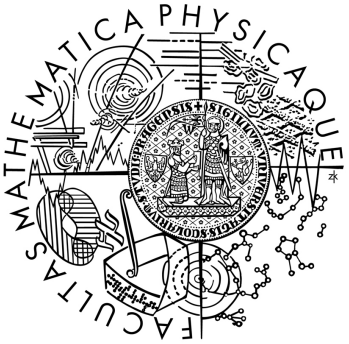
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Ondřej Kafka

Lot-sizing problém

Katedra pravděpodobnosti a matematické statistiky

Vedoucí bakalářské práce: RNDr. Martin Branda, Ph.D.

Studijní program: Matematika

Studijní obor: Obecná matematika

Praha 2011

Na tomto místě bych velice rád poděkoval vedoucímu mé bakalářské práce RNDr. Martinovi Brandovi, Ph.D., za nápad, vedení práce a cenné připomínky.

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

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

V ..... dne ..... Podpis autora

Název práce: Lot-sizing problém

Autor: Ondřej Kafka

Katedra: Katedra pravděpodobnosti a matematické statistiky

Vedoucí bakalářské práce: RNDr. Martin Branda, Ph.D.

Abstrakt: V předložené práci se seznámíme se základními pojmy z oblasti lot-sizingu. Představíme si Wagner-Whitinův problém a odvodíme algoritmus dynamického programování, jak jej řešit. Dále se podíváme na případ problému PCLSP (Profit maximizing capacitated lot size problem) s pevnými cenami a zanedbatelnými přípravnými náklady a budeme jej řešit pomocí speciálního algoritmu lineárního programování. Vše se pokusíme vysvětlit na konkrétních příkladech. V závěru práce ověříme efektivitu uvedených algoritmů pomocí numerické studie na náhodných datech, porovnáme rychlost naprogramovaných algoritmů s profesionálním optimalizačním nástrojem Gurobi.

Klíčová slova: Lot sizing, dynamické programování, lineární programování

Title: Lot-sizing problem

Author: Ondřej Kafka

Department: Department of probability and mathematical statistics

Supervisor: RNDr. Martin Branda, Ph.D.

Abstract: In the present work, we define the basic concepts of lot-sizing. We introduce Wagner-Whitin's dynamic lot size problem and derive a dynamic programming algorithm for the solution. Next we look at the case of PCLSP (Profit maximizing capacitated lot size problem) problem with fixed prices and negligable setup costs and solve it using specialized linear programming algorithm. Everything we try to explain with concrete examples. In the end we verify the efficiency of those algorithms by numerical study on random data comparing the performance of programmed algorithms with the professional optimization solver Gurobi.

Keywords: Lot-sizing, dynamic programming, linear programming

Obsah

1. Úvod

1.1 Seznámení s problémem

Management společnosti stojí před úkolem naplánování výrobního procesu. Cílem je najít realizovatelný plán časů výroby a příslušných množství jednotlivých produktů, který pokryje poptávku a bude splňovat další případné podmínky. Z ekonomických důvodů nestačí jen najít takový plán, ale je třeba vybrat ten nejlepší. Produkce je posuzována funkcí sčítající náklady na výrobu a uskladnění. Obecně lot-sizing zahrnuje určování optimální velikosti výroby (případně dodávek, objednávek) a jejich načasování uspokojující poptávku na nějakou danou dobu do budoucna a to za účelem minimalizace nákladů. Existuje celá řada lot-sizing modelů a často se dají formulovat jako úlohy (celočíselného) lineárního programování. Řešit veliké úlohy pomocí lineárního programování může být časově velmi náročné až nemožné, proto matematici hledají rychlejší algoritmy nebo různé heuristiky, které se dokáží optimálnímu řešení alespoň přiblížit.

Lot-sizing modely jsou dnes užitečné například pro společnosti, jakými jsou např. farmaceutické firmy, stáčírny nealkoholických nápojů, kosmetické firmy, výrobci nábytku atd., které nevytváří neustále jeden a tentýž výrobek, ale jednotlivé výrobky produkují v určitých dávkách, šaržích a své zboží naskladní. Další výrobu započnou až nějaký druh zboží začne docházet.

1.2 Klasifikace lot-sizing modelů a základní terminologie

Lot-sizing modelem nazveme úlohu hledání ekonomicky nejvýhodnějšího plánu produkce. V této sekci uvedeme ty nejčastější charakteristiky užitečné při rozlišování jednotlivých modelů.

- Plánovací horizont: Konečný nebo nekonečný.
- Časová stupnice: Může být buď spojitá nebo diskrétní. Diskrétní stupnici rozdělujeme na rovnoměrnou a nerovnoměrnou.
- Dostupnost jednotlivých parametrů: Pokud známe všechna potřebná data a parametry (nejčastěji např. poptávka), mluvíme o modelu deter-

ministickém. Naopak o stochastický model se jedná v případě, že musíme nějaký parametr předpovídat.

- Účelová funkce: Většinou účelovou funkcí lot-sizing modelu bývá minimalizace součtu všech nákladů, nicméně občas ji lze definovat např. jako maximalizaci zisku, minimalizaci času.

- Náklady: V lot-sizing modelech existují hlavně dva základní typy. Náklady na uchovávání, skladování (holding costs, inventory costs) a jisté přípravné náklady (setup costs). Skladovací náklady obvykle vyjadřuje lineární funkce závislá na množství výrobků na skladě. Tyto náklady zahrnují různé skladové operace jako např. pronájem, pojištění, daně nebo zastarávání.
  S každým výrobním procesem jsou navíc spojeny náklady na přípravu výroby, mezi které patří různé výdaje např. na přenastavování a čištění výrobních strojů.
  Nesmíme zapomínat ani na samotné výrobní náklady, ty ale bývají často považovány za konstantní v čase, a tedy pro účely hledání optima nezajímavé.

- Kapacita: Množství, která je schopen vytvořit výrobní zdroj v jednotlivých periodách, mohou být omezená (capacitated) nebo neomezená (uncapacitated). Stejně tak může být omezena kapacita skladu ať už shora nebo zdola.

- Počet vyráběných produktů: Rozlišujeme mezi jedním (single) a více než jedním (multi).

- Neuspokojená poptávka: Ve většině modelů se chce, aby bylo vyhověno poptávce, existují ale modely, kde tomu tak není. Neuspokojená poptávka po zboží může být odložena na další období (backlogging) a to za určitý trestný poplatek v účelové funkci nebo jednoduše ztracena (lost sales). Backlogging můžeme uvažovat např. v modelu výrobce automobilů nebo kvalitního nábytku, kdy je zákazník ochoten si nějakou dobu počkat na své zboží. Naopak třeba zákazníci výrobce potravin v případě, že nedostanou, co chtějí, jdou jinam (lost sales).

Poznamenejme, že model nemusíme nutně definovat přímo ve výrobním prostředí, ale také pro případ, kdy zboží nevyrábíme, ale objednáváme od dodavatele. Kromě uchovávacích nákladů pak namísto přípravných nákladů uva-

žujeme výdaje za uskutečnění objednávek (ordering costs), např. přepravní náklady. Obdobně se nepoužívá pojem backlogging, ale backordering.

1.3 Historická zmínka

Nejstarším a také pravděpodobně nejznámějším lot-sizingovým modelem je model EOQ (Economic order quantity, Harris [4], 1913), který v českém „objednávacím“ prostředí můžeme formulovat následovně:

$$EOQ = \sqrt{\frac{2DS}{H}},$$

kde

$$\begin{aligned} EOQ &= \text{optimální velikost objednávky,} \\ S &= \text{náklady na jednu objednávku (Kč),} \\ D &= \text{celková roční poptávka po produktu (ks),} \\ H &= \text{roční náklady na skladování jednotky zboží (Kč/ks),} \end{aligned}$$

za platných předpokladů: jeden druh zboží, konstantní a známá výše poptávky, konstantní jednotkové nákupní náklady a konstantní jednotkové přepravní náklady, neomezený plánovací horizont a spojitá časová stupnice.

Na několik desítek let se EOQ modelu nedostalo příliš velkého uznání i třeba kvůli probíhajícímu období velké hospodářské krize. Po druhé světové válce se začaly firmy potýkat s nadvýrobou a musely optimalizovat své výrobní plány, což také vedlo k odvození dalších modelů. Zásadní krok vpřed učinili Wagner a Whitin [7] v roce 1958, kdy uvažovali model se známou proměnlivou poptávkou, a ten řešili pomocí algoritmu dynamického programování. Jejich legendární článek, který významně ovlivnil další výzkum, podrobně rozebereme v druhé kapitole. Kvůli složité rekurzivní struktuře jejich algoritmu a tehdejším vysokým výpočetním nárokům se další výzkum ubral směrem k vylepšování algoritmu a hledání heuristik přibližujících se k optimálnímu řešení. Další generace modelů již kombinují různá kapacitní omezení, obecnější nákladové funkce (backordering), počet druhů zboží, spojitou časovou stupnici atd.

2. Wagner-Whitin model

Při nekonstantní poptávce či dokonce nekonstantních skladovacích nákladech nejsou splněny předpoklady EOQ modelu, a tedy při jeho použití nedostaneme optimální řešení. Takový problém popsali a navrhli algoritmus Wagner a Whitin v článku [7], který si nyní představíme.

2.1 Matematická formulace

Opět předpokládejme, že výrobní (nákupní) náklady jsou konstantní po celý plánovací horizont, který se skládá z N období (period).
V čase $t = 1, 2, \ldots, N$, označme:

d_t = poptávané, požadované množství (předem známé),
i_t = náklady na uchování jednotky zboží z t-té do $(t+1)$-ní periody,
s_t = přípravné náklady,
x_t = vyrobené (objednané) množství.

Přičemž všechny poptávky a náklady jsou nezáporné. Úkolem je najít plán

$$x_t \geq 0,\ t = 1, 2, \ldots, N,$$

takový, že vyhoví všem poptávkám a to za minimálních celkových nákladů. Na kapacitu výroby přitom neklademe žádná omezení. Abychom dostali přímo problém celočíselného lineárního programování, musíme úlohu přepsat do tvaru

$$\min \sum_{t=1}^{N} s_t b_t + i_t y_t,$$

za podmínek

$$\begin{aligned} y_{t-1} + x_t - y_t &= d_t, & t &= 1, 2, \ldots, N, \\ x_t - M b_t &\leq 0, & t &= 1, 2, \ldots, N, \\ x_t &\geq 0, & t &= 1, 2, \ldots, N, \\ y_t &\geq 0, & t &= 1, 2, \ldots, N, \\ b_t &\in \{0, 1\}, & t &= 1, 2, \ldots, N, \\ x_t, y_t &\in \mathbb{Z}, & t &= 1, 2, \ldots, N, \end{aligned}$$

kde d_t, i_t, s_t, x_t jsou definovány jako výše a navíc

$b_t = \begin{cases} 1, & \text{v čase t se uskuteční výroba (objednávka),} \\ 0, & \text{jinak,} \end{cases}$
$y_t =$ množství uchovávaného zboží zboží z t-té do $(t+1)$-ní periody,
$y_0 = 0$,
$M > 0$ je dostatečně velké číslo.

Číslo M musí být větší než maximální velikost výroby, v našem neomezeném případě postačí například $M > \sum_{t=1}^{N} d_t$. Druhá podmínka pak zajistí požadavek:

$$x_t > 0 \Rightarrow b_t = 1,\ t = 1, 2, \ldots, N.$$

Jedním ze způsobů řešení je prozkoušení všech 2^{N-1} kombinací (za předpokladu, že v prvním období vyrábíme resp. objednáváme). Efektivnější algoritmus lze získat využitím dynamického programování. Nechť I_t značí množství zboží na skladě při vstupu do t-tého období a I_1 výchozí množství zboží na skladě. Potom pro období $t \in \{1, 2, \ldots, N\}$ platí

$$I_t = I_1 + \sum_{j=1}^{t-1} x_j - \sum_{j=1}^{t-1} d_j \geq 0. \tag{2.1}$$

Nyní podle [7] a [2] můžeme definovat rekurzivní funkci představující minimální náklady za období t až N pro dané množství zboží na skladě $I = I_t$ jako

$$f_t(I) = \min_{\substack{x_t \geq 0 \\ I + x_t \geq d_t}} \left[i_{t-1} I + \delta(x_t)\, s_t + f_{t+1}(I + x_t - d_t) \right], \tag{2.2}$$

kde

$$\delta(x_t) = \begin{cases} 0, & \text{když } x_t = 0, \\ 1, & \text{když } x_t > 0. \end{cases} \tag{2.3}$$

V období N máme

$$f_N(I) = \min_{\substack{x_N \geq 0 \\ I + x_t = d_N}} \left[i_{N-1} I + \delta(x_N)\, s_N \right]. \tag{2.4}$$

Postupně získáme f_1 jako funkci I, a tedy $f_1(I_1)$ bude hledaná optimální hodnota. Spočítat přímo optimální hodnotu z takto definované funkce f_1 může být velmi obtížné. Proto se budeme snažit využít vlastnosti problému, abychom bez ztráty optimality řešení výrazně omezili množiny hodnot, kterých mohou nabývat jednotlivá I_t a x_t. K tomu budeme potřebovat několik tvrzení. Navíc předpokládejme, že $d_1 \geq 0$ je zbytek poptávky v čase 1,

na který již nestačí počáteční zásoby I_1, a poté položme $I_1 = 0$[1]. Následujících pět tvrzení pak lze včetně důkazů nalézt v článku [7].

Tvrzení 1. *Existuje optimální plán takový, že $I_t x_t = 0$ pro všechna $t \in \{1, 2, \ldots, N\}$, kde I_t značí zboží na skladě vstupující do periody t.*

Důkaz. Zřejmě $I_1 x_1 = 0$. Mějme optimální plán a $t \in \{2, 3, \ldots, N\}$ takové, že $I_t x_t > 0$. Potom lze změnit plán tak, že zboží I_t nebudeme přenášet do času t, ale rovnou zařadíme do x_t. Nevzniknou tak žádné další náklady za přípravu výroby (objednávky). Naopak se ušetří náklady $i_{t-1} I_t \geq 0$. □

Tvrzení 2. *Existuje optimální plán takový, že pro všechna $t \in \{1, 2, \ldots, N\}$ platí*

$$x_t = 0 \text{ nebo } x_t = \sum_{j=t}^{k} d_j \text{ pro nějaké } k \in \{t, t+1, \ldots, N\}.$$

Důkaz. Protože všechny poptávky d_t musí být splněny, jakákoli jiná hodnota nějakého x_t implikuje existenci periody t^* takové, že $I_{t^*} x_{t^*} > 0$, ale podle tvrzení 1 stačí uvažovat pouze takové plány, ve kterých taková možnost nenastává. □

Při hledání minima ve (2.2) se můžeme omezit pouze na hodnoty x_t dané tvrzením 2. Je-li $I_1 = 0$, pak stačí otestovat dohromady $N(N+1)/2$ hodnot $x_t, t = 1, \ldots, N$, k nalezení řešení problému o N periodách.

Tvrzení 3. *Existuje optimální plán takový, že je-li poptávka d_{t^*} uspokojena nějakým $x_{t^{**}}$, $t^{**} < t^*$, pak také poptávky $d_t, t = t^{**} + 1, \ldots, t^* - 1$, jsou uspokojeny $x_{t^{**}}$.*

Důkaz. V plánu, který by nesplňoval podmínku z tvrzení, je $I_{t^{**}} > 0$, a tedy $I_{t^{**}} x_{t^{**}} > 0$, nebo existuje $t' \in \left\{t^{**} + 1, \ldots, t^* - 1\right\}$ takové, že $x_{t'} > 0$, a potom $I_{t'} x_{t'} > 0$, ale opět podle prvního tvrzení stačí uvažovat pouze ta řešení, ve kterých $I_t x_t = 0$ pro všechna $t \in \{1, 2, \ldots, N\}$. □

Další tvrzení říká, kdy je možné rozdělit náš problém na dva menší.

Tvrzení 4. *Nechť pro nějaké období $t \in \{2, 3, \ldots, N\}$ platí $I_t = 0$. Pak je optimální řešit samotný problém pro období $\{1, \ldots, t-1\}$.*

[1] Pokud $I_1 > d_1$, pak za první index považujeme nejmenší t takové, že $I_1 \leq \sum_{\tau=1}^{t} d_\tau$, ve kterém odstartujeme pozdější algoritmus.

Důkaz. Podle (2.2) pro model o N periodách je v období $t-1$

$$f_{t-1}(I) = \min_{\substack{x_{t-1}\geq 0 \\ I+x_{t-1}=d_{t-1}}} \left[i_{t-2}I + \delta\left(x_{t-1}\right)s_{t-1} + f_t(0)\right] \tag{2.5}$$

a pro model o $t-1$ periodách podle (2.4) platí

$$g_{t-1}(I) = \min_{\substack{x_{t-1}\geq 0 \\ I+x_{t-1}=d_{t-1}}} \left[i_{t-2}I + \delta\left(x_{t-1}\right)s_{t-1}\right]. \tag{2.6}$$

Tyto funkce se liší pouze konstantou $f_t(0)$. Tedy co je optimální pro (2.5), je optimální i pro (2.6) a z rekurzivní struktury modelu plyne tentýž závěr i pro všechna dřívější období. □

Nyní můžeme nabídnout alternativní formulaci ke (2.2). Nechť $F(t)$ značí minimální náklady za období $1,\ldots,t$. Potom

$$F(t) = \begin{cases} \min\limits_{1\leq j\leq t}\left[s_j + \sum_{h=j}^{t-1}\sum_{k=h+1}^{t} i_h d_k + F(j-1)\right], & t \geq 1, \\ 0, & t = 0. \end{cases} \tag{2.7}$$

Tedy minimální náklady pro prvních t období se skládají z přípravných nákladů v čase $j \in \{1,\ldots,t\}$, skladovacích nákladů pro pokrytí poptávek d_k, $k = j+1,\ldots,t$, a hodnotou samotné optimální strategie v obdobích $\{1,\ldots,j-1\}$. Tvrzení 2,3,4 nám zaručí nalezení optimálního plánu tohoto typu. Minimum v (2.7) nemusí být nutně jednoznačně určeno, a tedy může existovat i více optimálních řešení.

Asi nejužitečnějším tvrzením ze článku [7] je tvrzení 5, díky kterému jsme schopni efektivně vynechávat nevhodná období pro výrobu.

Tvrzení 5. (o plánovacím horizontu)
Pokud se minima v (2.7) *pro období* t^* *nabývá v* $j_0 = t^{**} \leq t^*$, *potom v* (2.7) *pro období* $t > t^*$ *stačí uvažovat pouze* $t^{**} \leq j \leq t$. *Speciálně je-li* $t^{**} = t^*$, *pak stačí uvažovat pouze takové plány, kde* $x_{t^*} > 0$.

Důkaz. Bez ztráty optimality se omezíme na plány specifikované ve tvrzeních 1–4. Mějme plán, ve kterém poptávka d_t je uspokojena $x_{t^{***}}$, kde

$$t^{***} < t^{**} \leq t^* < t.$$

Pak podle tvrzení 3 je d_{t^*} také uspokojena $x_{t^{***}}$. Ale podle předpokladu víme, že náklady se nezvýší přeplánováním tak, aby poptávka d_{t^*} byla pokryta až $x_{t^{**}} > 0$. □

2.2 Příklad

Uvažujme následující vymyšlený příklad. Výrobce uvádí na trh výrobek a na rok dopředu očekává poptávku a náklady při stejném značení v jednotlivých měsících t podle tabulky 2.1. Výrobní náklady na jednotku jsou konstantní.

t	1	2	3	4	5	6	7	8	9	10	11	12
d_t	30	40	50	45	35	29	30	28	25	10	21	26
s_t	50	50	50	50	70	70	70	70	50	50	50	50
i_t	1	1	1	1	1	1	1	1	1	1	1	1

Tabulka 2.1: Příklad Wagner-Whitinova problému

Počítejme podle (2.7).

$$\begin{aligned}
F(1) &= s_1 = \mathbf{50}, \\
F(2) &= \min\left\{s_1 + i_1 d_2, s_2 + F(1)\right\} \\
&= \min\{50 + 1 \cdot 40,\ 50 + 50\} \\
&= \min\{90, 100\} = \mathbf{90}, \\
F(3) &= \min\left\{s_1 + i_1 d_2 + (i_1 + i_2)\, d_3, s_2 + i_2 d_3 + F(1), s_3 + F(2)\right\} \\
&= \min\{50 + 1 \cdot 40 + 2 \cdot 50, 50 + 1 \cdot 50 + 50, 50 + 90\} \\
&= \min\{190, 150, 140\} = \mathbf{140}.
\end{aligned}$$

Nyní již lze podle tvrzení 5 (případ $t^{**} = t^* = 3$) prohlásit, že v našem optimálním plánu se bude vyrábět v časech 1 a 3 a tím se problém redukuje pouze na období 3 až 12.

$$\begin{aligned}
F(4) &= \min\left\{s_3 + i_3 d_4 + F(2), s_4 + F(3)\right\} \\
&= \min\{50 + 1 \cdot 45 + 90, 50 + 140\} \\
&= \min\{185, 190\} = \mathbf{185}.
\end{aligned}$$

Dále již ve výpočtech pokračujme v tabulce.

t	**Poslední měsíc, ve kterém se vyrábí** (tj. hodnota j ze (2.7))											
	1	2	3	4	5	6	7	8	9	10	11	12
1	**50**											
2	**90**	100										
3	190	150	**140**									
4			**185**	190								
5			255	**225**	255							
6				**283**	284	295						
7				373	344	**325**	353					
8						381	**381**	395				
9							431	**420**	431			
10								**440**	441	470		
11								503	**483**	491	490	
12									561	543	**516**	533

Tabulka 2.2: Tabulkové řešení příkladu

Opět hojně využíváme tvrzení 5. Jakmile se v nějakém řádku nabyde minima ve sloupci j, přestáváme prozkoumávat výrobu v dřívějších obdobích. Tučně zvýrazněná čísla v řádcích udávají jednotlivá $F(t)$. Tedy optimálním řešením problému je $F(12) = 516$ peněžních jednotek.

- Hodnoty v tabulce se dají získat dosazováním t a j do (2.7) nebo lze využít znalosti předchozího řádku. Na t-tém místě v úhlopříčce leží vždy $s_t + F(t-1)$ a čísla ve sloupcích pod sebou se liší pouze o součet příslušných uchovávacích nákladů.

- Proto lze z tabulky 2.2 také snadno vyčíst plán x_t. Nechť $I_{m,n}, m < n$, značí celkové náklady v optimálním plánu na skladování z času m do n. Zřejmě

$$\begin{aligned} F(12) &= s_{11} + F(10) + I_{10,12}, \\ F(10) &= s_8 + F(7) + I_{8,10}, \\ F(7) &= s_6 + F(5) + I_{6,7}, \\ F(5) &= s_4 + F(3) + I_{4,5}, \\ F(3) &= s_3 + F(2), \\ F(2) &= s_1 + I_{1,2}. \end{aligned}$$

  $F(12)$ se skládá z s_1, s_3, s_4, s_6, s_8 a s_{11}, tedy výroba proběhne v obdobích 1,3,4,6,8,11. Z tvrzení 1 a 2 dopočteme

$$
\begin{aligned}
x_1 &= 30 + 40 = 70, \\
x_3 &= 50, \\
x_4 &= 45 + 35 = 80, \\
x_6 &= 29 + 30 = 59, \\
x_8 &= 28 + 25 + 10 = 63, \\
x_{11} &= 21 + 26 = 47.
\end{aligned}
$$

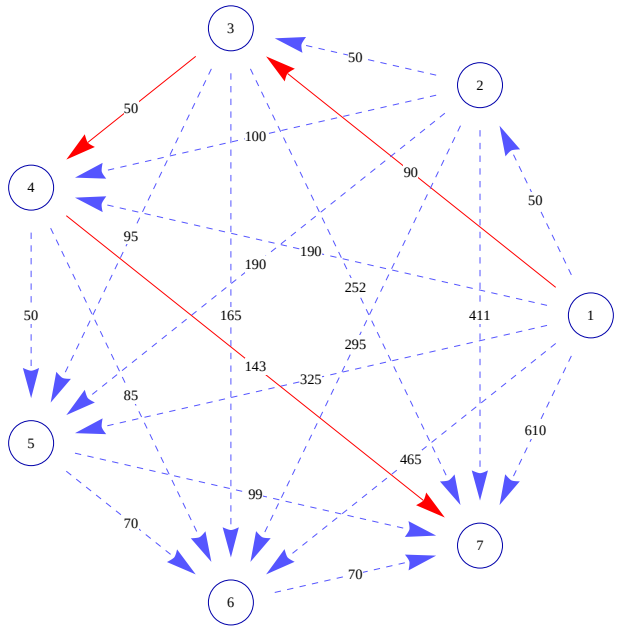
- V nejhorším případě se může stát, že vyplníme celý levý dolní roh tabulky. Jsou-li $N \times N$ její rozměry, budeme potřebovat maximálně $N(N+1)/2$ výpočtů. Algoritmus (2.7) je pak v nejhorším případě stejně efektivní jako (2.2) s využitím tvrzení 2.

- Budeme-li řešit stejný problém jen pro prvních osm měsíců, pak zřejmě existují 2 různá optimální řešení.

- Princip fungování algoritmu lze také ukázat na teorii grafů. Důsledkem tvrzení 1 je možnost převést úlohu na problém hledání nejkratší cesty v orientovaném grafu, jehož vrcholy představují jednotlivá období a hrany $(i, j),\ i < j$, se ocení podle vzorce

$$
w_{ij} = s_i + \sum_{k=i+1}^{j-1} \sum_{t=i+1}^{k} i_t d_k.
$$

  Ohodnocení hrany (i, j) zahrnuje přípravné náklady v období i a součet potřebných skladovacích nákladů pokrývající poptávku až do období $j-1$. Logicky tak přibyde jeden pomocný cílový vrchol. Omezme se v našem příkladu na prvních 6 měsíců. Příslušný graf máme na obrázku 2.1 a použitím vhodného algoritmu nejkratší cesty dostaneme vyznačené optimální řešení. Optimální hodnotu účelové funkce získáme sečtením ohodnocení těchto hran.



Obrázek 2.1: Řešení problému metodou nejkratší cesty v grafu

2.3 Algoritmus

V čase t, $t = 1, 2, \ldots, N$, může algoritmus podle [7] vypadat následovně:

1. Pro každé $j \in \{1, 2, \ldots, t\}$ uvažte možnost výroby v čase j takové, že pokryje poptávky $d_j, d_{j+1}, \ldots, d_t$.

2. U těchto t různých strategií určete celkové náklady dané součtem skladovacích a přípravných nákladů spojených s výrobou v čase j a dříve vypočtených optimálních nákladů za období 1 až $j - 1$.

3. Vyberte tu, která má tyto náklady nejnižší, a označte ji za optimální pro období 1 až t.

4. Pokud $t < N$, přejděte k $t + 1$.

Pro plné využití tvrzení o plánovacím horizontu si stačí pamatovat argument minima a z předchozího kroku a v 1. nechat j probíhat $\{a, a + 1, \ldots, t\}$.

3. PCLSP model

V této kapitole si ukážeme příklad, jakým směrem se ubírá současný výzkum na univerzitní vysoké škole Molde v Norsku. Představíme si speciální algoritmus ze článku [5] řešící jednoduchý lot-sizing problém maximalizace zisku s omezenou produkcí nazvaný PCLSP (Profit maximizing capacitated lot-size problem) za předpokladu, že ceny zboží jsou pevně dány a přípravné náklady jsou zanedbatelné. Za takových předpokladů se problém v praxi nemusí nevyskytovat příliš často, a tak hlavní motivací k jeho řešení bude fakt, že se může objevit jako častý subproblém při řešení samotného PCLSP modelu. Pořádné využití algoritmu pro řešení PCLSP by podle [5] mělo být předmětem dalšího výzkumu.

3.1 Představení problému

PCLSP byl odvozen od CLSP (Capacitated lot-size problem) modelu, který ve své nejjednodušší podobě obnáší minimalizaci celkových nákladů vyvažováním přípravných a skladovacích nákladů za jistých podmínek omezení produkce. Takový problém je obtížnosti NP, jak ukázali Bitran a Yanasse [3], a proto se hledají různé alternativní přístupy řešení. Jakmile v modelu v nějakém období má dojít k překročení kapacity produkce, výroba se musí přesunout do jiného období. Z praktického hlediska však existuje více možností. Výrobce může navýšit kapacitu produkce, nechat zákazníky čekat nebo změnit ceny zboží. Právě poslední možností se zabývá PCLSP model. Jedním z důvodů náročnosti řešení CLSP modelu je pevně daná poptávka, v PCLSP modelu je s ní možné změnou cen hýbat. Učiníme silný předpoklad monopolistického trhu, poptávkové funkce potom volíme lineární.

Nyní se už podle [6] podívejme na verzi PCLSP modelu se zanedbatelnými přípravnými náklady pro jeden druh zboží.

$$\max Z = \sum_{t=1}^{T} \left[(\alpha_t - \beta_t \cdot p_t)\, p_t - h_t I_t - c_t x_t \right], \tag{3.1}$$

za podmínek

$$a_t x_t \leq R_t, \qquad t = 1, \ldots, T, \tag{3.2}$$

$$x_t + I_{t-1} - I_t = \alpha_t - \beta_t \cdot p_t, \qquad t = 1, \ldots, T, \tag{3.3}$$

$$x_t \geq 0, \qquad t = 1, \ldots, T, \tag{3.4}$$

$$I_t \geq 0, \qquad t = 1, \ldots, T, \tag{3.5}$$

$$\frac{\alpha_t}{\beta_t} \geq p_t \geq 0, \qquad t = 1, \ldots, T, \tag{3.6}$$

s proměnnými (x_t, I_t, p_t) a nezápornými parametry $(T, \alpha_t, \beta_t, h_t, c_t, a_t, R_t)$:

x_t = vyrobené množství v období t,
I_t = množství zboží skladovaného z t-tého do $(t+1)$-ního období,
p_t = cena jednotky zboží v období t,
T = celkový počet období,
α_t = konstanta v lineární poptávkové funkci v období t,
β_t = směrnice přímky v lineární poptávkové funkci v období t,
h_t = náklady na uchování jednotky zboží z t-té do $(t+1)$-ní periody,
c_t = jednotková výrobní cena v období t,
a_t = spotřeba omezeného zdroje v období t,
R_t = množství dostupných zdrojů v období t.

3.2 Zjednodušující předpoklady

Podle [5] uvažujme následující zjednodušení problému.

3.2.1 Omezení produkce

Bez újmy na obecnosti lze nerovnost (3.2) nahradit $x_t \leq \hat{R}_t$, kde $\hat{R}_t = \frac{R_t}{a_t}$.

3.2.2 Pevné ceny

Za předpokladu, že ceny $p_1, \ldots, p_N$ jsou dány, řekněme $\hat{p}_1, \ldots, \hat{p}_N$, účelovou funkci (3.1) přepíšeme jako:

$$\max \; Z = \sum_{t=1}^{T} (\alpha_t - \beta_t \cdot \hat{p})\, \hat{p}_t - \sum_{t=1}^{T} [h_t I_t + c_t x_t] = C - \sum_{t=1}^{T} [h_t I_t + c_t x_x] \tag{3.7}$$

neboli

$$\min \ \hat{Z} = \sum_{t=1}^{T} \left[h_t I_t + c_t x_t \right]. \tag{3.8}$$

Dodatečně položením $\hat{D}_t = \alpha_t - \beta_t \cdot \hat{p}_t$ lze problém (3.1)–(3.6) definovat jako úlohu lineárního programování:

$$\min \ \hat{Z} = \sum_{t=1}^{T} \left[h_t I_t + c_t x_t \right], \tag{3.9}$$

za podmínek

$$x_t \leq \hat{R}_t, \qquad t = 1, \ldots, T, \tag{3.10}$$

$$x_t + I_{t-1} - I_t = \hat{D}_t, \qquad t = 1, \ldots, T, \tag{3.11}$$

$$x_t \geq 0, \qquad t = 1, \ldots, T, \tag{3.12}$$

$$I_t \geq 0, \qquad t = 1, \ldots, T. \tag{3.13}$$

3.2.3 Rozumné předpoklady na výrobní a skladovací náklady

Protože lot-sizing problémy tohoto typu obvykle nemívají příliš velký plánovací horizont, je celkem logické předpokládat

$$c_1 = c_2 = \ldots = c_T = c \text{ a } h_1 = h_2 = \ldots = h_T = h. \tag{3.14}$$

Potom můžeme účelovou funkci v (3.9) vyjádřit:

$$\sum_{t=1}^{T} \left[h_t I_t + c_t x_t \right] = h \sum_{t=1}^{T} I_t + c \sum_{t=1}^{T} x_t. \tag{3.15}$$

Dále je zřejmé, že sečtením levých a pravých stran rovností (3.11) dostaneme

$$\sum_{t=1}^{T} x_t = I_T - I_0 + \sum_{t=1}^{T} \hat{D}_t. \tag{3.16}$$

Pravá strana rovnosti (3.16) je konstantní, h a c taktéž. V důsledku toho můžeme opět přeformulovat účelovou funkci jako:

$$\min \ \bar{Z} = \sum_{t=1}^{T} I_t \tag{3.17}$$

neboli chceme minimalizovat celkové skladované množství. Nyní při vynechání podmínky (3.10) je optimální řešení zbylého problému jasné:

$$x_t^* = \hat{D}_t \text{ a } I_t^* = 0,\ t = 1, \ldots, T. \tag{3.18}$$

Vezmeme-li zpět v úvahu podmínku (3.10) je opět jednoduché si uvědomit, že pro účelovou funkci (3.17) se optimální řešení sestrojí následovně: Při jakémkoli překročení kapacity $\hat{R}_t$ se nejmenší nutné množství přenese do výroby v předchozích co nejbližších obdobích.

3.2.4 Algoritmus

Algoritmus ze článku [5] pro optimální řešení problému (3.9)–(3.13) za předpokladu parametrů podle (3.14) může být formulován takto:

1. Položte $x_t^* = \hat{D}_t, t = 1, \ldots, T$.
2. Pokud $x_t^* \leq \hat{R}_t\ \forall t$, pak konec. ($x_t^*$ je optimální.)
3. Pokud příští období bude $T + 1$, pak konec.
4. Najděte další období τ, kde $x_\tau^* > \hat{R}_\tau$, a nechte vyrobit množství $x_\tau^* - \hat{R}_\tau$ v předchozích nejbližších možných obdobích. (Pokud to nelze, problém nemá přípustné řešení, konec.)
5. Položte $x_\tau^* = \hat{R}_\tau$ a přenastavte $\{x_{\tau-1}^*, x_{\tau-2}^*, \ldots\}$ odpovídajícím způsobem.
6. Jděte na 3.

3.3 Obecnější struktura nákladů

3.3.1 Konstantní výrobní náklady

Pokud předpokládáme

$$c_1 = c_2 = \ldots = c_T = c \text{ a obecné ceny } h_1, \ldots, h_T, \tag{3.19}$$

pak obdobně jako v (3.15)–(3.17) dostaneme mírně pozměněnou účelovou funkci

$$\min\ \hat{Z} = \sum_{t=1}^{T} h_t I_t. \tag{3.20}$$

Naštěstí nová účelová funkce s sebou nepřináší žádné změny algoritmu ze subsekce 3.2.4. Zřejmě stále platí řešení (3.18) při vynechání podmínky (3.10). Podobně při přeplánovávání výroby kvůli podmínce nemá smysl rozšiřovat výrobu přes více období než je nutné, vedlo by to pouze k navýšení nákladů na skladování. V důsledku toho se účelová funkce (3.20) také minimalizuje pomocí algoritmu 3.2.4.

3.3.2 Nerostoucí výrobní náklady

Za situace

$$c_1 \geq c_2 \geq \ldots \geq c_T \text{ a obecných cen } h_1, \ldots, h_T, \tag{3.21}$$

je řešení (3.18) stále optimální pro problém bez omezení kapacity a je vždy nejlevnější zvolit nejbližší možná období, kam přesunout výrobu. Algoritmus 3.2.4 tedy zaručí optimální řešení všech případů kromě těch, kdy nějak rostou výrobní náklady. Taková situace by se měla vyskytovat jen zřídka, většinou je výrobce spíš schopen vyrábět levněji v čase.

3.3.3 Wagner-Whitin costs

Předpokládejme nyní

$$h_t + c_t - c_{t+1} \geq 0, \ t = 1, \ldots, T. \tag{3.22}$$

Tento předpoklad bývá v odborné literatuře označován jako Wagner-Whitin costs. Přepišme rovnost (3.11) do tvaru

$$x_t = \hat{D}_t + I_t - I_{t-1} \tag{3.23}$$

a dosaďme za x_t do účelové funkce (3.9). Dostaneme

$$\hat{Z} = \sum_{t=1}^{T} \left[h_t I_t + c_t \left(\hat{D}_t + I_t - I_{t-1} \right) \right]. \tag{3.24}$$

Nechť „$\stackrel{c}{=}$“ označuje známou relaci rovnosti „až na konstantu“. Počítejme

$$\hat{Z} \stackrel{c}{=} \sum_{t=1}^{T} \left[h_t I_t + c_t I_t \right] - \underbrace{\underbrace{\sum_{t=1}^{T} c_t I_{t-1}}_{=\sum_{t=0}^{T-1} c_{t+1} I_t}}_{\stackrel{c}{=}\sum_{t=1}^{T} c_{t+1} I_t} \stackrel{c}{=} \sum_{t=1}^{T} I_t \left(h_t + c_t - c_{t+1} \right), \tag{3.25}$$

neboť c_t, h_t a $\hat{D}_t$ jsou konstanty, počáteční naplnění skladu I_0 je dáno, $I_T = 0$ a položíme $h_{T+1} = 0$. Tedy účelovou funkci $\hat{Z}$ lze nahradit:

$$\min \ \underline{Z} = \sum_{t=1}^{T} \hat{h}_t I_t, \tag{3.26}$$

kde $\hat{h}_t = h_t + c_t - c_{t+1} \geq 0$ a $h_{T+1} = 0$. Porovnáme-li účelové funkce (3.26) a (3.20), lze náš algoritmus použít i pro případ splňující Wagner-Whitin costs.

V článku [5] lze také nalézt předpoklad $\frac{c_t}{h_t} = \text{konst}$. Tedy

$$c_t = c \cdot h_t, \ c \in \mathbb{R}, \ t = 1, \ldots, T. \tag{3.27}$$

Potom

$$\hat{h}_t = h_t + c \cdot h_t - c \cdot h_{t+1}. \tag{3.28}$$

Bez delšího odvozování je v [5] uvedena účelová funkce (3.26) s tím rozdílem, že

$$\hat{h}_t = h_t + c \cdot h_t + c \cdot h_{t+1}. \tag{3.29}$$

V takovém případě by nemohlo být $\hat{h}_t$ záporné a bylo by možné rovnou použít náš algoritmus. Ukažme si malý protipříklad, ve kterém není splněn předpoklad Wagner-Whitin costs, podle tabulky 3.1.

t	1	2	3	$\sum h_t I_t + c_t x_t$
c_t	1	3	3	
h_t	1	3	3	
R_t	50	40	30	
$\hat{D}_t$	30	35	40	
x_t	35	40	30	280
x_t^*	50	25	30	265

Tabulka 3.1: Příklad nesprávného použití algoritmu

Zřejmě $\frac{c_t}{h_t} = 1$. Pomocí algoritmu dostaneme x_t, přitom optimálním řešením je x_t^*.

3.4 Více vyráběných produktů

Jednodruhový problém lineárního programování (3.9)–(3.13) je snadno rozšiřitelný na případ více druhů zboží.

$$\min \ \hat{\underline{Z}} = \sum_{i=1}^{I} \sum_{t=1}^{T} \left[h_{it} I_{it} + c_{it} x_{it} \right], \tag{3.30}$$

za podmínek

$$\sum_{i=1}^{I} a_{it} x_{it} \leq R_t, \qquad t = 1, \ldots, T, \tag{3.31}$$

$$x_{it} + I_{i,t-1} - I_{i,t} = \hat{D}_{it}, \qquad t = 1, \ldots, T, \ i = 1, \ldots, I, \tag{3.32}$$

$$x_{it} \geq 0, \qquad t = 1, \ldots, T, \ i = 1, \ldots, I, \tag{3.33}$$

$$I_{it} \geq 0, \qquad t = 1, \ldots, T, \ i = 1, \ldots, I. \tag{3.34}$$

Objevuje se nám tu nový index i značící jednotlivé výrobky a jejich celkový počet je I. Navíc předpokládáme společný zdroj výrobních prostředků v období, což vede k podmínce (3.31). Konečně a_{it} značí spotřebu výrobních prostředků pro produkt i v čase t. Vrátíme-li se zpět k původním předpokladům konstantních výrobních a skladovacích nákladů, je zřejmé, že po vynechání podmínky (3.31) nám zůstane problém strukturálně podobný jednodruhovému případu. Proto stačí aplikovat jen lehce pozměněný algoritmus ze subsekce 3.2.4.

Nechť kapacita výrobního zdroje je překročena v období τ. Je třeba jen rozhodnout, který produkt ve volném období začít vyrábět jako první. Zřejmě je nejefektivnější vždy zvolit produkt s nejmenším poměrem $\frac{h_{it}}{a_{it}}$. Takto postupně „opravíme“ všechna problematická období. Nejjednodušší způsob, jak algoritmus naprogramovat, je postupovat odzadu. Z období t stačí přesunout nejnutnější množství nejvýhodnějších položek do $t-1$ a přejít k $t-1$. Bude-li nakonec v prvním období překročena kapacita, úloha nemá řešení. Jaké problémy mohou nastat v případě jiné implementace si ukážeme na příkladech. Bohužel je tato procedura příliš jednoduchá pro obecnější případy nákladů, ale jak již bylo řečeno, pro krátký časový horizont jsou konstantní náklady rozumným předpokladem.

3.5 Příklady

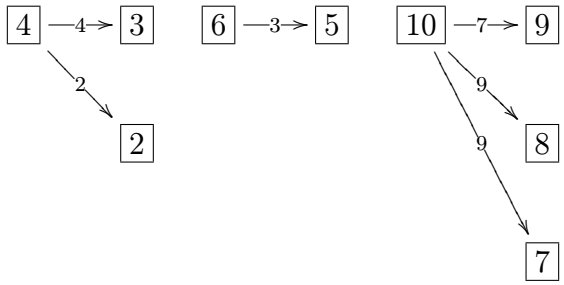
3.5.1 Jednodruhový případ

Podívejme se na jednoduchý již upravený příklad podle tabulky 3.2.

t	1	2	3	4	5	6	7	8	9	10
c_t	1	1	1	1	1	1	1	1	1	1
h_t	1	1	1	1	1	1	1	1	1	1
$\hat{R}_t$	50	50	39	36	37	42	39	43	34	29
$\hat{D}_t$	34	34	35	42	26	45	27	34	27	54

Tabulka 3.2: Příklad jednodruhového PCLSP

Zřejmě poptávka $\hat{D}_t$ převyšuje kapacitu výrobního zdroje $\hat{R}_t$ pro období $t = 4, 6, 10$. Náklady c_t a h_t jsou konstantní, stačí tedy podle našeho algoritmu zvolit $x_t = \hat{D}_t$, $t = 1, 2, \ldots, 10$, a v problematických obdobích výrobu přeplánovat do nejbližších předchozích období. Výrobu tedy přesuneme do dřívějších období podle šipek z diagramu na obrázku 3.1.



Obrázek 3.1: Přeplánování výroby

A docílíme tak optimálního řešení x_t z tabulky 3.3.

t	1	2	3	4	5	6	7	8	9	10
$\hat{R}_t$	50	50	39	36	37	42	39	43	34	29
$\hat{D}_t$	34	34	35	42	26	45	27	34	27	54
x_t	34	36	39	36	29	42	36	43	34	29
I_t	0	2	6	0	3	0	9	18	25	0

Tabulka 3.3: Optimální řešení příkladu

3.5.2 Vícedruhový případ

Mějme úlohu o 4 plánovacích obdobích a 3 druhy zboží vyráběné ze společného výrobního zdroje s vlastnostmi:

$$a_{1t} = 1,\ h_{1t} = 1,\ t = 1, \ldots, 4,$$
$$a_{2t} = 3,\ h_{2t} = 1,\ t = 1, \ldots, 4,$$
$$a_{3t} = 3,\ h_{3t} = 2,\ t = 1, \ldots, 4.$$

Nechť kapacita zdroje je 50 ve všech obdobích, vyrobní náklady každého výrobku jsou konstantní a poptávky volíme podle tabulky 3.4.

období	1	2	3	4
druh 1	$\hat{D}_{11} = 0$	$\hat{D}_{12} = 16$	$\hat{D}_{13} = 21$	$\hat{D}_{14} = 20$
druh 2	$\hat{D}_{21} = 0$	$\hat{D}_{22} = 2$	$\hat{D}_{23} = 3$	$\hat{D}_{24} = 2$
druh 3	$\hat{D}_{31} = 0$	$\hat{D}_{32} = 6$	$\hat{D}_{33} = 10$	$\hat{D}_{34} = 11$

Tabulka 3.4: Předpokládáné poptávky

Za těmito čísly si každý může představit nějaký oblíbený druh podnikání od výrobce metrového textilu až po majitele pískovny. Důležité je, aby jeho produkty byly v rámci možností nekonečně dělitelné, měl minimální náklady na přípravu výroby a na výrobu používal jeden společný zdroj (přístroj, pracovník).

Položme $x_{it} = \hat{D}_{it}$.

	$R_1 = 50$	$R_2 = 50$	$R_3 = 50$	$R_4 = 50$
	$x_{11} = 0$	$x_{12} = 16$	$x_{13} = 21$	$x_{14} = 20$
	$x_{21} = 0$	$x_{22} = 2$	$x_{23} = 3$	$x_{24} = 2$
	$x_{31} = 0$	$x_{32} = 6$	$x_{33} = 10$	$x_{34} = 11$
$\sum a_{it}x_{it}$	0	40	60	59

Tabulka 3.5: První krok algoritmu

Kapacita je překročena v období 3 a 4. Spočteme podíly

$$\frac{h_{1t}}{a_{1t}} = 1,\ \frac{h_{2t}}{a_{2t}} = \frac{1}{3},\ \frac{h_{3t}}{a_{3t}} = \frac{2}{3}.$$

Nejvýhodnější je tedy přesouvat položky druhu 2, pak 3 a nakonec 1.

Začněme s přeplánováním odzadu popsaným algoritmem.

	$R_1 = 50$	$R_2 = 50$	$R_3 = 50$	$R_4 = 50$
	$x_{11} = 0$	$x_{12} = 16$	$x_{13} = 21$	$x_{14} = 20$
	$x_{21} = 0$	$x_{22} = 2$	$x_{23} = 5$	$x_{24} = 0$
	$x_{31} = 0$	$x_{32} = 6$	$x_{33} = 11$	$x_{34} = 10$
$\sum a_{it}x_{it}$	0	40	69	50

Tabulka 3.6: Druhý krok algoritmu

Až nakonec dojdeme k optimálnímu řešení.

	$R_1 = 50$	$R_2 = 50$	$R_3 = 50$	$R_4 = 50$
	$x_{11} = 0$	$x_{12} = 16$	$x_{13} = 21$	$x_{14} = 20$
	$x_{21} = 3$	$x_{22} = 4$	$x_{23} = 0$	$x_{24} = 0$
	$x_{31} = 0$	$x_{32} = 7\frac{1}{3}$	$x_{33} = 9\frac{2}{3}$	$x_{34} = 10$
$\sum a_{it}x_{it}$	9	50	50	50

Tabulka 3.7: Optimální řešení příkladu

Snadno se dopočtou vzniklé náklady.

$$(3) + (5 + \frac{4}{3} \cdot 2) + (2 + 1 \cdot 2) = \frac{44}{3}.$$

Kdybychom postupovali podobně jako v jednodruhovém případě postupně přesunem do nejbližších volných, zřejmě 10, co přebývá ve třetím období, přesuneme do rezervy téže velikosti v předchozím období a čtvrté období vyrovnáme s prvním. Dostaneme řešení z tabulky 3.8

	$R_1 = 50$	$R_2 = 50$	$R_3 = 50$	$R_4 = 50$
	$x_{11} = 0$	$x_{12} = 16$	$x_{13} = 21$	$x_{14} = 20$
	$x_{21} = 2$	$x_{22} = 5$	$x_{23} = 0$	$x_{24} = 0$
	$x_{31} = 1$	$x_{32} = 6\frac{1}{3}$	$x_{33} = 9\frac{2}{3}$	$x_{34} = 10$
$\sum a_{it}x_{it}$	9	50	50	50

Tabulka 3.8: Ukázka nesprávného řešení problému

s náklady $\frac{47}{3}$, což není optimální. Méně výhodné druhy je potřeba přesouvat co nejméně a přes nejkratší úseky. To nám tento postup nezaručí, ani kdybychom ho zkoušeli odzadu.

4. Numerická studie

Prokázání efektivity lot-sizingových algoritmů se často provádí pomocí numerické studie. Ná náhodných datech se testuje jejich rychlost a srovnává se s nějakým osvědčeným způsobem řešení (optimalizační úloha řešená komerčním softwarem).

Představané algoritmy byly naprogramovány v jazyce Free Pascal a jejich rychlost srovnáme optimalizačním nástrojem Gurobi [8] nejnovější verze 4.5 na počítači s dvoujádrovým procesorem AMD Athlon™ QL-62 2GHz, pamětí 3GB a 64-bitovým operačním systémem Windows 7. Jedním z důvodů volby softwaru Gurobi byla jeho obstojná rychlost. Po vyladění[1] parametrů řešení bylo často dosaženo nejrychlejšího času řešení ve srovnání s jinými komerčními optimalizátory, zejména v případech lineárního programování byly výsledky skvělé, u velkých celočíselných úloh horší, ale v rámci možností pořád srovnatelné. Poslední dobou je navíc Gurobi jeden z nejrychleji se vyvíjejících programů v této oblasti. Neméně důležitým faktem při výběru byla legálnost použití, akademickou licenci pro vědecké účely si může obstarat každý student jakékoli ověřené vzdělávací instituce.

Způsob, jakým studii provedeme, je následující. Nejprve se vygeneruje úloha, vyřeší se algoritmem a zaznamená se doba řešení. Tutéž úlohu pak jako soubor formátu lp předáme optimalizátoru, nastavíme potřebné parametry, spustíme vhodnou metodu řešení a porovnáme výsledné časy[2]. Takový způsob předávání byl zvolen, protože je velmi jednoduchý a hlavně univerzální pro různé řešiče optimalizačních úloh. Použití nějakého modelovacího jazyka by nás pouze zdržovalo. Navíc pro získání přibližné průměrné hodnoty času řešení je potřeba pokus se stejnými parametry několikrát opakovat (nejlépe ne hned po sobě kvůli minimalizaci dalších vlivů). Moderní optimalizační nástroje jsou vybaveny řadou zjednodušujících procedur a doby řešení „ořezaných“ úloh se mohou velmi lišit. U většiny úloh provedeme toto opakování alespoň desetkrát, v případě podezřelých výsledků i vícekrát. O veškeré generování, spouštění, zaznamenávání informací a opakování pokusů se stará počítač tak, aby tytéž pokusy bylo možné kdykoli zopakovat. Vybrané procedury a zdrojové kódy lze nalézt v přílohách.

[1] Většina parametrů je nastavena na „automaticky“ a pouze vybíráme vhodnou metodu řešení a způsoby ořezávání.

[2] Uplynulý čas tzv. wall clock time, v současné době vícejádrových procesorů, „turbo boostů“ atd. CPU čas ztrácí na významu. Pro čisté srovnání algoritmů by možná měl svůj smysl, ale to bychom pouze znevýhodnili Gurobi využívající dvoujádro

4.1 Wagner-Whitin algoritmus

Algoritmus (v příloze A.1) porovnáváme s úlohou celočíselného lineárního programování, kterou jsme uvedli již v druhé kapitole. Tu řešíme jako úlohu MIP (mixed integer programming) v programu Gurobi[3]. Abychom se trochu přiblížili reálnému problému, modelujeme poptávku d_t Poissonovým rozdělením, s_t pak diskrétním rovnoměrným rozdělením.

$$\begin{aligned} d_t &\sim Po(\lambda), \\ i_t &= 1, \\ s_t &\sim R(\{40, 45, 50, 55, 60\}). \end{aligned}$$

Číslo n označuje počet plánovacích období. Následují dvě tabulky výsledků pro různé intenzity poptávky. V tabulkách jsou uplynulé časy v sekundách.

n=1000	$\lambda = 5$	$\lambda = 10$	$\lambda = 15$	$\lambda = 20$	$\lambda = 25$
Algoritmus	0.00054s	0.00045s	0.00041s	0.00039s	0.00038s
Gurobi	17.802s	17.142s	8.368s	3.021s	1.910s

Tabulka 4.1: Průměrné časy řešení Wagner-Whitinova problému

n=2000	$\lambda = 5$	$\lambda = 10$	$\lambda = 15$	$\lambda = 20$	$\lambda = 25$
Algoritmus	0.00083s	0.00065s	0.00058s	0.00055s	0.00051s
Gurobi	71.242s	23.377s	24.143s	10.528s	7.194s

Tabulka 4.2: Průměrné časy řešení Wagner-Whitinova problému

Je jasné, že s nižší mírou poptávky vzniká složitější problém, protože je třeba zkoušet mnohem větší počet období, kdy vyrábět (např. méně efektivní využití tvrzení o plánovacím horizontu), což můžeme pozorovat na chování doby řešení algoritmu. U Gurobi metody MIP to nemusí být pravidlem, jak dokládá poslední řádek tabulky 4.2. Již z uvedených tabulek se zdá, že porovnávané metody jsou doslova nesrovnatelné. Přesto pokud chceme vidět nějaké další zajímavé srovnání, zvolme $\lambda = 25$ a zvyšujme n.

[3]Parametry Gurobi MIP: Cuts=0, FlowCoverCuts=2, FlowPathCuts=2, PreCrush=1.

$\lambda = 25$	$n = 5000$	$n = 10000$	$n = 15000$
Algoritmus	0.00091s	0.00159s	0.00227s
Gurobi	32.960s	157.601s	389.740s

Tabulka 4.3: Průměrné časy řešení Wagner-Whitinova problému

Ač se volbou parametrů podařilo snížit dobu řešení programem Gurobi téměř na polovinu, je stále velmi vysoká. Nutno poznamenat, že takto formulovaná úloha MIP není zrovna ideálním řešením, přesto po ní asi kdekdo sáhne díky jednoduchému použití.

V druhé kapitole jsme ukázali, že se problém dá popsat také jako úloha hledání nejkratší cesty v grafu. Nabízí se třeba Dijkstrův algoritmus nebo lze dokonce nejkratší cestu v grafu formulovat přímo jako problém lineárního programování.

Věta. *Je dán orientovaný graf $G = (V, A)$ a pro každou hranu $(i, j) \in A$ ohodnocení $w_{ij} \geq 0$. Hledání nejkratší cesty z vrcholu $s \in V$ do $t \in V$ v grafu G je úloha lineárního programování*

$$Min \sum_{(i,j)\in A} w_{ij}x_{ij},$$

za podmínek

$$x_{ij} \geq 0, \ \forall (i,j) \in A,$$

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} 1 & i = s, \\ -1 & i = t, \ \forall i : (i,j) \in A, \\ 0 & jinak, \end{cases}$$

s proměnnými x_{ij}. Každé optimální řešení (pokud existuje) má všechny x_{ij} rovny 0 nebo 1, přitom množina hran $\{(i, j) \in A : x_{ij} = 1\}$ tvoří nejkratší cestu z vrcholu s do t.

Důkaz. Lze nalézt v [1]. □

Převedeme-li pro zajímavost s pomocí věty původní úlohu až na problém lineárního programování (LP)[4], dosáhneme mnohem lepších výsledků.

[4]Parametry Gurobi LP: Method=3, PreSolve=0, Cuts=0.

	$\lambda = 25$			$\lambda = 5$
	$n = 5000$	$n = 10000$	$n = 15000$	$n = 2000$
Algoritmus	0.00091s	0.00159s	0.00227s	0.00083s
Gurobi MIP	32.960s	157.601s	389.740s	71.242s
Gurobi LP	0.544s	1.108s	1.849s	0.363s

Tabulka 4.4: Průměrné časy řešení Wagner-Whitinova problému

Nicméně porovnávání těchto časů není tak úplně správné. K sestavení grafu a ocenění hran je totiž potřeba obdobný počet výpočetních operací jako v samotném algoritmu (při efektivním vynechávání zbytečných hran).

4.2 PCLSP

V případě PCLSP se jedná rovnou o problém lineárního programování, můžeme tedy očekávat srovnatelnější výsledky.

4.2.1 Jednodruhová verze

Úlohy generujeme takto:

$$\begin{aligned} R_t &\sim R(\{26, \ldots, 55\}), \\ \hat{D}_t &\sim R(\{22, \ldots, 55\}), \\ c &= 1, \\ h &= 1. \end{aligned}$$

Přičemž za R_1 zvolíme nějaké vysoké číslo, aby naše úlohy měly vždy řešení, a porovnáváme pro různý počet plánovacích období n s metodou duální simplex[5]. Zdrojový kód algoritmu v příloze A.2.

	$n = 100$	$n = 1000$	$n = 10000$	$n = 100000$	$n = 10^6$
Algoritmus	0.00049s	0.00065s	0.00213s	0.01768s	0.17172s
Gurobi	0.001s	0.021s	0.379s	5.144s	63.634s

Tabulka 4.5: Průměrné časy řešení jednodruhového PCLSP

Z tabulky 4.5 je vidět, že algoritmus si vede v porovnání s Gurobi velmi dobře, nicméně výrazného rozdílu dosahuje až při úlohách téměř nesmyslné velikosti.

[5]Method=1.

4.2.2 Vícedruhová verze

V předchozím případě jsme si definovali úlohy tak, aby nám i optimální řešení vycházela celočíselná a snadno se programovalo, nyní tomu tak již nebude. Do hry nám navíc vstupuje nový parametr m - počet položek. A na něm také závisí způsob generování úloh. Pro každý výrobek tedy náhodně vybereme

$$a_{it} \in \{1,2,3\},\ h_{it} \in \{1,2,3,4,5\},\ c_{it} \in \{5,6,7,8,9\},$$

konstantní přes všechna období. Označme $\hat{a}$ průměr hodnot a_{it}. Dále jednoduše vygenerujeme poptávky a kapacity s rozdělením:

$$\begin{aligned} \hat{D}_{it} &\sim R(\{20,\ldots,59\}),\\ R_t &\sim R(\{20,\ldots,59\}) \cdot \hat{a} \cdot m + R(\{0,\ldots,10m-1\}),\\ R_1 &= \infty. \end{aligned}$$

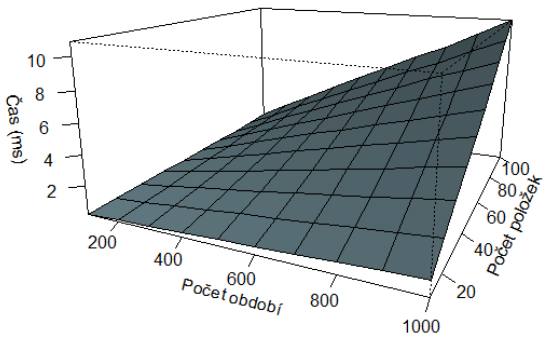
Ukazuje se, že naše úlohy mají opět z velké části strukturu sítě a je nejvýhodnější použít simplexovou metodu s ořezáváním „NetworkCuts“[6]. Provedeme srovnání např. pro $m = 10$, $m = 5$ a plánovací horizont n. Zdrojový kód algoritmu v příloze A.3.

$m = 10$	$n = 100$	$n = 1000$	$n = 10000$	$n = 100000$
Algoritmus	0.00010s	0.00105s	0.01110s	0.11063s
Gurobi	0.009s	0.167s	3.919s	80.229s
$m = 5$				
Algoritmus	0.00006s	0.00057s	0.00597s	0.06121s
Gurobi	0.004s	0.075s	1.830s	34.641s

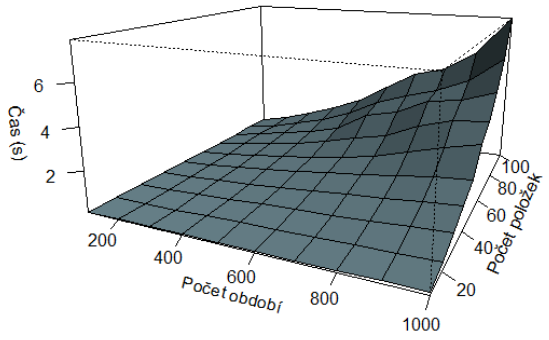
Tabulka 4.6: Průměrné časy řešení vícedruhového PCLSP

Zatímco u algoritmu lze pozorovat téměř lineární závislost na n, doba řešení programem Gurobi roste se zvyšujícím se n mnohonásobně rychleji.

[6]Method=0, PreCrush=1, NetworkCuts=2.



Obrázek 4.1: Graf závislosti doby řešení algoritmem



Obrázek 4.2: Graf závislosti doby řešení pomocí Gurobi

Závěr

V práci jsme rozebrali jeden historicky nejdůležitější a jeden neméně zajímavý lot-sizing problém. Čtenáři určitě pomůže s pochopením a snadnou implementací prezentovaných algoritmů. Text lze brát také jako úvod do problematiky a startovní můstek k dalším mnohem složitějším lot-sizing problémům a pokročilejším algoritmům, které nezapadají do rámce bakalářské práce. Odvodili jsme rychlé algoritmy k nalezení řešení probíraných problémů. V závěru práce jsme skrze numerickou studii prokázali výhodnost představených algoritmů v porovnání v současnosti s jedním z nejrychlejších optimalizačních software, i když jako formální důkaz ji samozřejmě brát nelze. Těžko uvádět nějaká procenta, o kolik je daný algoritmus rychlejší, protože, jak jsme se přesvědčili, velmi záleží na typu a velikosti úlohy. Z obrázků 4.1 a 4.2 lze alespoň usoudit mnohem klidnější chování funkce času řešení algoritmu PCLSP v závislosti na velikosti úlohy než v případě řešení optimalizačním softwarem.

Celkem bylo ve studii vyřešeno přes 300 úloh a od každého druhu je k dispozici ukázkový lp soubor na přiloženém CD.

Reference

[1] Ahuja R.K., Magnanti T.L., Orlin J.B.: *Network flows: theory algorithms and applications*, Prentice Hall, (1993).

[2] Bellman R.: *Dynamic programming*, Princeton University Press, (1957).

[3] Bitran G.R., Yanasse H.H.: *Computational Complexity of the Capacitated Lot Size Problem*, Management Science **28**, 10, (1982), 1174-1186.

[4] Harris F.W.: *How many parts to make at once*, Operations Research **38**, (1990), 947-950.

[5] Haugen K.K., Olstad A., Bakhrankova K., van Eikenhorst E.: *The single (and multi) item profit maximizing capacitated lot-size (PCLSP) problem with fixed prices and no set-up*, Kybernetika **46**, 3 (2010), 415-422.

[6] Haugen K.K., Olstad A., Pettersen B.I.: *The profit maximizing capacitated lot-size (PCLSP) problem*, European Journal of Operational Research **176**, 1, (2007), 165-176.

[7] Wagner H.M., Whitin T.M.: *Dynamic Version of the Economic Lot Size Model*, Management science **5**, 3 (1958), 89-96.

[8] Gurobi Optimizer Version 4.5. Houston, Texas: Gurobi Optimization, Inc., 2011.

A. Přílohy

A.1 Zdrojový kód algoritmu Wagner-Whitinova problému

```
const nmax=1000000;

var s,i,d:array[0..nmax] of integer;
       n:longint;


function poisson_random(lambda:real):longint;

var l,p,u:real;
    k:longint;

begin
l:=exp(-lambda);
k:=0;
p:=1;
 while p > l do
  begin
  u:=random;
  p:=p*u;
  inc(k);
  end;
poisson_random:=k-1;
end;


procedure create_problem;
var j:longint;
begin

for j:=1 to n do
 begin
 s[j]:=60-5*random(5);
 i[j]:=1;
 d[j]:=poisson_random(25);
 end;
end;


function inventory(l,j:longint):longint;
var k:longint;

 begin
 inventory:=0;

 for k:=j to l-1 do
  inventory:=inventory+i[k]*d[l];
 end;
```

```
procedure solve_problem;
var    l,j:longint;
      cost:longint; //naklady
         a:longint; //argument minima
         o:array[0..nmax] of longint; //pomocna promenna - (radek z tab. 2.2)
   argmins:array[0..nmax] of longint; //zapamatovani obdobi ve kterych vyrabime
         x:array[0..nmax] of longint; //optimalni plan


begin
o[0]:=0;
cost:=0;
a:=1;

for l:=1 to n do
 begin
 o[l]:=cost+s[l]+inventory(a,l);
 for j:=a to l-1 do o[j]:=o[j]+inventory(l,j);
 cost:=maxlongint;

 for j:=a to l do
  if cost > o[j] then begin cost:=o[j];
                            a:=j;
                      end;

 argmins[l]:=a;    //pro pozdejsi rychle nalezeni planu
 end;
writeln('solved ',cost);


//zpetne vycteni planu x
l:=n;
a:=argmins[l];
for j:=1 to n do x[j]:=0;
repeat
 for j:=a to l do x[a]:=x[a]+d[j];
 j:=a;
 a:=argmins[a-1];
 l:=j-1;
until a=1;
for j:=1 to l do x[1]:=x[1]+d[j];
end;


 begin

  n:=150000;
  create_problem;
  solve_problem;

 end.
```

A.2 Zdrojový kód algoritmu jednoduchého PCLSP

```
const nmax=2000000;

var c,h,r,d:array[0..nmax] of longint;
          n:longint;

procedure create_problem;
var i:longint;
begin
c[1]:=1;
h[1]:=1;
r[1]:=maxlongint;
d[1]:=34;

for i:=2 to n do
 begin
 c[i]:=1;
 h[i]:=1;
 r[i]:=55-random(30);
 d[i]:=55-random(34);
 end;
end;


procedure solve_problem;
var i,j,obj:longint;
      x,inv:array[0..nmax] of longint;

  function optimal:boolean;
  var i:longint;
  begin
  optimal:=false;
  for i:=n downto 1 do
   if x[i] > r[i] then exit;
  optimal:=true;
  end;

begin
for i:=1 to n do x[i]:=d[i];

j:=1;
if not optimal then
while j<=n do
 begin
 while (x[j] <= r[j]) and (j<=n) do inc(j);
 i:=j;
 repeat
 x[i-1]:=x[i-1]+x[i]-r[i];
 x[i]:=r[i];
 dec(i);
 until x[i]<=r[i]
 end;

inv[0]:=0;
for i:=1 to n do inv[i]:=x[i]+inv[i-1]-d[i];
```

```
obj:=0;
for i:=1 to n do obj:=obj+x[i]+inv[i];
writeln('obj= ',obj);
end;

 begin
  n:=1000000;
  create_problem;
  solve_problem;
 end.
```

A.3 Zdrojový kód algoritmu PCLSP pro více položek

```
const nmax=100000;
const mmax=500;

type rank=record
           r:real;
           i:integer;
          end;
var  n,m:longint;
 c,h,a,d:array[1..mmax,1..nmax] of longint;
       r:array[1..nmax] of longint;
       x:array[0..mmax,0..nmax] of real; //opt. plan
       i:array[0..mmax,0..nmax] of real; //presouvana mnozstvi optimalniho planu
     sum:array[0..nmax] of real; //soucet a_it*x_it
       p:array[1..mmax] of rank; //poradi vyhodnosti presouvani

procedure create_problem;
var p,q,o,j:integer;
          t:longint;
         ap:longint;
begin
ap:=0;
 for j:=1 to m do
  begin
  p:=random(3)+1;
  ap:=ap+p;
  o:=random(5)+1;
  q:=random(5)+5;
  for t:=1 to n do
   begin
   h[j,t]:=o;
   c[j,t]:=q;
   a[j,t]:=p;
   d[j,t]:=20+random(40);
   end;
  end;
for t:=2 to n do r[t]:=(random(40)+20)*ap+random(10*m);
r[1]:=maxlongint;
end;
```

```
function min(a,b:real):real;
begin
if a < b then min:=a else min:=b;
end;


procedure move(j:integer;t:longint);
//presune nejnutnejsi mnozstvi do predchoziho obdobi
var i,k:longint;
mp:real;
begin
if j > m then exit;
i:=p[j].i; //zvolim j-ty nejvyhodnejsi
k:=t-1;
mp:=min(x[i,t],(sum[t]-r[t])/a[i,t]);
x[i,k]:=x[i,k]+mp;  sum[k]:=sum[k]+mp*a[i,k];
x[i,t]:=x[i,t]-mp;  sum[t]:=sum[t]-mp*a[i,t];
if x[i,t]=0 then move(j+1,t);
end;


procedure quicksort(l,r:integer);
var i,j:integer;
    piv:real;
    pom:rank;
begin
i:=l;j:=r;
piv:=p[(l+r) div 2].r;

repeat
 while p[i].r < piv do inc(i);
 while piv < p[j].r do dec(j);
 if i<=j then
  begin
  pom:=p[i];
  p[i]:=p[j];
  p[j]:=pom;
  inc(i);dec(j);
  end;
until i>j;
if j>l then quicksort(l,j);
if i<r then quicksort(i,r);
end;


procedure solve_problem;
var k,j,t:longint;
      obj:real; //opt. hodnota ucelove funkce
begin
for j:=1 to m do
 for t:=1 to n do
  x[j,t]:=d[j,t];

for t:=1 to n do sum[t]:=0;
for j:=1 to m do
 for t:=1 to n do
  sum[t]:=sum[t]+a[j,t]*x[j,t];
```

```
for j:=1 to m do
 begin
 p[j].r:=h[j,t]/a[j,t];
 p[j].i:=j;
 end;

quicksort(1,m); //serazeni produktu podle vyhodnosti jejich presouvani

t:=n;
for t:=n downto 2 do
 if sum[t] > r[t] then move(1,t);

if sum[1] < r[1] then
 begin


  for j:=j to m do i[j,0]:=0;
  for j:=1 to m do
   for t:=1 to n do i[j,t]:=x[j,t]-d[j,t]+i[j,t-1];
  obj:=0;
  for j:=1 to m do
   for t:=1 to n do
    obj:=obj+h[j,t]*i[j,t]+c[j,t]*x[j,t];
  writeln('objective = ',obj);

 end
else writeln('infeasible');

end;

 begin
  n:=100000;
  m:=10;
  create_problem;
  solve_problem;
 end.
```