VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

REAL-TIME OPTIMALIZACE OPERACÍ V PRŮMYSLOVÉ VÝROBĚ

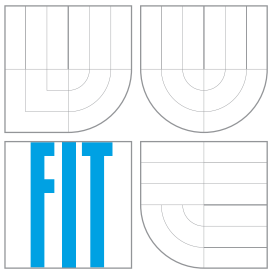
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL KŘEN

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

REAL-TIME OPTIMALIZACE OPERACÍ V PRŮMYSLOVÉ VÝROBĚ

REAL-TIME OPTIMIZATIONS IN INDUSTRIAL PRODUCTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL KŘEN
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN HRUBÝ, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Tato diplomová práce se zabývá rozvrhováním výrobních operací v průmyslové výrobě. Tento problém je formálně popsán jako Resource-Constrained Project Scheduling Problem, jehož cílem je nalezení optimálního přiřazení množiny operací na omezené zdroje. V této práci byl nejprve vytvořen základní optimalizátor výrobních operací založený na genetickém algoritmu. Následně byl navržen model poruch a byl vytvořen systém zahrnující real-time optimalizátor, který je schopen plynule reagovat na vznikající problémy ve výrobě. V real-time optimalizátoru bylo implementováno několik metod řešení a byly s nimi prováděny četné experimenty. Zmíněný systém rovněž umožňuje simulovat provádění výrobních operací a vykreslovat Ganttův diagram.

Abstract

The thesis deals with the scheduling problem of manufacturing operations in industrial production. This problem is described as the well-known the Resource-Constrained Project Scheduling Problem. The objective of this problem is to find an optimal assignment of operations to limited resources. Optimizer created for the thesis uses a genetic algorithm to solve the scheduling problem. For the purpose of a dynamic scheduling, a failures model was designed and a system with real-time optimizer, that is able to repair the original schedule fluently, was created. In the real-time optimizer, several solution methods were implemented and these solution methods underwent a number of experiments. The system thus created is also able to simulate manufacturing operations and draw a Gantt chart.

Klíčová slova

plánování, rozvrhování, optimalizace, simulace, RCPSP, genetické algoritmy, dynamické rozvrhování

Keywords

planning, scheduling, optimization, simulation, RCPSP, genetic algorithms, dynamic scheduling

Citace

Michal Křen: Real-Time optimalizace operací v průmyslové výrobě, diplomová práce, Brno, FIT VUT v Brně, 2012

Real-Time optimalizace operací v průmyslové výrobě

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Martina Hrubého, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Křen
23. května 2012

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu práce Ing. Martinu Hrubému, Ph.D. za jeho odborné rady při konzultacích. Dále bych rád poděkoval Ing. Janu Tykalovi a paní Evě Šimčíkové z firmy Visteon-Autopal za praktické informace o plánování výroby. Na závěr bych také poděkoval mým blízkým za podporu při vytváření této práce.

© Michal Křen, 2012.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

Plánování a rozvrhování, které lze obecně popsat jako optimální přiřazení úkolů na zdroje v čase, patří mezi rozhodovací procesy. Zdroji obecně myslíme stroje, lidi, přistávací dráhy na letišti apod. — naopak úkoly rozumíme činnosti vykonávané na strojích a lidmi, přistávání a vzlety letadel apod. V běžném životě lze najít mnoho příkladů, ve kterých plánování a rozvrhování nalézají praktické uplatnění. Patří mezi ně například rozvrhování směn pro zdravotní sestry, vývoj a nasazení velkého informačního systému, tvorba rozvrhů a jízdních řádů, rozvrhování výrobní linky v průmyslové výrobě a mnoho dalších. Rozdíl mezi plánováním (planning) a rozvrhováním (scheduling) je, že plánování se zabývá především kauzálními vztahy mezi akcemi a výběrem akcí tak, aby bylo dosaženo cíle. Zatímco rozvrhování se soustředí na přiřazení naplánovaných akcí určitému času a prostoru [3]. V češtině se často „scheduling“ překládá jako rozvrhování i plánování.

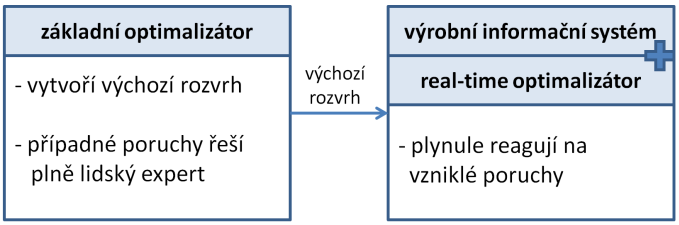
V současné době se ve výrobních firmách střetávají dva protichůdné požadavky. Vedoucí výroby požadují dopředu znát přesně požadavky výroby, aby dokázali efektivně naplánovat jednotlivé operace a údržbu s ohledem na společné strojní vybavení ostatní výroby. Naopak vedoucí logistiky musí neustále měnit požadavky výroby s ohledem na aktuální potřeby zákazníka. Tento konflikt lze vyřešit nasazením SW nástroje, který dokáže vytvořit výrobní rozvrh pro celou výrobu a efektivně tak využít zdroje. Tímto problémem, který lze formálně reprezentovat jako „Resource-Constrained Project Scheduling Problem“ (RCPSP), se zabývá tato práce [16].

V oblasti rozvrhování výroby je již po dlouhou dobu veden poměrně intenzivní výzkum. Většina prací se zaměřuje pouze na statické rozvrhování, kde jsou všechny výrobní operace a omezení známy dopředu. V běžné výrobě ovšem často dochází k neočekávaným událostem, mezi které lze zařadit poruchu stroje, změnu doby trvání výrobní operace a podobně. Právě z těchto důvodů je výhodné mít k dispozici SW nástroj, který by dokázal reagovat na vzniklé problémy ve výrobě. Nově vytvořený rozvrh by měl být sestavený v co nejkratším čase, měl by být podobný původnímu a zároveň být dostatečně optimální vzhledem k optimalizačnímu kritériu. Na řešení tohoto problému je zaměřena tato diplomová práce. Fakta uvedená v prvních dvou odstavcích byla čerpána z [21], [22] a [3].

Tato diplomová práce má následující strukturu. Po úvodu následují dvě kapitoly, které obsahují převážně teoretická východiska práce doplněná o mé vlastní příklady. Tedy druhá kapitola podrobně popisuje RCPSP, pomocí kterého lze popsat problém rozvrhování výrobních operací. Následující kapitola obsahuje přehled různých způsobů řešení tohoto problému.

V rámci této diplomové práce byly vytvořeny dva SW nástroje, jak je zobrazeno na obrázku 1.1. Prvním z nich je základní optimalizátor výrobních operací, který ještě neuvažuje poruchy ve výrobě. Tento základní optimalizátor je popsaný ve čtvrté kapitole. Součástí

této kapitoly je také popis praktického nasazení tohoto optimalizátoru v průmyslové výrobě. V případě narušení vytvořeného rozvrhu výroby by musely být prováděny potřebné změny lidským expertem.



Obrázek 1.1: SW nástroje vytvořené v této práci.

Druhý SW nástroj v této práci imituje chování části výrobního informačního systému a obsahuje real-time optimalizátor. Tento optimalizátor je popsaný v páté kapitole a umožňuje provádět dynamické přeplánování v případě narušení základního rozvrhu. Real-time optimalizátor není možné přímo připojit na reálný výrobní informační systém, proto byl vytvořen nástroj, který emuluje tento systém. Tento nástroj je popsán v šesté kapitole a rovněž umožňuje provádět simulaci vykonávání výrobních operací.

Sedmá kapitola obsahuje experimenty s vytvořenými nástroji a rovněž prezentuje zjištěné výsledky. Poslední kapitolou této práce je závěr, který ji hodnotí a popisuje možnosti dalšího vývoje práce.

První tři kapitoly v této práci včetně úvodu byly z velké části převzaty ze semestrálního projektu.

Kapitola 2

Problém rozvrhování výrobních operací

Problém rozvrhování výrobních operací lze formálně reprezentovat mnoha způsoby. Často je v literatuře tento problém popisován jako „Job-Shop“ problém nebo jeho zobecnění „Resource-Constrained Project Scheduling Problem“ (RCPSP). Teorie rozvrhování je ve skutečnosti popsána neomezeným počtem rozvrhovacích problémů [6]. Pro klasifikaci těchto problémů se často používá tzv. Grahamova notace skládající se ze tří částí. První obsahuje charakteristiku zdrojů, následuje charakteristika operací a poslední je optimalizační kritérium [3]. Standardní RCPSP je tedy popsáno jako $PS|prec|C_{max}$ [10]. Také lze problém rozvrhování výrobních operací kódovat v jiném formalismu např. SAT nebo CSP. Tato kapitola je zaměřena na detailní vysvětlení RCPSP.

2.1 Resource-Constrained Project Scheduling Problem

Obecně se RCPSP skládá z množiny operací a z množiny obnovitelných zdrojů s omezenou kapacitou. Operace mají být vykonány na zdrojích takovým způsobem, aby byly splněny dva druhy omezujících podmínek. První zajišťuje dodržení kapacity zdrojů a druhá zajišťuje dodržení relace následovníka mezi operacemi. Cílem RCPSP je nalezení řešení s minimální dobou trvání (makespan). RCPSP lze formálně popsat jako:

- $\mathcal{J} = \{0, 1, ..., n, n+1\}$ označující množinu všech nepřerušitelných operací, přičemž operace s číslem 0 odpovídá začátku projektu a operace s číslem $n+1$ odpovídá konci projektu. V anglicky psané literatuře se v tomto kontextu setkáme s pojmy „job“ nebo „activity“. Jak již název napovídá, jedná se o jednu dílčí operaci, aktivitu nebo úkol, který se má vykonat v rámci projektu. V této práci je použit pojem operace, neboť se běžně používá při řízení a rozvrhování výroby.

- p_j vyjadřující dobu trvání operace $j \in \mathcal{J}$, kterou nelze během vykonávání přerušit. Nemá žádnou konkrétní časovou jednotku, ale v praxi se obvykle využívá funkce mapující tuto dobu trvání na reálnou dobu trvání (hodiny, pracovní dny atd.). Pro první a poslední operaci platí, že $p_0 = p_{n+1} = 0$.

- $\mathcal{P}_j$ označuje množinu všech předchůdců operace j. Tyto operace musí být dokončeny před startem operace j.

- $\mathcal{K}^\rho = \{1, ..., K^\rho\}$ reprezentující množinu všech obnovitelných zdrojů,

- R_k^ρ určující kapacitu obnovitelného zdroje $k \in \mathcal{K}^\rho$, která je stejná po celou dobu projektu.
- $r_{j,k}^\rho$ určující požadavky operace j na obnovitelný zdroj k. Pro první a poslední operaci platí, že $r_{0,k}^\rho = r_{n+1,k}^\rho = 0$ pro všechny $k \in \mathcal{K}^\rho$.

Koncový čas operace j lze označit jako F_j. Poté vektor obsahující koncové časy operací $S = (F_1, F_2, ..., F_n)$ určuje rozvrh. Množina všech operací, které jsou vykonávány v čase t, je popsána jako $\mathcal{A}_{(t)} = \{j \in \mathcal{J} \mid F_j - p_j \leq t < F_j\}$. Poté lze rozhodovací problém popsat následovně:

$$\min F_{n+1} \tag{2.1}$$

$$F_h \leq F_j - p_j \qquad j = 1, ..., n+1; h \in \mathcal{P}_j \tag{2.2}$$

$$\sum_{j \in \mathcal{A}_{(t)}} r_{j,k} \leq R_k^\rho \qquad k \in \mathcal{K}^\rho; t \geq 0 \tag{2.3}$$

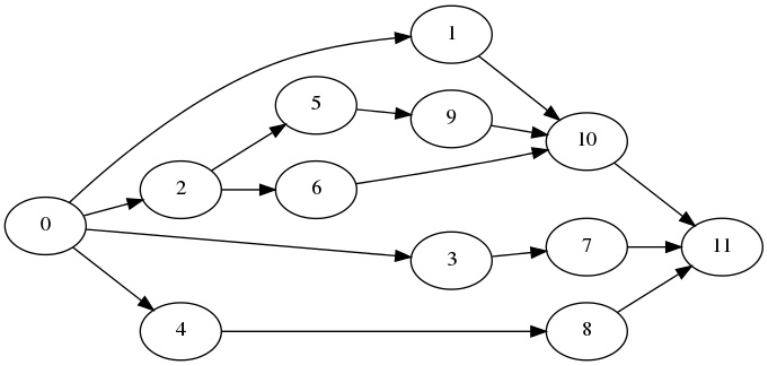
$$F_j \geq 0 \qquad j = 1, ..., n+1 \tag{2.4}$$

Jak již bylo řečeno, cílem RCPSP je minimalizovat dobu trvání projektu (2.1). Druhý vztah (2.2) popisuje, že operace nemůže začít dříve, dokud nejsou dokončeni všichni její předchůdci. Dalším omezením je vztah (2.3) udávající, že během vykonávání operací nesmí být nikdy překročena kapacita zdrojů. Poslední vztah (2.4) definuje nezápornost koncových časů operací. Označení a vztahy byly přejaty z [14].

V následujícím jednoduchém příkladu je demonstrován výše popsaný RCPSP. Množina operací $\mathcal{J}$ obsahuje $n = 10$ skutečných operací a $|\mathcal{K}^\rho| = 2$ zdrojů s kapacitami $R_1^\rho = 7$ a $R_2^\rho = 4$. Doba trvání operací popsaná vektorem p a požadavky operací na zdroje popsané maticí r jsou uvedeny v tabulce 2.1. Zadání problému bylo inspirováno problémem z [1].

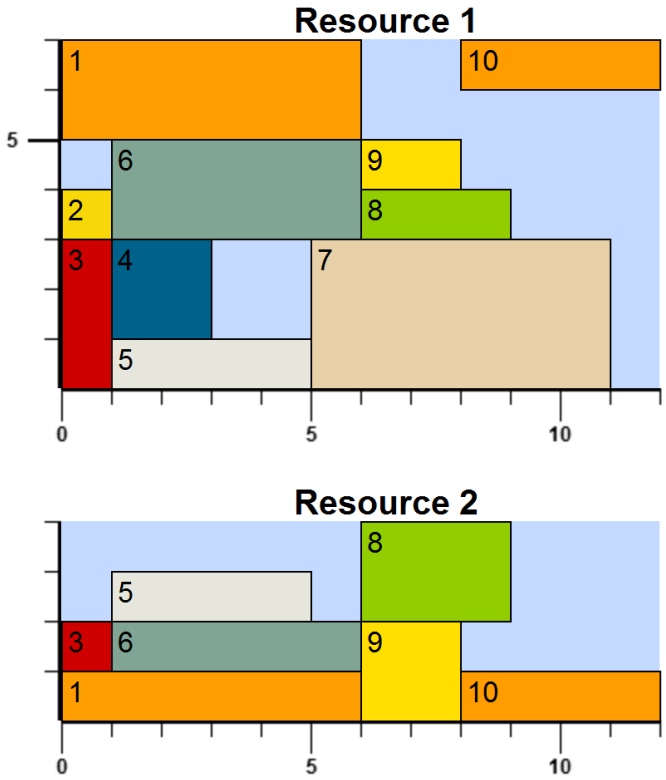
j	0	1	2	3	4	5	6	7	8	9	10	11
p_j	0	6	1	1	2	4	5	6	3	2	4	0
$r_{j,1}^\rho$	0	2	1	3	2	1	2	3	1	1	1	0
$r_{j,2}^\rho$	0	1	0	1	0	1	1	0	2	2	1	0

Tabulka 2.1: Vektor p a matice r u RCPSP se 2 zdroji a 10 operacemi



Obrázek 2.1: Relace předchůdce mezi jednotlivými operacemi v souladu s $\mathcal{P}_j$

Graf[1] zobrazující předchůdce každé operace j v souladu s $\mathcal{P}_j$ lze vidět na obrázku 2.1. Řešení tohoto problému s minimální možnou dobou trvání 12, které bylo vypočteno optimalizátorem vytvořeným v této práci, je zobrazeno formou Ganttova diagramu na obrázku 2.2. Ganttův diagram je pojmenován po jeho tvůrci a průkopníkovi v oblasti plánování Henry Laurence Ganttovi (1861-1919). Vytvořil jej během první světové války za účelem hodnocení harmonogramu výroby [21].



Obrázek 2.2: Vygenerovaný Ganttův diagram zobrazuje využití zdroje v čase. Osa x reprezentuje čas a osa y využití zdroje.

2.2 Rozšíření RCPSP o více módů

V předchozí kapitole popsaný RCPSP lze rozšířit na „Multi-Mode Resource-Constrained Project Scheduling Problem“ (MRCPSP), který zavádí pro každou operaci možnost definice více módů jejího provádění. Umožňuje to modelovat vlastnost, že lze stejnou operaci provést na různých strojích s různou spotřebou neobnovitelných zdrojů. Každá operace v MRCPSP musí běžet právě v jednom z definovaných módů. U tohoto problému má již smysl uvažovat i o neobnovitelných zdrojích, protože operace v každém módu může spotřebovat rozdílné množství neobnovitelných zdrojů (materiálu, financí atd.). Tedy spotřeba materiálu nebude vždy stejná, jak by tomu bylo v případě RCPSP, protože ten obsahuje pouze jeden mód pro každou operaci. Předchozí formální popis RCPSP je rozšířen o:

- $\mathcal{K}^{\nu} = \{1, ..., K^{\nu}\}$ reprezentující množinu všech neobnovitelných zdrojů,

[1]Graf byl vytvořen pomocí nástroje dot – graphviz. Jeho předpis je automaticky generován nástrojem vytvořeným v rámci tohoto projektu.

- R_k^ν určující kapacitu neobnovitelného zdroje $k \in \mathcal{K}^\nu$,
- $\mathcal{M}_j = \{1, ..., M_j\}$ reprezentující pro každou operaci $j \in \mathcal{J}$ množinu všech módů. Tedy operace j může být vykonána v jednom módu z množiny M_j. Dále je definováno, že mód, který má delší dobu trvání a zároveň větší požadavky na zdroje než všechny ostatní, se nazývá neefektivní a mód, který má větší požadavky na zdroje, než je jejich kapacita, se nazývá neproveditelný [20].

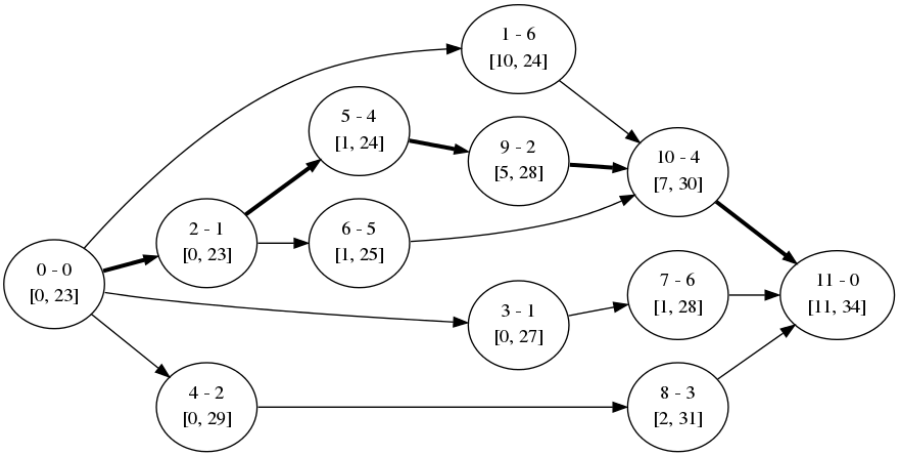
Doba trvání operace a požadavky na zdroje jsou rozšířeny o příslušný mód následovně:

- $p_{j,m}$ vyjadřuje dobu trvání operace j vykonané v módu $m \in \mathcal{M}_j$. Opět platí, že operaci nelze přerušit během vykonávání ani měnit její mód běhu.
- $r^\rho_{j,m,k}$ určuje požadavky operace j vykonané v módu m na obnovitelný zdroj $k \in \mathcal{K}^\rho$.
- $r^\nu_{j,m,k}$ určuje požadavky operace j vykonané v módu m na neobnovitelný zdroj $k \in \mathcal{K}^\nu$. Opět pro první a poslední operaci platí, že $r^\nu_{0,k} = r^\nu_{n+1,k} = 0$ pro všechny $k \in \mathcal{K}^\nu$.

Rozhodovací model popsaný vztahy (2.1) – (2.4) zůstává shodný, pouze by se rozšířil o skutečnost, že každá operace je provedena v jednom z vybraných módů. Dále by se přidala podmínka, že nesmí být překročena kapacita neobnovitelných zdrojů podobně jako v (2.3). Označení a vztahy byly přejaty z [9].

2.3 Základní analýza RCPSP

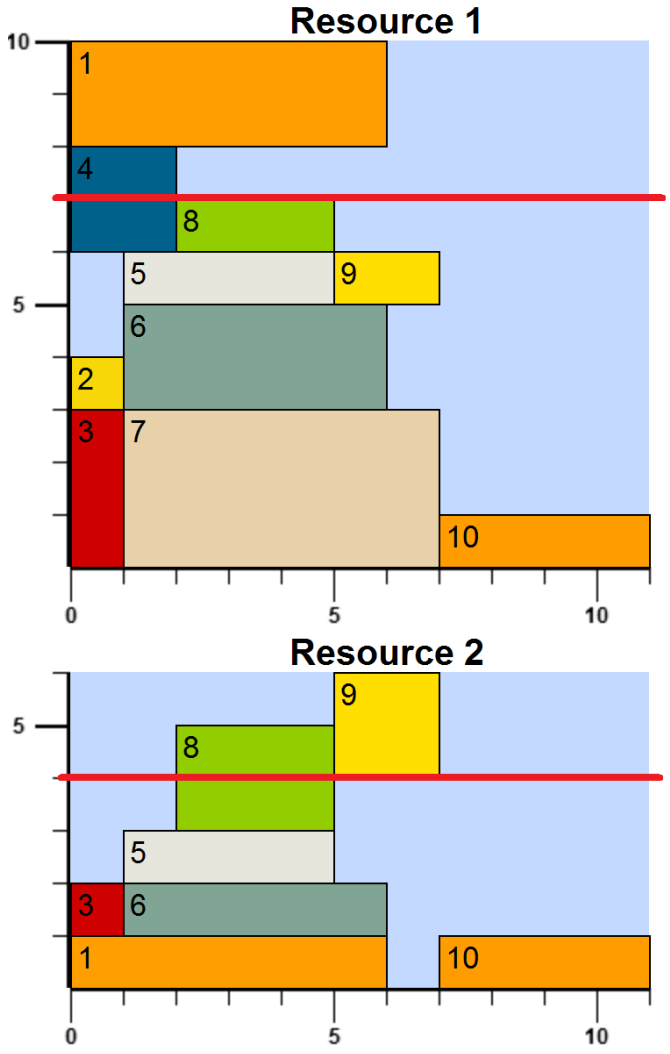
Pro mnoho metod řešení RCPSP je často užitečné provést základní výpočet nejdřívějšího možného startu operace (es_j) a nejpozdějšího možného startu operace (ls_j). Tyto hodnoty jsou užitečné pro omezení prohledávaného prostoru nebo pro lepší než náhodné generování proveditelných plánů. Výpočet těchto hodnot lze realizovat pomocí dopředné a zpětné rekurze nad acyklickým grafem předchůdců. Ve své podstatě se jedná o hledání nejdelší cesty v grafu pro každý uzel, přičemž hrany jsou ohodnoceny dobou trvání operace.



Obrázek 2.3: Výpočet nejdřívějšího a nejpozdějšího možného startu operace (es_j, ls_j). První číslo v každém uzlu značí číslo operace, následuje doba trvání operace a $[es_j, ls_j]$.

Vypočtené hodnoty pro příklad z podkapitoly 2.1 jsou zobrazeny na obrázku 2.3. Tučně zvýrazněná cesta grafem ($0 \to 2 \to 5 \to 9 \to 10 \to 11$) představuje nejdelší možnou cestu.

V anglicky psané literatuře je označována jako „critical path“. Její hodnota je v našem případě 11 a určuje nejkratší možnou dobu trvání rozvrhu (lower bound – LB) při zanedbání omezující podmínky na zdroje (2.3) a respektování relací precedence (2.2). Na obrázku 2.4 je zobrazen rozvrh, který nedodržuje kapacitu zdrojů $R_1^\rho = 7$ a $R_2^\rho = 4$. Nejdelší možná doba trvání projektu je vypočtena jako $T_{max} = \sum_{j \in \mathcal{J}} p_j$ (upper bound – UB) a její hodnota je v našem případě 34. Metody, které dokážou $[es_j, ls_j]$ přesněji určit, jsou popsány v [18]. Jednou z nich je například vybrání operace stojící mimo nejdelší cestu v grafu, pro kterou platí, že ji po nejdelší dobu nelze vykonávat současně s operacemi ležícími na nejdelší cestě grafu. Tato doba je poté připočtena k hodnotě nejdelší cesty grafu a tato hodnota bude určovat LB. Tato podkapitola také čerpá z [1] a [21].



Obrázek 2.4: Rozvrh nerespektující kapacitu zdrojů

Kapitola 3

Metody řešení RCPSP

V roce 1983 Blazewicz a kol. dokázal, že RCPSP, jakožto zobecnění „job-shop“ problému, patří mezi „NP-hard“ problémy [4]. MRCPSP, jakožto rozšíření RCPSP je rovněž „NP-hard“ problém [20]. To znamená, že nejsou známy algoritmy na nalezení optimálního řešení s polynomiální časovou složitostí. Obecně lze metody řešení tohoto problému rozdělit do dvou kategorií — hledající optimální řešení a hledající pouze suboptimální řešení. Do kategorie hledající optimální řešení patří například metoda celočíselného programování. Při větším počtu operací narážejí tyto metody na svá omezení, proto jsou v těchto případech s výhodou používány stochastické metody využívající heuristické funkce při prohledávání stavového prostoru. Ty ovšem nabízejí pouze suboptimální řešení v rozumném výpočetním čase, které může, ale také nemusí být optimální. Mezi ně lze zařadit genetické algoritmy, simulované žíhání, optimalizaci mravenčí kolonií (Ant Colony Optimization) a další. Tato kapitola se zaměřuje převážně na vysvětlení genetických algoritmů a jejich použití při řešení plánovacích problémů. Také obsahuje krátký popis dalších možných metod.

3.1 Přehled metod hledajících optimální řešení

3.1.1 Lineární programování

Lineární programování (LP) je variantou matematického programování, které se obecně zabývá maximalizací nebo minimalizací ohodnocující funkce vhodnou volbou hodnot omezených proměnných. Touto metodou lze s výhodou řešit úlohy typu, že výrobce vyrábí určitý počet výrobků, které při výrobě spotřebují určitý počet zdrojů a ze kterých má různý zisk. Zdroje jsou omezené a cílem výpočtu je zjistit, kolik kterého výrobku vyrábět, aby byl zisk maximální. Na řešení problémů lineárního programování se velmi často používá Simplexová metoda, která efektivně dokáže nalézt řešení obvykle v polynomiálním čase. Vlastností matematického programování je, že vždy nalézá optimální řešení problému [22], [7].

Celočíselné programování (integer programming IP) je variantou lineárního programování, kde proměnné mohou nabývat pouze celočíselných hodnot. Toto omezení je velmi užitečné právě pro řešení plánovacích problémů. Také RCPSP lze vyřešit pomocí celočíselného programování [18] a dále je naznačen postup, jak toho lze docílit.

Pro řešení RCPSP metodou celočíselného programování je nejprve provedena základní analýza za účelem zjištění nejdřívějšího a nejpozdějšího možného startu operací $[es_j, ls_j]$. Výpočet je popsán v podkapitole 2.3. Následně zavedeme proměnnou $\xi_{jt} \in \{0, 1\}$, která nabývá hodnoty jedna, jestliže operace j začne v čase t. V opačném případě má hodnotu 0.

Nyní může být RCPSP popsán následujícím způsobem:

$$\min \sum_{t=es_j}^{ls_j} t \cdot \xi_{jt} \tag{3.1}$$

$$\sum_{t=es_j}^{ls_j} \xi_{jt} = 1, \qquad j \in \mathcal{J} \tag{3.2}$$

$$\sum_{t=es_j}^{ls_j} t \cdot \xi_{jt} - \sum_{t=es_h}^{ls_h} t \cdot \xi_{ht} \geq p_j, \qquad h \in \mathcal{P}_j, \tag{3.3}$$

$$\sum_{j=1}^{J} r_{j,k} \sum_{\tau=\sigma(t,j)}^{t} \xi_{j\tau} \leq R_k, \qquad t = 0, ..., T_{max}; k \in \mathcal{K} \tag{3.4}$$

kde $\sigma(t, j) = \max(0, t - p_j + 1)$. Cílem je tedy minimalizovat dobu trvání projektu (3.1). Vztah (3.2) určuje, že je přípustný pouze jeden začátek operace. Omezení vzhledem k předchůdcům a kapacitě zdroje popisují vztahy (3.3) a (3.4).

Takto popsaný problém celočíselného programování se obvykle řeší pomocí metody větví a mezí (branch-and-bound). Ta spočívá v procházení všech potenciálních řešení, přičemž na základě horní a dolní hranice lze některé řešení vyloučit a tím zmenšit prohledávaný prostor. Další možnou metodou na řešení celočíselného programování je například metoda řezné roviny (cutting plane).

3.1.2 SAT

Další možností je na problém plánování výrobních operací nahlížet jako na SAT-problém (problém splnitelnosti booleovských formulí). Základní myšlenka spočívá v kódování rozvrhovacího problému jako výrokových formulí. Příkladem takové výrokové formule může být, že operace 3 začíná v čase 5. Opět jako v případě celočíselného programování jsou zde předpočítány údaje typu nejdřívější možný start operace a nejpozdější možný start operace. Tímto krokem dojde k omezení hodnot, kterých může například proměnná představující začátek operace nabývat. Následně se pomocí prostředků SATu zjistí splnitelnost této formule. Při řešení se obvykle postupuje způsobem, že je vyžadováno první obvykle neoptimální řešení problému v krátkém čase. Následně dochází ke snižování koncového času poslední operace a ověřování, zda je ještě možno nalézt nějaký proveditelný rozvrh. Z pravdivostních hodnot proměnných lze poté zpětně dekódovat řešení rozvrhovacího problému [11], [3].

3.1.3 Programování s omezujícími podmínkami

Jiným přístupem je nahlížet na problém plánování výrobních operací jako na CSP (Constraint Satisfaction Problem), přičemž SAT se dá považovat za speciální případ CSP. V tomto případě se problém převede na množinu proměnných s konečnými doménami a na množinu podmínek mezi nimi. Cílem je nalezení ohodnocení proměnných tak, aby byly splněny všechny podmínky. Velkou výhodou při tomto přístupu je existence mnoha nástrojů, které dokáží CSP řešit. Tyto nástroje obvykle pracují na podobném principu jako jazyk Prolog [3].

3.2 Algoritmy pro stochastické metody s heuristikou

„Schedule generation schemes“ (SGS) jsou jádrem většiny heuristických funkcí při řešení RCPSP. Existují dvě varianty — „serial“ a „parallel“. První zmíněná varianta se vyznačuje tím, že při provádění postupuje po jednotlivých operacích, kdežto „parallel“ postupuje po čase. Tyto algoritmy lze s výhodou využít pro potřeby vygenerování nového proveditelného rozvrhu a pro vypočtení doby trvání zadané posloupnosti operací (plánu).

3.2.1 Serial Schedule Generation Schemes

Serial Schedule Generation Schemes (SSGS) začíná tvořit rozvrh od první operace projektu v čase 0 a skládá se z $g = 1, ..., n$ kroků. V každém z nich naplánuje aktuálně zvolenou operaci na nejdřívější možný čas jejího začátku při splnění omezujících podmínek (2.2) a (2.3). V každém kroku g jsou také k dispozici dvě disjunktní množiny. První množina $\mathcal{S}_g$ obsahuje již naplánované operace, druhá množina $\mathcal{D}_g = \{j \in \mathcal{J} \setminus \mathcal{S}_g \mid \mathcal{P}_j \subseteq \mathcal{S}_g\}$ obsahuje všechny operace, které jsou připraveny k naplánování. Jinak řečeno, že všechny operace předcházející tuto operaci jsou již hotové. $\tilde{R}_k(t) = R_k - \sum_{j \in \mathcal{A}_{(t)}} r_{j,k}$ udává zbývající kapacitu zdroje k v čase t. Množina všech koncových časů je popsána jako $\mathcal{F}_g = \{F_j \mid j \in \mathcal{S}_g\}$.

Inicializace: $F_0 = 0, \mathcal{S}_0 = \{0\}$,
`For` $g = 1$ `to` n `do`
  Spočítej $\mathcal{D}_g, \mathcal{F}_g, \tilde{R}_k(t)(k \in \mathcal{K}; t \in \mathcal{F}_g)$
  Vyber jednu $j \in \mathcal{D}_g$
  $ES_j = \max_{h \in \mathcal{P}_j}\{F_h\}$
  $F_j = \min\{t \in [ES_j, LF_j - p_j] \cap \mathcal{F}_g \mid r_{j,k} \leq \tilde{R}_k(\tau), k \in \mathcal{K}, \tau \in [t, t + p_j] \cap \mathcal{F}_g\} + p_j$
  $\mathcal{S}_g = \mathcal{S}_{g-1} \cup \{j\}$
$F_{n+1} = \max_{h \in \mathcal{P}_{n+1}}\{F_h\}$

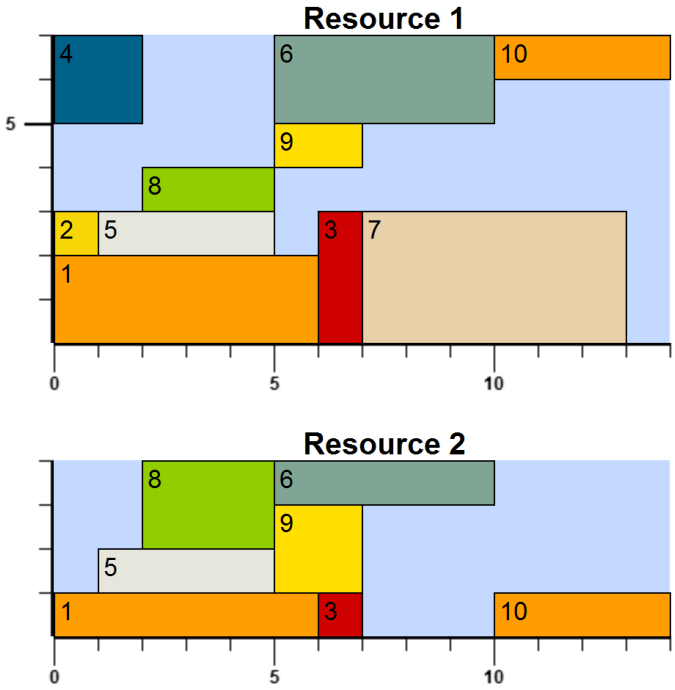
Na začátku výše uvedeného algoritmu je nejprve naplánována první operace reprezentující start projektu na čas 0. Poté se spočítají všechny požadované množiny včetně $\mathcal{D}_g$, která by v prvním kroku, v případě námi uvažovaného problému z podkapitoly 2.1, obsahovala operace $\mathcal{D}_g = \{1, 2, 3, 4\}$. Následně je jedna z těchto operací vybrána na základě dostupného plánu. Pokud by se jednalo o generování nového proveditelného rozvrhu, tak by byla pravděpodobně vybrána náhodně. Následně se určí nejdřívější možný start operace ES_j jako maximální hodnota koncových časů předcházejících operací. Nejdřívější možný začátek operace je tedy roven nebo větší než ES_j a leží v množině koncových časů $\mathcal{F}_g$. Zároveň pro provedení operace musí být dostatek zdrojů po celou dobu jejího vykonávání. Přičtením doby operace je určen koncový čas operace a ta je uložena do již naplánovaných operací. Celkovou dobu projektu určuje poslední operace reprezentující konec projektu F_{n+1}. Teoretická složitost tohoto algoritmu je $\mathcal{O}(n^2 \cdot K)$. Tato podkapitola vychází z [14].

Následující příklad uvedený v tabulce 3.1 názorně demonstruje princip výše popsaného algoritmu na RCPSP uvedeném v podkapitole 2.1. Z množiny operací připravených k naplánování $\mathcal{D}_g = \{1, 2, 3, 4\}$ je v prvním kroku vybrána operace 4, která skončí v čase 2. Po naplánování operace 4 se do množiny $\mathcal{D}_g$ dostává její jediný následovník a to operace 8. V dalším kroku je vybrána operace 2 a celý algoritmus pokračuje stejným způsobem dále. Výsledným plánem reprezentovaným posloupností vybíraných operací je $(4, 2, 5, 8, 1, 9, 6, 10, 3, 7)$. Tento plán není optimální a jemu odpovídající rozvrh s celkovou dobou trvání 14 lze vidět na obrázku 3.1.

g	1	2	3	4	5	6
j	4	2	5	8	1	9
$\mathcal{D}_g$	$\{1,2,3,4\}$	$\{1,2,3,8\}$	$\{1,3,5,6,8\}$	$\{1,3,6,8,9\}$	$\{1,3,6,9\}$	$\{3,6,9\}$
F_j	2	1	5	5	6	7

g	7	8	9	10	11
j	6	10	3	7	11
$\mathcal{D}_g$	$\{3,6\}$	$\{10,3\}$	$\{3\}$	$\{7\}$	$\{11\}$
F_j	10	14	7	13	14

Tabulka 3.1: Příklad výpočtu SSGS



Obrázek 3.1: Ukázka neoptimálního rozvrhu s plánem $(4, 2, 5, 8, 1, 9, 6, 10, 3, 7)$

3.2.2 Parallel Schedule Generation Schemes

Jak již bylo uvedeno, Parallel Schedule Generation Schemes (PSGS) postupuje při svém provádění po čase. V každém kroku algoritmu g je tedy určen čas t_g. Dále je definována množina již provedených operací $\mathcal{C}_g = \{j \in \mathcal{J} | F_j \leq t_g\}$ pro daný čas. Množina všech právě probíhajících operací v čase t_g je určena jako $\mathcal{A}_g = \mathcal{A}(t_g) = \{j \in \mathcal{J} \mid F_j - p_j \leq t_g < F_j\}$. Operace, které byly již naplánovány, jsou tedy právě v $\mathcal{A}_g$ nebo $\mathcal{C}_g$. Klíčovou je množina $\mathcal{D}_g$, která určuje operace připravené k provedení. Tyto operace tedy splňují obě omezující podmínky — na zdroje (2.3) a na předchůdce (2.2). Množina je určena jako

$$\mathcal{D}_g = \{j \in \mathcal{J} \setminus (\mathcal{A}_g \cup \mathcal{C}_g) \mid \mathcal{P}_j \subseteq \mathcal{C}_g \wedge r_{j,k} \leq \tilde{R}_k(t_g)(k \ \in \mathcal{K})\}.$$

Podobně jako u serial SGS udává $\tilde{R}_k(t_g) = R_k - \sum_{j \in \mathcal{A}_{(t)}} r_{j,k}$ zbývající kapacitu obnovitelného zdroje k v čase t_g. Nyní je zde popsán pseudokód PSGS:

Inicializace: $g = 0, t_g = 0, \mathcal{A}_0 = \{0\}, \mathcal{C}_0 = \{0\}, \tilde{R}_k(0) = R_k$
`While` $|\mathcal{A}_g \cup \mathcal{C}_g| \leq n$ `do`

(1) $g := g + 1$
$t_g := \min_{j \in \mathcal{A}_g}\{F_j\}$
Spočítej $\mathcal{C}_g, \mathcal{A}_g, \tilde{R}_k(t_g), \mathcal{D}_g$

(2) `While` $\mathcal{D}_g \neq \emptyset$ `do`
Vyber jednu $j \in \mathcal{D}_g$
$F_j = t_g + p_j$
Spočítej $\tilde{R}_k(t_g), \mathcal{A}_g, \mathcal{D}_g$

$F_{n+1} = \max_{h \in \mathcal{P}_{n+1}}\{F_h\}$

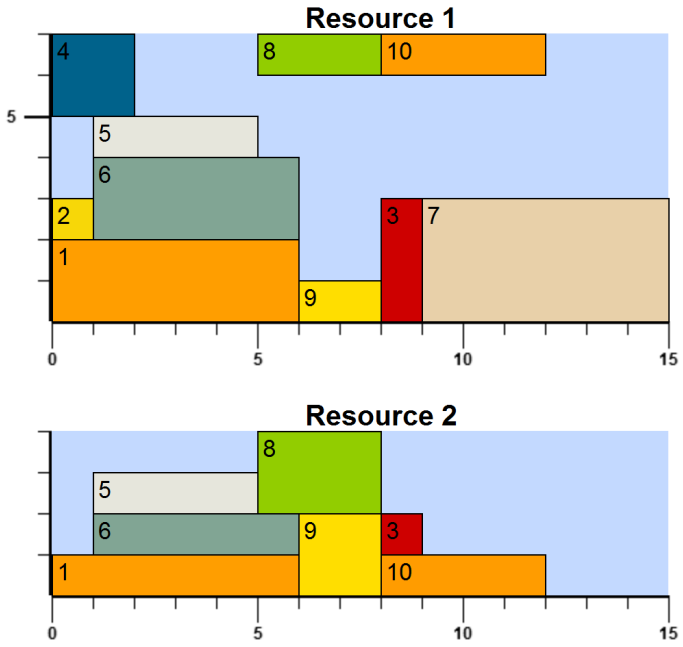
Obecně řečeno z pohledu Ganttova diagramu, SSGS vždy vezme jednu operaci a začne pro ni hledat dostatek zdrojů (místa). Hledání začíná od času, ve kterém jsou dokončeni všichni její předchůdci. Namísto toho PSGS v každém kroku g a v jemu odpovídajícím čase t_g vypočte klíčovou množinu $\mathcal{D}_g$, ve které jsou operace splňující podmínky na své zahájení v daný okamžik. Následně je spuštěn vnořený cyklus (2), který vždy vybere jednu operaci j z $\mathcal{D}_g$. Výběrem dojde k aktualizaci zbývajících kapacit zdrojů $\tilde{R}_k(t_g)$ a k přepočtení množiny $\mathcal{D}_g$, ze které mohou být odstraněny operace, které již nemají dostatečnou kapacitu na zdrojích na své zahájení v daném čase t_g. Teoretická složitost tohoto algoritmu je opět $\mathcal{O}(n^2 \cdot K)$, ovšem reálná časová složitost je obvykle horší než v případě SSGS [13].

g	1	1	1	2	2	3	4	5
t_g	0	0	0	1	1	2	5	6
$\mathcal{D}_g$	$\{1,2,3,4\}$	$\{1,2,3\}$	$\{1,3\}$	$\{3,5,6\}$	$\{6\}$	$\{\}$	$\{3,8,9\}$	$\{3,9\}$
j	4	2	1	5	6		8	9

g	6	6	7	8	9
t_g	8	8	9	12	15
$\mathcal{D}_g$	$\{3,10\}$	$\{3\}$	$\{7\}$	$\{\}$	$\{11\}$
j	10	3	7		11

Tabulka 3.2: Příklad výpočtu PSGS

Následující příklad se zadáním použitým z podkapitoly 2.1 názorně ilustruje princip algoritmu. Vybírání operací bylo prováděno podle plánu z minulého příkladu na SSGS a to $(4, 2, 5, 8, 1, 9, 6, 10, 3, 7)$. Celý výpočet je uveden v tabulce 3.2. V čase 0 jsou postupně vybírány operace 4, 2 a 1. Operace 5 a 8 nejsou v daném čase v množině proveditelných operací $\mathcal{D}_g$ a není je možno tedy vybrat. Naopak z množiny $\{1, 3\}$ je vybrána operace 1, protože v uvedeném plánu leží na nižším indexu než operace 3. Toto je také obecné pravidlo pro výběr konkrétní operace j z $\mathcal{D}_g$. Naopak v případě SSGS je zaručeno, že můžeme brát operace v pořadí, které přesně odpovídá zadanému plánu. Při výpočtu může také nastat situace jako v kroku $g = 3$ v čase $t_g = 2$, kdy není možno naplánovat žádnou operaci, protože není dostatečná kapacita na zdrojích. Výsledný rozvrh je možno vidět na obrázku 3.2 a má celkovou dobu trvání 15, což je horší než v případě SSGS. Daný rozvrh lze také popsat pomocí plánu $(4, 2, 1, 5, 6, 8, 9, 10, 3, 7)$. Vypovídá to o skutečnosti, že jeden konkrétní rozvrh lze popsat více plány, ale naopak jeden konkrétní plán vždy určuje pouze jeden konkrétní



Obrázek 3.2: Ukázka neoptimálního rozvrhu s plánem $(4, 2, 1, 5, 6, 8, 9, 10, 3, 7)$

rozvrh, pokud uvažujeme pouze jednu zvolenou verzi algoritmu. Tato podkapitola vychází z [14].

3.3 Genetické algoritmy

Genetické algoritmy (GA), založené na principech evoluční biologie, se řadí mezi stochastické metody prohledávání stavového prostoru s heuristickou funkcí. Základní myšlenkou je vytvořit množinu řešení problému a snažit se převážně ty nejlepší řešení zkombinovat za účelem nalezení ještě lepšího řešení. Jejich hlavní uplatnění se nalézá v optimalizačních úlohách. Podobně jako u jiných stochastických metod je jejich hlavním problémem uváznutí v lokálním extrému.

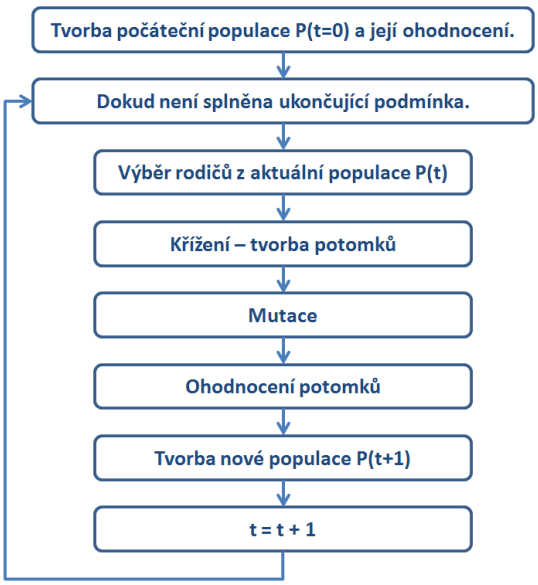
V následujícím výčtu je uveden přehled nejpoužívanější terminologie v genetických algoritmech [5] a [23]:

- *jedinec* (individual) – představuje jedno konkrétní řešení,
- *vhodnost/fitness* (fitness) – v biologii představuje schopnost jedince přežít a reprodukovat se v daném prostředí. V genetických algoritmech fitness odpovídá kvalitě daného jedince (řešení),
- *fenotyp* (phenotype) – představuje výsledek genotypu, tedy řešení, které může být ohodnoceno,
- *genotyp* (genotype) – reprezentace (kódování) řešení,
- *gen* (gene) – jedna část genotypu,
- *alela* (allele) – hodnota genu,

- *mutace* (mutation) – náhodný operátor, který změní řešení,
- *křížení* (crossover) – binární operátor, který zkombinuje informaci ze dvou rodičů a vyprodukuje nového potomka,
- *rodič* (parent) – jedinec vybraný k produkci dalších jedinců,
- *potomek* (child/offspring) – nový jedinec vytvořený křížením a mutací,
- *populace* (population) – soubor jedinců,
- *generace* (generation) – jedna iterace genetického algoritmu.

Obecné schéma genetického algoritmu je zobrazeno na obrázku 3.3. Nejprve je vytvořena počáteční populace jedinců, kde každý jedinec představuje právě jedno řešení problému zakódované v jeho „genech“. Následně je použita ohodnocující funkce (fitness), která ohodnotí kvalitu každého jedince vzhledem k řešenému problému. Poté se hodnota ohodnocující funkce zohlední při výběru rodičů. Existuje mnoho metod výběru, ale obecně platí, že jedinci představující kvalitnější řešení mají větší šanci být vybráni. U vybraných rodičů je následně provedena rekombinace jejich genů (křížení) a jsou vytvořeni noví potomci. S jistou malou pravděpodobností dochází k mutaci nového jedince, tedy změně náhodně vybraného genu. Noví jedinci následně nahradí část původní populace, případně přibudou ještě naprosto noví jedinci (imigranti). Tímto krokem je vytvořena nová populace a celý proces se může opakovat, dokud není splněna ukončující podmínka. Tou může být celkový počet nových jedinců v průběhu algoritmu nebo zjištění stagnace populace.

Existuje také mnoho dalších rozšíření tohoto základního přístupu a to například o práci s více nezávislými populacemi (island model), pamatování zajímavých jedinců apod. V následujících podkapitolách je popsáno, jak lze využít genetické algoritmy k řešení MRCPSP. V případě řešení pouze RCPSP je postup stejný jako u řešení MRCPSP, ale nemusíme uvažovat módy. Fakta jsou čerpána z [9], [5] a [23].



Obrázek 3.3: Postup provádění genetických algoritmů [27]

3.3.1 Reprezentace jedinců a jejich ohodnocení

Existuje mnoho způsobů reprezentace jedince (rozvrhu) v řešení RCPSP. Jednou z nejpřirozenějších reprezentací je zakódovat jedince jako posloupnost (seznam) naplánovaných operací (activity list). Pokud zároveň uvažujeme o rozšíření o více módů – MRCPSP, tak se nabízí přidat ke každé operaci mód, ve kterém bude prováděna. Jedinec je tedy reprezentován dvojicí $I = (\lambda, \mu)$, kde $\lambda = (j_1, ..., j_J)$ představuje proveditelnou posloupnost operací. Proveditelná je, pokud předchůdci operace jsou v plánu dříve než operace, kterou mají předcházet:

$$\mathcal{P}_{j_i} \subseteq \{j_1, ..., j_{i-1}\} \text{ pro } i = 1, ..., J$$

O přiřazení módu každé operaci se stará mapující funkce $\mu(j) \in \mathcal{M}_j$, která jednoznačně přiřazuje operaci j mód. Jedinec je tedy reprezentován:

$$I = \begin{pmatrix} j_i & \dots & j_J \\ \mu(j_i) & \dots & \mu(j_J) \end{pmatrix}$$

Každý takto reprezentovaný jedinec představuje unikátní rozvrh. Naopak jeden unikátní rozvrh může být reprezentován více různými seznamy operací — to může vzniknout, pokud dvě nezávislé operace začínají ve stejný čas. Vzniká zde tedy redundance. Za povšimnutí také stojí, že v λ není obsažena první ani poslední operace projektu, protože ty by musely být vždy na začátku a konci seznamu operací.

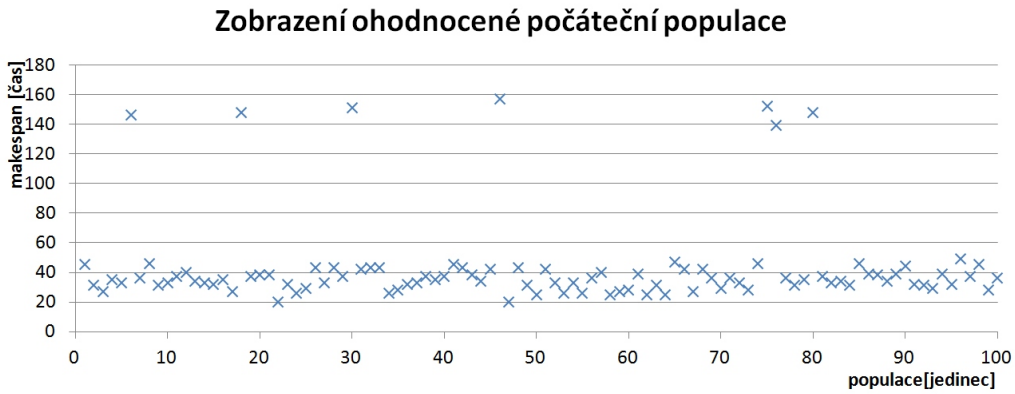
Důležité je si také uvědomit, že tato reprezentace spolu s SGS zaručuje splnění omezujících podmínek popsaných vztahy (2.2) a (2.3) — tedy dodržení předchůdců operací a nepřekročení kapacity u obnovitelných zdrojů. Nezaručuje ovšem splnění podmínky o nepřekročení kapacity u neobnovitelných zdrojů, protože nalezení takového proveditelného plánu není možné provést v polynomiálním čase. Je tedy mnohem lepší zahrnout do populace i jedince, kteří nedodržují toto omezení, a až následně je penalizovat pomocí ohodnocující funkce. Na základě cíle naší optimalizace se provádí ohodnocení jedince. Obvykle se zohledňuje celkový čas projektu, případně ještě zbylá kapacita neobnovitelných zdrojů.

3.3.2 Tvorba počáteční populace

Jedním z nejjednodušších způsobů tvorby počáteční populace je využití SSGS nebo PSGS, přičemž je vždy vybrána náhodná operace z množiny operací připravených k naplánovaní. Zcela náhodný výběr neposkytuje příliš dobré výsledky, proto se upřednostňují operace například s nejmenším nejpozdějším možným koncovým časem operace (LFT). Výpočet těchto časů je vysvětlen v podkapitole 2.3. Další způsoby, jak lze vybírat operace jinak než náhodně, jsou popsány v [14].

Mód operace je následně zvolen také náhodně. Jedince překračující neobnovitelné zdroje lze v dalších generacích omezit penalizací. Další možný způsob je zvolit pořadí operací i módy zcela náhodně a poté se je snažit „opravovat“ za účelem dodržení omezujících podmínek. Dalším přístupem popsaným v [9] může být nejprve zvolení náhodného módu pro každou operaci. Pokud dochází k překročení kapacity neobnovitelných zdrojů, tak se lokální prohledávací funkce snaží tento problém vylepšit. Na závěr je upraveno pořadí operací respektující omezení na předchůdce operací.

V grafu na obrázku 3.4 lze vidět vytvořenou počáteční populaci a její ohodnocení k MRCPS problému `j107_1.mm` z knihovny problémů PSPLIB. Ohodnocením je v tomto případě doba trvání projektu, přičemž plány, které nerespektují neobnovitelné zdroje, jsou penalizovány přičtením konstanty 60 krát počet překročených zdrojů. Populace byla vygenerována s využitím SSGS s náhodnou volbou operací.



Obrázek 3.4: Graf zobrazující ohodnocení počáteční populace

3.3.3 Výběr rodičů

Na výběr rodičů lze použít běžně užívané metody pro řešení obecných problému, tedy MRCPSP řešený GA není v tomto případě nijak výrazně specifický. Obecně platí, že je vhodné upřednostňovat kvalitní jedince, ale zároveň udržovat dostatečně různorodou populaci. Jednou z nejpoužívanějších metod na výběr rodičů je algoritmus rulety (roulette-wheel selection). Ten spočívá v tom, že každému jedinci je proporcionálně přiřazena část výseče rulety vzhledem k jeho fitness funkci. Poté je náhodně vygenerováno číslo, které jakoby představuje „jazýček“ rulety. Vybraný je právě ten jedinec, do jehož rozsahu padlo takto náhodně vygenerované číslo. Proporcionální pravděpodobnost výběru se tedy určí podle vztahu:

$$P_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

Dalším často používaným algoritmem je algoritmus turnaje (tournament selection). Ten spočívá v tom, že se náhodně vybere jistá podmnožina populace a nejlepší jedinec v této populaci je zvolen za rodiče. Pokud by v populaci existoval nějaký výrazně dominantní jedinec, tak by tento přístup mohl zamezit jeho neustálému výběru.

3.3.4 Křížení jedinců

V případě křížení je situace odlišná než u výběru rodičů a je nutné běžně používané operátory křížení upravit. Je tomu z důvodu specifické reprezentace jedince seznamem operací $\lambda = (j_1, ..., j_J)$, tedy nutnosti dodržení předchůdců operací. Pro zjednodušení jsou operátory nejprve vysvětleny na variantě RCPSP, tedy bez rozšíření o více módů. Tento princip lze velmi snadno použít i v MRCPSP pouze s faktem, že se s každou operací j bude přenášet i informace o jejím módu $\mu(j)$.

Nejjednodušším běžně používaným operátorem je operátor jednobodového křížení. Nejprve označíme matku jako M, otce F a budeme uvažovat pouze o jednom nově vzniklém jedinci N. Následně se vybere náhodné číslo q v rozsahu $1 \leq q < J$. Nově vzniklý jedinec bude obsahovat operace na pozicích $i = 1, ..., q$ od matky:

$$j_i^N := j_i^M.$$

Zbytek operací na pozicích $i = q + 1, ..., J$ bude přejat od otce F s faktem, že již použité operace se nebudou znovu opakovat:

$$j_i^N := j_k^F \text{ kde } k \text{ je nejnižší index, takový že } j_k^F \notin \{j_1^N, ..., j_{i-1}^N\}.$$

Tímto vznikne nový jedinec, který respektuje předchůdce každé operace. Nyní je níže uveden příklad tohoto operátoru při zvoleném $q = 3$.

$$M = (1, 3, 2, 5, 4, 6),\ F = (2, 4, 6, 1, 3, 5),\ N = (1, 3, 2, 4, 6, 5)$$

Dalším běžně používaným operátorem je dvoubodové křížení, což je rozšíření výše uvedeného operátoru. Jak již název napovídá, jsou celkem použita dvě náhodně zvolená čísla $1 \leq q_1 < q_2 < J$. Ta rozdělí seznam operací na celkem tři části, přičemž první část do nového jedince je opět použita od matky M, druhá opět od otce F a třetí zase od matky M. Opět platí pravidlo, že se operace nesmí opakovat.

Dalším operátorem křížení je uniformní křížení. V tomto případě se nejprve pro každou operaci vygeneruje náhodné číslo $p_i \in \{0, 1\},\ i = 1, ..., J$. Pokud má toto náhodné číslo hodnotu $p_i = 1$, tak je na indexu i použit gen od matky M:

$$j_i^N := j_k^M \text{ kde } k \text{ je nejnižší index, takový že } j_k^M \notin \{j_1^N, ..., j_{i-1}^N\}.$$

Jinak řečeno, taková operace, která není ještě použita v novém jedinci. Stejným způsobem, pokud má náhodné číslo hodnotu $p_i = 0$, tak je na indexu i použit gen od otce F:

$$j_i^N := j_k^F \text{ kde } k \text{ je nejnižší index, takový že } j_k^F \notin \{j_1^N, ..., j_{i-1}^N\}.$$

Praktické použití tohoto operátoru lze pro náhodně vygenerovaná čísla $(0, 1, 1, 0, 1, 1)$ vidět na příkladu níže [8].

$$M = (1, 3, 2, 5, 4, 6),\ F = (2, 4, 6, 1, 3, 5),\ N = (2, 1, 3, 4, 5, 6)$$

Jak již bylo zmíněno, v případě MRCPSP lze použít také tyto metody se skutečností, že se bude přenášet také informace o použitém módu u každé operace. Jiný způsob popsaný v [9] je vygenerování dvou náhodných čísel $1 \leq q_1, q_2 \leq J$. Přičemž první z nich bude sloužit k jednobodovému křížení stejně, jak bylo popsáno výše. Druhé z nich q_2 bude sloužit v podstatě také k jednobodovému křížení, ale u módů. Tedy módy operací na pozicích $i = 1, ..., q_2$ budou použity od matky M:

$$\mu^N(j_i^N) := \mu^M(j_i^N).$$

A módy na zbývajících pozicích $i = q_2 + 1, ..., J$ jsou použity od otce F:

$$\mu^N(j_i^N) := \mu^F(j_i^N).$$

3.3.5 Mutace jedinců

S jistou malou pravděpodobností dochází ke změně náhodně vybraného genu u náhodně zvoleného jedince. Tento proces se nazývá mutace, která dokáže vytvořit i takové varianty, které nelze vytvořit pouze pomocí operátoru křížení. Princip mutace je, že operace j_i na pozici $i = 1, ..., J - 1$ je zaměněna s j_{i+1} s pravděpodobností p_{mutace}. V případě varianty MRCPSP je mód operace přesunut také. Následně je operátor mutace aplikován i na mód operace a to způsobem, že je pro náhodně zvolenou operaci j_i na pozici $i = 1, ..., J$ vybrán náhodně jiný patřičný mód $\mu(j_i) \in \mathcal{M}_{j_i}$.

3.3.6 Tvorba nové populace

Opět existuje mnoho způsobů, jak začlenit nové jedince do stávající populace. Obvykle má populace pevnou velikost během celé doby provádění algoritmu, proto je nutné některé jedince ze stávající populace nahradit. Jednou z nejpoužívanějších metod je seřazení jedinců podle výsledků ohodnocující funkce a nahrazení nejhůře ohodnocených jedinců novými jedinci.

Kapitola 4

Základní optimalizátor výrobních operací

V kapitole 2.1 byl podrobně popsán RCPSP a v předchozí kapitole možnosti řešení tohoto problému. Tato kapitola vychází z předchozích dvou kapitol a aplikuje popsané myšlenky do základního optimalizátoru výrobních operací, který neuvažuje poruchy ve výrobě. Na začátku této kapitoly jsou nejprve popsány uvažované praktické rozšíření základního RCPSP. Následně je popsán nový zápis RCPSP včetně nově uvažovaných rozšíření v XML. Další podkapitola je věnována implementaci základního optimalizátoru výrobních operací. Závěr kapitoly za zabývá možností praktického nasazení základního optimalizátoru.

4.1 Rozšíření RCPSP

Standardní MRCPSP i RCPSP jsou dostatečně robustní a nalézají uplatnění jako základní model v mnoha praktických situacích. Ovšem RCPSP nedokáže pokrýt všechny praktické požadavky, a proto je nutné tento základní model rozšířit o praktické prvky, které vždy reflektují daný reálný problém. V následujících podkapitolách jsou popsány jednotlivé implementované praktické rozšíření RCPSP. Dále již není uvažováno rozšíření o více módů MRCPSP. Základní možnosti přístupu byly čerpány z článku [10].

4.1.1 Zarovnávání operací v rozvrhu

Při modelování nějaké skutečné průmyslové výroby se nejprve musíme rozhodnout, zda chceme operace v rozvrhu zarovnávat doprava nebo doleva. Výhodou zarovnávání operací doprava (right-justified schedules) je, že operace jsou vykonávány právě na čas (JIT – Just in time) a není potřeba skladovat mnoho výrobků apod. Je to za cenu toho, že musí být zajištěna vysoká spolehlivost celého procesu, aby docházelo k dodržování požadovaných termínů [12]. Pokud firma vyrábí mírně na sklad a až například další týden dodává výrobky zákazníkům, tak postačí zarovnávat operace doleva, tak jak bylo dosud uvažováno. Tento přístup je také zvolen v této práci.

4.1.2 Proměnlivá kapacita zdroje v čase

Standardní RCPSP model předpokládá, že zdroj má stejnou kapacitu po celou dobu trvání. V praxi ovšem často existují technologická a provozní omezení, která znemožňují nepřetržitý

provoz stroje. Z technologických omezení je to například plánovaná údržba, odstávka a podobně. Dále například nemusí být na stroji třísměnný provoz.

Pokud bychom chtěli toto rozšíření implementovat do algoritmu vycházejícího ze SSGS, tak se jako nejjednodušší způsob nabízí vložit při startu algoritmu „falešné“ operace reprezentující dobu, po kterou stroj není v provozu. Následně se tato skutečnost promítne ve vektorech zbývajících kapacit $\tilde{R}_k(t)$ a operace se tedy nebudou na daný čas plánovat, protože nemají dostatečnou kapacitu na své provedení.

V případě implementace tohoto rozšíření do algoritmu vycházejícího z PSGS je situace poněkud složitější. Algoritmus nemá ze své podstaty potřebu kontrolovat dostatek zdrojů pro danou operaci po celou dobu jejího trvání, protože stačí tuto kontrolu provést pouze při startu operace. Je tomu tak z důvodu, že algoritmus postupuje po čase a v následujících časech nemůže být zbývající kapacita zdrojů nižší, než je ta aktuální. V případě uvažování stejného principu jako u SSGS (promítnutí provozní doby stroje do vektoru zbývajících kapacit $\tilde{R}_k(t)$) musíme zjišťovat, zda má operace dostatečnou kapacitu po celou dobu jejího trvání. Toho lze dosáhnout stejným způsobem, jaký je implementován v SSGS. V tabulce 4.1 jsou uvedeny zbývající kapacity zdrojů popsané $\tilde{R}_k(t)$. Jedná se o případ dvou zdrojů s kapacitou $R_1^\rho = 7$ a $R_2^\rho = 4$ a nefunkčností zdroje 1 v čase od 3 do 6 a stroje 2 od 4 do 10. S hodnotami uvedenými v $\tilde{R}_k(t)$ by tedy algoritmus začínal svou činnost.

t	0	3	4	6	10
$\tilde{R}_1(t)$	7	0	0	7	7
$\tilde{R}_2(t)$	4	4	0	0	4

Tabulka 4.1: Počáteční zbývající kapacita na strojích

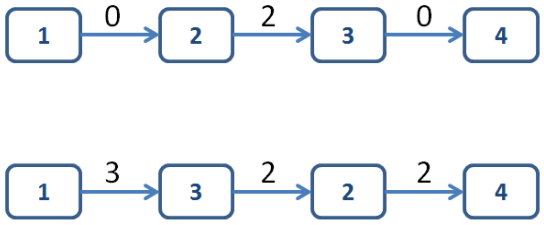
4.1.3 Instalační doba operace

Standardní RCPSP explicitně neuvažuje instalační dobu operace na stroj. Nepřímo toho lze dosáhnout přičtením instalační doby k době provádění operace. Problémem ovšem je, že tato doba často závisí například na předchozí prováděné operaci na stroji, protože je například nutné vyměnit nástroj a podobně. Instalační doba operace může být sekvenčně závislá na všech předchozích operacích na stroji, pouze na poslední operaci nebo pokud uvažujeme RCPSP s více módy pouze na stroji. Celková doba trvání operace se tedy určí jako součet zadané doby trvání operace a instalační doby.

V této práci je uvažována varianta, že instalační doba závisí pouze na předchozí operaci na stroji. Je tedy nutné pro každou operaci definovat vektor instalačních časů po každé operaci. Jednoduchá ukázka, jak se projeví různá sekvence provádění operací na stroji, je zobrazena na obrázku 4.1. V případě provádění operací na stroji v pořadí $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ je nutné uvažovat instalační dobu s hodnotou 2 pouze při přechodu mezi operacemi $2 \rightarrow 3$. Naopak, pokud bychom prováděli operace na stroji v nevhodném pořadí $1 \rightarrow 3 \rightarrow 2 \rightarrow 4$, tak je často nutné měnit například lisovací nástroje, což způsobí zbytečné plýtvání času.

Implementace tohoto rozšíření je v případě PSGS poměrně jednoduchá, neboť si stačí pamatovat vždy poslední operaci, která byla prováděna na daném stroji. Následně je pro plánovanou operaci ve vektoru instalačních časů nalezena její instalační doba vzhledem k předchozí operaci.

V případě implementace tohoto rozšíření do algoritmu založeném na SSGS je situace mnohem komplikovanější. Problémem je, že plánování operací se neprovádí postupně v čase,



j	1	2	3	4
1	-	0	3	0
2	0	-	2	2
3	3	2	-	0
4	0	2	0	-

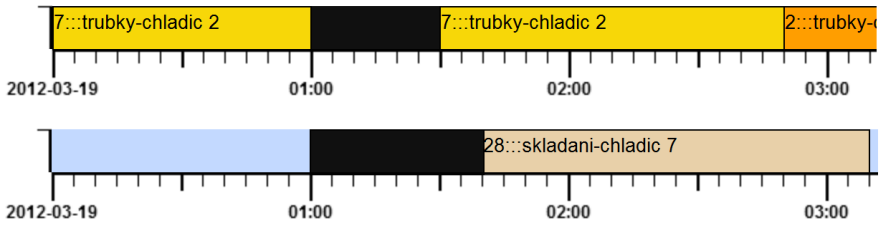
Obrázek 4.1: Sekvenčně závislá instalační doba operací

ale běžně nastává, že se nějaká operace naplánuje časově před již naplánovanou operaci. To znemožňuje určení instalační doby na základě předchozí operace, neboť ji v daném momentě ještě neznáme. Jedním z řešení by mohlo být uvažovat instalační doby až v dalším průchodu.

4.1.4 Přerušitelnost operace

Standardní RCPSP předpokládá, že operaci po jejím zahájení není možno přerušit. V praxi ovšem může být přerušení operace výhodné, protože výsledný rozvrh poté lépe využívá čas na strojích a omezuje vznik nevyužitých časových úseků na strojích. Neomezené přerušování výrobní operace ovšem není v praxi příliš žádoucí, neboť zahájení a přerušení výrobní operace obvykle spotřebuje určitý čas. Z tohoto důvodu se často uvažuje nejkratší možná doba provádění operace, aby mohlo nastat její přerušení. Dále je také možno definovat maximální možný počet přerušení pro danou operaci. Více možných přístupů lze nalézt v [10].

V této práci je uvažováno přerušení operace převážně v souvislosti s plánovanou nefunkčností stroje. Je tedy možno pro každou operaci definovat, zda si přejeme, aby byla zahájena i s vědomím, že v průběhu jejího provádění bude přerušena například z důvodu prováděné údržby na stroji. Také lze definovat, že si operaci nepřejeme přerušovat. Ukázka přerušitelné operace je na obrázku 4.2 nahoře a dole je ukázka nepřerušitelné operace. Černé místo znamená plánovanou údržbu na stroji.



Obrázek 4.2: Ukázka přerušitelné a nepřerušitelné výrobní operace

V případě implementace této vlastnosti do SGS je nutné upravit rozhodování, zda je operace zdrojově proveditelná a dále je nutné upravit výpočet zbývajících kapacit zdrojů $\tilde{R}_k(t)$ po naplánování operace. Pokud uvažujeme PSGS a plánovou údržbu strojů 4.1.2, tak pro určení množiny proveditelných operací $\mathcal{D}_g$ bude záležet na skutečnosti, zda je operace přerušitelná nebo ne. V případě, že je přerušitelná, tak dostačuje, aby měla výrobní operace dostatečnou kapacitu na zdrojích pouze při svém zahájení. Pokud dojde v průběhu jejího

provádění k přerušení, tak se musí konec operace upravit tak, aby byla operace provedena celá. Jinak řečeno, aby požadovaná doba trvání operace odpovídala skutečně prováděnému času operace. V případě, že operace není přerušitelná, tak platí skutečnost, že operace musí mít dostatek zdrojů po celou dobu svého trvání, aby mohla být zařazena do $\mathcal{D}_g$.

4.1.5 Prioritní operace

Při standardní činnosti optimalizátoru dochází k výběru operací v takovém pořadí, aby bylo co nejlépe naplněno optimalizační kritérium. V praxi ovšem často požadujeme upřednostňovat některé výrobní operace, protože mohou být například součástí důležité prioritní zakázky. Pokud bychom ovšem většině operací nadefinovali jejich různé priority a optimalizátor by je musel striktně dodržovat, tak nastává situace, že optimalizátor je de facto zbytečný. Je tomu tak z důvodu, že již uživatel si nadefinovat pořadí provádění jednotlivých operací a optimalizátor je může měnit buď minimálně, nebo vůbec.

Do základního algoritmu SGS lze striktní dodržování priorit operací zapracovat poměrně jednoduše. Pokud existuje v množině operací připravených k provedení $\mathcal{D}_g$ nějaká operace, která má vyšší prioritu než ostatní, tak je vybrána právě ona.

4.1.6 Zavedení reálného času

Ve standardním RCPSP doba trvání operace p_j nemá žádnou konkrétní časovou jednotku. Při praktickém použití optimalizátoru je ovšem vyžadováno, aby vypočtené časy začátků operací odpovídaly konkrétnímu datu a času. Řešení je v tomto případě poměrně jednoduché. Nejprve je v této práci za základní časovou jednotku celého RCPSP prohlášena minuta, protože pro potřeby rozvrhování výrobních operací v průmyslové výrobě je dostatečně přesná. Dále musí být RCPSP rozšířen o konkrétní datum a čas, který představuje start celého projektu a odpovídá tedy startu první operace $j = 0$ v čase $t = 0$. Poté je možno mapovat vypočtené relativní časy startů na konkrétní datum a čas. Formát vstupního i výstupního zápisu data a času je následující: 2012-03-19 12:35.

4.2 Zápis RCPSP a jeho rozšíření v XML

V první implementované verzi umožňoval optimalizátor řešit pouze RCPSP a MRCPSP popsané v textovém souboru ve formátu, který odpovídá problémům z PSPLIB. Zde ovšem není možno popsat uvažované praktické rozšíření. Z tohoto důvodu byl navržen vlastní zápis RCPSP v XML a jeho základní rysy jsou zde vysvětleny. V zápisu je také uvažována verze s více módy provádění a s neobnovitelnými zdroji.

Kořenový uzel celého problému je definován následovně.

```
<rcpsp name="case study" jobs="33" resources="11" materials="1" orders="9"
    start="2012-03-19 6:00" />
```

Problém obsahuje 31 reálných výrobních operací, přičemž dvě navíc představují začátek a konec projektu. Tyto operace lze vykonávat na celkem jedenácti strojích a jsou vázány na jednu z devíti zakázek (výrobků). Rozvrh je plánován od 19. března 2012 od 6:00.

V následujícím zápisu je uveden způsob definování obnovitelných a neobnovitelných zdrojů. První uvedený uzel obsahuje stroje a druhý materiál. U strojů je možno také nadefinovat dobu, po kterou nejsou v provozu — uzel `interruption`.

```
<res_availabilities>
  <resource id="3" name="pájecí pec" capacity="1" >
```

```
    <interruption name="pravidelná údržba"  from="2012-03-19 12:00"
       to="2012-03-19 12:30" />
  </resource>
  <resource>...</resource>
</res_availabilities>

<mat_availabilities>
  <material id="0" name="plechy" capacity="100" />
</mat_availabilities>
```

Následně je v zápisu uveden seznam všech zakázek.

```
<orders>
  <order id="0" name="n/a" />
  <order id="1" name="chladič A" />
  ...
</orders>
```

Poté jsou definováni pro každou operaci její následovníci – `successor`. Také je uvedeno, co operace představuje – `name`, k jaké zakázce je vázána – `order` a v kolika módech ji lze provádět – `modes`.

```
<precedences>
  <job id="0" name="dummy start" order="0" modes="1">
    <successor id="1" />
    <successor id="2" />
    ...
  </job>
  <job id="1" name="lisování" order="1" modes="1">
    <successor id="3" />
  </job>
```

V poslední části popisu je pro každou operaci a její určitý mód definován požadavek na zdroje – `resources`, materiál – `materials`, instalační dobu před určitou operací – `setups`, dobu trvání – `duration`, povolení přerušení – `preemptive` a prioritu – `priority`.

```
<requests>
  <job id="0">
    <mode id="0" duration="0:00" preemptive="false" >
      <resources>
      </resources>
      </mode>
  </job>

  <job id="1">
    <mode id="0" duration="2:20" preemptive="false" priority="10">
      <resources>
        <res id="0" request="1" />
      </resources>
      <materials>
        <mat id="0" request="30" />
      </materials>
      <setups>
        <before job="0" time="1:20" />
      </setups>
      </mode>
  </job>
</requests>
```

4.3 Implementace optimalizátoru

Základní optimalizátor výrobních operací je kompletně implementován v jazyce C++. Při svém spuštění očekává soubor s RCPSP ve formátu odpovídající problému z PSPLIB nebo ve vlastním navrženém formátu XML. Výstupem je soubor XML obsahující nejlepší plán a také XML soubor, který obsahuje startovní časy všech operací odpovídající zmíněnému plánu. Pro práci s XML byla využita v celém projektu knihovna TinyXML[1]. Po načtení je problém reprezentován objektem vytvořeným ze třídy `Mrcpsp`. Následně je nad tímto problémem zavolána metoda `Solve()` ze třídy `GaSolver` reprezentující genetické algoritmy.

Existuje celá řada knihoven, které dokážou řešit genetické algoritmy. Pro C++ je jednou z nejznámějších knihovna GAlib[2]. Knihovna obsahuje celou řadu běžně používaných operátorů a metod výběru jedinců. Také nabízí podrobné statistiky z průběhu genetického algoritmu. Ovšem, jak již bylo popsáno, na řešení RCPSP je nutné specifikovat vlastní operátory křížení, mutace a používat specifický způsob reprezentace jedince. To GAlib umožňuje, ale pokud zvážíme výhody, které nám při implementaci ulehčí (výběr rodičů, začlenění nových jedinců do populace) a jejich náročnost vlastní implementace, tak se výhody použití této knihovny ztrácejí. Z tohoto důvodu byl implementován vlastní genetický algoritmus dle popisu z kapitoly 3.3.

Třída `GaSolver` obsahuje privátní metodu `Crossover()` implementovanou pomocí jednobodového křížení a privátní metodu `SelectParents()` implementovanou pomocí algoritmu rulety. Jedinec je reprezentován jako objekt vytvořený ze třídy `Individual`. Tato třída obsahuje dynamicky alokované pole pro uložení seznamu operací (activity list). Pro vytvoření počáteční populace a ohodnocení potomků byly implementovány ve třídě `Scheduling` obě verze popsaných algoritmů — SSGS z podkapitoly 3.2.1 a PSGS z podkapitoly 3.2.2.

Pro implementaci všech uvažovaných rozšíření popsaných v podkapitole 4.1 bylo nutné převážně upravit algoritmus SGS a vhodně reprezentovat nové uvažované vlastnosti ve třídě `Mrcpsp`. Implementovaný genetický algoritmus není potřeba dále rozšiřovat. Při návrhu algoritmu vycházejícího ze SGS jsem nejprve musel rozhodnout, zda je výhodnější postupovat po čase (PSGS) nebo po operacích (SSGS). V publikovaném článku [14] je experimentálně dokázáno, že SSGS v kombinaci s implementovaným LFT poskytuje lepší výsledky u problémů s počtem operací 30. Naopak u problémů s vyšším počtem operací vykazuje lepší hodnoty PSGS. Proměnlivá kapacita zdrojů v čase by se lépe implementovala do SSGS než do PSGS. Naopak závislost instalační doby na předchozí operaci by se do SSGS implementovala velmi obtížně. Z těchto všech důvodů jsem se rozhodl vyjít z algoritmu PSGS a provádět zjišťování zdrojově proveditelných operací podobným způsobem jako v SSGS.

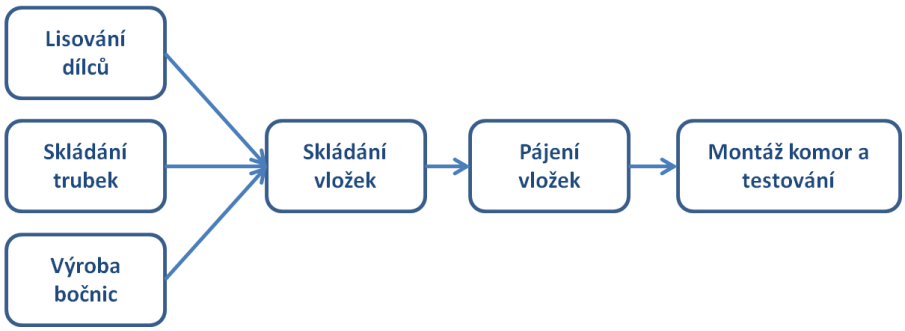
4.4 Nasazení optimalizátoru v průmyslové výrobě

Tuto práci jsem měl možnost konzultovat s firmou Visteon-Autopal, s.r.o. sídlící v Hluku. Tato firma svým počtem zaměstnanců patří mezi velké. Při konzultacích v této firmě jsme probírali případné praktické nasazení základního optimalizátoru v průmyslové výrobě. V této kapitole jsou popsány základní myšlenky, jak by bylo možno vytvořený optimalizátor uvést do praxe.

[1] http://www.grinninglizard.com/tinyxml/
[2] http://lancet.mit.edu/ga/

4.4.1 Základní popis výroby

Firma ve svém závodě vyrábí různé typy chladičů pro automobilový průmysl. Výroba každého chladiče se skládá z několika výrobních operací, které jsou obvykle velmi podobné pro různé chladiče. Liší se obvykle pouze strojem, na kterém má být operace provedena. Příklad takového technologického postupu je na obrázku 4.3.



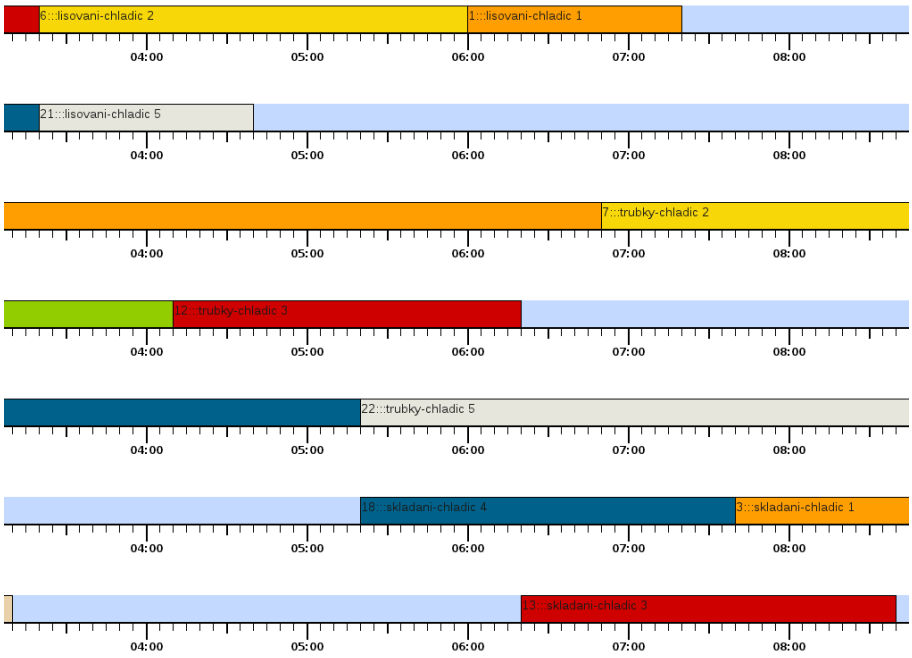
Obrázek 4.3: Základní technologický postup při výrobě chladiče

4.4.2 Propojení s podnikovým informačním systémem

Optimalizátor sám o sobě je v podstatě připraven k optimalizaci výrobních operací v dané firmě. Ovšem popis problému ve formátu XML, který je vysvětlen v podkapitole 4.2, je pro běžně užívání nepoužitelný. Vyžadovalo by to totiž neefektivní zdlouhavý zápis mnoha operací. Řešením je tento soubor generovat nebo rozšířit optimalizátor o možnost načítání požadovaných dat z jiných zdrojů (databáze).

Obrázek 4.4: Možnost generování RCPSP

Na obrázku 4.4 je naznačen postup, jak by mohlo probíhat vygenerování požadovaného XML souboru s popisem RCPSP. V podnikovém informačním systému (ERP) se obvykle hromadí požadavky zákazníků na výrobu. Tyto systémy obvykle umožňují export těchto informací do běžně používaných formátů. Tento soubor, který by obsahoval seznam výrobků a jejich požadovaný počet, by byl jedním ze vstupů do uvažovaného generátoru. Dále musí generátor znát pro každý výrobek jeho přesný technologický postup. Musí tedy existovat seznam výrobních operací a jejich závislostí, které je nutno provést pro výrobu daného produktu. U každé operace musí být také jasně definován stroj, na kterém má být operace provedena a odhadovaná výrobní doba pro určitý počet kusů. Databáze by také měla obsahovat případné instalační doby operací. Dále musí mít generátor k dispozici seznam strojů a jejich provozní doby. Program by také měl uživateli umožňovat měnit jednotlivé položky ve vygenerovaném RCPSP a definovat případné priority. Po provedení optimalizace



Obrázek 4.5: Příklad vygenerované části Ganttova diagramu

by výstupem byl pravděpodobně Ganttův diagram, ze kterého lze snadno vyčíst, kdy má být jaká operace zahájena. Příklad tohoto diagramu je zobrazen na obrázku 4.5.

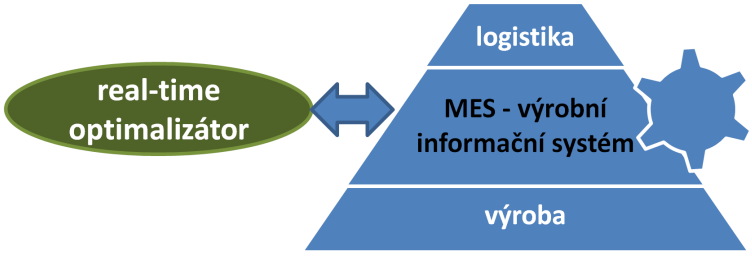
Kapitola 5

Real-time optimalizátor výrobních operací

V předchozích kapitolách byl podrobně popsán způsob řešení RCPSP a možnosti jeho použití při rozvrhování výrobních operací. Tento základní optimalizátor ovšem předpokládá přesnou znalost všech požadovaných údajů a také předpokládá vykonávání výrobních operací ve statickém, deterministickém prostředí. Na skutečnou výrobu je ovšem nutné nahlížet jako na dynamické prostředí, a proto je výhodné mít SW nástroj, který by dokázal reagovat na aktuální změny ve výrobě. Tímto nástrojem je real-time optimalizátor, který je popsán v této kapitole. Jeho přínos a začlenění do výrobního informačního systému je uveden na začátku této kapitoly. Poté jsou popsány všechny uvažované neočekávané události ve výrobě ve formě modelu poruch. Následuje kapitola, která analyzuje možnosti přístupu k dynamickému přeplánování. Na závěr této kapitoly je popsán návrh a implementace real-time optimalizátoru.

5.1 Přínos real-time optimalizátoru

V předchozí kapitole byl popsán základní optimalizátor, který umožňuje vytvořit efektivní rozvrh výroby. Tento rozvrh je ovšem do jisté míry pouze teoretickým, neboť při vykonávání výrobních operací často dochází k neočekávaným událostem. Mezi ně lze zařadit například poruchu stroje, novou výrobní operaci, změnu požadovaného množství výrobků a podobně. Tyto neočekávané události obvykle způsobí, že se původní rozvrh výroby stává neproveditelným a je nutné jej upravit pomocí real-time optimalizátoru.



Obrázek 5.1: Začlenění real-time optimalizátoru a MES mezi ostatní

V reálném průmyslovém nasazení by existoval výrobní informační systém (MES), který by získával aktuální informace z terminálů umístěných ve výrobě a z logistiky. Tento systém

má tedy k dispozici aktuální stav výroby, což je předpoklad pro správné fungování real-time optimalizátoru, který by s tímto systémem spolupracoval. V případě narušení rozvrhu by MES předal aktuální stav výroby real-time optimalizátoru k přeplánování. Jejich postavení v rámci řízení výroby je zobrazeno na obrázku 5.1. Cílem real-time optimalizátoru je tedy v krátkém čase vytvořit nový kvalitní rozvrh, který je podobný tomu původnímu. Celá tato kapitola se tedy zabývá dynamickými změnami do výrobního rozvrhu vytvořeného základním optimalizátorem.

5.2 Model poruch

Tato podkapitola je zaměřena na detailní vysvětlení poruch, které mohou nastat ve výrobě. Pokud bychom požadovali vyjmenovat kompletní seznam všech poruch v rámci RCPSP, tak by se jednalo o možnost editace všech parametrů tohoto problému — tedy přidání, odstranění a modifikace zdrojů a operací. V praxi jsou ovšem některé poruchy méně časté a významně upravují celý model výroby. Jedná se například o přidání nového stroje (zdroje) nebo úpravu technologického postupu (změna relací předchůdce). V těchto případech by bylo výhodnější vytvořit nový rozvrh pomocí základního optimalizátoru, a proto nejsou tyto poruchy uvažovány mezi dynamickými. Navržený model poruch popsaný XML pokrývá následující události:

- přidání nové operace — porucha `newjob`,
- modifikace operace — porucha `job` na změnu doby trvání,
- odstranění operace — pouze nepřímo nastavením doby trvání na 0,
- modifikace zdroje — porucha `resource` na změnu kapacity v čase,
- odstranění zdroje — pouze nepřímo nastavením kapacity 0 po celou dobu.

5.2.1 Změna doby trvání operace

Základní optimalizátor výrobních operací při výpočtu rozvrhu výroby počítal se zadanou dobou trvání jednotlivých operací. Postupem času se ovšem může stát, že původně odhadovaná doba trvání operace již není platná. Může tomu tak být například z důvodu, že zákazník požaduje jiný počet finálních výrobků, a proto se i patřičně změní doba trvání jednotlivých výrobních operací. Také vlivem nějakého technologického problému může dojít ke zpomalení výroby na určitém stroji, které způsobí delší dobu provádění výrobní operace. V tomto výčtu možných příčin změny doby trvání operace by se dalo pokračovat.

Nejjednodušším možným způsobem lze tuto změnu popsat následovně:

```
<failures problemID="XXYY" >
  <job id="10" mode="absolute" time="16" />
</failures>
```

Jak je ze zápisu patrné, jedná se o nastavení nové doby trvání na hodnotu 16 u operace číslo 10. Postihnutou operaci lze specifikovat třemi způsoby:

- číslem, které odpovídá požadované operaci,
- znakem `?`, který znamená náhodný výběr jedné operace,
- znakem `*`, který znamená aplikaci poruchy na všechny operace.

Také je možno změnu doby trvání operace popsat relativně vzhledem k její původní době, jak je ukázáno níže.

```
<failures problemID="XXYY" >
  <job id="?" mode="relative">
    <percent val="150" />
  </job>
</failures>
```

V tomto případě by se doba trvání náhodně zvolené operace zvýšila o 50 %. V případě potřeby by se dal zápis rozšířit o specifikaci doby určitým pravděpodobnostním rozložením. Pokud je uvažován model s reálným časem, tak je nutné tuto dobu trvání zapsat ve formátu `hh:mm`, kde první číslo znamená hodinu a druhé minutu.

5.2.2 Nefunkčnost stroje

V reálné průmyslové výrobě se každý stroj vyznačuje určitou mírou poruchovosti. U lisovacích strojů může dojít například k poškození lisovacího nástroje nebo obecně k poruše například na elektrickém zařízení stroje. Dále například v určité části výroby může dojít k výpadku proudu a podobně. Všechny tyto neočekávané události způsobí nefunkčnost stroje a naplánované operace na nich nelze provést. V následující ukázce je popsán příklad takové poruchy.

```
<failures problemID="XXYY" >
  <resource id="2" from="2012-03-19 1:00" to="2012-03-19 4:00" />
</failures>
```

Opět, jako v případě specifikace operace, lze zdroj specifikovat stejnými třemi způsoby. V uvedeném příkladě by se tedy jednalo o nefunkčnost stroje číslo 2 v čase od 1:00 do 4:00 dne 19. března 2012. V případě neuvažování reálného času lze zapsat dobu poruchy pouze číselnými hodnotami.

5.2.3 Nová operace

Lidé z logistiky musí obvykle přizpůsobovat výrobu aktuálním požadavkům zákazníka. Tímto vzniká často potřeba vykonat nějakou předem neplánovanou výrobní operaci. Tuto operaci je mnohdy požadováno vykonat i s vyšší prioritou.

```
<failures problemID="XXYY" >
  <newjob reporttime="20" duration="10" resource="1" request="2" priority="5"
     />
</failures>
```

Výše uvedená ukázka demonstruje příklad zápisu nové výrobní operace. Tato operace byla hlášena v čase 20 a požaduje kapacitu 2 na zdroji číslo 1 po dobu 10. Priorita této nové operace je 5. V případě RCPSP s reálným časem lze tento čas zapsat stejným způsobem jako v předchozích případech.

5.3 Analýza možností řešení dynamických změn ve výrobě

Z obecného pohledu lze k dynamickým změnám přistupovat dvěma způsoby. První možností je zaujmout aktivnější přístup a vytvářet robustní rozvrh výroby, který již anticipuje s případnými budoucími poruchami. Jedním ze způsobů, jak toho lze docílit je, že nebudeme uvažovat předem známou konstantní dobu trvání operací. Doba trvání jednotlivých

operací bude tedy popsána pomocí určitého pravděpodobnostního rozložení. Tímto vznikne problém, který je v literatuře označován jakožto „stochastic RCPSP“ (SRCPSP) [2]. Další možností aktivnějšího přístupu je vytvořit prvotní rozvrh běžným způsobem a následně se jej snažit učinit robustnějším vkládáním časových prodlev [26]. Obecně ovšem není možno vytvořit natolik robustní rozvrh, který by byl odolný vůči všem poruchám. Je tedy nutné tento aktivnější přístup zkombinovat s pasivnějším (reaktivním) přístupem, který je popsán v dalších odstavcích.

Druhou možností je tedy zaujmout pasivnější přístup a reagovat na problém až v okamžiku jeho vzniku. Jednou z možností je provést přeplánování pomocí metod hledajících přesné řešení. Tento přístup je ovšem velmi časově náročný i pro poměrně malé problémy [1].

Dalším možným přístupem popsaným v článku [25] je využít více možných způsobů reprezentace rozvrhu (priority lists) a různých typů algoritmů SGS. Ve zmiňovaném článku je kromě algoritmů PSGS a SSGS, které jsou popsány v této práci, uvažována jejich tzv. robustní verze. Ta spočívá v případě PSGS v myšlence, že v každém rozhodovacím čase t mohou být naplánovány pouze ty operace, které měly svůj původní start plánovány na dřívější nebo stejný čas jako čas t. V případě robustního SSGS myšlenka spočívá v umísťování operací do rozvrhu na proveditelné místo, které je svým časem začátku operace co nejvíce podobné původnímu plánovanému času. Uvažovanými způsoby reprezentace jsou různé prioritní listy — s náhodnými prioritami, s prioritou nejpozdějšího možného startu operace (LST) a podobně. Jakmile nastane potřeba provést přeplánování, tak jsou propočítány všechny uvedené kombinace SGS algoritmů se zmíněnými způsoby reprezentace rozvrhu. Následně je vybrán takový rozvrh, který má nejmenší odchylku startovních časů jednotlivých operací od původně plánovaných startovních časů. Dle prezentovaných informací v uvedeném článku nabízí tento přístup poměrně kvalitní nový rozvrh, který je podobný původnímu a je sestaven v krátkém výpočetním čase.

Další možností pasivnějšího přístupu je provést přeplánování podobným způsobem jako výpočet základního rozvrhu. V případě této práce by se tedy jednalo o genetické algoritmy. Případně lze určit operace postihnuté danou poruchou a pouze je se snažit přeplánovat.

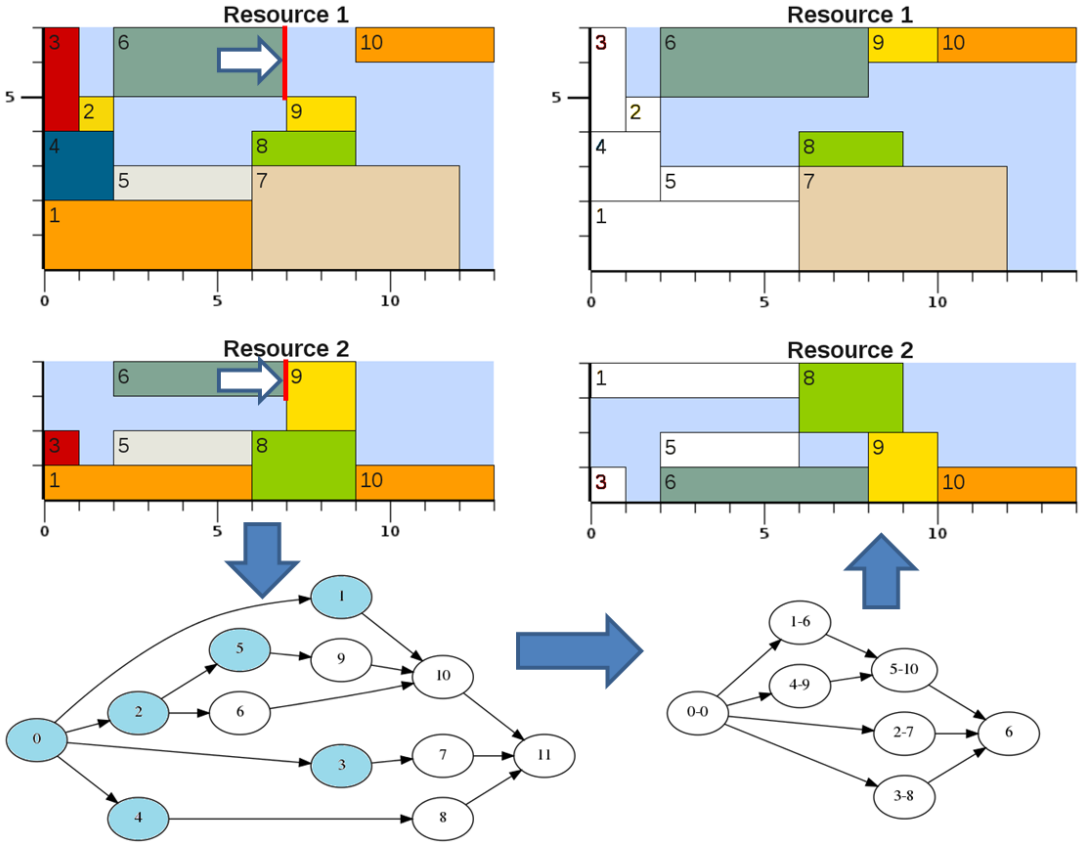
5.4 Návrh real-time optimalizátoru

V případě potřeby provést přeplánování je nejprve nutné vytvořit aktuální model popsaný RCPSP. Aktuální RCPSP lze sestavit z původního RCPSP a aktuálního stavu výroby. Příklad činnosti real-time optimalizátoru a vytvoření aktuálního RCPSP je zobrazen na obrázku 5.2. Jedná se opět o zadání problému z podkapitoly 2.1. Základní optimalizátor vytvořil neoptimální rozvrh s celkovým časem 13. Doba trvání operace číslo 6 se zvýšila z doby 5 na dobu 6. To zapříčiní, že v čase 7 vzniká narušení původního rozvrhu výroby, neboť operace 9 nemá dostatek zdrojů na své zahájení. Aktuální stav v čase 7 je charakterizován následovně:

- zbývající kapacitou na zdrojích (zdroj 1 – 1, zdroj 2 – 1),
- operacemi, které právě probíhají $\{6, 7, 8\}$,
- operacemi, které již byly vykonány $\{0, 1, 2, 3, 4, 5\}$,
- operacemi, které jsou připraveny na své zahájení $\{9\}$,
- operacemi, které nejsou připraveny na své zahájení $\{10, 11\}$,

- operacemi, jejichž provádění bylo přerušeno {}.

Postihnutými operacemi jsou prohlášeny všechny právě vykonávané operace a všechny operace, které ještě nebyly zahájeny. Následně je z postihnutých operací sestaven aktuální RCPSP na základě původního problému. U právě probíhajících operací je nutno zkrátit novou dobu trvání podle již provedeného času. V uvažovaném příkladě na obrázku 5.2 je aktuální RCPSP zobrazen vpravo dole. Tento problém již obsahuje pouze operace $\{6, 7, 8, 9, 10, 11\}$, které jsou v uzlu uvedeny jako druhé číslo. První číslo značí nové číslo operace. Nad tímto aktuálním RCPSP je následně provedeno přeplánování a jsou určeny nové startovní časy jednotlivých operací. Přeplánovaný rozvrh s celkovým časem 14 lze vidět na zmiňovaném obrázku vpravo nahoře.



Obrázek 5.2: Příklad vytvoření aktuálního RCPSP v čase 7

V podkapitole 5.3 byly rozebrány možnosti řešení samotného přeplánování. V této práci není uvažován aktivní přístup k přeplánování, protože vyžaduje věrohodnou znalost poruchovostí jednotlivých prvků ve výrobě. Tyto informace je obvykle náročné získat a také by případně nepokryly všechny možné neočekávané problémy. Tato práce uvažuje tedy pouze pasivní přístup a na neočekávanou událost je reagováno jedním z následujících způsobů:

1. Přepočítáním původního plánu (activity list) pomocí SGS. Plán je určen vybráním postihnutých operací v pořadí odpovídajícímu základnímu plánu.

2. Zcela novým výpočtem nad aktuálním RCPSP. Tento výpočet je realizován genetickým algoritmem podobně jako u základního optimalizátoru, jen je nutné snížit počet vypočtených plánů, aby se snížila doba výpočtu.

3. Stejným způsobem jako v druhém případě, jen je přidán do populace jedinec (plán), který odpovídá základnímu vypočtenému rozvrhu.

Dále je možno v SW nástroji dvěma způsoby specifikovat, jaké operace mají vstupovat do real-time optimalizace:

1. Všechny právě vykonávané a doposud nezahájené operace, jak bylo dosud uvažováno.
2. Pouze operace přímo postihnuté danou poruchou. Tento způsob je podrobněji popsán v podkapitole 5.4.1.

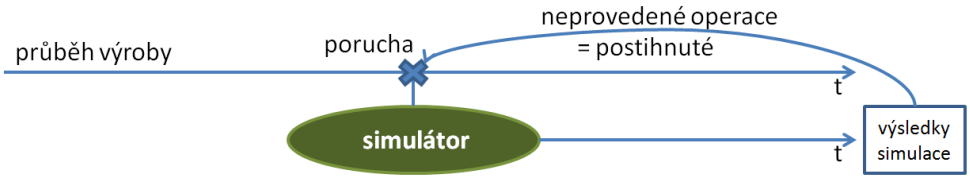
Poslední možností nastavení je určení momentu zahájení dynamického přeplánování:

1. Provést dynamické přeplánování okamžitě v momentě nahlášení neočekávané události.
2. Provést dynamické přeplánování až dojde k narušení původního rozvrhu.

5.4.1 Určení přímo postihnutých operací

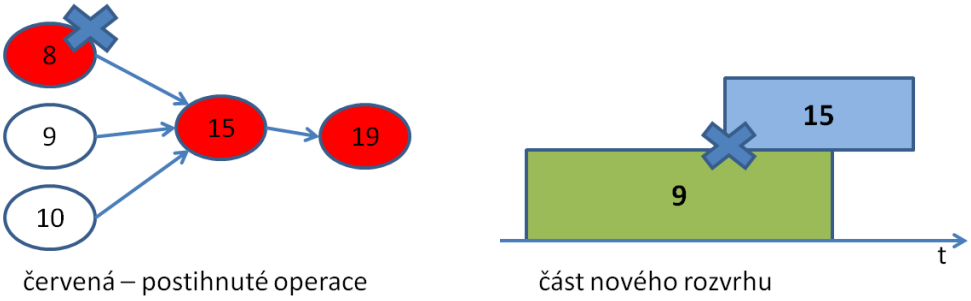
Jedním z nápadů, jak urychlit real-time optimalizaci je, že každá porucha přímo postihne pouze určitou množinu operací. Operace, které nebyly poruchou postihnuté, se nebudou v dynamickém přeplánování uvažovat a zůstanou na svém původním místě v rozvrhu. Toho lze dosáhnout patřičným snížením kapacity zdrojů v novém RCPSP.

Množinu přímo postihnutých operací lze určit jako operaci, u které nastal problém ve výrobě včetně všech jejich následovníků. Obecně jedna porucha může přímo narušit více operací, a proto je tento způsob určení postihnutých operací poměrně náročný. Lepším způsobem je využít simulátor průmyslové výroby a inicializovat ho aktuálním stavem výroby. Následně je provedena simulace výroby se zadanou poruchou. Simulátor odpoví na otázku, jaké operace nebylo možno podle rozvrhu zahájit a právě tyto operace jsou prohlášeny za přímo postihnuté. Tato myšlenka je zobrazena na obrázku 5.3.



Obrázek 5.3: Ukázka určení postihnutých operací pomocí simulátoru

Tento přístup skrývá problém, protože může dojít k situaci, která je zobrazena na obrázku 5.4. Množina $\{8, 15, 19\}$ obsahuje postihnuté operace určené k přeplánování. Pro operaci 15 je následně vypočten nový startovní čas, který je menší než čas konce operace číslo 9, která je předchůdcem této operace. Tento rozvrh je tedy neproveditelný. Problém vzniká z důvodu, že zanikla informace o všech předchůdcích operace 15 v novém RCPSP. Řešením tohoto problému by bylo zahrnout i tyto předchůdce do přímo postihnutých operací. Úlohy z PSPLIB jsou obvykle velmi provázané relacemi předchůdce, a proto by zahrnutí i všech předchůdců přímo postihnutých operací znamenalo v podstatě uvažování všech doposud nevykonaných operací. Z tohoto důvodu není v této práci tento problém řešen, a pokud k němu dojde, tak se musí provést nové přeplánování.



Obrázek 5.4: Ukázka problému s nerespektováním relace předchůdce

5.5 Implementace real-time optimalizátoru

Objekt představující real-time optimalizátor je vytvořen ze třídy `FastGaSolver`, která je zděděna od základního optimalizátoru `GaSolver`. `FastSolve()` je klíčovou metodou v této třídě, protože provede dynamické přeplánování jednou ze tří uvažovaných variant. Varianta s pouhým přepočítáním plánu je realizována pomocí metody `ParallelScheduleGeneration-Scheme()` ze třídy `Scheduling`.

Real-time optimalizátor je součástí emulovaného výrobního informačního systému, který je popsán v kapitole 6. Zda má dojít k přeplánování určuje návratová hodnota metody `Execute()` ze třídy představující určitou událost. Tato metoda zohledňuje nastavený moment zahájení přeplánování. Pokud je vyžadováno přeplánování, tak je zavolána metoda `Reschedule()` ze třídy `Simulation`. Tato metoda nejprve určí postihnuté operace jedním ze dvou uvažovaných způsobů a následně sestaví nový RCPSP. Nad tímto problémem je zavolána již zmiňovaná metoda `FastSolve()`.

Kapitola 6

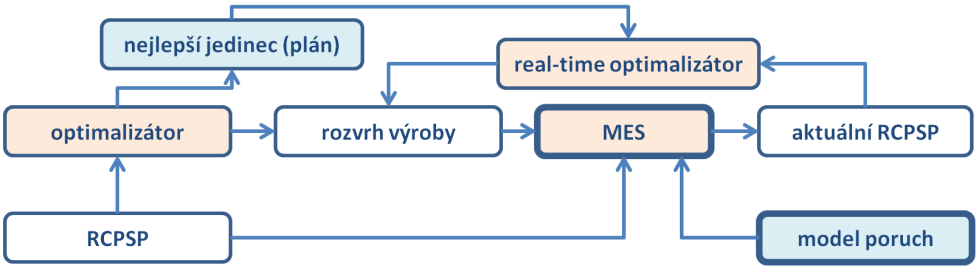
Výrobní informační systém

V této kapitole je popsán nástroj, který imituje činnost části výrobního informačního systému (MES) a také umožňuje simulovat provádění výrobních operací. Na začátku kapitoly je nejprve vysvětlen význam MES v rámci této práce. Poté je vysvětleno, na jakém principu provádí tento SW nástroj svou činnost a jak je navržen. V další podkapitole je detailně popsáno reagování na jednotlivé události ve výrobě. Následuje podkapitola popisující implementaci tohoto nástroje (simulátoru). Poslední podkapitola je věnována vlastní implementaci vykreslování Ganttova diagramu.

6.1 Význam výrobního informačního systému

MES (Manufacturing execution system) je informační systém, který řídí provádění výrobních operací. Také v reálném čase zaznamenává a předává ostatním systémům informace o výrobě. Organizace MESA vydala v roce 1997 „MESA-11“ model, který jednoznačně definuje 11 základních funkcí tohoto systému [17]. Tato nezisková organizace sdružuje společnosti zabývající se vývojem a implementací MES systémů.

V předchozí kapitole 5 byl podrobně popsán real-time optimalizátor výrobních operací. Tento optimalizátor pro svou činnost vyžaduje aktuální stav výroby, který je v reálné výrobě k dispozici v MES. Cílem této práce není vytvořit MES, ovšem pro ověření funkčnosti real-time optimalizátoru jsou data z tohoto systému potřebná. Z tohoto důvodu bylo nutné vytvořit nástroj, který bude emulovat potřebnou část MES.



Obrázek 6.1: Začlenění MES v tomto projektu

Na obrázku 6.1 je zobrazeno postavení uvažovaného nástroje (MES) v rámci tohoto projektu. Nástroj vyžaduje při spuštění rozvrh výroby se startovními časy jednotlivých operací a model výroby popsaný jako RCPSP. Model poruch a nejlepší původní plán jsou

volitelné. Pokud by měl být real-time optimalizátor nasazen do výroby, tak by reálný MES získával data ze skutečných terminálů ve výrobě namísto modelu poruch.

Nástroj, který je popisován v této kapitole, ve své podstatě umožňuje dvě činnosti:

- Již zmíněnou imitaci chování části MES. V případě narušení původního rozvrhu výroby je požádán real-time optimalizátor o přeplánování.
- Simulovat provádění výrobních operací. S modelem poruch lze provádět i tzv. what-if analýzu. Ze statistik simulace lze poté vyčíst, které operace byly postihnuté a které naopak proběhly v pořádku. Lze tedy získat nové užitečné informace o systému, což je obecně smyslem simulace. Důležité je si také uvědomit, že simulátor se bude pohybovat na úrovni výrobních operací, a proto není možné pomocí něj simulovat například zaplnění skladovacích prostor, fronty u strojů a podobně. Pro tyto účely existuje celá řada běžně používaných simulačních nástrojů. Doposud nepojmenovaný nástroj bude v celé této kapitole nazýván simulátorem.

6.2 Analýza a návrh simulátoru

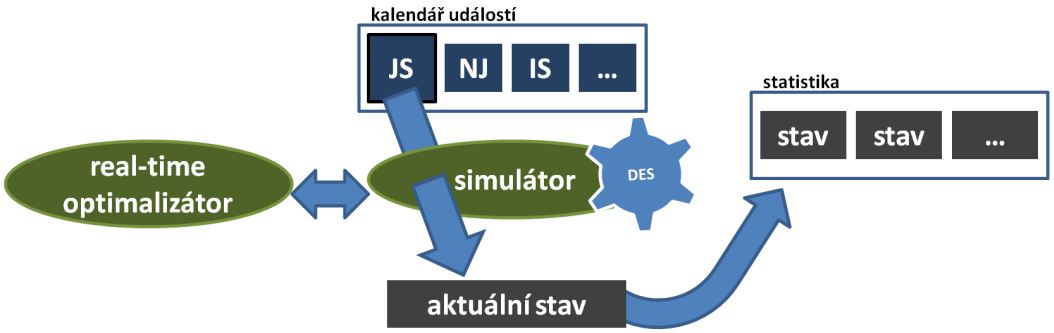
Existuje celá řada možností, jakými lze diskrétní simulaci výrobních operací popisovat. Jednou z možností je využít některý simulační jazyk/knihovnu a pouze nadefinovat vlastní chování jednotlivých událostí. Příkladem simulační knihovny pro C++ je například SIMLIB [19]. Mezi simulační jazyky lze zařadit například SIMSCRIPT III[1] a mezi simulační nástroje například Simulink[2]. Také je možno uvažovaný systém popisovat například pomocí Petriho sítí, Discrete Event System Specification (DEVS), Communicating Sequential Processes (CSP) a podobně. Pro daný problém jsem ovšem nenalezl příliš mnoho výhod, které by pramenily z použití některého z uvedených formalismů. Uvažovaný systém je poměrně specifický, a proto by se výhody použití již hotové simulační knihovny zúžily v podstatě na ušetření práce s implementací vlastní řídící smyčky diskrétní simulace. Tento základní algoritmus popsaný v následujícím odstavci je poměrně jednoduchý, a proto jsem se rozhodl jít v této práci cestou jeho vlastní implementace v obecném programovacím jazyce.

Diskrétní simulace při implementaci pomocí kalendáře událostí (next-event) se řídí následujícím algoritmem [19].

```
čas = POČÁTEČNÍ-ČAS-SIMULACE;
kalendář.Inicializace();
model.Inicializace();
while(not kalendář.Prázdný()) {
     aktivačníZáznam = kalendář.NajdiNejdřívější();
     if (aktivačníZáznam.čas > KONCOVÝ-ČAS-SIMULACE) {
          break;
     }
     kalendář.OdstraňNejdřívější();
     čas = aktivačníZáznam.čas;
     aktivačníZáznam.událost.Proveď();
}
čas = KONCOVÝ-ČAS-SIMULACE;
```

[1] http://www.simscript.com/
[2] http://www.mathworks.com/products/simulink/

Kalendářem událostí je myšlena uspořádaná datová struktura, která uchovává aktivační záznamy budoucích naplánovaných událostí. Každý tento aktivační záznam obsahuje trojici (čas, priorita, vlastní událost). Čas udává, na kdy je daná událost naplánována. Priorita určuje přednost událostí v případě výskytu více událostí ve stejný čas. Vlastní událost obsahuje popis činností, které se mají vykonat [19].



Obrázek 6.2: Návrh činnosti simulátoru

Jako základ pro simulátor tedy poslouží algoritmus diskrétní simulace s kalendářem událostí. Při inicializaci kalendáře se do něj načtou z rozvrhu výroby všechny naplánované starty operací. V případě existence modelu poruch jsou tyto poruchy do kalendáře rovněž načteny. Příklad počátečního stavu kalendáře a celkovou činnost simulátoru lze vidět na obrázku 6.2. Událost označená jako `JS` značí start operace. Porucha `IS` představuje zahájení výpadku stroje a `NJ` značí novou neplánovanou operaci. Prováděním těchto událostí dochází ke změně stavu výroby. V případě, že prováděná událost vrací hodnotu `false`, znamená to narušení rozvrhu výroby. Na tento problém je možno reagovat přeplánováním pomocí real-time optimalizátoru. Při inicializaci simulace jsou všechny operace označeny jako čekající na své provedení. Během simulace jsou ukládány jednotlivé stavy výroby do historie kvůli pozdějším statistikám. Každý stav simulace v určitém čase je určen:

- vytížením jednotlivých zdrojů,
- operacemi, které právě probíhají,
- operacemi, které již byly vykonány,
- operacemi, které jsou připraveny na své zahájení,
- operacemi, které nejsou připraveny na své zahájení (předchůdci nebyli ještě dokončeni),
- operacemi, jejichž provádění bylo přerušeno.

Prioritu všech uvažovaných událostí určuje tabulka 6.1. Při výskytu více událostí ve stejném čase obecně platí, že nejprve dochází k ukončení jednotlivých operací a tedy i k uvolnění zdrojů a až poté jsou spouštěny nové operace. V následující podkapitole je popsáno chování jednotlivých událostí.

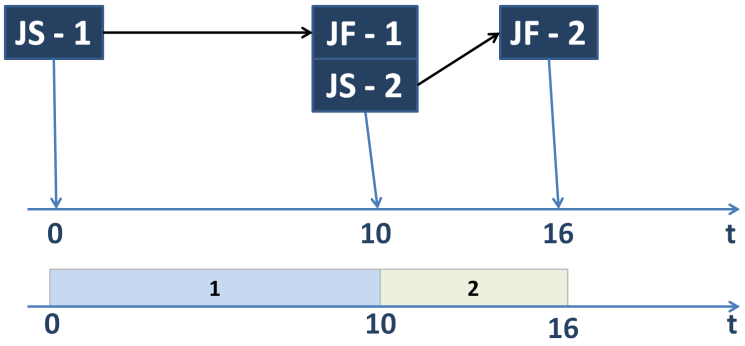
zkratka	popis	priorita
JD	změna doby trvání operace	60
IF	konec výpadku stroje	50
JF	ukončení výrobní operace	20
NJ	nová neplánovaná operace	10
JS-0	zahájení první operace	2
JS	zahájení výrobní operace	1
IS	zahájení výpadku stroje	0

Tabulka 6.1: Priorita jednotlivých událostí

6.3 Popis chování jednotlivých událostí

6.3.1 Zahájení a ukončení operace

Na obrázku 6.3 je zobrazena základní funkce simulátoru — tedy simulace výrobních operací, které jsou nahrány do kalendáře při startu (`JS`). V příkladu je ukázáno provedení operace číslo 1 s dobou trvání 10 a operace číslo 2 s dobou trvání 6. Při provádění této události je nejprve ověřeno, zda operace patří do množiny operací připravených na své zahájení (všichni předchůdci již byli dokončeni), nebo zda patří do množiny přerušených operací. Pokud tato podmínka není splněna, tak je událost ukončena s návratovou hodnotou, že nelze provést (`false`). Poté je ověřeno, zda má operace dostatek zdrojů na své zahájení. V případě nesplnění této podmínky je opět navrácena hodnota „neprovedeno“. V případě splnění obou podmínek operace zahajuje svoji činnost a je přidána do množiny operací, které právě probíhají. Také je uložen čas jejího startu a je odstraněna z operací připravených k zahájení nebo z přerušených operací. Následně jsou aktualizovány kapacity jednotlivých zdrojů v souladu s požadavky operace. Také je operace uložena jako poslední prováděná na používaných strojích. Posledním úkolem této události je naplánovat novou událost představující konec operace (`JF`). Pokud byla operace přerušena, tak je koncový čas operace určen s ohledem na již proběhlou dobu operace. Pokud byla právě zahájena, tak je podle RCPSP určena její doba provádění a instalační doba v závislosti na předchozí operaci na stroji.



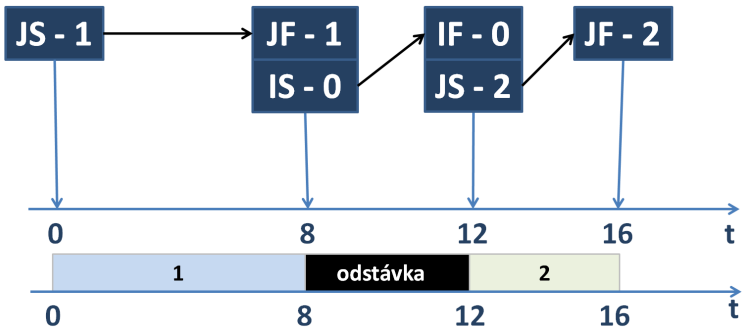
Obrázek 6.3: Na obrázku výše je zobrazen kalendář událostí při základní simulaci výrobních operací. Níže je naznačen Ganttův diagram pro uvažované operace a stroj.

Při události provádějící konec operace jsou nejprve zkontrolovány dvě podmínky — zda

je operace v množině právě probíhajících a zda již uplynula veškerá požadovaná doba výrobní operace. Pokud nejsou tyto podmínky splněny, tak je navrácena hodnota `true`, která v tomto případě znamená, že se nic neprovedlo. V případě splnění podmínek je prováděna ve své podstatě opačná činnost jako při zahájení operace. Nejprve je operace uvolněna z jednotlivých strojů, které využívala. Poté je smazána z množiny probíhajících operací a je přidána do množiny již ukončených operací. Na závěr je aktualizována množina operací připravených k provedení.

6.3.2 Přerušení a obnovení činnosti stroje

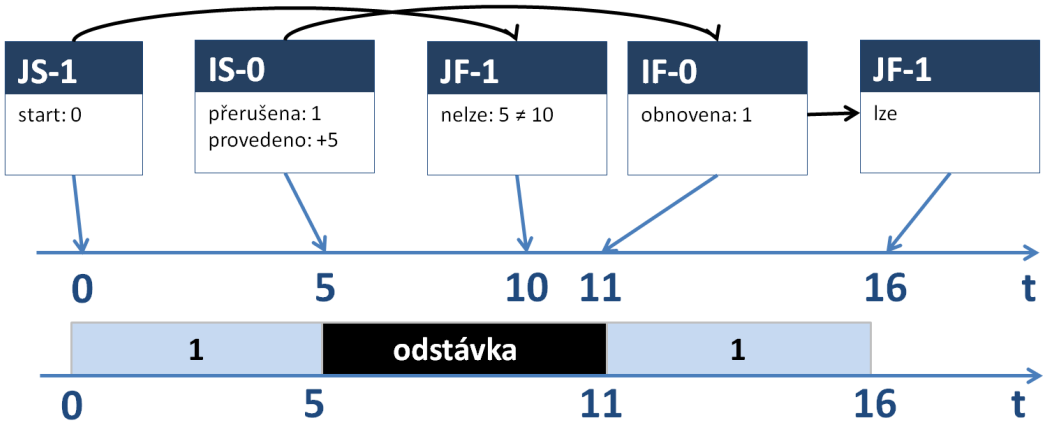
V předchozí podkapitole byla vysvětlena základní simulace výrobních operací. Simulátor dále umožňuje simulovat plánované odstávky strojů. Na obrázku 6.4 je uveden příklad odstávky (od 8 do 12) a dvou operací s dobami trvání 8 a 4. Díky prioritám je v čase 8 nejprve událostí `JF` ukončena operace číslo 1 a poté je zahájena plánovaná odstávka stroje (`IS`), která je ukončena událostí `IF`. V uvedeném příkladě je činnost událostí `IS` a `IF` velmi jednoduchá. Při události `IS` dojde pouze k nastavení aktuální kapacity zdroje na 0 a při události `IF` dojde k nastavení na maximální kapacitu zdroje.



Obrázek 6.4: Příklad proveditelného rozvrhu s plánovanou odstávkou stroje

V případě přerušitelné operace je situace komplikovanější, neboť je nutné právě probíhající operaci při odstávce přerušit. Princip tohoto přerušení je zobrazen na obrázku 6.5. Operace má celkovou dobu trvání 10 a odstávka trvá 6. Při zahájení odstávky v čase 5 je právě probíhající operace přesunuta do množiny přerušených operací. Zároveň je uložena již provedená doba operace — v tomto případě 5. Plánovaný konec výrobní operace v čase 10 nelze provést, protože neproběhla celá požadovaná doba operace. V čase 11 dochází ke konci odstávky stroje (`IF`). Tato událost projde množinu přerušených operací, a pokud je možno některou z nich obnovit, tak to provede. V uvažovaném případě lze obnovit operaci číslo 1 a je naplánován její konec s ohledem na již provedenou dobu na čas 16.

Pokud by v modelu poruch byl definován neplánovaný výpadek stroje, tak je tato skutečnost vložena do kalendáře událostí opět jako událost (`IS`). Při jejím vykonávání je navíc tento výpadek uložen do aktuálního modelu RCPSP pro potřeby real-time optimalizace. Opět dojde k přerušení aktuálně prováděných operací na daném stroji. Rozdíl je, že při neplánovaném výpadku událost `IS` navrací hodnotu `false`, která znamená, že nastal problém. V případě reakce na tento problém přeplánováním dojde k určení nových startovních časů přerušených operací — tedy k vložení nových začátků operací `JS`.



Obrázek 6.5: Příklad přerušitelné operace při plánované odstávce stroje

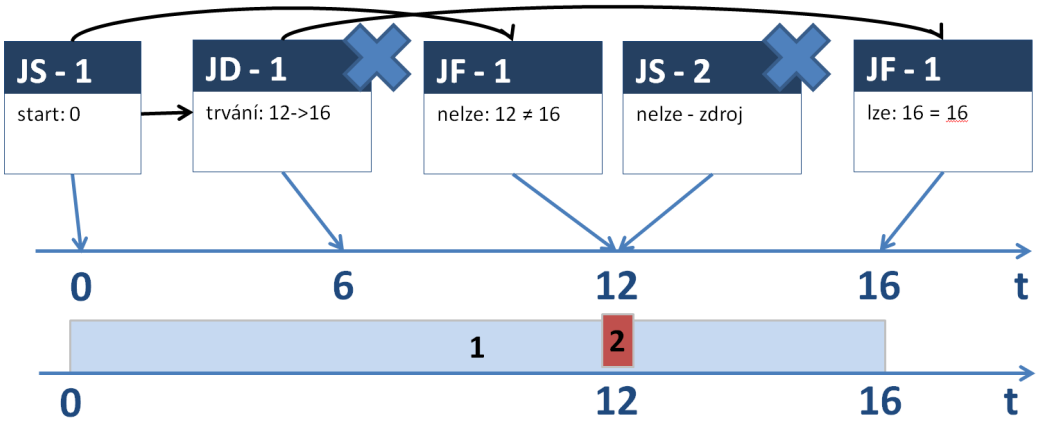
6.3.3 Změna doby trvání operace

Další událost, která se vyskytuje v kalendáři událostí, je změna doby trvání operace (`JD`). Tato událost představuje například hlášení z výroby, že vzhledem k nějakému problému dochází k prodloužení plánované doby trvání operace. Na obrázku 6.6 je uveden takový příklad změny z 12 na 16 pro operaci jedna. Událost představující tuto změnu (`JD`) je vytvořena při provádění události `JS` a je vždy vložena mezi `JS` a `JF` s menším koncovým časem. Vložení této události (`JD`) neprobíhá při inicializaci kalendáře, protože doba trvání operace je v tuto dobu neznámá, neboť je závislá na předchozí operaci na stroji.

Vlastní činnost události `JD` spočívá ve změně doby trvání dané operace v modelu popsaném RCPSP. Dále tato událost naplánuje nový konec dané operace (`JF`). V uvažovaném příkladě nastává změna trvání operace v čase 6 a nový konec operace je naplánován na čas 16. V případě volby aktivního přístupu je navrácenou hodnotou této události `false` a dochází obvykle k přeplánování. V uvažovaném příkladě je ovšem pasivní přístup, a proto simulace pokračuje bez problémů dále. Problém ovšem vznikne v čase 12, ve kterém se měla operace 12 ukončit, což nyní nelze, protože neproběhla celá. Zároveň měla v tomto čase začít nová operace číslo dva (`JF-2`). Tato operace nemůže začít, protože požadovaný zdroj je obsazený první operací. V daném momentě tedy nastává problém a je nutné na něj zareagovat jednou z uvažovaných možností.

6.3.4 Nová operace

Poslední uvažovanou poruchou v modelu poruch je vznik nové neplánované operace. Tato porucha je v kalendáři reprezentována událostí `NJ` a je do kalendáře vložena při inicializaci. Při jejím provádění dojde k vložení této nové operace do modelu výroby popsaného RCPSP. V aktuální verzi je operace vložena jako nezávislá. To znamená, že jediným předchůdcem nové operace je operace představující start projektu a jediným následovníkem nové operace je operace představující konec celého projektu. Nová operace je také vložena do množiny operací čekajících na své provedení. Událost končí vždy s návratovou hodnotou `false`.



Obrázek 6.6: Příklad změny doby trvání operace

6.4 Implementace simulátoru

Vlastní realizace simulátoru vychází z navrženého schématu na obrázku 6.2. Simulátor je implementován v obecném programovacím jazyce C++. Jak již bylo uvedeno, simulátor je navržen jako diskrétní simulace s kalendářem událostí. Kalendář událostí je implementován jako vlastní třída, která je postavena na datové struktuře prioritní fronty (`priority_queue`). Tato datová struktura obsahuje objekty reprezentující jednotlivé události. Tyto objekty jsou instance tříd, které byly odvozeny od abstraktní třídy `Event` a implementují její čistě virtuální metodu `Execute()`. Podle návratové hodnoty metody `Execute()` je zjištěno, zda je dodržován rozvrh výroby. V případě výskytu problému je jednou z implementovaných variant zavolání metody `FastSolve()` z objektu real-time optimalizátoru, která určí nové startovní časy jednotlivých operací. Na základě těchto nových časů je patřičně upraven kalendář událostí.

Objekt představující aktuální stav výroby je vytvořen ze třídy `State`. Tato třída převážně uchovává množiny, které obsahují například právě probíhající operace, dokončené operace atd. Více je popsáno v podkapitole 6.2. Jako datový typ pro tyto množiny byl zvolen datový typ `set` ze standardní knihovny C++. Zajišťuje unikátnost prvků a poskytuje rychlé hledání prvků s logaritmickou časovou složitostí. Historie stavů je implementována jako datová struktura `map`, přičemž klíčem je čas a hodnotou zmiňovaný stav. Zajišťuje to, že pro jeden konkrétní čas bude existovat pouze nejaktuálnější stav výroby. Na základě této historie stavů je poté vytvořena statistika simulace.

6.5 Vykreslování Ganttova diagramu

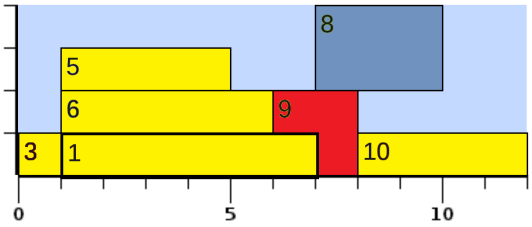
Výstupem simulátoru je soubor se statistikou, který obsahuje informace typu, kdy jaká operace byla v jakém stavu. Tyto informace jsou pro účel základního přehledu provádění výrobních operací poměrně nečitelné, a proto se velmi často zobrazují ve formě Ganttova diagramu. Existuje celá řada nástrojů, které dokáží Ganttův diagram vykreslovat. Mezi tyto nástroje lze zařadit například Microsoft Project[3] nebo skript nad Gnuplotem[4]. Nevýhodou těchto nástrojů je, že uvažují kapacitu zdrojů pouze jedna. Během hledání jsem také narazil

[3] http://www.microsoft.com/cze/project2010/
[4] http://se.wtb.tue.nl/sewiki/wonham/gantt.py

na SW RESCON[5], který umožňuje vykreslovat Ganttův diagram přesně v požadovaném formátu. Problémem ovšem je, že účelem nástroje je provádět optimalizační výpočty a nikoliv vykreslovat Ganttův diagram, a proto není možno zadat požadovaná data na vykreslení.

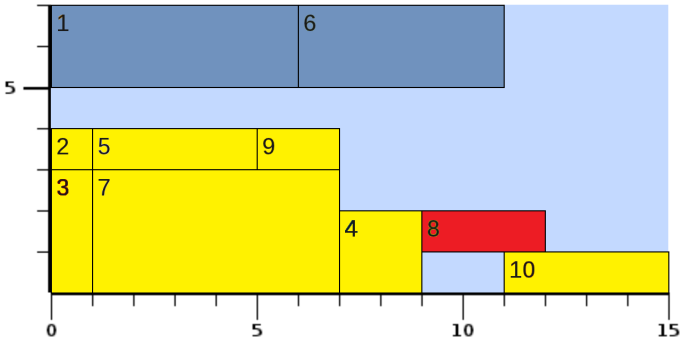
V předchozím odstavci byly popsány důvody, proč jsem se v této práci rozhodl implementovat vlastní vykreslování Ganttova diagramu. Jako výstupní formát byl zvolen HTML5 a to hlavně z důvodu snadného vykreslování pomocí elementu `canvas` a jednoduchého přidávání událostí pomocí JavaScriptu a knihovny jQuery[6].

Nejsložitější částí při implementaci Ganttova diagramu bylo vymyslet algoritmus na rozmístění jednotlivých operací, aby docházelo k co nejmenšímu překryvu. Jelikož diagram bude sloužit jen pro základní představu, tak jsem se spokojil i s řešením, při kterém bude občas docházet k překryvu, jak je zobrazeno na obrázku 6.7. Po najetí kurzorem na požadovanou operaci dojde k jejímu přesunutí do nejvyšší vrstvy, jak lze vidět také na uvedeném obrázku u operace jedna.



Obrázek 6.7: Příklad překrývajících se operací v Ganttově diagramu

Samotný algoritmus pracuje na principu, který je ilustrován na obrázku 6.8. Nejprve jsou jednotlivé položky (operace) na vykreslení seřazeny sestupně podle doby trvání. V tomto pořadí jsou vykreslovány nejprve odspodu. Přičemž pokud by nastala situace, při které by položka nebyla celou svou plochou na jiné položce nebo na „dnu“, tak nebude vykreslena. Operace vykreslené při prvním průchodu jsou v uvedeném obrázku vyznačeny žlutou barvou. Poté je tento algoritmus opakován jen s tím rozdílem, že skládání operací probíhá od shora (operace 1 a 6 v příkladu). Pokud zůstane nějaká položka, která ani po tomto průchodu není vykreslena, tak jsou testovány všechny možnosti jejího umístění. Následně je vložena na to místo, kde dochází k co nejmenšímu překrývání s ostatními položkami. V uvažovaném příkladě je to operace 8.



Obrázek 6.8: Příklad skládání jednotlivých operací v Ganttově diagramu

[5] http://www.econ.kuleuven.be/rescon/
[6] http://jquery.com/

Kapitola 7

Experimenty a výsledky

Tato kapitola popisuje prováděné experimenty se základním a s real-time optimalizátorem. Na začátku kapitoly je představena knihovna PSPLIB, která obsahuje množinu různých RCPSP, na kterých jsou prováděny experimenty. Poté je zde podkapitola věnovaná porovnání implementovaného algoritmu optimalizátoru s ostatními publikovanými. V další podkapitole je testována časová složitost optimalizátoru. Poté je experimentováno s různými variantami real-time optimalizace. Na závěr jsou shrnuty zjištěné výsledky.

7.1 Project Scheduling Problem Library

„Project scheduling problem library“ (PSPLIB) obsahuje různé testovací sady problémů s RCPSP, MRCPSP a dalšími problémy. Jednotlivé množiny s testovacími problémy se liší počtem operací (30, 60 a 120) a každá tato množina obsahuje řádově stovky úloh s RCPSP.

PSPLIB rovněž obsahuje optimální řešení (případně nejlepší doposud známé řešení) každého problému. Tyto množiny problémů slouží k testování výkonnosti algoritmů řešící tyto problémy a k objektivnímu porovnání s ostatními. Objektivního porovnání je obvykle docíleno volbou konce algoritmu po 1000 nebo po 5000 sestaveném rozvrhu. Jednotlivé množiny problémů byly systematicky vygenerovány pomocí generátoru ProGen[1]. Více o této knihovně problému a zmíněném generátoru lze najít v článku [15].

7.2 Experimentování se základním optimalizátorem

V této podkapitole je objektivně porovnán implementovaný algoritmus základního optimalizátoru s ostatními publikovanými řešeními. Pro porovnání byla zvolena datová sada (j30.sm) z PSPLIB obsahující celkem 480 RCPSP, kde každý problém obsahuje 30 operací. Ukončovacím kritériem bylo zvoleno 1000 sestavených plánů.

Během implementace základního optimalizátoru bylo experimentováním zjištěno, že tvorba počáteční populace má velký vliv na celkovou kvalitu řešení. Při prvotním generování jedinců se ukázalo jako výhodné nevybírat pořadí operací zcela náhodně, ale upřednostňovat operace s menším nejpozdějším možným koncovým časem (LFT). V této práci byly implementovány obě verze SGS a jejich současné použití při generování počáteční populace se rovněž projevilo jako velmi výhodné.

Samotné porovnání implementovaného genetického algoritmu s ostatními lze vidět v tabulce 7.1. Velikost populace byla nastavena na 200 jedinců s pravděpodobností mutace 15 %.

[1] Generátor i testovací množiny jsou na adrese: http://129.187.106.231/psplib/.

Hodnoty v tabulce vyjadřují průměrnou odchylku od optimálního řešení a byly převzaty z [24].

Algoritmus	typ SSG	Reference	průměrná odchylka
GA – Self-adapting	Oba	Hartmann	0,38 %
GA – Activity list	Serial	Hartmann	0,54 %
GA – LFT	Oba	Tento projekt	1,09 %
Sampling – LTF	Parallel	Kolisch	1,4 %
Sampling – Random	Parallel	Kolisch	1,77 %
GANS	Serial	Proon and Jin	1,83 %
GA – Problem space	Parallel	Leon and Ramamoorthy	2,08 %

Tabulka 7.1: Porovnání algoritmu s ostatními – RCPSP j30 z PSPLIB [24]

7.3 Experimentování s časovou složitostí optimalizátoru

V tomto testu byla zkoumána časová složitost vypočtu RCPSP v závislosti na počtu operací (n) pro potřeby real-time optimalizace. Teoretická časová složitost je pro SSGS i pro PSGS v literatuře uváděna jako $\mathcal{O}(n^2|\mathcal{K}|)$ [14]. Testovací úlohy s různým počtem operací (30, 60 a 120) byly převzaty z knihovny problémů PSPLIB. Z každé množiny problémů (j30, j60 a j120) bylo vybráno náhodně 5 úloh. Omezujícím kritériem výpočtu bylo maximálně 1000 sestavených plánů. Výsledky měření pro PSGS jsou zobrazeny v tabulce 7.2. Z naměřených dat lze vidět, že reálná časová složitost takřka odpovídá teoretické složitosti. Všechny testované úlohy obsahovaly celkem čtyři zdroje. S přibývajícím počtem operací se časová složitost zvyšuje kvadraticky za předpokladu konstantního počtu zdrojů.

	naměřená		teoretická
počet operací	5 úloh [s]	1 úloha [s]	1 úloha [s]
30 (j30)	2,431	0,486	0,486
60 (j60)	8,537	1,707	1,945
120 (j120)	40,237	8,047	7,779
480	N/A	N/A	124,467
960	N/A	N/A	497,868

Tabulka 7.2: Časová složitost výpočtu v závislosti na počtu operací - PSGS

Dále byl testován optimalizační výpočet s využitím SSGS pro sestavování rozvrhů. Ostatní podmínky měření zůstaly shodné jako v předchozím případě. Implementovaná varianta SSGS vykazuje výrazně lepší výsledky než předešlá 7.3, ovšem nelze ji s ní přímo porovnávat, neboť PSGS obsahuje mnoho rozšíření navíc (údržba, priority atd.). Také ve zkoumaném rozsahu počtu operací není patrná kvadratická závislost.

V reálné výrobě se počet operací může pohybovat kolem tisíce. V případě výskytu poruchy hned na začátku a v případě postihnutí většiny operací by real-time přeplánování mohlo trvat až cca 8 minut v případě PSGS za předpokladu platnosti kvadratické závislosti. Tato doba je příliš dlouhá, neboť je předpoklad, že by měla trvat maximálně cca 1 minutu. Toho lze nejjednodušším způsobem docílit snížením maximálního počtu sestavených plánů

počet operací	naměřená 5 úloh [s]	naměřená 1 úloha [s]	teoretická 1 úloha [s]
30 (j30)	1,093	0,219	0,219
60 (j60)	2,341	0,468	0,874
120 (j120)	5,57	1,11	3,498

Tabulka 7.3: Časová složitost výpočtu v závislosti na počtu operací - SSGS

na $1000/8 = 125$. Pokud bychom uvažovali o běžnější situaci – například 300 postihnutých operací, pak by výpočet trval i při 1000 sestavených plánech do jedné minuty. Zdrojů bude v reálné výrobě více, takže pro další testy bude uvažována hodnota maximálně 400 sestavených plánů v real-time optimalizátoru. Pro úplnost uvádím, že experimenty byly prováděny ve virtualizovaném OS spuštěném na procesoru Intel(R) Core(TM) i5 – 2,3 GHz.

7.4 Experimentování s real-time optimalizátorem

V kapitole 5, která popisuje real-time optimalizátor, bylo uvedeno několik implementovaných možností přístupu k real-time optimalizaci. V této podkapitole je experimentováno s těmito variantami za účelem zjištění vhodného přístupu. Na začátku práce byly stanoveny tři základní požadavky na kvalitní real-time optimalizátor. Nejprve tedy bylo nutné vymyslet způsob měření těchto tří parametrů:

- *Kvalita nového rozvrhu (K).* Je určena jako procentuální odchylka od optimálního nebo doposud nejlepšího známého řešení.
- *Odlišnost od původního rozvrhu (O).* Tento parametr lze určit jako procentuální hodnotu udávající počet operací, které v novém rozvrhu začínají ve stejném čase jako v původním. To by ovšem mohlo výrazně poškozovat hodnocení algoritmu provádějící pouhé přepočítání původního rozvrhu, protože obvykle u něj dochází k malému posunutí všech operací. Pořadí operací je ovšem zachováno. Z tohoto důvodu je odlišnost měřena jako průměrná odchylka nových startovních časů od původních.
- *Časová složitost real-time optimalizace (D).* Tato vlastnost je měřena jako reálná doba výpočtu na procesoru.

Tyto prezentované parametry tedy hodnotí kvalitu jednotlivých variant. Všechny níže popsané experimenty byly prováděny na všech RCPSP z množiny problémů j60 z PSPLIB. Experimenty byly realizovány následujícím způsobem. Nejprve byly základním optimalizátorem (PSGS, LFT) vypočteny základní rozvrhy všech problémů. Následně bylo vytvořeno několik modelů poruch, které pokrývají různé situace a jsou vypsány v tabulce 7.4. Poté pomocí automatického skriptu byly prováděny experimenty s různým nastavením real-time optimalizátoru a s různými modely poruch.

Před samotnými experimenty byla ověřena funkčnost simulátoru (emulátoru MES) bez modelu poruch. Průměrná procentuální odchylka od nejlepších známých řešení byla $3,58$ %, což zcela odpovídá hodnotě vypočtené základním optimalizátorem. Simulace 480 rozvrhů výroby trvala celkem $18,4$ sekundy (jeden cca 38 ms), přičemž přibližně třetinu času zabralo vytváření Ganttova diagramu a výpisu simulace.

zkratka	typ poruchy	parametry (hodnota změny)
JD-1-200	změna doby trvání	1 operace (200 %)
JD-2-200	změna doby trvání	2 operace (obě 200 %)
JD-4-200	změna doby trvání	4 operace (všechny 200 %)
JD-1-50	změna doby trvání	1 operace (50 %)
JD-2-50	změna doby trvání	2 operace (obě 50 %)
JD-5-50	změna doby trvání	5 operací (všechny 50 %)
JD-2-50-200	změna doby trvání	2 operace (50 %, 200 %)
IS-0-10-12	porucha stroje	stroj č. 0 od 10 do 12
NJ-1-10	nová operace	čas 10, doba trvání 5 – stroj č. 0

Tabulka 7.4: Přehled použitých poruch v experimentech

7.4.1 Experimentování s vhodným momentem zahájení přeplánování

Cílem tohoto experimentu je zjistit, zda je výhodnější spustit real-time optimalizaci okamžitě při zjištění poruchy (I) nebo až v momentě, kdy dojde k narušení původního rozvrhu výroby (II). Výsledky experimentů jsou uvedeny v tabulce 7.5 pro variantu pouhého přepočítání původního plánu a v tabulce 7.6 pro variantu zcela nového výpočtu. Zkratky uvedené v tabulce odpovídají měřeným parametrům, které jsou vysvětleny na začátku podkapitoly 7.4 (K – kvalita, O – odlišnost a D – časová složitost).

	okamžité (I)			nezbytné (II)		
porucha	K [%]	O [t]	D [s]	K [%]	O [t]	D [s]
JD-1-200	6,91	0,90	17,50	6,75	0,82	17,41
JD-2-200	10,08	1,93	18,10	9,67	1,75	17,83
JD-4-200	15,35	3,42	19,30	14,68	3,14	18,64
JD-1-50	3,84	0,33	17,70	3,58	0,00	17,01
JD-2-50	4,02	0,72	17,90	3,58	0,00	16,90
JD-5-50	3,36	1,39	19,50	3,58	0,00	17,45
JD-2-50-200	7,24	1,09	18,10	6,55	0,59	17,40
IS-0-10-12	6,36	1,54	18,10	6,36	1,54	17,99
NJ-1-10	3,83	1,29	18,13	3,83	1,29	18,68

Tabulka 7.5: Výsledky experimentů s pouhým přepočítáním původního plánu

Z výsledků v tabulce 7.5 vyplývá, že i při malé změně původního rozvrhu (JD-1-50) již není původní plán příliš efektivní. Přeplánování se tedy vyplácí provádět až v momentě, kdy je to opravdu nezbytné, neboť při této variantě přeplánování dochází ke zhoršení kvality rozvrhu. Až v případě pěti operací, které snížily svou dobu trvání na polovinu (JD-5-50), byl získán kvalitnější rozvrh než v případě nereagování na tuto pozitivní poruchu.

Tabulka 7.6 obsahuje naměřené výsledky pro případ zcela nového výpočtu. Je důležité si uvědomit, že real-time optimalizátor nemá při tomto výpočtu tolik času jako základní optimalizátor, a proto obecně vytváří méně kvalitní rozvrh než základní optimalizátor. Toto je patrné například u jedné operace, která zdvojnásobila svou dobu trvání (JD-1-200), protože občas nastává situace, že tato změna nenaruší původní rozvrh. Skutečnost, že některé rozvrhy nebyly narušeny, lze pozorovat z celkové doby výpočtu (nebylo nutné

porucha	okamžité (I) K [%]	O [t]	D [s]	nezbytné (II) K [%]	O [t]	D [s]
JD-1-200	6,08	1,30	135,20	5,87	1,10	101,85
JD-2-200	8,40	2,39	272,14	8,14	2,17	193,90
JD-4-200	13,52	3,89	532,83	13,02	3,53	348,94
JD-1-50	3,35	0,79	130,00	3,58	0,00	16,84
JD-2-50	2,99	1,68	252,28	3,58	0,00	16,96
JD-5-50	1,82	2,60	604,50	3,58	0,00	16,95
JD-2-50-200	5,95	1,91	261,16	5,83	0,76	69,54
IS-0-10-12	5,90	2,94	281,97	6,21	2,97	274,97
NJ-1-10	4,20	3,36	307,73	4,29	3,40	305,63

Tabulka 7.6: Výsledky experimentů se zcela novým výpočtem.

provádět přeplánování). Z tohoto důvodu je mírně výhodnější provádět přeplánování až v případě narušení rozvrhu. Naopak v případě zkrácení doby trvání operací je výhodné provést okamžité přeplánování.

7.4.2 Experimentování s různými variantami přeplánování

Tento experiment si klade za cíl porovnat výhody a nevýhody tří implementovaných variant přeplánování, které byly popsány v podkapitole 5.4. Při provádění tohoto experimentu byl uvažován pouze přístup okamžitého reagování na poruchu. Varianta označená „nový výpočet 2“ znamená nový výpočet GA s využitím původního nejlepšího plánu.

porucha	přepočítání (I) K [%]	O [t]	D [s]	nový výpočet (II) K [%]	O [t]	D [s]	nový výpočet 2 (III) K [%]	O [t]	D [s]
JD-1-200	6,91	0,90	18,84	5,94	1,27	148,69	5,77	1,10	144,30
JD-2-200	10,08	1,93	18,16	8,54	2,44	285,57	8,58	2,30	295,96
JD-4-200	15,35	3,42	19,02	13,48	3,91	559,53	13,56	3,74	561,31
JD-1-50	3,84	0,33	17,45	3,32	0,81	142,52	3,24	0,66	136,40
JD-2-50	4,02	0,72	17,50	3,16	1,74	291,76	2,97	1,36	270,30
JD-5-50	3,36	1,40	19,06	1,87	2,62	673,89	1,73	2,32	651,04
JD-2-50-200	7,24	1,09	18,22	6,12	1,92	275,29	5,81	1,61	280,67
IS-0-10-12	6,36	1,54	17,81	6,09	2,94	301,98	5,59	2,23	302,03
NJ-1-10	3,83	1,29	18,13	4,20	3,36	307,73	3,77	2,16	304,90

Tabulka 7.7: Porovnání různých variant přeplánování

Výsledky tohoto experimentu jsou uvedeny v tabulce 7.7. Z těchto naměřených dat plyne, že výhodou varianty pouhého přepočítání původního plánu (I) je velmi krátká doba výpočtu (cca 18 s/480 = 37,5 ms) a také větší podobnost s původním rozvrhem. Naopak jednoznačnou nevýhodou je skutečnost, že takto vytvořené rozvrhy nejsou příliš kvalitní.

Dále se ukázalo jako výhodné zanést do nového výpočtu genetického algoritmu informaci o minulém nejlepším jedinci (plánu) (III). Tento přístup vykazuje ve všech sledovaných údajích lepší výsledky než genetický algoritmus prováděný zcela od znovu (II).

7.4.3 Experimentování s přeplánováním pouze postihnutých operací

Tento experiment si klade za cíl zjistit, zda je výhodné při přeplánování přímo nepostihnuté operace nechat v rozvrhu na svém původním místě nebo je také začlenit do výpočtu. Způsob určení přímo postihnutých operací je vysvětlen v podkapitole 5.4.1. Experimenty byly prováděny s dvěmi variantami přeplánování — (I) přepočítání původního plánu a (II) nový výpočet GA.

	přepočítání (I)			nový výpočet (II)		
porucha	K [%]	O [t]	D [s]	K [%]	O [t]	D [s]
JD-1-200	7, 21	0, 90	18, 25	6, 10	1, 10	178, 01
JD-2-200	10, 06	1, 89	18, 91	8, 51	2, 22	336, 58
JD-4-200	15, 42	3, 40	20, 60	13, 45	3, 72	621, 62
JD-2-50-200	6, 99	0, 98	18, 79	5, 86	1, 62	330, 83

Tabulka 7.8: Výsledky experimentů s přeplánováním pouze postihnutých operací

Výsledky tohoto experimentu jsou uvedeny v tabulce 7.8. Naopak výsledky experimentů, při nichž byly přeplánovány všechny doposud nevykonané operace, jsou uvedeny v tabulce 7.7. U přístupu přeplánování pouze přímo postihnutých operací bychom očekávali, že doba výpočtu bude kratší než v případě uvažování všech operací. Z naměřených dat ovšem plyne, že doba výpočtu je dokonce mírně vyšší. Je tomu tak z důvodu, že tato metoda nevytváří vždy proveditelný rozvrh, a proto je často nutné jej ještě jednou přeplánovat. Výhodou tohoto přístupu je naopak mírně lepší podobnost s původním rozvrhem. U silně provázaných úloh z PSPLIB se tento přístup tedy mnoho neosvědčil.

7.5 Shrnutí výsledků

V této části práce jsou zhodnoceny naměřené hodnoty popsané v předchozích podkapitolách. Důležité je si uvědomit, že měření probíhala na teoretických problémech z knihovny PSPLIB. Tyto úlohy se vyznačují poměrně složitými relacemi předchůdců, velkou kapacitou zdrojů a podobně. Níže prezentované výsledky proto mají platnost pouze pro úlohy podobného typu, jako jsou v PSPLIB.

Z naměřených hodnot vyplývají užitečné vlastnosti, které lze použít v rozhodování, jakým způsobem je nejlepší se v daném okamžiku zachovat. V následujícím seznamu je uveden výpis zjištěných skutečností.

- *K optimálnímu řešení někdy nevede cesta.* Tabulka 7.1 prezentuje porovnání různých algoritmů na řešení RCPSP. Náhodné generování proveditelných rozvrhů (Sampling – Random) nevykazuje velkou odchylku od optimálního řešení, jak by mohlo být předpokládáno. Je tomu tak z důvodu, že optimální řešení někdy nemá v podstatě žádnou podobnost s kvalitními řešeními, které jsou získávány během provádění genetického algoritmu.
- *Okamžité přeplánování je výhodné pouze u poruch, kde dochází ke zkrácení doby trvání operace.* Je tomu tak z důvodu, že real-time optimalizátor neposkytuje natolik kvalitní rozvrh jako základní optimalizátor. Z tohoto důvodu je výhodné provádět přeplánování, jen pokud je to opravdu nutné nebo se jedná o zkrácení doby trvání operace.

- *Přeplánování se nevyplatí vždy ani u zkrácení doby trvání operace.* Za předpokladu reagování na poruchu pouhým přepočtením původního plánu nastává často následující situace. Na zdroji se objeví dostatečná kapacita na zahájení operace, která byla v původním rozvrhu odkládána právě kvůli nedostatečné kapacitě. To ovšem má obvykle za následek, že další operace začínají později a nové uspořádání nemusí být příliš výhodné.

- *Varianta s přepočítáním původního plánu nalezne v krátkém čase nový rozvrh, který je podobný původnímu.* Nevýhodou ovšem je, že kvalita rozvrhu není příliš velká v porovnání s novým výpočtem.

- *Při novém výpočtu genetického algoritmu se vyplatí vložit do populace původní nejlepší plán (jedince).* Tyto rozvrhy jsou obecně mírně kvalitnější a podobnější původnímu rozvrhu než v případě genetického algoritmu bez tohoto jedince. Pokud by bylo cílem získávat co nejvíce efektivní rozvrhy výroby, byl by zvolen tento přístup.

- *Jedním z řešení v reálné výrobě by bylo nechat uživatele vybrat jeden rozvrh z několika nabízených.* Lidský expert zohledňuje daleko více aspektů, než je uvažováno v RCPSP. Může se například jednat o slíbenou dovolenou a podobně.

Kapitola 8

Závěr

V rámci této diplomové práce jsem vytvořil základní optimalizátor výrobních operací založený na vlastním genetickém algoritmu. Tento algoritmus vykazuje na testovacích problémech z PSPLIB srovnatelné výsledky jako podobné publikované řešení. Rovněž jsem navrhl vlastní model poruch a vytvořil jsem systém, který je schopen plynule reagovat na vznikající problémy ve výrobě. Systém také umožňuje provádět diskrétní simulaci nad zadaným výrobním rozvrhem s modelem poruch.

V real-time optimalizátoru bylo implementováno několik variant přeplánování a různých přístupů. S různými modely poruch a s různými možnostmi přístupu k real-time optimalizaci byly prováděny četné experimenty. Jejich výsledky jsou popsány v předchozí kapitole.

Kromě řešení teoretických problémů z knihovny PSPLIB jsem se v této práci také zaměřil na rozvrhování reálné průmyslové výroby. Pro tento účel bylo nutné základní RCPSP rozšířit o proměnlivou kapacitu zdrojů v čase, instalační dobu operací a podobně. Možnost praktického využití tohoto SW byla konzultována s firmou Visteon-Autopal, která je předním světovým výrobcem chladičů pro automobilový průmysl. Během jednání s pracovníky z logistiky a z vytěžování zdrojů jsem měl možnost získat cenné praktické informace o reálném rozvrhování výroby.

Tato diplomová práce pokrývá poměrně široké téma a převážně v oblasti RCPSP s prvky neurčitosti je nyní veden poměrně intenzivní výzkum. V rámci dalšího rozvoje této práce by bylo možno podrobně zkoumat další metody řešení tohoto problému — například SAT. Dále by mohl být optimalizátor rozšířen o anticipaci případných poruch ve výrobě a vytvářet tak robustnější rozvrh výroby. Rovněž by bylo možné akcelerovat genetický algoritmus s využitím GPU. Pro případné praktické použití optimalizátoru by jej bylo nutné mírně přizpůsobit dané výrobě.

Tato práce byla prezentována na studentské konferenci a soutěži Student EEICT 2012. V kategorii s názvem Inteligentní systémy se umístila na třetím místě [16].

Literatura

[1] Artigues, C.; Demassey, S.; Néron, E.: *Resource-constrained project scheduling - Models, Algorithms, Extensions and Applications*. Wiley, 2008, ISBN 978-1-84821-034-9, 320 s.

[2] Ballestín, F.; Leus, R.: Resource-Constrained Project Scheduling for Timely Project Completion with Stochastic Activity Durations. *Production and Operations Management*, ročník 18, č. 4, 2009: s. 459–474, ISSN 1937-5956, doi:10.1111/j.1937-5956.2009.01023.x.

[3] Barták, R.: Plánování a rozvrhování (přednášky) [online]. 2012 [cit. 2012-05-17]. Dostupné z: http://ktiml.mff.cuni.cz/~bartak/planovani/index.html.

[4] Blazewicz, J.; Lenstra, J. K.; Rinnooy, K. A.: Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, ročník 5, 1983: s. 11–24.

[5] Branke, J.: *Evolutionary Optimization in Dynamic Environments*. Norwell, MA, USA: Kluwer Academic Publishers, 2001, ISBN 0792376315.

[6] Brucker, P.: *Scheduling Algorithms*. Springer, čtvrté vydání, 2004, 978-3540205241.

[7] Dantzig, B.; Thapa, N.: *Linear programming 1: Introduction*. Springer, 1997, 978-0387948331.

[8] Hartmann, S.: A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling. *Naval Research Logistics*, ročník 45, 1998: s. 733–750.

[9] Hartmann, S.: Project Scheduling with Multiple Modes: A Genetic Algorithm. *Annals of Operations Research*, ročník 102, 2001: s. 111–135.

[10] Hartmann, S.; Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, ročník 207, č. 1, 2010: s. 1 – 14, ISSN 0377-2217, doi:10.1016/j.ejor.2009.11.005.

[11] Horbach, A.: A Boolean satisfiability approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, ročník 181, 2008: s. 89–107.

[12] Jurová, M.: *Řízení výroby*. Akademické nakladatelství CERM, 2011, ISBN 978-80-214-4370-9.

[13] Klimek, M.: A genetic algorithm for the project scheduling with the resource constraints. *Ann. UMCS, Inf.*, ročník 10, č. 1, Leden 2010: s. 117–130, ISSN 1732-1360, doi:10.2478/v10065-010-0042-8.

[14] Kolisch, R.; Hartmann, S.: Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. *Project scheduling: Recent models, algorithms and applications*, 1999: s. 147–178.

[15] Kolisch, R.; Sprecher, A.: PSPLIB - A project scheduling library. *European Journal of Operational Research*, ročník 96, 1996: s. 205–216.

[16] Křen, M.: Effective Production planning and scheduling. *Proceeding of the 18th Conference STUDENT EEICT 2012*, 2012: s. 293 – 295.

[17] MESA International: *MESA Model* [online]. 2012 [cit. 2012-05-17]. Dostupné z: http://www.mesa.org/en/modelstrategicinitiatives/MESAModel.asp.

[18] Mingozzi, A.; Maniezzo, V.; Ricciardelli, S.; aj.: An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science*, ročník 44, č. 5, 1998: s. 714–729.

[19] Peringer, P.: *Modelování a simulace – Studijní opora*. FIT VUT v Brně, 2008.

[20] Peteghem, V. V.; Vanhoucke, M.: A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, ročník 201, č. 2, 2010: s. 409 – 418, ISSN 0377-2217, doi:10.1016/j.ejor.2009.03.034.

[21] Pinedo, M.: *Scheduling: theory, algorithms, and systems*. New York: Springer, čtvrté vydání, 2012, ISBN 978-1-4614-1986-0.

[22] Rudová, H.: PA167 Rozvrhování (přednášky) [online]. 2012 [cit. 2012-05-17]. Dostupné z: http://www.fi.muni.cz/~hanka/rozvrhovani/prusvitky/all.pdf.

[23] Schwarz, J.; Sekanina, L.: *Aplikované evoluční algoritmy – Studijní opora*. FIT VUT v Brně, 2006.

[24] Sepehr, P.; Mingzhou, J.: A genetic algorithm with neighborhood search for the resource-constrained project scheduling problem. *Naval Research Logistics*, ročník 58, 2011: s. 73–82.

[25] de Vonder, S. V.; Ballestín, F.; Demeulemeester, E.; aj.: Heuristic procedures for reactive project scheduling. *Computers and Industrial Engineering*, ročník 52, č. 1, 2007: s. 11 – 28, ISSN 0360-8352, doi:10.1016/j.cie.2006.10.002.

[26] Vonder, S. V. D.; Demeulemeester, E.; Herroelen, W.: Heuristic procedures for generating stable project baseline schedules. In *In proceedings of Third Euro Conference for Young OR researchers and practitioners ORP3*, 2005, s. 11–19.

[27] Zbořil, F.: *Přednáška č. 9 – Genetické algoritmy – předmět SFC (přednášky)*. FIT VUT v Brně, 2010.

Dodatek A

Obsah CD

Písemná zpráva — adresář thesis

V uvedeném adresáři je tato písemná technická zpráva ve formátu `pdf`. Také jsou zde obsaženy zdrojové texty této zprávy v systému LaTeX.

Zdrojové texty programů — adresář src

Tento adresář obsahuje všechny zdrojové soubory potřebné pro přeložení obou programů. Také je zde přiložena knihovna TinyXml použitá pro práci s XML. Soubor `README` obsahuje pokyny k přeložení.

Programová dokumentace — adresář refman

V tomto adresáři je obsažena vygenerovaná programová dokumentace celého projektu.

Výsledky experimentů — adresář results

Tento adresář obsahuje výsledky experimentů publikovaných v této práci. Rovněž jsou zde přiloženy automatické skripty a je možno tedy experimenty opakovat. Více informací je uvedeno v přiloženém souboru `README`.

Spustitelné aplikace — adresář apps

Tento adresář obsahuje přeložené spustitelné aplikace vytvořené v rámci této práce. Také jsou zde přiloženy testovací úlohy. Soubor `README` obsahuje krátký návod k aplikacím.