ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Diplomová práce

Optimalizační algoritmus pro dávkování a rozvrhování

Bc. Pavel Vitvera

vedoucí práce: prof. Dr. Ing. Zdeněk Hanzálek

studijní program: Otevřená informatika, magisterský
obor: Počítačové inženýrství

Praha 2016

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Pavel Vitvera**

Studijní program: Otevřená informatika
Obor: Počítačové inženýrství

Název tématu: **Optimalizační algoritmus pro dávkování a rozvrhování**

Pokyny pro vypracování:

1. Formulujte problém dávkování a rozvrhování výroby několika typů výrobků
2. Seznamte se s ERP systémem SAP a exportujte vhodná typová data
3. Navrhněte a implementujte optimalizační algoritmy pro jeden zdroj a pro více zdrojů
4. Zvolte vhodné metriky a vyhodnoťte algoritmy z hlediska efektivity výroby

Seznam odborné literatury:

[1] Fardin Ahmadizar, Soma Farhadi, Single-machine batch delivery scheduling with job release dates, due windows and earliness, tardiness, holding and delivery costs, Computers & Operations Research, Volume 53, January 2015, Pages 194-205.
[2] L.L. Liu, C.T. Ng, T.C.E. Cheng, Scheduling jobs with release dates on parallel batch processing machines, Discrete Applied Mathematics, Volume 157, Issue 8, 28 April 2009, Pages 1825-1830.

Vedoucí: prof. Zdeněk Hanzálek Dr. Ing.

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 1. 2. 2016

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 27.5.2016 .....

Poděkování

Tímto bych chtěl poděkovat všem, kteří mi pomáhali při vzniku této diplomové práce. Především panu prof. Dr. Ing. Zdeňku Hanzálkovi za čas, který si na mě vyhradil při konzultacích a také za nabyté zkušenosti se sofwarem SAP, s kterým jsem se mohl seznámit.

Abstract

This diploma thesis focuses on production line optimization problems. We propose an algorithm for batching, which computes number of batches and returns suboptimal solution of our batching problem in polynomial time for one and multiple types of products. Batches are then scheduled by MILP on one or multiple sources, where weighted tardiness of batches and setup times are minimized. Furthermore we introduce two methods on how to export data from software for planning and controlling SAP, which is widely used by many companies worldwide. All algorithms were implemented in MATLAB, where they were also tested and evaluated on exported and generated data.

Key words:
Batching, Scheduling, weighted tardiness, SAP, SAP data export

Abstrakt

Tato diplomová práce se zabývá úlohou optimalizace výrobní linky. Je navržen a implementován dávkovací algoritmus, který určí počet dávek a najde suboptimální řešení instance problému v polynomiálním čase pro jeden a více typů výrobků. Poté jsou dávky pomocí MILP rozvrženy na jeden či více zdrojů tak, aby bylo minimalizováno jejich vážené spoždění a přestavbové časy strojů linky. Dále jsou představeny dva způsoby exportu dat z programu pro plánování a kontroling SAP, který je hojně firmami využíván. Veškerá implementace byla provedena v programu MATLAB, kde také byly algoritmy na exportovaných datech a na datech generovaných testovány a vyhodnoceny.

Klíčová slova:
Dávkování, Rozvrhování, vážené spoždění, SAP, SAP data export

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tato diplomová práce se zabývá dávkováním úloh a jejich následným rozvrhováním na jeden či více zdrojů. Samotné dávkování (angl. Batching) je NP-obtížný problém. Je navrženo rychlejší řešení tohoto problému, které najde suboptimální řešení v rychlejším čase. Do některých dávek jsou poté přidány úlohy tak, aby na konci algoritmu bylo požadované množství výrobků na skladu v případě nedostatku zboží nebo naopak úlohy odebrány pokud je potřeba sklad vyprázdnit.

Výsledné dávky jsou poté rozvrhnuty na jeden zdroj a více dedikovaných zdrojů pomocí MILP (mixed integer linear programing), kde je minimalizováno tzv. vážené zpoždění dávek (weighted tardiness) a čas přestaveb výrobní linky. Oba algoritmy jsou testovány na generovaných datech a demo-datech, která byla exportována z programu SAP. Pro export bylo uvažováno několik možností, jak testovací data získat. V této práci jsou představeny 2, které nevyžadují znalost programovacího jazyku ABAP, jež SAP využívá. Jedním je manuální export tabulek a druhý export automatický, kde jsou data získána pomocí definovaných programů a uživatelských událostí.

1.1 Motivace

Výrobní procesy jsou v dnešní době velmi komplexní, trh vyžaduje vysokou variabilitu v produktech a krátké dodací lhůty (ang. due date). To nutí manufakturní firmy k vysoké flexibilitě a tedy schopnosti výroby daného výrobku v relativně krátkém čase a hlavně dodržení dodací lhůty. Rozvrhovací problémy se podle Tapan et al. (2003) v základu dají rozdělit do 4 kategorií:

flowshop: Všechny zakázky projdou stejným procesem v daném pořadí. Na každém stroji se vykoná právě jedna operace

openshop: Zakázky mohou projít stroje v různém pořadí a na strojích se vykoná právě jedna operace

jobshop: Zakázky mohou projít stroje v různém pořadí a navštívit některé stroje vícekrát. Stroje dokáží provádět více než jednu operaci

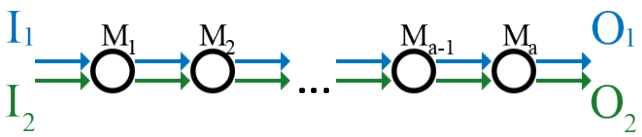
parallel-machine: Firma obsahuje více identických strojů a zakázka může projít kterýmkoliv z nich.

Podle Tapan et al. (2003) také celá čtvrtina firem zabývající se linkovu výrobou je postavena jako flowshop.

Sledování výrobních procesů, které můžou zahrnovat i stovky úkonů, je velice komplexní a složitá záležitost. Je tedy potřeba dodržovat plánovaní a kontroling, na který se hojně využívají specializované softwary. Ty jsou schopny sledovat cestu výroby daného produktu a operace, kterými musí projít (tzv. rountings), rychlost práce a přestavbové časy strojů, materiály potřebné pro výrobu, stav skladu a další informace o výrobních procesech. Zároveň jsou tyto systémy schopny přijímat a automaticky zpracovávat zakázky nebo odesílat objednávky k subdodavatelům.

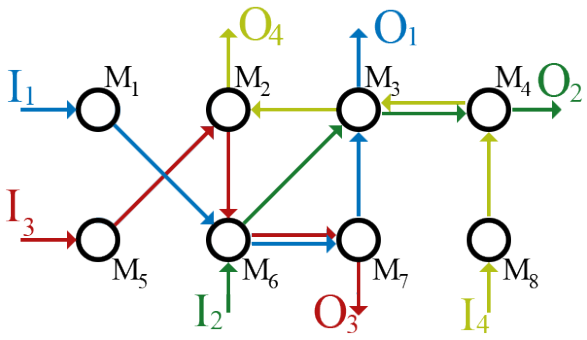
Pracovní postupy a prostředky má každá firma jiné. Proto se tyto systémy nastavují podle požadavků zákazníka. Například některé firmy nemají vlastní sklady (zejména firmy pro automobilový průmysl) a využívají strategii tzv. On-time delivery, při které objednané zboží dorazí v přesně daném časovém intervalu (v řádu hodin), které ihned putuje na výrobní linku. V takovýchto případech sice odpadá nutnost vlastnit sklady, na druhou stranu, pokud se objednávka opozdí, vznikají firmě vysoké ztráty. Více do hloubky na toto téma pojednává [4].

Při samotné výrobě, například právě automobilů, se využívá modelu flowshop, při kterém se polotovar pohybuje od jednoho stanoviště k druhému v přesně daném pořadí, kde se na každém místě vykoná určitá operace a poté putuje dále. Na konci je hotový produkt, který prošel všemi stanovišti ve výrobním procesu. Tento proces je graficky znázorněn na obrázku 1.1. Flowshop model se využívá pro velké dávky a linky produkují jeden výrobek v řádu měsíců. Jejich přestavba je zpravidla velice časově (a tím pádem i finančně) náročná.



Obrázek 1.1: Flowshop proces na a zdrojích

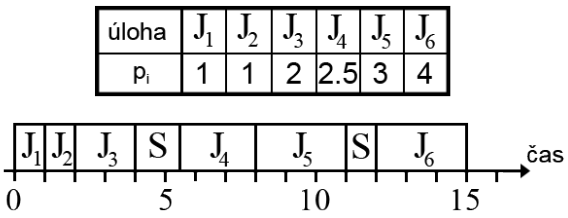
Na druhou stranu pro zakázkovou výrobu malých a středních objednávek s vysokým stupněm variablility výrobků se používá Jobshop model, kde každý druh výrobku potřebuje projít pouze určité kroky výroby. Stroje jsou zpravidla schopny provádět více operací a přestavba linky je časově velice rychlá oproti flowshopu.



Obrázek 1.2: Jobshop proces na osmi ztrojích

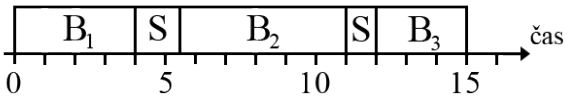
Samotné rozvrhování (angl. Scheduling), vzhledem k NP-obtížnosti problému, je při vysokých počtech zakázek velice časově náročná úloha, kterou lze zjednodušit spojením několika zakázek do jedné. Tento proces spojování zakázek se nazývá dávkování.

Pro jednoduchost uvažujme šest úloh $J_1, \ldots, J_6$, jejichž processing time a vytvořený rozvrh je znázorněn na obrázku 1.3 níže.



Obrázek 1.3: Příklad Rozvrhnutí šesti úloh s přestavbovými časy S

Z příkladu lze vypozorovat, že mezi jednotlivými přestavbovými časy S, lze jednotlivé úlohy spojit do jedné (viz Obrázek 1.4). Poté bychom při rozvrhování pracovali pouze se třemi dávkami, což by při větších instancích tohoto problému vedlo k významné redukci času běhu rozvrhovacího algoritmu.



Obrázek 1.4: Příklad rozvrhnutí dávek

Problém nastává ve chvíli, kdy se snažíme zakázky nadávkovat ještě před samotným rozvrhováním a nejsme schopni s jistotou říci, které objednávky seskupit. Jedním z možných způsobů je jednotlivé zakázky seřadit vzestupně podle release časů a postupně je seskupovat. Tento postup ale není velice optimální pro produkci více typů výrobků a je třeba se s tímto vypořádat efektivnějšími algoritmy.

Kapitola 2

Cíle diplomové práce

Cílem práce je navrhnout a implementovat algoritmy pro optimalizaci výroby, které jsou následně testovány na demo datech pro plánování podnikových zdrojů SAP a na datech generovaných. Algoritmus dávkování má za úkol ze zakázek utvořit dávky. Tento problém je NP-obtížný a je tedy žádoucí najít rychlejší, suboptimální řešení tohoto problému, které je popsáno v kapitole 4.4.1 . Pro druhý algoritmus, algoritmus plánování, je použit MILP, který tyto dávky rozvrhne (viz kapitolu 4.4.3).

Po dohodě se zadavatelem práce jsou tyto dva algoritmy odděleny, aby se popřípadě jednotlivé části daly nahradit jinými, jmenovitě algoritmus rozvrhování, které za použití heuristik dokáže pracovat efektivněji.

Dalším faktorem ovlivňující výsledný rozvrh je přítomnost skladu, kde je dán počáteční a také konečný (požadovaný) stav a je tedy třeba dovyrobit některé typy výrobků nebo se naopak některých zbavit.

Dále jsou vyexportována data z programu SAP pro určitou firmu, která jsou zpracována tak, aby vyhovovala vstupu algoritmu a jsou na něm testována. Veškeré testy probíhají na monoprocesoru, tedy jednom zdroji nebo více dedikovaných zdrojích.

Implementace je provedena v programu MATLAB, kde proběhlo i veškeré testování.

2.1 Související práce

O rozvrhovacích problémech a jejich variantách pojednává [9], které jsou kategorizovány do čtyř základních skupin jmenovaných v kapitole 1.1. Podrobněji je také v této práci na stranách 5 - 8 rozebrán problém flowshopu, který je rozdělen na několik dalších kategorií cyklických a necyklických úloh, které mohou být dále rozděleny. Práce [10] se zabývá resource constrained scheduling problémem (RCPSP), kde jsou minimalizovány časové prodlevy mezi úlohami. Problém je formulován jako ILP

a úlohy jsou rozvrhovány na jeden a více zdrojů s přestavbovými časy.

Dávkováním na jednom zdroji se zabývá práce [3], která v kapitole 2 ukazuje polynomiální řešení nadávkování již rozvrhnutých úloh do k dávek. Tento problém je řešitelný v $\mathcal{O}(n)$. Dále je zde dokázána NP-obtížnost pro 3 druhy dávkovacích problémů. Jedním z nich je problém minimalizace váženého zpoždění úloh $\cdot|\cdot|\sum w_i f_i$, na kterém je ukázána NP-obtížnost redukcí z 3-PARTITION problému a o kterém pojednává i tato práce.

Při dávkování je třeba najít způsob, jakým jednotlivé zakázky do dávek spojit a zároveň se pokusit dodržet jejich release date a due date. Tímto problémem se zabývá [2], kde jsou zvažovány různé strategie pro dávkování. Některé uvažují stejný processing time u všech objednávek, jiné zas omezení počtu objenávek v jedné dávce. Jeden z algoritmů například sjednocuje objednávky od největších po nejmenší s maximální velikostí dávky b zakázek. Tento postup se nezdá být špatný, nicméně je hůře aplikovatelný pro dávkování více druhů výrobků. Také se zde uvažuje zpracování zakázek v dávce paralelně, tedy prosessing time dávky B je $max(p_i) \quad i \in B$. Tento způsob samozřejmě také lze v některých případech aplikovat, nicméně v této práci jsou všechny zakázky zpracovávány sériově, tedy pro dávku B je celkový processing time $\sum(p_i) \quad i \in B$.

Dávkováním jednoho a více druhů produktů se zabývá práce [5], kde jsou uvažovány i setup časy mezi jednotlivými dávkami.

Kapitola 3

Definice problému

Dávkování a rozvrhování úloh jsou jedny z klíčových problémů pro firmy zabývající se linkovou výrobou. Při správném dávkování je firma schopna zvýšit své zisky a zároveň snížit náklady na výrobu, ať už jde o samotný materiál, minimální počet přestaveb strojů nebo dalších faktorů ovlivňujících plynulost chodu firmy. Také pokud jsou úlohy nadávkovány, vytvoření samotného rozvrhu není časově tolik náročné, díky snížení počtu úloh.

3.1 Terminologie

úloha je definována parametry release time, due date, processing time, váha, typ, kapacita a identifikátor do jaké zakázky úloha patří (podrobné vysvětlení v kapitole 4.1).

zakázka 1 či více úloh, kde každá obsahuje požadavek na jeden typ výrobku. Jednotlivé úlohy mají stejný release time, due date, váhu a číslo zakázky.

zdroj monoprocesor schopný zpracovávat maximálně jednu úlohu v daném časovém okamžiku.

dedikovaný zdroj zdroj schopný vykonávat pouze jeden typ operace.

dávkování proces jehož cílem je seskupit úlohy a tím redukovat celkový počet úloh.

dávka je definována stejnými parametry jako úloha a obsahuje 1 či více úloh. Přesná definice dávky je popsána v kapitole 4.3

3.2 Požadavky

Dávkovací problémy jsou NP-obtížné a tedy hlavním požadavkem je navrhnout a implementovat algortimus, který nalezne řešení v rychlejším čase i za cenu toho, že řešení nebude vždy optimální. Dále je třeba splnit požadavky skladu na konci běhu algoritmu, tedy na sklad produkty vyrobit nebo je z něj odebrat. Z výsledných dávek poté bude vytvořen rozvrh, který bude minimalizovat vážené zpoždění dávek a čas přestaveb strojů linky. Zároveň lze úlohy rozvrhnout na jeden zdroj nebo více dedikovaných zdrojů typu flowshop. Tzn. všechny úlohy procházejí stejným procesem a na každém zdroji je provedena určitá operace.

Dále jsou prozkoumány možnosti exportu dat z programu SAP, na kterých je algoritmus testován.

3.3 Cíle

Cílem první části je navrhnout algoritmus dávkování, který bude splňovat požadavky na rychlost a zároveň bude vykazovat dobré výsledky. Tzn. výsledné vážené zpoždění zakázek bude minimální. Dále je třeba navrhnout rozvrhovací algoritmus na vážené zpoždění dávek, který bude úlohy rozvrhovat na jeden zdroj nebo více dedikovaných zdrojů, jejichž počet je určen uživatelem.

V druhé části je navrhnuta strategie exportu dat z programu SAP, které budou použity jako vstup pro testování efektivity algoritmu.

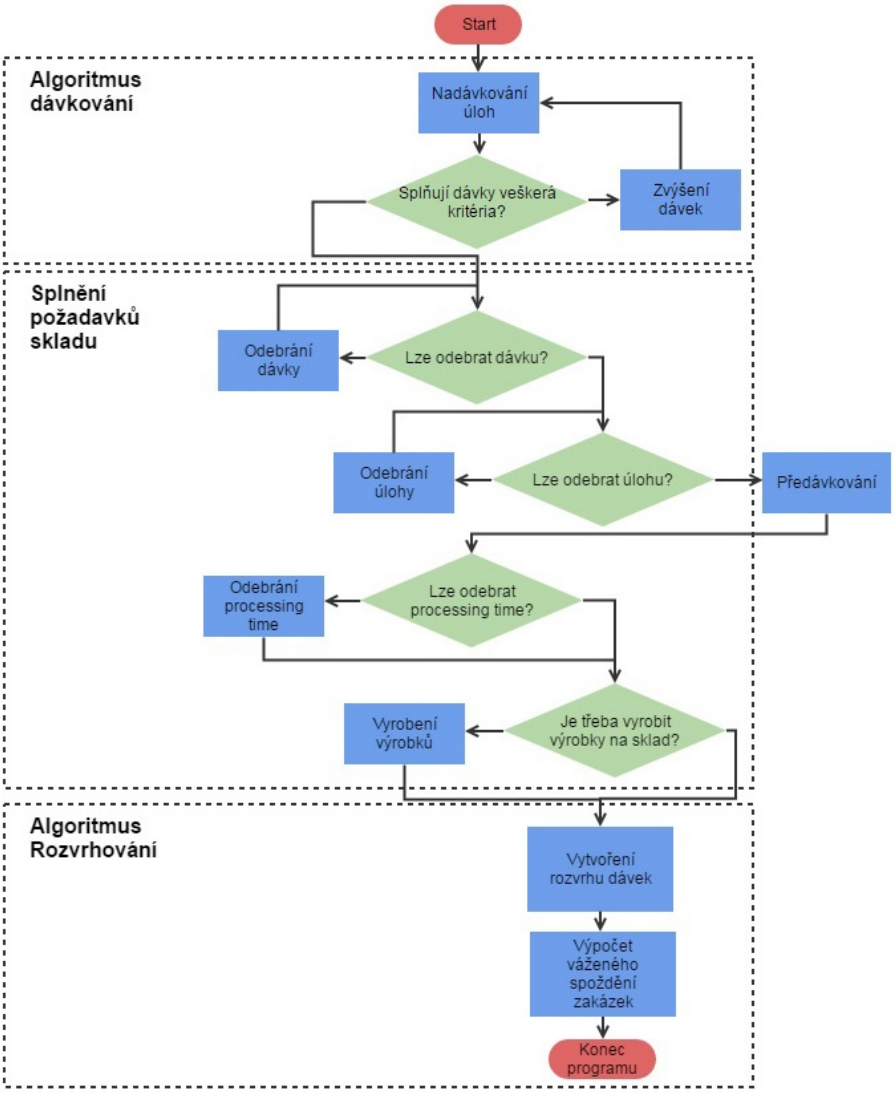
3.4 Navrhovaná řešení

Jedním z navrhovaných řešení bylo vyjádření dávkování a rozvrhování jako jeden MILP, které se nakonec ukázalo jako velice pomalé a tudíž neefektivní řešení.

V druhém návrhu byl algoritmus rozdělen na tři části (viz obr 3.1).

Nejprve jsou úlohy nadávkovány tak, aby splnily veškerá kritéria dávkovacího algoritmu (viz kapitola 4.4.1). Poté jsou splněny požadavky skladu. Z obrázku je vidět, že po odebrání dávek nebo zakázek je provedeno předávkování, jelikož je možné nalézt lepší řešení. Nakonec jsou výsledné dávky rozvrhnuty na jeden nebo více dedikovaných zdrojů typu flowshop.

Hodnota kriteriální funkce rozvrhování není konečná, neboť zpoždění dávky nemusí nutně znamenat zpoždění všech úloh, které dávka obsahuje. To plyne z definice dávky, která je popsána v kapitole 4.3. Výsledné vážené zpoždění zakázek bude tedy menší nebo rovno hodnotě kriteriální funkce.



Obrázek 3.1: Části algoritmu

Algoritmy byly testovány na generovaných zakázkách, kde každá obsahovala pouze jednu úlohu a také na zakázkách exportovaných ze SAPu, kde jedna zakázka mohla obsahovat úloh více. Pro export dat byly použity 2 způsoby (viz 6.2) a byla získána data o objednávkách, bills of material a stavu skladu.

Zároveň před spuštěním samotného algoritmu jsou všechny vstupní parametry zkontrolovány. Pokud jsou některé špatně zadány, program vypíše varování a chybu opraví nebo vypíše error a program skončí. Všechny varovné a chybové hlášky jsou vysvětleny v kapitole 5.2.

Kapitola 4

Implementace

Tato kapitola je věnována podrobnému popisu funkčnosti všech částí programu, které byly v rámci této práce implementovány. Implementace algoritmů byla provedena v programu MATLAB, za využití TORSCHE scheduling toolboxu, vyvíjeného na ČVUT v Praze. TORSCHE toolbox je použit pro výpočet MILP v rozvrhovací části.

4.1 Definice úlohy a zakázky

Na vstupu mějme n úloh a m typů výrobků. Každá úloha $J_i, i \in 1, ..., n$ obsahuje právě jeden typ výrobku a je definovaná jako vektor sedmi hodnot v tomto pořadí:

- release date $r_i, \quad r_i \geq 0$
- due date $d_i, \quad d_i > r_i$
- processing time $p_i, \quad p_i > 0$
- typ výrobku $f_i, \quad f_i = 1, \ldots, m$
- váha $w_i, \quad w_i > 0$

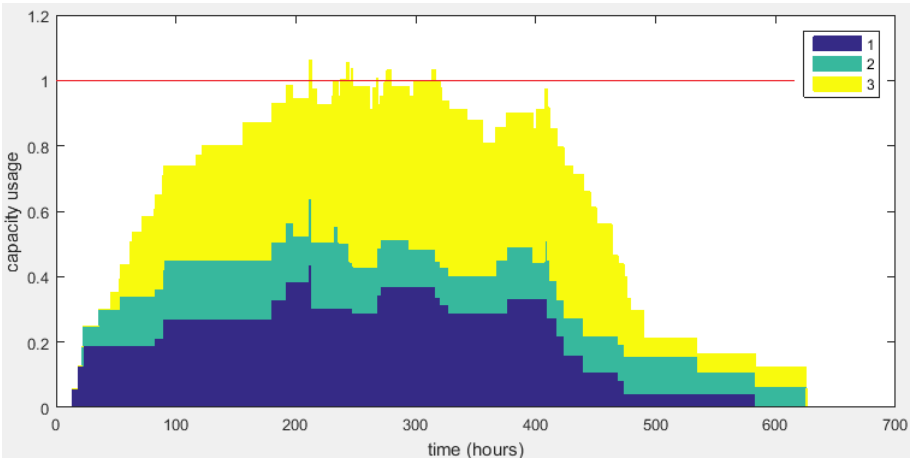
Z těchto hodnot lze dopočítat šestou hodnotu - kapacitu využití stroje pro každou úlohu mezi jejím release timem a due datem:

$$c_i = p_i/(d_i - r_i) \tag{4.1}$$

Kapacita nám říká, jak velkou část z intervalu $< p_i, d_i >$ úloha potřebuje na její zpracování. Na obrázku 4.1 je tento poměr znázorněn pro 2 úlohy s kapacitami 0.5 a 0.25 žlutě a jejich náležité processing times červeně.

Obrázek 4.1: Kapacita (žlutě) oproti processing time (červeně)

Na obrázku 4.2 jsou takto vykresleny kapacity 40 úloh, kde x-ová osa znázorňuje čas a y-ová právě kapacitu využití stroje. Barvy v grafu značí typ výrobku, které úlohy obsahují. V tomto případě jsou 3 typy výrobků, tedy $m = 3$. Také lze pozorovat, že úloh typu 3 je znatelně více než úloh ostatních typů. Jednotlivé kapacity jsou vykresleny nad sebe a tedy celková kapacita využití stroje je součet kapacit úloh v daném čase. Lze tedy pozorovat, že v intervalu 200 až 300 kapacita v některých místech přesahuje hodnotu jedna a tedy v daném okamžiku stroj nestihne úlohy zpracovat.



Obrázek 4.2: Kapacity úloh v čase

To ale neznamená, že zpoždění některých úloh je jisté. Jak bylo ukázáno na obrázku 4.1, některé úlohy mohou již být zpracovány a tedy celková reálná kapacita v daném čase bude stejná nebo nižší než je znázorněna v grafu.

Sedmý parametr určuje číslo zakázky, do které úloha patří. Generované úlohy obsahují právě jeden typ výrobku a tedy pouze jednu úlohu. Celkový počet zakázek je tedy roven počtu úloh. Oproti tomu exportované zakázky ze SAPu mohou obsahovat v jedné zakázce požadavek až na m druhů výrobků. Potom je zakázka rozdělena do m úloh, každá s jedním typem výrobku. Takovéto úlohy mají vždy stejný release time, due date, váhu (která je rovna váze celé zakázky) a číslo zakázky.

4.2 Vstupní parametry algoritmu

Všechny vstupní parametry jsou definovány v souboru `Main.m` a lze je rozdělit do několika skupin:

- Parametry pro generování zakázek
- Parametry rozvrhování
- Parametry skladu

Pokud se boolean proměnné *loadSAPdata* nebo *loadGendata* rovnají jedné, nahrají se data z `tasksm.mat`, což je matice napoledy vygenerovaných dat resp. z `tasksSAP.mat`, což jsou data ze SAPu.

Pokud jsou obě tyto hodnoty nulové, zakázky se vygenerují podle následujících parametrů:

- n - počet generovaných zakázek
- m - počet typů produktů
- *period* - doba maximálního release timu úlohy
- *ddmin*/*ddmax* - minimální/maximální časové okno pro 1 zakázku
- *procmean* - průměrný processing time jedné zakázky
- *procdev* - maximální odchylka od procmean
- *rndweight* - maximální možná váha zakázky
- $famratios$ - vektor délky m určující poměry počtů zakázek pro daný typ produktu

Parametry rozvrhování jsou následující:

- *setupweight* - váha penalizace za jednotku času přestavby stroje
- $familysetups$ - matice $m \times m$ určující dobu přestavby stroje z výroby produktu i na produkt j

- q - počet dedikovaných zdrojů, kterými každá zakázka projde
- *srctimes* - vektor délky $m \times q$, kde každý řádek určuje poměr časů strávených na zdroji i, $\quad i = 1, \ldots, q$ z celkového processing time dávky

Posledními parametry jsou parametry skladu:

- *initialstock* - vektor délky m určující stav skladu před během algoritmu v jednotkách processing time
- *endstock* - vektor délky m určující požadovaný stav skladu po běhu algoritmu v jednotkách processing time

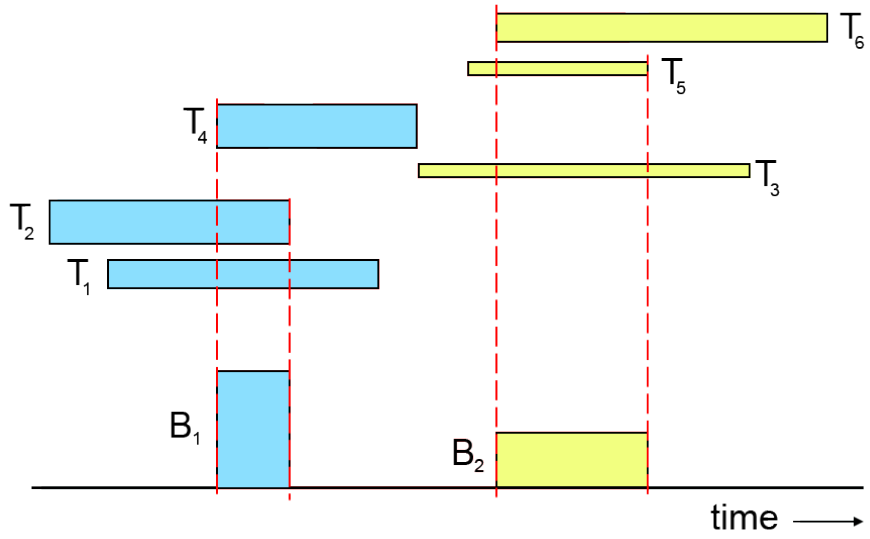
Všechny vstupní parametry jsou zkontrolovány funkcí `TestInput.m`, kde pokud jsou parametry nastaveny špatně, jsou upraveny a je zobrazena varovná hláška nebo je vypsán error a program skončí.

4.3 Dávkování úloh

Při dávkování je třeba myslet na splnění due dates zakázek a také je potřeba začít výrobu až po dodání všech potřebných materiálů pro výrobu. Také lze dávkovat pouze úlohy stejného typu, jelikož mají stejný výrobní proces. Mějme tedy dávku B, která je definována prvními šesti parametry jako úloha a obsahuje úlohy $J_i \in B$. Potom parametry dávky B určíme takto:

- Release time r_B dávky B je dáno jako $max(r_i)$
- Due date d_B dávky B je dáno jako $min(d_i)$
- Processing time p_B dávky B je dáno jako $\sum(p_i)$
- Typ f_B dávky B je roven typu f_i (všechny úlohy v dávce mají stejný typ)
- Váha w_B dávky B je dána jako $\sum(w_i)$

Kapacita dávky se vypočítá stejným způsobem podle vzorce 4.1. Z obrázku 4.3 lze vidět aplikované vlastnosti uvedené výše. Zároveň lze sledovat, jak se navýší kapacita dávky při změně release timu a due datu. Dávka po přidání úlohy totiž bude vždy mít vyšší kapacitu, jelikož interval dávky $< r_B, d_B >$ bude menší nebo stejný a celkový processing time dávky se navýší o processing time nově přidané úlohy. Due date d_B je definován tak, že při splnění d_B je zaručeno splnění due dates všech úloh, které dávka obsahuje.



Obrázek 4.3: Proces dávkování 6 úloh do 2 dávek

4.4 Algoritmy

V této části je podrobně vysvětlena funkcionalita implementovaných algoritmů. Pseudokódy pro tyto algoritmy jsou v kapitole 5.1.

4.4.1 Algoritmus dávkování

Při navrhování algoritmu bylo uvažováno několik kritérií, která by měla být splněna. Nejzásadnější bylo, aby všechny dávky byly splnitelné v daném čase, tedy aby dávka B splňovala $p_B < (d_B - r_B)$ nebo $c_B < 1$. Dalším kritériem také byl počet dávek, který se snažíme minimalizovat.

V prvním kroku algoritmu vytvoříme dávky tak, že seřadíme pro každý typ výrobku úlohy i podle release timu r_i a postupně je sjednocujeme do dávky B dokud je dodrženo:

$$r_i < d_B \tag{4.2}$$

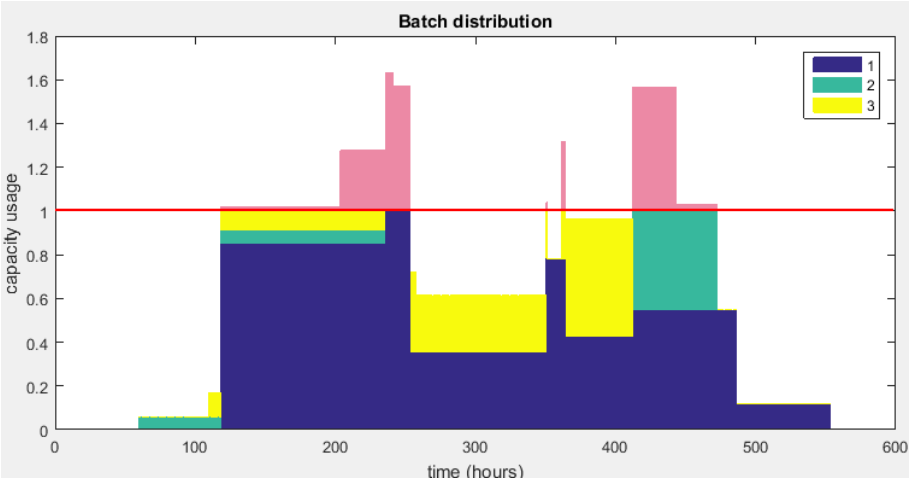
Tedy úlohu i přidáme do dávky B pokud release time úlohy je menší než due date dávky.

Pokud je toto pravidlo porušeno, je vytvořena nová dávka, do které je úloha i přidána. Následující úlohy jsou přidávány k této nově vytvořené dávce dokud opět není porušeno pravidlo 4.2. Tímto dosáhneme počátečního nadávkování (=počáteční konfigurace). Tu uložme do struk-

tury *batchset*, která uchovává všechny dávky a také úlohy, které každá dávka obsahuje.

Počáteční nadávkování je rychlé a také v drtivé většině případů neoptimální, nicméně tímto způsobem získáme minimální počet dávek pro každý typ výrobku. Jelikož v prvním kroku pouze seřadíme pole, poté ho lineárně projdeme a utvoříme dávky, je náročnost prvního kroku $\mathcal{O}(n \cdot log(n))$. Počáteční konfigurace dávek je znázorněna na obrázku 4.6 vlevo nahoře, která obsahuje 2 dávky každého typu. Z obrázku je také vidět, že kapacity u některých dávek mnohonásobně přesahují hodnotu 1 a tedy nejsou proveditelné mezi jejich release timem a due datem. Takovéto dávky jsou nepřípustné, jelikož zaručují že due date dávky d_B nebude splněn a je třeba tyto dávky penalizovat.

Zaveďme tedy proměnnou *overload*, která udává penalizaci dané konfigurace jako plochu kapacit S_o překračující hodnotu 1. Graficky je tato hodnota zobrazena na následujícím obrázku, kde obsah růžové plochy je právě roven hodnotě *overload*.



Obrázek 4.4: Výpočet hodnoty overload

Samotná tato podmínka ale pro výpočet *overload* nestačí, jelikož hodnotu lze snížit, pokud jsou úlohy nadávkovány tak, že interval $< r_B, d_B >$ dávky B je velice malý. Tento problém nastává kvůli výpočtu obsahu numerickým integrováním v diskrétních časových intervalech. Při dostatečně malém časovém úseku mezi r_B a d_B je tedy možné dosáhnout vysoké kapacity, která nebude započítána do celkové hodnoty *overload*.

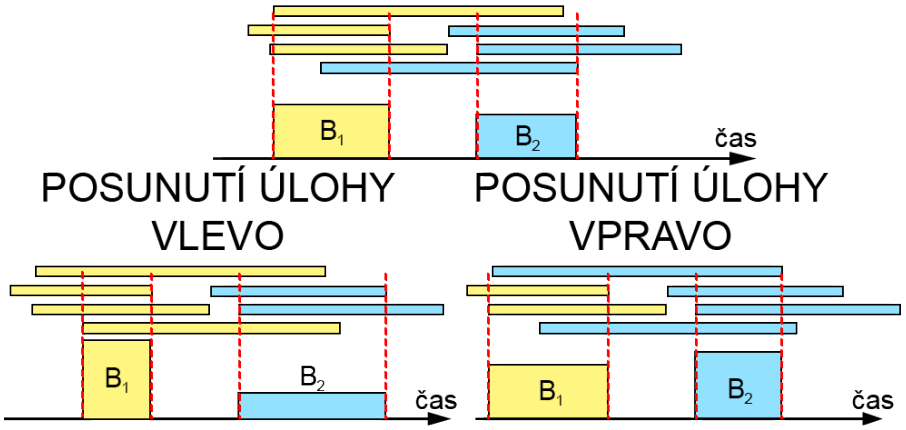
Je tedy třeba penalizovat samotné dávky, které mají vysokou kapacitu a to vyšší než hodnota *caplim*, což je interní proměnná nastavená na hodnotu 0.95. Celkový výpočet *overloadu* bude tedy následující:

$$overload = S_o + \sum_{1}^{n}(c_i \cdot hpen) \text{ pro } c_i > caplim \tag{4.3}$$

Konstanta *hpen* udává penalizaci za překročení hodnoty *caplim* a je nastavena defaultně na hodnotu 30.

S počáteční konfigurací dávek a její penalizací se dostáváme k druhé fázi algoritmu, kde si dávky "vyměňují"úlohy následujícím způsobem. Máme j dávek, které jsou seřazeny vzestupně podle release time r_j. Z dávky $B_a, a = 1, \ldots, j$ je vybrána úloha s nejnižším $d_i \in B_a$, tedy taková úloha, která definuje due date dávky. Ta je následně, pokud je to možné, vložena do dávky B_{a-1}, která leží v intervalech nižších než tato dávka. Tento pohyb nazvěme posunutí úlohy vlevo. Stejným způsobem je z dávky B_a vybrána úloha s nejvyšším r_i, tedy úloha definující release time dávky B_a a ta je, pokud je to možné, přidána do dávky B_{a+1}. Tento pohyb nazvěme posunutí úlohy vpravo.

Úlohu lze posunout do dávky pokud průnik dávky $< r_a, d_a >$ a úlohy $< r_i, d_i >$ není prázdný. Neboli dávka a úloha sdílí alespoň z části časový interval ve kterém leží. Následující obrázek ukazuje oba výše popsané postupy:



Obrázek 4.5: Posouvání zakázek vlevo/vpravo

Z obrázku lze sledovat, jak se při posunutí úlohy vlevo z dávky B_2 do dávky B_1 změní release time dávky B_1 a také kapacita, která vzroste kvůli přidání úlohy do dávky a také kvůli zvýšení release timu. Tyto samé vlastnosti lze také pozorovat při posunutí úlohy vpravo z dávky B_1 do dávky B_2.

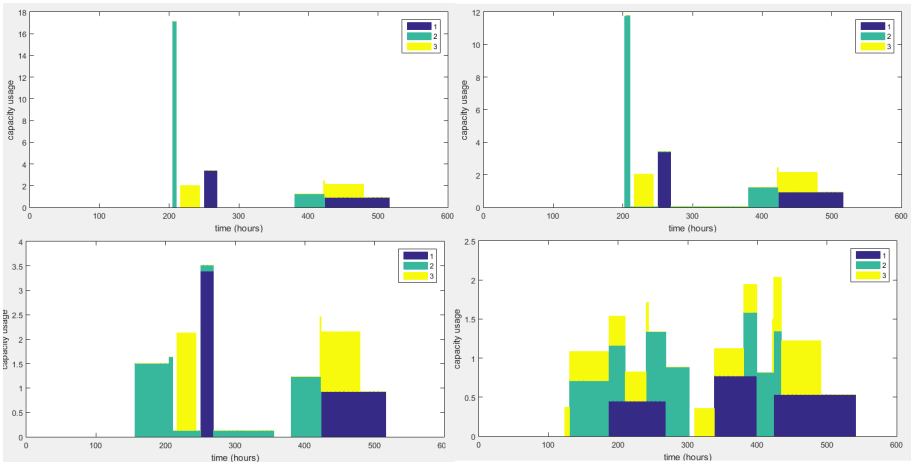
Takto posuneme ze stávající konfigurace z každé dávky úlohy vlevo i vpravo a spočítáme *overload*. Pokud dojde ke zlepšení alespoň o δ, hodnota *overload* a také konfigurace dávek jsou uloženy do pole. Defaultní

hodnota $\delta = 10$. Po všech možných posunech ze stávající konfigurace je vybráno z uložených hodnot 5 nejlepších konfigurací. Ty jsou uchovány do dalších iterací a zbytek je smazán. Všechny konfigurace dávek jsou v poli řazeny podle *overload* vzestupně a pro další posouvání zakázek je vždy vybrána konfigurace s nejmenším *overload*.

Pokud již není možné *overload* zlepšit, je vrácena konfigurace dávek s nejnižší takovouto hodnotou. Pokud v takovéto konfiguraci existuje dávka s kapacitou $c_i > 1$, je počet možných dávek pro tento typ zvýšen o 1 a je tedy možnost lepšího rozdistribuování zakázek do dávek a je opakován druhý krok algoritmu od dosavadní nejlepší nalezené konfigurace. Prázdná dávka je vložena vedle dávky s nejvyšší kapacitou a tedy v další iteraci algoritmu se do této dávky vloží zakázka, jelikož v takovémto případě dojde k nejlepšímu zlepšení *overload*.

Nejhorší případ je, kdy máme n úloh se stejnými r, d, f a s kapacitou $c = 1$. Potom prvotní konfigurace bude pouze 1 dávka. V dalších iteracích se bude počet dávek zvyšovat až do hodnoty n. V každé konfiguraci si d dávek vymění $(d-1) \cdot 2$ zakázek doleva a doprava. Takto výměny proběhnout n krát. Tedy celkový počet výměn bude $n \cdot (d-1) \cdot 2$, kde počet dávek d se zvětšuje od jedné do n. Časová náročnost druhého kroku a tedy i celého algoritmu je tedy $\mathcal{O}(n^2)$.

Jak se zlepšuje overload v různých iteracích algoritmu lze pozorovat na obrázku 4.6, kde v 1. iteraci (levý horní obrázek) se *overload* = 1021 a v poslední (pravý spodní obázek) *overload* = 76.49:



Obrázek 4.6: Konfigurace dávek v různých iteracích

Z poslední konfigurace (vpravo dole) je vidět, že všechny dávky mají kapacitu menší než 1. Což je, jak již bylo zmíněno, podmínkou, aby byla dávka splnitelná v jejím časovém intervalu. Samozřejmě v některých časech součty kapacit dávek mohou přesáhnout hodnotu 1 a tím stoupá

overload. Je tedy snaha najít takovou konfiguraci, kde se takováto situace vyskytuje minimálně.

4.4.2 Využití skladu

Po nadávkování úloh je třeba odebrat/přidat ze skladu požadované množství výrobků, aby konečné hodnoty odpovídaly hodnotám zadaným uživatelem. To probíhá v těchto čtyřech fázích:

1. Odebrání celé dávky, pokud je dostatečné množství na skladu.
2. Odebrání úlohy, pokud je dostatečné množství na skladu.
3. Odebrat processing time z dávky, pokud jsou nějaké produkty na skladu.
4. Přidání processing time k dávkám, aby bylo vyrobeno požadované množství na sklad.

Při odebírání dávek jsou detekovány všechny možné dávky, které lze odebrat. Poté je postupně každá dávka z konfigurace odebrána a je spočítán *overload* konfigurace bez této dávky. Nakonec se vybere ta konfigurace, která má nejnižší *overload* a odečte se ze skladu processing time odebrané dávky. Poté se algoritmus pokusí znovu odebrat dávku, dokud je dostatečné množství výrobků na skladu.

Nutno upozornit, že když hovoříme o odebrání processing time ze skladu, tak odebíráme processing time stejného typu jako je dávka, jelikož sklad obsahuje processing time pro každý typ výrobku.

Pokud další dávky nelze odebrat, algoritmus se pokusí odebrat úlohy z dávek a to ty, které definují release time r nebo due date d dávek, jelikož odebráním takovýchto úloh opět dosáhneme nejvyššího zlepšení *overload*.

Pokud je odebrána pouze jedna nebo více úloh z dávky, je třeba upravit parametry dávky, jelikož mohlo dojít k odebrání úlohy s maximálním r_i, nebo minimálním d_i. Mimo tyto 2 hodnoty se při odebrání úloh mění hodnoty p_i, w_i a tedy i kapacita dávky c_i. Po provedení prvních 2 fází se dávky předávkují, jelikož odebráním dávek nebo úloh z dávek lze pravděpodobně nalézt lepší konfiguraci než doposud. Ve třetí fázi je odebrán processing time z dávky, která nejvíce zlepší hodnotu *overload*.

Jak již bylo zmíněno, ve fázích 1 až 3 odebíráme vždy dávky, úlohy nebo processing time takový, který nejvíce vylepší *overload*. Naopak ve 4. fázi přidáme processing time k dávce, která *overload* navýší nejméně. To ale nemusí nutně znamenat přidání celého processing timu k jedné dávce, proto je tento čas rozdělen na několik menších, které jsou postupně rozdistribuovány k dávkám příslušného typu. Nutno dodat, že

úlohy pro výrobu na sklad, lze pouze přidávat k již existujícím dávkám a nelze vytvářet nové. Proto některá kapacita dávky v této fázi algoritmu může přesáhnout hodnotu 1 a tím pádem nebude stihnut due date dávky. Tento přístup byl zvolen, aby se předešlo novým dávkám, které by obsahovaly pouze úlohy pro výrobu na sklad a tím pádem by jejich parametry byly voleny velmi benevolentně, jako např. velmi vysoký rozsah intervalu $< r, d >$ či nulová váha. Takový přístup je nežádoucí.

Poté co jsou uspokojeny požadavky uživatele na stav skladu, jsou dávky rozvrhnuty.

4.4.3 Algoritmus rozvrhování

Rozvrhování s minimalizací váženého zpoždění (scheduling of minimizing weighted tardiness) je NP-obtížný problém řešený jako mixed integer linear program (MILP). Uvažujme tedy nepreemptivní výrobní linky typu flowshop, tedy všechny výrobky stejného typu projdou všemi zdroji v přesně daném pořadí. Zároveň uvažujeme přestavbové časy linky. Vstupem algoritmu je n dávek spočítaných dávkovacím algoritmem, ke kterým jsou přidány 2 "dummy"úlohy s $p = 0$, kde první bude předcházet všechny ostatní úlohy a druhá všechny následovat. Dále matice časů přestaveb linky $fs \in \mathbb{R}^{m \times m}$, *setupweight*, počet zdrojů q a matice *srctimes*, z které jsou získány hodnoty p_i^a udávající processing time dávky i na zdroji a (všechny tyto parametry jsou popsány v kapitole 4.2).

Mějme matici následností $x \in \mathbb{R}^{(n+2)\times(n+2)}$, kde $x_{i,j} = 1$ říká, že dávka i je zpracována před dávkou j. Vektor $C_i^j, \quad i = 1, \ldots, n+2 \quad j = 1, \ldots, q$ určuje čas, kdy je dávka i dokončena na zdroji j. T_i udává zpoždění dávky, matice $D \in \mathbb{R}^{(n+2)\times(n+2)}$ určuje, zda je dávka i jiného typu než dávka j, tedy $D(i, j) = 1$ pokud $f_i \neq f_j$. Jinak obsahuje matice nuly. Hodnota S pak reprezentujme čas přestaveb linky.

Vektor proměnných c vypadá tedy následovně:

$$c = \left[x_{i,j}, C_1^1, \ldots, C_{n+2}^1, \ldots, C_{n+2}^q, T_i, S\right]$$

Potom jsme schopni popsat problém vzorcem 4.4. První dvě podmínky zaručí, že každá dávka je rozvrhnuta právě jednou. Třetí a čtvrtá podmínka zaručí, že dummy úlohy jsou vykonány jako první a poslední. Pátá podmínka je platná pouze pokud $x_{i,j} = 1$. Potom čas dokončení úlohy j na zdroji a je větší než součet časů dokončení úlohy i na zdroji a, jejího processing time p_i^a a čas přestavby linky z výroby produktu typu i na typ j. Šestá podmínka určí zpoždění dávky T_i z času dokončení dávky na posledním zdroji C_i^q. Sedmá podmínka zaručí, že dávka se začne vykonávat až po jejím release time a osmá zaručí, že dávka na zdroji a je

dokončena dříve než je zpracovávána na zdorji $a+1$. Poslední podmínka určí celkový čas přestavby strojů.

$$
\begin{aligned}
\min \quad & \sum_{2}^{n+1}(w_i \cdot T_i) + S \cdot setupweight \\
\text{s.t.} \quad & \sum_{i=1}^{n+2}(x_{i,j}) = 1,\ j = 2, \ldots, n+2 \\
& \sum_{j=1}^{n+2}(x_{i,j}) = 1,\ i = 1, \ldots, n+1 \\
& \sum_{i=1}^{n+2}(x_{i,j}) = 0,\ j = 1 \\
& \sum_{j=1}^{n+2}(x_{i,j}) = 0,\ i = n+2 \\
& C_j^a + M(1 - x_{i,j}) \geq C_i^a + p_j^a + fs_{i,j}, \forall a, i \neq j \\
& T_i \geq C_i^q - d_i,\ i = 1 \ldots n+2 \\
& C_i^1 \geq r_i + p_i^1,\ i = 1 \ldots n+2 \\
& C_i^{a+1} \geq C_i^a + p_i^a \ \ a = 1, \ldots, q-1 \\
& S = \sum_{i,j=1}^{i,j=n+2}(x_{i,j}) \cdot D,\ i = 1 \ldots n+2 \\
\text{where} \quad & x_{i,j} \in \{0, 1\}, \\
& C_i^j, T_i, S \in \mathbb{R}
\end{aligned}
\tag{4.4}
$$

Výstupem takového algoritmu je hodnota váženého zpoždění dávek. Abychom získali vážené zpoždění zakázek, je třeba spočítat penalizaci pro jednotlivé zakázky. Jak bylo popsáno v dávkovacím algoritmu, konfigurace dávek je uložena v proměnné *batchset*. Jsme tedy schopni zjistit, do jaké dávky úloha patří a také které úlohy tvoří zakázku. Potom výsledná hodnota zpoždění $penalty_j$ zakázky j se tedy spočítá následovně:

$$penalty_j = max(max(C_B - d_i, 0), penalty_j) \cdot w_i \ \ i \in B, B = 1, \ldots, n$$

kde i jsou úlohy obsažené v dávce B, j je číslo zakázky a vektor *penalty* je inicializován na 0. Pokud zakázka obsahuje více úloh, výsledná penalizace je maximální hodnota z úloh náležící zakázce.

4.5 Výstup algoritmu

Výstupem algoritmu je graf konfigurace nejlepšího nadávkování a také výsledný rozvrh dávek. Dále také jsou vypsány následující parametry:

- *overload* původního nadávkování úloh
- *overload* konečného nadávkování úloh
- Původní počet dávek pro typ výrobku
- Konečný počet dávek pro typ výrobku
- Orientační vytížení linky
- Doba běhu dávkovacího algoritmu
- Hodnota kriteriální funkce rozvrhovacího algoritmu
- Doba běhu rozvrhovacího algoritmu
- Dávky které byly utvořeny
- Časy spoždění dávek

Výstup v MATLABU je znázorněn na obrázku 4.7, kde jsou vypsány hodnoty ve stejném pořadí jako výše uvedené hodnoty. Zároveň se v průběhu běhu algoritmu zobrazují výpisy pro lepší přehled ve kterém stavu se nachází (pro jednotlivé stavy viz 3.1).

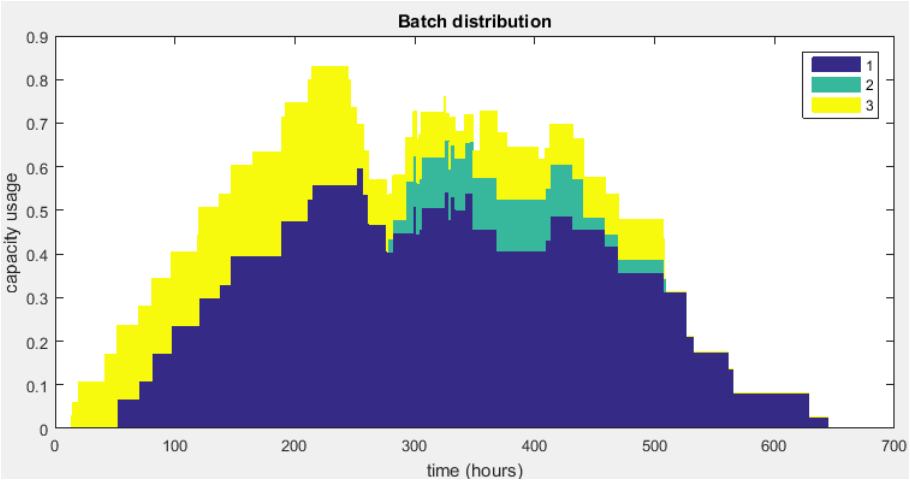
```
Batching...
Satisfying stock limits...
Rebatching...
Scheduling...
----------------------------------------
Orig overload  : 794.73
Min overload   : 31.505
Initial batches:     2     1     2

Final batches  :     3     1     2

Line utilization : 0.47
Batch weighted tardiness : 1043.00
Order weighted tardiness : 560.00
Batching time : 52.22
Scheduling time : 0.27
Batches:
  146.0000  257.0000   76.0000   33.0000    1.0000    0.6847
  298.0000  348.0000   46.0000   20.0000    1.0000    0.9200
  412.0000  507.0000   80.0000   29.0000    1.0000    0.8421
  292.0000  440.0000   20.0000    7.0000    2.0000    0.1351
  164.0000  244.0000   53.0000   30.0000    3.0000    0.6625
  354.0000  376.0000   20.0000    5.0000    3.0000    0.9091

Batches Ti:     0     0     0     0    34     0
```

Obrázek 4.7: Příklad výsledného rozvrhu dávek

Lze tedy pozorovat navýšení dávek pro 1. typ výrobků ze 2 na 3 dávky. Dále z algoritmu rozvrhování je hodnota váženého zpoždění dávek 1043, nicméně vážené zpoždění zakázek je pouze 560. Také z vektoru T_i vidíme, že pouze jedna dávka byla opožděna a to dávka pátá.

4.6 Příklad

V této sekci rozeberme detailně jeden příklad a ukažme na něm jednotlivé kroky algoritmu. Následující příklad obsahuje 30 vygenerovaných zakázek a 3 typy výrobků. Jejich distribuce a využití kapacity stroje je znázorněna na následujícím obrázku,

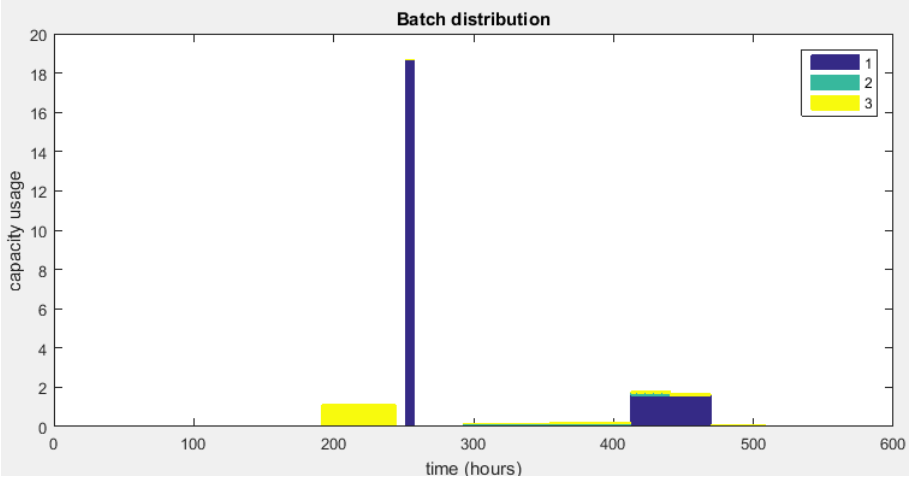


Obrázek 4.8: Geneorvané zakázky

kde je vidět jasná převaha úloh na výrobek typu 1 oproti zbývajícím dvěma typům. Z těchto dat lze určit orientační vytížení linky jako

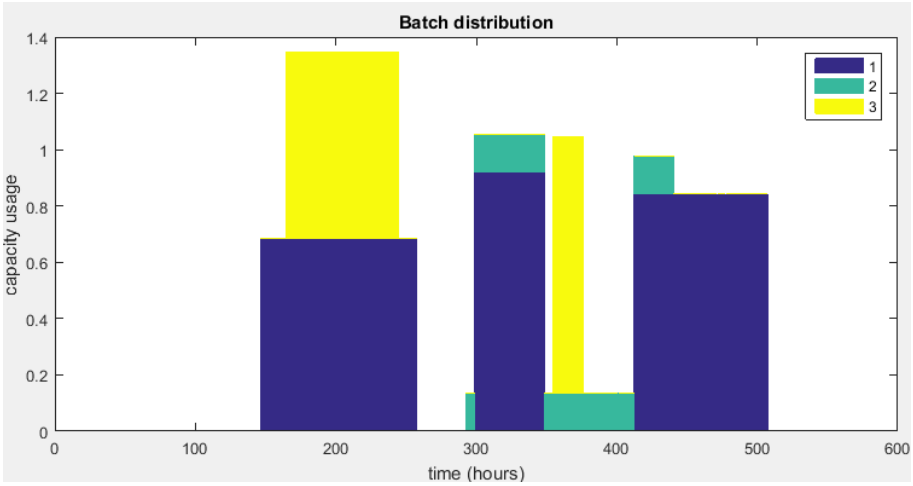
$$\frac{\sum(p_i)}{(max(d_i) - min(r_i))} \tag{4.5}$$

Kde $i, i = 1, \ldots, n$ jsou indexy úloh. Nyní jsou z úloh vytvořeny prvotní (neoptimální!) dávky, které jsou na obázku 4.9. Počáteční konfigurace obsahuje dvě dávky pro typ 1 a 3 a jednu dávku pro typ 2. Je také spočítán *overload* této konfigurace, který je popsán blíže v kapitole 4.4.1, což je skóre penalizující vysoké kapacity zakázek. *overload* je roven hodnotě 794.73.



Obrázek 4.9: Počáteční konfigurace dávek

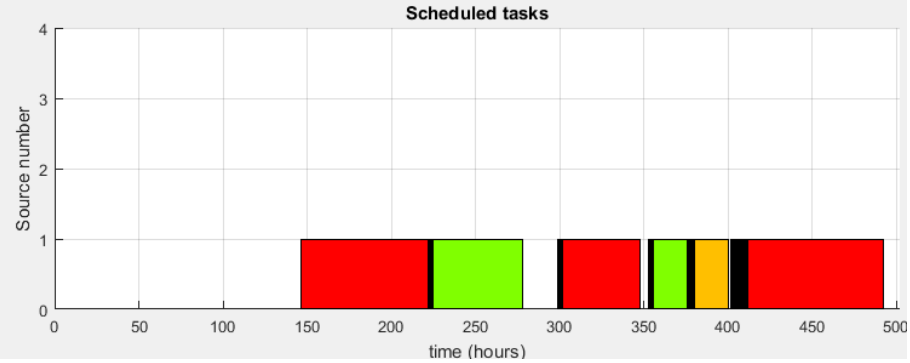
Po průběhu dávkovacího algoritmu dostaneme optimální řešení, které splňuje požadavky na to, aby každá dávka měla kapacitu menší než $caplim = 0.95$. Z následujícího obrázku je vidět, že všechny dávky tuto podmínku splňují. Také lze pozorovat, že počet dávek typu 1 byl navýšen a tedy celkový počet dávek pro tento typ jsou tři. Také si lze všimnout, že první 2 dávky v součtu přesahují kapacitu 1. Toto je zcela validní a kvůli překročení kapacity je také tato konfigurace penalizována podle vzorce 4.3. Nicméně i přes tuto penalizaci tato konfigurace má stále nejlepší *overload* oproti konfiguracím jiným.



Obrázek 4.10: Konečná konfigurace dávek

Konečný *overload* je roven hodnotě 31.51. Dále jsou splněny požadavky skladu, kde se odeberou dávky, úlohy, popřípadě části úloh, pokud je

přebytek zboží na skladu nebo naopak přidá processing time k dávkám, pokud je potřeba nějaké zboží na sklad vyrobit. Po odebrání dávek nebo úloh je znovu puštěn algoritmus dávkování, který se pokusí stávající konfiguraci zlepšit.



Obrázek 4.11: Rozvrh dávek

Poté je spuštěn algoritmus rozvrhování, který vytvoří samotný rozvrh dávek, který je znázorněn na obrázku 4.11. Zároveň jsou vypsány hodnoty, které jsou popsány v předchozí kapitole. Parametry tohoto příkladu jsou zobrazeny na obr. 4.7.

Jak již bylo v předchozí kapitole zmíněno, jediná dávka, která nesplnila due date, je 5. dávka, což je 1. dávka typu tři. Jsme tedy schopni zjistit úlohy, které dávka obsahuje z proměnné $batchset\{3, 1\}$, kde první parametr je typ dávky a druhý parametr pořadí dávky v rámci jejího typu.

```
  13.0000  244.0000    7.0000    1.0000    3.0000    0.0303    2.0000
  14.0000  251.0000    7.0000    3.0000    3.0000    0.0295   10.0000
  19.0000  251.0000   11.0000    7.0000    3.0000    0.0474    4.0000
  41.0000  246.0000   13.0000    8.0000    3.0000    0.0634    5.0000
 118.0000  325.0000    8.0000    4.0000    3.0000    0.0386   21.0000
 164.0000  402.0000    7.0000    7.0000    3.0000    0.0294   12.0000
```

Obrázek 4.12: Úlohy obsažené v 5. dávce

Zde druhý sloupec značí due dates jendotlivých úloh (viz definice úloh a zakázek 4.1). due date dávky je 244, jelikož je to minimální due date ze všech úloh, které dávka obsahuje a zpoždění dávky je $T_5 = 34$. Jsme tedy schopni dopočítat, které úlohy jsou zpožděny. V tomto příkladu jsou to úlohy 1, 2, 3 a 4, jejichž celkové vážené zpoždění je 560.

Kapitola 5

Software design dokument

V této části je popsán software a jeho jednotlivé moduly. Také jsou zde uvedeny pseudoalgoritmy, které byly v této práci implementovány a které jsou popsány v kapitole 4.4.

5.1 Implementované algoritmy

V této části jsou uvedeny pseudoalgoritmy implementovaných algoritmů.

5.1.1 Algoritmus požadavků skladu

Na vstupu mějme konfiguraci dávek *batchset* obsahující dávky a úlohy, které do těchto dávek patří a seznam dávek $(batches_1, \ldots, batches_n)$. Dále počáteční stav skladu $(initstick_1, \ldots, initstock_m)$ a konečný stav skladu $(endstock_1, \ldots, endstock_m)$ pro každý typ výrobku, kterých je m. Vzhledem k délce pseudokódu je algoritmus rozdělen na 2 části. V první části jsou na řádcích 1 až 15 odebrány dávky a na řádcích 16 - 40 jsou odebrány úlohy, definující release time nebo due date dávek. Poté jsou dávky předávkovány, jelikož je možnost nalezení lepšího řešení. V Druhé části na řádcích 2 až 16 je odebrán processing time. Ve zbytku pak je přidán processing time k dávkám, pokud je potřeba vyrobit produkty na sklad. Celý processing time je rozdělen na 10 částí, které jsou postupně přidávány tak, aby *overload* byl minimální.

Algorithm 1: Algoritmus skladu část 1

```
Data: batchset, batches, initstock, endstock
Result: batchset
cantake = max(initstock − endstock, 0);
minoverload = Inf;
while any(batches(p_i)) < cantake(f_i) do
    for i = 1 to n do
        if batches(p_i) < cantake(f_i) then
            remove batch i and compute overload ;
            if overload < minoverload then
                minoverload = overload;
                minbatch = i;
            end
        end
    end
    remove batches(minbatch); minoverload = Inf;
    update batchset, n, m, cantake;
end
while 1 do
    for i = 1 to n do
        curtask = pick task with max r_i from batch i;
        if curtask(p) < cantake(f_i) then
            remove task and compute overload ;
            if overload < minoverload then
                minoverload = overload;
                mintask = i;
            end
        end
        curtask = pick task with min d_i from batch i;
        if curtask(p) < cantake(f_i) then
            remove tasi and compute overload ;
            if overload < minoverload then
                minoverload = overload;
                mintask = i;
            end
        end
    end
    if minoverload == Inf then
        break;
    end
    remove batchset(mintask); minoverload = Inf;
    update batchset, n, m, cantake;
end
```

Algorithm 2: Algoritmus skladu část 2

```
1  REBATCH;
2  for i = 1 to m do
3      if cantake(i) > 0 then
4          for j = 1 to n do
5              if batches(f_j) == i then
6                  remove p_j and compute overload;
7                  if overload < minoverload then
8                      minoverload = overload;
9                      min = j;
10                 end
11             end
12         end
13         remove p from batch min; minoverload = Inf;
14         update cantake;
15     end
16 end
17 togive = max(endstock − initialstock, 0);
18 for i = 1 to m do
19     if togive(i) > 0 then
20         partp = togive(i)/10;
21         for k = 1 to 10 do
22             for j = 1 to n do
23                 if batches(f_j) == i then
24                     add p_j and compute overload;
25                     if overload < minoverload then
26                         minoverload = overload;
27                         min = j;
28                     end
29                 end
30             end
31             add partp to batch min; minoverload = Inf;
32         end
33     end
34 end
```

5.1.2 Algoritmus dávkování

Vstupem dávkovacího algoritmu je n úloh a parametr *caplim* určující maximální povolenou kapacitu dávky. Ve vektoru *overloads* jsou uloženy hodnoty *overload* a v *batchsets* jejich příslušné konfigurace. Vektor *batchlim* definuje maximální dovolený počet dávek pro každý typ. Nejdříve na

řádku 2 je utvořena počáteční konfigurace a spočítán *overload*, počet dávek pro každý typ *batchlim* a je uložena tato konfigurace do *batchset*. Poté si dávky předávájí úlohy a pokud se *overload* zlepší alespoň o δ, jsou uloženy do pole *overloads* a *batchsets*. Pokud se již overload nedá zlepšit, tak je zkontrolováno, zda některá z dávek nemá vyšší kapacitu jak *caplim*. Pokud ano, je počet dávek pro tento typ dávky zvýšen a algoritmus se opakuje.

Algorithm 3: Algoritmus dávkování

```
   Data: n úloh, parametr caplim
   Result: r dávek a konfigurace dávek batchset
 1 sort tasks from min r_i ;
 2 make batches till r_i < d_B, compute overload, batchset, batchlim;
 3 vector overloads;
 4 curoverload = overload;
 5 curbatchset = batchset;
 6 while any(c_i) > caplim do
 7     while size(overloads) > 0 do
 8         for pro každou dávku do
 9             give job to the left/right and compute overload;
10             if overload < curoverload − δ then
11                 add overload to overloads;
12                 add batchset to batchsets;
13             end
14         end
15         pick 5 best values from overloads and batchsets, delete
            rest;
16         pick best overload and batchset as curoverload and
            curbatchset;
17     end
18     if any(c_i) > caplim then
19         Increase number of batches batchlim for type with
            c_i > caplim;
20     end
21 end
```

5.2 Kontrola vstupních parametrů

Vstupní parametry jsou zadávány do souboru `Main.m` a jsou poté kontrolovány funkcí `TestInput.m`, která je buď opraví a vypíše varovnou hlášku nebo vypíše error a program skončí.

Zde je výpis všech možných hlášení při špatně zadaném parametru:

n

není integer - Error: *Wrong number of tasks (n) - isn't integer*
je menší než nula - Warning: *Wrong number of tasks (n)... setting n=20*

m

není integer - Error: *Wrong number of product types (m) - isn't integer*
je menší než nula - Warning: *Wrong number for item types (m) ... setting to m=2*

period

je menší než 10 - Warning: *Wrong period - is less than 10, setting period to period = 10*

ddmin

je menší než 1 - Warning: *Wrong ddmin - is less than 1, setting ddmin = 1*

ddmax

je menší než ddmin+1 - Warning: *Wrong ddmax - is less than ddmin, setting ddmax = ddmin + 1*

procmean

je menší než 1 - Warning: *Wrong procmean - is less than 1, setting procmenan = 1*

procdev

je menší než 0 - Warning: *Wrong procdev, must be positive or zero, setting prodev = 1*

rndweight

je menší než 1 - Warning: *Wrong procdev - is less than 1, setting rndweight = 2*

famratios

nerovná se délce m - Error: *Wrong length of famratios vector (must be m elements long)*

prvek menší než 0 - Error: *Negative or zero element in famratios vector*

setupweight

menší než 0 - Warning: *Wrong setupweight value, is negative .. setting setupweight = 1*

familysetups

matice větší než $m \times m$ - Warning: *Familysetups matrix is bigger than m x m, adjusting matrix*
matice menší než $m \times m$ - Error: *Wrong familysetup matrix size, has to be m x m*
prvek menší než 0 - Error: *Negative family setup times*

initialstock

vektor delší než m - Warning: *Wrong initialstock vector size, is bigger tha m elements, ... adjusting to m element size*
vektor menší než m - Error: *Wrong initialstock vector size, has to have m elements*
prvek menší než 0 - Error: *Wrong initialstock values, has negative values*

endstock

vektor delší než m - Warning: *Wrong endstock vector size, is bigger tha m elements, ... adjusting to m element size*
vektor menší než m - Error: *Wrong endstock vector size, has to have m elements*
prvek menší než 0 - Error: *Wrong endstock values, has negative values*

q

není integer - Error: *Wrong number of sources (q) - isn't integer*
menší než 0 - Warning: *Wrong number of sources (q)... setting q=1*

srctimes

matice není než $m \times q$ - Error: *Wrong srctimes matrix size, has to be m x q*
prvek menší než 0 - Warning: *Wrong srctimes values, has negative values*

5.3 Metody

Main.m

Hlavní soubor, kde se definují vstupní parametry algoritmu nebo se data načítají ze seuborů `tasksm.mat` pro načtení posledních generovaných dat nebo `tasksSAP.mat` pro načtení dat exportovaných ze SAPu.

Batch.m

Hlavní algoritmus dávkování.

- Vstupní parametry
  tasks - množina úloh
  familysetups - matice setup times pro přestavbu linky
  setupweight - váha penalizace za přestavbu linky
  caplimit - interní proměnná definující maximální kapacitu dávky
- Výstupní parametry
  minbatchset - konfigurace optimnálních dávek
  minbatches - seznam dávek
  minoverload - minimální hodnota *overload*
  batchlim - počet dávek pro jednotlivé typy výrobků
  globoverload - *overload* prvotního nadávkování
  batchlimorig - počet dávek pro jednotlivé typy po prvotním nadávkování

Batchsetb.m

Funkce pro vytvoření seznamu dávek z konfigurace dávek.

- Vstupní parametry
  batchset - konfigurace dávek
- Výstupní parametry
  batches - seznam dávek

GenTasks.m

Generování úloh podle vstupních parametrů definovaných v `Main.m`.

- Vstupní parametry
  n - počet úloh
  m - počet typů výrobků
  ddmin - minimální interval due date
  ddmax - maximální interval due date
  procmean - průměrný processing time úlohy

procdev - maximální výchylka od procmean
rndweight - maximální váha úlohy
period - maximální čas, do kterého se budou generovat zakázky
famratios - vektor udávající poměr zakázek na typy výrobků

- Výstupní parametry
  tasks - vygenerované úlohy

GiveBLeft.m
Předání úlohy z dávky dávce vlevo.

- Vstupní parametry
  batchset - konfigurace dávek
  num - číslo dávky
  fam - typ dávky
  batchlim - maximální počet dávek pro typ výrobku
- Výstupní parametry
  batches - nový seznam dávek
  batchset - nová konfigurace dávek
  same - indikátor, zda jsou výstupní dávky jiné než vstupní

GiveRight.m
Předání úlohy z dávky dávce vpravo.

- Vstupní parametry
  batchset - konfigurace dávek
  num - číslo dávky
  fam - typ dávky
  batchlim - maximální počet dávek pro typ výrobku
- Výstupní parametry
  batches - nový seznam dávek
  batchset - nová konfigurace dávek
  same - indikátor, zda jsou výstupní dávky jiné než vstupní

graphVals2.m
Počítání *overload* a vykreslení dávek/úloh

- Vstupní parametry
  tasks - seznam úloh/dávek
  caplimit - interní proměnná, udává penalizaci za překročení kapacity této hodnoty
- Výstupní parametry
  overload - hodnota *overload*

InitBatch.m

Prvotní neoptimální nadávkování úloh.

- Vstupní parametry
  famlengths - počet úloh pro každý typ výrobku
  taskset - seznam úloh pro každý typ výrobku
  m - počet typů výrobků
- Výstupní parametry
  batchset - konfigurace dávek
  batchlim - kapacity utvořených dávek

pickBestRes.m

Funkce pro výběr nejlepších konfigurací. Tedy konfigurací s nejmenším *overload*.

- Vstupní parametry
  olist - pole *overload*
  blist - seznam konfigurací dávek
  num - počet nejlepších konfigurací k vrácení
- Výstupní parametry
  olist - nové pole *overload* s num hodnotami
  blist - num konfigurací dávek

runBatch.m

Nalezení nejlepšího *overload* pro danou konfiguraci dávek pomocí posouvání úloh vlevo/vpravo.

- Vstupní parametry
  batchlim - maximální počet dávek pro typ výrobku
  blist - seznam konfigurací dávek
  ovprah - číslo dávky
  olist - pole *overload*
  caplimit - interní proměnná, udává penalizaci za překročení kapacity této hodnoty
- Výstupní parametry
  minbatchset - nejlepší konfigurace dávek
  minbatches - seznam dávek nejlepší konfigurace
  minoverload - hodnota *overload* nejlepší konfigurace dávek

Scheduling.m

MILP pro rozvrhování dávek/úloh na jeden a více zdrojů.

- Vstupní parametry
  tasks - seznam dávek/úloh
  familysetups - matice setup time
  setupweight - váha penalizace setup time
  q - počet zdrojů
  srctimes - matice určující poměr časů strávených na jendotlivých zdrojích
- Výstupní parametry
  out - vektor obsahující hodnotu kriteriální funkce a hodnoty proměnných

takeBatches.m

Odebrání dávky, pokud je dostatečné množství zboží na skladu.

- Vstupní parametry
  initialstock - počáteční stav skladu
  endstock - konečný stav skladu
  batches - seznam dávek
  batchlim - počet dávek pro typy výrobků
  batchset - konfigurace dávek
  overload - hodnota *overload*
  caplimit - interní proměnná, udává penalizaci za překročení kapacity této hodnoty
- Výstupní parametry
  cantake - počet výrobků, které lze ještě ze skladu odebrat po odebrání celých dávek
  batchset - nová konfigurace dávek
  batchlim - nový počet dávek pro typy výrobků
  batches - nový seznam dávek

takeTask.m

Odebrání úlohy z dávky, pokud je dostatečné množství na skladu. Odebírány jsou prioritně úlohy, které definují release time nebo due date dávky.

- Vstupní parametry
  batchset - konfigurace dávek
  cantake - počet, který lze odebrat pro typy výrobků
  batchlim - počet dávek pro typy výrobků
  caplimit - interní proměnná, udává penalizaci za překročení kapacity této hodnoty

- Výstupní parametry
  batchset - nová konfigurace dávek
  cantake - počet výrobků, které lze ještě ze skladu odebrat po odebrání celých úloh

Visualization.m
Vytvoření a grafické zobrazení rozvrhu dávek/úloh

- Vstupní parametry
  fmin - hodnota kriteriální funkce rozvrhu
  xmin - hodnoty proměnných z vytvořeného rozvrhu
  tasks - seznam dávek/úloh
  familysetups - matice setup times
- Výstupní parametry
  taskorder - pořadí dávek/úloh

exportPlantData.m
Skript na parsování .htm dat získaných ze SAPu

- Vstupní soubory
  P1000.htm - obsahuje objednávky
  složka BOM - obsahuje soubory s informacemi o výrobním procesu výrobků.
  složka WH - obsahuje stavy skladů pro výrobky v různých časech
- Výstupní parametry
  tasksSAP.mat - výstuppní matice obsahující úlohy ve formátu vyhovující vstupu algoritmu

Kapitola 6

SAP

Podniky, obzvláště velké korporace, potřebují pro maximální efektivitu a ziskovost získat veškeré důležité informace o své firmě, na základě kterých mohou vydávat rozhodnutí, jak daný podnik bude fungovat. Je proto důležité udržovat přehled nad zaměstnanci (human resources), audity, pracovními postupy, financemi, správou majetku firmy, údržbou a opravami, komunikací mezi pracovníky, analýzou dat a mnoho dalšími interními záležitostmi. Ať už firma potřebuje všechny právě zmiňované moduly nebo jen některé, je dobré používat software, který tyto věci sleduje, zaznamenává a vyhodnocuje. Jedním z takovýchto programů je program Systems, Applications & Products in Data Processing neboli SAP.

Tento program využívá na světě přes 290000 firem ze 190ti zemí světa a je jedním z nejúspěšnějších softwarů pro podnikové plánování.

Sap R\3 se skládá z následujících modulů:

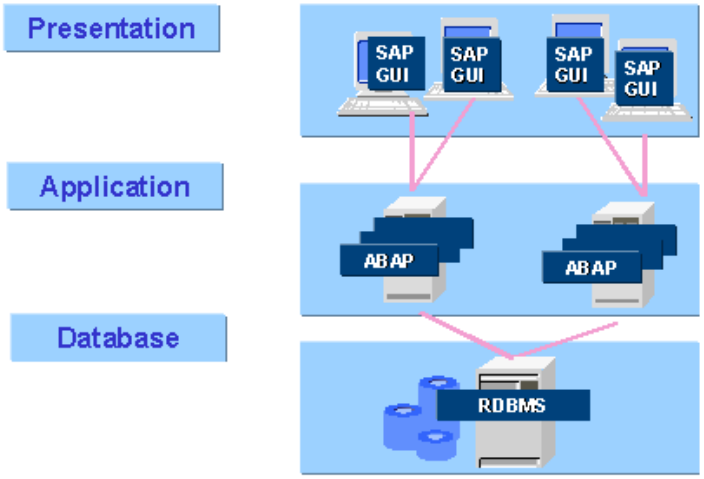
- FI: (Financial Accounting) Finanční účetnictví
- CO: (Controlling) Kontroling
- AM: (Asset Management) Evidence majetku
- PS (Project system) Plánování dlouhodobých projektů
- WF (Workflow) Řízení oběhu dokumentů
- IS (Industry Solutions) Specifická řešení různých odvětví
- HR (Human Resources) Řízení lidských zdrojů
- PM (Plant Maintenance) Údržba
- MM (Materials Management) Skladové hospodářství a logistika

- QM (Quality Management) Management kvality
- PP (Production Planning) Plánování výroby
- SD (Sales and Distribution) Podpora prodeje

Funkčnost systému SAP R\3 je programována v jazyce ABAP (Advanced Business Application Programming), jenž je jazykem umožňujícím vytvářet jednoduché programy. SAP obsahuje vývojové prostředí, které umožňuje vývojářům modifikovat stávající kód či doprogramovávat vlastní funkcionalitu od reportů až po transakční systémy. ABAP komunikuje s databází pomocí SQL dotazů. Nastavení SAPu je vysoce komplexní záležitost, jelikož je nastaven pro specifické potřeby každé firmy. Proto si podniky najímají konzultanty, kteří přizpůsobují systém potřebám dané společnosti.

SAP je klient-server aplikace využívající třívrstvý model (obr. 6.1). R znamená "Real-time data processing"a 3 je užito pro "3-vrstvý", kde jednotlivé vrstvy jsou následující:

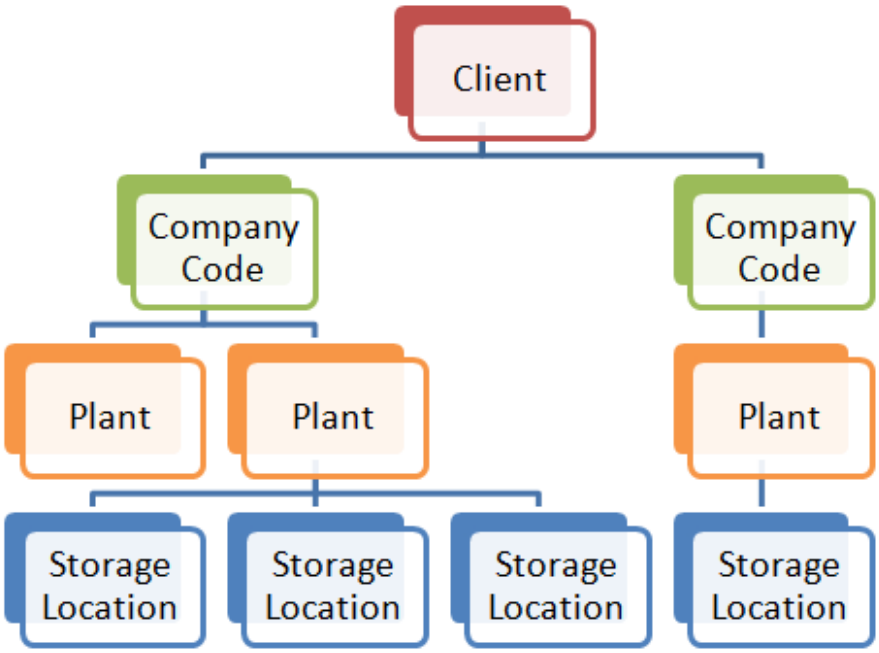
1. Prezenční vrstva
2. Aplikační vrstva
3. Databázová vrstva



Obrázek 6.1: Architegrutra SAPu R\3

6.1 Modelová firma

Jako modelová firma byla vybrána továrna firmy Best-Run Germany se sídlem v Hamburku. Tato firma vyrábí pumpy a další strojírenské produkty. Tato továrna byla vybrána hlavně z důvodu velkého počtu zakázek, a tedy možnosti lepšího testování na implementovaných algoritmech. V SAP systému je tato továrna označována číslem 1000 (plant number). Základní struktura firmy v SAPu vypadá následovně:



Obrázek 6.2: Struktura firmy definovaná v SAPu

Klient je struktura, která definuje práva uživatelů pro různé pracovní pozice v organizaci. Například účty pro SAP vývojáře a sekretářku budou jiná a jejich práva jsou definovaná v systému. Vývojář je pak např. schopen psát programy a vytvářet logické systémy oproti sekretářce, která může vytvářet objednávky a řešit další administrativní záležitosti.

Každá firma obsahuje jeden nebo více Company codes, která definuje nejmenší organizační jednotku a jsou přes ní zpravovány finanční transakce firmy. Tyto organizační jednotky mohou obsahovat fyzické fabriky tzv. plants, v kterých probíhá samotná výroba produktů. S nimi jsou svázány sklady (storage locations), které obsahují materiály pro danou fabriku. Pro lepší představu uvažujme firmu HP, kde jednotlivé company codes odpovídají místům, kde firma působí, tedy např. HP USA, HP India atd.. V nich jsou definované fabriky pro výrobu produktů společnosti

HP, které také mají vlastní sklady pro své potřeby.

6.2 Export dat

Pro export dat ze SAPu pro testování na implementovaných algoritmech bylo zvažováno několik postupů, jak data získat. Nakonec byly zvoleny automatický export a export manuální, jelikož nevyžadují znalost programovacího jazyka ABAP.

V následujících kapitolách jsou popsány oba postupy, jak data získat.

6.2.1 Automatický export dat

Ze SAPu lze automaticky exportovat data pomocí programu sapevt.exe umístěném na serveru, který spouští eventy, na které slyší nadefinované programy. Pro vytvoření uživatelské události je třeba v SAPu nastavit následující položky:

1. Vytvoření logického systému
2. Vytvoření distribučního modelu
3. Vytvoření partnerské dohody
4. Vytvoření uživatelské události
5. Vytvoření varianty programu
6. Vytvoření spouštějícího jobu
7. Spuštění samotné události

Ve výše jmenovaných krocích je vytvořen logický systém, u kterého je definováno, s jakými typy zpráv umí pracovat. Vytvořením a přiřazením partnerské dohody logickému systému jsou definovány IDocy pro typy zpráv, s kterými systém umí pracovat a také port, který určí, kam se data budou posílat a ukládat. IDoc, neboli Imtermediate Document je formát SAP dokumentu, který se používá pro transakce a výměnu dat, ať už s jiným SAP systémem nebo s koncovým uživatelem. Svojí funkcí je tento formát podobný XML, ale struktura je jiná. Oproti XML IDoc využívá formát tabulek s daty a meta-daty a obsahuje také část, kde je popsáno, jakými procesy dokument samotný již prošel, popřípadě má projít. Toto umožňuje sledovat cestu daného IDocu a zpětně trasovat.

Dále je vytvořena uživatelská událost a také varianta programu, která definuje, která data se mají posílat. Kdybychom například pro export dat materiálů nedefinovali materiál nebo množinu materiálů, které

chceme dostat, vyexportovali bychom všechny definované materiály v systému, což v některých případech může trvat i v řádu hodin.

Pro různé zprávy jsou definovány různé IDocy, které definují přesný formát poslaných dat. IDoců pro jeden typ zprávy může být více. To je dáno hlavně kvůli kompatibilitě se staršími systémy které se v některých parametrech liší. Například tedy pro zprávu MATMAS existují IDocy MATMAS01 až MATMAS05. Je tedy třeba mít tyto rozdíly na paměti při práci s více SAP systémy.

Základní definované zprávy jsou vypsány v následující tabulce:

Data	Typ zprávy
Produkční/procesní objednávky	LOIPRO
Plánované objednávky	LOIPLO
Požadavky skladu	LOISTD
Rozvrh	LOIRSH
Materiály	MATMAS
Bill of material (BOM)	LOIBOM
Pracovní centra	LOIWCS
Síť zdrojů	LOIRNH
Hiearchie pracovních center	LOIRNH
Routings	LOIROU
Kalendář	LOICAL
Klasifikace objektů	CLFMAS
Počet poslaných IDoců	LOINUM
Matice přechodů	LOITMX
Skupina produktů	LOIPGR
Stav skladu	LOIMSO
Produkční kampaně	LPOPCM

Tabulka 6.1: Nadefinované IDocy v SAPu

Je nutné podotknout, že export těchto IDoců může vzhledem k jejich velikosti trvat i několik hodin a lze pro export jen některých dat využít lepší způsoby, jako je naprogramování vlastího exportu v ABAPu.

6.2.2 Manuální export dat

Data, která je potřeba získat k otestování algoritmů jsou:

1. Data objednávek
2. Data Bill of material
3. Data stavu skladu

Objednávky lze získat z transakce va03, kde byly vybrány pouze zakázky pro továrnu 1000. Exportovaná data pro zakázky jsou uložena v souboru P1000.htm (viz příloha B). Pro export Bill of material, kde je obsažen postup výroby pro určitý materiál, byla použita transakce na vyhledávání v tabulkách se16. Zde v tabulce MAPL zjistíme číslo BOM v poli PLNNR a podle něj najdeme v tabulce PLPO proces výroby pro tento materiál. Informace ke každému materiálu o jeho výrobě jsou uloženy ve složce BOM. Poslední je stav skladu pro každý materiál. K těmto datům lze přistoupit přes tabulku MBEWH. Data o stavu skladu jsou uložena ve složce WH.

Exportovaná data jsou pro tyto materiály:

HD-1300	Glad Boy configurable
P-100	Pump PRECISION 100
P-101	Pump PRECISION 101
P-102	Pump PRECISION 102
P-103	Pump PRECISION 103
P-104	Pump PRECISION 104
P-109	Pump cast steel IDESNORM 170-230
P-402	Pump standard IDESNORM 100-402

6.2.3 Parsování exportovaných dat

Pro export dat je napsán skript `exportPlantData.m` který se nachází ve složce SAPExport a který rozparsuje soubor zakázek a data uloží do matice `orders.mat`. Poté za ve skriptu uživatel definuje úsek který chce vyexportovat a výstup je uložen do matice `tasksSAP.mat`, který obsahuje:

- Seznam úloh
- Počáteční stav skladu
- Přestavbové časy linky

Seznam zakázek obsahuje objednávky i na produkty, pro které není definován BOM. To jsou většinou výrobky typu HAWA, což jsou již hotové výrobky a jsou pouze určeny k přeprodeji. Je tedy třeba takovéto zakázky odstranit. Dále jedna zakázka může obsahovat více úloh, kde každá má jiný due date. Takovéto zakázky je nutné rozdělit, jelikož zakázka je definována tak, že úlohy které obsahuje mají stejný release date a due date. Dále byly vyfiltrovány zakázky, u kterých due date chyběl úplně.

Ze seznamu zakázek jsme tedy schopni zjistit následující informace:

- Release date

- Release time
- Číslo zakázky
- Cena
- Typ požadovaného výrobku
- Množství požadovaného výrobku

Release time byl použit pro upřesnění, kdy úloha může být nejdříve zpracována. Váha zakázek byla určena jejich cenou a jelikož se ceny pohybovaly od cca 5000 německých marek do několika milionů, tyto hodnoty byly normalizovány na interval $< 0, 10 >$.

Ze souborů BOM pro materiály byly získány přestavbové časy linky a také processing time pro jeden výrobek. Pomocí tohoto údaje a počtu požadovaných výrobků ze zakázky jsme schopni určit processing time celé úlohy. Ze souborů WH byl získán parametr pro počáteční stavu skladu.

Kapitola 7

Testování

Při testování pozorujeme několik hodnot a to:

- Približný odhad vytížení linky, který spočítáme jako $\sum(p_i)/(max(d_i) - min(r_i))$ přes všechny úlohy před spuštěním algoritmu
- Počet dávek při prvním (neoptimálním) nadávkování
- Počet dávek po běhu dávkovacího algoritmu
- *overload* při prvním nadávkování
- *overload* po běhu dávkovacího algoritmu
- Hodnotu kriteriální funkce rozvrhovacího algoritmu
- Čas běhu dávkovacího algoritmu
- Čas běhu rozvrhovacího algoritmu

Při exportu dat ze SAPu je na vstupu obdrženo n úloh, matice familysetups s přestavbovými časy a vektor *initialstock* s počátečními hodnotamy skladu.

7.1 Parametry stroje

Veškeré testy proběhly na jednom stroji s těmito parametry:

MB: Intel ASUSTeK P8Z77-V LX2

Procesor: Intel Core i5 3570K, 1600 MHz

RAM: 16 Gbyte DDR3, 1600 MHz

7.2 Testování na genrovaných datech

7.2.1 Příklad 1

Vstupní parametry:

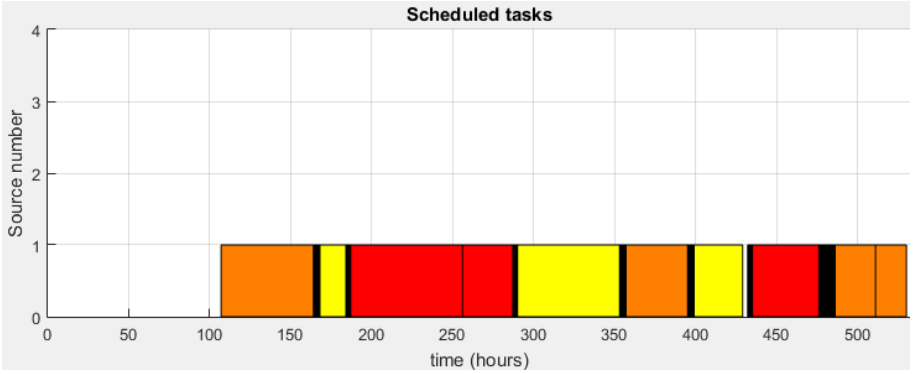
n	m	period	ddmin	ddmax	procmean	procdev	rndw	q
40	3	430	144	240	10	5	8	1

famratios	[1 1 1]
setupweight	1
familysetups	[0 10 3; 10 0 4; 3 4 0]
initialstock	[5 5 5]
endstock	[10 10 10]
srctimes	[1; 1; 1]

Výstup:

Vytížení linky	0.65
Počet dávek na začátku	2 2 2
Počet dávek po optimalizaci	3 4 3
Původní overload	746.75
Konečný overload	12.361
Hodnota kriteriální funkce	67
Vážené zpoždění zakázek	2
Čas běhu dávkování	163.47 s
Čas běhu rozvrhování	4.02 s

Výsledný rozvrh:



Obrázek 7.1: Rozvrh příkladu 1

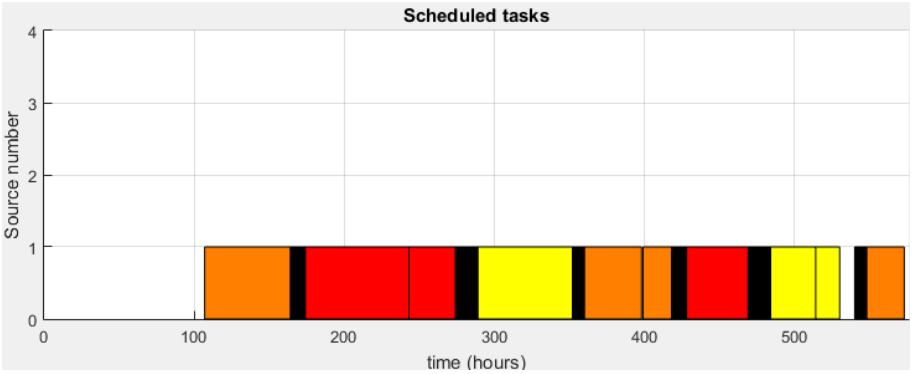
Zde pozorujme hlavně vážené zpoždění zakázek, které se zhoršuje s narůstajícími přestavbovými časy. Upravme tedy matici *familysetups* následovně a algoritmus spusťme znovu na té samé instanci:

familysetups	[0 10 15; 10 0 8; 15 8 0]

Výstup:

Vytížení linky	0.65
Počet dávek na začátku	2 2 2
Počet dávek po optimalizaci	3 4 3
Původní overload	746.75
Konečný overload	12.361
Hodnota kriteriální funkce	993
Vážené zpoždění zakázek	816
Čas běhu dávkování	162.84 s
Čas běhu rozvrhování	17.34 s

Výsledný rozvrh:



Obrázek 7.2: Rozvrh příkladu 1 s upravenými přestavbovými časy

Vážené zpoždění zakázek tedy znatelně vzrostlo a zároveň si lze všimnout, že vzrostl i čas rozvrhování. Samozřejmě *overload* zůstal stejný, jelikož matice přestavbových časů nemá na algoritmus dávkování vliv. Nyní zkusme redukovat toto zpoždění, opět té samé instance, rozvrhnutím dávek na více zdrojů s těmito parametry:

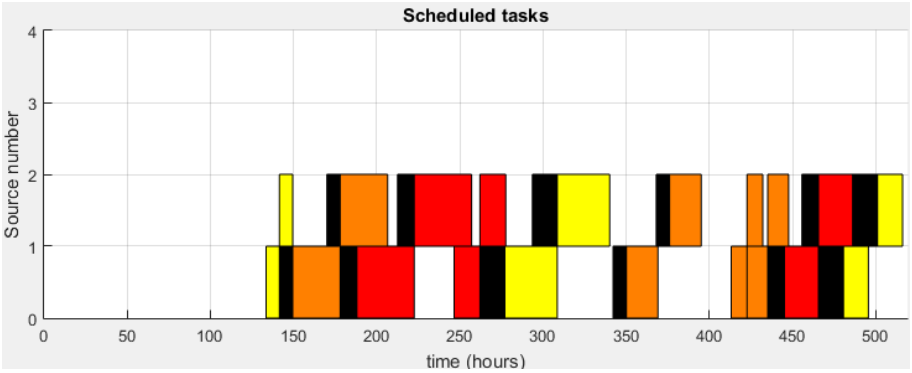
q	2
srctimes	[1 1; 1 1; 1 1]

Matice *srctimes* určuje poměr času, který úloha stráví na zdroji 1 a 2. Tedy polovinu processing time úlohy každého typu stráví na zdroji 1 i na zdroji 2.

Výstup:

Vytížení linky	0.65
Počet dávek na začátku	2 2 2
Počet dávek po optimalizaci	3 4 3
Původní overload	746.75
Konečný overload	12.361
Hodnota kriteriální funkce	66
Vážené zpoždění zakázek	0
Čas běhu dávkování	165.49 s
Čas běhu rozvrhování	6.03 s

Výsledný rozvrh:



Obrázek 7.3: Rozvrh příkladu 1 na 2 zdrojích

Jak lze pozorovat, zpoždění zakázek je nulové. Nicméně tento způsob redukce zpoždění není použitelný v praxi, jelikož počet zdrojů linky je přesně daný a nelze ho modifikovat.

7.2.2 Příklad 2

Vstupní parametry:

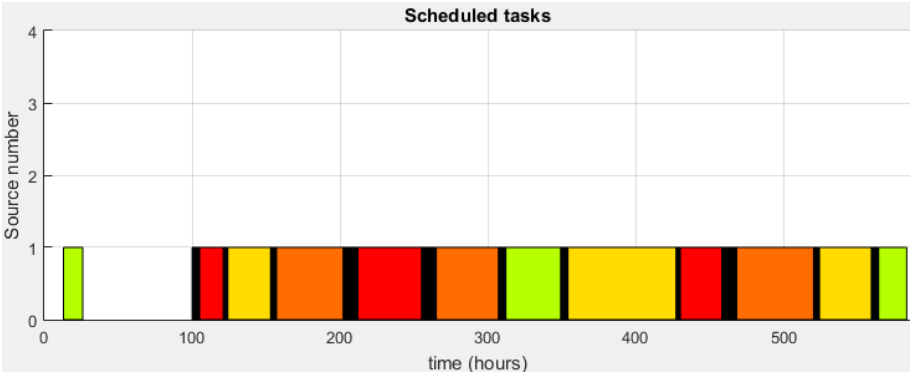
n	m	period	ddmin	ddmax	procmean	procdev	rndw	q
45	4	430	144	240	10	5	8	1

famratios	[1 1 1 1]
setupweight	1
familysetups	[0 10 3 5; 10 0 4 5; 3 4 0 5; 5 5 5 0]
initialstock	[5 5 5 5]
endstock	[10 10 10 10]
srctimes	[1; 1; 1; 1]

Výstup:

Vytížení linky	0.65
Počet dávek na začátku	3 2 2 3
Počet dávek po optimalizaci	3 3 3 3
Původní overload	1307.9
Konečný overload	75.855
Hodnota kriteriální funkce	1429
Vážené zpoždění zakázek	330
Čas běhu dávkování	129.3 s
Čas běhu rozvrhování	1007.89 s (16.8 min)

Výsledný rozvrh:



Obrázek 7.4: Rozvrh příkladu 2

V tomto příkladu si lze všimnout vyššího času rozvrhování, který byl pro 45 úloh a 4 typy výrobků. Zkusme snížit hodnotu zpoždění zakázek úpravou parametrů skladu následnově:

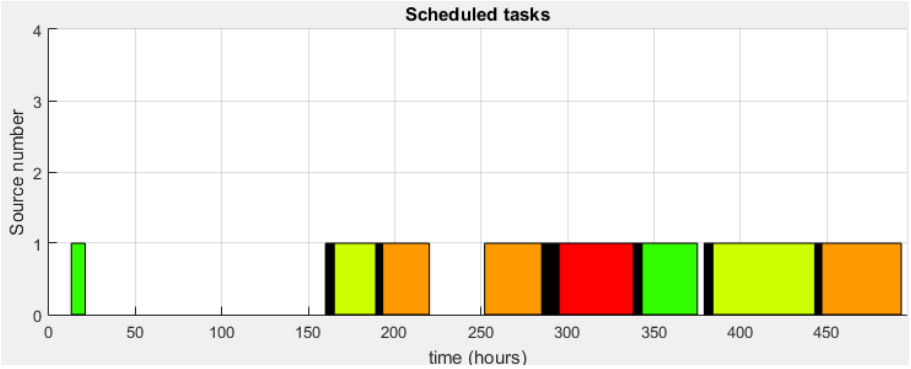
initialstock	[50 50 50 50]
endstock	[10 20 0 25]

Processing time dávek tedy tímto snížíme, jelikož budeme sklad vyprazdňovat. S těmito upravenými parametry pusťme algoritmus na stejné instanci.

Výstup:

Vytížení linky	0.65
Počet dávek na začátku	3 2 2 3
Počet dávek po optimalizaci	1 3 2 2
Původní overload	1307.9
Konečný overload	75.855
Hodnota kriteriální funkce	33
Vážené zpoždění zakázek	0
Čas běhu dávkování	126.66s
Čas běhu rozvrhování	4.85 s

Výsledný rozvrh:



Obrázek 7.5: Rozvrh s redukovaným počtem dávek

Nyní můžeme oba výstupy porovnat, zejména počet dávek. V první iteraci jsme měli 12 dávek a po odebrání processing time ze skladu jich zbylo pouze 8. Zároveň jsme snížili zpoždění zakázek na 0. Díky tomu také algoritmus vyhodnotil danou instanci znatelně rychleji.

7.2.3 Příklad 3

Vstupní parametry:

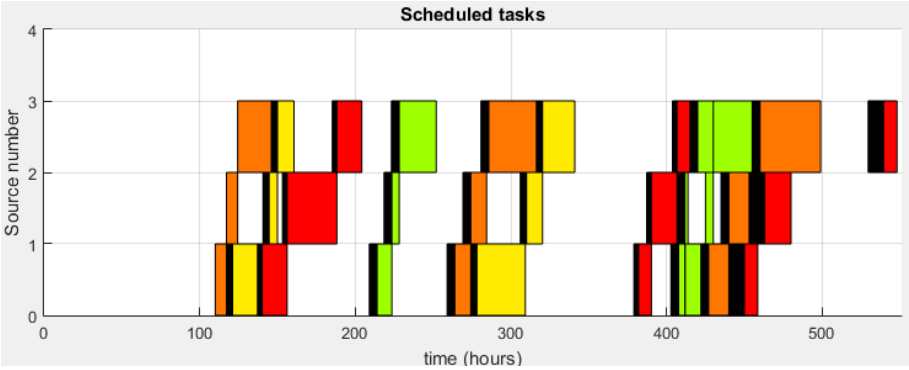
n	m	period	ddmin	ddmax	procmean	procdev	rndw	q
50	4	430	144	240	10	5	8	3

famratios	[1 1 1 1]
setupweight	1
familysetups	[0 10 3 5; 10 0 4 5; 3 4 0 5; 5 5 5 0]
initialstock	[5 5 5 5]
endstock	[10 10 10 10]
srctimes	[1 1 3; 3 1 2; 1 2 1; 2 1 5]

Výstup:

Vytížení linky	0.75
Počet dávek na začátku	2 2 2 2
Počet dávek po optimalizaci	3 3 2 3
Původní overload	3694.74
Konečný overload	58.315
Hodnota kriteriální funkce	44
Vážené zpoždění zakázek	0
Čas běhu dávkování	172.34s
Čas běhu rozvrhování	37.44 s

Výsledný rozvrh:



Obrázek 7.6: Rozvrh na 3 zdrojích

Zde je definována matice *srctimes* s různými poměry peocessing time na různých zdrojích. Lze tedy sledovat, že např. zdroj 3 je vytížen nejvíce oproti předchozím dvěma zdrojům.

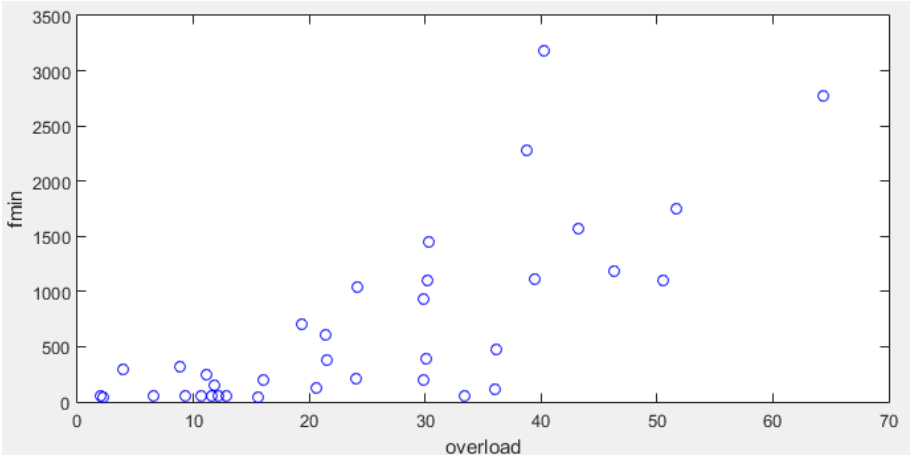
7.2.4 Příklad 4

Pro zjištění, jaká je závislost mezi overloadem a hodnotou kriteriální funkce, byly generovány instance s těmito vstupními parametry:

n	m	period	ddmin	ddmax	procmean	procdev	rndweight
40	2	430	144	240	10	5	8

famratios	[0.5 0.5]
setupweight	1
familysetups	[0 10; 10 0]
initialstock	[5 5]
endstock	[10 10]
caplimit	0.9

Na následujícím grafu je vyznačen *overload* a hodnota kriteriální funkce rozvrhování čtyřiceti instancí. Lze pozorovat, jak se tato hodnota v závislosti na *overload* snižuje. Na základě tohoto testování lze říci, že snížením *ovelroad* snížíme i hodnotu kriteriální funkce.



Obrázek 7.7: Závislost kriteriální funkce na *overload*

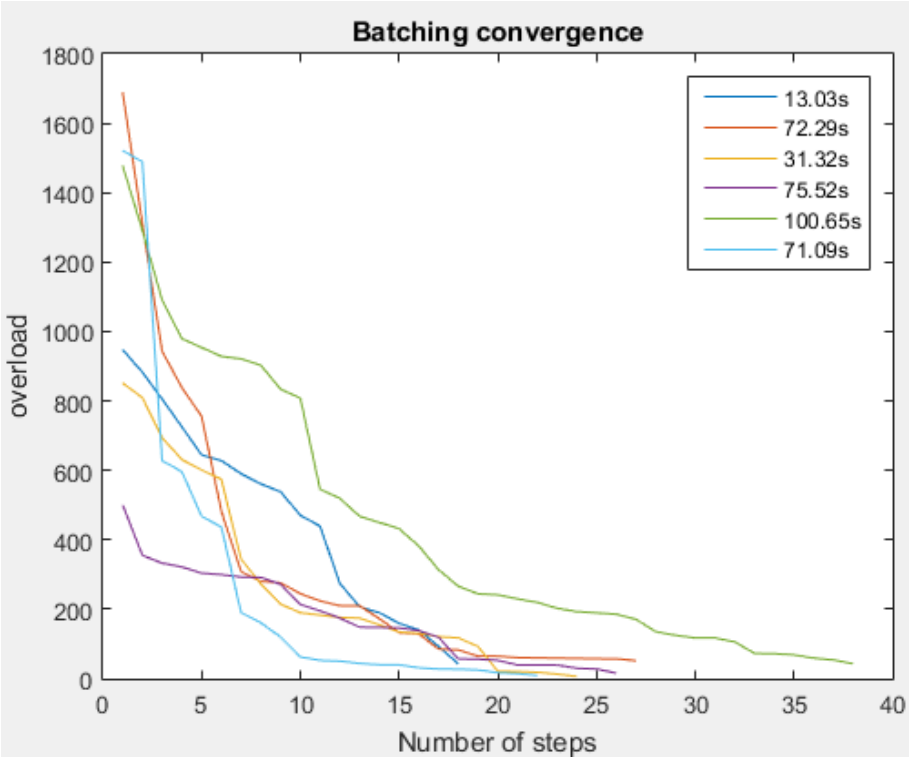
7.2.5 Příklad 5

Vstupní parametry:

n	m	period	ddmin	ddmax	procmean	procdev	rndw	q
40	2	430	144	240	10	5	8	1

famratios	[1 1]
setupweight	1
familysetups	[0 10; 10 0]
initialstock	[5 5]
endstock	[10 10]
srctimes	[1; 1]

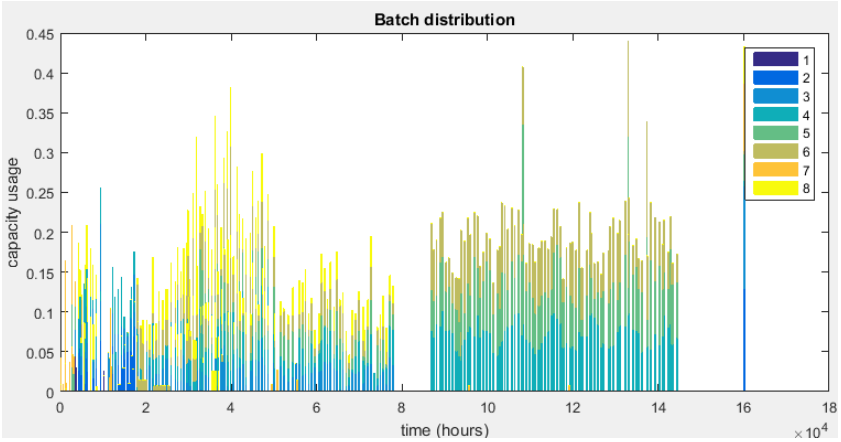
Bylo generováno 6 instancí s parametry uvedenými výše, u kterých byl spuštěn algoritmus dávkování. Následující graf ukazuje jejich rychlost konvergence a hodnotu *overload* v jednotlivých krocích algoritmu. Z grafu lze sledovat, že v některých místech rychlost konvergence zpomaluje. To je dáno hledáním nejlepšího řešení při daném počtu dávek. Poté dojde opět ke zrychlení, jelikož byl počet dávek navýšen a tedy lze úlohy do dávek lépe rozdistribuovat.



Obrázek 7.8: Rychlost konvergence dávkovacího algoritmu

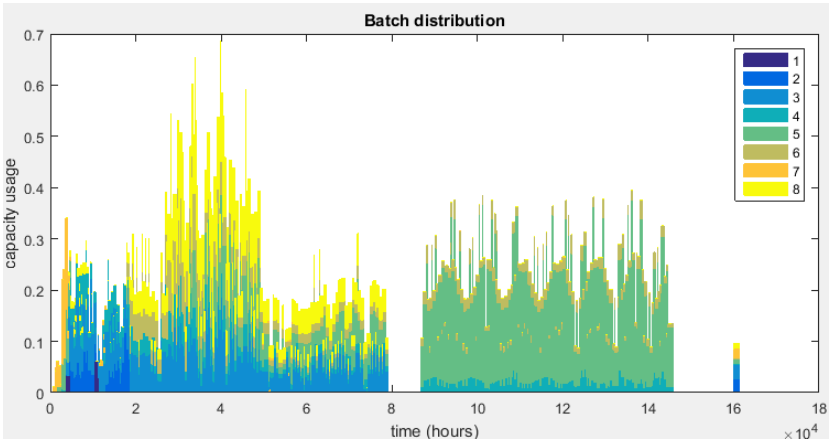
7.3 Testování na demo-datech

Vzhledem k celkovému počtu úloh, které exportované zakázky obsahují (celkem 1540 úloh), je vždy použit pouze vymezený úsek, v kterém jsou úlohy dávkovány a rozvrhovány. Dále, jak z následujících testů vyplyne, je potřeba upravit release time úloh, jelikož úlohy nesdílí jejich časové, jak je znázorněno na následujícím obázku:



Obrázek 7.9: Distribuce exportovaných úloh ze SAPu

Release time úloh byl tedy posunut o 50 dní dříve tak, abychom se zbavili "hluchých"míst mezi jednotlivými zakázkami. Distribuce upravených úloh vypadá takto:



Obrázek 7.10: Upravené release times exportovaných úloh

Instance k následujícím příkladům jsou uloženy v maticích `tasksSAPEx1.mat`, `tasksSAPEx2.mat`, `tasksSAPEx2mod.mat` a `tasksSAPEx3mod.mat`.

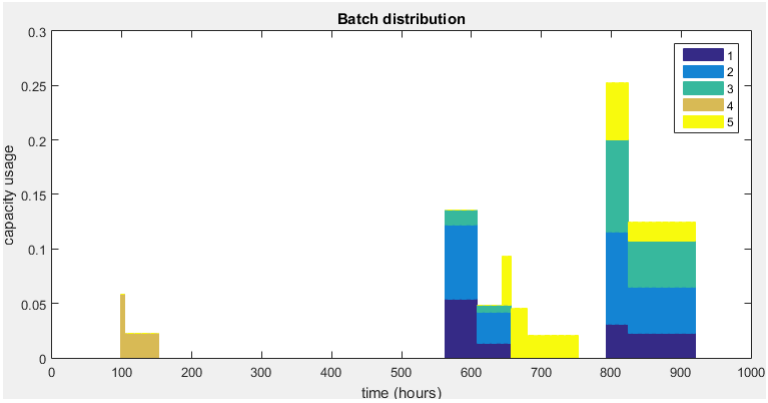
7.3.1 Příklad 1

Vstupní parametry:

n	m	q
18	5	1

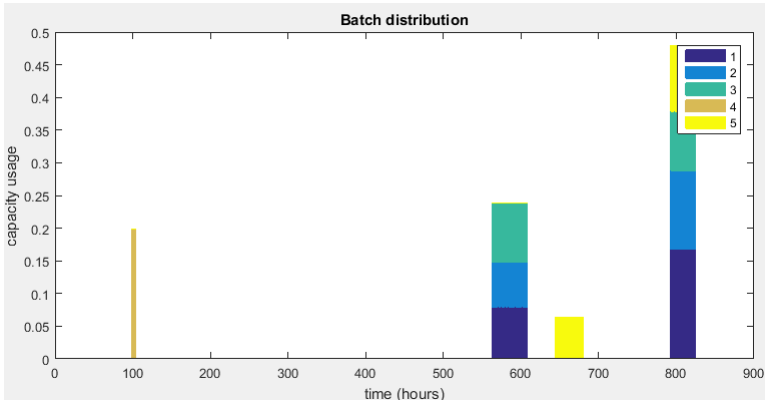
setupweight	1
initialstock	[0 0 0 0 0]
endstock	[0 0 0 0 0]
srctimes	[1; 1; 1; 1; 1]

Distribuce zakázek vypadá takto:



Obrázek 7.11: Distribuce úloh

Zde vidíme, že se překrývají pouze 2 zakázky od každého typu. Ty jsou tedy nadávkovány následovně:

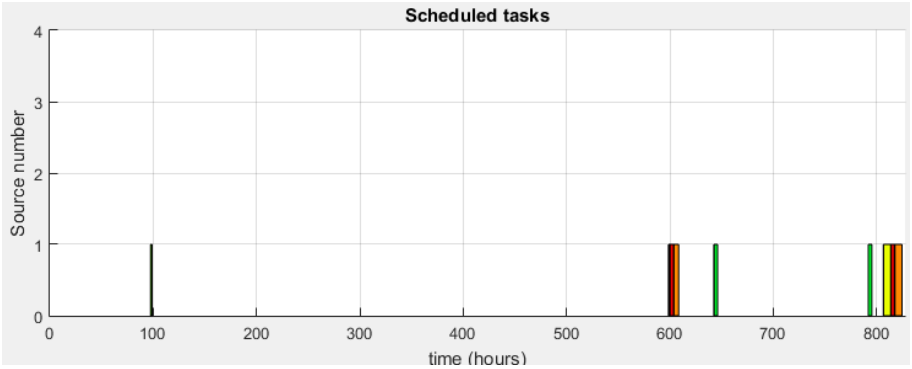


Obrázek 7.12: Vytvořené dávky

Výstup:

Vytížení linky	0.04
Počet dávek na začátku	2 2 2 1 2
Počet dávek po optimalizaci	2 2 2 1 2
Původní overload	0
Konečný overload	0
Hodnota kriteriální funkce	300
Vážené zpoždění zakázek	0
Čas běhu dávkování	0.25s
Čas běhu rozvrhování	0.46 s

Výsledný rozvrh:

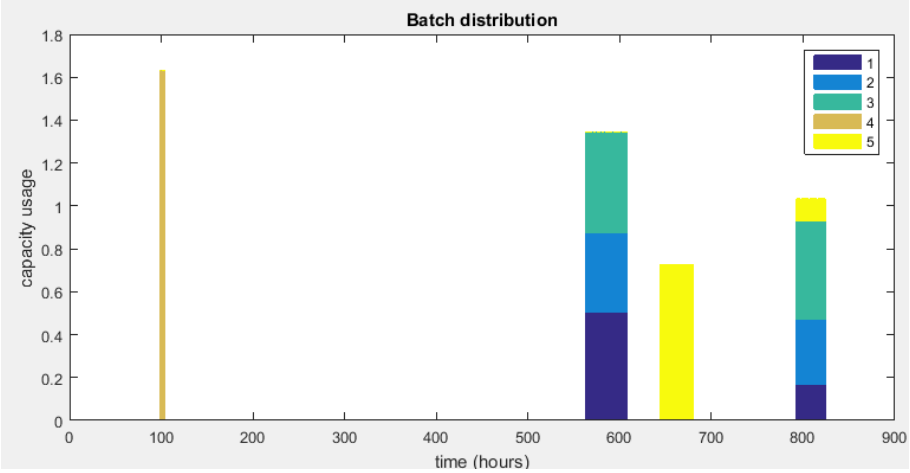


Obrázek 7.13: Rozvrhnuté dávky na jeden zdroj

Zredukovali jsme tedy počet dávek z 18 úloh na 9 dávek. A i přesto, že kapacity těchto dávek jsou malé, dále dávky stejného typu nelze spojit jelikož nesdílí intervaly $< r, d >$. Nicméně změňme parametr *endstock*:

endstock	[20 20 30 10 25]

Po přidání výrobků ze skladu se tedy celkové kapacity navýší, jak je znázorněno na následujícím obrázku. Lze také pozorovat, že dávka typu 4 překračuje kapacitu 1. To je dáno přiřazením úloh, které produkují výrobky na sklad k již existujícím dávkám. A jelikož je tato dávka tohoto typu jediná, výroba byla přiřazena právě sem. Toto zaručuje, že nebude stihnut due date dávky a tedy že požadavek na výrobu na sklad je moc vysoký.

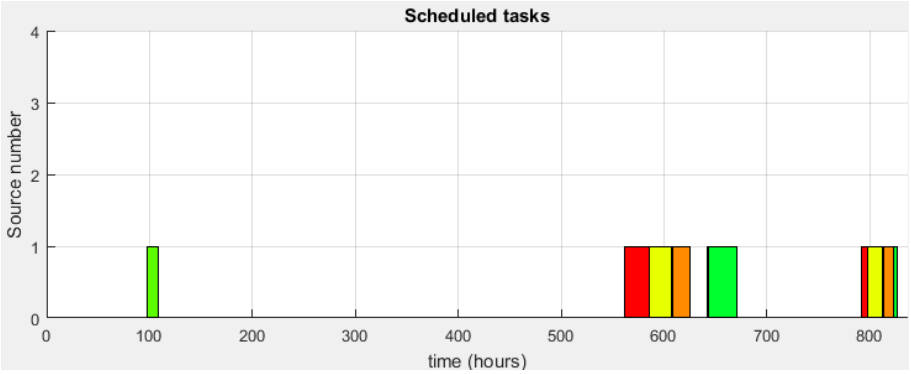


Obrázek 7.14: Dávky po přidání výroby pro sklad

Výstup:

Vytížení linky	0.04
Počet dávek na začátku	2 2 2 1 2
Počet dávek po optimalizaci	2 2 2 1 2
Původní overload	0
Konečný overload	0
Hodnota kriteriální funkce	279.82
Vážené zpoždění zakázek	137.91
Čas běhu dávkování	0.48s
Čas běhu rozvrhování	2.36 s

Výsledný rozvrh:



Obrázek 7.15: Rozvrh dávek na jeden zdroj i s úlohami pro výrobu na sklad

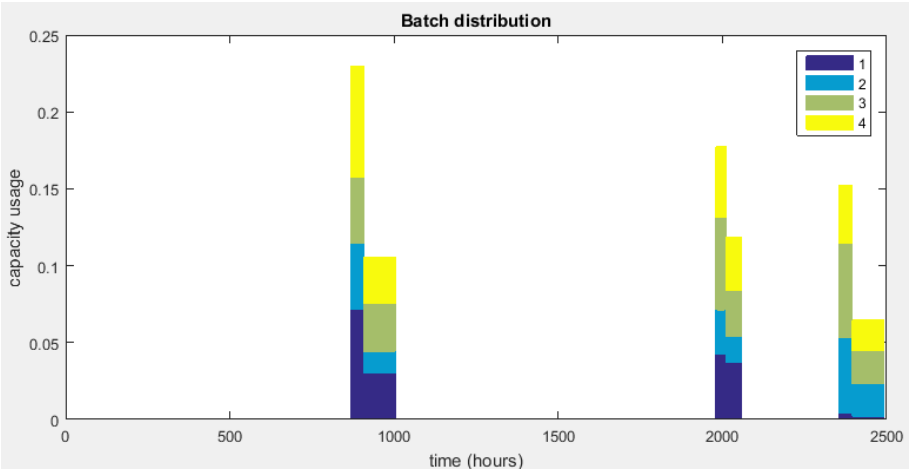
7.3.2 Příklad 2

Vstupní parametry:

n	m	q
24	4	1

setupweight	1
initialstock	[95 45 108 67]
endstock	[105 55 118 77]
srctimes	[1; 1; 1; 1; 1]

Distribuce úloh:

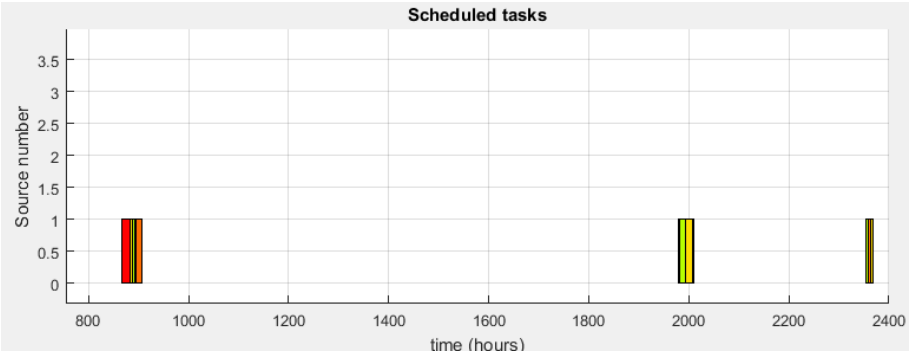


Obrázek 7.16: Distribuce úloh

Výstup:

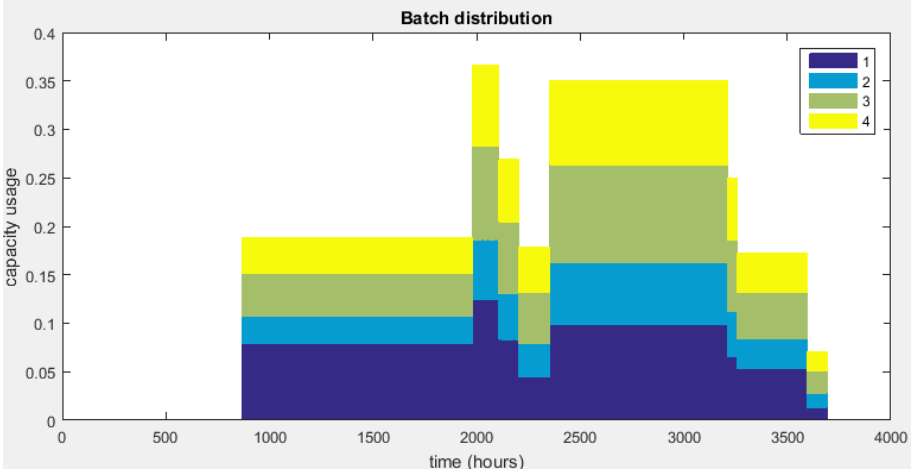
Vytížení linky	0.03
Počet dávek na začátku	3 3 3 3
Počet dávek po optimalizaci	3 3 3 3
Původní overload	0
Konečný overload	0
Hodnota kriteriální funkce	3
Vážené zpoždění zakázek	0
Čas běhu dávkování	0.5s
Čas běhu rozvrhování	23.46 s

Výsledný rozvrh:



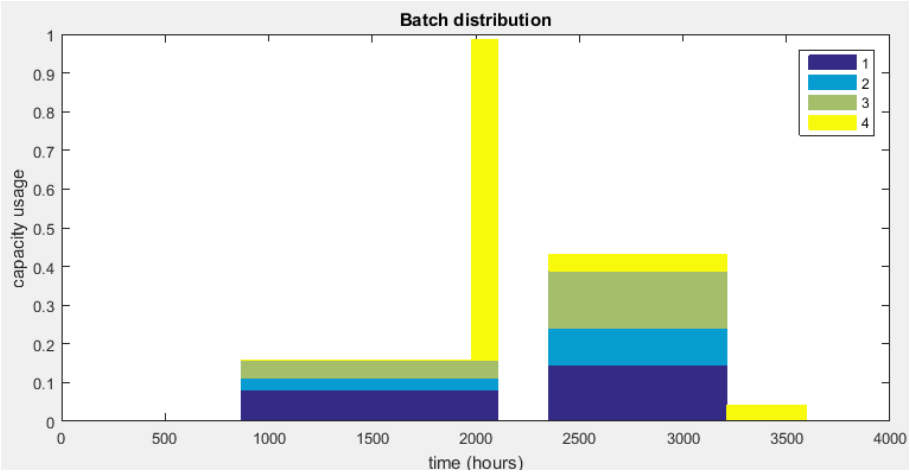
Obrázek 7.17: Rozvrhnuté dávky na jedne zdroj

Opět vidíme, že se úlohy moc nepřekrývají a nelze je tedy efektivně nadávkovat. Pozměňme tedy release date úloh, abychom dosáhli lepší distribuce.



Obrázek 7.18: Distribuce úloh s upraveným release time

Výsledné dávky:

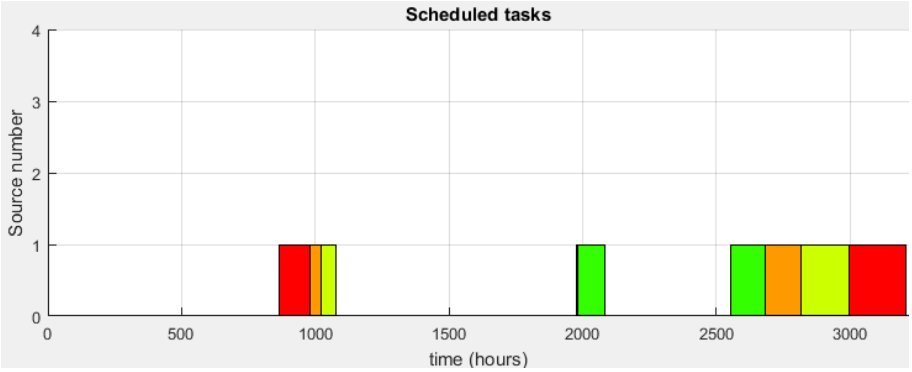


Obrázek 7.19: Dávky modifikovaných úloh

Výstup:

Vytížení linky	0.24
Počet dávek na začátku	2 2 2 2
Počet dávek po optimalizaci	2 2 2 2
Původní overload	403.61
Konečný overload	0
Hodnota kriteriální funkce	2
Vážené zpoždění zakázek	0
Čas běhu dávkování	28.32s
Čas běhu rozvrhování	0.51 s

Výsledný rozvrh:



Obrázek 7.20: Rozvrh dávek na jeden zdroj

Vzhledem k většímu počtu úloh, které se překrývají, běh dávkovacího

algoritmu byl delší, nicméně počet výsledných úloh se snížil. Z původních 12 dávek jsme díky posunutí release times úloh získali dávek 8.

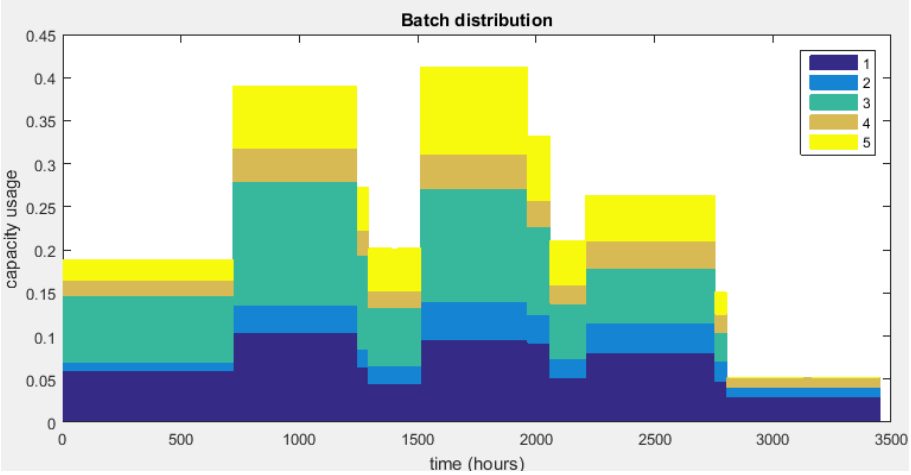
7.3.3 Příklad 3

Opět posuňme release times úloh na této instanci, aby se úlohy překrývaly. Vstupní parametry:

n	m	q
31	5	1

setupweight	1
initialstock	[46 21 298 120 74]
endstock	[46 21 298 120 74]
srctimes	[1; 1; 1; 1; 1]

Distribuce úloh:



Obrázek 7.21: Distribuce modifikovaných úloh

Výstup:

Vytížení linky	0.24
Počet dávek na začátku	2 2 2 2 2
Počet dávek po optimalizaci	2 2 2 2 2
Původní overload	0
Konečný overload	0
Hodnota kriteriální funkce	3
Vážené zpoždění zakázek	0
Čas běhu dávkování	0.59s
Čas běhu rozvrhování	7.81 s

Výsledný rozvrh:

Obrázek 7.22: Rozvrh dávek na jeden zdroj

7.4 Vyhodnocení výsledků

Při testování na generovaných datech byly v prvních 3 příkladech ukázány funkce a chování algoritmu. Ve čtvrtém příkladu byla testována závislost *overload* na hodnotě kriteriální funkce váženého zpoždění zakázek, kde jsme pozorovali, že při snižování *overload* klesá i hondota kriteriální funkce. V pátém příkladu jsme sledovali rychlost konvergence dávkovacího algoritmu, která by se dala zlepšit lepším odhadem konečného počtu dávek. V tomto algoritmu na začátku spočítáme minimální množství dávek, které jsou pak v průběhu navyšovány.

U generovaných dat se jednotlivé intervaly zakázek neprotínaly a tedy nebylo jak efektivně nadávkovat úlohy. Nicméně s úpravou release times jsme toho byli schopni částečně docílit redukce počtu dávek, které byly poté rozvrhnuty na 1 zdroj.

Kapitola 8

Závěr

Cílem této diplomové práce bylo navrhnout a implementovat dávokvací algoritmus, který by za pomoci heuristiky našel řešení, které nemusí být vždy optimální. Navržený model funguje na principu minimalizace overloadu, tedy je snaha o minimalizaci kapacit dávek a také o minimalizaci počtu dávek. Dále byl navrhnut algoritmus pro rozvrhování úloh na jeden a více zdrojů. Dále bylo nutné se seznámit se systémem SAP a vhodným způsobem z něj exportovat data. Zde byly představeny dva způsoby, jak toho docílit.

Algoritmy poté byly testovány na generovaných datech a datech exportovaných. U exportovaných dat nebyl algoritmus velice efektivní, jelikož exportované zakázky nebyly určeny k dávkování a tedy typově nebyly zcela vyhovující. Nicméně byly modifikovány tak, aby na nich mohla být aslespoň z části ukázána funkcionalita implementovaných algoritmů.

Práce nabízí několik možností, jak stávající implementaci zlepšit. Jednou z nich je například lepší odhad počtu dávek a tím pádem snížení doby běhu dávkovacího algoritmu. Dále také lze do výpočtu penalizace dávkovacího algoritmu uvažovat přestavbové časy strojů a tím algoritmus ještě více zrychlit.

Literatura

[1] Cheng, T., Ng, C., Yuan, J., & Liu, Z., *Single machine scheduling to minimize total weighted tardiness.* European Journal of Operational Research, 165(2), 423-443, 2005

[2] Koole G., Righter R., *A Stochastic Batching and Scheduling Problem.* Probability in the Engineering and Informational Sciences, 15, pp465-479, 2001

[3] Albers S., Brucker P., *The complexity of one-machine batching problems.* Discrete Applied Mathematics, Volume 47, Issue 2, 1993

[4] Fliedner, M., Briskorn, D., & Boysen, N., *Vehicle scheduling under the warehouse-on-wheels policy.* Discrete Applied Mathematics, 205, 52-61, 2016

[5] Naddef D., Santos C., *One-pass batching algorithms for the one-machine problem.* Discrete Applied Mathematics, Volume 21, Issue 2, 1988

[6] K.C. Tan, R. Narasimhan, *Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach.* Omega, Volume 25, Issue 6, December 1997

[7] S. Mohri, T. Masuda and H. Ishii, *Batch scheduling problem with multiple due-dates constraint.* Computers and Industrial Engineering (CIE), 2010 40th International Conference on, Awaji 2010

[8] Xiangtong Qi, Fengsheng Tu, *Earliness and tardiness scheduling problems on a batch processor.* Discrete Applied Mathematics, Volume 98, Issues 1–2, October 1999

[9] Bagchi, T., Gupta, J., & Sriskandarajah, C., *A review of TSP based approaches for flowshop scheduling.* European Journal of Operational Research, 169(3), 816-854, 2006

[10] Čapek, R., Šůcha, P., Hanzálek, Z., *Production Scheduling with Alternative Process Plans.* European Journal of Operational Research, Volume 217, Issue 2, March 2012,

Příloha A

Terminologie a zkratky

ABAP = Advanced bussines application programming, je programovací jazyk který je využíván v systému SAP

Batching (dávkování) = proces dávkování úloh

Dávka = batch, shluk objednávek pospojované do jedné

Flow shop Proces ve kterém všechny výrobky projdou stejným procesem výroby

IDoc Intermediate Document, formát který využívá SAP pro komunikaci s jinými SAP systémy a exportu dat

Job shop Proces ve kterém jednotlivé výrobky mohou projít různými výrobními procesy

Konfigurace dávek Daná množina dávek

MILP = mixed integer linear programming

On-time delivery Proces ve kterém firma nevlastní sklad a spoléhá na dovezení materiálu v přesně daný čas.

Počáteční stav skladu Obsah skladu v počátku

Routings Informace o výrobním procesu produktu. Jakými stroji má projít, jaké operace jsou uskutečněny na jakých stanovištích atd..

SAP software pro plánování výroby

Scheduling = Rozvrhování, proces při kterém jsou úlohy/dávky rozvrhnuty na 1 či více zdrojů

Úloha Je definována parametry release date, due date, processing time, váhou, typem, kapacitou a číslem zakázky

Zakázka 1 či více úloh sdílících release date, due date, váhu, typ a číslo zakázky

Příloha B

Obsah CD

- `readme.txt` ..... stručný popis obsahu CD
- `Code` ..... zdrojové kódy implementace
  - `SAPExport` ..... skript pro export dat SAPu
    - `BOM` ..... soubory BOM materiálů
    - `WH` ..... soubory stavu skladu materiálů
- `VyrobniLinka.pdf` ..... text práce ve formátu PDF

Příloha C

SAP: důležité transakční kódy, tabulky a pole tabulek

C.1 Transakční kódy

bd64 distribuční modely

mm03 informace o materiálech

sale vytvoření logického systému

se16 zobrazení tabulek

se38 ABAP programy a varianty programů

sm36 vytvoření jobu

sm62 eventy (události)

va03 seznam zakázek

we20 parnerské dohody

we21 porty

C.2 Tabulky

MAPL kódy procesů pro daný materiál v dané továrně

PLPO výrobní proces pro daný kód (z tabulky MAPL)

T001K seznam firem a jejich okruh ocenění

T001W detailní informace o firmě

VBAK seznam zakázek

VBAP položky zakázek

C.3 Pole tabulek

BURKS company code

BWKEY valuation area

DATUV Datum vytvoření reportu

LTXA1 Popis akce

MATNR čísla materiálů

WERKS plant