Mendelova univerzita v Brně
Provozně ekonomická fakulta

Optimalizace plánování SMT výroby pomocí měkkých výpočtů

Diplomová práce

Vedoucí práce:

Ing. Vít Ondroušek, Ph.D. **Bc. Jan Helán**

Brno 2016

Na tomto místě bych rád poděkoval panu Ing. Vítu Ondrouškovi, Ph.D. za jeho vedení, odborné rady a cenné podněty při vyhotovení této diplomové práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Optimalizace plánování SMT výroby pomocí měkkých výpočtů**
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 20. května 2016 _____

Abstract

Helán, J. Optimization of SMT production planning using soft computing. Diploma thesis. Brno: Mendel University in Brno, 2016.

This diploma thesis deals with production planning optimization using soft computing at an environment of a company. The theoretical part describes the production-scheduling problem and introduces possible methods of solution. The practical part focuses on the design and implementation of a web application that is used to optimize the production planning process. The application is implemented using the C# programming language and the ASP.NET framework. At the end of the thesis, tests are performed to verify the effectiveness of the bee colony optimization algorithm on test data.

Keywords

Production scheduling, production plan optimization, Bee Colony Optimization, BCO, web application, C#, ASP.NET.

Abstrakt

Helán, J. *Optimalizace plánování SMT výroby pomocí měkkých výpočtů*. Diplomová práce. Brno: Mendelova univerzita v Brně, 2016.

Diplomová práce se zabývá problematikou optimalizace plánování výroby v prostředí výrobní firmy za pomoci měkkých výpočtů. V teoretické části práce je popsán problém rozvrhování výroby a představeny možné metody řešení. Praktická část práce je zaměřena na návrh a tvorbu webové aplikace, která slouží k optimalizaci procesu plánování výroby. Webová aplikace je implementována pomocí frameworku ASP.NET a programovacího jazyka C#. Na závěr jsou provedeny testy ověřující efektivitu použitého optimalizačního algoritmu včelího roje na testovacích datech.

Klíčová slova

Rozvrhování výroby, optimalizace výrobního plánu, optimalizace včelím rojem, BCO, webová aplikace, C#, ASP.NET.

Obsah

1 Úvod a cíl práce 11
1.1 Úvod 11
1.2 Cíl práce 11

2 Rozvrhování 12
2.1 Rozvrhování výroby 13
2.2 Rozvrhování proudové výroby 15
2.3 Rozvrhování zakázkové výroby 17

3 Rozvrhování zakázkové výroby 19
3.1 Kritéria rozvrhu 19
3.2 Reprezentace dat 20
3.3 Problémy rozvrhování v reálných podmínkách 23
3.3.1 Montážní a distribuční operace 25
3.3.2 Čas přípravy, technologický čas a dopravní čas 26
3.4 Metody řešení 27
3.4.1 Evoluční algoritmy 29
3.4.2 Hejnové algoritmy 31

4 Popis problému a možná řešení 34
4.1 Analýza prostředí 34
4.2 Výběr algoritmu pro řešení 35
4.3 Specifikace cílů a metodika 36
4.3.1 Metodika 36
4.3.2 Specifikace cílů 37

5 Navržené řešení 38
5.1 Implementace problému plánování zakázkové výroby 38
5.2 Implementace optimalizačního algoritmu 41
5.2.1 Třída *Bee* 42
5.2.2 Třída *Hive* 42

5.2.3 Metoda *Solve* ... 43
5.3 Aplikace pro plánování výroby ... 46
5.3.1 Návrh aplikace ... 46
5.3.2 Databázová vrstva ... 47
5.3.3 Aplikační vrstva ... 49
5.3.4 Prezentační vrstva ... 52
5.4 Popis řešení z uživatelského hlediska ... 52
5.4.1 Formulář Data ... 52
5.4.2 Formulář Plan ... 55
5.5 Testování ... 57
5.5.1 První testovací sada dat ... 58
5.5.2 Druhá testovací sada dat ... 59
5.5.3 Třetí testovací sada dat ... 59
6 Zhodnocení výsledků 61
6.1 Shrnutí ... 61
6.2 Diskuze ... 61
7 Závěr 63
8 Literatura 64
9 Seznam použitých zkratek a značení 66
10 Seznam obrázků 68
11 Seznam tabulek 70
A Ilustrace optimalizace výrobního plánu 72
B Obsah přiloženého CD 73

1 Úvod a cíl práce

1.1 Úvod

V dnešní době je udržení konkurenceschopnosti jedním z primárních cílů každé uvědomělé společnosti. K dosažení a udržení tohoto stavu je nutné optimalizovat co nejvíce vnitropodnikových procesů a jedním z těchto procesů je, v případě všech výrobních firem, bezesporu plánování výroby.

Tento problém plánování výroby, jinak řečeno rozvrhování, je určen konečnou množinou úloh, které mají být v daném časovém období vyrobeny a konečným počtem strojů, na kterých lze požadované výrobky vyrábět. Každý výrobek musí projít zpracováním na více strojích, aby se dal považovat za finální produkt. Celkovou výrobu jednoho typu výrobku nazveme úloha (*job*) a úkony na jednotlivých strojích operacemi (*operation*). Na jakých strojích a v jakém pořadí se výrobek vyrábí, určuje technologie výroby jednotlivých výrobků. V teorii se můžeme setkat s dvěma hlavními druhy rozvrhování výroby – rozvrhování zakázkové a rozvrhování proudové výroby. U rozvrhování proudové výroby mají všechny výrobky identický technologický postup – pořadí strojů při výrobě finálního produktu je stejné, naproti tomu u zakázkové výroby může dojít k situaci, že každý finální produkt má různý technologický postup. Hlavním cílem rozvrhování je nalézt nejlepší časový plán – rozvrh (*schedule*) provádění operací, jinak řečeno, určit optimální pořadí operací na jednotlivých strojích. Jako kritérium rozvrhu se udává celková doba zpracování všech úloh (*makespan*) nebo minimalizace ztrát spojených s nedodržením termínu.

Úspěšná optimalizace plánování výroby znamená sestavení takového výrobního rozvrhu, který za určitou časovou jednotku dokáže vyprodukovat více výstupů, než tomu bylo před počátkem optimalizace. Problém optimalizování je možné řešit širokou množinou stochasticko-heuristických metod (tzv. měkkých výpočtů). Tyto metody jsou typické možností volby v průběhu výpočtu (např. v pořadí zpracování dat) a také určitou mírou nahodilosti. Mnohé z těchto metod nalézají inspiraci v procesech známých z biologie.

Tato diplomová práce se zabývá právě problémem rozvrhování zakázkové výroby a jejího optimalizování pomocí měkkých výpočtů v prostředí skutečné mezinárodní výrobně – montážní firmy.

1.2 Cíl práce

Cílem této práce je vyřešit problém rozvrhování výroby pomocí vhodné optimalizační strategie, která bude uživatelům zprostředkována pomocí webové aplikace, implementované na základě požadavků konkrétní výrobní firmy. K dosažení tohoto cíle bude nutné rozebrat existující metody přístupu k rozvrhování výroby, vybrat vhodný způsob optimalizace problémů tohoto typu a v neposlední řadě analyzovat vlastní prostředí firmy.

2 Rozvrhování

Než přejdeme k vlastnímu tématu rozvrhování výroby, určitě je vhodné se nejdříve stručně seznámit s tématem rozvrhování všeobecně.

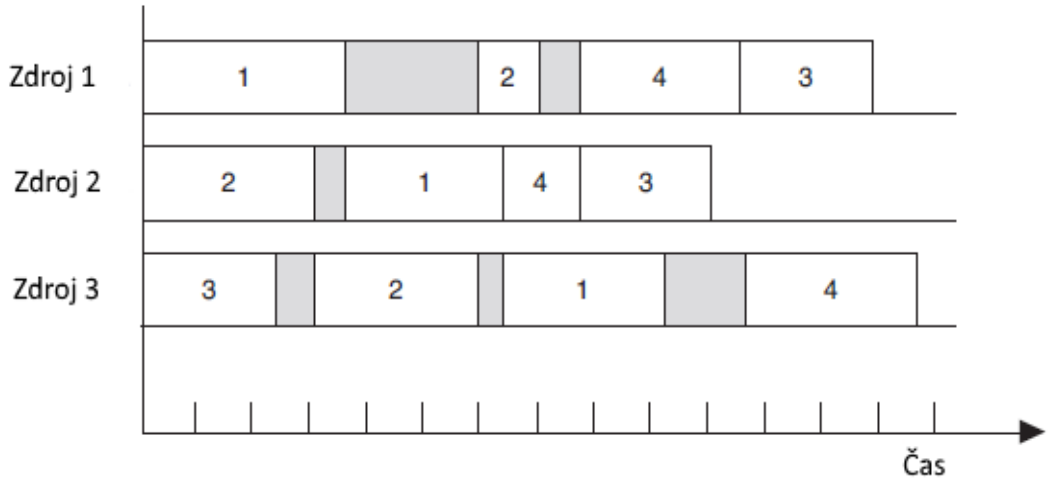
Rozvrhování můžeme definovat jako přiřazování omezených zdrojů aktivitám. Pod aktivitami si můžeme představit úlohy v oboru informatiky, postupy při strojírenských projektech, předměty na univerzitě nebo operace ve výrobním procesu. Zdroje představují dělníci, učitelé, stroje apod. (Robert, Vivien, 2010, s. 1).

Jedním ze základních plánovacích problémů je tzv. problém plánování projektu s omezenými zdroji (*RCPSP – resource-constrained project scheduling problem*). Tento problém můžeme definovat jako posloupnost n aktivit $j = 1, \ldots, n$ a r obnovitelných zdrojů (strojů) $k = 1, \ldots, r$. Počet dostupných zdrojů je v rámci problému neměnný. Každá aktivita j je zpracovávána po určitý čas p_j. Během této doby je obsazen určitý počet jednotek r_{jk} zdroje k. Dále mohou být definovány určitá precedenční omezení mezi aktivitami. Toto omezení je určeno relací $i \rightarrow j$, což znamená, že aktivita j nemůže začít, dokud není hotová aktivita i. (Robert, Vivien, 2010, s. 2)

Úkolem rozvrhování je stanovit počáteční časy S_j aktivit $j = 1, \ldots, n$ takovým způsobem, že:

- V každém čase t je počet požadavků na zdroje menší nebo roven dostupným zdrojům.
- Daná precedenční omezení jsou splněna.
- Celkový výrobní čas je minimalizován.

Vektor počátečních časů $S = (S_j)$ poté tvoří výsledný rozvrh projektu. Rozvrh je proveditelný, pokud jsou splněny všechny omezení. Výsledný rozvrh lze vizualizovat několika způsoby. Nejjednodušším a nejrozšířenějším modelem je Ganttův diagram. Ve své běžné podobě znázorňuje využití zdrojů v průběhu času. Jednotlivé zdroje jsou znázorněny na vertikální ose a čas na horizontální ose.



Obr. 1 Ganttův diagram

Diagram, který je znázorněn na obrázku 1.1, nám pomáhá představit si výsledný rozvrh, protože jasně zobrazuje jednotlivé zdroje a k nim přiřazené úlohy.

2.1 Rozvrhování výroby

V průběhu kapitoly rozvrhování výroby se budeme často setkávat s následujícími pojmy, je vhodné si je tedy nejprve přesně definovat.

- Stroj (*machine*) je zařízení, na kterém se zpracovávají jednotlivé operace. V kontextu *RCPSP* mluvíme o zdroji.
- Úloha (*job*) je posloupnost operací, které je třeba vykonat k dosažení finálního výstupu. Můžeme se setkat také s označením práce. V rámci *RCPSP* se jedná o termín aktivita.
- Operace (*operation*) je základní technologický úkon.
- Doba zpracování (*processing time*) je čas potřebný na zpracování úlohy *j*.
- Celkový výrobní čas (*makespan*) je čas ukončení poslední zpracovávané úlohy.

Nejjednodušší plánovací problém je ten, ve kterém je výrobní rozvrh určen pouze pořadím úloh. Tento problém může nastat, uvažujeme-li pouze jeden zdroj (stroj) a pokud je doba zpracování každé úlohy deterministická. Ač se tento problém může zdát triviální, je velmi důležitý, protože představuje základ pro pochopení složitějších problémů a často se vyskytuje jako část složitějšího plánovacího problému. Jako příklad můžeme uvést řešení úzkého místa (*bottleneck*) v případě více strojových problémů, ve kterém jeho vyřešení ovlivní celkovou podobu rozvrhu.

V návaznosti na limitace jednoho stroje je tento problém charakterizován těmito podmínkami (Baker, Trietsch, 2009, s. 11):

- C1. Každá úloha se skládá pouze z jedné operace.

- C2. V čase 0 existuje n úloh dostupných pro zpracování.
- C3. Stroj může zaráz zpracovávat nanejvýš jednu úlohu.
- C4. Časy přípravy úloh na zpracování jsou nezávislé na pořadí zpracovávání úloh a jsou zahrnuty v čase zpracování.
- C5. Všechny úlohy jsou známé dopředu.
- C6. Stroj je neomezeně dostupný (nemá přestávky / výpadky).
- C7. Jakmile začne zpracování úlohy, nemůže dojít k přerušení zpracovávání.
- C8. Pokud existuje úloha čekající na zpracování, stroj není nikdy nečinný.
- C9. Existuje $n!$ různých výsledných rozvrhů, kde n je počet úloh.

Abychom mohli podrobněji charakterizovat výsledný rozvrh a jednotlivé úlohy, používáme informace vyplývající z výsledného rozvrhu. Jednou z hlavních informací, kterou nám výsledný rozvrh poskytne je:

- Čas dokončení C_j (*Completion time*) – Čas, ve kterém je dokončeno zpracování úlohy *j*.

Další kvantifikovatelná měřítka jsou obvykle vyvozeny z času dokončení. Mezi dvě nejdůležitější zařadíme:

- Čas v systému F_j (*Flowtime*) – Čas, který úloha j stráví v systému. Vypočteme jej jako rozdíl času dokončení a času, ve který je tato úloha připravena ke zpracování.

$$F_j = C_j - r_j \tag{1}$$

- Zpoždění L_j (*Lateness*) – Čas, o který výsledný čas dokončení úlohy *j* překročil požadovaný čas dokončení této úlohy (tzv. *due date*).

$$L_j = C_j - d_j \tag{2}$$

Tyto kvantifikátory nám poskytují dvojí službu. Čas v systému měří odezvu systému na jednotlivé požadavky a znázorňuje časový interval od přijetí požadavku po jeho vyřízení. Zpoždění měří shodu rozvrhnuté úlohy s jejím původním požadovaným datem ukončení. Toto zpoždění je často spojováno s penalizacemi od zadavatele. Pokud zpoždění dosáhne negativní hodnoty, znamená to, že úloha byla dokončena dříve, než bylo požadováno. Tento výsledek ale nepřináší žádný užitek, proto se často místo kvantifikátoru zpoždění používá kvantifikátor nedochvilnost T_j (*Tardiness*), který zahrnuje pouze kladné výskyty hodnot zpoždění.

Rozvrh je obecně hodnocen agregováním kvantifikátorů, které obsahují informace o všech úlohách, do jedno-úrovňového měřidla kvality. Pro příklad uvažujme, že n úloh má být naplánováno. Agregované měřidlo kvality je potom definováno následovně:

- Celkový čas v systému:

$$F = \sum_{j=1}^{n} F_j \tag{3}$$

- Celková nedochvilnost:

$$T = \sum_{j=1}^{n} T_j \tag{4}$$

- Nejdelší čas v systému:

$$F_{\max} = \max_{1 \le j \le n} \{F_j\} \tag{5}$$

- Největší nedochvilnost:

$$T_{\max} = \max_{1 \le j \le n} \{T_j\} \tag{6}$$

- Nejdelší čas dokončení:

$$C_{\max} = \max_{1 \le j \le n} \{C_j\} \tag{7}$$

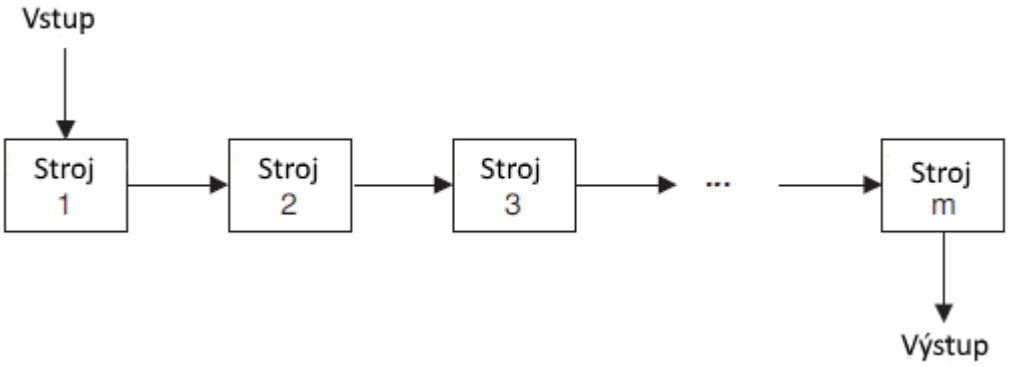
Pro měřítko nejdelší čas dokončení se také často udává pojmenování *makespan*.

2.2 Rozvrhování proudové výroby

Rozvrhování proudové výroby (*flow shop*) je složitějším plánovacím problémem, ve kterém existuje více sériově zapojených strojů. V tomto modelu úlohy plynou (*flow*) od počátečního stroje, přes libovolný počet následujících až po konečný stroj. Až po zpracování na tomto stroji je úloha kompletní.

Při tomto rozvrhování jsou úlohy rozdělené na jednotlivé úkoly zvané operace, kdy každá tato operace je prováděna na jiném stroji. Jinak řečeno, úloha je zde množina operací s danou precedenční strukturou. V této struktuře má kromě první operace každá operace přesně jednoho předchůdce a kromě poslední operace také přesně jednoho následovníka.

Proudová výroba obsahuje *m* různých strojů a v ryzím modelu proudové výroby počítáme s faktem, že každá úloha se skládá z *m* operací prováděných na různých strojích. Obrázek č. 2 znázorňuje ryzí model proudové výroby.

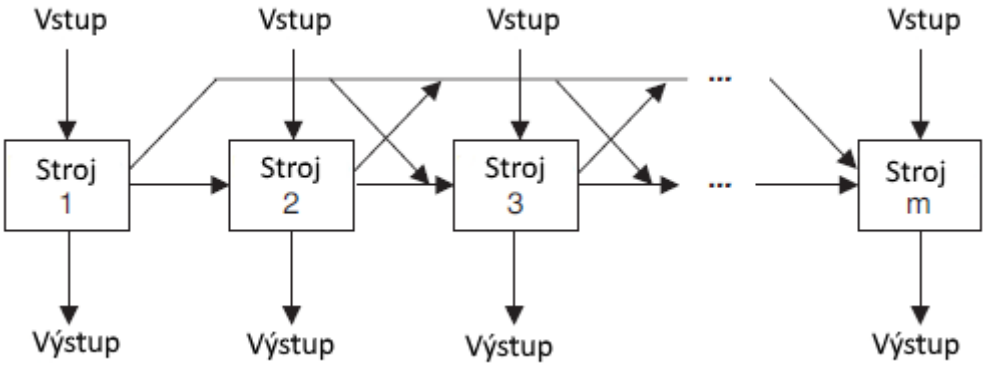


Obr. 2 Ryzí model proudové výroby

Ve většině případů se ale setkáme spíše s obecným modelem proudové výroby. V tomto případě, kde je opět *m* strojů, platí následující upřesnění:

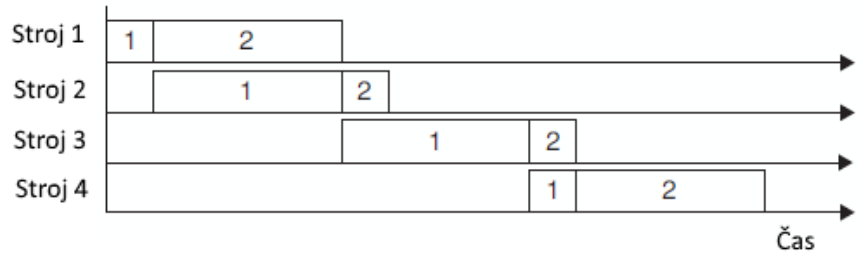
- Každá úloha se oproti ryzímu modelu může skládat z méně než *m* operací.
- Operace se nemusí vždy provádět na přímo sousedícím stroji.
- Úvodní a konečná operace nemusí být prováděna vždy na stroji 1, respektive stroji *m*.

Nicméně tento obecný model stále pracuje s jednosměrným proudem výroby a lze vyjádřit jako ryzí model proudové výroby s nulovými operacemi (operacemi, které netrvají žádný čas). Obecný model je vyobrazen na následujícím obrázku.



Obr. 3 Obecný model proudové výroby

Omezující podmínky pro tento model jsou stejné jako pro plánování výroby pro jeden stroj s jednou výjimkou a to, že u více strojového plánování výroby může dojít k situaci, že bude stroj v nečinnosti kvůli dosažení lepšího rozvrhu. Příklad rozvrhu s prostoji na strojích vidíme na obrázku č. 4.

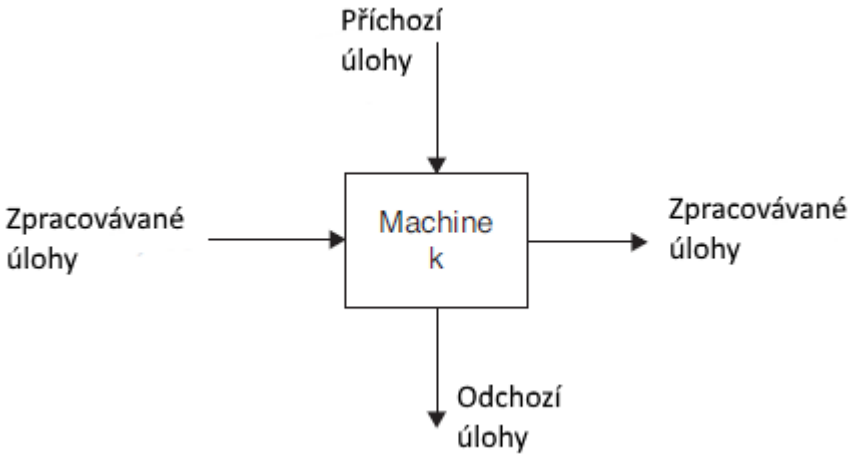


Obr. 4 Rozvrh proudové výroby s prostoji

V rozvrhování výroby pro jeden stroj je nutné projít až $n!$ různých variant pro dosažení optimálního řešení. U rozvrhování pro více strojů je při m strojích a n úlohách nutné prozkoumat až $(n!)^m$ různých rozvrhů (Baker, Trietsch, 2009, s. 227). Pro řešení tohoto problému se proto používají optimalizační metody a heuristické přístupy, které budou probrány v dalších kapitolách této práce.

2.3 Rozvrhování zakázkové výroby

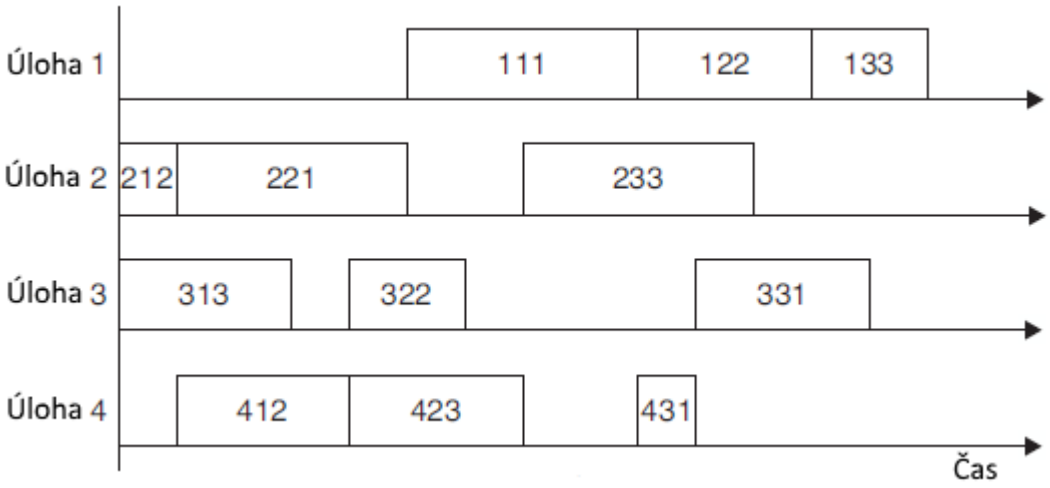
Rozvrhování zakázkové výroby je typický plánovací problém, ve kterém je množina m strojů a n úloh. Každá úloha je tvořena libovolným počtem operací se stejnou precedenční strukturou jako u proudové výroby. Na rozdíl od proudové výroby se rozvrhování zakázkové výroby liší v jednom důležitém aspektu – proces zpracování úloh není jednosměrný. Ačkoliv může mít úloha libovolný počet operací, nejběžnější případ je, že má stejný počet operací jako je strojů. Nicméně, je možné připustit situaci, kdy úloha obsahuje více operací pro stejný stroj, nebo že některý stroj vynechá. Proces zpracování výroby je znázorněn na následujícím obrázku.



Obr. 5 Model funkce stroje u zakázkové výroby

Na rozdíl od proudové výroby v tomto systému není žádný úvodní nebo koncový stroj, který by prováděl pouze první, respektive poslední operaci. Na obrázku č. 6 je znázorněn problém zakázkové výroby z pohledu úloh a operací.

Na ose Y jsou znázorněny úlohy tvořené jednotlivými operacemi. Každá operace je jednoznačně určená pomocí trojice celočíselných hodnot. Tato trojice se skládá z čísla úlohy, pořadí zpracování v úloze a číslo stroje, na kterém se tato operace zpracovává. Například označení operace 423 znamená, že patří do úlohy s číslem 4, zpracovává se jako druhá v pořadí úlohy a že je zpracovávána na stroji 3. Skutečnost, že se zpracovává druhá v pořadí, znamená, že nemůže být zpracována, dokud neskončí zpracovávání operace 412.



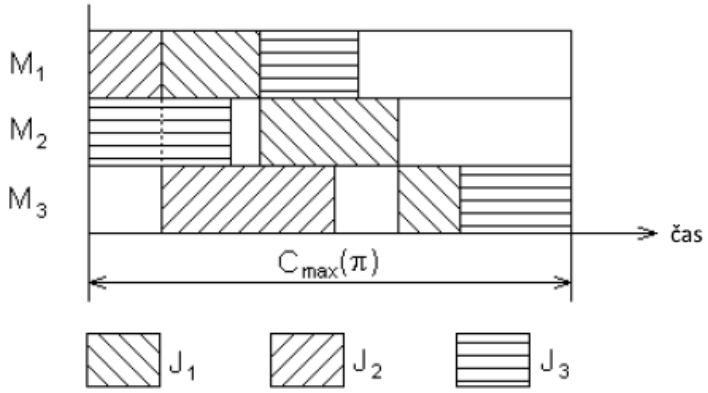
Obr. 6 Úlohy a operace v problému zakázkové výroby

Podobně jako u plánování proudové výroby i u tohoto problému řešíme hledání alespoň částečně optimálního rozvrhu z $(n!)^m$ možných řešení (Baker, Trietsch, 2009, s. 327). Opět je tedy nutné použít optimalizace a heuristické a jiné přístupy k hledání optimálního rozvrhu. Vzhledem k zaměření této diplomové práce právě na rozvrhování zakázkové výroby bude celý problém plánování zakázkové výroby rozebrán podrobněji v následující kapitole.

3 Rozvrhování zakázkové výroby

Jak bylo řečeno v předchozí kapitole, problém rozvrhování zakázkové výroby je formálně definován jako množina úloh (*jobs*) $J = \{J_1, \dots, J_n\}$ a množina strojů (*machines*) $M = \{M_1, \dots, M_m\}$. Naším úkolem je naplánovat zpracování všech n úloh, přičemž každá úloha se skládá z určitého počtu operací. Doby trvání jednotlivých operací u dvou nestejných úloh mohou být různě dlouhé a tyto doby trvání mohou být i nulové. Pořadí operací v rámci úloh je předem definováno jejich technologickým postupem. Toto pořadí nelze měnit. Množina operací všech úloh je definována jako $O = \{O_1, \dots, O_o\}$. Doba trvání k-té operace je p_k. Pro zpracování každé operace je nutné ji jednoznačně přiřadit stroj z množiny *M*. Na jednotlivém stroji je možné současně zpracovávat nejvýše jednu operaci jedné úlohy. Zpracovávání operací nemůže být přerušeno. Zpracování všech úloh tedy znamená vykonání všech jejích operací v určeném pořadí na daných strojích (Blazewicz, 1996).

Výsledný rozvrh π (*schedule*) můžeme formulovat jako přiřazení libovolných proveditelných pořadí operací na všech strojích z množiny *M*. Příklad proveditelného výsledného rozvrhu pro *n = 3* úlohy a *m = 3* stroje se nachází na následujícím obrázku.



Obr. 7 Proveditelný rozvrh pro n = 3 úlohy a m = 3 stroje

Úkolem rozvrhování je nalézt optimální rozvrh π, to znamená, určit nejvhodnější pořadí úloh na všech strojích. Jednotlivá kritéria kvality rozvrhu budou rozebrány v příští podkapitole.

3.1 Kritéria rozvrhu

Každá firma může mít jinak stanové priority výrobního procesu a podle toho je třeba přizpůsobovat výsledný rozvrh. Toto subjektivní hodnocení označíme pojmem kritérium rozvrhu. Nejčastěji používaným kritériem je co nejmenší čas dokončení všech úloh (*makespan*). Pokud označíme symbolem r_k nejdříve možný termín začátku k-té operace a symbolem c_k termín dokončení k-té operaci, pak platí:

$$c_k \geq r_k + p_k \tag{8}$$

Cílem rozvrhování je poté minimalizovat největší z těchto hodnot:

$$\text{Minimalizace } C_{\max}(\pi) = \max_{1 \leq k \leq o} \{c_k\} \tag{9}$$

Jiné kritérium použijeme v situaci, kdy firma zná požadovaný termín dokončení určité výrobní dávky nebo jednotlivé úlohy. V tomto případě se snažíme o nalezení rozvrhu, které minimalizuje ztráty s nedodržením termínů. Pokud požadovaný termín dokončení úlohy označíme T_i a koeficient ztráty spojené s opožděným dokončením *i*-té úlohy z_i, pak toto kritérium můžeme vyjádřit následujícím způsobem:

$$\text{Minimalizace } \sum_{i=1}^{n} \left[z_i \max\left\{0, c_{Last(i)} - T\right\} \right] \tag{10}$$

V tomto výrazu použijeme funkci *Last(i)*, která nám vrátí pozici poslední operace i-té úlohy.

V případě plánování procesu výroby systémem *just in time (JIT)* je kritérium rozvrhu ještě náročnější a požaduje mimo jiné také minimalizaci času předčasného dokončení úloh. Důvodem mohou být například náklady spojené s nutností skladování předčasně dokončených výrobků. Pro koeficient nákladu spojeného s předčasným dokončením *i*-té úlohy použijeme označení s_i. Výsledné kritérium je formálně definováno:

$$\text{Minimalizace } \sum_{i=1}^{n} \left[z_i \max\left\{0, c_{Last(i)} - T\right\} + s_i \max\left\{0, T_i - c_{Last(i)}\right\} \right] \tag{11}$$

Pro dosažení účelného rozvrhu, který splňuje nejenom kritéria představená výše, je zapotřebí použít specifické heuristicko-stochastické metody aplikované na vhodnou reprezentaci problému rozvrhování. (Majer, 2001).

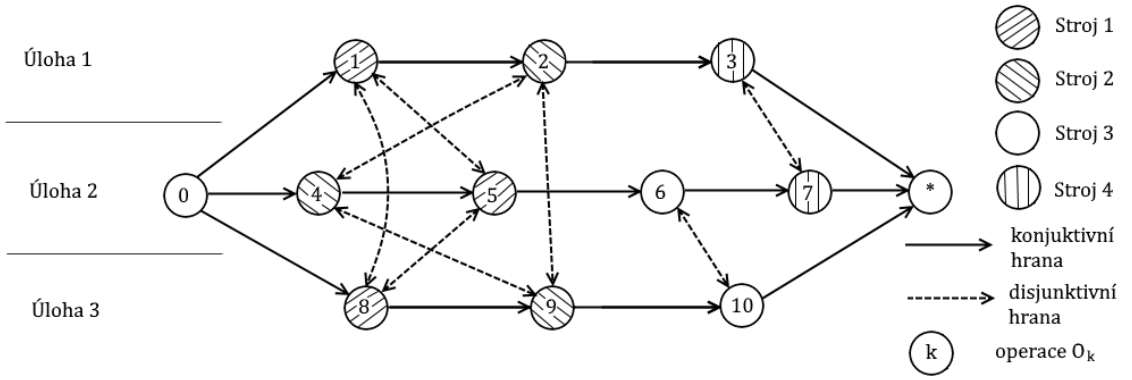
3.2 Reprezentace dat

Reprezentaci dat si můžeme představit jako funkci, která znázorňuje posloupnosti operací na jednotlivých strojích jako specifický druh seznamu, nad kterým lze dobře provádět složité výpočty obsažené v optimalizačních algoritmech. Do dnešní doby bylo objeveno mnoho způsobů znázornění neboli reprezentací problému rozvrhování zakázkové výroby. Nejčastěji se setkáváme s reprezentací v podobě *disjunktivního* grafu nebo pomocí *preferenčního seznamu* (Šeda, 1999).

Preferenční seznam je takový seznam, který je složen z *m* řetězců. Ke každému stroji je přiřazen jeden řetězec obsahující čísla operací prováděných právě na tomto stroji. Řetězce neobsahují informaci o pořadí operací na strojích, nýbrž informaci o preferencích jednotlivých operacích na stroji. Řetězce si můžeme představit jako fronty, ze kterých jsou postupně vybírány dostupné operace. V případě rozhodování, kterou operaci vybrat přednostně, zvolíme tu, která je v preferenčním seznamu

dříve na řadě. Kvůli následnému použití optimalizačních algoritmů se jako výhodnější reprezentace problému rozvrhování jeví reprezentace pomocí *disjunktivního grafu*.

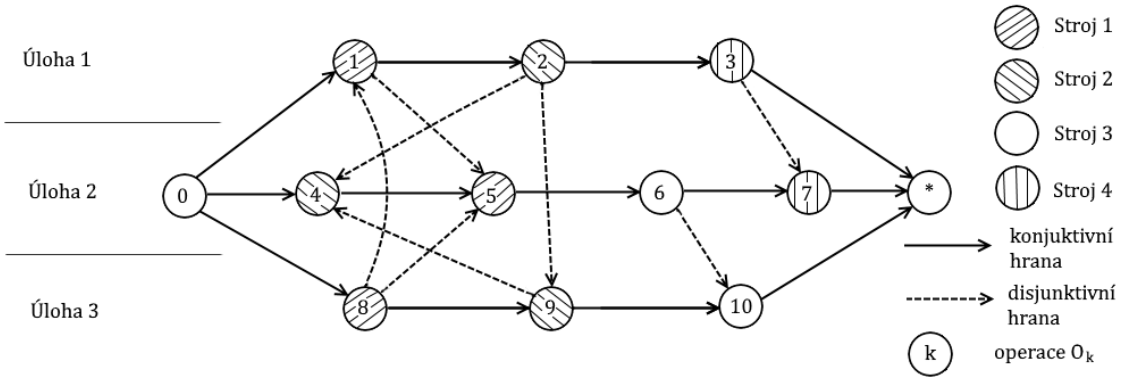
Disjunktivní graf můžeme definovat jako orientovaný graf *G*, tvořený množinou uzlů *V* a dvěma množinami hran *C* a *D*. Uzly v množině *V* představují všechny operace, které musí být vykonány v *n* úlohách. Množina *V* ještě navíc obsahuje dva speciální neutrální uzly – počáteční uzel *0* a koncový uzel *. Počáteční uzel obsahuje *n* výchozích hran vedoucích do prvních operací *n* úloh. Do koncového uzlu poté míří *n* vstupních hran z *n* posledních operací v úlohách. Množina *C* obsahuje takzvané konjunktivní hrany, které reprezentují posloupnost operací v úloze. Operace, patřící do různých úloh, prováděné na stejném stroji, jsou spojeny neorientovanými disjunktivními hranami. Tyto disjunktivní hrany poté tvoří *m* klik[1], pro každý stroj přesně jednu. Uzly (operace) v jedné klice musí být zpracovány na jednom stejném stroji. Všechny uzly v grafu mají ohodnocení, které se rovná době zpracování (*processing time*) daného uzlu (operace). Jedinými výjimkami jsou počáteční a koncový uzel, které jsou ohodnoceny nulovou velikostí. Takovýto graf značíme $G = (V, C \cup D)$. Příklad reprezentace disjunktivním grafem v případě problému s třemi úlohami a čtyřmi stroji vidíme na následujícím obrázku.



Obr. 8 Disjunktivní graf

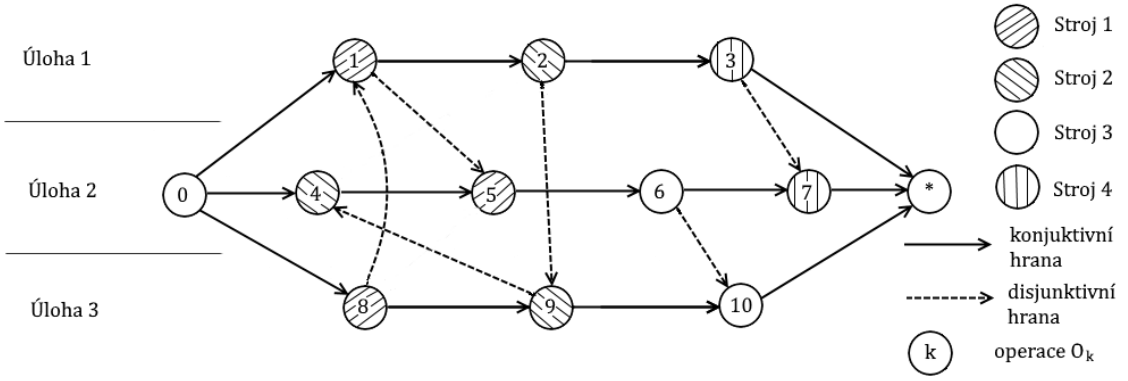
Výsledný výrobní rozvrh je sestaven určením pořadí operací na jednotlivých strojích. Určení pořadí operací docílíme stanovením orientace disjunktivních hran. Množinu všech orientovaných disjunktivních hran *S* poté nazýváme *úplnou selekcí* (viz obrázek č. 9). Výsledná úplná selekce nám definuje přípustný rozvrh π jen v případě, že výsledný graf $G(\pi) = (V, C \cup S)$ je acyklický. Rozvrh, znázorněný na obrázku č. 9 můžeme formálně zapsat $\Pi = \{(8,1,5), (2,9,4), (3,7), (6,10)\}$, kde jednotlivé *n*-tice představují pořadí operací na strojích. Nejdelší cesta z počátečního uzlu do koncového uzlu nám poté udává celkovou dobu zpracování všech úloh. Tuto nejdelší cestu označíme kritickou cestou $u(\pi)$.

[1] Klika je pojem z teorie grafů, který popisuje takový podgraf, který je úplným grafem. To znamená, že se jedná o takový graf, ve kterém jsou každé dva uzly spojené hranou



Obr. 9 Orientovaný disjunktivní graf – úplná selekce

Před určením kritické cesty $u(\pi)$ je vhodné vytvořený acyklický orientovaný graf zjednodušit. Můžeme si všimnout, že kromě počátečního a koncového uzlu, má každý uzel grafu $G(\pi)$ maximálně dva následníky a maximálně dva předchůdce, a z toho vždy nejvýše jednoho bezprostředního následníka na stejném stroji. Vypuštěním těch disjunktivních hran, které nespojují bezprostředně po sobě následující operace, nenarušíme pořadí operací na strojích a vznikne zjednodušený orientovaný disjunktivní graf, jak vidíme na následujícím obrázku č. 10.



Obr. 10 Zjednodušený orientovaný disjunktivní graf

Po vytvoření zjednodušeného disjunktivního grafu určíme kritickou cestu tohoto grafu. Hlavní kroky algoritmu pro nalezení kritické cesty lze dle Majera (2001, str. 4) formulovat následovně:

1. Stanovíme $r_0 = 0$ a označíme indexem všechny uzly takovým způsobem, že pokud existuje přímá cesta z uzlu *i* do uzlu *j*, pak index uzlu *i* < index uzlu *j*.
2. Všechny uzly poté procházíme vzestupně dle indexů a počítáme nejdříve možné termíny začátků operací. Nejdříve možný termín začátku *j*-té operace lze spočítat jako $r_j = \max_{k \in P(j)} \{r_k + p_k\}$, kde množinu uzlů, z nichž do uzlu *j* vstupují orientované hrany, označíme jako $P(j)$.

3. Po zpracování všech uzlů jsme schopni vyčíst délku kritické cesty $C_{max}(\pi)$ z hodnoty nejdříve možného začátku koncového uzlu r_*. Termín nejpozdějšího možného konce koncového uzlu je $d_* = r_*$, protože $p_* = 0$.
4. Nyní procházíme všechny uzly sestupně podle indexů a počítáme nejpozději možné termíny ukončení operací. Nejpozději možný konec *j*-té operace je roven $d_j = \min_{k \in N(j)} \{d_k - p_k\}$, kde $N(j)$ je množina uzlů, do kterých z uzlu *j* vstupují orientované hrany.
5. Posledním krokem algoritmu je stanovení kritických uzlů. Nejdříve si vypočítáme prodlevu každého uzlu $w_k = d_k - (r_k + p_k)$. Všechny uzly, pro které platí, že $w_k = 0$, označíme jako kritické uzly. Posloupnost kritických uzlů následně tvoří kritickou cestu. Graf může obsahovat i více kritických cest.

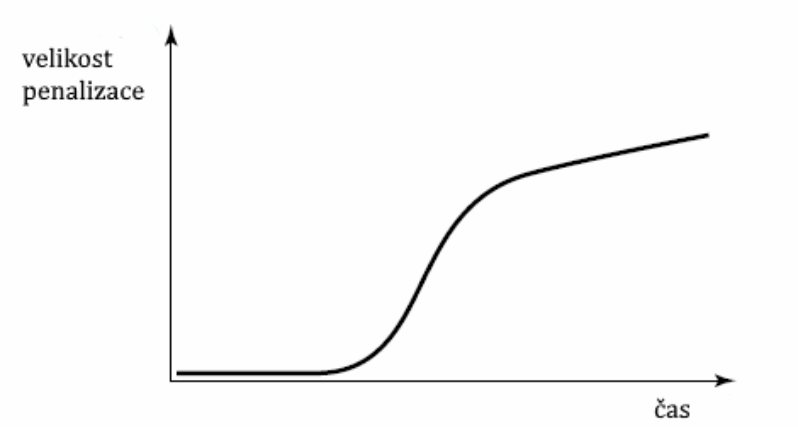
Výsledná délka kritické cesty $C_{max}(\pi)$ odpovídá celkové době zpracování všech úloh v rozvrhu.

3.3 Problémy rozvrhování v reálných podmínkách

Problém rozvrhování výroby ve skutečných podmínkách se bohužel obvykle velmi liší od teoretických modelů. Není v našich silách vyčíslit všechny rozdíly mezi problémy skutečného světa a teoretickými modely, neboť skoro každý reálný problém má své specifické výstřednosti. Nicméně, některé rozdíly jsou společné pro většinu problémů a proto stojí za uvedení (Pineda, 2008, str. 428).

- Teoretické modely předpokládají, že existuje *n* úloh požadovaných k rozvržení, a že po skončení rozvrhu došlo k dokončení právě *n* úloh. V praxi ale dochází k tomu, že v libovolném neurčitém čase je v systému právě *n* úloh a průběžně jsou přidávány nové úlohy. To znamená, že rozvrhování současných *n* úloh musí být prováděno bez znalosti budoucích plánů a z tohoto důvodu musí být zavedena některá opatření. Mezi tato opatření můžeme zavést například zařazení zdánlivě zbytečných prostojů strojů (*slack times*), díky kterým ovšem můžeme pokrýt případnou várku nových úloh nebo poruchu některého stroje.
- Teoretické modely obvykle nezahrnují do úvahy problém přerozvrhování (*resequencing*). V praxi se ale tento problém vyskytuje vcelku často, například, že existuje naplánovaný rozvrh rozvržený na základě jistých požadavků a dojde k nějaké nečekané události, která vyžaduje menší nebo větší změny v existujícím rozvrhu. Tento proces změny rozvrhu, někdy označován jako *reaktivní rozvrhování,* musí splňovat dodatečné požadavky. Jedním z těchto požadavků může být například zachování stávajícího rozvrhu v co největší míře i za cenu toho, že nebude dosaženo optimálního rozvrhu. Z tohoto důvodu může být výhodné sestavovat výrobní plány, které jsou takzvaně *robustní*, tj. že přerozvrhováním dojde k pouze malým změnám ve stávajícím výrobním plánu. Protiklad robustního výrobního plánu se označuje jako *křehký* rozvrh.

- Prostředí strojů v reálném světě je většinou také mnohem více komplikované, než předpokládají teoretické modely, protože na jednotlivé stroje se mohou vázat různá omezení a výjimky, se kterými teoretické modely nepočítají.
- V teoretických modelech jsou váhy (resp. *priority*) úloh neměnné, tzn., nemění se v průběhu výrobního plánu. V praxi běžně dochází k situaci, že původně úloha s nízkou prioritou se stane vysoce prioritní úlohou.
- Teoretické modely neberou v úvahu *preference* úloh. To znamená, že v modelech úloha buďto může (hodnota 1) nebo nemůže (hodnota 0) být zpracována na určitém stroji. Ve skutečnosti je možná situace, že úloha může být zpracována na nějakém stroji, ale je rozumnější ji zpracovat na jiném stroji. Zpracování na méně preferovaném stroji by proběhlo, pokud by došlo například k poruše vhodnějšího stroje.
- Většina teoretických modelů počítá s nepřetržitou dostupností strojů. V praxi ale stroje obvykle nepřetržitě dostupné nejsou. Existuje mnoho důvodů, díky kterým se stane stroj nedostupný. Jako příklad můžeme uvést absenci obsluhy kvůli střídání směn, nebo poruchu na stroji. K minimalizování možnosti poruchy se do rozvrhu často přidává preventivní údržba.
- Zajímavostí je, že na rozdíl od teoretickými modely předpokládaného lineárního tvaru křivky penalizací za zpoždění, je ve skutečnosti tvar této křivky více podobný tvaru písmene „S“ (viz obrázek 11).



Obr. 11 Tvar křivky penalizace v praxi

- Většina teoretických modelů předpokládá pouze jedno kritérium. V reálném světě je běžně požadováno mnoho kritérií zároveň. A nejenom, že existuje více kritérií, v praxi se také setkáváme s rozdílnými prioritami těchto kritérií, které se navíc mohou měnit v průběhu času. Častou kombinací kritérií je minimalizace zpoždění a minimalizace času na nastavení stroje (*setup time*).

- Další aspekt, který teoretické modely vůbec nezohledňují, je proces učení nebo naopak úpadku. Ve výrobě, kterou vykonávají ručně lidé, dochází díky zdokonalování techniky výroby (resp. automatizování provádění činnosti) postupem času k redukování průměrného času potřebného k vykonání dané operace. A naopak, ve výrobě prováděné na strojích, může dojít k zpomalení procesu výroby kvůli postupnému stárnutí a opotřebování stroje.

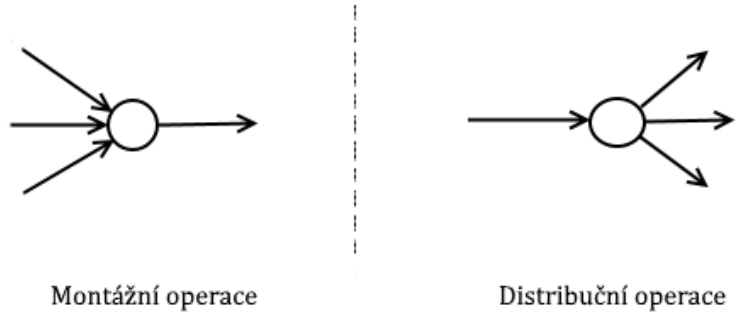
Kromě výše zmíněných rozdílností se v reálném světě setkáváme ještě s několika dalšími důležitými rozdíly, které budou podrobněji popsány v následujících podkapitolách.

3.3.1 Montážní a distribuční operace

Ve výrobním procesu často dochází k situaci, že je nutné výrobek kompletovat z více částí a tento složený výrobek pokračuje dál procesem, dokud nevznikne konečný výrobek. Tato kompletace se ve výrobním procesu nazývá montáží.

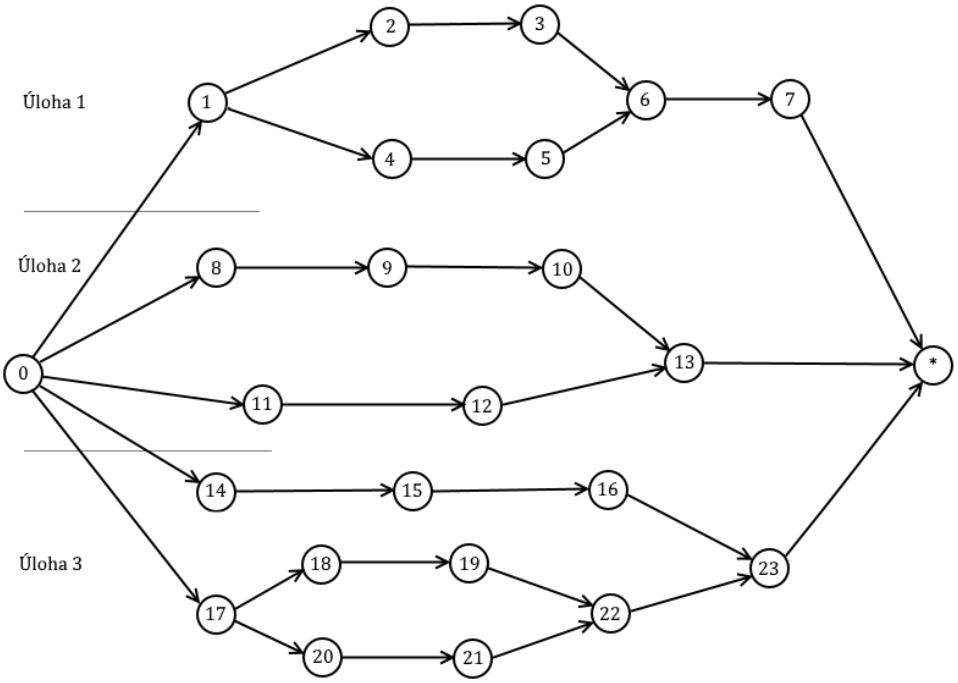
Opačným případem je situace, kdy se vlivem určité operace rozdělí výrobek na více částí a tyto části se distribuují na další nadcházející operace. Toto rozdělení výrobku, například polotovaru na více částí, označíme distribuční operací. Často se poté při plánování dostaneme do situace, ve které je úkolem rozvrhnout výrobu, která obsahuje současně jak montážní tak distribuční operace.

Montážní a distribuční operace lze snadno integrovat do disjunktivního grafu. Ve standardní podobě disjunktivního grafu platí, že každý uzel, kromě počátečního a koncového, má pouze jednoho následníka a jednoho předchůdce. Montážní operace můžeme snadno namodelovat tak, že připustíme, že daný uzel může mít více předcházejících uzlů spojený s tímto uzlem konjunktivními hranami. Podobným způsobem lze reprezentovat distribuční operaci, pro kterou bude platit, že z ní vychází více výstupních konjunktivních hran do následujících operací. Obě situace jsou znázorněny na následujícím obrázku.



Obr. 12 Montážní a distribuční operace

Disjunktivní graf obohacený o montážní nebo distribuční operace nazveme rozšířeným disjunktivním grafem. Příklad rozšířeného disjunktivního grafu s třemi úlohami prezentuje obrázek č. 13. Pro přehlednost jsou v grafu skryty disjunktivní hrany mezi uzly.



Obr. 13 Disjunktivní graf rozšířený o montážní a distribuční operace

Jednou úlohou v tomto grafu rozumíme množinu operací, které jsou propojeny pomocí konjunktivních hran. Na obrázku č. 13 vidíme tedy rozšířený disjunktivní graf složený ze tří úloh. První úloha obsahuje operace jedna až sedm, druhá úloha operace 8 až třináct a třetí úloha operace čtrnáct až dvacet tři. Mezi montážní operace můžeme zařadit operace číslo šest, třináct a dvacet tři. Do množiny distribučních operací patří operace číslo jedna a sedmnáct.

3.3.2 Čas přípravy, technologický čas a dopravní čas

Většina teoretických modelů zabývající se rozvrhováním zakázkové výroby bere v potaz pouze jeden daný výrobní čas operace (*process time*), který označujeme p_k. V reálném prostředí v sobě doba zpracování k-té operace skrývá některé další časy, se kterými musíme počítat. Jedná se o čas přípravy, technologický čas a dopravní čas. V disjunktivním grafu je výrobní čas operace znázorněn uzlovým ohodnocením, naproti tomu, tyto doplňkové časy můžeme zobrazit jako ohodnocení jednotlivých hran mezi uzly.

- Čas přípravy operace (*setup time*) je takový časový úsek, ve kterém ještě není úplně nezbytné, aby byla dokončena předcházející operace stejné úlohy na od-

lišném stroji. Tento čas se využívá například k seřízení stroje (*setup*) před začátkem výroby odlišného výrobku. Čas přípravy operace lze formálně definovat následovně (Majer, 2003):

Nechť každá disjunktivní hrana $\{x, y\}$ je vyjádřena dvěma opačnými orientovanými hranami (x, y) a (y, x). Předmětem optimalizace času přípravy je poté vybrat pouze jednu z této dvojice hran, přičemž rozlišujeme, jestli jsou časy přípravy na pořadí operací závislé nebo nezávislé. V případě časů přípravy závislých na pořadí, ohodnotíme každou disjunktivní hranu (x, y) hodnotou s_{ij}, která reprezentuje čas přípravy operace o_j, kterou předchází operace o_i prováděná na stejném stroji. V případě nezávislých časů přiřadíme každé disjunktivní hraně (x, y) hodnotu s_j, která vyjadřuje čas přípravy operace o_j.

- Technologický čas (*technologic time*) je takový časový úsek, o který je proces zpracování úlohy zdržen po skončení zpracování dané operace, než může začít zpracování na dalším stroji. Stroj, na kterém byla daná operace zpracovávána, může být použit pro zpracování další operace jiné úlohy. Pod technologickým časem si můžeme představit například zaschnutí lepidla na výrobku. Technologický čas označujeme symbolem τ_k.
- Dopravní čas (*transport time*) je doba dopravy výrobku po zpracování operace na následující stroj. Dopravní čas označujeme t_k.

Technologický a dopravní čas mají stejný dopad na posun času začátku následující operace, proto jejich hodnoty sčítáme a výslednou hodnotou přiřadíme konjunktivní hraně vedoucí k následující operaci stejné úlohy.

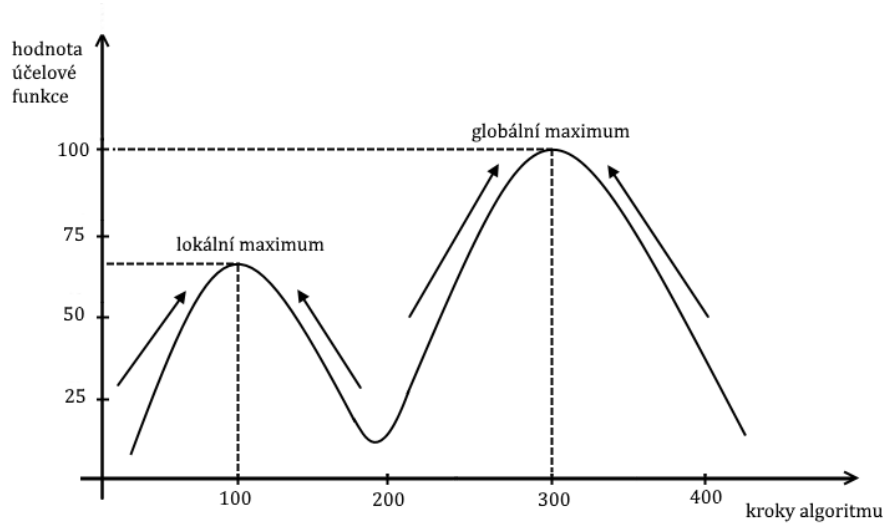
3.4 Metody řešení

Stěžejní částí k vyřešení každého problému rozvrhování výroby je optimalizace účelové funkce popisující sekvence operací na jednotlivých strojích. Většina takovýchto úloh obsahuje velké množství kombinací sekvencí operací a z toho vyplývá, že definiční obor všech možných řešení může být opravdu velký. K řešení úloh rozvrhování výroby můžeme použít exaktní metody, ale od určitého rozsahu definičního oboru problému přestávají být efektivní. Z tohoto důvodu se při řešení těchto problému preferuje používání heuristických metod.

Heuristické algoritmy jsou takové algoritmy, které používají dodatečnou expertní informaci k řešení úloh. Jejich předností je schopnost řešení úloh se značnou velikostí nebo složitou strukturou. K nalezení výsledného řešení používají rychlejší a efektivnější metody než klasické deterministické algoritmy, ovšem na úkor toho, že nezaručují nalezení optimálního řešení. Heuristické algoritmy nedokáží většinou ani určit, jak daleko od globálního optima daný výsledek je. Nicméně, praxe ukázala, že tyto výsledná řešení jsou vzhledem k mnohem menšímu výpočetnímu času naprosto přijatelná a v množině NP těžkých úloh je použití těchto algoritmů jediným způsobem, jak se v relativně krátkém čase dostat k výsledku. Vzhledem k použití

heuristických metod v této práci se jim budeme věnovat podrobněji než deterministickým algoritmům a na dalších řádcích si představíme některé základní heuristické algoritmy (Kokash, 2016).

Lokální hledání (*local search*) je základním heuristickým algoritmem. Tato metoda je klasické prohledávání, které se soustředí jen na určitou část problému. Metoda lokálního hledání může být prováděna několika způsoby, takzvaný horolezecký algoritmus (*hill-climbing*) je jednou z nich. Tato technika funguje tak, že prohledává všechny sousední řešení aktuálního řešení a pokud je některé ze sousedních řešení lepší, stane se novým aktuálním řešením. Problémem této metody je fakt, že může snadno uváznout v lokálním maximu (obrázek č. 14). Jediným řešením, jak z tohoto stavu dostat, je poté spustit toho hledání znovu s jiným výchozím bodem.



Obr. 14 Lokální a globální maximum

Simulované žíhání (*Simulated annealing*) je algoritmus, který používá metody podobné horolezeckému algoritmu, s tou výjimkou, že příležitostně přijme řešení horší než je současné. Pravděpodobnost této akceptace klesá v průběhu času. Základy metody simulovaného žíhání byly poprvé publikovány již v roce 1953 a vedly k simulaci chladnutí materiálu v horké lázni. V roce 1983 pánové Kirkpatrick, Gelatt a Vecchi objevili, že existuje určitá podobnost tohoto přírodního procesu s procesem řešení optimalizačních problémů a popsal jej ve své publikaci.

Metoda simulovaného žíhání je inspirována fyzikálními procesy vznikajícími při odstraňování anomálií (defektů) krystalické mřížky. Opracovávaný krystal se zahřeje na vysokou teplotu a následně se pomalu ochlazuje (*žíhá*). Při vysoké teplotě mají defekty krystalické mřížky vysokou pravděpodobnost zániku a při následném pomalém ochlazování naopak klesá pravděpodobnost vzniku defektů nových. Tímto procesem se snažíme dostat soustavu do stavu, kdy krystalická mřížka neobsahuje žádné defekty.

Jak bylo zmíněno v úvodu, v průběhu výpočtu je možné zvolit postup směřující k přijmutí horšího řešení. Pravděpodobnost tohoto postupu je dána funkcí reprezentující zákon termodynamiky, ve které je při teplotě t pravděpodobnost růstu energie o δE popsána vztahem:

$$P(\delta E) = e^{\left(\frac{-\delta E}{kt}\right)} \tag{12}$$

Kde E je energie v joulech, t je teplota v kelvinech a k je Boltzmanova konstanta. Podle Metropolisova vzorce (1963) se při změně stavu systému spočítá příslušná změna energie. Pokud došlo ke snížení energie, systém přejde do nového stavu, jestliže ale dojde ke zvýšení energie, změna stavu systému je závislá na pravděpodobnosti dané rovnicí (12).

Zakázané prohledávání (*tabu search*) rozšiřuje myšlenku lokálního prohledávání pomocí užití paměťových struktur. Tato metoda přenáší základy z horolezeckého algoritmu, kde se pro aktuální řešení generuje množina všech sousedních řešení. V průběhu algoritmu je nejlepší řešení zaznamenáno a po skončení běhu algoritmu je poslední nejlepší řešení bráno za optimum. Nedostatkem tohoto řešení je, že algoritmus může uváznout v lokálním řešení. Tento problém odstranil Glover (1989) pomocí tzv. *krátkodobé* a *dlouhodobé paměti*.

V krátkodobé paměti (*tabu list*) se nachází informace o řešeních použitých v předcházejících iteracích algoritmu. Při hledání sousedních řešení se algoritmus podívá do této paměti, a pokud tam toto sousední řešení již existuje, nezahrne ho do okolí aktuálního řešení. Po naplnění krátkodobé paměti se při každé iteraci jedno zakázané řešení odebere. U zakázaného prohledávání je důležitá správně zvolená délka krátkodobé paměti. Pokud bychom zvolili příliš velkou velikost tabu listu, vznikne velká pravděpodobnost vynechání některého lokálního extrému, který by mohl být zároveň i globálním extrémem. Naopak při příliš malé velikosti tabu listu může dojít k zacyklení algoritmu.

V dlouhodobé paměti se uchovávají informace o řešeních, které sice nejsou v tabu listu, ale často se vyskytovali jako sousední řešení v předcházejících krocích algoritmu. Hodnoty těchto řešení jsou poté penalizovány v závislosti na četnosti jejich výskytu.

Použití metody *tabu search* je již z principu nejvýhodnější ve spojení s horolezeckým algoritmem. Je možné ji ovšem použít i v kombinaci s jinými algoritmy, kdy ale není tak efektivní, protože další metody neprohledávají celou množinu sousedních řešení aktuálního řešení a pravděpodobnost výběru sousedního řešení nacházejícího se v zakázaných řešení je tedy dost malá.

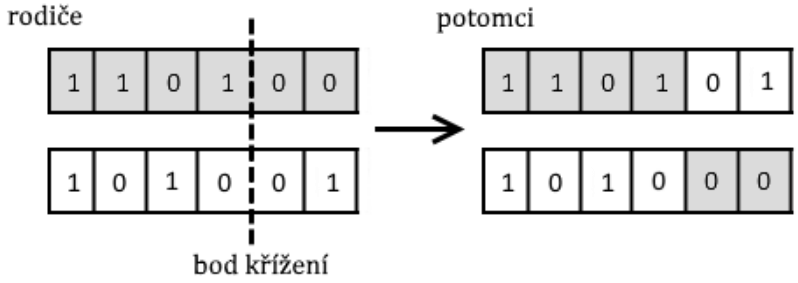
Další metody heuristických výpočtů jsou větší tematicky rozdělené skupiny algoritmů, blíže se na ně podíváme v následujících podkapitolách.

3.4.1 Evoluční algoritmy

Evoluční algoritmy jsou metodami, které využívají nápady biologické evoluce, jako jsou reprodukce, mutace a křížení pro nalezení optimálního řešení optimalizačního

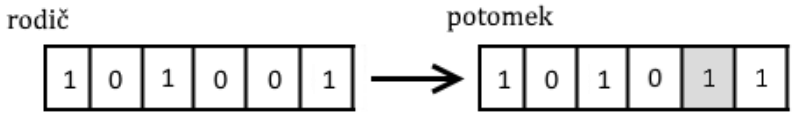
problému. Aplikují principy přežití na množinu potenciálních řešení, aby se pomocí těchto principů postupně přiblížili k optimu. Tato aproximace je vytvořena pomocí výběru (*selekce*) jedinců na základě jejich objektivního ohodnocení, které se v rámci evolučních algoritmů nazývá zdatnost (*fitness*), a pomocí kombinace (*křížení*) vybraných jedinců – rodičů, použitím genetických operátorů. Tento proces vede k evoluci populace jedinců, kteří jsou pro dané prostředí vhodnější než jejich předchůdci.

Existuje více evolučních technik, které se liší v detailech implementace v závislosti na problému, na které jsou aplikovány. Nejúspěšnějším z evolučních algoritmů jsou genetické algoritmy (*Genetic Algorithms – GAs*). Genetické algoritmy byly vyzkoumány v roce 1975 panem Johnem Hollandem, který dokázal demonstrovat jejich účinnost. Genetické algoritmy jsou založeny na faktu, že mutace zlepšuje zdatnost jedinců pouze zřídka, a proto spoléhá hlavně na operaci křížení. Hledání řešení se nejčastěji provádí formou řetězce binárních čísel.



Obr. 15 Operátor křížení

Křížení (obr. 15) je operace, kdy ze dvou rodičů vzniknou dva potomci. Každý potomek získá část genetické informace z jednoho rodiče a zbytek z druhého. Rodiče zahynou a potomci zůstávají pro pokračování evolučního procesu. Tento stav je nazýván jednou generací genetického algoritmu. Stejně jako v biologii, zdatnost jednotlivých jedinců je rozdílná. Jedinci s nízkou zdatností mají vysokou pravděpodobnost úmrtí v jejich generaci a v tom případě jsou odstraněni z generace algoritmu. Jedinci s vysokou zdatností v generaci zůstávají a přechází do výběru ke křížení s ostatními vysoce zdatnými jedinci. Jedinci s vyšší hodnotou zdatnosti mají poté větší šanci na reprodukci. Samotná selekce může probíhat více způsoby. Tradičním je takzvaný ruletový způsob výběru (*roulette wheel*), ve kterém jsou hodnoty zdatnosti jednotlivých jedinců v poměru k celkové hodnotě zdatnosti populace. Dalším způsobem může být například soutěžení (*tournament*), ve kterém soutěží část populace a její vítěz se pak stane rodičem.



Obr. 16 Operátor mutace – bitová inverze

Mutace (obr. 16) je v biologii vzácná a na efekt ne příliš významná změna. Taktéž při implementaci genetického algoritmu se operátor mutace vyskytuje zřídka – pravděpodobnost mutace se nastavuje okolo 2 %. Vhodná míra pravděpodobnosti mutace záleží na daném typu problému, velikosti populace a dalších faktorech. Navzdory malé pravděpodobnosti, je mutace důležitá, protože poskytuje možnost vzniku nové informace v populaci (Simon, 2013).

Základem evolučních algoritmů jsou následující kroky:

1. Inicializujeme a vyhodnotíme počáteční populace.
2. Provedeme selekci vhodných jedinců.
3. Použijeme genetické operátory k vytvoření nové populace.
4. Vyhodnotíme populace.
5. Dokud nejsou splněny ukončující podmínky, opakujeme od bodu 2.

3.4.2 Hejnové algoritmy

V posledních dvou dekádách přitáhly hejnové algoritmy, které jsou považovány za odvětví umělé inteligence, velkou pozornost výzkumníků a byla úspěšně aplikována na různé problémy. Na hejno se můžeme dívat jako na skupinu agentů, sdílející určitý vzor chování za účelem dosažení nějakého cíle (Chee, Jain, 2009). Existuje spousta různých modelů hejnových algoritmů, z nichž nejběžnějšími jsou optimalizace mravenčí kolonií (*ACO – Ant Colony Optimization*), optimalizace hejnem částic (*PSO – Particle Swarm Optimization*) nebo optimalizace včelím rojem (*BCO – Bee Colony Optimization*).

Obecně platí, že hejnové algoritmy se zabývají modelováním kolektivního chování jednotlivých agentů, komunikujících s okolním prostředím a mezi sebou navzájem, které vede ke vzniku soudržného funkčního globálního vzoru. Tyto modely jsou inspirovány sociálním chováním hmyzu a ostatních zvířat. Z technického hlediska se jedná o výpočetní algoritmy, které jsou užitečné pro řešení optimalizačních problémů. Základní princip staví na pravděpodobnostních vyhledávacích algoritmech. Každý jedinec hejna představuje jednoho agenta. Komunikace mezi agenty je stručná a neexistuje žádný centrální ovládací mechanismus, který by tyto agenty řídil. Tyto vlastnosti dělají hejnové algoritmy jednoduché na realizaci a zároveň rozšiřitelné na vysoký stupeň robustnosti systému.

Podle Millonase (1994) existuje pět základních principů hejnových algoritmů:

1. Blízkost (*proximity*) – schopnost provádět jednoduché výpočty času a prostoru v reakci na podněty z okolního prostředí.

2. Kvalita (*quality*) – schopnost rozeznat faktory kvality, např. kvalitu potravy.
3. Rozmanitost reakce (*diverse response*) – schopnost přerozdělovat zdroje a ochránit se před změny okolního prostředí.
4. Stabilita (*stability*) – schopnost udržet stálé chování skupiny i přes změny v okolním prostředí.
5. Přizpůsobivost (*adaptibility*) – schopnost změnit chování skupiny aby se docílilo lepšího přizpůsobení okolnímu prostředí.

Jako ve všech systémech, i zde je nutné dosáhnout rovnováhy mezi principy stability a přizpůsobivosti.

Na dalších řádkách si stručně popíšeme tři nejběžnější výše uvedené modely – mravenčí kolonii, hejno částic a včelí roj.

Optimalizace mravenčí kolonií (ACO) je inspirována sociálním chováním a volbou trasy mravenců při hledání potravy. Mravenci jsou schopni stanovit nejkratší cestu z jejich kolonie ke zdroji potravy a zpátky. Při hledání potravy, mravenci nejdřív náhodně putují po okolí jejich kolonie. Jakmile najdou zdroj potravy, vrátí se do kolonie a po cestě za sebou zanechávají vyznačenou cestu z chemických látek zvaných feromony. Ostatní mravenci objeví označenou cestu a vydají se po ní. Feromony jsou tedy jakýmsi způsobem komunikace mezi mravenci. Tím, že se po cestě vydá více mravenců, se zvýší množství feromonů na cestě a čím silnější toto označení je, tím je pravděpodobnější, že jej najde a bude sledovat více mravenců. Nicméně, feromony v průběhu času vyprchají, a pokud nejsou obnovovány, vyznačení postupně slábne. Kratší cesty jsou tedy obnovovány rychleji než dlouhé, neboť je na nich frekvence pohybu mravenců vyšší. Tímto způsobem se mravenci dopracují k tomu, že zůstane vyznačená pouze nejkratší cesta ke zdroji potravy.

Z technického hlediska je díky vyprchání feromonů zabránění uvíznutí na lokálním maximu.

Optimalizace hejnem částic (PSO) je inspirována sociálním chováním hejna ptáků a ryb. Při optimalizace hejnem částic se vytvoří populace částic, které se pohybují skrz více dimenzionální prohledávací prostor s danou rychlostí. Každá častíc zachycuje jeden průnik mezi všemi dimenzemi. V každé generaci, rychlost částice je nahodile upraveny podle historicky nejlepší hodnoty sebe sama a jejích sousedů. Pohyb každé částice se tedy postupně vyvíjí k optimálnímu nebo téměř optimálnímu řešení.

Optimalizace hejnem částic má několik výhod. Vyhledávací mechanismus je robustní a efektivní v udržování rozmanitosti. Hlavní výhodou je, že má vysokou pravděpodobnost dosažení optimálního řešení.

Optimalizace včelím rojem (BCO) je metoda inspirována chováním včel při hledání potravy. Základní myšlenkou této metody je vytvoření více agentního systému (roj umělých včel) schopných vyřešit obtížné kombinatorické optimalizační problémy.

Chování včel při hledání a sběru nektaru lze popsat následujícím způsobem. Včely průzkumnice se vydávají z úlu a náhodně hledají zdroj potravy. Po nalezení zdroje a nabrání nektaru se vracejí do úlu, kde v místě zvaném „taneční parket“

(*dance floor*) provádějí „vrtící tanec“ (*waggle dance*). Tímto tancem znázorňují, kolik sousedních zdrojů se v jimi nalezeném místě nachází. Jakmile se ostatní včely rozhodnou opustit úl, vyberou si jednu z včel tanečnic a následují ji. Po nasbírání dostatku potravy se s ní vrací do úlu a odevzdá ji k uskladnění. Po odevzdání má včela čtyři možnosti, jak se může zachovat:

- Pokud je zdroj již vyčerpaný, opustí ho a vybírá si další včelu tanečnici, kterou bude následovat.
- Stává se průzkumnicí.
- Pokračuje ve sbírání na daném zdroji potravy.
- Stane se tanečnicí a rekrutuje nové včely ke sbírání na daném zdroji potravy.

Včely si mezi těmito čtyřmi možnostmi vybírají podle toho, za jak moc kvalitní považují daný zdroj potravy, to znamená, kolik zbývá potravy na daném zdroji a kolik existuje sousedních zdrojů potravy.

V průběhu času bylo objeveno několik implementací této metody. Jednou z těchto metod je algoritmus umělého včelího roje (*Artificial Bee Colony – ABC*) navržený p. Karabogou (2005). Tato metoda spočívá ve vytvoření umělého roje složeného z následujících typů agentů:

- včely sběračky (*employed bees*) – létají k zdroji potravy;
- včely divačky (*onlookers*) – sledují tancující včely a rozhodují se mezi zdroji potravy;
- včely průzkumnice (*scouts*) – náhodně objevují zdroje potravy;

V ABC algoritmu, půlka roje je složena ze včel sběraček a druhá půlka z včel divaček. Algoritmus poté provádí hledání v cyklech. Každý cyklus se skládá z těchto kroků:

1. Včely sběračky letí ke zdroji potravy, sbírají nektar a vrací se do úlu. V úlu se změří množství jejich nasbíraného nektaru.
2. Informace o nasbíraných množstvích nektaru je předána včelám divačkám. Ty se rozhodnou pro nejlepší zdroj potravy.
3. Včely, které se rozhodly, že se stanou průzkumnicemi, se vydají hledat nový zdroj potravy.

Při optimalizaci pomocí umělého včelího roje je původní řešení náhodně vygenerováno a při následujících průchodech algoritmu se včely sběračky a průzkumnice starají o modifikace stávajícího řešení.

4 Popis problému a možná řešení

4.1 Analýza prostředí

Než začneme s vlastním návrhem řešení problému, je nutné nejdříve analyzovat stávající situaci ve firmě.

Cílem této práce je vytvořit webovou aplikaci pro optimalizaci plánování SMT výroby. *SMT* (*surface mount technology*), v překladu povrchová montáž, je proces výroby, kdy se elektronické součástky osazují pájením přímo na povrch plošného spoje. Plošný spoj může být vícevrstvý a osazování probíhá vždy na vnějších stranách. Součástky určené pro povrchovou montáž nazýváme *SMD* (*surface mount device*). Povrchová montáž vděčí rozšíření hlavně díky miniaturizaci elektroniky v průběhu 80. let a v současnosti je používána v téměř každém sériově vyráběném přístroji.

Zadavatelem požadavku na webovou aplikaci je výrobní firma, ve které se v současné době nachází devět výrobních linek sestávajících se z tří různých typů – *SMT*, *Axial* a *Radial* linka. Na těchto linkách lze vyrobit přibližně sedm set různých produktů. Výsledné produkty se mohou vyrábět buďto na jedné lince daného typu, nebo postupně na několika linkách v předem daném pořadí. Ve firmě probíhá klasický třísměnný provoz.

Na výrobu každého produktu je potřeba určitého nastavení (*setup*) výrobní linky. Některé produkty mohou mít toto nastavení identické, ale ve většině případů je potřeba při změně vyráběného produktu provést na dané výrobní lince přenastavení (*changeover*), ve kterém se změní nastavení výrobní linky na nadcházející produkt.

V současné době probíhá plánování výroby manuálně. V podniku se udržuje několik tabulkových dokumentů, kde jeden dokument reprezentuje jednu výrobní linku a pracovníci na pozici plánovač výroby je každodenně aktualizují vzhledem k nadcházejícím směnám. Toto plánování výroby je neefektivní a zabírá velkou část pracovní doby příslušným pracovníkům, proto byl zadán požadavek na vytvoření webové aplikace, ulehčující práci nejenom zaměstnancům na pozici plánovačů.

Požadovaná webová aplikace by měla umět zobrazovat informace o aktivních linkách, přehled produktů a interaktivní plán výroby na danou směnu. Plán bude obsahovat informace o produkci na jednotlivých linkách, změnách výroby, plánovaných údržbách a odstávkách. Aplikace tento plán dokáže vhodným způsobem vizualizovat a poskytne nástroje na jeho úpravu. Výrobní plán musí respektovat všechny podmínky pro výrobu a bude informovat plánovače v případě, že z nějakého důvodu není plán realizovatelný. Aplikace musí použít efektivní způsoby generování a optimalizace plánu, aby bylo dosaženo dostatečného ušetření času plánovače.

4.2 Výběr algoritmu pro řešení

Na základě provedené rešerše o plánování zakázkové výroby byla vybrána reprezentace tohoto problému pomocí disjunktivního grafu. Jedním z kritérií při rozhodování byl fakt, že reprezentace pomocí disjunktivního grafu je mnohem vhodnější pro následnou optimalizaci za použití heuristických metod. Druhým kritériem byla úspěšnost algoritmu. Díky experimentům provedeným Majerem (2001) bylo možné provést srovnání úspěšnosti řešení problému zakázkové výroby na datech reprezentovaných disjunktivním grafem (DG) a daty reprezentovanými preferenčním seznamem (PS). Na jednotlivé struktury byly puštěny tři heuristické optimalizační algoritmy. Jednalo se o metody simulovaného žíhání, zakázaného prohledávání a genetický algoritmus. Výsledky srovnání jsou uvedeny v následující tabulce.

Tab. 1 Srovnání reprezentace problému zakázkové výroby

Příklad	Struktura	Nejlepší hodnota	Průměrná hodnota
Fischer a Thompson 6x6; optimum: 55 Čas optimalizace: 60s	PS	68	74
	DG	55	55
Lawrence 10x5; optimum: 597 Čas optimalizace: 120s	PS	844	881
	DG	623	632
Adams, Balas a Zawack 10x10; optimum: 1234 Čas optimalizace: 300s	PS	1856	1938
	DG	1300	1321
Sumarizace	**PS**	**923**	**965**
	DG	**659**	**670**

Zdroj: Majer, 2001

Ze sumarizačního řádku jasně vyplývá, že reprezentace disjunktivním grafem je účinnější než reprezentace pomocí preferenčního seznamu.

Druhým významným rozhodnutím byla volba vhodné optimalizační metody. Na základě informací, získaných v teoretické části práce, byla za vhodný optimalizační algoritmus zvolena moderní metoda vhodná na řešení složitých kombinatorických problému – optimalizace včelím rojem. Toto rozhodnutí bylo podpořeno preferencí ze strany zadavatele a také výsledky různých studií zaměřených na toto téma. Jedním z autorů těchto studií je pan Karaboga, který se tomuto tématu věnuje ve více publikacích (2005, 2007, 2008), ve kterých vyzdvihuje schopnost algoritmu dostat se spolehlivě z lokálního extrému a kde také udává, že se hodí pro složité optimalizační problémy (Karaboga, Basturk, 2007). Dalším podnětem byl článek, ve kterém Sato a Hagiwara (1997) uvádějí, že včelí algoritmus má lepší účinnost než konvenční genetický algoritmus. Zajímavostí je, že pan Sato řadí tento algoritmus spíše do kategorie evolučních algoritmů než do kategorie hejnových algoritmů, kam ho řadí ostatní literatura (Teodorović, 2009).

4.3 Specifikace cílů a metodika

4.3.1 Metodika

Prvním krokem při vytvoření plánovacího algoritmu bude správně pojmenovat a definovat problém nacházející se v procesu výroby. V tomto případě se jedná o problém plánování zakázkové výroby.

Dalším krokem bude volba vhodné reprezentace dat v tomto problému. Při rozhodování o vhodné reprezentaci bude nutné brát v potaz vhodnost metody pro následné použití heuristických metod a efektivnost metody. Poté bude nutné stanovit počáteční rozvrh výroby (tzn. úplnou selekci) a pomocí metody kritické cesty zjistit časy začátků, konců operací, případných rezerv a kritickou cestu (popřípadě cesty) grafu.

Následujícím krokem bude zvolení správné heuristické metody pro daný problém. Po implementaci algoritmu bude nutné správně nastavit počáteční parametry algoritmu, zvolit operace přechodu do sousedního řešení a určit kritéria rozvrhu.

Po úspěšném aplikování algoritmu na disjunktivní graf bude plánovací algoritmus připraven na integraci do vytvořené webové aplikace. Při realizaci aplikace bude nutné dodržovat následující metodický postup.

Počátečním krokem implementace webové aplikace bude navržení datového modelu systému a následná realizace databáze. Navazujícím dalším logickým krokem je navržení objektového modelu a transformace tohoto modelu do základních tříd webové aplikace, sloužících ke komunikaci aplikační vrstvy s databází a ke správě různých druhů oprávnění uživatelů vyskytujících se v systému.

Po vybudování aplikačního základu webové aplikace bude následovat návrh a implementace uživatelského prostředí. V něm bude nutné dbát především na správně navrženou navigaci v aplikaci, přehledné zobrazení všech požadovaných informací a uživatelskou přívětivost při zadávání nových údajů do systému.

Implementace uživatelského prostředí je úzce spojená s rozšiřováním funkcí systému. Stávající třídy proto budou rozšířeny o třídy reprezentující jednotlivé výrobky, výrobní linky, výrobní plán a další. V tomto momentě dojde také k integrování optimalizačního algoritmu do aplikační části programu. Tato integrace bude spočívat v komunikaci systému s algoritmem, za účelem získání vstupních dat do algoritmu a pro zobrazení znalostí získaných optimalizací do prostředí systému.

Poslední částí implementace webové aplikace bude její zpětné testování a vyhodnocení výsledků. Tato zpětná vazba bude důležitá pro možnost případných úprav dodaného řešení.

4.3.2 Specifikace cílů

Globální cíl, vytvoření informačního systému pro plánování výroby, je možné rozdělit do dvou dílčích cílů – vytvoření plánovacího algoritmu a vytvoření prostředí informačního systému. Tyto dva cíle obsahují několik konkrétních úkolů, které je nutné splnit pro úspěšné dokončení projektu. Tímto způsobem rozdělení je možné předejít případným nejasnostem v procesu implementace a získat lepší kontrolu nad průběhem projektu.

Při implementaci informačního systému je nutné dbát na dodržení požadavků od zadavatele. Jednotlivé dílčí cíle lze vyjmenovat následovně:

- Navrhnout a realizovat strukturu databáze.
- Navrhnout a realizovat objektový model informačního systému.
- Navrhnout a realizovat uživatelské prostředí informačního systému.
- Implementovat požadovanou funkcionalitu informačního systému.

Požadovanou funkcionalitu opět můžeme shrnout do následujícího výčtu:

- Implementovat algoritmus včelího roje pro optimalizaci plánování výroby.
- Implementovat různé úrovně oprávnění jednotlivých uživatelů.
- Implementovat pohled na data v systému:
    - Přehled linek.
    - Přehled produktů.
- Implementovat funkce pro plánování rozvrhu výroby:
    - Tvorba výrobního rozvrhu na danou směnu včetně vizualizace.
    - Vizualizace historických rozvrhů.
    - Editace plánu (změna priority výroby).
    - Kontrola realizovatelnosti rozvrhu.
- Zaznamenávání proběhlých událostí pro lepší kontrolu.

5 Navržené řešení

Na základě požadavků od zadavatele byl proveden návrh výsledného řešení – informačního systému pro optimalizaci plánování výroby. Tento informační systém je reprezentován webovou aplikací v jazyce *C#*, pomocí technologie *ASP.NET*. Související databázi byla realizována v prostředí *Microsoft SQL Server 2012.*

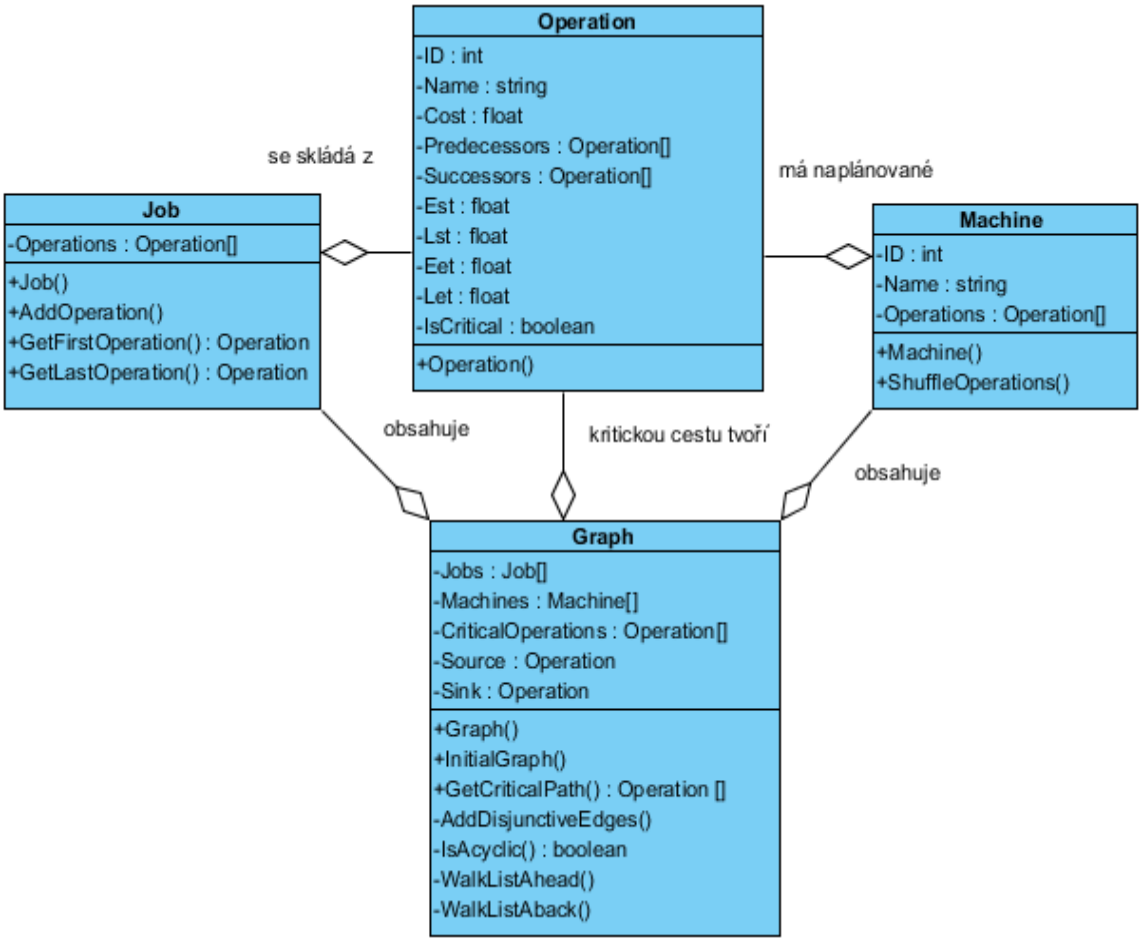
K vývoji informačního systému byl použit profesionální vývojářský programu *Visual Studio* od firmy *Microsoft*, který byl vytvořen přímo pro vývoj *.NET* aplikací. Jednou z velkých předností programu *Visual Studio* je zabudovaný vývojový webový server, na kterém je možné vyvíjené aplikace jednoduše spouštět, testovat a ladit. Kromě toho poskytuje příjemné uživatelské prostředí s mnoha zabudovanými nástroji, užitečnými při vývoji aplikací. Další výhodou tohoto profesionálního programu je, že poskytuje pro studenty bezplatnou licenci. Správa databáze proběhla pomocí programu *SQL Server Management Studio* od firmy *Microsoft.*

5.1 Implementace problému plánování zakázkové výroby

Před začátkem vývoje vlastního informačního systému bylo nutné reprezentovat a vyřešit stěžejní problém plánování zakázkové výroby. Tento problém reprezentujeme pomocí disjunktivního grafu, který můžeme znázornit pomocí čtyř tříd:

- *Machine* – jednoduchá třída, uchovávající informace o jednom stroji. Obsahuje parametry jako ID stroje, název stroje, typ stroje a seznam operací, které jsou na tomto stroji naplánovány. Tato třída obsahuje také metodu *ShuffleOperation*, sloužící k náhodnému prohození operací naplánovaných na tomto stroji.
- *Operation* – třída reprezentující jednotlivé operace. Každá instance této třídy obsahuje základní informace jako je název operace, čas potřebný na výrobu a seznamy předcházejících a následujících operací. Dále obsahuje informace vztahující se ke kritické cestě, to znamená časy začátku výroby a konce výroby a příznak, jestli tato operace leží na kritické cestě.
- *Job* – kontejnerová třída obsahující pouze seznam operací pro danou úlohu. Součástí této třídy jsou metody pro inicializaci úlohy, nebo získání první a poslední operace v dané úloze.
- *Graph* – další především kontejnerová třída symbolizující disjunktivní graf. Obsahuje seznamy strojů, operací a úloh, vyskytujících se v řešeném grafu. Tato třída obsahuje mimo jiné metody na inicializaci grafu a zjištění kritické cesty.

Vztahy mezi jednotlivými třídami nalezneme na následujícím obrázku.



Obr. 17 Objektová reprezentace disjunktivního grafu

Disjunktivní graf byl implementován následujícím způsobem. Získáme a inicializujeme požadované operace a zjistíme, jaké jsou k dispozici stroje. Operace seskupíme do jednotlivých úloh a provedeme počáteční inicializaci grafu. Tato inicializaci zahrnuje přidání požadovaných úloh a následně disjunktivních hran mezi jednotlivé operace.

Při přidávání disjunktivních hran je nutné dbát na to, že výsledný disjunktivní graf musí být acyklický. Pro zjištění, jestli graf obsahuje cyklus, je použita metoda *topologického řazení* (Kahn, 1962). Takzvaný Kahnův algoritmus je možné popsat následovně:

```
L ← prázdný seznam, který bude obsahovat seřazené uzly
S ← množina uzlů bez vstupních hran
dokud (S <> ∅) proveď
  odstraň uzel n z množiny S
  přidej uzel n do seznamu L
  pro každý uzel m s hranou e z n do m proveď
    odstraň hranu e z grafu
    jestliže uzel m nemá žádné další vstupní hrany potom
      vlož uzel m do množiny S
jestliže graf nemá žádné hrany potom
  graf obsahuje alespoň jeden uzel
jinak
  vrať topologicky seřazený seznam uzlů L
```

Přidáním disjunktivních hran vznikne náhodný přípustný rozvrh, kterému je možné přiřadit hodnotu účelové funkce.

Dalším krokem implementace je pomocí metody kritické cesty zjistit hodnotu účelové funkce *(makespan)* a zjistit nejdříve a nejpozději možné časy počátku a dokončení zpracování každé operace. Metodu kritické cesty *(CPM)* tvoří dva algoritmy, které postupně prochází sousedy jednotlivých uzlů:

- Průchod následovníky uzly. V tomto algoritmu začínáme v počátečním uzlu grafu a postupně u všech jeho následovníků nastavujeme nejdříve možné časy začátku *(Est)* a nejdříve možné časy konce *(Eet)* zpracování uzlu. Algoritmus poté rekurzivně zpracuje všechny uzly nacházející se v grafu.
- Průchod předchůdci uzlu. Po skončení nastavení nejdříve možných časů je nutné nastavit nejpozději možné časy začátku *(Lst)* a konce *(Let)* zpracování uzlu. Tentokrát algoritmus začíná od konečného uzlu a pokračuje rekurzivně přes všechny předchůdce jednotlivých uzlů. Ukázka kódu použitého v práci je zobrazena na obrázku č. 18 na následující stránce.

Po provedení těchto dvou algoritmů je možné jednoduše získat uzly ležící na kritické cestě. Jsou to ty uzly, u nichž platí, že mají nulovou velikost rezervy. Tuto podmínku lze slovy vyjádřit tak, že rozdíl mezi nejdříve možným čas dokončení a nejpozději možným čas dokončení zpracování uzlu je nulový a také je zároveň nulový rozdíl mezi nejdříve možným časem začátku a nejpozději možným časem začátku zpracování uzlu.

```
public static void WalkListAback(Operation activity)
{
    bool processAgain = true;
    // maximální počet průchodů = počet předchůdců uzlu
    int maxLoop = activity.Pred.Count;
    int loopCounter = 0;
    while (processAgain)
    {
        foreach (var predecessor in activity.Pred)
        {
            // pokud nebyl předchůdce ještě nastaven a zároveň už neexistuje
            // žádný následovník, který by nebyl nastaven
            if (predecessor.Let == -1 &&
                predecessor.Succ.Where(op => op.Lst == -1).ToList().Count == 0)
            {
                foreach (var successor in predecessor.Succ)
                {
                    if (predecessor.Let == -1)
                        predecessor.Let = successor.Lst;
                    else if (predecessor.Let > successor.Lst)
                        predecessor.Let = successor.Lst;
                }
                predecessor.Lst = predecessor.Let - predecessor.Cost;
            }
        }
        loopCounter++;

        // všichni předchůdci mají nastavený Lst => nezpracovávat znovu
        processAgain = !(activity.Pred.Where(op => op.Lst == -1).ToList().Count == 0);

        if (loopCounter == maxLoop)
        {
            processAgain = false;
        }
    }
    foreach (var predecessor in activity.Pred)
    {
        WalkListAback(predecessor);
    }
}
```

Obr. 18 Algoritmus zpětného průchodu disjunktivním grafem

Nalezením kritické cesty je zároveň zjištěna hodnota účelové funkce, která reprezentuje celkový čas dokončení všech úloh. Tento čas je v následujícím kroku implementace minimalizován pomocí algoritmu včelího roje.

5.2 Implementace optimalizačního algoritmu

Pro implementaci optimalizačního algoritmu včelího roje byla zvolena struktura, kterou tvoří dvě třídy. Třída *Hive* (roj) a třída *Bee* (včela).

5.2.1 Třída *Bee*

Třída Bee je vcelku jednoduchá a obsahuje pouze čtyři atributy:

- *Status* – tento atribut obsahuje celočíselnou hodnotu, která označuje typ včely. Může nabývat hodnot 0 až 2, kde 0 značí neaktivní včelu, 1 aktivní a 2 včelu průzkumnici.
- *MeasureOfQuality* – obsahuje desetinné číslo, které reprezentuje hodnotu účelové funkce. Tato účelová funkce se vztahuje pro řešení, které má v paměti jedna instance této třídy (včela). V našem případě se jedná o celkovou dobu trvání výroby. Tento parametr je měřítkem, které řešení (výrobní rozvrh), je lepší než jiný, zde platí, že čím nižší hodnota, tím lepší.
- *NumberOfVisits* – tento celočíselný atribut znázorňuje počet návštěv daného zdroje potravy (hledaného řešení) včelou, aniž by našla lepší sousední řešení. Tento parametr je použit k simulování létání včely ke zdroji potravy, dokud se zdroj nevyčerpá. Pro aktivní včely platí, že pokud hodnota přesáhne prahovou hodnotu, řešení se zahodí a včela se stane neaktivní.
- *MemoryMatrix* – atribut, který obsahuje seznam všech operací, tvořících disjunktivní graf vztahující se na jedno řešení.

Konstruktor třídy Bee přijímá všechny čtyři atributy jako jeho parametry. Kromě tohoto konstruktoru obsahuje tato třída pouze metodu *ToString*, která slouží k výpisu hodnot atributů a je užitečná při ladění běhu algoritmu.

5.2.2 Třída *Hive*

Třída *Hive* je oproti *Bee* významně obsáhlejší a obsahuje větší počet atributů, které je podrobně popsán v následujícím výčtu:

- *TotalNumberBees* – celkový počet včel v roji. Větší počet bude dosahovat lepších výsledků, ale zpomaluje chod algoritmu. Celkový počet včel se dělí na tři typy s doporučeným procentuálním poměrem 75:10:15, kde jednotlivé složky tvoří aktivní, neaktivní včely a včely průzkumnice.
- *MaxNumberVisits* – maximální možný počet návštěv zdroje potravy. V každé iteraci algoritmu je kontrolováno, zda včela našla lepší sousední řešení, v opačném případě je zvýšeno počítadlo *NumberOfVisits*. Literatura (Karaboga, 2007) doporučuje použít takovou hodnotu, která se rovná pětinásobku počtu sousedních řešení aktuálního řešení.
- *MaxNumberCycles* – je proměnná používaná ke kontrole počtu iterací algoritmu. Jedná se o nezbytnou proměnnou, protože algoritmus není schopen poznat, zda dosáhl optimálního řešení.
- *ProbPersuasion* – je pravděpodobnostní hodnota, používaná k určení, zda neaktivní včela, která sleduje vábení aktivní včely, se nechá přesvědčit a použije toto řešení. Obvykle se používá hodnota 0.9, která znamená, že existuje 90% šance, že včela toto řešení přijme.

- *ProbMistake* – je další pravděpodobnostní hodnota, která určuje, jestli včela neudělá chybu. Tato chyba může spočívat v tom, že včela odmítne prokazatelně lepší řešení, nebo naopak přijme horší řešení než je její aktuální řešení. U tohoto atributu používáme hodnotu 0.01, která reprezentuje 1% šanci na chybu.
- *Bees* – je atribut, obsahující pole všech instancí třídy *Bee*. Počet je odvozen od hodnoty *TotalNumberBees*.
- *BestMemoryMatrix* – obsahuje seznam operací, které představují aktuálně nejlepší řešení problému.
- *BestMeasureOfQuality* – je hodnota účelové funkce aktuálně nejlepšího řešení uloženého v atributu *BestMemoryMatrix*.
- *Data* – atribut, obsahující odkaz na objekt třídy *Graph*, reprezentující disjunktivní graf řešeného problému.
- *Random* – je atribut typu *Random*, který slouží ke generování pseudonáhodných čísel pro rozhodovávání u atributů *ProbPersuasion* a *ProbMistake*.

Třída dále obsahuje konstruktor a několik důležitých tříd, které budou nyní detailně probrány. Konstruktor slouží k inicializaci a počátečnímu nastavení všech důležitých proměnných třídy *Hive*. Mezi nejdůležitější atributy patří *BestMemoryMatrix*, kterému je právě v konstruktoru přiřazena počáteční hodnota rovnající se náhodně vygenerovanému pořadí operací na strojích, tvořícímu přípustný rozvrh. Následně zjistíme hodnotu účelové funkce tohoto řešení a uložíme jej do globálního proměnné *BestMeasureOfQuality* obsahující prozatímní nejlepší hodnotu účelové funkce.

V další části konstruktoru je iterováno skrz všechny instance třídy *Bee* a dochází k inicializaci. Napřed se nastaví typ včely tak, aby byl splněn procentuální poměr definovaný výše, a poté je pro každou včelu vygenerováno náhodné lokální řešení. Následně je zjištěna hodnota účelové funkce a poměřena s globální hodnotou uloženou v proměnné *BestMeasureOfQuality*. Pokud je toto lokální řešení lepší, je nastaveno jako nové globální řešení.

5.2.3 Metoda *Solve*

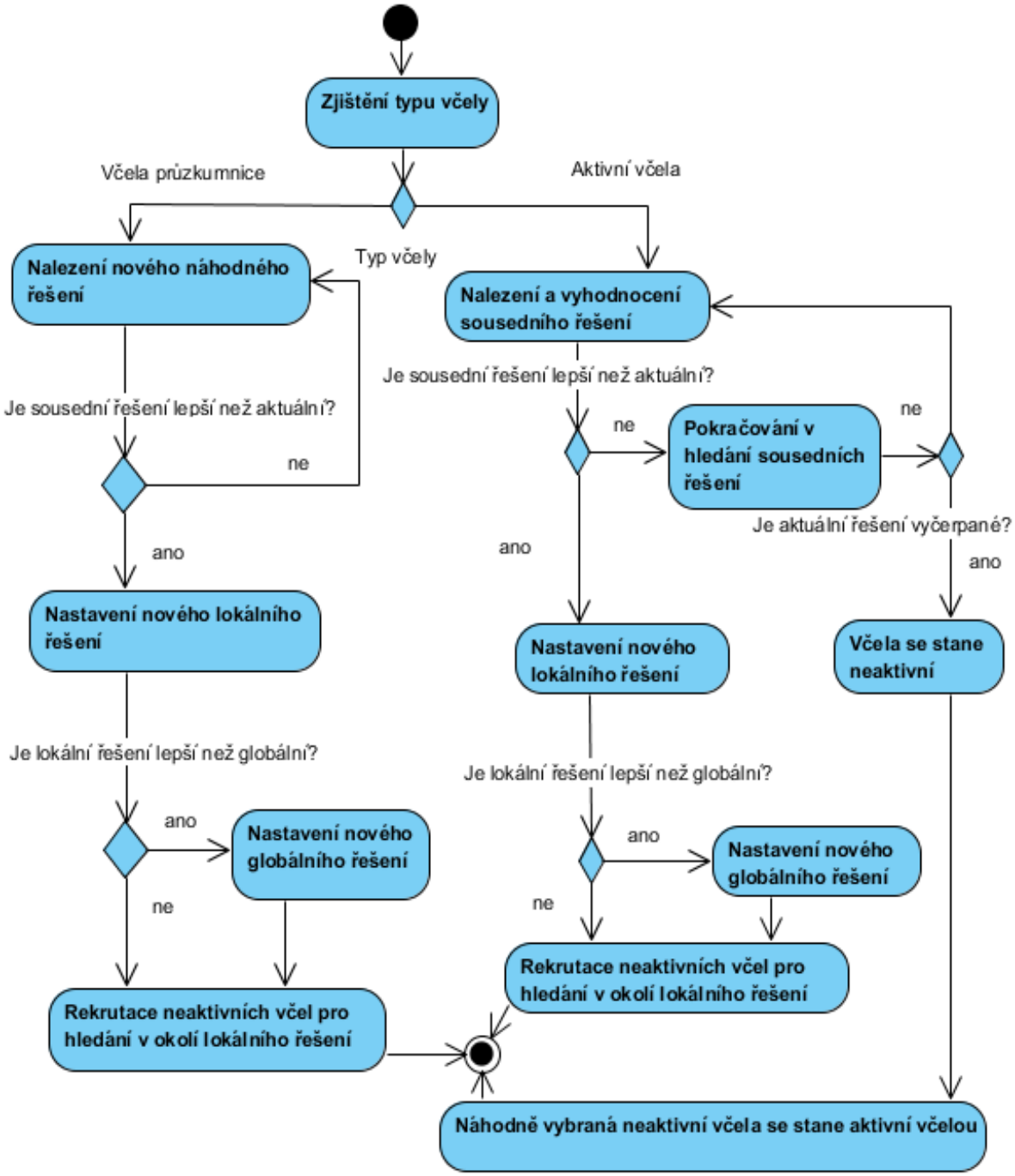
Další stěžejní částí je metoda *Solve*, která obsahuje všechnu logiku simulující chování včel ve snaze vyřešit daný problém. Hlavní část je uvnitř cyklu, který kontroluje, aby nedošlo k více iteracím, než je povoleno atributem *MaxNumberCycles*. Uvnitř tohoto cyklu následuje další cyklus skrze všechny instance třídy *Bee*, u kterých v závislosti na jejich typu vyvoláváme příslušné metody. U aktivních včel je to metoda *ProcessActiveBee*, která slouží především k prohledávání sousedních řešení.

Uvnitř této metody je nejdříve vygenerováno sousední řešení pomocí metody *GenerateNeighborMemoryMatrix* a následně ohodnoceno vzniklé sousední řešení. Poté porovnáváme hodnotu sousedního řešení s řešením, které je aktuálně v paměti této včely. Pokud je nové řešení lepší vygenerujeme si pseudonáhodné číslo v intervalu 0 až 1 a porovnáme ho s prahovou hodnotou v atributu *probMistake*, abychom zjistili, zda včela náhodou neudělala chybu při rozhodování o přijmutí nového řešení. Pokud ano, nové řešení zahodíme. Pokud ne, aktualizujeme stávající řešení

v paměti této včely a pokračujeme v algoritmu. V případě, že nové řešení není lepší, opět zjistíme, zda nedošlo k chybě v rozhodování a nové řešení buďto přijmeme nebo zahodíme. Dále rozhodujeme, jestli dané řešení již není vyčerpané, tzn., jestli jsme nepřekročili *MaxNumberVisits* u aktuálního řešení v paměti včely. Pokud došlo k vyčerpání, aktuální včela změní stav z aktivní na neaktivní a náhodně se vybere některá z neaktivních včel a stane se naopak aktivní. Jestliže k vyčerpání nedošlo, zjistíme, jestli je lokální řešení lepší než stávající globální řešení a zavoláme metodu *DoWaggleDance* – proveď vábící tanec. Tato metoda spočívá k přesvědčení neaktivních včel, aby se vydaly prozkoumávat lokální řešení tancující včely. V principu tato metoda prochází přes všechny neaktivní včely a porovnává lokální řešení tancující a neaktivní včely. Pokud má tancující včela lepší řešení, opět vygenerujeme pseudonáhodné číslo a porovnáváme ho s prahovou hodnotou *probPersuasion*. V případě úspěchu se do paměti přesvědčené včely uloží řešení tancující včely.

Důležitou metodou, která ještě nebyla objasněna, je metoda sloužící ke generování sousedních řešení *GenerateNeighborMemoryMatrix*. Klíčovým konceptem hejnových algoritmů je fakt, že každé řešení má dané okolí definované množinou sousedních řešení. Existuje několik způsobů, jak tvořit sousedství v rozvrzích založených na reprezentaci disjunktivním grafem. V tomto algoritmu pracujeme se sousedstvím, které vytvoříme tím, že zaměníme pořadí dvou bezprostředně následujících operací na jednom stroji v případě, že obě operace leží na kritické cestě.

Kromě metody *ProcessActiveBee*, zpracovávající aktivní včely, se v metodě *Solve* setkáme s metodou *ProcessScoutBee*, která simuluje chování včel průzkumnic ve snaze náhodně nalézt nový zdroj potravy. Tato metoda tedy spočívá ve vygenerování nového náhodného řešení, jeho ohodnocení a následného porovnání se stávajícím lokálním a globálním řešením. Pokud je nalezené řešení lepší, včela provede vábící tanec.



Obr. 19 Stavový diagram jedné iterace metody *Solve*

Pro přehlednost je na výše uvedeném diagramu znázorněna jedna iterace metody *Solve*, u které se mění průběh v závislosti na typu aktuálně zpracovávané včely.

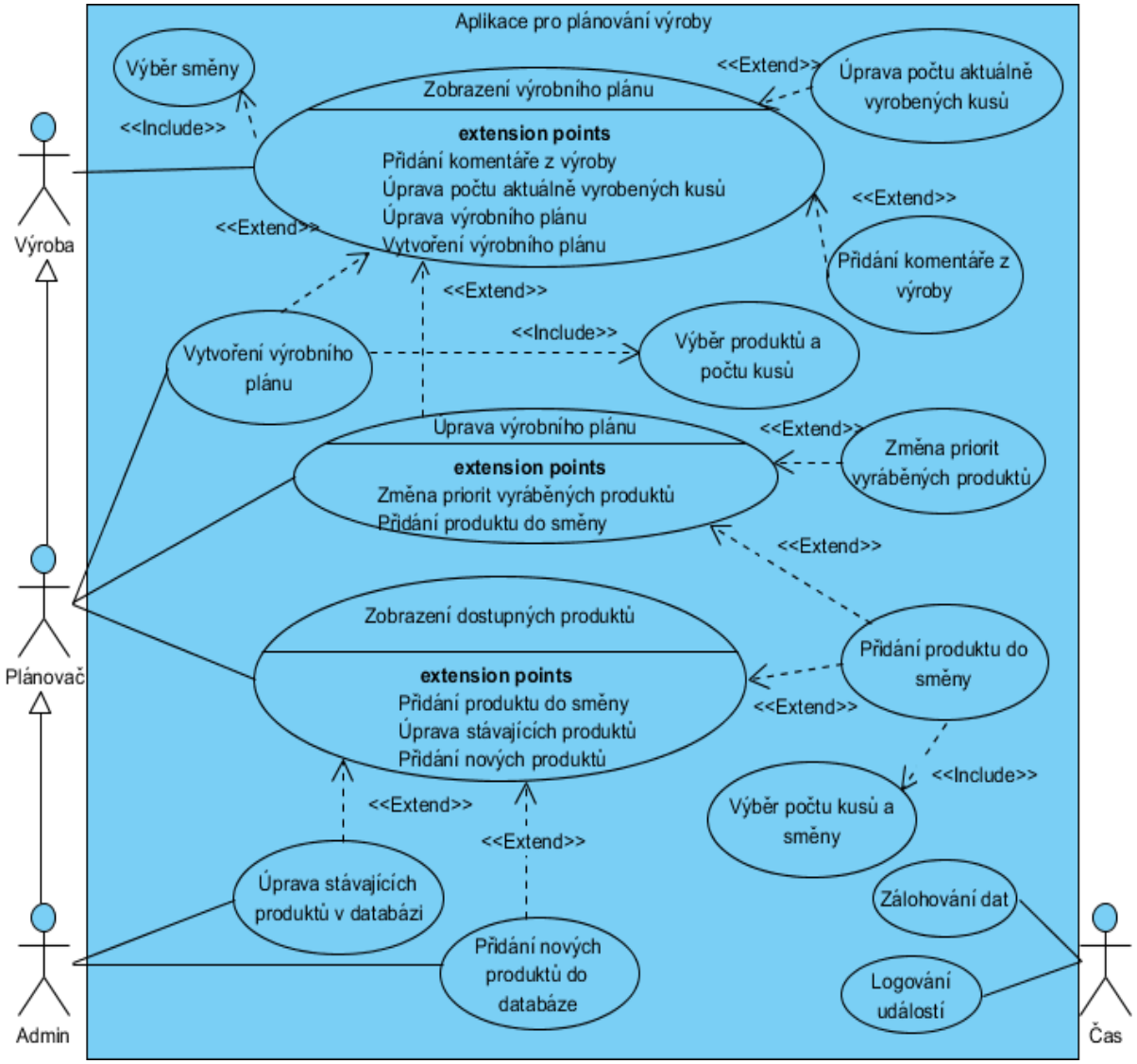
Metoda *Solve* skončí v momentě, kdy počet iterací dosáhne hodnoty *MaxNumberCycles*. V tomto momentě v atributu *BestMemoryMatrix* nalezneme aktuálně nejlepší možné řešení.

5.3 Aplikace pro plánování výroby

5.3.1 Návrh aplikace

Na základě požadavků od zadavatele je potřeba navrhnout a implementovat aplikaci, která zjednoduší zaměstnancům na pozici plánovač výroby jejich každodenní práci a zefektivní proces plánování výroby.

Pro reprezentaci všech funkčních požadavků je použit diagram případů použití (obrázek č. 20), který zároveň znázorňuje i jednotlivé typy uživatelů, kteří pracují s aplikací, a definuje jejich pravomoci.



Obr. 20 Diagram případů použití

V aplikaci se budou nacházet tři typy uživatelských účtů, odvozených od pracovních pozic zaměstnanců, kteří s aplikací budou pracovat:

- Výroba (*production*) – uživatelé s nejnižší úrovní oprávnění. Jedná se především o montážní dělníky, kteří s aplikací budou pracovat v průběhu jejich směny a budou ji používat ke zjištění, které výrobky se mají na jaké lince vyrábět (případ použití *Zobrazení výrobního plánu)* a k aktualizaci informací o dosud vyrobených kusech, nebo přidání komentáře k vyráběnému produktu.
- Plánovač (*planner*) – uživatelský účet pro zaměstnance plánující výrobu. Tito zaměstnanci budou nejčastějšími uživateli aplikace. Jejich úkolem je naplánovat výrobu na následující směny (případ použití *Vytvoření výrobního plánu*), popřípadě upravit nadcházející výrobní plány změnou priorit výroby jednotlivých produktů. Tito uživatelé mají, na rozdíl od dělníků, přístup k zobrazení všech produktů v aplikaci (případ použití *Zobrazení produktů*), odkud mohou přidat další výrobky do dosud neprovedených výrobních plánů.
- Admin (*admin*) – správcovský uživatelský účet, kterému náleží všechna uživatelská práva v aplikaci. Kromě funkcí nižších uživatelských účtů může administrátor aplikace navíc upravovat stávající a přidávat nové výrobky do databáze výrobků.

V aplikaci platí hierarchie uživatelských účtů, to znamená, že plánovači dědí kromě svých oprávnění také oprávnění z účtu výroba. Administrátoři mají neomezený přístup. Posledním aktérem, znázorněným v diagramu případů použití je obecný uživatel Čas, který slouží pro znázornění pravidelných aktivit jako je zálohování dat nebo zaznamenávání (*logování*) uskutečněných operací v aplikaci.

Aplikace bude k přihlašování využívat metodu autentizace prostřednictvím systému *Windows*. To znamená, že aplikace pozná, pod jakým uživatelským jménem je uživatel přihlášen do systému *Windows* a následně se v databázi zjistí, jaká uživatelská role tomuto uživateli náleží. Model databáze bude představen v následující kapitole.

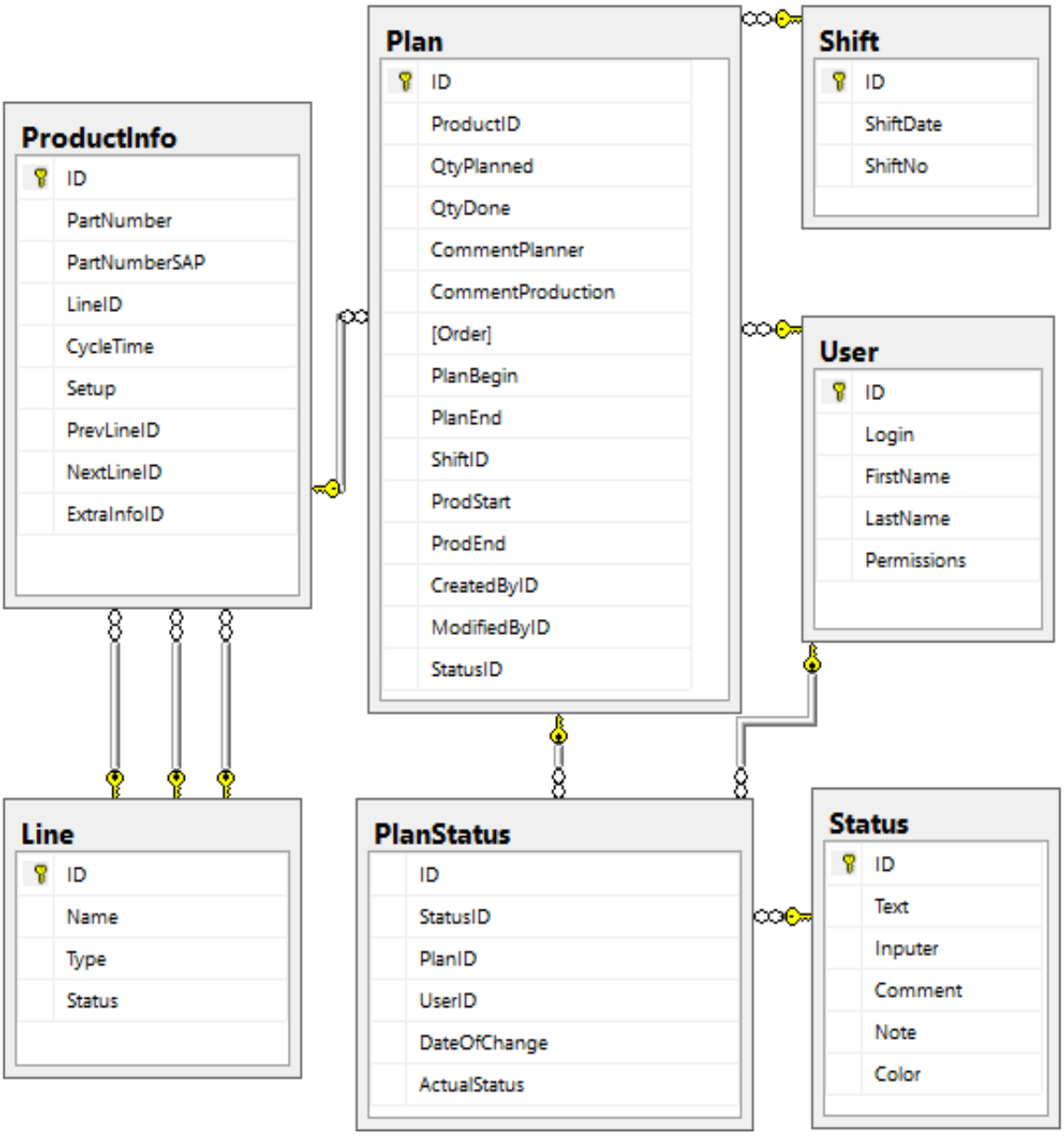
5.3.2 Databázová vrstva

Na pozadí aplikace stojí databáze obsahující sedm tabulek. Mezi nimi se nachází i tabulka *User* zmíněná v předcházející kapitole. Tato tabulka reprezentuje uživatele, kteří mají přidělený přístup do aplikace. Každý uživatel se skládá z unikátního označení ID, křestního jména a příjmení, přihlašovacího jména, které se shoduje s přihlašovacím jménem do systému Windows a slouží k párování uživatelů, a v poslední řadě z řetězce označujícím roli uživatele v aplikaci.

V databázi se dále nachází tabulka *Line*, symbolizující jednotlivé výrobní linky. Každá linka je jednoznačně identifikována pomocí celočíselného ID, názvu, typu výrobní linky a stavu výrobní linky. Stav linky je dán bitovým příznakem 0 – neaktivní, 1 – aktivní.

Jak ukazuje obrázek č. 21, na tabulku *Line* je napojená tabulka *ProductInfo*, ve které jsou uloženy informace o jednotlivých produktech určených k výrobě. Produkt, který k vyrobení musí projít více linkami, je v této tabulce uložen vícekrát

se stejným atributem *PartNumber*, ale rozdílnými dalšími vlastnostmi jako je požadované nastavení výrobní linky, čas potřebný na výrobu a především identifikátor výrobní linky, na které je produkt, nebo část produktu vyráběn.



Obr. 21 Entitně–relační schéma databáze

Nejdůležitější tabulkou je výstupní tabulka *Plan*, která obsahuje naplánovaný výrobní plán na jednu směnu. Každý záznam zde představuje jeden naplánovaný produkt, informace o počtu kusů naplánovaných k výrobě, počtu už vyrobených kusů, směnu pro, kterou je daný plán vytvořen, pořadí výroby na lince v dané směně, začátek a konec výroby daného produktu, komentáře od plánovače a z výroby, informace o tom, kdo tento plán vytvořil, popřípadě upravil a odkaz do spojovací tabulky

PlanStatus, která udržuje informace o změně stavu plánu. Na tuto tabulku je zároveň napojena tabulka *Status*, obsahující číselník všech možných stavů. V databázi je evidováno 11 stavů plánu – od předběžného požadavku, přes probíhající výrobu až po přesun zakázky na další zpracování.

Poslední tabulka – *Shift*, reprezentuje jednotlivé směny. Jediné informace, které obsahuje, jsou datum směny a pořadí směny v daném dni.

5.3.3 Aplikační vrstva

Aplikační vrstvu aplikace tvoří třídy objektového programovacího jazyka *C#*. Všechny třídy je možné rozdělit do pěti vzájemně komunikujících komponent aplikace.

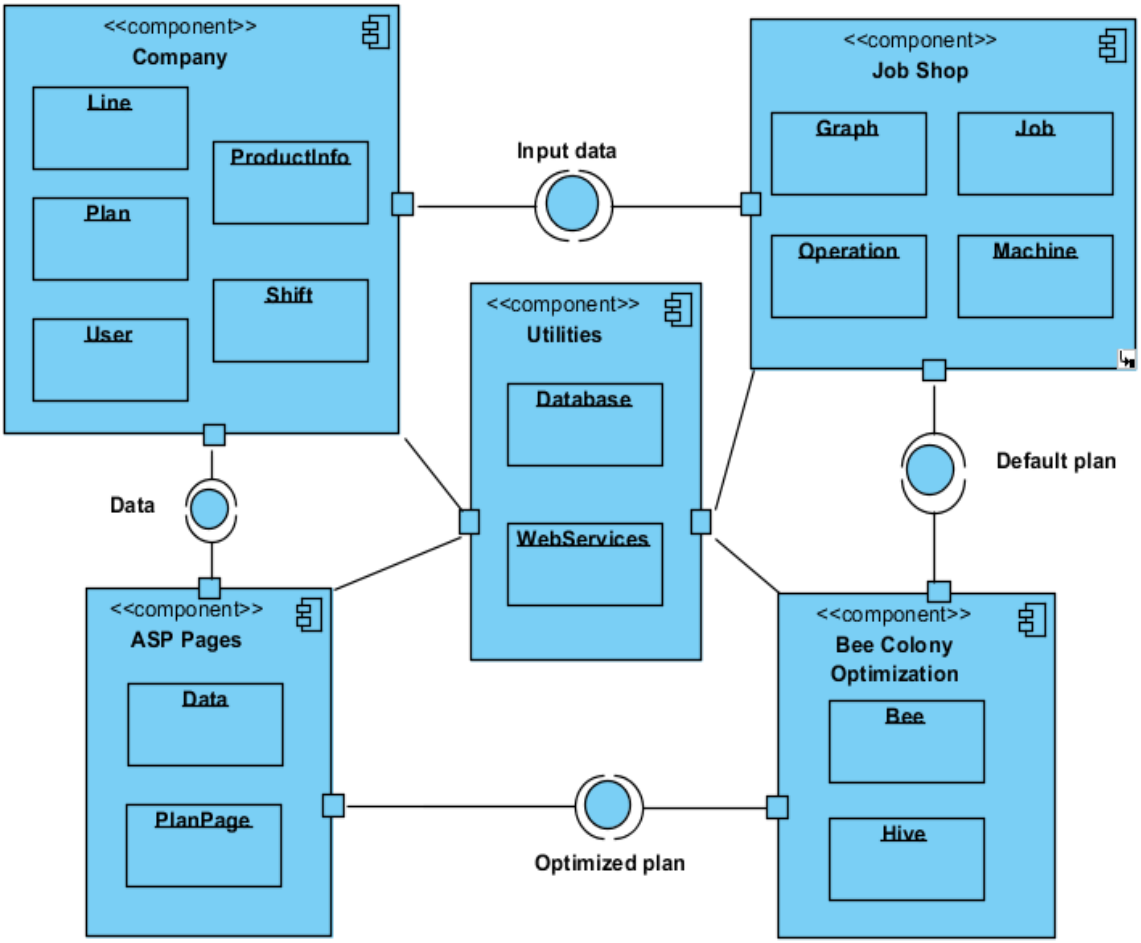
První komponentu tvoří třídy reprezentující entity nacházející se v prostředí firmy. Do této kategorie patří třídy *Line* a *ProductInfo*, znázorňující jednotlivé výrobní linky, respektive vyráběné produkty, dále třída *Shift* popisující jednotlivé směny, třída *Status* – číselník obsahující jednotlivé stavy plánu, třída *User* reprezentující uživatele aplikace a důležitá třída *Plan*, která obsahuje informace o naplánovaném výrobním plánu pro jednu směnu.

Do druhé komponenty je možné zařadit třídy určené pro implementaci problému plánování zakázkové výroby, takzvaný *jobshop*. Jedná se o třídy *Graph*, *Job*, *Operation* a *Machine* blíže představené v kapitole 5.1.

Třetí komponentu tvoří třídy sloužící k realizaci optimalizačního algoritmu včelího roje. Patří zde třída *Hive* a *Bee*, které byly detailně rozebrané v kapitole 5.2.

Čtvrtá komponenta je složená z tříd vytvořených technologií *ASP.NET* při tvorbě jednotlivých stránek webové aplikace. Jedná se o stránku *Data.aspx* a k ní náležící třídu *Data* a stránku *PlanPage.aspx* a k ní náležící stejnojmennou třídu *PlanPage*.

Poslední komponentu tvoří pomocné třídy, které v sobě ukrývají spoustu důležité funkcionality. Jedná se o třídu *Database*, která obsahuje metody pro rychlejší a bezpečnější práci s databází, třídu *Utility*, obsahující pomocné statické metody a třídu *WebServices*, uvnitř které se nachází webové služby volané *JavaScriptem*.



Obr. 22 Diagram komponent v aplikaci

Na předcházejícím obrázku jsou znázorněny vazby mezi jednotlivými komponentami aplikace.

Třídy reprezentující reálné objekty nacházející se v prostředí firmy (komponenta *Company*) obsahují převážně stejné atributy, které se nachází v databázovém modelu aplikace. Metody uvnitř těchto tříd slouží primárně k načtení jejich instancí z databáze za použití unikátního *ID*. Při načítání dat z databáze jsou použity metody třídy *Database*, které obalují základní funkce *frameworku .NET* určené pro práci s databází funkcemi starajícími se o ošetření případných výjimek v aplikaci. Pečlivé ošetření všech výjimek je důležité z hlediska zachování konzistence dat v aplikaci. Jednou z tříd je třída *User*, reprezentující instance uživatele. Jak již bylo řečeno, uživatelé se do aplikace přihlašují pomocí autentizace integrované v systému *Windows*. Přidělení uživatelského přístupu je možné provést dvěma způsoby. Prvním způsobem je enumerace jednotlivých uživatelů v konfiguračním souboru aplikace. Uživatelé jsou zde identifikováni pomocí uživatelského jména a domény, ve které se nachází. Příklad přidělení přístupu výčtem uživatelů je vidět na následujícím obrázku.

```
<configuration>
  <system.web>
    <authorization>
      <allow users="domainname\user1,domainname\user2,domainname\user3" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

Obr. 23 Autentizace pomocí enumerace jednotlivých uživatelů

Druhý způsob je vhodnější pro specifikaci většího počtu uživatelů. Proti prvnímu způsobu se liší v tom, že v konfiguračním souboru aplikace jsou uvedeny uživatelské skupiny, pro jejichž členy bude platit, že mají povolený přístup do aplikace. Příklad autentizace pomocí uživatelských skupin je na obrázku č. 24.

```
<configuration>
  <system.web>
    <authorization>
      <allow roles="domainname\Managers" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

Obr. 24 Autentizace pomocí specifikování skupiny uživatelů

V aplikaci jsou použity obě metody. Pomocí uživatelských skupin je přístup umožněn všem uživatelům, kteří jsou součástí například skupiny *production* a správce aplikace je autentizován pomocí uživatelského jména.

Metody v třídě *WebServices* se nazývají webové služby. Framework *ASP.NET* vyžaduje pro správnou funkci webových služeb označení příznakem *[WebMethod]*, takto označené metody mají veřejně dostupné rozhraní, které je možné volat pomocí funkcí jazyka *JavaScript*. Výhody tohoto přístupu spočívají především v omezení počtu načtení celé stránky, takzvaného *postbacku*. Při použití webových služeb se pomocí *JavaScriptu* pošle požadavek na server, kde se po zpracování vrátí odpověď na klientskou část aplikace a poté se opět pomocí *JavaScriptu* projeví provedené změny na stránce. Tento přístup je v aplikaci použit především při práci s plánem výroby ve funkcích, jako jsou *GetShift* (výpis naplánovaných produktů pro určitou směnu), *Add / RemoveProductToShift* (přidání nebo odebrání produktu ve směně) nebo *ChangeOrderOfProduct* (změna pořadí vyráběných produktů v plánu).

5.3.4 Prezentační vrstva

Základ prezentační vrstvy aplikace je tvořen webovými formuláři (*web forms*), které jsou klíčovou myšlenkou při vývoji pomocí technologie *ASP.NET*. Kromě webových formulářů je součástí aplikace takzvaná *master page*, která slouží pro definování jednotného vzhledu všech stránek. Aplikace obsahuje dva webové formuláře – *Data.aspx* a *PlanPage.aspx*.

Celková podoba prezentační vrstvy aplikace je výsledkem kombinace webových formulářů technologie *ASP.NET*, klasických prvků jazyka *HTML* upravených pomocí kaskádových stylů a pomocí jazyka *JavaScript*. Kromě klasické podoby *JavaScriptu* jsou v aplikaci použity volně dostupné knihovny, obohacující tento jazyk o velké množství další funkcionality. Jedná se především o dobře známou knihovnu *jQuery* a další menší, ale užitečné knihovny jako je *jQuery UI* sloužící například k usnadnění práce s dialogovými okny nebo knihovnu *DataTables*, které slouží k jednoduchému vytváření a správě tabulek dynamicky napojených na zdrojová data a to vše na klientské straně aplikace.

Vizuální podobu aplikace určuje především grafický vzor poskytnutý zadavatelem. K jeho doplnění byl kromě klasických kaskádových stylů použit *framework Bootstrap*, který obsahuje velké množství vizuálních stylů zlepšujících vzhled například tlačítek a celkově usnadňuje tvorbu výsledného vzhledu stránky.

5.4 Popis řešení z uživatelského hlediska

Jak bylo řečeno v předcházející kapitole, aplikace obsahuje dva webové formuláře, které zprostředkovávají funkcionalitu aplikace uživateli. Jednotlivé funkce aplikace jsou popsány v následujících podkapitolách.

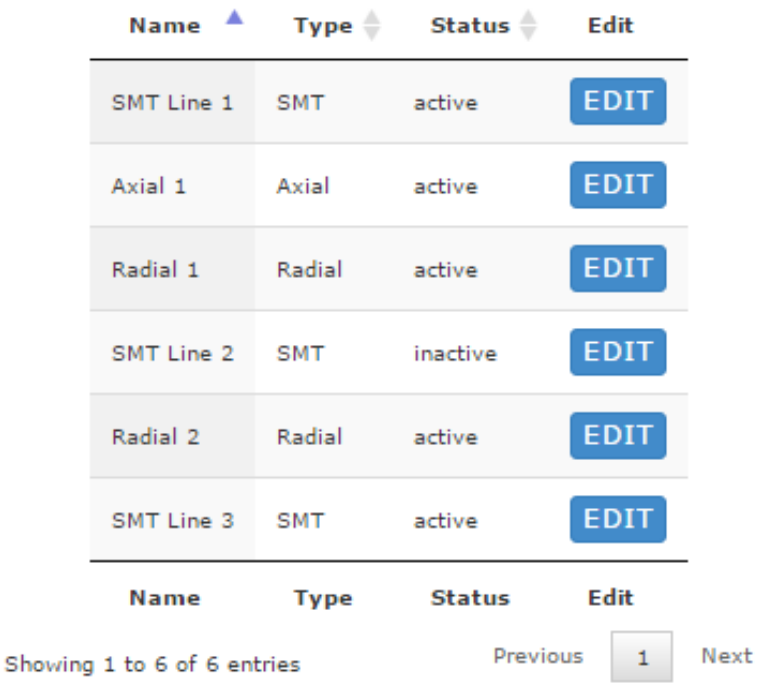
5.4.1 Formulář Data

Formulář *Data* obsahuje veškeré informace o dostupných produktech, výrobních linkách a historických nebo nadcházejících směnách.

Ve výpisu výrobních linek jsou znázorněny následující informace:

- Název linky
- Typ linky
- Status linky (aktivní / neaktivní)

Kromě zobrazení aktuálních informací může uživatel s příslušným oprávněním také zadané informace upravovat a přidávat. Výřez části aplikace, který zobrazuje informace o výrobních linkách, je vidět na následujícím obrázku.



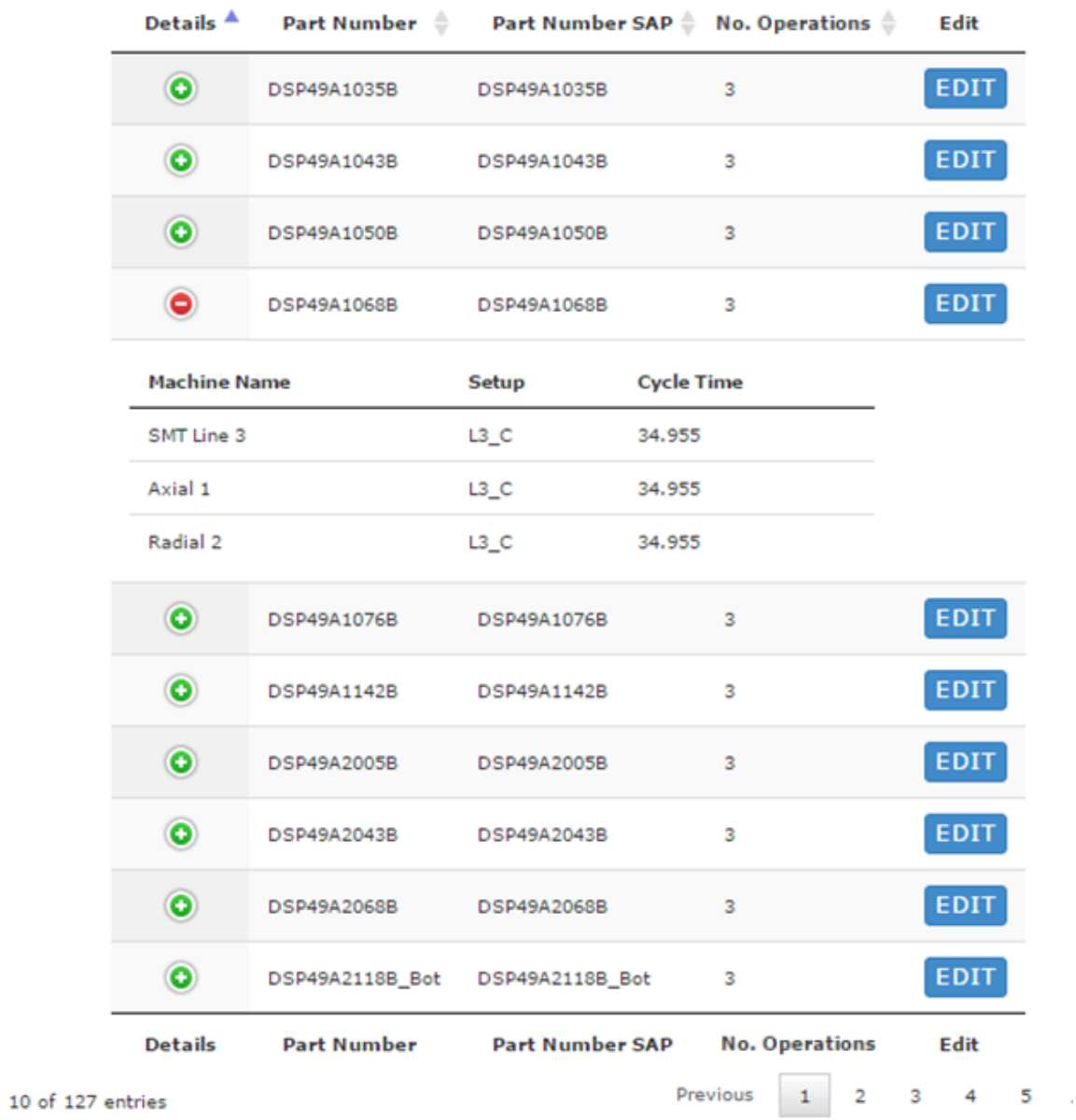
Obr. 25 Aplikace – seznam výrobních linek

Kromě prostého zobrazení umožňuje tabulka seřazení výpisu podle jednotlivých sloupců a stránkování výsledků.

Pro zobrazení historické nebo nadcházející směny je nutné nejprve vybrat požadované datum z kalendáře a následně zvolit příslušnou směnu v daném dni. V případě nadcházející směny jsou zobrazeny dostupné produkty, které může uživatel aplikace přidat plánu a tím vytvořit předběžný výrobní plán pro danou směnu. Pro dostupné produkty jsou v základním zobrazení k dispozici následující informace:

- Číslo produktu
- Číslo produktu v systému SAP
- Počet operací nutných k výrobě daného produktu

Pro zachování přehlednosti byly ostatní informace skryty pod tlačítko *Details*, které po stisku zobrazí doplňující tabulku, která obsahuje informace o jednotlivých operacích produktu. Snímek aplikace, zachycující zobrazení seznamu produktů, je na obrázku č. 26.



Obr. 26 Aplikace – seznam produktů

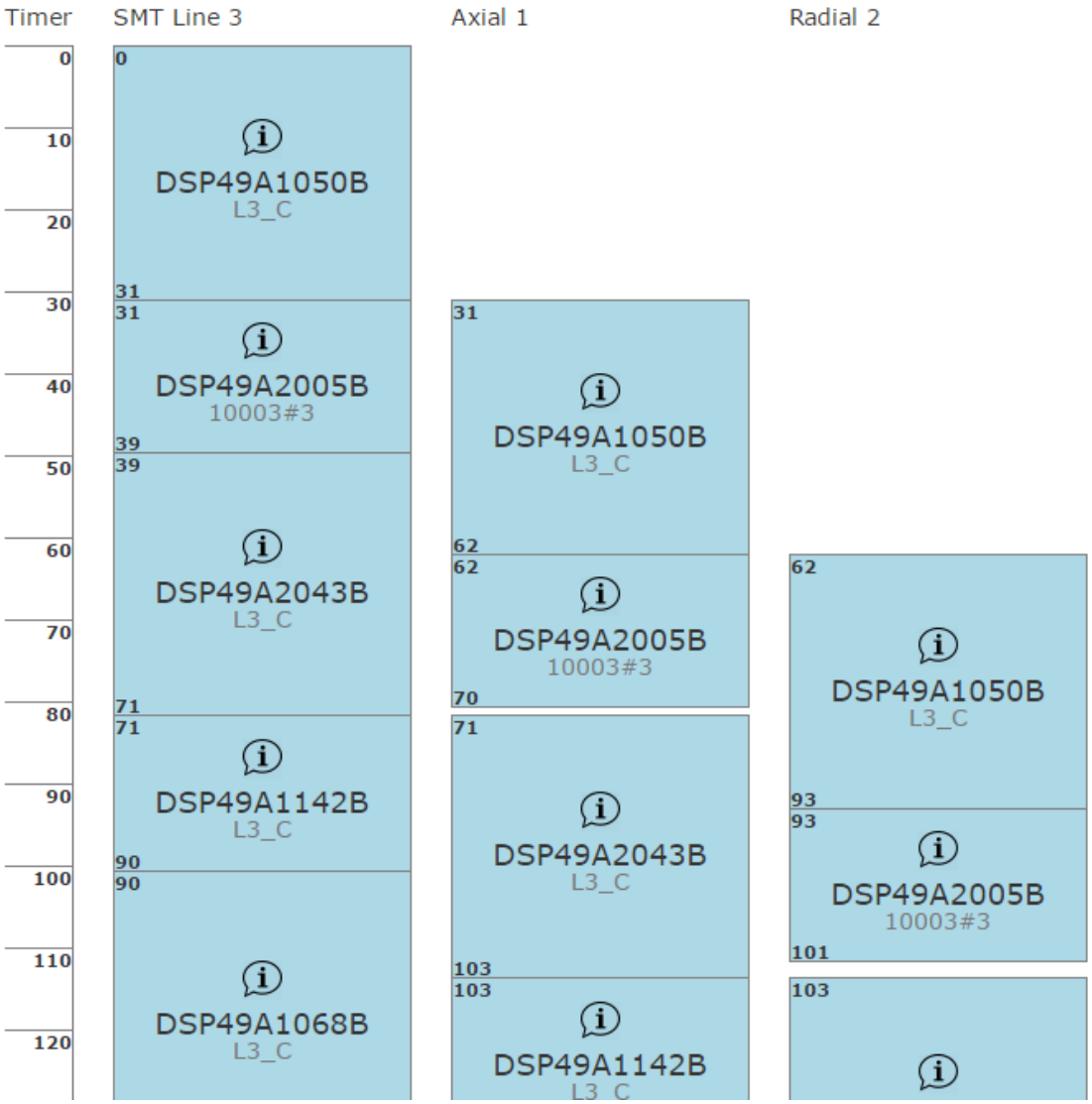
Detail produktu skrývá následující informace pro jednotlivé operace:

- Název linky, na které bude probíhat výroba
- Informace o nutném nastavení linky (*Setup*)
- Výrobní čas operace na lince (*Cycle Time*)

Seznam produktů opět obsahuje doplňující funkcionalitu jako je řazení podle jednotlivých sloupců, stránkování a možnost úpravy a přidávání produktů.

5.4.2 Formulář Plan

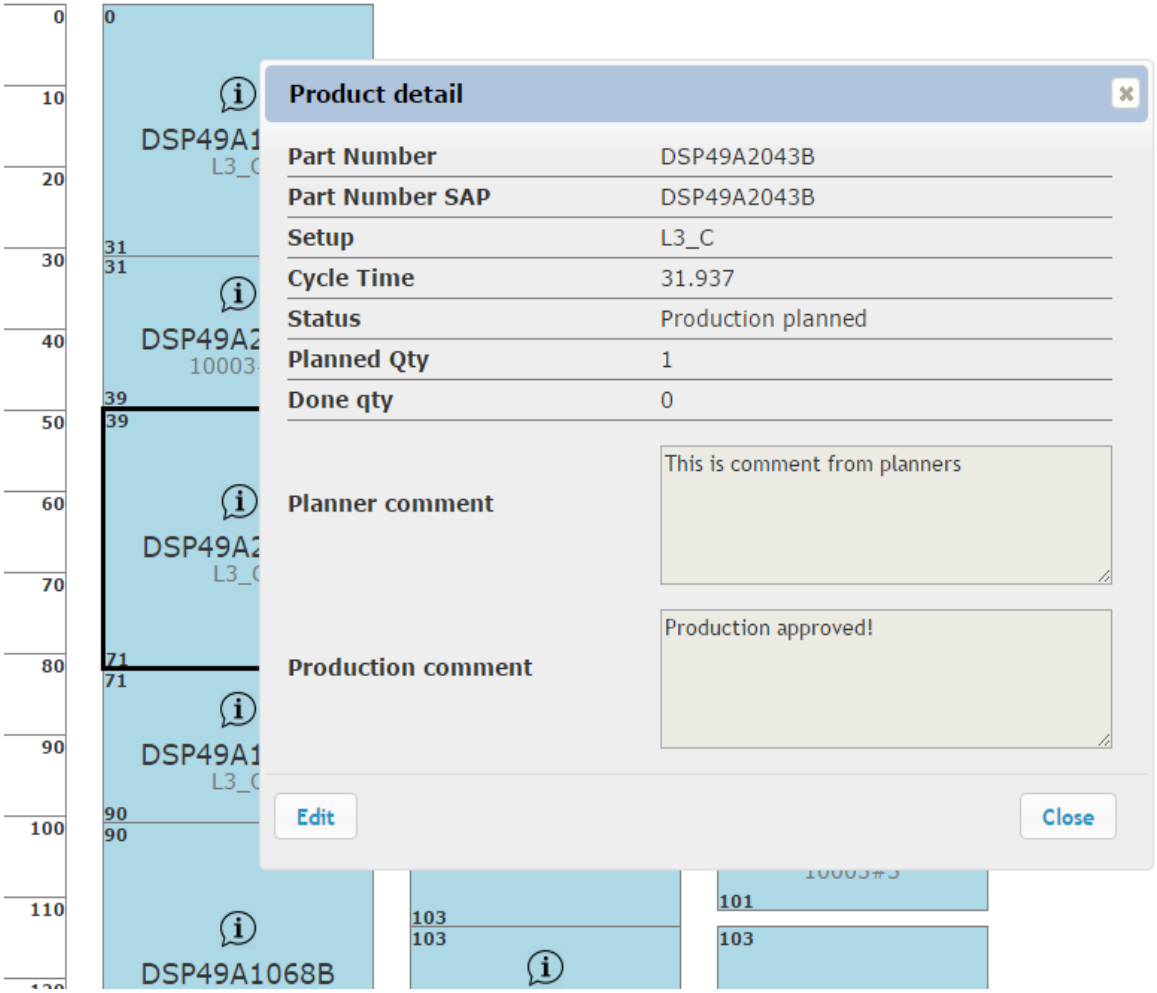
Druhý webový formulář – *Plan* je zaměřen na zobrazení, plánování a úpravy výrobního plánu. Výrobní plán je znázorněn formou vertikální fronty naplánovaných produktů, ve které jsou jednotlivé produkty zobrazeny jako obdélníkové objekty, jejichž výška symbolizuje dobu trvání výroby. Každé výrobní lince je přiřazena jedna fronta. Následující obrázek znázorňuje část výrobního plánu vizualizovaného aplikací.



Obr. 27 Aplikace – vizualizace výrobního plánu

Jednotlivé objekty poskytují uživatelům také další informace. Jedná se o identifikační číslo produktu (textový řetězec znázorněn největším písmem), nutné nastavení výrobní linky (textový řetězec pod číslem produktu) a začátek a konec vykonávání operace.

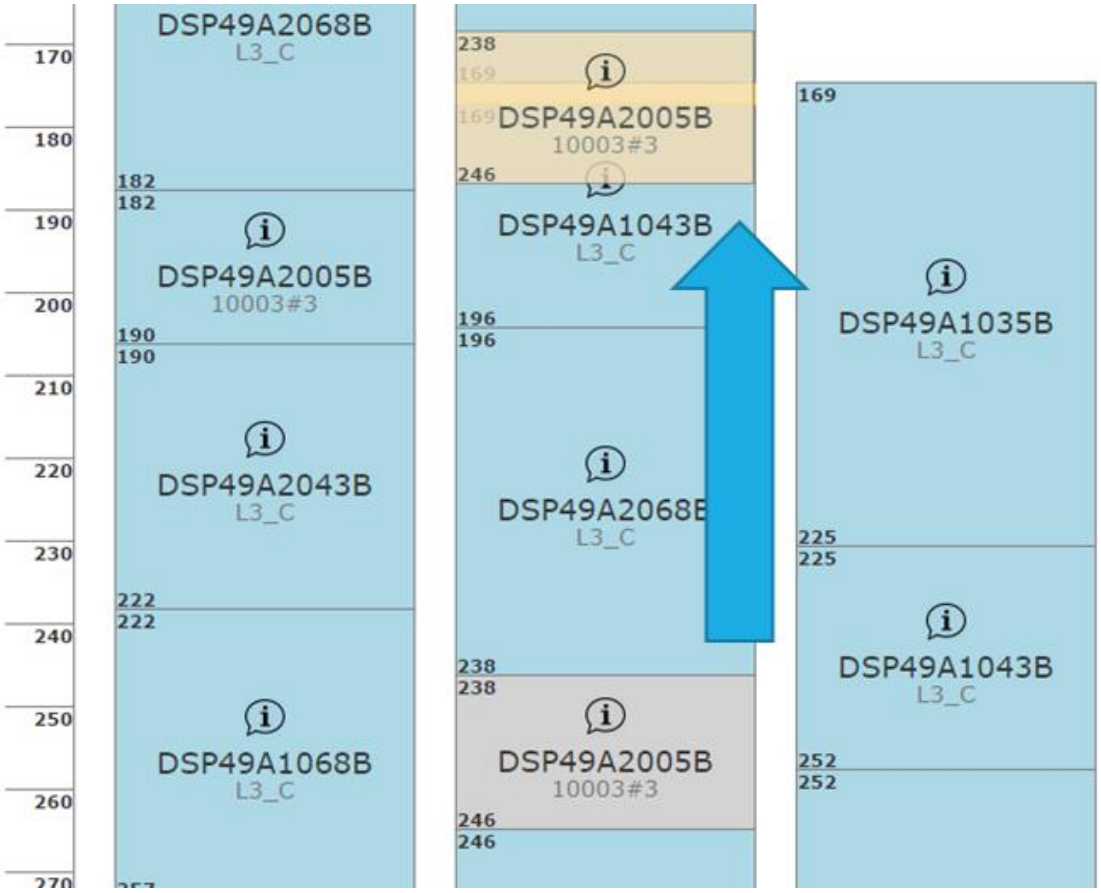
Kromě výše uvedených informací lze zobrazit dodatečné informace o požadované operaci stisknutím informační ikonky, která je umístěna uprostřed každého objektu nad číslem produktu.



Obr. 28 Aplikace – detail operace

Podrobnosti o vybrané operaci se zobrazí v dialogovém okně, ve kterém je zároveň možné provádět úpravy nebo přidávat komentáře. Podrobnosti obsahují kromě základních informací údaje o číslu výrobku použitém v systému SAP, aktuální stav operace, čas potřebný na provedení operace, počet naplánovaných a dosud vyrobených produktů nebo poznámky z výroby a od plánovače.

Další implementovanou funkcionalitou je možnost změny pořadí provádění naplánovaných operací. Tato funkce slouží k přeplánování výrobního rozvrhu v případě náhlé změny priority určité produktu. Uživatelé aplikace mohou tuto změnu priority provést jednoduchým způsobem – takzvanou metodou *drag-and-drop* (chyť-přetáhni-pusť) přímo na výrobním plánu. Znázornění této metody je na obrázku č. 29.



Obr. 29 Aplikace – změna pořadí naplánovaných operací

Funkce je realizována tak, že uživatel uchopí kurzorem vybranou operaci a poté ji přesune na požadované místo výrobního plánu. Pohyb je umožněn pouze v rámci stejné linky. Po přesunutí operace se spustí kontrolní algoritmus, který zkontroluje, zda je tento nový výrobní rozvrh realizovatelný a případně přepočítá časy začátku a dokončení jednotlivých operací nebo zobrazí uživateli upozornění o neproveditelném plánu.

Hlavní funkcí webové aplikace je automatická optimalizace plánu. Tato funkce optimalizuje výrobní plán tím, že se pokusí přeskládat pořadí jednotlivých operací na všech strojích tak, aby docházelo k co nejmenším prostojům na strojích, a aby se co nejvíce zkrátil celkový výrobní čas. Úspěšnost této optimalizace je popsána v následující kapitole.

5.5 Testování

Nejdůležitějším z faktorů, který popisuje úspěšnou implementaci optimalizačního algoritmu, je množství ušetřeného celkového výrobního času na výrobním plánu oproti stavu před optimalizací. Pro zjištění skutečných výsledků bylo nutné provést

několik testů na předpřipravených datech. Při testování byla vybrána množina produktů určených k výrobě, tyto výrobky přidány do fronty výrobků určených k naplánování a poté byl vytvořen výrobní plán. Z výrobního plánu byl zjištěn celkový výrobní čas pro daný rozvrh a následně byl nad tímto rozvrhem spuštěn optimalizační algoritmus. Veškeré úkony byly provedeny prostřednictvím vytvořené webové aplikace, aby došlo k co nejvěrnějšímu přiblížení provozu v reálném prostředí. Při testování byly použity 3 různé sady testovacích dat s rozdílnou velikostí. Nejmenší sadu dat tvořil výrobní plán pro deset typů 10 produktů, které byly vyráběny prostřednictvím 30 operací prováděných na 3 strojích. Další sada dat byla určena opět pro 3 stroje, ale obsahovala 30 typů produktů. Poslední sadu dat tvořil plán pro 20 typů výrobků složených ze 74 operací, které jsou vyráběny na 5 strojích. Použité produkty, včetně jejich operací a výrobních časů byly převzaty z testovacích dat obdržených od zadavatele. Pro jednoduchost bylo při testování použito konstantní množství kusů jednotlivých výrobků, celkové výrobní časy jsou proto spíše orientační a slouží především k porovnání situace před a po optimalizaci. Počáteční výrobní čas je u jednotlivých pokusů záměrně rozdílný, neboť při vytváření počátečního rozvrhu hraje roli pravděpodobnost, která ovlivňuje pořadí provádění operací na jednotlivých strojích.

5.5.1 První testovací sada dat

Prvně testovaný výrobní plán obsahoval 10 typů produktů, u kterých platilo, že každý finální výrobek je nutné vyrobit pomocí 3 operací. Na této sadě bylo provedeno 10 testovacích měření. Výsledné výrobní časy, seřazené od nejdelších po nejkratších před optimalizací, jsou zobrazeny v následující tabulce:

Tab. 2 Porovnání výsledných časů – první testovací data

Test č.	Výrobní čas [s]	Výrobní čas po optimalizaci [s]	Zlepšení [%]	Zlepšení [s]
1	632	400	37 %	232
2	605	400	34 %	205
3	597	400	33 %	197
4	581	400	31 %	181
5	565	400	29 %	165
6	531	400	25 %	131
7	517	400	23 %	117
8	478	400	16 %	78
9	466	400	14 %	66
10	400	400	0 %	0
Průměr	**537 ± 69,1**	**400 ± 0,0**	**24 % ± 10,7 %**	**137 ± 69,1**

V posledním řádku tabulky jsou uvedeny zprůměrované výsledky testování. Z výsledků lze konstatovat, že při plánování výroby 10 výrobků dochází díky optimalizaci ke zlepšení výrobních časů o průměrně 24 %. Při 10. pokusu nedošlo k žádnému zlepšení. Tento stav je možné okomentovat tím, že algoritmus sestavení počátečního rozvrhu nalezl díky malému množství možných kombinací nejlepší možné řešení a použitím optimalizačního algoritmu poté už nedošlo k žádné změně ve výrobním plánu.

Na obrázku č. 27 umístěného v Příloze A je pro ilustraci uvedena část výrobního plánu po sestavení počátečního rozvrhu (levá strana obrázku) a po optimalizaci (pravá strana obrázku). Tato ilustrace byla pořízena při testovacím pokusu č. 2.

5.5.2 Druhá testovací sada dat

Druhá testovací sada měla podobnou strukturu jako první, ale obsahovala větší počet dat. V tomto výrobním plánu byla naplánována výroba 30 typů produktů, u nichž platilo, že každý výrobek je postupně vyráběn na 3 strojích. Výrobní plán tedy dohromady obsahoval 90 operací.

Tab. 3 Porovnání výsledných časů – druhá testovací data

Test č.	Výrobní čas [s]	Výrobní čas po optimalizaci [s]	Zlepšení [%]	Zlepšení [s]
1	2762	1504	46 %	1258
2	2730	1504	45 %	1226
3	2695	1504	44 %	1191
4	2613	1504	42 %	1109
5	2580	1504	42 %	1076
6	2504	1504	40 %	1000
7	2472	1504	39 %	968
8	2316	1504	35 %	812
9	2153	1504	30 %	649
10	2079	1504	28 %	575
Průměr	**2490 ± 225,9**	**1504 ± 0,0**	**39 % ± 5,9 %**	**986 ± 225,9**

Při testování na druhé sadě dat vykazoval optimalizační algoritmus velmi vysokou míru procentuálního zlepšení. V nejlepším případě dosáhl na 46% zlepšení výrobního času, což znamená téměř dvojnásobné zkrácení celkového výrobního času.

5.5.3 Třetí testovací sada dat

Poslední testovací sada obsahovala složitější výrobní plán z hlediska struktury procesu výroby produktů. Produkty v tomto plánu byly celkově vyráběny na 5 různých

strojích, s tím, že jednotlivé produkty byly vyráběny na 2, 3 a případně i všech 5 strojích. V plánu bylo přítomno 20 typů výrobků sestávajících se ze 74 operací. Výsledky měření jsou uvedeny v následující tabulce:

Tab. 4 Porovnání výsledných časů – třetí testovací data

Test č.	Výrobní čas [s]	Výrobní čas po optimalizaci [s]	Zlepšení [%]	Zlepšení [s]
1	2923	2376	19 %	547
2	2912	2336	20 %	576
3	2889	2309	20 %	580
4	2851	2250	21 %	601
5	2835	2341	17 %	494
6	2769	2264	18 %	505
7	2758	2301	17 %	457
8	2691	2335	13 %	356
9	2689	2313	14 %	376
10	2561	2310	10 %	251
Průměr	**2787 ± 110,4**	**2313 ± 35,1**	**17 % ± 3,3 %**	**474 ± 108,6**

Při provádění testů na třetí sadě testovacích dat dosahoval algoritmus průměrně 17% zlepšení výrobního času. Zajímavým poznatkem z testování bylo, že optimalizační algoritmus zde nikdy nenašel dvě stejné řešení.

Při testování byly také pečlivě sledovány časové nároky na optimalizaci. U první testovací sady dat byla optimalizace provedena téměř okamžitě. Časová náročnost optimalizace druhé testovací sady se pohybovala v řádu jednotek sekund. U poslední nejsložitější testovací sady dat, trvala optimalizace výrobního plánu několik desítek sekund.

6 Zhodnocení výsledků

6.1 Shrnutí

Práce se zabývá problematikou optimalizace plánování výroby pomocí měkkých výpočtů a její následné implementace do webové aplikace sloužící k plánování výroby v prostředí výrobní firmy.

V kapitole č. 2 je všeobecně popsán problém rozvrhování se zaměřením na tématiku plánování výroby. Především jsou zde popsány základní principy a představeny dva základní typy – plánování zakázkové a proudové výroby.

Kapitola č. 3 se podrobněji věnuje problému rozvrhování zakázkové výroby a jsou zde definovány všechny pojmy nutné k pochopení dané problematiky. Hlavními body této kapitoly jsou představení jednotlivých kritérií výrobního rozvrhu, reprezentace dat, problémy rozvrhování v reálných podmínkách a také jsou zde nastíněny možné metody řešení problému rozvrhování výroby.

Prostředí výrobní firmy je představeno v kapitole č. 4. Kromě analýzy prostředí je zde uvedeno zamyšlení nad výběrem vhodné optimalizační strategie a specifikována metodika včetně stanovení jednotlivých dílčích cílů.

Praktická část práce, tj. navržení a implementace webové aplikace v prostředí jazyka *C#* a *frameworku ASP.NET*, je popsána v kapitole č. 5. Tato kapitola je rozdělena do tematicky seřazených celků, kde v kapitole 5.1 je představena implementace problému rozvrhování výroby za použití reprezentací pomocí disjunktivního grafu. Kapitola 5.2 se věnuje návrhu a implementaci optimalizace pomocí algoritmu včelího roje (*Bee Colony Optimization*). Návrh a implementace webové aplikace je řešena v kapitole 5.3, kde jsou podrobně rozebrány jednotlivé vrstvy aplikace. Kapitola 5.4 je věnována rozebráním výsledné webové aplikace z uživatelského hlediska, to znamená ilustrací uživatelského rozhraní a představením jednotlivých funkcí. Praktická část práce je ukončena představením výsledků testování optimalizačního algoritmu na různých sadách testovacích dat.

6.2 Diskuze

Výsledkem práce je webová aplikace vytvořená pomocí jazyka *C#* a *frameworku ASP.NET*, sloužící k plánování výroby v prostředí reálné výrobní firmy. Stěžejní částí aplikace je vytvoření výrobního rozvrhu pro zvolenou směnu. Výrobní rozvrh se skládá ze zvolených produktů, které mají být v dané směně vyrobeny. Smyslem aplikace je nejenom umožnit správu dostupných produktů a jejich naplánování do směny, ale především co největší optimalizace výsledného výrobního plánu. Této optimalizace je dosaženo použitím hejnového algoritmu včelího roje. Aplikace bere v potaz různé úrovně oprávnění uživatelů a určuje jejich pravomoci na základě autentizace prostřednictvím systému *Windows*. Prototyp aplikace, vytvořený v rámci diplomové práce, bude sloužit vybrané výrobní firmě k představení možností plá-

nování zakázkové výroby pomocí moderních metod, které jsou schopny ušetřit a zefektivnit práci zaměstnancům na pozici plánovač výroby a tím poskytnout výrobní firmě konkurenční výhodu.

V průběhu implementace aplikace bylo nutné překonat několik překážek. Největší překážkou byla absence jakéhokoliv teoretického základu na poli plánování výroby a z toho vyplývajících potíží s implementací problému zakázkové výroby jakožto základu pro použití optimalizační technik. Tato překážka byla nakonec vyřešena pomocí reprezentace problému disjunktivním grafem, který se ukázal jako ideální prostředek pro následnou optimalizaci včelím rojem.

Optimalizační algoritmus včelího roje se ukázal být efektivním způsobem optimalizace pro tento typ problému. Provedené testy ukázaly spolehlivost a pro účely prototypu dostačující účinnost a rychlost této metody. Z výsledků testování je možné dojít k závěru, že účinnost optimalizačního algoritmu stoupá s počtem operací, kdy při testování na výrobním plánu s 30 výrobky byla průměrná účinnost pouze 24 %, ale při testech na sadě dat obsahujících 90 výrobků dosáhl algoritmus až 46% zlepšení celkového výrobního času. Toto chování je způsobeno faktem, že při vyšším počtu operací má algoritmus větší možnost změny pořadí provádění výrobků na jednotlivých strojích. Při testování na sadě dat se složitou strukturou (produkty se vyráběly postupně až na 5 strojích) dosahoval optimalizační algoritmus průměrného 17% zlepšení výrobního času. Oproti předchozím testům algoritmus na této sadě dat nikdy nenašel dvě stejné řešení, ale výsledné řešení se vždy lišilo na základě nalezeného počátečního řešení. Nižší procentuální zlepšení než u ostatních testovacích dat lze zdůvodnit tvrzením, že komplikovaná struktura výrobního procesu neposkytovala optimalizačnímu algoritmu dostatek prostoru na optimalizaci.

Při testování se ukázalo, že největší slabinou při realizaci problému plánování výroby je náhodně stanovený počáteční výrobní rozvrh. Vhodněji zvoleným algoritmem pro nalezení počátečního rozvrhu by bylo možné eliminovat některé sice proveditelné, ale velmi neefektivní výrobní rozvrhy, které sloužily jako startovní bod pro optimalizační algoritmus. Dalším nedostatkem současného řešení je malé přizpůsobení problému podmínkám skutečné výroby. Většího přizpůsobení pro potřeby podniku by bylo dosaženo zahrnutím času přípravy a dalších specifických časů do výrobního času jednotlivých produktů.

Možnosti vylepšení webové aplikace se nachází ve větší míře integrace do stávajícího informačního systému firmy. Jedná se především o využití výstupu z předcházejícího procesu výroby, který by sloužil jako vstupní data pro plánování další fáze výroby. Proces plánování výroby by se tímto krokem ještě více zautomatizoval a zefektivnil.

Realizovaná webová aplikace není komerčním řešením, které by kompletně obsáhlo veškeré specifika plánování zakázkové výroby, ale splňuje všechna stanovená kritéria a je připravena na svůj případný další vývoj.

7 Závěr

Cílem této práce bylo vyřešit problém rozvrhování výroby pomocí vhodné optimalizační strategie a poskytnout nástroj v podobě webové aplikace, implementované na základě požadavků konkrétní výrobní firmy.

Cíl práce byl splněn, vytvořený prototyp webové aplikace je schopen rozvrhnout a následně efektivně optimalizovat výrobní plán. Efektivita optimalizace byla ověřena v řadě provedených testů, které prokázaly správnost a použitelnost navrženého konceptu řešení.

8 Literatura

BAKER, KENNETH R., TRIETSCH, D. *Principles of sequencing and scheduling*. Hoboken, N.J.: John Wiley, c2009. ISBN 0470391650.

BLAZEWICZ, J. et al. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, Berlin, 1996. 491 p.

GLOVER, F., *Tabu Search-Part I*, ORSA Journal on Computing, Vol. 1, 1989, 206 p.

CHEE, P.L.,JAIN, C.L. *Advances in Swarm Intelligence In Innovations in swarm intelligence.* New York: Springer, c2009, s. 1-7. ISBN 9783642042249.

KAHN, ARTHUR B. ,*Topological sorting of large networks*, Communications of the ACM 5 (11): 558–562, doi:10.1145/368996.369025, 1962

KARABOGA, D. *An idea based on honey bee swarm for numerical optimization* (Technical Report-Tr06, October, 2005), Erciyes University, Engineering Faculty Computer Engineering Department Kayseri/Türkiye, 2005

KARABOGA, D., BASTURK, B. *A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm.* J. Global. Optim. 39, 2007, 459–471p.

KARABOGA, D., BASTURK, B. *On the performance of artificial bee colony (ABC) algorithm.* Appl. Soft. Comput. 8, 2008, 687–697p.

KIRKPATRICK, S., GELATT C. D. , VECCHI M. P. *Optimization by Simulated Annealin*, Science, New Series, Vol. 220, No. 4598. (May 13, 1983), pp. 671-680.

KOKASH, N. *An introduction to heuristic algorithms*. Department of Informatics and Telecommunications University of Trento, Italy, 2016.

MAJER, P., *Reprezentace problému rozvrhování zakázkové výroby disjunktivním grafem*. XXVI. ASR '2001 Seminar, Instruments and Control, Ostrava, 2001.

METROPOLIS, N., et al. *Equation for State Calculation for Fast Computing Machines*, J. Chem. Phys., 1963, vol. 21, pp. 1087-1092.

MILLONAS, M., *Swarms, Phase Transitions, and Collective Intelligence In Langton*, C.G.(ed.) Artificial Life III, vol. XVII, Addison-Wesley Publishing Company,Reading, 1994, pp. 417–445.

PINEDA, M. *Scheduling: theory, algorithms, and systems*. 3rd ed. New York: Springer, c2008. ISBN 978-0-387-78935-4.

SATO, T., HAGIWARA, M. *Bee System: Finding Solution by a Concentrated Search*. In*: Proceedings of the IEEE International Conference on Systems*, Man and Cybernetics Computational Cybernetics and Simulation, Orlando, FL, USA, 1997, pp. 3954–3959

ROBERT, Y., VIVIEN, F. *Introduction to scheduling.* Boca Raton: CRC Press, c2010. Chapman & Hall/CRC computational science series. ISBN 1420072730.

SIMON, D. *Evolutionary optimization algorithms: biologically-inspired and population-based approaches to computer intelligence*. Hoboken: John Wiley & Sons, c2013. ISBN 978-0-470-93741-9.

ŠEDA, M., DVOŘÁK, J. & BURDA, J. *Scheduling Job Shops Using Genetic Algorithms and Local Search Framework,* In Proceedings of the 6th International Conference on Soft Computing MENDEL '1999, Brno, Czech Republic, 1999, p. 131-138.

TEODOROVIĆ, D. *Bee Colony Optimization (BCO) In Innovations in swarm intelligence.* New York: Springer, c2009, s. 39-60. ISBN 9783642042249.

9 Seznam použitých zkratek a značení

.NET	Soubor technologií vytvořených firmou Microsoft pro vývoj softwaru.
ACO (Ant Colony Optimization)	Optimalizace mravenčí kolonií. Jedná se o hejnový algoritmus inspirovaný chováním mravenců při hledání potravy.
ASP.NET	ASP.NET je součást .NET Frameworku pro tvorbu webových aplikací a služeb.
autentizace	Proces ověření identity subjektu.
BCO (Bee Colony Optimization)	Optimalizace včelím rojem. Jedná se o hejnový algoritmus inspirovaný chováním včel při sběru potravy.
Bootstrap	Volně dostupná sada nástrojů pro tvorbu webových aplikací. Obsahuje návrhářské šablony založené na HTML a CSS.
bottleneck	Úzké místo v systému.
C_j (Completion time)	Čas, ve kterém je dokončeno zpracování úlohy j.
CPM (Critical Path Method)	Metoda kritické cesty je matematický algoritmus plánování průběhu projektu.
drag-and-drop	Gesto, ve kterém uživatel uchytí objekt a posune ho na jiné místo, což následně vyvolá nějakou reakci.
Eet	Nejdříve možný čas konce zpracování operace.
Est	Nejdříve možný čas začátku zpracování operace.
F_j (Flowtime)	Čas, který úloha j stráví v systému.
flow shop	Termín pro rozvrhování proudové výroby.
framework	Aplikační rámec. Struktura, která slouží jako podpora při vývoji softwaru.
GA (Genetic Algorithms)	Genetické algoritmy. Patří do kategorie evolučních algoritmů.
JavaScript	JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk.
JIT (Just-in-time)	Termín pro přístup k výrobě, který umožňuje podniku vyrábět výrobky v určeném množství a určeném čase dle požadavků zákazníka.
job	Množina operací definující úlohu.
job shop	Termín pro rozvrhování zakázkové výroby.
jQuery	Multiplatoformní JavaScriptová knihovna určená k zjednodušení interakce JavaScriptu s HTML.
Let	Nejpozději možný čas konce zpracování operace.
L_j (Lateness)	Čas, o který výsledný čas dokončení úlohy j překročil požadovaný čas dokončení této úlohy.
Lst	Nejpozději možný čas začátku zpracování operace.
machine	Zařízení, na kterém se zpracovávají jednotlivé operace

makespan	Celkový výrobní čas.
operation	Základní technologický úkon.
postback	Proces, ve kterém jsou data poslány na stejnou stránku, jako ze které byly poslány.
processing time	je čas potřebný na zpracování úlohy.
PSO (Particle Swarm Optimization)	Optimalizace hejnem částic.
RCPSP (resource-constrained project scheduling problem)	Problém plánování projektu s omezenými zdroji.
SAP	Softwarový produkt, který slouží pro řízení podniku.
setup time	Čas potřebný k nastavení stroje.
schedule π	Výrobní rozvrh.
slack times	Doba, po které je stroj v nečinnosti.
SMT (surface mount technology)	Technologie povrchové montáže.
technologic time	Časový úsek, o který je proces zpracování úlohy zdržen po skončení zpracování dané operace, než může za-čít zpracování na dalším stroji.
T_j (tardiness)	Kvantifikátor nedochvilnosti. Obsahuje pouze kladné hodnoty zpoždění úlohy j.
transport time	Doba dopravy výrobku po zpracování operace na následující stroj.
Web Service	Webová služba je softwarový systém umožňující interakci dvou strojů na síti.

10 Seznam obrázků

Obr. 1 Ganttův diagram 13

Obr. 2 Ryzí model proudové výroby 16

Obr. 3 Obecný model proudové výroby 16

Obr. 4 Rozvrh proudové výroby s prostoji 17

Obr. 5 Model funkce stroje u zakázkové výroby 17

Obr. 6 Úlohy a operace v problému zakázkové výroby 18

Obr. 7 Proveditelný rozvrh pro n = 3 úlohy a m = 3 stroje 19

Obr. 8 Disjunktivní graf 21

Obr. 9 Orientovaný disjunktivní graf – úplná selekce 22

Obr. 10 Zjednodušený orientovaný disjunktivní graf 22

Obr. 11 Tvar křivky penalizace v praxi 24

Obr. 12 Montážní a distribuční operace 25

Obr. 13 Disjunktivní graf rozšířený o montážní a distribuční operace 26

Obr. 14 Lokální a globální maximum 28

Obr. 15 Operátor křížení 30

Obr. 16 Operátor mutace – bitová inverze 31

Obr. 17 Objektová reprezentace disjunktivního grafu 39

Obr. 18 Algoritmus zpětného průchodu disjunktivním grafem 41

Obr. 19 Stavový diagram jedné iterace metody *Solve* 45

Obr. 20 Diagram případů použití 46

Obr. 21 Entitně–relační schéma databáze 48

Obr. 22 Diagram komponent v aplikaci 50

Obr. 23 Autentizace pomocí enumerace jednotlivých uživatelů 51

Obr. 24 Autentizace pomocí specifikování skupiny uživatelů 51

Obr. 25 Aplikace – seznam výrobních linek 53

Obr. 26 Aplikace – seznam produktů 54

Obr. 27 Aplikace – vizualizace výrobního plánu 55

Obr. 28 Aplikace – detail operace 56

Obr. 29 Aplikace – změna pořadí naplánovaných operací 57

Obr. 30 Část výrobního plánu před a po optimalizaci 72

11 Seznam tabulek

Tab. 1 Srovnání reprezentace problému zakázkové výroby 35

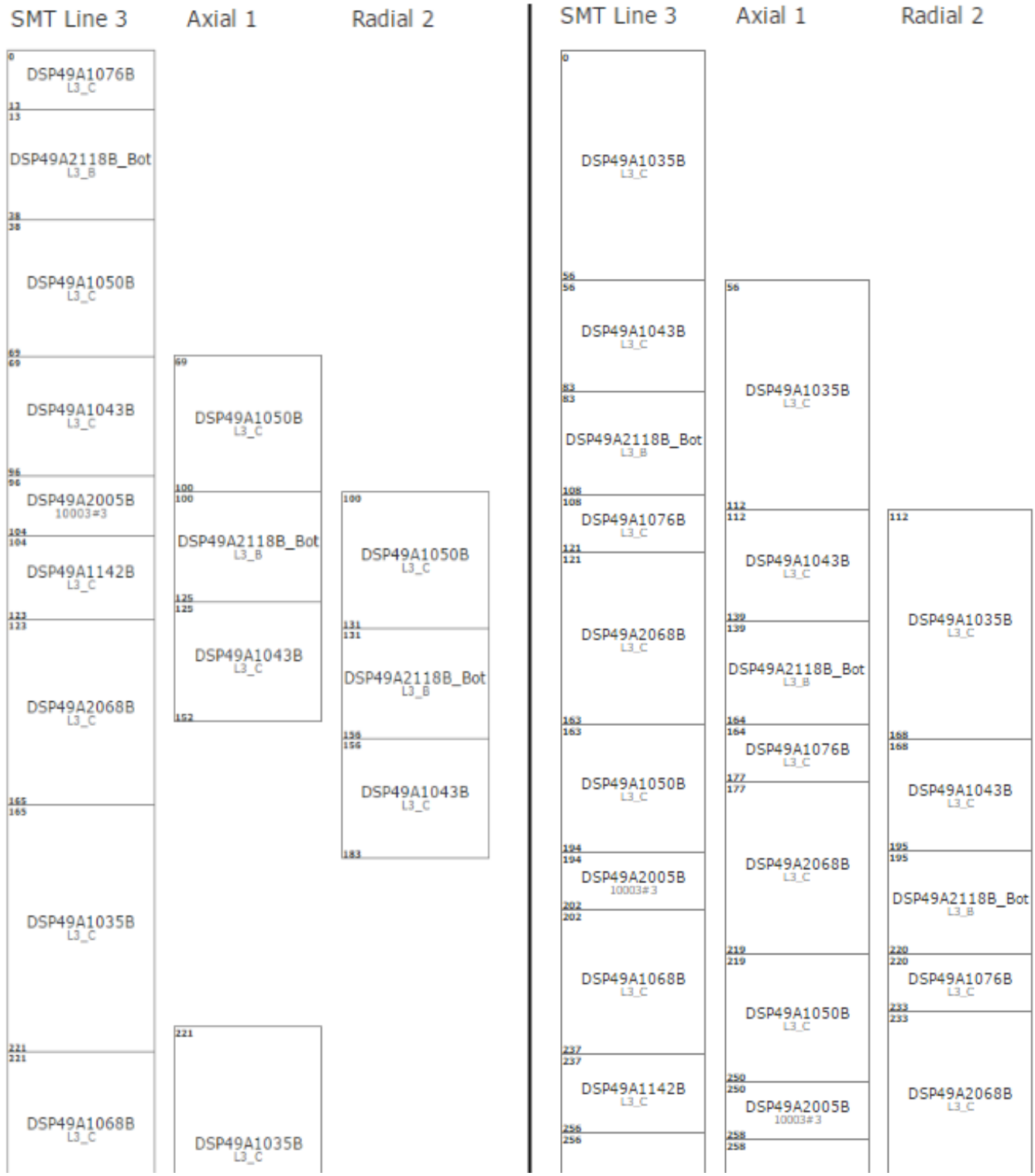
Tab. 2 Porovnání výsledných časů – první testovací data 58

Tab. 3 Porovnání výsledných časů – druhá testovací data 59

Tab. 4 Porovnání výsledných časů – třetí testovací data 60

Přílohy

A Ilustrace optimalizace výrobního plánu



Obr. 30 Část výrobního plánu před a po optimalizaci

B Obsah přiloženého CD

Na přiloženém CD lze najít následující data:

- Sql_scripts.zip – skripty sloužící k vytvoření databáze.
- ePlanner_DB_backup.bak – záloha databáze s testovacími daty.
- ePlanner_solution.zip – zdrojové soubory webové aplikace.