TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

WEBOVÉ ROZHRANÍ PRO PROGRAM LP SOLVE

Bakalářská práce

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 1802R022 – Informatika a logistika

Autor práce: **Daniel Pišna**
Vedoucí práce: Ing. Petr Rálek, Ph.D.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

WEB INTERFACE FOR LP SOLVE SOFTWARE

Bachelor thesis

Study programme: B2612 – Electrical Engineering and Informatics
Study branch: 1802R022 – Informatics and Logistics

Author: **Daniel Pišna**
Supervisor: Ing. Petr Rálek, Ph.D.

Liberec 2014

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Daniel Pišna**
Osobní číslo:	**M10000044**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Informatika a logistika**
Název tématu:	**Webové rozhraní pro program LP Solve**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se s teorií a výpočetními metodami lineárního programování.
2. Seznamte se se softwarem Lpsolve a možností použití jeho knihoven v různých programovacích jazycích.
3. Vytvořte v jazyce php webové uživatelské rozhraní pro zadávání a řešení úloh lineárního programování za použití knihoven Lpsolve.
4. Formulujte obecnou úlohu optimalizace provozu výrobních linek podle různých kritérií.
5. Otestujte online provoz aplikace na této úloze.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **cca 45 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **RÁLEK, Petr. Operační výzkum. Elektronická skripta, Liberec, 2008. 26 s.**
[2] **MAŇAS, Miroslav. Optimalizační metody. Praha: SNTL, 1979.**
[3] **lpsolve, http://sourceforge.net/projects/lpsolve/**

Vedoucí bakalářské práce: **Ing. Petr Rálek, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání bakalářské práce: **21. října 2013**

Termín odevzdání bakalářské práce: **16. května 2014**

V. Kopecký

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 21. října 2013

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Děkuji tímto Ing. Petru Rálkovi, Ph.D. za vedení mé bakalářské práce, za cenné rady, podněty a připomínky.

Abstrakt

Práce popisuje stavbu webové aplikace využívající jazyka PHP, která bude uživateli umožňovat využívat funkce programu LP Solve, k řešení úloh lineárního programování. V prvních částech jsou popsány základní informace o lineárním programování, o metodách řešení takových úloh a zmíněny simplexová metoda a metoda větví a mezí, kterých LP Solve ve svých výpočtech využívá. Další část práce představuje LP Solve a popisuje možnosti, jakými lze využít jeho funkcí. V třetí části bakalářské práce je popsána struktura webového rozhraní, je vysvětleno, jakým způsobem se zprostředkovávají funkce LP Solve API a jakým způsobem lze funkčnost tohoto procesu otestovat. Dále je popsán obecný tvar úloh, se kterými aplikace pracuje a následuje popis řešení práce s nimi. Hlavní pozornost je věnována úloze optimalizace výrobních linek, která hledá optimální výrobní plán pro určitý počet linek, které mají při daných omezeních svých výrobních kapacit uspokojit poptávku po jimi vyráběných produktech.

Klíčová slova: lineární programování, LP Solve, optimalizace procesu výrobních linek, webové rozhraní, LP Solve API

Abstract

This thesis describes building of the web application using language PHP, which will provide functions of software LP Solve for the user, for the purpouses of solving the linear programing problems. In the first parts of thesis there are described basic informations abou linear programing, the methods of solvig these problems and there are mentioned the simplex method and the method of brounch and bounds, which are used by the LP Solve in its calculations. The next part of thesis represents the LP Solve and the options of how its functions can be used. In the third part of thesis a structure of the web interface is described, there is explained by which methods are functions of LP Solve API provided and how can this process be tested. Then there is described a general form of the problems with which the aplication operates and that is followed by description of how are these problems handled. The main focus is given to the problem of the production lines process optimization, which looks for the optimum production plan of the certain count of lines, which has to satisfy the demand despite the line's capacity limitations.

Klíčová slova: linear programing, LP Solve, production lines process optimization, web interface, LP Solve API

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů

$\boldsymbol{A} \in R^{n,m}$ - matice o n×m prvcích,

a_{ij} – j-tá složka i-tého řádku matice $\boldsymbol{A}$

$\boldsymbol{x} \in R^{n}$ - vektor o n prvcích,

x_i – i-tá složka vektoru $\boldsymbol{x}$,

$\hat{n}$ – množina prvků o n prvcích jdoucích od 0 do $(n-1)$,

Úvod

Účelem této práce je vytvořit webové rozhraní pro software LP Solve, který slouží k řešení úloh lineárního programování. Hlavním úkolem webového rozhraní je potom zprostředkovat uživateli možnost řešit úlohy lineárního programování bez znalostí metod jeho řešení nebo práce se specializovaným softwarem. Nadstavbou k tomuto potom je možnost řešení úlohy optimalizace procesu výrobních linek, bez nutnosti vlastní algoritmizace úlohy a znalosti programovacího jazyka.

K pochopení realizace je třeba mít základní povědomí o tom, co to lineární programování je, což bude vysvětleno v první části této práce. Další část se bude věnovat programu LP Solve jako takovému. Budou v ní popsány některé jeho funkce, i způsob jak jich uživatel může využívat. Třetí část se dostane k vlastní realizaci projektu popisem stavby a funkcí webového rozhraní. Poslední část se bude věnovat řešení obou typů úloh, nejprve definováním vstupů a výstupů a potom vlastním algoritmům řešení úloh. Práce bude ukončena zhodnocením celého procesu tvorby webového rozhranní i této zprávy.

1 Lineární programování

Lineární programování (zkráceně LP) je druh matematického programování spadajícího do oboru Operační výzkum. Matematické programování je soubor metod řešících úlohy s velkým počtem parametrů a kritérií, pro které ze všech možných řešení hledá to nejmenší nebo největší, což je dáno typem úlohy (např.: minimalizace výrobních nákladů nebo maximalizace zisku). Pokud veškeré vztahy mezi proměnnými jsou dány lineárními funkcemi, mluvíme o lineárním programování. Typickou úlohu lineárního programování zobrazuje příklad 1.

Příklad 1:

Čokoládovna vyrábí pět druhů výrobků. Na každý výrobek spotřebovává tři suroviny: tuk, kakao, cukr. Každý den může využít maximálně 1500 kg tuku, 300 kg kakaa a 450 kg cukru. Spotřeba surovin na 1 kg výrobku včetně odbytových cen je uvedena v tabulce 1. Stanovte výrobní plán tak, aby hodnota výroby byla maximální.

tabulka 1 Spotřeba surovin a odbytové ceny pro příklad 1.

	V_1	V_2	V_3	V_4	V_5
Tuk		0,40	0,30	0,60	0,60
Kakao	0,05	0,20	0,10	0,10	
Cukr	0,10	0,20	0,20	0,10	0,20
Odbytová cena (Kč/kg)	20	120	100	140	40

Metody lineárního programování hledají maximum nebo minimum lineární funkce více proměnných neboli účelové funkce (1), omezené podmínkami vyjádřenými rovnicemi nebo nerovnicemi – soustavou omezení (2). Každý takový výsledný vektor proměnných, který vyhovuje omezujícím podmínkám, se nazývá přípustné řešení. Takové přípustné řešení, ve kterém funkce nabývá svého extrému, označujeme za optimální řešení.[1]

Níže uvedené definice (1) a (2) zobrazují obecný tvar úlohy lineárního programování, kde:

x_j je označení j-té proměnné modelu,
c_j je cenový koeficient j-té proměnné – konstanta,
b_i je pravá strana i-tého omezení – konstanta,
a_{ij} je strukturní koeficient – konstanta. [2]

$$\begin{Bmatrix} min \\ max \end{Bmatrix} \sum_{j=0}^{n-1} c_j x_j \quad n \in \mathrm{N}, \tag{1}$$

$$\sum_{j=0}^{n-1} a_{ij}\, x_j \begin{Bmatrix} \geq \\ = \\ \leq \end{Bmatrix} b_i\,, \quad i = 0, \dots, m-1 \quad m, n \in \mathrm{N}, \tag{2}$$
$$x_j\{\geq 0\}, \quad j = 0, \dots, n-1 \quad n \in \mathrm{N}.$$

Úloha LP se dá také zapsat v maticovém tvaru zobrazeném rovnicemi (3) a (4):

$$\begin{Bmatrix} min \\ max \end{Bmatrix} c^T x, \tag{3}$$

s podmínkami:

$$Ax \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b, \qquad x \geq 0. \tag{4}$$

A – matice složená ze strukturních koeficientů a_{ji} o rozměrech m×n**,** zobrazena níže (5).

$$A = \begin{pmatrix} a_{0,0} & \cdots & a_{0,n} \\ \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,n-1} \end{pmatrix}, \tag{5}$$

c$^{\mathbf{T}}$ – řádkový vektor cenových koeficientů c_j, zobrazen níže (6):

$$c^T = (c_0, c_2, \dots, c_{n-1}), \tag{6}$$

b – sloupcový vektor pravých stran b_i, zobrazen na (7):

$$b = \begin{pmatrix} b_0 \\ \vdots \\ b_{m-1} \end{pmatrix}, \tag{7}$$

x – sloupcový vektor neznámých x_j, zobrazen na (8):

$$x = \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix}. \tag{8}$$

Nyní si pro představu uvedeme zápis příkladu 1 pomocí modelu lineárního programování. Hledáme maximální zisk, daný součtem odbytových cen všech vyrobených produktů. Tomu odpovídá účelová funkce (6).

$$\max z = 20x_0 + 120x_1 + 100x_2 + 140x_3 + 40x_4. \tag{6}$$

Protože nemůžeme využít více surovin, než je na daný den poskytnuto, omezení úlohy bude vyjadřovat, že součet surovin spotřebovaných na výrobu každého produktu musí být menší nebo stejný jako poskytované suroviny. Tomu odpovídají nerovnice (7):

$$\begin{aligned} 0{,}4x_1 + 0{,}30x_2 + 0{,}60x_3 + 0{,}60x_4 &\leq 1500, \\ 0{,}05x_0 + 0{,}20x_1 + 0{,}10x_2 + 0{,}10x_3 &\leq 300, \\ 0{,}10x_0 + 0{,}20x_1 + 0{,}20x_2 + 0{,}10x_3 + 0{,}20x_4 &\leq 450. \end{aligned} \tag{7}$$

1.1 Způsoby řešení úloh lineárního programování

Pro řešení úloh lineárního programování se využívá několik metod, z nich si popíšeme jen simplexovou metodu a metodu větví a mezí, protože právě ty LP Solve využívá.

1.1.1 Simplexová metoda

Nejužívanější metodou pro řešení úloh lineárního programování a zároveň metodou, kterou využívá LP Solve k řešení LP problémů, je tzv. simplexová metoda. Tato metoda vychází z primárně přípustné simplexové tabulky[1] a postupnou iterací hledá další přípustné řešení úlohy takové, které zlepší nebo alespoň nezhorší hodnotu účelové funkce, a to tak dlouho, dokud nenajde optimální řešení.

1.1.2 Metoda větví a mezí

V případě počítání úlohy, kde by bylo požadavkem celočíselné řešení, samotný simplexový algoritmus by nám nestačil. Celočíselný výsledek by se musel následně dopočítat jedním z algoritmů pro celočíselné programování např. Gomoryho algoritmem nebo algoritmem větví a mezí. Po nás je důležitý hlavně algoritmus větví a mezí, neboli Branch and Bound (zkráceně BB), protože právě ten využívá software LP Solve k řešení celočíselných úloh.

Algoritmus větví a mezí se využívá pro úlohy, kde je žádoucí úplné nebo jen částečné celočíselné řešení. Jeho princip spočívá v tom, že pro každý prvek z množiny výsledků, u kterého je požadována celočíselnost, vytvoří další dvě úlohy, které budou vycházet z předchozí úlohy. Následně ke každé z úloh přidá podmínku vyjadřující požadavek celočíselnosti a obě úlohy vyřeší. Pokud takto nalezne celočíselný výsledek hledaného prvku, pokračuje s ostatními neceločíselnými výsledky, jinak vybere jednu z předešlých nerozvětvených úloh a proces opakuje.

2 LP Solve

LP Solve je svobodný řešič úloh smíšeného lineárního programování, vystupující pod licencí LGPL (GNU Lesser General Public License). Oficiální web GNU jí definuje takto:

„*Program, který neobsahuje žádnou část odvozenou od kterékoli části knihovny, ale je navržen tak, aby s knihovnou pracoval zkompilováním nebo slinkováním s ní, se nazývá jako "dílo, které používá knihovnu". Takové dílo, izolované, není odvozeným dílem knihovny, a proto spadá mimo působnost této licence.*“[3].

2.1.1 Historie

Za vývojem LP Solve původně stojí Michel Berkelaar z Eindhoven University of Technology. Následě byl LP Solve Jeroenem Dirksem převeden z verze 1.5 do plné verze 2.0, kde byla implementovaná MPS[1] čtečka, opraveny známé chyby a bylo provedeno optimalizování stávajícího kódu. Navíc byl přeložen algoritmus 2.0 verze z C do Javy. Do vydání verze 3.0, kdy byl zaštítěn licencí LGPL, bylo možné jej využívat pouze k nekomerčním účelům. Do té doby na projektu LP Solve pracovalo mnoho dalších vývojářů, nicméně jejich identita není známa. Aktuálně je program ve verzi 5.5, která je od předešlých verzí výkonnější co se týče rychlosti řešení, stabilnější a schopna řešit velké modely.[6]

2.1.2 Využití

LP Solve řeší úlohy lineárního programování pomocí simplexové metody (resp. metody větví a mezí při požadavcích celočíselnosti). Dokáže řešit modely lineární, smíšené i celkové celočíselné, částečně lineární (proměnné mohou nabývat hodnot omezených jejich maximem a minimem nebo nuly, což platí i pro případ, že v soustavě omezení je řečeno jinak) nebo SOS[2]. Pro výpočet modelů celočíselných, částečně lineárních nebo SOS se zde využívá metody větví a mezí. Dokáže zpracovat modely formátované jako

[1]MPS - Master Production Schedule formát – je to starý, sloupcově orientovaný formát, ve kterém každá položka musí mít svoje jméno

[2] SOS - Special Ordered Sets představují množinu proměnných, ve které pouze předem definovaný počet proměnných smí nabývat jiných hodnot než 0. Toto je navíc v LP Solve implementováno tak, že tyto nenulové proměnné musí být vzájemně sousední a seřazené podle jejich definovaných důležitostí

modely lineárního programování (lp[3]) i modely výrobních plánů (MPS). Ačkoliv LP Solve neomezuje velikost zadaného modelu, může se stát, že některé, především větší modely, mohou řešení hledat dlouho nebo ho vůbec nenajít.

2.1.3 Způsob předávání dat

LP Solve je knihovna, která může být volána téměř z jakéhokoliv programovacího jazyku. Je několik způsobů jak metody LP Solve využít:

- přes API,
- pomocí vstupní soubory,
- skrze IDE.

Vyžití LP Solve přes API

API je soubor metod, volaný daným programovacím jazykem, sloužící ke stavbě a prací s modelem v paměti, jeho řešení a vrácení výsledků. API obsahuje mnoho metod, které je možné při práci s modelem využít, tyto budou rozebrány v další části práce.

Využívání LP Solve pomocí vstupních souborů

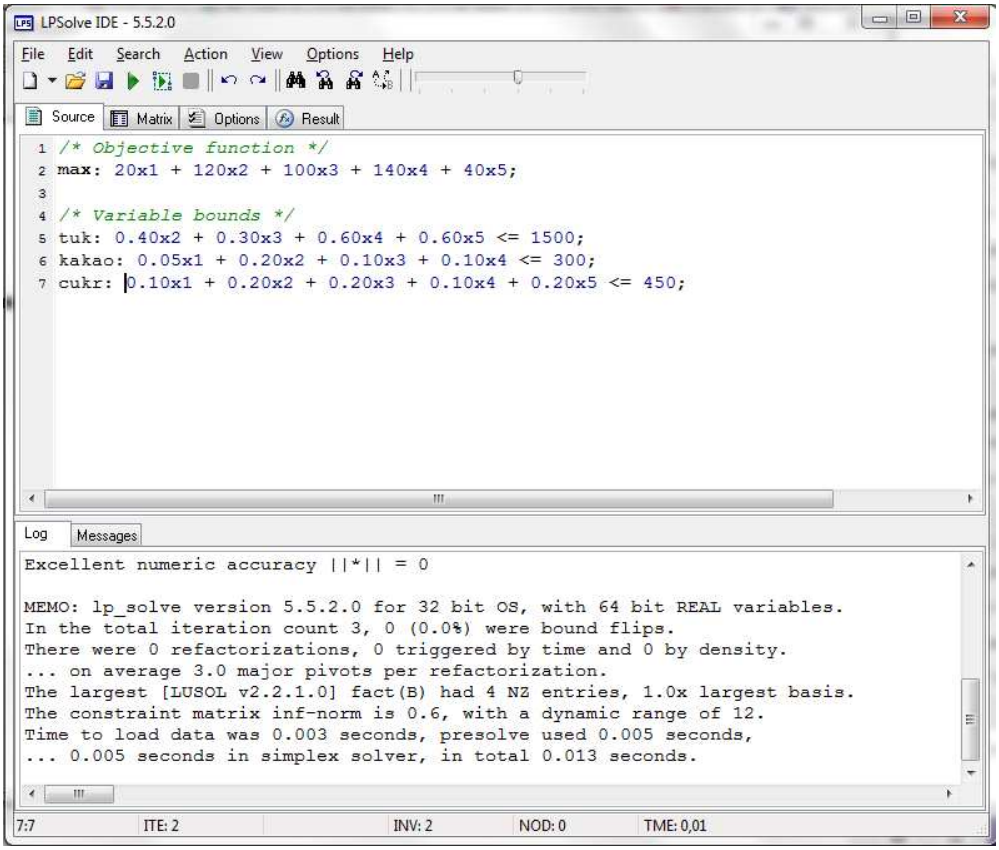
V základu jsou podporovány formáty MPS a lp. LP Solve dokáže využít uživatelem definované metody pro vkládání modelů (skrze XLI – viz. [4]). Dále je možno model ze souboru předat LP Solve pomocí příkazové řádky příkazem lp_solve s parametry definovanými zde [5]. Model tak bude vyřešen bez jakékoliv znalosti API nebo programovacích jazyků.

Využívání LP Solve skrze IDE

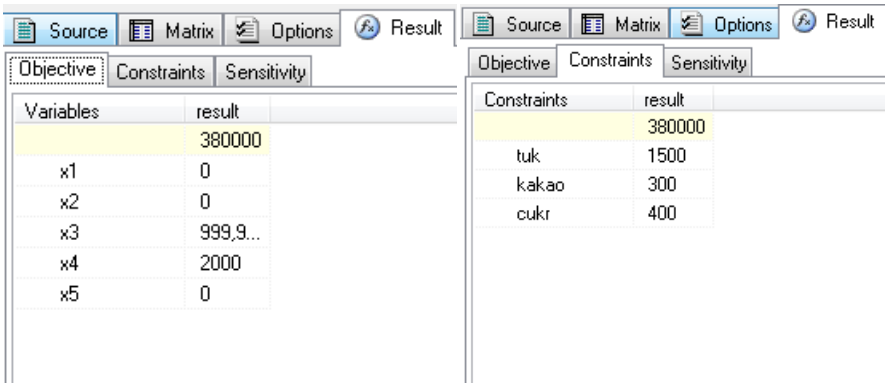
LP Solve IDE je software zprostředkovávající metody API uživateli skrze Windows aplikaci. Tato aplikace maximálně usnadňuje práci s úlohami lineárního programování tak, že do textového pole uživatel zadá požadovaný model, nechá aplikaci jej vyřešit a následně se podívá na vypsané výsledky. IDE samozřejmě poskytuje i možnost nastavit různé parametry výpočtů, jednoduchou práci s textem, načítání a ukládání souborů nebo zobrazit matici zadaného modelu. Na obr. 1 je předvedeno, jakým způsobem se s LP

[3] LP – lp formát je základní formát LP Solve, který obsahuje řádek s objektovou funkcí, soustavu omezení a definici typů proměnných

Solve IDE pracuje na konkrétní úloze továrny na čokoládu z příkladu 1. Obr. 2 potom zobrazuje výpis výsledků poskytnutý IDE.



obr. 1 Ukázka LP Solve IDE – Účelová funkce a soustava omezení příkladu 1.



obr. 2: Ukázka LP Solve IDE – Výsledky příkladu 1.

3 Webové rozhranní

Webové rozhraní je realizováno pomocí značkovacího jazyka HTML, kaskádových stylů CSS a skriptovacího programovacího jazyka PHP. HTML a CSS zajišťuje možnost přehledného vkládání dat, skripty PHP data zpracovávají a předávají LP Solve API.

3.1 Zajištění spolupráce PHP a LP Solve API

Aby spolupráce mezi PHP a knihovnou LP Solve byla možná, je nejprve třeba webovému serveru poskytnout knihovnu LP Solve.

3.1.1 Práce s knihovnami

Na tuto knihovnu odkazuje řídící soubor `php_phplpsolve55.dll` pro Windows a `phplpsolve55.so` pro Unix. Tento soubor je třeba uložit na lokální úložiště, nejlépe do složky s ostatními knihovnami, které PHP na daném serveru využívá. A následně upravit konfiguraci PHP v souboru `php.ini`. Tento soubor se pod Windows obvykle nachází v `\Program Files\php\`, v Unixu zas pod `/etc/php5/cli` nebo `/etc/php5/apache2`. Po otevření souboru v textovém editoru je třeba se podívat, kam odkazuje položka `extension_dir=` (viz. obr 3), do tohoto umístění se LP Solve knihovna uloží. Následně se do souboru přidá položka `extension = php_phplpsolve55.dll` pod Windows (viz. obr. 4), nebo `extension = phplpsolve55.so` pro Unix. Aby změny byly aktivní, je nakonec nutno restartovat webovou službu.

```
; Directory in which the loadable extensions (modules) reside.
extension_dir = "c:/wamp/bin/php/php5.2.9-2/ext/"
```

obr. 3: Umístění knihovny LP Solve.

```
;extension=php_sybase_ct.dll
;extension=php_tidy.dll
;extension=php_xmlrpc.dll
;extension=php_xsl.dll
;extension=php_zip.dll
extension=php_phplpsolve55.dll
;;;;;;;;;;;;;;;;;;;
; Module Settings ;
;;;;;;;;;;;;;;;;;;;

[Date]
; Defines the default timezone used by the date functions
;date.timezone =
```

obr. 4: Povolení knihovny pro PHP.

Ovladač `php_phplpsolve.dll` volá sdílenou LP Solve knihovnu `lpsolve55.dll` pod Windows (resp. `liblpsolve55.so` pod Unix). Toto řešení má tu výhodu, že ovladač LP Solve nemusí být rekompilován pokaždé, kdy se změní verze LP Solve. Tato knihovna obsahující API LP Solve se musí nalézat na místě, kam odkazuje systémová proměnná PATH.

3.1.2 Ověření funkčnosti instalace knihovny

Schopnost PHP využívat API LP Solve můžeme ověřit např. příkazem `lpsolve()`, ten by měl vrátit výstup (čísla verzí zobrazená ve výstupu se budou lišit podle užívaných verzí):

```
LP Solve PHP Interface version 5.5.0.6
using LP Solve version 5.5.2.0

Usage: ret = LP Solve("functionname", arg1, arg2, ...)
```

Pokud takový výstup neobdržíme, je to nejčastěji způsobeno tím, že PHP dokázalo najít knihovnu ovladače, ale už ne knihovnu s LP Solve API.

3.2 Funkce LP Solve API

Základní syntaxe LP Solve funkcí pro PHP je zobrazena v posledním řádku výstupu 1, pro podrobnější popis si tento řádek uvedeme znovu na výstupu 2:

```
ret = LP Solve('functionname', arg1, arg2, ...).
```

Návratová hodnota se mění v závislosti na typu funkce, může to být skalár, vektor, případně pole. Jméno funkce musí být uvedeno vždy v uvozovkách a je case sensitive. Typ jednotlivých argumentů i jejich počet je závislý na použité funkci. Některé funkce mohou mít počet argumentů proměnlivý a jejich chování se pak odvíjí od typu užitých argumentů. Ve většině metod LP Solve API vystupuje jako první argument ukazatel na lp model, který má tvar skaláru. Vlastní model je pak spravován ovladačem LP Solve.

Téměř všechny funkce LP Solve API jsou k nalezení v lp_solve API referenci [6]. Většina z uvedených funkcí je pro PHP stejná jako v referenčním seznamu například `make_lp`, která vytváří model o daných rozměrech v paměti. Vrací inkrementované číslo

začínající na nule, které slouží jako ukazatel na vzniklý model. Některé se liší syntaxí jako funkce `get_column,` která užívá proměnnou pro uložení výsledku `column` standardně jako parametr, ale v případě funkce pro PHP se předává jako návratová hodnota. Některé funkce nejsou realizované pro PHP vůbec jako například `str_add_constraint,` ta slouží k přidání omezujícího řádku do soustavy omezení a data přitom čte z řetězce. Použité LP Solve API funkce jsou rozebrány v příloze.

3.3 Typy řešených úloh

Úkolem webového rozhraní je jednak zprostředkovat funkce LP Solve API pro řešení úloh lineárního programování, jednak poskytnout uživateli možnost jednoduše vypočítat plán výrobních linek se zadanými parametry tak, aby jejich čas byl využit co možná nejlépe.

3.3.1 Obecná úloha LP

Hledáme minimum nebo maximum lineární funkce n proměnných ohodnocených cenou c_i, kde $i \in \hat{n}$, čemuž odpovídá účelová funkce (8):

$$\min z = \sum_{i=0}^{n-1} c_i x_i,$$

maticový zápisem pak:

$$\min z = (c_0 \quad \cdots \quad c_{n-1}) \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix}. \tag{8}$$

Účelová funkce je navíc omezena soustavou omezení danou s m řádky, kde jeden řádek $j \in \hat{m}$ má tvar (9):

$$\sum_{i=0}^{n-1} a_{ij} x_i \blacksquare b_j,$$

nebo maticově:

$$\begin{pmatrix} a_{0,0} & \cdots & a_{0,m} \\ \vdots & \ddots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,m-1} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} \blacksquare \begin{pmatrix} b_0 \\ \vdots \\ b_{m-1} \end{pmatrix}. \tag{9}$$

Pro $(a_{ij})_{\substack{i \in \hat{m} \\ j \in \hat{p}}}$ – strukturní koeficient proměnné x_i, b_j– pravá strana j-tého omezení,

$\blacksquare =$ '$\geq$', '$\leq$'$nebo$ '='.

3.3.2 Optimalizace procesu výrobních linek

Určitý počet linek vyrábí množinu výrobků, kterou musí uspokojit poptávku odběratelů. Výroba se plánuje pro určité období skládající se z n výrobních cyklů (směn, dnů, týdnů, ...). Tyto cykly se mohou lišit v požadavcích na výrobu produktů. Každá linka má omezený pracovní čas (tzv. volný čas linky), který říká, kolik časových jednotek během jednoho cyklu linka vyrábí. Stejně jako požadavky na produkci se tento čas může pro každý cyklus a každou linku lišit. Linky se navíc liší náklady na výrobu každého kusu produktu a také kapacitou výroby, říkající jaké množství, kterého produktu je linka schopna za časovou jednotku vyrobit.

Pokud linka nevyužije v daném cyklu všechen svůj volný čas, bude možno využít zbylého (tzv. rezervního) volného času na výrobu zásob do budoucna nebo na doplnění produkce předešlých cyklů.

Cílem úlohy je najít takový výrobní plán pro dané období, při kterém budou náklady na výrobu minimální. Výrobní plán bude rozdělen na n částí, z nichž každá bude obsahovat informace o tom kolik, která linka má vynaložit času na výrobu kterého produktu. Dále bude zobrazovat, kolik volného času linek nebylo využito a jak by tyto rezervy měly být využity, aby všechny cykly vyrobily požadovaný objem produkce.

Pro úlohu optimalizace procesu výrobních linek platí toto (10):

$$\boldsymbol{A} = a_{ij} = \begin{pmatrix} a_{0,0} & \cdots & a_{0,p} \\ \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,p-1} \end{pmatrix},$$

a_{ij} − množství výrobků j, které linka i vyrobí za jednotku času,

$$\boldsymbol{C} = c_{ij} = \begin{pmatrix} c_{0,0} & \cdots & c_{0,p} \\ \vdots & \ddots & \vdots \\ c_{m-1,0} & \cdots & c_{m-1,p-1} \end{pmatrix},$$

c_{ij} − náklady na časovou jednotku výroby výrobku j linkou i,

$$\boldsymbol{B} = b_{kj} = \begin{pmatrix} b_{0,0} & \cdots & b_{0,p} \\ \vdots & \ddots & \vdots \\ b_{n-1,0} & \cdots & b_{n-1,p-1} \end{pmatrix}, \tag{10}$$

b_{kj} − požadavky na vyrobené množství produktu j pro k-tý cyklus

$$\boldsymbol{T} = t_{ki} = \begin{pmatrix} t_{0,0} & \cdots & t_{0,m} \\ \vdots & \ddots & \vdots \\ t_{n-1,0} & \cdots & t_{n-1,m-1} \end{pmatrix},$$

t_{ki} − časové omezení i-té linky v cyklu k.

Z důvodu složitosti postupu úlohu zjednodušíme jen na jeden cyklus k. Podrobný popis řešení pro celé období bude popsán v dalších částech práce.

Hledáme minimální náklady, tomu odpovídá účelová funkce (11):

$$\min z = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} c_{ij} x_{ij},$$

maticově:

$$\min z = \begin{pmatrix} c_{0,0} & \cdots & c_{0,p} \\ \vdots & \ddots & \vdots \\ c_{m-1,0} & \cdots & c_{m-1,p-1} \end{pmatrix} \begin{pmatrix} x_{0,0} & \cdots & x_{0,p} \\ \vdots & \ddots & \vdots \\ x_{m-1,0} & \cdots & x_{m-1,p-1} \end{pmatrix}. \tag{11}$$

Soustava omezení bude vyjadřovat nutnost nepřekročit volné časy linek t, vyjádřené rovnicí (12) a potřebu splnit požadavky na výrobu b vyjádřené rovnicí (13).

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{ij} \leq t_i ,$$

maticově pak:

$$\begin{pmatrix} x_{0,0} & \cdots & x_{0,p} \\ \vdots & \ddots & \vdots \\ x_{m-1,0} & \cdots & x_{m-1,p-1} \end{pmatrix} \leq \begin{pmatrix} t_0 \\ \vdots \\ t_{m-1} \end{pmatrix}, \tag{12}$$

$$\sum_{j=0}^{n-1} \sum_{i=0}^{m-1} a_{ij} x_{ij} = b_j,$$

a pomocí matic:

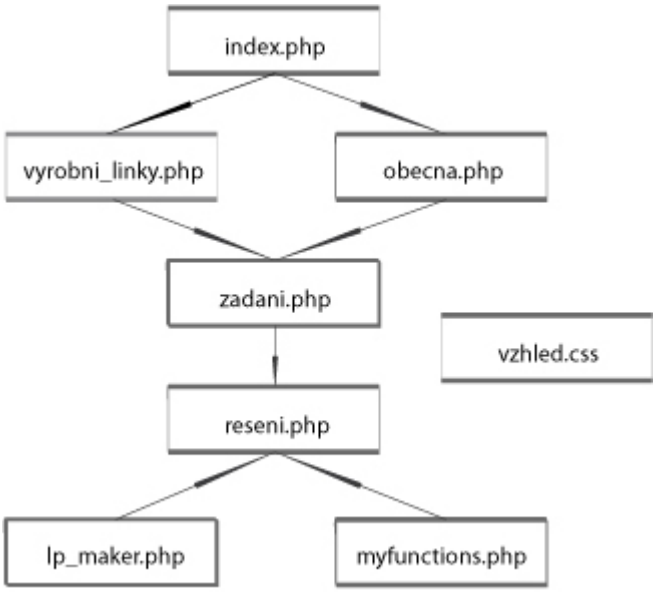
$$\begin{pmatrix} a_{0,0} & \cdots & a_{0,p} \\ \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,p-1} \end{pmatrix} \begin{pmatrix} x_{0,0} & \cdots & x_{0,p} \\ \vdots & \ddots & \vdots \\ x_{m-1,0} & \cdots & x_{m-1,p-1} \end{pmatrix} \geq \begin{pmatrix} b \\ \vdots \\ b_{p-1} \end{pmatrix}. \tag{13}$$

Samozřejmě není možné vyrobit záporný počet produktů, proto $\forall x_{ij} \geq 0$.

3.4 Struktura webového rozhraní

Po otevření aplikace ve webovém prohlížeči se uživateli jako první zobrazí soubor `index.php`. Zde má pak na výběr ze dvou možností: Výrobní linky a Obecná LP úloha. Po vybrání jedné z možností se uživateli zobrazí soubor `vyrobni_linky.php` (resp. `obecna.php` v případě vybrání obecné úlohy). Následně je zadavateli úlohy nabídnuto vyplnění 1.úrovně parametrů úloh. Dle typu vybrané úlohy je pak z těchto parametrů skriptem `zadani.php` vytvořena stránka pro zadání 2.úrovně parametrů úloh. O vytvoření úloh ze zadaných parametrů a následné vypsání výsledků se potom postará

skript `reseni.php`, který při řešení úlohy optimalizace linek dále využívá souborů `lp_maker.php` a `myfunctions.php`. Nad tím vším řídí vzhled aplikace soubor kaskádových stylů `vzhled.css`. Graficky je struktura stránek znázorněna na obr. 5:



obr 5. Struktura webového rozhraní.

3.5 Způsob řešení úloh

Veškerá data se aplikaci předávají skrze formuláře HTML. Potom jsou PHP skripty ukládány do jim odpovídajících datových struktur a pomocí funkcí LP Solve API se s nimi následně pracuje.

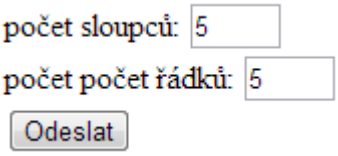
3.5.1 Obecná úloha lineárního programování

Aplikace skládá model obecné úlohy z takovýchto parametrů:

1. úroveň – obecna.php

- n – počet sloupců neboli neznámých,
- m – počet řádků omezujících podmínek.

V této části aplikace se nastavují rozměry úlohy viz. obr 6. Hodnota n říká, kolik proměnných daná úloha bude mít. Hodnota v m nastavuje počet řádků soustavy omezení.



obr. 6 Rozměry úlohy.

2. úroveň – zadani.php

- *min/max* – určuje, jestli je úloha typu maximalizačního nebo minimalizačního,
- $\boldsymbol{c} = (c_0, \ldots, c_{n-1})$ – vektor cenových koeficientů proměnných,
- $\boldsymbol{A} = (a_{ij})_{\substack{i \in \hat{n} \\ j \in \hat{m}}}$ – matice strukturních koeficientů,
- $\boldsymbol{e} = (e_0, \ldots, e_{m-1})$ – vektor koeficientů určujících typ omezení,
- $\boldsymbol{b} = (b_0, \ldots, b_{m-1})$ – vektor pravých stran,
- $\boldsymbol{int} = (int_0, \ldots, int_{n-1})$ – vektor celých čísel,
- $\boldsymbol{neg} = (neg_0, \ldots, neg_{n-1})$ – vektor záporných čísel.

V druhé části uživatel vybere, jestli chce počítat minimalizační nebo maximalizační úlohu a nastaví jednotlivé cenové koeficienty proměnných. Následně vyplní omezující podmínky - vyplní matici strukturních koeficientů, nastaví typy omezení a zadá pravé strany. Pokud je vyžadováno, aby výstupními hodnotami mohla být záporná čísla nebo aby výsledky byly pouze celočíselné, u každé proměnné se v poslední fázi zatrhnou tyto možnosti. Defaultně jsou typy proměnných nastaveny na nezáporná reálná čísla. Ukázka na obr. 7.

obr. 7 Zadání obecné úlohy.

Vlastní řešení

Po odeslání formuláře s 2. úrovní parametrů skript `zadani.php` zkontroluje, jestli se jedná o parametry obecné úlohy a následně začne ukládat data z formuláře do proměnných PHP. Ač se jedná o jazyk PHP, je nutné dbát na to, aby ukládaná data měla správný datový typ, protože data dále budeme posílat LP Solve API, které je psané v jazyce C.

Vlastní algoritmus skládání a řešení potom funguje takto:

1. vytvoří se model v paměti o rozměrech $n \times m$,
2. nastaví se druh úlohy – minimalizace nebo maximalizace,
3. složí se účelová funkce z vektoru $\boldsymbol{c}$,
4. z každého j-tého řádku matice strukturních koeficientů, j-tého sloupce vektoru $\boldsymbol{e}$ a vektoru $\boldsymbol{b}$, kde $j \in \hat{m}$, se vytvoří jeden řádek soustavy omezení,
5. pro každý i-tý prvek $\boldsymbol{neg}$ a $\boldsymbol{int}$, kde $i \in \hat{n}$, se nastaví typ proměnné x_i,
6. model se vyřeší,

7. pokud není nalezeno řešení, na výstupu se zobrazí "Model nemá řešení." a algoritmus se ukončí,
8. pokud je nalezeno řešení výsledek se vypíše v tomto formátu:

   $z^* = "hodnota\_účelové\_funkce"$

   $x^* = (x_0, x_1, ..., x_{n-1})$
9. model je uvolněn z paměti.

3.5.2 Úloha optimalizace výrobních linek

Úloha má tyto vstupní parametry:

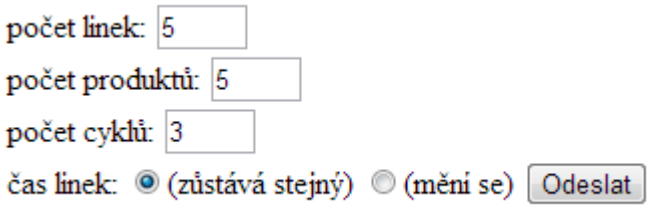
1. **úroveň – vyrobni_linky.php**

   m – počet linek,

   p – počet produktů,

   n – počet cyklů predikce.

Podobně jako u obecné úlohy LP, nejprve se určí základní vlastnosti úlohy, kterých se bude využívat při stavbě modelu. Předvedeno na obr. 8.



obr. 8 Základní vlastnosti

2. **úroveň – zadani.php**

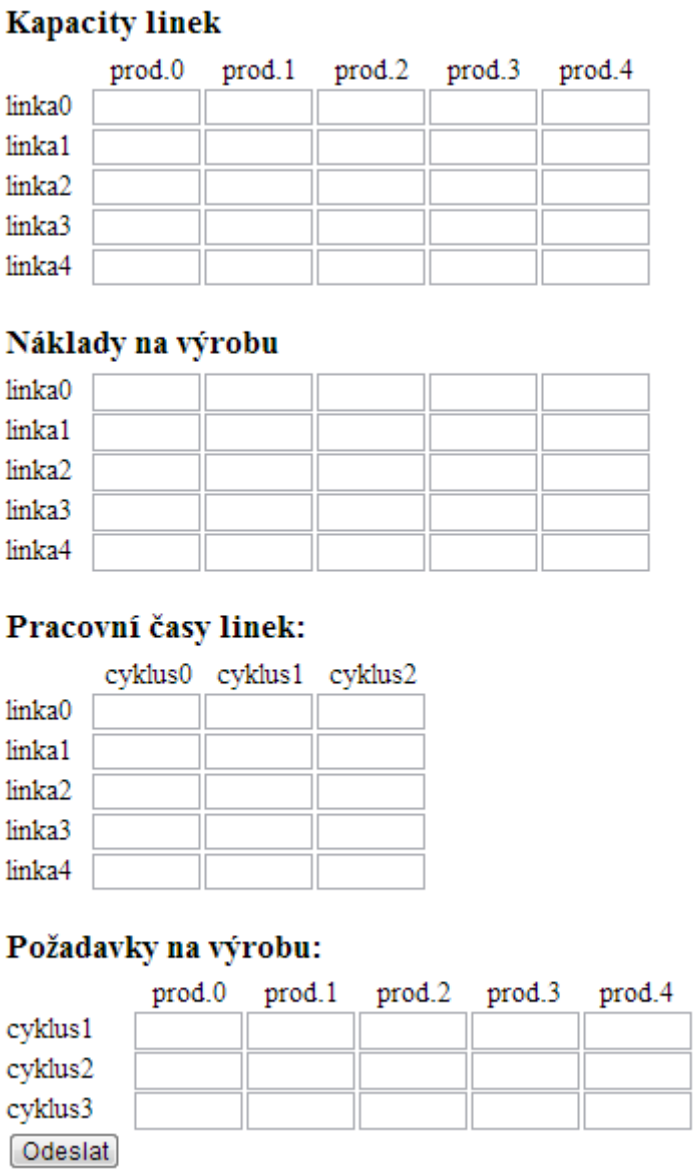
   $\boldsymbol{A} = (a_{ij})_{\substack{i\in\hat{m}\\ j\in\hat{p}}}$ – matice kapacit linek,

   $\boldsymbol{C} = (c_{ij})_{\substack{i\in\hat{m}\\ j\in\hat{p}}}$ – matice nákladů na výrobu produktu,

   $\boldsymbol{T} = (t_{ki})_{\substack{k\in\hat{n}\\ i\in\hat{m}}}$ – matice volných časů linek,

   $\boldsymbol{B} = (b_{kj})_{\substack{k\in\hat{n}\\ j\in\hat{p}}}$ – matice požadavků na výrobu.

Jak je vidět na obr. 9, ve druhé části zadávání parametrů se vyplní matice kapacit pro každou linku a produkt. Stejně tak se určí náklady výroby produktu na lince a nastaví se, kolik volného času může linka pro daný cyklus využít. Nakonec se zadají požadavky na odběr produktů pro každý cyklus.



obr. 9 Parametry úlohy.

Vlastní řešení

Řešení úlohy se skládá z těchto částí:

1. Provede se kontrola splnitelnosti požadavků, při daných omezeních úlohy tak, že se spočte, jestli je úloha řešitelná jako celé období, tedy při součtu požadavků a omezení času linek všech cyklů.
2. Vyřeší se jednotlivé cykly výroby při současném splnění omezujících podmínek celého predikčního období a zapíší se případné neřešitelné cykly.
3. Vyřeší se úlohy nevyřešených cyklů s tím, že je k jejich volným časům linek přidáván rezervní čas předešlých (resp. následujících cyklů).
4. Dopočítají se výrobní plány linek vyrábějících ve využitých rezervních časech.
5. Vypíše se výrobní plán pro celé období predikce.

Kontrola řešitelnosti úlohy

Úloha je řešitelná v případě, že existuje optimální řešení úlohy pro celé období predikce. Musí tedy být splněn součet požadavků všech cyklů ten definuje rovnice (13) jako $\boldsymbol{b'}$, při nepřekročení celkového času linek definovaného rovnicí (14) nebo (15) jako $\boldsymbol{t'}$.

Nejprve je tedy nutno spočítat požadavky pro celé predikční období:

$$\boldsymbol{b'} = \sum_{k=0}^{n-1} \boldsymbol{b_k}. \tag{13}$$

Dále celkový čas linek na období (pro případ kdy se časy linek v každém cyklu liší):

$$\boldsymbol{t'} = \sum_{k=0}^{n-1} \boldsymbol{t_k}, \tag{14}$$

případně (pro případ kdy se časy linek v průběhu času nemění):

$$\boldsymbol{t'} = \boldsymbol{t} * n. \tag{15}$$

Pro parametry získané při zadávání úlohy hledáme takové časové rozložení, při kterém minimalizujeme výrobní náklady c produktu j, na lince i a to pro celé období výroby skládající se z n cyklů. Tomu odpovídá účelová funkce (16):

$$\min z = \sum_{k=0}^{n-1} \sum_{i=0}^{m-1} \sum_{j=0}^{p-1} c_{ij} x_{kij}. \tag{16}$$

Dále je třeba vytvořit soustavu omezení. Ta je tvořena omezeními času linek, kdy celkový využitý čas nesmí překračovat celkový volný čas a požadavky na minimální cel-

kové množství vyrobených produktů. Část omezení linek se bude skládat z m omezujících řádků a bude pro každý řádek $i \in \hat{m}$ mít tvar zobrazení rovnicí (17):

$$\sum_{k=0}^{n-1}\sum_{j=0}^{p-1} x_{kij} \leq t'_i. \tag{17}$$

Část požadavků na produkty bude mít p řádků a pro každý řádek $j \in \hat{p}$ bude její tvar (18):

$$\sum_{k=0}^{n-1}\sum_{i=0}^{m-1} a_{kij}x_{kij} \geq b'_j. \tag{18}$$

Pokud po vyřešení úlohy s těmito s omezeními bude nalezeno optimální řešení, je možno přikročit k řešení jednotlivých cyklů. V opačném případě se na výstupu vypíše, že úloha nemá řešení a program bude ukončen.

Řešení cyklů se základními časy

Každý cyklus je třeba řešit ve spojení s úlohou pro celé období, aby se zajistilo, že zůstane zachována řešitelnost celého období. V opačném případě by nám při řešení cyklů bez přípustných řešení nestačily rezervy vypůjčené z předešlých (resp. následujících) cyklů, na výrobu dostačujícího množství produktu, aby byly splněny požadavky odběratelů.

Účelovou funkci a podmínky pro řešitelnost celkového období budou všechny cykly mít společné. Po každý jednotlivý cyklus tedy vytvoříme model celého období a přidáme k němu omezující podmínky řešeného cyklu. Soustava omezení cyklu $k \in \hat{n}$, se bude skládat z $m + p$ řádků.

První část týkající se omezení linek bude mít pro řádek $i \in \hat{m}$ cyklu k takovýto tvar (19):

$$\sum_{j=0}^{p-1} x_{kij} \leq t_{ki}. \tag{19}$$

Požadavky na produkty skládající druhou část soustavy omezení cyklu k budou mít pro řádek $j \in \hat{p}$ tvar (20):

$$\sum_{i=0}^{m-1} a_{ij} x_{kij} \geq b_{kj}. \tag{20}$$

Pro každý cyklus takový model vyřešíme. V této fázi můžeme mít dva typy výsledků:

a. nalezeno optimální řešení
b. nenalezeno přípustné řešení

Pokud bylo nalezeno optimální řešení cyklu k, potom z celkového vektoru výsledků $\boldsymbol{x}^*$, kde $\boldsymbol{x}^* = (x_0, \ldots, x_{(n*m*p)-1})$, získáme matici výsledků k-tého cyklu zobrazenou zápisem (21):

$$\boldsymbol{X}_{\boldsymbol{k}}^* = x_{kij}^* = \begin{pmatrix} x_{0,0} & \cdots & x_{0,p} \\ \vdots & \ddots & \vdots \\ x_{m-1,0} & \cdots & x_{m-1,p-1} \end{pmatrix}, \tag{21}$$

a spočítáme časové rezervy linek pro cyklus k. Rezervy linky i získáme tak, že sečteme produkční časy linky pro každý produkt a následně je odečteme od volného času linky dle vzorce (22):

$$r_{ki} = t_{ki} - \sum_{j=0}^{p-1} x_{kij}^*. \tag{22}$$

V případě, že pro daný cyklus nebylo nalezeno řešení, uložíme číslo cyklu do zásobníku nevyřešených cyklů.

Řešení cyklů s přidáváním rezerv

Vezmeme úlohu h ze zásobníku a nastavíme číslo úlohy právě poskytující rezervy (tzv. půjčující úlohy) jako $k = h$ a uložíme nynější pracovní časy linek úlohy h do vektoru $\boldsymbol{tn_h}$. Dále je třeba zavést řídící proměnnou u, nastavující jestli počítáme s předešlými ($u = 0$) nebo následujícími ($u = 1$) cykly. Proměnnou u nastavíme na 0.

1. Pokud u je vypnuto a cyklus k neodpovídá prvnímu pracovnímu cyklu, jak říká vzorec (23):

$$u = 0 \cap k > 0, \tag{23}$$

potom číslo půjčující úlohy je $k = k - 1$. Jinak nastavíme řídící proměnnou u na počítání s následujícími cykly a číslo úlohy k na h, dle vzorce (24):

$$u = 1, \qquad k = h. \tag{24}$$

Pokud je řídící proměnná u nastavena na 1. Číslo cyklu poskytujícího rezervy získáme dle vzorce (25):

$$k = k + 1. \tag{25}$$

2. Uložíme předešlý volný čas linek $\boldsymbol{tn_h}' = \boldsymbol{tn_h}$ a přičteme k volným časům nek $\boldsymbol{tn_h}$ časové rezervy úlohy k jak zobrazuje (26):

$$\boldsymbol{tn_h} = \boldsymbol{tn_h} + \boldsymbol{r_k}. \tag{26}$$

3. Upravíme pravou stranu cyklu h tak, aby bylo využito rezerv viz. (27):

$$\boldsymbol{rhs_h} = \begin{pmatrix} \boldsymbol{t}' \\ \boldsymbol{b}' \\ \boldsymbol{tn_h} \\ \boldsymbol{b_h} \end{pmatrix}. \tag{27}$$

4. Vyřešíme. Pokud nebylo nalezeno řešení, vracíme se k bodu 1.
5. Přepočítáme rezervy. Můžou nastat dvě situace:
   a. Celkový využitý čas linky i se vejde do původního volného času – platí (28):

$$\sum_{j=0}^{p-1} x_{hij}^{*} \leq t_{hi}. \tag{28}$$

   Časové rezervy linky i pro cyklus h spočteme stejně jako u řešitelné úlohy a rezervy půjčujících cyklů zůstanou netknuté, viz (29):

$$r_{hi} = t_{hi} - \sum_{j=0}^{p-1} x_{hij}^{*}. \tag{29}$$

   b. Linka využila část nebo celé poskytnuté rezervy, zobrazeno na (29):

$$t_{hi} \leq \sum_{j=0}^{p-1} x_{hij}^{*} \leq tn_{hi}. \tag{29}$$

   Rezervy linky $i \in \hat{m}$ pro cyklus h jsou nulové (čas linek byl využit celý). Všechny půjčující cykly, kromě posledního, mají rezervy linky i rovněž celé využity. Linka i posledního půjčujícího cyklu, přijde o rezervní čas rovný

rozdílu mezi aktuální využitou a předešlou časovou kapacitou linky $tn_{hi}{}'$ viz. (31):

$$r_{ki} = r_{ki} - \left(\sum_{j=0}^{p-1} x^*_{hij} - \; tn'_{hi} \right) \tag{31}$$

6. Nalezení výrobního plánu cyklů poskytujících svoje rezervy. Abychom zjistili, na výrobu kterých výrobků svůj čas daná linka využije, je třeba složit další model, který bude obohacený o půjčující linky. Pro každý samostatně neřešitelný cyklus h vytvoříme novou úlohu.

Nový model bude mít stejnou účelovou funkci, jako předešlé viz. (32):

$$\min z = \sum_{k=0}^{n-1} \sum_{i=0}^{m-1} \sum_{j=0}^{p-1} c_{ij} x_{kij}. \tag{32}$$

První částí soustavy omezení bude časové omezení linek úlohy h, kde řádek i má tvar (33):

$$\sum_{j=0}^{p-1} x_{hij} \leq \; t_{hi} \tag{33}$$

Druhá část představuje požadavky na výrobu pro cyklus h, ta bude oproti omezením linek obohacena o rezervní časy linek $i_r \in \widehat{k_r}$, pro každý rezervy poskytující cyklus $k_r \in \widehat{K}$, kde $\widehat{k_r}$ je množina linek, jejichž čas je třeba využít na splnění požadavků a $\widehat{K}$ je množina cyklů, ve kterých se bude vyrábět rezervní produkt. Pro řádek $j \in \hat{p}$ platí rovnice (33):

$$\sum_{i=0}^{m-1} a_{ij} x_{hij} + \sum_{\mathrm{k_r}=0}^{\mathrm{K}-1} \sum_{\mathrm{i_r}=0}^{\mathrm{k_r}-1} a_{i_r j} x_{k_r i_r j} = \; b_{hj} \tag{33}$$

V poslední části budou vystupovat časová omezení linek poskytujících rezervní čas. Počet jejích řádků se bude odvíjet od počtu všech půjčujících linek ($i_r \in \widehat{k_r,}$) každého cyklu ($k_r \in \widehat{K}$), ve kterém se bude vyrábět rezervní produkt. Soustava bude mít následující podobu (34):

$$\sum_{\mathrm{k_r}=0}^{\mathrm{K}-1} \sum_{\mathrm{i_r}=0}^{\mathrm{k_r}-1} \sum_{\mathrm{j}=0}^{\mathrm{p}-1} x_{k_r i_r j} \leq \; r_{k_r i_r} \tag{34}$$

Z výsledného vektoru $\boldsymbol{x}^* = (x_0, \ldots, x_{(n*m*p)-1})$ potom spočítáme výrobní plán cyklu k_r pro rezervní čas $r_{k_r i_r}$.

4 Implementace

O vlastní práci s daty stará skript `reseni.php`, ten ze získaných parametrů (popsáno v kapitole 3.6) nejprve zjistí, jestli hledáme řešení obecné úlohy LP nebo chceme optimalizovat proces výroby na výrobních linkách.

4.1 Obecná LP úloha

Předně je třeba načíst parametry úlohy do proměnných a zajistit aby měly typ, který API LP Solve očekává. Následně se vytvoří model s rozměry $n * m$ a nastaví se, které informace má program vypisovat. Zkontroluje se, jestli úloha je minimalizační nebo maximalizační a zavolá se API, aby vytvořilo účelovou funkci vytvořeného modelu. Cyklus přes všechny řádky soustavy omezení potom přidává k modelu omezující podmínky. Poslední věcí před řešením modelu je zkontrolovat jestli proměnná i má být pouze nezáporná nebo/a celočíselná, případně jí na takovou nastavit. Úloha se vyřeší, a pokud bylo nalezeno optimální řešení, získá se z ní výsledná hodnota účelové funkce a vektor výsledných proměnných, tyto se potom vypíší. V případě, že řešení neuspělo, vypíše se na výstupu "Model nemá řešení". Nakonec se model uvolní z paměti.

4.2 Úloha optimalizace výrobních linek

Nejprve se načtou parametry úlohy a ve správném datovém typu se uloží do proměnných a polí. Zavolá se funkce `lp_maker`[4] s těmito parametry: účelová funkce pro celé predikční období, matice strukturních koeficientů, matice požadavků výroby, vektor (resp. matice pokud se omezení pro různé cykly liší) časových omezení linek, vektor rovností, NULL, NULL, NULL, 1, 1.

Funkce `lp_maker` nejprve vytvoří model pro celé predikční období o rozměrech $p * m * n$ a nastaví se, jaké informace o průběhu výpočtů budou vypisovány a jakým způsobem škálovat. Dále nastaví účelovou funkci, kterou mají všechny modely společnou. Pomocí funkce `createMainTMatrix` ze skriptu `myfunctions.php` vytvoří

[4]lp_maker je součástí skriptu lp_maker.php poskytovaného společně s distribucí LP Solve pro php. Pro účely této práce je tento skript upraven, aby ze zadaných parametrů stavěl rovnou modely optimalizace výrobních linek. Přehled jejích parametrů můžete vidět v příloze.

tzv. časovou[5] část soustavy omezení, jako parametry se funkci předají ukazatel na model a součet časů linek pro celé období predikce. Následuje jí funkce `createMainPMatrix`, jíž se jako parametry předají: ukazatel na model, požadavky na výrobu pro celé období a matice strukturních koeficientů. Ta vytvoří tzv. produktovou[6] část soustavy omezení. Jako poslední se nastaví, typ účelové funkce na minimalizační. Tím je paměti vytvořen model pro kontrolu řešitelnosti úlohy - základní úloha.

V další části funkce `lp_maker` sestaví modely jednotlivých cyklů. To provede tak, že pro každý cyklus zkopíruje základní úlohu, ke které potom funkcemi `createDayTMat` a `createDayPMat` přidá časovou část a produktovou část daného cyklu. Parametry funkcí `createDayTMat` a `createDayPMat` jsou samozřejmě ukazatele na model daného cyklu, pravé strany omezujících podmínek (pro časovou část vektor volných časů linek daného cyklu, pro produktovou část vektor požadavků pro daný cyklus), číslo cyklu, jehož model stavíme, a nakonec, pouze u `createDayPMat,` matici strukturních koeficientů.

V tuto chvíli jsou modely celkového období a jednotlivých cyklů postaveny a ukazatele na ně jsou uloženy v proměnných. Nejprve se vyřešením úlohy pro celé období otestuje, jestli dané podmínky na období jsou splnitelné. Pokud ano, přistoupí se k řešení jednotlivých cyklů. Pro každý pracovní cyklus k se nastaví jeho vektor pravých stran a následně se vyřeší. V případě nalezeného optimálního řešení se vektor výsledků uloží k tomuto cyklu, stejně tak jeho zbylé pracovní časy, které jsou přiřazeny vektoru rezerv cyklu k. Pokud nebylo nalezeno řešení, číslo cyklu se uloží do zásobníku a vektor rezerv se nastaví na volné časy linek aktuálního cyklu.

Dokud zásobník nevyřešených cyklů není prázdný, vezme se první hodnota z něj a nastaví se jako aktuální cyklus h, číslo této úlohy a její vektor rezerv se uloží, číslo úlohy se bude dále dekrementovat (resp. inkrementovat) při získávání rezerv, vektor rezerv bude využit pro pozdější výpočet půjčených rezerv. Dále se sníží číslo úlohy k, pokud $k < 0$, nastaví se změna čísla úlohy na inkrementaci, a číslo úlohy $k = h + 1$. Pokud $k = n$, vyskytla se nespecifikovaná chyba a algoritmus se s chybovou hláškou ukončí.

[5] tzv. časová část představuje tu část soustavy omezení, která vymezuje podmínky na nepřekročení volných časů linek

[6] tzv. produktová část představuje tu část soustavy omezení, která určuje požadavky množství vyrobených produktů

Nyní se přičte k časové části omezení úlohy h vektor rezerv cyklu k. Zavolá se funkce `solve`, pro pokus o vyřešení úlohy h. Pokud nebylo nalezeno řešení, postup se opakuje.

V případě, že se povedlo najít řešení, výsledky se přiřadí k vektoru výsledků dané úlohy. Nyní se pro každou linku porovná, jestli její využitý čas je menší nebo stejný jako její původní volný čas, pokud ano, nastaví se rezerva linky jako rozdíl původního času a využitého času. V opačném případě se rezerva dané linky rovná nule. Rezervy aktuální linky cyklu k se rovnají původním rezervám zmenšeným o rozdíl aktuálně využitého času linky a původního časového omezení linky cyklu h. Informace o tom, která linka poskytla kolik času v cyklu k, cyklu h se uloží do pole, pro pozdější výpočet plánu výroby z rezerv. Nakonec, pokud nebylo číslo úlohy nastaveno na inkrementaci, rezervy linky všech cyklů nižších než h a vyšší než k, se vynulují. V případě, že k již bylo inkrementováno, vynulují se rezervy dané linky pro všechny cykly nižší než h a zároveň všechny vyšší než h, ale nižší než k. Následně se úloze h vrátí její původní omezení linek.

V případě už již je zásobník nevyřešených cyklů prázdný, zavolá se funkce `doReserves`, která jako parametry vyžaduje vektor odkazů na modely jednotlivých cyklů, matici strukturních koeficientů a nakonec trojrozměrné pole, ve kterém jsou uloženy informace o tom, kolik cyklus h využil rezerv z linek cyklů k. Návratovou hodnotou této funkce potom je pole rezervních výrobních plánů všech cyklů k.

Poslední část skriptu `reseni.php` zajišťuje výpis získaných dat, v podobě výrobního plánu linek pro všechny cykly období a uvolnění vytvořených modelů z paměti.

Veškeré zdrojové kódy jsou uloženy na CD v příloze práce. V době psaní práce bylo zprovozňování aplikace na univerzitním serveru v procesu. Aplikace bude dostupná na `ov.nti.tul.cz`.

5 Závěr

Cílem bakalářské práce bylo vytvořit webové rozhranní programu LP Solve a tím umožnit uživatelům rozhraní řešit jak obecné úlohy lineárního programování, tak i specializovaný typ úlohy optimalizace procesu výrobních linek. Úkol se podařilo splnit, i když ne k úplné spokojenosti autora. Webové rozhraní má primitivní vzhled a výstup úlohy výrobních linek by mohl vypisovat výrobní plán přehledněji, případně by mohl poskytovat další možnosti práce se získanými i vkládanými daty. Při realizaci bakalářské práce jsem si připomenul vědomosti z předmětu Operační výzkum, procvičil si schopnosti programování v jazyce PHP a naučil se pracovat s funkcemi LP Solve. V případě další práce na tomto projektu by, s poznatky získanými při realizaci práce, jistě nebyl problém pole působnosti projektu webového rozhraní programu LP Solve dále rozšířit.

Použité zdroje

[1] Rálek, Petr. Operační výzkum. Elektronická skripta, Liberec, 2008. 26 s.

[2] KOŘENÁŘ, Václav a Milada LAGOVÁ. *Optimalizační metody*. Vyd. 1. V Praze: Oeconomica, 2003, 187 s. ISBN 80-245-0609-2.

[3] Gnu.org. [online]. [cit. 2014-05-03]. Dostupné z: https://www.gnu.org/licenses/old-licenses/lgpl-2.0.htm

[4] XLI. [online]. [cit. 2014-05-03]. Dostupné z: http://lpsolve.sourceforge.net/5.5/XLI.htm

[5] Příkaz lp_solve. [online]. [cit. 2014-05-03]. Dostupné z: http://lpsolve.sourceforge.net/5.5/lp_solve.htm

[6] Reference lp_solve API. [online]. [cit. 2014-05-03]. Dostupné z: http://lpsolve.sourceforge.net/5.5/

[7] Rálek, Petr. Operační výzkum. Řešené příklady, Liberec, 2009. 28 s.

Manuál k používání webového rozhraní LP Solve

Na první straně je třeba vybrat, jaký typ úlohy chceme řešit. Na výběr je ze dvou možností:

1. obecná úloha lineárního programování
2. optimalizace provozu výrobních linek

Obecná úloha lineárního programování

Úlohu si pro lepší pochopení ukážeme na příkladě 1. Hledáme minimum účelové funkce (1). Omezené podmínkami (2).

Příklad 1:

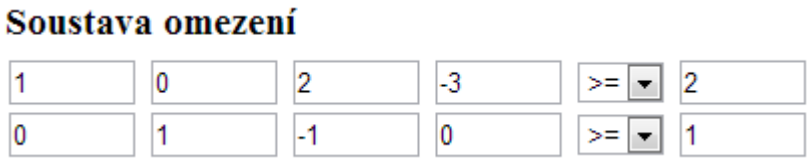
$$\min z = x_1 + x_2 + 2x_3 + x_4, \tag{1}$$

$$\begin{aligned} x_1 + 2x_3 - 3x_4 &\geq 2, \\ x_2 - x_3 &\geq 1, \\ x_i &\geq 0 \ \forall i. \end{aligned} \tag{2}$$

Nejprve se zadají základní rozměry úlohy – počet neznámých a počet omezujících podmínek, pro náš příklad - 4 neznámé a 2 omezující podmínky. V další části se zobrazí možnost výběru typu účelové funkce a pole pro zadání cen proměnných. Dle příkladu, vybereme minimalizaci a vyplníme pole cen, po řadě 1, 1, 2, 1. Jak je vidět na obr. 1.

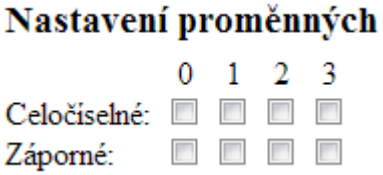
obr. 10 Zadávání účelové funkce.

Dále je nutno vyplnit strukturní koeficienty proměnných – pro první řádek po řadě 1, 0, 2, -3 a 0, 1, -1, 0 pro druhý. Potom vybereme z možností $\geq, =, \leq$ pro nastavení typů podmínek a nakonec zadáme pravé strany podmínek tedy 2 a 1. To je vidět na obr. 2.



obr. 11 Soustava omezení.

Poslední částí zadávání úlohy je nastavení, u kterých proměnných je vyžadována celočíselnost nebo které proměnné mohou nabývat záporných hodnot. Protože náš příklad takové požadavky na proměnné nemá, kolonky nevyplňujeme. To zobrazuje obr. 3.



obr. 12 Nastavení proměnných.

Po potvrzení formuláře se v případě nalezení optimálního řešení objeví výsledek úlohy, zobrazený na obr. 4.

```
z* = 3
x* = {2,1,0,0}
```

obr. 13 Výstup obecné úlohy.

Optimalizace provozu výrobních linek

Úlohu výrobních linek si předvedeme na příkladu 2, získaném ze sbírky příkladů pro předmět Operační výzkum [7]:

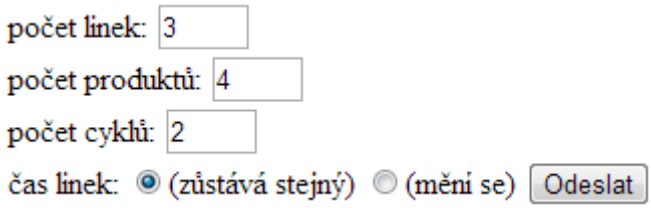
Příklad 2:

Výrobce obalů vyrábí čtyři druhy pivních lahví. Požadavky odběratelů jsou 200, 50, 80 a 30 tis. ks. Obaly jsou vyráběny na třech lisech s měsíční kapacitou 160, 320 a 160 hodin. Na obr. 5 je zobrazeno kolik tisíc kusů lahví vyrobí každá linka za hodinu a s jakými náklady. Úkolem je stanovit plán výroby aby náklady byly minimální.

tabulka 2 Data k příkladu 2.

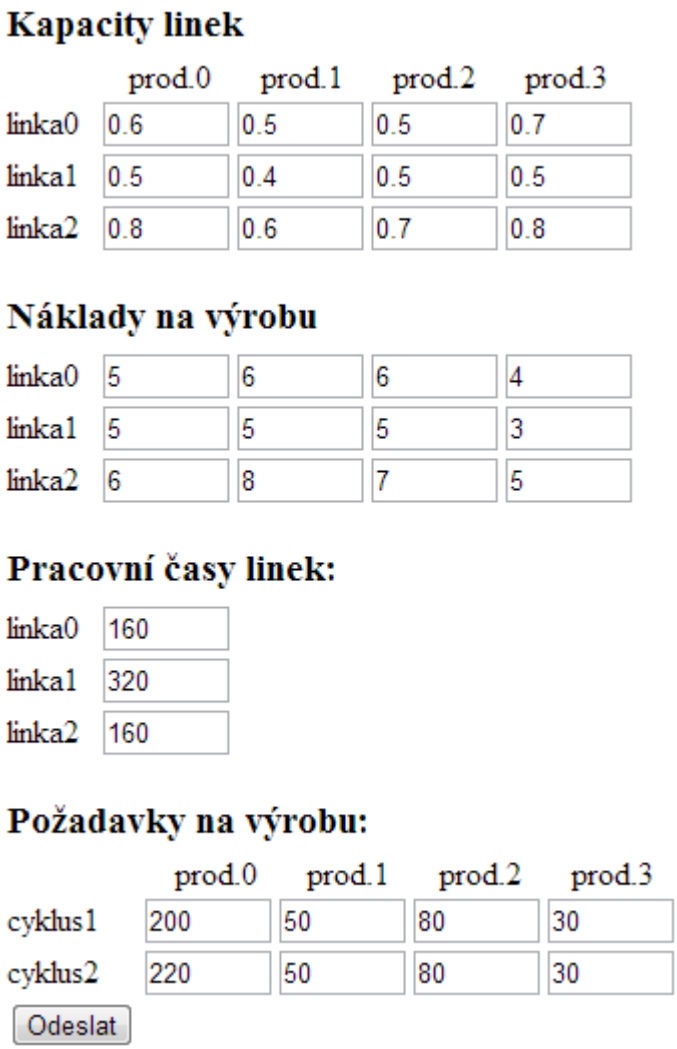
	linka 1	linka 2	linka 3	požadavek (tis. ks)
Obal 1	0.6, 5	0.5, 5	0.8, 6	200
Obal 2	0.5, 6	0.4, 5	0.6, 8	50
Obal 3	0.5, 6	0.5, 5	0.7, 7	80
Obal 4	0.7, 4	0.5, 3	0.8, 5	30
kapacita linky (h)	160	320	160	

V první části nastavení úlohy zadáme, pro kolik linek hledáme časový rozvrh a kolik produktů budou linky vyrábět. Dále zadáme, jak dlouhé predikční období nás zajímá, tedy pro kolik cyklů výroby se úloha bude řešit, příklad 2 lehce upravíme tak, že ho nastavíme pro 2 cykly. Nakonec nastavíme, jestli se budou volné časy linek v průběhu období měnit, v našem případě nikoliv. Tuto část můžeme vidět na obr. 6.



obr. 14 Formulář pro nastavení úlohy výrobních linek

Po stisknutí odeslat se zobrazí formulář na zadání konkrétních parametrů úlohy. Nejprve zadáme první část ze sloupců linka do polí Kapacity linek. Dále nastavíme náklady na výrobu, kterým v příkladě 2 odpovídá druhá část sloupců linka. Do polí v oddílu Pracovní časy linek se zadají data z posledního řádku příkladu 2. Nakonec se po řádcích zadají požadavky pro jednotlivé produkty, z posledního sloupce příkladu 2. Pro druhý cyklus nastavíme požadavky o trochu vyšší, aby ve výsledku bylo třeba využít časových rezerv prvního cyklu, zvedneme tedy požadavek na obal1o 20. To je vidět na obr. 7.



obr. 15 Formulář pro parametry úlohy výrobních linek.

Po odeslání formuláře se zobrazí na výstupu výrobní plán pro oba cykly. Jak je vidět na obr. 8, pod číslem linky jsou ve sloupcích vypsány časy, po které linka má vyrábět jednotlivé produkty. V pravé části jsou potom vypsány informace o volných rezervách a jejich poskytování ostatním cyklům. V našem výstupu pro příklad 2 je tedy vidět, že v cyklu1 se bude 10hodin na lince2 vyrábět obal3.

cyklus0

linka0:	linka1:	linka2:
x0=140	x0=0	x0=145
x1=20	x1=100	x1=0
x2=0	x2=160	x2=0
x3=0	x3=60	x3=0

Původní rezervy: r0=0 r1=0 r2=15
Nové rezervy: r0=0 r1=0 r2=5
cyklus1(linka2=10) : x(0,0,0,0,0,0,0,0,10,0,0,0);

cyklus1

linka0:	linka1:	linka2:
x0=140	x0=0	x0=170
x1=20	x1=100	x1=0
x2=0	x2=160	x2=0
x3=0	x3=60	x3=0

Původně bez řešení
Nové rezervy: r0=0 r1=0 r2=0

obr. 16 Výrobní plán příkladu 2.

Popis funkcí LP Solve API

make_lp – vytvoří model a vrátí číslo ukazatele na model

- `lp = lpsolve('make_lp', rows, columns)`
  - rows – počet řádků,
  - columns – počet sloupců

add_constraint – přidá k danému modelu (lp) jedno omezení a vrátí TRUE pokud byla operace úspěšná. Pro PHP je add_constraint implementovaná stejně jako add_constraintex, která je optimalizovaná pro práci s řídkými maticemi

- `lpsolve ('add_constraint', lp, row, colno, constr_type, rh)`
  - `lp – existující lp model,`
  - `row – vektor koeficientů v řádku`
  - `constr_type – typ rovnice LE – less or equal, EQ – equal, GE – greater or equal`
  - `rh – hodnota pravé strany řádku`

set_verbose – nastavuje, jaké informace se budou reportovat uživateli

- `lpsolve('set_verbose', lp, verbose)`
  - `verbose` – číslo nastavující výstup dle tabulky 1

tabulka 3 Nastavení funkce verbose.

NEUTRAL (0)	Budou zobrazovány jen výstupy debugovacích metod
CRITICAL (1)	Budou vypisovány pouze zásadní chyby (např.: nedostatek paměti)
SEVERE (2)	Vypisuje pouze chyby.

IMPORTANT (3)	Vypisuje varování a chyby.
NORMAL (4)	Vypisuje standardní informace.
DETAILED (5)	Vypisuje podrobnosti jako velikost modelu, nebo zlepšující kroky B&B
FULL (6)	Vypíše všechny zprávy.

set_obj_fn – nastavuje hodnoty účelové funkce

- `lpsolve('set_obj', lp, [values])`
  - `[values]` - vektor cen

solve – vyřeší daný model, pokud je zadaný set_timeout, po překročení této doby, pokud bylo nalezeno neoptimální řešení, ho vrátí jako výsledek. Pro seznam návratových hodnot viz. tabulku2

- `lpsolve('solve', lp)`

tabulka 4 Návratové hodnoty funkce solve.

NOMEMORY (-2)	Nedostatek paměti
OPTIMAL (0)	Bylo nalezeno optimální řešení
SUBOPTIMAL (1)	Bylo nalezeno přípustné řešení. • Byl dosáhnut časový limit • set_break_at_first byl nastaven (vrací první nalezené řešení) • set_break_at_value byl nastaven a bylo nalezené lepší řešení než zadaná hodnota • set_mip_gap bylo nastaveno • byla nainstalována funkce put_abortfunc a tato vrátila hodnotu TRUE • v určité době se narazilo na nedostatek paměti
INFEASIBLE (2)	Model nemá přípustné řešení
UNBOUNDED (3)	Model má nekonečně mnoho řešení
DEGENERATE (4)	Model je degenerativní
NUMFAILURE (5)	Početní chyba
USERABORT (6)	Funkce abort vrátila TRUE.
TIMEOUT (7)	Byl překročen časový limit.
PRESOLVED (9)	Řešením modelu je přednastavené řešení
PROCFAIL (10)	B&B funkce se nezdařila
PROCBREAK (11)	B&B se nezdařilo kvůli funkcím break-at-first nebo break-at-value
FEASFOUND (12)	Bylo nalezeno přípustné B&B řešení
NOFEASFOUND (13)	Nebylo nalezeno přípustné B&B řešení

get_objective – vrátí funkcí solve nalezené řešení

- `lpsolve('get_objective', lp);`

get_variables – vrátí řešení jednotlivých proměnných získaných funkcí solve

- `[var, return] = lpsolve('get_variables', lp,)`
  - `var` - pole obsahující hodnoty proměnných

    - `return` – true pokud byla operace úspěšná

free_lp– funkce uvolní paměť alokovanou pro uložení modelu
- `lpsolve('free_lp', lp)`

set_rh_vec – nastaví vektor pravích stran
- `lpsolve('set_rh_vec', lp, [rh])`
    - `[rh]` - vektor pravých stran

set_minim – nastaví typ úlohy na minimalizační
- `lpsolve('set_minim', lp)`

set_maxim – nastaví typ úlohy na maximalizační
- `lpsolve('set_maxim', lp)`

set_unbounded – nastaví omezení proměnné na -nekonečno – nekonečno
- `lpsolve('set_unbounded', lp, i)`
    - `i` - číslo proměnné, kterou chceme nastavit

set_int – nastaví proměnnou jako celočíselnou
- `lpsolve('set_int', lp, i)`
    - `i`- číslo proměnné, kterou chceme nastavit

read_params – přečte parametry pro výpočet ze souboru nastavení
- `lpsolve('read_params', lp, path)`
    - `path` - udává cestu k souboru nastavení

set_lowbo – nastaví dolní omezení proměnných
- `lpsolve('set_lowbo', lp, [values])`
    - `[values]`- pole s hodnotami omezení

set_upbo – nastaví horní omezení proměnných
- `lpsolve('set_upbo', lp, [values])`
    - `[values]` - pole s hodnotami omezení

set_scaling – nastaví parametr škálování
- `lpsolve('set_scaling', lp, scalemode)`
    - `scalemode` - nastavuje algoritmus pro škálování, pro podrobnosti viz. [6]

copy_lp – zkopíruje vybraný model v paměti a vrátí na něj ukazatel
- `clp = lpsolve('copy_lp', lp)`

set_row_name – pojmenuje jeden řádek modelu, nemá vliv na výpočty
- `lpsolve('set_row_name', lp,j,name)`
    - `j` - číslo řádku pro pojmenování, nultý řádek zastupuje účelovou funkci
    - `name` - budoucí jméno řádku

get_rh – vrátí pravou stranu řádku
- `rh = lpsolve('get_rh', lp, j)`
    - `j` - číslo řádku

del_constraint – smaže jeden řádek ze soustavy omezení
- `lpsolve('del_constraint', lp, j)`
    - `j` - číslo řádku

lp_maker – vytvoří model pro základní úlohu výrobních linek, na nějž vrátí ukazatel jako návratovou hodnotu, a modely jednotlivých cyklů s ukazateli uloženými v poli `day`. Pro účely práce jsou nutné jen první 4 atributy – `f, a, b, t.`

- `lp_maker(f,a,b,t,e,vlb,vub,xint,scalemode,setminim)`
  - `f` – vektor cen koeficientů pro účelovou funkci
  - `a` – m×p matice strukturních koeficientů
  - `b` – n×p matice pravých stran
  - `t` – n×m matice volných časů výrobních linek