popsán postup grafického řešení a podrobněji krok po kroku algoritmus simplexové metody. V závěru teoretické části je popsán význam pojmů: duální úloha, duální cena, nedočerpaní omezení a citlivostní analýza. Tyto pojmy úzce souvisí s lineárním programováním, a proto jsou v práci popsány.

Cílem práce bylo především vytvoření web-aplikace pro řešení problému úloh lineárního programování, jejíž realizace je podrobně popsána v praktické části včetně informací o použitých nástrojích a knihovnách.

Výsledkem práce je vytvořená aplikace využívající algoritmus zmiňované simplexové metody. Aplikace umožňuje uživateli zadat úlohu v textovém formátu či formou tabulky, kde nejprve zvolí počet proměnných tedy sloupců a následně počet omezujících podmínek, tedy řádků. Aplikace umožňuje uživateli také zvolit zobrazování řešení po krocích, či jednorázově a vybrat metodu výpočtu celočíselně, v desetinných číslech nebo ve zlomcích. Po odeslání požadavku aplikace provede výpočet řešení, a pokud existuje, je uživateli zobrazeno spolu s možností stažení úlohy ve formě PDF. V případě že řešení neexistuje, či jsou data zadána chybně, je uživateli zobrazeno patřičné chybové hlášení.

Vlastním přínosem je realizace zobrazování kontroly správnosti výpočtu, hodnoty nedočerpaní omezení a stínové ceny. Tyto hodnoty se zobrazují spolu s výsledkem výpočtu.

Aplikace je vytvořená s použitím moderních technologií klient-server. Pro vývoj serverové aplikace jsou to především Java EE, JSF 2.0, Pretty Faces a klientská část využívá technologie XHTML a CSS3.

Během vytváření webové aplikace byla řešena řada překážek a chyb, ať již s použitými technologiemi, tak s algoritmy simplexové metody. Největším problémem bylo řešení

podmínek nezápornosti, protože bez implementace odstranění tohoto porušení úloha často sice řešení najde, ale to však porušuje omezující podmínky a není tedy platným řešením.

Web aplikaci je možné dále rozšířit o další metody řešení úloh lineárního programování nebo o uživatelské účty, které by umožňovaly přihlášeným uživatelům ukládat řešené úlohy.

Závěrem lze konstatovat, že zadání se zdařilo realizovat v plném rozsahu.

CONCLUSION

The issues of optimization methods, particularly linear programming problems, had been explored. The simplex method, which is the most frequently used method when solving this type of tasks, was deeply researched.

The theoretical part includes references related to the history of optimization, description of the optimization task and its basic types. The information about optimization are followed by detailed description of linear programming tasks, methodology of its model compilation, definition of canonical form and procedures to transfer various tasks to canonical one. While the procedure of graphic solution is described rather briefly, the algorithm of the simplex method is elaborated in more details. In the end of the theoretical part the following terms are explained: dual problem, dual price, un-drawing of limits and sensitivity analysis. These terms are closely related with linear programming and therefore they are mentioned and described in the thesis.

The aim of this work was mainly to create the problem solving web application for linear programming tasks; the realization of which is described in details in the practical part including the information about the tools and sources used.

The result of the thesis is the created application which uses the algorithm of the mentioned simplex method. The application enables the user to set a task in the form of text or a table, where he/she chooses the number of variables - columns - and then the number of the limiting conditions - rows. The application allows the user to choose also the manner how the solution is displayed - either by steps or all at once, and also to choose method of the calculation in whole numbers, decimal numbers or fractions. After sending off the request, the application carries out the solution and if it exists, it is displayed together with the possibility of downloading the task in PDF format. In case that the solution does not exist, or the data are incorrectly entered, the appropriate error message is displayed.

The main contribution is the checking display of the correct calculation, the values of un-drawing of limits and shadow prices. These figures are displayed together with the calculation result.

The application is created using the modern client-server technology. For the development of the server application Java EE, JSF 2.0, Pretty Faces were used and the client part uses the technologies XHTML and CSS3.

During the creation of the web application a number of several obstacles and errors were solved both problems with the used technologies and with the algorithm of the simplex method. The biggest issue was to solve the conditions of non-negativity because without the implementation of the elimination of this violation, the task often finds the solution, however it breaks the limited conditions and therefore the solution is not valid.

The Web application can be furthermore extended with methods of solving the linear programming tasks or with user accounts that would allow the registered users to store the solved tasks.

Finally, we can state that the assignment was successfully accomplished in its full extent.

SEZNAM POUŽITÉ LITERATURY

- [1] PAVLÍKOVÁ, Pavla. *Z historie lineárního programování George B. Dantzig (1914–2005)*. Praha: Jednota českých matematiků a fyziků, 2008.
- [2] DEMEL, Jiří. *Operační výzkum 1*. Praha: České vysoké učení technické, 2007.
- [3] KOŘENÁŘ, Václav a Milada LAGOVÁ. *Optimalizační metody*. Vyd. 1. V Praze: Oeconomica, 2003, 187 s. ISBN 80-245-0609-2.
- [4] GAJDA, Bohumil. *Speciální metody optimalizace*. Zlín, 2009. Diplomová práce. UTB ve Zlíně, Fakulta aplikované informatiky.
- [5] OBORNÝ, Petr. *Simplexový algoritmus pro rozsáhlé úlohy lineárního programování*. Ostrava, 2007. Bakalářská práce. VŠB – Technická univerzita Ostrava Fakulta elektrotechniky a informatiky.
- [6] ŠVRČEK, Jaroslav. *Lineární programování v úlohách*. 2., přeprac. vyd. Olomouc: Univerzita Palackého, 2003, 103 s. Skripta (Univerzita Palackého). ISBN 80-244-0705-1.
- [7] LINDA, Bohdan a Josef VOLEK. *Lineární programování*. Vyd. 3. Pardubice: Univerzita Pardubice, 2009, 139 s. ISBN 978-80-7395-207-5.
- [8] HARSHBARGER, Ronald J. *Mathematical applications for the management, life, and social sciences*. 10 edition. Boston: Wadsworth Publishing Co Inc, 2008, 1 volume (various pagings). ISBN 11-331-0847-4.
- [9] ŽENČÁK, Pavel. *Texty k přednáškám z lineárního programování*. Olomouc: Universita Palackého v Olomouci, 2011.
- [10] LARSON, Ron. *Elementary linear algebra*. 7th ed. Boston, MA: Brooks/Cole, Cengage Learning, 2013, 1 v. (various paging). ISBN 11-331-1087-8.
- [11] OCPSOFT. *Pretty URLs for JavaServer Faces and Java Application Servers* [online]. 2013 [cit. 2013-05-09]. Dostupné z: <http://ocpssoft.org/prettyfaces/>
- [12] WOLFRAM RESEARCH. *Wolfram Mathematica* [online]. 2013 [cit. 2013-05-09]. Dostupné z: <http://www.wolfram.com/mathematica/>
- [12] WOLFRAM RESEARCH. *Wolfram Mathematica* [online]. 2013 [cit. 2013-05-09]. Dostupné z: <http://www.wolfram.com/mathematica/>
- [13] MATHWORKS. *MATLAB - The Language of Technical Computing* [online]. 2013 [cit. 2013-05-09]. Dostupné z: <http://www.mathworks.com/products/matlab/>

- [14] MAPLESOFT. *Technical Computing Software – Maple 17* [online]. 2013 [cit. 2013-05-09]. Dostupné z: <http://www.maplesoft.com/products/Maple/professional/>
- [15] LINDO SYSTEMS INC. *LINDO Systems - Optimization Software* [online]. 2013 [cit. 2013-05-09]. Dostupné z: <http://www.lindo.com/>
- [16] *Linear Program Solver* [online]. 2013 [cit. 2013-05-09]. Dostupné z: <http://sourceforge.net/projects/lipside/>
- [17]. *Lpsolver* [online]. 2009 [cit. 2013-05-09]. Dostupné z: <https://code.google.com/p/lpsolver/>
- [18] Simplex Method Tool. [online]. 2010 [cit. 2013-05-09]. Dostupné z: <http://www.zweigmedia.com/RealWorld/simplex.html>
- [19] Simplex me - the simple simplex solver. [online]. 2009 [cit. 2013-05-09]. Dostupné z: <http://www.simplexme.com/en/>
- [online]. [cit. 2013-05-09].
- [20] Simplex On Line Calculator. *Mathstools* [online]. 2004 [cit. 2013-05-09]. Dostupné z: <http://www.mathstools.com/section/main/SimplexOnLine>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CSS	Kaskádový styl.
EJB	Serverové komponenty umožňující modulární tvorbu aplikací.
GC	Automatický správce paměti.
GUI	Uživatelské rozhraní.
HTML	Hypertextový značkovací jazyk.
HTTP	Hypertextový přenosový protokol.
HTTPD	HTTP démon.
HTTPS	Bezpečné HTTP.
IDE	Vývojové prostředí.
Java EE	Pokročilá edice Javy.
JPA	Java framework pro práci s perzistentními objekty.
JSF	Framework pro práci s GUI v Java EE.
JSP	Nástroj pro tvorbu dynamických stránek v Java EE.
JVM	Virtuální stroj platformy Java.
LP	Lineární programování.
LPP	Problém lineárního programování.
PDF	Formát přenosného dokumentu.
PHP	PHP hypertextový procesor.
SQL	Standardizovaný dotazovací jazyk.
URL	Jednotný lokátor zdrojů.
USA	Spojené státy americké.
XHTML	Rozšiřitelný hypertextový značkovací jazyk.
XML	Rozšiřitelný značkovací jazyk.
XSS	Metoda narušení WWW stránek využitím bezpečnostních chyb ve skriptech.

SEZNAM OBRÁZKŮ

Obrázek 1 – Grafická reprezentace oblasti přípustných řešení.....	17
Obrázek 2 – Nekonvexní množina.....	26
Obrázek 3 – Konvexní množina	26
Obrázek 4 – Příklad řešení dvourozměrné úlohy LP	28
Obrázek 5 – vývojový diagram algoritmu simplexové metody.....	29
Obrázek 6 – Diagram případů užití.....	41
Obrázek 7 – Vazba Java SE a Java EE	45
Obrázek 8 – Celková architektura aplikace	45
Obrázek 9 – Propojení aplikačního a webového serveru.....	47
Obrázek 10 – Životní cyklus požadavku na Servlet JSF	48
Obrázek 11 – Algoritmus řešení úlohy včetně zobrazení kroků.....	49
Obrázek 12 – Struktura projektu.....	51
Obrázek 13 – Stránka - textové zadání	57
Obrázek 14 – Stránka – zadání pomocí průvodce část 1	58
Obrázek 15 – Stránka – zadání pomocí průvodce část 2	58
Obrázek 16 – Výpočet a výsledek úlohy	59
Obrázek 17 – Nastavení ISAPI A CGI	78
Obrázek 18 – Nastavení filtru ISAPI.....	78

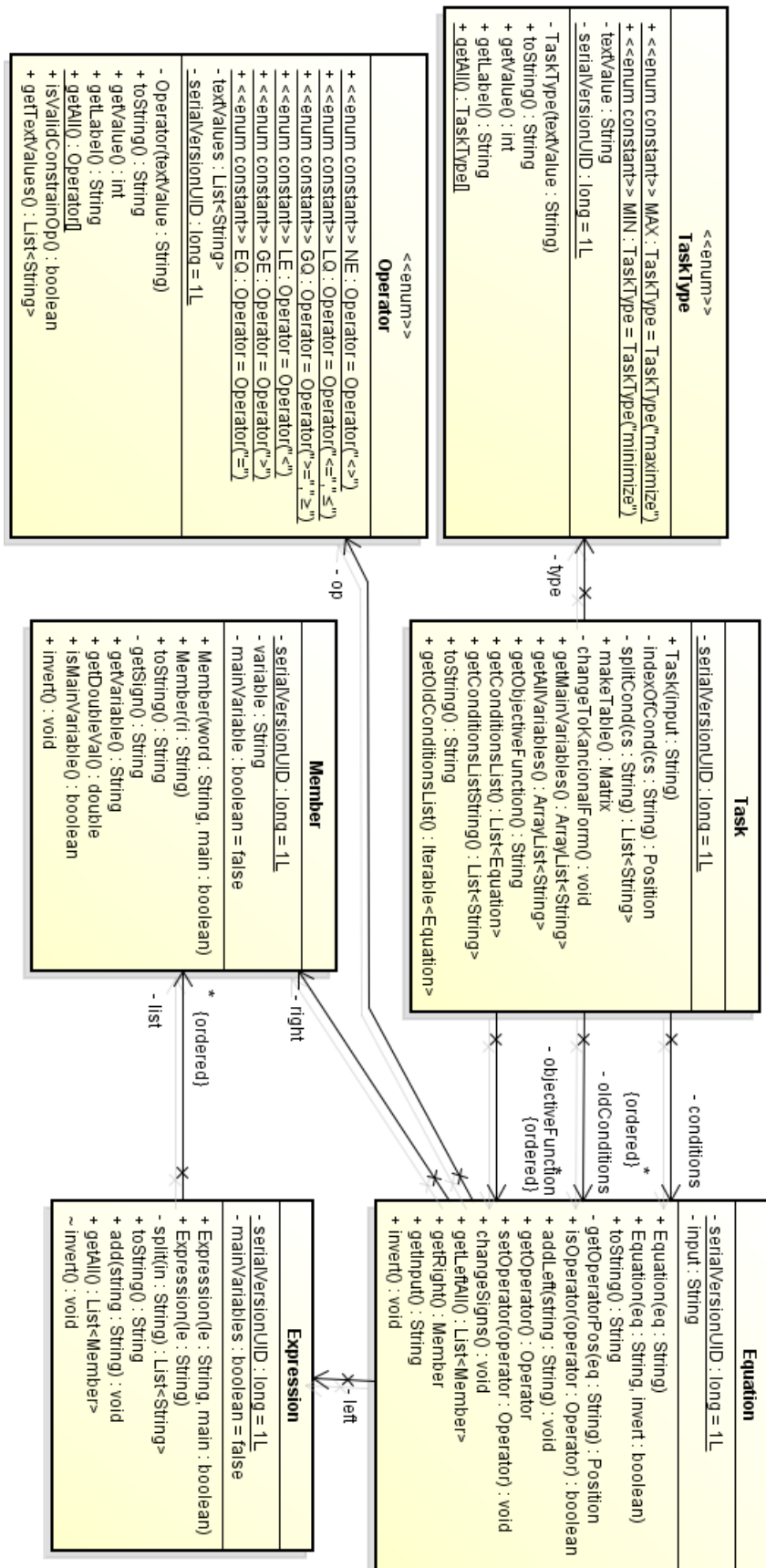
SEZNAM TABULEK

Tabulka 1 – Vztah matematického a verbálního modelu	20
Tabulka 2 – Obecná simplexová tabulka	30
Tabulka 3 – Příklad sestavení simplexové tabulky	30
Tabulka 4 – Příklad volby sloupce v simplexové tabulce	31
Tabulka 5 – Příklad volby sloupce v simplexové tabulce	32
Tabulka 6 – Příklad transformace řešení	33
Tabulka 7 – Výsledná simplexová tabulka	33
Tabulka 8 – Volba klíčového prvku v případě porušení podmínky nezápornosti	35
Tabulka 9 – Scénář případu užití změny jazyku stránky	41
Tabulka 10 – Scénář případu užití zadání úlohy	42
Tabulka 11 – Scénář případu užití zobrazení výsledku	43
Tabulka 12 – Scénář případu užití stažení vypočtené úlohy ve formátu PDF	43
Tabulka 13 – Tabulka k zadání úlohy č. 1	60
Tabulka 14 – Výsledek výpočtu příkladu č. 1	61
Tabulka 15 – Zkouška a rezerva omezení příkladu č. 1	61
Tabulka 16 – Stínové ceny příkladu č. 1	61
Tabulka 17 – Tabulka k zadání úlohy č. 2	61
Tabulka 18 – Výsledky výpočtu příkladu č. 2	62
Tabulka 19 – Zkouška a rezerva omezení příkladu č. 2	62
Tabulka 20 – Stínové ceny příkladu č. 2	62
Tabulka 21 – Výsledky výpočtu příkladu č. 3	63
Tabulka 22 – Zkouška a rezerva omezení příkladu č. 3	64
Tabulka 23 – Stínové ceny příkladu č. 3	64

SEZNAM PŘÍLOH

- PI Diagram tříd reprezentujících úlohu
- PII Instalační manuál pro provoz aplikace na serveru Microsoft IIS
- PIII Instalační manuál pro provoz aplikace na serveru Apache HTTPD
- PIV CD se softwarem

PŘÍLOHA P I: DIAGRAM TŘÍD REPREZENTUJÍCÍCH ÚLOHU



PŘÍLOHA P II – INSTALAČNÍ MANUÁL PRO PROVOZ APLIKACE NA SERVERU MICROSOFT IIS

Instalace platformy Java

Ze stránek platformy Java⁶ stáhněte instalační balíček JDK, spusťte a proved'te instalaci dle průvodce.

Instalace Apache Tomcat

Ze stránek projektu Apache Tomcat⁷ stáhněte balíček *service installer* a proved'te instalaci dle průvodce.

Nasazení aplikace.

V adresáři nainstalovaného aplikačního serveru vyhledejte složku webapps např. `C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps`

Do této složky nakopírujte war archiv aplikace, aplikační server poté automaticky provede její rozbalení a spuštění. Aplikace následně bude dostupná na adrese `http://localhost:8080/název_archivu`. Aplikaci je možné nasadit také na kořen aplikačního serveru. Pokud existuje složka ROOT, je nutné ji nejprve smazat, a následně archiv aplikace přejmenovat na `ROOT.war`. Veškeré tyto činnosti je vhodné provádět při vypnutém aplikačním serveru a po nakopírování archivu provést jeho spuštění.

Konfigurace IIS

Celá konfigurace je podrobněji popsána v manuálu Apache Tomcat⁸. Nejprve je nutné stáhnout konektor ze stránek projektu, konkrétně ze sekce connectors⁹. Tento konektor uložte do složky isapi serveru Apache Tomcat:

`C:\Program Files\Apache Software Foundation\Tomcat 7.0\isapi`

Konektor čte nastavení ze souboru uloženého ve stejném adresáři, tedy:

`C:\Program Files\Apache Software Foundation\Tomcat 7.0\isapi`

⁶ <http://java.com/>

⁷ <http://tomcat.apache.org/>

⁸ <http://tomcat.apache.org/connectors-doc/reference/iis.html>

⁹ <http://tomcat.apache.org/download-connectors.cgi>

Jehož obsah je následující:

```
extension_uri = /jakarta/isapi_redirect.dll  
log_file = C:\Program Files\Apache Software Foundation\Tomcat 7.0\logs\isapi_redirect.log  
log_level = error  
worker_file = C:\Program Files\Apache Software Foundation\Tomcat  
7.0\conf\workers.properties.minimal  
worker_mount_file = C:\Program Files\Apache Software Foundation\Tomcat  
7.0\conf\uriworkermap.properties
```

Dále ve složce: C:\Program Files\Apache Software Foundation\Tomcat 7.0\conf

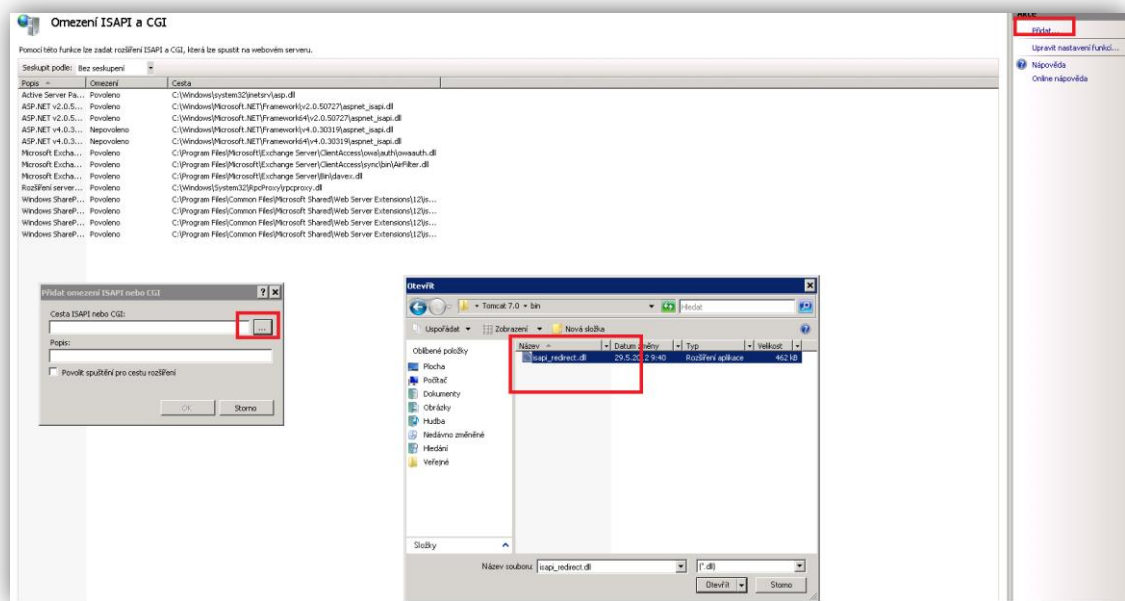
Vytvořte soubor workers.properties.minimal, obsahující:

```
worker.list = worker1  
worker.worker1.host=localhost  
worker.worker1.port=8009  
worker.worker1.type=ajp13
```

a soubor uriworkermap.properties, obsahující:

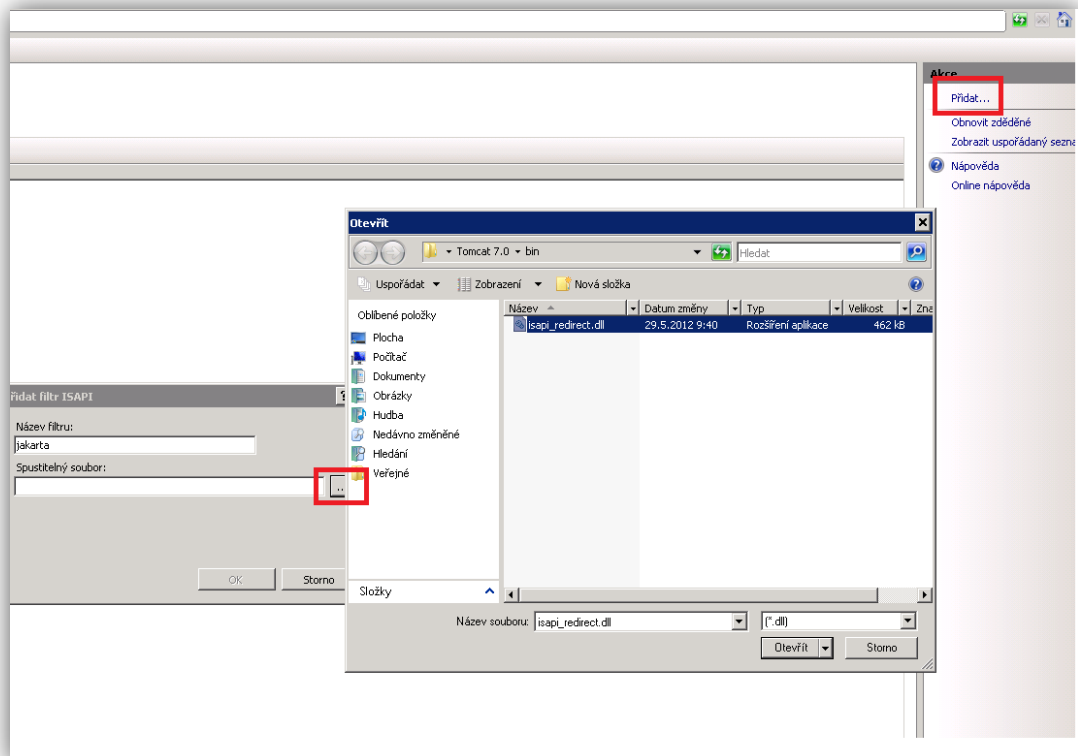
```
/*=worker1
```

Následně v administraci IIS serveru v sekci ISAPI a CGI přidejte nové omezení a jak je zobrazeno na následujícím obrázku vyberte uložený konektor.



Obrázek 17 – Nastavení ISAPI A CGI

Toto omezení pojmenujte *isapi_redirect* a povolte rozšířené cesty. Následně v sekci Weby zvolte defaultní web service a filtry ISAPI. Následně zvolte Přidat a opět vyberte konektor tak, jak je zobrazeno na následujícím obrázku a filtr pojmenujte *jakarta*.



Obrázek 18 – Nastavení filtru ISAPI

Následně pravým kliknutím na Default web site zvolte Přidat virtuální adresář, jehož název zadejte opět *jakarta* a cestu ke složce isapi serveru Tomcat. Nyní otevřete nově vzniklou složku *jakarta* a v ní zvolit sekci mapování obslužných rutin, kde pomocí rychlé volby, upravte oprávněné akce a povolte provádění skriptů.

Nakonec stačí restartovat obě serverové služby a nastavení je dokončeno.

PŘÍLOHA P III: INSTALAČNÍ MANUÁL PRO PROVOZ APLIKACE NA SERVERU APACHE HTTPD

Následující manuál předpokládá nainstalovaný a funkční webový server Apache HTTPD a Apache Tomcat. Servery budou v provozu oba dva. HTTPD bude na standardním portu 80 či 443 a určité požadavky bude předávat serveru Apache Tomcat.

Instalace mod_jk

Mimo zmíněných serverů je třeba doinstalovat i mod_jk do Apache HTTPD což na Linux distribuce Debian provedete následujícím příkazem:

```
apt-get install libapache2-mod-jk
```

Konfigurace Apache Httpd

Na Debian Linux automaticky proběhne i jeho aktivace vytvořením simlinku v adresáři mods-enabled. Pokud se toto nestane, je nutné přidat do konfiguračního souboru následující řádek:

```
LoadModule jk_module /usr/lib/apache2/modules/mod_jk.so
```

Dále je nutné nastavit konfiguraci Jk_workeru, která se nachází v souboru /etc/apache2/workers.properties kam přidáme následující řádky:

```
workers.tomcat_home=/usr/lib/apache-tomcat
workers.java_home=/usr/lib/jvm/default-java
ps=/
worker.list=worker1
worker.default.port=8009
worker.default.host=localhost
worker.default.type=ajp13
worker.default.lbfactor=1
```

Následně do konfigurace Apache mimo sekci VirtualHost nastavit konfiguraci workeru:

```
JkWorkersFile /etc/apache2/workers.properties
```

Nakonec zbývá nastavit mapování URL, tedy které URL směřovat na Apache Tomcat, což zajistí následující dva řádky:

```
JkMount /lpp worker1
JkMount /lpp/* worker1
```


Tímto je zajištěno, že veškeré URL začínající znaky lpp budou předány na vyřízení serveru Apache Tomcat.

Konfigurace Apache Tomcat

V konfiguraci Apache Tomcat je třeba ještě povolit AJP connector. Tato konfigurace je definována v souboru `/etc/tomcat6/server.xml` kde je následující konfigurační řádek obvykle zakomentována a stačí jej pouze povolit.

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8080" />
```

Nakonec stačí restartovat obě serverové služby a nastavení je dokončeno.